Name: Mohit Kerkar **Div: D15C** **Roll No: 23**

EXPERIMENT NO.6

Aim: To implement Classification modelling

A. **Choose a classifier for classification problems.**
B. **Evaluate the performance of classifier**

Perform Classification using the below 4 classifiers on the same dataset:

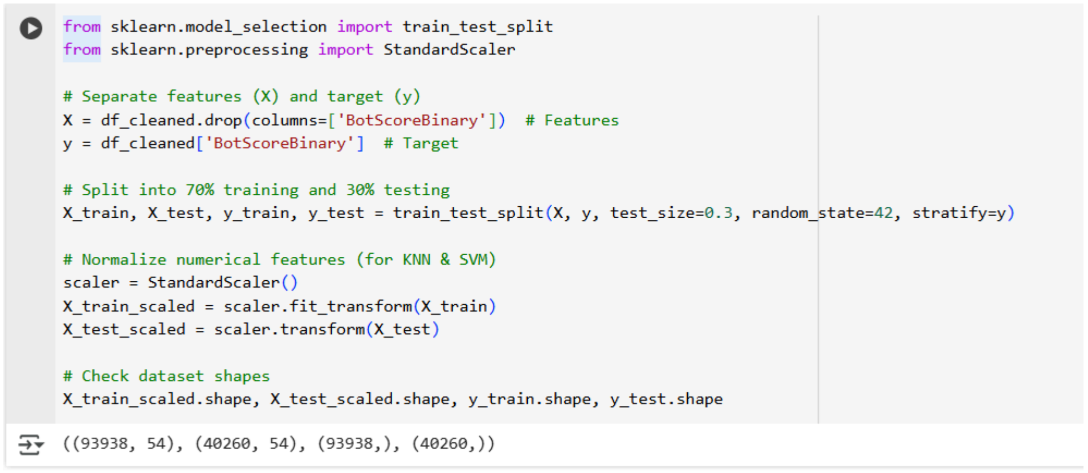
1. **K-Nearest Neighbors (KNN)**
2. **Naive Bayes**
3. **Support Vector Machines (SVMs)**
4. **Decision Tree**

Theory:

1. K-Nearest Neighbors (KNN):

K-Nearest Neighbors (KNN) is a supervised learning algorithm used for classification and regression, making predictions based on the majority class or average of the k closest data points. It determines similarity using distance metrics such as Euclidean, Manhattan, or Minkowski distance. As a non-parametric and instance-based method, KNN is simple to implement and effective for small datasets. However, it can be computationally expensive for large datasets and is sensitive to irrelevant features and the choice of k, which significantly impacts its performance.

Effective preprocessing enhances model performance by preparing data for training. This involves separating features (X) and target labels (y), then splitting the dataset into training (70%) and testing (30%) sets. Since models like KNN and SVM are sensitive to feature scales, numerical features are standardized using StandardScaler to ensure uniformity, improving model accuracy and efficiency.

