BRAC UNIVERSITY

Inspiring Excellence

**ARTIFICIAL INTELLIGENCE**

**CSE422, Section: 05**

# Project Report: Mice Protein Expression.

**Date : 03/09/2025**

**Submitted by:**

| | |
|---|---|
| Name: Pretom Areefin Pranto | Name: Fareya Hossain. |
| [redacted] | [redacted] |

# Table of Contents

## Introduction

### Objective

The project investigates collected samples of various protein expressions which are derived from mice, under different circumstances. The associated data on protein expression serves as a useful indicator for researching any kind of genetic disorders and neurological problems, alongside medication side effects. Our goal is to construct a proper machine learning model, consisting of appropriate algorithms in order to classify, define and categorize the patterns from the dataset. The incentive behind this project arises from the possibility that timely detection and understanding of these patterns may significantly aid scientific studies and treatment strategies.

## Dataset Description

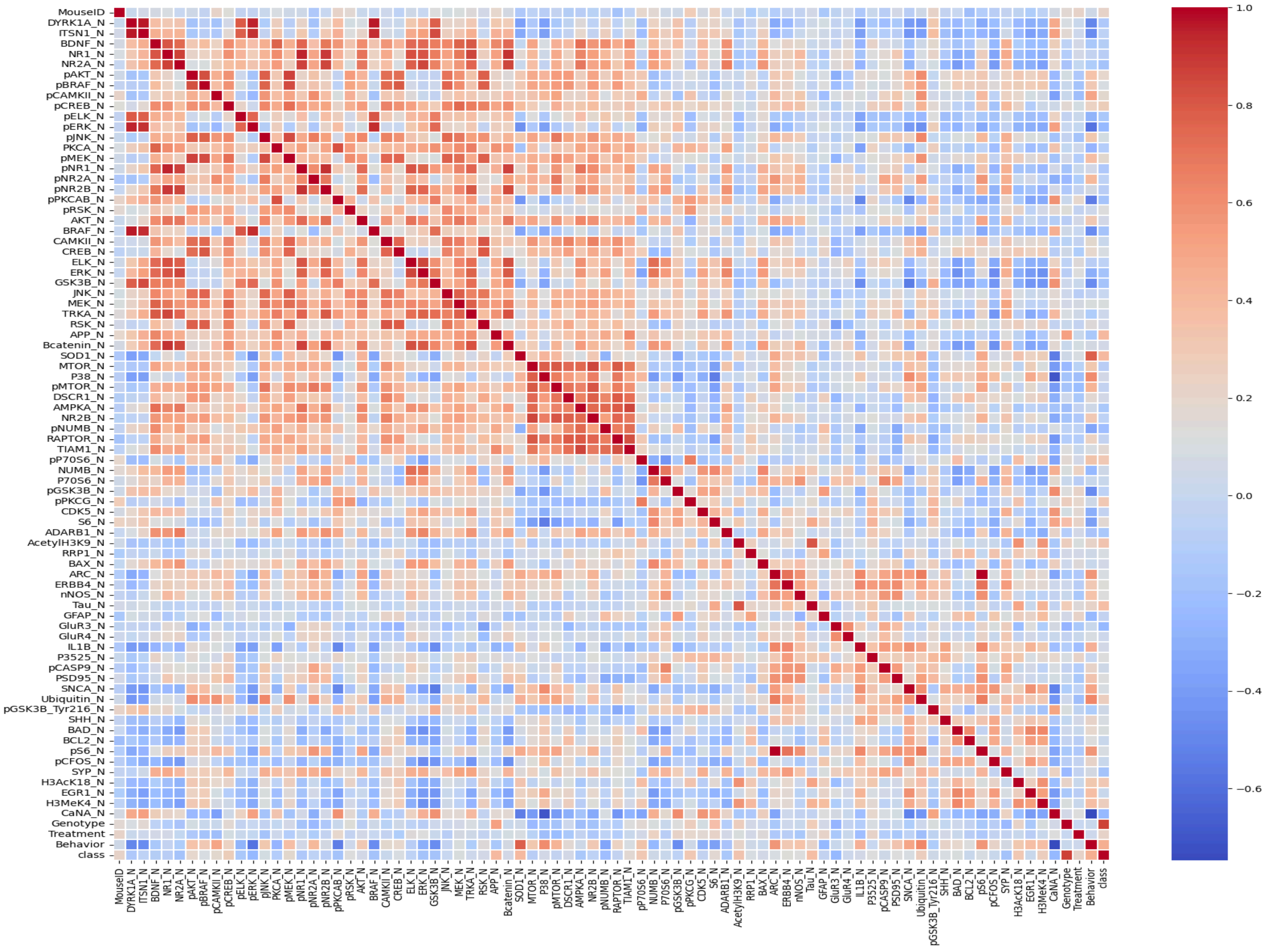
The dataset is a set of collected information regarding different protein expressions in mice. It is in a comma-separated value (csv) format. The dataset has numerous intriguing characteristics, which has been described below:

1. **The number of features:**
   The dataset contains **77 numerical protein expression features**, plus **one** identifier column, which has been labeled as Mouse ID.
2. **The type of dataset:**
   The dataset can be stated as a **classification** type of problem. We have selected a special target variable (i.e. class) for analysis and evaluation. Because the target variable is actually a type of category or label. Furthermore, the target variable is not a continuous number. Thus, it is a classification based problem. To further confirm our thought process, the dataset is not a regression problem because we are not trying to predict a number along a continuous scale, which is done primarily for regression analysis.
3. **Amount of data points:**
   In this particular dataset, there are 1050 data and 88560 points (observations).
4. **Types of features present in the dataset:**
   There are two specific types of features, which are:
   - Numeric: Mostly quantitative and continuous measurements of protein expression levels (e.g., DYRK1A_N, BDNF_N, ITSN1_N) etc.
   - Categorical: Mostly consists of parameters such as MouseID, Genotype, Treatment etc.

   In all, there are 82 features remaining.
5. **Do you need to encode the categorical variables, why or why not?**
   Yes, we do need to encode categorical variables. Categorical variables such as MouseID, Genotype, Treatment require encoding in order to modify the textual based format into a

numerical one, so that the machine learning model is able to thoroughly pursue the data and interpret properly.

