Import Necessary Package

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

```
data=pd.read_csv('C:\Rohith\Backup\Desktop\SEM 6\Machine Learning Lab\Practices\Online Sh
```

```
data.head()
```

	Administrative	Administrative_Duration	Informational	Informational_Duration	Pr
0	0	0.0	0	0.0	
1	0	0.0	0	0.0	
2	0	0.0	0	0.0	
3	0	0.0	0	0.0	
4	0	0.0	0	0.0	

```
data.describe()
```

	Administrative	Administrative_Duration	Informational	Informational_Duration
count	12330.000000	12330.000000	12330.000000	12330.000000
mean	2.315166	80.818611	0.503569	34.472398
std	3.321784	176.779107	1.270156	140.749294
min	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000
50%	1.000000	7.500000	0.000000	0.000000
75%	4.000000	93.256250	0.000000	0.000000
max	27.000000	3398.750000	24.000000	2549.375000

```
data.shape
```

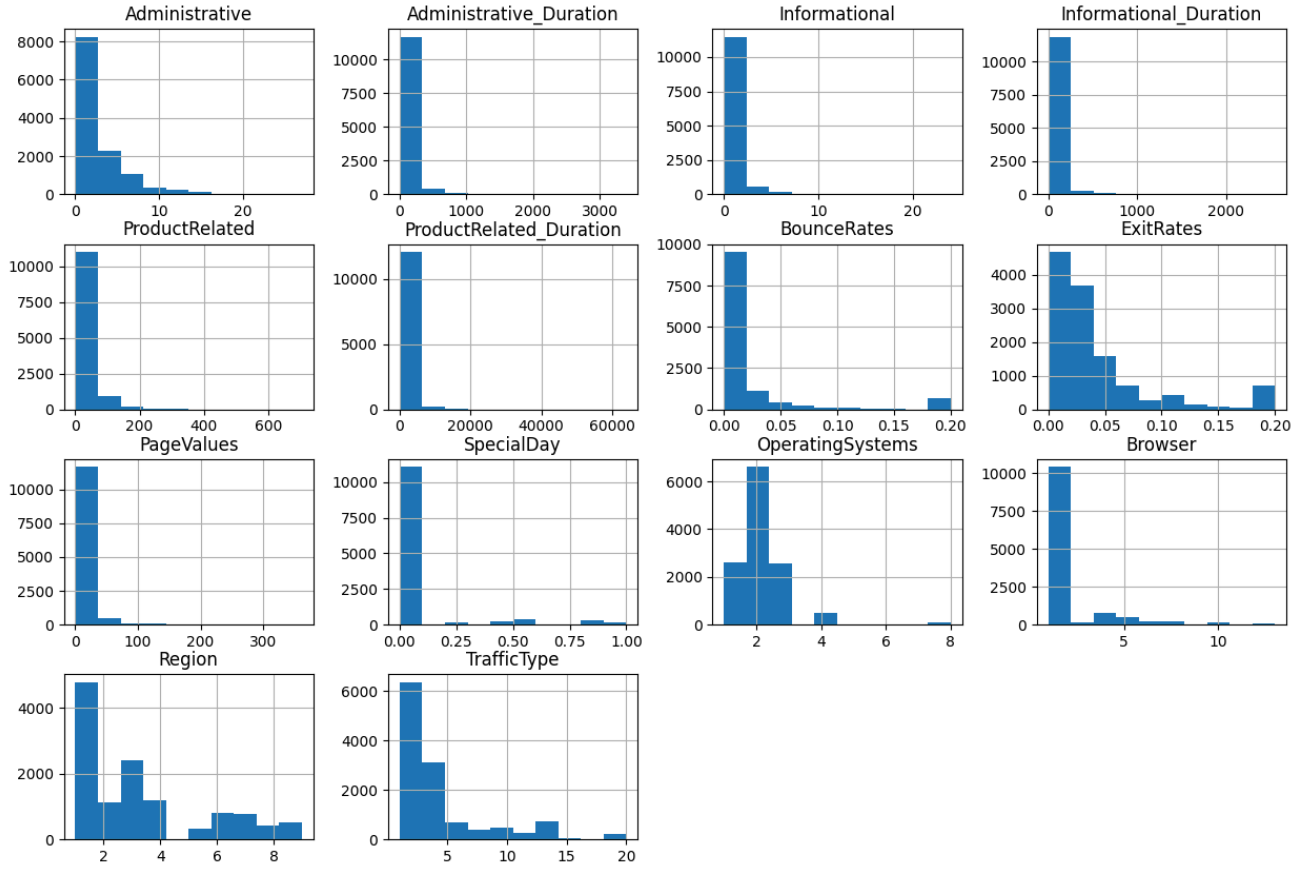
```
(12330, 18)
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12330 entries, 0 to 12329
Data columns (total 18 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   Administrative           12330 non-null  int64  
 1   Administrative_Duration  12330 non-null  float64
 2   Informational            12330 non-null  int64  
 3   Informational_Duration   12330 non-null  float64
 4   ProductRelated           12330 non-null  int64  
 5   ProductRelated_Duration  12330 non-null  float64
 6   BounceRates              12330 non-null  float64
 7   ExitRates                12330 non-null  float64
 8   PageValues               12330 non-null  float64
 9   SpecialDay               12330 non-null  float64
 10  Month                    12330 non-null  object 
 11  OperatingSystems         12330 non-null  int64  
 12  Browser                  12330 non-null  int64  
 13  Region                   12330 non-null  int64  
 14  TrafficType              12330 non-null  int64  
 15  VisitorType              12330 non-null  object 
 16  Weekend                  12330 non-null  bool   
 17  Revenue                  12330 non-null  bool   
dtypes: bool(2), float64(7), int64(7), object(2)
memory usage: 1.5+ MB
```

```
data.hist(figsize=(15,10))
```

```
array([[<Axes: title={'center': 'Administrative'}>,
        <Axes: title={'center': 'Administrative_Duration'}>,
        <Axes: title={'center': 'Informational'}>,
        <Axes: title={'center': 'Informational_Duration'}>],
       [<Axes: title={'center': 'ProductRelated'}>,
        <Axes: title={'center': 'ProductRelated_Duration'}>,
        <Axes: title={'center': 'BounceRates'}>,
        <Axes: title={'center': 'ExitRates'}>],
       [<Axes: title={'center': 'PageValues'}>,
        <Axes: title={'center': 'SpecialDay'}>,
        <Axes: title={'center': 'OperatingSystems'}>,
        <Axes: title={'center': 'Browser'}>],
       [<Axes: title={'center': 'Region'}>,
        <Axes: title={'center': 'TrafficType'}>, <Axes: >, <Axes: >]],
      dtype=object)
```



```
#Checking the null values
data.isnull().sum()
```

```
Administrative             0
Administrative_Duration    0
Informational              0
Informational_Duration     0
ProductRelated             0
ProductRelated_Duration    0
BounceRates                0
ExitRates                  0
PageValues                 0
SpecialDay                 0
Month                      0
OperatingSystems           0
Browser                    0
Region                     0
TrafficType                0
VisitorType                0
Weekend                    0
Revenue                    0
dtype: int64
```

```
data.duplicated().sum()
```

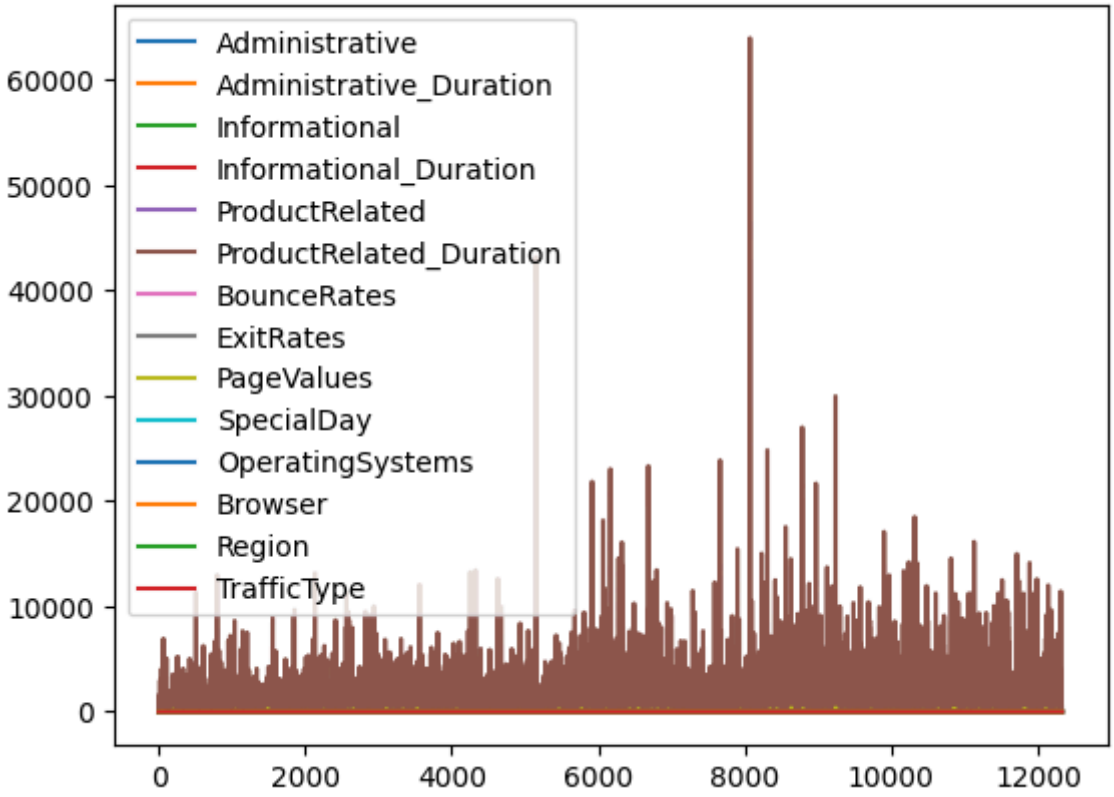
```
125
```

```
data.drop_duplicates(inplace=True)
data.shape
```

```
(12205, 18)
```

```
data.plot()
```

<Axes: >




```
data['ProductRelated_Duration'].unique()
```

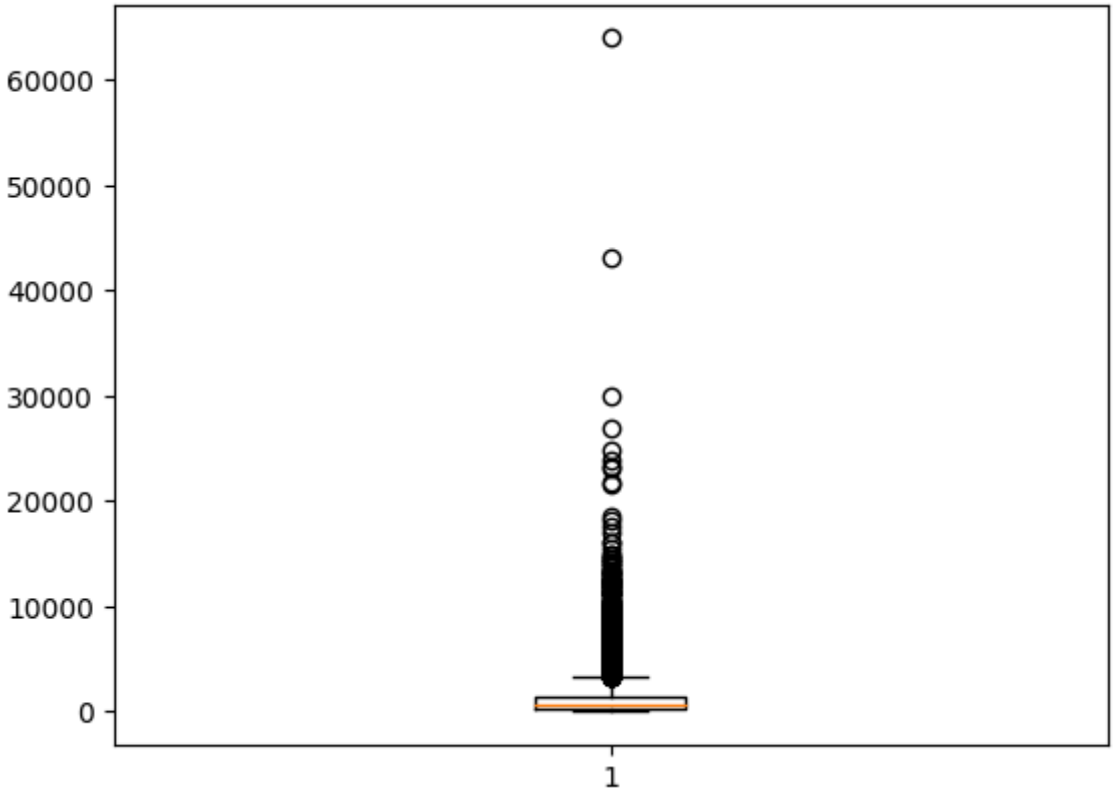
```
array([  0.        ,  64.        ,   2.66666667, ..., 465.75      ,
       184.25      , 346.        ])
```

```
max(data['ProductRelated_Duration'].unique())
```

```
63973.52223
```

```
plt.boxplot(data['ProductRelated_Duration'])
```

```
{'whiskers': [<matplotlib.lines.Line2D at 0x22a5e659270>,
  <matplotlib.lines.Line2D at 0x22a5e659510>],
 'caps': [<matplotlib.lines.Line2D at 0x22a5e6597b0>,
  <matplotlib.lines.Line2D at 0x22a5e659a50>],
 'boxes': [<matplotlib.lines.Line2D at 0x22a5efe2410>],
 'medians': [<matplotlib.lines.Line2D at 0x22a5e659cf0>],
 'fliers': [<matplotlib.lines.Line2D at 0x22a5e659ff0>],
 'means': []}
```



