UCS2612 Machine Learning Laboratory

A5 – K-Nearest Neighbor Algorithm

Name: C B Ananya
Reg No: 3122215001010

Question:

Download the Online Shoppers Purchasing Intention Dataset dataset from the link given below:
https://archive.ics.uci.edu/dataset/468/online+shoppers+purchasing+intention+dataset

The dataset consists of 12,330 sessions, 84.5% (10,422) were negative class samples that did not end with shopping, and the rest (1908) were positive class samples ending with shopping.

Develop a python program to predict the Online Shoppers Purchasing Intention using K-Nearest Neighbor algorithm. Visualize the features from the dataset and interpret the results obtained by the model using Matplotlib library. [CO1, K3]
Use the following steps to do implementation:

1. Loading the dataset.
2. Pre-Processing the data (Handling missing values, Encoding, Normalization, Standardization).
3. Exploratory Data Analysis.
4. Feature Engineering techniques.
5. Split the data into training, testing and validation sets.
6. Provide test data.
7. Measure the performance of the model.
8. Represent the results in terms of ROC curves using graphs.

GitHub Main Branch Link:
https://github.com/CB-Ananya/ML-Lab

Importing necessary libraries and functions

In [ ]:
```
!jupyter nbconvert --to html /content/drive/MyDrive/MLLab/A5-CBAnanya-RegNo10.ipynb
```

In [2]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [1]:
```
import pandas as pd
import numpy as np
from sklearn import preprocessing
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,roc_auc_score,pairwise_distances, roc_curve, auc
from sklearn.model_selection import train_test_split
```

Loading the dataset

In [3]:
```
df=pd.read_csv('/content/drive/MyDrive/MLLab/online_shoppers_intention.csv')
df
```

Out[3]:

	Administrative	Administrative_Duration	Informational	Informational_Duration	ProductRelated	ProductRelated_Duration	BounceRa
0	0	0.0	0	0.0	1	0.000000	0.200
1	0	0.0	0	0.0	2	64.000000	0.000
2	0	0.0	0	0.0	1	0.000000	0.200
3	0	0.0	0	0.0	2	2.666667	0.050
4	0	0.0	0	0.0	10	627.500000	0.020
...	...	...	...	...	...	...	
12325	3	145.0	0	0.0	53	1783.791667	0.007
12326	0	0.0	0	0.0	5	465.750000	0.000
12327	0	0.0	0	0.0	6	184.250000	0.083
12328	4	75.0	0	0.0	15	346.000000	0.000
12329	0	0.0	0	0.0	3	21.250000	0.000

12330 rows × 18 columns

Preprocessing and Exploratory Data Analysis

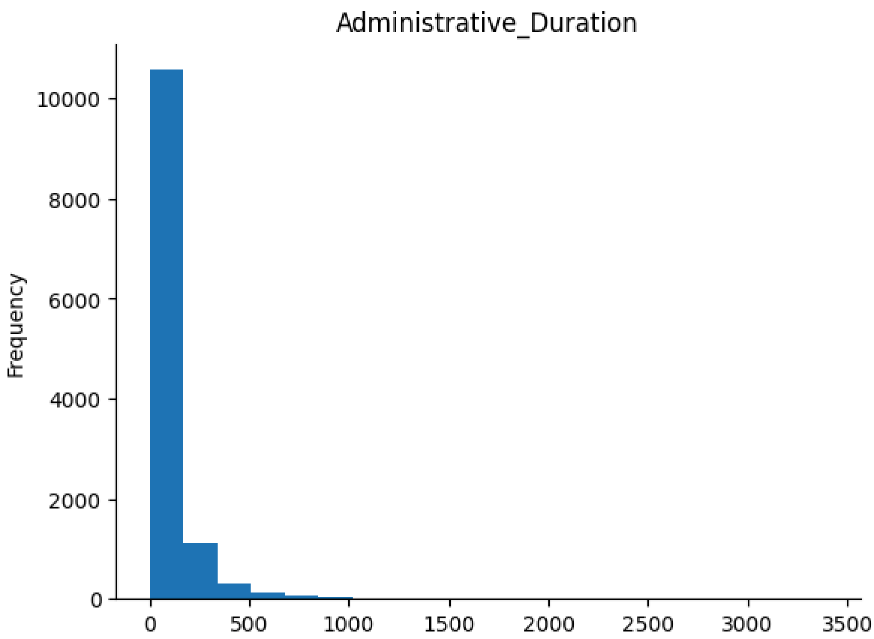
In [4]:
```
# Number of columns and data type of each
df.info()
```

```
<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
```

