Car price prediction Model

Importing Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import (
    accuracy_score, classification_report, confusion_matrix, roc_auc_score,
    roc_curve, matthews_corrcoef
)

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
```

Fetching Data

```
from google.colab import files
uploaded = files.upload()
```

Choose Files CarPricesPrediction.csv

- **CarPricesPrediction.csv**(text/csv) - 48598 bytes, last modified: 2/10/2024 - 100% done

```
Saving CarPricesPrediction.csv to CarPricesPrediction (1).csv
```

```
import os
os.listdir()
```

```
['.config',
 'CarPricesPrediction (1).csv',
 'CarPricesPrediction.csv',
 'sample_data']
```

```
import pandas as pd

# Replace filename if it's different (check uploaded name in left panel or use `os.listdir()`)
df = pd.read_csv("CarPricesPrediction.csv")
```

Performing EDA

```
# 1. Display first 5 rows
print("📌 First 5 Rows:")
display(df.head())

# 2. Summary statistics
print("\n📊 Summary Statistics:")
display(df.describe(include='all'))

# 3. Dataset info
print("\n📦 Dataset Info:")
df.info()

# 4. Missing values check
print("\n❌ Missing Values:")
print(df.isnull().sum())
```

📌 First 5 Rows:

	Unnamed: 0	Make	Model	Year	Mileage	Condition	Price
0	0	Ford	Silverado	2022	18107	Excellent	19094.75
1	1	Toyota	Silverado	2014	13578	Excellent	27321.10
2	2	Chevrolet	Civic	2016	46054	Good	23697.30
3	3	Ford	Civic	2022	34981	Excellent	18251.05
4	4	Chevrolet	Civic	2019	63565	Excellent	19821.85

📊 Summary Statistics:

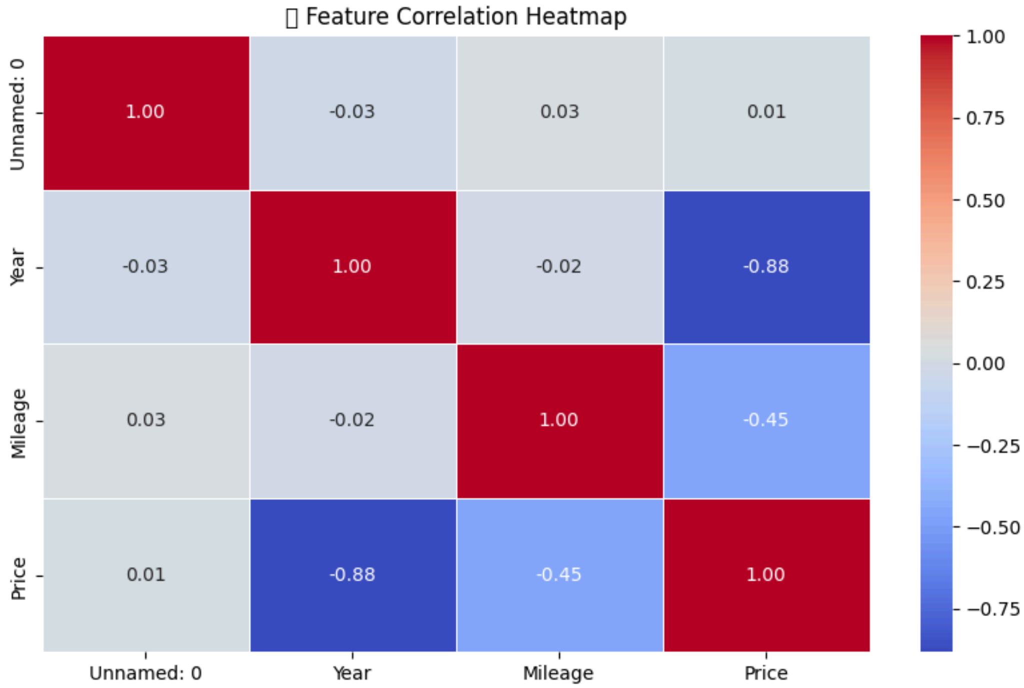
	Unnamed: 0	Make	Model	Year	Mileage	Condition	Price
count	1000.000000	1000	1000	1000.00000	1000.000000	1000	1000.000000
unique	NaN	5	5	NaN	NaN	3	NaN
top	NaN	Chevrolet	Altima	NaN	NaN	Excellent	NaN
freq	NaN	209	226	NaN	NaN	595	NaN
mean	499.500000	NaN	NaN	2015.86500	78796.927000	NaN	22195.205650
std	288.819436	NaN	NaN	3.78247	39842.259941	NaN	4245.191585
min	0.000000	NaN	NaN	2010.00000	10079.000000	NaN	12613.000000
25%	249.750000	NaN	NaN	2013.00000	44942.750000	NaN	18961.862500
50%	499.500000	NaN	NaN	2016.00000	78056.500000	NaN	22247.875000
75%	749.250000	NaN	NaN	2019.00000	112366.250000	NaN	25510.275000
max	999.000000	NaN	NaN	2022.00000	149794.000000	NaN	31414.900000