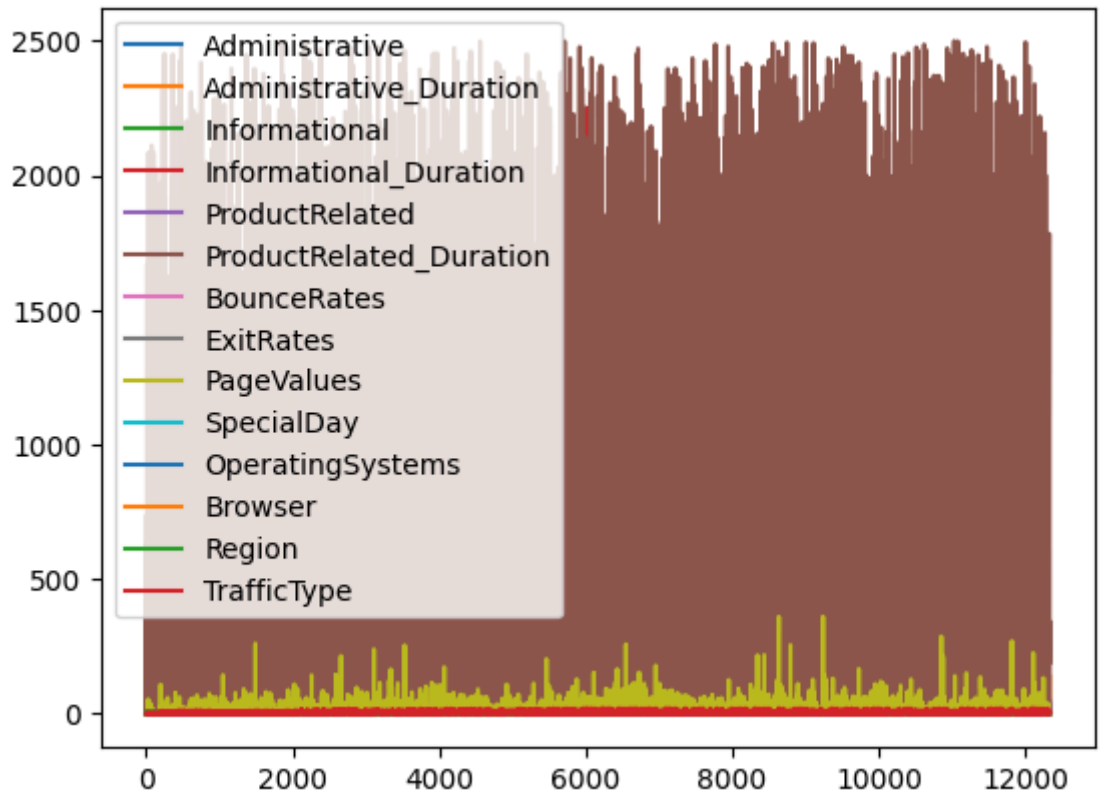

```
df=data[data['ProductRelated_Duration']<=2500]
df
```

	Administrative	Administrative_Duration	Informational	Informational_Duratio
0	0	0.0	0	0.(
1	0	0.0	0	0.(
2	0	0.0	0	0.(
3	0	0.0	0	0.(
4	0	0.0	0	0.(
...	...	...	...	.
12325	3	145.0	0	0.(
12326	0	0.0	0	0.(
12327	0	0.0	0	0.(
12328	4	75.0	0	0.(
12329	0	0.0	0	0.(

10644 rows × 18 columns

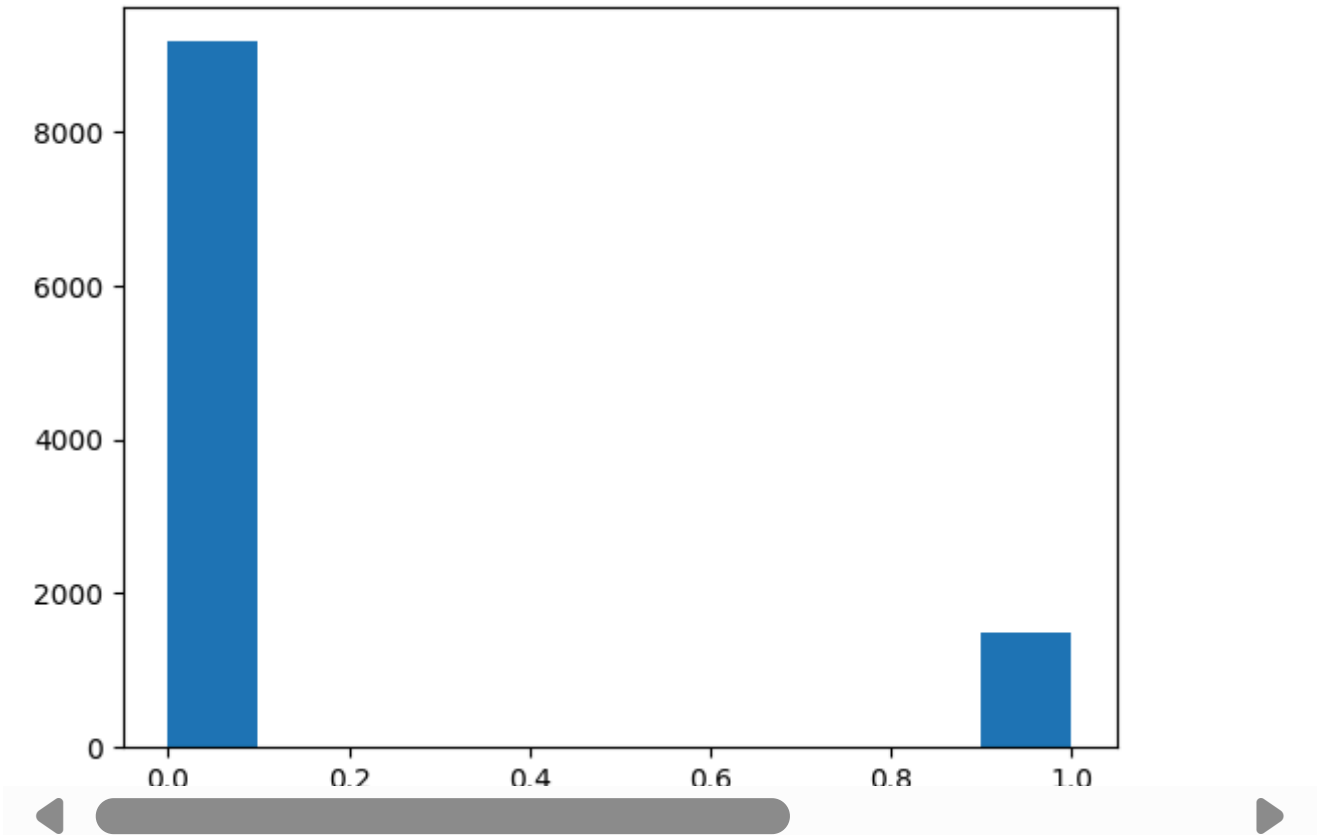
```
df.plot()
```

```
<Axes: >
```



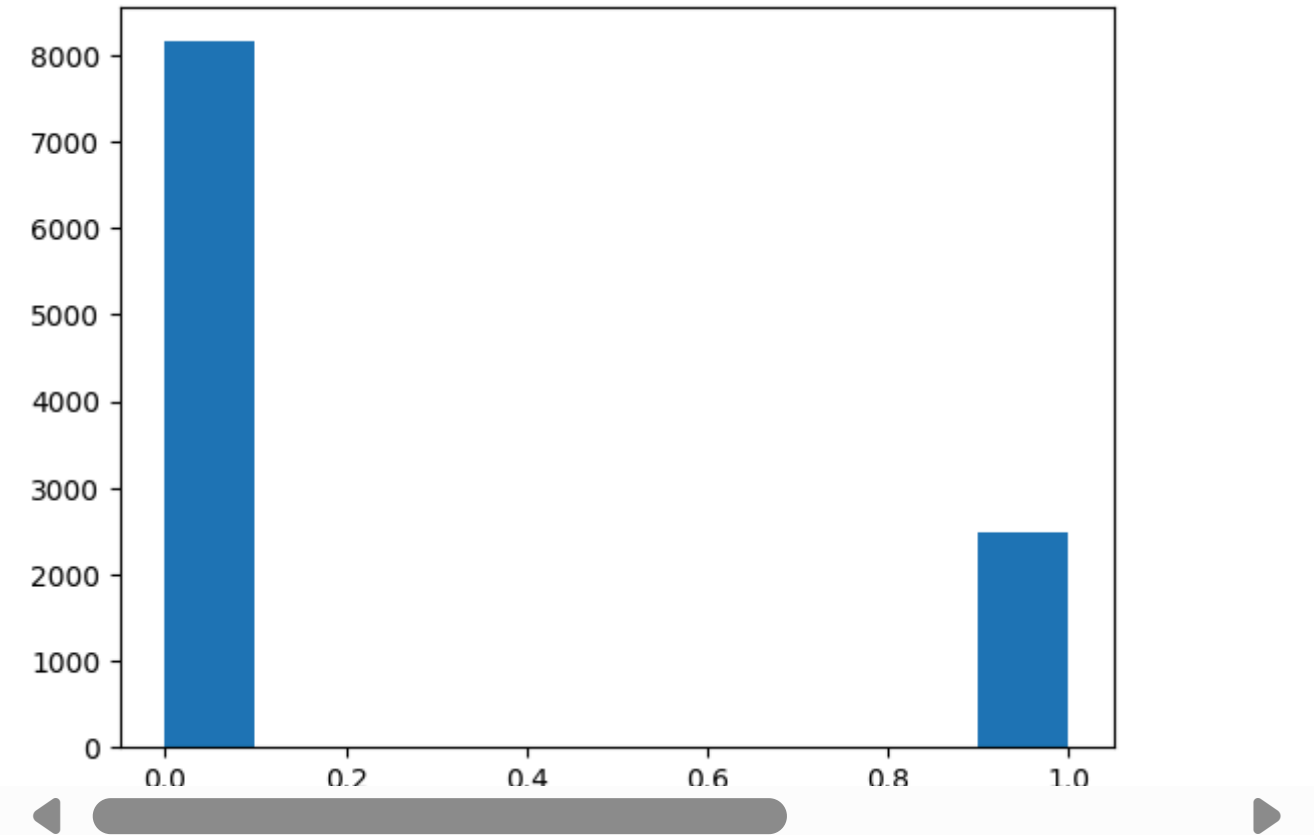

```
df['Revenue']=np.where(df['Revenue'],1,0)       #Converting True=1 and False=0
plt.hist(df['Revenue'])
```

```
C:\Users\rohit\AppData\Local\Temp\ipykernel_17388\3012526398.py:1: SettingWithCopyWar
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
  df['Revenue']=np.where(df['Revenue'],1,0)       #Converting True=1 and False=0
(array([9164.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,
        1480.]),
 array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ]),
 <BarContainer object of 10 artists>)
```