## 6. Correlation of all the features by the help of a plotted heatmap:

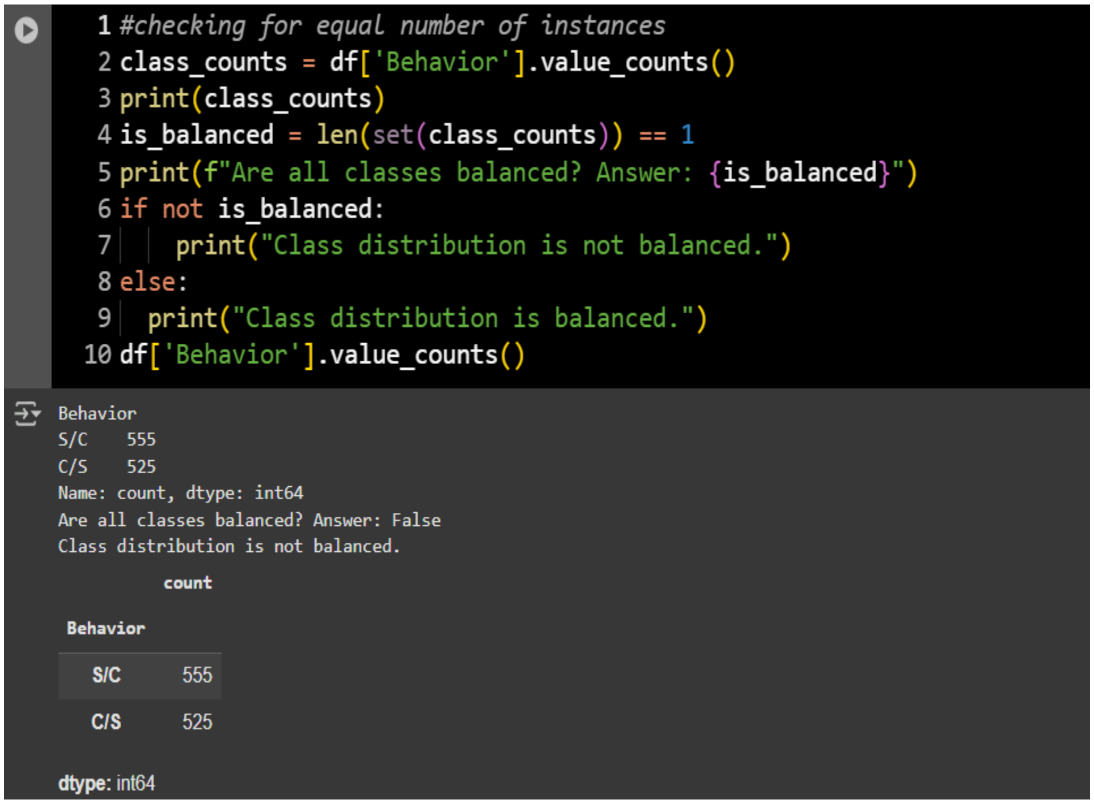


## 7. Understanding of the correlation test:

The correlation matrix generally shows the interrelationship between numerical values of parameters in a dataset, based on their strength and direction. Positive values mean variables tend to increase simultaneously, whilst the negative means one value tends to increase while the other one tends to decrease, in a particular order. Meanwhile, zero defines no relationship, helping us to quickly spot different patterns and relationships in data before further data analysis. In the aspect of the mice protein expression dataset, we have plotted the parameters in a correlation heatmap, which is a colorful grid-like plot of the Pearson correlation coefficient (r) between pairs of variables (BDNF_N, pERK_N, AKT_N etc.)

**Imbalance Dataset:**

a. For the output feature, do all unique classes have an equal number of instances or not? From analysis:



```python
#checking for equal number of instances
class_counts = df['Behavior'].value_counts()
print(class_counts)
is_balanced = len(set(class_counts)) == 1
print(f"Are all classes balanced? Answer: {is_balanced}")
if not is_balanced:
    print("Class distribution is not balanced.")
else:
  print("Class distribution is balanced.")
df['Behavior'].value_counts()
```

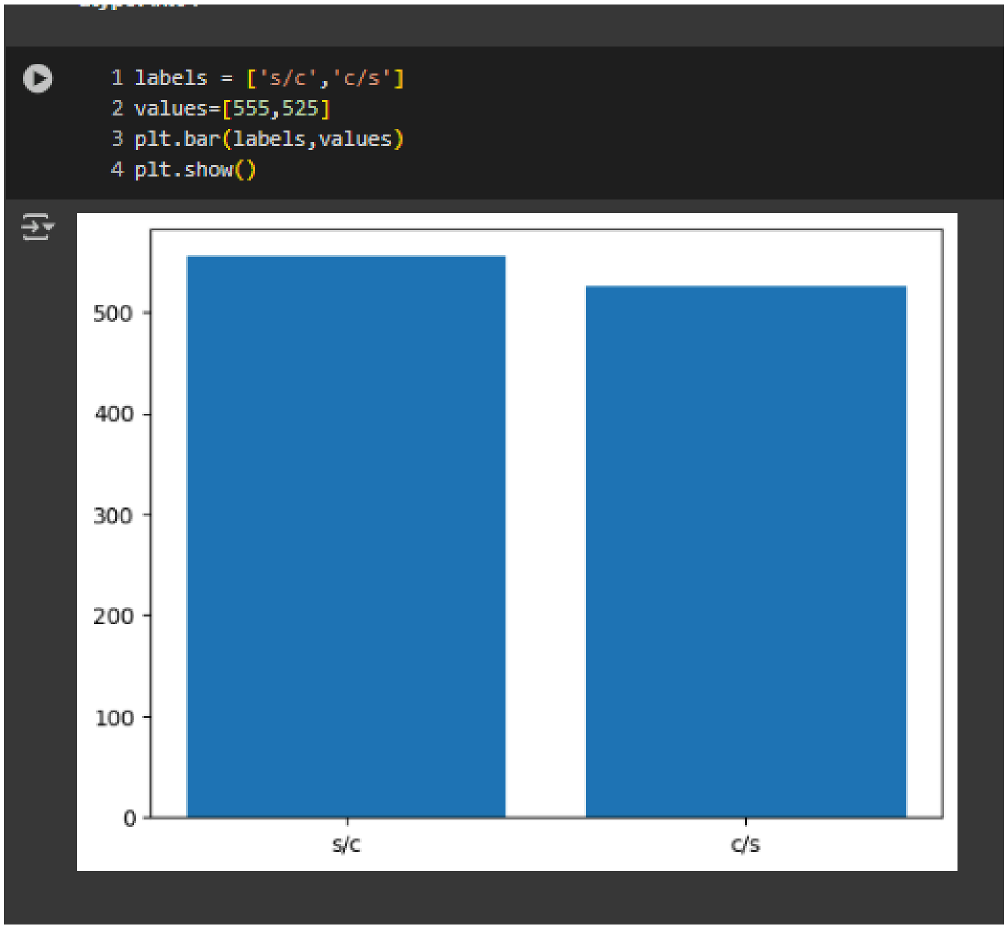
```
Behavior
S/C    555
C/S    525
Name: count, dtype: int64
Are all classes balanced? Answer: False
Class distribution is not balanced.
```

| Behavior | count |
|---|---|
| S/C | 555 |
| C/S | 525 |

dtype: int64

sHere, it is seen that the classes are **unbalanced** with class 1 S/C being 555 and class 2 C/S being 525 (i.e. an unequal distribution).

b. Representation using a bar chart where, N=number of classes in the dataset:



```
labels = ['s/c','c/s']
values=[555,525]
plt.bar(labels,values)
plt.show()
```

**EDA (exploratory data analysis) to extract some important relationships from the dataset:**
During EDA, several important observations are provided below:

1. Minor imbalance in the output classes where, class 1 S/C being 555 and class 2 C/S being 525 (i.e. an unequal distribution).
2. Missing values were checked, and appropriate handling steps were taken (referring to the **pre-processing** section).
3. The dataset seems suitable for classification models without heavy transformations.

