Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

Experiment No 6

Aim: 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 experiment no 5:

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

Theory:

Classification Modeling: Theory & Techniques

Classification modeling is a type of supervised learning in machine learning where the goal is to predict the category or class of a given data point based on input features. The model is trained using labeled data (i.e., data where the output class is known).

Classification problems can be:

- **Binary Classification:** Two classes (e.g., spam vs. not spam).
- **Multiclass Classification:** More than two classes (e.g., classifying types of flowers).
- **Multi-label Classification:** Each sample can belong to multiple classes.

1. K-Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN) is a simple, non-parametric classification algorithm based on proximity to labeled examples.

Working Principle:

1. Choose a value for K (number of neighbors).
2. Compute the distance between the new data point and all training samples.
3. Select the K nearest neighbors.
4. Assign the majority class among the K neighbors as the predicted class.

Common Distance Metrics:

- **Euclidean Distance:** $d=(\sum(xi-yi)^2)^{½}$ (Most commonly used)
- **Manhattan Distance:** $d=\sum|xi-yi|$
- **Minkowski Distance:** A generalization of Euclidean and Manhattan distances.

2. Naïve Bayes (NB)

Naïve Bayes is a probabilistic classifier based on **Bayes' Theorem**, assuming independence between predictors.

Bayes' Theorem:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Where:

- P(A|B) = Probability of class A given data B
- P(B|A) = Probability of data B given class A
- P(A) = Prior probability of class A
- P(B) = Prior probability of data B

Types of Naïve Bayes Classifiers:

1. **Gaussian Naïve Bayes:** Assumes normal distribution of features.
2. **Multinomial Naïve Bayes:** Used for text classification (e.g., spam detection).
3. **Bernoulli Naïve Bayes:** Used when features are binary (e.g., word presence in spam detection).

3. Support Vector Machines (SVMs)

SVM is a powerful classification algorithm that finds the optimal hyperplane to separate data points into different classes.

Working Principle:

1. **Hyperplane:** A decision boundary that maximizes the margin between two classes.
2. **Support Vectors:** Data points that lie closest to the hyperplane and influence its position.
3. **Kernel Trick:** SVM can handle non-linearly separable data using kernel functions to transform the input space.

Common Kernel Functions:

Linear Kernel: $K(x, y) = x^T y$

Polynomial Kernel: $K(x, y) = (x^T y + c)^d$

Radial Basis Function (RBF) Kernel: $K(x, y) = e^{-\gamma ||x-y||^2}$

Sigmoid Kernel: $K(x, y) = \tanh(\alpha x^T y + c)$

4. Decision Tree

A Decision Tree is a flowchart-like structure where internal nodes represent features, branches represent decisions, and leaves represent class labels.

Working Principle:

1. **Splitting Criteria:** Choose the best feature to split the data.
   - **Gini Index:** Measures impurity (Gini=1−∑pi^2).
   - **Entropy (Information Gain):** Measures information gained from a split.
2. **Recursive Splitting:** Continue splitting nodes until a stopping criterion is met.
3. **Pruning:** Reduces overfitting by trimming branches.

Types of Decision Trees:

- **ID3 (Iterative Dichotomiser 3)** – Uses entropy for splitting.
- **C4.5 & C5.0** – Improvement over ID3 (handles continuous data).
- **CART (Classification and Regression Trees)** – Uses Gini Index.

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

Steps:

Step 1: Load the Dataset

The dataset is loaded from a CSV file using pandas and First 5 entries in the Dataset is shown using df.head() and Total rows and columns are printed using df.shape[n].

```
[4] import pandas as pd
```

```
file_path = "/content/drive/MyDrive/Semester 6/AIDS/AIDS Lab/Clean_Dataset_Categorized.csv"
df = pd.read_csv(file_path)

# Display dataset overview
print(f"Total Entries: {df.shape[0]}")
print(f"Total Columns: {df.shape[1]}")
```

```
Total Entries: 300153
Total Columns: 13
```

```
[6] df.head()
```

