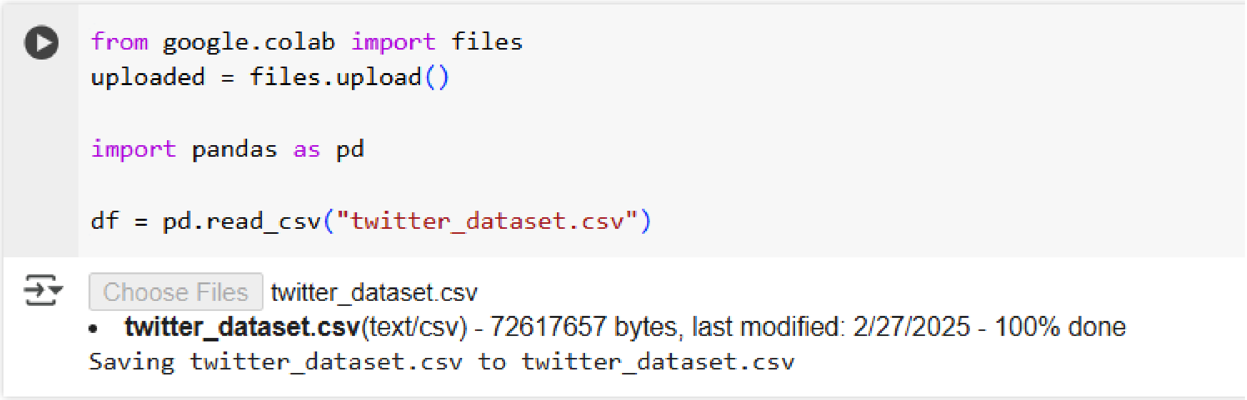
Experiment No: 5

Aim: Perform Regression Analysis using Scipy and Sci-kit learn.

In this experiment, we performed Logistic Regression using SciPy and Scikit-Learn to analyze and predict user behavior on Twitter. The dataset consists of various attributes such as followers count, friends count, statuses count, BotScore, mentions, and engagement metrics like retweets, replies, and quotes. The target variable, BinaryNumTarget, classifies users into two categories, potentially indicating bots or real users.

Logistic Regression, a widely used classification algorithm, was employed to model the relationship between these features and the binary outcome. The dataset was preprocessed by handling missing values, normalizing features, and splitting it into training and testing sets. The model's performance was evaluated using accuracy, confusion matrix, and classification metrics such as precision, recall, and F1-score.

1. **Upload the Dataset to Google Colab**

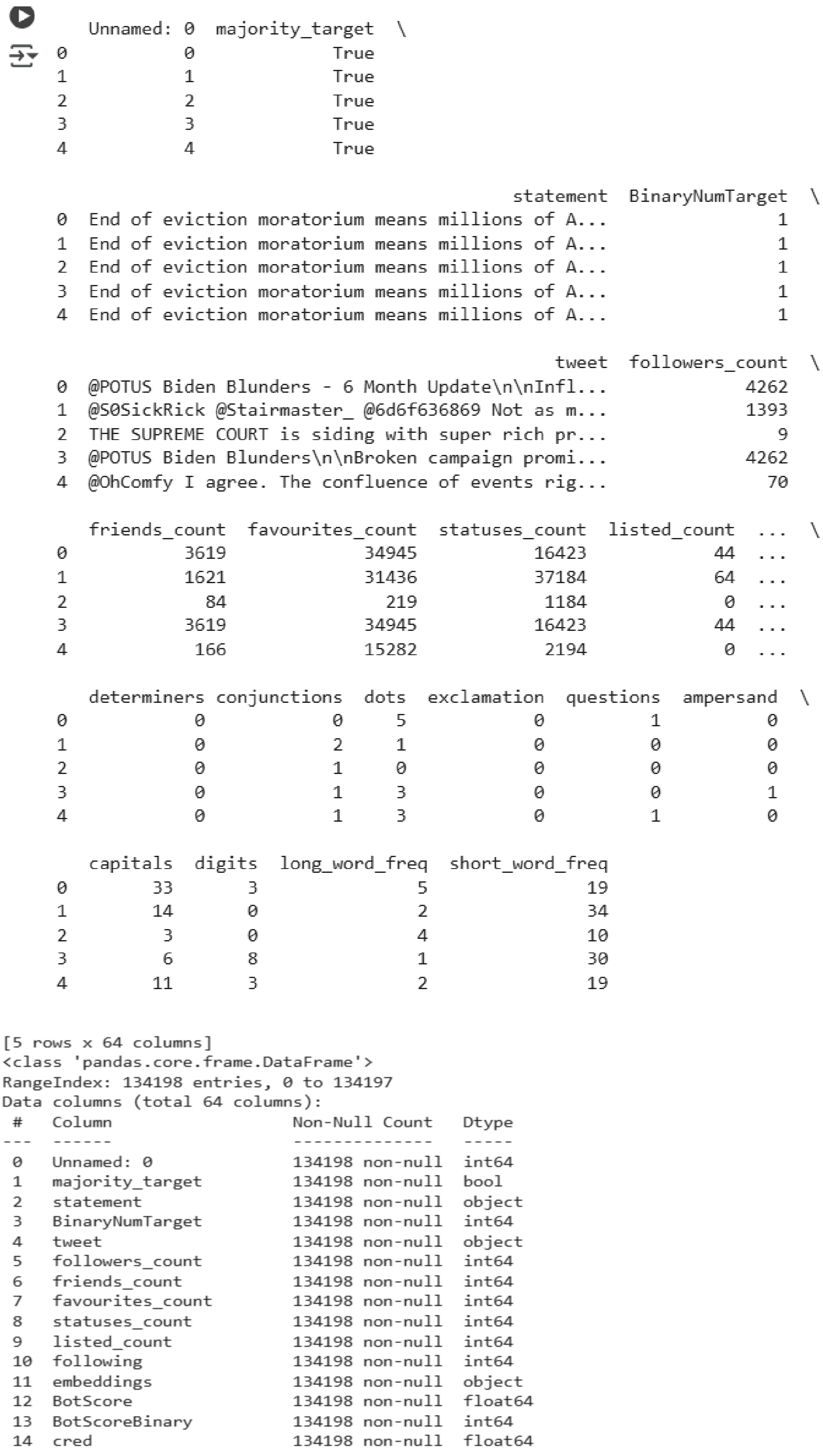


```
from google.colab import files
uploaded = files.upload()

import pandas as pd

df = pd.read_csv("twitter_dataset.csv")
```

