Sample

COMPUTATIONAL STUDY OF NEGATIVE DETERMINANTS IN TRANSCRIPTION FACTOR SPECIFICITY EVOLUTION

EXPLORING STERIC CLASHES, ELECTROSTATIC REPULSION, AND UNPAIRED POLAR ATOMS

Sample Gene RFX1

1. Introduction to RFX1 Gene.

Regulatory Factor X1 (RFX1) gene is a member of the RFX transcription factor family, which plays a crucial role in regulating gene expression, particularly in immune response, neurodevelopment, and cell cycle control. RFX1 contains a conserved winged-helix DNA-binding domain, allowing it to bind to cis-regulatory elements in promoter regions and regulate target genes. This gene is widely studied for its involvement in transcriptional repression, epigenetic modifications, and potential links to diseases such as cancer and neurodevelopmental disorders.

2. Methodology.

I followed a systematic computational approach to analyze RFX1 orthologs, extract transcription factor sequences, identify DNA-binding domains, and construct phylogenetic trees. My pipeline includes the following steps:

2.1 Data Collection and Preprocessing.

1. Creating a Directory Structure:

```
Bash
mkdir Sample_RFX1 && cd Sample_RFX1 && mkdir D.C && cd D.C && mkdir
Ortholog && mkdir DBD && mkdir MSA && cd Ortholog && mkdir unzip
```

Explanation:

I created a directory named **Sample RFX1**, and within it, a subdirectory **D.C**, in that I created 3 directory named **Ortholog** -> to store ortholog data, **DBD** -> to store DNA Binding Domain data, **MSA** -> to store Multiple Sequence Alignment data, then enter Ortholog Directory.

2. Downloading Ortholog Data:

I executed the following command to retrieve RFX1 orthologs from the NCBI database:

```
Bash
datasets download gene symbol RFX1 --ortholog all
```

Explanation:

- datasets is an NCBI command-line tool used to download biological datasets.
- `download gene symbol RFX1` specifies that I want to download the dataset for the RFX1 gene using its symbol.
- `--ortholog all` ensures that I retrieve ortholog data across multiple species.

3. Extracting Files:

Since the data is downloaded as a ZIP file, I extracted it using:

```
Bash
unzip /home/zero/Videos/Sample_RFX1/D.C/Ortholog/ncbi_dataset.zip -d
/home/zero/Videos/Sample_RFX1/D.C/Ortholog/unzip
```

Explanation:

- `unzip`: Command to extract the contents of a .zip file.
- `/home/zero/Videos/Sample_RFX1/D.C/Ortholog/ncbi_dataset.zip`: Path to the zip file.
- `-d /home/zero/Videos/Sample_RFX1/D.C/Ortholog/unzip`: Specifies the destination directory where the extracted files should be placed.

- This results include multiple files and folder:
  - 📂 unzip/
    ├── 📄 README.md
    ├── 📄 md5sum.txt
    ├── 📂 ncbi_dataset/
    │ ├── 📂 data/
    │ │ ├── 📄 rna.fna
    │ │ ├── 📄 protein.faa
    │ │ ├── 📄 data_report.jsonl
    │ │ └── 📄 dataset_catalog.json

2.2 Transcription Factor (TF) Sequence Extraction

1. I navigated to the data directory:

```
Bash
cd
/home/zero/Videos/Sample_RFX1/D.C/Ortholog/unzip/ncbi_dataset/data
```

2. I extracted transcription factor sequences from protein.faa based on relevant keywords:

```
Bash
awk '
    /^>/ {header=$0; next}
    /transcription factor|TF|DNA-binding|zinc finger|helix-turn-
helix|homeobox/ {print header; print; capture=1; next}
    capture {print}
' protein.faa > TF_sequences.faa
```

Explanation:

`awk` is a powerful text-processing tool in Unix/Linux used for pattern matching, text extraction, and data manipulation. It operates on a line-by-line basis and processes text based on defined patterns and actions.

