GENETIC MUTATION PREDICTION BAED ON ML

Overview

Genetic mutations are a central focus in bioinformatics, as they are key to understanding genetic diseases and disorders. Mutations, including deletions, insertions, and replacements, as well as subtypes like frameshift, silent, missense, and nonsense mutations, require precise identification and classification. However, the limited availability of labeled datasets often makes this task challenging.

This project addresses this gap by utilizing a dataset derived from the SARS-CoV-2 genome (NC_045512.2) published on the NCBI GeneBank. The dataset includes simulated mutations to classify genetic sequences. The goal is to determine if a DNA sequence is mutated and categorize the mutation type, whether at the primary level (e.g., deletion, insertion) or the subtype level (e.g., frameshift, silent).

By analyzing this dataset, the project aims to contribute to the understanding of mutation effects and support advancements in diagnostic and therapeutic solutions. Through machine learning and data analysis, this study provides a structured approach to mutation classification and its implications.

Significance of the Study

Understanding genetic mutations is not only critical for decoding the mechanisms of diseases but also for designing targeted treatments. Accurate classification of mutations allows researchers to explore patterns and connections between specific mutations and health outcomes. This deeper understanding could aid in predicting the potential risks associated with certain mutations.

Additionally, such research lays the groundwork for developing innovative therapies that address the root causes of genetic disorders. By identifying these key patterns, scientists can design precision treatments tailored to individual patients, ultimately improving outcomes and advancing the field of modern medicine. These advancements hold promise for reducing the burden of genetic diseases on both individuals and healthcare systems.

project dataset

- The dataset used for this project is designed to analyze and identify genetic mutations in DNA sequences. It contains detailed information about the changes occurring at specific positions within the DNA sequence.

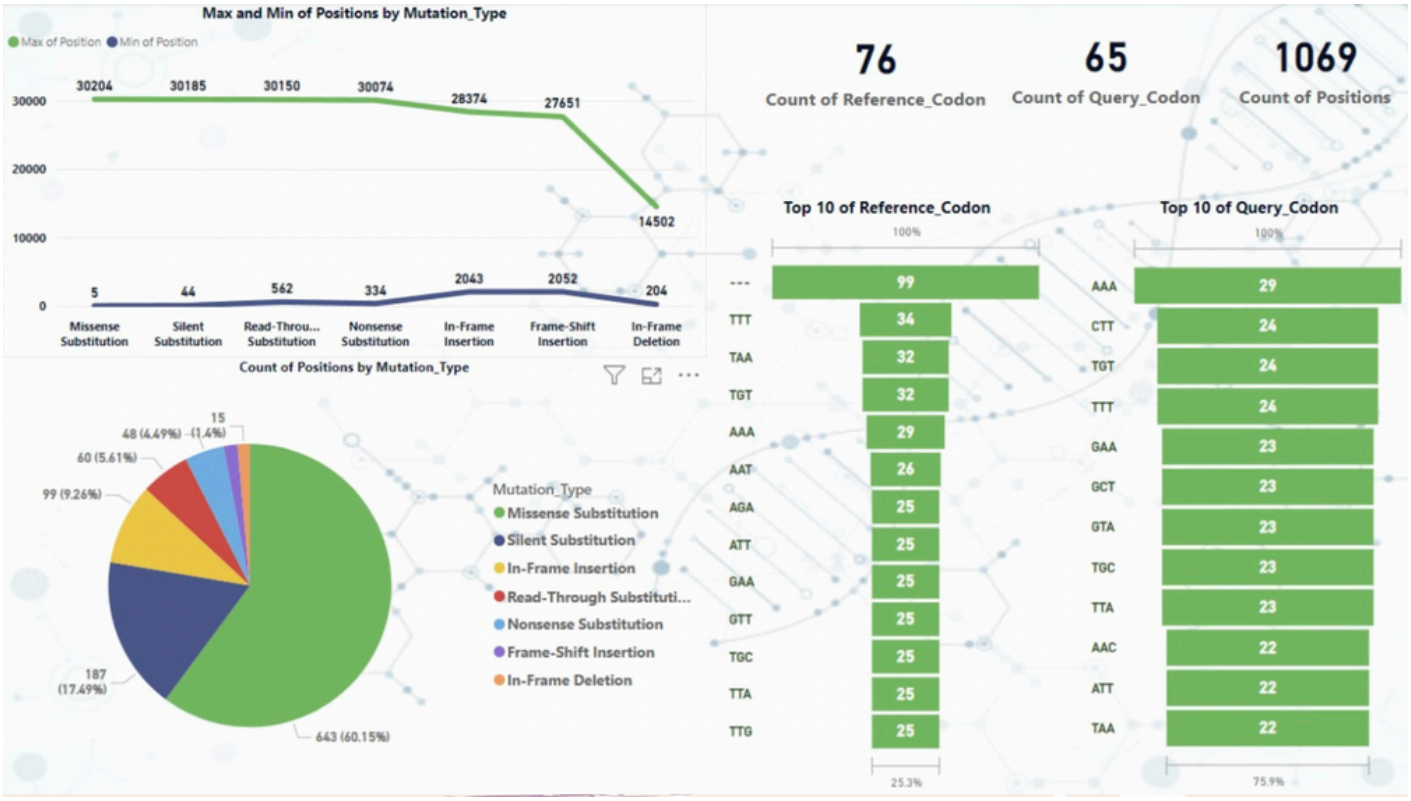
- The dataset consists of 1069 rows and 4 columns, with each row representing a mutation instance. The columns are:
- Position: The exact location of the mutation within the DNA sequence.
- Reference_Codon: The codon (three DNA bases) at the specified position in the reference sequence.
- Query_Codon: The observed codon at the same position in the analyzed sequence.
- Mutation_Type: The type of mutation identified, such as substitution, insertion, or deletion.

- Mutation Breakdown:
- Primary Types:
  - Deletion (30% of the dataset).
  - Insertion (35% of the dataset).
  - Replacement (35% of the dataset).

- Subtypes:
- Frameshift and in-frame mutations for deletion and insertion.
- Silent, missense, nonsense, and read-through mutations for replacements.

- Purpose:
- Designed for training machine learning models to classify and analyze mutations.
- Simulates real-world mutation scenarios to support research and tool development.