```
🧾 Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   Unnamed: 0  1000 non-null   int64  
 1   Make        1000 non-null   object 
 2   Model       1000 non-null   object 
 3   Year        1000 non-null   int64  
 4   Mileage     1000 non-null   int64  
 5   Condition   1000 non-null   object 
 6   Price       1000 non-null   float64
dtypes: float64(1), int64(3), object(3)
memory usage: 54.8+ KB

❌ Missing Values:
Unnamed: 0    0
Make          0
Model         0
Year          0
Mileage       0
Condition     0
Price         0
dtype: int64
```

Correlation Heatmap

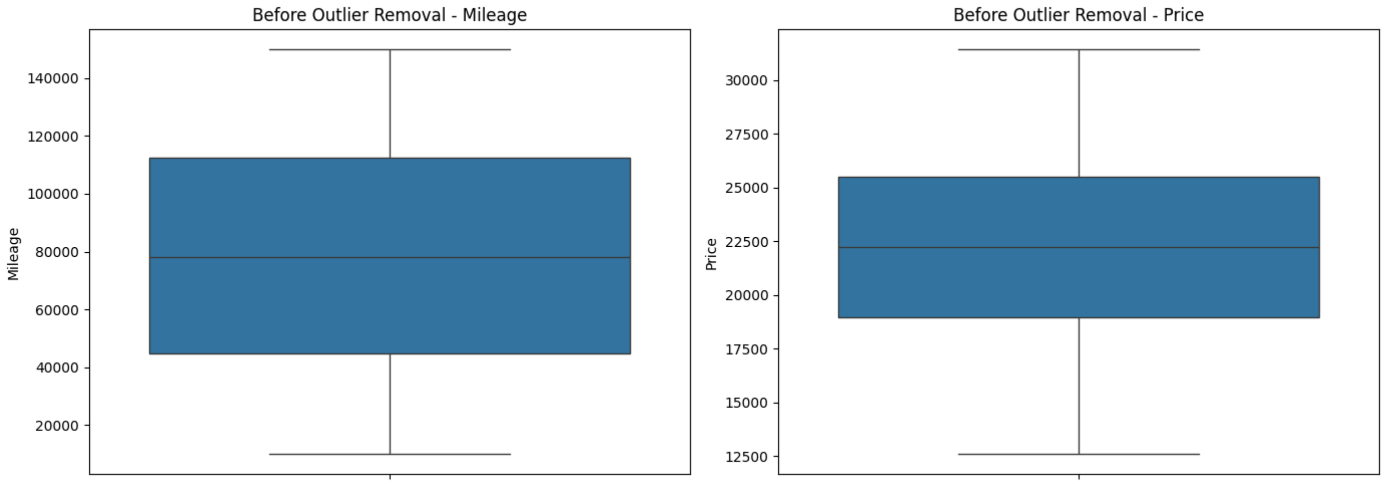
```
import seaborn as sns
import matplotlib.pyplot as plt

