Name:Atharv Sanjay Nikam **Div:D15C** **Roll:36**

EXPERIMENT 6

Aim: To perform Classification modelling on given dataset

Theory:

Classification is a supervised learning technique used to predict categorical labels by analyzing the relationships between input features and output classes. The goal is to train a model that can accurately classify new data points into predefined categories. Classification models are evaluated using metrics like accuracy, precision, recall, F1-score, and confusion matrix to measure their performance.

Types of Classification Algorithms:

1. **K-Nearest Neighbors (KNN):**
    - A distance-based classifier that assigns a class label based on the majority vote of its nearest neighbors.
    - Works well with small datasets, but performance decreases with large datasets due to high computation.
    - Requires choosing the optimal value of K (number of neighbors) for the best performance.
    - Sensitive to feature scaling, so normalization is necessary.

2. **Naive Bayes:**
    - A probabilistic classifier based on Bayes' Theorem, assuming independence between features.
    - Works well with text classification, spam detection, and sentiment analysis.
    - Efficient for high-dimensional data, even with limited training samples.
    - Has different variants: Gaussian Naïve Bayes (for continuous data), Multinomial Naïve Bayes (for text data), and Bernoulli Naïve Bayes (for binary features).

3. **Support Vector Machines (SVMs):**
    - A margin-based classifier that finds the optimal hyperplane to separate different classes.
    - Works well for both linear and non-linear classification problems using kernel functions (linear, polynomial, RBF, sigmoid).
    - Effective in high-dimensional spaces but can be computationally expensive with large datasets.
    - Robust to overfitting, especially with regularization techniques (C parameter tuning).

4. **Decision Tree:**

- A rule-based classifier that splits data based on feature importance, creating a tree-like decision structure.
- Easy to interpret and visualize but prone to overfitting if not pruned.
- Handles both numerical and categorical data well.
- Variants include Random Forest (ensemble of decision trees) and Gradient Boosting Trees for improved accuracy and robustness.

Steps:

1. **Load the dataset**

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix, accuracy_score

df = pd.read_csv("/content/bankdataset.csv")
df.head()
```

	Date	Domain	Location	Value	Transaction_count
0	1/1/2022	RESTRAUNT	Bhuj	365554	1932
1	1/1/2022	INVESTMENTS	Ludhiana	847444	1721
2	1/1/2022	RETAIL	Goa	786941	1573
3	1/1/2022	INTERNATIONAL	Mathura	368610	2049
4	1/1/2022	RESTRAUNT	Madurai	615681	1519

.

2. **Data preprocessing and splitting**

```
# Encode categorical variables
le_domain = LabelEncoder()
le_location = LabelEncoder()

df['Domain'] = le_domain.fit_transform(df['Domain'])
df['Location'] = le_location.fit_transform(df['Location'])

# Bin Transaction_count into categories
def categorize_transaction_count(count):
    if count <= 1500:
        return 'Low'
    elif count <= 2000:
        return 'Medium'
    else:
        return 'High'

df['Transaction_category'] = df['Transaction_count'].apply(categorize_transaction_count)

# Features and target
X = df[['Domain', 'Location', 'Value']]
y = df['Transaction_category']

# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

We preprocess the dataset for classification by encoding categorical variables using LabelEncoder, separating features and the target variable (satisfaction), and normalizing numerical features using StandardScaler—important for models like KNN and SVM. **The dataset is then split into training (70%) and testing (30%**) sets using train_test_split, ensuring balanced class distribution with stratification