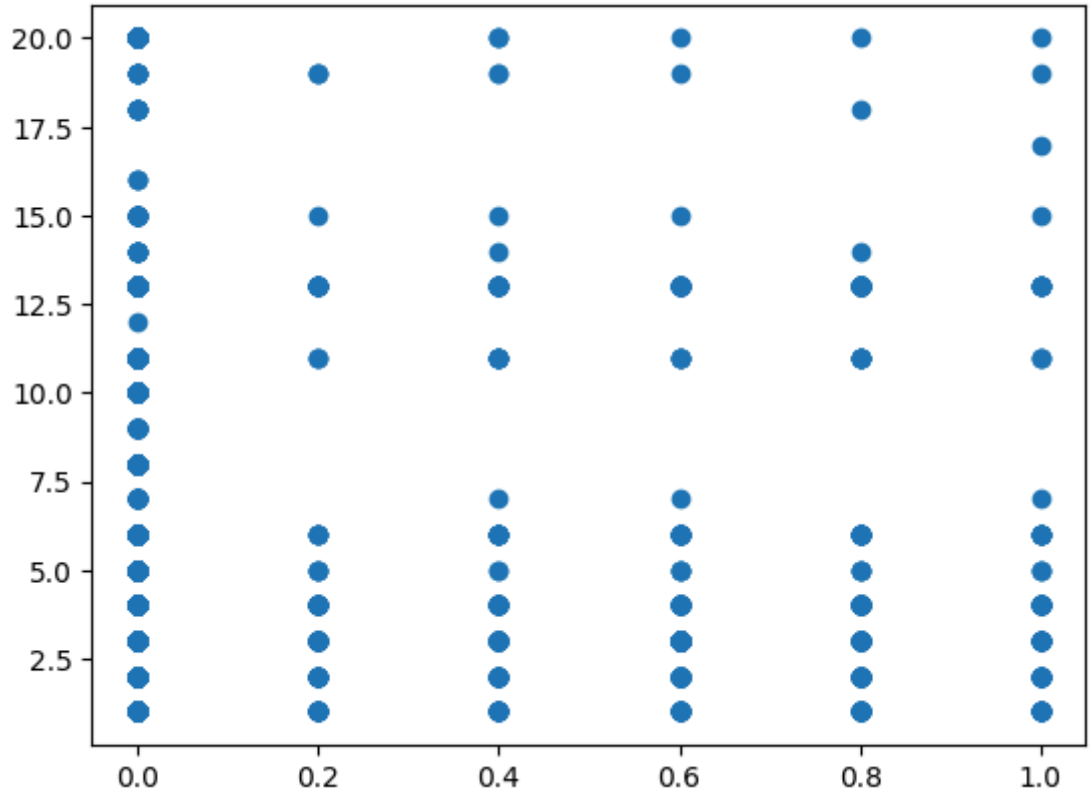

```
df['Weekend']=np.where(df['Weekend'],1,0)
plt.hist(df['Weekend'])
```

```
C:\Users\rohit\AppData\Local\Temp\ipykernel_17388\1244464083.py:1: SettingWithCopyWar
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
  df['Weekend']=np.where(df['Weekend'],1,0)
(array([8154.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,
        2490.]),
 array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ]),
 <BarContainer object of 10 artists>)
```




```
plt.scatter(x=df['SpecialDay'],y=df['TrafficType'])
```

```
<matplotlib.collections.PathCollection at 0x22a5e41a860>
```




```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()           #Convert Object types into numeric values


le.fit(df['Month'])
df['Month']=le.transform(df['Month'])
```

```
C:\Users\rohit\AppData\Local\Temp\ipykernel_17388\3157372182.py:2: SettingWithCopyWar
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
  df['Month']=le.transform(df['Month'])
```

```
le.fit(df['VisitorType'])
df['VisitorType']=le.transform(df['VisitorType'])
```

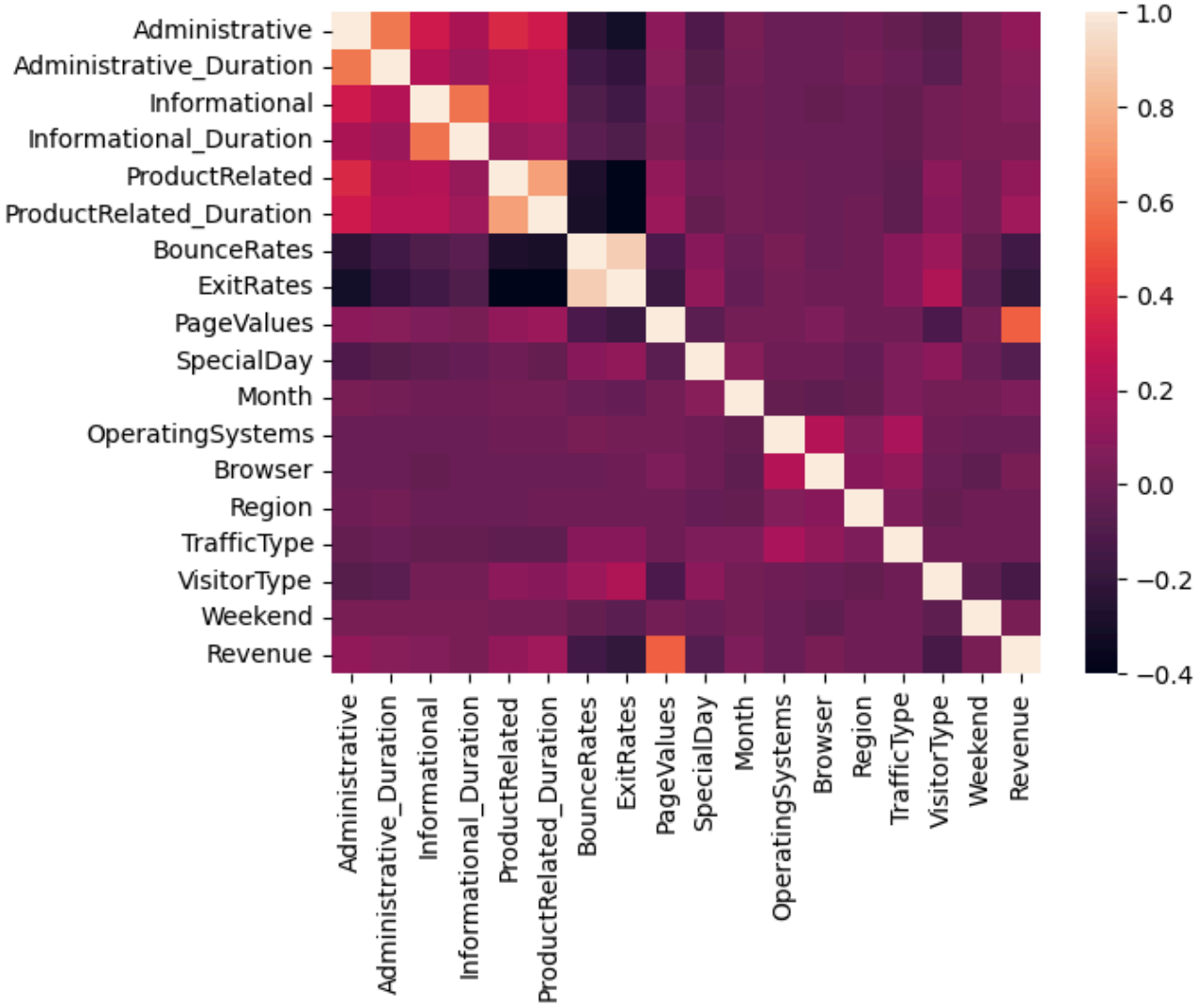
```
C:\Users\rohit\AppData\Local\Temp\ipykernel_17388\3298818332.py:2: SettingWithCopyWar
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
  df['VisitorType']=le.transform(df['VisitorType'])
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 10644 entries, 0 to 12329
Data columns (total 18 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   Administrative           10644 non-null  int64  
 1   Administrative_Duration  10644 non-null  float64
 2   Informational            10644 non-null  int64  
 3   Informational_Duration   10644 non-null  float64
 4   ProductRelated           10644 non-null  int64  
 5   ProductRelated_Duration  10644 non-null  float64
 6   BounceRates              10644 non-null  float64
 7   ExitRates                10644 non-null  float64
 8   PageValues               10644 non-null  float64
 9   SpecialDay               10644 non-null  float64
 10  Month                    10644 non-null  int32  
 11  OperatingSystems         10644 non-null  int64  
 12  Browser                  10644 non-null  int64  
 13  Region                   10644 non-null  int64  
 14  TrafficType              10644 non-null  int64  
 15  VisitorType              10644 non-null  int32  
 16  Weekend                  10644 non-null  int32  
 17  Revenue                  10644 non-null  int32  
dtypes: float64(7), int32(4), int64(7)
memory usage: 1.4 MB
```

```
sns.heatmap(df.corr())
```

```
<Axes: >
```



Training and testing

```
X=df.drop('Revenue',axis=1)
y=df['Revenue']
```

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=42)
```

```
print("X_train = ",X_train.shape)
print("X_test = ",X_test.shape)
print("y_train = ",y_train.shape)
print("y_test = ",y_test.shape)
```

```
X_train =  (8515, 17)
X_test =  (2129, 17)
y_train =  (8515,)
y_test =  (2129,)
```

Model 1 Linear Regression

```
from sklearn.linear_model import LinearRegression
li=LinearRegression()
li
```

```
LinearRegression()
```

```
li.fit(X_train,y_train)
y_pred=li.predict(X_test)
accuracy=li.score(X_test,y_test)
accuracy
```

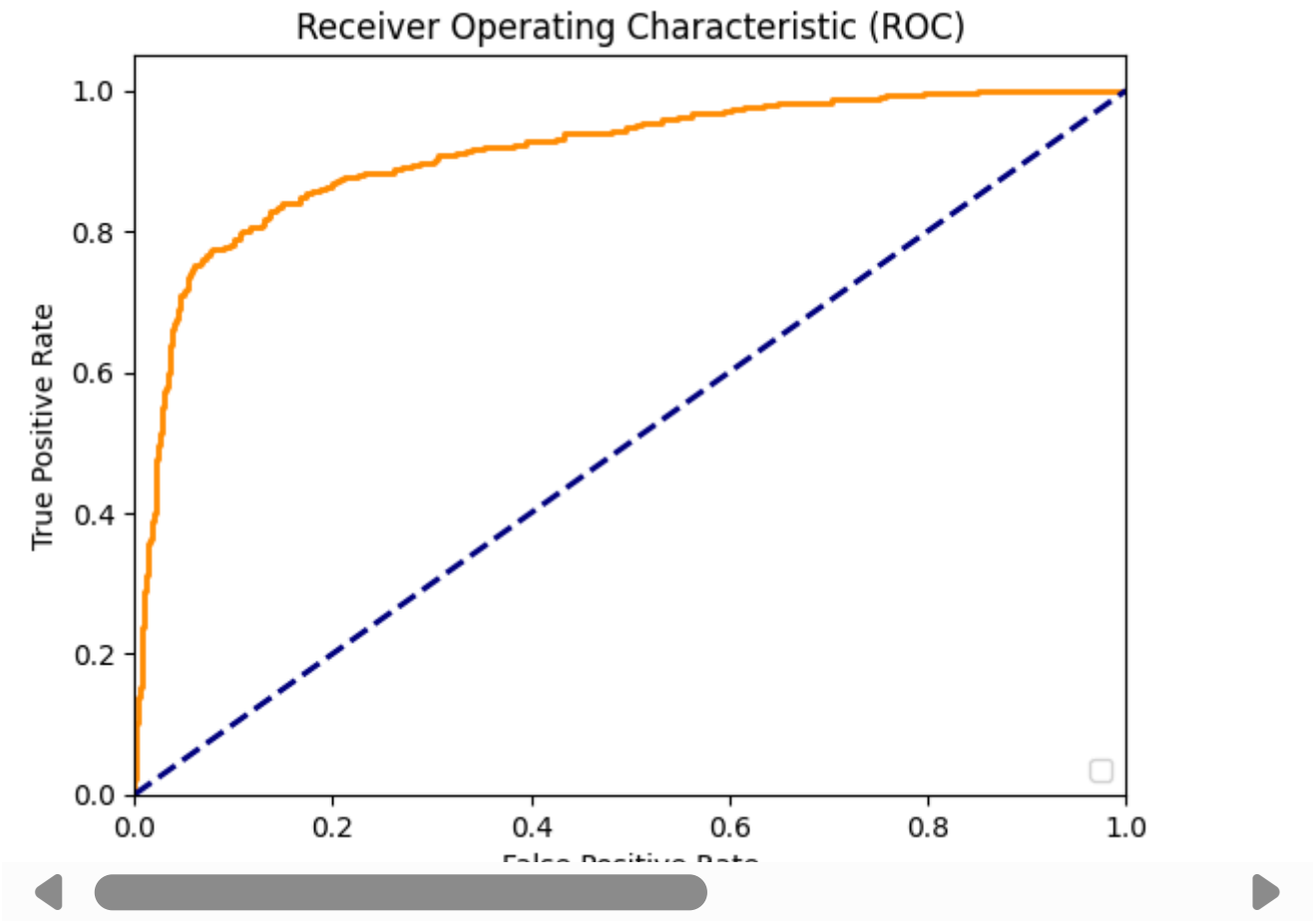
```
0.3305981754646069
```