1. `/^>/ {header=$0; next}`
   - When encountering a sequence header (`>...`), store it in the `header` variable and move to the next line.
2. `/transcription factor|TF|DNA-binding|zinc finger|helix-turn-helix|homeobox/ {print header; print; capture=1; next}`
   - If the line contains any of the listed TF-related keywords, print the stored header and the current line, then activate `capture=1`.
3. `capture {print}`
   - If `capture=1` (i.e., we found a TF keyword), print the entire sequence until the next header is encountered.

3. Copy `TF_sequences.faa` to the **DBD** directory:

```
Bash
cp
/home/zero/Videos/Sample_RFX1/D.C/Ortholog/unzip/ncbi_dataset/data/
TF_sequences.faa /home/zero/Videos/Sample_RFX1/D.C/DBD
```

2.3 Identifying DNA-Binding Domains (DBDs)

1. I ran hmmscan against the Pfam database to identify DNA-binding domains:

```
Bash
hmmscan --tblout DBDs.tsv ~/Pfam-A.hmm TF_sequences.faa
```

Explanation:

- `hmmscan` searches for protein domains in sequences.
- `--tblout DBDs.tsv` saves the results in a tabular format.
- `~/Pfam-A.hmm` specifies the Pfam database containing DNA-binding domains.
- `TF_sequences.faa` is my input file containing transcription factor sequences.

2. I filtered relevant DNA-binding domains (DBDs):

```
Bash
grep -E "PF00096|PF00172|PF00176|PF00319|PF00439|PF00847|PF01429|
PF02178|PF02365|PF07716|PF02257|" DBDs.tsv > filtered_DBD.tsv
```

Explanation:

1. `grep`
    - `grep` is used to search for patterns in a file.
2. `-E (Extended Regular Expressions)`
    - The `-E` flag enables "Extended Regular Expressions" (ERE), allowing the use of the pipe (`|`) operator as an OR condition without escaping it.
3. `"PF00096|PF00172|PF00176|PF00319|...|PF02257"`
    - This is a regular expression that searches for any of the listed PFAM domain IDs (`PF00096`, `PF00172`, etc.).
    - The pipe (`|`) acts as a logical OR, meaning `grep` will match any line containing **at least one** of these domain IDs.

    Breaking Down Pfam IDs

    A Pfam ID consists of:
    - **Prefix:** `"PF"` (indicating it belongs to the Pfam database)
    - **Number:** A unique identifier for the protein domain or family

    For example:
    - `PF00096` → **Homeobox domain** (DNA-binding domain in transcription factors)
    - `PF00172` → **bZIP transcription factor** domain
    - `PF00176` → **Zinc finger domain**
    - `PF00319` → **Nuclear hormone receptor ligand-binding domain**
    - `PF00439` → **Basic helix-loop-helix (bHLH) domain**
    - `PF00847` → **T-box transcription factor domain**
    - `PF01429` → **Forkhead transcription factor domain**
    - `PF02178` → **C2H2-type zinc finger domain**
    - `PF02365` → **Myb-like DNA-binding domain**
    - `PF07716` → **Ets domain (involved in gene regulation)**
    - `PF02257` → **HMG (High Mobility Group) DNA-binding domain**
4. `DBDs.tsv`
    - This is the input file that contains protein domain data.
5. `> filtered_DBD.tsv`
    - The `>` operator redirects the output (matched lines) into a new file called `filtered_DBD.tsv`, instead of printing it to the terminal.

3. Now I extracts the **third column** from filtered_DBD.tsv and saves it into DBD_sequence_Ids.txt using this command :

```
Bash
awk '{print $3}' filtered_DBD.tsv > DBD_sequence_Ids.txt
```

Explanation :

1. `awk`
    - `awk` is a powerful text-processing tool that operates on columns in a file.
2. `'{print $3}'`
    - `{print $3}` tells `awk` to print the **third column** (`$3`) of each line in `filtered_DBD.tsv`.
    - `$1`, `$2`, `$3`, etc., refer to different columns in a tab- or space-separated file.
3. `filtered_DBD.tsv`
    - This is the input file that contains only the filtered sequences (from your previous `grep` command).
4. `> DBD_sequence_IDs.txt`
    - Redirects (`>`) the output (extracted third column) into a new file called `DBD_sequence_IDs.txt`.