- Reliability:
- Based on a well-documented and curated genome sequence.
- Ensures high-quality data for reproducible and impactful research.



Dashboard Insights

- Mutation Distribution:
  - Visualizes the count and positions of different mutation types.
  - Highlights dominant types like "In-Frame Insertion" (60.15%) and "Frame-Shift Insertion" (17.49%).
- Codon Analysis:
  - Displays the top 10 reference and query codons (e.g., "AAA" and "CTT").
  - Analyzes their frequency and significance in the dataset.
- Summary Metrics:
  - Total positions analyzed: 1069.
  - Count of reference codons: 76.
  - Count of query codons: 65.

- The dashboard provides a clear summary of mutation trends, helping to prioritize significant mutation types.
- Codon patterns and mutation distributions directly support the classification tasks.
- These insights enhance model training and improve the understanding of mutation effects, aligning with the project's goal of advancing diagnostic tools.

```python
# Importing necessary libraries

from sklearn.preprocessing import LabelEncoder,OrdinalEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import BaggingClassifier
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sn
```

Tools and Techniques

- Python Libraries:
  - NumPy: Used for numerical operations, such as matrix calculations and data preprocessing.
  - Pandas: Essential for handling and analyzing tabular data, enabling operations like filtering, grouping, and transformation of the dataset.
  - Matplotlib & Seaborn: For creating visualizations like heatmaps and scatter plots to highlight patterns in the mutation data.
- Machine Learning Frameworks:
  - Scikit-learn:
    - Implements classification models (e.g., Random Forest, Logistic Regression, SVM).
    - Includes tools like train-test splitting, cross-validation, and hyperparameter tuning.
    - Feature Selection: RFECV is used to identify the most important features.
- Data Visualization:
  - Tools like heatmaps and bar charts are employed to illustrate the relationship between mutations and their positions, making insights more accessible.
- Bioinformatics Tools:
  - Genome mapping and codon analysis are integrated to ensure the biological relevance of findings.
- Purpose of Techniques:
  - These tools streamline data preprocessing, enhance model accuracy, and provide clear insights into mutation trends, directly supporting the project's objectives.

Data Preprocessing:

Data preprocessing is an important step to ensure that the dataset is ready for machine learning models. It involves transforming and cleaning the data to make it suitable for training. In this project, two key preprocessing steps were applied: converting categorical data to numerical values and identifying outliers.

1. Encoding Categorical Data:

We transformed text-based columns into numerical values using LabelEncoder to make them compatible with machine learning models.

```
from sklearn.preprocessing import LabelEncoder

# Convert categorical columns to numerical
lab = LabelEncoder()
for column in data.select_dtypes(include='object').columns.values:
    data[column] = lab.fit_transform(data[column])

# Display the updated dataset
print(data)
```

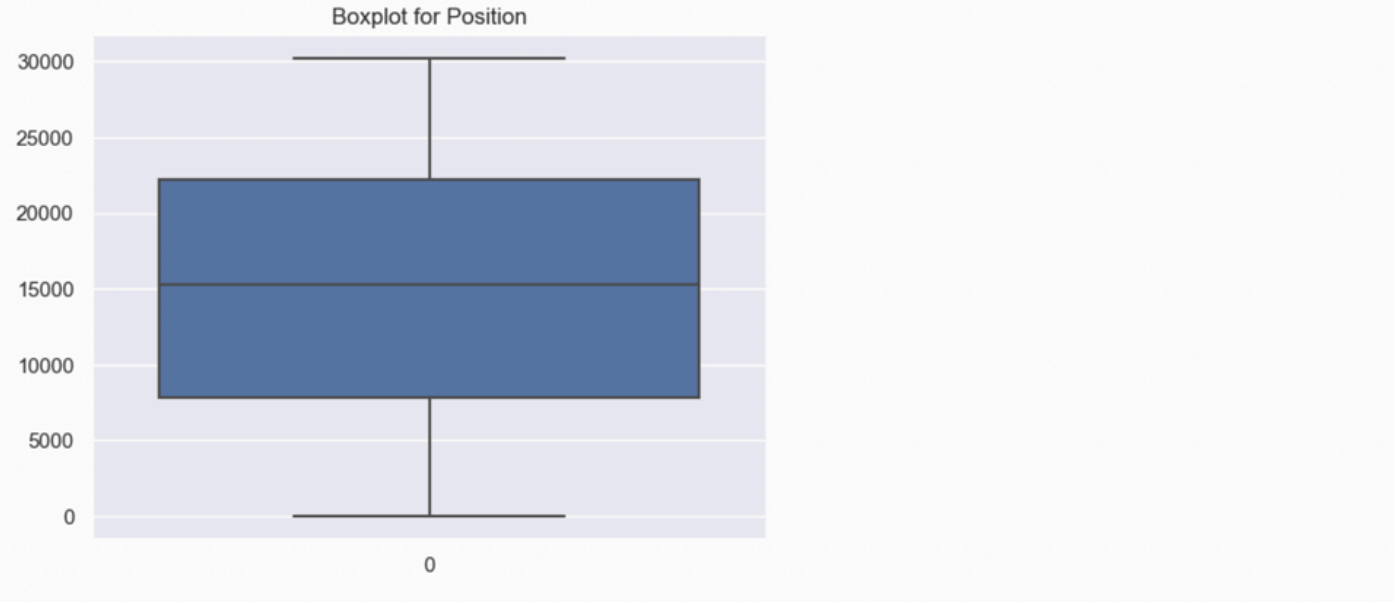
```
      Position  Reference_Codon  Query_Codon  Mutation_Type
0            5                7            2              3
1           44               65           53              7
2           81               21           62              3
3           99               33           43              3
4          186               17           27              7
...        ...              ...          ...            ...
1064     30153               68           10              3
1065     30165               55           63              3
1066     30181               66           52              3
1067     30185               60           49              7
1068     30204                7           17              3

[1069 rows x 4 columns]
```

- What?: LabelEncoder replaces text values in columns (e.g., Reference_Codon, Query_Codon, Mutation_Type) with unique integers.
- Why?: Models cannot process text data directly. Encoding converts text into numbers that models can use.
- How?: For every categorical column, we applied the encoder to assign a number to each unique text value.
- Outcome: All text data in the dataset was successfully converted into numerical format.