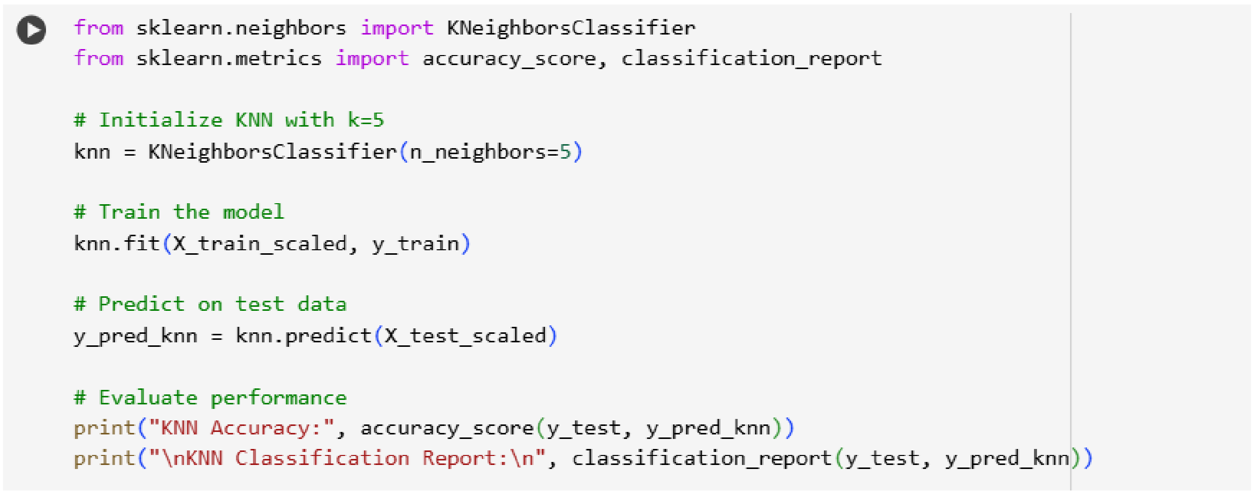


```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Separate features (X) and target (y)
X = df_cleaned.drop(columns=['BotScoreBinary'])  # Features
y = df_cleaned['BotScoreBinary']  # Target

# Split into 70% training and 30% testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)

# Normalize numerical features (for KNN & SVM)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Check dataset shapes
X_train_scaled.shape, X_test_scaled.shape, y_train.shape, y_test.shape
```

```
((93938, 54), (40260, 54), (93938,), (40260,))
```

After preprocessing the data, the K-Nearest Neighbors (KNN) algorithm is implemented for classification. The model is initialized with k = 5, meaning it considers the five closest data points when making predictions. The training process involves fitting the model to the scaled training data, allowing it to learn patterns based on feature similarities.

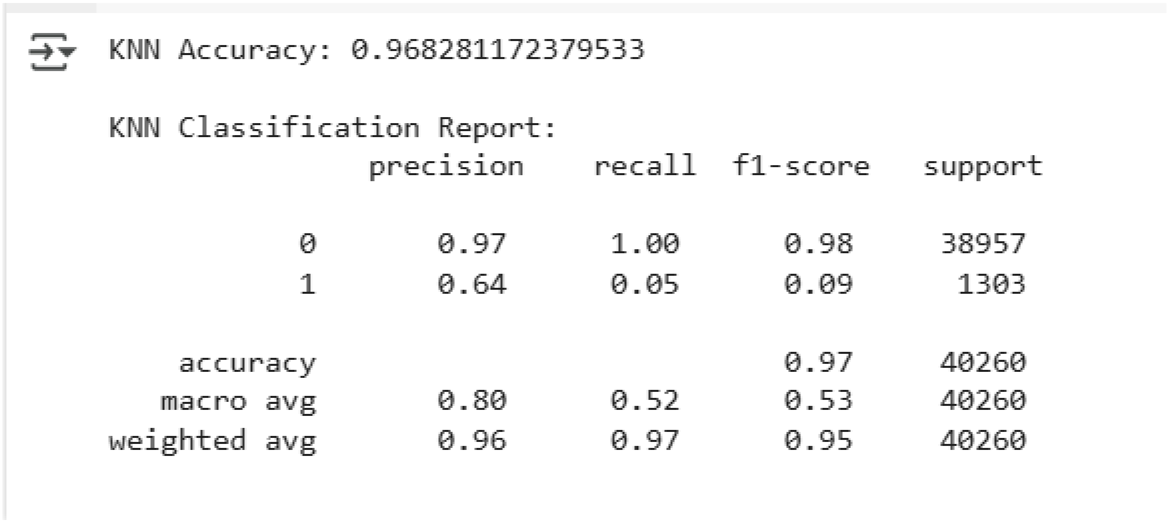
Name: Mohit Kerkar **Div: D15C** **Roll No: 23**



```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report

# Initialize KNN with k=5
knn = KNeighborsClassifier(n_neighbors=5)

# Train the model
knn.fit(X_train_scaled, y_train)

# Predict on test data
y_pred_knn = knn.predict(X_test_scaled)

# Evaluate performance
print("KNN Accuracy:", accuracy_score(y_test, y_pred_knn))
print("\nKNN Classification Report:\n", classification_report(y_test, y_pred_knn))
```