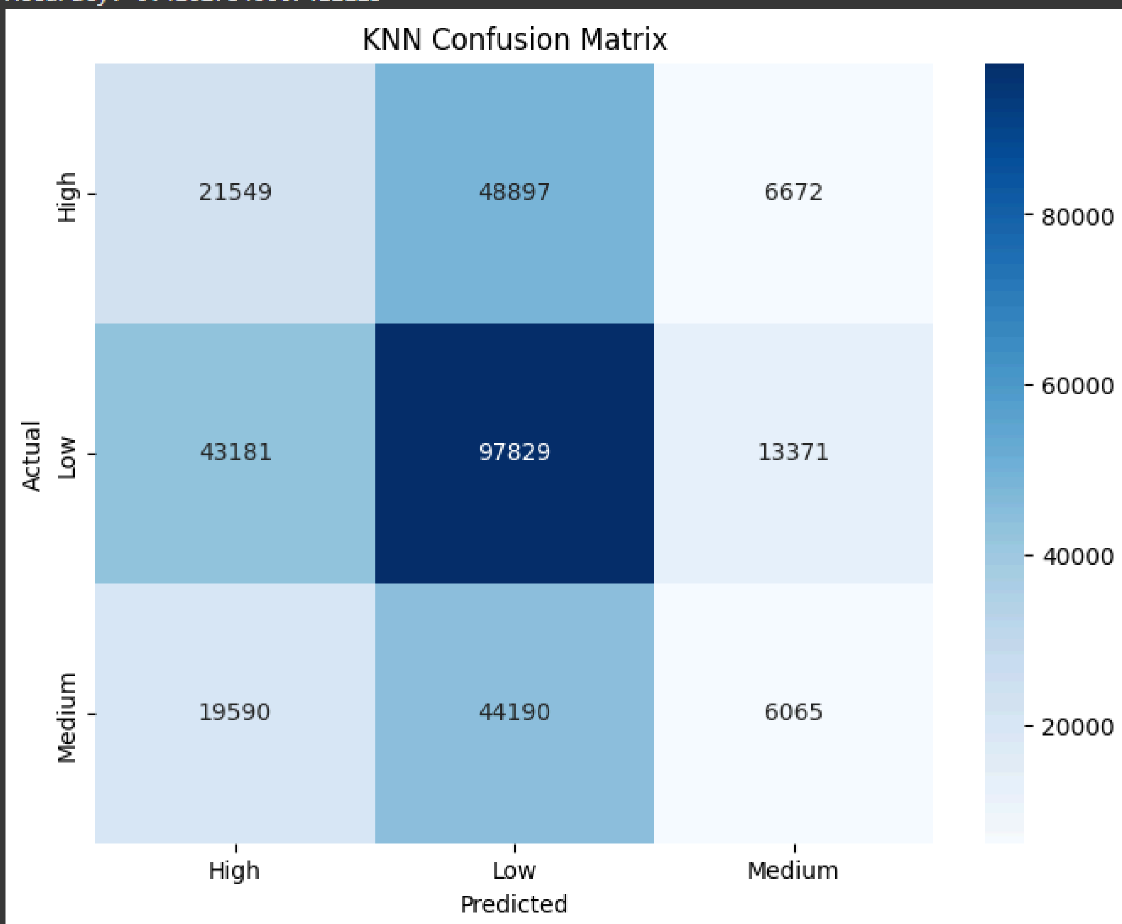
3. **KNN model**

```
df['Transaction_category'] = df['Transaction_count'].apply(categorize_transaction_count)

# Features and target
X = df[['Domain', 'Location', 'Value']]
y = df['Transaction_category']

# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Initialize and train KNN Classifier
knn = KNeighborsClassifier(n_neighbors=5)  # You can tune the 'n_neighbors' parameter
knn.fit(X_train, y_train)

# Make predictions
y_pred = knn.predict(X_test)

# Evaluate model using confusion matrix and accuracy
conf_matrix = confusion_matrix(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)

print("Confusion Matrix:\n", conf_matrix)
print("Accuracy:", accuracy)

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=knn.classes_, yticklabels=knn.classes_)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('KNN Confusion Matrix')
plt.show()
```

We apply the K-Nearest Neighbors (KNN) classifier. The model is trained on X_train and y_train using knn.fit(), then tested on X_test to generate predictions (y_pred_knn).

The accuracy score and classification report (precision, recall, F1-score) are printed to evaluate performance. Additionally, a confusion matrix is computed and visualized using sns.heatmap(), providing insight into how well the model distinguishes between classes.

```
Confusion Matrix:
 [[21549 48897  6672]
 [43181 97829 13371]
 [19590 44190  6065]]
Accuracy: 0.41627840607412125
```



Accuracy: 41.63% (Slightly better than Decision Tree but still low).