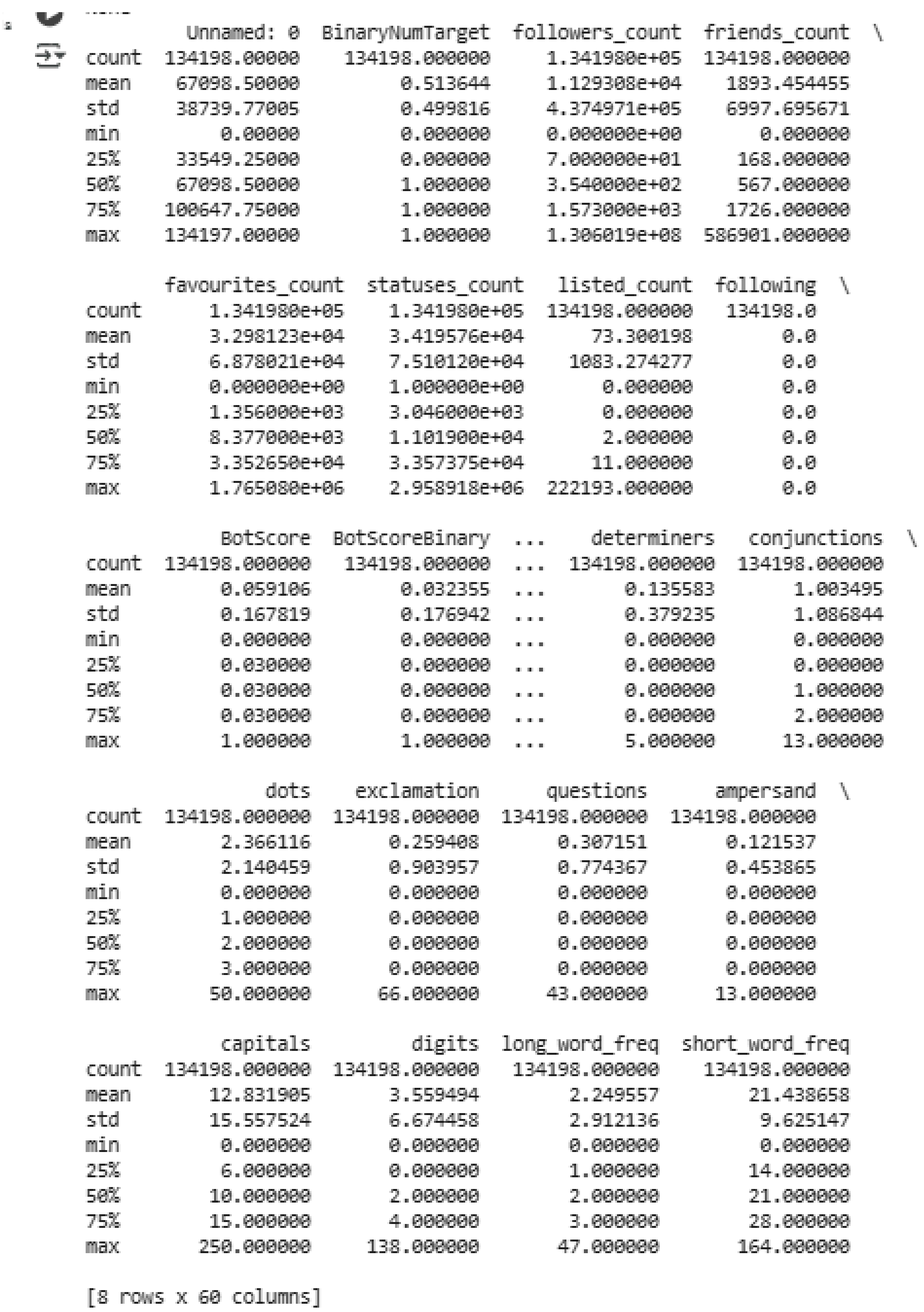
```
twitter_dataset.csv
• twitter_dataset.csv(text/csv) - 72617657 bytes, last modified: 2/27/2025 - 100% done
Saving twitter_dataset.csv to twitter_dataset.csv
```