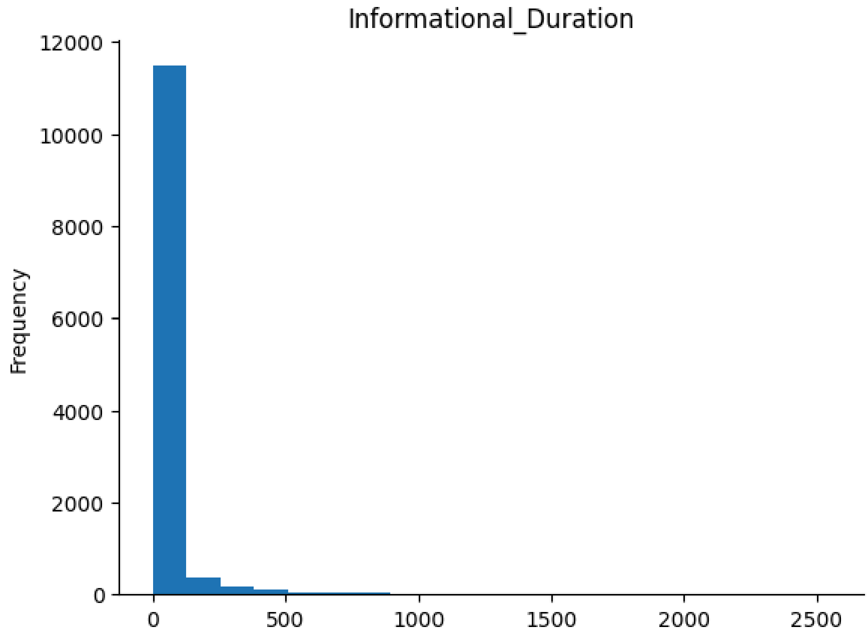
In [5]:
```
# Checking for missing values
df.isnull().sum()
```

Out[5]:
```
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
```

In [21]:
```
from matplotlib import pyplot as plt
df['Administrative_Duration'].plot(kind='hist', bins=20, title='Administrative_Duration')
plt.gca().spines[['top', 'right',]].set_visible(False)
```



In [22]:
```
from matplotlib import pyplot as plt
df['Informational_Duration'].plot(kind='hist', bins=20, title='Informational_Duration')
plt.gca().spines[['top', 'right',]].set_visible(False)
```