	Unnamed: 0	airline	flight	source_city	departure_time	stops	arrival_time	destination_city	class	duration	days_left	price	price_category
0	0	SpiceJet	SG-8709	Delhi	Evening	zero	Night	Mumbai	Economy	2.17	1	5953	Cheap
1	1	SpiceJet	SG-8157	Delhi	Early_Morning	zero	Morning	Mumbai	Economy	2.33	1	5953	Cheap
2	2	AirAsia	I5-764	Delhi	Early_Morning	zero	Early_Morning	Mumbai	Economy	2.17	1	5956	Cheap
3	3	Vistara	UK-995	Delhi	Morning	zero	Afternoon	Mumbai	Economy	2.25	1	5955	Cheap
4	4	Vistara	UK-963	Delhi	Morning	zero	Morning	Mumbai	Economy	2.33	1	5955	Cheap

Step 2: Data Preprocessing

1) Drop Unnecessary Columns

For the following experiment, the columns such as Unnamed and price are not required for classification. So to start preprocessing, we drop such columns using **df.drop(columns=[])** command.

```
[7] # Drop 'Unnamed: 0' (index) and 'price' (to prevent data leakage)
df.drop(columns=['Unnamed: 0', 'price'], inplace=True, errors='ignore')

# Check updated dataset structure
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 300153 entries, 0 to 300152
Data columns (total 11 columns):
 #   Column            Non-Null Count   Dtype  
---  ------            --------------   -----  
 0   airline           300153 non-null  object 
 1   flight            300153 non-null  object 
 2   source_city       300153 non-null  object 
 3   departure_time    300153 non-null  object 
 4   stops             300153 non-null  object 
 5   arrival_time      300153 non-null  object 
 6   destination_city  300153 non-null  object 
 7   class             300153 non-null  object 
 8   duration          300153 non-null  float64
 9   days_left         300153 non-null  int64  
 10  price_category    300153 non-null  object 
dtypes: float64(1), int64(1), object(9)
memory usage: 25.2+ MB
```

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

2) Handling Missing Values

We fill up the missing values such that the numeric columns are filled with the median values, and the categorical columns are filled with mode of that column.

```
[8] # Check for missing values
    missing_values = df.isnull().sum()
    print(missing_values[missing_values > 0])  # Show only columns with missing values

    # Fill missing values
    df.fillna(df.median(numeric_only=True), inplace=True)  # Fill numeric columns with median
    df.fillna(df.mode().iloc[0], inplace=True)  # Fill categorical columns with mode
```

```
Series([], dtype: int64)
```

3) Encode Categorical Variables

The categorical columns are encoded so that it becomes suitable for the algorithm to make it easier for the algorithm to make the classification.

```
from sklearn.preprocessing import LabelEncoder

# Identify categorical columns
categorical_columns = df.select_dtypes(include=['object']).columns

# Apply Label Encoding
le = LabelEncoder()
for col in categorical_columns:
    df[col] = le.fit_transform(df[col])

# Check transformed data
df.head()
```

	airline	flight	source_city	departure_time	stops	arrival_time	destination_city	class	duration	days_left	price_category
0	4	1408	2	2	2	5	5	1	2.17	1	0
1	4	1387	2	1	2	4	5	1	2.33	1	0
2	0	1213	2	1	2	1	5	1	2.17	1	0
3	5	1559	2	4	2	0	5	1	2.25	1	0
4	5	1549	2	4	2	4	5	1	2.33	1	0

4) Save Data In New File

```
# Define the file path to save the processed dataset
processed_file_path = "/content/drive/MyDrive/Semester 6/AIDS/AIDS Lab/Converted_Clean_Dataset_Categorized.csv"

# Save the DataFrame to a new CSV file
df.to_csv(processed_file_path, index=False)

print(f"Preprocessed dataset saved as: {processed_file_path}")
```

```
Preprocessed dataset saved as: /content/drive/MyDrive/Semester 6/AIDS/AIDS Lab/Converted_Clean_Dataset_Categorized.csv
```