# Calculate correlation matrix
correlation_matrix = df.corr(numeric_only=True)

# Plot heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5, fmt=".2f")
plt.title("📊 Feature Correlation Heatmap")
plt.show()
```

```
/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 128202 (\N{BAR CHART}) missing from font(s) D
  fig.canvas.print_figure(bytes_io, **kw)
```



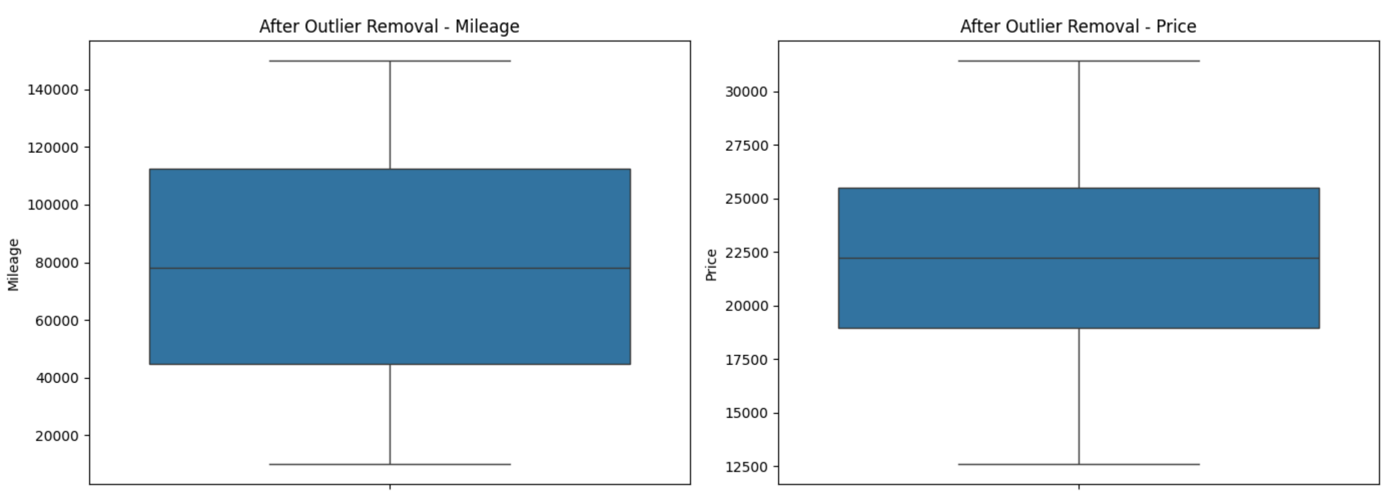
Outlier Handling

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

# OPTIONAL: Reload dataset if 'Price' was dropped earlier
df = pd.read_csv("CarPricesPrediction.csv")

# Drop Unnamed column if exists
if "Unnamed: 0" in df.columns:
    df.drop(columns=["Unnamed: 0"], inplace=True)

# Step 1: Boxplots BEFORE Outlier Handling
plt.figure(figsize=(14, 5))

plt.subplot(1, 2, 1)
sns.boxplot(y=df["Mileage"])
plt.title("Before Outlier Removal - Mileage")

plt.subplot(1, 2, 2)
sns.boxplot(y=df["Price"])
plt.title("Before Outlier Removal - Price")

plt.tight_layout()
plt.show()
```




```
def remove_outliers_iqr(df, column):
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower = Q1 - 1.5 * IQR
    upper = Q3 + 1.5 * IQR
    return df[(df[column] >= lower) & (df[column] <= upper)]

# Remove outliers from Mileage and Price
df_clean = remove_outliers_iqr(df, "Mileage")
df_clean = remove_outliers_iqr(df_clean, "Price")
```

```
plt.figure(figsize=(14, 5))

plt.subplot(1, 2, 1)
sns.boxplot(y=df_clean["Mileage"])
plt.title("After Outlier Removal - Mileage")

plt.subplot(1, 2, 2)
sns.boxplot(y=df_clean["Price"])
plt.title("After Outlier Removal - Price")

plt.tight_layout()
plt.show()
```



