Name : Himanshu Naik **Div : D15c** **Roll No : 63**

Experiment No. - 6 :

Problem Statement: Classification modelling
a. Choose a classifier for a classification problem.
b. Evaluate the performance of the classifier.
Perform Classification using the below 4 classifiers on the same dataset which you have used for

- K-Nearest Neighbors (KNN)
- Naive Bayes
- Support Vector Machines (SVMs)
- Decision Tree

Introduction :

Classification algorithms are crucial in predicting categorical outcomes and analyzing labeled data. In this experiment, we applied two supervised learning techniques — Naive Bayes and K-Nearest Neighbors (KNN) — to classify air quality based on the Air Quality Index (AQI) dataset.

Naive Bayes is a probabilistic classifier based on Bayes' theorem, known for its simplicity, efficiency, and effectiveness, especially with small datasets or when feature independence is assumed. On the other hand, KNN is a distance-based classifier that assigns a class label based on the majority of the nearest neighbors, making it intuitive yet sensitive to feature scaling and noise.

The primary objective was to predict the trip duration of NYC taxi rides based on features such as pickup and drop-off coordinates, time-related variables, and passenger count. By applying various regression algorithms and evaluating their performance

Implementation Process :

Step 1 : We perform a series of data pre-processing operations that involve calculating the missing percentage for each column ,using mean imputation for columns that have less than 20 % missing values,using iterative imputation methods for columns between 20 - 50% missing values and dropping the xylene column that had around 60 % missing values.

```
from sklearn.impute import SimpleImputer

mean_imputer = SimpleImputer(strategy='mean')

columns_to_impute = ['passenger_count', 'pickup_longitude', 'pickup_latitude',
                     'dropoff_longitude', 'dropoff_latitude', 'trip_duration']

for col in columns_to_impute:
    if col in df.columns:
        df[col] = mean_imputer.fit_transform(df[[col]])
```

```
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer

# Define the iterative imputer
iter_imputer = IterativeImputer(max_iter=10, random_state=42)

# Select numeric columns with 20%-50% missing values (example list — update based on actual %)
columns_to_impute = ['pickup_longitude', 'pickup_latitude',
                     'dropoff_longitude', 'dropoff_latitude']

# Apply the imputer if these columns exist
existing_columns = [col for col in columns_to_impute if col in df.columns]
df[existing_columns] = iter_imputer.fit_transform(df[existing_columns])

# Show remaining missing values
print("Remaining Missing Values:\n", df[existing_columns].isnull().sum())
```

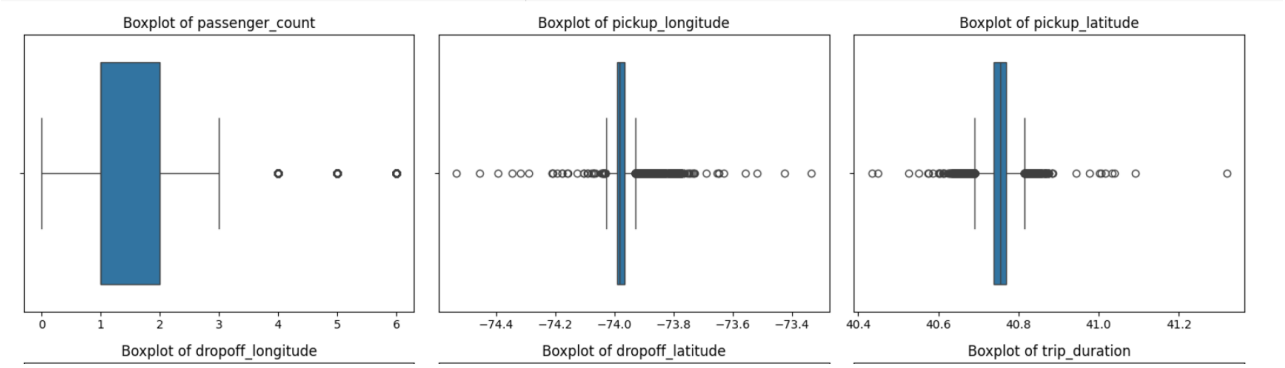
```
df.drop('store_and_fwd_flag', axis=1, inplace=True)
```