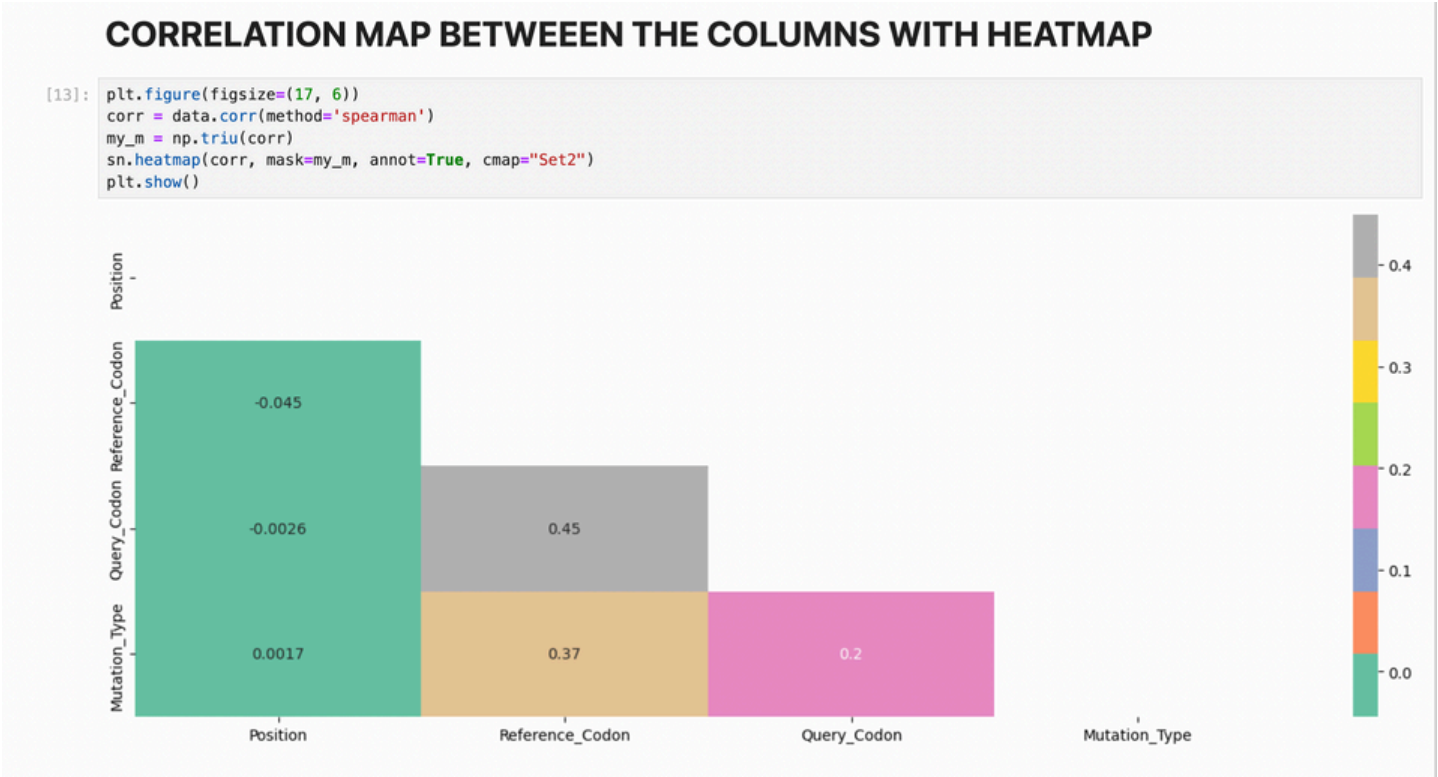
2. Detecting Outliers

We used boxplots to visualize and detect extreme values (outliers) in the numeric columns.

```
# Exploratory Data Analysis (EDA) and Visualization
# Boxplot for each column
numeric_columns = data.select_dtypes(include=['number']).columns
for col in numeric_columns:
    sn.boxplot(data[col])
    plt.title(f'Boxplot for {col}')
    plt.show()
```



- What?: Boxplots highlight the range of normal data values and show outliers as points outside the main range.
- Why?: Outliers can negatively impact the accuracy of machine learning models, so they need to be identified.
- How?: For each column, we plotted a boxplot to visualize the distribution and find values outside the range.
- Outcome: We identified potential outliers that may require special handling or removal.



Correlation Map Between the Columns with Heatmap

- A correlation map (heatmap) helps us understand the relationships between different columns in the dataset. It shows how much one column is related to another using values between -1 and 1. This is useful for identifying patterns in the data.

Purpose:

The heatmap helps us:

