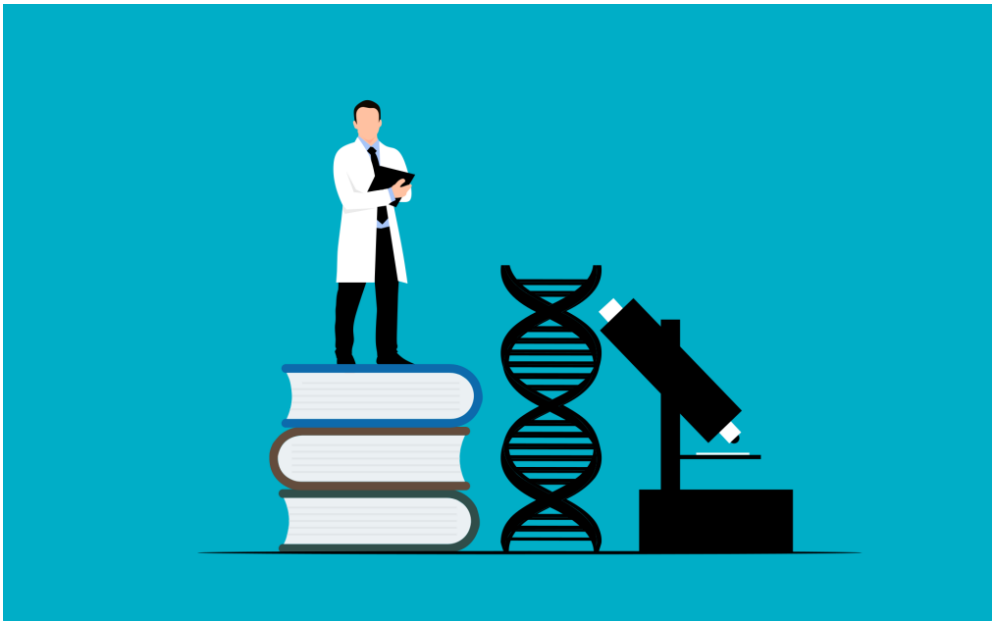
Pune Vidyarthi Griha's College of Engineering and Technology & G.K. Pate (Wani) Institute of Management, Pune
Approved by AICTE, DTE (CODE: 6274) Affiliated to SPPU, Pune |

Artificial Intelligence and Data Science Department

Bioinformatics Lab Manual



FOR THE PROGRAMME

B. E. (ARTIFICIAL INTELLIGENCE AND DATA SCIENCE)-2020 Course

Institute's Vision Statement

"To achieve excellence in engineering education"

Institute's Mission Statement

- To satisfy all stakeholders
- To develop ethical, highly motivated engineering professionals with good human values, requisite skills and competencies.
- To adopt innovative teaching mechanisms. To promote research culture
- To contribute to the country's economic developments.
- To be responsive to changes In technology, socio-economic and environmental conditions

Artificial Intelligence and Data Science Department

Vision

"To become a center of excellence for education in Artificial Intelligence and Data Science with a holistic development approach."

Mission

- To create a research based educational ecosystem by adopting interdisciplinary and multidisciplinary approach
- To impart fundamental and advanced engineering knowledge to provide sustainable solutions to the industry and other stakeholders
- To nurture social and ethical values in students for the progress of society and nation

Program Educational Objectives (PEOs)

Graduate of the program will

PEO1 Apply problem solving skills to address the societal and environmental issues

PEO2 Practice Ethical AI and become Responsible Data Scientist

PEO3 Embrace lifelong learning to accommodate fast paced technical world of Artificial Intelligence

PEO4 Exhibit interpersonal and intra-personal skills to become a leader in the domain of AI&DS

PROGRAM OUTCOMES (POs)

Engineering graduates will possess:

PO1: Engineering Knowledge

Apply the knowledge of mathematics, science, Engineering fundamentals, and an Engineering specialization to the solution of complex Engineering problems.

PO2: Problem Analysis

Identify, formulate, review research literature, and analyze complex Engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and Engineering sciences.

PO3: Design/Development of Solutions

Design solutions for complex Engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and Environmental considerations.

PO4: Conduct Investigations of Complex problems

Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5: Modern Tool Usage

Create, select, and apply appropriate techniques, resources, and modern Engineering and IT tools including prediction and modeling to complex Engineering activities with an understanding of the limitations.

PO6: The Engineer and Society

Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7: Environment and Sustainability

Understand the impact of the professional Engineering solutions in societal and Environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8: Ethics

Apply ethical principles and commit to professional ethics and responsibilities and norms of the Engineering practices.

PO9: Individual and Team Work

Function effectively as an individual, and as a member or leader in diverse teams, and multidisciplinary settings.

PO10: Communication

Communicate effectively on complex Engineering activities with the Engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions

PO11: Project Management and Finance

Demonstrate knowledge and understanding of the Engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary Environments.

PO12: Life-long Learning

Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change

PROGRAM SPECIFIC OUTCOMES (PSOs)

Program Specific Outcomes (PSO) AI&DS Engineering graduate will

- Demonstrate proficiency in collecting, cleaning, and analyzing diverse datasets using state-of the-art tools and techniques
- Exhibit the machine learning expertise and deep learning competency to design and train neural networks for various applications
- Adhere to professional standards and ethical principles in their AI and data science work, respecting data privacy, confidentiality, and intellectual property rights

List of Experiments

1. DNA Sequence Analysis
2. RNA-Seq Data Analysis
3. Protein Structure Prediction
4. Machine Learning for Genomic Data

List of Mini Project

1. Molecular Docking

Curriculum-Practical

Savitribai Phule Pune University Fourth Year of Artificial Intelligence and Data Science (2020 Course) 417526: Computer Laboratory II: Bioinformatics		
Teaching Scheme: **PR: 04 Hours/Week**	**Credit** **02**	**Examination Scheme and Marks** **Term Work (TW): 50 Marks** **Practical (PR): 25 Marks**
Prerequisites Courses: Statistics (217528), Artificial Intelligence (310253), Data Science (317529)		
Companion Course: Elective III: Bioinformatics (417523(D))		
Course Objectives: • To refer appropriate, suitable datasets • To study appropriate Bioinformatics tools		
Course Outcomes: On completion of the course, learners will be able to **CO1:** Use suitable datasets for various problems **CO2:** Demonstrate and apply appropriate Bioinformatics tools		
Instructions: 1. Practical can be performed on suitable development platform. 2. Perform total 4 experiments (Group A) and one mini-project (Group B)		
List of Assignments		
1. Assignment: DNA Sequence Analysis. Task: Analyze a given DNA sequence and perform basic sequence manipulation, including finding motifs, calculating GC content, and identifying coding regions. Deliverable: A report summarizing the analysis results and any insights gained from the sequence.		
2. Assignment: RNA-Seq Data Analysis. Task: Analyze a provided RNA-Seq dataset and perform differential gene expression analysis. Deliverable: A detailed report presenting the differentially expressed genes, their functional annotations, and any potential biological interpretations		
3. Assignment: Protein Structure Prediction. Task: Predict the 3D structure of a given protein sequence using homology modeling or threading techniques. Deliverable: A report presenting the predicted protein structure, along with an analysis of its potential functions and interactions.		
4. Assignment: Molecular Docking and Virtual Screening. Task: Perform molecular docking simulations to predict the binding affinity between a protein target and a small molecule ligand. Additionally, conduct virtual screening to identify potential drug candidates. Deliverable: A report summarizing the docking results, including the binding poses and potential lead compounds.		
5. Assignment: Machine Learning for Genomic Data. Task: Apply machine learning algorithms, such as random forests or support vector machines, to classify genomic data based on specific features or markers. Deliverable: A comprehensive analysis report presenting the classification results, model performance evaluation, and insights into the predictive features.		

CO-PO Matrix

The CO-PO Mapping Matrix												
CO/ PO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1	3	2	1	3	3	2	-	-	-	-	1	-
CO2	3	2	1	3	3	1	-	-	-	-	1	-

Assessment Criterion

Sr. No.	Criteria	Marks
1	Attendance	5
2	Coding Efficiency	5
3	Timely Submission	5
4	Oral	5
5	Mock Practical	5
Total		25

Assignment 1

Title: DNA Sequence Analysis

Problem Statement: **Task**: Analyze a given DNA sequence and perform basic sequence manipulation, including finding motifs, calculating GC content, and identifying coding regions. **Deliverable**: A report summarizing the analysis results and any insights gained from the sequence.

Dataset: The DNA sequence used was downloaded from NCBI.

- Homo sapiens chromosome 19
- NCBI Reference Sequence: NC_000019.10
- Format: FASTA

Description: The goal of this assignment is to analyze a DNA sequence from Homo sapiens chromosome 19 (NC_000019.10) in FASTA format by searching for motifs, calculating GC content, and identifying coding regions (ORFs). This analysis helps in understanding the sequence's structure, coding potential, and biological significance.

Tasks to Perform:

1. **Obtain and Load the DNA Sequence:**

- Download the Homo sapiens chromosome 19 sequence in **FASTA format** from the **NCBI Reference Sequence Database**.
- Read the sequence from the FASTA file using Python code, ignoring the header and processing only the nucleotide sequence.

2. **Analyze the DNA Sequence:**

- **Find Coding Regions (Open Reading Frames)**:
  - Search for **start codon (ATG)** and **stop codons (TAA, TAG, TGA)**.
  - Extract all valid **coding sequences** (ORFs) with lengths divisible by 3.
- **Calculate GC Content**:
  - Compute the percentage of **guanine (G)** and **cytosine (C)** in the DNA sequence to assess stability and coding potential.

- **Search for Motifs**:
    - Identify important **motif patterns**, such as recurring elements or regulatory sequences, if present.

3. **Summarize the Findings:**

- Record all **identified ORFs** and their lengths.
- Report the **overall GC content** and **any motifs** found in the DNA sequence.
- Provide insights about the **potential coding regions** and **functional patterns** in the analyzed DNA sequence.