4. now I **extracts sequences** from `TF_sequences.faa` whose headers match the IDs listed in `DBD_sequence_IDs.txt` and saves them in `DBD_sequences.faa`.

```
Bash
seqkit grep -f DBD_sequence_IDs.txt TF_sequences.faa >
DBD_sequences.faa
```

Explanation :

1. `seqkit grep`
    - `seqkit` is a **fast and efficient** toolkit for working with FASTA/Q files.
    - `grep` is a subcommand that **searches for matching sequences** based on headers.
2. `-f DBD_sequence_IDs.txt`
    - `-f` (file input mode) tells `seqkit grep` to read search patterns (sequence IDs) from `DBD_sequence_IDs.txt`.
    - Each line in `DBD_sequence_IDs.txt` should contain one sequence ID (matching FASTA headers).
3. `TF_sequences.faa`
    - The input **FASTA file** containing **all transcription factor (TF) sequences**.
4. `> DBD_sequences.faa`
    - Redirects the output (matching sequences) into a new FASTA file `DBD_sequences.faa`.

5. Copy `DBD_sequences.faa` to MSA directory.

```
Bash
cp /home/zero/Videos/Sample_RFX1/D.C/DBD/DBD_sequences.faa
/home/zero/Videos/Sample_RFX1/D.C/MSA
```

2.4 Multiple Sequence Alignment (MSA).

Now I **performs multiple sequence alignment (MSA)** using **MAFFT**, then **fixes duplicate sequence headers** using awk.

Step 1: MAFFT - Multiple Sequence Alignment

```
Bash
mafft --auto --reorder --inputorder DBD_sequences.faa >
DBD_MSA_full.fasta
```

Explanation:

- `mafft` → Performs multiple sequence alignment.
- `--auto` → Automatically selects the best alignment strategy.
- `--reorder` → Reorders the output based on similarity.
- `--inputorder` → Keeps sequences in the input order (instead of clustering them first).
- `DBD_sequences.faa` → Input FASTA file containing DNA-binding domain (DBD) sequences.
- `> DBD_MSA_full.fasta` → Saves the aligned sequences into `DBD_MSA_full.fasta`.

Step 2: Fixing Duplicate Sequence Headers Using awk

```
Bash
awk '/^>/ {split($0, arr, " "); acc=arr[1]; count[acc]++; if
(count[acc] > 1) arr[1] = acc "_" count[acc]; print join(arr);
next} {print} function join(a, i, s){s=a[1]; for(i=2; i in a; i++)
s=s" "a[i]; return s}' DBD_MSA_full.fasta > DBD_MSA_fixed.fasta
```

What This Does

This awk script **modifies sequence headers** to ensure uniqueness while preserving annotations.

Breaking It Down

1. `/^>/` → Identifies FASTA headers (lines starting with >).
2. `split($0, arr, " ")` → Splits the header line into an array `arr` using **spaces** as delimiters.
3. `acc=arr[1]` → Extracts the main sequence ID (e.g., `>XP_026575699.1`).
4. `count[acc]++;` → Keeps track of how many times this sequence ID appears.
5. `If count[acc] > 1, append _X to duplicate headers:`
    - First occurrence: `>XP_026575699.1`
    - Second occurrence: `>XP_026575699.1_2`
    - Third occurrence: `>XP_026575699.1_3`
6. `print join(arr); next` → Prints the modified header and skips to the next line.
7. `{print}` → Prints non-header lines (sequence data) unchanged.
8. `function join(a, i, s){...}` → Reconstructs the header line with modifications.