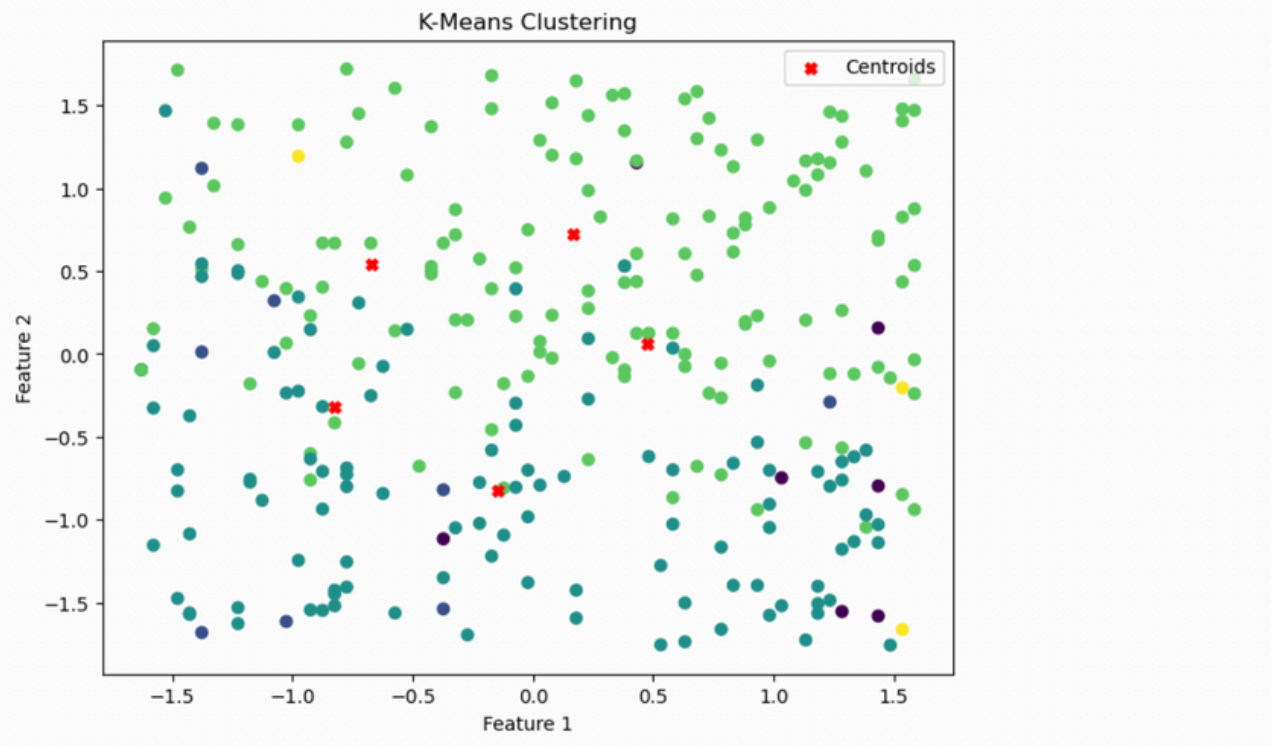
1. Identify if any columns are strongly or weakly correlated.
2. Understand how columns might affect the target variable (Mutation_Type in this case).
3. Decide which features are more important for the model.

The heatmap highlights how different columns in the dataset are related. This helps in feature selection by identifying columns with strong or weak correlations, which can affect the model's performance.

K-Means Clustering

```
# Assuming 'data' is your DataFrame with features and target
X = data[['Reference_Codon', 'Query_Codon', 'Position']]  # Use your desired features
y = data['Mutation_Type']  # Assuming 'Mutation_Type' is the target variable

# One-hot encoding 'Reference_Codon' if needed
X = pd.get_dummies(X, columns=['Reference_Codon'])

# Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)

# Scaling the features using StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Applying K-Means Clustering with 5 clusters
kmeans = KMeans(n_clusters=5, random_state=42)
kmeans.fit(X_train_scaled)

# Predicting the cluster labels for the test set
y_pred = kmeans.predict(X_test_scaled)

# Calculating the Silhouette Score to evaluate clustering quality
silhouette_avg = silhouette_score(X_test_scaled, y_pred)
print(f"Silhouette Score: {silhouette_avg:.3f}")

# Plotting the clusters and centroids
plt.figure(figsize=(8, 6))
plt.scatter(X_test_scaled[:, 0], X_test_scaled[:, 1], c=y_pred, cmap='viridis', marker='o')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], color='red', marker='X', label='Centroids')
plt.title('K-Means Clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.show()
```

```
Silhouette Score: 0.023
```

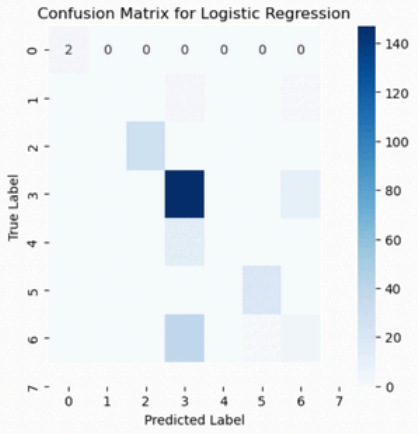


This graph shows the results of K-Means clustering applied to the dataset, using features such as 'Reference_Codon', 'Query_Codon', and 'Position'. The scatter plot visualizes how the data points are grouped into clusters, with centroids marked in red, and the Silhouette Score is calculated to assess the clustering quality.

Logistic Regression Classifier

```python
# Logistic Regression

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.preprocessing import StandardScaler
import seaborn as sn
import matplotlib.pyplot as plt

# Preparing the data
X = data[['Reference_Codon', 'Position']]
y = data['Mutation_Type']

X = pd.get_dummies(X, columns=['Reference_Codon'])

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

# Scaling the features using StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Initializing and training the Logistic Regression model
lr = LogisticRegression(max_iter=500)
lr.fit(X_train, y_train)

# Making predictions
y_pred_lr = lr.predict(X_test)

# Evaluating the model's accuracy
lr_accuracy = accuracy_score(y_test, y_pred_lr)
print(f"Logistic Regression Accuracy: {lr_accuracy:.2f}")

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred_lr)
print("Confusion Matrix for Logistic Regression:\n", conf_matrix)

# Displaying True and Predicted Mutations
results = pd.DataFrame({
    'True Mutation': y_test,
    'Predicted Mutation': y_pred_lr
})
print(results.head())

# Plotting the confusion matrix
plt.figure(figsize=(5, 5))
sn.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=np.unique(y), yticklabels=np.unique(y))
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix for Logistic Regression')
plt.show()

# Classification report (precision, recall, f1-score, support)
print("\nClassification Report:")
print(classification_report(y_test, y_pred_lr, zero_division=1))
```

```
Logistic Regression Accuracy: 0.75
Confusion Matrix for Logistic Regression:
 [[  2   0   0   0   0   0   0]
 [  0   0   0   2   0   0   1]
 [  0   0  29   0   0   0   0]
 [  0   0   0 147   0   0  11]
 [  0   0   0  13   0   0   0]
 [  0   0   0   0   0  20   0]
 [  0   0   0  38   0   1   4]]
     True Mutation  Predicted Mutation
708              3                   3
215              7                   3
882              3                   3
88               4                   3
842              3                   3
```