2. **Explore the Dataset**

# Display basic information about the dataset
df.info()
# Display the first few rows of the dataset
df.head()
# Display summary statistics of numerical columns
df.describe()



```
   Unnamed: 0  majority_target  \
0           0             True
1           1             True
2           2             True
3           3             True
4           4             True

                                           statement  BinaryNumTarget  \
0  End of eviction moratorium means millions of A...                1
1  End of eviction moratorium means millions of A...                1
2  End of eviction moratorium means millions of A...                1
3  End of eviction moratorium means millions of A...                1
4  End of eviction moratorium means millions of A...                1

                                               tweet  followers_count  \
0  @POTUS Biden Blunders - 6 Month Update\n\nInfl...             4262
1  @S0SickRick @Stairmaster_ @6d6f636869 Not as m...             1393
2  THE SUPREME COURT is siding with super rich pr...                9
3  @POTUS Biden Blunders\n\nBroken campaign promi...             4262
4  @OhComfy I agree. The confluence of events rig...               70

   friends_count  favourites_count  statuses_count  listed_count  ...  \
0           3619             34945           16423            44  ...
1           1621             31436           37184            64  ...
2             84               219            1184             0  ...
3           3619             34945           16423            44  ...
4            166             15282            2194             0  ...

   determiners conjunctions  dots  exclamation  questions  ampersand  \
0            0            0     5            0          1          0
1            0            2     1            0          0          0
2            0            1     0            0          0          0
3            0            1     3            0          0          1
4            0            1     3            0          1          0

   capitals  digits  long_word_freq  short_word_freq
0        33       3               5               19
1        14       0               2               34
2         3       0               4               10
3         6       8               1               30
4        11       3               2               19

[5 rows x 64 columns]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 134198 entries, 0 to 134197
Data columns (total 64 columns):
 #   Column                  Non-Null Count   Dtype
---  ------                  --------------   -----
 0   Unnamed: 0              134198 non-null  int64
 1   majority_target         134198 non-null  bool
 2   statement               134198 non-null  object
 3   BinaryNumTarget         134198 non-null  int64
 4   tweet                   134198 non-null  object
 5   followers_count         134198 non-null  int64
 6   friends_count           134198 non-null  int64
 7   favourites_count        134198 non-null  int64
 8   statuses_count          134198 non-null  int64
 9   listed_count            134198 non-null  int64
 10  following               134198 non-null  int64
 11  embeddings              134198 non-null  object
 12  BotScore                134198 non-null  float64
 13  BotScoreBinary          134198 non-null  int64
 14  cred                    134198 non-null  float64
```



```
         Unnamed: 0  BinaryNumTarget  followers_count  friends_count  \
count  134198.00000    134198.000000     1.341980e+05  134198.000000
mean    67098.50000         0.513644     1.129308e+04    1893.454455
std     38739.77005         0.499816     4.374971e+05    6997.695671
min         0.00000         0.000000     0.000000e+00       0.000000
25%     33549.25000         0.000000     7.000000e+01     168.000000
50%     67098.50000         1.000000     3.540000e+02     567.000000
75%    100647.75000         1.000000     1.573000e+03    1726.000000
max    134197.00000         1.000000     1.306019e+08  586901.000000

       favourites_count  statuses_count   listed_count  following  \
count      1.341980e+05    1.341980e+05  134198.000000   134198.0
mean       3.298123e+04    3.419576e+04      73.300198        0.0
std        6.878021e+04    7.510120e+04    1083.274277        0.0
min        0.000000e+00    1.000000e+00       0.000000        0.0
25%        1.356000e+03    3.046000e+03       0.000000        0.0
50%        8.377000e+03    1.101900e+04       2.000000        0.0
75%        3.352650e+04    3.357375e+04      11.000000        0.0
max        1.765080e+06    2.958918e+06  222193.000000        0.0

            BotScore  BotScoreBinary  ...    determiners    conjunctions  \
count  134198.000000   134198.000000  ...  134198.000000  134198.000000
mean        0.059106        0.032355  ...       0.135583       1.003495
std         0.167819        0.176942  ...       0.379235       1.086844
min         0.000000        0.000000  ...       0.000000       0.000000
25%         0.030000        0.000000  ...       0.000000       0.000000
50%         0.030000        0.000000  ...       0.000000       1.000000
75%         0.030000        0.000000  ...       0.000000       2.000000
max         1.000000        1.000000  ...       5.000000      13.000000

                dots    exclamation      questions      ampersand  \
count  134198.000000  134198.000000  134198.000000  134198.000000
mean        2.366116       0.259408       0.307151       0.121537
std         2.140459       0.903957       0.774367       0.453865
min         0.000000       0.000000       0.000000       0.000000
25%         1.000000       0.000000       0.000000       0.000000
50%         2.000000       0.000000       0.000000       0.000000
75%         3.000000       0.000000       0.000000       0.000000
max        50.000000      66.000000      43.000000      13.000000

            capitals         digits  long_word_freq  short_word_freq
count  134198.000000  134198.000000   134198.000000    134198.000000
mean       12.831905       3.559494        2.249557        21.438658
std        15.557524       6.674458        2.912136         9.625147
min         0.000000       0.000000        0.000000         0.000000
25%         6.000000       0.000000        1.000000        14.000000
50%        10.000000       2.000000        2.000000        21.000000
75%        15.000000       4.000000        3.000000        28.000000
max       250.000000     138.000000       47.000000       164.000000

[8 rows x 60 columns]
```

3. Check for null values

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

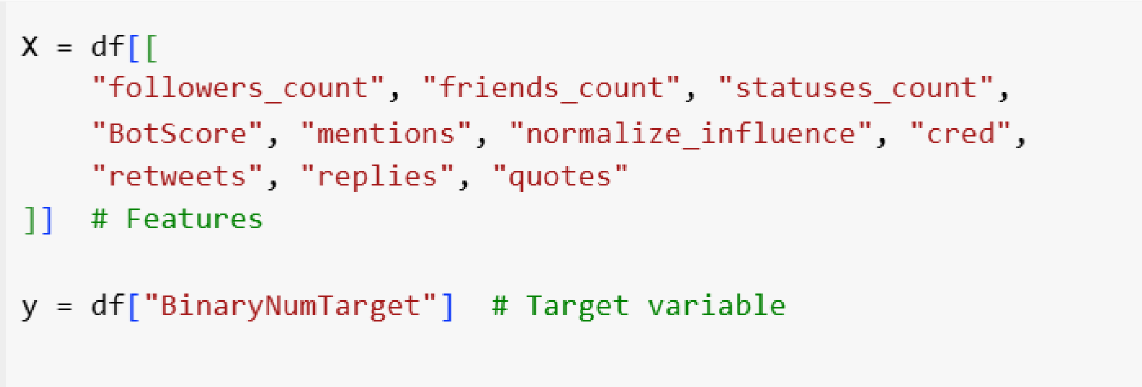
We found no null values in the data so there is no need to take care of any missing values.