```
from sklearn.metrics import mean_absolute_error,mean_squared_error

# Calculate Mean Absolute Error (MAE)
mae = mean_absolute_error(y_test, y_pred)
print("Mean Absolute Error (MAE):", mae)

# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error (MSE):", mse)

# Calculate Root Mean Squared Error (RMSE)
rmse = np.sqrt(mse)
print("Root Mean Squared Error (RMSE):", rmse)
```

```
Mean Absolute Error (MAE): 0.1747474516040395
Mean Squared Error (MSE): 0.0814856359227879
Root Mean Squared Error (RMSE): 0.28545688977985434
```

```
#Plotting ROC curve
from sklearn.metrics import roc_curve,auc,accuracy_score

fpr,tpr,threshold=roc_curve(y_test,y_pred)
roc_auc=auc(fpr,tpr)

plt.figure()
plt.plot(fpr,tpr,color='darkorange',lw=2)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc="lower right")
plt.show()
```

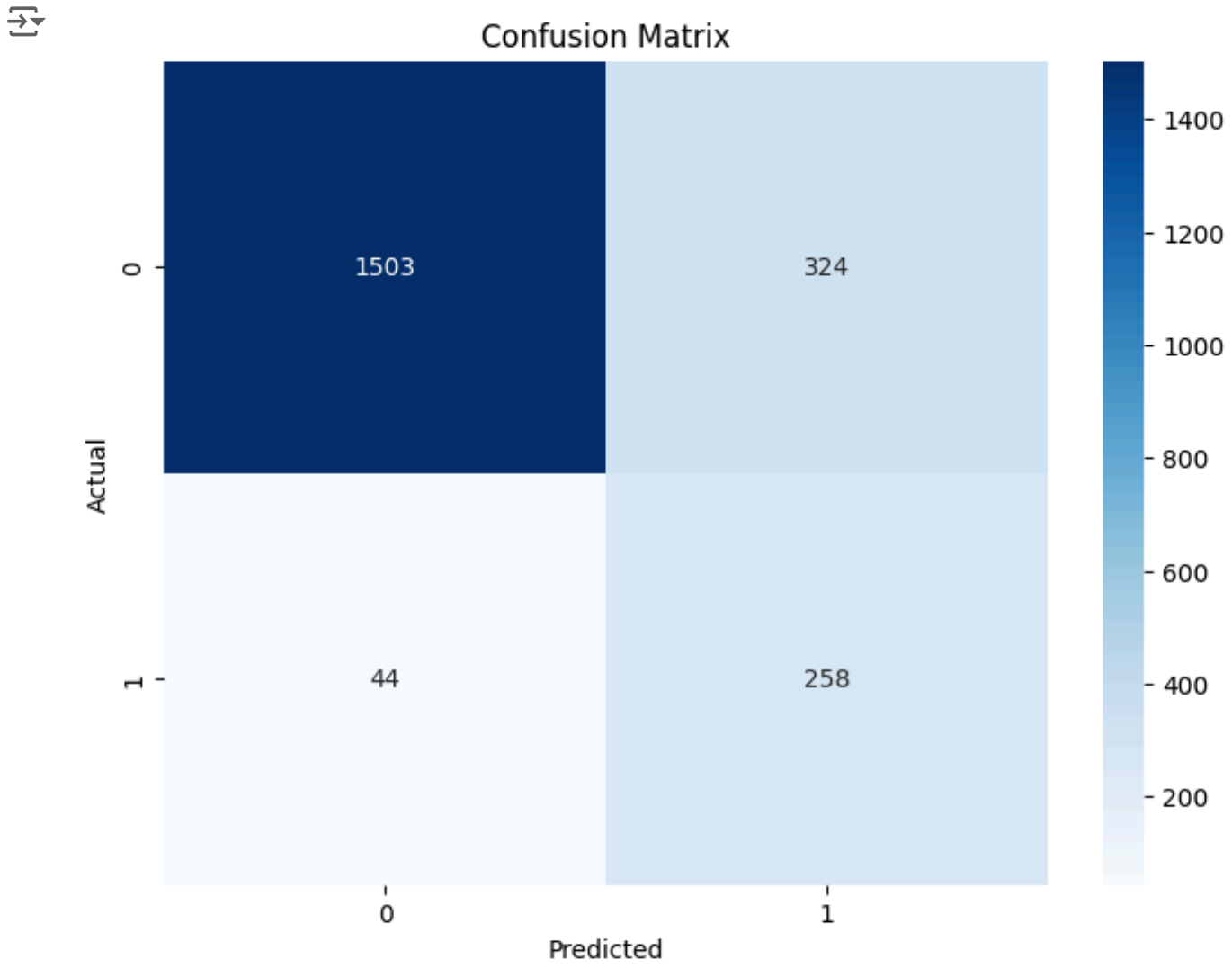
```
No artists with labels found to put in legend.  Note that artists whose label start w
```



```
from sklearn.metrics import confusion_matrix

# Convert regression output to binary classification
threshold = y_test.mean()  # Example threshold, you can set your own
y_test_class = (y_test > threshold).astype(int)
y_pred_class = (y_pred > threshold).astype(int)

# Compute confusion matrix
cm = confusion_matrix(y_test_class, y_pred_class)
accuracy = accuracy_score(y_test_class, y_pred_class)
print("Linear Regression Accuracy: \n ",accuracy)
print("Linear Regression Confusion Matrix: \n",cm)
```

```
Linear Regression Accuracy:
  0.8271488961953969
Linear Regression Confusion Matrix:
 [[1503  324]
 [  44  258]]
```

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




```
from sklearn.metrics import classification_report
print(classification_report(y_test_class,y_pred_class))
```

```
              precision    recall  f1-score   support

           0       0.97      0.82      0.89      1827
           1       0.44      0.85      0.58       302

    accuracy                           0.83      2129
   macro avg       0.71      0.84      0.74      2129
weighted avg       0.90      0.83      0.85      2129
```

Model 2 Logistic Regression

```
from sklearn.linear_model import LogisticRegression
lr=LogisticRegression()
lr
```

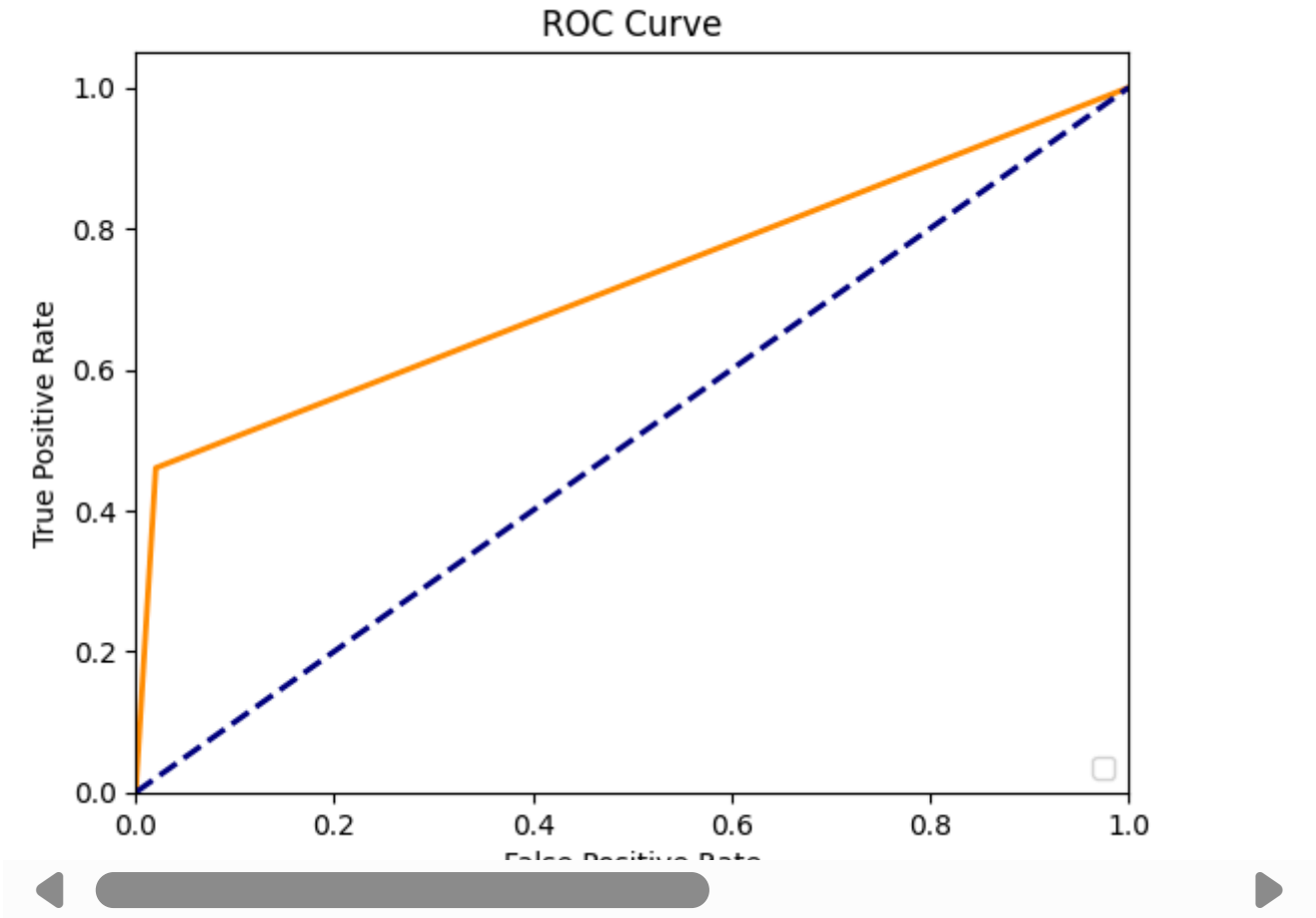
```
LogisticRegression()
```

```
lr.fit(X_train,y_train)
y_pred=lr.predict(X_test)
accuracy=lr.score(X_test,y_test)
accuracy
```

```
c:\Python310\lib\site-packages\sklearn\linear_model\_logistic.py:469: ConvergenceWarn
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(
0.9060591827148896
```

