## Imports

```
from matplotlib.colors import ListedColormap
import numpy as np
import pandas as pd
import time
import math
import random
import seaborn as sns
import operator
from matplotlib import pyplot as plt
from sklearn.model_selection import KFold, train_test_split
from sklearn.metrics import confusion_matrix, f1_score, accuracy_score, classification_report, precision_score, recall_score, r2_score, mean_
import sklearn.metrics as metrics
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.neighbors import KNeighborsRegressor
from sklearn import metrics
from sklearn.svm import SVC
from sklearn import tree
from sklearn.tree import _tree
from sklearn import svm, metrics
from sklearn.model_selection import KFold, cross_val_score, cross_val_predict
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import DecisionTreeRegressor
```

# PART 1

## Method Definitons

```
def manhattanDistance(a, b, size):
    distance = 0
    for i in range(size):
        distance += math.fabs(a[i] - b[i])

    return distance
```

Manhattan Distance Method

```
def euclidDistance(a, b, size):
    distance = 0
    for i in range(size):
        distance += pow(a[i] - b[i], 2)

    return math.sqrt(distance)
```

Euclidian Distance Method

```
def display_confusion_matrix(y_test, y_pred):
    cm = confusion_matrix(y_test, y_pred)

    plt.rcParams["figure.figsize"] = [12,9]
    disp = ConfusionMatrixDisplay(confusion_matrix=cm)
    disp.plot()
    plt.show()
```

Print Confusion Matrix

```
def report_performance(y_test, y_pred):
    report = classification_report(y_test, y_pred)
    print(report)
```

precision - recall - f1-score - support

```
def display_roc(y_test, y_pred):
  print('Original Model')
  fpr, tpr, thres = metrics.roc_curve(y_test,  y_pred)

  plt.plot(fpr,tpr)
  plt.ylabel('True Positive Rate')
  plt.xlabel('False Positive Rate')
  plt.show()
  print(thres)
```

Create and print Roc Curve

```
def getNeighbors(X_train, y_train, x_test, k, dist_method):
    all_dist = []
    test_len = len(x_test)
    for i in range(len(X_train)):
        dist = dist_method(x_test, X_train[i], test_len)
        all_dist.append((X_train[i], y_train[i], dist))

    all_dist.sort(key=operator.itemgetter(2))
    return all_dist[:k]
```

X_train : input data

y_train : output data

x_test : test data

dist_method : euclidDistance or manhattanDistance

return : k# neighbors as : (X_train, y_train, distance)

```
def PreProcess(x_train, x_test):
    scx = StandardScaler()
    x_train = scx.fit_transform(x_train.astype(float))
    x_test = scx.transform(x_test.astype(float))

    return x_train, x_test
```

Pre Processing - Normalization - Standart Scaler

```
def maxRepeatItem(neighbors, class_index):
    arr = {}
    for i in range(len(neighbors)):
        result = neighbors[i][class_index]
        if result in arr:
            arr[result] += 1
        else:
            arr[result] = 1
    sortedArr = sorted(arr.items(), key=operator.itemgetter(1), reverse=True)
    return sortedArr[0][0]
```

Classifying helper method

Max repeated item's count

```
def KNNClassify(x_train, y_train, x_test, k=4, method='euclidean'):

    dist_method = euclidDistance if method == 'euclidean' else manhattanDistance

    knn = []
    for i in range(len(x_test)):
        neighbors = getNeighbors(x_train, y_train, x_test[i], k, dist_method)
        knn.append(maxRepeatItem(neighbors, 1))

    return np.array(knn)
```

KNN Classifier

x_train : input data

y_train : output data

x_test : test data

dist_method : euclidDistance or manhattanDistance

## Implementation

```
audit_risk = pd.read_csv("audit_data/audit_risk.csv")
trial = pd.read_csv("audit_data/trial.csv")
```

Read Data's

```
trial.columns = ['Sector_score','LOCATION_ID', 'PARA_A', 'Score_A', 'PARA_B', 'Score_B',  'TOTAL', 'numbers', 'Marks', 'Money_Value', 'MONEY
trial['Score_A'] = trial['Score_A']/10
trial['Score_B'] = trial['Score_B']/10
merged_df = pd.merge(audit_risk, trial, how='outer', on = ['History', 'LOCATION_ID', 'Money_Value', 'PARA_A', 'PARA_B', 'Score', 'Score_A',
```