Misclassifications: Significant confusion between "High" and "Low" categories.

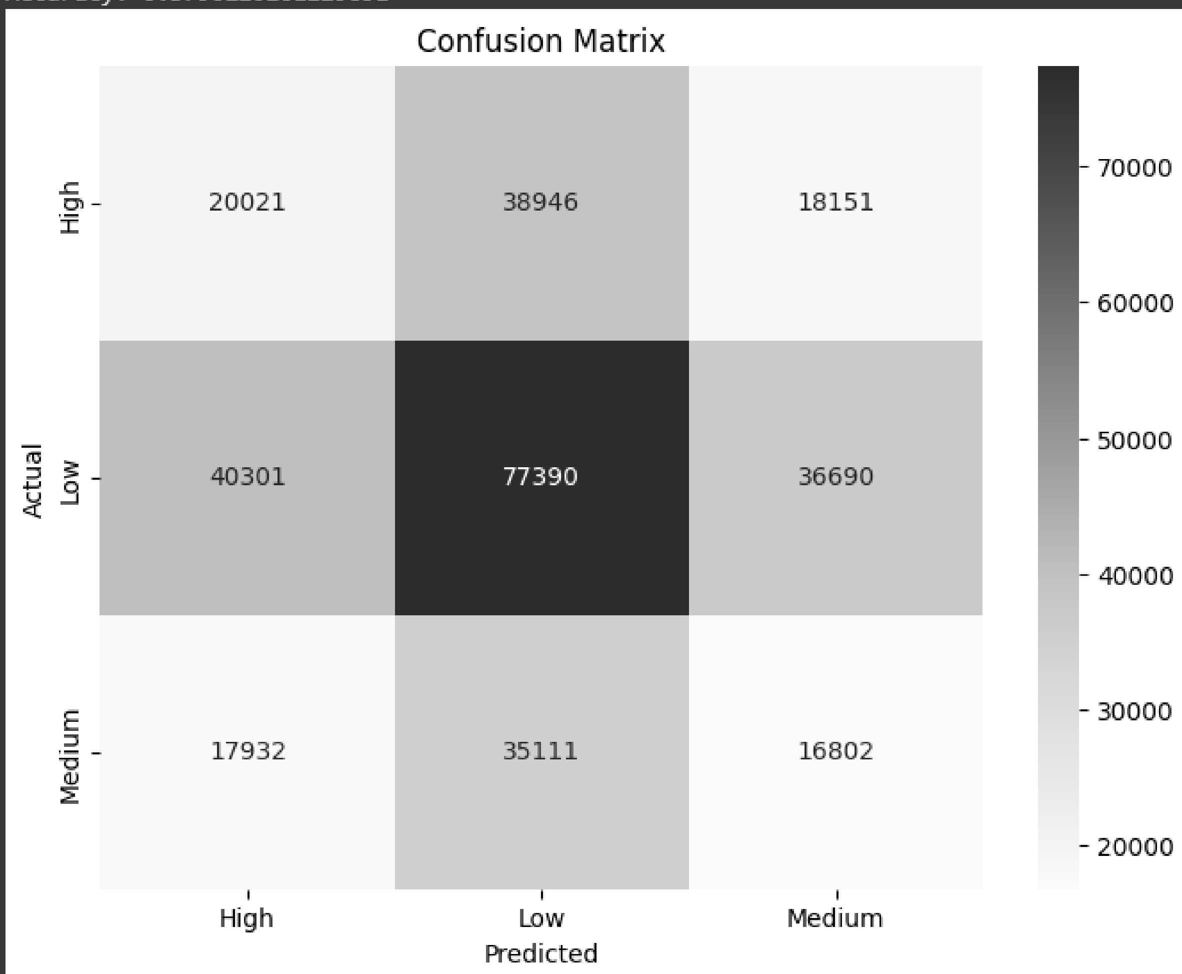
Model Bias: Over-predicts the Low category.