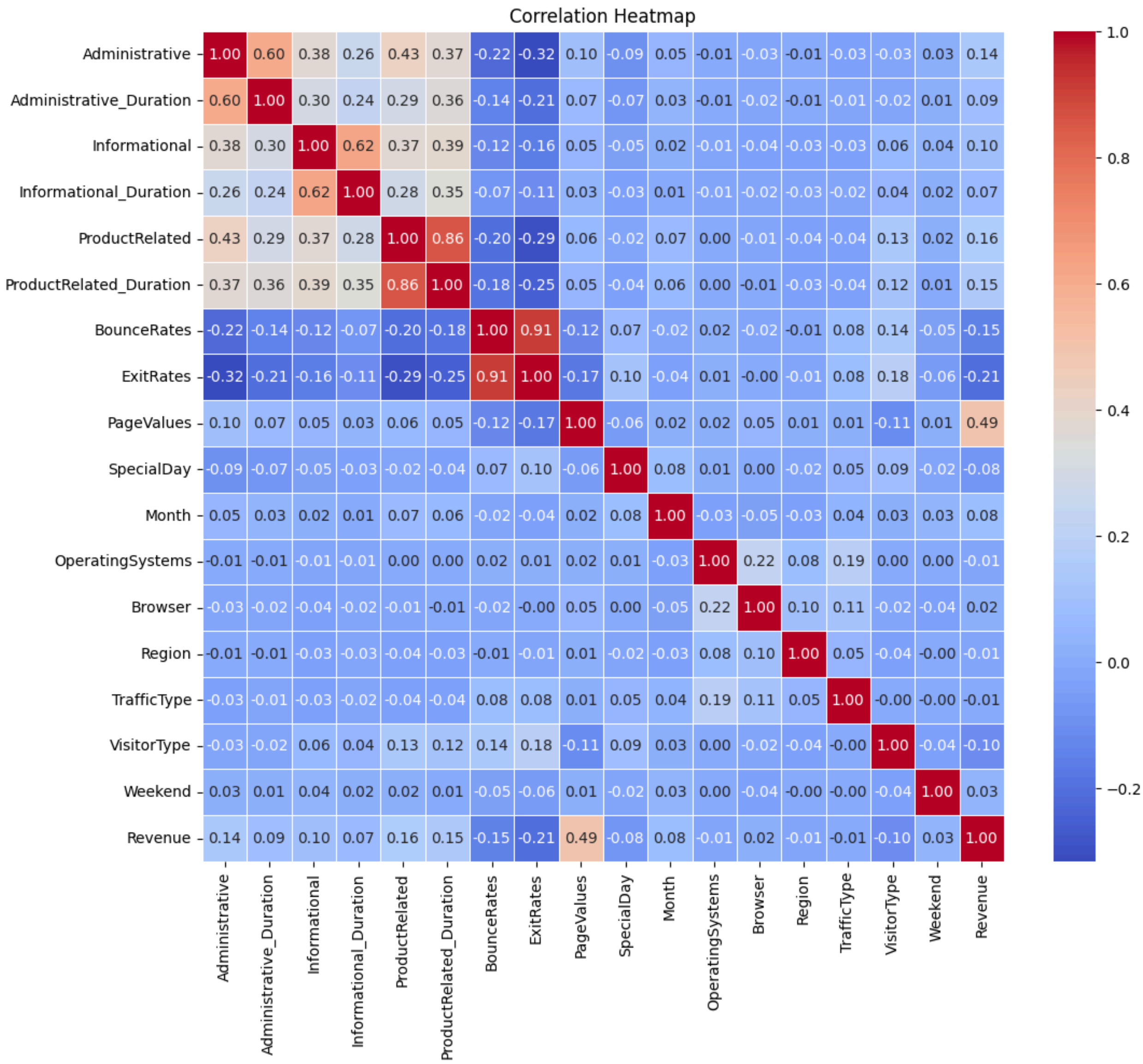


In [6]:
```
# Encode categorical columns (type object)

for col in df.select_dtypes(include=['object','bool']).columns:

    print(f"{col}: {df[col].unique()}")
    df[col] = df[col].astype(str)

    # Initialize a LabelEncoder object
    label_encoder = preprocessing.LabelEncoder()

    # Fit the encoder to the unique values in the column
    label_encoder.fit(df[col].unique())

    # Transform the column using the encoder
    df[col] = label_encoder.transform(df[col])

    print(f"{col}: {df[col].unique()}")
```

```
Month: ['Feb' 'Mar' 'May' 'Oct' 'June' 'Jul' 'Aug' 'Nov' 'Sep' 'Dec']
Month: [2 5 6 8 4 3 0 7 9 1]
VisitorType: ['Returning_Visitor' 'New_Visitor' 'Other']
VisitorType: [2 0 1]
Weekend: [False  True]
Weekend: [0 1]
Revenue: [False  True]
Revenue: [0 1]
```

In [7]:
```
# Correlation pairwise-columns to build heat-map
correlation_matrix = df.corr()
```

In [8]:
```
plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=.5)
plt.title('Correlation Heatmap')
plt.show()
# PageValues attribute seemingly is more related to Revenue than other attributes
```