## Dataset pre-processing

The dataset underwent pre-processing in order to rid any problems and make it more analysis friendly through numerous steps. The problems and their remedies are mentioned below:

a. **Null/missing values**
   There were some missing values from the original dataset. We used a detection method in order to detect it by utilizing **df.isna().sum().**
   **Solution:**

The rows containing the missing values were either imputed (by the aid of mean/mode based upon the feature type), or dropped if the amount was lessened.

**Reason:**

If a dataset has missing values, the machine learning model can become quite biased towards specific values in the dataset, reducing accuracy significantly and so, it ends up impacting the model training.

b. **Categorical values:** The target feature, Behaviour is actually categorical. In order to remedy this problem, we employed the following;

**Solution:**

We used a special type of encoding, known as 'Label encoding'. This encoding allows for the assigning of numerical values to each category, which are unique. For instance, in this case we used S/C → 0 and C/S → 1.

**Reason:**

Machine learning algorithms typically are able to analyze numerical data as inputs, not textual. So we assigned unique numbers to the categories for machine learning analysis.

c. **Feature Scaling:**

There are numerous types of features present, some of them are larger whilst others are comparatively smaller. This ends up causing inconsistencies.

**Solution:**

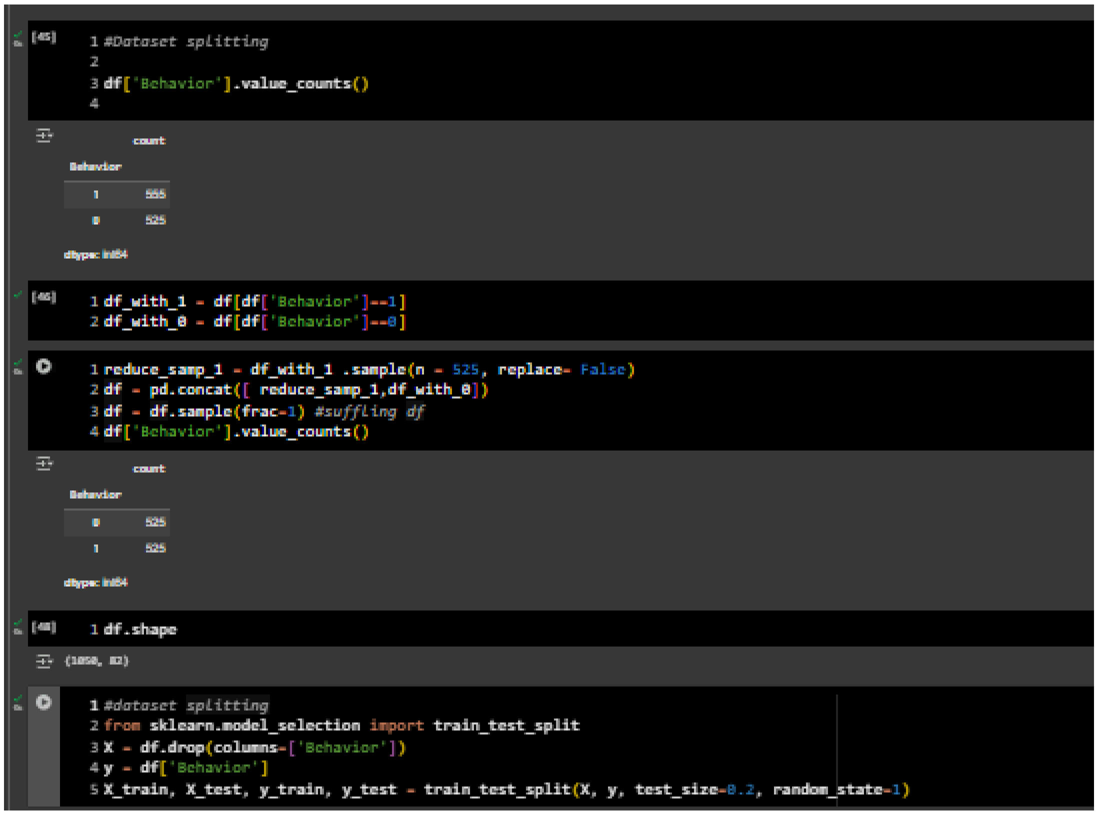
We have applied StandardScaler (Z-score normalization) for this particular dataset. It is optimal for biological data.

**Reason:**

Scaled data is essential for distance-based machine learning models such as KNN and Neural Networks. It allows for better performance and it handles outliers better than others.

## Dataset splitting:

1. Here, the dataset is not balanced at the beginning. It had class 1 S/C being 555 and class 2 C/S being 525 (i.e. an unequal distribution). In this instance, stratified sampling is more appropriate. However, if we balance the data, we can ensure an equal distribution instead. For this case, random splitting becomes sufficient and there remains no need for a stratified sampling as each class now has the same values.



```
[45]
#Dataset splitting

df['Behavior'].value_counts()

          count
Behavior
       1    555
       0    525

dtype: int64
```

```
[46]
df_with_1 = df[df['Behavior']==1]
df_with_0 = df[df['Behavior']==0]
```

```
reduce_samp_1 = df_with_1 .sample(n = 525, replace= False)
df = pd.concat([ reduce_samp_1,df_with_0])
df = df.sample(frac=1) #suffling df
df['Behavior'].value_counts()

          count
Behavior
       0    525
       1    525

dtype: int64
```

```
[48]
df.shape

(1050, [illegible])
```

```
#dataset splitting
from sklearn.model_selection import train_test_split
X = df.drop(columns=['Behavior'])
y = df['Behavior']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)
```