Normalization

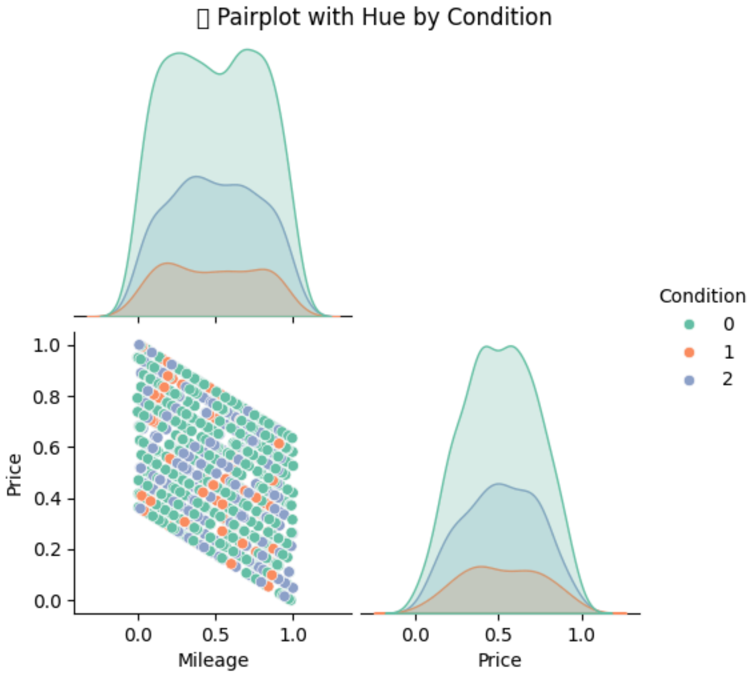
```
from sklearn.preprocessing import MinMaxScaler

# Step 1: Select numerical features to normalize
numeric_cols = ["Mileage", "Price"]

# Step 2: Initialize scaler
scaler = MinMaxScaler()

# Step 3: Fit and transform the data
df_clean[numeric_cols] = scaler.fit_transform(df_clean[numeric_cols])

# Step 4: Display normalized values
print("✅ Normalized Data (first 5 rows):")
print(df_clean[numeric_cols].head())
```

```
✅ Normalized Data (first 5 rows):
    Mileage     Price
0  0.057460  0.344739
1  0.025044  0.782267
2  0.257488  0.589531
3  0.178234  0.299866
4  0.382822  0.383411
```