1. The KNN classifier achieved 96.83% accuracy, indicating strong overall performance. However, an in-depth analysis of the classification report highlights an imbalance in predictive capability between the two classes:

- Class 0 (Not a Bot): The model performed exceptionally well, with 97% precision and 100% recall, meaning almost all non-bot accounts were correctly classified.
- Class 1 (Bot): The model struggled with detecting bots, achieving 64% precision but only 5% recall, indicating that a large proportion of actual bot accounts were misclassified as non-bots.



```
KNN Accuracy: 0.968281172379533

KNN Classification Report:
               precision    recall  f1-score   support

           0       0.97      1.00      0.98     38957
           1       0.64      0.05      0.09      1303

    accuracy                           0.97     40260
   macro avg       0.80      0.52      0.53     40260
weighted avg       0.96      0.97      0.95     40260
```

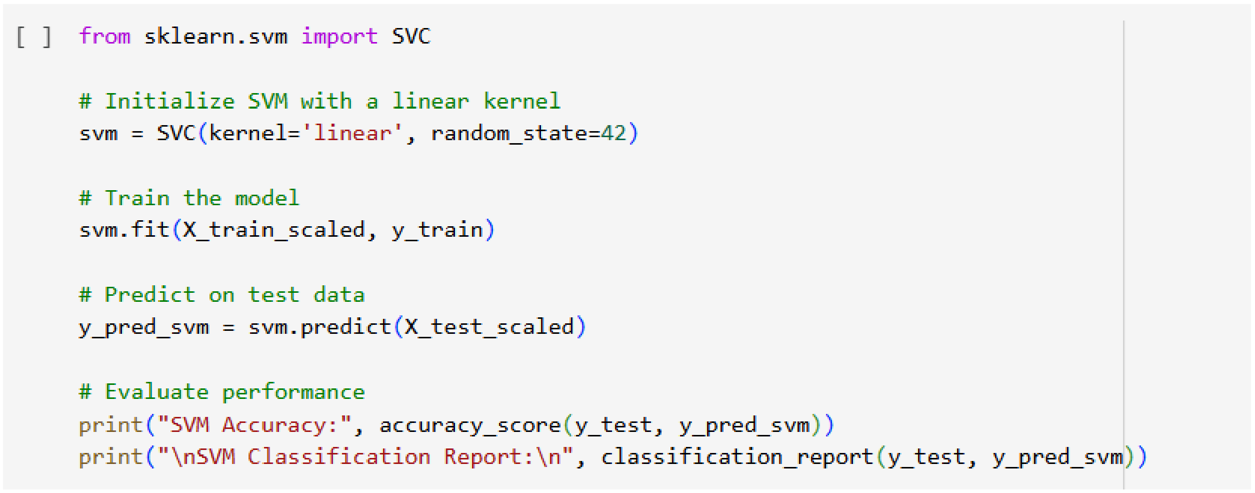
2. From the classification report, we observe:

- True Negatives (TN): Majority of non-bot accounts were correctly classified.
- False Positives (FP): A small number of non-bot accounts were misclassified as bots.
- False Negatives (FN): A significant number of actual bot accounts were misclassified as non-bots.
- True Positives (TP): Very few bot accounts were correctly identified.

3. While the model performs well overall, the extremely low recall for bot detection suggests that many bots are not being correctly identified. This is likely due to class imbalance, where

non-bot accounts dominate the dataset.