Train Test Split

In [9]:
```
# Features and Target Variable
x=df.drop('Revenue',axis=1)
y=df['Revenue']
```

In [ ]:
```

```

In [10]:
```
# 80-20
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42,stratify=y)
```

Scikit-Learn KNeighborsClassifier

In [11]:
```
# Using different distance metrics (euclidean, manhattan and minkowski), and k=3 and comparing accuracies
def KNN(k, distance):
    # Initialize the KNN classifier
    knn_classifier_sk = KNeighborsClassifier(n_neighbors=k, metric=distance)

    # Train the model
    knn_classifier_sk.fit(x_train, y_train)
    # Predict on the test set
    y_pred = knn_classifier_sk.predict(x_test)

    # Evaluate the model
    accuracy = accuracy_score(y_test, y_pred)
    return accuracy, knn_classifier_sk

print("Accuracy:")
accuracy,knn_classifier_sk = KNN(3,"euclidean")
print("Using Euclidean Distance: ",accuracy)
print("Using Manhattan Distance: ",KNN(3, "manhattan")[0])
print("Using Mikowski Distance: ",KNN(3, "minkowski")[0])
```

```
# With manhattan distance metric, slightly better accuracy is obtained
```

```
Accuracy:
Using Euclidean Distance:  0.8398215733982157
Using Manhattan Distance:  0.8438767234387672
Using Mikowski Distance:  0.8398215733982157
```

In [12]:
```
# Using different values of k with the euclidean metric
print("k=5: Accuracy: ", KNN(5, "euclidean")[0])
print("k=10: Accuracy: ", KNN(10, "euclidean")[0])
print("k=15: Accuracy: ", KNN(15, "euclidean")[0])
print("k=20: Accuracy: ", KNN(20, "euclidean")[0])

# Accuracy seems to increase as the value of k increases
```

```
k=5: Accuracy:  0.8499594484995945
k=10: Accuracy:  0.8572587185725872
k=15: Accuracy:  0.8596918085969181
k=20: Accuracy:  0.8600973236009732
```

User Defined KNN Classifier

In [13]:
```
# Using distance metric - euclidean and k=3
class KNNClassifier:
    def __init__(self, k=3, distance='euclidean'):
        self.k = k
        self.distance = distance

    def fit(self, X, y):
        self.X_train = X
        self.y_train = y

    def predict(self, X):
        distances = pairwise_distances(X, self.X_train, metric=self.distance)
        y_pred = [self._predict(dist) for dist in distances]
        return np.array(y_pred)

    def _predict(self, distances):
        k_indices = np.argsort(distances)[:self.k]
        k_nearest_labels = [self.y_train[i] for i in k_indices]
        most_common = max(set(k_nearest_labels), key=k_nearest_labels.count)
        return most_common
    # for plotting ROC Curves
    def predict_proba(self, X):
        distances = pairwise_distances(X, self.X_train, metric=self.distance)
        y_probs = []
        for dist in distances:
            k_indices = np.argsort(dist)[:self.k]
            k_nearest_labels = [self.y_train[i] for i in k_indices]
            class_probs = [k_nearest_labels.count(c) / self.k for c in np.unique(self.y_train)]
            y_probs.append(class_probs)
        return np.array(y_probs)

def KNN_user(k=3, distance="euclidean"):
    knn_classifier = KNNClassifier(k=3)
    knn_classifier.fit(x_train.values, y_train.values)
    y_pred = knn_classifier.predict(x_test.values)
    accuracy = np.mean(y_pred == y_test.values)
    return accuracy, knn_classifier

accuracy, knn_classifier = KNN_user(k=3)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.8398215733982157
```

In [14]:
```
# Using different values of k with the euclidean metric
print("k=5: Accuracy: ", KNN_user(5, "euclidean")[0])
print("k=10: Accuracy: ", KNN_user(10, "euclidean")[0])
print("k=15: Accuracy: ", KNN_user(15, "euclidean")[0])
print("k=20: Accuracy: ", KNN_user(20, "euclidean")[0])

## Accuracy stays the same
```

```
k=5: Accuracy:  0.8398215733982157
k=10: Accuracy:  0.8398215733982157
k=15: Accuracy:  0.8398215733982157
k=20: Accuracy:  0.8398215733982157
```