Merge test and train data parts in merged_df

```
df = merged_df.drop(['Risk_trial', 'Detection_Risk', 'Risk_F'], axis = 1)
```

Pre Processing - Delete some columns from df

```
df['Money_Value'] = df['Money_Value'].fillna(df['Money_Value'].median())
```

Pre Processing - Fill empty nodes

```
df = df[(df.LOCATION_ID != 'LOHARU')]
df = df[(df.LOCATION_ID != 'NUH')]
df = df[(df.LOCATION_ID != 'SAFIDON')]
df = df.astype(float)
```

Pre Processing - Filter some columns

```
df = df.drop_duplicates(keep = 'first')
df = df[['Risk_A', 'Risk_B', 'Risk_C', 'Risk_D','RiSk_E', 'Prob', 'Score', 'CONTROL_RISK', 'Audit_Risk', 'Risk', 'MONEY_Marks', 'Loss']]

audit_class_df = df.drop("Audit_Risk", axis = 1)
audit_class_df.info()
audit_class_df.to_csv("audit_data/audit_clean_data.csv", index=False)
print("Updated number of rows in the dataset: ",len(df))
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 760 entries, 0 to 809
Data columns (total 11 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Risk_A        760 non-null    float64
 1   Risk_B        760 non-null    float64
 2   Risk_C        760 non-null    float64
 3   Risk_D        760 non-null    float64
 4   RiSk_E        760 non-null    float64
 5   Prob          760 non-null    float64
 6   Score         760 non-null    float64
 7   CONTROL_RISK  760 non-null    float64
 8   Risk          760 non-null    float64
 9   MONEY_Marks   760 non-null    float64
 10  Loss          760 non-null    float64
dtypes: float64(11)
memory usage: 71.2 KB
Updated number of rows in the dataset:  760
```

Pre Processing - Delete duplicated datas

```
%%time
dataset = audit_class_df

cX = dataset.drop(['Risk'], axis=1)
cy = dataset['Risk']

X_train_org, x_test_org, Y_train, y_test = train_test_split(cX, cy, test_size = 0.30, random_state = 0)

x_train = X_train_org.values.tolist()
x_test = x_test_org.values.tolist()
y_train = Y_train.tolist()
y_test = y_test.tolist()

y_pred = KNNClassify(x_train, y_train, x_test, k=4, method='manhattan')

total_y_test = np.array(y_test)
total_y_pred = np.array(y_pred)
print("manhattan_distance")

print("Accuracy : %",accuracy_score(total_y_test, total_y_pred) * 100)
```

```
manhattan_distance
Accuracy : % 96.9298245614035
CPU times: user 373 ms, sys: 1.65 ms, total: 375 ms
Wall time: 384 ms
```