PairPlot

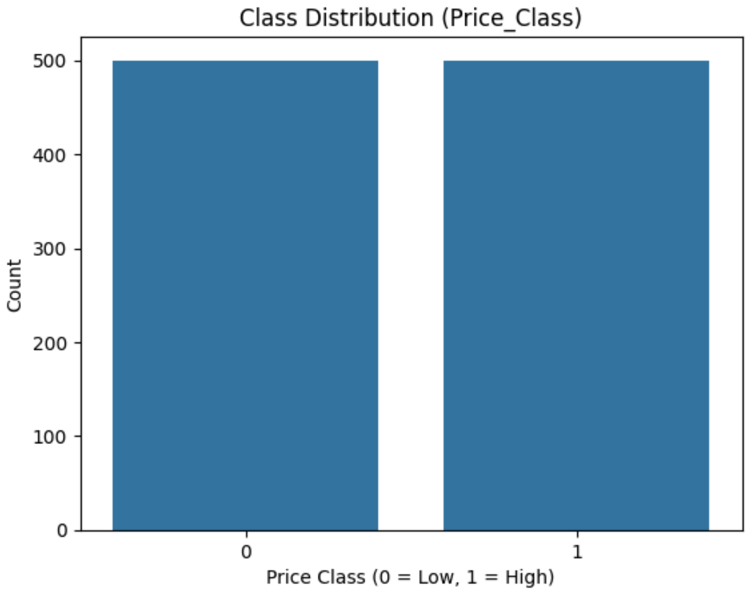
```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder

# Encode Condition column if it's still object type
if df_clean["Condition"].dtype == "object":
    le = LabelEncoder()
    df_clean["Condition"] = le.fit_transform(df_clean["Condition"])

# Pairplot with hue
sns.pairplot(df_clean[["Mileage", "Price", "Condition"]],
             diag_kind="kde",
             corner=True,
             hue="Condition",
             palette="Set2")

plt.suptitle("📊 Pairplot with Hue by Condition", y=1.02)
plt.show()
```

```
/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 128202 (\N{BAR CHART}) missing from font(s) D
  fig.canvas.print_figure(bytes_io, **kw)
```



Class Balancing Approach

```
# Recalculate median price
median_price = df_clean["Price"].median()

# Recreate binary class label
df_clean["Price_Class"] = (df_clean["Price"] > median_price).astype(int)



import seaborn as sns
import matplotlib.pyplot as plt

# Count plot
sns.countplot(x='Price_Class', data=df_clean)
plt.title("Class Distribution (Price_Class)")
plt.xlabel("Price Class (0 = Low, 1 = High)")
plt.ylabel("Count")
plt.show()

# Print counts
print(df_clean["Price_Class"].value_counts())
```




```
Price_Class
0    500
1    500
Name: count, dtype: int64
```

```
from sklearn.preprocessing import LabelEncoder

# Make a copy to avoid modifying original data
df_encoded = df_clean.copy()

# Encode all object (categorical) columns
for col in df_encoded.select_dtypes(include='object').columns:
    le = LabelEncoder()
    df_encoded[col] = le.fit_transform(df_encoded[col])
```

```
from imblearn.over_sampling import SMOTE

# Features and target
X = df_encoded.drop("Price_Class", axis=1)
y = df_encoded["Price_Class"]

# Apply SMOTE
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)

# Confirm balance
print("Class distribution after SMOTE:")
print(pd.Series(y_resampled).value_counts())
```

```
Class distribution after SMOTE:
Price_Class
0    500
1    500
Name: count, dtype: int64
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X_resampled, y_resampled, test_size=0.2, random_state=42, stratify=y_resampled
)
```

Splitting features

```
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import seaborn as sns
```

```
import pandas as pd

# Split data after SMOTE
X_train, X_test, y_train, y_test = train_test_split(
    X_resampled, y_resampled, test_size=0.2, stratify=y_resampled, random_state=42
)

# Combine into DataFrames for visualization
train_df = pd.DataFrame({'Price_Class': y_train, 'Set': 'Train'})
test_df = pd.DataFrame({'Price_Class': y_test, 'Set': 'Test'})
combined_df = pd.concat([train_df, test_df])

# Plot
plt.figure(figsize=(7, 5))
sns.countplot(x="Price_Class", hue="Set", data=combined_df, palette="Set2")
plt.title("📊 Class Distribution After Train-Test Split")
plt.xlabel("Price Class (0 = Low, 1 = High)")
plt.ylabel("Count")
plt.grid(True)
plt.show()
```

```
/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 128202 (\N{BAR CHART}) missing from font(s) D
  fig.canvas.print_figure(bytes_io, **kw)
```