4. **Selecting Features and Target Variable**
   Choosing Independent Variables (X):

- We select relevant user attributes that may influence the classification.
- The chosen features include engagement metrics (retweets, replies, quotes), user statistics (followers_count, friends_count, statuses_count), and credibility scores (BotScore, normalize_influence, cred, mentions).

Defining the Target Variable (y):

- The target variable, BinaryNumTarget, represents a binary classification (0 or 1), where the model predicts whether a user belongs to a specific category.



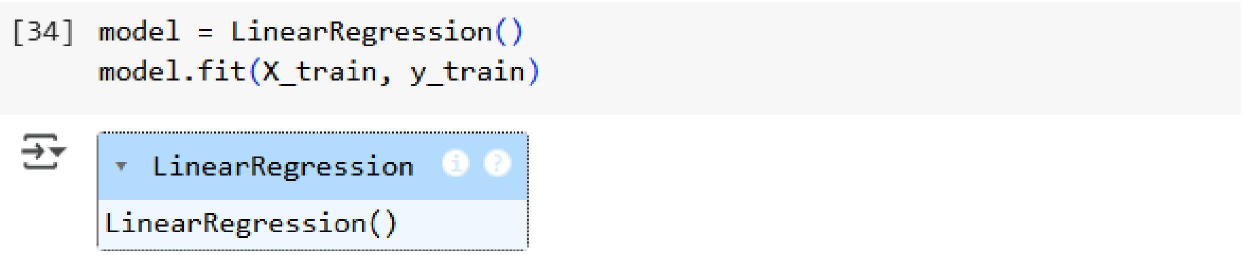
```
X = df[[
    "followers_count", "friends_count", "statuses_count",
    "BotScore", "mentions", "normalize_influence", "cred",
    "retweets", "replies", "quotes"
]]  # Features

y = df["BinaryNumTarget"]  # Target variable
```

5. **Regression**

Linear Regression:

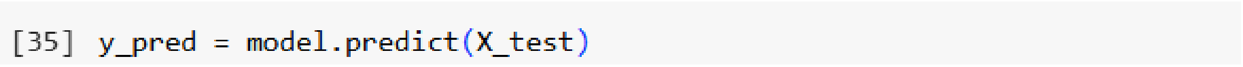
Training the Model:
After preparing the dataset, we train a Linear Regression model to predict the target variable based on the selected features. Linear Regression is a fundamental regression algorithm that models the relationship between independent and dependent variables by fitting a straight line to the data. It aims to find the optimal weights for each feature to minimize the error between the actual and predicted values using the least squares method.

In this step, we initialize and train the model using the Scikit-Learn LinearRegression() function.



```
[34] model = LinearRegression()
     model.fit(X_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

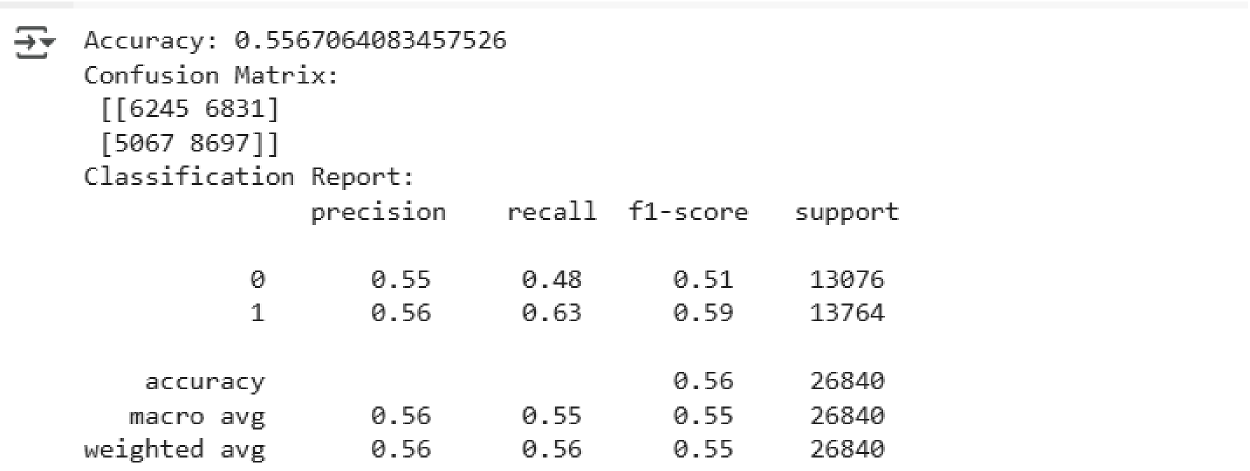


```
[35] y_pred = model.predict(X_test)
```

After training the **Linear Regression** model, we evaluate its performance by interpreting the predictions in a classification context. Since Linear Regression outputs continuous values, we convert them into binary classes (0 or 1) based on a threshold (e.g., 0.5).

Model Evaluation

We then compute key classification metrics:
Accuracy – Measures the overall correctness of predictions.
Confusion Matrix – Shows the number of correct and incorrect classifications.
Classification Report – Provides precision, recall, and F1-score for each class.



