**Machine Learning Lab (PMCA507P)**

> **Guided by :** Dr.B.Saleena Ma'am
>
> **Reg No :** 23MCA1030
>
> **Name :** Vinayak Kumar Singh

Collab Url: https://colab.research.google.com/drive/1x3icdCrYa1sty92Ju0IW7kqX4fpHEx50?usp=sharing

## Import necessary libraries

```
import pandas as pd
import numpy as np
from sklearn.metrics import precision_score, recall_score, f1_score
```

## Load the dataset

```
data = pd.read_csv('/content/heart_Dataset_2.csv')
data.info();
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       303 non-null    int64
 1   sex       303 non-null    int64
 2   cp        303 non-null    int64
 3   trestbps  303 non-null    int64
 4   chol      303 non-null    int64
 5   fbs       303 non-null    int64
 6   restecg   303 non-null    int64
 7   thalach   303 non-null    int64
 8   exang     303 non-null    int64
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64
 11  ca        303 non-null    int64
 12  thal      303 non-null    int64
 13  target    303 non-null    int64
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

```
data
```

| | age | sex | cp | trestbps | chol | fbs | restecg | thalach | exang | oldpeak | slope | ca | thal | target |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 63 | 1 | 3 | 145 | 233 | 1 | 0 | 150 | 0 | 2.3 | 0 | 0 | 1 | 1 |
| **1** | 37 | 1 | 2 | 130 | 250 | 0 | 1 | 187 | 0 | 3.5 | 0 | 0 | 2 | 1 |
| **2** | 41 | 0 | 1 | 130 | 204 | 0 | 0 | 172 | 0 | 1.4 | 2 | 0 | 2 | 1 |
| **3** | 56 | 1 | 1 | 120 | 236 | 0 | 1 | 178 | 0 | 0.8 | 2 | 0 | 2 | 1 |
| **4** | 57 | 0 | 0 | 120 | 354 | 0 | 1 | 163 | 1 | 0.6 | 2 | 0 | 2 | 1 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **298** | 57 | 0 | 0 | 140 | 241 | 0 | 1 | 123 | 1 | 0.2 | 1 | 0 | 3 | 0 |
| **299** | 45 | 1 | 3 | 110 | 264 | 0 | 1 | 132 | 0 | 1.2 | 1 | 0 | 3 | 0 |
| **300** | 68 | 1 | 0 | 144 | 193 | 1 | 1 | 141 | 0 | 3.4 | 1 | 2 | 3 | 0 |
| **301** | 57 | 1 | 0 | 130 | 131 | 0 | 1 | 115 | 1 | 1.2 | 1 | 1 | 3 | 0 |
| **302** | 57 | 0 | 1 | 130 | 236 | 0 | 0 | 174 | 0 | 0.0 | 1 | 1 | 2 | 0 |

303 rows × 14 columns

Next steps: Generate code with data | View recommended plots

## Data preprocessing

## Check for missing values in the dataset

```
missing_values = data.isnull().sum()
print("Missing Values:")
print(missing_values)
```

```
Missing Values:
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

## check unique value in dataset

```
data.nunique()
```

```
age          41
sex           2
cp            4
trestbps     49
chol        152
fbs           2
restecg       3
thalach      91
exang         2
oldpeak      40
slope         3
ca            5
thal          4
target        2
dtype: int64
```

## Mapping numeric values to categories

```
data['sex'] = data['sex'].map({1: 'male  ', 0: 'female'})
```

# Implement the Model

## A. Logistic Regression

```
import pandas as pd

data = pd.read_csv("/content/heart_Dataset_2.csv")

data.dropna(inplace=True)

X = data.drop('target', axis=1)
y = data['target']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = LogisticRegression()

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```

```
Accuracy: 0.8852459016393442
Precision: 0.8787878787878788
Recall: 0.90625
F1 Score: 0.8923076923076922
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

## B. Naive Bayes Classifier

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

data = pd.read_csv("/content/heart_Dataset_2.csv")

data.dropna(inplace=True)

X = data.drop('target', axis=1)
y = data['target']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = GaussianNB()

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```

```
Accuracy: 0.8688524590163934
Precision: 0.9
Recall: 0.84375
F1 Score: 0.870967741935484
```