Step 3: Split Into Train and Test

The dataset it then split into training and testing such that the models are trained on 80% of the dataset and 20% is used to test the models for their accuracy.

```python
from sklearn.model_selection import train_test_split

# Define Features (X) and Target (y)
X = df.drop(columns=['price_category'])  # Target column
y = df['price_category']

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

# Display the split
print(f"Training Samples: {X_train.shape[0]}")
print(f"Testing Samples: {X_test.shape[0]}")
```

```
Training Samples: 240122
Testing Samples: 60031
```

Step 4: Train & Evaluate Classifiers

1)K-Nearest Neighbors (KNN)

From sklearn.neighbors library, we import the KNeighboursClassifier. We call this function and pass a parameter for the number of neighbours to be used. Here, we are passing 5 neighbours. After that, we fit the model with our training datasets (X and y) and create a variable to store the predicted values.

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

# Train KNN Model
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)

# Predictions
y_pred_knn = knn.predict(X_test)
```

I) Classification Report

Using the classification_report function, we generate the classification report which would give the performance metrics such as accuracy, precision, recall, f1-score, support, etc.

```python
# Classification Report
classification_report_knn = classification_report(y_test, y_pred_knn, output_dict=True)
pd.DataFrame(classification_report_knn).transpose()
```

	precision	recall	f1-score	support
0	0.863818	0.854738	0.859254	33202.00000
1	0.822540	0.833240	0.827856	26829.00000
accuracy	0.845130	0.845130	0.845130	0.84513
macro avg	0.843179	0.843989	0.843555	60031.00000
weighted avg	0.845370	0.845130	0.845221	60031.00000

II) Accuracy

II) Accuracy

```
[14] from sklearn.metrics import accuracy_score

     # Accuracy
     accuracy_knn = accuracy_score(y_test, y_pred_knn)
     accuracy_knn
```

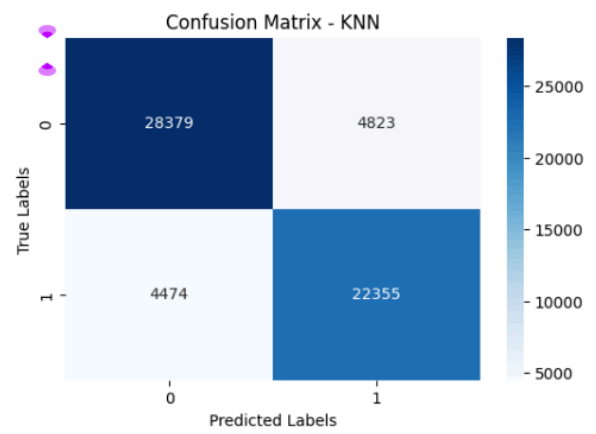
```
0.8451300161583182
```

III) Confusion Matrix

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Confusion Matrix
conf_matrix_knn = confusion_matrix(y_test, y_pred_knn)

# Plot Confusion Matrix
plt.figure(figsize=(6,4))
sns.heatmap(conf_matrix_knn, annot=True, cmap="Blues", fmt='g')
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Confusion Matrix - KNN")
plt.show()
```



The KNN model achieves 84.51% accuracy, performing well across both classes. Class 0 has slightly higher precision (0.8633) and recall (0.8547) than Class 1 (0.8225, 0.8332), indicating a minor bias. The confusion matrix shows 28,379 True Negatives (TN) and 22,355 True Positives (TP), with 4,823 False Positives (FP) and 4,474 False Negatives (FN). Misclassification is fairly balanced, though Class 1 has slightly higher errors. Tuning K-values, distance metrics, and handling class imbalance can improve performance.