Step 2 : We verify that all of the missing values have been removed and then move to identifying and eliminating the outliers using boxplot analysis.

```
import matplotlib.pyplot as plt
import seaborn as sns

# Relevant numeric columns in NYC Taxi dataset
numeric_columns = ['passenger_count', 'pickup_longitude', 'pickup_latitude',
                   'dropoff_longitude', 'dropoff_latitude', 'trip_duration']

# Plot boxplots
plt.figure(figsize=(15, 8))
for i, col in enumerate(numeric_columns, 1):
    plt.subplot(2, 3, i)
    sns.boxplot(x=df[col])
    plt.title(f'Boxplot of {col}')
    plt.xlabel("")  # Optional: Hide x-label if too long
plt.tight_layout()
plt.show()
```



Step 3 : Apply standardizing operations using the z-score method.

```
import numpy as np

# Define threshold for z-scores
threshold = 3

# NYC Taxi dataset's relevant numeric columns
numeric_columns = ['passenger_count', 'pickup_longitude', 'pickup_latitude',
                   'dropoff_longitude', 'dropoff_latitude', 'trip_duration']

# Apply z-score based capping
for col in numeric_columns:
    mean_val = df[col].mean()
    std_dev = df[col].std()

    # Clip values outside ±3 standard deviations
    df[col] = np.clip(df[col], mean_val - threshold * std_dev, mean_val + threshold * std_dev)
```

Step 4 : We then encode categorical features in the dataset to prepare it for machine learning algorithms, as our model requires numerical data.

The LabelEncoder converts the categorical values of the AQI_Bucket column (like "Good", "Satisfactory", "Poor", etc.) into integer labels (0, 1, 2, etc.).

One-Hot Encoding creates binary columns for each unique value in the City column (like Delhi, Mumbai, etc.).

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
df['trip_duration_bucket'] = le.fit_transform(df['trip_duration_bucket'])

print("Label Encoded 'trip_duration_bucket':")
print(df['trip_duration_bucket'].unique())
```

Step 5 : We then perform splitting of the dataset into a 70 : 30 split to proceed with the preparation and evaluation of the models.

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

# Dropping unnecessary columns (like IDs and datetime)
X = df.drop(['id', 'pickup_datetime', 'dropoff_datetime', 'trip_duration'], axis=1)  # Features
y = df['trip_duration']  # Target variable

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

# Standardizing numerical features only
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Convert back to DataFrame (optional)
X_train_scaled = pd.DataFrame(X_train_scaled, columns=X.columns)
X_test_scaled = pd.DataFrame(X_test_scaled, columns=X.columns)

print("Scaled Features (First 5 rows):")
print(X_train_scaled.head())
```

Step 6 : The preparation of the Naive Bayes Classifier for application on to the dataset.

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Handle missing values and format 'trip_duration_bucket'
df['trip_duration_bucket'] = df['trip_duration_bucket'].fillna('Unknown').astype(str)

# Drop datetime-related columns and set features/target
X = df.drop(columns=['trip_duration_bucket', 'pickup_datetime', 'dropoff_datetime'])
y = df['trip_duration_bucket']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Standardizing only numeric features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train.select_dtypes(include=[float, int]))
X_test_scaled = scaler.transform(X_test.select_dtypes(include=[float, int]))

# Naive Bayes Model
nb_model = GaussianNB()
nb_model.fit(X_train_scaled, y_train)
nb_pred = nb_model.predict(X_test_scaled)

# Evaluation
print(" 🔹 Naive Bayes Evaluation 🔹 ")
print(f"Accuracy: {accuracy_score(y_test, nb_pred):.2f}")
print("Classification Report:\n", classification_report(y_test, nb_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, nb_pred))
```

This classifier is then evaluated for accuracy and a classification report and a confusion matrix is prepared for the same.

