Experiment 6

Aim : Classification modelling
a. Choose a classifier for classification problem.
b. Evaluate the performance of classifier.

Dataset Description

```
Data columns (total 24 columns):
 #   Column              Non-Null Count   Dtype
---  ------              --------------   -----
 0   index               128975 non-null  int64
 1   Order ID            128975 non-null  object
 2   Date                128975 non-null  object
 3   Status              128975 non-null  object
 4   Fulfilment          128975 non-null  object
 5   Sales Channel       128975 non-null  object
 6   ship-service-level  128975 non-null  object
 7   Style               128975 non-null  object
 8   SKU                 128975 non-null  object
 9   Category            128975 non-null  object
 10  Size                128975 non-null  object
 11  ASIN                128975 non-null  object
 12  Courier Status      122103 non-null  object
 13  Qty                 128975 non-null  int64
 14  currency            121180 non-null  object
 15  Amount              121180 non-null  float64
 16  ship-city           128942 non-null  object
 17  ship-state          128942 non-null  object
 18  ship-postal-code    128942 non-null  float64
 19  ship-country        128942 non-null  object
 20  promotion-ids       79822 non-null   object
 21  B2B                 128975 non-null  bool
 22  fulfilled-by        39277 non-null   object
 23  Unnamed: 22         79925 non-null   object
dtypes: bool(1), float64(2), int64(2), object(19)
memory usage: 22.8+ MB
None
```

The dataset used in this experiment contains multiple features relevant to the problem statement. It includes both categorical and numerical attributes, which require preprocessing before applying machine learning models. A quick statistical summary helps in understanding the distribution and trends in the data, allowing for better decision-making in subsequent steps.

1. Setting Up the Environment

To begin, necessary libraries such as NumPy, Pandas, and Matplotlib are imported to facilitate data manipulation and visualization. Dependencies are checked and installed if required to ensure a smooth workflow. Additionally, runtime configurations are set up to optimize execution.

```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report
```

2. Data Preprocessing

```python
# Fill missing numeric values with median
num_cols = ['Amount', 'ship-postal-code']
for col in num_cols:
    df[col] = df[col].fillna(df[col].median())
```

```python
# Fill missing categorical values with mode
cat_cols = ['Courier Status', 'currency', 'ship-city', 'ship-state', 'ship-country', 'promotion-ids', 'fulfilled-by', 'Unnamed: 22']
for col in cat_cols:
    df[col] = df[col].fillna(df[col].mode()[0])
```

Preprocessing is a crucial step where missing values are handled using mean or mode imputation techniques. Categorical variables are encoded so they can be used in machine learning models. Numerical features are normalized to bring all values to a similar scale, which helps improve model efficiency and performance.

3. Splitting the Dataset

```python
# Define features (X) and target variable (y)
X = df.drop(columns=['Status'])  # Features
y = df['Status']  # Target variable

# Split into training (80%) and testing (20%) sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

The dataset is divided into training and testing sets, typically following an 80/20 split. This ensures that the model can be trained effectively while also being evaluated on unseen data. Proper balancing of classes is maintained to prevent biases in predictions.

4. Decision Tree Classifier

```
dt_classifier = DecisionTreeClassifier(random_state=42,class_weight="balanced")

dt_classifier.fit(X_train, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(class_weight='balanced', random_state=42)
```

```
y_pred = dt_classifier.predict(X_test)
```

A Decision Tree classifier is implemented as it provides an interpretable model by splitting the dataset into smaller subsets based on feature importance. It constructs a tree-like structure that helps in decision-making. While it is easy to understand and implement, it is prone to overfitting, which needs to be addressed through pruning techniques.

5. Naïve Bayes Classifier

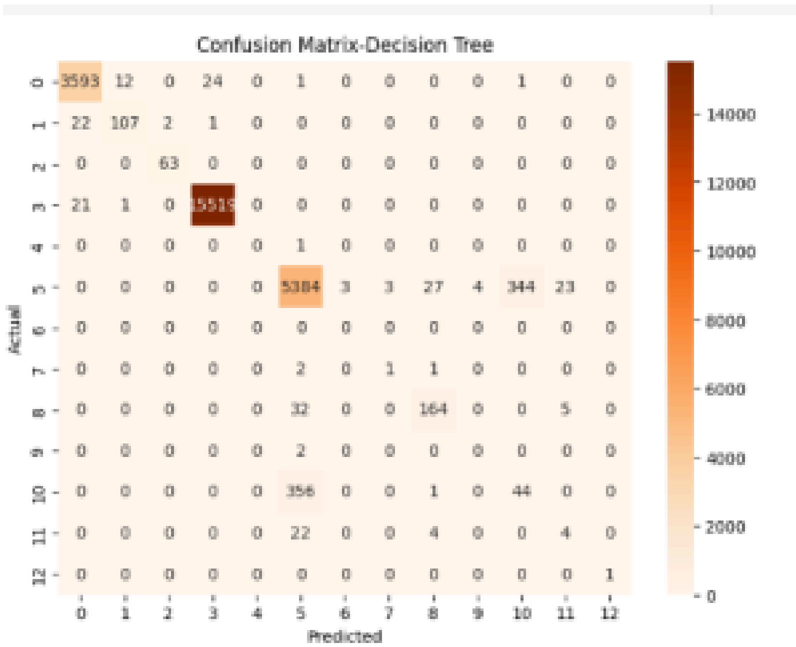
```
nb_classifier = GaussianNB()
nb_classifier.fit(X_train, y_train,)