Step 3: Now I navigates directories, creates a new folder (`Phylogeny`), moves into it, and copies an alignment file (`DBD_MSA_fixed.fasta`) into the new folder.

```
Bash
cd ../../ && mkdir Phylogeny && cd Phylogeny && cp
/home/zero/Videos/Sample_RFX1/D.C/MSA/DBD_MSA_fixed.fasta
/home/zero/Videos/Sample_RFX1/Phylogeny
```

2.5 Phylogenetic Tree Construction.

Step 1: Now I **replaces spaces with underscores (_)** in the file `DBD_MSA_fixed.fasta` and saves the modified version as `DBD_MSA_fixed_no_spaces.fasta`.

```
Bash
sed 's/ /_/g' DBD_MSA_fixed.fasta > DBD_MSA_fixed_no_spaces.fasta
```

Step 2: Now I **build a phylogenetic tree** from a multiple sequence alignment (`DBD_MSA_fixed_no_spaces.fasta`) using **FastTree** and saves the tree in **Newick format** (`DBD_phylogenetic_tree.nwk`).

```
Bash
FastTree -gamma -nome DBD_MSA_fixed_no_spaces.fasta >
DBD_phylogenetic_tree.nwk
```

Explanation:

1. `FastTree`
   - **FastTree** is a software tool that **constructs approximately-maximum-likelihood phylogenetic trees** from multiple sequence alignments (MSA).
   - It is **optimized for large datasets** and faster than traditional methods like **RAxML** or **IQ-TREE**.
2. `-gamma`
   - This **enables gamma-distributed rate variation** across sites.
   - Different positions in a sequence evolve at different rates (some change faster than others).
   - Using a **gamma model** improves accuracy by allowing sites to evolve at different rates.
3. `DBD_MSA_fixed_no_spaces.fasta`
   - This is the **input file**, a FASTA file containing aligned sequences **without spaces** in sequence names.
   - Spaces in sequence names can cause issues in phylogenetic software, so they are usually removed beforehand.
4. `> DBD_phylogenetic_tree.nwk`
   - Redirects (>) the output **phylogenetic tree** to a file in **Newick format** (`.nwk`).
   - **Newick format** is a standard format for storing phylogenetic trees, used in visualization tools like:
     - FigTree
     - iTOL (Interactive Tree of Life)
     - MEGA

Step3: Now I analyzes a **multiple sequence alignment (MSA)** to detect **mutations** and classify them based on **biochemical properties**, using my script.

Python Script: `Detect_mutation.py`

`Link:`
https://github.com/dronak-cyber/Sample/blob/main/Sample_RFX1/Phylogeny/Detect_mutation.py

Detailed Line-by-Line Explanation of `Detect_mutation.py`

This script analyzes a **multiple sequence alignment (MSA)** to detect **mutations** and classify them based on **biochemical properties**.

1️⃣ Import Required Libraries

```
python
from Bio import AlignIO
from collections import Counter
```

- **`Bio.AlignIO`**: Part of Biopython, used to read alignment files in **FASTA** or other formats.
- **`Counter`**: A dictionary subclass that helps count occurrences of elements.

2️⃣ Load the Multiple Sequence Alignment (MSA) File

```
python
msa_file = "DBD_MSA_fixed.fasta"
alignment = AlignIO.read(msa_file, "fasta")
```

- **`msa_file`**: Specifies the **input FASTA file** containing aligned protein sequences.
- **`AlignIO.read(msa_file, "fasta")`**: Reads the MSA into an **alignment object**.

3️⃣ Get Sequence Length

```
python
seq_length = alignment.get_alignment_length()
```

- **`alignment.get_alignment_length()`**: Returns the **number of columns (alignment positions)** in the MSA.