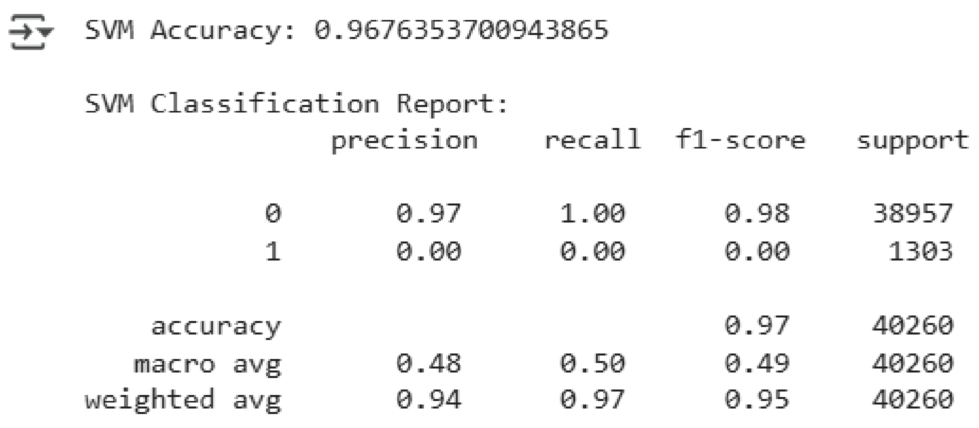
2. Support Vector Machines (SVMs):
A Support Vector Machine (SVM) finds the optimal hyperplane to separate classes, using support vectors to maximize the margin. It employs the kernel trick (Linear, Polynomial, RBF) for non-linearly separable data. SVMs perform well on high-dimensional data and small datasets but can be computationally expensive and require careful kernel selection.



```
[ ]  from sklearn.svm import SVC

     # Initialize SVM with a linear kernel
     svm = SVC(kernel='linear', random_state=42)

     # Train the model
     svm.fit(X_train_scaled, y_train)

     # Predict on test data
     y_pred_svm = svm.predict(X_test_scaled)

     # Evaluate performance
     print("SVM Accuracy:", accuracy_score(y_test, y_pred_svm))
     print("\nSVM Classification Report:\n", classification_report(y_test, y_pred_svm))
```

1. The SVM classifier achieved 96.76% accuracy, indicating strong overall performance. However, a deeper analysis of the classification report highlights a severe imbalance in predictive capability between the two classes:

- Class 0 (Not a Bot): The model performed exceptionally well, with 97% precision and 100% recall, meaning nearly all non-bot accounts were correctly classified.
- Class 1 (Bot): The model failed to detect bot accounts, achieving 0% precision and 0% recall, meaning that no actual bots were correctly classified.



```
SVM Accuracy: 0.9676353700943865

SVM Classification Report:
               precision    recall  f1-score   support

           0       0.97      1.00      0.98     38957
           1       0.00      0.00      0.00      1303

    accuracy                           0.97     40260
   macro avg       0.48      0.50      0.49     40260
weighted avg       0.94      0.97      0.95     40260
```

2. From the classification report, we observe:

- True Negatives (TN): Almost all non-bot accounts were correctly classified.

- False Positives (FP): A small number of non-bot accounts were misclassified as bots.
- False Negatives (FN): All actual bot accounts were misclassified as non-bots.
- True Positives (TP): None of the bot accounts were correctly identified.

3. While the model appears to perform well overall, the complete failure in identifying bots (Class 1) indicates a major issue, likely due to class imbalance. The dominance of non-bot accounts in the dataset causes SVM to heavily favor classifying all instances as non-bots.

Conclusion:
Both the K-Nearest Neighbors (KNN) and Support Vector Machine (SVM) models achieved high overall accuracy (96.83% and 96.76%, respectively) in classifying Twitter accounts as bots or non-bots. However, further analysis reveals that accuracy alone is misleading due to severe class imbalance in the dataset.

- KNN performed slightly better in detecting bots, achieving 64% precision but only 5% recall, meaning that while some bot accounts were correctly identified, the model still misclassified the majority of them as non-bots.

- SVM completely failed to identify bots, with 0% precision and 0% recall, meaning it classified all accounts as non-bots, making it ineffective for bot detection.