## C. SVM (Linear SVM)

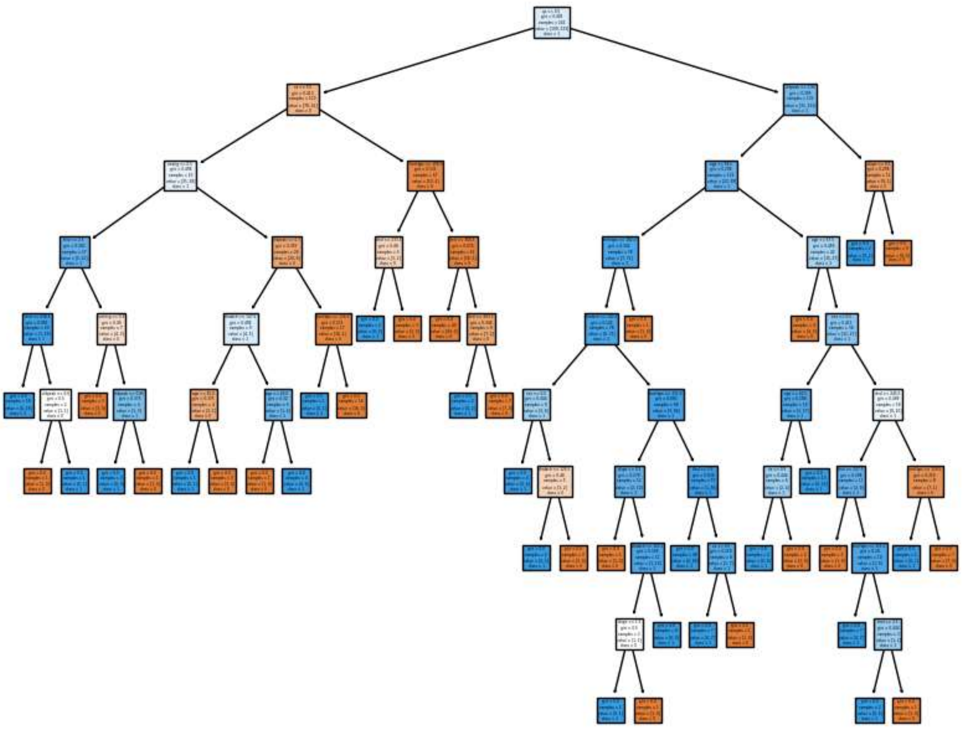
```
from sklearn.svm import SVC

# Load the dataset
data = pd.read_csv("/content/heart_Dataset_2.csv")

# Drop rows with missing values
data.dropna(inplace=True)

# Separate features and target variable
X = data.drop('target', axis=1)
y = data['target']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a Linear SVM model
model = SVC(kernel='linear')

# Train the model on the training data
model.fit(X_train, y_train)

# Make predictions on the test data
y_pred = model.predict(X_test)

# Evaluate the model performance
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```

```
Accuracy: 0.8688524590163934
Precision: 0.875
Recall: 0.875
F1 Score: 0.875
```

## D. Decision Tree

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
# Load the dataset
data = pd.read_csv("/content/heart_Dataset_2.csv")

# Drop rows with missing values
data.dropna(inplace=True)

# Separate features and target variable
X = data.drop('target', axis=1)
y = data['target']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a Decision Tree model
model = DecisionTreeClassifier()

# Train the model on the training data
model.fit(X_train, y_train)

# Make predictions on the test data
y_pred = model.predict(X_test)

# Evaluate the model performance
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)

# Visualize the decision tree
plt.figure(figsize=(10, 8))
plot_tree(model, feature_names=X.columns, class_names=['0', '1'], filled=True)
plt.show()
```

```
Accuracy: 0.819672131147541
Precision: 0.8888888888888888
Recall: 0.75
F1 Score: 0.8135593220338982
```