```
#Plotting ROC curve
from sklearn.metrics import roc_curve,auc,accuracy_score

fpr,tpr,threshold=roc_curve(y_test,y_pred)
roc_auc=auc(fpr,tpr)

plt.figure()
plt.plot(fpr,tpr,color="darkorange",lw=2)
plt.plot([0,1],[0,1],color="navy",lw=2,linestyle='--')
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc='lower right')
plt.show()
```

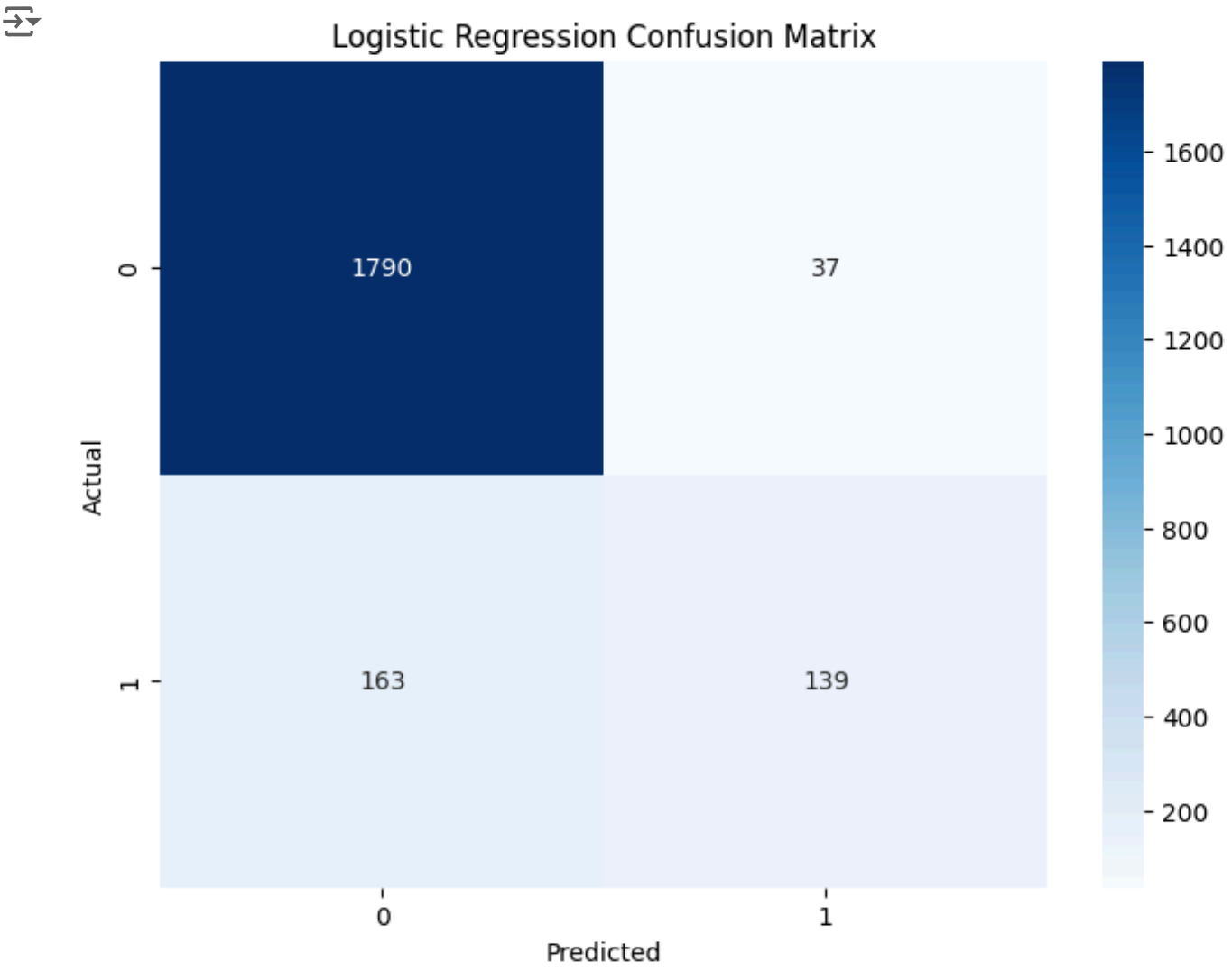
```
No artists with labels found to put in legend.  Note that artists whose label start w
```




```
#Confusion matrix
from sklearn.metrics import confusion_matrix

cm=confusion_matrix(y_test,y_pred)
print("Logistic REgression Confusion matrix: \n",cm)
```

```
Logistic REgression Confusion matrix:
 [[1790   37]
 [ 163  139]]
```

```
#Plot Confusion Matrix
plt.figure(figsize=(8,6))
sns.heatmap(cm,annot=True,fmt='d',cmap='Blues')
plt.title('Logistic Regression Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```




```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.92      0.98      0.95      1827
           1       0.79      0.46      0.58       302

    accuracy                           0.91      2129
   macro avg       0.85      0.72      0.76      2129
weighted avg       0.90      0.91      0.90      2129
```

Model 3 SVM

```
from sklearn.svm import SVC
svm_classifier=SVC(probability=True)
svm_classifier
```

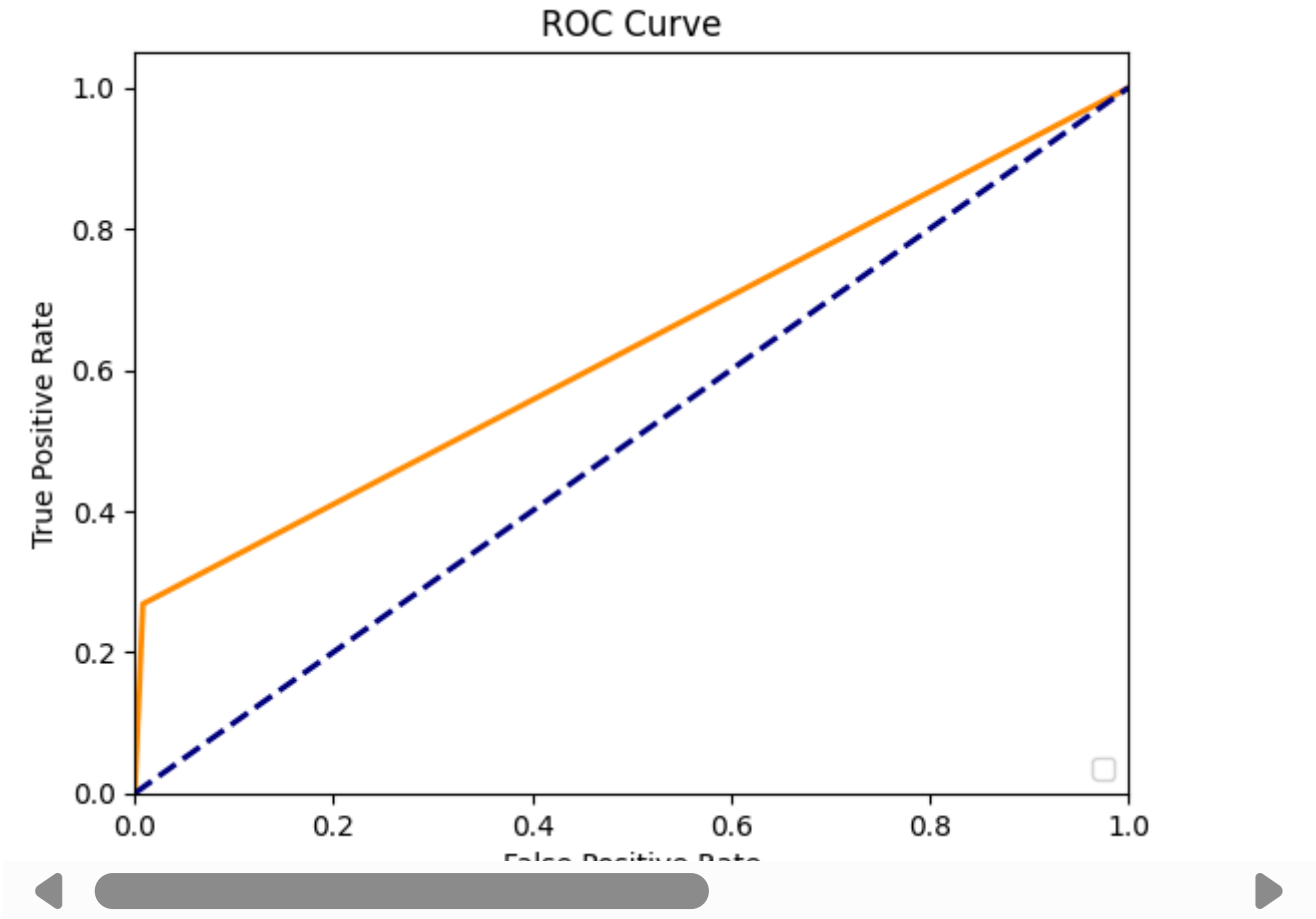
```
SVC
SVC(probability=True)
```

```
svm_classifier.fit(X_train,y_train)
y_pred=svm_classifier.predict(X_test)
accuracy=accuracy_score(y_test,y_pred)
accuracy
```

```
0.8891498356035697
```

```
#Plotting ROC_curve
from sklearn.metrics import roc_curve,auc,accuracy_score

fpr,tpr,threshold=roc_curve(y_test,y_pred)
roc_auc=auc(fpr,tpr)

plt.figure()
plt.plot(fpr,tpr,color='darkorange',lw=2)
plt.plot([0,1], [0,1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc='lower right')
plt.show()
```

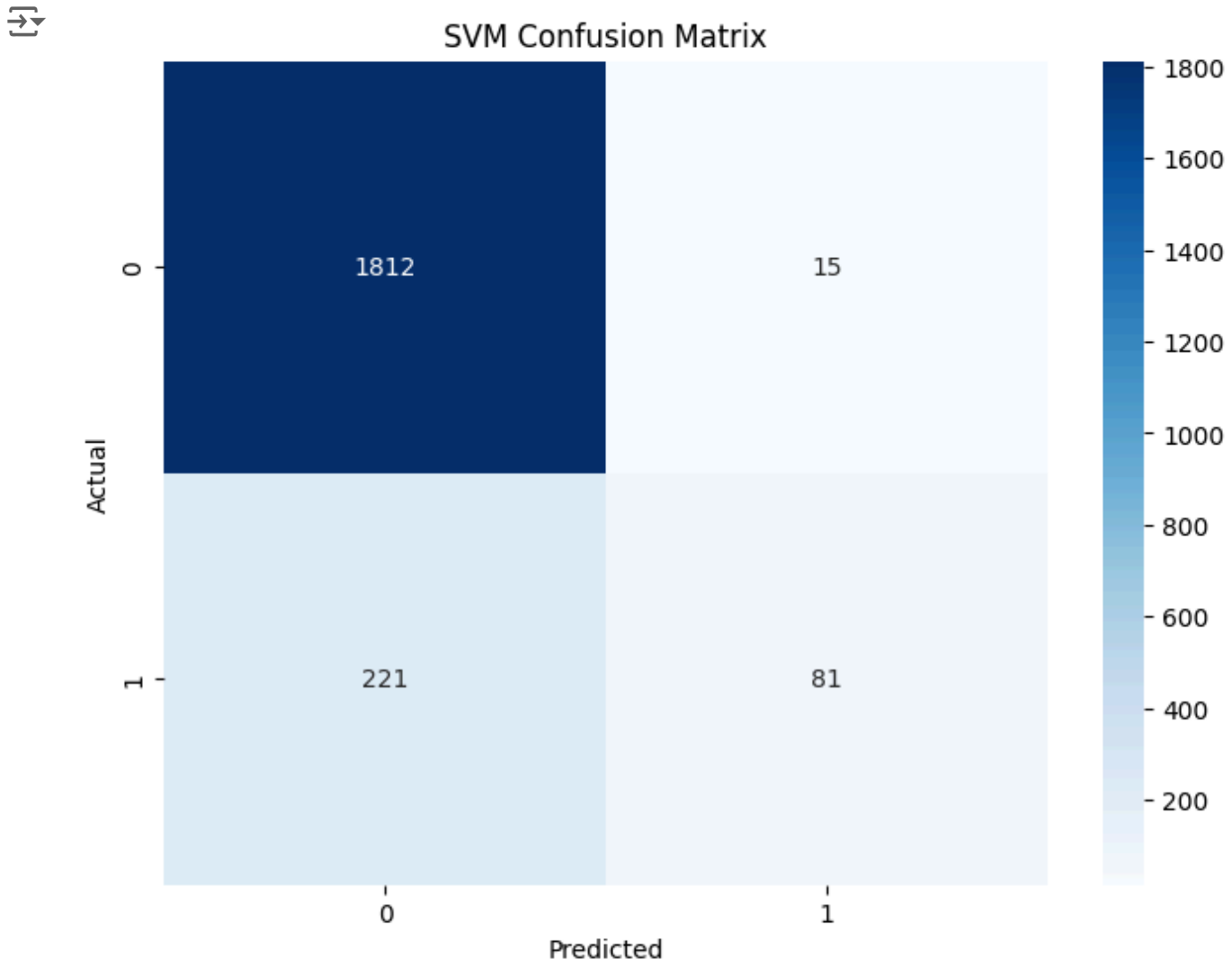
```
No artists with labels found to put in legend.  Note that artists whose label start w
```




```
#Confusion matrix
from sklearn.metrics import confusion_matrix

cm=confusion_matrix(y_test,y_pred)
print("SVM Confusion Matrix: \n",cm)
```

```
SVM Confusion Matrix:
 [[1812   15]
 [ 221   81]]
```

```
#Plot Confusion Matrix
plt.figure(figsize=(8,6))
sns.heatmap(cm,annot=True,fmt='d',cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('SVM Confusion Matrix')
plt.show()
```




```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.89      0.99      0.94      1827
           1       0.84      0.27      0.41       302

    accuracy                           0.89      2129
   macro avg       0.87      0.63      0.67      2129
weighted avg       0.88      0.89      0.86      2129
```

Model 4 Decision Tree

```
from sklearn.tree import DecisionTreeClassifier

dt_classifier=DecisionTreeClassifier()
dt_classifier
```