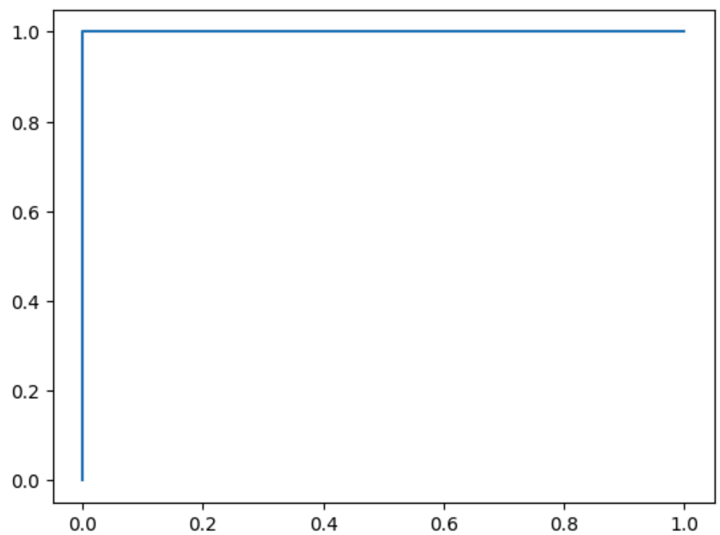
Train Models

Logistic Regression

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, roc_auc_score, roc_curve, matthews_corrcoef
import matplotlib.pyplot as plt

# Logistic Regression Model
logreg = LogisticRegression()

# Train the model
logreg.fit(X_train, y_train)
y_train_pred = logreg.predict(X_train)
y_test_pred = logreg.predict(X_test)
y_test_proba = logreg.predict_proba(X_test)[:, 1]

# Evaluation
print("===== Logistic Regression =====")
print("Train Accuracy:", accuracy_score(y_train, y_train_pred))
print("Test Accuracy:", accuracy_score(y_test, y_test_pred))
print("\nClassification Report:\n", classification_report(y_test, y_test_pred))
```

```
cm = confusion_matrix(y_test, y_test_pred)
tn, fp, fn, tp = cm.ravel()
print("Confusion Matrix:\n", cm)
print("Specificity:", tn / (tn + fp))
print("MCC:", matthews_corrcoef(y_test, y_test_pred))
print("ROC AUC Score:", roc_auc_score(y_test, y_test_proba))

# ROC curve
fpr, tpr, _ = roc_curve(y_test, y_test_proba)
plt.plot(fpr, tpr, label=f"Logistic Regression (AUC = {roc_auc_score(y_test, y_test_proba):.2f})")
```

```
===== Logistic Regression =====
Train Accuracy: 0.995
Test Accuracy: 1.0

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       100
           1       1.00      1.00      1.00       100

    accuracy                           1.00       200
   macro avg       1.00      1.00      1.00       200
weighted avg       1.00      1.00      1.00       200

Confusion Matrix:
 [[100   0]
 [  0 100]]
Specificity: 1.0
MCC: 1.0
ROC AUC Score: 1.0
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: 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(
[<matplotlib.lines.Line2D at 0x787f5b8c0fd0>]
```

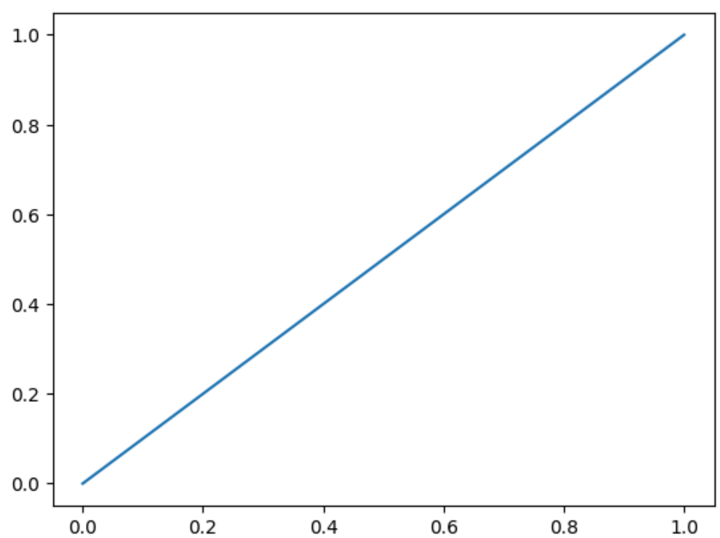


Support Vector Machine Model