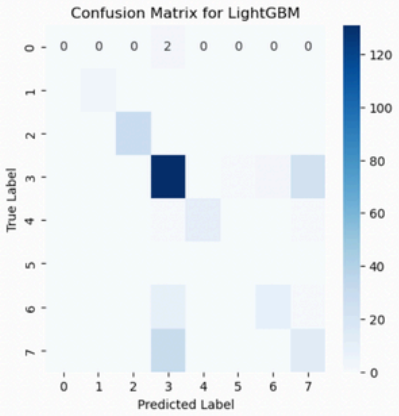



```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         2
           1       1.00      0.00      0.00         3
           2       1.00      1.00      1.00        29
           3       0.73      0.93      0.82       158
           4       1.00      0.00      0.00        13
           6       0.95      1.00      0.98        20
           7       0.25      0.09      0.14        43

    accuracy                           0.75       268
   macro avg       0.85      0.57      0.56       268
weighted avg       0.72      0.75      0.69       268
```

LightGBM Classifier

```python
# LightGBM Classifier

from lightgbm import LGBMClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import seaborn as sn
import matplotlib.pyplot as plt

# Preparing the data
X = data[['Reference_Codon', 'Query_Codon', 'Position']]
y = data['Mutation_Type']

X = pd.get_dummies(X, columns=['Reference_Codon'])

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

# Scaling the features using StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Initializing and training the LightGBM model
lgbm = LGBMClassifier(n_estimators=100, learning_rate=0.1, random_state=42)
lgbm.fit(X_train, y_train)

# Predicting the test set results
y_pred_lgbm = lgbm.predict(X_test)

# Evaluating the model's accuracy
lgbm_accuracy = accuracy_score(y_test, y_pred_lgbm)
print(f"LightGBM Accuracy: {lgbm_accuracy:.2f}")

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred_lgbm)
print("Confusion Matrix for LightGBM:\n", conf_matrix)

# Displaying True and Predicted Mutations
results = pd.DataFrame({
    'True Mutation': y_test,
    'Predicted Mutation': y_pred_lgbm
})
print(results.head())

# Plotting the confusion matrix
plt.figure(figsize=(5, 5))
sn.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=np.unique(y), yticklabels=np.unique(y))
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix for LightGBM')
plt.show()

# Classification report (precision, recall, f1-score, support)
print("\nClassification Report:")
print(classification_report(y_test, y_pred_lgbm, zero_division=1))
```

```
LightGBM Accuracy: 0.73
Confusion Matrix for LightGBM:
 [[  0   0   0   2   0   0   0   0]
 [  0   3   0   0   0   0   0   0]
 [  0   0  29   0   0   0   0   0]
 [  0   0   0 131   0   1   2  24]
 [  0   0   0   1  11   0   0   1]
 [  0   0   0   0   0   0   0   0]
 [  0   0   0  10   0   0   9   1]
 [  0   0   0  30   0   0   0  13]]
     True Mutation  Predicted Mutation
708              3                   3
215              7                   3
882              3                   3
88               4                   4
842              3                   3
```

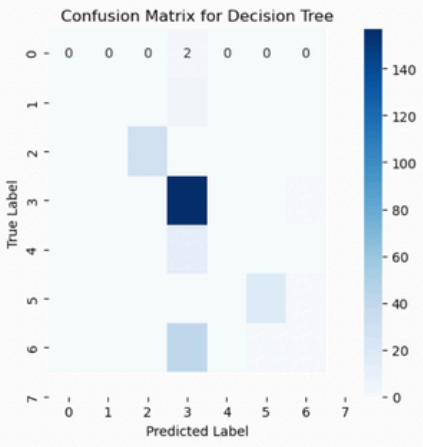



```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.00      0.00         2
           1       1.00      1.00      1.00         3
           2       1.00      1.00      1.00        29
           3       0.75      0.83      0.79       158
           4       1.00      0.85      0.92        13
           5       0.00      1.00      0.00         0
           6       0.82      0.45      0.58        20
           7       0.33      0.30      0.32        43

    accuracy                           0.73       268
   macro avg       0.74      0.68      0.58       268
weighted avg       0.73      0.73      0.72       268
```

Decision Tree Classifier

```python
# Decision Tree Classifier

from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import seaborn as sn
import matplotlib.pyplot as plt

# Preparing the data
X = data[['Reference_Codon', 'Position']]
y = data['Mutation_Type']

X = pd.get_dummies(X, columns=['Reference_Codon'])

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

# Initializing and training the Decision Tree model
dt = DecisionTreeClassifier(max_depth=5)
dt.fit(X_train, y_train)

# Making predictions
y_pred_dt = dt.predict(X_test)

# Evaluating the model's accuracy
dt_accuracy = accuracy_score(y_test, y_pred_dt)
print(f"Decision Tree Accuracy: {dt_accuracy:.2f}")

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred_dt)
print("Confusion Matrix for Decision Tree:\n", conf_matrix)

# Displaying True and Predicted Mutations
results = pd.DataFrame({
    'True Mutation': y_test,
    'Predicted Mutation': y_pred_dt
})
print(results.head())

# Plotting the confusion matrix
plt.figure(figsize=(5, 5))
sn.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=np.unique(y), yticklabels=np.unique(y))
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix for Decision Tree')
plt.show()

# Classification report (precision, recall, f1-score, support)
print("\nClassification Report:")
print(classification_report(y_test, y_pred_dt, zero_division=1))
```

```
Decision Tree Accuracy: 0.77
Confusion Matrix for Decision Tree:
 [[  0   0   0   2   0   0   0]
 [  0   0   0   3   0   0   0]
 [  0   0  29   0   0   0   0]
 [  0   0   0 157   0   0   1]
 [  0   0   0  13   0   0   0]
 [  0   0   0   0   0  19   1]
 [  0   0   0  41   0   1   1]]
     True Mutation  Predicted Mutation
708              3                   3
215              7                   3
882              3                   3
88               4                   3
842              3                   3
```

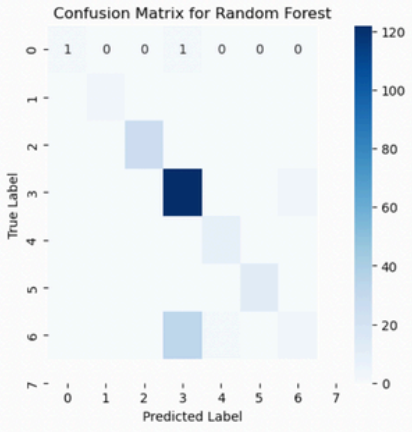



```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.00      0.00         2
           1       1.00      0.00      0.00         3
           2       1.00      1.00      1.00        29
           3       0.73      0.99      0.84       158
           4       1.00      0.00      0.00        13
           6       0.95      0.95      0.95        20
           7       0.33      0.02      0.04        43

    accuracy                           0.77       268
   macro avg       0.86      0.42      0.40       268
weighted avg       0.73      0.77      0.68       268
```