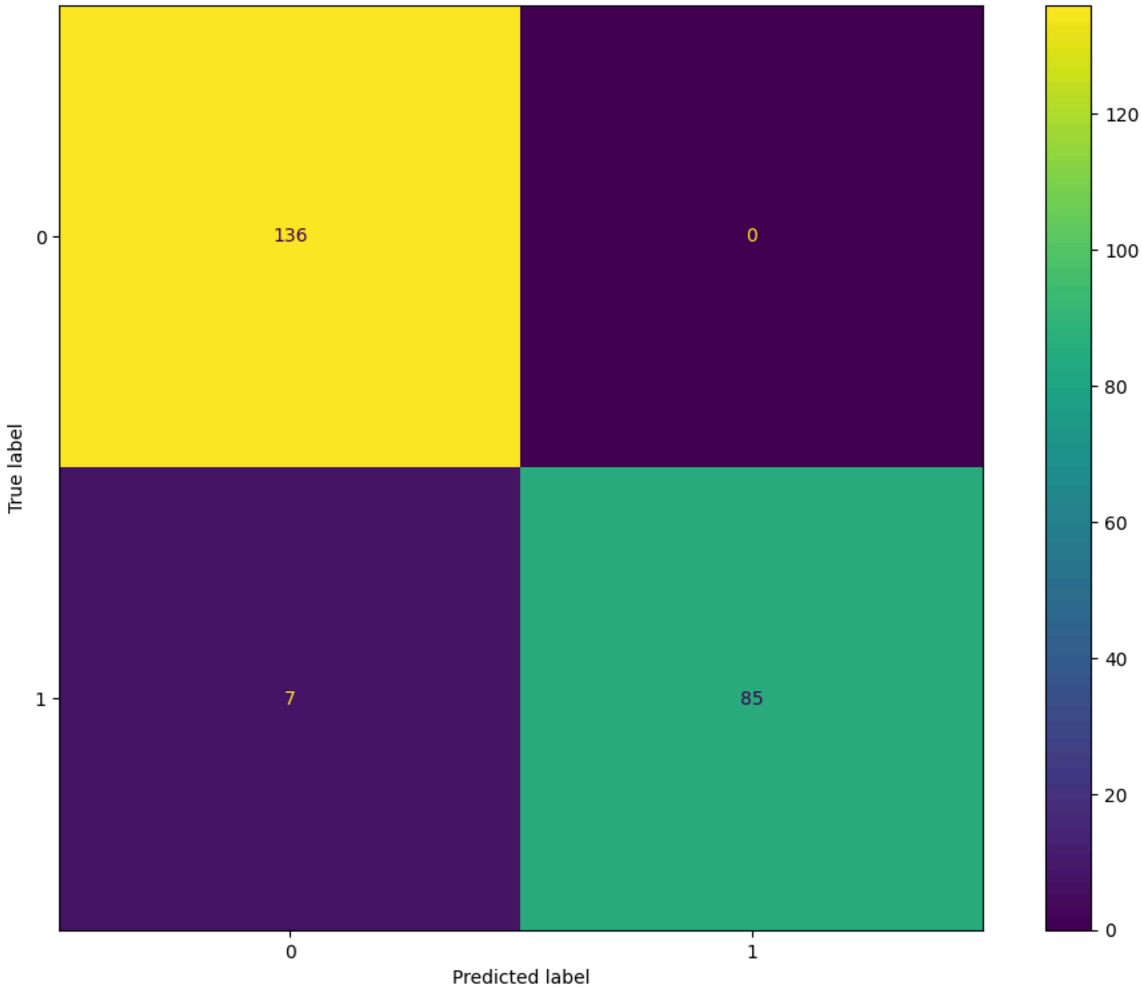
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

         0.0       0.95      1.00      0.97       136
         1.0       1.00      0.92      0.96        92

    accuracy                           0.97       228
   macro avg       0.98      0.96      0.97       228
weighted avg       0.97      0.97      0.97       228
```