```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, roc_auc_score, roc_curve, matthews_corrcoef
import matplotlib.pyplot as plt

# Support Vector Machine Model
svm = SVC(probability=True)

# Train the model
svm.fit(X_train, y_train)
y_train_pred = svm.predict(X_train)
y_test_pred = svm.predict(X_test)
```

```
y_test_proba = svm.predict_proba(X_test)[:, 0]

# Evaluation
print("===== Support Vector Machine (SVM) =====")
print("Train Accuracy:", accuracy_score(y_train, y_train_pred))
print("Test Accuracy:", accuracy_score(y_test, y_test_pred))
print("\nClassification Report:\n", classification_report(y_test, y_test_pred))

cm = confusion_matrix(y_test, y_test_pred)
tn, fp, fn, tp = cm.ravel()
print("Confusion Matrix:\n", cm)
print("Specificity:", tn / (tn + fp))
print("MCC:", matthews_corrcoef(y_test, y_test_pred))
print("ROC AUC Score:", roc_auc_score(y_test, y_test_proba))

# ROC curve
fpr, tpr, _ = roc_curve(y_test, y_test_proba)
plt.plot(fpr, tpr, label=f"SVM (AUC = {roc_auc_score(y_test, y_test_proba):.2f})")
```

```
===== Support Vector Machine (SVM) =====
Train Accuracy: 0.8875
Test Accuracy: 0.9

Classification Report:
               precision    recall  f1-score   support

           0       0.93      0.87      0.90       100
           1       0.88      0.93      0.90       100

    accuracy                           0.90       200
   macro avg       0.90      0.90      0.90       200
weighted avg       0.90      0.90      0.90       200

Confusion Matrix:
 [[87 13]
 [ 7 93]]
Specificity: 0.87
MCC: 0.801443899700861
ROC AUC Score: 0.5
[<matplotlib.lines.Line2D at 0x787f5b76e4d0>]
```

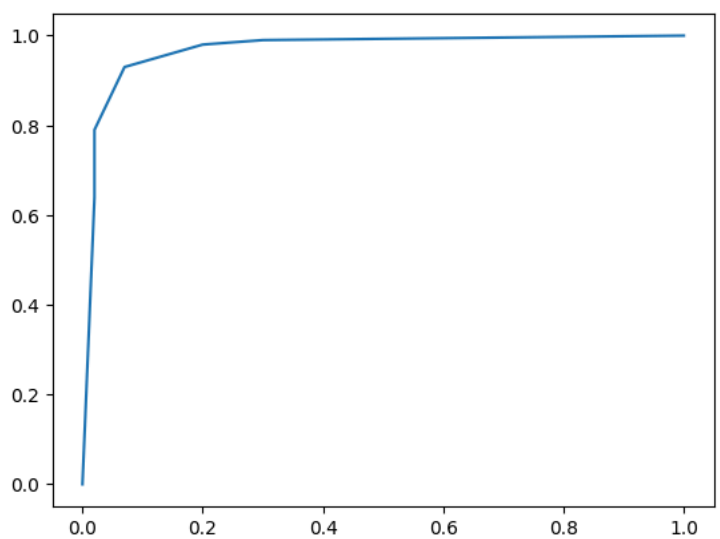


K-Nearest Neighbors Model

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, roc_auc_score, roc_curve, matthews_corrcoef
import matplotlib.pyplot as plt

# K-Nearest Neighbors Model
knn = KNeighborsClassifier()

# Train the model
knn.fit(X_train, y_train)
y_train_pred = knn.predict(X_train)
```