```
Accuracy: 0.5567064083457526
Confusion Matrix:
 [[6245 6831]
 [5067 8697]]
Classification Report:
              precision    recall  f1-score   support

           0       0.55      0.48      0.51     13076
           1       0.56      0.63      0.59     13764

    accuracy                           0.56     26840
   macro avg       0.56      0.55      0.55     26840
weighted avg       0.56      0.56      0.55     26840
```

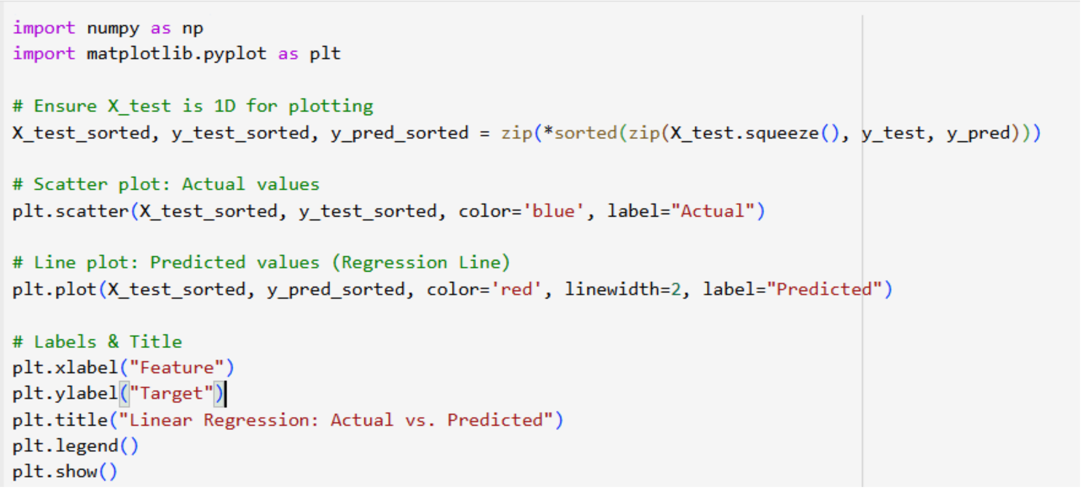
The results show that after converting Linear Regression outputs to binary (0 or 1), the model achieved 56% accuracy.

- Confusion Matrix: The model correctly classified 6245 instances of class 0 and 8697 of class 1, but misclassified 6831 and 5067 instances, respectively.

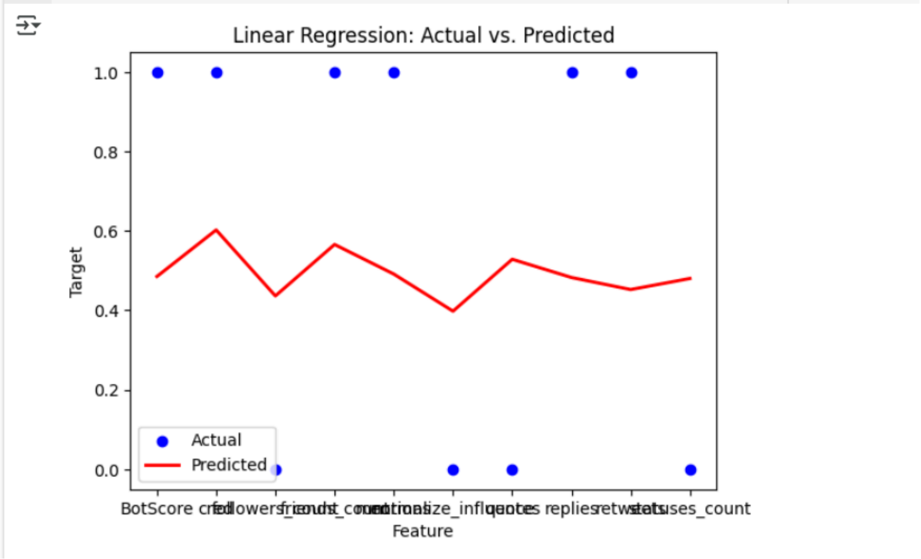
- Classification Report: The model has a precision of 0.55 for class 0 and 0.56 for class 1, with an overall F1-score of ~0.55–0.56, indicating moderate performance.

This suggests that Linear Regression may not be the best choice for classification tasks, as it lacks the probabilistic decision-making of Logistic Regression.

The next step is to **evaluate the performance** of the Linear Regression model using appropriate regression metrics.



```
import numpy as np
import matplotlib.pyplot as plt

# Ensure X_test is 1D for plotting
X_test_sorted, y_test_sorted, y_pred_sorted = zip(*sorted(zip(X_test.squeeze(), y_test, y_pred)))

# Scatter plot: Actual values
plt.scatter(X_test_sorted, y_test_sorted, color='blue', label="Actual")

# Line plot: Predicted values (Regression Line)
plt.plot(X_test_sorted, y_pred_sorted, color='red', linewidth=2, label="Predicted")