Random Forest Classifier

```python
# Random Forest Classifier

from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.preprocessing import StandardScaler, OneHotEncoder
import matplotlib.pyplot as plt

# Preparing the data
X = data[['Reference_Codon', 'Query_Codon', 'Position']]
y = data['Mutation_Type']

# Encoding categorical features
encoder = OneHotEncoder()
X_encoded = encoder.fit_transform(X).toarray()
X = pd.DataFrame(X_encoded, columns=encoder.get_feature_names_out())

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardizing the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Initializing and training the Random Forest model
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

# Making predictions
y_pred_rf = rf.predict(X_test)

# Evaluating the model's accuracy
rf_accuracy = accuracy_score(y_test, y_pred_rf)
print(f"Random Forest Accuracy: {rf_accuracy:.2f}")

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred_rf)
print("Confusion Matrix for Random Forest:\n", conf_matrix)

# Displaying True and Predicted Mutations
results = pd.DataFrame({
    'True Mutation': y_test.reset_index(drop=True),
    'Predicted Mutation': y_pred_rf
})
print(results.head())

# Plotting the confusion matrix
plt.figure(figsize=(5, 5))
sn.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=np.unique(y), yticklabels=np.unique(y))
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix for Random Forest')
plt.show()

# Classification report (precision, recall, f1-score, support)
print("\nClassification Report:")
print(classification_report(y_test, y_pred_rf, zero_division=1))
```

```
Random Forest Accuracy: 0.82
Confusion Matrix for Random Forest:
 [[  1   0   0   1   0   0   0]
 [  0   2   0   0   0   0   0]
 [  0   0  26   0   0   0   0]
 [  0   0   0 122   0   0   4]
 [  0   0   0   0  10   0   0]
 [  0   0   0   0   0  11   0]
 [  0   0   0  33   1   0   3]]
   True Mutation  Predicted Mutation
0              3                   3
1              7                   3
2              3                   3
3              4                   4
4              3                   3
```

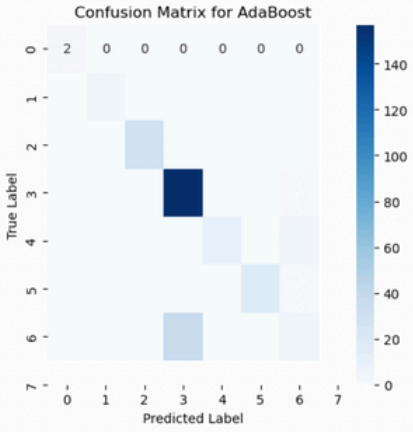



```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.50      0.67         2
           1       1.00      1.00      1.00         2
           2       1.00      1.00      1.00        26
           3       0.78      0.97      0.87       126
           4       0.91      1.00      0.95        10
           6       1.00      1.00      1.00        11
           7       0.43      0.08      0.14        37

    accuracy                           0.82       214
   macro avg       0.87      0.79      0.80       214
weighted avg       0.77      0.82      0.77       214
```

AdaBoost Classifier

```python
# AdaBoost Classifier

from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import seaborn as sn
import matplotlib.pyplot as plt

# Preparing the data
X = data[['Reference_Codon','Query_Codon', 'Position']]
y = data['Mutation_Type']

X = pd.get_dummies(X, columns=['Reference_Codon', 'Query_Codon'])

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

# Scaling the features using StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# AdaBoost Classifier with Decision Tree as base estimator
base_model = DecisionTreeClassifier(max_depth=3)
adb = AdaBoostClassifier(estimator=base_model, n_estimators=200, learning_rate=0.05, random_state=42)
adb.fit(X_train, y_train)

# Evaluating the model's accuracy
adb_accuracy = adb.score(X_test, y_test)
print(f"AdaBoost Accuracy: {adb_accuracy:.2f}")

# Predicting the test set results
predictions = adb.predict(X_test)

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, predictions)
print("Confusion Matrix for AdaBoost:\n", conf_matrix)

# Displaying True and Predicted Mutations
results = pd.DataFrame({
    'True Mutation': y_test,
    'Predicted Mutation': predictions
})
print(results.head())

# Plotting the confusion matrix
plt.figure(figsize=(5, 5))
sn.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=np.unique(y), yticklabels=np.unique(y))
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix for AdaBoost')
plt.show()

# Classification report (precision, recall, f1-score, support)
print("\nClassification Report:")
print(classification_report(y_test, predictions, zero_division=1))
```

```
AdaBoost Accuracy: 0.84
Confusion Matrix for AdaBoost:
 [[  2   0   0   0   0   0   0]
 [  0   3   0   0   0   0   0]
 [  0   0  29   0   0   0   0]
 [  0   0   0 157   0   0   1]
 [  0   0   0   0  10   0   3]
 [  0   0   0   0   0  19   1]
 [  0   0   0  37   0   0   6]]
     True Mutation  Predicted Mutation
708              3                   3
215              7                   3
882              3                   3
88               4                   4
842              3                   3
```

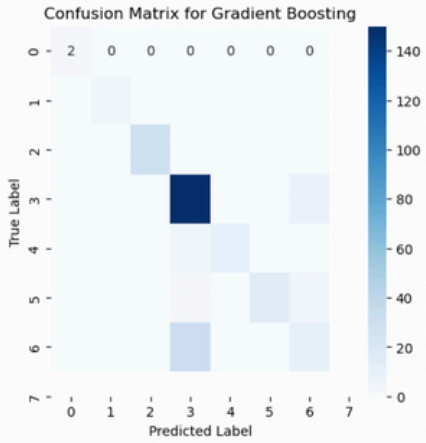



```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         2
           1       1.00      1.00      1.00         3
           2       1.00      1.00      1.00        29
           3       0.81      0.99      0.89       158
           4       1.00      0.77      0.87        13
           6       1.00      0.95      0.97        20
           7       0.55      0.14      0.22        43

    accuracy                           0.84       268
   macro avg       0.91      0.84      0.85       268
weighted avg       0.81      0.84      0.80       268
```