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

2) Naive Bayes

```
from sklearn.naive_bayes import GaussianNB

# Train Naive Bayes Model
nb = GaussianNB()
nb.fit(X_train, y_train)

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

I) Classification Report

```
# Classification Report
classification_report_nb = classification_report(y_test, y_pred_nb, output_dict=True)
pd.DataFrame(classification_report_nb).transpose()
```

	precision	recall	f1-score	support
0	0.820803	0.722065	0.768274	33202.000000
1	0.700613	0.804913	0.749150	26829.000000
accuracy	0.759091	0.759091	0.759091	0.759091
macro avg	0.760708	0.763489	0.758712	60031.000000
weighted avg	0.767088	0.759091	0.759727	60031.000000

II) Accuracy

```
[18] # Accuracy
accuracy_nb = accuracy_score(y_test, y_pred_nb)
accuracy_nb
```

```
0.7590911362462728
```

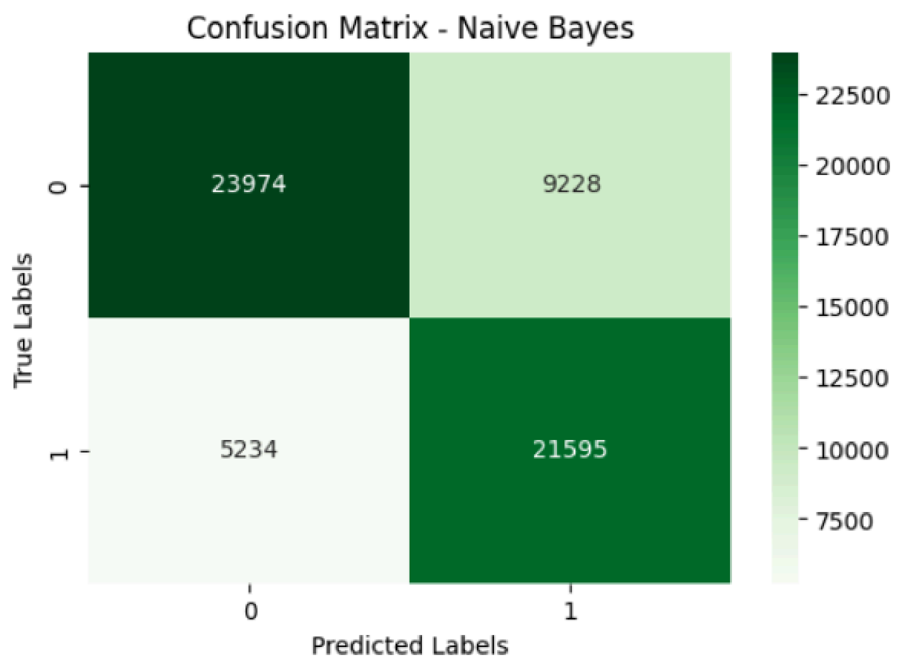
Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

III) Confusion Matrix

```
# Confusion Matrix
conf_matrix_nb = confusion_matrix(y_test, y_pred_nb)

# Plot Confusion Matrix
plt.figure(figsize=(6,4))
sns.heatmap(conf_matrix_nb, annot=True, cmap="Greens", fmt='g')
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Confusion Matrix - Naive Bayes")
plt.show()
```



The Naive Bayes model (GaussianNB) achieved an accuracy of 75.91%. It performed slightly better for class 1 in terms of recall (80.49%), but precision was higher for class 0 (82.08%), showing a trade-off between the two. The F1-scores were moderately balanced, around 0.75–0.76 for both classes. The confusion matrix revealed more false positives (9,228) and false negatives (5,234) than KNN, indicating it was more prone to misclassification. Overall, Naive Bayes performed decently but not as accurately as KNN.

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

3)Decision Tree (With Visualization)