## Q2

```
# Logistic Regression
accuracy_lr = accuracy_score(y_test, y_pred_lr)
precision_lr = precision_score(y_test, y_pred_lr)
recall_lr = recall_score(y_test, y_pred_lr)
f1_lr = f1_score(y_test, y_pred_lr)
error_rate_lr = 1 - accuracy_lr

# Naive Bayes
accuracy_nb = accuracy_score(y_test, y_pred_nb)
precision_nb = precision_score(y_test, y_pred_nb)
recall_nb = recall_score(y_test, y_pred_nb)
f1_nb = f1_score(y_test, y_pred_nb)
error_rate_nb = 1 - accuracy_nb

# SVM
accuracy_svm = accuracy_score(y_test, y_pred_svm)
precision_svm = precision_score(y_test, y_pred_svm)
recall_svm = recall_score(y_test, y_pred_svm)
f1_svm = f1_score(y_test, y_pred_svm)
error_rate_svm = 1 - accuracy_svm

# Decision Tree
accuracy_dt = accuracy_score(y_test, y_pred_dt)
precision_dt = precision_score(y_test, y_pred_dt)
recall_dt = recall_score(y_test, y_pred_dt)
f1_dt = f1_score(y_test, y_pred_dt)
error_rate_dt = 1 - accuracy_dt

print("Logistic Regression:")
print("- Accuracy:", accuracy_lr)
print("- Precision:", precision_lr)
print("- Recall:", recall_lr)
print("- F1 Score:", f1_lr)
print("- Error rate:", error_rate_lr)

print("\nNaive Bayes:")
print("- Accuracy:", accuracy_nb)
print("- Precision:", precision_nb)
print("- Recall:", recall_nb)
print("- F1 Score:", f1_nb)
print("- Error rate:", error_rate_nb)

print("\nSVM:")
print("- Accuracy:", accuracy_svm)
print("- Precision:", precision_svm)
print("- Recall:", recall_svm)
print("- F1 Score:", f1_svm)
print("- Error rate:", error_rate_svm)

print("\nDecision Tree:")
print("- Accuracy:", accuracy_dt)
print("- Precision:", precision_dt)
print("- Recall:", recall_dt)
print("- F1 Score:", f1_dt)
print("- Error rate:", error_rate_dt)
```

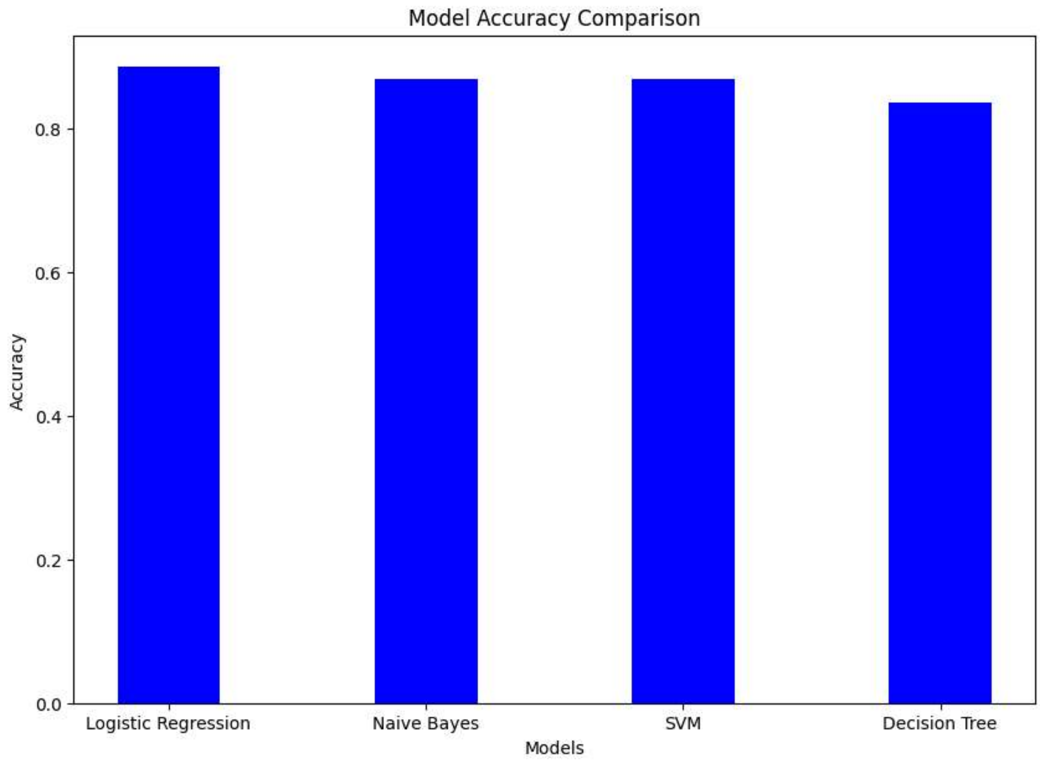
```
Logistic Regression:
- Accuracy: 0.8852459016393442
- Precision: 0.8787878787878788
- Recall: 0.90625
- F1 Score: 0.8923076923076922
- Error rate: 0.11475409836065575

Naive Bayes:
- Accuracy: 0.8688524590163934
- Precision: 0.9
- Recall: 0.84375
- F1 Score: 0.870967741935484
- Error rate: 0.1311475409836066

SVM:
- Accuracy: 0.8688524590163934
- Precision: 0.875
- Recall: 0.875
- F1 Score: 0.875
- Error rate: 0.1311475409836066

Decision Tree:
- Accuracy: 0.8360655737704918
- Precision: 0.8928571428571429
- Recall: 0.78125
- F1 Score: 0.8333333333333334
- Error rate: 0.16393442622950816
```