# Labels & Title
plt.xlabel("Feature")
plt.ylabel("Target")
plt.title("Linear Regression: Actual vs. Predicted")
plt.legend()
plt.show()
```



The plot visualizes the **Linear Regression** model's performance:

- Blue dots represent the actual target values.
- Red line represents the model's predicted values.

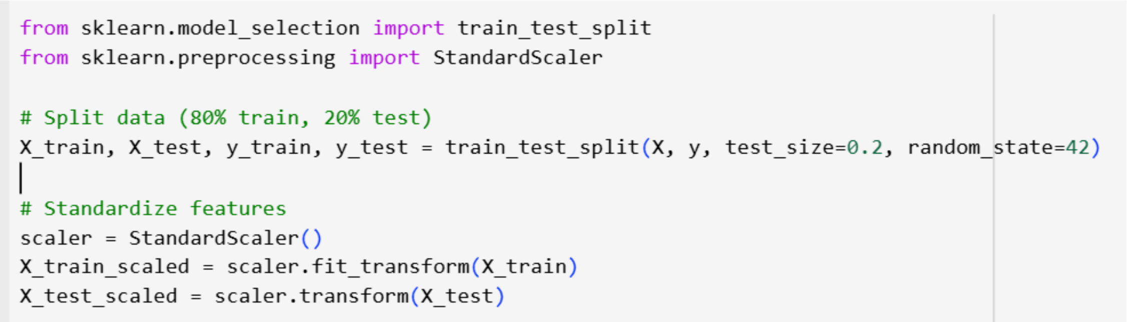
Observations:

- The **actual values (blue dots) are binary (0 or 1)**, meaning the target variable is categorical.
- The **predicted values (red line) are continuous**, which is expected in Linear Regression but may not be ideal for classification.
- The model’s predictions do not perfectly align with actual values, indicating possible underfitting.

Logistic Regression:

After preparing the dataset, we train a Logistic Regression model to classify users based on the selected features. Logistic Regression is a widely used classification algorithm that predicts the probability of an instance belonging to a particular class using the sigmoid function. It finds the optimal weights for each feature to minimize classification errors.

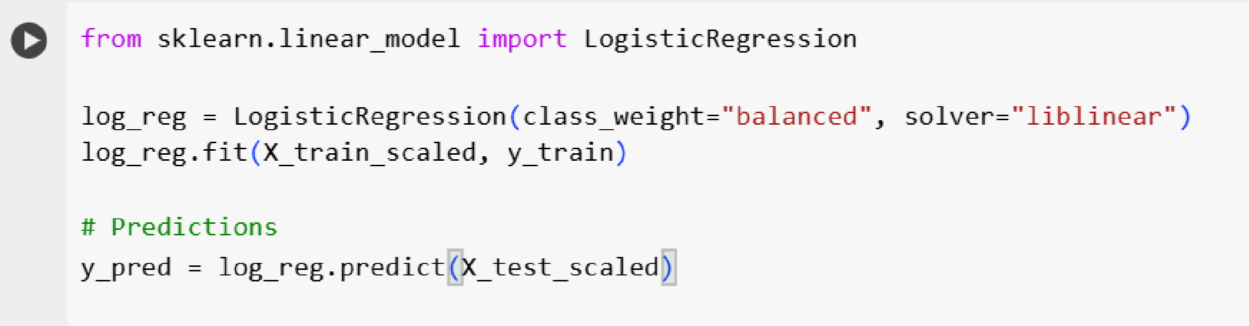
In this step, we initialize and train the model using the Scikit-Learn LogisticRegression() function. We also set class_weight="balanced" to handle any potential class imbalance and use the "liblinear" solver, which is efficient for smaller datasets.



```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Split data (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Model Prediction

Once the Logistic Regression model is trained, we use it to make predictions on the test dataset. The model applies the learned coefficients to the test data and classifies each instance as 0 or 1 based on the sigmoid function’s output.



```
from sklearn.linear_model import LogisticRegression

log_reg = LogisticRegression(class_weight="balanced", solver="liblinear")
log_reg.fit(X_train_scaled, y_train)

# Predictions
y_pred = log_reg.predict(X_test_scaled)
```

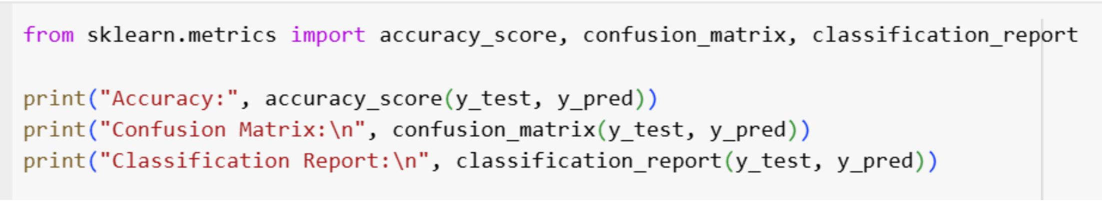
Model Evaluation

After making predictions, we evaluate the performance of our Logistic Regression model using accuracy, confusion matrix, and classification report from sklearn.metrics.

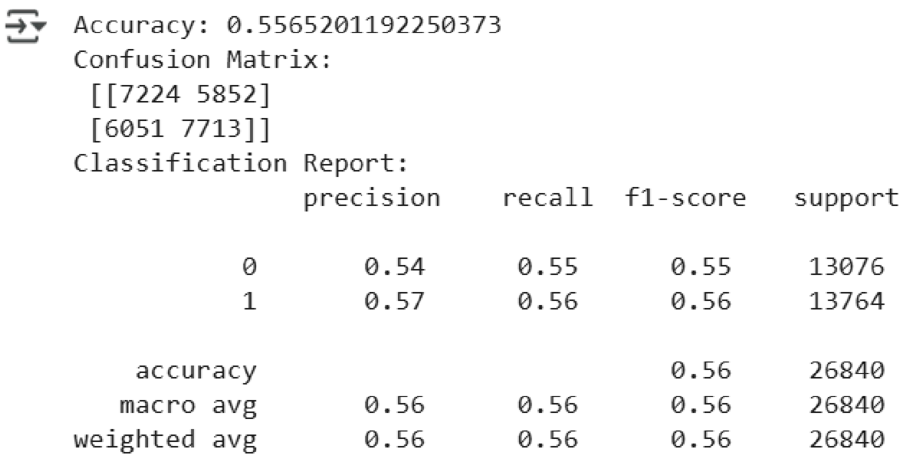
accuracy_score(y_test, y_pred): Measures the overall correctness of predictions.

confusion_matrix(y_test, y_pred): Displays how many predictions were correctly or incorrectly classified.

classification_report(y_test, y_pred): Provides precision, recall, and F1-score for both classes (0 and 1).



