भाकृअनुप
ICAR

हर कदम, हर डगर
किसानों का हमसफर
भारतीय कृषि अनुसंधान परिषद
Agrisearch with a human touch

ऑनलाइन प्रशिक्षण कार्यक्रम

ट्रांसक्रिप्टोमिक डेटा विश्लेषण के लिए सांख्यिकीय एवं कृत्रिम बुद्धिमत्ता आधारित दृष्टिकोण

Online Training Programme

Statistical and AI/ML Approaches for Transcriptomic Data Analysis

Sponsoring Project

DBT - Establishment of Centre for Bioinformatics and Computational Biology in Agriculture-BIC at ICAR-Indian Agricultural Statistics Research Institute

E-Manual

September 01-08, 2026

Course Director: Dr. Girish Kumar Jha

Course Coordinators: Dr. Mohammad Samir Farooqi

Dr. Sudhir Srivastava

Dr. Sharanbasappa

Division of Agricultural Bioinformatics

ICAR-Indian Agricultural Statistics Research Institute

Library Avenue, PUSA, New Delhi - 110012

https://iasri.res.in/

आमुख

जैव सूचना विज्ञान एक अंतःविषय क्षेत्र है, जिसमें जीव विज्ञान, सांख्यिकी, कंप्यूटर विज्ञान तथा कम्प्यूटेशनल तकनीकों का समन्वय किया जाता है। पिछले दो दशकों के दौरान उच्च-प्रवाह अनुक्रमण तथा अन्य उन्नत जैव प्रौद्योगिकीय तकनीकों के तीव्र विकास के परिणामस्वरूप जैविक विज्ञान के क्षेत्र में अत्यधिक मात्रा में डेटा उत्पन्न हुआ है। विशेष रूप से, ट्रांसक्रिप्टोमिक तकनीकों ने विभिन्न जीवों तथा प्रयोगात्मक परिस्थितियों में जीन अभिव्यक्ति के स्वरूप, नियामक तंत्र, आणविक प्रतिक्रियाओं तथा जैविक पथों के अध्ययन के लिए व्यापक अवसर प्रदान किए हैं। ट्रांसक्रिप्टोमिक डेटा के विश्लेषण में जैव सूचना विज्ञान तथा सांख्यिकीय तकनीकों का प्रयोग डेटा के कुशल प्रसंस्करण, विश्लेषण, एनोटेशन, व्याख्या एवं दृश्यांकन में महत्वपूर्ण भूमिका निभाता है। जैविक डेटा की मात्रा एवं जटिलता में निरंतर वृद्धि के साथ कृत्रिम बुद्धिमत्ता (Artificial Intelligence) एवं मशीन लर्निंग (Machine Learning) आधारित तकनीकें भी ट्रांसक्रिप्टोमिक डेटा से पैटर्न की पहचान करने, महत्वपूर्ण जैविक जानकारी प्राप्त करने, पूर्वानुमानात्मक मॉडल विकसित करने तथा जैविक अंतर्दृष्टि प्राप्त करने के लिए महत्वपूर्ण टूल के रूप में उभर रही हैं। सांख्यिकीय एवं कृत्रिम बुद्धिमत्ता/मशीन लर्निंग आधारित दृष्टिकोणों का समन्वित उपयोग कृषि एवं संबद्ध जैविक अनुसंधान में डेटा-आधारित अध्ययन को आगे बढ़ाने के लिए नई संभावनाएँ प्रदान करता है।

इसी संदर्भ में, कृषि जैव सूचना विज्ञान प्रभाग, भा.कृ.अनु.प.–भारतीय कृषि सांख्यिकी अनुसंधान संस्थान (भा.कृ.अनु.प.–भा.कृ.सां.अनु.सं.), नई दिल्ली द्वारा डीबीटी–प्रायोजित परियोजना "भा.कृ.अनु.प.-भारतीय कृषि सांख्यिकी अनुसंधान संस्थान में कृषि में जैव सूचना विज्ञान और कम्प्यूटेशनल जीव विज्ञान केंद्र बीआईसी की स्थापना" के अंतर्गत दिनांक 01–08 सितम्बर, 2026 के दौरान "ट्रांसक्रिप्टोमिक डेटा विश्लेषण के लिए सांख्यिकीय एवं कृत्रिम बुद्धिमत्ता आधारित दृष्टिकोण" पर एक ऑनलाइन प्रशिक्षण कार्यक्रम आयोजित किया गया है। यह प्रशिक्षण कार्यक्रम राष्ट्रीय कृषि अनुसंधान एवं शिक्षा प्रणाली (NARES) में कार्यरत वैज्ञानिक, तकनीकी एवं संविदात्मक अनुसंधान कार्मिकों के लिए आयोजित किया गया है।

यह प्रशिक्षण कार्यक्रम निम्नलिखित उद्देश्यों के साथ तैयार किया गया है:

- प्रतिभागियों को ट्रांसक्रिप्टोमिक डेटा विश्लेषण में प्रयुक्त सांख्यिकीय एवं कृत्रिम बुद्धिमत्ता/मशीन लर्निंग (AI/ML) तकनीकों की व्यापक समझ प्रदान करना।
- एक्सपर्ट लेक्चर, सॉफ्टवेयर डेमोंस्ट्रेशन और प्रैक्टिकल सेशन के माध्यम से इन तकनीकों के अनुप्रयोग को प्रदर्शित करना।

इस प्रशिक्षण कार्यक्रम में (1) Linux, Python और R का परिचय, (2) एन.जी.एस. डेटा जनरेशन: तकनीकें, समस्याएं और चुनौतियां, (3) NGS डेटा: प्री-प्रोसेसिंग, असेंबली और क्वांटिफिकेशन, (4) आर.एन.ए.-सेक डेटा का एनोटेशन, (5) बल्क और एकल-कोशिका आर.एन.ए.-सेक डेटा का विश्लेषण, (6) AI/ML तकनीकें और ट्रांसक्रिप्टोमिक्स में उनका उपयोग, (7) ट्रांसक्रिप्टोमिक डेटा से AI/ML-आधारित फंक्शनल और जैविक जानकारी, और (8) माइक्रो आर.एन.ए. की पहचान और टारगेट प्रेडिक्शन, को शामिल किया गया है। इस कार्यक्रम में थ्योरेटिकल और प्रैक्टिकल, दोनों तरह के सेशन शामिल हैं, ताकि प्रतिभागियों को ट्रांसक्रिप्टोमिक अनुसंधान में इस्तेमाल होने वाले कंप्यूटेशनल वर्कफ़्लो और विश्लेषणात्मक तरीकों की जानकारी मिल सके।

इस प्रशिक्षण कार्यक्रम के आयोजन में प्राप्त बहुमूल्य मार्गदर्शन, सहयोग एवं समर्थन के लिए हम डॉ. कैरम नरसैया [सहायक महानिदेशक (प्रक्रिया अभियांत्रिकी), भा.कृ.अनु.प. एवं निदेशक, भा.कृ.अनु.प.–भारतीय कृषि सांख्यिकी अनुसंधान संस्थान] तथा डॉ. गिरीश कुमार झा [प्रधान प्रभाग, कृषि जैव सूचना विज्ञान एवं परियोजना के मुख्य अन्वेषक] के प्रति हार्दिक आभार व्यक्त करते हैं।

हमें आशा है कि यह प्रशिक्षण पुस्तिका प्रतिभागियों के लिए एक उपयोगी संदर्भ सामग्री के रूप में कार्य करेगी तथा उन्हें ट्रांसक्रिप्टोमिक डेटा विश्लेषण में सांख्यिकीय एवं कृत्रिम बुद्धिमत्ता/मशीन लर्निंग आधारित दृष्टिकोणों को समझने और कृषि अनुसंधान में प्रभावी रूप से प्रयोग करने में सहायता प्रदान करेगी।

लेखकगण

PREFACE

Bioinformatics is an interdisciplinary field that integrates biology, statistics, computer science, and computational approaches to address complex biological problems. Over the past two decades, rapid advances in high-throughput sequencing technologies have resulted in the generation of enormous volumes of biological data. In particular, transcriptomic technologies have provided unprecedented opportunities to study gene expression patterns, regulatory mechanisms, molecular responses, and biological pathways across diverse organisms and experimental conditions. The application of bioinformatics and statistical approaches enables efficient processing, analysis, annotation, interpretation, and visualization of transcriptomic data. With the increasing volume and complexity of biological datasets, Artificial Intelligence and Machine Learning (AI/ML) techniques are also becoming increasingly important for identifying patterns, extracting meaningful biological information, developing predictive models, and supporting data-driven biological research. The integration of statistical and AI/ML approaches can therefore provide valuable opportunities for advancing agricultural research and addressing complex biological problems. In this context, an Online Training Program on “Statistical and AI/ML Approaches for Transcriptomic Data Analysis” has been organized by the Division of Agricultural Bioinformatics, ICAR–Indian Agricultural Statistics Research Institute (ICAR-IASRI), New Delhi, under the DBT-funded project “Establishment of Centre for Bioinformatics and Computational Biology in Agriculture-BIC at ICAR-Indian Agricultural Statistics Research Institute” during September 01–08, 2026. The training programme is intended for Scientific, Technical, and Contractual Research Personnel working in the National Agricultural Research and Education System (NARES).

The training programme has been formulated with the following objectives:

- To provide participants with a comprehensive understanding of statistical and AI/ ML techniques used in transcriptomic data analysis.
- To demonstrate the application of these techniques through expert lectures, software demonstrations, and hands-on practical sessions.

In this training programme, lectures related to (1) Introduction to Linux, Python and R, (2) NGS Data Generation: Techniques, Issues and Challenges, (3) NGS Data: Preprocessing, Assembly and Quantification, (4) RNA-Seq Data Annotation, (5) Bulk

and Single-Cell RNA-Seq Data Analysis, (6) AI/ML Techniques and Their Applications in Transcriptomics, (7) AI/ ML-Based Functional and Biological Insights from Transcriptomic Data, and (8) microRNA Identification and Target Prediction, have been included. The programme includes both theoretical and practical sessions to provide participants with exposure to computational workflows and analytical approaches used in transcriptomic research.

We sincerely express our gratitude to Dr. Kairam Narsaiah [Assistant Director General (Process Engineering), ICAR and Director, ICAR–Indian Agricultural Statistics Research Institute] and Dr. Girish Kumar Jha [Head, Division of Agricultural Bioinformatics, ICAR–IASRI and Principal Investigator of the Project], for their valuable guidance, support, and cooperation in organizing this training programme.

We hope that this training manual will serve as a useful reference for the participants and provide them with the necessary knowledge and practical exposure to apply statistical and AI/ML approaches effectively in transcriptomic data analysis and agricultural research.

Authors

CONTENTS

S. No.	Topic	Page No.
1	Overview of Linux	1 – 11
2	NGS Data Generation: Techniques, Issues and Challenges	12 – 15
3	NGS Data: Pre-processing, Assembly and Quantification	16 – 22
4	Annotation of RNA-Seq Data	23 – 34
5	Annotation of RNA-Seq Data (Practical)	35 – 46
6	Introduction to Python	47 – 61
7	Introduction to R and RStudio	62 – 85
8	Overview of RNA-Seq Data Analysis	86 – 91
9	Differential Gene Expression Analysis in RNA-Seq Data using R	92 – 97
10	Overview of AI/ ML Techniques	98 – 131
11	AI/ML in Transcriptomic Data Analysis	132 – 138
12	Single-Cell RNA-Seq Data Analysis	139 – 149
13	Extracting Functional and Biological Insights from Transcriptomic Data Using AI/ML	150 – 156
14	miRNA Identification and Target Prediction	157 – 164

Overview of Linux

Anu Sharma and S. B. Lal

ICAR-Indian Agricultural Statistics Research Institute, New Delhi

The Linux operating system is basically a variant of the UNIX operating system, and Linux has probably all that UNIX offers and more. It is a multi-user, multitasking, network operating system which also has a user friendly Graphical User Interface (GUI).

Every desktop computer uses an operating system. The most popular operating systems are Windows, Mac OS, UNIX, Linux.

What is an Operating System?

An operating system is the first piece of software that the computer executes when a system is turned on. The operating system loads itself into memory and begins managing the resources available in the computer. It provides those resources to other applications that the user wants to run. Typical services that an operating system provides include:

A task scheduler - The task scheduler is able to allocate the execution of the CPU to a number of different tasks. Some of those tasks are the different applications that the user is running, and some of them are operating system tasks.

A memory manager - The memory manager controls the system's RAM and normally creates a larger virtual memory space using a file on the hard disk.

A disk manager - The disk manager creates and maintains the directories and files on the disk. When a file is needed, the disk manager makes it available from the disk.

A network manager - The network manager controls all data moving between the computer and the network.

Other I/O services manager - The OS manages the keyboard, mouse, video display, printers, etc.

Security manager - The OS maintains the security of the information in the computer's files and controls who can access the computer.

An operating system normally also provides the default user interface for the system. The standard "look" of Windows 98 includes the Start button, the task bar, etc. The Mac OS provides a completely different look and feel for Macintosh computers.

To understand why Linux has become so popular, it is helpful to know a little bit about its history.

Background on Linux

Linux, a UNIX-like operating system, is based on Minix and has been invented by Linus Benedict Torvalds in 1991. The following is an excerpt of a newsgroup, called "comp.os.minix" where Linus posted this text on 08/01/91: "...As I mentioned a month ago, I'm working on a free version of a Minix-look-alike for AT-386 computers. It has finally reached the stage where it's even usable (though may not be, depending on what you want), and I am willing to put out the sources for wider distribution. It is just version 0.02... but I've successfully run bash, gcc, gnu-make, gnu-sed, compress, etc. under it."

Linux is a free version of UNIX that continues to be developed by the cooperative efforts of volunteer groups of programmers, primarily on the Internet, who exchange code, report bug, and fix problems in an open-ended environment. As a result, the world now has a powerful, robust, and full-featured operating system that continues to change and grow.

In other words, Linux is little bit harder to manage than something like Windows, but offers more flexibility and configuration options.

Linux is licensed under the GPL (General Public license) from the GNU organization, under which the kernel is provided with the source code, and is available for free. As a result, Linux is considered to be more secure and stable than closed source or proprietary systems like Windows because anyone can analyse the source code written in the C language and find bugs or add new features. One important point that should be noted is that even though the source is free, anyone is allowed to sell it for profit.

Linux is known as an *open source* operating system and also called *free software* because everything about Linux is accessible to the public and is freely available to anyone. Since the Linux source code is available, anyone can copy, modify, and distribute this software. This allows for various companies such as SuSE, Red Hat, Caldera and others to sell and distribute Linux; however, at the same time, these companies must keep their Linux distribution code open for public inspection, comment, and changes. Despite of the command-line origins of Linux, these distributing companies are working to make the Graphical User Interface (GUI).

The GNU General Public License

To make software free, you need a license that defines the rights and the limits, that have to be regarded by the open source developer that wants to obtain, edit and eventually redistribute your source code. Because of that exists the GNU GPL (General Public License). Of course, there are also other licenses, but today's most open source programs are distributed under this popular license.

The GNU project was started in 1984 and "GNU is recursive acronym for "GNU's Not Unix"; The Free Software Foundation, which stands for the freedom, the security and the protection of free source code therefore founded this kind of license, designed to protect open source code. GNU is also founder and maintainer of many software packages for the Linux operating system, such as basic tools and file system software.

Is Linux Right for you?

It depends on you and what you would like to do. Linux is not an all-purpose operating system and it would probably be more suited for some people and not so pleasing for others. If you are a person using your computer for some entertainment at home and are satisfied with your Windows system there are no compelling reasons for switching over to Linux, but you do have a choice now. There are several other reasons to consider Linux. Linux is not just a simple operating system. It is an entire server and desktop environment, equipped with add-ons, GUI tools and interfaces, and supplementary programs.

You can use Linux at home and even in college to understand the commands and even the internal workings of UNIX systems.

Distributions

When people use the name Linux they are probably referring to a particular distribution of Linux. There are several software packages provided for Linux over the Internet but selecting and downloading one is a complicated task not necessarily manageable for new users who want to try out Linux. This is exactly where a distribution kicks in.

A distribution is a set of software packages that are tested and provided on CD by a company for a small fee just like Windows. The advantages of using distributions are the support and manuals, as well as the fact that Linux can be specialized for use in a particular area. For example, if you would like using Linux for embedded systems a distribution may offer just the right amount of required software, leaving out optional things like the graphical user interface. So you get what you want instead of a general package for all users.

The mainstream distributions, which are seemingly popular, are RedHat, SuSE, Caldera and Debian. Among these distributions RedHat seems to be most widespread.

Caldera is probably more suited for those who are already using Windows. SuSE is a German based distribution known for its large number of bundled packages and support. Debian is unique because its not owned by a company and it’s a non-profit volunteer-based distribution developed solely by users.

Getting Started with Linux

Once the installation is complete, the system will reboot and start up with Linux. There are a series of messages on the screen while booting of the system regarding the hardware enabled, services started etc. After a while, the system will display a login: prompt. You can now log in.

Some systems are configured to start graphical mode with a box in the middle containing both login: and Password: prompts. Press *[CTRL]-[ALT]-[F1]* to switch to the virtual console (text login screen), where you can log in to the system in the usual way.

Accounts and Privileges

Linux is a multi-user system, meaning that many users can use one Linux system simultaneously, from different terminals. So to avoid confusion, each user's workspace must be kept separate from the others.

Even if a particular Linux system is a stand-alone personal computer with no other terminals physically connected to it, it can be shared by different people at different times, making the separation of user workspace is important.

This separation is accomplished by giving each individual user an *account* on the system. You need an account in order to use the system; with an account you are issued an individual workspace to use, and a unique *username* that identifies you to the system and to other users. It is the name along with the password by which the system will recognize the user.

Logging into the System

To begin a session on a Linux system, you need to *log in*. Do this by entering your username at the login: prompt on your terminal, and then entering your password when asked.

Every Linux system has its own name, called the system's *hostname*; a Linux system is sometimes called a *host*, and it identifies itself with its hostname at the login: prompt. It's important to name your system -- like a username for a user account, a hostname gives name to the system you are using (and it becomes especially important when putting the system on a network). The system administrator usually names the system when it is being initially configured (the hostname can be changed later; its name is kept in the file `/etc/hostname'). The name of the terminal you are connecting from is displayed just after the hostname.

To log in to the system, type your username (followed by) at the login: prompt, and then type your password when asked (also followed by); for security purposes, your password is not displayed on the screen when you type it.

Once you've entered your username and password, you are "logged in" to the system. You can then use the system and run commands.

As soon as you log in, the system displays the contents of `/etc/motd', the "Message of the Day" file. The system then displays the time and date of your last login, and reports whether or not you have electronic mail waiting for you. Finally, the system puts you in a *shell*---the environment in which you interact with the system and give it commands. Bash is the default shell on most Linux systems.

The dollar sign (`$') displayed to the left of the cursor is called the *shell prompt*; it means that the system is ready and waiting for input. By default, the shell prompt includes the name of the current directory.

Logging Out of the System

To end your session on the system, type *logout* at the shell prompt. This command logs you out of the system, and a new login: prompt appears on your terminal.

- To log out of the system

  $ *logout*

You can also logout by just pressing *Ctrl+d*.

Logging out of the system frees the terminal you were using and ensures that nobody can access your account from this terminal.

Console Basics

A Linux *terminal* is a place to put input and get output from the system, and usually has at least a keyboard and monitor.

When you access a Linux system by the keyboard and monitor that are directly connected to it, you are said to be using the *console* terminal.

Linux systems feature *virtual consoles*, which act as separate console displays that can run separate login sessions, but are accessed from the same physical console terminal. Linux systems are configured to have seven virtual consoles by default. When you are at the console terminal, you can switch between virtual consoles at any time, and you can log in and use the system from several virtual consoles at once.

Switching Between Consoles

To switch to a different virtual console, press **[ALT]-[F*n*],** where *n* is the number of the console to switch to.

- To switch to the fourth virtual console, press **[ALT]-[F4].**

You can also cycle through the different virtual consoles with the left and right arrow keys. To switch to the next-lowest virtual console, press [ALT]-[←]and to the next-highest virtual console, press **[ALT]-[→].**

- To switch from the fourth to the third virtual console, press **[ALT]-[←]**

The seventh virtual console is reserved for the X Window System. If X is installed, this virtual terminal will never show a login: prompt, but when you are using X, this is where your X session appears. If your system is configured to start X immediately, this virtual console will show an X login screen.

You can switch to a virtual console from the X Window System using [CTRL] in conjunction with the usual [ALT] and function keys. This is the only console manipulation keystroke that works in X.

- To switch from X to the first virtual console, press: [CTRL]-[ALT]-[F1]

Running a Command

A *command* is the name of a tool that performs a certain function along with the options and arguments. Commands are case sensitive.

To run the hostname command just type the command in front of prompt ($)

$ *hostname*

Options always begin with a hyphen character, `-', which is usually followed by one alphanumeric character. Always separate the command, each option, and each argument with a space character.

Long-style options begin with two hyphen characters (`--').

For example, many commands have an option, `--version', to output the version number of the hostname.

$ *hostname --version*

Sometimes, an option itself may take an argument. For example, hostname has an option for specifying a file name to use to read the hostname from, `-F'; it takes as an argument the name of the file that hostname should read from. To run hostname and specify that the file `host.info' is the file to read from

$ *hostname -F host.info*

Changing Your Password

To change your password, use the passwd command. It prompts you for your current password and a new password to replace it with. You must type it exactly the same way both times, or passwd will not change your password.

$ passwd username

Listing Your Username

Use whoami to output the username of the user that is logged in at your terminal.

$ *whoami*

Listing Who Is on the System

Use who to output a list of all the users currently logged in to the system. It outputs a minimum of three columns, listing the username, terminal location, and time of login

for all users on the system. A fourth column is displayed if a user is using the X Window System.

```
$ who
abc    tty1     Oct 20 20:09
def     tty2     Oct 21 14:37
def     ttyp1    Oct 21 15:04 (:0.0)
$
```

The output in this example shows that the user abc is logged in on tty1 (the first virtual console on the system), and has been on since 20:09 on 20 October. The user def is logged in twice -- on tty2 (the second virtual console), and ttyp1, which is an X session with a window location of `(:0.0)'.

Listing the Last Times a User Logged In

Use last to find out who has recently used the system, which terminals they used, and when they logged in and out.

```
$ last abc
```

Listing System Activity

When you run a command, you are starting a *process* on the system, which is a program that is currently executing. Every process is given a unique number, called its *process ID*, or "PID."

Use ps to list processes on the system. By default, ps outputs 5 columns: process ID, the name of the terminal from which the process was started, the current status of the process (including `S' for *sleeping*, meaning that it is on hold at the moment, `R' meaning that it is running, and `Z' meaning that it is a process that has already died), the total amount of time the CPU has spent on the process since the process started, and finally the name of the command being run.

Listing Your Current Processes

Type *ps* with no arguments to list the processes you have running in your current shell session.

```
$ ps
  PID TTY STAT TIME COMMAND
  193   1 S    0:01 -bash
  204   1 S    0:00 ps
$
```

Listing All of a User's Processes

To list all the running processes of a specific user, use ps and give the username to list as an argument with the `-u' option.

```
$ ps -u abc
```

Listing All Processes on the System

To list all processes running by all users on the system, use the `aux' options.

```
$ ps aux
```

Listing Processes by Name or Number

To list processes whose output contains a name or other text to match, list all processes and pipe the output to grep. This is useful for when you want to see which users are running a particular program or command.

To list all the processes whose commands contain reference to an `sbin' directory in them

$ *ps aux | grep sbin*

To list any processes whose process IDs contain a 13 in them

$ *ps aux | grep 13*

To list the process, which corresponds to a process ID, give that PID as an argument to the `-p' option (PID is 344)

$ ps -p 344

Finding the System Manual of a Command

Use the man command to view a page in the system manual. As an argument to man, give the name of the program whose manual page you want to view.

$ *man ps*

Use the up and down arrow keys to move through the text. Press [Q] to stop viewing the manual page and exit man.

Working with Shell

S*hell* is a program that reads your command input and runs the specified commands. The shell environment is the most fundamental way to interact with the system -- you are said to be in a shell from the very moment you've successfully logged in to the system.

The `$' character preceding the cursor is called the *shell prompt*; it tells you that the system is ready and waiting for input.

If your shell prompt shows a number sign (`#') instead of a `$', this means that you're logged in with the superuser, or root, account. Beware: the root account has complete control over the system; one wrong keystroke and you might accidentally break it something awful. You need to have a different user account for yourself, and use that account for your regular use.

Every Linux system has at least one shell program, and most have several. The standard shell on most Linux systems is bash("Bourne again shell").

Running a List of Commands

To run more than one command on the input line, type each command in the order you want them to run, separating each command from the next with a semicolon (`;'). For example, to clear the screen and then log out of the system

$ *clear; logout*

Redirecting Input and Output

The shell moves text in designated "streams." The *standard output* is where the shell streams the text output of commands -- the screen on your terminal, by default. The

standard input, typically the keyboard, is where you input data for commands. You can redirect these streams -- to a file, or even another command -- with *redirection*.

Redirecting Input to a File

To redirect standard input to a file, use the `<' operator. To do so, follow a command with < and the name of the file it should take input from. For example, to redirect standard input for ls -l to file `listing'

$ ls -l < listing

Redirecting Output to a File

Use the `>' operator to redirect standard output to a file. If you redirect standard output to an existing file, it will overwrite the file, unless you use the `>>' operator to *append* the standard output to the contents of the existing file. For example, to append the standard output of *ls -l* to an existing file `commands'

$ *ls -l>> commands*

Redirecting Output to another Command's Input

Piping is to connect the standard output of one command to the standard input of another. You do this by specifying the two commands in order, separated by a vertical bar character, `|' (also called as a "pipe"). Commands built in this fashion are called *pipelines*.

For example, it's often useful to pipe commands that display a lot of text output to more for perusing text.To pipe the output of apropos bash shell shells to less

$ *ls –l | more*

Managing Jobs

The processes you have running in a particular shell are called your *jobs*. You can have more than one job running from a shell at once, but only one job can be active at the terminal, reading standard input and writing standard output. This job is the *foreground* job, while any other jobs are said to be running in the *background*.

The shell assigns each job a unique *job number*. Use the job number as an argument to specify the job to commands. Do this by giving the job number preceded by a `%' character.

Suspending a Job

Type *Ctrl+z* to suspend or stop the foreground job. This is useful when you want to do something else in the shell and return to the current job later. The job stops until you either bring it back to the foreground or make it run in the background.

For example, if you are finding a file at Linux partition from root (/), typing *Ctrl+z* will suspend the find program and return you to a shell prompt where you can do something else. The shell outputs a line giving the job number (in brackets) of the suspended job, the text `Stopped' to indicate that the job has stopped, and the command line itself, as shown here:

[1]+ Stopped find / -name abc

In this example, the job number is 1 and the command that has stopped is `find / -name abc'. The `+' character next to the job number indicates that this is the most recent job.

If you have any stopped jobs when you log out, the shell will tell you this instead of logging you out:

$ *logout*

There are stopped jobs.

$

At this point you can list your jobs, stop any jobs you have running and then log out.

Putting a Job in the Background

New jobs run in the foreground unless you specify otherwise. To run a job in the background, end the input line with an ampersand (`&'). This is useful for running non-interactive programs that perform a lot of calculations. To run the command find / -name abc > shell-commands as a background job

$ *find / -name abc > shell-commands &*

[1] 6575

$

The shell outputs the job number (in this case, 1) and process ID (in this case, 6575), and then returns to a shell prompt. When the background job finishes, the shell will list the job number, the command, and the text `Done', indicating that the job has completed successfully:

[1]+ Done find / -name abc >shell-commands

To move a job from the foreground to the background, first suspend it and then type *bg* (for "background").

- For example, to start the command *find / -name abc > shell-commands* in the foreground, suspend it, and then specify that it finish in the background, you would type:

  $ *find / -name abc > shell-commands*

  Ctrl+z

  [1]+ Stopped find / -name abc >shell-commands

  $ *bg*

  [1]+ find / -name abc &

  $

If you have suspended multiple jobs, specify the job to be put in the background by giving its job number as an argument. TFor example, to run job 4 in the background

$ *bg %4*

Putting a Job in the Foreground

Type *fg* to move a background job to the foreground. By default, fg works on the most recent background job. For example, to bring the most recent background job to the foreground

$ *fg*

To move a specific job to the foreground when you have multiple jobs in the background, specify the job number as an option to fg. To bring job 3 to the foreground

$ *fg %3*

Listing Your Jobs

To list the jobs running in the current shell, type *jobs*.

$ *jobs*

[1]- Stopped find / -name abc >shell-commands

[2]+ Stopped find / -name abc >bash-commands

$

This example shows two jobs--- *find / -name abc > shell-commands* and *find / -name abc > bash-commands*. The `+' character next to a job number indicates that it's the most recent job, and the `-' character indicates that it's the job *previous* to the most recent job. If you have no current jobs, jobs returns nothing.

Stopping a Job

Typing *Ctrl+c* interrupts the foreground job before it completes, exiting the program. To interrupt *cat*, a job running in the foreground

$ *cat*

Ctrl+c

$

Use kill to interrupt ("kill") a background job, specifying the job number as an argument. To kill job number 2

$ *kill %2*

Command History

Your command *history* is the sequential list of commands you have typed, in the current or previous shell sessions. The commands in this history list are called *events*.

By default, bash remembers the last 500 events, but this number is configurable.

Your command history is stored in a text file in your home directory called `.bash_history'; you can view this file or edit it like you would any other text file.

Viewing Your Command History

Use history to view your command history. To view your command history

$ *history*

1 who

2 apropos shell >shell-commands

3 apropos bash >bash-commands

4 history

$

This command shows the contents of your command history file, listing one command per line prefaced by its *event number*. Use an event number to specify that event in your history. To search your history for the text `find'

$ *history | grep find*

Specifying a Command from Your History

You can specify a past event from your history on the input line, in order to run it again.

The simplest way to specify a history event is to use the up and down arrow keys at the shell prompt to browse your history. The up arrow key takes you back through past events, and the down arrow key moves you forward into recent history. When a history event is on the input line, you can edit it as normal, and type to run it as a command; it will then become the newest event in your history.

To run a history event by its event number, enter an exclamation mark (`!') followed by the event number (1).

$ *!1*

NGS Data Generation: Techniques, Issues and Challenges

Dwijesh Chandra Mishra

Introduction

DNA sequencing is a biochemical method in order to determine the correct order of nucleotide bases in a DNA macromolecule by using sequencing machines. Earlier sequencing was based on a single type of method that is Sanger sequencing. In 2005, Next Generation Sequencing (NGS) Technologies emerged and changed the view of the analysis and understanding of living beings. Over the last two decade, considerable progress has been made on new sequencing methods. NGS is modern high-throughput DNA sequencing technologies. They are faster, cheaper, rapid and parallel. They require much less template preparation than the Sanger sequencing technique.

First Generation Sequencing

1. Dideoxy method or chain termination method:

This method is developed by Sanger and Coulson in 1977. In this method one strand of the double stranded DNA is used as template to be sequenced. This sequencing is based on using chemically modified nucleotides called dideoxy-nucleotides (ddNTPs). The dideoxynucleotides are used in elongation of DNA complementary strand, once incorporated into the DNA strand they prevent the further elongation. The sequencing reaction is carried out in four test tubes which consist of various components besides the templates. These components are a small stretch of DNA sequence called primer, DNA polymerase enzyme, a mixture of four deoxy nucleotide triphosphate (A, T, G, and C) and one of the dideoxy nucleotide, i.e. either ddATP, ddTTP, ddGTP or ddCTP labeled with radioactive substances or non-radioactive substances like dig or biotin. The synthesis of new DNA strand continues in the presence of DNA polymerase enzyme until a dideoxynucleotide is added in the complementary DNA strand which results in the generation of different sized DNA fragments, ending with labeled ddNTPs. After the reaction is complete the reaction mixture of all the four tubes are loaded adjacent to each other on a polyacrylamide sequencing gel. The four lanes specific to ddATP, ddCTP, ddGTP and ddTTP produce fragments of varying length upon electrophoresis and autoradiography. The position of bands in the gel is used to directly read DNA sequences from bottom to top. The automated version of this method uses ddNTPs that are labeled with different color fluorescent dyes so that all four reactions can be run in a single tube.

2. Chemical cleavage method of DNA sequencing:

This method is developed by Maxam and Gilbert in 1977. This method uses chemicals to break DNA molecules of specific bases, thus creating fragments of different sizes. DNA molecule to be sequenced is radiolabeled. The sequencing reaction is devised into four tubes along with a fifth reference tube.

Chemicals	**Reaction**
Dimethyl sulphate	Alters guanine at N7 position by methylation
Acid	Alters either adenine or guanine
Hydrazine	Alters either thymine or cytosine
Hydrazine + NaCl	Alters cytosine
NaOH	Reference

For removing altered basepairs from the sequencing reaction, piperidine is added in each tube. Piperidine breaks the DNA molecules at the sugar residue from the point of altered nucleotide thus making different sized fragments of DNA. The mixture of DNA fragments are separated on high resolution polyacrylamide gels by loading the contents of all the four tubes in adjacent lanes. After electrophoresis, the gels are exposed to x-ray film for developing autoradiographs of the DNA bands from which sequence is read.

Second Generation Sequencing

The first generation of sequencing especially Sanger sequencing was extensively used for three decades, however, the cost and time was a major drawback. In 2005, second generation sequencing technologies came into the market which eliminates the limitations of the first generation sequencing. The basic characteristics of second generation sequencing technology are: (1) The generation of many millions of short reads in parallel, (2) The speed up of sequencing the process compared to the first generation, (3) The low cost of sequencing and (4) The sequencing output is directly detected without the need for electrophoresis. Short read sequencing approaches divided under two wide approaches: sequencing by ligation (SBL) and sequencing by synthesis (SBS).

1. 454/Roche:

Roche/454 sequencing appeared on the market in 2005, using pyrosequencing technique which is based on the detection of pyrophosphate, released after each nucleotide incorporation in the new synthetic DNA strand (http://www.454.com). The pyrosequencing technique is a sequencing-by-synthesis approach. DNA samples are randomly fragmented and each fragment is attached to a bead whose surface carries primers that have oligonucleotides complementary to the DNA fragments so each bead is associated with a single fragment. Then, each bead is isolated and amplified using PCR emulsion which produces about one million copies of each DNA fragment on the surface of the bead. The beads are then transferred to a plate containing many wells called picotiter plate (PTP) and the pyrosequencing technique is applied which consists in activating of a series of downstream reactions producing light at each incorporation of nucleotide. By detecting the light emission after incorporation of each nucleotide, the sequence of the DNA fragment is deduced. The use of the picotiter plate allows hundreds of thousands of reactions occur in parallel, considerably increasing sequencing throughput. The latest instrument launched by Roche/454 called GS FLX+ that generates reads with lengths of up to 1000 bp and can produce ~1Million reads per run. The Roche/454 is able to generate relatively long reads which are easier to map to a reference genome. The main errors detected of sequencing are insertions and deletions due to the presence of homopolymer regions. Indeed, the identification of the size of homopolymers should be determined by the intensity of the light emitted by pyrosequencing. Signals with too high or too low intensity lead to under or overestimation of the number of nucleotides which causes errors of nucleotides identification.

2. Illumina/ Solexa:

Illumina technology is sequencing by synthesis approach and is currently the most used technology in the NGS market. During the first step, the DNA samples are randomly fragmented into sequences and adapters are ligated to both ends of each sequence. Then, these adapters are fixed themselves to the respective complementary adapters, the latter are hooked on a slide with many variants of adapters (complementary) placed on a solid plate. During the second step, each attached sequence to the solid plate is amplified by PCR bridge amplification that creates several identical copies of each sequence. A set of sequences made from the same original sequence is called a cluster. Each cluster contains approximately one million copies of the same original sequence. The last step is to determine each nucleotide in the sequences, Illumina uses the sequencing by synthesis approach that employs reversible terminators in which the four modified nucleotides, sequencing primers and DNA polymerases are added as a mix, and the primers are hybridized to the sequences. Then, polymerases are used to extend the primers using the modified nucleotides. Each type of nucleotide is labeled with a fluorescent specific in order for each type to be unique. The nucleotides have an inactive 3'-hydroxyl group which ensures that only one nucleotide

is incorporated. Clusters are excited by laser for emitting a light signal specific to each nucleotide, which will be detected by a coupled-charge device (CCD) camera and Computer programs will translate these signals into a nucleotide sequence. The process continues with the elimination of the terminator with the fluorescent label and the starting of a new cycle with a new incorporation.

3. ABI/SOLiD:

Sequencing by Oligonucleotide Ligation and Detection (SOLiD) is a NGS sequencer Marketed by Life Technologies (http:// www.lifetechnologies.com). In 2007, Applied Biosystems (ABI) has acquired SOLiD and developed ABI/SOLID sequencing technology that adopts the ligation (SBL) approach. The ABI/SOLiD process consists of multiple sequencing rounds. It starts by attaching adapters to the DNA fragments, fixed on beads and cloned by PCR emulsion. These beads are then placed on a glass slide and the 8-mer with a fluorescent label at the end is sequentially ligated to DNA fragments, and the color emitted by the label is recorded. Then, the output format is color space which is the encoded form of the nucleotide where four fluorescent colors are used to represent 16 possible combinations of two bases. The sequencer repeats this ligation cycle and each cycle the complementary strand is removed and a new sequencing cycle starts at the position n-1 of the template. The cycle is repeated until each base is sequenced twice. The recovered data from the color space can be translated to letters of DNA bases and the sequence of the DNA fragment can be deduced.

4. Ion Torrent:

Life Technologies commercialized the Ion Torrent semiconductor sequencing technology in 2010 (https//www.thermofisher.com/us/en/home/brands/ion-torrent.html). It is similar to 454 pyrosequencing technology but it does not use fluorescent labeled nucleotides like other second-generation technologies. It is based on the detection of the hydrogen ion released during the sequencing process. First, emulsion PCR is used to clonally amplify adapter ligated DNA fragments on the surface of beads. The beads are subsequently distributed into micro-wells where sequencing by synthesis reaction occurs. Ion torrent chip consists of a flow compartment and solid state pH sensor micro-arrayed wells that are manufactured using processes built on standard complementary metal oxide semiconductor (CMOS) technology. The release of H+ during extension of each nucleotide is detected as a change in the pH within the sensor wells. Since there is no detectable difference for H+ released from a A, T, G or C bases, the individual dNTPs are applied in multiple cycles of consecutive order. The speed of sequencing is 2-8 hrs depending on the machine and chip used. Error rate for substitutions is ~0.1%, similar to Illumina. Homopolymer repeats more than 6bp lead to increased error rates.

Third Generation Sequencing

The second generation sequencing technologies generally require PCR amplification step which is a long and expensive procedure. Also, it became clear that the genomes are very complex with many repetitive areas that second generation sequencing technologies are incapable to solve them and the relatively short reads made genome assembly more difficult. Third generation sequencing technologies are remedy to these problems. These third generations of sequencing have the ability to cover a low sequencing cost and easy sample preparation without the need PCR amplification. The execution time reduces significantly than second generation sequencing technologies. The most widely used third generation sequencing technology approach is SMRT (Pacific Biosciences) and Oxford Nanopore sequencing.

1. Pacific biosciences SMRT sequencing

Pacific Biosciences (http//www.pacificbiosciences.com/) developed the first genomic sequencer using SMRT approach and it's the most widely used third-generation sequencing technology. Template preparation involves ligation of single stranded hairpin adapters onto the ends of digested DNA or cDNA molecules, generating a capped template called SMRT-bell. This technology works with single molecule

detection which does not require any amplification step. By using a strand displacing polymerase, the original DNA molecule can be sequenced multiple times, thereby increasing accuracy.

DNA synthesis occurs in zeptoliter sized chambers, called zero-mode waveguides (ZMWs). These ZMW are small reaction wells that each ideally contains one complex consisting of template molecule, sequencing primer and DNA polymerase bound to the bottom of the ZMW. The fluorescent signals of the extended nucleotides are recorded in real time at 75 frames per second for the individual ZMWs. This is achieved by powerful optical system that illuminates the individual ZMWs with red and green laser beamlets from the bottom of the SMRT cell and a parallel confocal recording system to detect the signal from the fluorescent nucleotides. When a nucleotide complementary to the template is bound in position by the polymerase within the illumination zone of the zmw, the identity of the nucleotide is recorded by its fluorescent label. Each SMRT cell produces ~50k reads and upto 1 gb of data in 4 hrs. The average read length is >14 kb. This technology has a high error rate of approximately 11%. This is useful for denovo assembly of small bacterial and viral genomes as well as large genome finishing.

2. Oxford nanopore sequencing

This technology is also based on single molecule strategy. This relies on the transition of DNA or individual nucleotides through a small channel called protein nanopore. A nanopore is a nanoscale hole made of proteins or synthetic materials. A sequencing flow cell comprises hundreds of independent micro-wells, each containing a synthetic bilayer perforated by biological nanopores. Sequencing is accomplished by measuring characteristic changes in ionic current that are induced as the bases are threaded through the pore by a molecular motor protein. Library preparation is minimal, involving fragmentation of DNA and ligation of adapters. The first adapter is bound with a motor enzyme as well as a molecular tether, whereas second adapter is a hairpin oligonucleotide that is bound by a second motor protein. This library design allows sequencing of both strands of DNA from a single molecule, which increases accuracy. The variation in ionic current is recorded progressively on a graphic model and then interpreted to identify the sequence. MinION is released in 2014 which generate longer reads, ensure better resolution structural genomic variants and repeat content.it is a mobile single –molecule nanopore sequencing measures connected by a USB 3.0 port of a laptop computer. PromethION is bigger than MinION, equivalent to 48 MinIONs.

References:

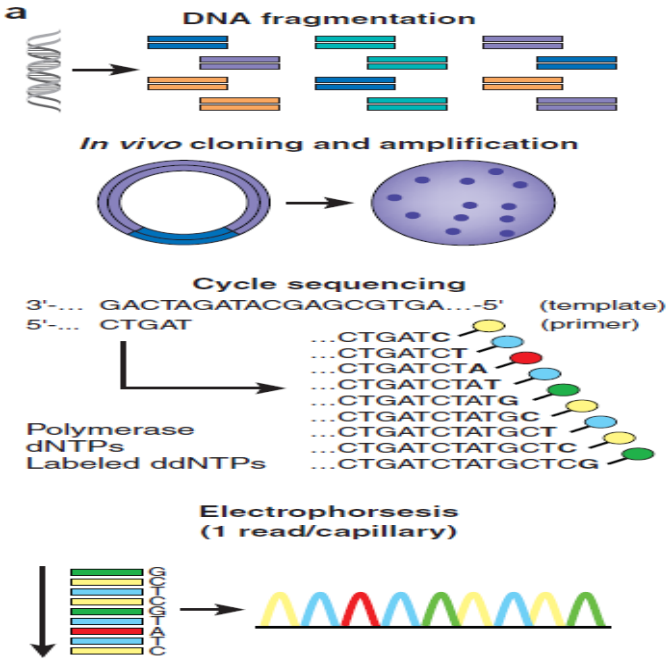
- Kchouk M, Gibrat JF, Elloumi M. (2017). Generations of Sequencing Technologies: From First to Next Generation. Biol Med (Aligarh) 9: 395. doi:10.4172/0974-8369.1000395.
- H.P.J. Buermans, J.T. den Dunnen. (2014). Next generation sequencing technology: Advances and applications, Biochimica et Biophysica Acta (BBA) - Molecular Basis of Disease, 1842(10): 1932-1941. https://doi.org/10.1016/j.bbadis.2014.06.015.