```
[20] from sklearn.tree import DecisionTreeClassifier

     # Train Decision Tree Model
     dt = DecisionTreeClassifier(max_depth=3, random_state=42)
     dt.fit(X_train, y_train)

     # Predictions
     y_pred_dt = dt.predict(X_test)
```

I) Classification Report

```
# Classification Report
classification_report_dt = classification_report(y_test, y_pred_dt, output_dict=True)
pd.DataFrame(classification_report_dt).transpose()
```

	precision	recall	f1-score	support
0	0.869279	0.871243	0.870260	33202.000000
1	0.840211	0.837862	0.839035	26829.000000
accuracy	0.856324	0.856324	0.856324	0.856324
macro avg	0.854745	0.854552	0.854647	60031.000000
weighted avg	0.856288	0.856324	0.856305	60031.000000

II) Accuracy

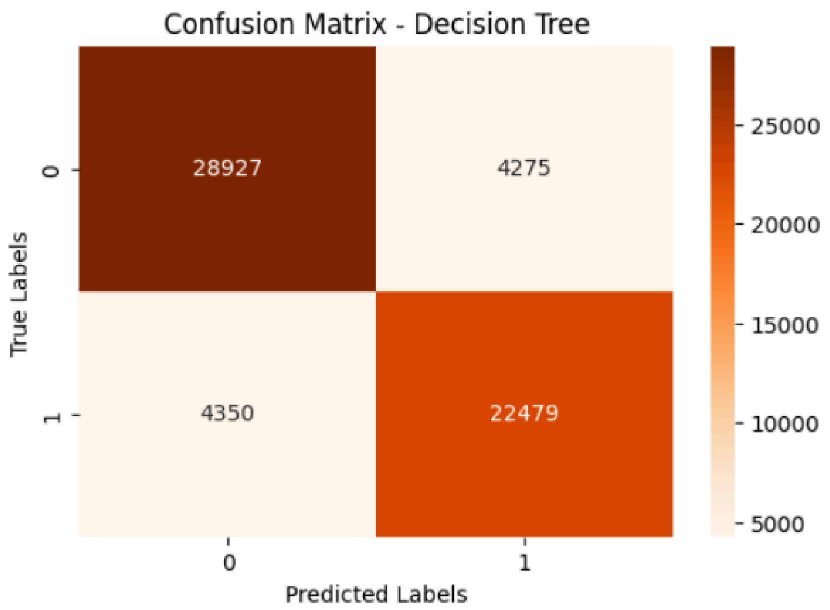
```
[22] # Accuracy
     accuracy_dt = accuracy_score(y_test, y_pred_dt)
     accuracy_dt
```

```
0.8563242324798854
```

III) Confusion Matrix

```
# Confusion Matrix
conf_matrix_dt = confusion_matrix(y_test, y_pred_dt)

# Plot Confusion Matrix
plt.figure(figsize=(6,4))
sns.heatmap(conf_matrix_dt, annot=True, cmap="Oranges", fmt='g')
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Confusion Matrix - Decision Tree")
plt.show()
```



The Decision Tree classifier achieved an overall accuracy of approximately 85.63%, demonstrating a balanced performance across both classes. From the classification report, we observe that class 0 had slightly higher precision and recall compared to class 1, indicating the model is slightly better at predicting class 0 instances. The F1-scores for both classes are close—0.870 for class 0 and 0.839 for class 1—showing a good trade-off between precision and recall. The confusion matrix further confirms this, with a high number of correctly predicted instances for both classes and a relatively low number of misclassifications.

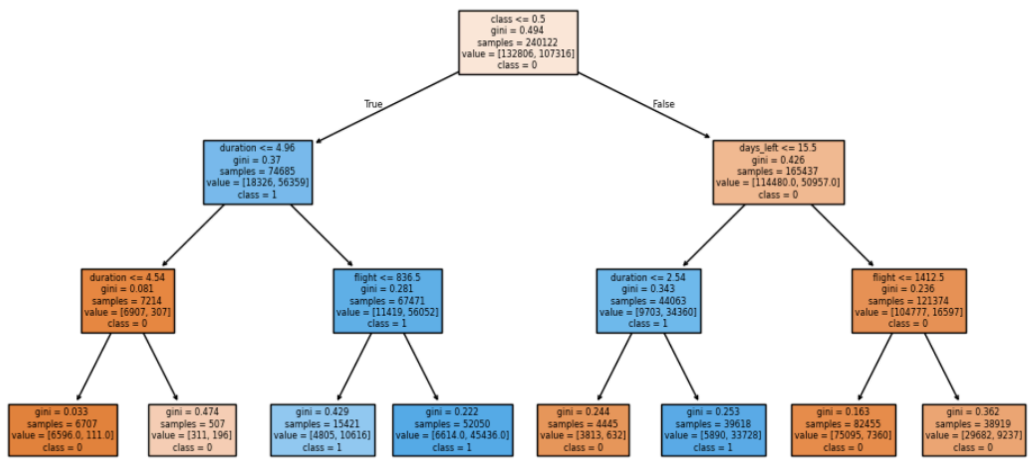
IV) Visualization