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

print("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```



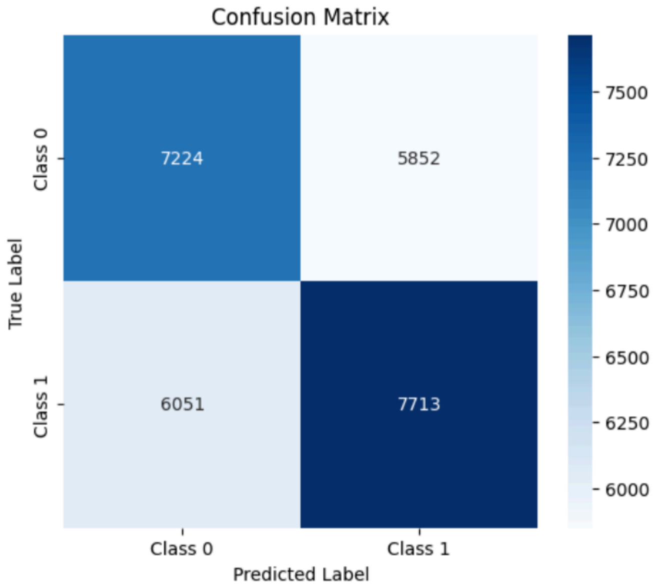
```
Accuracy: 0.5565201192250373
Confusion Matrix:
 [[7224 5852]
 [6051 7713]]
Classification Report:
               precision    recall  f1-score   support

           0       0.54      0.55      0.55     13076
           1       0.57      0.56      0.56     13764

    accuracy                           0.56     26840
   macro avg       0.56      0.56      0.56     26840
weighted avg       0.56      0.56      0.56     26840
```

The evaluation results show that our Logistic Regression model achieved an accuracy of approximately 55.65%, indicating moderate predictive performance. The confusion matrix reveals that the model correctly classified 7,224 instances as Class 0 and 7,713 as Class 1, but misclassified 5,852 as Class 1 and 6,051 as Class 0. The classification report shows nearly equal precision and recall for both classes, ranging between 0.54 - 0.57, suggesting the model performs similarly for detecting bots and non-bots.

Making the Confusion Matrix

To visually interpret the model's performance, we plotted a confusion matrix heatmap using Seaborn. This heatmap provides a clearer understanding of how well the Logistic Regression model classified the data. The x-axis represents the predicted labels, while the y-axis represents the true labels. The correctly classified instances are shown along the diagonal, while misclassified cases appear in the off-diagonal cells. The intensity of the blue color indicates the number of samples in each category, making it easier to analyze the model's strengths and weaknesses in classification.

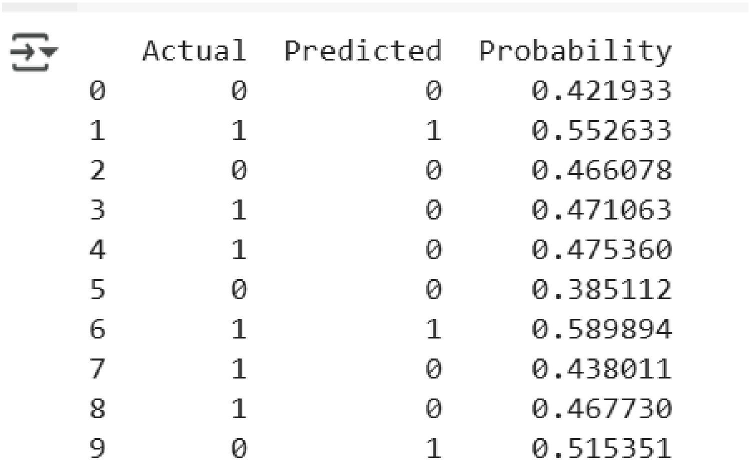
```
cm = confusion_matrix(y_test, y_pred)

# Plot heatmap
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["Class 0", "Class 1"],
yticklabels=["Class 0", "Class 1"])
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()
```



Model Predictions and Confidence Scores

In this step, we predict probabilities for each test sample using the predict_proba function, which returns the probability of belonging to each class. We extract the probability of Class 1 and use it to make final class predictions. Then, we create a DataFrame to compare actual labels, predicted labels, and their respective probabilities, allowing us to analyze how confident the model is in its predictions.

```
# Predict probabilities for test data
y_prob = log_reg.predict_proba(X_test_scaled)[:, 1] # Probability of Class 1
# Predict final class labels
y_pred = log_reg.predict(X_test_scaled)
# Show some predictions
predictions_df = pd.DataFrame({"Actual": y_test.values, "Predicted": y_pred,
"Probability": y_prob})
print(predictions_df.head(10)) # Show first 10 predictions
```

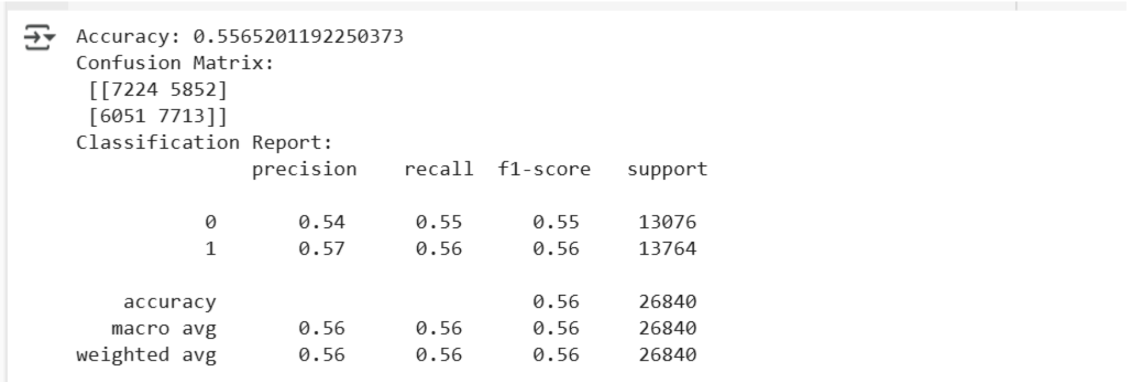


```
   Actual  Predicted  Probability
0       0          0     0.421933
1       1          1     0.552633
2       0          0     0.466078
3       1          0     0.471063
4       1          0     0.475360
5       0          0     0.385112
6       1          1     0.589894
7       1          0     0.438011
8       1          0     0.467730
9       0          1     0.515351
```

Adjusting the Decision Threshold for Optimized Classification

In this step, we adjust the decision threshold for classification. Instead of using the default threshold of 0.5, we set it to 0.6, meaning a sample is classified as 1 only if its predicted probability is at least 0.6. This helps in tuning model performance by balancing precision and recall. The adjusted confusion matrix and classification report show the new results, where Class 0 has higher recall, but Class 1's recall has dropped significantly. This trade-off is useful when optimizing for a specific metric, like minimizing false positives or false negatives.