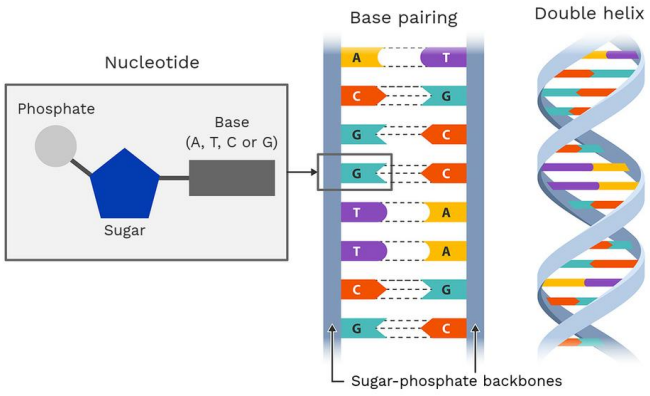
Prerequisite:

1. Understanding of DNA sequences (A, T, C, G).
2. Knowledge of codons and open reading frames (ORFs).
3. Basics of FASTA format handling.
4. Familiarity with Python programming and string manipulation.

Theory:

DNA (Deoxyribonucleic Acid) is the molecule that carries the genetic instructions for the development, functioning, and reproduction of all living organisms. It consists of two long strands that form a double helix structure, where each strand is made up of smaller units called nucleotides.

Structure of DNA



1. Nucleotides: Each nucleotide is composed of three components:

- Phosphate group

- Deoxyribose sugar (5-carbon sugar)
- Nitrogenous base: One of four bases –
    - Adenine (A)
    - Thymine (T)
    - Cytosine (C)
    - Guanine (G)

2. Base Pairing Rules: DNA is double-stranded.

- Adenine (A) pairs with Thymine (T) via 2 hydrogen bonds.
- Cytosine (C) pairs with Guanine (G) via 3 hydrogen bonds.

3. Antiparallel Orientation:

- The two strands run in opposite directions: One from 5′ to 3′ and the other from 3′ to 5′.
- This directionality is essential for replication and transcription.

Genetic Code and Codons

The DNA sequence contains instructions for building proteins, which are vital for cellular functions. These instructions are organized into triplets of nucleotides called codons. Each codon corresponds to a specific amino acid or a stop signal during protein synthesis.

Start Codon: ATG encodes the amino acid methionine and signals the start of translation.

Stop Codons: TAA, TAG, and TGA do not encode any amino acid and signal the end of translation.

Open Reading Frames (ORFs)

An ORF is a sequence of DNA that contains:

1. A start codon (ATG).
2. A continuous stretch of codons that do not encounter a stop codon.
3. Ends with one of the stop codons (TAA, TAG, or TGA).

ORFs are important because they represent potential protein-coding regions.

Identifying ORFs helps predict genes in a sequence.

GC Content in DNA

GC content refers to the percentage of guanine (G) and cytosine (C) bases in a DNA sequence.

- Importance:

Regions with higher GC content are usually more stable because G-C pairs form three hydrogen bonds, compared to two in A-T pairs.

GC content is used to predict genomic stability and coding potential.

- Formula:

$$GC\ Content\ (\%) = \frac{(G+C)}{Total\ Bases} x\ 100$$

Motifs in DNA

A motif is a short, recurring pattern in DNA that has a biological function. Motifs often appear in:

1. Promoters (regions where RNA polymerase binds for transcription)
2. Regulatory regions controlling gene expression
3. Splice sites within genes

Example: The TATA box is a common promoter motif found in many eukaryotic genes.

FASTA Format

The FASTA format is a widely used text-based format for storing biological sequences. It consists of:

1. Header line: Starts with > followed by the sequence name or description.
2. Sequence lines: Contain the nucleotide sequence in standard letters (A, T, C, G).

Example:

>NC_000019.10 Homo sapiens chromosome 19

ATGGCGGCTGAATGCTTAG...

Code:

```
import re
sequence = ""
with open("sequence.fasta") as f:
    lines = f.readlines()
    sequence = ''.join(line.strip() for line in lines[1:]).upper()
g_count = sequence.count('G')
c_count = sequence.count('C')
total_count = len(sequence)
gc_percent = ((g_count + c_count) / total_count) * 100
print(f"G Count: {g_count}")
print(f"C Count: {c_count}")
print(f"Total Count: {total_count}")
print(f"GC Percent: {gc_percent:.2f}%")

a_count = sequence.count('A')
t_count = sequence.count('T')
at_percent = ((a_count + t_count) / total_count) * 100
print(f"A Count: {a_count}")
print(f"T Count: {t_count}")
print(f"AT Percent: {at_percent:.2f}%")

ratio = (a_count + t_count) / (g_count + c_count)
print(f"AT/GC Ratio: {ratio:.2f}")
start_codon = 'ATG'
stop_codons = ['TAA', 'TAG', 'TGA']
coding_regions = []
start_index = sequence.find(start_codon)
```

```
print(start_index)

while start_index != -1:
    for stop_codon in stop_codons:
        stop_index = sequence.find(stop_codon, start_index + 3)
        if stop_index != -1 and (stop_index - start_index) % 3 == 0:
            coding_region = sequence[start_index:stop_index + 3]
            coding_regions.append(coding_region)
            break
    start_index = sequence.find(start_codon, start_index + 1)
if coding_regions:
    print("Coding Regions Found")
    for i, coding_region in enumerate(coding_regions, start=1):
        print(f"\nRegion {i}: {coding_region}\nLength:
{len(coding_region)}")
else:
    print("No coding regions found")

def find_motifs(sequence, motif):
    print(f"\nSearching for motif: {motif}")
    matches = [match.start() for match in re.finditer(motif,
sequence)]
    if matches:
        print(f"Motif '{motif}' found at positions: {matches}")
    else:
        print(f"Motif '{motif}' not found")

motif = "TATAA"
find_motifs(sequence, motif)
```

Output:

```
PS D:\College\Bioinformatics\Practicals> & C:/Users/Admin/AppData/Local/Programs/Python/
Python312/python.exe d:/College/Bioinformatics/Practicals/dna_sequence.py
G Count: 1130
C Count: 1067
Total Count: 3598
GC Percent: 61.06%
A Count: 667
T Count: 734
AT Percent: 38.94%
AT/GC Ratio: 0.64
73
Coding Regions Found

Region 1: ATGAGCTCAGGGGCCTCTAGAAAGAGCTGGGACCCTGGGAACCCCTGGCCTCCAGGTAGTCTCAGGAGAGCTACTCGG
GGTCGGGCTTGGGGAGAGGAGGAGCGGGGGTGAGGCAAGCAGCAGGGGACTGGACCTGGGAAGGGCTGGGCAGCAGAGACGACCCGAC
CCGCTAGAAGGTGGGGTGGGGAGAGCAGCTGGACTGGGATGTAA
Length: 210
AGCCCCTGGTGGAAGACATGCAGCGCCAGTGGGCCGGGCTGGTGGAGAAGGTGCAGGCTGCCGTGGGCACCAGCGCCGCCCCTGTGCC
CAGCGACAATCACTGAACGCCGAAGCCTGCAGCCATGCGACCCCACGCCACCCCGTGCCTCCTGCCTCCGCGCAGCCTGCAGCGGGAG
ACCCTGTCCCCGCCCCAGCCGTCCTCCTGGGGTGGACCCTAG
Length: 354

Region 31: ATGCAGCGCCAGTGGGCCGGGCTGGTGGAGAAGGTGCAGGCTGCCGTGGGCACCAGCGCCGCCCCTGTGCCCAGCGA
CAATCACTGAACGCCGAAGCCTGCAGCCATGCGACCCCACGCCACCCCGTGCCTCCTGCCTCCGCGCAGCCTGCAGCGGGAGACCCTG
TCCCCGCCCCAGCCGTCCTCCTGGGGTGGACCCTAG
Length: 201

Region 32: ATGCGACCCCACGCCACCCCGTGCCTCCTGCCTCCGCGCAGCCTGCAGCGGGAGACCCTGTCCCCGCCCCAGCCGTC
CTCCTGGGGTGGACCCTAG
Length: 96

Searching for motif: TATAA
Motif 'TATAA' found at positions: [1799]
```

Conclusion:

This assignment involved analyzing the Homo sapiens chromosome 19 dataset by reading a FASTA file, identifying coding regions, calculating GC content, and searching for motifs. These tasks provide insights into genetic structure, gene prediction, and sequence stability.

Assignment 2

Title: RNA-Seq Data Analysis.

Problem Statement: **Task**: Analyze a provided RNA-Seq dataset and perform differential gene expression analysis. **Deliverable**: A detailed report presenting the differentially expressed genes, their functional annotations, and any potential biological interpretations

Dataset: Gene Count Matrix (pasilla_gene_counts.tsv) and Sample Metadata (pasilla_sample_annotation.csv). The **Pasilla RNA-Seq dataset** is a widely used example dataset in bioinformatics education and practice. It originates from a study on **Drosophila melanogaster (fruit fly)** and focuses on the effect of **Pasilla (ps)**, a splicing factor gene, on alternative splicing and gene expression.

Description: The goal of this experiment is to analyze RNA-Seq data and identify differentially expressed genes between conditions using the DESeq2 package in R. RNA-Seq is a method for studying gene expression by sequencing RNA molecules. We analyze count data and metadata from the pasilla dataset to create a DESeq2 object for differential expression analysis. The results, including log2 fold change and p-values, highlight genes with significant expression changes, providing insights into gene regulation and affected biological pathways.

Tasks to Perform:

1. **Install and Load Required Packages**

- Install the necessary **Bioconductor packages** such as Biomanager, DESeq2 and pasilla.

2. **Load the RNA-Seq Data**

- Read the **count matrix** and **metadata file** using read.csv().
- Ensure the row names of metadata and column names of the count matrix match for correct alignment.

3. **Create a DESeqDataSet Object**

- Construct a **DESeqDataSet object** using the DESeqDataSetFromMatrix() function.
- Use the experimental **design formula** to specify the condition being tested.

4. **Perform Differential Gene Expression Analysis**

- Run the **DESeq()** function to perform the statistical analysis.
- Use the results() function to extract the differentially expressed genes.

5. Annotate the Results

- Classify genes as **upregulated, downregulated, or not significant** based on **log2 fold change** and **p-values**.

6. Generate a Volcano Plot

- Visualize the results using a **volcano plot** created with the ggplot2 library to display upregulated, downregulated, and non-significant genes.