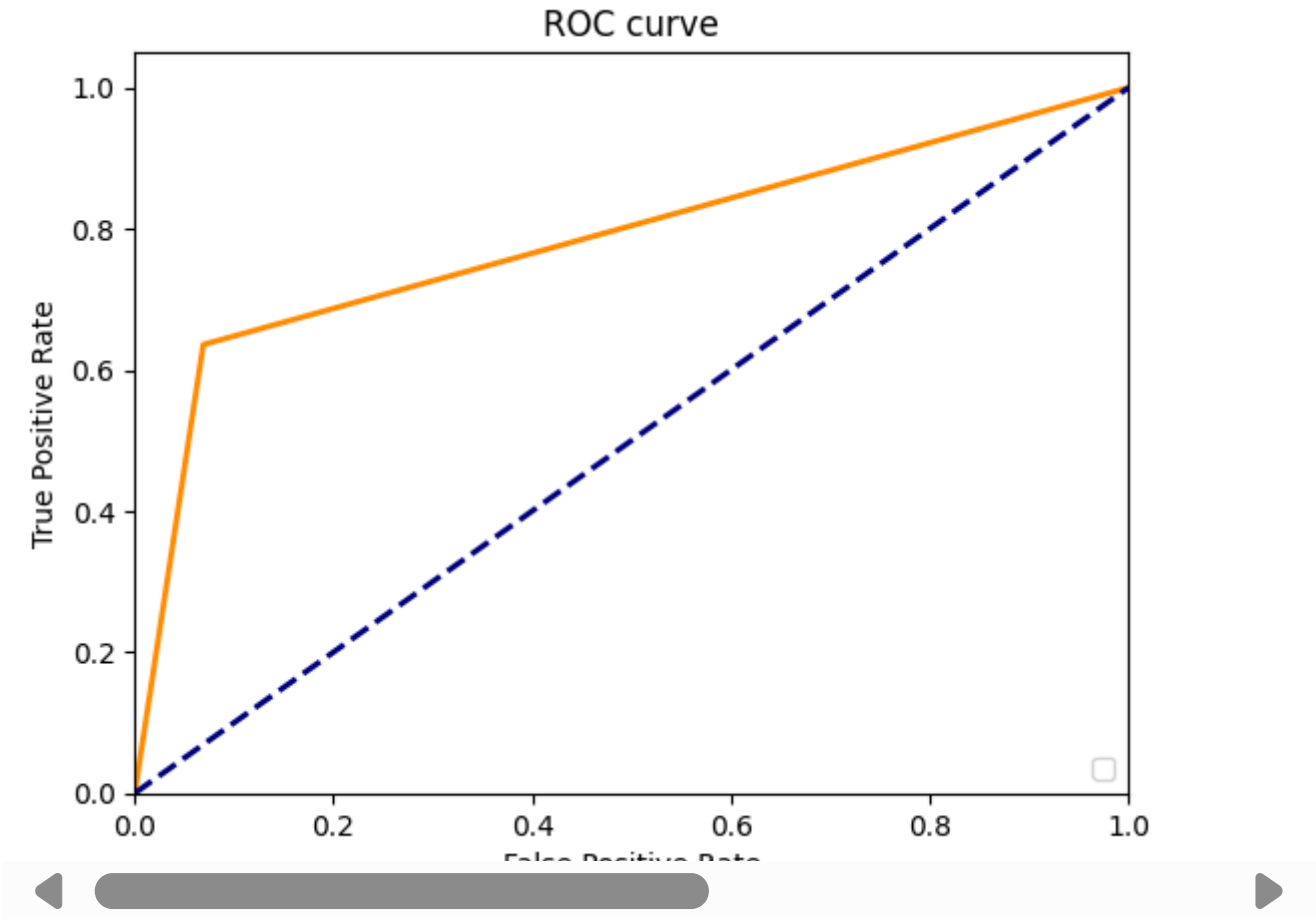
```
DecisionTreeClassifier
DecisionTreeClassifier()
```

```
dt_classifier.fit(X_train,y_train)
y_pred=dt_classifier.predict(X_test)
accuracy=accuracy_score(y_test,y_pred)
accuracy
```

```
0.8891498356035697
```

```
#Plotting ROC Curve
from sklearn.metrics import roc_curve,auc,accuracy_score

fpr,tpr,threshold=roc_curve(y_test,y_pred)
roc_auc=auc(fpr,tpr)

plt.figure()
plt.plot(fpr,tpr,color='darkorange',lw=2)
plt.plot([0,1], [0,1], color='navy',linestyle='--',lw=2)
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC curve')
plt.legend(loc='lower right')
plt.show()
```

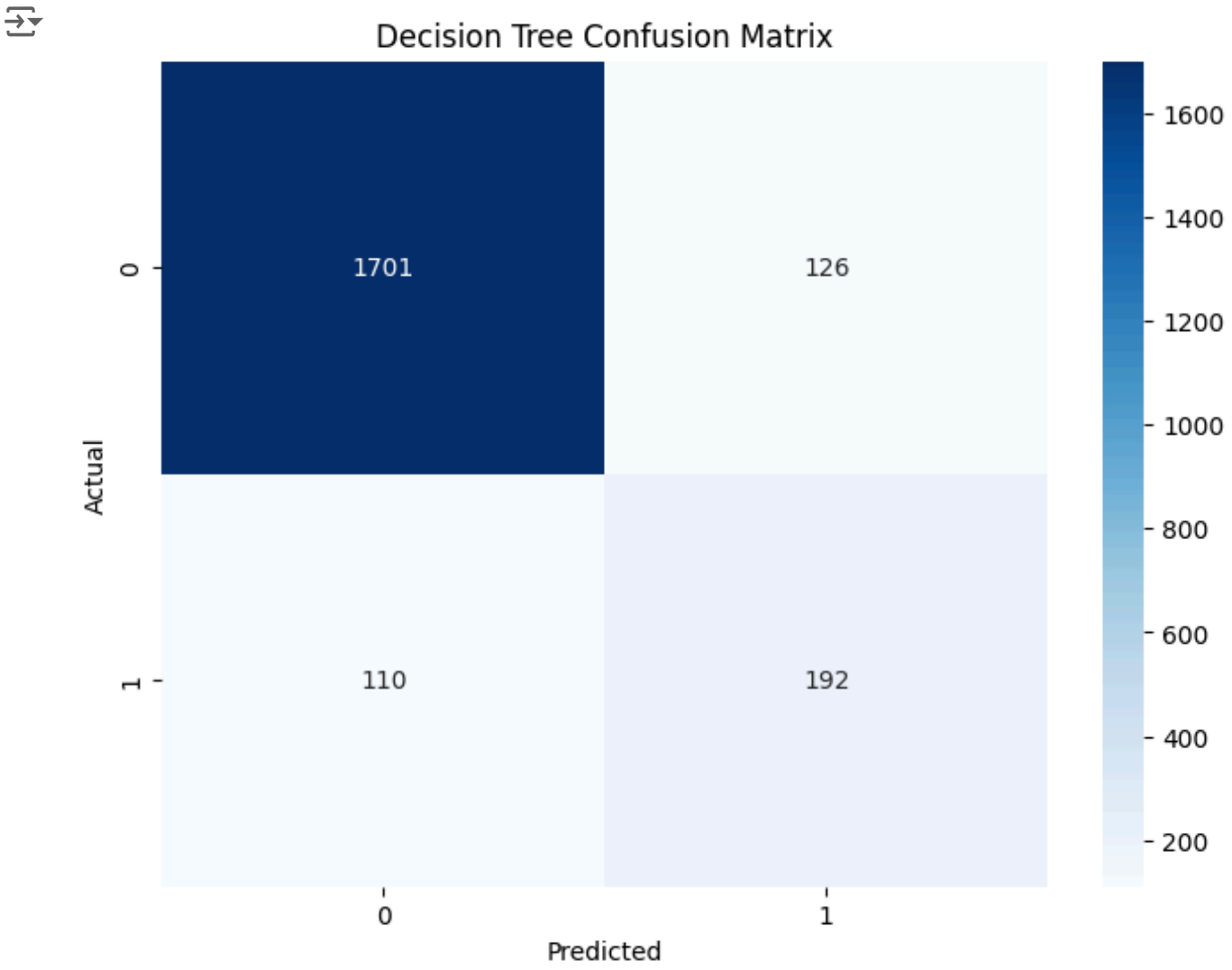
```
No artists with labels found to put in legend.  Note that artists whose label start w
```




```
#Confusion Matrix
from sklearn.metrics import confusion_matrix

cm=confusion_matrix(y_test,y_pred)
print("Decision Tree Confusion Matrix: \n",cm)
```

```
Decision Tree Confusion Matrix:
 [[1701  126]
 [ 110  192]]
```

```
#Plot Confusion Matrix
plt.figure(figsize=(8,6))
sns.heatmap(cm,annot=True,fmt='d',cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Decision Tree Confusion Matrix')
plt.show()
```




```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.94      0.93      0.94      1827
           1       0.60      0.64      0.62       302

    accuracy                           0.89      2129
   macro avg       0.77      0.78      0.78      2129
weighted avg       0.89      0.89      0.89      2129
```

Model 5 Random Forest

```
from sklearn.ensemble import RandomForestClassifier

rf_classifier=RandomForestClassifier()
rf_classifier
```

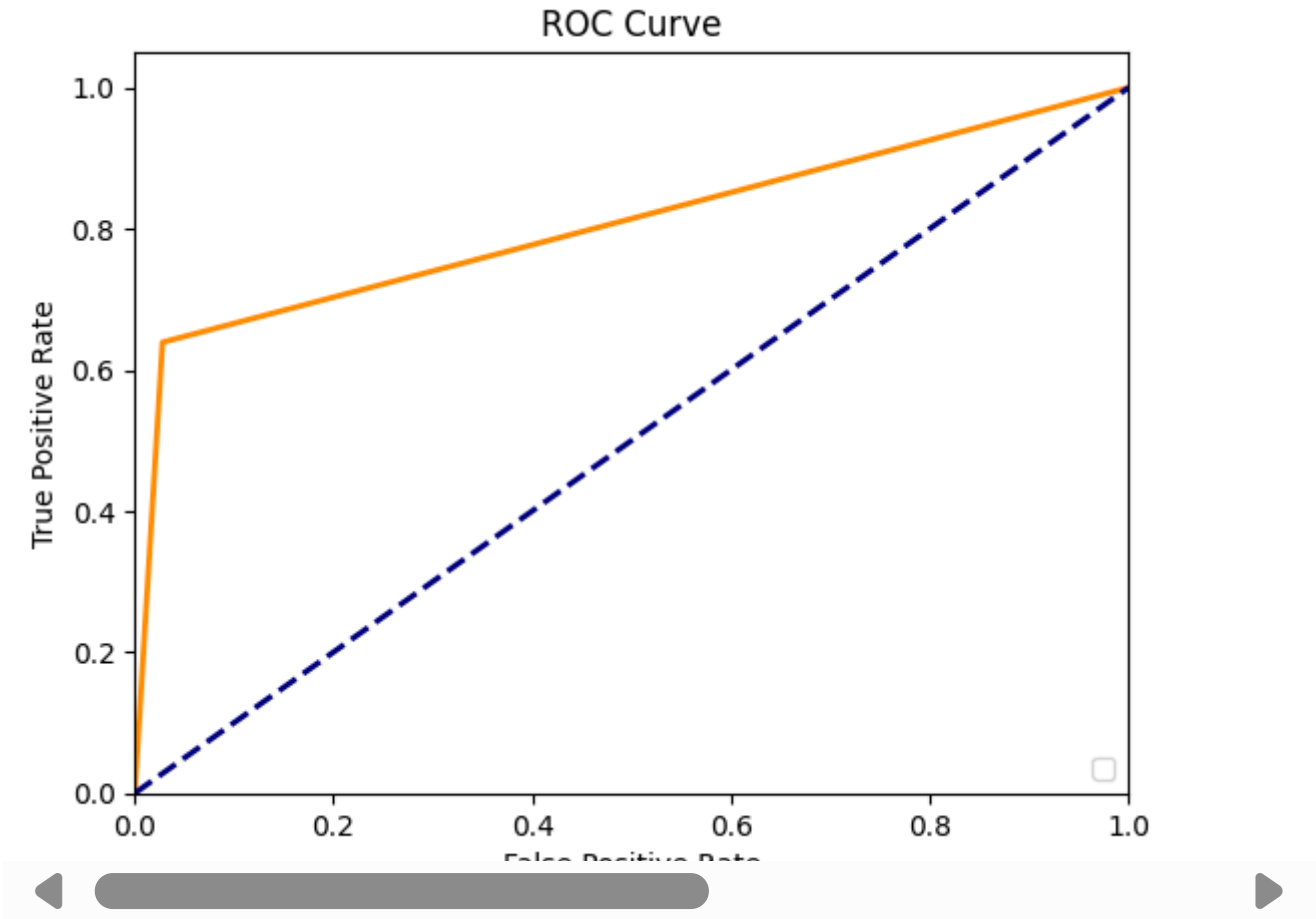
```
RandomForestClassifier
RandomForestClassifier()
```

```
rf_classifier.fit(X_train,y_train)
y_pred=rf_classifier.predict(X_test)
accuracy=accuracy_score(y_test,y_pred)
accuracy
```

```
0.9248473461719117
```

```
#Plotting ROC curve

from sklearn.metrics import roc_curve,auc,accuracy_score

fpr,tpr,threshold=roc_curve(y_test,y_pred)
roc_auc=auc(fpr,tpr)

plt.plot(fpr,tpr,color='darkorange',lw=2)
plt.plot([0,1], [0,1], color='navy',lw=2,linestyle='--')
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc='lower right')
plt.show()
```