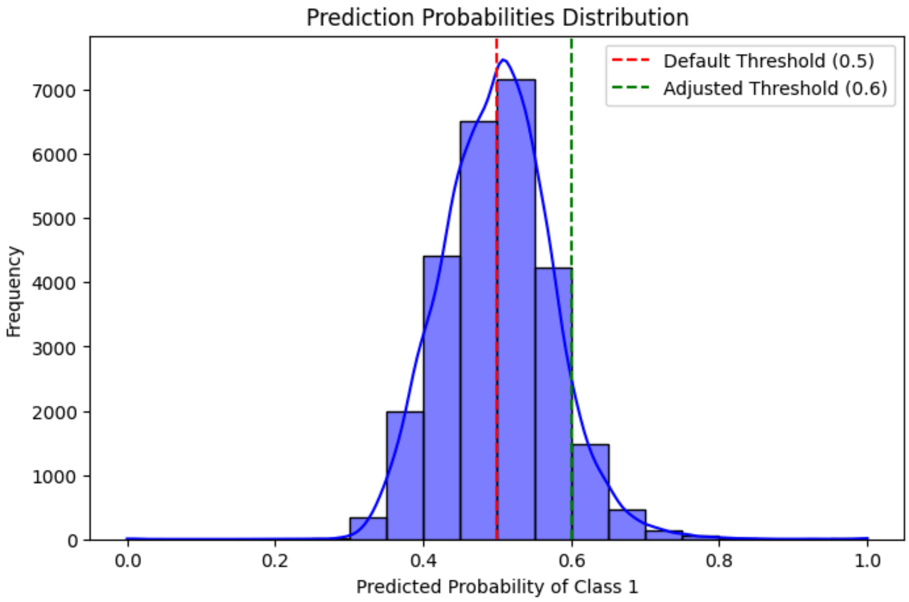
threshold = 0.6 # Change this to tune performance
y_pred_adjusted = (y_prob >= threshold).astype(int)
# Check performance
print("Adjusted Confusion Matrix:\n", confusion_matrix(y_test, y_pred_adjusted))
print("Adjusted Classification Report:\n", classification_report(y_test, y_pred_adjusted))



```
Accuracy: 0.5565201192250373
Confusion Matrix:
 [[7224 5852]
 [6051 7713]]
Classification Report:
               precision    recall  f1-score   support

           0       0.54      0.55      0.55     13076
           1       0.57      0.56      0.56     13764

    accuracy                           0.56     26840
   macro avg       0.56      0.56      0.56     26840
weighted avg       0.56      0.56      0.56     26840
```

Evaluating Threshold Impact on Model Performance

```
plt.figure(figsize=(8, 5))
sns.histplot(y_prob, bins=20, kde=True, color="blue")
plt.axvline(0.5, color="red", linestyle="dashed", label="Default Threshold (0.5)")
plt.axvline(0.6, color="green", linestyle="dashed", label="Adjusted Threshold (0.6)")
plt.xlabel("Predicted Probability of Class 1")
plt.ylabel("Frequency")
plt.title("Prediction Probabilities Distribution")
plt.legend()
plt.show()
```

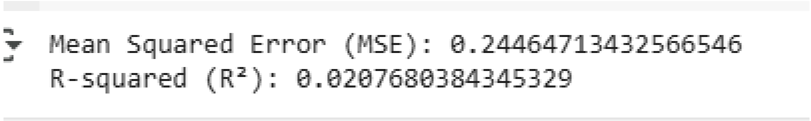


Mean Squared Error and R² Score Calculation
In this step, we evaluate the model’s prediction performance using Mean Squared Error (MSE) and R² Score:

- MSE (Mean Squared Error): Measures the average squared difference between actual and predicted probabilities. A lower MSE indicates better predictions.
- R² Score (Coefficient of Determination): Measures how well the predicted probabilities explain the variability in the actual values. A value closer to 1 suggests better model performance, while a value closer to 0 indicates poor predictive power.

```
from sklearn.metrics import mean_squared_error, r2_score
# Get predicted probabilities of class 1
```

```
y_prob = log_reg.predict_proba(X_test_scaled)[:, 1] # Probabilities instead of 0/1 predictions
# Compute MSE
mse = mean_squared_error(y_test, y_prob)
print("Mean Squared Error (MSE):", mse)
# Compute R² score
r2 = r2_score(y_test, y_prob)
print("R-squared (R²):", r2)
```



```
Mean Squared Error (MSE): 0.24464713432566546
R-squared (R²): 0.0207680384345329
```

Conclusion:

In this experiment, we implemented both Logistic Regression and Linear Regression to classify users based on selected features, but both models achieved an accuracy of only ~56%. While Logistic Regression is designed for classification, the low accuracy suggests that the data is not well-separated, possibly due to overlapping class distributions or weak predictive features. Linear Regression, on the other hand, is meant for continuous predictions, and converting its outputs to binary values further reduces classification performance. The confusion matrices and classification reports indicate misclassification in both models, reinforcing the need for better feature engineering or a more advanced model. To improve accuracy, we can explore non-linear models like Decision Trees or Neural Networks to optimize model performance.