Prerequisite:

1. **RNA Sequences (A, U, C, G)**: Basics of RNA bases, with uracil replacing thymine.
2. **Gene Expression**: Understanding mRNA's role in protein synthesis.
3. **Differential Gene Expression (DGE)**: Knowledge of varying gene expression across conditions.
4. **FASTQ/FASTA Formats**: Familiarity with RNA-Seq data storage.
5. **Statistical Models**: Basics of fold change, p-values, and significance.
6. **R Programming**: Experience with package installation, data handling, and visualization.

Theory:

What is RNA?

RNA (Ribonucleic Acid) is a **nucleic acid molecule** essential for gene expression and protein synthesis. It plays a central role in **transferring genetic information** from DNA to proteins. Unlike DNA, which is double-stranded, RNA is **single-stranded** and contains the nitrogenous base **uracil (U)** instead of **thymine (T)** found in DNA. It also contains **ribose** sugar instead of deoxyribose, making RNA structurally less stable than DNA.

Types of RNA

- **mRNA (Messenger RNA)**: Carries the genetic code from DNA to ribosomes for

protein synthesis.

- **tRNA (Transfer RNA)**: Brings amino acids to the ribosome during protein synthesis.
- **rRNA (Ribosomal RNA)**: Forms the structural and functional core of ribosomes.
- **Non-coding RNAs** (e.g., microRNA, lncRNA): Regulate gene expression without being translated into proteins.

What is RNA-Seq?

RNA sequencing (RNA-Seq) is a powerful technology used to study the transcriptome—the complete set of RNA molecules expressed by a cell or tissue at a given time. It provides insights into gene expression levels, alternative splicing, non-coding RNAs, and more. RNA-Seq allows researchers to measure how active certain genes are under various conditions, such as healthy versus diseased states.

Steps in RNA-Seq Data Analysis

RNA-Seq data analysis typically involves several key steps:

1. **Preprocessing of Raw Reads**
   Raw RNA-Seq data obtained from a sequencing platform (like Illumina) requires preprocessing to remove **low-quality reads** and **adapter sequences**. Tools like **FASTQC** (for quality control) and **Trimmomatic** (for trimming low-quality reads) are used at this stage.
2. **Alignment to a Reference Genome**
   After cleaning the reads, the next step is to **align them to a reference genome** (e.g., Homo sapiens genome) using tools such as **HISAT2** or **STAR**. The aligned reads are stored in **BAM files**, which represent the positions where the RNA sequences map onto the genome.
3. **Quantification of Gene Expression**
   The aligned reads are then used to **quantify expression levels** for each gene. Tools like **featureCounts** or **HTSeq** can count the number of reads aligned to each gene,

producing a **count matrix**.

4. **Differential Gene Expression Analysis**

   The goal of differential gene expression (DGE) analysis is to identify **genes that are expressed at different levels** between two or more conditions (e.g., treated vs. untreated). This step is often performed using **DESeq2**, which normalizes the data and applies statistical methods to detect significant changes in gene expression.

   Wald Test for Differential Expression

- The **Wald test** is a statistical test used in DESeq2 to determine whether the expression of a gene significantly differs between two conditions (e.g., treated vs. control).
- It calculates the **log2 fold change** between conditions and tests the null hypothesis (no difference).
- **p-values** from the Wald test indicate the significance of observed changes, which are further adjusted to **padj** values to control for multiple testing errors.

Volcano Plot

- A **volcano plot** is a scatter plot that displays the relationship between the **log2 fold change** (effect size) on the x-axis and **-log10(p-value)** (significance) on the y-axis.
- It helps identify **upregulated and downregulated genes**:
  - **Upregulated genes**: Genes with higher expression in the test condition (positive log2 fold change).
  - **Downregulated genes**: Genes with lower expression in the test condition (negative log2 fold change).
- In the plot, dashed lines mark thresholds for significance and fold change.

Upregulated and Downregulated Genes

- **Upregulated genes**: Identified by a **log2 fold change > 0.1** and **p-value < 0.05**.
- **Downregulated genes**: Identified by a **log2 fold change < -0.1** and **p-value < 0.05**.

- The identified genes are saved to separate text files (e.g., upregulated_genes.txt, downregulated_genes.txt) for further analysis.

Annotation Using DAVID

- **DAVID** (Database for Annotation, Visualization, and Integrated Discovery) provides functional annotation tools to interpret the biological meaning behind significant genes.
- Exported gene lists (from differential expression) are uploaded to DAVID to obtain **biological processes** and **molecular functions**.
- Results include key processes (like metabolic pathways) and molecular interactions associated with the significant genes.

Concepts in Differential Gene Expression Analysis

1. **Normalization:** RNA-Seq data is normalized to account for differences in sequencing depth across samples.
2. **Log2 Fold Change (LFC):** Measures the change in gene expression between two conditions.
3. **p-value and Adjusted p-value (padj):** Used to determine the statistical significance of the observed changes. Adjusted p-values control for **false discovery rate (FDR)**.

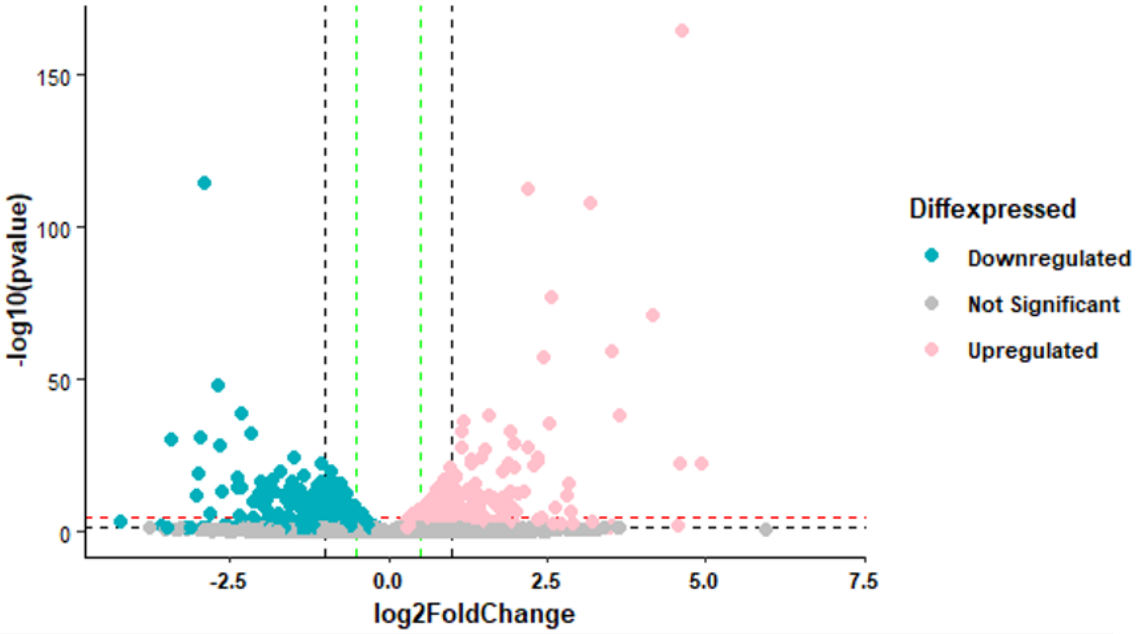
Code:

```
if (!require("BiocManager", quietly = TRUE))
  install.packages("BiocManager")

BiocManager::install("DESeq2")

BiocManager::install("pasilla")

library(BiocManager)
library(DESeq2)
library(pasilla)
```

```
# Input for count data
pasCts <- system.file("extdata",
                      "pasilla_gene_counts.tsv",
                      package="pasilla", mustWork=TRUE)

# Input for metadata
pasAnno <- system.file("extdata",
                       "pasilla_sample_annotation.csv",
                       package="pasilla", mustWork=TRUE)

# Count Matrix
count_csv = read.csv(pasCts,sep="\t",row.names="gene_id")
cts <- as.matrix(count_csv)
coldata <- read.csv(pasAnno, row.names=1)
coldata$condition <- factor(coldata$condition)

rownames(coldata) <- sub("fb", "", rownames(coldata))

# Check if data same
all(rownames(coldata) %in% colnames(cts))

# Check if in same order
all(rownames(coldata) == colnames(cts))

# To make in same order
cts <- cts[, rownames(coldata)]
all(rownames(coldata) == colnames(cts))

# creating dds object for analysis
dds <- DESeqDataSetFromMatrix(countData = cts,
                              colData = coldata,
                              design = ~ condition)

# Differential Expression Analysis
dds <- DESeq(dds)

res <- results(dds)

write.csv(res, file="DESeq_Analysis.csv")

df <- read.csv("./DESeq_Analysis.csv", header=TRUE)

df$Diffexpressed <- "NO"

df$Diffexpressed[df$log2FoldChange > 0.1 & df$pvalue < 0.05] <- "UP"
df$Diffexpressed[df$log2FoldChange < -0.1 & df$pvalue < 0.05] <- "DOWN"
```

```
ggplot(data = df, aes(x = log2FoldChange, y = -log10(pvalue), col =
Diffexpressed)) +
  geom_vline(xintercept = c(-1, 1), col = "black", linetype = 'dashed')
+
  geom_vline(xintercept = c(-0.5, 0.5), col ="green", linetype =
"dashed") +
  geom_hline(yintercept = -log10(0.00003), col = "red", linetype =
"dashed") +
  geom_hline(yintercept = -log10(0.05), col = "black", linetype =
"dashed") +
  geom_point(size = 2) +
  theme(panel.grid.major = element_blank(), panel.grid.minor =
element_blank(),
        panel.background = element_blank(), axis.line =
element_line(colour = "black"),
        axis.title.y = element_text(size = 10, colour = "black", face =
"bold"),
        axis.title.x = element_text(size = 10, colour = "black", face =
"bold"),
        axis.text.x = element_text(size = 8, colour = "black", face =
"bold"),
        axis.text.y = element_text(size = 8, colour = "black"),
        legend.title = element_text(size = 10, colour = "black",
face="bold"),
        legend.text = element_text(size = 8, colour="black",
face="bold"))+
  scale_color_manual(values = c("#00AFBB", "grey", "pink"),
                     labels = c("Downregulated", "Not Significant",
"Upregulated"))

upregulated_genes <- rownames(df[df$Diffexpressed == "UP", ])
downregulated_genes <- rownames(df[df$Diffexpressed == "DOWN", ])

# Export upregulated genes to a text file
write(upregulated_genes, file = "upregulated_genes.txt")

# Export downregulated genes to a text file
write(downregulated_genes, file = "downregulated_genes.txt")

biological_processes <- "./Biological_Processes.txt"
david_data <- read.delim(biological_processes, header = TRUE, sep =
"\t", stringsAsFactors = FALSE)

molecular_data <- read.delim("./Molecular_Functions.txt", header = TRUE,
sep = "\t", stringsAsFactors = FALSE)
```