```
from sklearn import tree

# Visualizing Decision Tree
plt.figure(figsize=(12,6))
tree.plot_tree(dt, filled=True, feature_names=X.columns, class_names=[str(c) for c in dt.classes_])
plt.title("Decision Tree Visualization")
plt.show()
```

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**



The Decision Tree visualization, limited to a maximum depth of 3, reveals that features such as duration, flight, and days_left played a key role in the classification. This controlled depth helps maintain interpretability while preventing overfitting. Overall, the model shows strong predictive capability and clear decision logic, making it a solid baseline for classification tasks.

4)Support Vector Machines (SVM)

1)Performed On Small Set

```
from sklearn.model_selection import train_test_split

# Reduce dataset size for SVM (e.g., use 10% of the original data)
small_df = df.sample(frac=0.1, random_state=42)  # Takes 10% of the data

# Define features (X) and target (y)
X_small = small_df.drop(columns=['price_category'])
y_small = small_df['price_category']

# Split into training (80%) and testing (20%)
X_train_small, X_test_small, y_train_small, y_test_small = train_test_split(
    X_small, y_small, test_size=0.2, random_state=42, stratify=y_small
)

# Print dataset sizes
print(f"Reduced Training samples: {X_train_small.shape[0]}, Testing samples: {X_test_small.shape[0]}")
```

```
Reduced Training samples: 24012, Testing samples: 6003
```

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

2) Model Train

```python
from sklearn.svm import SVC

# Train SVM on reduced dataset
svm_model = SVC(kernel='rbf', random_state=42)
svm_model.fit(X_train_small, y_train_small)

# Predictions
y_pred_svm = svm_model.predict(X_test_small)
```

I) Classification Report

```python
# Classification Report for SVM (Updated for small dataset)
classification_report_svm = classification_report(y_test_small, y_pred_svm, output_dict=True)

# Convert to DataFrame for better visualization
pd.DataFrame(classification_report_svm).transpose()
```

	precision	recall	f1-score	support
0	0.710333	0.754197	0.731608	3336.000000
1	0.666802	0.615298	0.640016	2667.000000
accuracy	0.692487	0.692487	0.692487	0.692487
macro avg	0.688568	0.684747	0.685812	6003.000000
weighted avg	0.690993	0.692487	0.690916	6003.000000

II) Accuracy

```python
# Accuracy
accuracy_svm = accuracy_score(y_test_small, y_pred_svm)
accuracy_svm
```