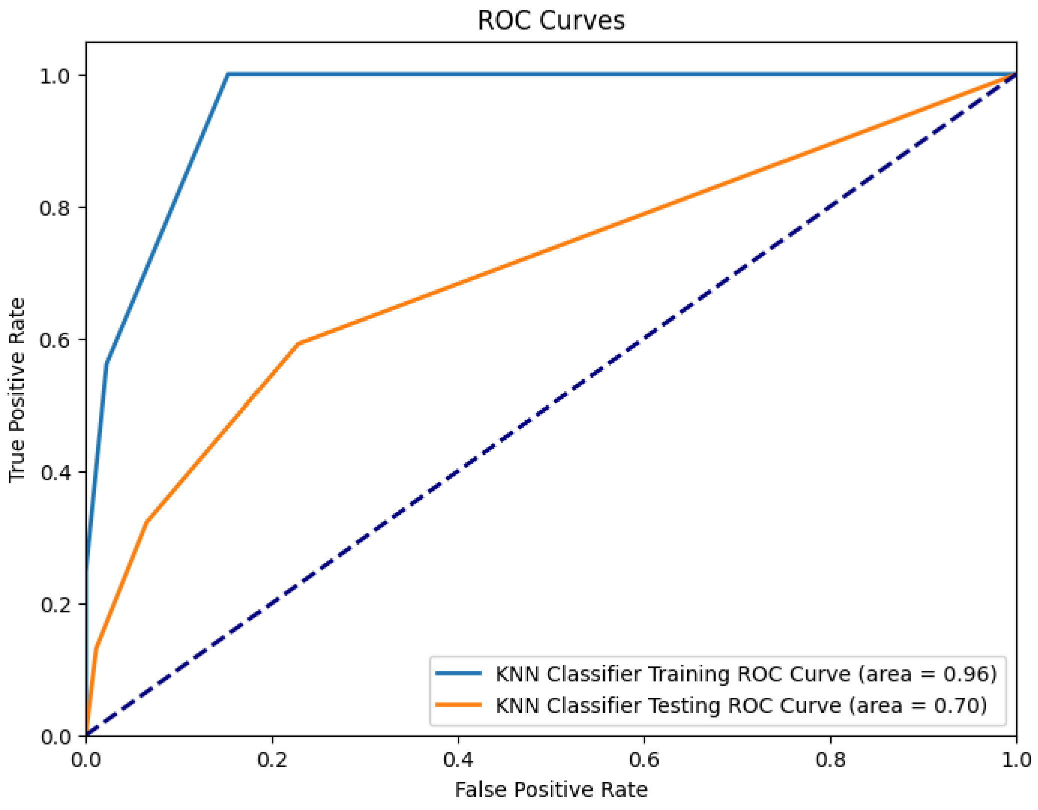
ROC Curves

In [15]:
```
from sklearn.metrics import roc_curve, auc
# Function to plot ROC curve
def plot_roc_curve(y_true, y_prob, title):
```

```
    fpr, tpr, _ = roc_curve(y_true, y_prob)
    roc_auc = auc(fpr, tpr)

    plt.plot(fpr, tpr, lw=2, label= title+' (area = %0.2f)' % roc_auc)

# Function to plot ROC curves for training and testing data on the same graph
def plot_roc_curves(classifier, x_train, y_train, x_test, y_test, title):
    classifier.fit(x_train, y_train)
    y_train_prob = classifier.predict_proba(x_train)[:, 1]
    y_test_prob = classifier.predict_proba(x_test)[:, 1]

    plt.figure(figsize=(8, 6))
    plot_roc_curve(y_train, y_train_prob, f'{title} Training ROC Curve')
    plot_roc_curve(y_test, y_test_prob, f'{title} Testing ROC Curve')

    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 Curves')
    plt.legend(loc="lower right")
    plt.show()
```