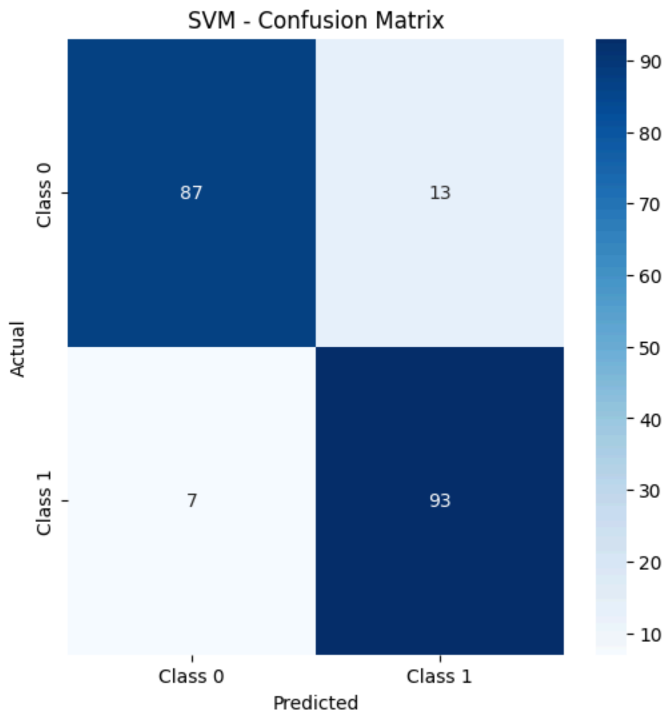
```
y_test_pred = knn.predict(X_test)
y_test_proba = knn.predict_proba(X_test)[:, 1]

# Evaluation
print("===== K-Nearest Neighbors (KNN) =====")
print("Train Accuracy:", accuracy_score(y_train, y_train_pred))
print("Test Accuracy:", accuracy_score(y_test, y_test_pred))
print("\nClassification Report:\n", classification_report(y_test, y_test_pred))

cm = confusion_matrix(y_test, y_test_pred)
tn, fp, fn, tp = cm.ravel()
print("Confusion Matrix:\n", cm)
print("Specificity:", tn / (tn + fp))
print("MCC:", matthews_corrcoef(y_test, y_test_pred))
print("ROC AUC Score:", roc_auc_score(y_test, y_test_proba))

# ROC curve
fpr, tpr, _ = roc_curve(y_test, y_test_proba)
plt.plot(fpr, tpr, label=f"KNN (AUC = {roc_auc_score(y_test, y_test_proba):.2f})")
```

```
===== K-Nearest Neighbors (KNN) =====
Train Accuracy: 0.94625
Test Accuracy: 0.93

Classification Report:
               precision    recall  f1-score   support

           0       0.93      0.93      0.93       100
           1       0.93      0.93      0.93       100

    accuracy                           0.93       200
   macro avg       0.93      0.93      0.93       200
weighted avg       0.93      0.93      0.93       200

Confusion Matrix:
 [[93  7]
 [ 7 93]]
Specificity: 0.93
MCC: 0.86
ROC AUC Score: 0.96855
[<matplotlib.lines.Line2D at 0x787f5b7e4190>]
```




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

# Train the SVM model
svm = SVC(probability=True)
svm.fit(X_train, y_train)

# Predict on test set
y_test_pred_svm = svm.predict(X_test)

# Generate Confusion Matrix
```

```
cm_svm = confusion_matrix(y_test, y_test_pred_svm)

# Plot Confusion Matrix
plt.figure(figsize=(6, 6))
sns.heatmap(cm_svm, annot=True, fmt='d', cmap='Blues', xticklabels=["Class 0", "Class 1"], yticklabels=["Class 0", "Class 1"])
plt.title('SVM - Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```




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

# Train the KNN model
knn = KNeighborsClassifier()
```