```
print(confusion_matrix(y_test, y_pred))
```

```
[[136   0]
 [  7  85]]
```

```
display_confusion_matrix(total_y_test.astype(int), total_y_pred.astype(int))
```



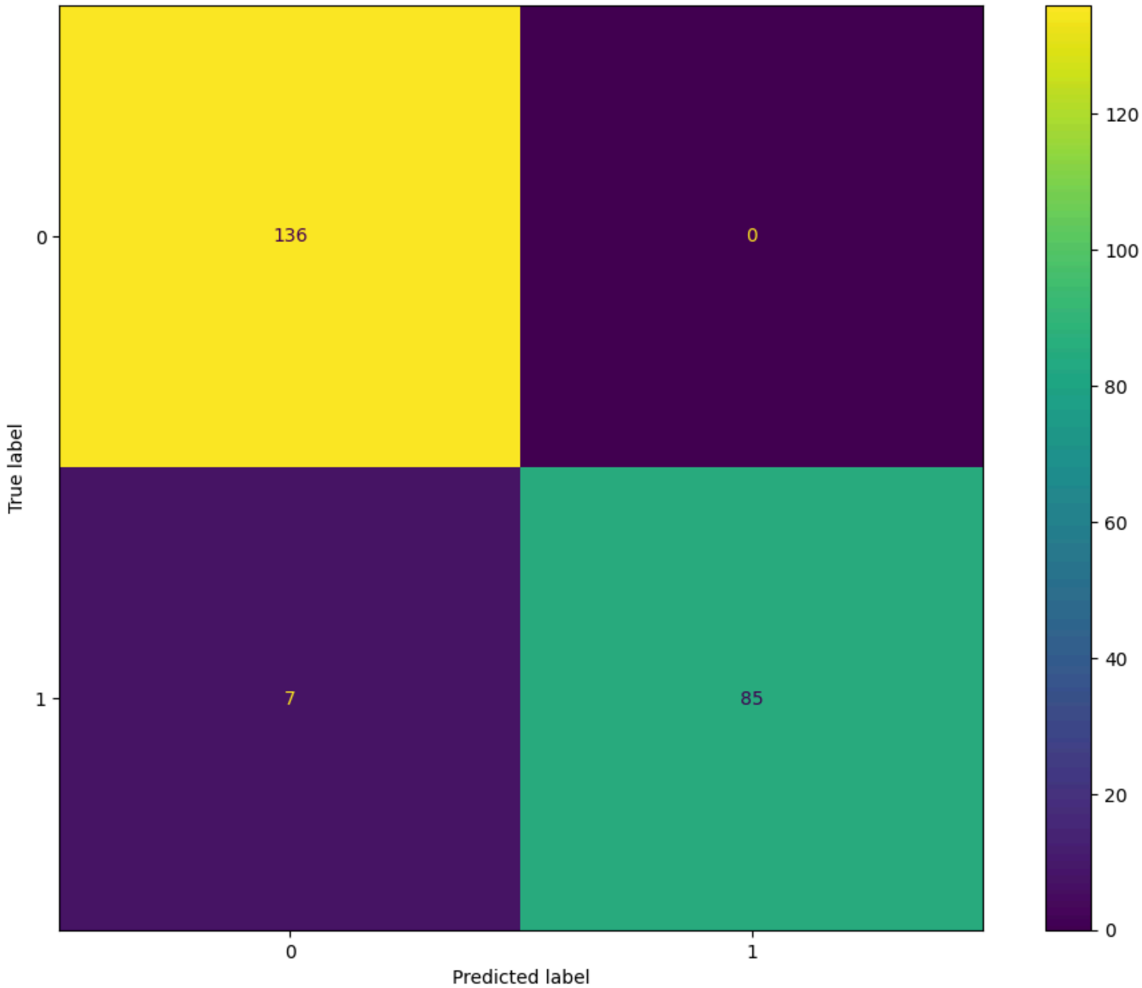

```
y_pred = KNNClassify(x_train, y_train, x_test, k=4, method='euclidean')

total_y_test = np.array(y_test)
total_y_pred = np.array(y_pred)
print("euclidean_distance")

print("Accuracy : %",accuracy_score(total_y_test, total_y_pred) * 100)

print(classification_report(y_test, y_pred))
```

```
euclidean_distance
Accuracy : % 96.9298245614035
              precision    recall  f1-score   support

         0.0       0.95      1.00      0.97       136
         1.0       1.00      0.92      0.96        92

    accuracy                           0.97       228
   macro avg       0.98      0.96      0.97       228
weighted avg       0.97      0.97      0.97       228
```

```
print(confusion_matrix(y_test, y_pred))
```

```
[[136   0]
 [  7  85]]
```

```
display_confusion_matrix(total_y_test.astype(int), total_y_pred.astype(int))
```




# PART2

## Method Definitions

```
def manhattanDistance(a, b, size):
  distance = 0
  for i in range(size):
    distance += math.fabs(a[i] - b[i])
  return distance
```

```
def sort_list(val, n):
  for i in range(len(val)):
    min = i
    for j in range(i+1, len(val)):
      if val[j] < val[min]:
        min = j
    val[i], val[min] = val[min], val[i]
  return val[:n]
```

Sorting Method

```
def get_index(val, n):
  sort_value = sort_list(list(val.values()), n)
  index = []
  for i in range(len(val)):
    if val[i] in sort_value:
      index.append(i)
    if len(index) == n:
      break
  return index
```

Sort the list and detect the n'th value

```
def sort_distance_man(X, t, n):
  distance = {}
  for j in range(len(X)):
    distance[j] = manhattanDistance(X[j], t, len(X[j]))
  index = get_index(distance, n)
  return index
```

Sort distance with Manhattan Distance

```
def get_y_value(y, index):
  total = 0
  for i in range(len(y)):
    if i in index:
      total += y[i]
  return total
```

Summation of values that index is equal in the y

```
def KNNRegressionpredict(X, y, T, n):
  total = []
  for i in range(len(T)):
    index = sort_distance_man(X, T[i], n)
    total.append((get_y_value(y, index)) / n)
  return total
```

KNN Regression Predict

For every index sum of the y values

## Implementation