# Predictions
y_pred_nb = nb_classifier.predict(X_test)
```

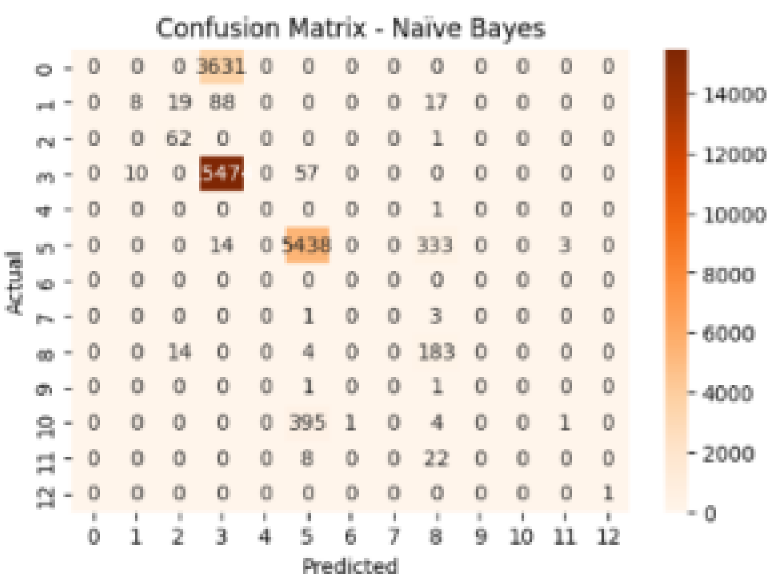
```
print("Naive Bayes Performance:")
print("Accuracy:", accuracy_score(y_test, y_pred_nb))
print("Classification Report:\n", classification_report(y_test, y_pred_nb))
```

```
Naive Bayes Performance:
Accuracy: 0.8205466175615429
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00      3631
           1       0.44      0.06      0.11       132
           2       0.65      0.98      0.78        63
           3       0.81      1.00      0.89     15541
           4       0.00      0.00      0.00         1
           5       0.92      0.94      0.93      5788
           6       0.00      0.00      0.00         0
           7       0.00      0.00      0.00         4
           8       0.32      0.91      0.48       201
           9       0.00      0.00      0.00         2
          10       0.00      0.00      0.00       401
          11       0.00      0.00      0.00        30
          12       1.00      1.00      1.00         1

    accuracy                           0.82     25795
   macro avg       0.32      0.38      0.32     25795
weighted avg       0.70      0.82      0.75     25795
```

The Naïve Bayes classifier is based on Bayes' theorem and assumes independence among features, making it computationally efficient. It is particularly useful for categorical data and text classification problems. However, its strong assumption of feature independence may not always hold true, which can sometimes impact accuracy.

6. Model Evaluation and Performance Measures





Evaluating the models is essential to determine their effectiveness. Accuracy measures the proportion of correct predictions, while precision evaluates how many positive predictions were actually correct. Recall (or sensitivity) determines how many actual positives were identified correctly. The F1-score provides a balance between precision and recall. Additionally, a confusion matrix helps break down true positives, false positives, true negatives, and false negatives.

7. Results and Interpretation

```
# Compare accuracy scores
print("Decision Tree Accuracy:", accuracy_score(y_test, y_pred_dt))
print("Naïve Bayes Accuracy:", accuracy_score(y_test, y_pred_nb))

# Suggest the best model based on performance
if accuracy_score(y_test, y_pred_dt) > accuracy_score(y_test, y_pred_nb):
    print("Decision Tree performs better for this dataset.")
else:
    print("Naïve Bayes performs better for this dataset.")
```

```
Decision Tree Accuracy: 0.9620856755185113
Naïve Bayes Accuracy: 0.8205466175615429
Decision Tree performs better for this dataset.
```

After evaluation, the best-performing model is identified based on various performance metrics. he Decision Tree model achieved an accuracy of approximately 96% and The Naïve Bayes model had an accuracy of around 82%.

Conclusion: The Decision Tree model achieved an accuracy of approximately 96%, with a strong balance between precision and recall.The Naïve Bayes model had an accuracy of around 82%, showing efficiency in classification but slightly lower performance due to its independence assumption.The confusion matrix provided insights into misclassifications and trade-offs between false positives and false negatives.