```
0.6924870897884391
```

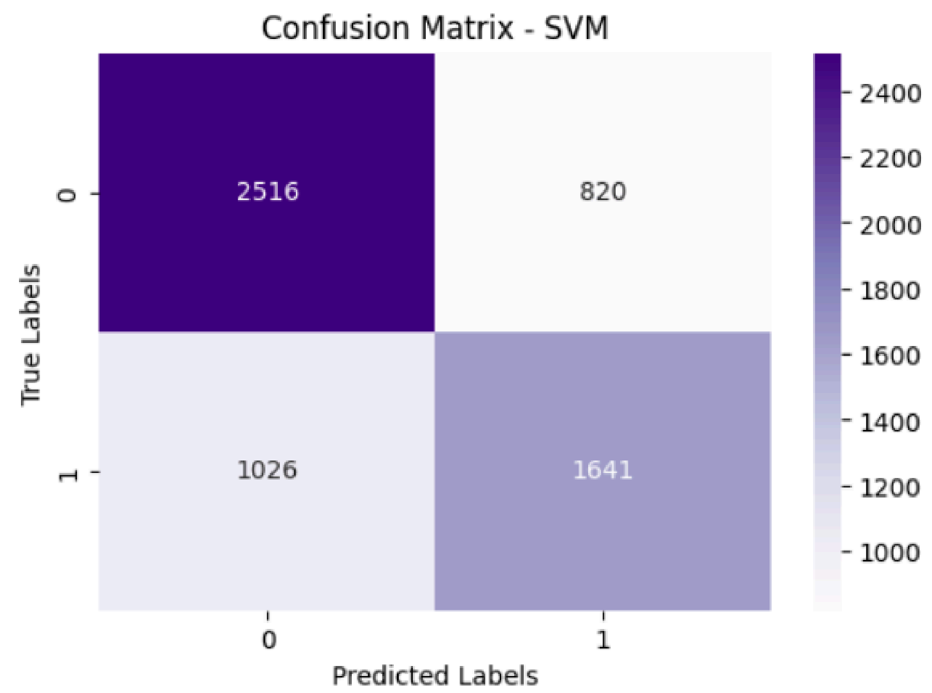
Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

III) Confusion Matrix

```
# Confusion Matrix
conf_matrix_svm = confusion_matrix(y_test_small, y_pred_svm)

# Plot Confusion Matrix
plt.figure(figsize=(6,4))
sns.heatmap(conf_matrix_svm, annot=True, cmap="Purples", fmt='g')
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Confusion Matrix - SVM")
plt.show()
```



To evaluate the Support Vector Machine (SVM) model, a reduced dataset consisting of 10% of the original data was used to optimize computational efficiency. The SVM, trained with an RBF kernel, achieved an accuracy of approximately 69.25%. From the classification report, class 0 had a higher precision (0.71) and recall (0.75), while class 1 showed slightly lower performance with a precision of 0.67 and recall of 0.62. This indicates that the model performs better in identifying class 0 instances. The confusion matrix shows 2,516 correct predictions for class 0 and 1,641 for class 1, while misclassifications include 820 and 1,026 instances respectively. Although the overall performance is lower compared to the Decision Tree model, the SVM still demonstrates decent generalization on a smaller dataset and could benefit from further hyperparameter tuning or feature scaling to improve classification of minority cases.

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

Step 5: Compare Classifiers

```
print("Classifier Performance:")
print(f"KNN Accuracy: {accuracy_score(y_test, y_pred_knn)}")
print(f"Naive Bayes Accuracy: {accuracy_score(y_test, y_pred_nb)}")
print(f"Decision Tree Accuracy: {accuracy_score(y_test, y_pred_dt)}")
print(f"SVM Accuracy: {accuracy_score(y_test_small, y_pred_svm)}")
```

```
Classifier Performance:
KNN Accuracy: 0.8451300161583182
Naive Bayes Accuracy: 0.7590911362462728
Decision Tree Accuracy: 0.8563242324798854
SVM Accuracy: 0.6924870897884391
```

Conclusion:

In the comparison of classifiers, the **Decision Tree** model achieved the **highest accuracy at 85.63%**, making it the best-performing model in this evaluation. The **K-Nearest Neighbors (KNN)** model followed closely with an accuracy of **84.51%**, demonstrating strong predictive performance as well. The **Naive Bayes** classifier achieved a moderate accuracy of **75.91%**, suggesting it may not capture the complexity of the data as effectively. The **Support Vector Machine (SVM)**, trained on a reduced dataset due to computational limitations, yielded the lowest accuracy at **69.25%**. Although SVM generally performs well with well-separated data, its performance here may be hindered by dataset reduction or lack of feature scaling. Overall, the Decision Tree appears to be the most suitable model for this task, balancing interpretability and performance effectively.