```
day_df = pd.read_csv('bike_data/day.csv')
hour_df = pd.read_csv('bike_data/hour.csv')
```

Read data from file

```
day_df.rename(columns={'instant':'id','dteday':'Date','yr':'Year','mnth':'Month',
                       'weathersit':'WeatherCondition','atemp':'FeelinTemp',
                       'hum':'Humidity','cnt':'TotalRentDay'},inplace=True)

hour_df.rename(columns={'instant':'rec_id','dteday':'datetime','holiday':'is_holiday',
                     'workingday':'is_workingday', 'weathersit':'weather_condition',
                     'hum':'humidity','mnth':'month', 'cnt':'total_count','hr':'hour',
                     'yr':'year'},inplace=True)
```

Pre Processing - Rename the dataset columns

```
day_df['Date']=pd.to_datetime(day_df.Date)
day_df['season']=day_df.season.astype('category')
day_df['Year']=day_df.Year.astype('category')
day_df['Month']=day_df.Month.astype('category')
day_df['holiday']=day_df.holiday.astype('category')
day_df['weekday']=day_df.weekday.astype('category')
day_df['workingday']=day_df.workingday.astype('category')
day_df['WeatherCondition']=day_df.WeatherCondition.astype('category')
day_df = day_df.drop(['Date'], axis=1)
```

Pre Processing - Type Declaration for every day column

```
hour_df['datetime']=pd.to_datetime(hour_df.datetime)
hour_df['season']=hour_df.season.astype('category')
hour_df['year']=hour_df.year.astype('category')
hour_df['month']=hour_df.month.astype('category')
hour_df['hour']=hour_df.hour.astype('category')
hour_df['is_holiday']=hour_df.is_holiday.astype('category')
hour_df['weekday']=hour_df.weekday.astype('category')
hour_df['is_workingday']=hour_df.is_workingday.astype('category')
hour_df['weather_condition']=hour_df.weather_condition.astype('category')
hour_df = hour_df.drop(['datetime'], axis=1)
```

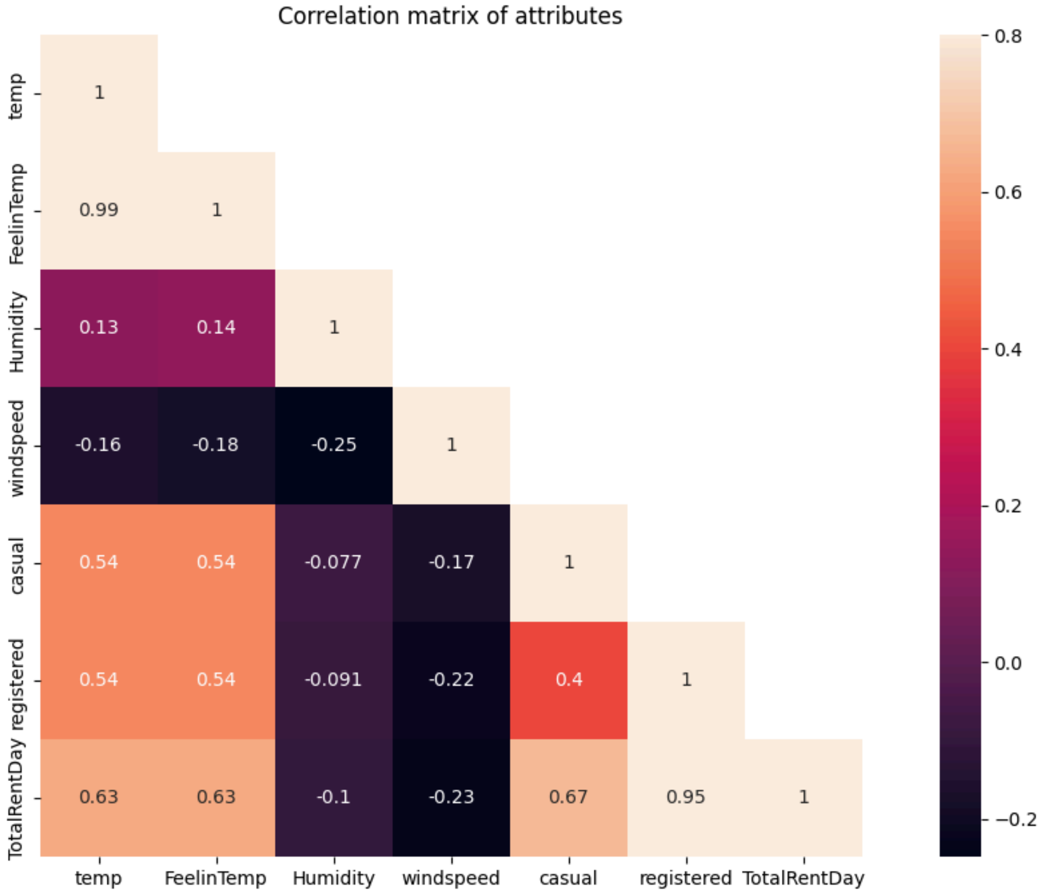
Pre Processing - Type Declaration for every hour column

```
print("DaysRents_df: {}\nHourlyRents_df: {}".format(day_df.shape, hour_df.shape))
print(hour_df.columns)
print(day_df.columns)
```

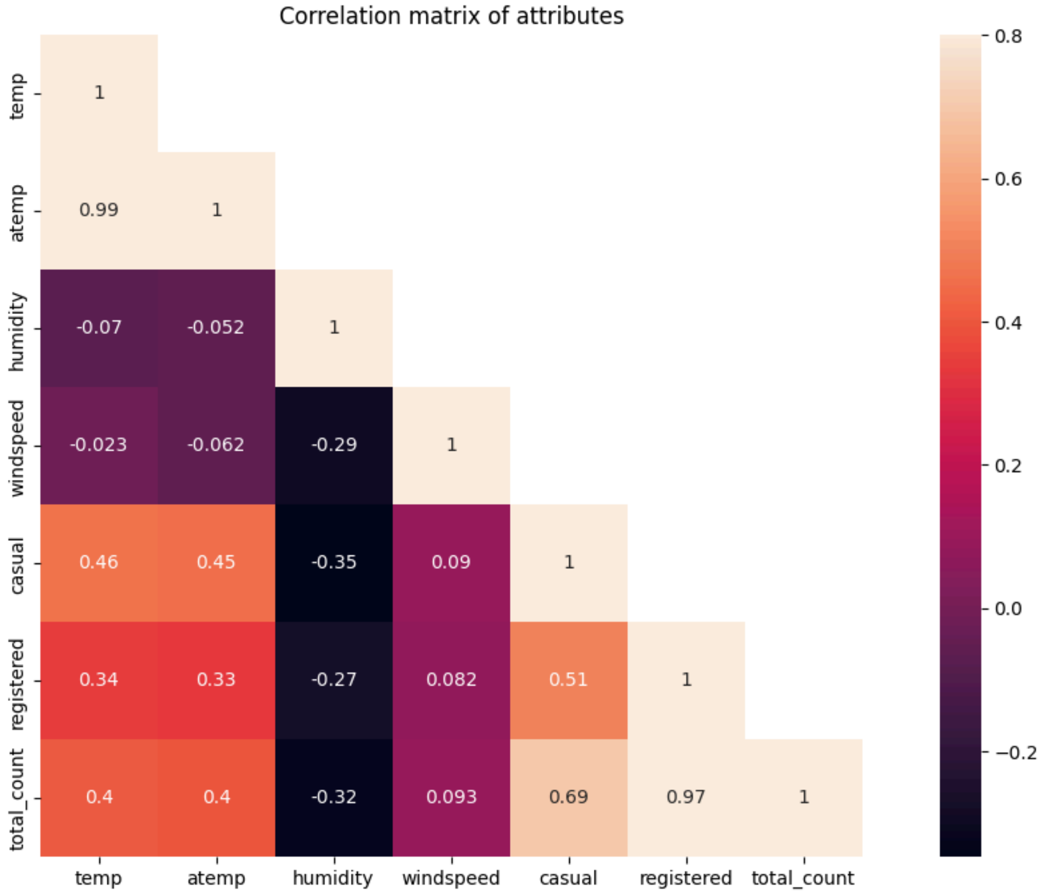
```
DaysRents_df: (731, 15)
HourlyRents_df: (17379, 16)
Index(['rec_id', 'season', 'year', 'month', 'hour', 'is_holiday', 'weekday',
       'is_workingday', 'weather_condition', 'temp', 'atemp', 'humidity',
       'windspeed', 'casual', 'registered', 'total_count'],
      dtype='object')
Index(['id', 'season', 'Year', 'Month', 'holiday', 'weekday', 'workingday',
       'WeatherCondition', 'temp', 'FeelinTemp', 'Humidity', 'windspeed',
       'casual', 'registered', 'TotalRentDay'],
      dtype='object')
```

Pre Processing - Day and Hour dataset's format declaration

```
#Day Dataset - Correlation Matrix
correMtr=day_df[["temp","FeelinTemp","Humidity","windspeed","casual","registered","TotalRentDay"]].corr()
mask=np.array(correMtr)
mask[np.tril_indices_from(mask)]=False

#Heat map
fig,ax=plt.subplots(figsize=(15,8))
sns.heatmap(correMtr,mask=mask,vmax=0.8,square=True,annot=True,ax=ax)
ax.set_title('Correlation matrix of attributes')
plt.show()

print()
print()
```




```
#Hour Dataset - Correlation Matrix
correMtr=hour_df[["temp","atemp","humidity","windspeed","casual","registered","total_count"]].corr()
mask=np.array(correMtr)
mask[np.tril_indices_from(mask)]=False

#Heat map
fig,ax=plt.subplots(figsize=(15,8))
sns.heatmap(correMtr,mask=mask,vmax=0.8,square=True,annot=True,ax=ax)
ax.set_title('Correlation matrix of attributes')
plt.show()
```




```
dataset = hour_df[["temp","atemp","humidity","windspeed","casual","registered","total_count"]]
target = hour_df['total_count']

X_train, X_test, y_train, y_test = train_test_split(dataset,target,test_size=0.3,random_state=0)
```

X train - X test Definitions

```
sc = StandardScaler()
x_train = sc.fit_transform(X_train)
x_test = sc.transform(X_test)
```

Pre Processing - Normalization

```
knn = KNeighborsRegressor(n_neighbors = 3)
knn.fit(x_train, y_train)
```

```
▾ KNeighborsRegressor
KNeighborsRegressor(n_neighbors=3)
```

Fit Regression

```
y_pred = knn.predict(x_test)
print ("Regression score is:",format(metrics.r2_score(y_test, y_pred),'.3f'), "for k_value:", 3)
```

```
Regression score is: 0.993 for k_value: 3
```

KNN Prediction

```
k_range = range(1, 10)
k_scores = []
for k in k_range:
    knn_org = KNeighborsRegressor(n_neighbors=k)
    scores = cross_val_score(knn_org, x_train, y_train, cv=10, scoring='neg_root_mean_squared_error')
    k_scores.append(scores.mean())
```

K-Cross Validation

k = 10

```
BestScore = [1 - x for x in k_scores]
best_k = k_range[BestScore.index(min(BestScore))]
```

Detect best k-score

```
classifier_org = KNeighborsRegressor(n_neighbors = best_k)
classifier_org.fit(x_train, y_train)
y_pred_org = classifier_org.predict(x_test)
```

Classifier Fitting

```
print('Original Model')
print('\nn_neighbors:',str(best_k))
print('\nR2: {:.2f}'.format(metrics.r2_score(y_test, y_pred_org)))
adjusted_r_squared = 1-(1-metrics.r2_score(y_test,y_pred_org))*(len(target)-1)/(len(target)-dataset.shape[1]-1)
print('Adj_R2: {:0.2f}'.format(adjusted_r_squared))
print('Mean Absolute Error: {:0.2f}'.format(metrics.mean_absolute_error(y_test, y_pred_org)))
print('Mean Squared Error: {:0.2f}'.format(metrics.mean_squared_error(y_test, y_pred_org)))
print('Root Mean Squared Error: {:0.2f}'.format(np.sqrt(metrics.mean_squared_error(y_test, y_pred_org))))
```

```
Original Model

n_neighbors: 9

R2: 0.99
Adj_R2: 0.99
Mean Absolute Error: 9.61
Mean Squared Error: 185.06
Root Mean Squared Error: 13.60
```

# PART 3

```
%%time

dataset = audit_class_df.drop(['Risk'], axis=1)
target = audit_class_df['Risk']
```

```
CPU times: user 2.52 ms, sys: 0 ns, total: 2.52 ms
Wall time: 3.57 ms
```

Delete "Risk" elements from dataset