Output:



Functional Annotation Chart

Help and Manual

Current Gene List: downregulated_genes
Current Background: Homo sapiens
603 DAVID IDs

Options

Rerun Using Options | Create Sublist

9 chart records Download File

Sublist	Category	Term	RT	Genes	Count	%	P-Value	Benjamini
☐	UP_KW_BIOLOGICAL_PROCESS	Host-virus interaction	RT		39	6.5	1.1E-4	1.3E-2
☐	UP_KW_BIOLOGICAL_PROCESS	Glucose metabolism	RT		5	0.8	5.6E-4	3.4E-2
☐	UP_KW_BIOLOGICAL_PROCESS	Carbohydrate metabolism	RT		10	1.7	2.0E-3	8.3E-2
☐	UP_KW_BIOLOGICAL_PROCESS	Apoptosis	RT		28	4.6	5.3E-3	1.6E-1
☐	UP_KW_BIOLOGICAL_PROCESS	Tricarboxylic acid cycle	RT		5	0.8	7.3E-3	1.8E-1
☐	UP_KW_BIOLOGICAL_PROCESS	Vision	RT		9	1.5	3.1E-2	5.8E-1
☐	UP_KW_BIOLOGICAL_PROCESS	Lipid metabolism	RT		32	5.3	3.3E-2	5.8E-1
☐	UP_KW_BIOLOGICAL_PROCESS	Spermidine biosynthesis	RT		2	0.3	5.7E-2	8.8E-1
☐	UP_KW_BIOLOGICAL_PROCESS	Steroid metabolism	RT		7	1.2	9.8E-2	9.6E-1

Functional Annotation Chart

Help and Manual

Current Gene List: downregulated_genes
Current Background: Homo sapiens
603 DAVID IDs

Options

Rerun Using Options | Create Sublist

7 chart records — Download File

Sublist	Category	Term	RT	Genes	Count	%	P-Value	Benjamini
☐	UP_KW_MOLECULAR_FUNCTION	Activator	RT		37	6.1	7.4E-3	6.9E-1
☐	UP_KW_MOLECULAR_FUNCTION	Actin-binding	RT		16	2.7	5.0E-2	8.2E-1
☐	UP_KW_MOLECULAR_FUNCTION	Kinase	RT		34	5.6	5.1E-2	8.2E-1
☐	UP_KW_MOLECULAR_FUNCTION	Lyase	RT		11	1.8	6.0E-2	8.2E-1
☐	UP_KW_MOLECULAR_FUNCTION	Monooxygenase	RT		7	1.2	7.9E-2	8.2E-1
☐	UP_KW_MOLECULAR_FUNCTION	Blood group antigen	RT		4	0.7	9.8E-2	8.2E-1
☐	UP_KW_MOLECULAR_FUNCTION	Protein phosphatase	RT		9	1.5	9.8E-2	8.2E-1

Conclusion:

This assignment demonstrates how RNA-Seq data can be used to identify genes that are differentially expressed between conditions using DESeq2. The results provide insights into gene regulation, enabling further biological interpretations and potential downstream analysis.

Assignment 3

Title: Protein Structure Prediction

Problem Statement: Task: Predict the 3D structure of a given protein sequence using homology modeling. Deliverable: A report presenting the predicted protein structure, along with an analysis of its potential functions and interactions.

Dataset: "6n37.pdb"

Description: The 6N37 dataset represents the structural conformation of the low complexity domain segment A of TDP-43, a protein implicated in neurodegenerative diseases such as amyotrophic lateral sclerosis (ALS) and frontotemporal dementia (FTD). The protein is classified as a fibril, indicating its involvement in amyloid-like structures that can form in pathological conditions. The dataset was expressed in Escherichia coli BL21(DE3), a common bacterial system used for protein expression. The data was deposited in the Protein Data Bank on November 14, 2018, and released for public access on June 26, 2019. The research was funded by the National Institutes of Health, highlighting its significance in understanding protein misfolding and aggregation mechanisms in human diseases.

Tasks to Perform:

1. **Obtain Target Sequence** from UniProt or other databases.
2. **Identify Templates** via BLAST search.
3. **Build a Model** with SWISS-MODEL or Modeller.
4. **Refine and Validate** model with Ramachandran plot.
5. **Analyze Functionality** and potential interactions.

Prerequisite:

1. **Basic Understanding of Molecular Biology**: Familiarity with protein structure and function, including the levels of protein organization (primary, secondary, tertiary, quaternary).
2. **Computational Biology Tools**: Knowledge of tools like UniProt, BLAST, and SWISS-MODEL for protein sequence analysis and structure prediction.
3. **Programming Skills**: Basic Python skills for writing scripts and handling web

requests.

4. **Structural Biology Concepts**: Understanding of structure validation metrics (e.g., ERRAT scores, Ramachandran plots).

Theory:

Protein Structure Basics

- **Proteins** are composed of amino acid sequences folded into specific 3D structures.
- Structure determines the **function** and interaction of proteins with other molecules.
- Proteins have four levels of structure:
    1. **Primary** – Linear amino acid sequence.
    2. **Secondary** – Alpha-helices and beta-sheets.
    3. **Tertiary** – 3D folding of secondary structures.
    4. **Quaternary** – Assembly of multiple protein subunits.

Understanding protein structure helps identify **biological functions**, binding partners, and drug targets.

What is Homology Modelling?

- **Homology Modelling** predicts the 3D structure of a target protein using a known **template** protein with a similar sequence.
- It assumes that similar sequences fold into similar structures, leveraging evolutionary relationships.

Types of Protein Structure Prediction Techniques

1. **Homology Modelling**:
    - Relies on a **template protein** to build the target structure.
    - Most effective when the sequence identity between the target and template is **30% or higher**.
2. **Threading (Fold Recognition)**:

- Used when no closely related template is available.
- Matches the target sequence to known **structural folds**.

3. **Ab Initio Modelling**:
   - Predicts the structure from scratch without any template.
   - Computationally intensive but useful for **novel proteins** with no known homologs.

Steps in Predicting the 3D Structure of a Protein

Step 1: Retrieve Protein Sequence from UniProt

1. **Access UniProt**: Go to the UniProt website.
2. **Search for the Target Protein**: Enter the name or accession number of the target protein (e.g., "xyz protein") in the search bar.
3. **Copy the Protein Sequence**: Locate the protein sequence in the results and copy it for further analysis.

Step 2: Predict the 3D Structure Using SWISS-MODEL

1. **Access SWISS-MODEL**: Visit the SWISS-MODEL website.
2. **Paste the FASTA Sequence**: In the modeling tool, paste the copied FASTA sequence of the target protein.
3. **Select Templates**: The tool will automatically search for suitable templates based on sequence similarity. Review the suggested templates and select the one with the highest alignment score or maximum sequence identity to your target protein.
4. **Run the Modeling Process**: Start the modeling process, which aligns the target sequence with the selected template and generates the predicted 3D structure in PDB format.

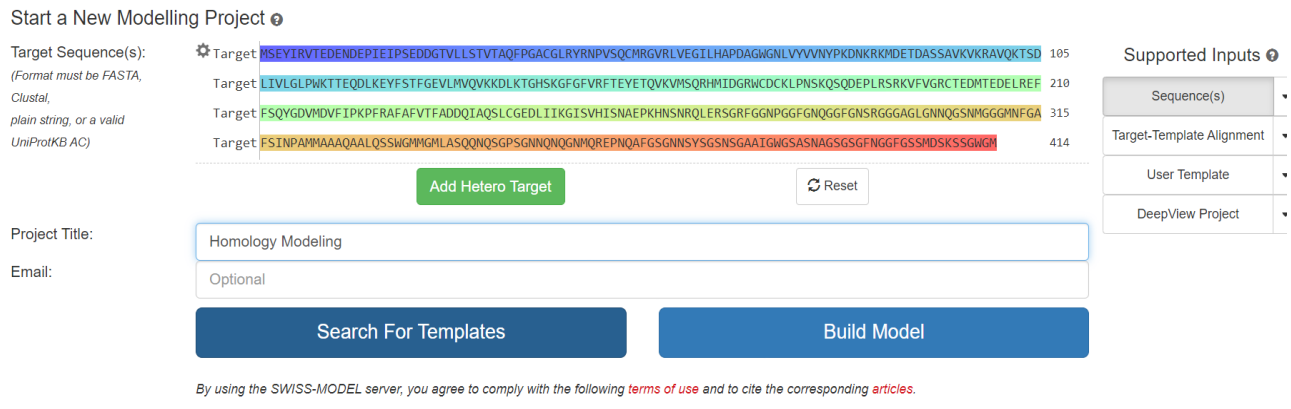
Step 3: Validate the Predicted Structure