Gradient Boosting Classifier

```
# Gradient Boosting Classifier

from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import seaborn as sn
import matplotlib.pyplot as plt

# Preparing the data
X = data[['Reference_Codon', 'Query_Codon', 'Position']]
y = data['Mutation_Type']

X = pd.get_dummies(X, columns=['Reference_Codon'])

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

# Scaling the features using StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Initializing and training the Gradient Boosting model
gb = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1, random_state=42)
gb.fit(X_train, y_train)

# Predicting the test set results
y_pred_gb = gb.predict(X_test)

# Evaluating the model's accuracy
gb_accuracy = accuracy_score(y_test, y_pred_gb)
print(f"Gradient Boosting Accuracy: {gb_accuracy:.2f}")

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred_gb)
print("Confusion Matrix for Gradient Boosting:\n", conf_matrix)

# Displaying True and Predicted Mutations
results = pd.DataFrame({
    'True Mutation': y_test,
    'Predicted Mutation': y_pred_gb
})
print(results.head())

# Plotting the confusion matrix
plt.figure(figsize=(5, 5))
sn.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=np.unique(y), yticklabels=np.unique(y))
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix for Gradient Boosting')
plt.show()

# Classification report (precision, recall, f1-score, support)
print("\nClassification Report:")
print(classification_report(y_test, y_pred_gb, zero_division=1))
```

```
Gradient Boosting Accuracy: 0.82
Confusion Matrix for Gradient Boosting:
 [[  2   0   0   0   0   0   0]
 [  0   3   0   0   0   0   0]
 [  0   0  29   0   0   0   0]
 [  0   0   0 150   0   0   8]
 [  0   0   0   3  10   0   0]
 [  0   0   0   2   0  15   3]
 [  0   0   0  32   0   0  11]]
     True Mutation  Predicted Mutation
708              3                   3
215              7                   3
882              3                   3
88               4                   4
842              3                   3
```

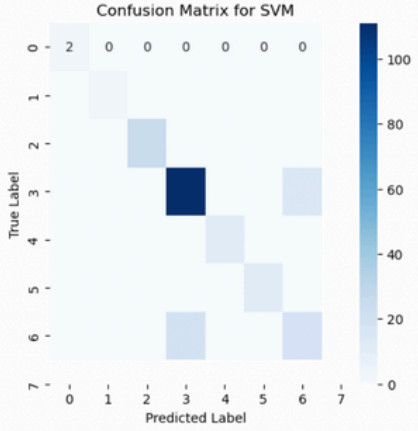



```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         2
           1       1.00      1.00      1.00         3
           2       1.00      1.00      1.00        29
           3       0.80      0.95      0.87       158
           4       1.00      0.77      0.87        13
           6       1.00      0.75      0.86        20
           7       0.50      0.26      0.34        43

    accuracy                           0.82       268
   macro avg       0.90      0.82      0.85       268
weighted avg       0.80      0.82      0.80       268
```

Support Vector Machine (SVM) Classifier

```python
# Support Vector Machine (SVM) Classifier

import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.preprocessing import StandardScaler
import seaborn as sn
import matplotlib.pyplot as plt

# Preparing the data
print(data.columns)

x = data[['Reference_Codon', 'Position']]  # Features
y = data['Mutation_Type']  # Target variable

# Splitting the data into train and test sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

# Scaling the features using StandardScaler
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)

# Initializing and training the SVM model with a linear kernel
svm = SVC(kernel='linear')
svm.fit(x_train, y_train)

# Making predictions
y_pred = svm.predict(x_test)

# Evaluating the model's accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"SVM Classifier Accuracy: {accuracy:.2f}")

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print('Confusion Matrix:\n', conf_matrix)

# Displaying True and Predicted Mutations
results = pd.DataFrame({
    'True Mutation': y_test,
    'Predicted Mutation': y_pred
})
print(results.head())

# Plotting the confusion matrix
plt.figure(figsize=(5, 5))
sn.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=np.unique(y), yticklabels=np.unique(y))
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix for SVM')
plt.show()

# Classification report (precision, recall, f1-score, support)
print("\nClassification Report:")
print(classification_report(y_test, y_pred, zero_division=1))
```

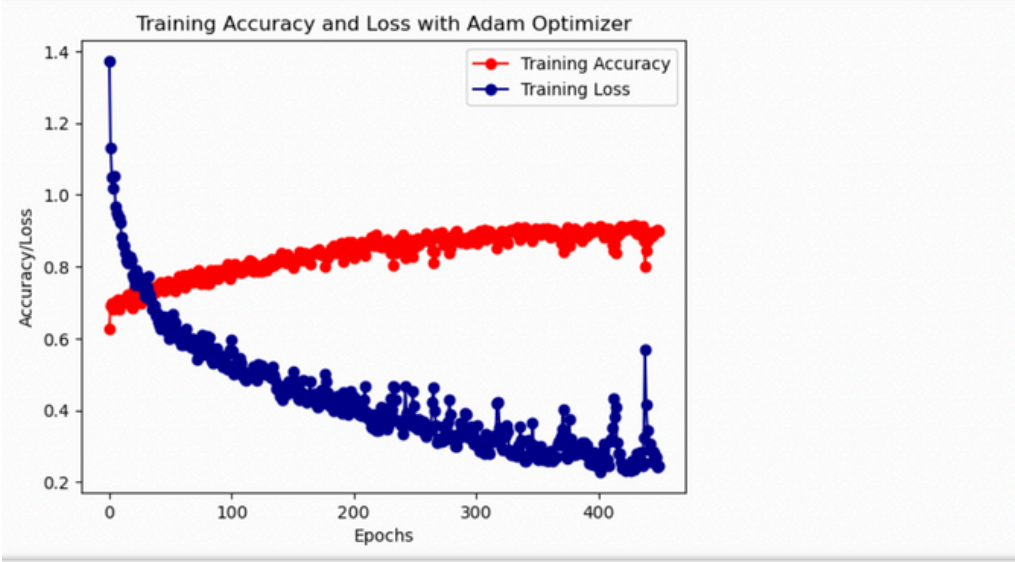
```
Index(['Position', 'Reference_Codon', 'Query_Codon', 'Mutation_Type'], dtype='object')
SVM Classifier Accuracy: 0.84
Confusion Matrix:
 [[  2   0   0   0   0   0   0]
 [  0   2   0   0   0   0   0]
 [  0   0  26   0   0   0   0]
 [  0   0   0 111   0   0  15]
 [  0   0   0   0  10   0   0]
 [  0   0   0   0   0  11   0]
 [  0   0   0  19   0   0  18]]
     True Mutation  Predicted Mutation
708              3                   3
215              7                   7
882              3                   3
88               4                   4
842              3                   3
```




```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         2
           1       1.00      1.00      1.00         2
           2       1.00      1.00      1.00        26
           3       0.85      0.88      0.87       126
           4       1.00      1.00      1.00        10
           6       1.00      1.00      1.00        11
           7       0.55      0.49      0.51        37

    accuracy                           0.84       214
   macro avg       0.91      0.91      0.91       214
weighted avg       0.84      0.84      0.84       214
```