```
 🔹 Naive Bayes Evaluation 🔹
Accuracy: 0.95
Classification Report:
              precision    recall  f1-score   support

           0       0.79      1.00      0.88       191
           1       0.92      0.94      0.93      1352
           2       0.95      0.93      0.94      6611
           3       0.95      0.97      0.96     12771
           4       0.97      0.84      0.90       403
           5       0.96      0.94      0.95      3834

    accuracy                           0.95     25162
   macro avg       0.92      0.93      0.93     25162
weighted avg       0.95      0.95      0.95     25162

Confusion Matrix:
 [[  191     0     0     0     0     0]
 [    0  1266    75     0    11     0]
 [    0    91  6133   387     0     0]
 [    0     0   242 12373     0   156]
 [   50    16     0     0   337     0]
 [    0     0     0   240     0  3594]]
```

Step 7 : The preparation of the KNN (K-Nearest Neighbors) for application on to the dataset.

```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Make sure the target column is clean
df['trip_duration_bucket'] = df['trip_duration_bucket'].fillna('Unknown').astype(str)

# Drop datetime columns and define features/target
X = df.drop(columns=['trip_duration_bucket', 'pickup_datetime', 'dropoff_datetime'])
y = df['trip_duration_bucket']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Standardizing only numeric features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train.select_dtypes(include=[float, int]))
X_test_scaled = scaler.transform(X_test.select_dtypes(include=[float, int]))

# KNN Classifier with distance-based weighting
knn_model = KNeighborsClassifier(n_neighbors=5, weights='distance', n_jobs=-1)
knn_model.fit(X_train_scaled, y_train)
knn_pred = knn_model.predict(X_test_scaled)

# Evaluation
print(" 🔹 Optimized KNN Evaluation 🔹 ")
print(f"Accuracy: {accuracy_score(y_test, knn_pred):.2f}")
print("Classification Report:\n", classification_report(y_test, knn_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, knn_pred))
```

This classifier is then evaluated for accuracy and a classification report and a confusion matrix is prepared for the same.

```
🔹 Optimized KNN Evaluation 🔹
Accuracy: 0.89
Classification Report:
              precision    recall  f1-score   support

           0       0.48      0.32      0.38       191
           1       0.63      0.55      0.59      1352
           2       0.85      0.88      0.86      6611
           3       0.93      0.95      0.94     12771
           4       0.53      0.32      0.39       403
           5       0.94      0.91      0.93      3834

    accuracy                           0.89     25162
   macro avg       0.73      0.65      0.68     25162
weighted avg       0.88      0.89      0.88     25162

Confusion Matrix:
 [[   61    55    10     0    65     0]
 [   14   749   540     0    49     0]
 [    0   174  5789   648     0     0]
 [    0     0   455 12104     0   212]
 [   53   202    21     0   127     0]
 [    0     0     0   330     0  3504]]
```

Conclusion :

Conclusion

Based on the classification results obtained from both the **Naive Bayes** and the **optimized K-Nearest Neighbors (KNN)** algorithms, the following conclusions can be drawn:

- **Model Performance**: The **Naive Bayes model achieved a higher accuracy of 95%**, compared to **89% for the optimized KNN model**. This indicates that **Naive Bayes is more effective** in accurately classifying trip durations into predefined categories for this dataset.

- **Evaluation Metrics**: Naive Bayes consistently outperformed KNN across all major metrics—**precision, recall, and F1-score**—especially for classes with larger support (like class labels 2, 3, and 5). This indicates that Naive Bayes is more reliable in correctly identifying the correct duration classes.

- **Confusion Matrix Analysis**: The **confusion matrix for Naive Bayes** shows fewer misclassifications across all classes compared to KNN. In contrast, the **KNN model misclassified a notable number of samples**, particularly between adjacent duration classes, suggesting it struggled more with borderline cases.

- **Model Suitability**: While **KNN** is a strong non-parametric model that can capture complex decision boundaries, its performance is **limited by the curse of dimensionality** and sensitivity to noisy data. **Naive Bayes**, on the other hand, demonstrates strong performance even with its assumption of feature independence, making it **computationally efficient and highly accurate** for this classification task.