## Q3

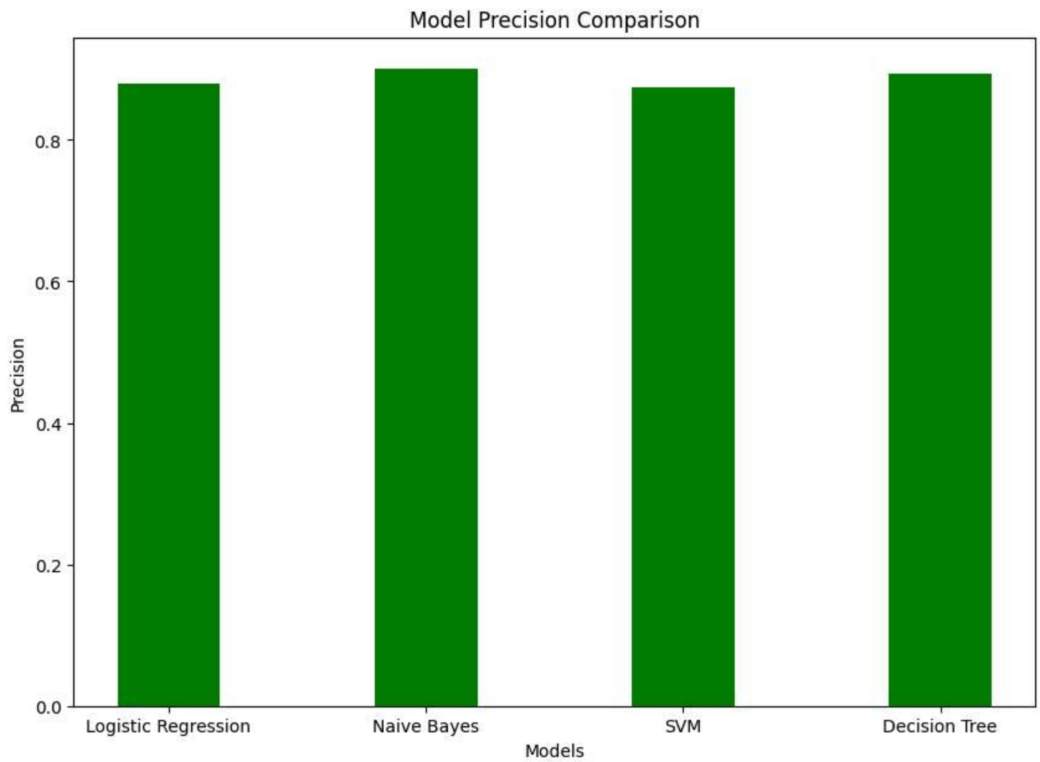
> Visualize the results of all the models and derive the inferences individually and also perform a comparative evaluation

```
models = ['Logistic Regression', 'Naive Bayes', 'SVM', 'Decision Tree']
accuracy_scores = [accuracy_lr, accuracy_nb, accuracy_svm, accuracy_dt]
```

```
plt.figure(figsize=(10, 7))
plt.bar(models, accuracy_scores, color='blue', width=0.4)
plt.xlabel('Models')
plt.ylabel('Accuracy')
plt.title('Model Accuracy Comparison')
plt.show()
```