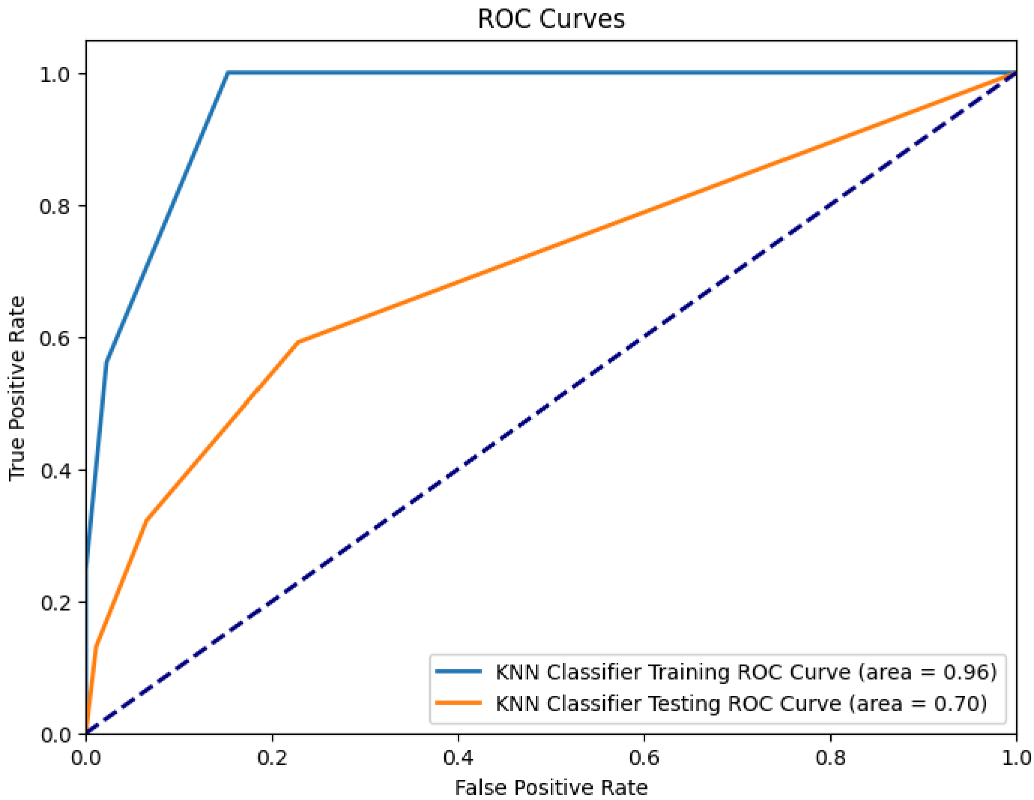
In [ ]:

```
# Plotting ROC Curves for sklearn KNeighborsClassifier Model
plot_roc_curves(knn_classifier_sk, x_train.values, y_train.values, x_test.values, y_test.values, 'KNN Classifier')
```



In [16]:

```
# Plotting ROC Curves for User-defined KNNClassifier
plot_roc_curves(knn_classifier, x_train.values, y_train.values, x_test.values, y_test.values, 'KNN Classifier')
```



Inferences

- For the dataset chosen, the accuracy score seems to increase with the value of k, with respect to scikit-learn's KNeighbors Classifier Model. Scikit-learn's KNeighborsClassifier is highly optimized and includes additional features like efficient data structures (e.g., KD-trees, Ball-trees) to speed up nearest neighbor searches. These optimizations might not only affect performance in terms of computation time but can also impact accuracy.
- Since the user-defined KNNClassifier includes a basic implementation of KNN with none of the optimizations that scikit-learn model offers, the accuracy does not increase with the value of k.
- For the given dataset, using manhattan distance as the distance metric seems to offer slightly better performance.

Learning Outcomes

- Implementing the KNN model from scratch for better understanding.
- Evaluating KNN model's performance using various metrics and visualization og ROC Curve.
- Learning how the model's performance is impacted by using different parameters (k and distance metric)
- Comparing scikit-learn and user defined KNN models, and understanding that the library model is highly optimized with the use of appropriate data structures

In [ ]: