UCS2612 machine learning lab

Assignment on K Means Clustering with User Defined Functions

Name : Mega V Roll No : 3122 21 5001 051

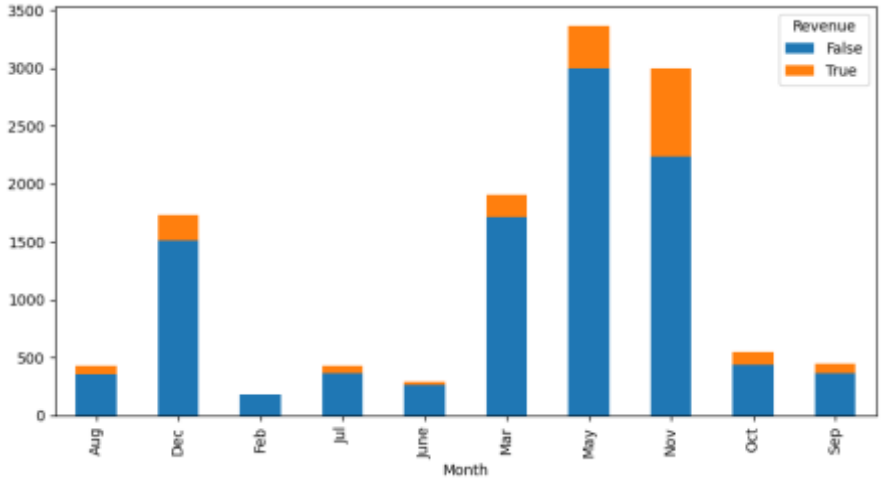
Write the python code from scratch to implement K Means Clustering Algorithm without using Scikit-learn library or built in functions.

Code and Output

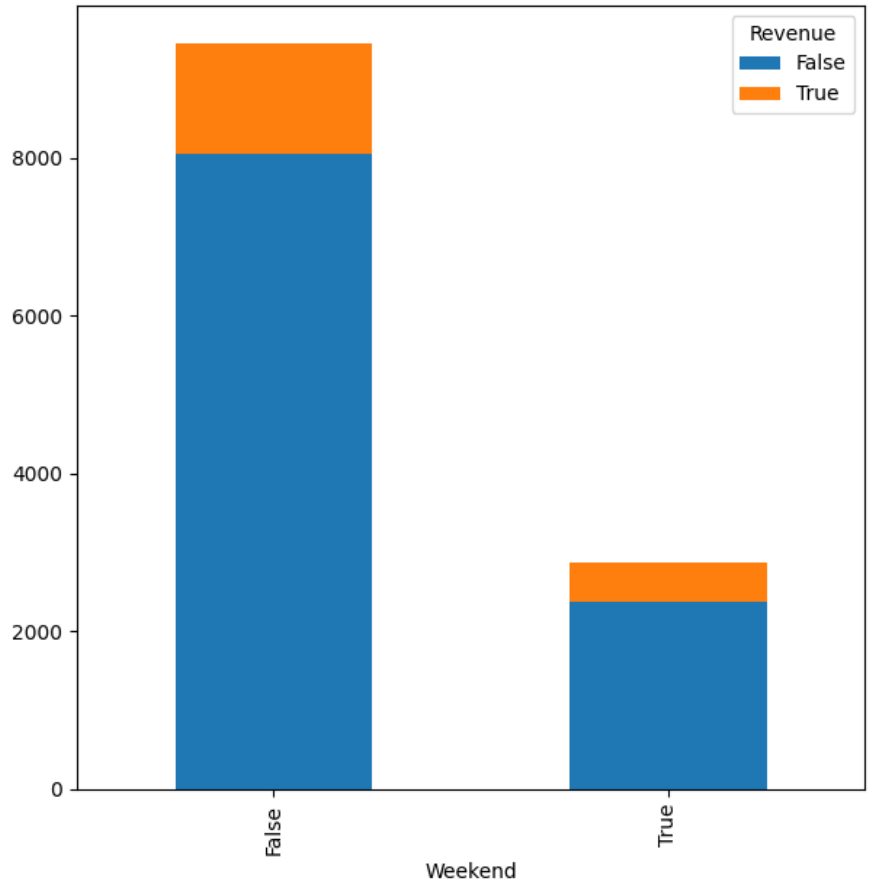
```
Loading the dataset
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
df = pd.read_csv("online_shoppers_intention.csv")
df.head()
```

```
<bound method NDFrame.head of       Administrative  Administrative_Duration  Informational  \
0                  0                     0.0              0
1                  0                     0.0              0
2                  0                     0.0              0
3                  0                     0.0              0
4                  0                     0.0              0
...              ...                     ...            ...
12325              3                   145.0              0
12326              0                     0.0              0
12327              0                     0.0              0
12328              4                    75.0              0
12329              0                     0.0              0

       Informational_Duration  ProductRelated  ProductRelated_Duration  \
0                         0.0               1                 0.000000
1                         0.0               2                64.000000
2                         0.0               1                 0.000000
3                         0.0               2                 2.666667
4                         0.0              10               627.500000
...                       ...             ...                      ...
12325                     0.0              53              1783.791667
12326                     0.0               5               465.750000
12327                     0.0               6               184.250000
12328                     0.0              15               346.000000
12329                     0.0               3                21.250000
...
12327       2       1          13  Returning_Visitor     True    False
12328       2       3          11  Returning_Visitor    False    False
12329       2       1           2        New_Visitor     True    False
```

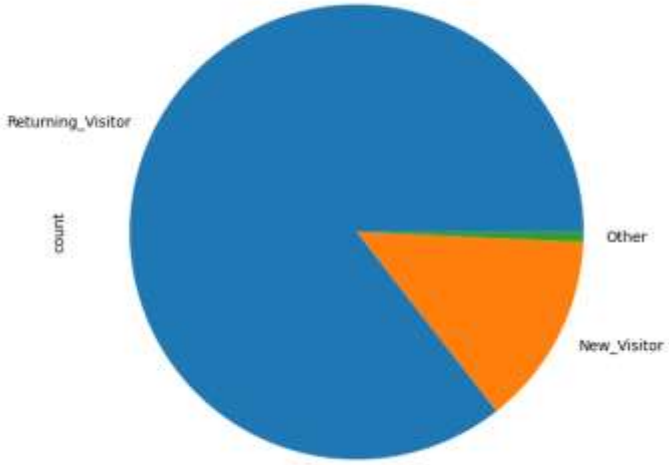
Data Visualization



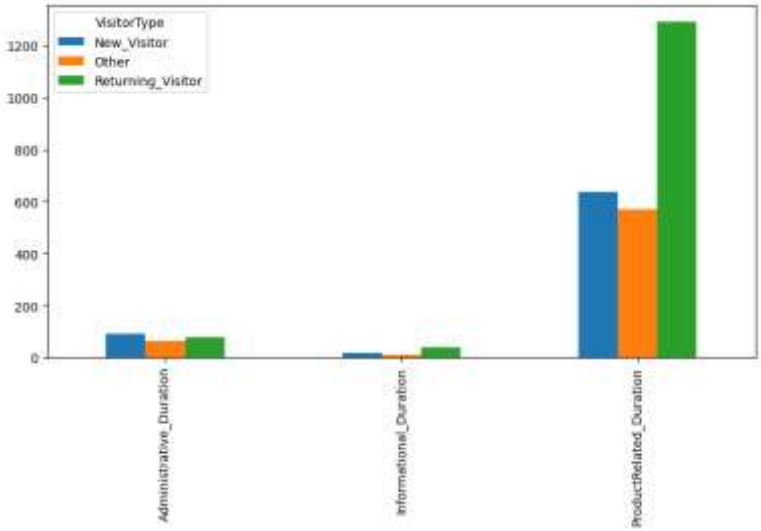

```
df.groupby('Weekend')['Revenue'].value_counts().unstack('Revenue').plot(kind='
bar', stacked=True, figsize=(7, 7))
```



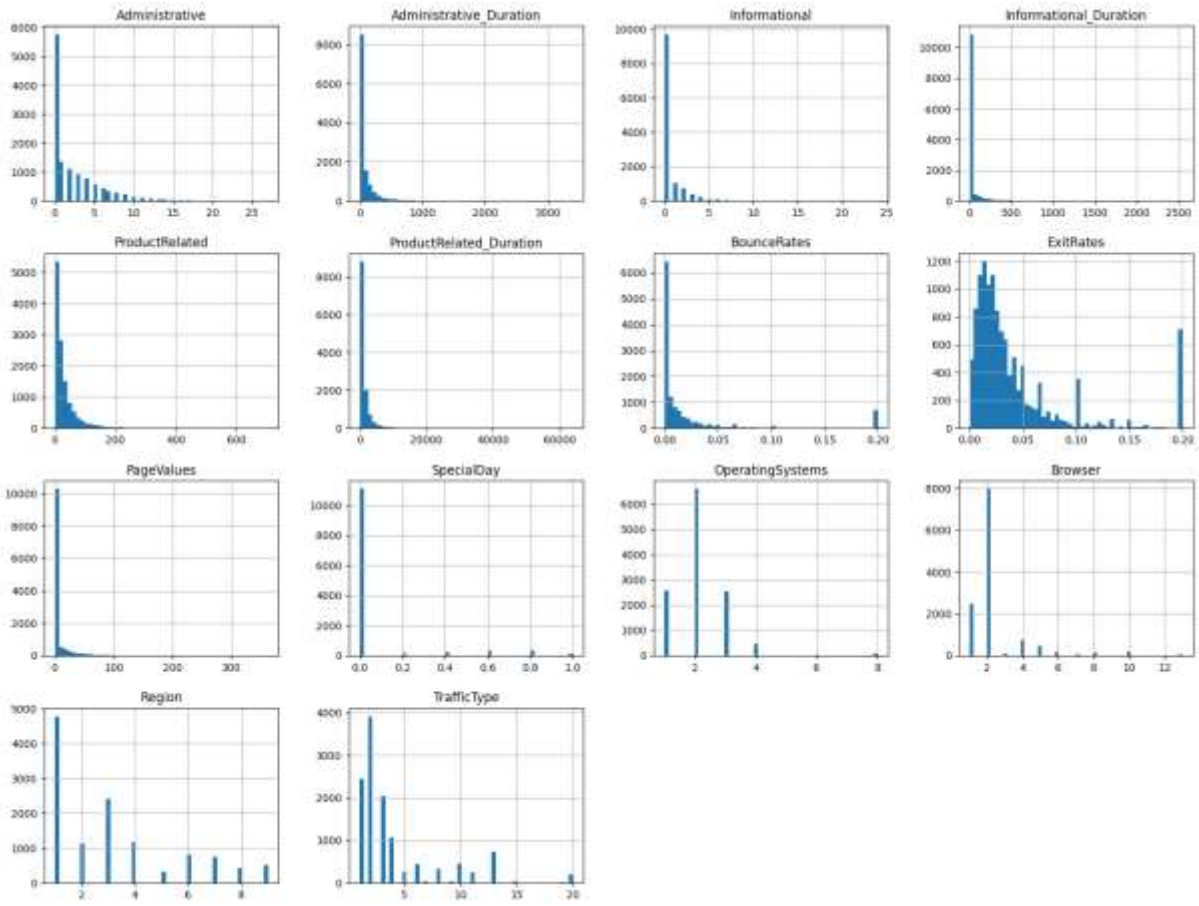
```
df['VisitorType'].value_counts().plot.pie(y='VisitorType', figsize=(7, 7))
```




```
df_pvt=df[['Administrative_Duration','Informational_Duration','ProductRelated_Duration','VisitorType']]
pd.pivot_table(df_pvt,
values=['Administrative_Duration','Informational_Duration','ProductRelated_Duration'],columns=['VisitorType'], aggfunc='mean').plot(kind='bar', figsize=(10, 5))
```



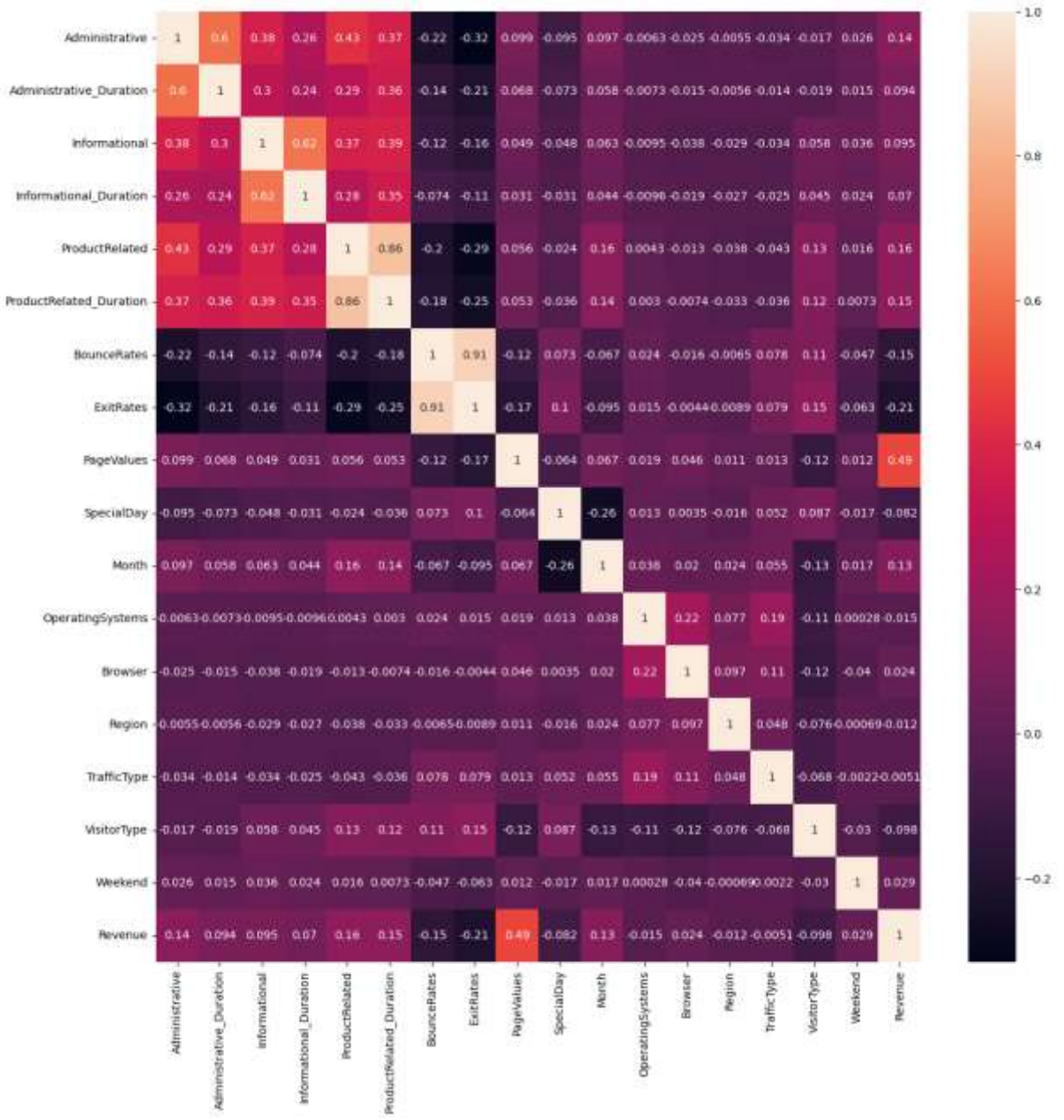
```
df.hist(bins=50, figsize=(20,15))
plt.show()
```



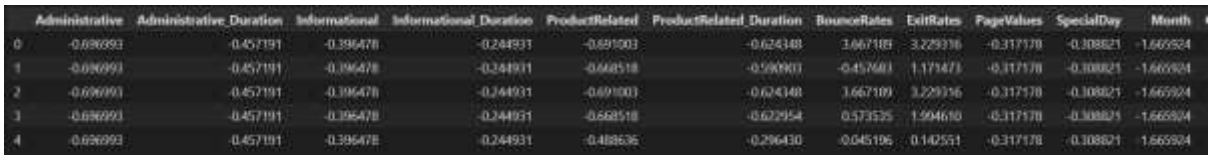
Data Cleaning and Standardization

```
Month={'Feb':2, 'Mar':3, 'May':5, 'Oct':10, 'June':6, 'Jul':7, 'Aug':8,
'Nov':11, 'Sep':9,'Dec':12}
df['Month']=df['Month'].map(Month)

VisitorType={'Returning_Visitor':3, 'New_Visitor':2, 'Other':1}
df['VisitorType']=df['VisitorType'].map(VisitorType)
d={True:1,False:0}
df['Weekend']=df['Weekend'].map(d)
df['Revenue']=df['Revenue'].map(d)
```

```
Var_Corr = df.corr()
fig, ax = plt.subplots(figsize=(15,15))
sns.heatmap(Var_Corr, xticklabels=Var_Corr.columns,
yticklabels=Var_Corr.columns, annot=True)
```



```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

scaler.fit(df.drop('Revenue', axis = 1))
scaled_features = scaler.transform(df.drop('Revenue', axis = 1))

df_feat = pd.DataFrame(scaled_features, columns = df.columns[:-1])
df_feat.head()
```



	Administrative	Administrative_Duration	Informational	Informational_Duration	ProductRelated	ProductRelated_Duration	BounceRates	ExitRates	PageValues	SpecialDay	Month
0	-0.696993	-0.457191	-0.396478	-0.244931	-0.691003	-0.624348	3.667189	3.229316	-0.317178	-0.308821	-1.665924
1	-0.696993	-0.457191	-0.396478	-0.244931	-0.668518	-0.580903	-0.457681	1.171473	-0.317178	-0.308821	-1.665924
2	-0.696993	-0.457191	-0.396478	-0.244931	-0.691003	-0.624348	1.667189	3.229316	-0.317178	-0.308821	-1.665924
3	-0.696993	-0.457191	-0.396478	-0.244931	-0.668518	-0.622954	0.573525	1.994610	-0.317178	-0.308821	-1.665924
4	-0.696993	-0.457191	-0.396478	-0.244931	-0.488636	-0.296430	-0.045196	0.142551	-0.317178	-0.308821	-1.665924

```
df.info()
```

```
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  int64  
 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  int64  
 16  Weekend                  12330 non-null  int64  
 17  Revenue                  12330 non-null  int64  
dtypes: float64(7), int64(11)
```

```
# K Nearest Neighbor - User Defined
```

```
import numpy as np

def knn_euclidean(k, train, test, y_train):
    predictions = []
    for i in range(len(test)):
        distances = []
        for j in range(len(train)):
            dist = np.sqrt(np.sum((test[i] - train[j])**2))
            distances.append((dist, j))
        distances.sort()
        neighbors = distances[:k]
        labels = [y_train[index] for dist, index in neighbors]
        predictions.append(max(set(labels), key=labels.count))
    return predictions

def accuracy(y_true, y_pred):
    correct = sum(1 for true, pred in zip(y_true, y_pred) if true == pred)
    total = len(y_true)
    return correct / total

y_pred = knn_euclidean(3, X_train, X_test, list(y_train))
acc = accuracy(y_test, y_pred)
print("Accuracy:", acc)
```

```
Accuracy: 0.8656393619897269
```

```
def knn_manhattan(k, train, test, y_train):
    predictions = []
    for i in range(len(test)):
        distances = []
        for j in range(len(train)):
            dist = np.sum(np.abs(test[i] - train[j]))
            distances.append((dist, j))
        distances.sort()
        neighbors = distances[:k]
        labels = [y_train[index] for dist, index in neighbors]
        predictions.append(max(set(labels), key=labels.count))
    return predictions
def accuracy(y_true, y_pred):
    correct = sum(1 for true, pred in zip(y_true, y_pred) if true == pred)
```

```
    total = len(y_true)
    return correct / total

y_pred_manhattan = knn_manhattan(3, X_train, X_test, list(y_train))
acc_manhattan = accuracy(y_test, y_pred_manhattan)
print("Accuracy (Manhattan):", acc_manhattan)
```

```
Accuracy (Manhattan): 0.8656393619897269
```

```
def knn_minkowski(k, train, test, y_train, p=2):
    predictions = []
    for i in range(len(test)):
        distances = []
        for j in range(len(train)):
            dist = np.power(np.sum(np.power(np.abs(test[i] - train[j]), p)),
1/p)
            distances.append((dist, j))
        distances.sort()
        neighbors = distances[:k]
        labels = [y_train[index] for dist, index in neighbors]
        predictions.append(max(set(labels), key=labels.count))
    return predictions

def accuracy(y_true, y_pred):
    correct = sum(1 for true, pred in zip(y_true, y_pred) if true == pred)
    total = len(y_true)
    return correct / total

y_pred_minkowski = knn_minkowski(3, X_train, X_test, list(y_train), p=3)
acc_minkowski = accuracy(y_test, y_pred_minkowski)
print("Accuracy (Minkowski):", acc_minkowski)
```

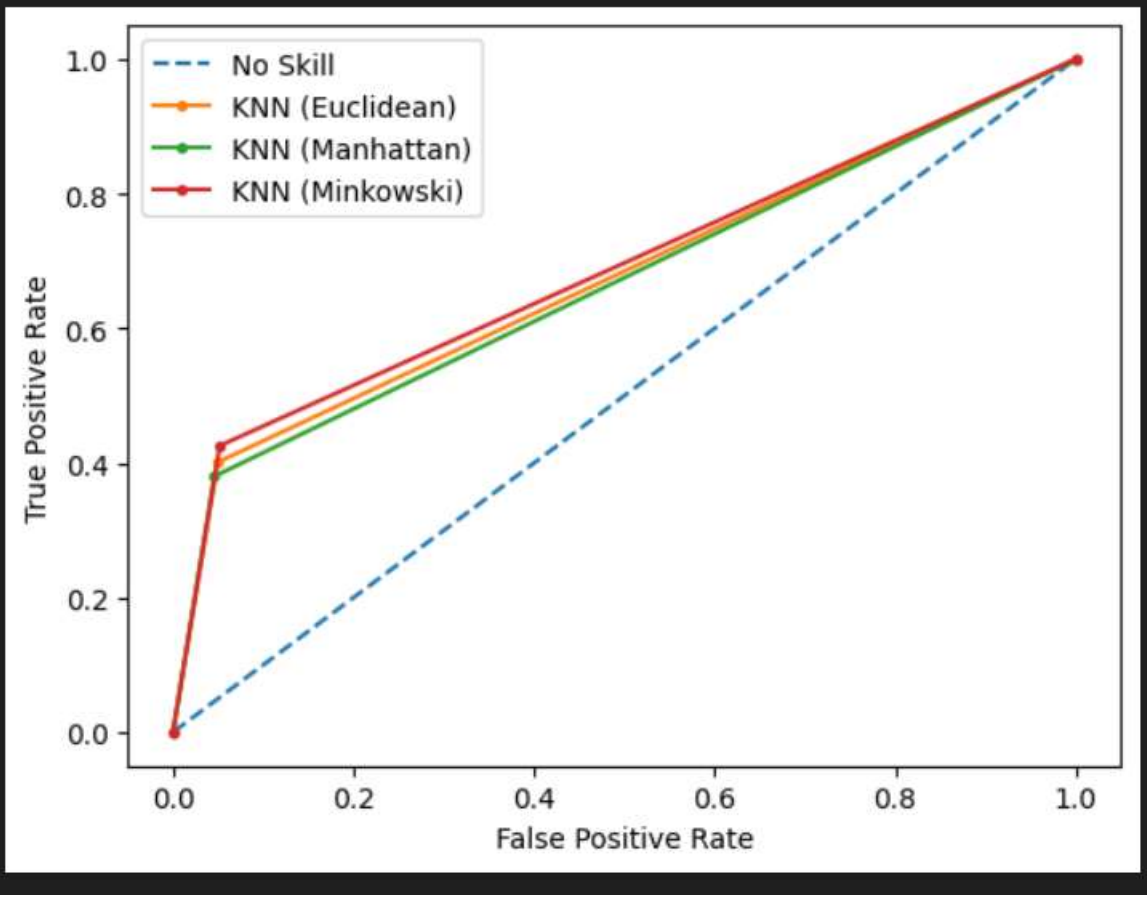
```
Accuracy (Minkowski): 0.8675317653419843
```