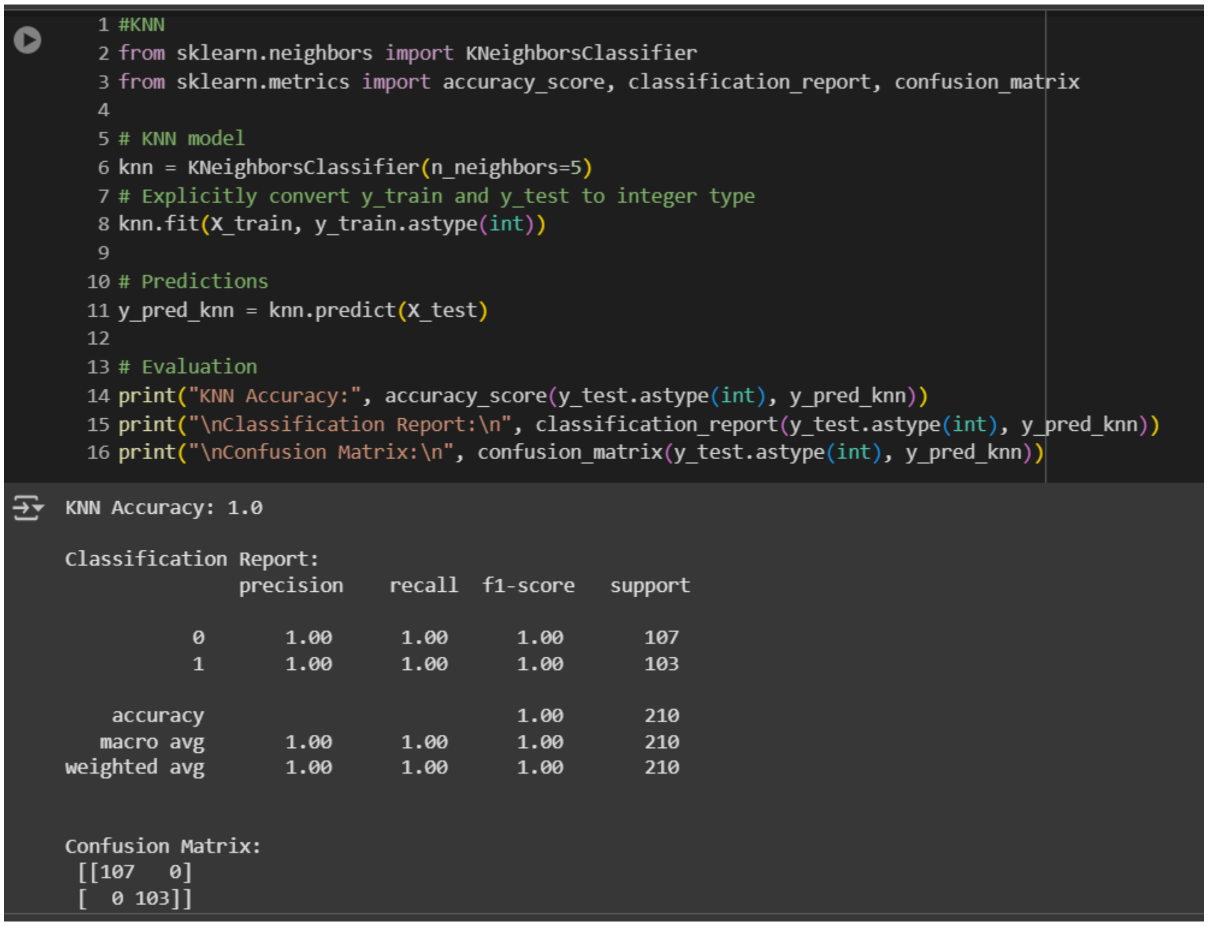
2. We have employed Train and Test sets as, Train = 80% and Test = 20%. It indicates that 20% data is used for testing while the remaining are used for training (1-0.2 = 0.8 i.e. 80%, when test_size = 0.2)

## Model training & testing:

In order to analyze the dataset, we have employed three specific machine learning algorithms. These are:

- KNN (for classification problem)
- Logistic Regression (for regression problem)
- Neural Network (for both)

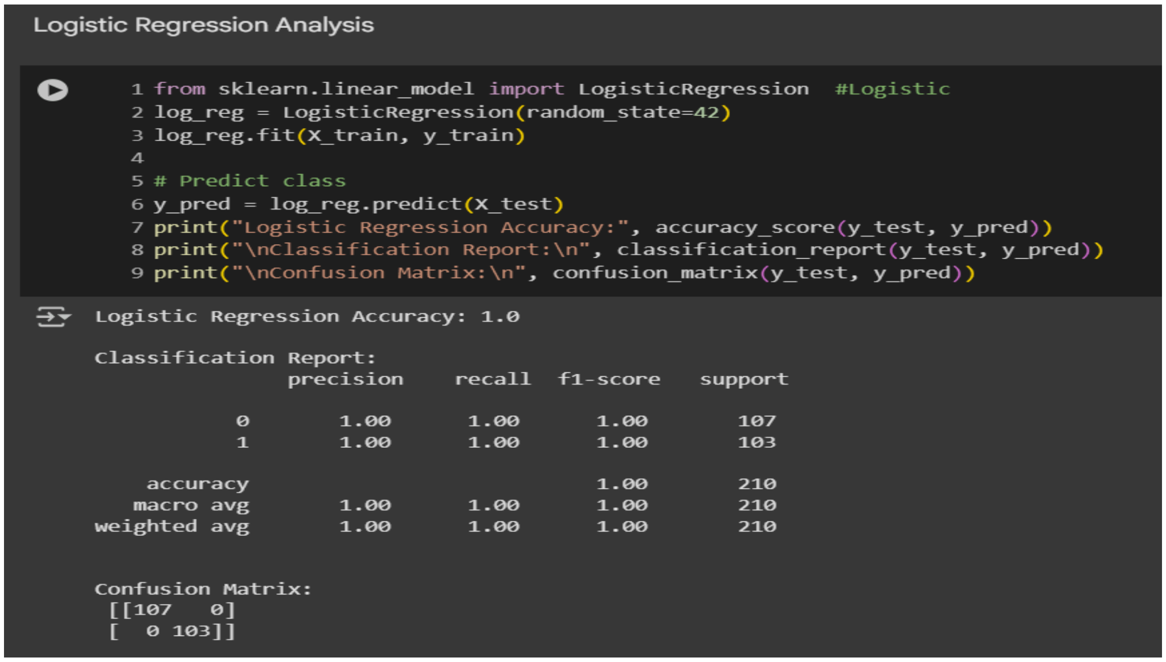
1. **KNN:** We have used KNN and this is the output:



```
#KNN
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# KNN model
knn = KNeighborsClassifier(n_neighbors=5)
# Explicitly convert y_train and y_test to integer type
knn.fit(X_train, y_train.astype(int))

# Predictions
y_pred_knn = knn.predict(X_test)

# Evaluation
print("KNN Accuracy:", accuracy_score(y_test.astype(int), y_pred_knn))
print("\nClassification Report:\n", classification_report(y_test.astype(int), y_pred_knn))
print("\nConfusion Matrix:\n", confusion_matrix(y_test.astype(int), y_pred_knn))
```

```
KNN Accuracy: 1.0

Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00       107
           1       1.00      1.00      1.00       103

    accuracy                           1.00       210
   macro avg       1.00      1.00      1.00       210
weighted avg       1.00      1.00      1.00       210


Confusion Matrix:
 [[107   0]
 [  0 103]]
```

2. **Logistic regression:**
   This is our logistic regression analysis. We have also plotted the confusion matrix of it.