Deep Neural Network for Mutation Classification

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Dense
import keras.activations
import keras.losses
import keras.optimizers
from sklearn.preprocessing import LabelEncoder

# Prepare Data
X = data[['Reference_Codon', 'Query_Codon']]
Y = pd.get_dummies(data['Mutation_Type'])  # One-hot encoding for the target

# Label encoding for 'Reference_Codon' and 'Query_Codon' if needed (for categorical data)
encoder = LabelEncoder()
X['Reference_Codon'] = encoder.fit_transform(X['Reference_Codon'])
X['Query_Codon'] = encoder.fit_transform(X['Query_Codon'])

# Split the data into training and testing sets
x_tran, x_tst, y_tran, y_tst = train_test_split(X, Y, test_size=0.25, random_state=42)

# Build the model
model = Sequential()

# Adding layers to the neural network
model.add(Dense(units=64, input_dim=x_tst.shape[1], activation=keras.activations.relu))
model.add(Dense(units=128, activation=keras.activations.relu))
model.add(Dense(units=256, activation=keras.activations.relu))
model.add(Dense(units=128, activation=keras.activations.relu))
model.add(Dense(units=64, activation=keras.activations.relu))

# Output layer with the same number of units as the target classes
model.add(Dense(units=Y.shape[1], activation=keras.activations.softmax))

# Compile the model
model.compile(optimizer='adam', metrics=['accuracy'], loss=keras.losses.categorical_crossentropy)

# Train the model
history = model.fit(x_tran, y_tran, batch_size=45, epochs=450, validation_data=(x_tst, y_tst))

# Save the model in Keras native format
model.save('adam_model.keras')

# Plot training history
plt.plot(history.history['accuracy'], label='Training Accuracy', marker='o', color='red')
plt.plot(history.history['loss'], label='Training Loss', marker='o', color='darkblue')
plt.title('Training Accuracy and Loss with Adam Optimizer')
plt.xlabel('Epochs')
plt.ylabel('Accuracy/Loss')
plt.legend()
plt.show()
```



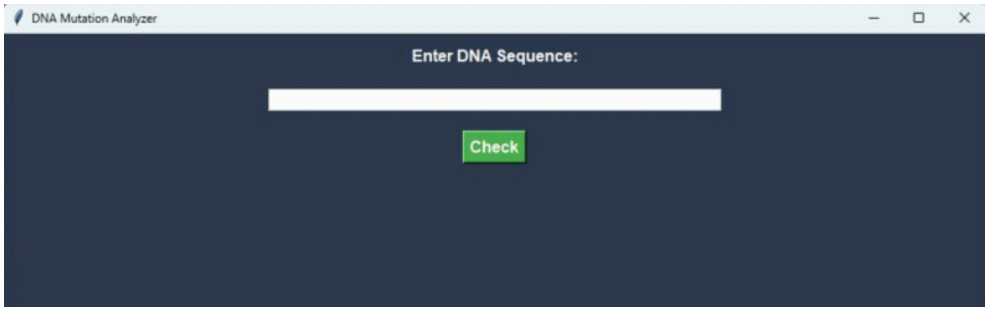
Genetic Mutation Detector for DNA Sequences

```
import tkinter as tk
from tkinter import messagebox
import numpy as np
```

```
def analyze_codon_sequence(dna_sequence, mutation_data):
    dna_sequence = dna_sequence.upper()
    mutations_detected = []


    codons = [dna_sequence[i:i+3] for i in range(0, len(dna_sequence), 3)]


    for _, row in mutation_data.iterrows():
        position = row['Position'] - 1
        reference_codon = row['Reference_Codon'].upper()
        query_codon = row['Query_Codon'].upper()
        mutation_type = row['Mutation_Type']


        if position >= len(codons):
            continue

        observed_codon = codons[position]


        if observed_codon == query_codon :
            mutation = mutation_type
            mutations_detected.append((position + 1, reference_codon, query_codon, observed_codon, mutation))

    return mutations_detected
```

```
def check_sequence():
    dna_sequence = entry.get()
    if not dna_sequence:
        messagebox.showwarning("Input Error", "Please enter a DNA sequence.")
        return
    dataset = pd.read_csv('Lable_Dataset.csv')
    mutations = analyze_codon_sequence(dna_sequence, dataset)

    if mutations:
        result = "Results of Analysis:\n"
        for mutation in mutations:
            result += (f"Position {mutation[0]}: Reference={mutation[1]}, Query={mutation[2]}, "
                       f"Observed={mutation[3]}, Status={mutation[4]}\n")
    else:
        result = "No Mutations Detected"

    messagebox.showinfo("Analysis Result", result)
```

```
root = tk.Tk()
root.title("DNA Mutation Analyzer")
root.configure(bg="#2E3B4E")


label = tk.Label(root, text="Enter DNA Sequence:", fg="#FFFFFF", bg="#2E3B4E", font=("Arial", 12, "bold"))
label.pack(pady=10)


entry = tk.Entry(root, width=50, bg="#FFFFFF", fg="#000000", font=("Arial", 12))
entry.pack(pady=10)


button = tk.Button(root, text="Check", command=check_sequence, bg="#4CAF50", fg="#FFFFFF", font=("Arial", 12, "bold"))
button.pack(pady=10)

root.mainloop()
```



Genetic Mutation Detector for DNA Sequences