ROC curve

```
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve, roc_auc_score
import matplotlib.pyplot as plt

ns_probs = [0 for _ in range(len(y_test))]

knn_probs_euclidean = [1 if pred == 1 else 0 for pred in y_pred]

knn_probs_manhattan = [1 if pred == 1 else 0 for pred in y_pred_manhattan]

knn_probs_minkowski = [1 if pred == 1 else 0 for pred in y_pred_minkowski]

ns_auc = roc_auc_score(y_test, ns_probs)
knn_auc_euclidean = roc_auc_score(y_test, knn_probs_euclidean)
knn_auc_manhattan = roc_auc_score(y_test, knn_probs_manhattan)
knn_auc_minkowski = roc_auc_score(y_test, knn_probs_minkowski)

print('No Skill: ROC AUC=%.3f' % (ns_auc))
print('KNN (Euclidean): ROC AUC=%.3f' % (knn_auc_euclidean))
print('KNN (Manhattan): ROC AUC=%.3f' % (knn_auc_manhattan))
print('KNN (Minkowski): ROC AUC=%.3f' % (knn_auc_minkowski))

ns_fpr, ns_tpr, _ = roc_curve(y_test, ns_probs)
knn_fpr_euclidean, knn_tpr_euclidean, _ = roc_curve(y_test,
knn_probs_euclidean)
knn_fpr_manhattan, knn_tpr_manhattan, _ = roc_curve(y_test,
knn_probs_manhattan)
knn_fpr_minkowski, knn_tpr_minkowski, _ = roc_curve(y_test,
knn_probs_minkowski)

plt.plot(ns_fpr, ns_tpr, linestyle='--', label='No Skill')
plt.plot(knn_fpr_euclidean, knn_tpr_euclidean, marker='.', label='KNN
(Euclidean)')
plt.plot(knn_fpr_manhattan, knn_tpr_manhattan, marker='.', label='KNN
(Manhattan)')
plt.plot(knn_fpr_minkowski, knn_tpr_minkowski, marker='.', label='KNN
(Minkowski)')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
```

```
plt.legend()
plt.show()
```

```
No Skill: ROC AUC=0.500
KNN (Euclidean): ROC AUC=0.676
KNN (Manhattan): ROC AUC=0.667
KNN (Minkowski): ROC AUC=0.687
```



Results and Inference

```
func = ['Euclidean','Manhattan','Minkowski']
t = [acc,acc_manhattan,acc_minkowski]
x = func
res = pd.DataFrame({"Distance Function": x, "Accuracy": t})
print(res)
```

	Distance Function	Accuracy
0	Euclidean	0.865639
1	Manhattan	0.865639
2	Minkowski	0.867532

Learning Outcomes

- Implement KNN using user-defined functions to understand its algorithmic principles.
- Apply data preprocessing techniques for improved performance in KNN classification.
- Calculate distances between data points using various metrics like Euclidean and Manhattan distances.
- Experiment with hyperparameter tuning, adjusting 'k' to optimize KNN model performance.
- Evaluate KNN model performance using accuracy, precision, recall, and F1-score.
- Recognize and mitigate overfitting and underfitting issues in KNN through experimentation.