4️⃣ Compute Consensus Sequence

```
python
consensus_seq = ""
for i in range(seq_length):
    column = alignment[:, i]  # Extract residues at position i
    freq = Counter(column)     # Count occurrences of each residue
    most_common_residue, _ = freq.most_common(1)[0]  # Get most common
residue
    consensus_seq += most_common_residue
```

Step-by-Step

- Iterates through **each column (position)** in the alignment.
- Extracts the **residues at that position**.
- Uses `Counter` to find the **most frequent residue** (consensus).
- Builds the **consensus sequence**.

5️⃣ Initialize Data Structures

```
python
conserved = []
variable_sites = {}
classified_mutations = []
```

- **`conserved`**: Stores **fully conserved** residues.
- **`variable_sites`**: Stores **mutation sites**.
- **`classified_mutations`**: Stores **mutation types**.

6️⃣ Define Amino Acid Properties

```
python
bulky_residues = {"W": "Tryptophan (Bulky, Aromatic,
Hydrophobic)", ... }
positive_residues = {"K": "Lysine (Positively charged, Basic,
Hydrophilic)", ... }
negative_residues = {"D": "Aspartic Acid (Negatively charged, Acidic,
Hydrophilic)", ... }
polar_residues = {"S": "Serine (Polar, Uncharged, Hydrophilic)", ... }
```

- **Categorizes amino acids** based on size, charge, and polarity.
- Helps classify mutations based on **biochemical changes**.

7️⃣ Analyze Each Alignment Position

```
python
for i in range(seq_length):
    column = alignment[:, i]  # Extract residues at position i
    freq = Counter(column)     # Count occurrences of each residue
    consensus_residue = consensus_seq[i]
```

- Iterates through **each position**.
- Extracts residues and their **frequency distribution**.
- Stores the **consensus residue** at that position.

8️⃣ Identify Conserved and Variable Sites

```
python
if len(freq) == 1:  # All sequences have the same residue
    conserved.append(consensus_residue)
else:  # Variable site (mutation)
```

```
    variable_sites[i + 1] = dict(freq)  # Store site & mutations
```

- **Conserved Sites**: If all sequences have the **same** residue at a position, it is **fully conserved**.
- **Variable Sites**: If different residues exist, the position is **mutated**.

9️⃣ Classify Mutations

```
python
for residue in freq.keys():
    if residue != consensus_residue:
        mutation_type = "General"

        if consensus_residue in bulky_residues and residue not in
bulky_residues:
            mutation_type = "Steric Clash"
        elif consensus_residue in positive_residues and residue in
negative_residues:
            mutation_type = "Electrostatic Repulsion"
        elif consensus_residue in negative_residues and residue in
positive_residues:
            mutation_type = "Electrostatic Repulsion"
        elif consensus_residue in polar_residues and residue not in
polar_residues:
            mutation_type = "Unpaired Polar Atom"

        classified_mutations.append(f"Position {i+1}:
{consensus_residue} -> {residue} ({mutation_type})")
```

- **Identifies the type of mutation**:
  - **Steric Clash**: Loss of a **bulky residue**.
  - **Electrostatic Repulsion**: **Charge swap** (e.g., positive to negative).
  - **Unpaired Polar Atom**: Loss of **hydrophilic properties**.

🔹 Output Results

```
python
print("\n✅ Conserved Residues:")
print("".join(conserved))
```

- **Displays fully conserved residues**.

```
python
print("\n🔬 Variable Residues (Mutations Across Species):")
for pos, residues in variable_sites.items():
    print(f"Position {pos}: {residues}")
```

- **Prints mutation sites**.

```
python
CopyEdit
print("\n🔍 Classified Mutations:")
for mutation in classified_mutations:
    print(mutation)
```

- **Prints classified mutations with type**.

🔹 Save Mutation Data

```
python
with open("mutation_list.txt", "w") as f:
    f.write("Position\tMutation\tType\n")
    for mutation in classified_mutations:
        f.write(mutation + "\n")
```

- **Saves mutation list to `mutation_list.txt`**.

```
python
print("\n📄 Mutation analysis saved to mutation_list.txt")
```

- **Confirmation message**.

Run the Script

```
bash
python3 Detect_mutation.py
```

- **Ensure you have Biopython installed** (`pip install biopython`).
- The script will read **`DBD_MSA_fixed.fasta`**, analyze mutations, and print results.

Step 4: Now I **maps mutations onto a phylogenetic tree** and **saves the annotated tree as an image**. It uses the `ete3` library for tree visualization, by using my script.
Python Script: `Map_mutation.py`
`Link:`
https://github.com/dronak-cyber/Sample/blob/main/Sample_RFX1/Phylogeny/Map_mutation.py

Detailed Line-by-Line Explanation of `Map_mutation.py`

1️⃣ Import Required Libraries

```
python
from ete3 import Tree, TextFace
```

- **`ete3.Tree`**: Loads and manipulates phylogenetic trees.
- **`TextFace`**: Adds text labels (mutation annotations) to tree nodes.

2️⃣ Load the Phylogenetic Tree

```
python
tree_file = "DBD_phylogenetic_tree.nwk"
tree = Tree(tree_file, format=1)
```

- **`DBD_phylogenetic_tree.nwk`**: The input phylogenetic tree in **Newick format**.
- **`Tree(tree_file, format=1)`**: Loads the tree with **format=1** (ensures correct parsing of branch lengths).

3️⃣ Load Mutation Data

```
python
mutation_file = "mutation_list.txt"
mutations = {}

with open(mutation_file, "r") as f:
    for line in f:
        if line.startswith("##") or line.strip() == "" or "Mutation
Type" in line:
            continue  # Skip headers and empty lines
        parts = line.strip().split("\t")
        if len(parts) >= 2:
            position = parts[1]  # Mutation position
            mutation_desc = parts[2]  # Mutation details
            mutations[position] = mutation_desc
```

What This Does

- Opens `mutation_list.txt`, which contains **mutation information**.
- **Skips** headers, empty lines, and comments (##).
- Extracts **mutation position** and **mutation type**.
- Stores this data in the `mutations` dictionary:

```
arduino
CopyEdit
{ "25": "Electrostatic Repulsion", "42": "Steric Clash", ... }
```

4️⃣ Annotate the Phylogenetic Tree

```
python
for leaf in tree.iter_leaves():
    species = leaf.name  # Extract species ID
```

```
    if species in mutations:
        mutation_text = mutations[species]
        leaf.add_face(TextFace(f" {mutation_text}", fsize=10,
fgcolor="red"), column=0)
```

Step-by-Step

- Iterates through **each leaf (species)** in the tree.
- Extracts the **species name** (leaf node name).
- If the species has mutations in `mutations`, it:
  - Adds a **text annotation** next to the species.
  - Displays **mutation type** in **red** (`fgcolor="red"`).

5️⃣ Save the Annotated Tree as an Image

```
python
tree.render("mutation_tree.png", w=800, units="px")
print("📄 Mutation tree saved as mutation_tree.png")
```

- Saves the annotated tree as **`mutation_tree.png`**.
- Sets the **width** to **800 pixels** for clarity.
- Prints a confirmation message.

🚀 How to Execute This Script

1️⃣ Install Required Library (`ete3`)

```
bash
pip install ete3
```

2️⃣ Run the Script

```
bash
python3 Map_mutation.py
```

Step 5: Now I **uses PyMOL to visualize mutations on a transcription factor (TF) structure**. It loads a **PDB file**, highlights mutation sites in **red**, and saves an **annotated image and modified PDB file**, by using my script.

First Download PDB file using this command:

```
Bash
wget https://files.rcsb.org/download/1DP7.pdb -O TF_structure.pdb
```

Second use the script

Python Sript: `Visualize_mutation_pymol.py`
`Link:`
https://github.com/dronak-cyber/Sample/blob/main/Sample_RFX1/Phylogeny/Visualize_mutation_pymol.py

Explanation of Visualize_mutation_pymol.py (Mutation Visualization in PyMOL).
This script **uses PyMOL to visualize mutations on a transcription factor (TF) structure**. It loads a **PDB file**, highlights mutation sites in **red**, and saves an **annotated image and modified PDB file**.