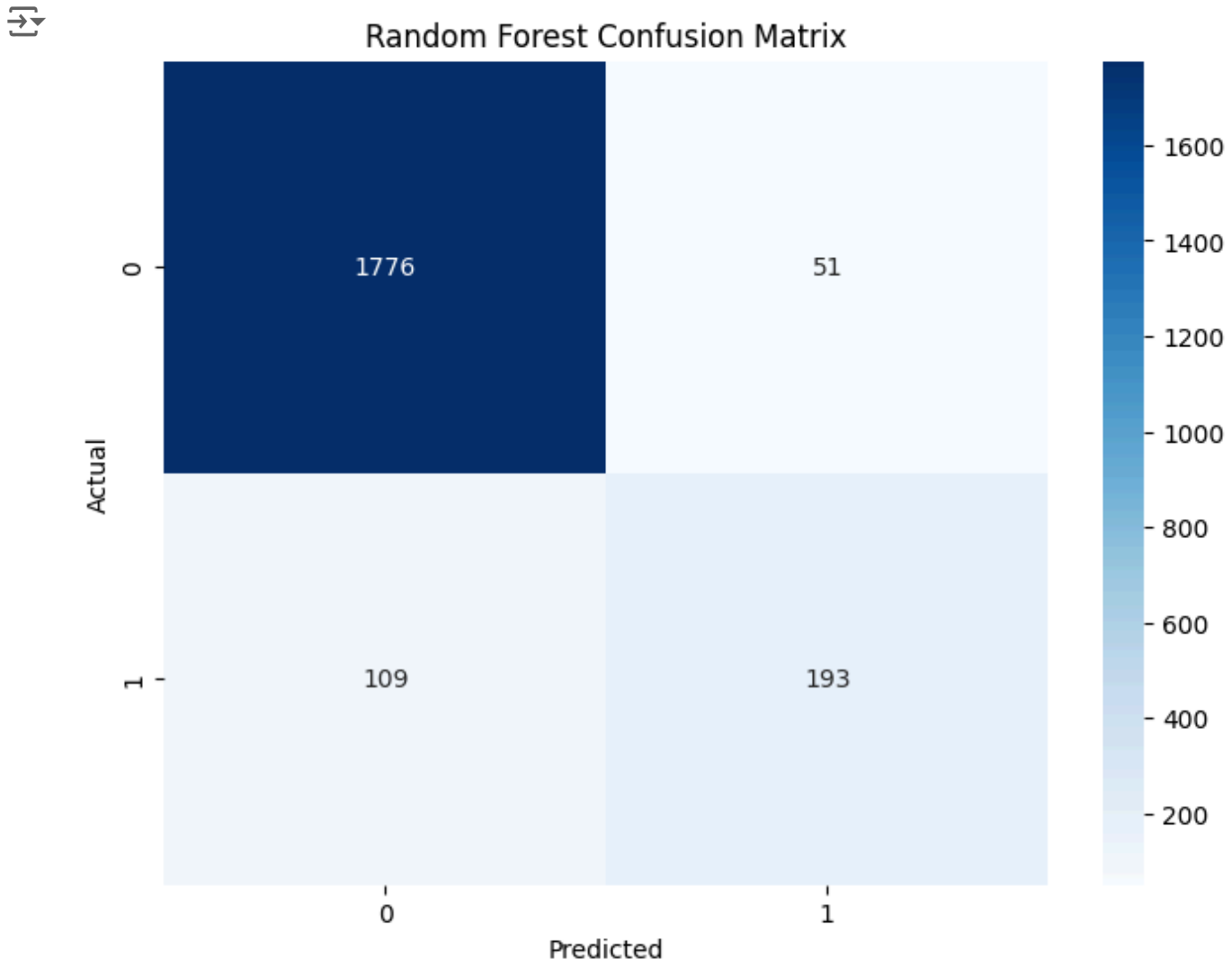
```
No artists with labels found to put in legend.  Note that artists whose label start w
```




```
#Confusion Matrix
from sklearn.metrics import confusion_matrix

cm=confusion_matrix(y_test,y_pred)
print("Random Forest Confusion Matrix: \n",cm)
```

```
Random Forest Confusion Matrix:
 [[1776   51]
 [ 109  193]]
```

```
#Plot Confusion Matrix
plt.figure(figsize=(8,6))
sns.heatmap(cm,annot=True,fmt='d',cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Random Forest Confusion Matrix')
plt.show()
```




```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.94      0.97      0.96      1827
           1       0.79      0.64      0.71       302

    accuracy                           0.92      2129
   macro avg       0.87      0.81      0.83      2129
weighted avg       0.92      0.92      0.92      2129
```

Model6 K Neighbors

```
from sklearn.neighbors import KNeighborsClassifier

knn_classifier=KNeighborsClassifier()
knn_classifier
```

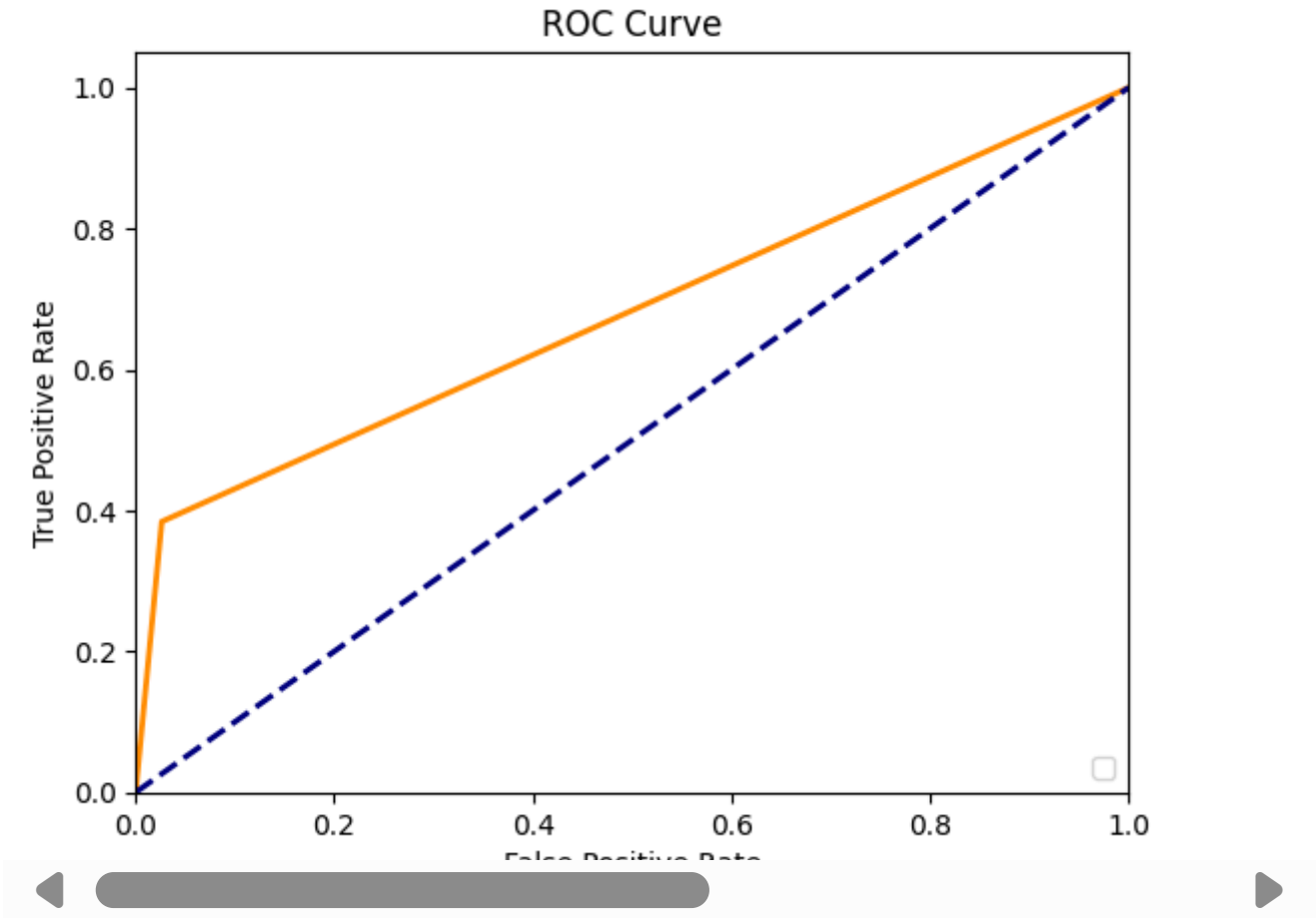
```
KNeighborsClassifier()
```

```
knn_classifier.fit(X_train,y_train)
y_pred=knn_classifier.predict(X_test)
accuracy=accuracy_score(y_test,y_pred)
accuracy
```

```
0.8900892437764208
```

```
#Plotting ROC_curve
from sklearn.metrics import roc_curve,auc,accuracy_score

fpr,tpr,threshold=roc_curve(y_test,y_pred)
roc_auc=auc(fpr,tpr)

plt.figure()
plt.plot(fpr,tpr,color='darkorange',lw=2)
plt.plot([0,1],[0,1],color='navy',lw=2,linestyle='--')
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc='lower right')
plt.show()
```