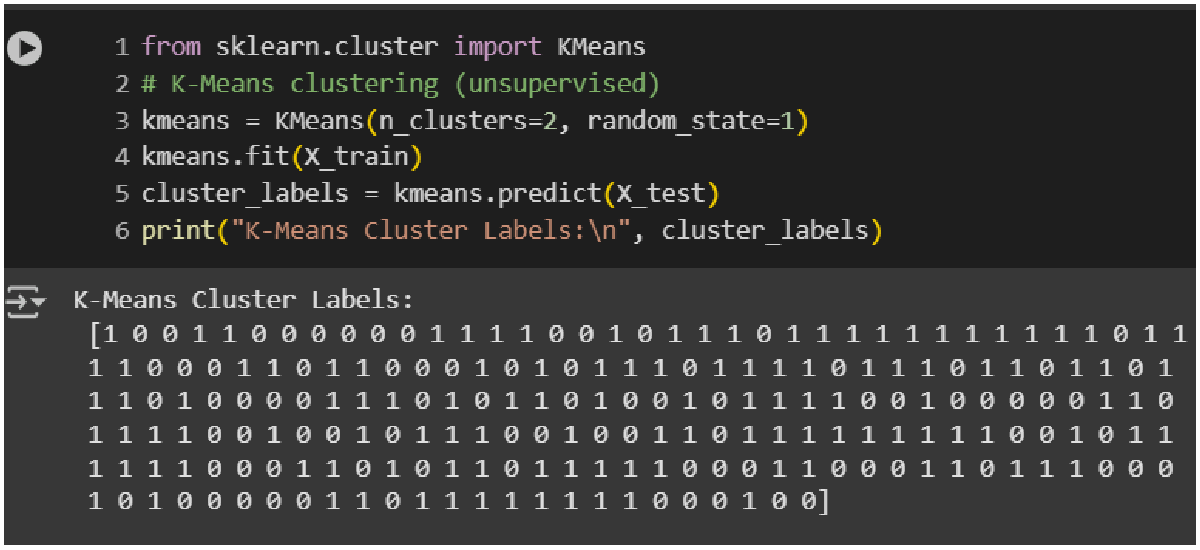


Logistic Regression Analysis

```
from sklearn.linear_model import LogisticRegression  #Logistic
log_reg = LogisticRegression(random_state=42)
log_reg.fit(X_train, y_train)

# Predict class
y_pred = log_reg.predict(X_test)
print("Logistic Regression Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

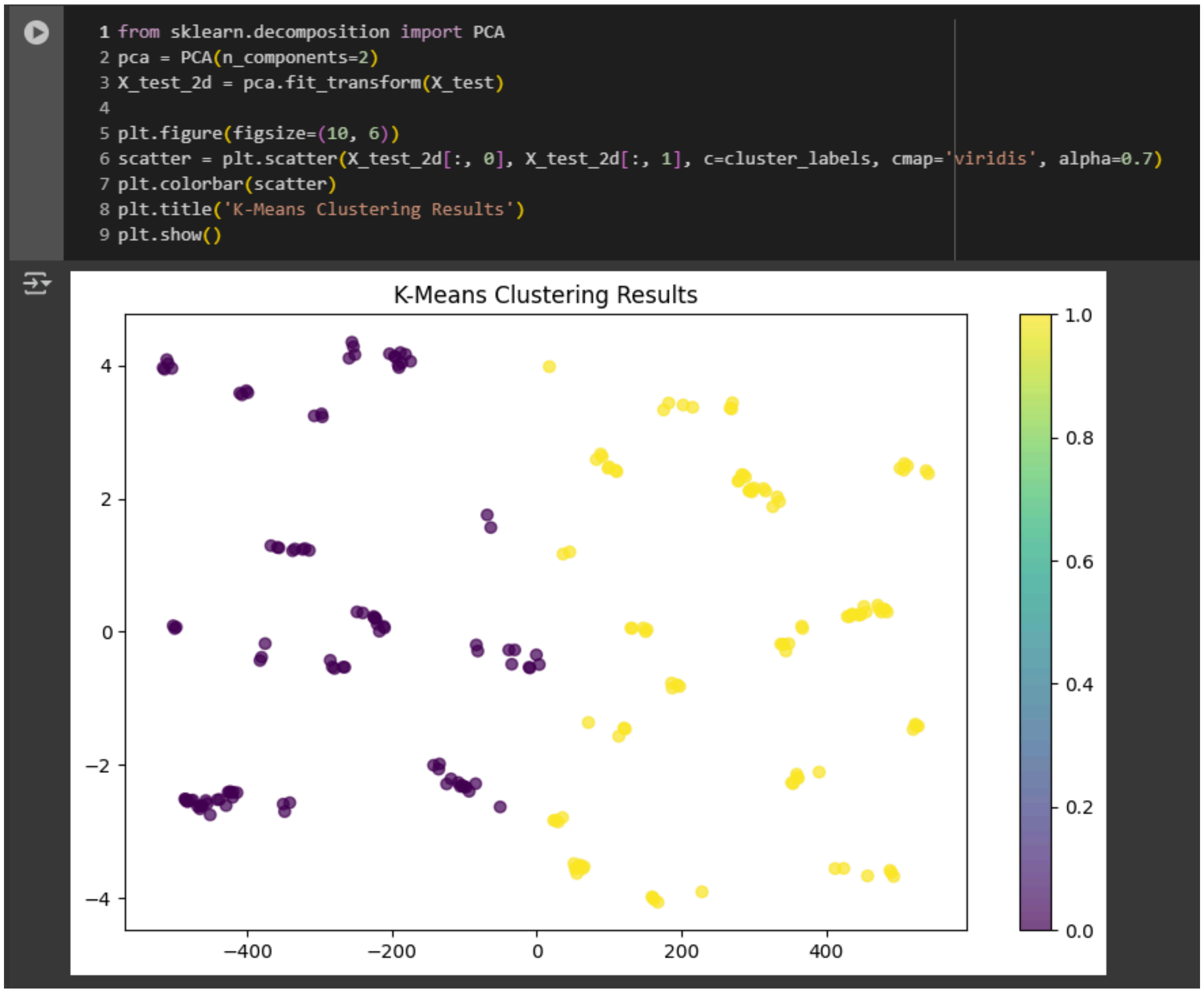
```
Logistic Regression Accuracy: 1.0

Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00       107
           1       1.00      1.00      1.00       103

    accuracy                           1.00       210
   macro avg       1.00      1.00      1.00       210
weighted avg       1.00      1.00      1.00       210


Confusion Matrix:
 [[107   0]
 [  0 103]]
```

We used **Kmeans** and constructed clusters for the dataset, treating it as an **unsupervised** learning problem:



```
from sklearn.cluster import KMeans
# K-Means clustering (unsupervised)
kmeans = KMeans(n_clusters=2, random_state=1)
kmeans.fit(X_train)
cluster_labels = kmeans.predict(X_test)
print("K-Means Cluster Labels:\n", cluster_labels)
```