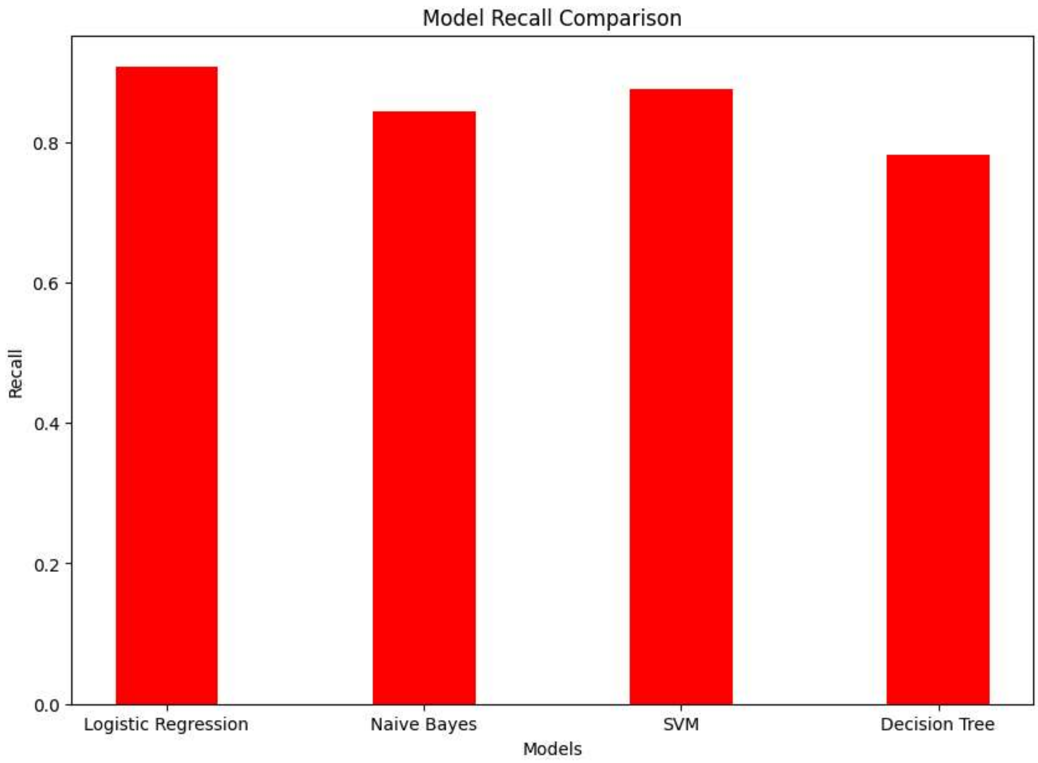



```
precision_scores = [precision_lr, precision_nb, precision_svm, precision_dt]

plt.figure(figsize=(10, 7))
plt.bar(models, precision_scores, color='green', width=0.4)
plt.xlabel('Models')
plt.ylabel('Precision')
plt.title('Model Precision Comparison')
plt.show()
```

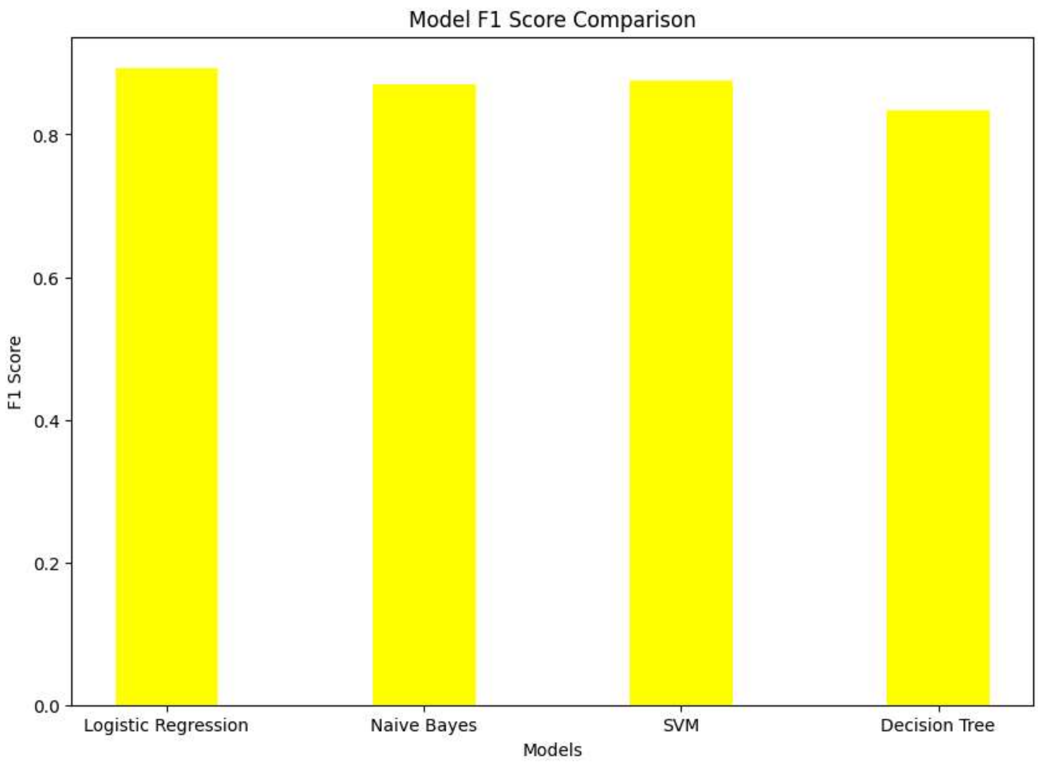



```
recall_scores = [recall_lr, recall_nb, recall_svm, recall_dt]

plt.figure(figsize=(10, 7))
plt.bar(models, recall_scores, color='red', width=0.4)
```

```
plt.xlabel('Models')
plt.ylabel('Recall')
plt.title('Model Recall Comparison')
plt.show()
```