NGS Data: Pre-processing, Assembly and Quantification

Dwijesh Chandra Mishra, Sanjeev Kumar and Neeraj Budhlakoti

Sanger Sequencing

- DNA is fragmented
- Cloned to a plasmid vector
- Cyclic sequencing reaction
- Separation by electrophoresis
- Readout with fluorescent tags



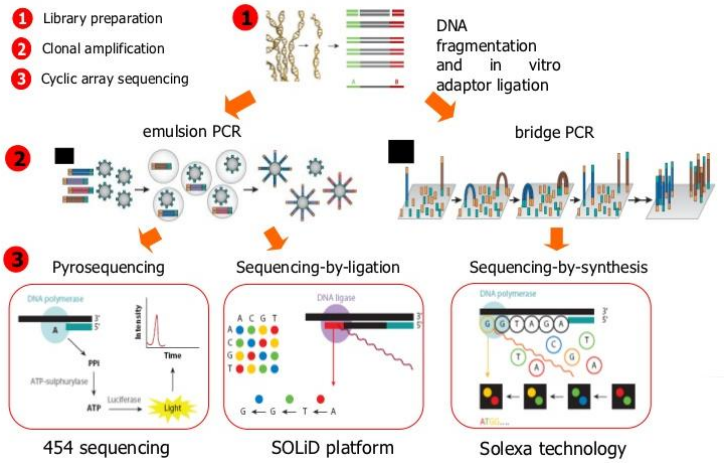
Sanger Vs NGS

- 'Sanger sequencing' has been the only DNA sequencing method for 30 years but…
- …hunger for even greater sequencing throughput and more economical sequencing technology…
- NGS has the ability to process millions of sequence reads in parallel rather than 96 at a time (1/6 of the cost)

NGS Platforms: Different sequencing techniques used for next generation sequencing are:

- Roche/454 FLX: 2004
- Illumina Solexa Genome Analyzer: 2006
- Applied Biosystems SOLiD™ System: 2007
- Helicos Heliscope™ : 2009
- Pacific Biosciencies SMRT: 2010

General Experimental Procedure

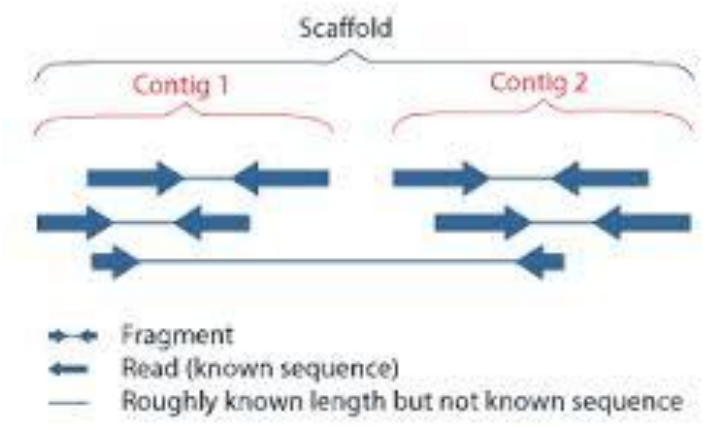


Sequencing Technology at a Glance

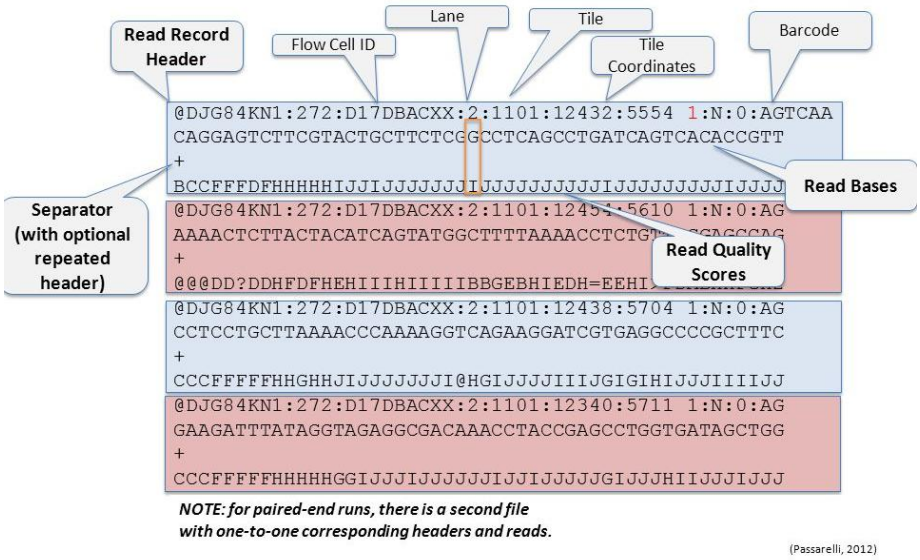
Method	Read length	Accuracy	Time per run	Cost per 1 million bases	Advantages	Disadvantages
Chain termination (Sanger sequencing)	400 to 900 bp	99.9%	20 minutes to 3 hours	Rs 144000	Long individual reads. Useful for many applications.	More expensive and impractical for larger sequencing projects.
Pyrosequencing (454)	700 bp	99.9%	24 hours	Rs 600	Long read size. Fast	Runs are expensive. Homopolymer errors.
Sequencing by synthesis (Illumina)	50 to 300 bp	98%	1 to 10 days, depending upon sequencer and specified read length	Rs 3 to 9	Potential for high sequence yield, depending upon sequencer model and desired application.	Equipment can be very expensive. Requires high concentrations of DNA.
Sequencing by ligation (SOLiD sequencing)	50+35 or 50+50 bp	99.9%	1 to 2 weeks	Rs 78	Low cost per base.	Slower than other methods. Have issue sequencing palindromic sequence.
Single-molecule real-time sequencing (Pacific Bio)	10,000 bp to 15,000 bp avg. (14,000 bp);	87%	30 minutes to 4 hours	Rs 7.8–36	Longest read length. Fast.	Moderate throughput. Equipment can be very expensive.

Reads, Contigs and Scaffolds

- Reads are what you start with (35bp-800bp)
- Fragmented assemblies produce contigs that can be kilobases in length
- Putting contigs together into scaffolds is the next step



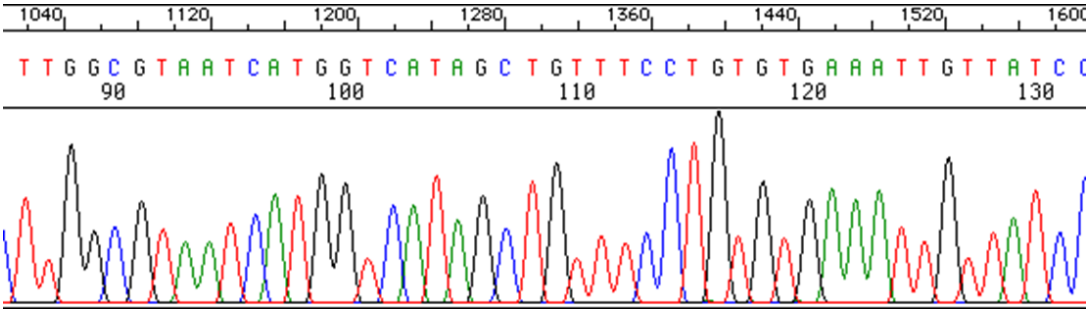
FASTQ Format



Before Assembly

Fragment readout

- DNA characters in a fragment are determined from chromatogram
- Base call is a DNA character that is determined from chromatogram



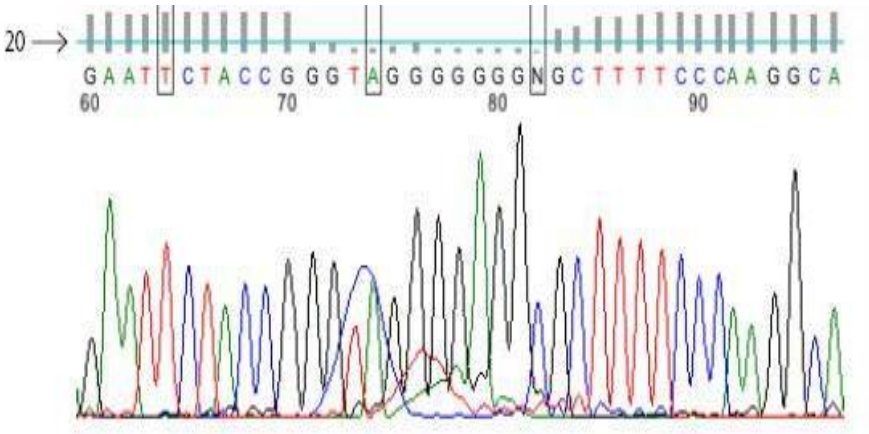
Fragment readout

- Phred Score- determine the quality value of a base

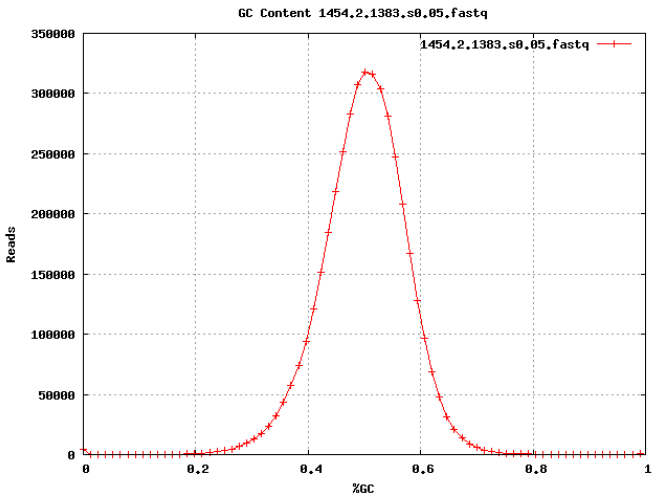
$$q = -10 \times log_{10}(\mathrm{p})$$

  where p is the estimated error probability for the base

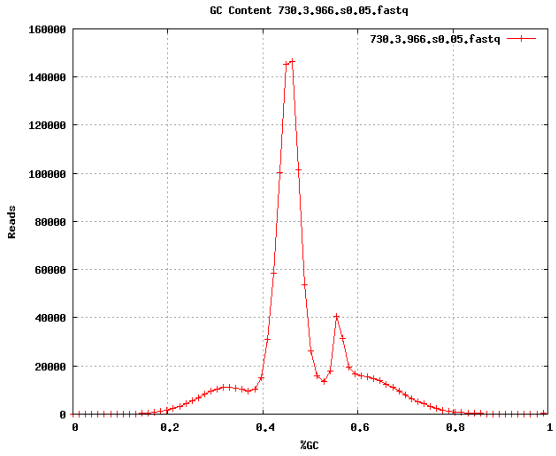
- if Phred assigns a quality score of 30 to a base, the chances that this base is called incorrectly are 1 in 1000
- The most commonly used method is to count the bases with a quality score of 20 and above
- Phred Score



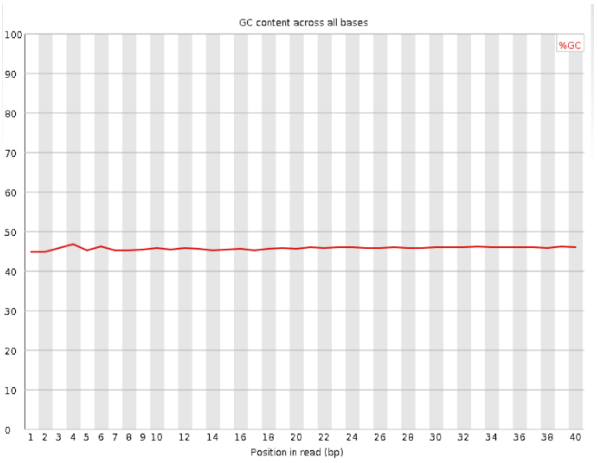
Genome Properties



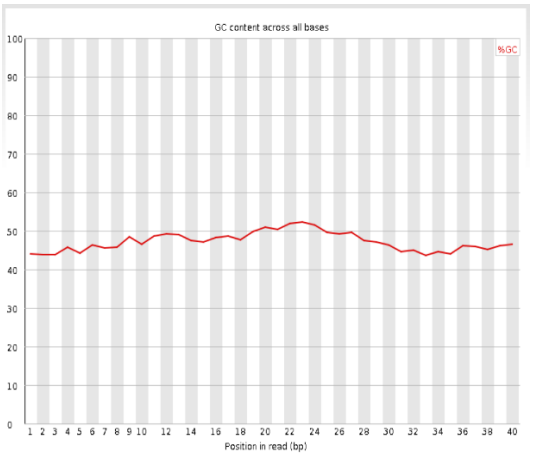
PASS



FAIL

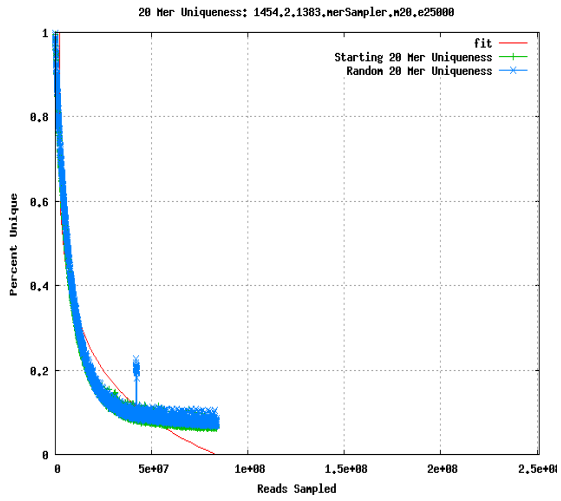


PASS

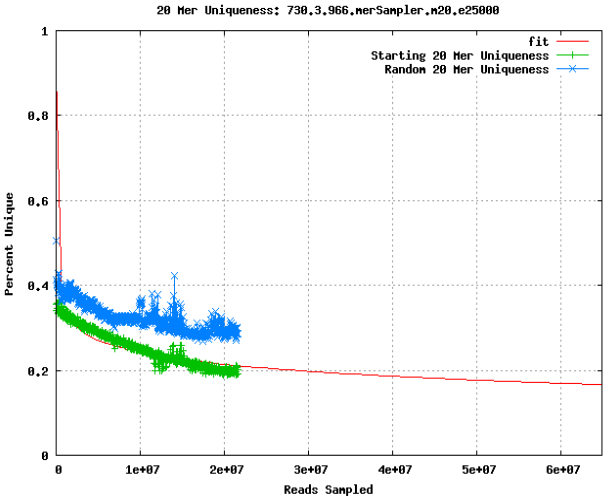


FAIL

Library Quality

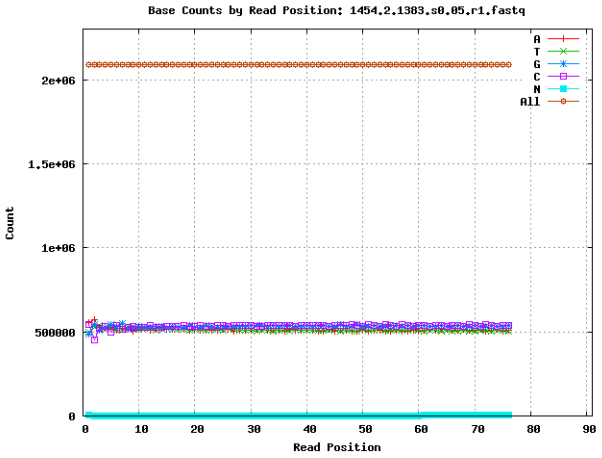


PASS

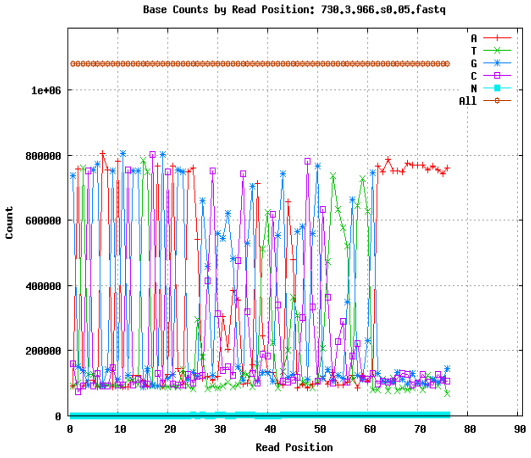


FAIL

Run Quality

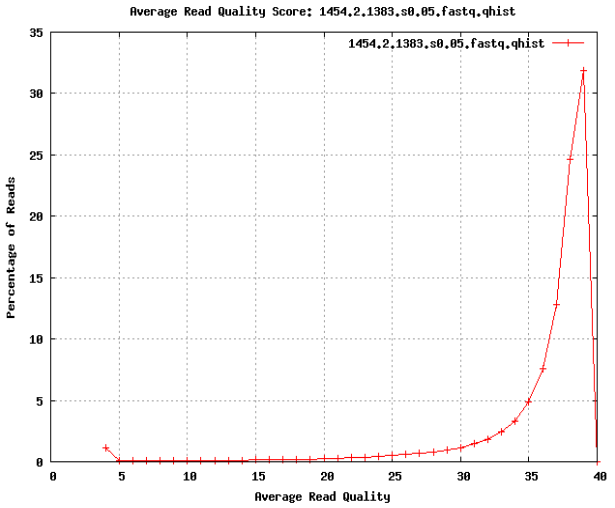


PASS

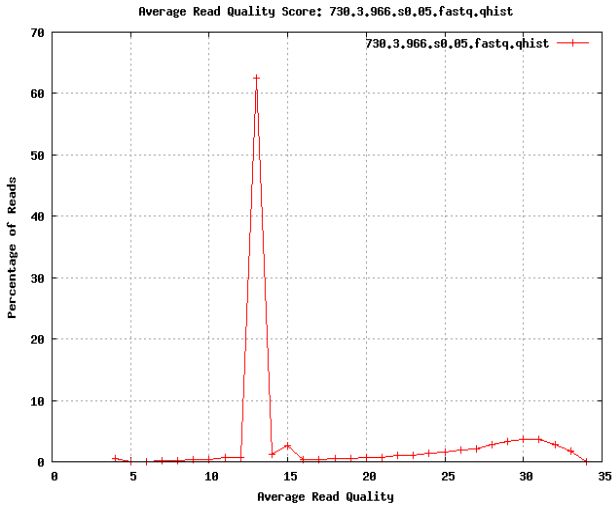


FAIL

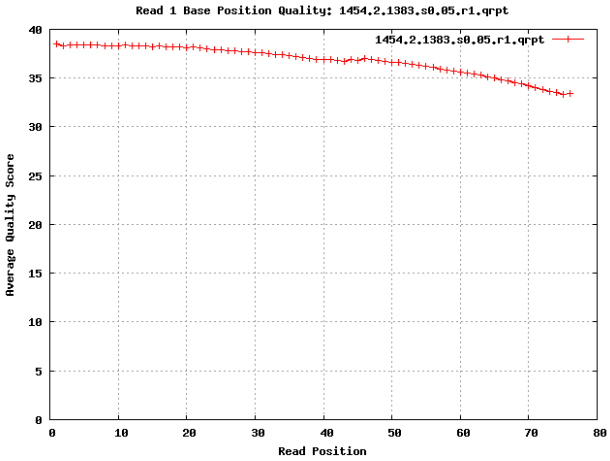
Read Quality



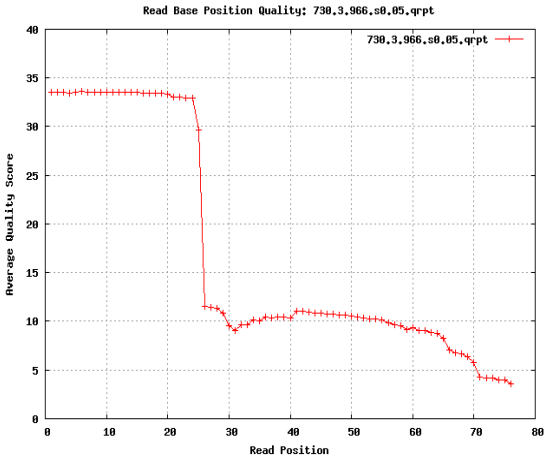
PASS



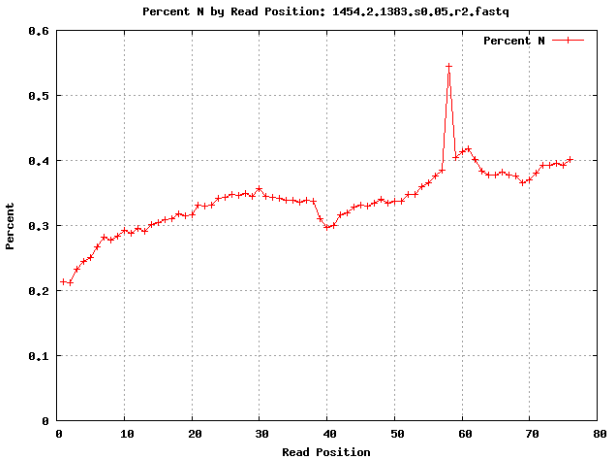
FAIL



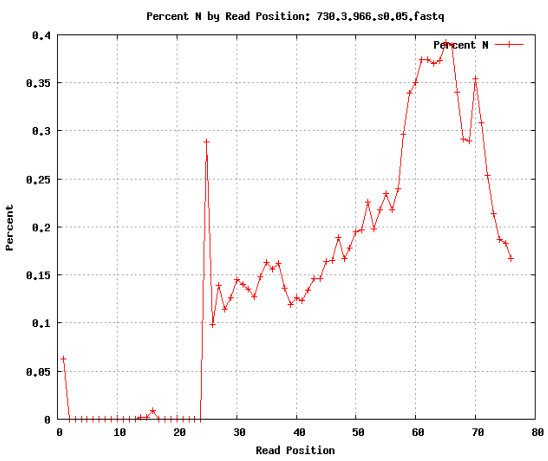
PASS



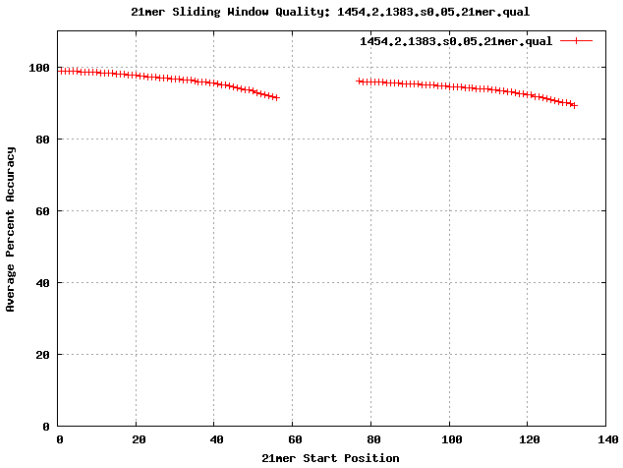
FAIL



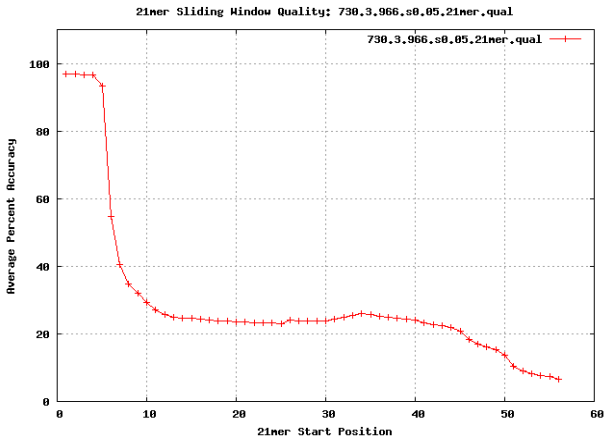
PASS



FAIL



PASS



FAIL

Trimming

- Trimming low-quality sequences

  -removal of reads containing poor quality base calls
- Trimming vector sequences

  -removal of reads containing vector sequences

Annotation of RNA-Seq Data

Sneha Murmu, Sanjeev Kumar, D. C. Mishra and Jyotika Bhati

Introduction

Until the genome revolution, genes were identified by researchers with specific interests in a particular protein or cellular process. Once identified, these genes were isolated, typically by cloning and sequencing cDNAs, usually followed by targeted sequencing of the longer genomics segments that code for the cDNAs. Once an organism's entire genome sequence becomes available, there is strong motivation for finding all the genes encoded by a genome at once rather than in a piecemeal approach. Such catalogue is immensely valuable to researchers, as they can learn much more from the whole picture than from a much more limited set of genes. For example, genes of similar sequence can be identified, evolutionary and functional relationships can be elucidated, and a global picture of how many and what types of genes are present in a genome can be seen. A significant portion of the effort in genome sequencing is devoted to the process of *annotation*, in which genes, regulatory elements, and other features of the sequence are identifies as thoroughly as possible and catalogued in a standard format in public databases so that researchers can easily use the information. Functional genomics research has expanded enormously in the last decade and particularly the plant biology research community. Functional annotation of novel DNA sequences is probably one of the top requirements in functional genomics as this holds, to a great extent, the key to the biological interpretation of experimental results.

Computational Gene Prediction

Computational gene prediction is becoming more and more essential for the automatic analysis and annotation of large uncharacterized genomic sequences. In the past two decades, many algorithms have been evolved to predict protein coding regions of the DNA sequences. They all have in common, to varying degree, the ability to differentiate between gene features like Exons, Introns, Splicing sites, Regulatory sites etc. Gene prediction methods predicts coding region in the query sequences and then annotates the sequences databases.

Gene Structure and Expression

The gene structure and the gene expression mechanism in eukaryotes are far more complicated than in prokaryotes. In typical eukaryotes, the region of the DNA coding for a protein is usually not continuous. This region is composed of alternating stretches of *exons* and *introns*. During transcription, both exons and introns are transcribed onto the RNA, in their linear order. Thereafter, a process called *splicing* takes place, in which, the intron

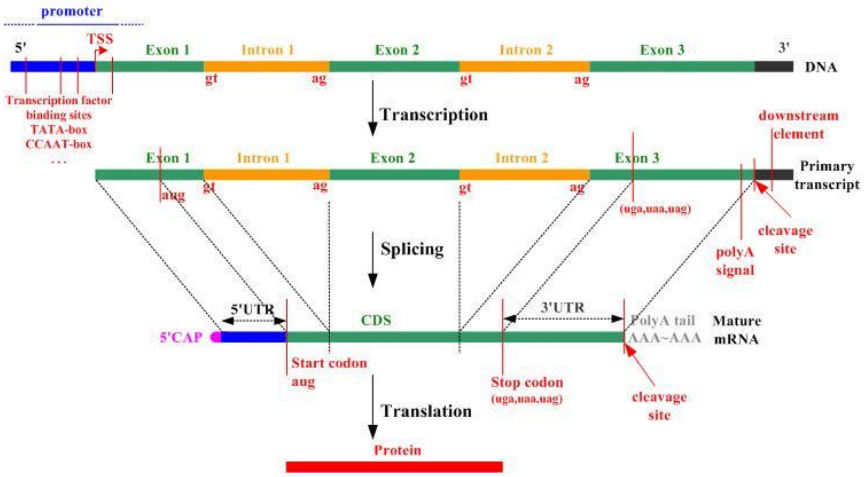


Fig. 1: Representative Diagram of Protein Coding Eukaryotic Gene

sequences are excised and discarded from the RNA sequence. The remaining RNA segments, the ones corresponding to the exons are ligated to form the mature RNA strand. A typical multi-exon gene has the following structure (as illustrated in Fig. 1). It starts with the promoter region, which is followed by a transcribed but non-coding region called *5' untranslated region (5' UTR)*. Then follows the initial exon which contains the start codon. Following the initial exon, there is an alternating series of introns and internal exons, followed by the terminating exon, which contains the stop codon. It is followed by another non-coding region called the *3' UTR*. Ending the eukaryotic gene, there is a polyadenylation (polyA) signal: the nucleotide Adenine repeating several times. The exon-intron boundaries (i.e., the splice sites) are signalled by specific short (2bp long) sequences. The 5'(3') end of an intron (exon) is called the *donor* site, and the 3'(5') end of an intron (exon) is called the *acceptor* site. The problem of gene identification is complicated in the case of eukaryotes by the vast variation that is found in gene structure.

Gene Prediction Methods

There are mainly two classes of methods for computational gene prediction (Fig. 2). One is based on sequence similarity searches, while the other is gene structure and signal-based searches, which is also referred to as Ab initio gene finding.

Sequence Similarity Searches

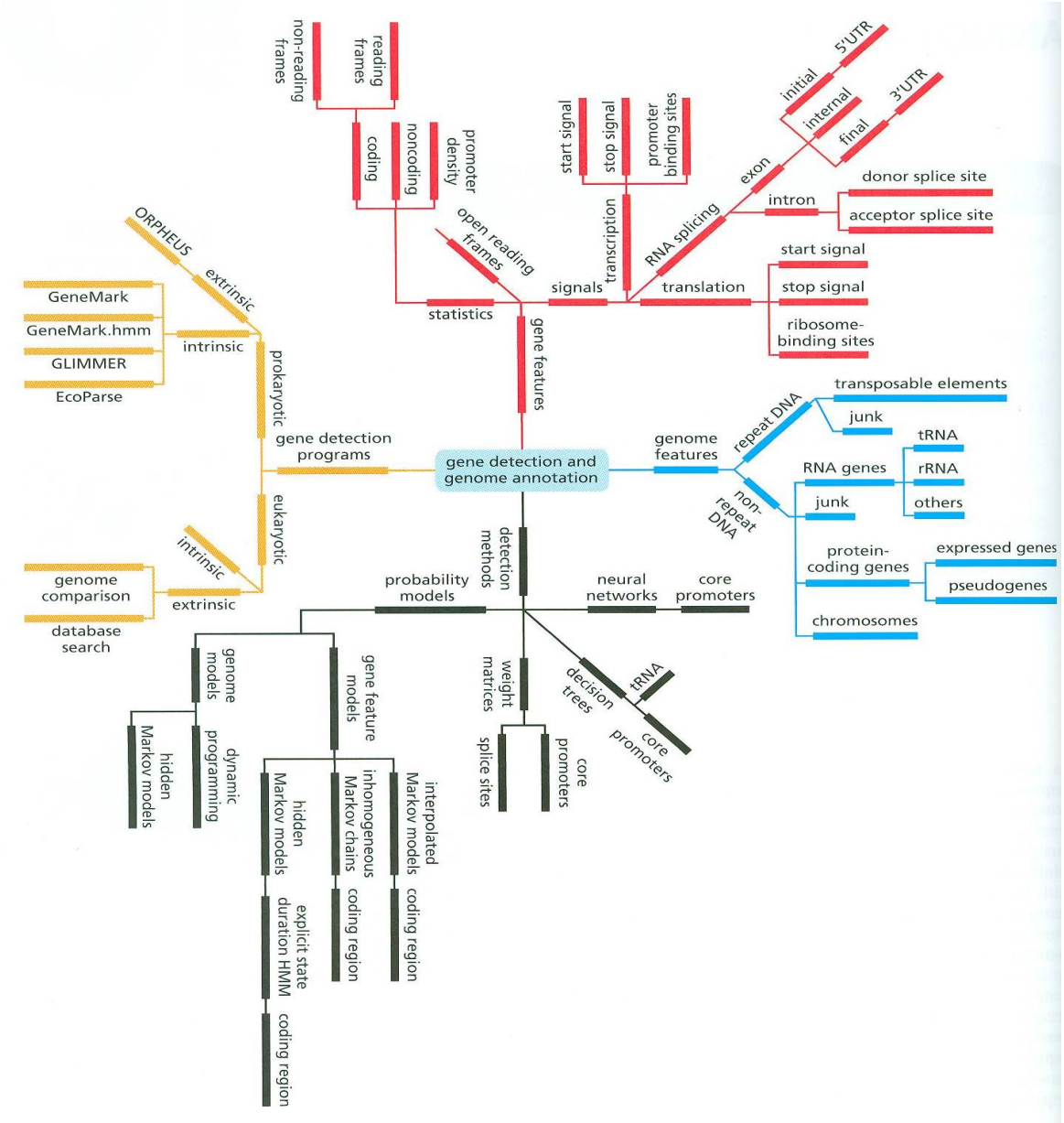
Sequence similarity search is a conceptually simple approach that is based on finding similarity in gene sequences between ESTs (expressed sequence tags), proteins, or other genomes to the input genome. This approach is based on the assumption that functional regions (exons) are more conserved evolutionarily than non-functional regions (intergenic or intronic regions). Once there is similarity between a certain genomic region and an EST, DNA, or protein, the similarity information can be used to infer gene structure or function of that region. EST-based sequence similarity usually has drawbacks in that ESTs only correspond to small portions of the gene sequence, which means that it is often difficult to predict the complete gene structure of a given region. Local alignment and global alignment are two methods based on similarity searches. The most common local alignment tool is the BLAST family of programs, which detects sequence similarity to known genes, proteins, or ESTs. The biggest limitation to this

type of approaches is that only about half of the genes being discovered have significant homology to genes in the databases.

Ab initio Gene Prediction Methods

The second class of methods for the computational identification of genes is to use gene structure as a template to detect genes, which is also called *ab initio* prediction. *Ab initio* gene predictions rely on two types of sequence information: signal sensors and content sensors. Signal sensors refer to short sequence motifs, such as splice sites, branch points, poly pyrimidine tracts, start codons and stop codons. Exon detection must rely on the content sensors, which refer to the patterns of codon usage that are unique to a species, and allow coding sequences to be distinguished from the surrounding non-coding sequences by statistical detection algorithms.

Many algorithms are applied for modeling gene structure, such as Dynamic Programming,



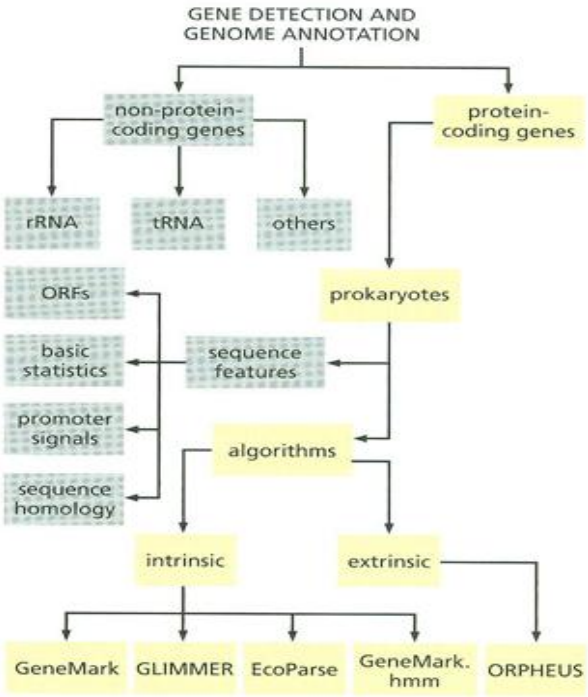
linear discriminant analysis, Linguist methods, Hidden Markov Model and Neural Network.

Based on these models, a great number of *ab initio* gene prediction programs have been developed.

Fig. 2: Diagrammatic Representation of Gene Prediction and Annotation

Gene Discovery in Prokaryotic Genomes

Discovery of genes in Prokaryote is relatively easy, due to the higher gene density typical of prokaryotes and the absence of introns in their protein coding regions. DNA sequences that encode proteins are transcribed into mRNA, and the mRNA is usually translated into proteins without significant modification. The longest ORFs (open reading frames) running from the first available start codon on the mRNA to the next stop codon in the same reading frame generally provide a good, but not assured prediction of the protein coding regions. Several methods have been devised that use different types of Markov models in order to capture the compositional differences among coding regions, "shadow" coding regions (coding on the opposite DNA strand), and noncoding DNA. Such methods, including ECOPARSE, the widely used GENMARK, and Glimmer program, appear to be able to identify most protein coding



genes with good performance (Fig. 3).

Fig. 3: Flow Diagram of Prokaryotic Gene Discovery

Gene Discovery in Eukaryotic Genome

It is a quite different problem from that encountered in prokaryotes. Transcription of protein coding regions initiated at specific promoter sequences is followed by removal of noncoding sequences (introns) from pre-mRNA by a splicing mechanism, leaving the protein encoding exons. Once the introns have been removed and certain other modifications to the mature RNA have been made, the resulting mature mRNA can be translated in the 5` to 3` direction, usually from the first start codon to the first stop codon. As a result of the presence of intron sequences

in the genomic DNA sequences of eukaryotes, the ORF corresponding to an encoded gene will be interrupted by the presence of introns that usually generate stop codons (Fig.4).

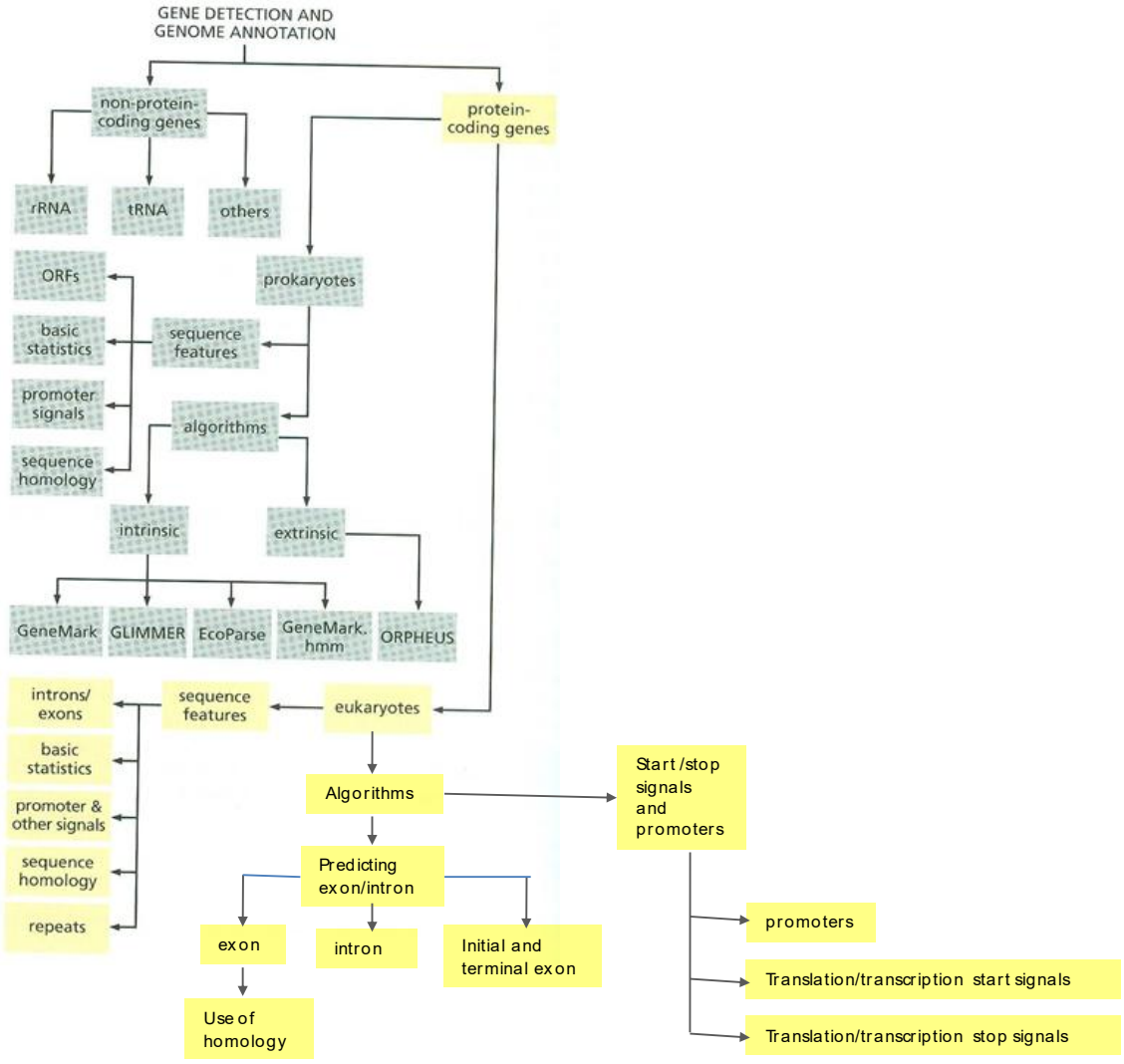


Fig. 4: Flow Diagram of Eukaryotic Gene Discovery

Gene Prediction Program

There are two basic problems in gene prediction: prediction of protein coding regions and prediction of the functional sites of genes. Gene prediction program can be classified into four generations. The first generation of programs was designed to identify approximate locations of coding regions in genomic DNA. The most widely known programs were probably TestCode and GRAIL. But they could not accurately predict precise exon locations. The second generation, such as SORFIND and Xpound, combined splice signal and coding region identification to predict potential exons, but did not attempt to assemble predicted exons into complete genes. The next generation of programs attempted the more difficult task of predicting complete gene structures. A variety of programs have been developed, including GeneID, GeneParser, GenLang, and FGENEH. However, the performance of those programs remained rather poor. Moreover, those programs were all based on the assumption that the input sequence contains exactly one complete gene, which is not often the case. To solve this

problem and improve accuracy and applicability further, GENSCAN and AUGUSTUS were developed, which could be classified into the fourth generation.

GeneMark

GeneMark uses a Markov Chain model to represent the statistics of the coding and noncoding frames. The method uses the dicodon statistics to identify coding regions. Consider the analysis of a sequence x whose base at the ith position is called x_i. The Markov chains used are fifth order, and consist of a terms such as $P(a/x_1x_2x_3x_4x_5)$, which represent the probability of the sixth base of the sequence x being given a given that the previous five bases in the sequence x where $x_1x_2x_3x_4x_5$, resulting in the first dicodon of the sequence being $x_1x_2x_3x_4x_5a$. These terms must be defined for all possible pentamers with the general sequence $b_1b_2b_3b_4b_5$. The values of these terms can be obtained of analysis of data, consisting of nucleotide sequence in which the coding regions have been actually identified. When there are sufficient data, they are given by

$$P(\frac{a}{b_1b_2b_3b_4b_5}) = \frac{n_{b_1b_2b_3b_4b_5a}}{\sum_{a=A,C,G,T} n_{b_1b_2b_3b_4b_5a}}$$

where, $n_{b_1b_2b_3b_4b_5a}$ is the number of times the sequence $b_1b_2b_3b_4b_5a$ occurs in the training data. This is the maximum likelihood estimators of the probability from the training data.

Glimmer

The core of Glimmer is Interpolated Markov Model (IMM), which can be described as a generalized Markov chain with variable order. After GeneMark introduces the fixed-order Markov chains, Glimmer attempts to find a better approach for modeling the genome content. The motivational fact is that the bigger the order of the Markov chain, the more non-randomness can be described. However, as we move to higher order models, the number of probabilities that we must estimate from the data increases exponentially. The major limitation of the fixed-order Markov chain is that models from higher order require exponentially more training data, which are limited and usually not available for new sequences. However, there are some oligomers from higher order that occur often enough to be extremely useful predictors. For the purpose of using these higher-order statistics, whenever sufficient data is available, Glimmer IMMs.

Glimmer calculates the probabilities for all Markov chains from 0^{th} order to 8^{th}. If there are longer sequences (e.g. 8-mers) occurring frequently, IMM makes use of them even when there is insufficient data to train an 8-th order model. Similarly, when the statistics from the 8-th order model do not provide significant information, Glimmer refers to the lower-order models to predict genes.

Opposed to the supervised GeneMark, Glimmer uses the input sequence for training. The ORFs longer than a certain threshold are detected and used for training, because there is high probability that they are genes in prokaryotes. Another training option is to use the sequences with homology to known genes from other organisms, available in public databases. Moreover, the user can decide whether to use long ORFs for training purposes or choose any set of genes to train and build the IMM.

GeneMark.hmm

GeneMark.hmm is designed to improve GeneMark in finding exact gene starts. Therefore, the properties of GeneMark.hmm are complementary to GeneMark. GeneMark.hmm uses GeneMark models of coding and non-coding regions and incorporates them into hidden Markov model framework. In short terms, Hidden Markov Models (HMM) are used to describe the transitions from non-coding to coding regions and vice versa. GeneMark.hmm predicts the most likely structure of the genome using the Viterbi algorithm, a dynamic programming algorithm for finding the most likely sequence of hidden states. To further improve the prediction of translation start position, GeneMark.hmm derives a model of the ribosome binding site (6-7 nucleotides preceding the start codon, which are bound by the ribosome when initiating protein translation). This model is used for refinement of the results.

Both GeneMark and GeneMark.hmm detect prokaryotic genes in terms of identifying open reading frames that contain real genes. Moreover, they both use pre-computed species-specific gene models as training data, in order to determine the parameters of the protein-coding and non-coding regions.

Orpheus

The ORPHEUS program uses homology, codon statistics and ribosome binding sites to improve the methods presented so far by using information that those programs ignored. One of the key differences is that it uses database searches to help determine putative genes, and is thus an extrinsic method. This initial set of genes is used to define the coding statistics for the organism, in this case working at the level of codon, not dicodons. These statistics are then used to define a larger set of candidate ORFs. From this set, those ORFs with an unambiguous start codon end are used to define a scoring matrix for the ribosome-binding site, which is then used to determine the 5` end of those ORFs where alternative start are present.

EcoParse

EcoParse is one of the first HMM model based gene finder, was developed for gene finding in *E.coli*. It focuses on the uses the codon structure of genes. With EcoParse a flora of HMM based gene finder, usuing dynamic programming and the viterbi algorithm to parse a sequence, emerged.

Evaluation of Gene Prediction Programs

In the field of gene prediction accuracy can be measured at three levels

a. Coding nucleotides (base level)

b. Exon structure (exon level)

c. Protein product (protein level)

At base level gene predictions can be evaluated in terms of *true positives (TP)* (predicted features that are real), *true negatives* (TN) (non-predicted features that are not real), *false positives (FP)* (predicted features that are not real), and *false negatives (FN)* (real features that were not predicted) Fig. 5. Usually the base assignment is to be in a coding or non coding segment, but this analysis can be extended to include non coding parts of genes, or any functional parts of the sequences.

Fig. 5: Four Possible Comparisons of Real and Predicted Genes

Sensitivity (Sn): The fraction of bases in real genes that are correctly predicted to be in genes is the sensitivity and interpreted as the probability of correctly predicting a nucleotide to be in a given gene that it actually is.

$$Sn = \frac{TP}{TP + FN}$$

Specificity (Sp): The fraction of those bases which are predicted to be in genes that actually are is called the specificity and interpreted as the probability of a nucleotide actually being in a gene given that it has been predicted to be.

$$Sp = \frac{TP}{TP + FP}$$

Care has to be taken in using these two values to assess a gene prediction program because, as with the normal definition of specificity, extreme results can make them misleading.

Approximate correlation coefficient (AC) has been proposed as a single measure to circumvent these difficulties. This defined as AC=2(ACP-0.5), where

$$ACP = \frac{1}{n}\left(\frac{TP}{TP + FN} + \frac{TP}{TP + FP} + \frac{TN}{TN + FP} + \frac{TN}{TN + FN}\right),$$

At the exon level, determination of prediction accuracy depends on the exact prediction of exon start and end points. There are two measures of sensitivity and specificity used in the field, each of which measures a different but useful property.

The sensitivity measures used are

S_{n1} = CE/AE and Sn2 = ME/AE

The specificity measures used are

S_{p1}=CE/PE and S_{p2}=WE/PE

Where,

AE = No of actual exons in the data

PE = No of predicted exons in the data

CE = No of correct predicted exons

ME = No of missing exons (rarely occurs)

WE = No of wrongly predicted exons (Figure-5)

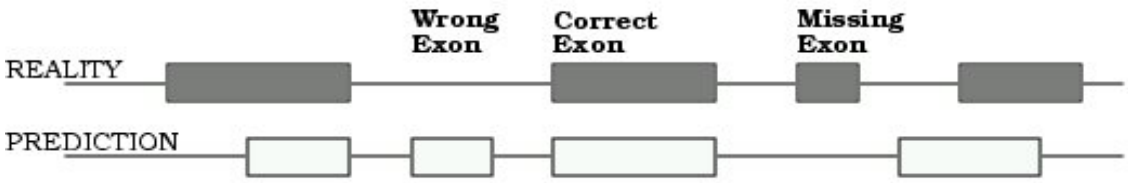


Fig. 6: Real and Predicted Exons

Gene Ontology

The gene ontology (GO, http:www.geneontology.org) is probably the most extensive scheme today for the description of gene product functions but other systems such as enzyme codes, KEGG pathways, FunCat, or COG are also widely used. Here, we describe the Blast2GO (B2G, www.blast2go.org) application for the functional annotation, management, and data mining of novel sequence data through the use of common controlled vocabulary schemas. The main application domain of the tool is the functional genomics of nonmodel organisms and it is primarily intended to support research in experimental labs. Blast2GO strives to be the application of choice for the annotation of novel sequences in functional genomics projects where thousands of fragments need to be characterized. Functional annotation in Blast2GO is based on homology transfer. Within this framework, the actual annotation procedure is configurable and permits the design of different annotation strategies. Blast2GO annotation parameters include the choice of search database, the strength and number of blast results, the extension of the query-hit match, the quality of the transferred annotations, and the inclusion of motif annotation. Vocabularies supported by B2G are gene ontology terms, enzyme codes (EC), InterPro IDs, and KEGG pathways.

Fig.7 shows the basic components of the Blast2GO suite. Functional assignments proceed through an elaborate annotation procedure that comprises a central strategy plus refinement functions. Next, visualization and data mining engines permit exploiting the annotation results to gain functional knowledge. GO annotations are generated through a 3-step process: blast, mapping, annotation. InterPro terms are obtained from InterProScan at EBI, converted and merged to GOs. GO annotation can be modulated from Annex, GOSlim web services and manual editing. EC and KEGG annotations are generated from GO. Visual tools include sequence color code, KEGG pathways, and GO graphs with node highlighting and filtering options. Additional annotation data-mining tools include statistical charts and gene set enrichment analysis functions.

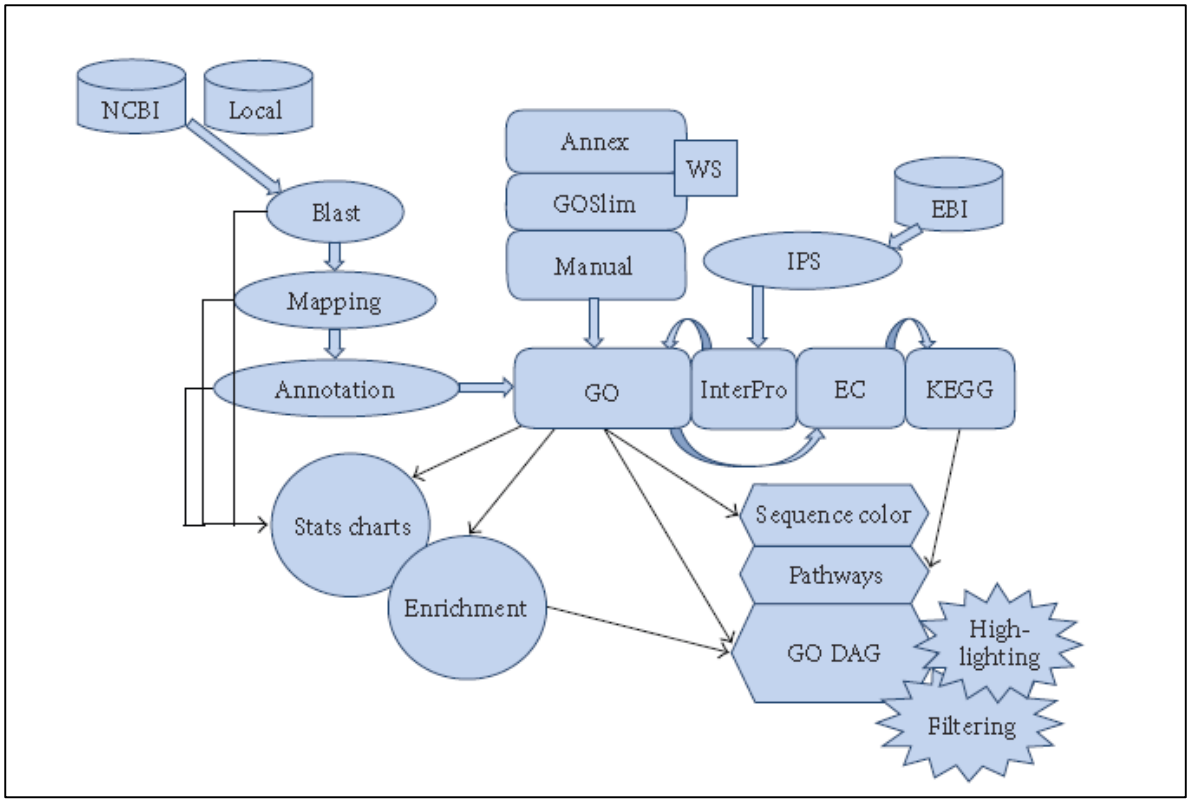


Fig. 7: Schematic Representation of Blast2GO Application.

The Blast2GO annotation procedure consists of three main steps: blast to find homologous sequences, mapping to collect GO terms associated to blast hits, and annotation to assign trustworthy information to query sequences.

Blast Step

The first step in B2G is to find sequences similar to a query set by blast. B2G accepts nucleotide and protein sequences in FASTA format and supports the four basic blast programs (blastx, blastp, blastn, and tblastx). Homology searches can be launched against public databases such as (the) NCBI nr using a query-friendly version of blast (QBlast). This is the default option and in this case, no additional installations are needed. Alternatively, blast can be run locally against a proprietary FASTA-formatted database, which requires a working www-blast installation. The Make Filtered Blast-GO-BD function in the Tools menu allows the creation of customized databases containing only GO annotated entries, which can be used in combination with the local blast option. Other configurable parameters at the blast step are the expectation value (e-value) threshold, the number of retrieved hits, and the minimal alignment length (hsp length) which permits the exclusion of hits with short, low e-value matches from the sources of functional terms. Annotation, however, will ultimately be based on sequence similarity levels as similarity percentages are independent of database size and more intuitive than e-values. Blast2GO parses blast results and presents the information for each sequence in table format. Query sequence descriptions are obtained by applying a language processing algorithm to hit descriptions, which extracts informative names and avoids low content terms such as “hypothetical protein” or “expressed protein”.

Mapping Step

Mapping is the process of retrieving GO terms associated to the hits obtained after a blast search. B2G performs three different mappings as follows.

a. Blast result accessions are used to retrieve gene names (symbols) making use of two mapping files provided by NCBI (geneinfo, gene2accession). Identified gene names are searched in the species-specific entries of the gene product table of the GO database.

b. Blast result GI identifiers are used to retrieve UniProt IDs making use of a mapping file from PIR (Non-redundant Reference Protein database) including PSD, UniProt, Swiss-Prot, TrEMBL, RefSeq, GenPept, and PDB.

c. Blast result accessions are searched directly in the DBXRef Table of the GO database.

Annotation Step

This is the process of assigning functional terms to query sequences from the pool of GO terms gathered in the mapping step. Function assignment is based on the gene ontology vocabulary. Mapping from GO terms to enzyme codes permits the subsequent recovery of enzyme codes and KEGG pathway annotations. The B2G annotation algorithm takes into consideration the similarity between query and hit sequences, the quality of the source of GO assignments, and the structure of the GO DAG. For each query sequence and each candidate GO term, an annotation score (AS) is computed (see Figure 8). The AS is composed of two terms. The first, direct term (DT), represents the highest similarity value among the hit sequences bearing this GO term, weighted by a factor corresponding to its evidence code (EC). A GO term EC is present for every annotation in the GO database to indicate the procedure of functional assignment.

$$DT = \max(\text{similarity} \times EC_{\text{weight}})$$
$$AT = (\#GO - 1) \times GO_{\text{weight}}$$
$$AR : \text{lowest.node}(AS(DT + AT) \geq \text{threshold})$$

Fig. 8: Blast2GO Annotation Rule

ECs vary from experimental evidence, such as inferred by direct assay (IDA) to unsupervised assignments such as inferred by electronic annotation (IEA). The second term (AT) of the annotation rule introduces the possibility of abstraction into the annotation algorithm. Abstraction is defined as the annotation to a parent node when several child nodes are present in the GO candidate pool. This term multiplies the number of total GOs unified at the node by a user defined factor or GO weight (GOw) that controls the possibility and strength of abstraction. When all ECw's are set to 1 (no EC control) and the GOw is set to 0 (no abstraction is possible), the annotation score of a given GO term equals the highest similarity value among the blast hits annotated with that term. If the ECw is smaller than one, the DT decreases and higher query-hit similarities are required to surpass the annotation threshold. If the GOw is not equal to zero, the AT becomes contributing and the annotation of a parent node is possible if multiple child nodes coexist that do not reach the annotation cutoff. Default values of B2G annotation parameters were chosen to optimize the ratio between annotation coverage and annotation accuracy. Finally, the AR selects the lowest terms per branch that exceed a user-defined threshold.

Blast2GO includes different functionalities to complete and modify the annotations obtained through the above-defined procedure. Enzyme codes and KEGG pathway annotations are generated from the direct mapping of GO terms to their enzyme code equivalents. Additionally, Blast2GO offers InterPro searches directly from the B2G interface. B2G launches sequence queries in batch, and recovers, parses, and uploads InterPro results. Furthermore, InterPro IDs can be mapped to GO terms and merged with blast-derived GO annotations to provide one integrated annotation result. In this process, B2G ensures that only the lowest term per branch remains in the final annotation set, removing possible parent-child relationships originating from the merging action.

References

1. Conesa, S. Gotz, J. M. Garcia-Gomez, J. Terol, M. Talon, and M. Robles, “Blast2GO: a universal tool for annotation, visualization and analysis in functional genomics research,” Bioinformatics, vol. 21, no. 18, pp. 3674–3676, 2005.
2. Conesa and S. Gotz, "Blast2GO: A Comprehensive Suite for Functional Analysis in Plant Genomics," International Journal of Plant Genomics, vol. 2008, 2008.
3. H. Ogata, S. Goto, K. Sato, W. Fujibuchi, H. Bono, and M. Kanehisa, “KEGG: Kyoto Encyclopedia of Genes and Genomes,” Nucleic Acids Research, vol. 27, no. 1, pp. 29–34, 1999.
4. J.D. Watson, R.M. Myers, A.A. Caudy and J.A. Witkowski, "Recombinant DNA: Genes and Genomes - A Short Course," 3rd Ed., 2007.
5. M. Ashburner, C. A. Ball, J. A. Blake, et al., “Gene Ontology: tool for the unification of biology. The Gene Ontology Consortium,” Nature Genetics, vol. 25, no. 1, pp. 25–29, 2000.
6. Ruepp, A. Zollner, D. Maier, et al., “The FunCat, a functional annotation scheme for systematic classification of proteins from whole genomes,” Nucleic Acids Research, vol. 32, no. 18, pp. 5539–5545, 2004.
7. R. L. Tatusov, N. D. Fedorova, J. D. Jackson, et al., “The COG database: an updated version includes eukaryotes,” BMC Bioinformatics, vol. 4, p. 41, 2003.
8. Schomburg, A. Chang, C. Ebeling, et al., “BRENDA, the enzyme database: updates and major new developments,” Nucleic Acids Research, vol. 32, Database issue, pp. D431–D433, 2004.
9. S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, “Basic local alignment search tool,” Journal of Molecular Biology, vol. 215, no. 3, pp. 403–410, 1990.
10. S. Myhre, H. Tveit, T. Mollestad, and A. Lægreid, “Additional Gene Ontology structure for improved biological reasoning,” Bioinformatics, vol. 22, no. 16, pp. 2020–2027, 2006.

Annotation of RNA-Seq Data (Practical)

Sneha Murmu

Introduction

Genome annotation is the process of identifying functional elements within a genome, such as genes, regulatory regions, and repeat elements. The goal of genome annotation is to create an accurate and comprehensive description of the genome's structure and function. This can be a time-consuming process, but it is essential for understanding how genes and other functional elements work together to control an organism's biology.

One powerful tool for genome annotation is Blast2GO (Conesa et al., 2005). Blast2GO is a commercial bioinformatics software suite that provides comprehensive functional annotation of nucleotide and protein sequences. It combines powerful sequence similarity search algorithms, such as BLAST (Altschul et al., 1997) and HMMER (Finn et al., 2011), with functional annotation tools, such as InterProScan (Zdobnov et al., 2001) and Gene Ontology (GO) mapping, to provide a detailed functional analysis of genomic and transcriptomic data.

Blast2GO works by first performing a sequence similarity search, typically using BLAST, to identify sequences with homology to known sequences in public databases. The resulting hits are then annotated using a variety of functional annotation tools, including InterProScan, which identifies conserved protein domains and functional motifs, and GO mapping, which assigns GO terms based on the functional categories of annotated genes.

Blast2GO also includes tools for statistical analysis and data visualization, allowing users to explore functional trends and patterns in their data. It can be used to analyze a wide range of genomic and transcriptomic data sets. One of the strengths of Blast2GO is its user-friendly interface, which allows even non-experts to perform complex functional annotation analyses. Blast2GO is also highly customizable, allowing users to tailor the annotation process to their specific needs and research questions.

Here are the four broad steps involved in genome annotation using Blast2GO:

- Sequence quality control and assembly: Before annotating a genome, it is important to ensure that the quality of the sequencing data is high and that the genome has been properly assembled. This may involve trimming low-quality sequences, filtering out contaminants, and performing de novo assembly or mapping to a reference genome.
- Sequence similarity search: The first step in genome annotation is to identify sequences with homology to known sequences in public databases. This is typically done using BLAST or a similar tool. The resulting hits can provide clues about the function and evolutionary relationships of the sequences in question.
- Functional annotation: Once sequences have been identified using a sequence similarity search, functional annotation tools can be used to identify functional domains and motifs, assign Gene Ontology terms, and perform other types of functional analysis. Blast2GO includes a number of annotation tools, including InterProScan, which searches for conserved domains and motifs in protein sequences, and GO mapping, which assigns Gene Ontology terms based on the functional categories of annotated genes.
- Data analysis and visualization: Once the sequences have been annotated with functional information, the data can be analyzed and visualized in a variety of ways. Blast2GO includes tools for statistical analysis and data visualization. The results of the analysis can be exported in a variety of formats for further analysis.

Installation of Blast2GO:

Following are the general steps to install Blast2GO:

1. System requirements: Check that your computer meets the system requirements for Blast2GO. Blast2GO is compatible with Windows, macOS, and Linux operating systems, and requires at least 8 GB of RAM.
2. Download Blast2GO: Visit the Blast2GO website (https://www.blast2go.com/) and download the appropriate installation file for your operating system. You may need to create an account and purchase a license, depending on your intended use of the software.
3. Install Blast2GO: Double-click the downloaded installation file and follow the on-screen instructions to install Blast2GO (as depicted in Figure 1). You may need to provide administrator permissions, depending on your operating system and security settings.

4. Configure Blast2GO: Once Blast2GO is installed, you will need to configure it to work with your specific computing environment. This may include setting preferences for sequence databases, annotation tools, and other settings.
5. Activate license: If you have purchased a license for Blast2GO, you will need to activate it before you can use the software. This typically involves entering a license key or activating the license through an online portal.

   Once Blast2GO is installed and configured, you can begin using it to analyze and annotate your genomic or transcriptomic data.

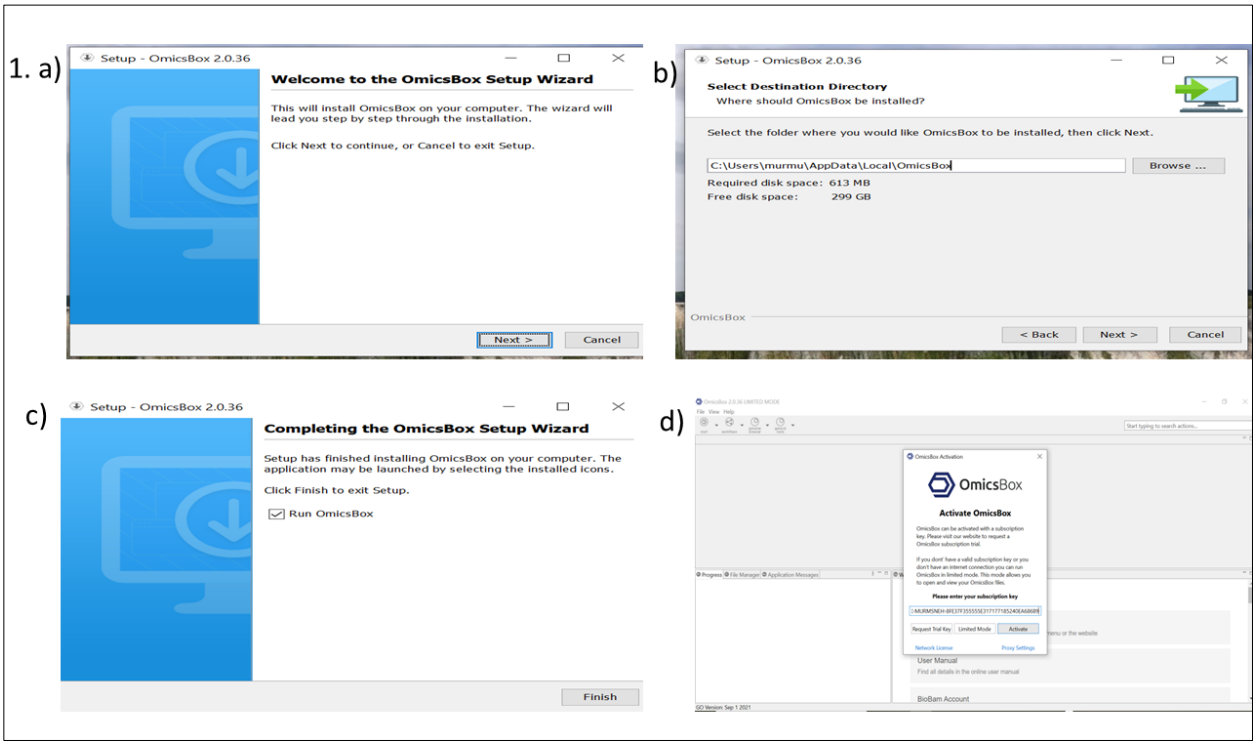


Figure 1: Installation steps of Blast2GO in Windows system.

Stepwise guide to perform annotation using Blast2GO

1. Open Blast2GO: Launch Blast2GO on your computer.
2. Load sequences: Load your sequence file(s) into Blast2GO. This can be done by clicking on "Load data" in the main menu and selecting the appropriate file type (e.g., FASTA).
3. Run **BLAST** search: In the main menu, click on "Run BLAST" and select the appropriate database for your search (e.g., NCBI non-redundant protein database) as shown in Figure 2. You can choose to run a BLASTP (protein query against protein database) or a BLASTX

(nucleotide query against protein database) search. You can also set various search parameters, such as the e-value threshold and the maximum number of hits to return.

4. View BLAST results: Once the BLAST search is complete, you can view the results in the BLAST results table (as shown in Figure 3). The table will show the sequence ID, the best hit, the e-value, the bit score, and other relevant information. You can sort the table by various columns to help you identify the best hits.
5. Import BLAST results: To import the BLAST results into the Blast2GO annotation pipeline, select the sequences you want to annotate and click on "Import selected hits". This will import the BLAST results and link them to the appropriate sequences in the annotation pipeline.

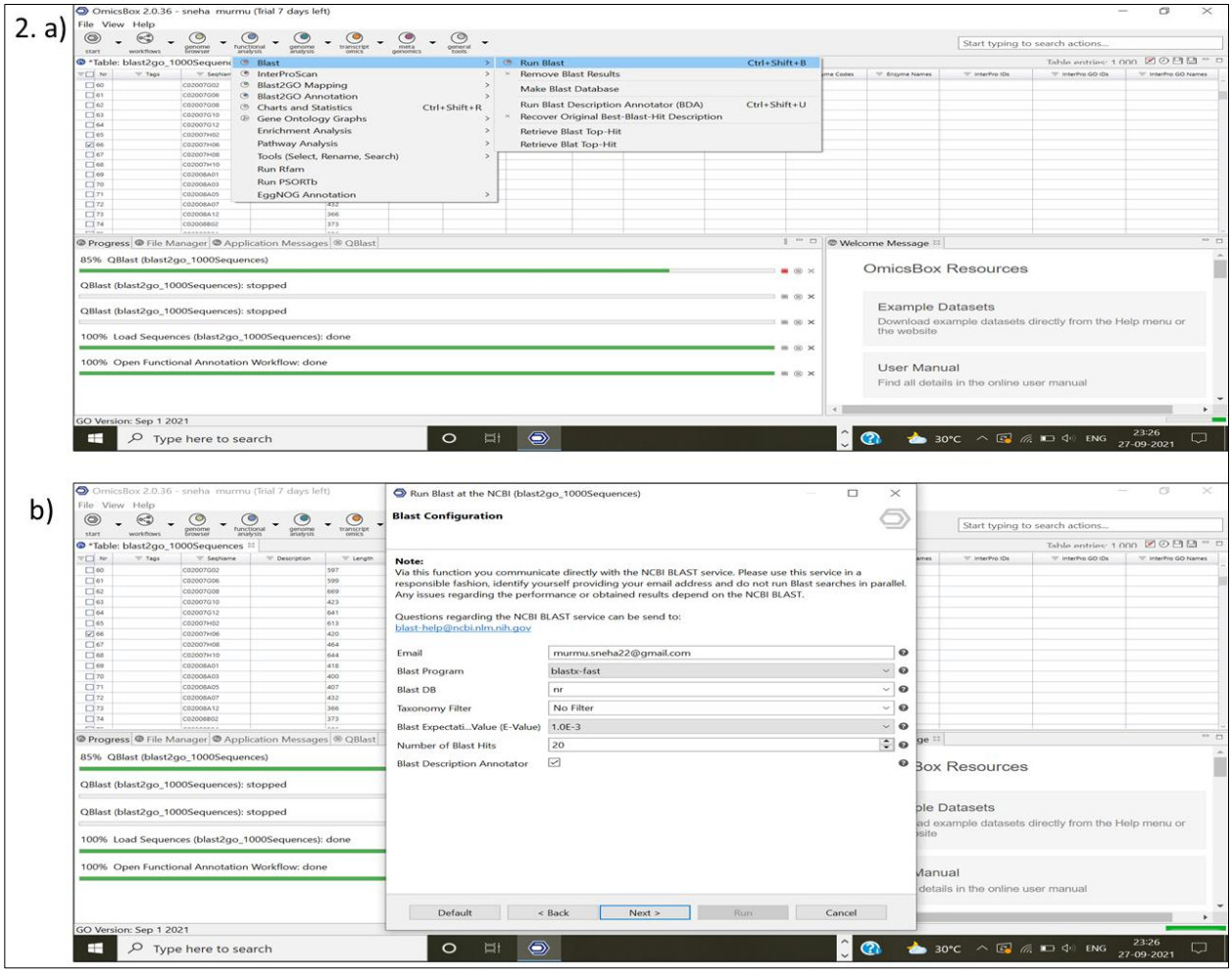


Figure 2: BLAST search.

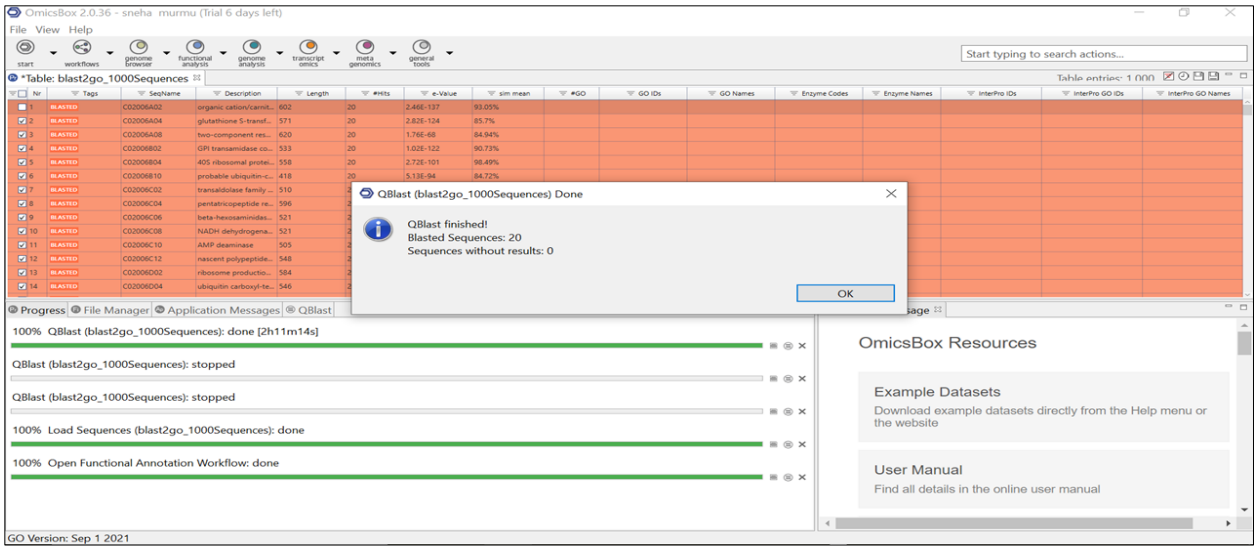


Figure 3: BLAST result.

6. Run **InterProScan**: In the main menu, click on "Run InterProScan" and select the appropriate database for your search (e.g., InterPro database). You can choose to run the search on protein or nucleotide sequences (Figure 4a).
7. Set search parameters: You can set various search parameters, such as the e-value threshold, the maximum number of sequences to align, and the type of analysis to perform (e.g., Pfam, Prosite, SMART, etc.) (Figure 4b).

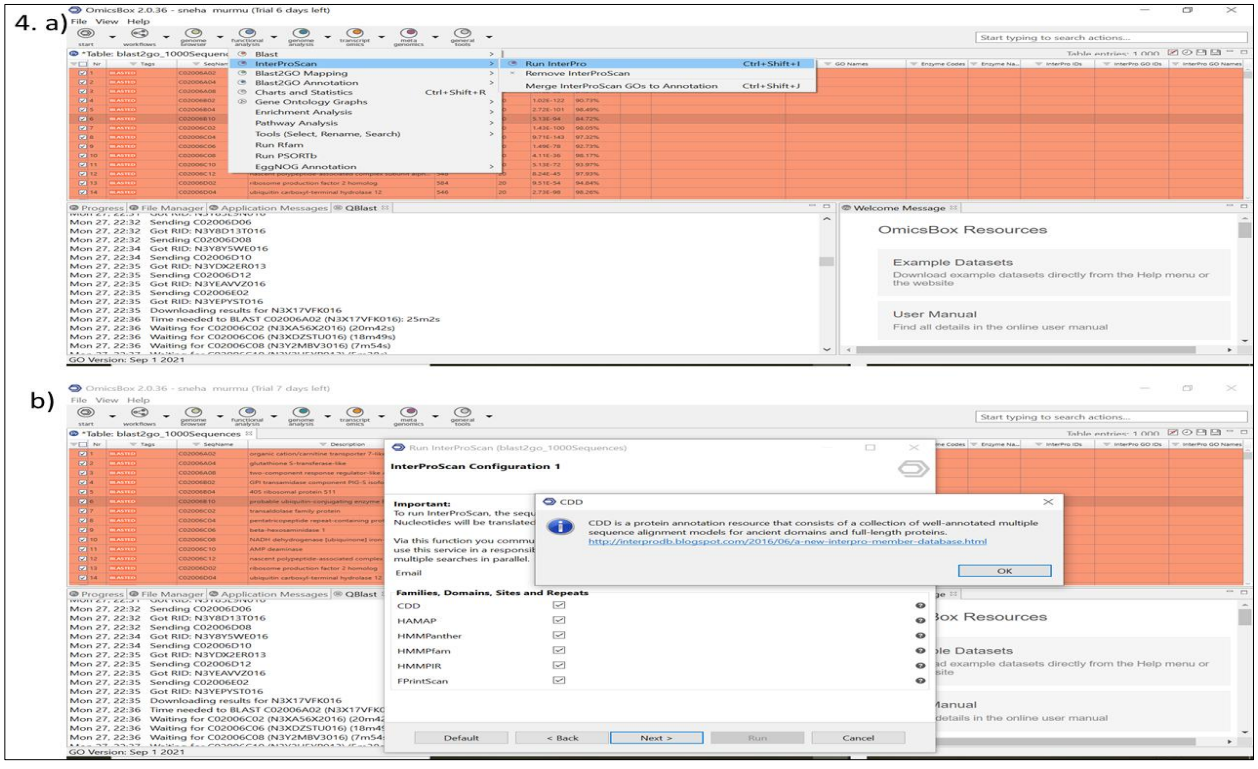


Figure 4: InterProScan search.

8. View InterProScan results: Once the InterProScan search is complete, you can view the results in the InterProScan results table. The table will show the sequence ID, the best match, the e-value, the score, and other relevant information (Figure 5). You can sort the table by various columns to help you identify the best matches.

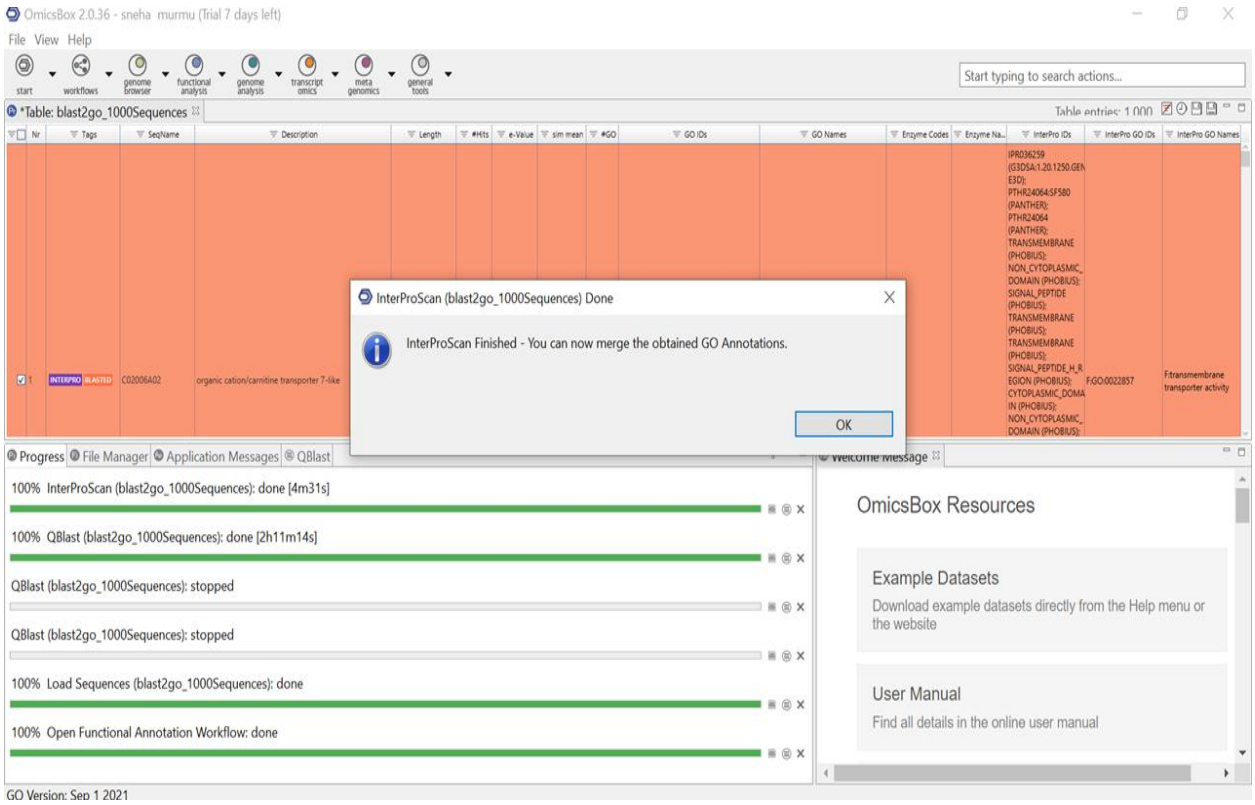


Figure 5: InterProScan result.

9. Import InterProScan results: To import the InterProScan results into the Blast2GO annotation pipeline, select the sequences you want to annotate and click on "Import selected hits". This will import the InterProScan results and link them to the appropriate sequences in the annotation pipeline.
10. Perform **mapping**: Once the BLAST results have been imported, you can use the Blast2GO mapping tools to map your sequences to Gene Ontology (GO) terms (Figure 6). This involves using the BLAST results to transfer functional annotations from similar sequences to your own sequences.

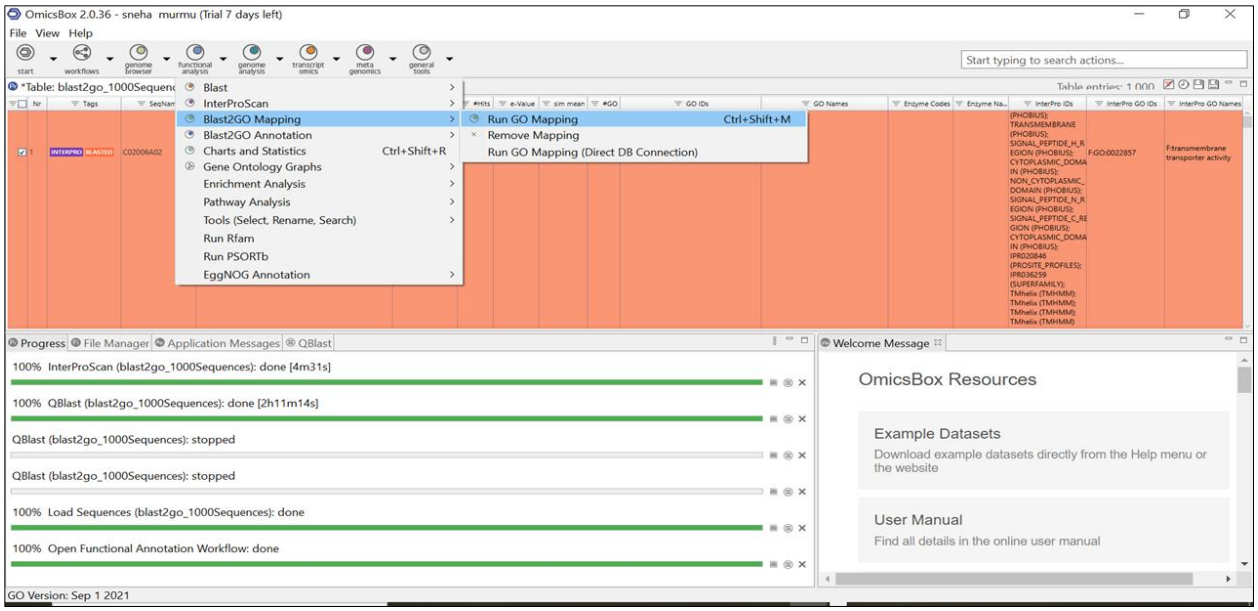


Figure 6: Mapping.

11. Edit mappings: You can edit the mappings manually, by adding or removing GO terms, or by changing the evidence codes. You can also remove or filter out low-confidence mappings, based on various criteria such as the e-value, the similarity score, or the GO term specificity.
12. Export mapping results: Once your sequences have been mapped, you can export the results in a variety of formats, such as tab-delimited text files or FASTA files (Figure 7). These results can be used for further analysis.

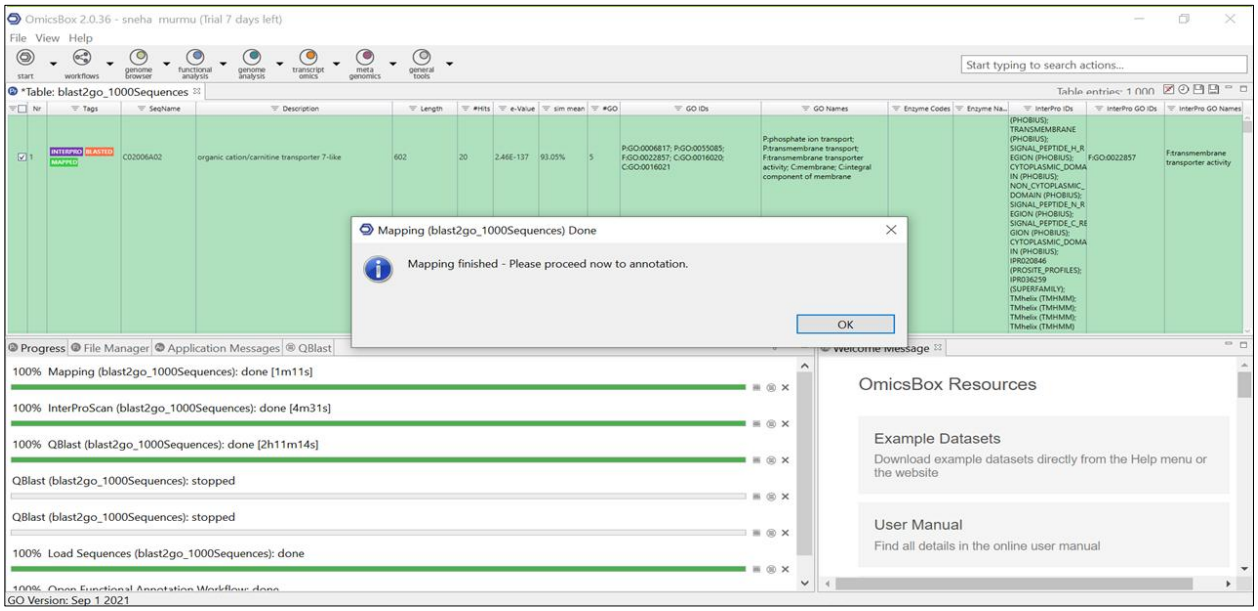


Figure 7: Mapping result.

13. **Annotate** sequences: Once the InterProScan results have been imported, you can use the Blast2GO annotation tools to assign functional information to your sequences (Figure 8). This may include mapping Gene Ontology (GO) terms, performing enrichment analysis, and performing other types of functional analysis.

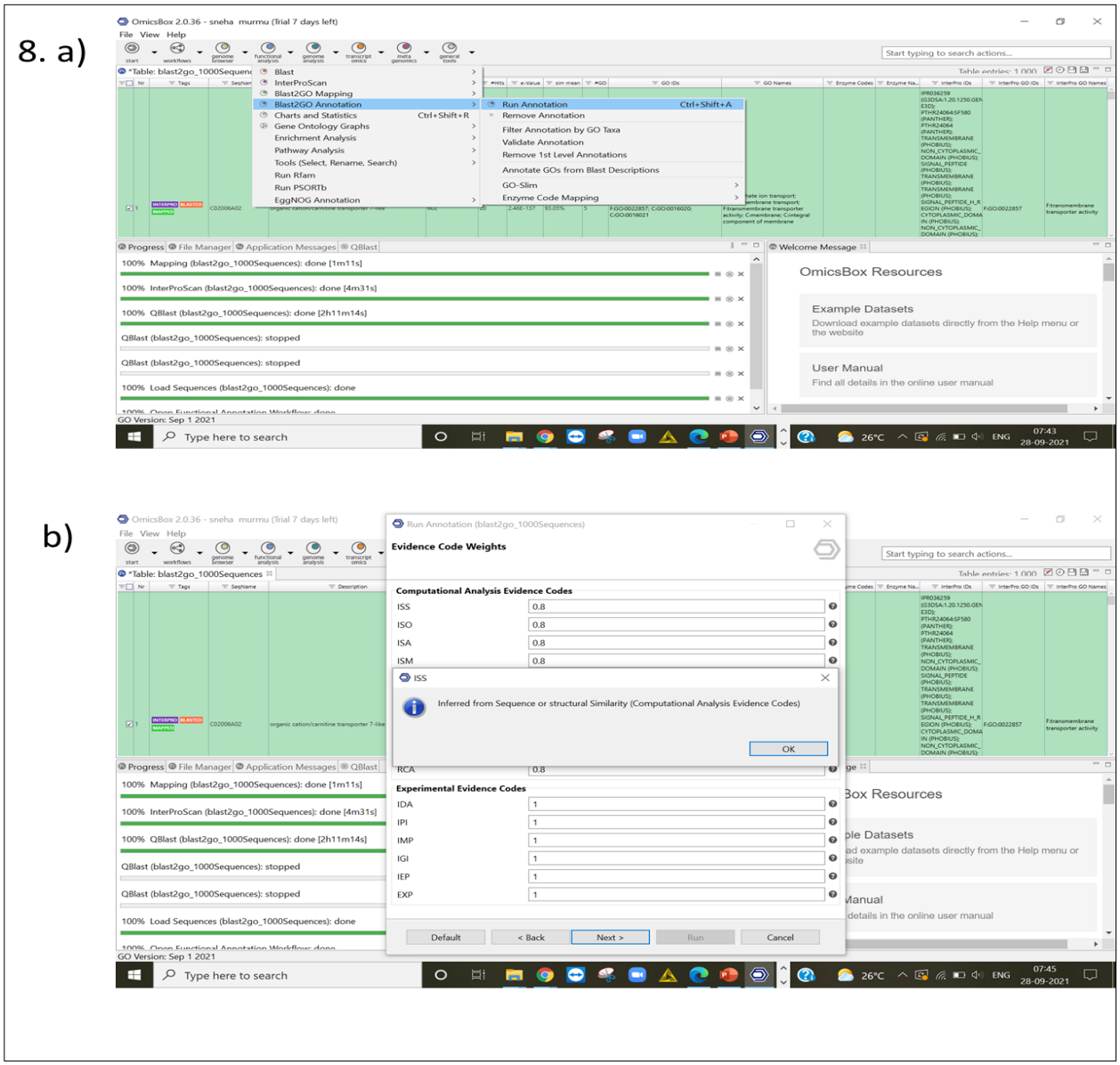


Figure 8: Annotate.

14. Export annotation results: Once your sequences have been annotated, you can export the results in a variety of formats, such as tab-delimited text files or FASTA files. These results can be used for further analysis, visualization, or sharing with collaborators.

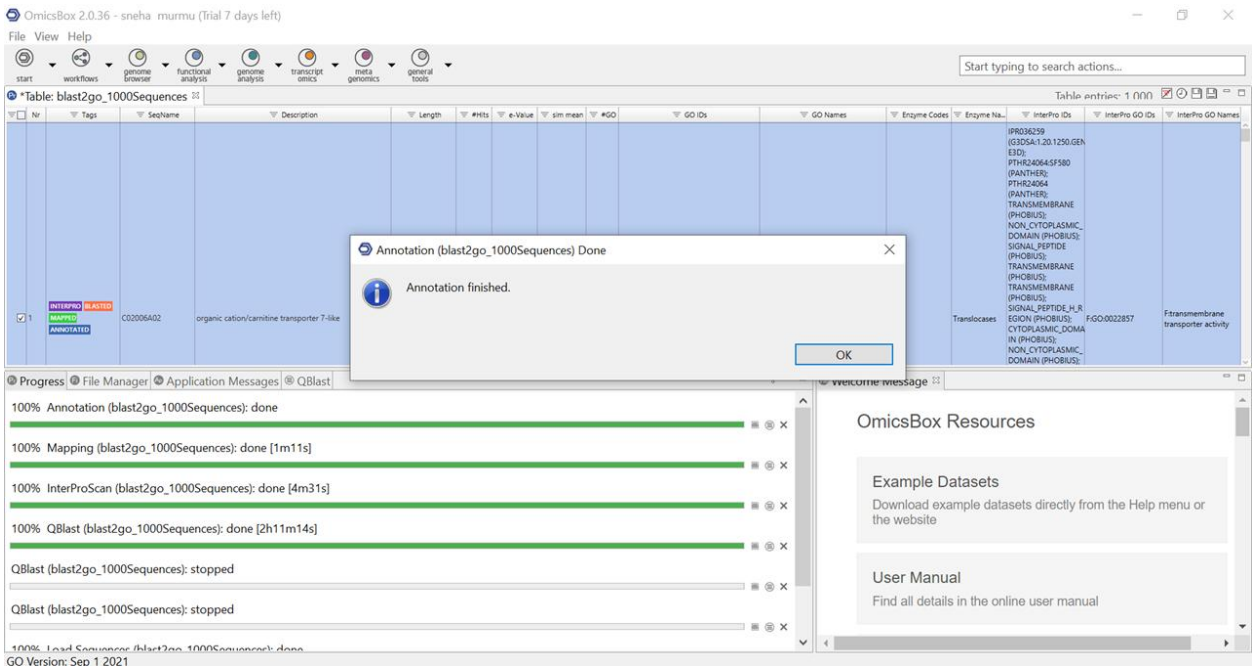


Figure 9: Annotate result.

15. Generate Gene Ontology (GO) graph: To create a GO graph in Blast2GO, click on "Graphs" in the main menu and select "GO Graph" (Figure 10). This will generate a graphical representation of the GO terms assigned to your sequences, based on the hierarchical structure of the Gene Ontology.

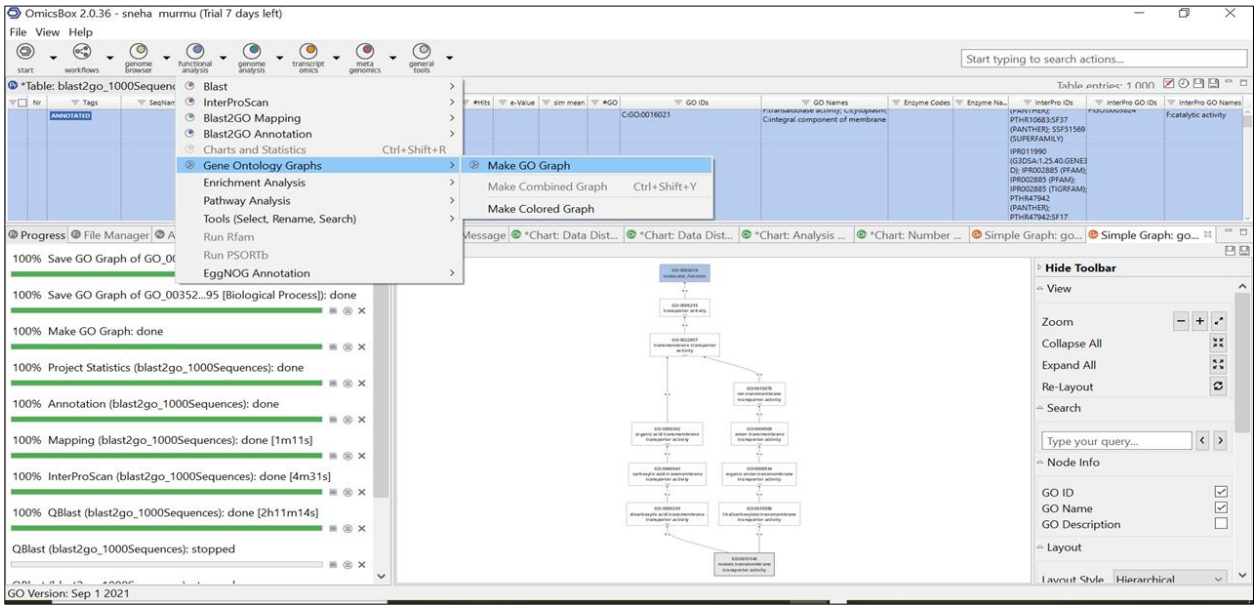


Figure 10. Generate GO graph.

16. Customize GO graph: You can customize the appearance of the GO graph by changing the colors, font sizes, or layout. You can also filter the GO terms based on various criteria such as

the level in the hierarchy, the number of sequences assigned to the term, or the statistical significance of the enrichment.

17. Analyze GO graph: Once you have generated a GO graph, you can use it to analyze the functional annotations of your sequences. This can include identifying overrepresented or underrepresented GO terms, comparing the GO profiles of different datasets or treatments, or visualizing the relationships between different biological processes, molecular functions, or cellular components (Figure 11).

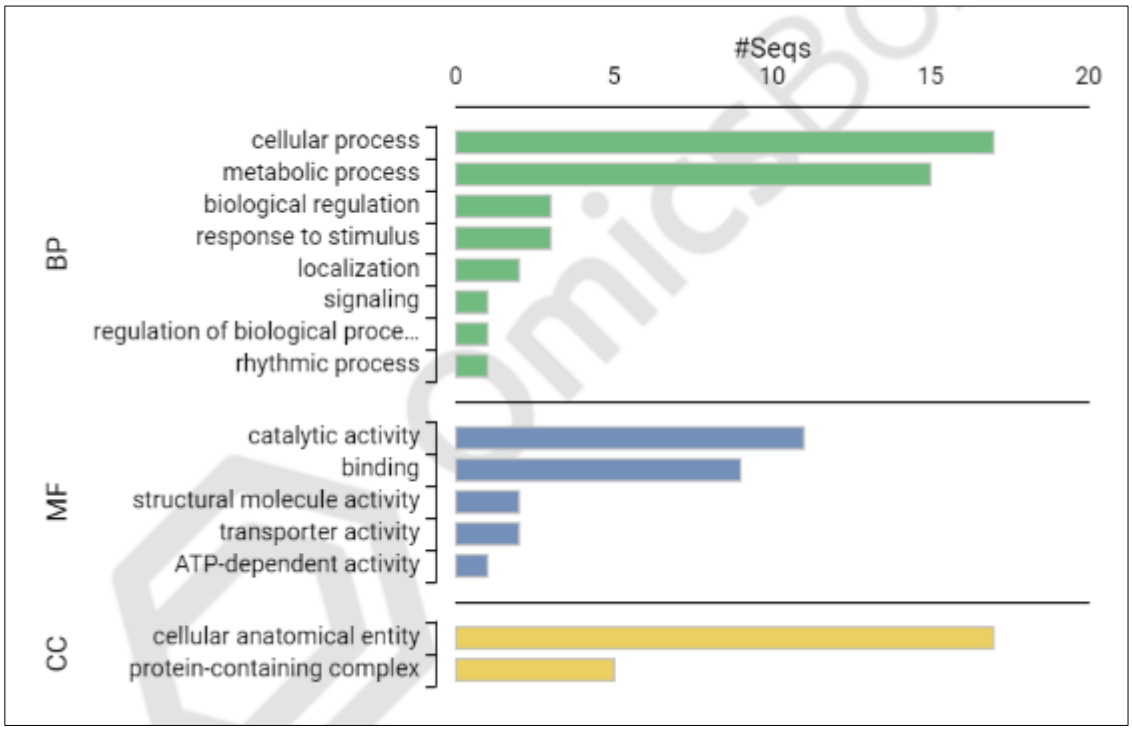


Figure 11: GO graph.

18. Export GO graph: Once you have customized and analyzed your GO graph, you can export it in a variety of formats, such as PNG, PDF, or SVG. These graphs can be used for presentations, publications, or further analysis with other tools or software.
19. Perform pathway analysis: To perform pathway analysis in Blast2GO, you need to use the KEGG (Kyoto Encyclopedia of Genes and Genomes) pathway database. In the main menu, click on "Annotation" and select "Pathway annotation". This will open the pathway annotation dialog box (Figure 12).

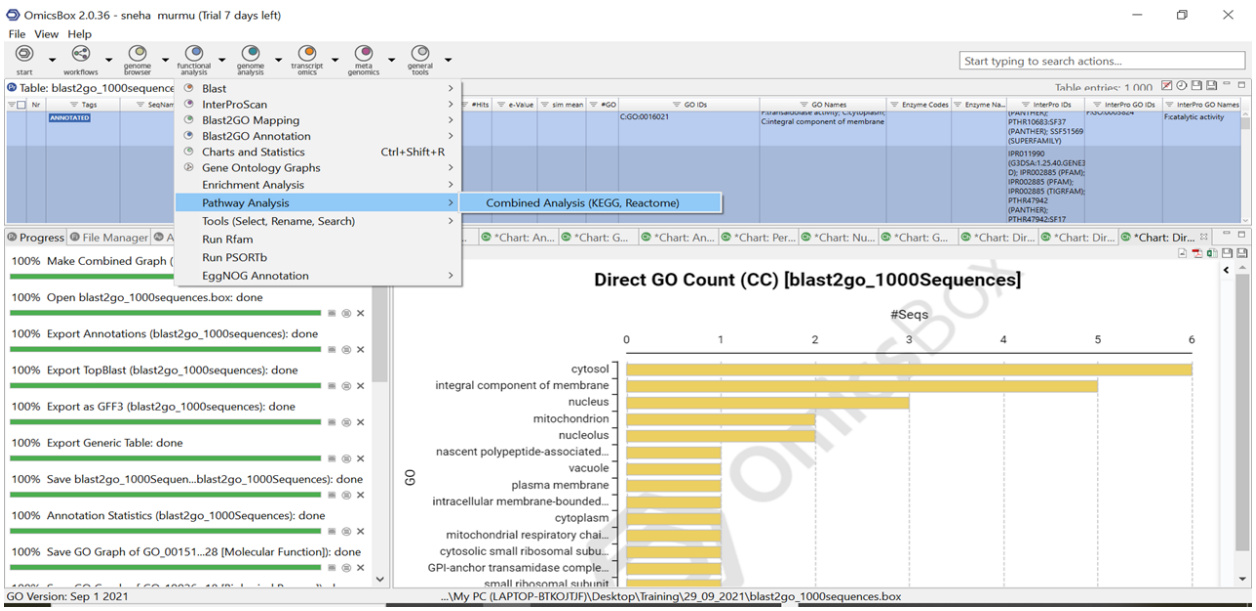


Figure 12. Run Pathway Analysis.

20. Select pathway database: In the pathway annotation dialog box, select the "KEGG" database and click on "Start". Blast2GO will download and install the latest version of the KEGG database on your computer.
21. Run pathway analysis: Once the KEGG database is installed, you can use the Blast2GO pathway analysis tools to identify the KEGG pathways that are enriched in your sequences. This involves comparing the frequency of KEGG pathway terms in your sequences to the frequency of these terms in a reference dataset, such as the entire KEGG database.
22. Filter and visualize pathways: Once the pathway analysis is complete, you can use the Blast2GO pathway analysis tools to filter and visualize the enriched pathways. This can involve setting statistical thresholds, such as the false discovery rate (FDR) or the p-value, or selecting specific pathways based on their relevance to your research question.
23. Analyze pathways: Once you have identified the enriched pathways, you can use the Blast2GO pathway analysis tools to analyze the functional annotations and gene products associated with these pathways. This can include identifying the key enzymes or regulators, comparing the pathway profiles of different datasets or treatments, or visualizing the relationships between different metabolic or signaling pathways (Figure 13).

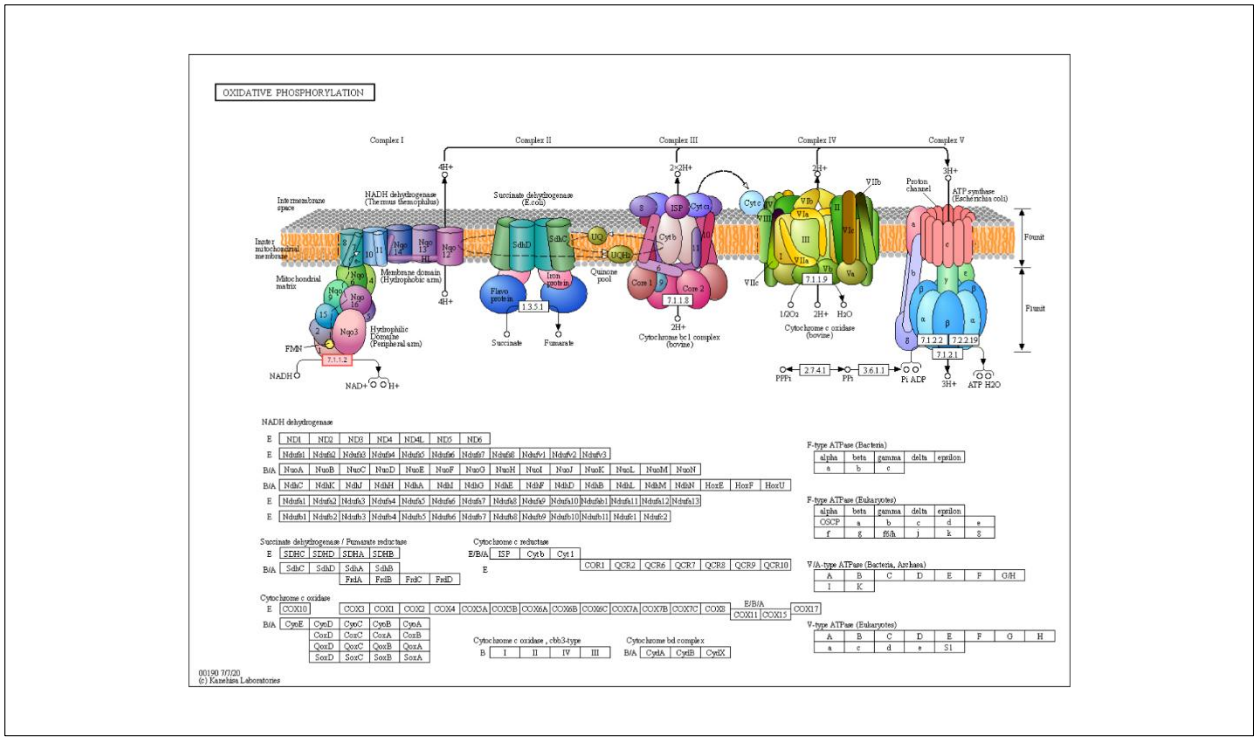


Figure 13. Pathway graph.

24. Export pathway data: Once you have customized and analyzed your pathway data, you can export it in a variety of formats, such as Excel, CSV, or XML. These data can be used for further analysis with other tools or software, or for visualizing and communicating the results of your pathway analysis.

References

Conesa, A., Götz, S., García-Gómez, J. M., Terol, J., Talón, M., & Robles, M. (2005). Blast2GO: a universal tool for annotation, visualization and analysis in functional genomics research. Bioinformatics, 21(18), 3674-3676.

Altschul, S. F., Madden, T. L., Schäffer, A. A., Zhang, J., Zhang, Z., Miller, W., & Lipman, D. J. (1997). Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. Nucleic acids research, 25(17), 3389-3402.

Finn, R. D., Clements, J., & Eddy, S. R. (2011). HMMER web server: interactive sequence similarity searching. Nucleic acids research, 39(suppl_2), W29-W37.

Zdobnov, E. M., & Apweiler, R. (2001). InterProScan–an integration platform for the signature-recognition methods in InterPro. Bioinformatics, 17(9), 847-848.

INTRODUCTION TO PYTHON

Sharanbasappa and Vinaykumar L N

ICAR-Indian Agricultural Statistics Research Institute, New Delhi-110012

1. Introduction to Python

1.1 What is Python?

Python is a high-level, interpreted, general-purpose programming language created by Guido van Rossum and first released in 1991 (Van Rossum & Drake, 2009). It emphasizes code readability through significant indentation and a clean, English-like syntax. Python supports multiple programming paradigms, including procedural, object-oriented, and functional programming (Lutz, 2013).

1.2 Key Features

- Easy to Learn and Read: Simple syntax close to natural language.
- Interpreted Language: Code executes line-by-line via an interpreter; no separate compilation step.
- Dynamically Typed: Variable types are determined at runtime, not declared explicitly.
- Cross-Platform: Runs on Windows, macOS, Linux, and more without modification.
- Extensive Standard Library: Ships with modules for file I/O, math, networking, and more ("batteries included").
- Large Ecosystem: Third-party packages available via PyPI (Python Package Index) using pip.
- Object-Oriented: Supports classes, objects, inheritance, and polymorphism.
- Open Source and Free: Distributed under the PSF license.

1.3 Applications of Python

Domain	Typical Use / Libraries
Web Development	Django, Flask
Data Science & ML	NumPy, Pandas, scikit-learn, TensorFlow
Automation / Scripting	os, shutil, subprocess
Software Testing	unittest, pytest
Networking	socket, requests
GUI Applications	Tkinter, PyQt

1.4 Python 2 vs Python 3

Python 2 was officially discontinued (end-of-life) on January 1, 2020; all modern development uses Python 3. Notable differences include: print is a function in Python 3 (print("x")) rather than a statement;

division of two integers with / returns a float in Python 3; and Python 3 handles strings as Unicode by default, whereas Python 2 distinguished between str and unicode types.

2. Installation and Program Execution

2.1 Installing Python

Python can be downloaded from python.org for Windows, macOS, and Linux. Most Linux distributions and macOS versions include Python pre-installed. After installation, the interpreter is invoked from the command line using the command python or python3 (Python Software Foundation, 2024).

2.2 Modes of Execution

1. Interactive Mode: Statements are typed directly at the >>> prompt and executed immediately; useful for testing small snippets.
2. Script Mode: Code is written in a file with a .py extension and executed using the command: python filename.py

```
$ python3 hello.py
Hello, World!
```

2.3 Writing the First Program

```
print("Hello, World!")
```

The print() function is a built-in function used to display output on the console. Statements in Python do not require a terminating semicolon, although one may be used to separate multiple statements on a single line.

3. Basic Syntax, Variables and Data Types

3.1 Identifiers and Keywords

An identifier is a name used to identify a variable, function, class, or module. Rules: must begin with a letter or underscore, may contain letters, digits, and underscores, and is case-sensitive. Keywords (e.g., if, else, for, while, def, class, import, return, True, False, None) are reserved and cannot be used as identifiers (Downey, 2015).

3.2 Variables

Python variables do not require explicit type declaration; the type is inferred from the assigned value. A variable is created the moment a value is first assigned to it.

```
x = 10          # integer
name = "Ravi"    # string
pi = 3.14        # float
is_valid = True  # boolean
```

3.3 Data Types

Type	Description	Example
int	Whole numbers, unlimited precision	10, -5, 2024
float	Decimal / floating-point numbers	3.14, -0.5
str	Sequence of Unicode characters	"Python"
bool	Boolean value	True, False
complex	Complex numbers	2+3j
list	Ordered, mutable collection	[1,2,3]
tuple	Ordered, immutable collection	(1,2,3)
dict	Key-value mapping	{"a":1}
set	Unordered collection of unique items	{1,2,3}

3.4 Type Conversion

Python provides built-in functions int(), float(), str(), and bool() to convert between data types explicitly (explicit/type casting). Implicit conversion occurs automatically when Python promotes a smaller data type to a larger one during an operation, such as int to float.

```
a = int("25")     # string to int
b = float(a)      # int to float
c = str(3.14)     # float to string
```

3.5 Comments and Indentation

A single-line comment begins with the # symbol and is ignored by the interpreter. Multi-line comments are typically written as consecutive single-line comments or as triple-quoted strings. Indentation (whitespace at the start of a line) is not merely stylistic in Python — it is syntactically required to define the boundaries of blocks such as loops, conditionals, and function bodies. Inconsistent indentation raises an IndentationError.

```
# This is a single-line comment
x = 5   # inline comment
"""
This is a multi-line
docstring / comment
"""
```

3.6 Taking Input

The built-in input() function reads a line of text from the user as a string. Since the return value is always a string, numeric input must be explicitly converted using int() or float().

```
age = int(input("Enter your age: "))
print("Next year you will be", age + 1)
```

4. Operators

4.1 Types of Operators

- Arithmetic Operators: +, -, *, /, // (floor division), % (modulus), ** (exponent).
- Relational (Comparison) Operators: ==, !=, >, <, >=, <=.
- Logical Operators: and, or, not.
- Assignment Operators: =, +=, -=, *=, /=, //=, %=, **=.
- Bitwise Operators: & (AND), | (OR), ^ (XOR), ~ (NOT), << (left shift), >> (right shift).
- Membership Operators: in, not in — test membership in a sequence.
- Identity Operators: is, is not — compare object identity, not value.

4.2 Operator Precedence

When multiple operators appear in an expression, Python evaluates them according to precedence rules: parentheses first, followed by exponentiation, unary operators, multiplicative operators, additive operators, comparisons, and finally logical operators. Parentheses can always be used to override default precedence and improve clarity.

5. Control Flow Statements

5.1 Conditional Statements

Python uses indentation (not braces) to define blocks of code. Conditional execution is achieved using if, elif, and else.

```
marks = 78
if marks >= 90:
    grade = "A"
elif marks >= 75:
    grade = "B"
else:
    grade = "C"
print(grade)
```

5.2 Looping Statements

for loop: used to iterate over a sequence (list, tuple, string, range, etc.).

```
for i in range(5):
    print(i)   # prints 0 1 2 3 4
```

while loop: repeats a block of statements as long as a given condition remains true.

```
count = 0
while count < 5:
    print(count)
    count += 1
```

5.3 Loop Control Statements

- break: terminates the loop immediately and transfers control to the statement after the loop.
- continue: skips the remaining statements in the current iteration and proceeds to the next.
- pass: a null operation used as a placeholder where a statement is syntactically required.

6. Strings

6.1 String Basics

A string is an immutable sequence of characters enclosed in single, double, or triple quotes. Triple-quoted strings can span multiple lines and are also used for docstrings (Lutz, 2013).

```
s = "Python Programming"
print(s[0])       # P  (indexing)
print(s[0:6])     # Python (slicing)
print(s[::-1])    # reversed string
```

6.2 Common String Methods

Method	Purpose
upper() / lower()	Convert case
strip()	Remove leading/trailing whitespace
split(sep)	Split string into a list
join(list)	Join list elements into a string
replace(old,new)	Replace substring
find(sub)	Return index of first occurrence
format() / f-strings	String formatting/interpolation

```
name = "World"
print(f"Hello, {name}!")   # f-string formatting
```

7. Data Structures: List, Tuple, Set, Dictionary

7.1 Lists

A list is an ordered, mutable collection that can hold heterogeneous elements. Lists are defined using square brackets (McKinney, 2022).

```
fruits = ["apple", "banana", "cherry"]
fruits.append("mango")
fruits[1] = "grape"
print(fruits)
```

7.2 Tuples

A tuple is similar to a list but immutable — elements cannot be modified after creation. Tuples are defined using parentheses and are typically used for fixed collections of data.

```
point = (3, 4)
print(point[0])   # 3
```

7.3 Sets

A set is an unordered collection of unique elements, defined using curly braces or the set() constructor. Sets support mathematical operations such as union, intersection, and difference.

```
a = {1, 2, 3}
b = {2, 3, 4}
print(a | b)   # union
print(a & b)   # intersection
```

7.4 Dictionaries

A dictionary stores data as key-value pairs, providing fast lookup by key. Keys must be unique and immutable; values can be of any type.

```
student = {"name": "Asha", "age": 21}
print(student["name"])
student["age"] = 22
```

7.5 Comparison Summary

Structure	Ordered	Mutable	Duplicates
List	Yes	Yes	Allowed
Tuple	Yes	No	Allowed
Set	No	Yes	Not allowed
Dictionary	Yes (3.7+)	Yes	Keys unique

7.6 Comprehensions

Comprehensions provide a concise syntax for constructing lists, sets, and dictionaries from existing iterables in a single readable line, often replacing an equivalent multi-line for-loop with append() calls.

```
squares = [x * x for x in range(10)]
evens = [x for x in range(20) if x % 2 == 0]
sq_set = {x * x for x in range(5)}
sq_dict = {x: x * x for x in range(5)}
```

8. Functions

8.1 Defining and Calling Functions

A function is a reusable block of code defined using the def keyword. Functions help avoid repetition and organize code into logical units (Downey, 2015).

```
def greet(name):
    return "Hello, " + name

print(greet("Sara"))
```

8.2 Function Arguments

- Positional Arguments: passed in the order the parameters are defined.
- Default Arguments: provide a default value if no argument is supplied, e.g., def greet(name="Guest").
- Keyword Arguments: passed using the parameter name, allowing any order.
- Variable-Length Arguments: *args collects extra positional arguments as a tuple; **kwargs collects extra keyword arguments as a dictionary.

8.3 Scope of Variables

A variable defined inside a function has local scope and is accessible only within that function. A variable defined outside all functions has global scope. The global keyword allows a function to modify a global variable.

8.4 Lambda Functions

A lambda function is a small, anonymous, single-expression function defined using the lambda keyword, commonly used for short operations passed to functions like map(), filter(), and sorted().

```
square = lambda x: x * x
print(square(5))   # 25
```

8.5 Recursion

A recursive function is a function that calls itself to solve smaller instances of the same problem, combined with a base case that stops the recursion. Recursive solutions are common for problems such as computing factorials, Fibonacci numbers, and traversing tree-like structures.

```
def factorial(n):
    if n == 0:
        return 1
    return n * factorial(n - 1)
```

```
print(factorial(5))   # 120
```

9. Modules and Packages

9.1 Modules

A module is a file containing Python code (functions, classes, variables) that can be imported and reused in other programs. Python's standard library provides many built-in modules such as math, random, datetime, and os.

```
import math
print(math.sqrt(16))    # 4.0

from random import randint
print(randint(1, 10))
```

9.2 Packages

A package is a directory containing multiple related modules along with an __init__.py file, allowing hierarchical organization of a program's namespace. Third-party packages are installed using the pip package manager, e.g., pip install numpy.

9.3 The __name__ Variable

Every Python module has a built-in attribute __name__. When a file is run directly, __name__ is set to "__main__"; when the same file is imported as a module elsewhere, __name__ is set to the module's filename. This allows a script to distinguish between direct execution and being imported.

```
def main():
    print("Running directly")

if __name__ == "__main__":
    main()
```

10. Exception Handling

10.1 Errors vs Exceptions

A syntax error occurs when code violates the grammar rules of the language and is detected before execution. An exception is an error detected during execution that disrupts the normal flow of the program (e.g., dividing by zero, accessing an invalid index).

10.2 try-except Block

Python handles runtime exceptions gracefully using try, except, else, and finally blocks, preventing abrupt program termination.

```
try:
    result = 10 / 0
except ZeroDivisionError:
    print("Cannot divide by zero")
else:
    print(result)
finally:
    print("Execution complete")
```

10.3 Common Built-in Exceptions

Exception	Cause
ZeroDivisionError	Division or modulo by zero
ValueError	Invalid value for an operation/function
TypeError	Operation on incompatible types
IndexError	Sequence index out of range
KeyError	Dictionary key not found
FileNotFoundError	File does not exist

10.4 Raising Exceptions and Custom Exceptions

The raise statement allows a program to trigger an exception deliberately, which is useful for enforcing constraints or validating input. Custom exceptions can be created by defining a class that inherits from the built-in Exception class, allowing programs to define domain-specific error types.

```
class InsufficientBalanceError(Exception):
    pass


def withdraw(balance, amount):
    if amount > balance:
```

```
        raise InsufficientBalanceError("Not enough funds")
    return balance - amount
```

> *Note: Multiple except blocks may be used to handle different exception types differently, and a bare except catches any exception type.*

11. File Handling

11.1 Opening and Closing Files

Files are opened using the built-in open() function, which requires a filename and a mode (e.g., "r" for read, "w" for write, "a" for append, "rb"/"wb" for binary). Using the with statement automatically closes the file once the block finishes executing, even if an exception occurs.

```
with open("data.txt", "r") as f:
    content = f.read()
    print(content)
```

11.2 File Modes

Mode	Meaning
r	Read (default); error if file does not exist
w	Write; creates file if absent, overwrites if present
a	Append; creates file if absent, adds to end if present
r+	Read and write, without truncating
rb / wb	Read/write in binary mode

11.3 Reading and Writing

- read(): reads the entire file content as a single string.
- readline(): reads a single line from the file.
- readlines(): returns a list of all lines in the file.
- write(text): writes a string to the file; does not add a newline automatically.

12. Object-Oriented Programming in Python

12.1 Classes and Objects

A class is a blueprint for creating objects, bundling data (attributes) and behavior (methods) together. An object is an instance of a class. The special method __init__() acts as a constructor, initializing object attributes when a new instance is created (Lutz, 2013).

```
class Student:
    def __init__(self, name, marks):
        self.name = name
        self.marks = marks

    def display(self):
        print(self.name, self.marks)

s1 = Student("Neha", 88)
s1.display()
```

12.2 Core OOP Principles

- Encapsulation: bundling data and methods together and restricting direct access to internal state using naming conventions (single or double leading underscore).
- Inheritance: a class (child/subclass) can acquire attributes and methods from another class (parent/superclass), promoting code reuse.
- Polymorphism: the same method name behaves differently depending on the object/class invoking it.
- Abstraction: hiding internal implementation details and exposing only the necessary functionality to the user.

12.3 Inheritance Example

```
class Person:
    def __init__(self, name):
        self.name = name

class Employee(Person):
    def __init__(self, name, salary):
        super().__init__(name)
        self.salary = salary
```

12.4 Method Overriding

A subclass can provide its own implementation of a method already defined in its parent class; this is called method overriding. The super() function allows the subclass to still access the parent class's version of the overridden method when needed.

12.5 Static and Class Methods

An instance method (the default) takes self as its first parameter and can access instance data. A class method, marked with the @classmethod decorator, takes cls as its first parameter and operates on the class itself rather than an instance. A static method, marked with @staticmethod, takes neither self nor cls and behaves like a plain function placed inside a class for organizational purposes.

```
class MathHelper:
    @staticmethod
    def add(a, b):
        return a + b

print(MathHelper.add(3, 4))   # 7
```

12.6 Multiple Inheritance

Python allows a class to inherit from more than one parent class simultaneously, unlike languages such as Java. When multiple parent classes define a method with the same name, Python resolves the call order using the Method Resolution Order (MRO), computed via the C3 linearization algorithm.

13. Iterators, Generators and Decorators

13.1 Iterators

An iterable is any object capable of returning its members one at a time (e.g., lists, tuples, strings, dictionaries). An iterator is an object that implements the __iter__() and __next__() methods and produces the next value in a sequence on demand, raising StopIteration once exhausted. The built-in iter() and next() functions work with any iterable.

```
nums = [1, 2, 3]
it = iter(nums)
print(next(it))   # 1
print(next(it))   # 2
```

13.2 Generators

A generator is a special type of function that uses the yield keyword instead of return to produce a sequence of values lazily, one at a time, without storing the entire sequence in memory. Each call to next() resumes execution from where the generator last paused.

```
def countdown(n):
    while n > 0:
        yield n
        n -= 1

for value in countdown(3):
    print(value)   # 3 2 1
```

13.3 Decorators

A decorator is a function that takes another function as an argument, adds some functionality to it, and returns a new function — without permanently modifying the original function's source code. Decorators are applied using the @ symbol placed immediately above a function definition.

```
def my_decorator(func):
    def wrapper():
        print("Before the function call")
        func()
        print("After the function call")
    return wrapper

@my_decorator
def say_hello():
    print("Hello!")

say_hello()
```

14. Overview of the Standard Library

Python's standard library provides ready-to-use modules that eliminate the need to write common functionality from scratch (Python Software Foundation, 2024).

Module	Purpose
math	Mathematical functions (sqrt, sin, log, etc.)
random	Random number generation
datetime	Date and time manipulation
os	Interacting with the operating system
sys	Access to interpreter variables and functions
json	Encoding and decoding JSON data
re	Regular expression matching
collections	Specialized container datatypes

Summary

Python's readable syntax, dynamic typing, extensive standard library, and rich third-party ecosystem make it a widely adopted language across web development, automation, data science, and artificial intelligence. A strong grasp of variables, data types, control flow, functions, data structures, exception handling, file handling, and object-oriented programming forms the foundation for further study of advanced Python applications and frameworks.

15. References

Downey, A. B. (2015). Think Python: How to think like a computer scientist (2nd ed.). O'Reilly Media.

Lutz, M. (2013). Learning Python (5th ed.). O'Reilly Media.

McKinney, W. (2022). Python for data analysis (3rd ed.). O'Reilly Media.

Python Software Foundation. (2024). The Python standard library. https://docs.python.org/3/library/

Van Rossum, G., & Drake, F. L. (2009). Python 3 reference manual. CreateSpace.

Introduction to R and RStudio

Neeraj Budhlakoti, Sudhir Srivastava, D. C. Mishra
ICAR-IASRI, New Delhi

Introduction

R is a powerful programming language for statistical analysis. This software is an implementation of S programming language which was designed by John Chambers at Bell Labs. R was created by Ross Ihaka and Robert Gentleman at the University of Auckland, New Zealand. It is currently developed by R Development Core Team. The name `R' is from the first names of first two authors and partly due to its inheritance from `S'. Recently R has become one of world's popular statistical analysis software, because of the following reasons:

(i) R is free, open source and capable of almost any statistical analysis including the most recently developed statistical methodologies.

(ii) R provides very good graphical facilities.

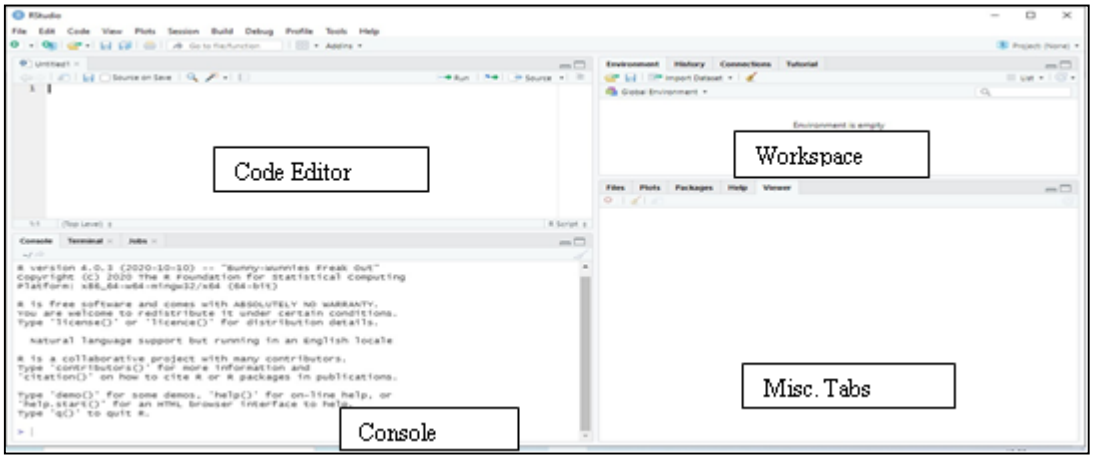
(iii) R is easily extensible through new contributions from statisticians and researchers around the globe, which also makes R quite different from other popular statistical analysis software. In fact, R community is highly active in terms of new contributions in the form of packages to R.

Working in R and RStudio

R can be installed in Linux, Unix, Windows and Mac platforms from www.r-project.org. For downloading R, please visit https://cloud.r-project.org/.

The R GUI

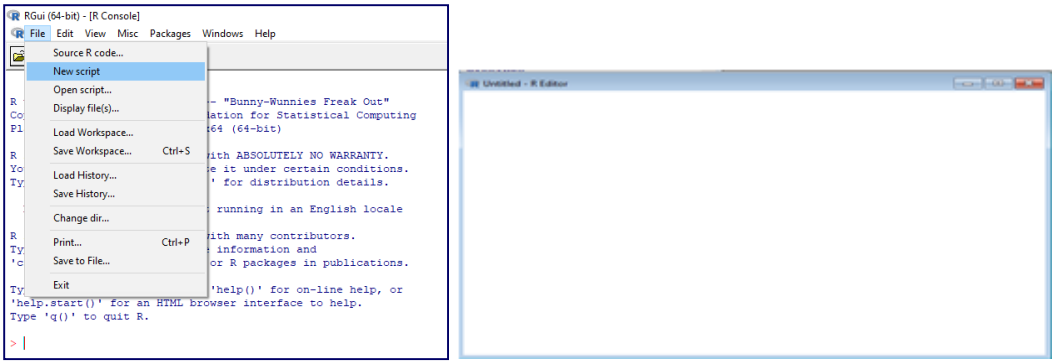
RStudio is a free, open-source IDE (integrated development environment) for R. It can be downloaded from https://www.rstudio.com/products/rstudio/download/. One must install R before installing RStudio. The interface is organized so that the user can clearly view graphs, data tables, R code, and output all at the same time.



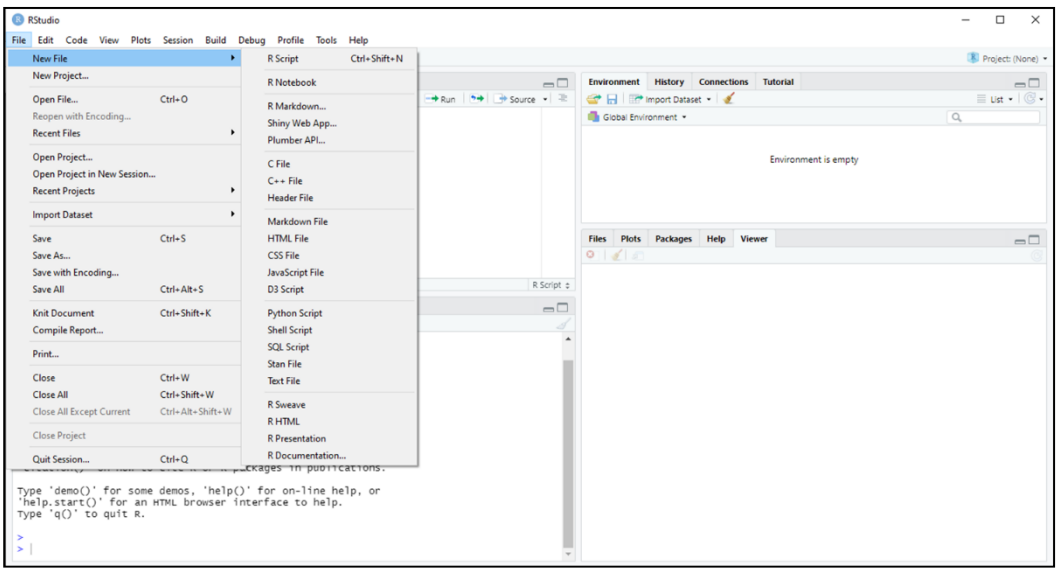
R Studio Interface

There are various ways for working in R:

- Work directly from the R editor to type in your script and execute the script completely (batch) or line-by-line (highlight and execute)
- Write script in an external editor (Notepad or software that interfaces with R) and execute in R by copy/paste or highlighting
- Beyond the native R GUI, external GUI can work with R to help in writing scripts, selecting functions, procedures, statistical tests, or graphics



Getting started: R



Getting started: RStudio

R is an expression language with a very simple syntax. It is case sensitive as are most UNIX based packages. For example, A and a are different symbols and refer to different variables. The set of symbols which can be used in R names depends on the operating system and country within which R is being run (technically on the locale in use). Normally all alphanumeric symbols are allowed (and in some countries this includes accented letters) plus '.' and '_', with the restriction that a name must start with '.' or a letter, and if it starts with '.' the second character must not be a digit. Elementary commands consist of either expressions or assignments. If an expression is given as a command, it is evaluated, printed (unless specifically made invisible), and the value is lost. An assignment evaluates an expression and passes the value to a variable but the result is not automatically printed. Commands

are separated either by a semi-colon (‘;’), or by a newline. Elementary commands can be grouped together into one compound expression by braces (‘{’ and ‘}’). Comments can be put almost anywhere, starting with a hashmark (‘#’), everything to the end of the line is a comment. If a command is not complete at the end of a line, R will give a different prompt, by default + on second and subsequent lines and continue to read input until the command is syntactically complete.

Getting and installing R

To be able to use R, it needs to be installed in computer. R is available for free download from any one of the mirror sites of Comprehensive R Archive Network (CRAN) in http://cran.r-project.org/. For downloading, it is better to select a mirror located nearer to you. R is available for installation in Windows/Macintosh/Unix platforms. To install R in a given machine, first double-click the downloaded file R.exe, then select language as `English'. R setup wizard window will appear. Select on `Next' and accept most of the default settings during the installation.

To start R, click on start menu → all programs → R → R 3.4.1 and a screen as shown in Fig. 1 appears. The white blank screen is called R Console and this is the place where all R codes are written and outputs appear, unless outputs are directed to some external files. There will be a toolbar at the top of the Console and a few menus in the R window. To know what the buttons on the toolbar does, hold your mouse on the button for some time, a description of the button will appear. There is an R editor which can be used to write and edit R codes. R editor window is just like a text editor with facilities for select, cut, copy, paste, typing text, deleting text etc. This window opens by clicking on File → New Script in the menu bar. The codes written in R editor window needs to be passed to the R console for execution by clicking on `Run line or selection button' on the toolbar in the R editor window.

Downloading and installing a package

To use an R package, download the package from CRAN and then install and load it in an R session. A package can be downloaded from within R or from outside R. On a Windows machine which is connected to internet, a package can be installed by clicking on Packages ! Install packages(s) from the menu bar. This will open a list of mirrors. Select a mirror and then, from the available list of packages in the website, select the desired packages. They will be installed into R.

One can also install the package from command line and then package can be loaded using following command.

> install.packages("agricolae")

> library(agricolae)

R Workspace

R workspace is temporary space on your CPU's RAM that "disappears" at the end of R session. It includes any user-defined objects (vectors, matrices, data frames, lists, functions). All data, analyses, output are stored as objects in the R workspace. This workspace is not saved on disk unless you tell R to do so. This means that your objects are lost when you close R and not save the objects, or worse when R or your system crashes on you during a session. When you close the RGui or the R console window, the system will ask if you want to save the workspace image. If you select to save the workspace image then all the objects in your current R session are saved in a file ".RData". ".RData" is a binary file located in the working directory of R, which is by default the installation directory of R. During your R session, you can also explicitly save the workspace image.

Go to the 'Session' menu and then select 'Save Workspace as'

> save.image("example1.Rdata")

If you have saved a workspace image and you start R the next time, it will restore the workspace. So all your previously saved objects are available again.

Go to the 'Session' menu and then select 'Load Workspace'.

> load.image("example1.Rdata")

- Windows uses a \ (left slash) to delineate locations in CPU:

  C:\Users\hp\Documents
- R uses / (right slash) to delineate locations in CPU:

  C:/Users/hp/Documents
- An alternative to R's / (single right) is \\ (two left) slashes:

  C:\\Users\\hp\\Documents
- There is no issue in the MAC OS/Linux as they have retained the / (right slash) as the basis for directory delineation
- Print the current working directory

  > getwd()
- List the objects in the current workspace

  > ls()
- Change to my directory

  > setwd(mydirectory)
- Display last 25 commands

  > history()
- Display all previous commands

  > history(max.show=Inf)
- Saving R workspace

  > x <- 5 # object x; x is assigned value 5

  > y <- 10 # object y; y is assigned value 10

  > z <- x+y # object z (addition of numbers x and y); z is assigned the value x+y

  > save(x, y, file = "example1_xy.RData") # save two specified objects x and y

  > save.image(file = "example1.RData") # save entire workspace
- Removing objects R workspace: Use rm()

```
> ls()
      [1]          "x" "y" "z"
> rm(x, y) # removes objects x and y
> ls()
[1] "z"
```

- Use load() to add previously saved objects or workspaces to your current R session.

```
> load(file = "example1.RData")
> ls()
[2]  "x" "y" "z"
```

Getting help with functions and features

R has an inbuilt help facility similar to the man facility of UNIX. To get more information on any specific named function, for example solve, the command is

```
> help(solve)
```

An alternative is

```
> ?solve
```

For a feature specified by special characters, the argument must be enclosed in double or single quotes, making it a “character string”: This is also necessary for a few words with syntacticmeaning including if, for and function.

```
> help("[[")
```

Either form of quote mark may be used to escape the other, as in the string "It’s important". Our convention is to use double quote marks for preference. On most R installations help is available in HTML format by running

```
> help.start()
```

which will launch a Web browser that allows the help pages to be browsed with hyperlinks. On UNIX, subsequent help requests are sent to the HTML-based help

system. The ‘Search Engine and Keywords’ link in the page loaded by help.start() is particularly useful as it is contains a high-level concept list which searches though available functions. It can be a great way to get your bearings quickly and to understand the breadth of what R has to offer. The help.search command (alternatively ??) allows searching for help in various ways. For example,

> ??solve

Try ?help.search for details and more examples.

The examples on a help topic can normally be run by

> example(topic)

Windows versions of R have other optional help systems: use

> ?help

Data types in R

There is a > symbol in the Console. The commands are typed after this symbol and then the Enter button needs to be pressed. When a command is written in the Console and the Enter button is pressed, R reads the commands and returns some results or some error message on the Console.

R is an object oriented language and therefore, all data types in R are some kind of object. Objects may be variables, vectors, matrices, arrays, character strings, functions, or more general structures built from such components. During an R session, objects are created and stored by name. Once can use the command

> objects()

to display the names of the objects which are currently stored within R. The collection of objects currently stored is called the workspace. One can remove objects using the function rm(). For example, the following code removes objects x and y form the workspace.

> rm(x, y)

An object created during an R session can be saved in a file for use in future R sessions. The entire workspace of an R session and the history of all the commands used during the session can also be saved. Some commonly encountered objects are discussed below.

Numbers

The most basic way to store a number is to make an assignment of a single number:

a <- 3

The "<-" tells R to take the number to the right of the symbol and store it in a variable whose name is given on the left. You can also use the "=" symbol. When you make an assignment R does not print out any information. If you want to see what value a variable has just type the name of the variable on a line and press the enter key:

```
> a
[1] 3
```

This allows you to do all sorts of basic operations and save the numbers:

```
> b <- sqrt(a*a+3)
> b
[1] 3.464102
```

Strings: One can not only limited to just storing numbers. We can also store strings. A string is specified by using quotes. Both single and double quotes will work:

```
> a <- "hello"
> a
[1] "hello"
> b <- c("hello","there")
> b
[1] "hello" "there"
> b[1]
[1] "hello"
```

Vectors

Simplest object in R is a vector. A vector is a collection of elements. For example, creates a vector of 5 numbers.

```
> x <- c(10.4, 5.6, 3.1, 6.4, 21.7)
```

Matrices

A matrix object also is a collection of elements but it has two dimensions. They can also be numeric, character or logical in nature. Following is an example of creating a matrix.

```
> x=matrix(c("a","b","c","d"),nrow=2)
> x
[,1] [,2]
[1,] "a" "c"
[2,] "b" "d"
```

Matrix facilities: As noted above, a matrix is just an array with two subscripts. However it is such an important special case it needs a separate discussion. R contains many operators and functions that are available only for matrices. For example t(X) is the matrix transpose function, as noted above. The functions nrow(A) and ncol(A) give the number of rows and columns in the matrix A respectively.

The operator %*% is used for matrix multiplication. An n by 1 or 1 by n matrix may of course be used as an n-vector if in the context such is appropriate. Conversely, vectors which occur in matrix multiplication expressions are automatically promoted either to row or column vectors, whichever is multiplicatively coherent, if possible, (although this is not always unambiguously possible, as we see later).

If, for example, A and B are square matrices of the same size, then

```
> A * B
```

is the matrix of element by element products and

> A %*% B

is the matrix product. If x is a vector, then

> x %*% A %*% x

is a quadratic form. The function crossprod() forms "crossproducts", meaning that crossprod(X, y) is the same as t(X) %*% y but the operation is more efficient.

Forming partitioned matrices with cbind() and rbind(): As we have already seen informally, matrices can be built up from other vectors and matrices by the functions cbind() and rbind(). Roughly cbind() forms matrices by binding together matrices horizontally, or column-wise, and rbind() vertically, or row-wise.

In the assignment

> X <- cbind(arg_1, arg_2, arg_3, ...)

the arguments to cbind() must be either vectors of any length, or matrices with the same column size, that is the same number of rows. The result is a matrix with the concatenated arguments arg 1, arg 2, . . . forming the columns. If some of the arguments to cbind() are vectors they may be shorter than the column size of any matrices present, in which case they are cyclically extended to match the matrix column size (or the length of the longest vector if no matrices are given).

The function rbind() does the corresponding operation for rows. In this case any vector argument, possibly cyclically extended, are of course taken as row vectors. Suppose X1 and X2 have the same number of rows. To combine these by columns into a matrix X, together with an initial column of 1s we can use

> X <- cbind(1, X1, X2)

The result of rbind() or cbind() always has matrix status. Hence cbind(x) and rbind(x) are possibly the simplest ways explicitly to allow the vector x to be treated as a column or row matrix respectively.

Arrays

Arrays are multi-dimensional generalization of vectors and matrices. A two-dimensional array is a matrix. Arrays can have more than two dimensions. Arrays can be constructed from vectors by the array function, which has the form

> Z <- array(data_vector, dim_vector)

For example, if the vector h contains 24 or fewer, numbers then the command

> Z <- array(h, dim=c(3,4,2))

Factors

Factor objects are used to specify categorical or classificatory or grouping variables. For example, males and females are two levels of a variable gender. Then gender can be thought of a factor object.

```
> gender=c("M", "F", "M")
> gender=as.factor(gender)
> levels(gender)
[1] "F" "M"
```

Factor variables are particularly useful in analysis of variance and in linear model with grouping variables.

Lists

A list is a collection of objects where each object can be of different type. For example, a list can have first object as a vector, second object as a matrix and third object as a data frame.

```
> mylist=list(x=c(10,20,30),y=matrix(1:6,nrow=3))
> mylist
$x
[1] 10 20 30
$y
[,1] [,2]
[1,] 1 4
[2,] 2 5
```

[3,] 3 6

> mylist[[1]]

Data frames

A data frame is a two dimensional object. But unlike matrices, different columns of data frame can be different types, for example some columns can be numeric, some columns can be character, some columns can be factors. Here a column generally refers to a variable.

> age=c(20,25,28,30,26)

> weight=c(50,53,54,55,51)

> mydata=data.frame(age,weight)

> mydata

age weight

1 20 50

2 25 53

3 28 54

4 30 55

5 26 51

The data.frame() function is used to create a data frame.

Functions

Functions in R are a kind of objects which takes one or more inputs and produces some result(s) as output. R has a number of in-built functions. R also provides facility to create new functions by users. R has huge number of in-built functions. As a simple example, to obtain the mean and variance of a set of numbers 10, 13, 21, 34, 51, 32, 45, 32, 17, 29, 41, 52, the following code can be used.

> x=c(10,13,21,34,51,32,45,32,17,29,41,52)

> mean(x)

[1] 31.41667

> var(x)

[1] 200.9924

Here, c(), mean() and var() are in-built functions of R. The function c() assigns those numbers to the object x. The commands mean(x) and var(x) computes the mean and variance of an object x. Here, x is the input, also called argument, to the function mean() and var().

Reading data from files

Large data objects will usually be read as values from external files rather than entered during an R session at the keyboard. R input facilities are simple and their requirements are fairly strict and even rather inflexible. There is a clear presumption by the designers of R that you will be able to modify your input files using other tools, such as file editors or Perl1 to fit in with the requirements of R. Generally this is very simple. If variables are to be held mainly in data frames, as we strongly suggest they should be, an entire data frame can be read directly with the read.table() function. There is also a more primitive input function, scan(), that can be called directly. For more details on importing data into R and also exporting data, see the R Data Import/Export manual.

Reading data from a spreadsheet

To read an entire data frame directly, the external file will normally have a special form. The first line of the file should have a name for each variable in the data frame. Each additional line of the file has as its first item a row label and the values for each variable.

```
> mydata2=read.table("rainfall.txt",header=TRUE,sep=",")
```

Reading data from Excel sheet

```
> library(xlsx)
> read.xlsx("myfile.xlsx", sheetName = "Sheet1")
```

Reading data from a particular Web page

```
> webdata=read.table("http://data.princeton.edu/wws509/datasets/effort.dat")
> webdata
```

Some Hands on with R

Examining the distribution of a set of data

Given a (univariate) set of data we can examine its distribution in a large number of ways. The simplest is to examine the numbers. Two slightly different summaries are given by summary function.

```
> attach(faithful)
> summary(eruptions)
Min. 1st Qu. Median Mean 3rd Qu. Max.
1.600 2.163 4.000 3.488 4.454 5.100
> fivenum(eruptions)
[1] 1.6000 2.1585 4.0000 4.4585 5.1000
> stem(eruptions)
> hist(eruptions)
## make the bins smaller, make a plot of density
> hist(eruptions, seq(1.6, 5.2, 0.2), prob=TRUE)
> lines(density(eruptions, bw=0.1))
> rug(eruptions) # show the actual data points
```

Fitting Linear models

The basic function for fitting ordinary multiple models is lm(), and a streamlined version of the call is as follows:

```
> fitted.model <- lm(formula, data = data.frame)
```

For example

```
> fm2 <- lm(y ~ x1 + x2, data = production)
```

would fit a multiple regression model of y on x1 and x2 (with implicit intercept term).

Fitting Generalized linear model

Since the distribution of the response depends on the stimulus variables through a single linear function only, the same mechanism as was used for linear models can still be used to specify the linear part of a generalized model. The family has to be specified in a different way. The R function to fit a generalized linear model is glm() which uses the form

```
> fitted.model <- glm(formula, family=family.generator, data=data.frame)
```

Linear equations and inversion

Solving linear equations is the inverse of matrix multiplication. When after

```
> b <- A %*% x
```

only A and b are given, the vector x is the solution of that linear equation system. In R,

```
> solve(A,b)
```

solves the system, returning x (up to some accuracy loss).

Eigenvalues and eigenvectors

The function eigen(Sm) calculates the eigenvalues and eigenvectors of a symmetric matrix Sm. The result of this function is a list of two components named values and vectors. The assignment

```
> ev <- eigen(Sm)
```

will assign this list to ev. Then ev$val is the vector of eigenvalues of Sm and ev$vec is the matrix of corresponding eigenvectors. Had we only needed the eigenvalues we could have used the assignment:

```
> evals <- eigen(Sm)$values
```

evals now holds the vector of eigenvalues and the second component is discarded. If the expression

```
> eigen(Sm)
```

is used by itself as a command the two components are printed, with their names. For large matrices it is better to avoid computing the eigenvectors if they are not needed by using the expression

> evals <- eigen(Sm, only.values = TRUE)$values

Generating Plots

One of the most frequently used plotting functions in R is the plot() function. This is a generic function: the type of plot produced is dependent on the type or class of the first argument.

plot(x, y)

plot(xy)

If x and y are vectors, plot(x, y) produces a scatterplot of y against x. The same effect can be produced by supplying one argument (second form) as either a list containing two elements x and y or a two-column matrix.

plot(x) If x is a time series, this produces a time-series plot. If x is a numeric vector, it produces a plot of the values in the vector against their index in the vector. If x is a complex vector, it produces a plot of imaginary versus real parts of the vector elements.

Descriptive Statistics

Descriptive statistics investigates the variables separately. Various descriptive statistics can be computed by using in-built R functions as given below.

Name of function	Use of function
mean	calculates the mean of an input
median	calculates the median of an input
var	calculates the variance of an input
sd	calculates the standard deviation of an input
IQR	calculates the interquartile range of an input
min	calculates the minimum value of an input
max	calculates the maximum of an input
range	returns a vector containing the minimum and maximum of all given arguments
summary	returns a vector containing a mixture of the above functions (minimum, first quartile, median, mean, third quartile, maximum)

```
> data(trees)
> head(trees)
  Girth Height Volume
1   8.3     70   10.3
2   8.6     65   10.3
3   8.8     63   10.2
4  10.5     72   16.4
5  10.7     81   18.8
6  10.8     83   19.7

> summary(trees)
     Girth           Height       Volume
 Min.   : 8.30   Min.   :63   Min.   :10.20
 1st Qu.:11.05   1st Qu.:72   1st Qu.:19.40
 Median :12.90   Median :76   Median :24.20
 Mean   :13.25   Mean   :76   Mean   :30.17
 3rd Qu.:15.25   3rd Qu.:80   3rd Qu.:37.30
 Max.   :20.60   Max.   :87   Max.   :77.00
```

> mean(trees$Height)
[1] 76
> sd(trees$Height)
[1] 6.371813
> range(trees$Height)
[1] 63 87

Graphics

Histogram plots the frequencies that data appears within certain ranges.

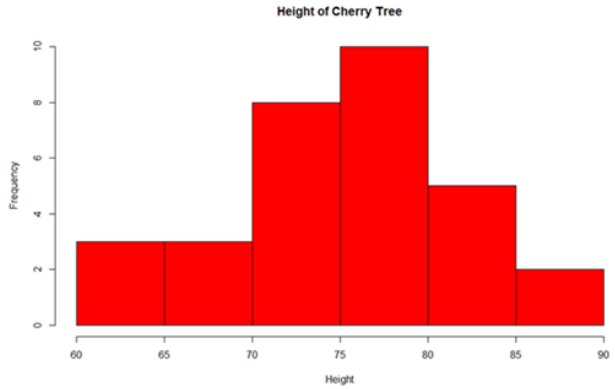
> data(trees)

***Add a title*:** The “main” statement will give the plot an overall heading.

***Add axis labels*:** Use “xlab” and “ylab” to label the X and Y axes, respectively.

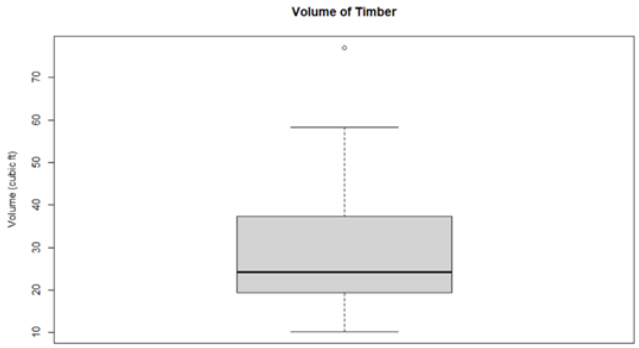
***Changing colors*:** Use the col statement

hist(trees$Height, main="Height of Cherry Tree", xlab="Height", ylab="Frequency", col="red")



A boxplot provides a graphical view of the median, quartiles, maximum, and minimum of a data set.

> boxplot(trees$Volume,main='Volume of Timber', ylab='Volume (cubic ft)')



Partitioning the Graphics Window

A useful facility before beginning is to divide a page into smaller pieces so that more than one figure can be displayed graphically.

par: used to set or query graphics parameters

par(mfrow=c(2,2))

This will create a window of graphics with 2 rows and 2 columns.

The windows are filled up row-wise.

Use mfcol instead of mfrow to fill up column-wise.

> data(trees)

> par(mfrow=c(2,2))

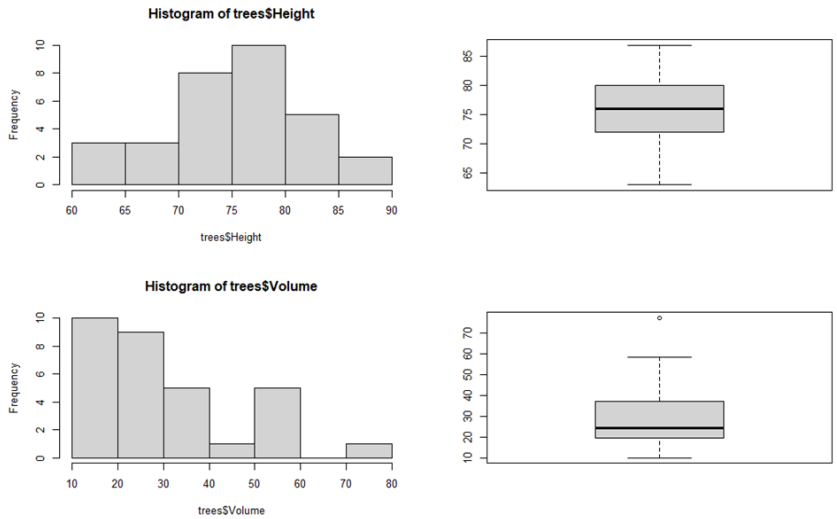
> hist(trees$Height)

> boxplot(trees$Height)

> hist(trees$Volume)

> boxplot(trees$Volume)

> par(mfrow=c(1,1))



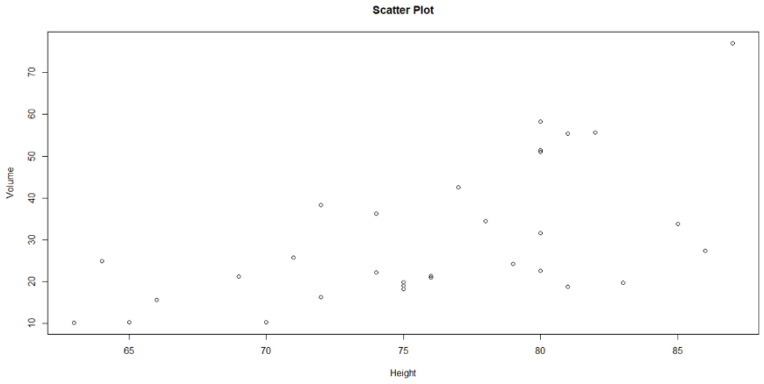
- Use layout()

Example: layout(matrix(1:4,2,2)) will partition the window into 4 equal parts

One can view the layout with layout show (n = 4)

A **scatter plot** provides a graphical view of the relationship between two sets of numbers.

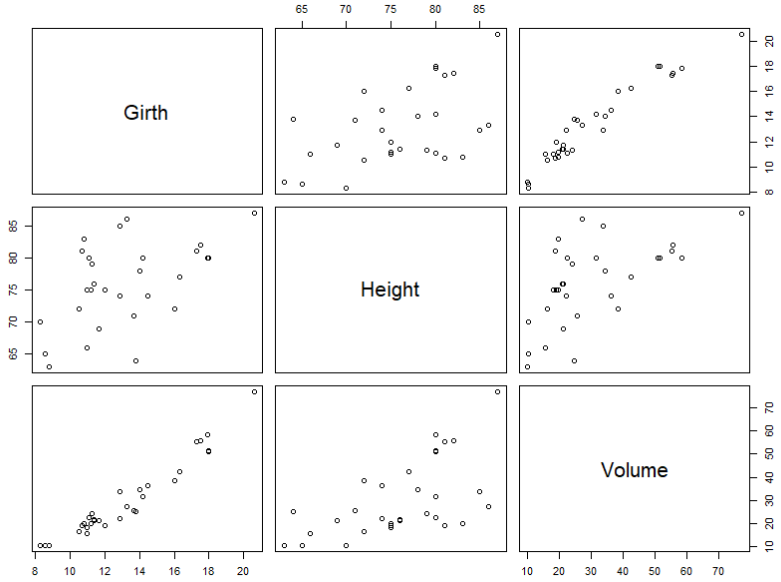
> plot(trees$Height, trees$Volume, xlab="Height", ylab="Volume", main="Scatter Plot", pch=20)



parameter pch stands for ‘plotting character’.

> pairs(trees)

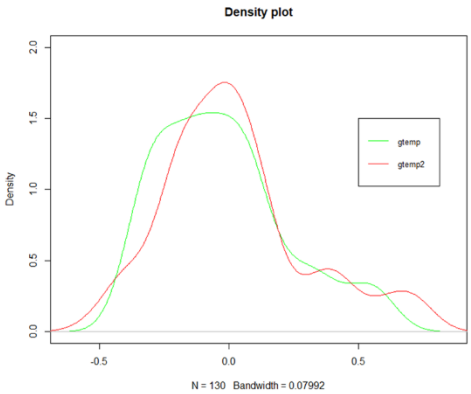
A matrix of scatterplots is produced.



Density plot is a representation of the distribution of a numeric variable that uses a kernel density estimate to show the probability density function of the variable.

In R Language we use the density() function which helps to compute kernel density estimates.

```
> plot(density(gtemp), ylim=c(0, 2), col = "green",main = "Density plot")
> lines(density(gtemp2), col="red")
> legend(0.5,1.5, cex=0.8, c("gtemp", "gtemp2"), col=c("green", "red"), lty=1:1)
```



R Packages for analysis of biological sequence analysis and retrieval of genomic data

- seqinr
- tidysq
- biomartr
- rentrez

R packages for sequence alignment

- Biostrings
- msa
- msaR
- ggmsa
- AlignStat

R Packages for differential gene expression analysis of microarray data

- amda
- maGUI
- maEndToEnd
- limma

- GEOlimma

R packages for differential gene expression analysis of RNA-Seq data

- edgeR
- DESeq2
- ideal
- DEvis

R Packages for protein structure analysis

- Bio3D
- Rpdb
- XLmap

R packages for protein-protein interaction graphs

- graph
- RBGL
- Rgraphviz
- crosstalkr
- igraph

R Packages for proteomics data analysis

- RforProteomcs
- protti
- Proteus
- DanteR
- MSstats
- MSqRob
- DAPAR

R Packages for metagenomics data analysis

- MicrobiomeExplorer
- matR
- MegaR

R Packages for GWAS and genomic selection

- statgenGWAS

- GWASTools
- BlueSNP
- rrBLUP
- lme4GS
- BWGS
- GSelection
- learnMET
- GAPIT

References

Giorgi, F. M., Ceraolo, C. and Mercatelli, D. (2022). The R Language: An Engine for Bioinformatics and Data Science. *Life (Basel, Switzerland)*, **12(5)**, 648. https://doi.org/10.3390/life12050648

Ihaka, R. and Gentleman, R (1996). R: A Language for Data Analysis and Graphics. *Journal of Computational and Graphical Statistics*, **5**, 299–314. doi: 10.1080/10618600.1996.10474713

W. N. Venables, D. M. Smith and the R Core Team. *An Introduction to R. Notes on R: A Programming Environment for Data Analysis and Graphics*, Version 4.2.2 (2022-10-31), URL: https://cran.r-project.org/doc/manuals/r-release/R-intro.pdf

https://en.wikipedia.org/wiki/R_(programming_language)

https://en.wikipedia.org/wiki/Bioconductor

https://www.cran.r-project.org/

https://www.bioconductor.org/

Overview of RNA-Seq Data Analysis

Mohammad Samir Farooqi and Sudhir Srivastava

Introduction

The advent of Next-Generation Sequencing (NGS) technology has transformed genomic studies. One important application of NGS technology is the study of the *transcriptome*, which is defined as the complete collection of all the RNA molecules in a cell. Various types of RNA that have been classified so far are shown in **Fig. 1**. All of these molecules are called *transcripts* since they are produced by process of transcription.

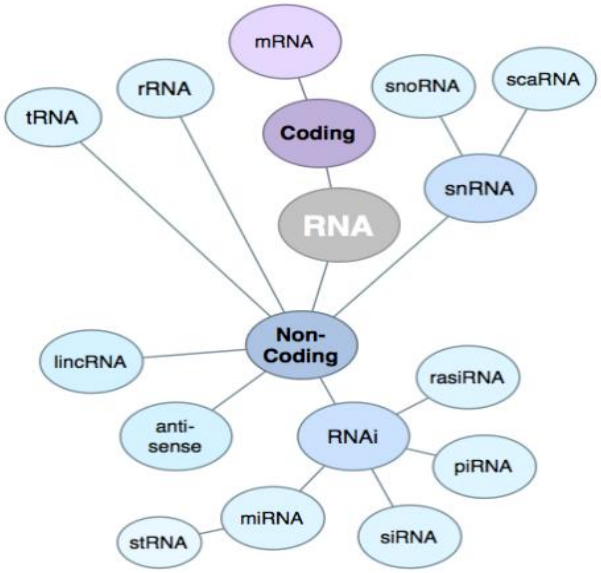


Fig. 1: Different types of RNA

(Image source: http://scienceblogs.com/digitalbio/2011/01/08/next-gene-sequencing)

Understanding the transcriptome is essential for interpreting the functional elements of the genome and revealing the molecular constituents of cells and tissues, and also for understanding development and disease [1]. The main purpose of transcriptomics are: to catalogue all species of transcript, including mRNAs, non-coding RNAs and small RNAs; to determine the transcriptional structure of genes, in terms of their start sites, 5′ and 3′ ends, splicing patterns and other post-transcriptional modifications; and to quantify the changing expression levels of each transcript during development and under different conditions.

The study of transcriptome is carried out through sequencing of RNAs. *RNA sequencing (RNA-Seq)* is a powerful method for discovering, profiling, and quantifying RNA transcripts [2]. RNA-Seq uses NGS datasets to obtain sequence reads from millions of individual RNAs. The RNA-Seq analysis is performed in several steps: First, all genes are extracted from the

reference genome (using annotations of type *gene*). Other annotations on the gene sequences are preserved (e.g.CDS information about coding sequences etc). Next, all annotated transcripts (using annotations of type *mRNA*) are extracted [3]. If there are several annotated splice variants, they are all extracted. An example is shown in below **Fig. 2(a).**

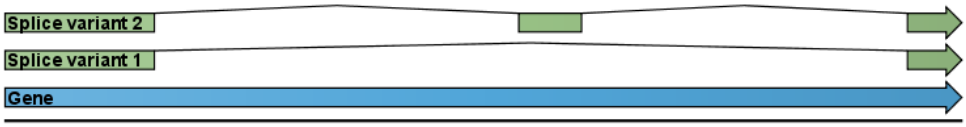


Fig. 2(a): A simple gene with three exons and two splice variants.

The given example is a simple gene with three exons and two splice variants. The transcripts are extracted as shown in **Fig. 2(b).**

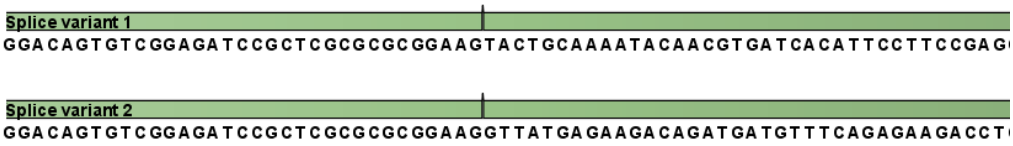


Fig. 2(b): All the exon-exon junctions are joined in the extracted transcript.

Next, the reads are mapped against all the transcripts plus the entire gene [see **Fig. 2(c)**].

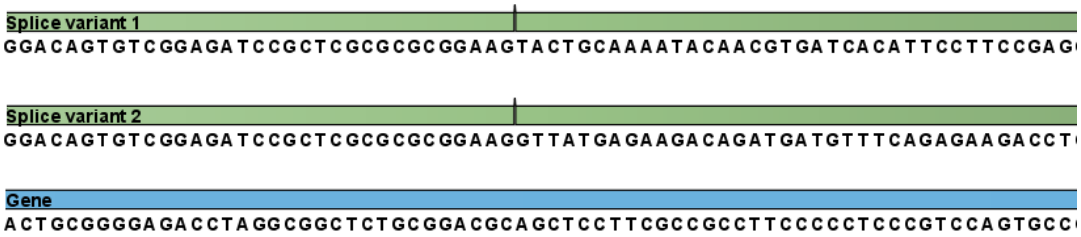


Fig. 2(c): The reference for mapping: all the exon-exon junctions and the gene.

(Image source: CLC Genomic workbench tutorials)

From this mapping, the reads are categorized and assigned to the genes and expression values for each gene and each transcript are calculated and putative exons are then identified.

RNA Sequencing Experiment

In a standard RNA-seq experiment, a sample of RNA is converted to a library of complementary DNA fragments and then sequenced on a high-throughput sequencing platform, such as Illumina's Genome Analyzer, SOLiD or Roche 454 [4]. Millions of short sequences, or reads, are obtained from this sequencing and then mapped to a reference genome (**Fig. 3**). The count of reads mapped to a given gene measures the expression level of this gene. The unmapped reads are usually discarded and mapped reads for each sample are assembled into gene-level, exon-level or transcript-level expression summaries, depending on the objectives of the experiment. The count of reads mapped to a given gene/exon/transcript measures the expression level for this region of the genome or transcriptome.

One of the primary goals for most RNA-seq experiments is to compare the gene expression levels across various treatments. A simple and common RNA-seq study involves two

treatments in a randomized complete design, for example, treated versus untreated cells, two different tissues from an organism, plants, etc. In most of the studies, researchers are particularly interested in detecting gene with differential expressions (DE). A gene is declared differentially expressed if an observed difference or change in read counts between two experimental conditions is statistically significant, i.e. if the difference is greater than what would be expected just due to random variation [5]. Detecting DE genes can also be an important pre-step for subsequent studies, such as clustering gene expression profiles or testing gene set enrichments.

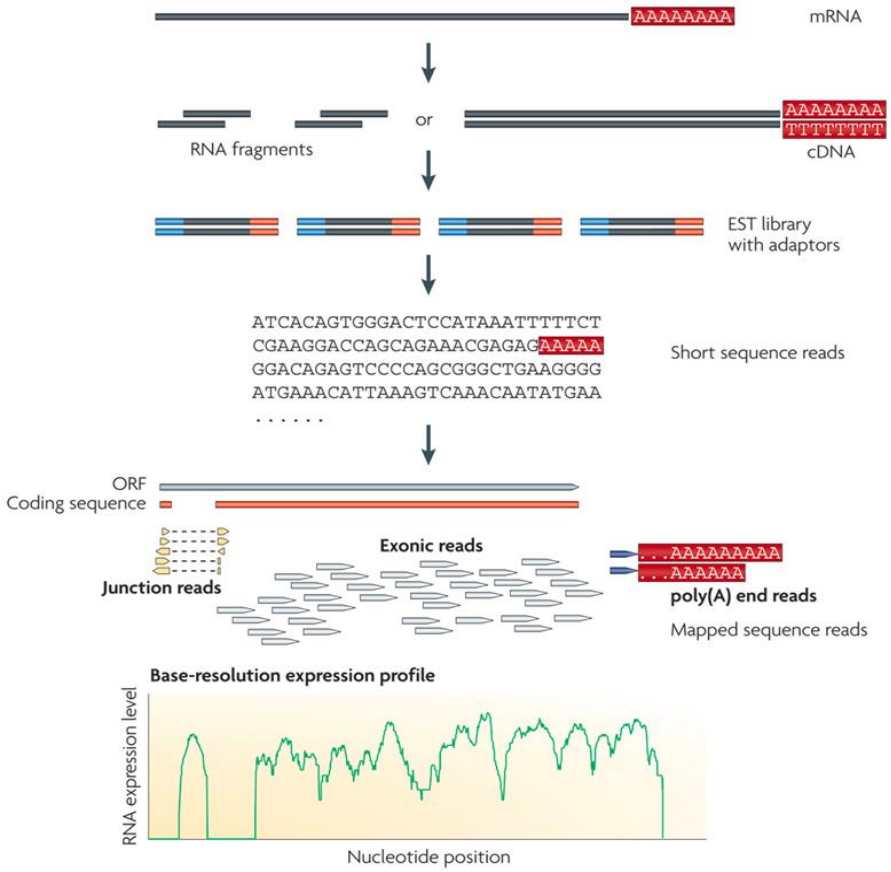


Fig. 3: General RNA-seq experiment. mRNA is converted to cDNA, and fragments from that library are used to generate short sequence reads. Those reads are assembled into contigs which may be mapped to reference sequences (Wang et al., 2009).

Analysing RNA-Seq data

RNA-seq experiments must be analyzed with robust, efficient and statistically correct algorithms. Fortunately, the bioinformatics community has been striving hard at work for incorporating mathematics, statistics and computer science for RNA-seq and building these ideas into software tools. RNA-seq analysis tools generally fall into three categories: (i) those for read alignment; (ii) those for transcript assembly or genome annotation; and (iii) those for transcript and gene quantification. Some of the open source softwares available for RNA-seq analysis are as follows:

- **Data preprocessing**
    - Fastx toolkit
    - Samtools

- **Short reads aligners**
    - Bowtie, TOPHAT, Stampy, BWA, Novoalign, etc
- **Expression studies**
    - Cufflinks package
    - R packages (DESeq, edgeR, *more…*)
- **Visualisation**
    - CummeRbund, IGV, Bedtools, UCSC Genome Browser, etc.

Besides there are commercially data analysis pipelines like GenomeQuest, CLCBio etc available for researchers to use. The most commonly used pipeline is to identify protein coding genes by aligning RNA-Seq data to annotate data from sources like RefSeq. After generating the alignments, the number of aligning sequences is counted for each position. Since each alignment represents a transcript, the alignments allow to count the number of RNA molecules produced from every gene.

Using NGS technology, RNA-Seq enables to count the number of reads that align to one of thousands of different cDNAs, producing results similar to those of gene expression microarrays [6]. Sequences generated from an RNA-Seq experiment are usually mapped to libraries of known exons in known transcripts. RNA-Seq can be used for discovery applications such as identifying alternative splicing events, allele-specific expression, and rare and novel transcripts [7]. The sequencing output files (compressed FASTQ files) are the input for secondary analysis. Reads are aligned to an annotated reference genome, and those aligning to exons, genes and splice junctions are counted. The final steps are data visualisation and interpretation, consisting of calculating gene- and transcript-expression and reporting differential expression. A general Bioinformatics workflow to map transcripts from RNA-seq data is shown in **Fig. 4**.

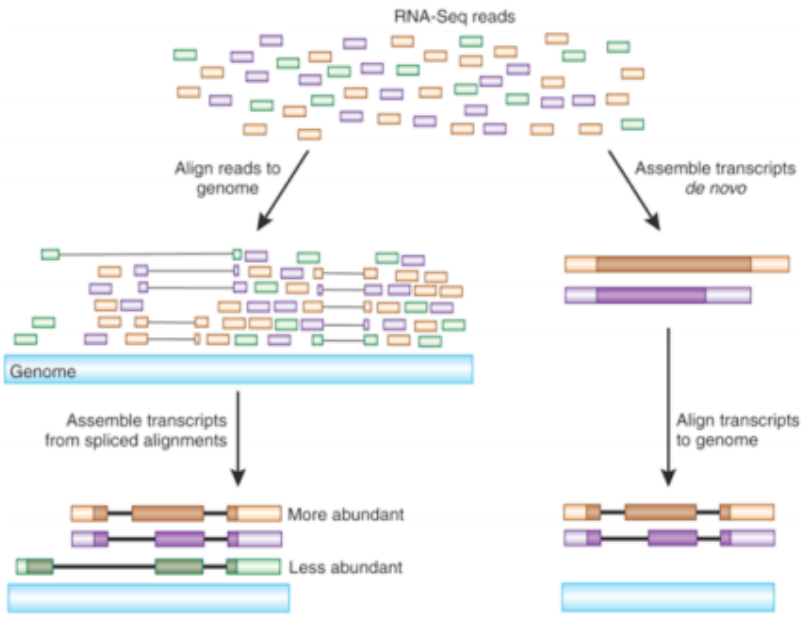


Fig. 4: RNA-seq workflow (Adapted from Advancing RNA-Seq analysis Brian J. Haas and Michael C. Zody Nature Biotechnology 28, 421-423 (2010)

RPKM (Reads per KB per million reads)

RNA-Seq provides quantitative approximations of the abundance of target transcripts in the form of counts. However, these counts must be normalized to remove technical biases inherent in the preparation steps for RNA-Seq, in particular the length of the RNA species and the sequencing depth of a sample. The most commonly used is RPKM (Reads Per Kilobase of exon model per Million mapped reads). The RPKM measure of read density reflects the molar concentration of a transcript in the starting sample by normalizing for RNA length and for the total read number in the measurement [8]. RPKM is mathematically represented as:

$$\text{RPKM} = \frac{total\ exon\ reads}{mapped\ reads\ (millions)\ \text{X}\ exon\ length\ (KB)}$$

Total exon reads

This is the number of reads that have been mapped to a region in which an exon is annotated for the gene or across the boundaries of two exons or an intron and an exon for an annotated transcript of the gene. For eukaryotes, exons and their internal relationships are defined by annotations of type mRNA.

Exon length

This is calculated as the sum of the lengths of all exons annotated for the gene. Each exon is included only once in this sum, even if it is present in more annotated transcripts for the gene. Partly overlapping exons will count with their full length, even though they share the same region.

Mapped reads

The total gene reads for a gene is the total number of reads that after mapping have been mapped to the region of the gene. A gene's region is that comprised of the flanking regions, the exons, the introns and across exon-exon boundaries of all transcripts annotated for the gene. Thus, the sum of the total gene reads numbers is the number of mapped reads for the sample.

Applications of RNA-seq

This technique can be used to:

- Measure gene expression
- Transcriptome assembly, gene discovery and annotation
- Detect differential transcript abundances between tissues, developmental stages, genetic backgrounds, and environmental conditions
- Characterize alternative splicing, alternative polyadenylation, and alternative transcription.

Future Directions

Although RNA-Seq is still in the infancy stages of use, it has clear advantages over previously developed transcriptomic methods. Compared with microarray, which has been the dominant approach of studying gene expression in the last two decades, RNA-seq technology has a wider measurable range of expression levels, less noise, higher throughput, and more information to detect allele-specific expression, novel promoters, and isoforms [9]. For these reasons, RNA-seq is gradually replacing the array-based approach as the major platform in gene expression studies. The next big challenge for RNA-Seq is to target more complex transcriptomes to identify and track the expression changes of rare RNA isoforms from all genes. Technologies that will advance achievement of this goal are pair-end sequencing, strand-specific sequencing and the use of longer reads to increase coverage and depth. As the cost of sequencing continues to fall, RNA-Seq is expected to replace microarrays for many applications that involve determining the structure and dynamics of the transcriptome.

References

1. https://www.genome.gov/13014330
2. Wang Z., Gerstein M., Synder M. (2009). Rna-seq: a revolutionary tool for transciptomics, Nat Rev Genet 10(1): 57–63.
3. http://scienceblogs.com/digitalbio/2011/01/08/next-gene-sequencing-results-a/
4. Shendure J, Ji H (2008) Next-generation RNA sequencing. Nature Biotechnology 26: 2514-2521
5. Anders S, Huber W (2010). Differential expression analysis for sequence count data. Genome Biol. 11:R106.
   Illumina, Inc,. (2011). Getting started with RNA-Seq Data Analysis. Pub. No. 470-2011-003.
6. Illumina, Inc,. (2011). RNA-Seq Data Comparison with Gene Expression Microarrays. A cross-platform comparison of differential gene expression analysis. Pub. No. 470-2011-004
7. Yaqing Si (2012). Statistical analysis of RNA-seq data from next-generation sequencing technology. PhD thesis. Iowa State University, Ames, Iowa.
8. Mortazavi, A., Williams, B. A., McCue, K., Schaeffer, L., and Wold, B. (2008). Mapping and quantifying mammalian transcriptomes by rna-seq. Nat Methods, 5(7):621-628.
9. Wang L., Si Y., Dedow L.K., Shao Y., Liu P., Brutnell T.P. (2010). A low-cost library construction protocol and data analysis pipeline for Illumina-based strand-specific multiplex RNA-seq. PLoS One 6(10):e26426.
10. Brian J. H. and Michael C. Z. (2010). Advancing RNA-Seq analysis Nature Biotechnology 28, 421-423.

Differential Gene Expression Analysis in RNA-Seq Data using R

Soumya Sharma[1]

[1]**ICAR-IASRI**

Identification of differentially expressed genes from the RNA-Seq data is an important area of bioinformatics data analysis. There are several packages available in R to carry out the differential gene expression analysis, like **DESeq2 (**Love et al., 2014), **edgeR (**Robinson et al., 2010), **limma (**Smyth et al., 2005) *etc.* After preprocessing and quantification of reads in RNA-Seq data, we get a matrix of read counts of each gene in every sample. Then we can use the "**DESeq2**" package to identify differentially expressed genes. Here, we demonstrate the differential gene expression analysis with R using a sample dataset available in the R package **airway (**Himes et al., 2014) in following steps.

i) Download the sample dataset from the "**airway**" package. The package contains 2 data files. One file contains read counts of 64102 genes in 8 samples obtained from the RNA-Seq experiment on 4 primary human airway smooth muscle cell lines treated with 1 micromolar dexamethasone for 18 hours. Another file contains sample-wise metadata information, *viz.*, treated or untreated. Import the count matrix and metadata file into RStudio.

R code to collect sample dataset from "airway" package:

```
# installing Bioconductor packages
if (!requireNamespace("BiocManager", quietly=TRUE))
  install.packages("BiocManager")
   BiocManager::install("airway")
library(airway)
data(airway)
airway
sample_info <- as.data.frame(colData(airway))
sample_info <- sample_info[,c(2,3)]
sample_info$dex <- gsub('trt', 'treated', sample_info$dex)
sample_info$dex <- gsub('untrt', 'untreated', sample_info$dex)
```

```
names(sample_info) <- c('cellLine', 'dexamethasone')

# Get the samplewise metadata file

write.table(sample_info, file = "/sample_info.csv", sep = ',', col.names = T, row.names = T, quote = F)

# Get the matrix of read counts for each gene in every sample

countsData <- assay(airway)

write.table(countsData, file = "/counts_data.csv", sep = ',', col.names = T, row.names = T, quote = F)
```

ii) Then we have to load the package "**DESeq2**" to perform the subsequent differential gene expression analysis. We have to create a DESeqDataSet object and then run the 'DESeq()' function to perform the said analysis.

Differential gene expression analysis using the "DESeq2" package in R

```
BiocManager::install("DESeq2")

library(DESeq2)

# read in counts data

counts_data <- read.csv('/counts_data.csv')

# read in sample info

colData <- read.csv('/sample_info.csv')

# making sure the row names in colData matches to column names in counts_data

all(colnames(counts_data) %in% rownames(colData))

# are they in the same order?

all(colnames(counts_data) == rownames(colData))

dds <- DESeqDataSetFromMatrix(countData = counts_data, colData = colData, design = ~ dexamethasone)

dds

#pre-filtering: removing rows with low gene counts

# keeping rows that have at least 10 reads total

keep <- rowSums(counts(dds)) >= 10

dds <- dds[keep,]

# set the factor level
```

```
dds$dexamethasone <- relevel(dds$dexamethasone, ref = "untreated")
# --------Run DESeq ----------------------
dds <- DESeq(dds)
res <- results(dds)
res
summary(res)
res0.01 <- results(dds, alpha = 0.01) # When padj = 0.01
summary(res0.01)
```

Here, we are trying to find the genes which are differentially expressed in Dexamethasone treated conditions as compared to untreated conditions. Hence, the reference level is set as 'untreated'. After the analysis, the result contains base means, $\log_2$FoldChange values, p-values, adjusted p-values, *etc.* for each gene. If at 1% level, the adjusted p-value for a gene is found as > 0.01, it means the result has been obtained purely by chance, *i.e.*, a non-significant result. Otherwise, that gene is differentially expressed if the adjusted p-value is < 0.01. In the latter case, if the log2FoldChange value is > 0, the gene is upregulated and if it is < 0, then that gene is downregulated. Thus, we can find out differentially expressed genes using R.

iii) Visualization of differentially expressed genes in R. After identifying differentially expressed genes, we can visualize the result in terms of various plots such as MA plot, volcano plot, heatmap, *etc*. Several R packages are available to develop these plots. MA plot can be generated using the 'plotMA()' function. We can use the "**ggplot2**" package to develop volcano plot. Similarly, R package "**heatmap2", "pheatmap"** *etc*. are useful to create heatmaps. MA plot (fig 1), volcano plot (fig 2) and heatmap (fig 3) created from the result of the previous analysis.

R code to visualize the result of differential gene expression analysis

```
# MA plot
plotMA(res)
# Volcano plot
library(ggplot2)
library(tidyverse)
```

```
df<-as.data.frame(res)

df$diffexpressed <- "non-significant"

# if log2Foldchange > 0 and padj < 0.01, set as "UP"

df$diffexpressed[df$log2FoldChange > 0 & df$padj < 0.01] <- "UP"

# if log2Foldchange < 0 and padj < 0.01, set as "DOWN"

df$diffexpressed[df$log2FoldChange < 0 & df$padj < 0.01] <- "DOWN"

ggplot(df, aes(log2FoldChange, -log10(padj), col=
diffexpressed))+geom_point()+scale_color_manual(values = c("red", "black", "green"))


# Developing Heatmap of first 10 genes for better demonstration

library(pheatmap)

library(RColorBrewer)

breaksList = seq(-0.4, 0.5, by = 0.04)

rowLabel = row.names(counts_data[1:10,])

pheatmap(df$log2FoldChange[1:10], color = colorRampPalette(c("dark blue", "white",
"yellow"))(25), breaks = breaksList, border_color = "black", cellheight = 25, cellwidth = 25,
cluster_rows = F,cluster_cols = F, fontsize = 12, labels_row = rowLabel)
```

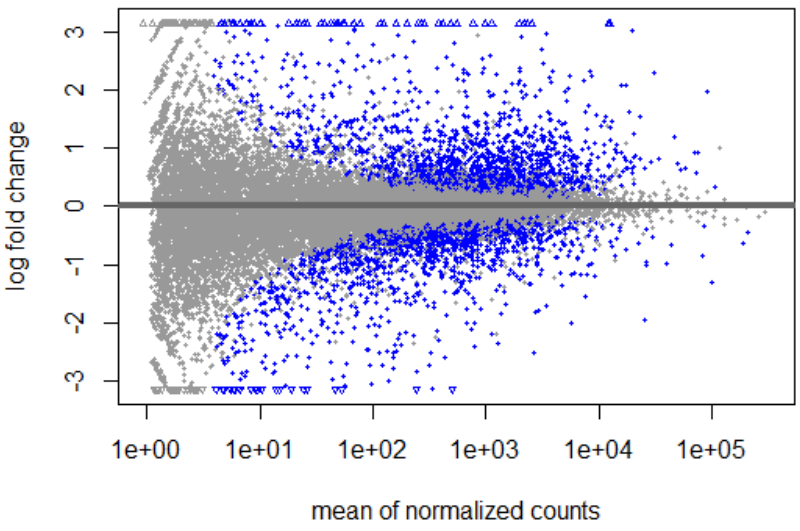


Fig 1: MA plot showing significantly upregulated and downregulated genes as blue dots.

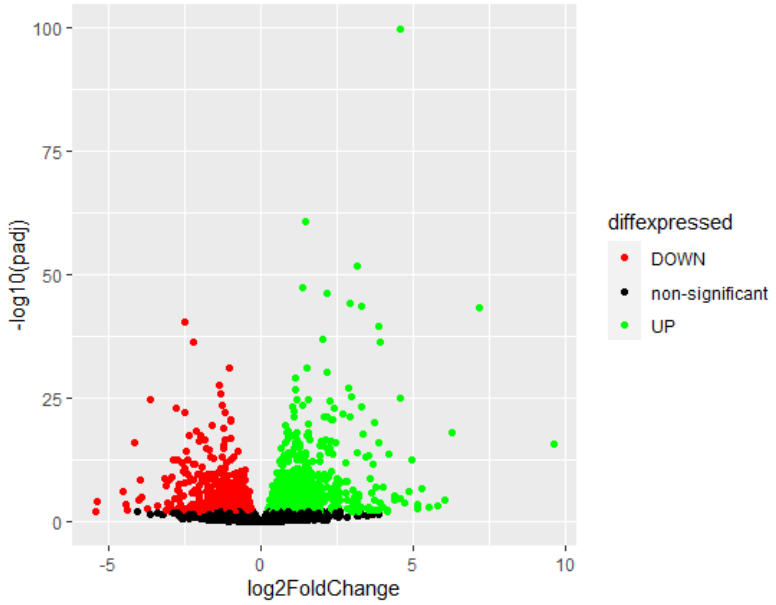


Fig 2: Volcano plot representing upregulated genes as green, downregulated genes as red and non-significant genes as black dots.

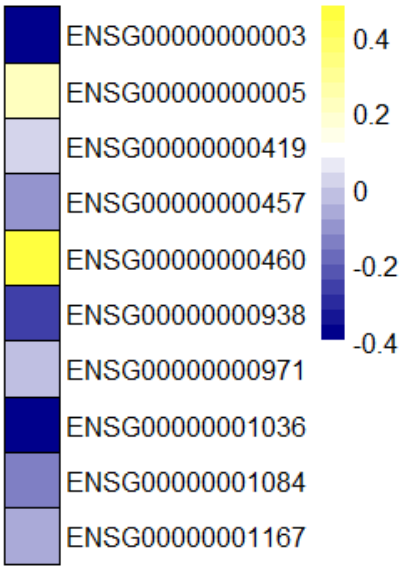


Fig 3: Heatmap representing the expression levels of first 10 genes in terms of log2FoldChange values in a scale of -0.4 to 0.4 where, blue colour represents downregulated genes, yellow represents upregulated genes and expression levels of remaining genes are represented by gradation of colour between blue and yellow.

Refrences:

Himes, B. E., Jiang, X., Wagner, P., Hu, R., Wang, Q., Klanderman, B., & Lu, Q. (2014). RNA-Seq transcriptome profiling identifies CRISPLD2 as a glucocorticoid responsive gene that modulates cytokine function in airway smooth muscle cells. *PloS one*, *9*(6), e99625.

Love, M. I., Huber, W., & Anders, S. (2014). Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2. *Genome biology*, *15*(12), 1-21.

Robinson, M. D., McCarthy, D. J., & Smyth, G. K. (2010). edgeR: a Bioconductor package for differential expression analysis of digital gene expression data. *bioinformatics*, *26*(1), 139-140.

Smyth, G. K. (2005). Limma: linear models for microarray data. *Bioinformatics and computational biology solutions using R and Bioconductor*, 397-420.

OVERVIEW OF AI / ML TECHNIQUES

Sharanbasappa and Vinaykumar L N

ICAR-Indian Agricultural Statistics Research Institute, New Delhi-110012

1. Introduction to Artificial Intelligence and Machine Learning

1.1 What is Artificial Intelligence?

Artificial Intelligence (AI) is the branch of computer science concerned with building systems that can perform tasks which typically require human intelligence, such as perception, reasoning, decision-making, and language understanding. Machine Learning (ML) is a subset of AI in which systems learn patterns from data rather than following explicitly programmed rules (Bishop, 2006). Deep Learning (DL) is, in turn, a subset of ML based on multi-layered artificial neural networks capable of automatically learning hierarchical feature representations (Goodfellow et al., 2016).

1.2 Brief Historical Timeline

Period	Milestone
1950s	Turing Test proposed; term "Artificial Intelligence" coined (1956, Dartmouth)
1960s-70s	Early expert systems and symbolic reasoning; first "AI winter" due to limited compute
1980s-90s	Rise of backpropagation, decision trees, and statistical learning theory (SVMs)
2000s	Growth of large datasets and ensemble methods (Random Forests, Boosting)
2012-present	Deep learning revolution driven by GPUs and large labeled datasets (ImageNet, AlexNet)
2017-present	Transformer architecture; large language models and generative AI

1.3 AI vs ML vs DL

Concept	Description
Artificial Intelligence	Broad field: any technique that enables machines to mimic intelligent behavior
Machine Learning	Algorithms that improve performance on a task through experience (data)

Concept	Description
Deep Learning	ML using deep neural networks with many layers, effective on unstructured data

1.4 Types of Machine Learning

- Supervised Learning: the model learns from labeled data (input-output pairs) to predict outputs for new inputs. Includes regression and classification.
- Unsupervised Learning: the model learns patterns and structure from unlabeled data. Includes clustering, dimensionality reduction, and association rule mining.
- Semi-Supervised Learning: uses a small amount of labeled data together with a large amount of unlabeled data during training.
- Self-Supervised Learning: generates its own labels from the structure of the input data itself (e.g., predicting masked words), widely used to pretrain large models.
- Reinforcement Learning: an agent learns to take actions in an environment to maximize cumulative reward through trial and error.

1.5 Typical Applications

Domain	Example Application
Healthcare	Disease diagnosis, medical image analysis, genomics
Finance	Fraud detection, credit scoring, algorithmic trading
Retail	Recommendation systems, demand forecasting
Computer Vision	Object detection, facial recognition, autonomous driving
NLP	Machine translation, chatbots, sentiment analysis
Bioinformatics	Gene expression analysis, protein structure prediction

1.6 Structured vs Unstructured Data

Structured data is organized in a fixed tabular format with well-defined columns (e.g., spreadsheets, relational database tables) and is typically handled well by classical ML algorithms such as tree ensembles. Unstructured data — images, audio, free text, and video — lacks this fixed structure and generally requires deep learning to automatically extract useful feature representations, since manual feature engineering is impractical at scale.

2. The Machine Learning Workflow

2.1 General Pipeline

1. Problem Definition: identify the business/scientific question and the type of learning task (regression, classification, clustering, etc.).
2. Data Collection: gather raw data from databases, sensors, APIs, or experiments.
3. Data Preprocessing: clean, transform, and structure the data for modeling.
4. Feature Engineering: select or construct informative variables (features).
5. Model Selection and Training: choose an algorithm and fit it to the training data.
6. Model Evaluation: assess performance on unseen (test/validation) data.
7. Hyperparameter Tuning: optimize model configuration to improve performance.
8. Deployment and Monitoring: integrate the model into production and track performance over time.

2.2 Train-Test Split and Validation

Data is typically split into a training set (used to fit the model) and a test set (used to evaluate generalization to unseen data). A validation set, or k-fold cross-validation, is often used additionally for hyperparameter tuning without touching the final test set. For imbalanced classification problems, stratified splitting preserves the original class proportions in each subset.

```
from sklearn.model_selection import train_test_split


X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=42
)
```

2.3 Reproducibility

Fixing random seeds (random_state), logging library versions, and version-controlling both code and data are essential practices for reproducible ML experiments, particularly in scientific and clinical applications where results must be independently verifiable.

2.4 Exploratory Data Analysis (EDA)

Before modeling, EDA summarizes the dataset's structure, distributions, and relationships using descriptive statistics and visualizations, helping to spot data quality issues, outliers, and candidate features early in the workflow.

```
import matplotlib.pyplot as plt
import seaborn as sns


```

```
print(df.describe())
print(df.isnull().sum())

sns.histplot(df["age"], kde=True)
sns.heatmap(df.corr(), annot=True, cmap="coolwarm")
plt.show()
```

3. Data Preprocessing and Feature Engineering

3.1 Handling Missing Values

Real-world datasets frequently contain missing values, which must be handled before model training — either by removing incomplete rows/columns or by imputing values (mean, median, mode, or model-based imputation such as k-nearest-neighbor imputation).

```
import pandas as pd
from sklearn.impute import SimpleImputer

df = pd.read_csv("data.csv")
imputer = SimpleImputer(strategy="mean")
df[["age", "income"]] = imputer.fit_transform(df[["age", "income"]])
```

3.2 Encoding Categorical Variables

Most ML algorithms require numeric input. Categorical variables are converted using label encoding (assigning an integer to each category, suitable for ordinal data) or one-hot encoding (creating a binary column per category, suitable for nominal data).

```
df_encoded = pd.get_dummies(df, columns=["gender", "city"])
```

3.3 Feature Scaling

Algorithms that rely on distance calculations (KNN, SVM, K-Means) or gradient descent (linear/logistic regression, neural networks) are sensitive to feature scale. Standardization rescales features to zero mean and unit variance; normalization (min-max scaling) rescales features to a fixed range, typically [0, 1]. Tree-based methods are generally scale-invariant and do not require this step.

```
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

3.4 Outlier Detection

Outliers can be identified using statistical rules (e.g., values beyond 1.5 times the interquartile range, or beyond 3 standard deviations from the mean) or model-based methods such as the Isolation Forest and Local Outlier Factor algorithms, which flag points that are easy to isolate or that lie in sparse neighborhoods.

```
from sklearn.ensemble import IsolationForest

iso = IsolationForest(contamination=0.05, random_state=42)
outlier_flags = iso.fit_predict(X)   # -1 indicates an outlier
```

3.5 Handling Imbalanced Data

When one class is much rarer than another (e.g., fraud detection), models tend to be biased toward the majority class. Common remedies include class-weighting the loss function, undersampling the majority class, oversampling the minority class, or generating synthetic minority examples using SMOTE (Synthetic Minority Over-sampling Technique).

```
from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_train, y_train)
```

3.6 Feature Selection

Feature selection reduces dimensionality and can improve generalization by retaining only the most informative variables. Common approaches include filter methods (e.g., correlation or chi-squared tests), wrapper methods (e.g., recursive feature elimination), and embedded methods (e.g., Lasso regression or tree-based feature importances).

```
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression

selector = RFE(LogisticRegression(max_iter=1000), n_features_to_select=10)
```

```
selector.fit(X_train, y_train)
selected_columns = X_train.columns[selector.support_]
```

4. Practical Case Study: End-to-End Classification Pipeline

This section walks through a complete, minimal ML pipeline on a tabular classification problem, tying together preprocessing, model training, and evaluation from the preceding sections into a single practical workflow (Géron, 2022).

4.1 Load and Inspect Data

```
import pandas as pd

df = pd.read_csv("patients.csv")
print(df.shape)
print(df.head())
print(df["outcome"].value_counts(normalize=True))
```

4.2 Preprocess Features

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer

X = df.drop(columns=["outcome"])
y = df["outcome"]

X = pd.DataFrame(SimpleImputer(strategy="median").fit_transform(X), columns=X.columns)

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=42
)

scaler = StandardScaler()
X_train_s = scaler.fit_transform(X_train)
X_test_s = scaler.transform(X_test)
```

4.3 Train and Compare Multiple Models

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, f1_score

models = {
    "Logistic Regression": LogisticRegression(max_iter=1000),
    "Random Forest": RandomForestClassifier(n_estimators=200, random_state=42),
    "SVM (RBF)": SVC(kernel="rbf", probability=True),
}

results = []
for name, model in models.items():
    model.fit(X_train_s, y_train)
    preds = model.predict(X_test_s)
    results.append({
        "model": name,
        "accuracy": accuracy_score(y_test, preds),
        "f1": f1_score(y_test, preds),
    })

print(pd.DataFrame(results).sort_values("f1", ascending=False))
```

4.4 Select and Persist the Best Model

```
best_model = models["Random Forest"]

import joblib
joblib.dump({"model": best_model, "scaler": scaler}, "pipeline.pkl")
```

> *Note: A production pipeline should wrap preprocessing and modeling steps together (e.g., using sklearn.pipeline.Pipeline) so that the exact same transformations are applied consistently at inference time.*

4.5 Combining Steps with a Pipeline

Wrapping preprocessing and the estimator inside a single sklearn Pipeline object avoids data leakage (e.g., accidentally fitting a scaler on the test set) and simplifies cross-validation, hyperparameter tuning, and deployment, since the entire pipeline can be fit, evaluated, and saved as one object.

```
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV

pipeline = Pipeline([
    ("scaler", StandardScaler()),
    ("clf", RandomForestClassifier(random_state=42)),
])

param_grid = {"clf__n_estimators": [100, 200], "clf__max_depth": [4, 8, None]}
grid = GridSearchCV(pipeline, param_grid, cv=5, scoring="f1")
grid.fit(X_train, y_train)

print("Best params:", grid.best_params_)
print("Best CV F1:", grid.best_score_)
```

5. Supervised Learning: Regression

5.1 Linear Regression

Linear regression models the relationship between a dependent variable and one or more independent variables by fitting a straight line (or hyperplane) that minimizes the sum of squared residuals (Ordinary Least Squares) (James et al., 2021).

```
from sklearn.linear_model import LinearRegression

model = LinearRegression()
model.fit(X_train, y_train)
predictions = model.predict(X_test)
```

5.2 Polynomial and Regularized Regression

Polynomial regression extends linear regression by adding polynomial terms of the features, allowing the model to capture non-linear relationships. Ridge regression (L2 penalty) shrinks coefficients toward zero

without eliminating them, helping with multicollinearity; Lasso regression (L1 penalty) can shrink some coefficients exactly to zero, performing automatic feature selection; Elastic Net combines both penalties.

```
from sklearn.linear_model import Ridge, Lasso, ElasticNet


ridge = Ridge(alpha=1.0).fit(X_train, y_train)
lasso = Lasso(alpha=0.1).fit(X_train, y_train)
elastic = ElasticNet(alpha=0.1, l1_ratio=0.5).fit(X_train, y_train)
```

5.3 Assumptions of Linear Regression

- Linearity: the relationship between predictors and the target is approximately linear.
- Independence of errors: residuals are not correlated with one another.
- Homoscedasticity: residual variance is constant across all levels of the predictors.
- Low multicollinearity: predictors are not highly correlated with each other.

5.4 Regression Evaluation Metrics

Metric	Description
MAE	Mean Absolute Error — average magnitude of errors
MSE	Mean Squared Error — penalizes larger errors more
RMSE	Root Mean Squared Error — same units as target variable
R-squared	Proportion of variance in the target explained by the model
Adjusted R-squared	R-squared adjusted for the number of predictors used

```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score


mse = mean_squared_error(y_test, predictions)
mae = mean_absolute_error(y_test, predictions)
r2 = r2_score(y_test, predictions)
```

5.5 Checking Residuals

Plotting residuals (actual minus predicted values) against predicted values or against each predictor helps verify the linearity and homoscedasticity assumptions; a random scatter around zero indicates a well-specified model, while a visible pattern (e.g., a funnel shape) suggests the assumptions are violated.

```
import matplotlib.pyplot as plt
```

```
residuals = y_test - predictions
plt.scatter(predictions, residuals)
plt.axhline(0, color="red", linestyle="--")
plt.xlabel("Predicted values")
plt.ylabel("Residuals")
plt.title("Residual Plot")
```

6. Supervised Learning: Classification

6.1 Logistic Regression

Despite its name, logistic regression is a classification algorithm. It applies the sigmoid function to a linear combination of features to output a probability between 0 and 1, which is then thresholded to assign a class label.

```
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

6.2 K-Nearest Neighbors (KNN)

KNN classifies a new data point based on the majority class among its k closest neighbors in the feature space, using a distance metric such as Euclidean distance. It is a non-parametric, instance-based ("lazy") learning algorithm requiring no explicit training phase, though prediction can be slow on large datasets.

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
```

6.3 Decision Trees

A decision tree recursively splits the feature space based on conditions that maximize information gain (or minimize Gini impurity), forming a tree of if-else decision rules that is easy to visualize and interpret, though prone to overfitting if grown too deep.

```
from sklearn.tree import DecisionTreeClassifier

tree = DecisionTreeClassifier(max_depth=5, random_state=42)
tree.fit(X_train, y_train)
```

6.4 Support Vector Machines (SVM)

SVM finds the hyperplane that maximizes the margin between classes. For non-linearly separable data, the kernel trick (e.g., radial basis function kernel) implicitly maps features into a higher-dimensional space where a linear separator can be found.

```
from sklearn.svm import SVC

svm = SVC(kernel="rbf", C=1.0, gamma="scale")
svm.fit(X_train, y_train)
```

6.5 Naive Bayes

Naive Bayes classifiers apply Bayes' theorem with the (naive) assumption that features are conditionally independent given the class label. Despite this simplifying assumption, Naive Bayes performs well in practice, particularly for text classification tasks such as spam detection, and trains very quickly even on large datasets.

6.6 Multi-Class Classification Strategies

Many algorithms (e.g., logistic regression, SVM) are inherently binary. Multi-class problems are handled either natively (if the algorithm supports it directly, as with decision trees and softmax-based neural networks) or via decomposition strategies: One-vs-Rest trains one binary classifier per class against all others, while One-vs-One trains a separate classifier for every pair of classes.

6.7 Choosing a Classification Algorithm

Algorithm	Strengths	Limitations
Logistic Regression	Simple, interpretable, fast	Assumes linear decision boundary
KNN	No training phase, simple concept	Slow prediction, sensitive to scale
Decision Tree	Interpretable, handles non-linearity	Prone to overfitting
Random Forest	Robust, handles non-linearity well	Less interpretable, larger memory use

Algorithm	Strengths	Limitations
SVM	Effective in high dimensions	Slower on large datasets, kernel tuning needed
Naive Bayes	Fast, works well on text data	Independence assumption often unrealistic

7. Ensemble Learning Methods

7.1 Bagging and Random Forests

Bagging (Bootstrap Aggregating) trains multiple instances of the same base model on different bootstrapped samples of the training data and averages (or votes on) their predictions, reducing variance. A Random Forest is a bagging ensemble of decision trees that additionally randomizes the subset of features considered at each split, further decorrelating the trees.

```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(n_estimators=200, max_depth=6, random_state=42)
rf.fit(X_train, y_train)
```

7.2 Boosting: AdaBoost and Gradient Boosting

Boosting trains base learners (typically shallow decision trees) sequentially, with each new learner focusing more on the examples that previous learners misclassified. AdaBoost re-weights misclassified samples at each round; Gradient Boosting instead fits each new learner to the residual errors (negative gradient of the loss) of the ensemble so far.

```
from sklearn.ensemble import GradientBoostingClassifier

gbc = GradientBoostingClassifier(n_estimators=150, learning_rate=0.1, max_depth=3)
gbc.fit(X_train, y_train)
```

7.3 XGBoost and LightGBM

XGBoost and LightGBM are optimized, regularized implementations of gradient boosting that add features such as tree pruning, handling of missing values, and parallelized/histogram-based split-finding, making them extremely popular for structured/tabular data competitions and industrial applications.

```
import xgboost as xgb

xgb_clf = xgb.XGBClassifier(n_estimators=300, max_depth=4, learning_rate=0.05)
xgb_clf.fit(X_train, y_train)
```

7.4 Voting and Stacking

A voting ensemble combines predictions from several different model types (e.g., logistic regression, SVM, random forest) by majority vote or averaged probability. Stacking goes further by training a meta-model that learns how to best combine the base models' predictions.

7.5 Key Ensemble Hyperparameters

Hyperparameter	Effect
n_estimators	Number of trees/base learners; more can improve performance up to a point
max_depth	Maximum depth of each tree; controls individual learner complexity
learning_rate	Boosting step size; smaller values need more estimators but generalize better
subsample	Fraction of samples used per tree; adds randomness, can reduce overfitting
min_samples_leaf	Minimum samples required at a leaf node; higher values regularize the tree

8. Classification Evaluation in Depth

8.1 Confusion Matrix and Derived Metrics

A confusion matrix summarizes correct and incorrect predictions across classes (true positives, false positives, true negatives, false negatives), from which further metrics are derived.

Metric	Formula / Meaning
Accuracy	(TP + TN) / Total — overall correctness
Precision	TP / (TP + FP) — correctness among predicted positives
Recall (Sensitivity)	TP / (TP + FN) — coverage of actual positives
F1-score	Harmonic mean of precision and recall
Specificity	TN / (TN + FP) — coverage of actual negatives

Metric	Formula / Meaning
ROC-AUC	Area under the ROC curve — ability to distinguish classes across thresholds

```
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score

print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
auc = roc_auc_score(y_test, clf.predict_proba(X_test)[:, 1])
```

8.2 ROC and Precision-Recall Curves

The Receiver Operating Characteristic (ROC) curve plots the true positive rate against the false positive rate at varying classification thresholds, summarizing performance independent of a single threshold choice. The Precision-Recall curve is often preferred over ROC for highly imbalanced datasets, since it focuses on performance with respect to the minority (positive) class.

```
from sklearn.metrics import roc_curve
import matplotlib.pyplot as plt

fpr, tpr, thresholds = roc_curve(y_test, clf.predict_proba(X_test)[:, 1])
plt.plot(fpr, tpr)
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
```

8.3 Learning Curves

A learning curve plots training and validation performance as a function of training set size (or training epochs), helping diagnose whether a model suffers from high bias (both curves plateau at a low score) or high variance (a large, persistent gap between training and validation performance).

9. Unsupervised Learning

9.1 K-Means Clustering

K-Means partitions data into k clusters by iteratively assigning points to the nearest cluster centroid and updating centroids as the mean of assigned points, until convergence. The number of clusters k is typically

chosen using the elbow method (plotting within-cluster variance against k) or the silhouette score (Pedregosa et al., 2011).

```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
labels = kmeans.fit_predict(X_scaled)
```

9.2 Hierarchical Clustering and DBSCAN

Hierarchical (agglomerative) clustering builds a tree of nested clusters (dendrogram) by successively merging the closest pairs of clusters, and does not require the number of clusters to be specified in advance. DBSCAN (Density-Based Spatial Clustering of Applications with Noise) groups points that are closely packed together, marking points in low-density regions as outliers/noise, and can discover clusters of arbitrary shape.

```
from sklearn.cluster import DBSCAN

dbscan = DBSCAN(eps=0.5, min_samples=5)
labels = dbscan.fit_predict(X_scaled)
```

```
from scipy.cluster.hierarchy import dendrogram, linkage
import matplotlib.pyplot as plt

Z = linkage(X_scaled, method="ward")
dendrogram(Z)
plt.xlabel("Sample index")
plt.ylabel("Distance")
plt.title("Hierarchical Clustering Dendrogram")
plt.show()
```

9.3 Dimensionality Reduction: PCA, t-SNE and UMAP

Principal Component Analysis (PCA) is a linear technique that projects high-dimensional data onto a smaller number of orthogonal components (principal components) that capture the maximum variance in the data, useful for visualization and noise reduction. t-SNE (t-distributed Stochastic Neighbor Embedding) and UMAP (Uniform Manifold Approximation and Projection) are non-linear techniques

primarily used for visualizing high-dimensional data in two or three dimensions by preserving local neighborhood structure.

```
from sklearn.decomposition import PCA


pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)
print(pca.explained_variance_ratio_)
```

9.4 Anomaly Detection

Anomaly (outlier) detection identifies rare items or observations that deviate significantly from the majority of the data, useful in fraud detection, network intrusion detection, and manufacturing defect detection. Common approaches include Isolation Forest, One-Class SVM, and autoencoder reconstruction error.

9.5 Association Rule Mining

Association rule mining discovers interesting relationships between variables in large transactional datasets, commonly summarized as "if X then Y" rules with associated support, confidence, and lift values. The Apriori and FP-Growth algorithms are classical approaches, widely used in market basket analysis (e.g., "customers who buy bread often also buy butter").

10. Model Selection, Overfitting and Hyperparameter Tuning

10.1 Bias-Variance Tradeoff

Bias refers to error introduced by overly simplistic assumptions in the model (underfitting), while variance refers to error introduced by excessive sensitivity to fluctuations in the training data (overfitting). The goal of model selection is to find a balance that minimizes total generalization error on unseen data.

10.2 Overfitting, Underfitting and Regularization

An overfit model performs well on training data but poorly on new data because it has learned noise rather than the underlying pattern; an underfit model performs poorly on both. Common remedies for overfitting include acquiring more training data, simplifying the model, applying regularization (L1/L2 penalties), pruning decision trees, and using dropout in neural networks.

10.3 Cross-Validation

k-fold cross-validation splits the training data into k subsets, training on k-1 folds and validating on the remaining fold, repeated k times, giving a more robust estimate of model performance than a single train-test split. Stratified k-fold preserves class balance in each fold for classification tasks.

```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(model, X, y, cv=5, scoring="accuracy")
print(scores.mean(), scores.std())
```

10.4 Hyperparameter Tuning

Grid search exhaustively evaluates every combination of a specified hyperparameter grid, while randomized search samples a fixed number of random combinations, often finding good configurations more efficiently for high-dimensional search spaces. Bayesian optimization further improves efficiency by using past evaluation results to choose the next combination to try.

```
from sklearn.model_selection import GridSearchCV

param_grid = {"n_estimators": [100, 200], "max_depth": [4, 6, 8]}
grid = GridSearchCV(RandomForestClassifier(), param_grid, cv=5, scoring="accuracy")
grid.fit(X_train, y_train)
print(grid.best_params_)
```

11. Model Interpretability

As models grow more complex, understanding why a model makes a given prediction becomes important for trust, debugging, and regulatory compliance, particularly in healthcare and finance.

- Feature Importance: tree-based models naturally provide a ranking of how much each feature contributed to reducing impurity across the ensemble.
- Permutation Importance: measures the drop in model performance when a feature's values are randomly shuffled, applicable to any model type.
- SHAP (SHapley Additive exPlanations): assigns each feature a contribution value for an individual prediction, based on cooperative game theory, providing both local and global interpretability (Lundberg & Lee, 2017).

- LIME (Local Interpretable Model-agnostic Explanations): approximates a complex model locally with an interpretable surrogate model (e.g., linear model) around a specific prediction (Ribeiro et al., 2016).

```
import shap

explainer = shap.TreeExplainer(rf)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
```

11.1 LIME Example

```
from lime.lime_tabular import LimeTabularExplainer

explainer = LimeTabularExplainer(
    X_train.values, feature_names=X_train.columns, class_names=["negative", "positive"],
mode="classification"
)

exp = explainer.explain_instance(X_test.iloc[0].values, clf.predict_proba)
exp.as_list()   # ranked (feature, contribution) pairs for this single prediction
```

12. Neural Networks and Deep Learning

12.1 The Perceptron and Multi-Layer Perceptron

A perceptron is the simplest neural network unit: it computes a weighted sum of inputs, adds a bias term, and passes the result through an activation function to produce an output. A Multi-Layer Perceptron (MLP) stacks multiple layers of such units (an input layer, one or more hidden layers, and an output layer), enabling it to learn non-linear decision boundaries (universal approximation) (Goodfellow et al., 2016).

12.2 Activation Functions

Function	Typical Use
Sigmoid	Binary classification output layer; squashes values to (0,1)
Tanh	Hidden layers; squashes values to (-1,1), zero-centered

Function	Typical Use
ReLU	Most common choice for hidden layers; fast and reduces vanishing gradient
Leaky ReLU	Variant of ReLU that avoids "dead neurons" by allowing a small negative slope
Softmax	Multi-class classification output layer; produces a probability distribution

12.3 Loss Functions and Optimizers

The loss function quantifies the discrepancy between predicted and actual outputs (e.g., mean squared error for regression, cross-entropy for classification). Optimizers such as Stochastic Gradient Descent (SGD), Adam, and RMSprop update model weights to minimize this loss, differing in how they adapt the learning rate and incorporate momentum.

12.4 Backpropagation

Neural networks are trained by minimizing a loss function using gradient descent. Backpropagation computes the gradient of the loss with respect to every weight in the network by applying the chain rule layer by layer, working backward from the output layer, and these gradients are used to update the weights iteratively.

12.5 Regularization in Deep Learning

- Dropout: randomly deactivates a fraction of neurons during each training step, preventing co-adaptation and reducing overfitting.
- Batch Normalization: normalizes layer inputs across a mini-batch, stabilizing and accelerating training.
- Early Stopping: halts training once validation performance stops improving, preventing the model from overfitting the training data.

12.6 Convolutional Neural Networks (CNNs)

CNNs use learnable filters (convolutions) that slide across the input to detect local spatial patterns such as edges, textures, and shapes; pooling layers subsequently reduce spatial dimensions while retaining the most salient features. This architecture makes CNNs highly effective and parameter-efficient for image data.

12.7 Recurrent Networks: RNN, LSTM and GRU

Recurrent Neural Networks (RNNs) maintain an internal hidden state that is updated at each time step, making them suited to sequential data such as text, speech, and time series. Standard RNNs suffer from vanishing gradients over long sequences; Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) architectures introduce gating mechanisms that allow the network to retain or forget information over longer sequences.

12.8 A Simple Neural Network in Keras

```
import tensorflow as tf
from tensorflow import keras

model = keras.Sequential([
    keras.layers.Dense(64, activation="relu", input_shape=(X_train.shape[1],)),
    keras.layers.Dropout(0.3),
    keras.layers.Dense(32, activation="relu"),
    keras.layers.Dense(1, activation="sigmoid")
])

model.compile(optimizer="adam", loss="binary_crossentropy", metrics=["accuracy"])
history = model.fit(X_train, y_train, epochs=20, batch_size=32, validation_split=0.2)
```

12.9 A Simple CNN for Image Classification

```
cnn = keras.Sequential([
    keras.layers.Conv2D(32, (3, 3), activation="relu", input_shape=(28, 28, 1)),
    keras.layers.MaxPooling2D((2, 2)),
    keras.layers.Conv2D(64, (3, 3), activation="relu"),
    keras.layers.MaxPooling2D((2, 2)),
    keras.layers.Flatten(),
    keras.layers.Dense(64, activation="relu"),
    keras.layers.Dense(10, activation="softmax")
])

cnn.compile(optimizer="adam", loss="sparse_categorical_crossentropy", metrics=["accuracy"])
cnn.fit(X_train_images, y_train, epochs=10, validation_split=0.1)
```

12.10 A Simple LSTM for Sequence Data

```
lstm_model = keras.Sequential([
    keras.layers.Embedding(input_dim=10000, output_dim=64),
    keras.layers.LSTM(64),
    keras.layers.Dense(1, activation="sigmoid")
])

lstm_model.compile(optimizer="adam", loss="binary_crossentropy", metrics=["accuracy"])
lstm_model.fit(X_train_seq, y_train, epochs=5, batch_size=64, validation_split=0.2)
```

13. Attention and Transformer Models

The Transformer architecture, introduced in 2017, replaced recurrence with a self-attention mechanism that allows every position in a sequence to directly attend to every other position, greatly improving parallelization and the modeling of long-range dependencies (Vaswani et al., 2017). Transformers underpin most modern large language models (e.g., the GPT and BERT families) (Wolf et al., 2020) and are increasingly applied beyond text, including to genomic and protein sequences.

13.1 Self-Attention (Conceptual)

Self-attention computes, for each token, a weighted combination of all other tokens' representations, where the weights (attention scores) reflect how relevant each other token is to the current one. These weights are learned from Query, Key, and Value projections of the input embeddings.

13.2 Transfer Learning and Fine-Tuning

Rather than training a large model from scratch, transfer learning starts from a model pretrained on a large, general dataset and fine-tunes it on a smaller, task-specific dataset, substantially reducing the data and compute required and often improving final performance.

14. Generative Models

Generative models learn the underlying distribution of the training data so that they can produce new, synthetic samples resembling that data, in contrast to discriminative models, which only learn to distinguish between classes.

14.1 Autoencoders

An autoencoder is a neural network trained to reconstruct its own input, consisting of an encoder that compresses the input into a lower-dimensional latent representation and a decoder that reconstructs the input from that representation. Autoencoders are used for dimensionality reduction, denoising, and anomaly detection (samples with high reconstruction error are flagged as anomalous).

```
encoder = keras.Sequential([keras.layers.Dense(32, activation="relu")])
decoder = keras.Sequential([keras.layers.Dense(784, activation="sigmoid")])

autoencoder = keras.Sequential([encoder, decoder])
autoencoder.compile(optimizer="adam", loss="mse")
autoencoder.fit(X_train, X_train, epochs=20, batch_size=256)
```

14.2 Generative Adversarial Networks (GANs)

A GAN consists of two competing neural networks trained together: a generator that produces synthetic samples from random noise, and a discriminator that tries to distinguish real samples from generated ones. Through this adversarial training process, the generator progressively learns to produce increasingly realistic samples.

14.3 Variational Autoencoders and Diffusion Models

Variational Autoencoders (VAEs) extend autoencoders by learning a probabilistic latent space, enabling smooth interpolation and sampling of new data points. Diffusion models, which underlie many modern image-generation systems, learn to reverse a gradual noising process, generating data by iteratively denoising a random starting point.

15. Reinforcement Learning

15.1 Core Concepts

Reinforcement Learning (RL) formalizes sequential decision-making as a Markov Decision Process (MDP), defined by a set of states, actions, transition probabilities, and rewards. An agent interacts with an environment, choosing actions according to a policy, and learns to maximize the expected cumulative (discounted) reward over time (Mnih et al., 2015).

15.2 Q-Learning

Q-Learning is a model-free, value-based RL algorithm that learns an action-value function Q(s, a), estimating the expected future reward of taking action a in state s and following the optimal policy

thereafter. The Q-values are updated iteratively using the observed reward and the estimated value of the next state.

```
import numpy as np

Q = np.zeros((n_states, n_actions))
alpha, gamma = 0.1, 0.9

for episode in range(1000):
    state = env.reset()
    done = False
    while not done:
        action = np.argmax(Q[state]) if np.random.rand() > 0.1 else env.action_space.sample()
        next_state, reward, done, _ = env.step(action)
        Q[state, action] += alpha * (reward + gamma * np.max(Q[next_state]) - Q[state, action])
        state = next_state
```

15.3 Deep Reinforcement Learning

When the state space is too large for a tabular Q-value representation, Deep Q-Networks (DQN) approximate the Q-function using a neural network. Policy-gradient methods (e.g., REINFORCE, PPO) instead directly learn a parameterized policy that maps states to action probabilities, which is often more effective for continuous action spaces.

16. Popular ML/DL Frameworks

Library	Purpose
scikit-learn	Classical ML algorithms, preprocessing, model evaluation
TensorFlow / Keras	Building and training deep neural networks, production deployment
PyTorch	Deep learning with dynamic computation graphs, popular in research
Pandas / NumPy	Data manipulation and numerical computation
Matplotlib / Seaborn	Data visualization
XGBoost / LightGBM	Gradient-boosted tree ensembles for structured data

Library	Purpose
Hugging Face Transformers	Pretrained transformer models for NLP and beyond

17. Introduction to Natural Language Processing

NLP applies AI/ML techniques to understand and generate human language. Text is first converted into numerical form through tokenization and vectorization (e.g., Bag-of-Words, TF-IDF, or dense word embeddings such as Word2Vec and GloVe) before being fed into classical ML models or neural networks such as RNNs and Transformers.

17.1 Text Preprocessing

- Tokenization: splitting text into words, subwords, or sentences.
- Stopword Removal: discarding common words ("the", "is", "and") that carry little semantic weight.
- Stemming/Lemmatization: reducing words to their root or base form (e.g., "running" to "run").
- Vectorization: converting tokens into numeric vectors (TF-IDF, word embeddings, or transformer embeddings).

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB

vectorizer = TfidfVectorizer(stop_words="english")
X_text = vectorizer.fit_transform(documents)

clf = MultinomialNB()
clf.fit(X_text, labels)
```

17.2 Word Embeddings

Unlike sparse Bag-of-Words/TF-IDF representations, word embeddings map each word to a dense, low-dimensional vector such that semantically similar words lie close together in the vector space. Word2Vec and GloVe learn static embeddings from large text corpora, while contextual embeddings from transformer models (e.g., BERT) produce a different vector for the same word depending on its surrounding context.

```
from gensim.models import Word2Vec

sentences = [doc.split() for doc in documents]
```

```
w2v = Word2Vec(sentences, vector_size=100, window=5, min_count=2, workers=4)
print(w2v.wv.most_similar("cancer"))
```

17.3 End-to-End Sentiment Classification Example

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

X_train_txt, X_test_txt, y_train, y_test = train_test_split(
    documents, labels, test_size=0.2, random_state=42
)

vectorizer = TfidfVectorizer(max_features=5000, ngram_range=(1, 2))
X_train_vec = vectorizer.fit_transform(X_train_txt)
X_test_vec = vectorizer.transform(X_test_txt)

clf = LogisticRegression(max_iter=1000)
clf.fit(X_train_vec, y_train)

print("Test accuracy:", accuracy_score(y_test, clf.predict(X_test_vec)))
```

18. Time Series Forecasting

Time series data consists of observations recorded sequentially over time, often exhibiting trend, seasonality, and autocorrelation. Classical statistical models such as ARIMA (AutoRegressive Integrated Moving Average) and modern tools such as Facebook Prophet model these components explicitly, while ML approaches (gradient boosting, LSTMs) treat forecasting as a supervised learning problem using lagged values as features.

```
from statsmodels.tsa.arima.model import ARIMA

model = ARIMA(series, order=(2, 1, 2))
fit = model.fit()
forecast = fit.forecast(steps=12)
```

18.1 Forecast Evaluation Metrics

Forecast accuracy is commonly measured using MAE, RMSE, and MAPE (Mean Absolute Percentage Error), the latter expressing error as a percentage of the actual value, which is useful for comparing forecast quality across series of different scales.

18.2 Forecasting with Prophet

```
from prophet import Prophet

df_prophet = df.rename(columns={"date": "ds", "sales": "y"})
model = Prophet(yearly_seasonality=True, weekly_seasonality=True)
model.fit(df_prophet)

future = model.make_future_dataframe(periods=90)
forecast = model.predict(future)
model.plot(forecast)
```

19. Model Deployment and MLOps

Once validated, models are typically serialized (e.g., using pickle or joblib) and deployed behind an API (e.g., Flask or FastAPI) or embedded into larger applications. MLOps practices — versioning of data, code and models, automated retraining pipelines, and continuous monitoring for performance degradation or data drift — help maintain reliable models in production over time.

```
import joblib

joblib.dump(model, "model.pkl")
loaded_model = joblib.load("model.pkl")
loaded_model.predict(X_new)
```

19.1 Serving via a REST API

```
from fastapi import FastAPI
import joblib

app = FastAPI()
```

```
model = joblib.load("model.pkl")


@app.post("/predict")
def predict(features: list):
    prediction = model.predict([features])
    return {"prediction": prediction.tolist()}
```

19.2 Monitoring and Data Drift

After deployment, the statistical distribution of incoming data may shift over time relative to the training distribution (data drift), or the relationship between features and target may change (concept drift), both of which can silently degrade model performance. Production systems typically monitor input feature distributions and prediction quality, triggering alerts or automated retraining when significant drift is detected.

19.3 Containerization

Packaging a model and its serving code inside a Docker container ensures that the exact runtime environment (library versions, system dependencies) is reproducible across development, testing, and production, and simplifies deployment to cloud platforms and orchestration systems such as Kubernetes.

```
FROM python:3.11-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install -r requirements.txt
COPY . .
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

20. Common Pitfalls in ML Projects

- Data Leakage: information from the test set (or from the future) inadvertently influences training, e.g., scaling features before splitting, or including a feature that is a proxy for the target.
- Target Leakage: a feature is available at training time but would not be available at prediction time in production (e.g., a field only recorded after the outcome is known).
- Look-Ahead Bias: in time series problems, using future information that would not have been available at the time of prediction, inflating apparent performance.

- Improper Cross-Validation: applying standard k-fold cross-validation to time series or grouped data (e.g., multiple samples per patient) without respecting temporal order or group structure, leading to overly optimistic estimates.
- Overfitting to the Test Set: repeatedly tuning a model based on test-set performance effectively turns the test set into a second validation set, inflating reported performance.
- Ignoring Class Imbalance: reporting only accuracy on an imbalanced dataset can mask a model that trivially predicts the majority class for every sample.

21. Ethical Considerations in AI/ML

- Bias and Fairness: models trained on biased historical data can perpetuate or amplify discrimination against certain groups.
- Explainability: complex models (especially deep networks) can behave as "black boxes", motivating interpretability techniques such as SHAP and LIME.
- Privacy: models trained on sensitive personal or medical data must comply with data protection regulations and anonymization practices.
- Accountability: clear responsibility must be established for decisions made or influenced by automated systems.
- Robustness and Security: models can be vulnerable to adversarial examples — small, deliberately crafted input perturbations that cause misclassification.

22. Applications of AI/ML in Transcriptomics

22.1 Background

Transcriptomics is the study of the complete set of RNA transcripts (the transcriptome) produced by the genome under specific circumstances, commonly measured through RNA sequencing (RNA-seq) or single-cell RNA sequencing (scRNA-seq). These experiments generate high-dimensional, noisy datasets — often tens of thousands of genes measured across many samples or cells — making them a natural fit for AI/ML techniques (Luecken & Theis, 2019).

22.2 Key Use Cases

- Dimensionality Reduction and Visualization: PCA, t-SNE, and UMAP are routinely used to project high-dimensional gene expression data (thousands of genes) into two or three dimensions to visualize sample relationships or distinct cell populations in scRNA-seq.

- Clustering for Cell-Type Identification: unsupervised clustering algorithms (K-Means, hierarchical clustering, Louvain/Leiden community detection) group cells with similar expression profiles into putative cell types or states in single-cell studies.
- Classification of Disease Subtypes: supervised classifiers (random forests, SVMs, logistic regression, neural networks) are trained on labeled expression data to distinguish disease subtypes, such as cancer classification from gene expression signatures.
- Differential Expression and Feature Selection: ML-based feature selection helps identify the subset of genes most predictive of an outcome, complementing traditional statistical differential expression tests (e.g., DESeq2, edgeR) (Love et al., 2014).
- Batch Effect Correction: techniques such as autoencoders and specialized algorithms (e.g., ComBat, Harmony, scVI) use ML to remove unwanted technical variation between experimental batches while preserving true biological signal.
- Imputation of Missing Data: scRNA-seq data suffers from high sparsity ("dropout"); deep learning-based autoencoders are used to impute missing/zero values and denoise expression matrices.
- Gene Regulatory Network Inference: ML models help infer regulatory relationships between transcription factors and target genes from expression data.
- Trajectory Inference: ML methods (e.g., pseudotime algorithms) order individual cells along inferred developmental or differentiation trajectories from single-cell expression profiles.

22.3 Illustrative Code: Clustering Gene Expression Data

The example below sketches a typical, simplified workflow for clustering samples based on gene expression using PCA followed by K-Means, analogous to steps used in exploratory transcriptomic analysis.

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans

# expr_df: rows = samples/cells, columns = genes (expression counts)
expr_df = pd.read_csv("expression_matrix.csv", index_col=0)

# Standardize expression values across genes
X_scaled = StandardScaler().fit_transform(expr_df)
```

```
# Reduce to top principal components capturing most variance
pca = PCA(n_components=20)
X_pca = pca.fit_transform(X_scaled)

# Cluster samples/cells into putative groups (e.g., cell types)
kmeans = KMeans(n_clusters=5, random_state=42, n_init=10)
expr_df["cluster"] = kmeans.fit_predict(X_pca)

print(expr_df["cluster"].value_counts())
```

> *Note: In practice, specialized bioinformatics toolkits such as Scanpy and Seurat implement optimized, biology-aware versions of these steps (library-size normalization, highly-variable-gene selection, Leiden clustering) rather than generic scikit-learn pipelines (Wolf et al., 2018).*

22.4 Illustrative Code: Classifying Samples by Disease Status

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

X = expr_df.drop(columns=["cluster", "label"])
y = expr_df["label"]   # e.g., "tumor" vs "normal"

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, stratify=y, random_state=42
)

clf = RandomForestClassifier(n_estimators=300, random_state=42)
clf.fit(X_train, y_train)

importances = pd.Series(clf.feature_importances_, index=X.columns)
top_genes = importances.sort_values(ascending=False).head(10)
print(top_genes)   # candidate marker genes
```

```
print(classification_report(y_test, clf.predict(X_test)))
```

Feature importances extracted from tree-based ensembles, as shown above, can highlight candidate marker genes associated with a phenotype, offering a data-driven complement to classical differential expression analysis.

22.5 Illustrative Code: 2D Visualization with UMAP

Non-linear embeddings such as UMAP are widely used to visualize single-cell transcriptomic data in two dimensions, often revealing distinct clusters corresponding to cell types or states more clearly than PCA alone.

```
import umap
import matplotlib.pyplot as plt

reducer = umap.UMAP(n_neighbors=15, min_dist=0.1, random_state=42)
embedding = reducer.fit_transform(X_pca)

plt.scatter(embedding[:, 0], embedding[:, 1], c=expr_df["cluster"], cmap="tab10", s=8)
plt.xlabel("UMAP 1")
plt.ylabel("UMAP 2")
plt.title("scRNA-seq Cell Clusters")
plt.show()
```

22.6 Practical Considerations

- Sample sizes in transcriptomic studies are often small relative to the number of genes (the "large p, small n" problem), increasing the risk of overfitting and demanding careful cross-validation and regularization.
- Normalization choices (e.g., counts per million, TPM, or variance-stabilizing transformations) can substantially influence downstream ML results and must be applied consistently between training and test data.
- Biological and technical batch effects, if uncorrected, can be learned by the model as spurious signal rather than true biological differences.

22.7 Commonly Used Tools

Tool	Typical Role
Scanpy	Python toolkit for scRNA-seq preprocessing, clustering, and visualization
Seurat	R toolkit for single-cell analysis, widely used alongside Scanpy
DESeq2 / edgeR	Statistical differential expression analysis for RNA-seq count data
scVI	Deep generative modeling framework for single-cell data (batch correction, imputation)
STAR / Salmon	Read alignment and transcript quantification from raw RNA-seq reads

23. Glossary of Key Terms

Term	Definition
Feature	An individual measurable input variable used by a model
Label / Target	The output variable a supervised model is trained to predict
Epoch	One complete pass of the training algorithm over the entire training dataset
Batch Size	Number of training samples processed before model weights are updated
Hyperparameter	A configuration value set before training (e.g., learning rate, tree depth)
Overfitting	A model that fits training data (including noise) too closely, harming generalization
Latent Space	A lower-dimensional learned representation capturing essential data structure
Embedding	A dense vector representation of a discrete entity (word, gene, user, etc.)
Loss Function	A function measuring the discrepancy between predictions and true values
Inference	The process of using a trained model to make predictions on new data

24. Summary

Machine learning provides a broad toolkit — from linear models and tree ensembles to deep neural networks, transformers, and reinforcement learning agents — for extracting patterns from data across supervised, unsupervised, and reward-driven paradigms. A sound workflow spanning data preprocessing, model selection, rigorous evaluation, interpretability, and responsible deployment is essential to building reliable AI/ML systems. These techniques extend naturally into specialized scientific domains such as transcriptomics, where dimensionality reduction, clustering, and classification support the analysis of complex, high-dimensional gene expression data.

25. References

Bishop, C. M. (2006). Pattern recognition and machine learning. Springer.

Géron, A. (2022). Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow (3rd ed.). O'Reilly Media.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT Press.

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2021). An introduction to statistical learning (2nd ed.). Springer.

Love, M. I., Huber, W., & Anders, S. (2014). Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2. Genome Biology, 15(12), 550. https://doi.org/10.1186/s13059-014-0550-8

Lundberg, S. M., & Lee, S. I. (2017). A unified approach to interpreting model predictions. Advances in Neural Information Processing Systems, 30, 4765-4774.

Luecken, M. D., & Theis, F. J. (2019). Current best practices in single-cell RNA-seq analysis: A tutorial. Molecular Systems Biology, 15(6), e8746. https://doi.org/10.15252/msb.20188746

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., & Hassabis, D. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540), 529-533. https://doi.org/10.1038/nature14236

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12, 2825-2830.

Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). "Why should I trust you?": Explaining the predictions of any classifier. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 1135-1144. https://doi.org/10.1145/2939672.2939778

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need. Advances in Neural Information Processing Systems, 30, 5998-6008.

Wolf, F. A., Angerer, P., & Theis, F. J. (2018). SCANPY: Large-scale single-cell gene expression data analysis. Genome Biology, 19, 15. https://doi.org/10.1186/s13059-017-1382-0

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Le Scao, T., Gugger, S., ... Rush, A. M. (2020). Transformers: State-of-the-art natural language processing. Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, 38-45. https://doi.org/10.18653/v1/2020.emnlp-demos.6

AI/ML in Transcriptomic Data Analysis

Dr. Ritwika Das

ICAR-Indian Agricultural Statistics Research Institute, Pusa, New Delhi – 110012

- **Introduction to Transcriptomics**

Genetic information flows from DNA to RNA to protein, and transcription is the first regulated step of this flow. The transcriptome is the complete set of RNA transcripts present in a cell or tissue at a given time and condition. The essential distinction is that the genome is static whereas the transcriptome is dynamic and condition-specific: the same genome gives rise to very different phenotypes depending on which genes are switched on, by how much, and when. Every method described below exists because expression data are dynamic, noisy and measured only indirectly. Three platforms dominate profiling today (Table 1), of which RNA-seq is the standard for bulk work and the data type assumed throughout.

Table 1: Comparison of major transcriptome profiling platforms

Feature	Microarray	RNA-seq	Single-cell RNA-seq
Principle	Hybridization to fixed probes	Sequencing of cDNA fragments	Sequencing per individual cell
Output value	Fluorescence intensity	Integer read counts	Sparse integer counts
Resolution	Bulk tissue average	Bulk tissue average	Individual cell
Dynamic range	Limited, saturates	Very wide	Wide but zero-inflated

The output value row matters most: a microarray yields continuous bounded intensities whereas RNA-seq yields over-dispersed integer counts, and that single difference dictates which normalization and which statistical model are appropriate. A bulk RNA-seq experiment proceeds through replicated sample collection, RNA extraction with verification of RIN and purity, library preparation, sequencing, quality control of the FASTQ files with FastQC and MultiQC, and quantification by alignment (HISAT2, STAR) or pseudoalignment (salmon, kallisto), yielding a gene × sample count matrix. Collection and extraction set the ceiling on data quality; randomising samples across extraction days and sequencing runs prevents batch effects that no software can later remove.

- **Gene Expression Data**

In an expression matrix, genes occupy the rows and samples the columns, and each cell holds the abundance of one gene in one sample (Table 2). A real matrix has 20,000 to 60,000 rows and often fewer than 50 columns, and this imbalance drives everything that follows.

Table 2: Toy expression matrix for three genes across two control and two stressed samples

Gene	Control_1	Control_2	Stress_1	Stress_2	Interpretation
Gene A	20	22	150	160	≈7-fold rise; candidate upregulated
Gene B	300	280	50	45	≈6-fold fall; candidate downregulated
Gene C	95	101	98	104	Unchanged; the majority class

Expression may be reported as raw read counts, as counts per million (CPM = counts / total counts × 10^6), as RPKM or FPKM (CPM divided by gene length in kb), as TPM (length-normalized then scaled to sum to 10^6), or as values normalized by TMM in edgeR or median-of-ratios in DESeq2. A gene with high counts is not necessarily highly expressed; it may sit in a deeply sequenced sample or be a very long transcript. A frequent error is comparing FPKM across samples: because the denominator differs per sample, FPKM columns do not sum to a constant and are not comparable. TPM corrects this, and TMM or median-of-ratios goes further by handling composition. For machine learning, use $\log_2(\text{CPM} + 1)$.

- **Statistical characteristics of expression data**
  - **High dimensionality:** tens of thousands of genes (p) on a few dozen samples (n), the p >> n regime. Some gene will separate any two groups by chance alone, so overfitting is the default state.
  - **Noise and biological variation:** technical noise from library preparation and sequencing is superimposed on genuine variability between individuals, so replication is mandatory.
  - **Batch effects:** systematic differences between sequencing runs, kits, operators or sites. If batch tracks condition, the model learns the batch instead of the biology.
  - **Sparsity:** many zero entries, mild in bulk RNA-seq and severe (80–95%) in single-cell data, so distance-based methods degrade.

- **Differential Gene Expression**

  Genes are assigned to three categories: upregulated, downregulated, and unchanged, the last being the large majority in any real experiment. Four parameters define them. Fold change (FC) is the ratio of mean expression in the treated group to that in the control group. $\log_2$ fold change is the same ratio on a logarithmic scale, so that +1 denotes a doubling and −1 a halving; being symmetric and additive, it is what is used in practice. The p-value is the probability of observing a difference this large if the gene were truly unchanged, and the adjusted p-value or FDR applies the Benjamini–Hochberg correction across thousands of simultaneous tests, which is essential because with 20,000 tests roughly 1,000 genes would reach $p < 0.05$ by chance alone.

  The typical calling rule is: upregulated if FDR < 0.05 and $\log_2\text{FC} \geq +1$, downregulated if FDR < 0.05 and $\log_2\text{FC} \leq -1$. Both conditions must be insisted upon, since a gene may show a large fold change with no statistical support because it is noisy and lowly expressed, or high significance with a trivial fold change because it is highly expressed with very low variance. Results are inspected using the volcano plot ($\log_2$FC against $-\log_{10}$ p-value, with regulated genes in the two upper corners), the MA plot (mean expression against log ratio, which signals a normalization problem if fold change depends on expression level), and the heatmap. If the shortlisted genes do not visibly separate the sample groups, the selection is probably noise whatever the p-values say.

- **Data Preprocessing**

  Missing values are common in microarray data and after merging platforms; they may be dropped, imputed with the row or column mean, or estimated by k-nearest-neighbour imputation. Note that scikit-learn never accepts NaN values. Genes with almost no reads inflate the multiple-testing burden, so a common rule is to retain genes with CPM > 1 in at least as many samples as the smallest group, which typically removes 40–60% of annotated genes. Genes with near-zero variance should also be dropped, and a sample correlating poorly with its own replicates is a failed library. Such filtering uses no label information and may safely be applied to the whole dataset before splitting.

Normalization is required because sequencing depth differs between samples and because counts span six orders of magnitude with variance growing with the mean. Without it, a model learns library size rather than the treatment effect. Log transformation, $\log_2(x + 1)$, compresses the long right tail and stabilises variance, the added constant avoiding log(0). Z-score scaling, $z = (x - \mu) / \sigma$, centres each feature at zero with unit variance and is required for SVM, k-NN, PCA and k-means, with μ and σ estimated on the training data only. Min-max normalization, $x' = (x - \min) / (\max - \min)$, rescales to [0, 1] but is very sensitive to a single outlier. The order matters: filter → normalize library size → log transform → split → scale on the training set. Scaling before splitting leaks the test distribution into training and inflates every downstream metric.

With thousands of genes and few samples, fewer and better features almost always outperform more features. Statistical selection ranks genes independently of any model, by differential expression (FDR < 0.05 and $|\log_2 FC| > 1$), by variance, or by mutual information with the outcome. Machine learning based selection includes Random Forest feature importance, recursive feature elimination, LASSO whose L1 penalty shrinks most coefficients exactly to zero, and elastic net, which retains correlated pathway genes together. Stability selection, running LASSO on many bootstrap resamples and keeping only genes chosen in 80% of runs, gives a far more reproducible panel. For visualization, PCA gives an interpretable linear projection ordered by variance, since its loadings name the driving genes, while t-SNE and UMAP preserve local neighbourhoods non-linearly. Separation along PC1 by condition means the treatment effect is strong enough to model; separation by sequencing run means a batch effect. Never run statistics on t-SNE or UMAP coordinates, as distances there are not meaningful.

- **Machine Learning Approaches**

  Machine learning is the subset of artificial intelligence concerned with algorithms that let a computer evolve its behaviour from empirical data without being explicitly programmed. Supervised learning builds a model from data in which inputs (x) are paired with labelled outputs (y), and divides into classification, where outputs are finite, and regression, where the output may take any real value; common algorithms are logistic regression, naïve Bayes, k-NN, SVM, decision trees, Random Forest and neural networks. Unsupervised learning works without labels, grouping points by similarity to reveal hidden structure; it divides into clustering and association, and includes hierarchical clustering, k-means, DBSCAN, PAM, CLARA, self-organising feature maps and PCA.

  The classification problem may be set up in two distinct ways, and the choice must be deliberate. In sample-level classification an instance is a sample and the features are genes, giving n instances and p features, the $p >> n$ regime; this predicts disease class, stress condition or resistant versus susceptible genotype. In gene-level classification an instance is a gene and the features are its values across samples; this predicts whether each gene is upregulated, downregulated or unchanged, and the transposition it requires is the most confusing part of the task for newcomers. In either case the pipeline runs from the expression matrix through preprocessing, feature selection, model training and prediction to biological interpretation, with roughly 80% of project effort in the first three stages.

  - **Support Vector Machine**

  An SVM separates two classes using a hyperplane, a plane in higher-dimensional space. Two parallel hyperplanes are constructed so that the classes, defined as $y = 1$ and $y = -1$, lie on either side. Only the points lying on these hyperplanes are needed to determine their equations, and these are termed support vectors. Classification is most accurate when the margin between them is maximised, and they are then called optimal hyperplanes. SVM is highly efficient for binary classification, and non-linearity is handled by replacing the

linear kernel with a non-linear one; applications include text classification, time series analysis and protein fold detection.

- **Decision Tree and Random Forest**

A decision tree is a tree-structured classifier in which internal nodes represent features, branches represent decision rules and leaf nodes represent outcomes. One criterion is applied at each step, splitting the data into subtrees according to the answer. Both classification and regression trees exist, and the principal disadvantage of a single tree is overfitting. Random Forest addresses this through ensemble learning: separate trees are grown on different subsets of the data, and the algorithm takes every tree's prediction and applies a majority voting rule. More trees raise accuracy and guard against overfitting. Random Forest handles large high-dimensional datasets, trains quickly, and tolerates missing data; in agricultural research it has been applied to disease risk prediction, land-use delineation and root-zone soil moisture estimation.

- **Training, Validation and Data Leakage**

The training set, typically 70–80% of the data, is used to fit the model; the test set, typically 20–30%, is held out completely and used once, to report final performance. Cross-validation repeatedly splits the training data so that model choice does not depend on one lucky split. In k-fold cross-validation the training data are divided into k equal parts, commonly five or ten; the model is trained on k−1 and validated on the remainder, rotating until every block has been held out once, and the k scores are averaged. Stratified cross-validation preserves class proportions in every fold and should be the default under imbalance. Leave-one-out sets k = n and is nearly unbiased but of very high variance. Nested cross-validation, with an inner loop for tuning and an outer loop for performance estimation, is the honest option whenever tuning is performed. Four mechanisms leak the test set into training:

1. Scaling or normalizing before the split, so the test distribution enters training.
2. Selecting genes on the full dataset and then cross-validating. This is the classic error and can turn pure noise into 90% apparent accuracy.
3. Batch perfectly confounded with class, so the model learns the sequencing run rather than the biology.
4. Tuning hyperparameters against the test set and reporting the best score as an independent estimate.

Filtering by expression level is label-free and therefore safe before splitting; anything that uses the outcome, feature selection above all, must sit inside the cross-validation loop.

- **Evaluation of Classification Performance**

The confusion matrix (Table 3) is the basis of all classification metrics.

Table 3: Confusion matrix for a binary classifier

	Predicted positive	Predicted negative
Actual positive	TP (True Positive)	FN (False Negative)
Actual negative	FP (False Positive)	TN (True Negative)

  - **Accuracy = (TP + TN) / N** — overall proportion correct, but misleading under imbalance: where 88% of genes are unchanged, a model predicting 'unchanged' for everything scores 0.88.
  - **Precision = TP / (TP + FP)** — of the genes called upregulated, how many really are. High precision means few wasted follow-ups.

- **Recall = TP / (TP + FN)** — of the truly upregulated genes, how many were found; usually what a biologist cares about most.
- **F1-score = 2PR / (P + R)** — harmonic mean of precision and recall; report the macro average so rare classes count equally.

The ROC curve plots true positive rate against false positive rate as the decision threshold is swept from strict to permissive, and the AUC is the area under it, interpretable as the probability that a randomly chosen positive is ranked above a randomly chosen negative. Values of 0.9–1.0 are excellent, 0.8–0.9 good, 0.7–0.8 fair and 0.5 no better than chance. Being threshold-free, AUC measures ranking quality rather than performance at one cut-off, but it is optimistic under strong imbalance because the large true-negative pool inflates it; when the positive class is rare, report the precision–recall curve too. Always state the majority-class rate as a baseline, and prefer macro F1, MCC and per-class recall under imbalance. In a worked three-class experiment a linear SVM and a Random Forest both exceeded ROC-AUC 0.96, which looks excellent in isolation; on macro F1, however, they scored 0.785 and 0.796, while a plain Welch t-test with Benjamini–Hochberg correction scored 0.840. The classical test won — not a failure of machine learning, but the correct answer for a clean two-group design.

- **Clustering in Transcriptomics**

  Classification is supervised: groups are predefined, labelled data are required, and results are evaluated against true labels. Clustering is unsupervised: groups are discovered from the data itself and evaluation is internal, by silhouette width, or external, against known biology. In transcriptomics it identifies co-expressed genes, often co-regulated by a shared transcription factor or belonging to a common pathway, which is the basis of guilt-by-association annotation; groups samples to reveal disease subtypes, tolerant versus susceptible genotypes, or unexpected batch structure; and discovers early, late and transient response programmes across a time course.

  Hierarchical clustering recursively partitions the dataset using distance measures such as Euclidean distance and Jaccard's coefficient, or similarity measures such as Pearson correlation. In agglomerative clustering each object initially forms its own cluster and clusters are successively merged (bottom-up); in divisive clustering all objects begin together and are successively divided (top-down). Merging criteria include single, complete and average linkage, centroid distance and Ward's distance. K-means is a non-hierarchical, centroid-based technique partitioning n objects into k clusters, assigning each point to its nearest centre and recalculating centroids until the grouping stabilises. The essential caution is that both methods always return an answer whether or not structure exists: hierarchical clustering will draw a tree for pure noise, and k-means will partition it into exactly k groups. Validation by bootstrap support or silhouette width is what separates genuine structure from decoration.

- **Hands-on Implementation in Python**

  The worked example uses a bulk RNA-seq dataset of 4,000 genes on 48 crop leaf samples, 24 control and 24 drought-stressed, of which 150 genes were truly up and 150 truly down by construction. Steps using no class labels may safely be run on the whole matrix before splitting.

```
import numpy as np, pandas as pd
from scipy import stats
counts = pd.read csv("gene expression counts.csv", index col=0)
grp    = pd.read csv("sample metadata.csv")["condition"].values

cpm    = counts / counts.sum(axis=0) * 1e6      # library-size normalization
keep   = (cpm > 1).sum(axis=1) >= 12            # low-expression filter
logcpm = np.log2(cpm[keep] + 1.0)               # variance stabilisation

c = logcpm.values[:, grp == "control"]; d = logcpm.values[:, grp == "drought"]
```

```
t, p = stats.ttest_ind(d, c, axis=1, equal_var=False)   # Welch t-test
lfc  = d.mean(axis=1) - c.mean(axis=1)
fdr  = stats.false_discovery_control(p)                 # Benjamini-Hochberg
de = pd.DataFrame({"gene": logcpm.index, "log2FC": lfc, "FDR": fdr})
de["reg"] = np.where((de.FDR < .05) & (de.log2FC >= 1), "Up",
             np.where((de.FDR < .05) & (de.log2FC <= -1), "Down", "Unchanged"))
# (3953, 48)   {'Unchanged': 3895, 'Down': 30, 'Up': 28}
```

Note scipy.stats.false_discovery_control, added in SciPy 1.11, which performs the Benjamini–Hochberg correction in one call; hand-coded versions are a common source of quiet errors. Classification follows, with feature selection and scaling inside the pipeline.

```
from sklearn.pipeline import Pipeline
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import (train_test_split, StratifiedKFold,
                                     cross_val_score)
X = logcpm.T.values                    # samples x genes
y = (grp == "drought").astype(int)
X_tr, X_te, y_tr, y_te = train_test_split(X, y, test_size=0.30,
                                          stratify=y, random_state=42)

svm = Pipeline([("select", SelectKBest(f_classif, k=100)),
                ("scale",  StandardScaler()),
                ("clf",    SVC(kernel="linear", C=1.0, probability=True,
                                class_weight="balanced", random_state=42))])
rf  = Pipeline([("select", SelectKBest(f_classif, k=100)),
                ("clf",    RandomForestClassifier(n_estimators=500,
                               max_features="sqrt", min_samples_leaf=2,
                               class_weight="balanced", oob_score=True,
                               n_jobs=-1, random_state=42))])

cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
for name, mdl in [("SVM", svm), ("RF", rf)]:
    sc = cross_val_score(mdl, X_tr, y_tr, cv=cv, scoring="f1_macro")
    print(name, "CV macro-F1: %.3f +/- %.3f" % (sc.mean(), sc.std()))
    mdl.fit(X_tr, y_tr)
# SVM CV macro-F1: 0.874 +/- 0.064   test accuracy 0.800, AUC 0.911
# RF  CV macro-F1: 0.872 +/- 0.182   OOB 0.909,  AUC 0.768
```

The single most important line is that SelectKBest sits inside the Pipeline, so the 100 genes are re-chosen from scratch within every fold; run on the whole matrix before splitting, the same code returns accuracy near 1.0 even on permuted labels. The two spreads are instructive: the SVM varied by 0.064 across folds against the forest's 0.182, so with 33 training samples the forest is far less stable. Its accuracy of 0.800 also conceals the real problem — it finds every control but only four of seven drought-stressed plants, so precision is perfect while recall is poor. The SVM ranks better at the same accuracy, and lowering its threshold trades precision for recall according to what each error costs.

Unsupervised analysis uses scipy.cluster.hierarchy.linkage with seaborn.clustermap on the top differential genes, z-scored by row, with Ward linkage. Cutting the sample dendrogram at k = 2 recovers the design with 85.4% agreement, and the samples it misplaces are the same weak responders the classifier misclassified. Running sklearn.cluster.KMeans over k = 2 to 8 on the 300 most significant genes and scoring with silhouette_score gives an ambiguous elbow but a silhouette peaking sharply at k = 2, the more principled criterion. The two modules of 163 and 137 genes rise and fall under drought and should next go separately to GO enrichment. For publication, prefer permutation importance to Gini importance, which is biased toward high-variance genes.

- **Summary and Key Takeaways**
  1. Expression data are high-dimensional, noisy, compositional and heavily imbalanced. Every preprocessing step answers one of these properties.
  2. Upregulated and downregulated are defined by two numbers together, an effect size ($\log_2$FC) and a confidence measure (FDR). Both thresholds are conventions and must be stated.

3. Normalize, then log transform, then split, then scale. Feature selection uses the labels and therefore belongs inside the cross-validation loop.
4. SVM finds one maximum-margin boundary while Random Forest averages hundreds of decorrelated trees. They fail differently, so running both is informative.
5. Accuracy is the wrong headline metric when 88% of genes are unchanged. Report macro F1, MCC, per-class recall and the confusion matrix.
6. Clustering always returns an answer whether or not structure exists. Validate with bootstrap support or silhouette width before believing a dendrogram.
7. A gene list is only a hypothesis until supported by enrichment analysis and confirmed on an independent cohort or by qRT-PCR.
8. Always compare against a classical statistical baseline: for a clean two-group design a t-test with FDR correction is often the better tool.

- **References**

Benjamini, Y. and Hochberg, Y. (1995). Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society: Series B*, **57**, 289 – 300.

Breiman, L. (2001). Random forests. *Machine Learning*, **45**, 5 – 32.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine Learning*, **20**, 273 – 297.

Pedregosa, F., Varoquaux, G., Gramfort, A. *et al*. (2011). Scikit-learn: machine learning in Python. *Journal of Machine Learning Research*, **12**, 2825 – 2830.

Robinson, M. D., McCarthy, D. J. and Smyth, G. K. (2010). edgeR: a Bioconductor package for differential expression analysis. *Bioinformatics*, **26**, 139 – 140.

Single-Cell RNA-Seq Data Analysis

Sudhir Srivastava and Mayank Rashmi

Introduction

Over the past two decades, the development of advanced high-throughput technologies has transformed research across diverse fields, including biomedicine, agriculture, and environmental sciences, by enabling investigations at the cellular and subcellular levels. The term 'omics' refers to the comprehensive studies of the genome, transcriptome, epigenome, proteome, and metabolome of a given sample using high-throughput approaches. Single-cell sequencing is a powerful approach that profiles individual cells to study their genome, transcriptome, epigenome, proteome, and other omics layers. Traditional sequencing, also known as bulk sequencing, analyzes the DNA or RNA from a group of cells, providing an average signal across the population. While useful for identifying general patterns in large groups of cells, bulk sequencing cannot capture the unique differences between individual cells. As a result, bulk sequencing is less suitable for studying complex systems with diverse cell types or for detecting rare cell populations and subtle cellular variations. Single-cell sequencing is a newer technology that enables the exploration of genomics, transcriptomics, and other omics layers at single-cell resolution, allowing researchers to distinguish cell populations, track cellular responses, and uncover evolutionary relationships among cells. To date, a number of technologies have been proposed for single-cell transcriptomic studies, and these are differentiated by at least one of the aspects, like cell isolation, cell lysis, reverse transcription, amplification, transcript coverage, strand specificity, and unique molecular identifiers (UMIs). UMI detects and quantifies the availability of unique transcripts. Single-cell RNA Sequencing (scRNA-Seq) allows the comparison of transcriptomes at the level of individual cells. Its major application is to assess transcriptional similarities and differences within a cell population. This enables the identification of rare cell types, characterization of cellular heterogeneity, analysis of developmental trajectories, and investigation of disease-associated transcriptional changes. scRNA-Seq was studied for the first time in 2009 by Tang and their group. Several other next-generation sequencing (NGS)-based assays have been updated for single-cell methods in addition to single-cell RNA-seq. These include assays for genomics, proteomics, and epigenetics, particularly single-cell ATAC-sequencing, which is frequently carried out in combination with scRNA-Seq. Various scRNA-Seq platforms and techniques differ

in terms of transcript coverage (3'/5' tag-based vs. whole-transcript) and throughput (number of cells), and are used to analyse differently. Single-cell RNA sequencing helps in quantification of mRNA expression levels in each cell. The Transposase-Accessible Chromatin Assay for Single-cell Assay utilizing Sequencing (scATAC) describes the openness of cis-regulatory elements in neighboring genes. When scRNA-Seq and scATAC data are analyzed together, they can reveal gene regulatory linkages associated with cellular heterogeneity and enhance the critical genetic information from other omics. Single-cell DNA sequencing (scDNA-seq) technologies offer an unusual opportunity to examine individual cell clonality and the sequence of mutations, both factors that have a big impact on therapeutic results by analyzing DNA mutations and copy number variations.

General Workflow of Single-cell Sequencing

Single-cell sequencing is a process that isolates a single cell for sequencing, then studies molecular mapping, cell heterogeneity, epigenetic change, and immune infiltration. Firstly, isolate the single cells from a tissue sample of interest that includes a few steps, like micro-dissection, microfluidic platforms, and droplet-based methods. Next to perform a list of single tasks in a way that preserves cellular mRNA. mRNA molecule captured by using poly(T) sequence primers that bind to mRNA poly(A) tails. Then convert poly (T)-primed mRNA into cDNA using reverse transcription. This reverse transcribed cDNA is usually amplified by PCR or in vitro transcription. A cDNA library is prepared by inserting index nucleotide barcodes that identify each library. After that pooled the cDNA libraries and made the sequencing libraries by NGS techniques.

There are many commercial kits and reagents are now available for the entire wet-lab procedure for scRNA-Seq. The steps from the beginning (lysing cells) to the end (sequencing) are depicted below in Figure 1. The workflow for scRNA-Seq data generation, from the initial step of cell lysis to the final step of sequencing, is depicted in Figure 1. Some of them are based on the switching mechanism at 5′ end of RNA template (SMARTer) procedure for mRNA capture, reverse transcription, and cDNA amplification. More recently, a number of droplet-based platforms like Chromium from 10x Genomics, ddSEQ from Bio-Rad Laboratories, InDrop from 1CellBio, and μEncapsulator from Dolomite Bio/Blacktrace Holdings are commercially available.

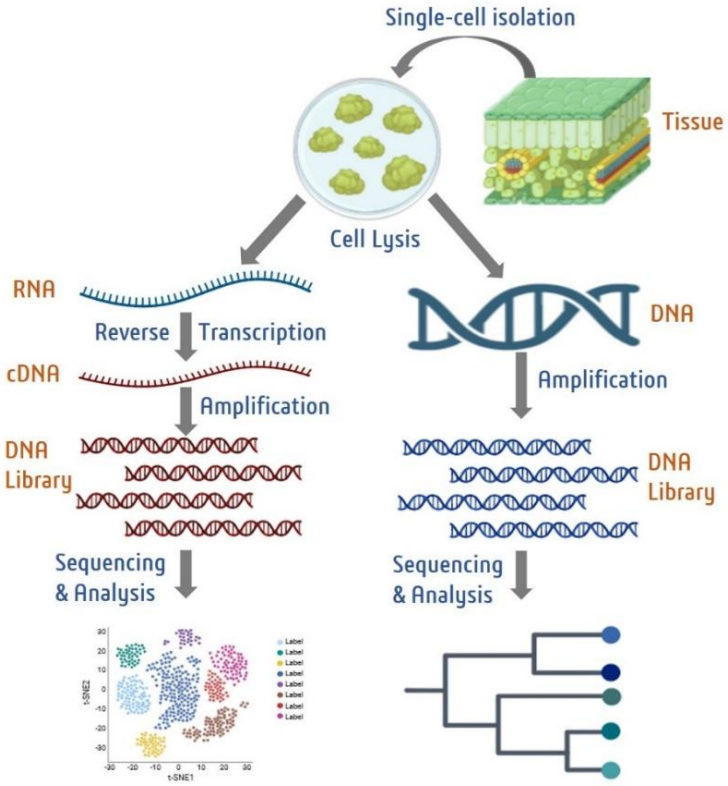


Figure 1. Basic steps of single-cell sequencing

A comparison of different scRNA-Seq technology and experimental protocols is given below in Table 1.

Table 1: A comparison of different scRNA-seq technology and experimental protocols

Platform / Protocol	Capture Strategy	Transcript Coverage	Throughput	Strengths	Limitations
Smart-Seq / Smart-Seq2	Plate-based (FACS)	Full-length	Hundreds	High sensitivity, detects splice variants	Low throughput, expensive per cell
Fluidigm C1	Microfluidic chip (single-cell capture)	Full-length	Hundreds	Automated handling, high-quality cDNA	Limited cell number, expensive chips
MATQ-seq	Plate-based / low-input	Full-length	Tens-Hundreds	Very sensitive, works for low RNA input	Low throughput, specialized protocol
Drop-seq	Droplet microfluidics	3′-end	Thousands	High throughput, low cost	Lower sensitivity, more dropouts

Experimental design plays a pivotal role in scRNA-Seq studies. Before selecting a protocol, it is essential to carefully assess key factors that may affect data quality, cost, and the biological insights obtained. First, the number of cells to be sequenced per experiment must be considered. This largely depends on the overall heterogeneity of the sample and the expected proportion of specific cell types, based on prior knowledge. An online estimator developed by the Satija Lab provides guidance on how many cells should be sampled in an scRNA-seq experiment to reliably capture a minimum number of cells from each cell type within the dataset (https://satijalab.org/howmanycells/). Second, cell size is an important factor. Each platform has its limitations in handling cells of different sizes. Smaller cells (typically < 25 μm in diameter) can usually be processed with minimal damage, whereas larger or irregularly shaped cells, such as adult cardiomyocytes and neurons, pose greater challenges. In such cases, single-nucleus RNA sequencing (snRNA-Seq) can serve as a practical alternative. Third, minimizing technical biases remains critical. Although new analytical methods continue to improve bias correction, distinguishing true biological variation from technical noise is still challenging. Therefore, careful experimental design that reduces confounding factors is essential.

Single-Cell Data Analysis

Sequenced data of scRNAs can be generated experimentally or retrieved from public repositories such as GEO, SRA, scRNASeqDB, and PlantscRNAdb, etc. Depending on the library preparation method used, the RNA sequences (reads or tags) may be derived either from the 3′ ends or 5′ ends of the transcripts (e.g., 10X Genomics, CEL-seq2, Drop-seq, inDrops) or from full-length transcripts (e.g., Smart-seq). Sequenced data is further analyzed by bioinformatics pipeline using various software and tools. The basic steps of scRNA-Seq data analysis are discussed below.

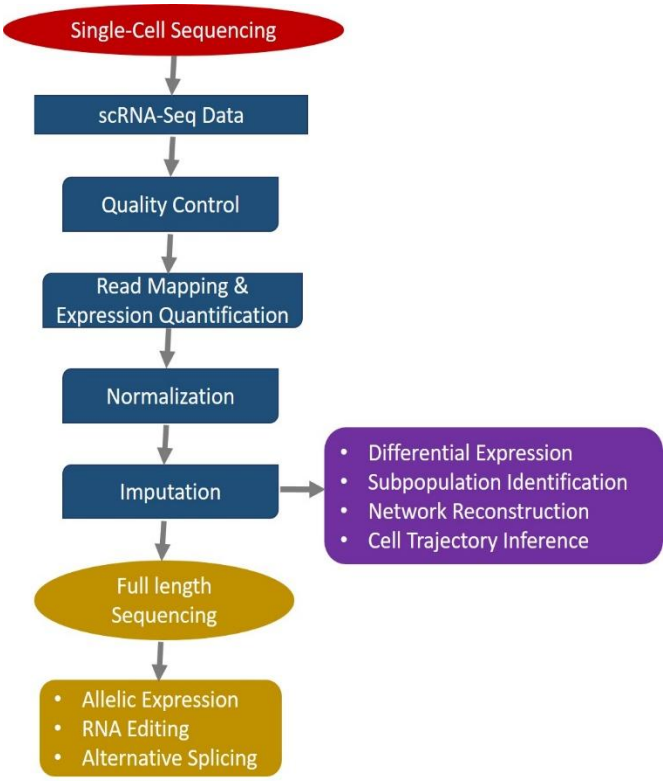


Figure 2. Basic steps of scRNA-Seq data analysis

- **Preprocessing raw scRNA-Seq data:** This step involves various steps such as formatting reads, demultiplexing samples, quality control, trimming, mapping, and quantification.

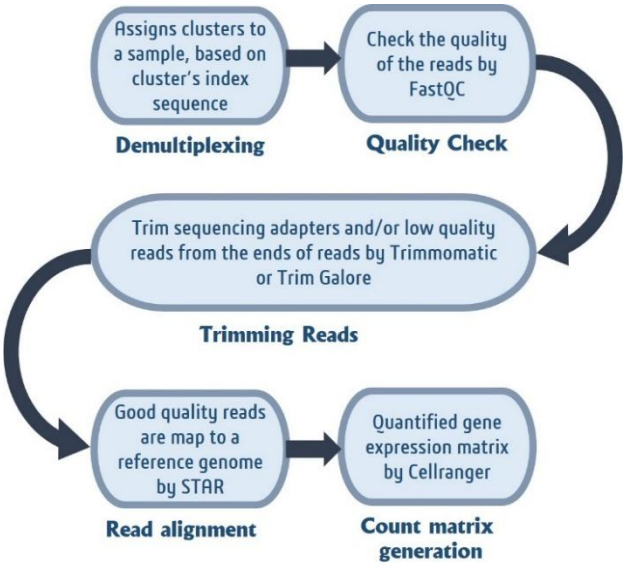


Figure 3. Processing of scRNA sequencing data

- **Demultiplexing:** Raw single-cell transcriptome sequencing data are typically generated in either FASTQ or BCL format, depending on the sequencing platform. Since only FASTQ files can be directly used for quality control, BCL files must first be converted into FASTQ format using appropriate tools. For example, the cellranger mkfastq pipeline, which wraps the bcl2fastq software, is commonly used for this conversion. A simple CSV sample sheet containing at least three columns (lane, sample, and index) must be provided along with the BCL file path. Once converted, the resulting FASTQ files can be assessed for quality using tools such as FastQC.
- **Quality control**: In scRNA-Seq experiments, some low-quality data are generated from cells that are broken, dead, or contaminated with multiple cells. Such low-quality cells can negatively impact downstream analyses and lead to misinterpretation of results. FastQC is a widely used tool for general quality checks and noise removal, while SinQC and Scater are commonly applied for scRNA-Seq–specific quality control.
- **Trimming**: This step removes sequencing adapters and low-quality bases from the ends of reads to improve data reliability. After trimming, data quality is reassessed to ensure clean input for downstream analysis. Commonly used trimming tools for scRNA-seq include Trimmomatic, Cutadapt, TrimGalore, fastp, *etc*.
- **Mapping**: High-quality reads are aligned to a reference genome to identify their gene of origin. STAR is widely used for scRNA-Seq, as it efficiently finds the longest matching sequence within the reference genome. Alternative aligners such as HISAT2, Kallisto, and Salmon are also employed, depending on the analysis goals and computational requirements. Unlike traditional alignment, Kallisto and Salmon use pseudoalignment, which rapidly assigns reads to transcripts without fully mapping each base, thereby reducing computational time and memory usage. Unique Molecular Identifiers (UMIs), which are short molecular tags added to transcripts prior to sequencing, help correct for amplification bias. Reads sharing the same UMI, gene, and cell barcode are collapsed into a single count, ensuring more accurate quantification.

- **Quantification/ Count matrix formation:** The number of reads or UMIs mapping to each gene is counted for every cell. In the resulting count matrix, genes are represented as rows and cells as columns, with each entry indicating the quantified expression level of a gene in a specific cell. For 10X Genomics data, the count matrix is typically generated using the Cell Ranger pipeline. Other commonly used quantification tools include STARsolo, Kallisto|bustools, Salmon/Alevin, HTSeq, Drop-seq Tools, *etc*. which provide flexibility for different scRNA-Seq platforms and analysis needs.

	Cell1	Cell2	Cell3	.....	CellN
Gene1	4	8	22	.	13
Gene2	7	2	3	.	6
Gene3	1	3	9	.	0
.....	.	.	.	.	.
.....	.	.	.	.	.
.....	.	.	.	.	.
GeneM	30	0	17	.	5

Figure 4. Count matrix representation

- **Batch Effect Correction:** The large data is generated in different scales of time, and sometimes data is produced by different laboratories using various protocols, library preparation, and sequencing platforms. So, there is some technical and biological variability, as well as systematic errors are defined as batch effects. The two batch correction methods are MNN (mutual nearest neighbor), used for similar cells in different batches, and kBET (k-nearest neighbor batch effect test), based on the $\chi 2$ method.
- **Normalization and Clustering:** Normalization of scRNA-Seq data is a critical step to remove technical biases (e.g., sequencing depth, capture efficiency) and enable meaningful biological comparisons. Common normalization approaches include log-normalization, scaling by library size, and advanced methods such as SCTransform (Seurat) or scran. Imputation is often applied to address missing values or dropouts, with tools such as MAGIC, SAVER, scImpute, and ALRA helping to recover gene expression signals. After normalization and imputation, dimensionality reduction and clustering are performed to explore cell-to-cell variability. Linear methods such as Principal Component Analysis (PCA) are used for initial dimensionality reduction, while non-linear approaches like t-SNE (t-distributed stochastic neighbor embedding) and UMAP (Uniform Manifold

Approximation and Projection) are widely used for visualization and clustering of cell populations.

- **Marker Identification and Analysis**: Marker identification is a key step in scRNA-Seq analysis to discover disease-relevant genes, understand transcriptional dynamics, and characterize both protein-coding and noncoding RNAs at the single-cell level. The key goals of the scRNA-Seq data analysis are as follows:

✓ Identify cell subpopulations (which are often distinct cell types) within a specific condition or tissue to unravel the heterogeneity of cells

Methods/Tools: Clustering (Seurat, Scanpy, SC3, SIMLR), trajectory inference (Monocle, Slingshot, PAGA).

✓ Find the significantly differentially expressed genes between distinct subpopulations or groups of cells

Methods/Tools: Seurat (FindMarkers), DESeq2, edgeR, MAST, limma-trend

✓ Alternative splicing: For the alternative splicing, generally five basic modes are recognized, including exon-skipping (cassette exon), mutually exclusive exons, alternative donor site, alternative acceptor site, and intron retention.

Methods/Tools: Seurat (FindMarkers), DESeq2, edgeR, MAST, limma-trend.

✓ Detection of RNA-editing dynamics

Tools: REDItools, RNAEditor, RES-Scanner, GIREMI.

✓ Identification of equally expressed genes between parental and maternal genomes by allelic expression analysis

Tools: WASP, ASEReadCounter (GATK), MBASED, scBASE.

✓ Gene regulatory network inference: The goal is to reveal transcription factor–target relationships and regulatory modules.

Tools: REDItools, RNAEditor, RES-Scanner, GIREMI

Advantages of Single-Cell Sequencing

- Genetic material at the single-cell level can be studied with the help of single-cell sequencing. This offers a deeper understanding of the variations inside cells.
- Rare cell types that are not identified by bulk sequencing can be found using it. This is helpful in areas like immunology and cancer studies.

- In a particular tissue, it offers extensive information about individual cells.
- By analyzing the cellular diversity and composition of every cell in a tissue sample, it can be employed to investigate complicated biological systems.
- It helps to improve our knowledge of complexities in tissues and various cell types.

Limitations of Single-Cell Sequencing

- The high cost of single-cell sequencing restricts its use in extensive research.
- Cells may experience stress during the isolation process that influences their viability and characteristics.
- Single-cell sequencing data is noisier than bulk sequencing data, which makes it less reliable.
- A large amount of data is missing or has zero values because single-cell sequencing frequently fails to identify every gene in every cell. It is possible that certain genes exist in a cell but are not detected by sequencing. Data analysis may be challenging due to this sparsity.
- Because it is gathered from a single cell, the data may not be accurate. It might be challenging to determine which measures are accurate and which are the result of technological mistakes.
- Accurately detecting genetic changes in individual cells can be challenging due to the potential for errors and biases introduced by the amplification step.

Applications of Single-Cell Sequencing

- Single-cell sequencing has uses in developmental biology, neurology, diabetes, and cancer research, among other scientific areas.
- Finding genetic differences within tumor cells is helpful in the study of cancer. It helps in understanding the nature of cancer cells and how they function. It is also helpful in the development of targeted therapies.
- Immunology uses it to investigate various immune cells and how they contribute to various diseases.
- In developmental biology, it supports the study of cellular differentiation and the modifications that take place throughout processes like embryogenesis.

- Through the identification of microbial species and their genetic composition, as well as functions in various processes such as antibiotic resistance, single-cell sequencing is also helpful in the study of microbes.
- It is also helpful for understanding the causes of diseases and for clinical diagnostics. It facilitates the identification of unusual or unique cell types and is also helpful for the development of treatment methods for various diseases.

Conclusion

Over the past 13 years, numerous scRNA-Seq protocols have been developed to enhance our understanding of cellular expression variability and dynamics. These methods enable the study of highly heterogeneous samples, including clinical specimens that have undergone fixation or freezing, thereby broadening the scope of experimental applications. Advanced computational analyses of scRNA-Seq data have driven the growth of single-cell transcriptomics, providing powerful tools for both biological and clinical research. By resolving gene expression at single-cell resolution, scRNA-Seq offers deep insights into cellular heterogeneity, transcriptional dynamics, and the regulatory mechanisms underlying diverse biological processes and disease states.

References

Andrews, T. S., Kiselev, V. Y., McCarthy, D., & Hemberg, M. (2021). Tutorial: guidelines for the computational analysis of single-cell RNA sequencing data. *Nature protocols*, **16(1)**, 1–9. https://doi.org/10.1038/s41596-020-00409-w

Buttner, M., Miao, Z., Wolf, F. A., Teichmann, S. A., and Theis, F. J. (2019). A test metric for assessing single-cell RNA-seq batch correction. *Nat. Methods* 16, 43–49. doi: 10.1038/s41592-018-0254-1.

Haghverdi, L., Lun, A. T. L., Morgan, M. D., and Marioni, J. C. (2018). Batch effects in single-cell RNA-sequencing data are corrected by matching mutual nearest neighbors. *Nat. Biotechnol.* 36, 421–427. doi: 10.1038/nbt.4091.

He, J., Lin, L., & Chen, J. (2022). Practical bioinformatics pipelines for single-cell RNA-seq data analysis. *Biophysics reports*, **8(3)**, 158–169. https://doi.org/10.52601/bpr.2022.210041

Ilicic, T., Kim, J. K., Kolodziejczyk, A. A., Bagger, F. O., McCarthy, D. J., Marioni, J. C., et al. (2016). Classification of low quality cells from single-cell RNA-seq data. *Genome Biol.* 17:29. doi: 10.1186/s13059-016-0888-1.

Jovic, D., Liang, X., Zeng, H., Lin, L., Xu, F., & Luo, Y. (2022). Single-cell RNA sequencing technologies and applications: A brief overview. *Clinical and translational medicine*, **12(3)**, e694. https://doi.org/10.1002/ctm2.694

Tang, F., Barbacioru, C., Wang, Y., Nordman, E., Lee, C., Xu, N., et al. (2009). mRNA-Seq whole-transcriptome analysis of a single cell. *Nat. Methods*, 6, 377–382. doi: 10.1038/nmeth.1315.

Important Links

https://satijalab.org/seurat/

https://satijalab.org/howmanycells/

https://www.bioinformatics.babraham.ac.uk/projects/fastqc/

https://www.bioinformatics.babraham.ac.uk/projects/trim_galore/

https://github.com/madsen-lab/valiDrops

https://github.com/alexdobin/STAR?tab=readme-ov-file

https://github.com/alexdobin/STAR/blob/master/docs/STARsolo.md

https://www.scrna-tools.org/

https://github.com/seandavi/awesome-single-cell

https://github.com/JiekaiLab/scDIOR

Seurat - Guided Clustering Tutorial

https://satijalab.org/seurat/articles/pbmc3k_tutorial#setup-the-seurat-object

Extracting Functional and Biological Insights from Transcriptomic Data Using AI/ML

Dr. Anshuman Singh

1. Introduction: From Transcriptomic Data to Biological Insight

Transcriptomics gives us a genome-wide view of the RNA molecules being produced by a biological system at a particular time and under a particular condition. In plant science, RNA sequencing (RNA-seq) has become a powerful way to study development, stress responses, disease resistance and differences among genotypes. But the real value of a transcriptome is not simply the production of a long list of differentially expressed genes. The important question is what those changes mean biologically.

A useful way to think about transcriptomics is to treat the transcriptome as a molecular phenotype. It provides a link between genotype, environment and phenotype and helps us ask why two genotypes behave differently under the same condition. The uploaded lecture therefore places a strong emphasis on moving from individual genes towards pathways, co-expression modules, regulatory networks, candidate genes and, finally, experimentally testable biological mechanisms.

1.1 The biology-first principle

A good transcriptomic analysis starts with a biological question, not with an algorithm. For example, if the objective is to understand drought tolerance, we may ask: Which genes respond to drought? Which pathways distinguish tolerant and susceptible genotypes? Which regulatory programmes coordinate the response? Which molecular changes are reflected in physiological traits? And which genes are strong enough candidates to take forward for validation or breeding?

This way of thinking is especially important when AI and machine learning are used. A model may find a strong association or make an accurate prediction, but that does not automatically mean that the identified gene is causal. Biological interpretation requires context, independent evidence and, wherever possible, experimental validation.

1.2 What can a transcriptome tell us?

At the simplest level, RNA abundance tells us which genes are active or altered in a tissue or condition. When samples are compared, the expression profiles can distinguish tissues, developmental stages, genotypes, treatments and stress states. Time-series experiments can also show how a response develops from an early signalling phase to later metabolic or physiological adjustment.

Transcriptomes can further reveal coordinated gene programmes. Transcription factors, regulatory RNAs and co-expression modules may point towards mechanisms controlling groups of genes. Functional annotation then connects these expression patterns with biological processes, molecular functions, cellular components and pathways. Thus, RNA-seq should be viewed as a starting point for understanding biology rather than simply as a technology for generating DEGs.

1.3 The RNA-seq foundation

The quality of every downstream AI/ML analysis depends on the quality of the transcriptomic dataset. A typical workflow begins with careful experimental design and biological replication, followed by RNA isolation and sequencing, quality control, read preprocessing, alignment or transcriptome assembly, transcript quantification and construction of an expression matrix. Statistical exploration and differential expression analysis then provide the first layer of biological evidence.

Factors such as genotype, treatment, tissue, developmental stage, time point and environment should be planned in advance. Batch effects and poor replication are particularly important because a machine-learning model can easily learn technical differences instead of biological differences. Sample-level quality

assessment, principal component analysis and correlation analysis are therefore useful before any predictive modelling is attempted.

2. From Differential Expression to Functional Biology

2.1 Differential expression is the beginning, not the end

Differential expression analysis tells us which genes change between defined conditions. Fold change, statistical significance and multiple-testing correction help identify robust expression differences. However, a DEG list by itself rarely provides a satisfactory biological explanation. Thousands of genes may respond to a stress, and many of them may be indirect consequences of the primary response.

The more useful question is therefore: which changes occur together and what biological process do they represent? This is where functional enrichment, gene-set analysis and pathway analysis become important. They help convert a large gene list into a smaller number of interpretable biological themes.

2.2 Gene Ontology and pathway interpretation

Gene Ontology analysis organizes genes into Biological Process, Molecular Function and Cellular Component. For example, a drought-responsive gene set may show enrichment for oxidation-reduction processes, response to water deprivation, carbohydrate metabolism, cell-wall organization or hormone signalling. Pathway analysis adds another layer by placing genes within metabolic and signalling pathways.

An important point for students is that enrichment is not merely a statistical exercise. The biological meaning should be checked against the experimental system. If a pathway appears enriched in a drought experiment, we should ask whether the pathway is consistent with the observed physiology and with what is already known about the crop and tissue being studied.

2.3 Co-expression analysis and networks

Individual genes rarely act alone. Co-expression analysis asks which genes show similar expression patterns across samples. Groups of highly coordinated genes can form modules that represent biological programmes such as photosynthesis, oxidative-stress management, hormone signalling or carbohydrate metabolism.

Highly connected genes within such modules are often described as hub genes. These genes are attractive candidates because their position in the network suggests that they may be important components of the observed response. However, a hub is not automatically a master regulator. Co-expression indicates association; it does not prove direct regulation or causation. This distinction should always be made clear when presenting network-based results.

2.4 Moving towards regulatory biology

The next step is to ask what controls the coordinated expression pattern. Transcription factors can regulate groups of downstream genes, while miRNAs and long non-coding RNAs can contribute to post-transcriptional or regulatory control. Promoter information, TF-binding evidence and alternative-splicing patterns can add further context.

This changes the biological question from 'Which genes are changing?' to 'Which regulatory programmes are coordinating these changes?' Such a shift is particularly valuable when the aim is to understand a complex trait rather than describe a single response.

3. Why AI and Machine Learning are Useful for Transcriptomics

Transcriptomic datasets are typically high-dimensional: thousands of genes may be measured across a relatively small number of biological samples. Conventional statistics remain essential, but AI/ML provides additional tools for discovering patterns, predicting phenotypes and ranking informative features.

3.1 Unsupervised learning

Unsupervised methods are useful when we want the data to reveal its internal structure. Principal Component Analysis (PCA) is commonly used to summarize the major sources of variation among samples. It can reveal genotype or treatment effects and can also alert us to possible batch effects.

Clustering groups samples or genes according to similarity and may reveal molecular subtypes, developmental states or stress-response classes. UMAP and t-SNE can provide intuitive visual representations of high-dimensional expression patterns. These methods are exploratory, so clusters should always be followed by biological interpretation using marker genes, pathways and phenotypic information.

4. Supervised Machine Learning and Explainable AI

4.1 Predicting biological phenotypes

Supervised learning is used when the biological outcome is already known. In plant research, classification can distinguish drought-tolerant from drought-sensitive genotypes, resistant from susceptible plants, or infected from healthy samples. Regression can be used when the target is quantitative, such as yield, biomass, metabolite concentration or a physiological measurement.

Random Forest is useful for nonlinear relationships and provides feature-importance information. Support Vector Machines are widely used for high-dimensional classification. Regularized regression methods such as LASSO and Elastic Net are useful when the number of genes is large and feature selection is needed. Neural networks and deep-learning models can represent more complex relationships, although they generally require larger datasets and careful validation.

4.2 The problem of overfitting

One of the biggest challenges in transcriptomic machine learning is the mismatch between the number of features and the number of samples. If thousands of genes are available but only a small number of genotypes are studied, a model can easily memorize the training data. It may then perform extremely well on the training set but poorly on new genotypes or environments.

For this reason, training and testing must be kept strictly separate. Feature selection should be performed within the training process, and independent validation is highly desirable. Cross-validation can provide a better estimate of model performance, but it does not replace genuine external validation.

4.3 Explainable AI: connecting prediction with biology

A biological researcher usually wants more than an accurate prediction. We also want to know why the model made that prediction. Explainable AI (XAI) addresses this problem by estimating how individual features contribute to model output.

Global explanations identify genes that repeatedly influence predictions across the dataset. Local explanations show which genes are important for a particular genotype or sample. Methods such as SHAP and permutation importance can help quantify these contributions. Once important genes are identified, they should be interpreted in the context of pathways, networks, transcription factors, QTLs and known biological functions.

The central caution is that model importance is not the same as biological causality. A predictive gene may be downstream of the true causal gene, correlated with a causal process, or even associated with a technical confounder. XAI therefore improves interpretation, but it does not eliminate the need for biological validation.

5. Candidate-Gene Prioritization Through Evidence Integration

AI/ML is most useful for candidate-gene discovery when it is treated as one evidence stream among several. The strongest candidates are those for which different types of evidence point in the same direction.

- Expression evidence: the gene is differentially expressed or strongly associated with the biological condition.
- AI/ML evidence: the gene makes a substantial contribution to phenotype prediction.
- Functional evidence: the gene has a relevant GO annotation, pathway role or biochemical function.
- Network evidence: the gene belongs to a relevant module or occupies a central network position.
- Genetic evidence: the gene is supported by QTL, GWAS, eQTL, haplotype or allele-specific evidence.
- Validation evidence: previous experiments, mutants, overexpression, silencing, genome editing or biochemical assays support its role.

The most convincing candidates are therefore those supported by converging evidence. A gene selected only because it has high ML importance is much weaker than one that is simultaneously differentially expressed, genetically associated with the trait, located in a relevant network module and supported by functional evidence.

6. Integrating Transcriptomics with Genetics, Physiology and Other Omics

Transcriptomics becomes particularly powerful when it is connected with genetic variation and phenotype. A useful conceptual framework is: genetic variation → QTL/GWAS → eQTL or expression → transcriptomic network → physiological trait → phenotype. QTL and GWAS can narrow a genomic region, while transcriptomics can reveal which genes in or near that region are active or responsive under the relevant condition.

This integration can substantially improve candidate-gene prioritization and can also support the development of functional markers. For crop improvement, the same framework can eventually contribute to genomic prediction and precision breeding, although the cost, stability and tissue-specific nature of transcriptomic measurements need to be considered.

6.1 Abiotic stress applications

Drought-responsive transcriptomes commonly involve ABA signalling, water transport, aquaporins, osmotic adjustment, root development, carbohydrate metabolism, reactive oxygen species (ROS) homeostasis and cell-wall remodelling. Heat stress often activates heat-shock proteins, protein-folding machinery, antioxidant systems and mechanisms that protect membranes and cellular proteins. Salinity responses involve ion transport and homeostasis, osmoprotection, hormone signalling and antioxidant defence. Nutrient stress can alter transporter genes, nutrient-signalling pathways and metabolic adjustment.

AI/ML can add value by identifying combinations of genes that distinguish tolerant and sensitive genotypes. This is important because stress tolerance is usually a complex phenotype controlled by interacting biological processes rather than by a single gene.

6.2 Biotic stress and disease biology

For disease studies, resistant and susceptible genotypes can be compared to identify defence-associated expression programmes. Time-series transcriptomics can help separate early pathogen recognition from later defence responses. Pattern-recognition receptors, hormone signalling, ROS metabolism, cell-wall modification and antimicrobial responses are among the processes that can be examined.

Machine learning can further be used to classify infection status or resistance phenotype and to identify predictive gene signatures. When appropriate, host and pathogen transcriptomes can also be analysed together to study host-pathogen interactions.

7. Case Studies

7.1 Chickpea drought tolerance

The lecture uses chickpea as an important example of how transcriptomics can reveal genotype- and developmental-stage-specific responses. Contrasting genotypes show differences involving carbohydrate metabolism, photosynthesis, redox homeostasis, cell-wall processes and transcription factors. One of the most useful lessons from this example is that drought tolerance is not a single, fixed molecular state; the response can change with developmental stage.

An AI/ML extension would be to use expression signatures to classify tolerant and sensitive genotypes and then rank the genes that contribute most strongly to that classification. Those candidates should then be examined alongside physiological measurements, genetic evidence and experimental validation.

7.2 Wheat heat tolerance

The wheat case presented in the lecture illustrates the value of combining RNA-seq with WGCNA and functional interpretation. Heat-response modules were associated with heat response, protein folding and ROS-related processes, and genes such as TaBI-1, TaCPD-1 and TaHSF-1 were prioritized. The case shows how expression, network information, pathway enrichment and candidate-gene evidence can be brought together before experimental validation.

7.3 Groundnut heat resilience

The groundnut example compares heat-tolerant and heat-sensitive material across developmental stages. Antioxidant and phenolic responses are prominent around flowering, while changes in sucrose and glucose partitioning and starch accumulation are important during pod development. This illustrates why developmental stage and physiological context must be considered when interpreting a stress transcriptome.

7.4 Multi-omics perspective

Transcriptomics can be combined with proteomics and metabolomics to connect changes at the RNA, protein and metabolite levels. Such integration can provide stronger mechanistic evidence than any single omics layer. As the number of molecular features increases, AI/ML becomes increasingly useful for finding relationships across these data types.

8. Emerging Directions and Practical Best Practices

The field is moving towards systems-level analysis in which genomics, transcriptomics, proteomics, metabolomics, epigenomics, phenomics and physiological measurements are considered together. Single-cell and spatial transcriptomics are adding cellular and tissue-level resolution, while long-read and direct RNA sequencing are improving the characterization of full-length transcripts and isoforms.

At the same time, several practical cautions remain essential. Small sample sizes, data leakage, overfitting, batch effects, feature instability and poor generalizability can undermine an otherwise sophisticated model. A model trained in one crop, tissue, developmental stage or environment may not automatically work in another. Most importantly, computational association, prediction and network centrality do not establish biological function by themselves.

8.1 Recommended end-to-end framework

A practical workflow is:

1. Define the biological question and phenotype;
2. Design a properly replicated RNA-seq experiment;
3. Perform quality control and quantification;
4. Explore the expression matrix;
5. Conduct differential expression analysis;

6. Interpret genes through GO, pathways and gene sets;
7. Construct networks where appropriate;
8. Apply unsupervised and supervised ML;
9. Use feature selection and XAI;
10. Integrate QTL/GWAS/eQTL and other genetic evidence;
11. Add physiological and other omics information when justified;
12. Prioritize candidates supported by multiple independent evidence streams;
13. Finally validate the strongest hypotheses experimentally.

9. Conclusion

The ultimate purpose of transcriptomic analysis is not to produce the largest DEG list or the most accurate machine-learning model. The purpose is to understand how molecular changes contribute to a biological phenotype. AI/ML is especially valuable because it can help us discover patterns, predict phenotypes and identify informative genes within complex datasets. Its greatest value, however, comes when computational results are combined with functional biology, networks, genetics, physiology and experimental validation.

The central message for researchers and students is simple: use AI/ML to ask better biological questions, not merely to obtain better predictions. The strongest conclusion is reached when expression, pathway, network, AI/ML, genetic and experimental evidence converge on a mechanism that can be tested and, ultimately, used to improve biological understanding or crop performance.

References

Upton, R.N., Correr, F.H., Lile, J., Reynolds, G.L., Falaschi, K., Cook, J.P. & Lachowiec, J. (2023). Design, execution, and interpretation of plant RNA-seq analyses. Frontiers in Plant Science, 14, 1135455. doi:10.3389/fpls.2023.1135455.

Tyagi, S., et al. (2022). Upcoming progress of transcriptomics studies on plants: An overview. Frontiers in Plant Science.

Novakovsky, G., Dexter, N., Libbrecht, M.W., Wasserman, W.W. & Mostafavi, S. (2023). Obtaining genetics insights from deep learning via explainable artificial intelligence. Nature Reviews Genetics, 24, 125–137. doi:10.1038/s41576-022-00532-2.

Garg, R., Shankar, R., Thakkar, B., et al. (2016). Transcriptome analyses reveal genotype- and developmental stage-specific molecular responses to drought and salinity stresses in chickpea. Scientific Reports, 6, 19228. doi:10.1038/srep19228.

Kudapa, H., Ghatak, A., Barmukh, R., et al. (2024). Integrated multi-omics analysis reveals drought stress response mechanism in chickpea. The Plant Genome, 17, e20337. doi:10.1002/tpg2.20337.

Bagudam, R., Mathew, A., Kancherla, E., et al. (2025). Transcriptome analysis of genes and carbon partitioning pathways involved in high temperature stress resilience in groundnut. Scientific Reports, 15, 29939. doi:10.1038/s41598-025-15509-4.

Deciphering plant transcriptomes: Leveraging machine learning for deeper insights. Current Plant Biology (2024). doi:10.1016/j.cpb.2024.100432.

Arif, A. & Srivastava, P. (2026). Artificial Intelligence Models for Analysing Plant Transcriptomics. In Crop Improvement with Artificial Intelligence. Wiley. doi:10.1002/9781394330485.ch10.

Zaman, W. & Park, S. (2026). Systems perspectives on plant growth regulation: integrating omics, signaling networks, and computational models. Plant Growth Regulation.

Li, et al. (2026). Artificial Intelligence Revolution in Transcriptomics: From Single Cells to Spatial Atlases. Advanced Science. doi:10.1002/advs.202518949.

Probing cellular landscapes in plants via single-cell transcriptomics. Plant Science (2026), 364, 112964.

Uploaded lecture presentation: Extracting Functional and Biological Insights from Transcriptomic Data Using AI/ML. ICAR–Indian Agricultural Statistics Research Institute, Division of Agricultural Bioinformatics.

miRNA Identification and Target Prediction

Sarika Sahu and Namrata Sachdeva

MicroRNAs (miRNAs) are a class of small non-coding RNAs, typically 18–23 nucleotides (nt) long, that play essential roles in post-transcriptional gene regulation across plants and animals. In plants, canonical miRNAs are predominantly ~21 nt endogenous small RNAs that regulate gene expression through target mRNA cleavage and/or translational repression, thereby contributing to diverse biological processes, including growth and development, differentiation, metabolism, and responses to environmental stresses. Their evolutionary conservation across species and miRNA families further highlights their functional importance in gene regulatory networks. Dysregulation of miRNA expression has also been associated with various human diseases, including cancer, cardiovascular disorders, and neurological conditions, emphasizing their potential as diagnostic biomarkers and therapeutic targets.

High-throughput sequencing of endogenous small RNAs (sRNA-seq) provides a powerful approach for the discovery and annotation of miRNA-producing loci. However, computational tools for miRNA identification are often optimized for specific organisms, and relatively few are specifically designed for plant miRNAs. For example, miRkwood is a web-based tool developed for plant miRNA discovery that accommodates the structural diversity of plant pre-miRNAs and supports the identification of canonical and non-canonical miRNAs (Guigon et al., 2019). These limitations highlight the need for robust, plant-oriented computational approaches for systematic miRNA identification and characterization.

1. **Computational tools for miRNA prediction and characterization**

miRBase: A comprehensive database for miRNA sequences and annotations. It serves as a valuable resource for comparing and validating predicted miRNAs.
miRDeep: A widely-used tool for the prediction of novel miRNAs from small RNA sequencing data. It integrates secondary structure analysis, sequence conservation, and machine learning algorithms for accurate prediction.
miRanda: This tool predicts miRNA target sites by examining sequence complementarity between miRNAs and potential target mRNAs. It considers both sequence complementarity and conservation across species.
TargetScan: A popular tool for miRNA target prediction, TargetScan predict miRNA target sites based on seed sequence matches, site accessibility, and evolutionary conservation.

RNAhybrid: A tool for predicting the hybridization energy and the minimum free energy (MFE) of RNA-RNA duplexes, commonly used to predict potential miRNA-mRNA interactions.
PITA (Probability of Interaction by Target Accessibility): This tool predicts miRNA target sites based on thermodynamic stability and target site accessibility, offering a probabilistic framework for target prediction.
miRDeep2: An updated version of miRDeep, miRDeep2 integrates small RNA sequencing data with genomic information to predict both known and novel miRNAs with improved accuracy.
miRPlant: Specifically designed for plant miRNA prediction, miRPlant incorporates features such as sequence conservation, secondary structure, and thermodynamic stability to identify potential miRNA candidates in plant genomes.
ShortStack: This tool integrates multiple small RNA sequencing data sets to identify and characterize miRNAs, including novel miRNAs and their targets, with a focus on plant species.
psRNATarget: A plant-specific tool for predicting miRNA targets, psRNATarget considers various factors such as target site accessibility and conservation across species to provide accurate predictions.

2. **General workflow for miRNA prediction**

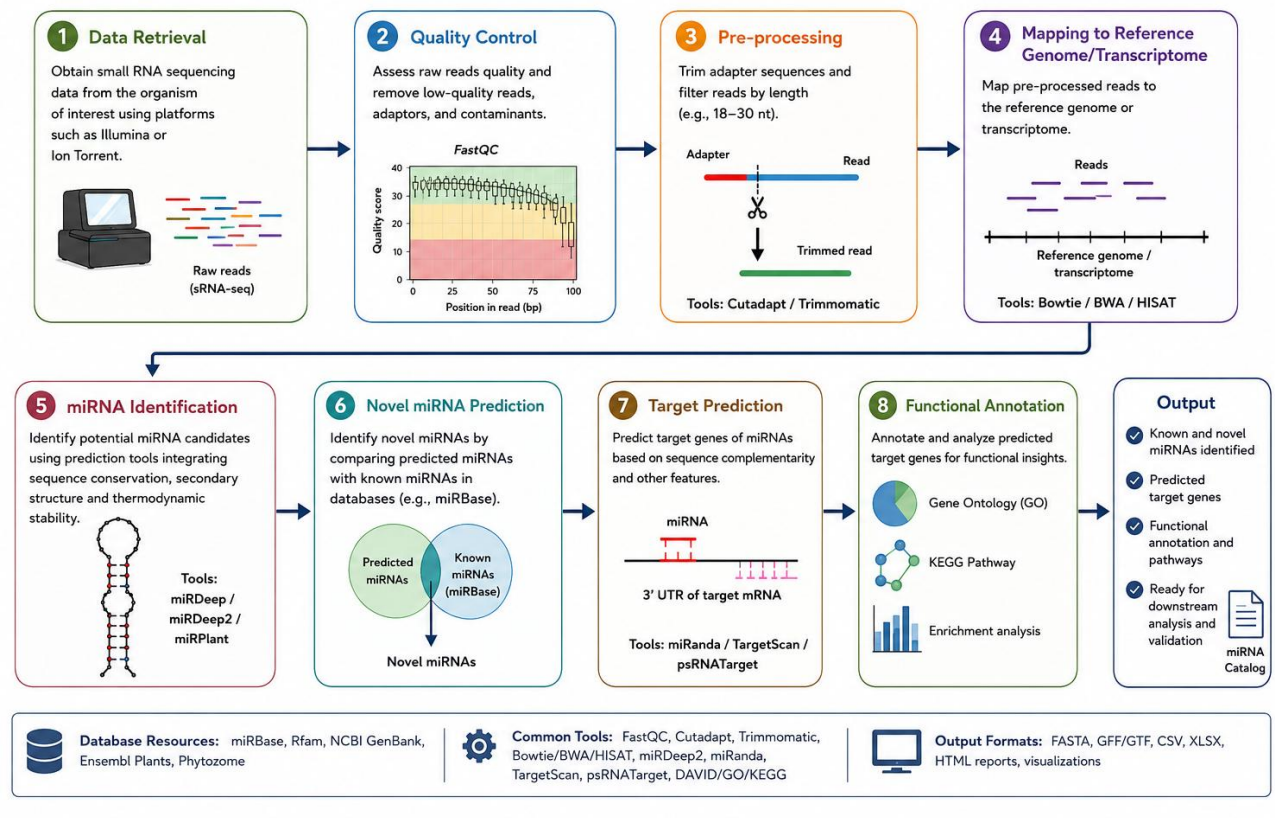


Figure 1: workflow of miRNA identification and characterization from transcriptome data

i. **Data retrieval:**

Obtain small RNA sequencing data from the organism of interest. This data can be generated from high-throughput sequencing platforms such as Illumina or Ion Torrent.

ii. Quality Control:

Perform quality control on the raw sequencing data to remove low-quality reads, adaptors, and contaminants. Tools like FastQC can be used for this purpose.

iii. Pre-processing:

Trim adapter sequences and filter out reads of inappropriate length. Tools such as Cutadapt or Trimmomatic can be used for this step.

iv. Mapping to Reference Genome:

Map the pre-processed reads to the reference genome or transcriptome using alignment tools like Bowtie, BWA, or HISAT.

v. miRNA Identification:

Use miRNA prediction tools such as miRDeep, miRDeep2, or miRPlant to identify potential miRNA candidates. These tools integrate various features such as sequence conservation, secondary structure, and thermodynamic stability to predict miRNAs.

vi. Novel miRNA Prediction:

Identify novel miRNAs by comparing predicted miRNAs with known miRNA sequences from databases like miRBase. Tools like miRDeep2 and ShortStack often include modules for predicting novel miRNAs.

vii. Target Prediction:

Predict miRNA target genes using tools like miRanda, TargetScan, or psRNATarget. These tools analyze sequence complementarity between miRNAs and potential target mRNAs, considering factors such as seed sequence matches, site accessibility, and evolutionary conservation.

viii. Functional Annotation:

Annotate predicted target genes to elucidate their biological functions and pathways. Tools such as DAVID, GO enrichment analysis, or KEGG pathway analysis can be used for functional annotation.

ix. Experimental Validation:

Experimentally validate predicted miRNAs and their targets using techniques such as qRT-PCR, luciferase reporter assays, or functional studies in cell lines or model organisms.

x. Integration and Visualization:

Integrate miRNA prediction results with other omics data (e.g., mRNA expression data, proteomics data) to gain a comprehensive understanding of miRNA-mediated regulatory networks. Visualization tools such as Cytoscape can be used to visualize miRNA-mRNA interaction networks.

xi. Validation and Interpretation:

Validate predicted miRNAs and their targets using experimental techniques. Interpret the results in the context of the biological system under study and generate hypotheses for further investigation.

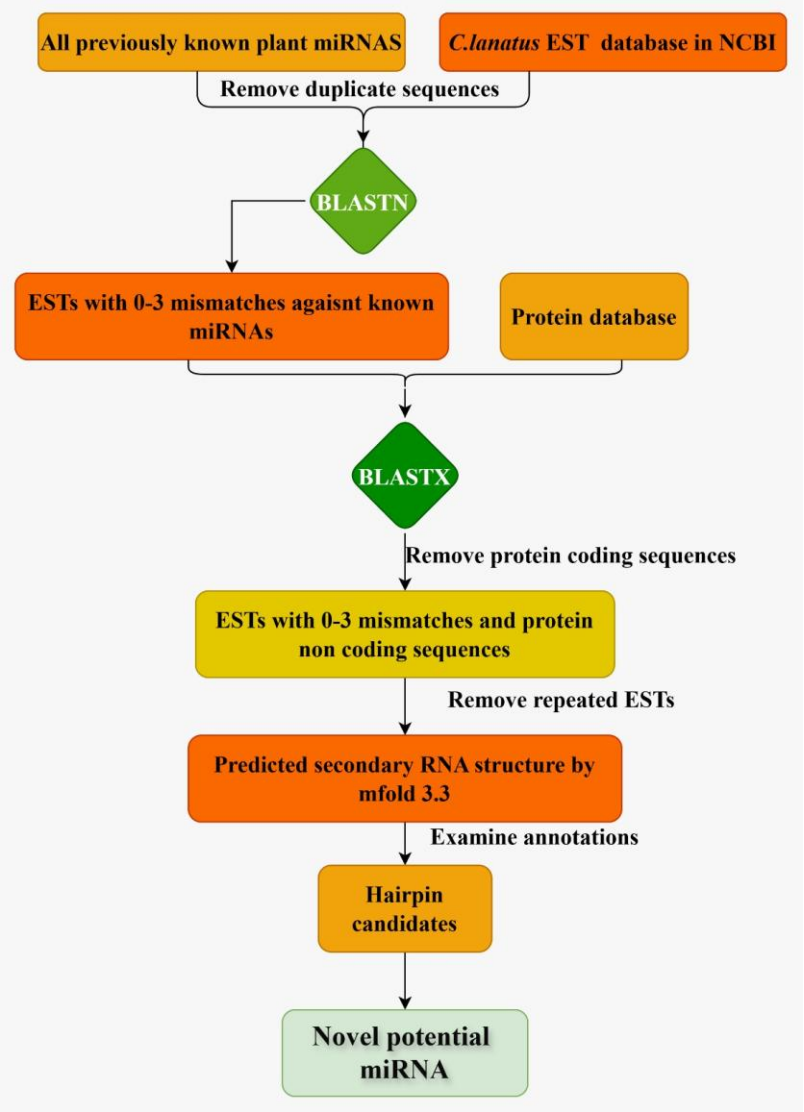


Figure 2: workflow of miRNA prediction from EST by identifying homologs of previously known miRNAs in plants (Zakeel et al., 2019)

MiRNA identification using comparative genomics approach

Identification of potential miRNAs

All previously known miRNA precursor sequences are downloaded from the miRBase database (Release 22.1; http://www.mirbase.org/ (2022). These precursor miRNAs are used as query sequences for BLASTN searches against the reference transcriptomes of species using default parameters and an E-value cut-off of 10. Only the best hit for each query sequence are retained and after elimination of redundant hits, these candidate primary miRNA sequences are scanned for hairpin-like secondary structures using the miRNA identification pipeline of the C-mii software. MirEval (http://mimirna.centenary.org.au/mireval/) was used to predict miRNA precursor sequences.

RNA sequences are considered as miRNA candidates only when they fulfilled the following criteria: (1) at least 18 nt length was assumed between the predicted and mature miRNAs and (2) 0–3 nt mismatches were allowed in sequence with all previously known plant mature miRNAs. A set of miRNA candidates were screened from ESTs that closely matched with the mature miRNAs which had hitherto been identified in plants. These miRNA candidates were then used for further screening to identify miRNA precursors by evaluating the miRNA precursor prediction properties using mirEval software. These precursor sequences are subjected to BLASTx analysis with protein database and the non-protein-coding sequences were retained for RNA secondary structure prediction.

Prediction of secondary structure and new miRNA

After the removal of protein coding sequences from the candidate miRNAs, the remaining precursor sequences of potential miRNA homologs are assessed for secondary structures using the Zuker folding algorithm by MFOLD software (Zuker, 2003). The following criteria are used in defining the RNA sequences as miRNA homologs: (1) The length of predicted mature miRNAs should be in the range of 19–25 nucleotides; (2) A maximum of two mismatches compared with known rice mature miRNAs should be allowed; (3) The mature miRNA should be localized in only one arm within the predicted stem–loop structure; (4) No more than five mismatches should be allowed between miRNA sequence and guide miRNA sequence in the stem–loop structure; (5) miRNAs should have high A + U content (30–70%); and (6) the predicted secondary structure should have higher minimal folding free energy index (MFEI) and negative minimal folding free energy (MFE) (Wang et al., 2011). The MFEI was calculated using the following equation:

$$\text{MFEI} = [(\text{MFE}/\text{length of the RNA sequence}) \times 100]/(\text{G} + \text{C})$$

%MFE denotes the negative folding free energy (ΔG)

The resulted precursor sequences are subjected to BLASTn with mRNA database to obtain new miRNA sequences. These novel miRNAs are named according to the miRNA nomenclature criteria (Griffiths-Jones et al., 2006). Novel mature miRNA sequences are highlighted in the secondary structure by small RNA workbench software.

Identification of miRNAs Targets

As previous studies have suggested that most miRNAs can bind to protein coding regions of the target mRNAs at a perfect or near-perfect complementation and interfere or degrade mRNA (Bartel, 2004; Chen, 2004), a simple homology search against EST and nucleotide

databases of *query species is performed* with the following criteria to predict targets of the potential cla-miRNAs: (1) the maximum number of mismatched nucleotides between the mature miRNA and its potential target genes was four; (2) the maximum number of mismatched nucleotides at positions 1–9 was one; (3) no mismatches were allowed at positions 10–11; (4) no more than two continuous mismatches at any position were allowed (Xie et al., 2010).

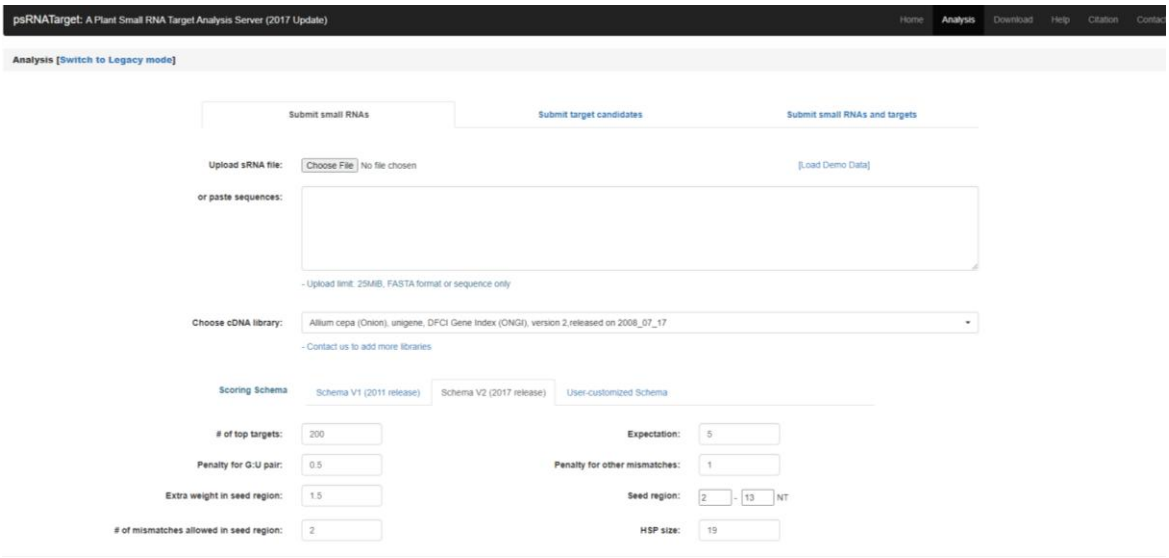


Figure 3: Home page of miRNA target of psRNATarget server

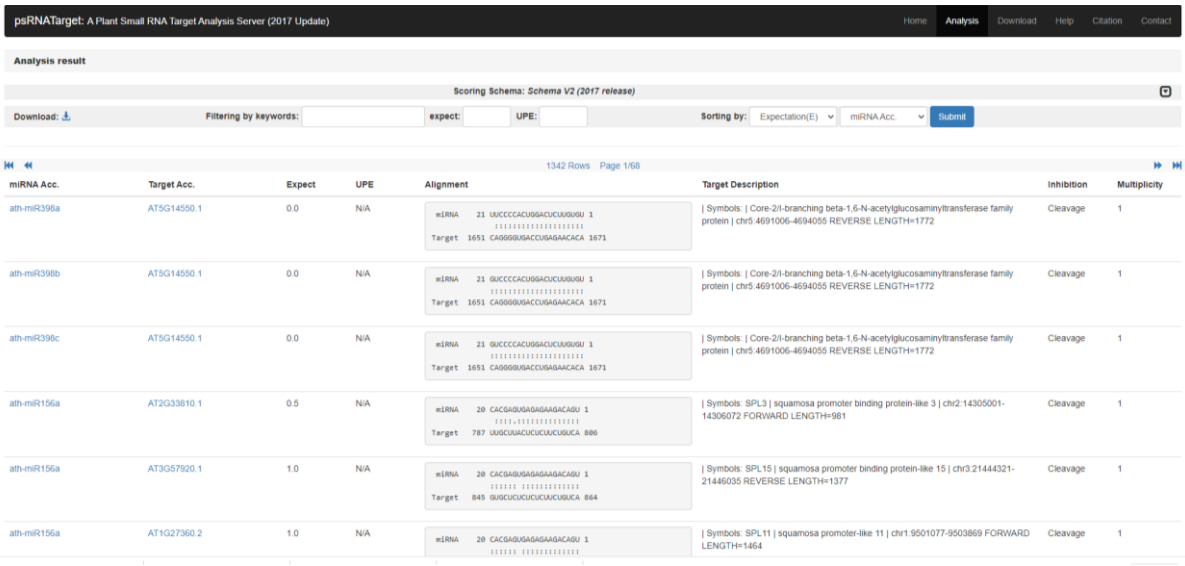


Figure 4: Result of the psRNATarget server

Tools commonly used for integrating omics data with miRNA data:

Tool/Database	Integration / Data Sources	Key Functions
miRWalk	miRNA and mRNA expression data	Predicts potential miRNA–target interactions and performs functional enrichment analysis.
miRGator	miRNA expression, mRNA expression, PPI networks, and pathway information	Visualizes miRNA–mRNA regulatory networks and identifies key regulatory modules.
DIANA-miRPath	miRNA expression, GO terms, and KEGG pathways	Predicts functional impacts of miRNA dysregulation and identifies enriched biological processes and pathways.
miEAA	miRNA expression and functional annotation databases (GO, KEGG)	Identifies miRNA-regulated biological processes and pathways and prioritizes functionally relevant miRNAs.
TarBase	Experimentally validated miRNA–target interactions and other omics datasets	Provides curated miRNA–target interactions based on experimental evidence and supports network analysis.
miRNA Target Filter	miRNA expression, TargetScan, miRanda, expression correlation, and prediction scores	Prioritizes high-confidence miRNA–target interactions.
miRNet	miRNA expression, PPI networks, TF–target interactions, and pathway databases	Constructs and visualizes miRNA-mediated regulatory networks and identifies key regulatory nodes.
miRNAtap	miRNA expression, gene expression, PPI networks, and pathway information	Identifies dysregulated miRNA–target interactions associated with biological processes or diseases.

These tools facilitate the integration of miRNA data with other omics data sets, enabling comprehensive analysis of miRNA-mediated regulatory networks and their functional implications.

Reference

1. Bartel, D. P. (2004). MicroRNAs: genomics, biogenesis, mechanism, and function. Cell, 116(2), 281-297. [DOI: 10.1016/S0092-8674(04)00045-5]
2. Ha, M., & Kim, V. N. (2014). Regulation of microRNA biogenesis. Nature Reviews Molecular Cell Biology, 15(8), 509-524. [DOI: 10.1038/nrm3838]
3. Jonas, S., & Izaurralde, E. (2015). Towards a molecular understanding of microRNA-mediated gene silencing. Nature Reviews Genetics, 16(7), 421-433. [DOI: 10.1038/nrg3965]
4. Mendell, J. T., & Olson, E. N. (2012). MicroRNAs in stress signaling and human disease. Cell, 148(6), 1172-1187. [DOI: 10.1016/j.cell.2012.02.005]
5. Lewis, B. P., Burge, C. B., & Bartel, D. P. (2005). Conserved seed pairing, often flanked by adenosines, indicates that thousands of human genes are microRNA targets. Cell, 120(1), 15-20. [DOI: 10.1016/j.cell.2004.12.035]
6. Bartel, D. P. (2009). MicroRNAs: target recognition and regulatory functions. Cell, 136(2), 215-233. [DOI: 10.1016/j.cell.2009.01.002]
7. Friedman, R. C., Farh, K. K., Burge, C. B., & Bartel, D. P. (2009). Most mammalian mRNAs are conserved targets of microRNAs. Genome Research, 19(1), 92-105. [DOI: 10.1101/gr.082701.108]

@ Disclaimer

The information contained in this reference manual has been taken from various web resources. The information is provided by "ICAR-IASRI" and whilst we endeavor to keep the information up-to-date and correct, we make no representations or warranties of any kind, express or implied, about the completeness, accuracy, reliability, suitability or availability with respect to the website or the information, products, services, or related graphics contained in the reference manual for any purpose. Any reliance you place on such information is therefore strictly at your own risk.

In no event will we be liable for any loss or damage including without limitation, indirect or consequential loss or damage, or any loss or damage whatsoever arising from loss of data or profits arise out of or in connection with the use of this manual. We have no control over the nature, content and availability of those sites. The inclusion of any links does not necessarily imply a recommendation or endorse the views expressed within them.

@ Citation

Md. Samir Farooqi, Sudhir Srivastava, Sharanbasappa and Girish Kumar Jha (2026). Statistical and AI/ML Approaches for Transcriptomic Data Analysis, Online Training Programme under the Project **DBT- Establishment of Centre for Bioinformatics and Computational Biology in Agriculture-BIC at ICAR-Indian Agricultural Statistics Research Institute,** E-Manual, ICAR-Indian Agricultural Statistics Research Institute, New Delhi.