```
K-Means Cluster Labels:
 [1 0 0 1 1 0 0 0 0 0 0 1 1 1 1 0 0 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1
 1 1 0 0 0 1 1 0 1 1 0 0 0 1 0 1 0 1 1 1 0 1 1 1 1 0 1 1 1 0 1 1 0 1 1 0 1
 1 1 0 1 0 0 0 0 1 1 1 0 1 0 1 1 0 1 0 0 1 0 1 1 1 1 0 0 1 0 0 0 0 0 1 1 0
 1 1 1 1 0 0 1 0 0 1 0 1 1 1 0 0 1 0 0 1 1 0 1 1 1 1 1 1 1 1 1 0 0 1 0 1 1
 1 1 1 1 0 0 0 1 1 0 1 0 1 1 0 1 1 1 1 1 0 0 0 1 1 0 0 0 1 1 0 1 1 1 0 0 0
 1 0 1 0 0 0 0 0 1 1 0 1 1 1 1 1 1 1 0 0 0 1 0 0]
```

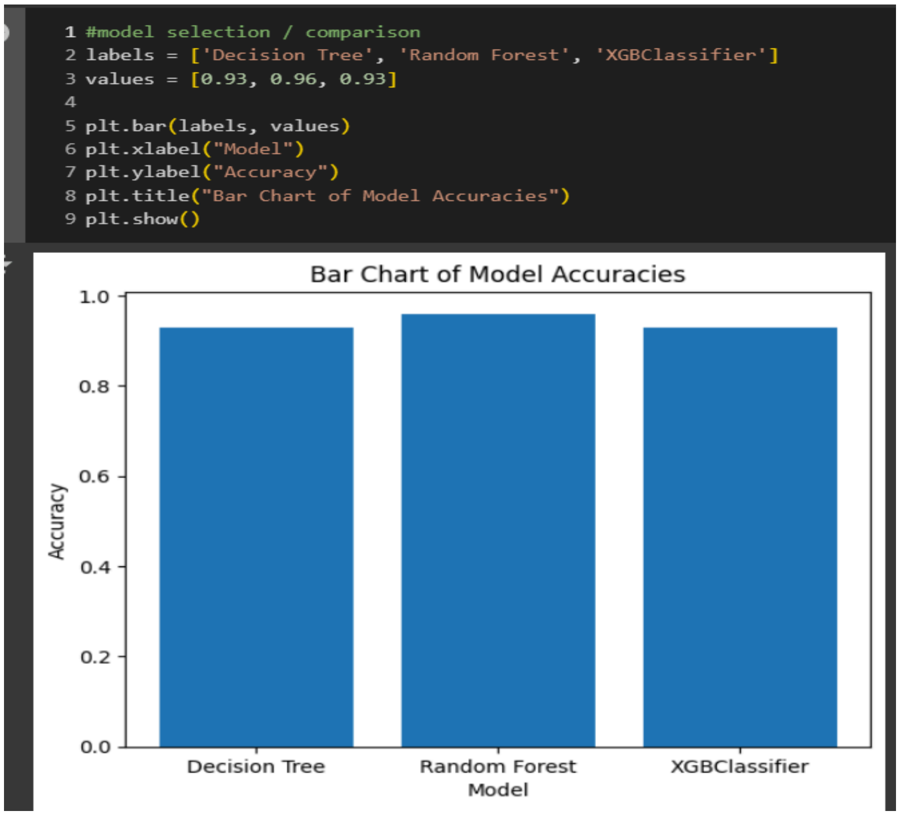


```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
X_test_2d = pca.fit_transform(X_test)

plt.figure(figsize=(10, 6))
scatter = plt.scatter(X_test_2d[:, 0], X_test_2d[:, 1], c=cluster_labels, cmap='viridis', alpha=0.7)
plt.colorbar(scatter)
plt.title('K-Means Clustering Results')
plt.show()
```

## Model selection/Comparison analysis

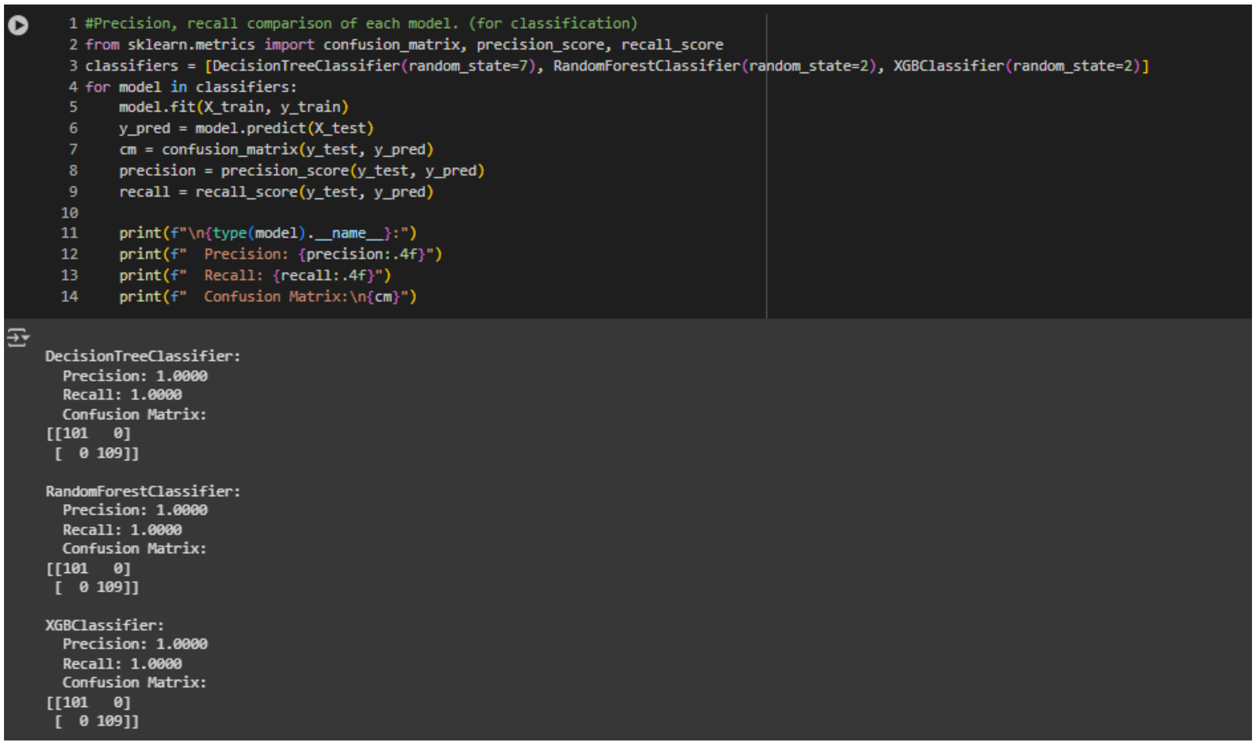
1. **Bar chart showcasing prediction accuracy of all models (for classification)**

This bar chart shows the predicted models with its accuracy:



```
#model selection / comparison
labels = ['Decision Tree', 'Random Forest', 'XGBClassifier']
values = [0.93, 0.96, 0.93]