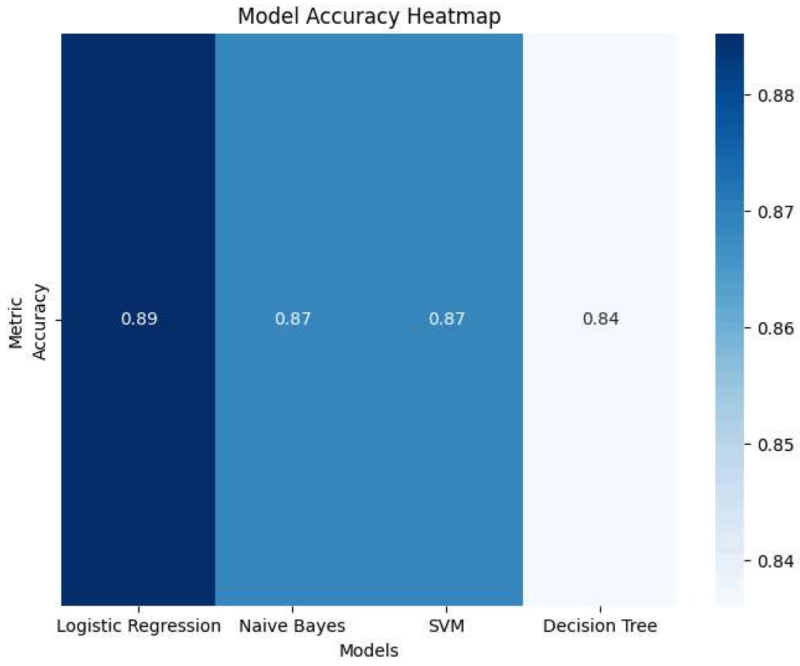



```
f1_scores = [f1_lr, f1_nb, f1_svm, f1_dt]

plt.figure(figsize=(10, 7))
plt.bar(models, f1_scores, color='yellow', width=0.4)
plt.xlabel('Models')
plt.ylabel('F1 Score')
plt.title('Model F1 Score Comparison')
plt.show()
```




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

# Create a list of models and their corresponding accuracy scores
models = ['Logistic Regression', 'Naive Bayes', 'SVM', 'Decision Tree']
accuracy_scores = [accuracy_lr, accuracy_nb, accuracy_svm, accuracy_dt]

# Create a heatmap
plt.figure(figsize=(8, 6))
```

```
plt.figure(figsize=(8, 6))
sns.heatmap(data=[accuracy_scores], xticklabels=models, yticklabels=['Accuracy'], annot=True, fmt=".2f", cmap="Blues")
plt.xlabel('Models')
plt.ylabel('Metric')
plt.title('Model Accuracy Heatmap')
plt.show()
```




## Q4

> Find the highly correlated parameters in the dataset

```
import pandas as pd

# Load the dataset
data = pd.read_csv("/content/heart_Dataset_2.csv")

# Calculate the correlation matrix
correlation_matrix = data.corr()

# Find the highly correlated parameters
highly_correlated_parameters = []
for i in range(len(correlation_matrix.columns)):
    for j in range(i + 1, len(correlation_matrix.columns)):
        if abs(correlation_matrix.iloc[i, j]) > 0.5:
            highly_correlated_parameters.append((correlation_matrix.columns[i], correlation_matrix.columns[j]))

# Print the highly correlated parameters
print("Highly Correlated Parameters:")
for pair in highly_correlated_parameters:
    print(f"- {pair[0]} and {pair[1]}")
```

```
Highly Correlated Parameters:
- oldpeak and slope
```