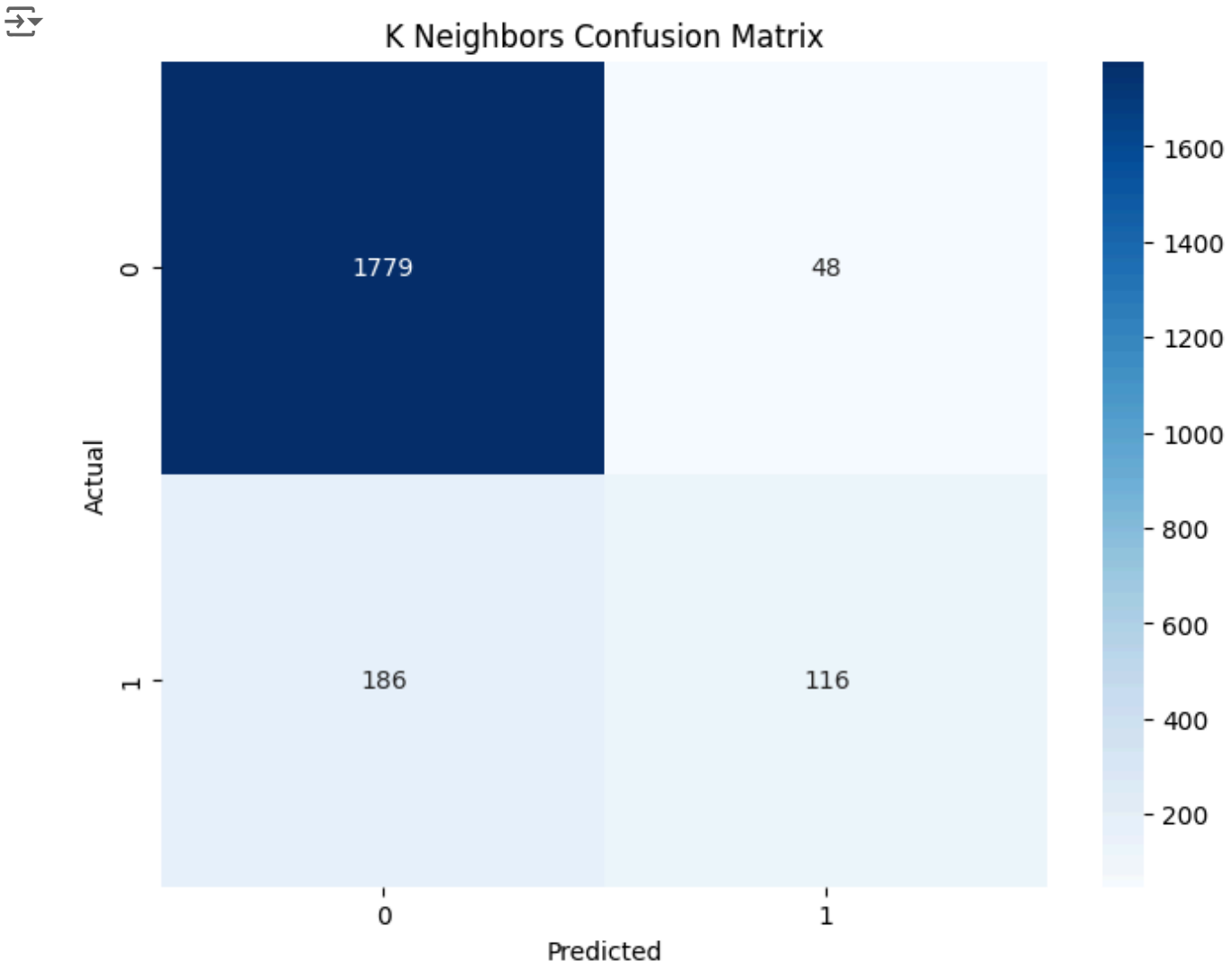
```
No artists with labels found to put in legend.  Note that artists whose label start w
```




```
#Confusion Matrix
from sklearn.metrics import confusion_matrix

cm=confusion_matrix(y_test,y_pred)
print("K Neighbors Confusion Matrix: \n",cm)
```

```
K Neighbors Confusion Matrix:
 [[1779   48]
 [ 186  116]]
```

```
#Plotting Confusion Matrix

plt.figure(figsize=(8,6))
sns.heatmap(cm,annot=True,fmt='d',cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('K Neighbors Confusion Matrix')
plt.show()
```




```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.91      0.97      0.94      1827
           1       0.71      0.38      0.50       302

    accuracy                           0.89      2129
   macro avg       0.81      0.68      0.72      2129
weighted avg       0.88      0.89      0.88      2129
```

Model 7 PLA Perceptron Learning Algorithm

```
from sklearn.linear_model import Perceptron

pla_classifier=Perceptron(max_iter=1000,tol=1e-3,random_state=42)
pla_classifier
```

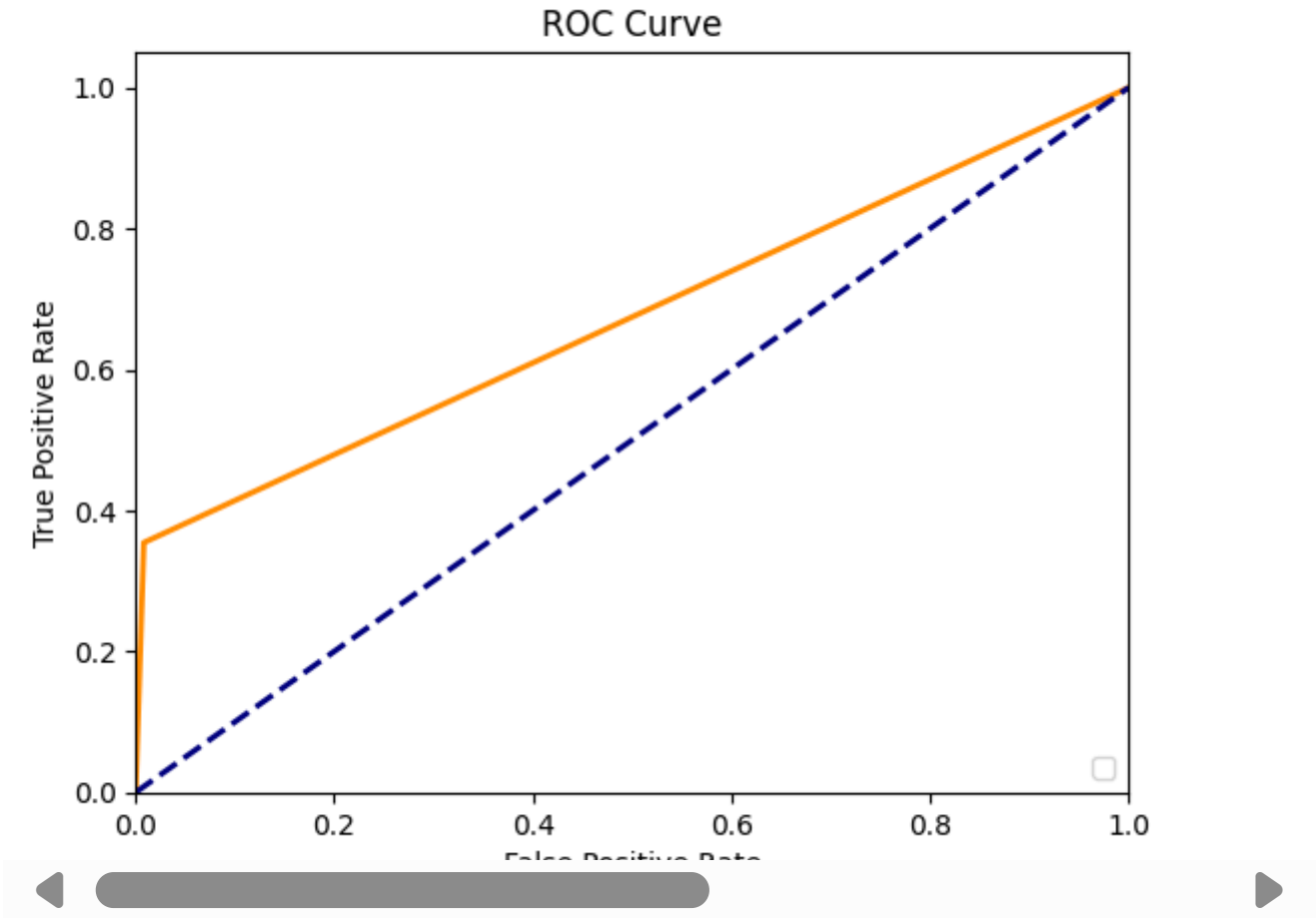
```
Perceptron
Perceptron(random_state=42)
```

```
pla_classifier.fit(X_train,y_train)
y_pred=pla_classifier.predict(X_test)
accuracy=accuracy_score(y_test,y_pred)
accuracy
```

```
0.9013621418506341
```

```
#Plotting ROC_curve
from sklearn.metrics import roc_curve,auc,accuracy_score

fpr,tpr,threshold=roc_curve(y_test,y_pred)
roc_auc=auc(fpr,tpr)

plt.figure()
plt.plot(fpr,tpr,color='darkorange',lw=2)
plt.plot([0,1],[0,1],color='navy',lw=2,linestyle='--')
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc='lower right')
plt.show()
```

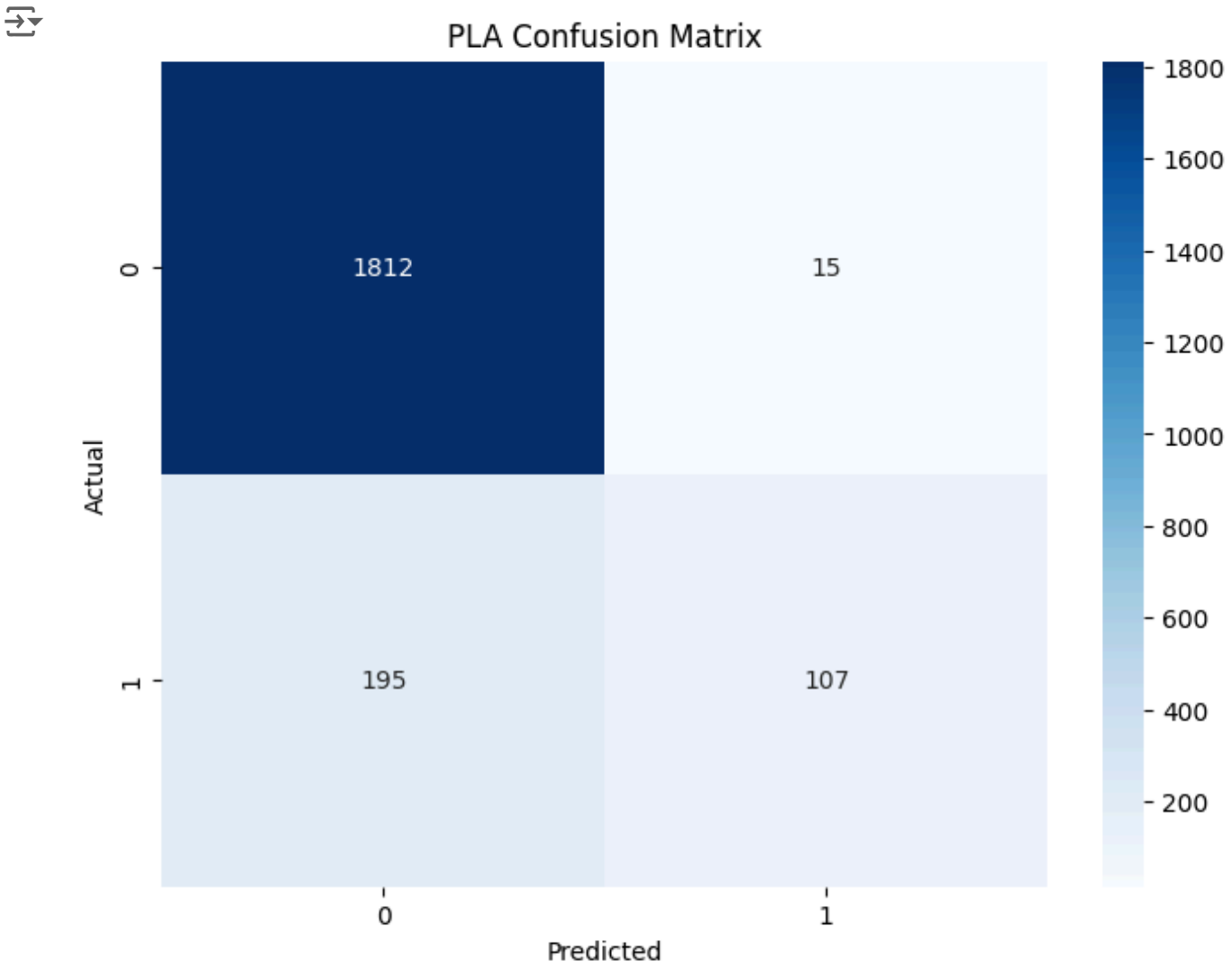
```
No artists with labels found to put in legend.  Note that artists whose label start w
```




```
#Confusion Matrix
from sklearn.metrics import confusion_matrix

cm=confusion_matrix(y_test,y_pred)
print("PLA Matrix: \n",cm)
```

```
PLA Matrix:
 [[1812   15]
 [ 195  107]]
```

```
#Plotting Confusion Matrix

plt.figure(figsize=(8,6))
sns.heatmap(cm,annot=True,fmt='d',cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('PLA Confusion Matrix')
plt.show()
```




```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.90      0.99      0.95      1827
           1       0.88      0.35      0.50       302

    accuracy                           0.90      2129
   macro avg       0.89      0.67      0.72      2129
weighted avg       0.90      0.90      0.88      2129
```

Model 8 MLP MultiLayer Perceptron

```
from sklearn.neural_network import MLPClassifier

mlp_classifier=MLPClassifier(hidden_layer_sizes=(100,),max_iter=300,random_state=42)
mlp_classifier
```

MLPClassifier

```
mlp_classifier.fit(X_train,y_train)
y_pred=mlp_classifier.predict(X_test)
accuracy=accuracy_score(y_test,y_pred)
accuracy
```

```
0.9173320807891029
```

```
#Plotting ROC_curve
from sklearn.metrics import roc_curve,auc,accuracy_score

fpr,tpr,threshold=roc_curve(y_test,y_pred)
roc_auc=auc(fpr,tpr)

plt.figure()
```