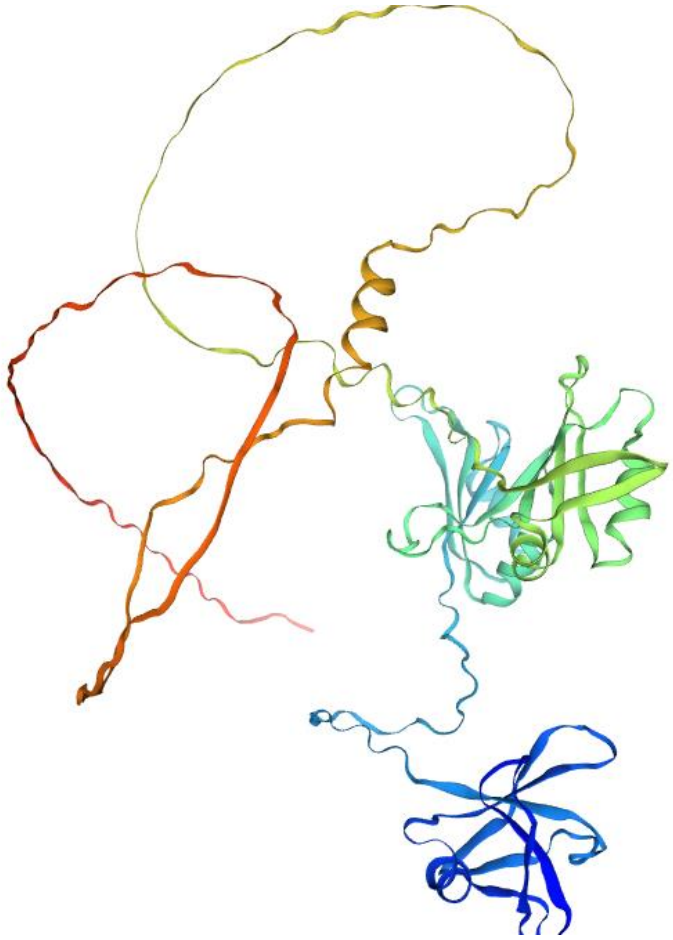
1. **Use SAVES Server**: Go to the SAVES Server for structural validation.
2. **Upload the Predicted PDB File**: Upload the generated PDB file of the predicted structure to the SAVES server.
3. **Check ERRAT Score**: The ERRAT tool will analyze the structure for non-ideal geometry. A high ERRAT score indicates potential structural issues.
4. **Use PROCHECK**: Utilize the PROCHECK tool to validate the geometry of the protein structure. PROCHECK provides information on the stereochemical quality of the structure, including:
    - **Ramachandran Plot**: Assessing the distribution of dihedral angles (phi and psi) of the amino acids.
    - **Bond Lengths and Angles**: Evaluating the accuracy of bond lengths and angles in the structure.

Tools Used for Homology Modelling

1. **SWISS-MODEL**:
    - An automated web-based tool for homology modeling.
    - Provides sequence alignment, model generation, and validation tools.

Output:





UCLA-DOE LAB — SAVES v6.1

UCLA

Job 71207 has been created

New Job

job #71207: 6n37.pdb [job link] [3D Viewer]

ERRAT Complete

Overall Quality Factor

98.4906

Results

Verify3D

Determines the compatibility of an atomic model (3D) with its own amino acid sequence (1D) by assigning a structural class based on its location and environment (alpha, beta, loop, polar, nonpolar etc) and comparing the results to good structures.

Start

PROVE

Temporarily down at the moment

WHATCHECK

Derived from a subset of protein verification tools from the WHATIF program (Vriend, 1990), this does extensive checking of many stereochemical parameters of the residues in the model.

Start

PROCHECK Complete

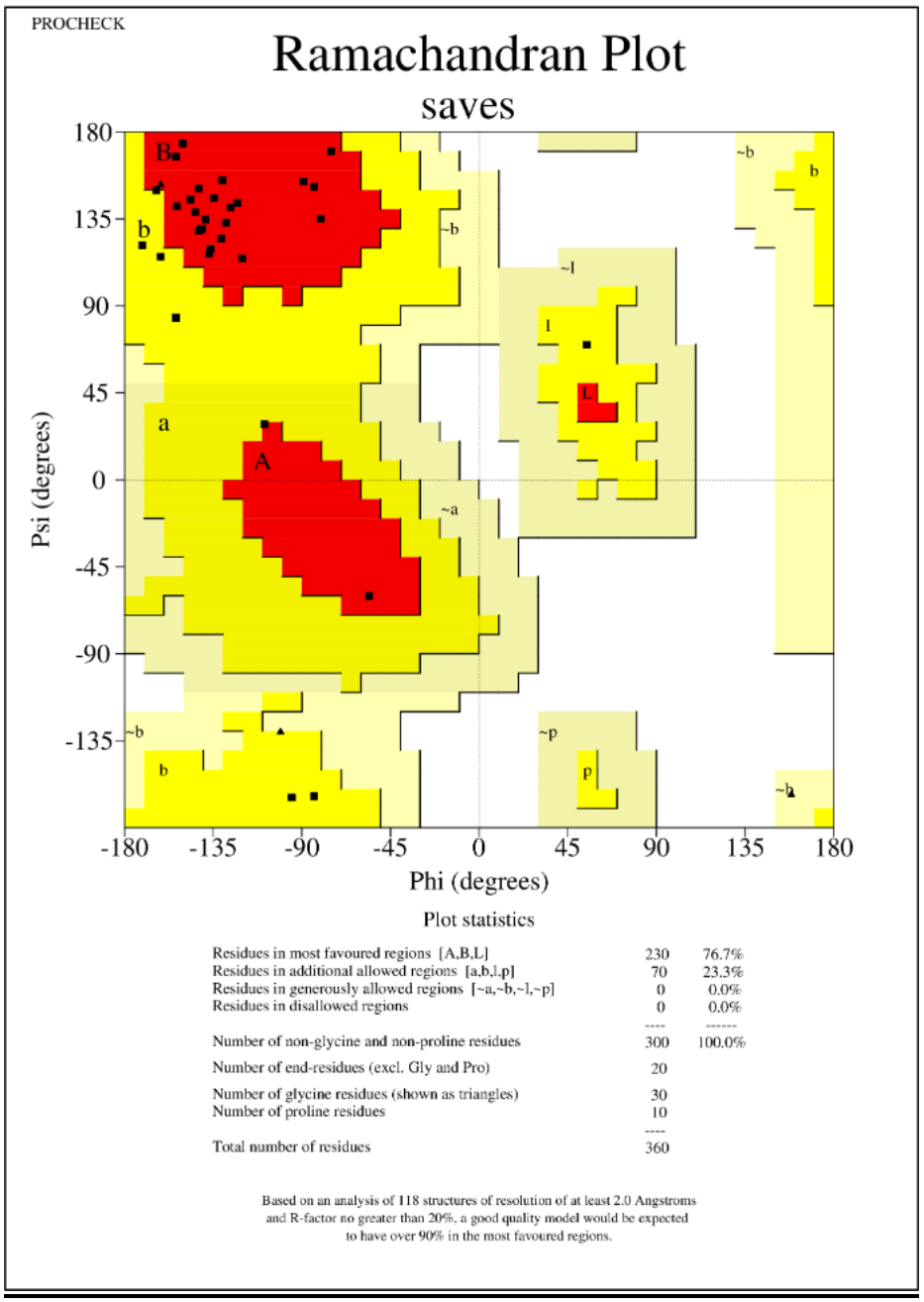
Out of 8 evaluations

- Errors: 2
- Warning: 3
- Pass: 3

Results

OPEN

We are open to suggestions for a 6th program to operate in this window. If you know of a program that we could run locally on our server that would be most useful, please let us know: email holton at mbi dot ucla dot edu with your suggestion



Conclusion:

Homology modelling is a widely used technique in structural bioinformatics to predict protein structures. It provides valuable insights into protein function, interactions, and potential drug binding sites. This method relies on evolutionary relationships, making it effective for proteins with homologous sequences and known 3D structures. Tools like **SWISS-MODEL** streamline the process, while visualization tools ensure accurate interpretation of results.

Assignment 4

Title: Machine Learning for Genomic Data

Problem Statement: Task: Apply machine learning algorithms, such as random forests or support vector machines, to classify genomic data based on specific features or markers. Deliverable: A comprehensive analysis report presenting the classification results, model performance evaluation, and insights into the predictive features.

Dataset: "METABRIC_RNA_Mutation.csv"

Description:

Name	Type	Description
patient_id	object	Unique identifier for each patient. Used as the index in the code.
age_at_diagnosis	float	Age of the patient at the time of diagnosis. Can be used as a feature in the prediction model.
chemotherapy	int	Whether the patient underwent chemotherapy (1 = Yes, 0 = No). Can affect survival outcomes.
hormone_therapy	int	Whether the patient received hormone therapy (1 = Yes, 0 = No).
radio_therapy	int	Whether the patient received radiotherapy (1 = Yes, 0 = No).
neoplasm_histologic_grade	int	Aggressiveness of the tumor, graded on a scale from 1 to 3. Higher values indicate more aggressive cancer.
tumor_size	float	Size of the tumor measured by imaging techniques. Larger sizes might correlate with worse outcomes.
tumor_stage	float	Cancer stage, indicating the extent of local and distant spread. Higher stages reflect more advanced disease.
mutation_count	float	Number of relevant gene mutations detected. May serve as a feature to predict survival.
lymph_nodes_examined_positive	float	Number of lymph nodes tested positive for cancer involvement. Higher counts are often associated with worse prognosis.
overall_survival_months	float	Duration from intervention to either death or the end of the study (in months). Useful in survival analysis.
overall_survival	object	**Target variable** in the code. Indicates whether the patient is alive or dead.

Tasks to Perform:

1. Data Preprocessing: Handle missing values (if any)

2. Encode categorical data: Split data into training and testing sets
3. Train Models: Train two models: Random Forest (RF) and Support Vector Classifier (SVC)
4. Evaluate Models: Use accuracy, F1 score, confusion matrix, and AUC-ROC for performance assessment
5. Visualize Results: Generate bar plots of F1 scores and accuracy for comparison
6. Extract Insights: Identify important genomic features that influence survival predictions

Theory:

Machine Learning Models Used

1. Random Forest Classifier (RF)

Concept: Random Forest is an **ensemble learning algorithm** that builds multiple **decision trees** and combines their predictions to provide more accurate results.

- **How it works**:
  - Each tree is trained on a random subset of the data.
  - The final prediction is made by taking a **majority vote** of all the trees (for classification tasks).
- **Advantages**:
  - **Handles high-dimensional data**: Random Forest can deal with many gene expression features.
  - **Resistant to overfitting**: As multiple trees are trained, the overall model becomes more robust.
  - **Feature Importance**: Random Forest can rank genes based on their contribution to the model's predictions, helping researchers identify **key biomarkers** associated with survival.

2. Support Vector Classifier (SVC)

Concept: SVC is a **supervised learning model** that finds the best **hyperplane** to separate data points into different classes.

- **How it works**:
  - In a 2D space, the hyperplane is a line that separates the two classes.
  - The goal is to **maximize the margin** between the hyperplane and the nearest data points (called **support vectors**).
- **Kernel Trick**: If the data is not linearly separable (as is often the case with biological data), SVC can use a **kernel function** (such as the RBF kernel) to map the data into a higher-dimensional space where it becomes separable.
- **Advantages**:
  - Works well with small datasets.
  - Effective when there is a **non-linear relationship** between features and outcomes.

Model Training and Evaluation

Once the models are trained, we need to **evaluate their performance** using various metrics. These metrics provide insights into the quality of the predictions and identify areas for improvement.

1. Accuracy Score

- **Formula**:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

- **Interpretation**: Accuracy indicates the proportion of correct predictions. However, it can be misleading for **imbalanced datasets**, where one class dominates the data.

2. F1 Score

- **Formula**:

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

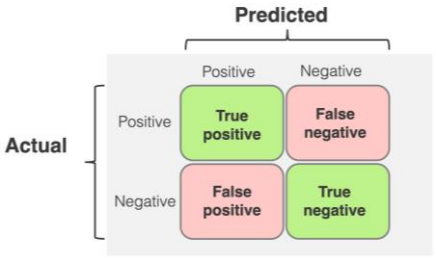
- **Interpretation**: The F1 score balances **precision** (the proportion of correct positive predictions) and **recall** (the proportion of actual positives identified). It is useful for datasets where **false positives and false negatives are costly**.

3. Confusion Matrix

The **confusion matrix** shows how many predictions were correct or incorrect, breaking it down into:

- **True Positive (TP)**: Correctly predicted positive samples.
- **True Negative (TN)**: Correctly predicted negative samples.
- **False Positive (FP)**: Incorrectly predicted positive samples.
- **False Negative (FN)**: Incorrectly predicted negative samples.

Example of a Confusion Matrix Visualization:



The matrix helps us see where the model is making mistakes.

4. AUC-ROC Score

- **Concept**: AUC-ROC measures the model's ability to discriminate between the classes.
- **Interpretation**:
  - **ROC Curve**: A plot of **True Positive Rate (TPR)** vs. **False Positive Rate (FPR)** at different thresholds.
  - **AUC (Area Under the Curve)**: A value close to **1.0** indicates that the model performs well in distinguishing between the survival and non-survival outcomes.

Visualizing Model Performance

The code also generates visualizations such as **bar plots of accuracy and F1 scores** for easier comparison between the models.

- **Heatmaps for Confusion Matrix**: These show the number of correct and incorrect predictions visually.
- **Bar Plots for Accuracy and F1 Scores**: Help in identifying which model performs better overall.

Code:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import (
    classification_report, confusion_matrix, roc_auc_score,
accuracy_score, f1_score
)
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC

# Load and process the dataset
metabric_df = pd.read_csv("./METABRIC_RNA_Mutation.csv")
metabric_df.shape
metabric_df.info(verbose=True)
metabric_df.sample(5)
metabric_df = metabric_df.set_index('patient_id')
df_expression = metabric_df.iloc[:,
30:519].join(metabric_df['overall_survival'], how='inner')
df_expression
# Dictionary to store F1 and accuracy scores of each model
metrics_summary = {"Model": [], "F1 Score": [], "Accuracy": []}
# Function to evaluate and display results

def evaluate_model(model, X_train, X_test, y_train, y_test,
model_name):
```

```
    model.fit(X_train, y_train)

    # Predictions and probabilities (if applicable)
    y_pred = model.predict(X_test)
    y_proba = model.predict_proba(X_test)[:, 1] if hasattr(
        model, 'predict_proba') else None

    # Calculate metrics
    f1 = f1_score(y_test, y_pred, average='weighted')
    accuracy = accuracy_score(y_test, y_pred)

    # Store the metrics
    metrics_summary["Model"].append(model_name)
    metrics_summary["F1 Score"].append(f1)
    metrics_summary["Accuracy"].append(accuracy)

    # Display classification metrics
    print(f"\n=== {model_name} ===")
    print("Classification Report:\n", classification_report(y_test,
y_pred))
    print(f"Accuracy: {accuracy:.4f}")
    print(f"F1 Score: {f1:.4f}")

    # Confusion Matrix Visualization
    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(6, 4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=[
```

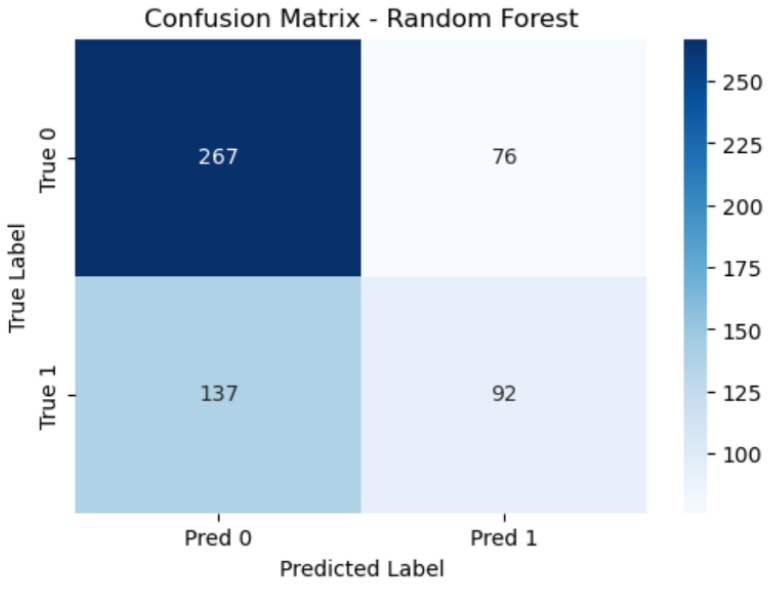
```
                'Pred 0', 'Pred 1'], yticklabels=['True 0', 'True
1'])
    plt.title(f'Confusion Matrix - {model_name}')
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')
    plt.show()

    # AUC-ROC Score (if applicable)
    if y_proba is not None:
        print(f"AUC-ROC Score: {roc_auc_score(y_test,
y_proba):.4f}")
# Main function to process the dataset and evaluate models


def main(df, target_column):
    # Prepare the dataset
    X = df.drop(target_column, axis=1)
    y = df[target_column]

    # Handle categorical variables
    X = pd.get_dummies(X, drop_first=True)
    if y.dtype == 'object':
        y = LabelEncoder().fit_transform(y)

    # Split the dataset
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.3, random_state=42)
```

```
    # Define classifiers to be evaluated
    models = {
        'Random Forest': RandomForestClassifier(),
        'Support Vector Classifier': SVC(probability=True),
    }

    # Train and evaluate all models
    for name, model in models.items():
        evaluate_model(model, X_train, X_test, y_train, y_test,
name)

    # Plot F1 Scores of all models
    plt.figure(figsize=(10, 5))
    sns.barplot(x=metrics_summary["Model"],
                y=metrics_summary["F1 Score"], palette='coolwarm')
    plt.xticks(rotation=45)
    plt.title('F1 Scores of All Models')
    plt.ylabel('F1 Score')
    plt.show()

    # Plot Accuracy Scores of all models
    plt.figure(figsize=(10, 5))
    sns.barplot(x=metrics_summary["Model"],
                y=metrics_summary["Accuracy"], palette='coolwarm')
    plt.xticks(rotation=45)
    plt.title('Accuracy Scores of All Models')
    plt.ylabel('Accuracy')
    plt.show()
```

```
# Load your dataset and specify the target column
if __name__ == "__main__":
    # Replace with your dataframe
    df = df_expression  # Example input dataframe
    target_column = 'overall_survival'  # Replace with the actual
target column name
    main(df, target_column)
```

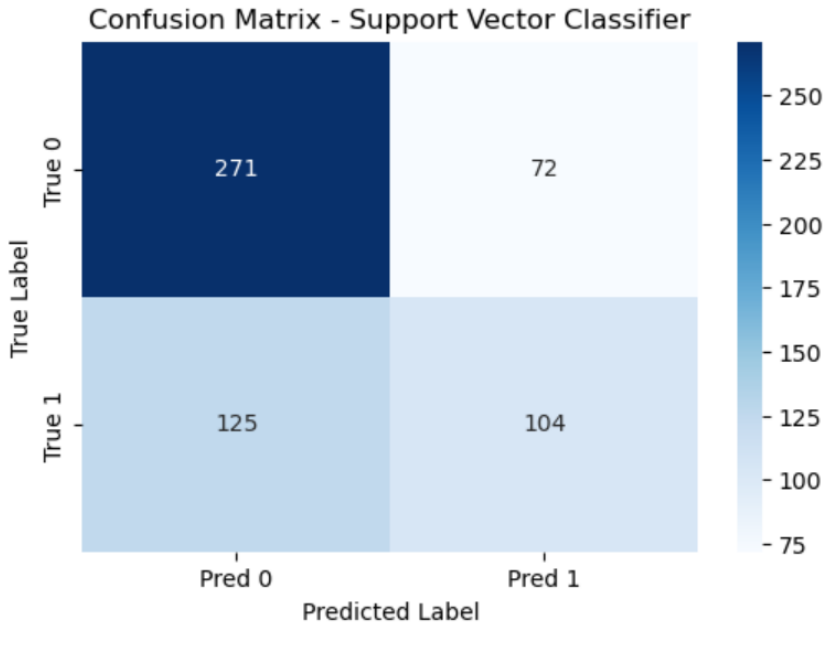
Output:

```
=== Random Forest ===
Classification Report:
              precision    recall  f1-score   support

           0       0.66      0.78      0.71       343
           1       0.55      0.40      0.46       229

    accuracy                           0.63       572
   macro avg       0.60      0.59      0.59       572
weighted avg       0.62      0.63      0.61       572

Accuracy: 0.6276
F1 Score: 0.6142
```

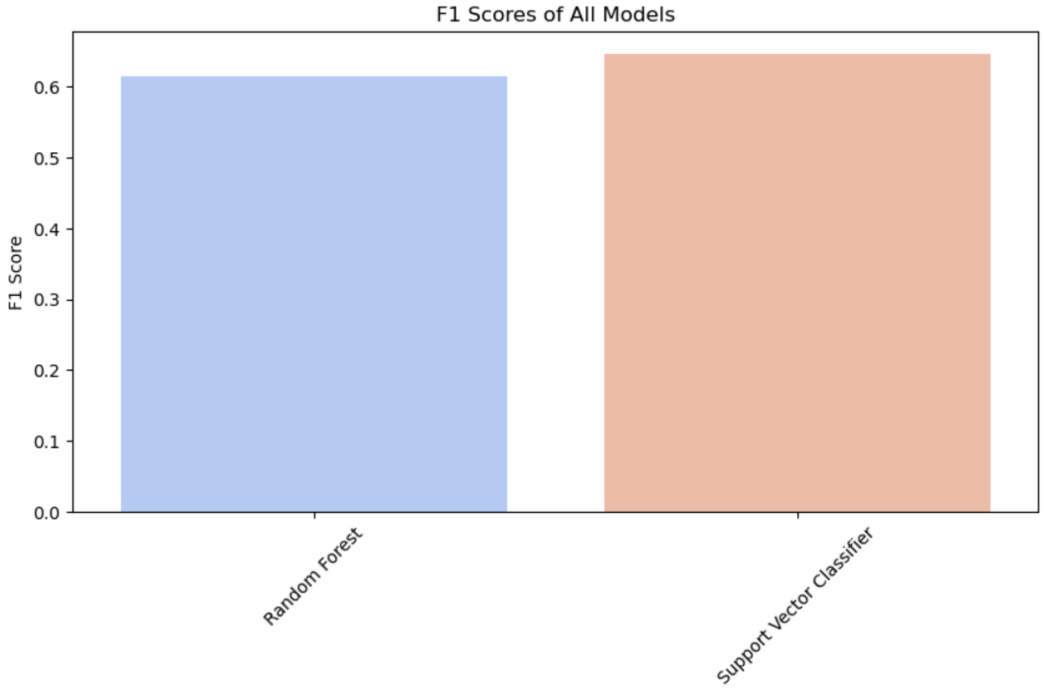


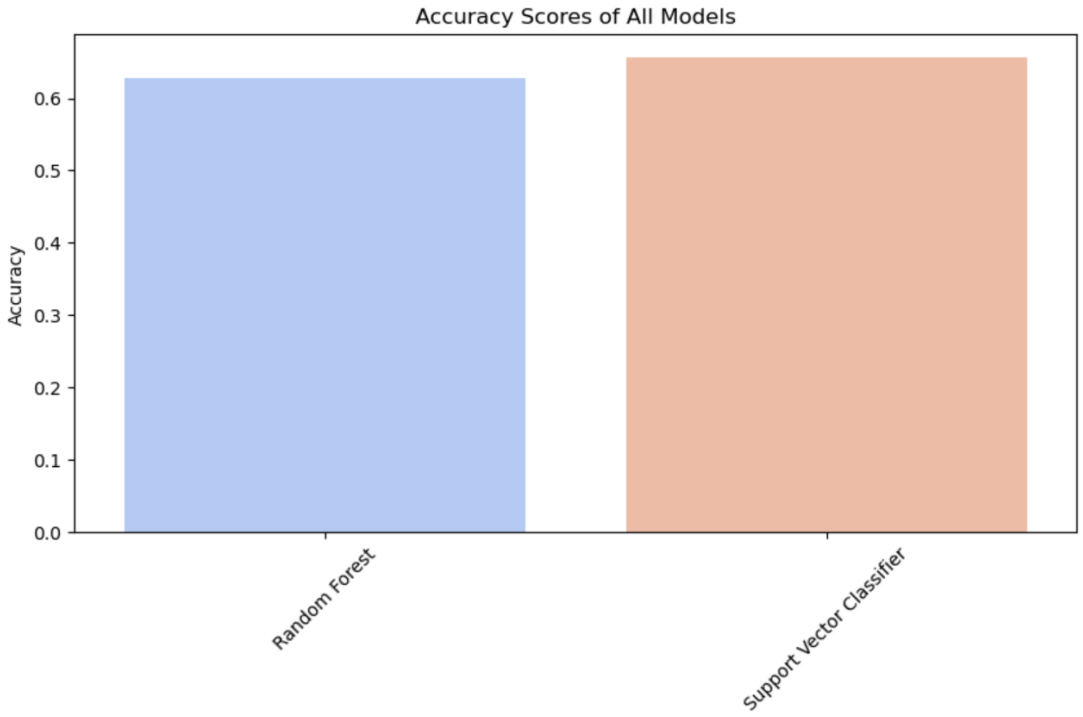

```
AUC-ROC Score: 0.6538
```

```
=== Support Vector Classifier ===
Classification Report:
              precision    recall  f1-score   support

           0       0.68      0.79      0.73       343
           1       0.59      0.45      0.51       229

    accuracy                           0.66       572
   macro avg       0.64      0.62      0.62       572
weighted avg       0.65      0.66      0.65       572