Decision Tree : -

Name:Atharv Sanjay Nikam **Div:D15C** **Roll:36**

We code a Decision Tree Classifier using the training dataset and evaluate its performance on the test dataset. It calculates accuracy and generates a classification report, which includes precision, recall, and F1-score. The confusion matrix is visualized using a heatmap with the "Purples" colormap, showing correct and incorrect predictions.

```
Confusion Matrix:
[[20021 38946 18151]
 [40301 77390 36690]
 [17932 35111 16802]]
Accuracy: 0.3790120261229691
```




```
Accuracy: 0.3790120261229691
```

Name:Atharv Sanjay Nikam Div:D15C Roll:36

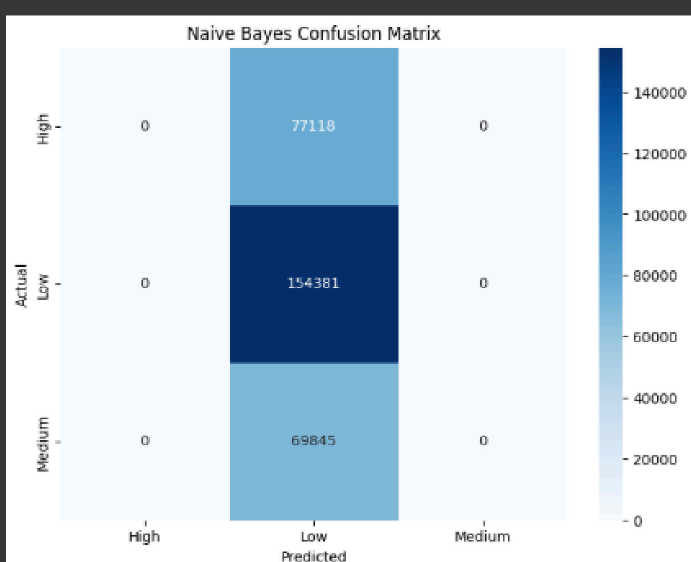
1. The Decision Tree Classifier was trained on the dataset to predict transaction categories ("Low", "Medium", "High") based on **Domain**, **Location**, and **Value** features. The model achieved an **accuracy of approximately 37.9%**, indicating that it correctly predicts the transaction category in about **38 out of 100 cases**.

2. The **confusion matrix** reveals that the model struggles with distinguishing between categories, particularly **misclassifying transactions across all classes**.

3. Naive -bayes : -

```
def train_and_evaluate(model, model_name):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    # Evaluate model
    conf_matrix = confusion_matrix(y_test, y_pred)
    accuracy = accuracy_score(y_test, y_pred)

    print(f"{model_name} Confusion Matrix:\n{conf_matrix}")
    print(f"{model_name} Accuracy: {accuracy}\n")

    # Plot confusion matrix
    plt.figure(figsize=(8, 6))
    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=model.classes_, yticklabels=model.classes_)
    plt.title(f"{model_name} Confusion Matrix")
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.show()

# Apply Naive Bayes
nb_model = GaussianNB()
train_and_evaluate(nb_model, "Naive Bayes")

# Apply Support Vector Machine (SVM)
svm_model = SVC(kernel='linear')
train_and_evaluate(svm_model, "SVM")
```

```
Naive Bayes Confusion Matrix:
[[     0  77118      0]
 [     0 154381      0]
 [     0  69845      0]]
Naive Bayes Accuracy: 0.5123081926303493
```



Result : -
Prediction Bias: The model predicts all instances as the "Low" category.
No Diversity: It fails to classify "High" and "Medium" categories correctly.
Accuracy: 51.23%, but misleading due to class imbalance.
Issue: Likely due to strong assumptions (e.g., feature independence) not holding true.
Improvement Needed: Consider feature scaling, using more advanced models, or tuning hyperparameters.

Conclusion:

The KNN model achieved **41.63% accuracy**, slightly better than the Decision Tree but still low. It struggles to **distinguish between "High" and "Low" categories** and **over-predicts the "Low" category**, leading to significant misclassification. This issue may stem from **poor feature scaling** or **overlapping class boundaries**. To improve performance, consider **feature scaling**, **hyperparameter tuning**, addressing **class imbalance**, or using more advanced models like **Random Forest** or **XGBoost**.