1️⃣ Import Required Modules

```
python
import sys
sys.path.append("/home/zero/pymol/lib/python3.8/site-packages")  # Ensure PyMOL's
Python path is included
from pymol import cmd
```

- **`sys.path.append()`** ensures that the script can access PyMOL's Python modules.
- **`from pymol import cmd`**: Imports PyMOL's command interface (`cmd`), which is used to manipulate molecular structures.

2️⃣ Define the Function to Visualize Mutations

```
python
```

```
def visualize_mutations(pdb_file, mutation_list_file):
    # Load the TF structure
    cmd.load(pdb_file, "TF_structure")
```

- **Loads the PDB file** (`pdb_file`) into PyMOL as `TF_structure`.
- The PDB file should contain a **3D structure of the transcription factor**.

3️⃣ Debugging: Print Available Residues in PyMOL

```
python
print("🔍 Available Residue Numbers in PDB:")
cmd.iterate("all", "print(resi)")
```

- **Prints all residue numbers** in the loaded PDB file.
- Helps in debugging if mutation positions **don't match the PDB numbering**.

4️⃣ Extract Mutation Sites from `mutation_list.txt`

```
python
mutation_sites = []
with open(mutation_list_file, "r") as f:
    for line in f.readlines()[1:]:  # Skip header
        line = line.strip()
        if line.startswith("Position"):
            parts = line.split(":")  # Extract the part after "Position"
            if len(parts) > 1:
                pos = parts[0].replace("Position", "").strip()  # Extract number
                if pos.isdigit():  # Ensure it's a valid number
                    mutation_sites.append(pos)
```

Step-by-Step

- Opens **mutation list file (`mutation_list.txt`)**.
- **Skips the header** and extracts **mutation positions**.
- Ensures extracted positions are **valid residue numbers**.

5️⃣ Handle Missing Mutation Sites

```
python
if not mutation_sites:
    print("⚠️ No valid mutation sites found! Please check mutation_sites.txt format.")
    return
```

- **Checks if mutations were extracted**.
- If **no valid sites** are found, prints a **warning** and exits.

6️⃣ Select and Highlight Mutations in PyMOL

```
python
for site in mutation_sites:
    print(f"🔹 Attempting to select: resi {site}")
    cmd.select(f"mutation_site_{site}", f"resi {site}")
    cmd.show("sticks", f"mutation_site_{site}")
    cmd.color("red", f"mutation_site_{site}")
```

What This Does

- **Loops through each mutation site**.
- **Selects** the residue in PyMOL (`resi {site}`).
- **Changes the representation** to **sticks**.
- **Colors the residue red** to highlight the mutation.

7️⃣ Zoom Into Mutations

```
python
cmd.zoom("mutation_site_*")
```

- Focuses the **camera view** on all mutation sites.

8️⃣ Save Image & Modified PDB File

```
python
cmd.png("mutation_visualization.png", width=1200, height=900, dpi=300)
print("📸 Image saved as mutation_visualization.png")
```

- Saves an **annotated image** of the structure.
- **Resolution**: `1200x900 px`, `300 dpi` (high quality).

```
python
cmd.save("TF_structure_mutations.pdb")
print("📄 Modified PDB saved as TF_structure_mutations.pdb")
```

- Saves the **modified PDB file** with **highlighted mutations**.

9️⃣ Final Message

```
python
print("✅ Mutations visualized in PyMOL!")
print("📌 Save the image manually or export the PDB from PyMOL if needed.")
```

- Informs the user that the **mutation visualization is complete**.

🚀 How to Run This Script

1️⃣ Ensure PyMOL is Installed

```
bash
pip install pymol-open-source
```

2️⃣ Run the Script

```
bash
python3 Visualize_mutation_pymol.py
```