Accuracy: 0.6556
F1 Score: 0.6454
```




```
AUC-ROC Score: 0.6740
```





Conclusion:

In this assignment, we applied Random Forest and SVC to classify patients based on genomic data. Both models were evaluated using metrics such as accuracy, F1 score, confusion matrix, and AUC-ROC. These metrics helped us understand the strengths and limitations of each model in predicting patient outcomes.

Mini Project

Title: Molecular Docking

Problem Statement: Task: Perform molecular docking simulations to predict the binding affinity between a protein target and a small molecule ligand. Deliverable: A report summarizing the docking results, including the binding poses and potential lead compounds.

Dataset:

Protein target structure and ligand molecules (retrieved from Protein Data Bank (PDB) or PubChem databases). Example: Delaverdine and Penicillin ligand molecules.

Tools used: AutoDock and MGM tools for molecular docking simulations.

Description: The aim of this assignment is to understand the process of molecular docking, which is a key technique in the field of computational drug discovery. The task involves predicting how a small molecule (ligand) fits into the active site of a protein (receptor), and how strongly it binds. This information helps in the identification of molecules that may act as potential inhibitors or modulators of the protein's activity, which is useful for developing new therapeutics.

Tasks to Perform:

Protein Preparation

- Get protein structure from **PDB**.
- Remove water, add hydrogens, assign charges, and define the **active site**.

Ligand Preparation

- Retrieve or draw ligands, **minimize energy**, and assign partial charges.

Docking Simulation

- Set a grid **box** and algorithm in AutoDock, then run the simulation.

Result Analysis

- Visualize poses, assess **interactions** (e.g., hydrogen bonds), and select the best one.

Report Generation

- Summarize **binding energy** and interactions, with visualizations.

Prerequisite:

- **Basic Chemistry Knowledge**: Understanding of **atoms, molecules**, and **chemical bonds**.
- **Protein Structures and Ligand Chemistry**: Knowledge about **protein active sites, ligand structures**, and **binding pocket identification**.
- **Familiarity with Docking Tools**: Experience using **AutoDock and MGM Tools** for molecular modeling.
- **Understanding of Computational Methods**: Awareness of **energy minimization, grid generation**, and **scoring functions** used in docking.

Theory:

What is Molecular Docking?

Molecular docking is a **computational method** used to predict the preferred orientation and **binding affinity** of a ligand when it interacts with a target protein. The docking process aims to **simulate the binding** of the ligand to the active site of the protein to find the **most stable conformations** based on energy calculations.

- **Receptor (Protein)**: The macromolecule, usually a protein, which has a specific site where the ligand can bind. The **binding site** is often the **active site** or a region where **biological activity** is regulated.
- **Ligand (Small Molecule)**: A small molecule that can interact with the protein's active site, potentially affecting the protein's function. **Ligands** could be **drug candidates**, **inhibitors**, or **substrates** for the protein.
- **Binding Affinity**: Refers to the **strength of interaction** between the ligand and the receptor, which is often expressed as a **binding energy**. The lower the binding energy (more negative), the stronger the interaction.

Steps in Molecular Docking

1. **Protein Preparation**:
    - **Structure Cleaning**: Removing **non-essential molecules** like water that could interfere with docking simulations.
    - **Adding Hydrogens and Assigning Charges**: Properly **placing hydrogens** to ensure the protein's structure is complete and assigning **partial charges** to atoms using force fields.
    - **Defining the Active Site**: The **region where docking** will be focused is usually the **enzyme's active site** or any **binding pocket** known for interaction with ligands.
2. **Ligand Preparation**:
    - **Energy Minimization**: The ligand's 3D structure is optimized to reach a **low-energy stable state**, ensuring that it is in the correct conformation for docking.
    - **Partial Charges and Atom Types Assignment**: Charges are calculated for each atom in the ligand, which influences how the ligand interacts with the protein's active site.
3. **Docking Execution**:
    - **Grid Generation**: A **grid box** is created around the binding site, allowing the docking software to search for the **best position and orientation** of the ligand within this region.
    - **Search Algorithm**: Algorithms such as **Genetic Algorithm (GA)** or **Simulated Annealing** are used to explore the **possible orientations** and **positions** of the ligand.
4. **Scoring and Ranking**:
    - The docking software calculates the **binding affinity** for each pose. Lower **binding energy scores** (more negative values) indicate stronger predicted binding.
    - The poses are then **ranked** based on their scores, and the **top poses** are selected for further analysis.

5. **Post-Docking Analysis**:
    - **Visualize Interactions**: Molecular visualization tools are used to observe **key interactions** such as **hydrogen bonds, hydrophobic contacts**, and **electrostatic interactions**.
    - **Identify the Best Pose**: The pose with the **lowest energy score** and **best interaction quality** is selected as the most likely binding mode.

Tools Used in Molecular Docking

1. **AutoDock**
    - A **widely-used software suite** for molecular docking simulations, AutoDock predicts **ligand binding modes** within the active site of the receptor.
    - **AutoDockTools (ADT)** is used to **prepare the protein and ligand**, including adding charges, setting atom types, and defining the grid box.
    - AutoDock employs **search algorithms** (e.g., **Lamarckian Genetic Algorithm**) for **pose prediction** and generates a **binding energy score** for each pose.
2. **MGM Tools**
    - These tools aid in **pre-processing the data** and can be used for **automated docking workflows**. They simplify **grid box setting, ligand preparation**, and **result analysis**.
    - Often used in combination with **AutoDock** to streamline the process, MGM Tools ensure the correct **preparation of ligands and protein structures** before the actual docking run.

Receptor-Ligand Interactions

In molecular docking, several types of **molecular interactions** help determine the **stability of the ligand-receptor complex**:

- **Hydrogen Bonds**: **Weak attractions** between a hydrogen atom and an electronegative atom, often stabilizing the binding pose.
- **Hydrophobic Interactions**: Occur between **non-polar groups**, contributing to the **binding energy**.
- **Electrostatic Interactions**: Attractions or repulsions between **charged groups** that can significantly affect the **binding affinity**.

Output:

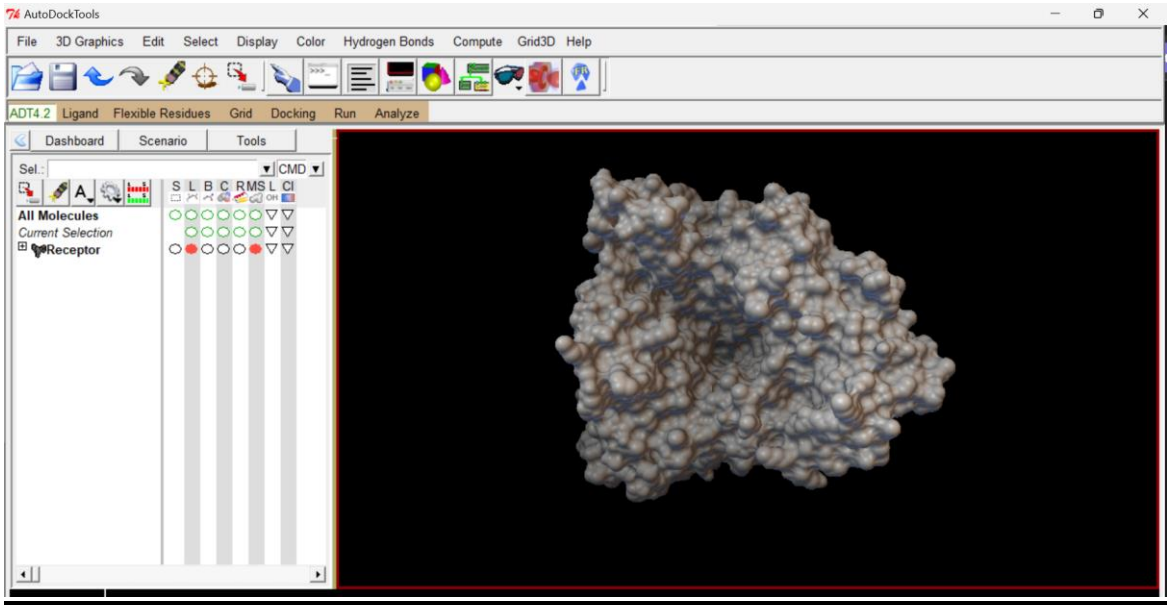


Fig. Protein Preparation

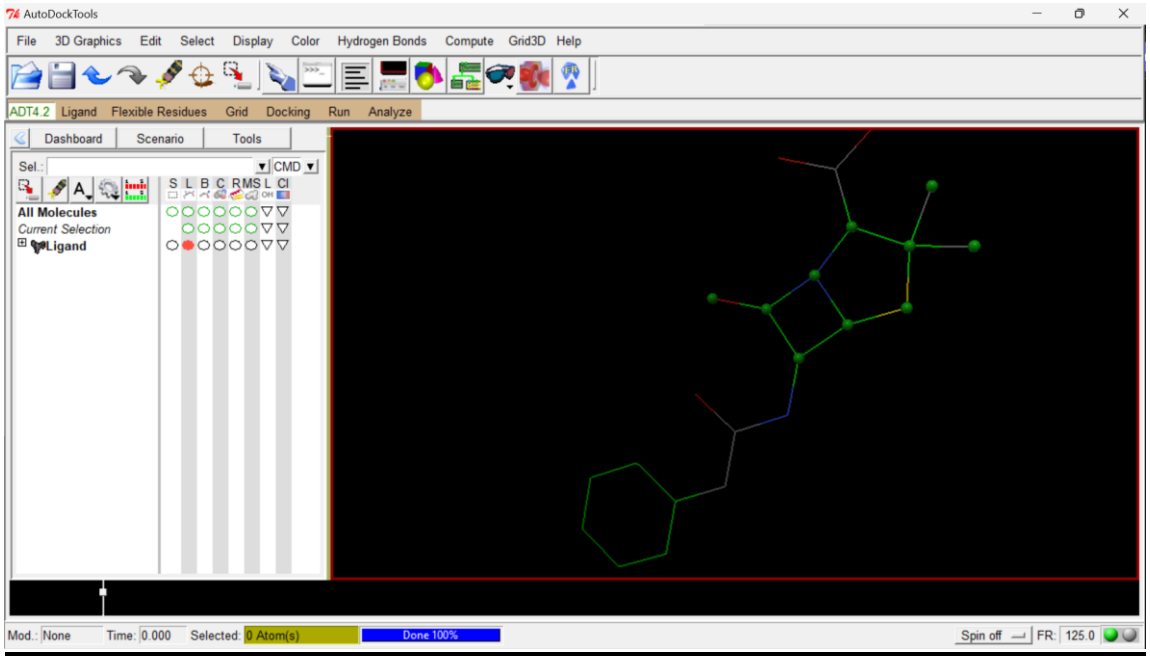


Fig. Ligand Preparation

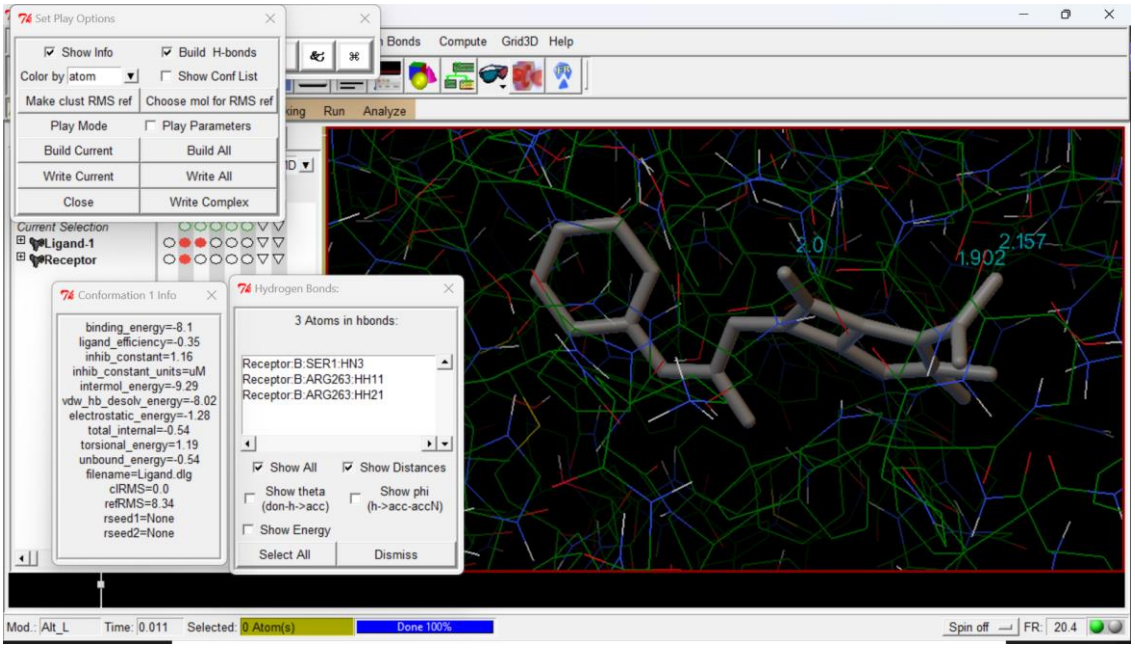


Fig. Interaction in Lowest Binding Energy State

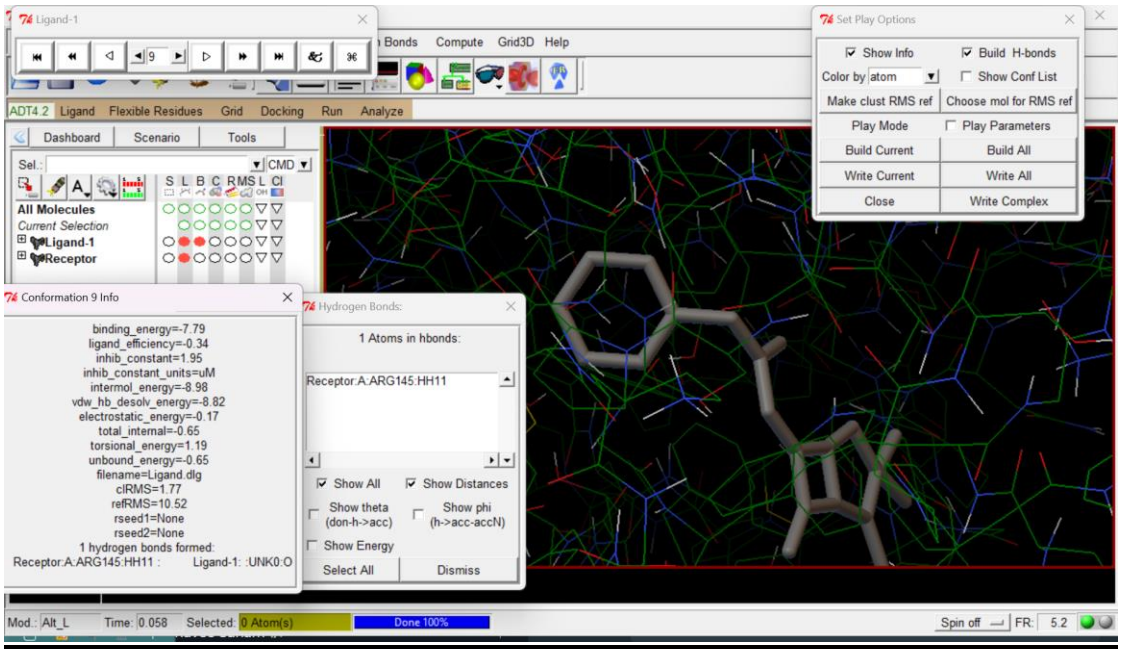


Fig. Interaction in Higher Binding Energy State

Conclusion:

Molecular docking is an essential computational approach in drug discovery for predicting the interaction strength and binding modes of small molecules to their protein targets. Using tools like AutoDock and MGM Tools, the docking process involves protein and ligand preparation, docking execution, and analysis of results, providing insights into the binding affinity and molecular interactions.