plt.bar(labels, values)
plt.xlabel("Model")
plt.ylabel("Accuracy")
plt.title("Bar Chart of Model Accuracies")
plt.show()
```

2. **Precision, recall comparison of each model. (for classification):**
   Here, we have generated the following:

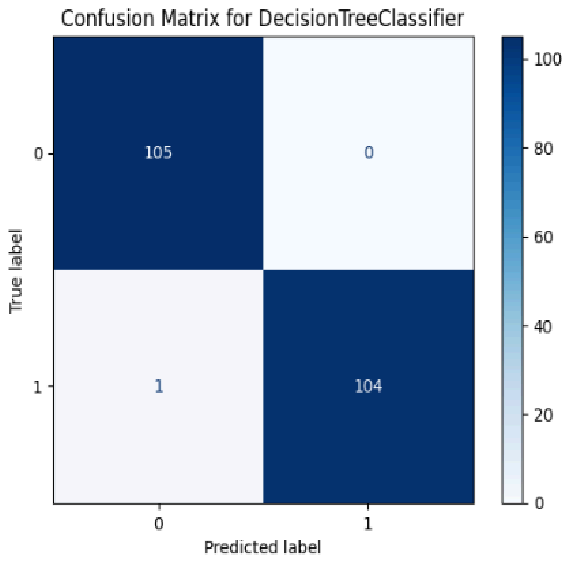


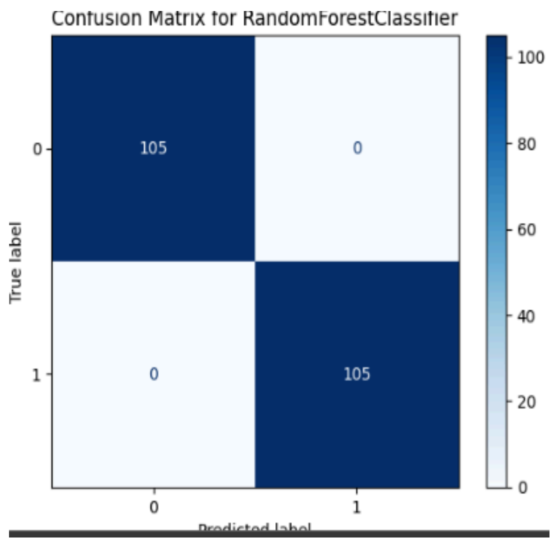
```
#Precision, recall comparison of each model. (for classification)
from sklearn.metrics import confusion_matrix, precision_score, recall_score
classifiers = [DecisionTreeClassifier(random_state=7), RandomForestClassifier(random_state=2), XGBClassifier(random_state=2)]
for model in classifiers:
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    cm = confusion_matrix(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)

    print(f"\n{type(model).__name__}:")
    print(f"  Precision: {precision:.4f}")
    print(f"  Recall: {recall:.4f}")
    print(f"  Confusion Matrix:\n{cm}")
```

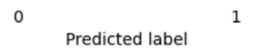
```
DecisionTreeClassifier:
  Precision: 1.0000
  Recall: 1.0000
  Confusion Matrix:
[[101   0]
 [  0 109]]

RandomForestClassifier:
  Precision: 1.0000
  Recall: 1.0000
  Confusion Matrix:
[[101   0]
 [  0 109]]

XGBClassifier:
  Precision: 1.0000
  Recall: 1.0000
  Confusion Matrix:
[[101   0]
 [  0 109]]
```

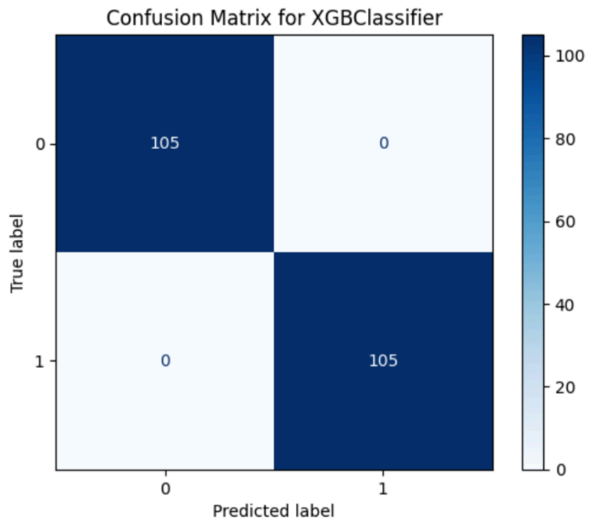
From the above snippet, we obtained precision for the decision tree classifier as 1.000, and recall as 1.000. Likewise, we obtained 1.000 recall and precision in the case of random forest classifier. Finally, for the XGB classifier we obtained precision as 1.000 and recall as 1.000, respectively.

3. **Confusion Matrix (for classification):**

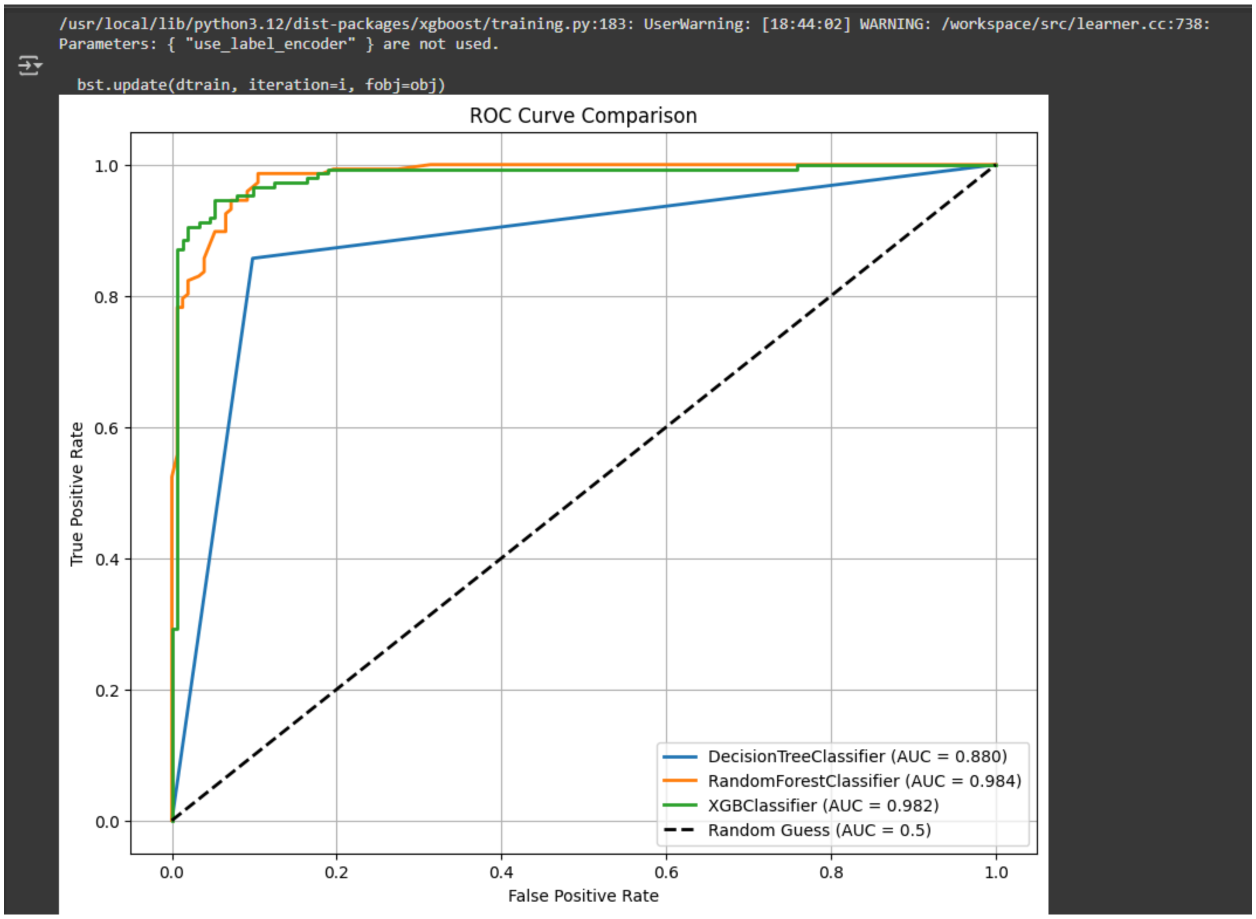








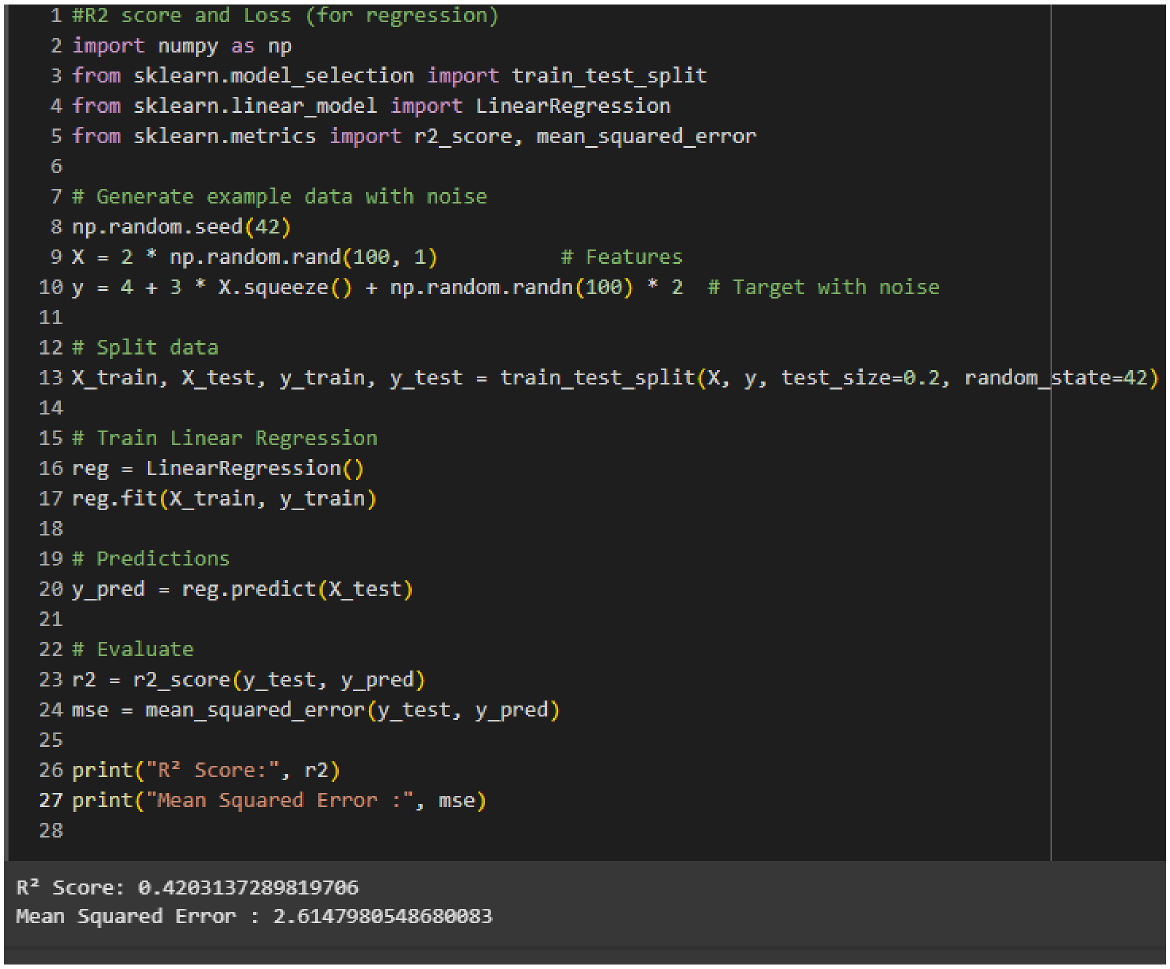
4. **AUC score, ROC curve for each model (for classification):**
   We have determined  the AUC score and ROC curve for each model:



```
/usr/local/lib/python3.12/dist-packages/xgboost/training.py:183: UserWarning: [18:44:02] WARNING: /workspace/src/learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
```

**5. R2 score and Loss (for regression):**

Here are the scores for R2 and loss, using regression:



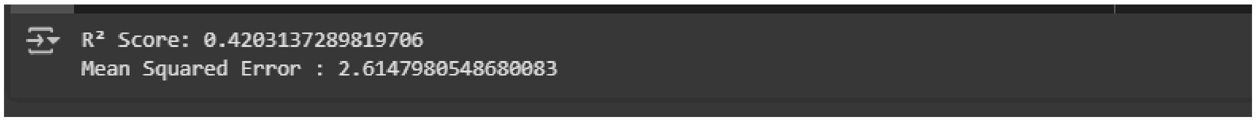
```
#R2 score and Loss (for regression)
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error

# Generate example data with noise
np.random.seed(42)
X = 2 * np.random.rand(100, 1)          # Features
y = 4 + 3 * X.squeeze() + np.random.randn(100) * 2  # Target with noise

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train Linear Regression
reg = LinearRegression()
reg.fit(X_train, y_train)

# Predictions
y_pred = reg.predict(X_test)

# Evaluate
r2 = r2_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)

print("R² Score:", r2)
print("Mean Squared Error :", mse)

```

```
R² Score: 0.4203137289819706
Mean Squared Error : 2.6147980548680083
```

A result comparison from all the above deployed metrics have been shown below:

1. **Bar chart Results:** From the analysis, we obtained
   Random Forest: 0.96 i.e. 96%
   XGBoost: 0.93 i.e. 93%
   Decision Tree: 0.93 i.e. 93%
2. **Precision and Recall comparison:** From the analysis, we obtained
   Random Forest: 1.000 precision and 1.000 recall
   XGBoost: 1.000 precision and 1.000 recall
   Decision Tree: 1.000 precision, and recall as 1.000
3. **Confusion matrix:** In case of the matrices,
   Random Forest: Classification with 105+105 =210
   Decision Tree: Classification with 105+104 = 209
   XGBoost: Classification with 105+105 =  210
   These indicate that highly accurate scores for both random forest and XGBoost are present while decision tree has a slightly lesser score.

4. **AUC score, ROC curve for each model (for classification):**
   Random Forest: 0.984
   XGBoost: 0.982
   Decision Tree: 0.880
   Random Guess AUC = 0.5
5. **R2 score and Loss (for regression):**
   The scores are:



```
R² Score: 0.4203137289819706
Mean Squared Error : 2.6147980548680083
```

## Conclusion:

The data analysis yielded outstanding performance. Almost all the obtained results had an accuracy ranging till 100%. This is quite an exceptional result. However, discrepancies might have influenced such an output.
We obtained such results after our entire analysis. The AUC scores were good. The scores of confusion matrices were very high, which may be traced back to either calculation based errors, or dataset inconsistencies.
We have faced numerous challenges in completion of this project. The dataset had an increasing number of imbalanced data. We also had to adjust to ensure there remains no data leakage. There may remain overly fitted and validation issues because of the unique properties of the data.