```
X_train_org, x_test_org, Y_train, y_test = train_test_split(dataset,target,test_size=0.3,random_state=0)
```

Train ve test splitting

```
sc = StandardScaler()
X_train = sc.fit_transform(X_train_org)
X_test = sc.transform(x_test_org)
```

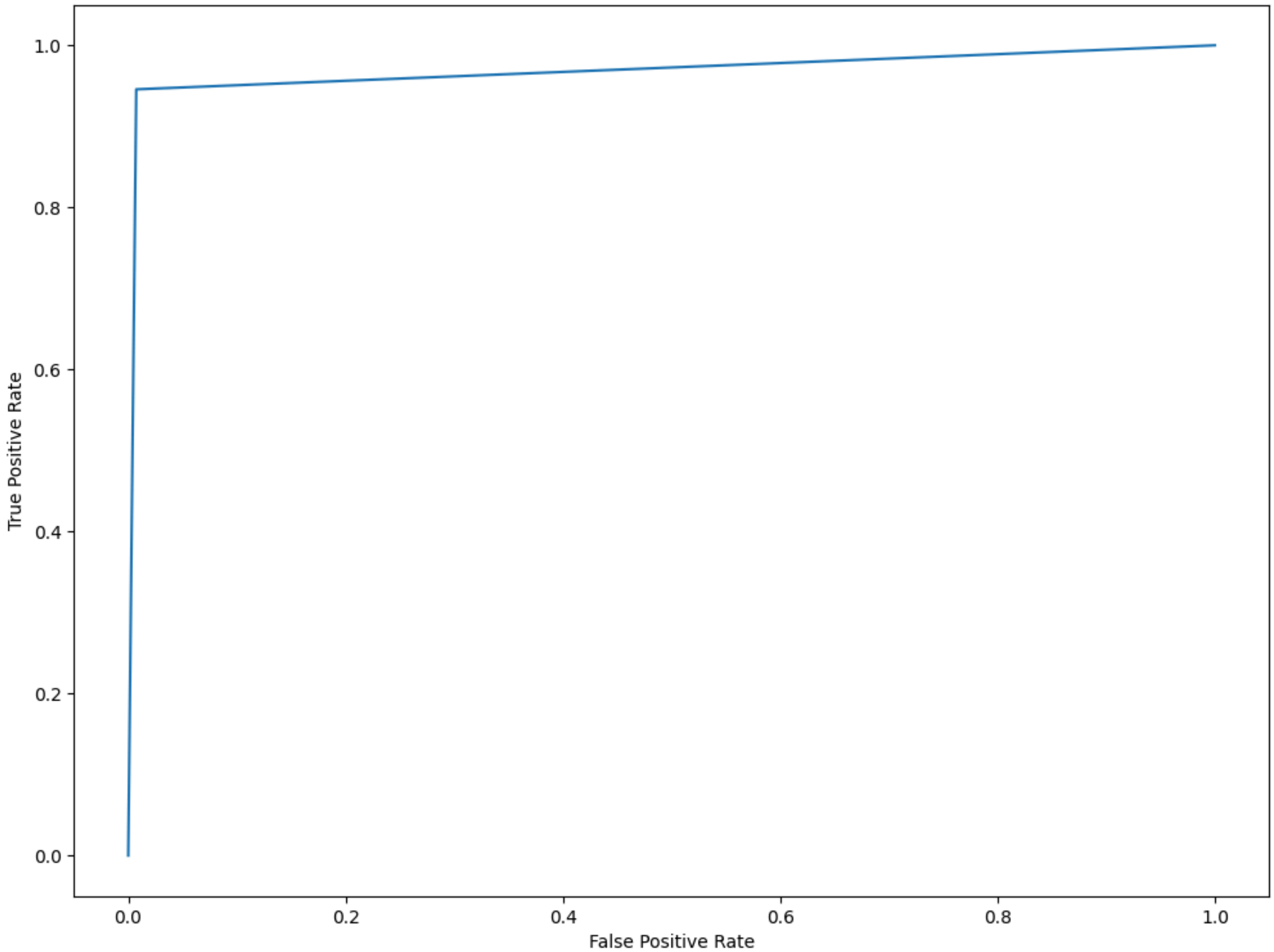
Pre Processing - Normalization

```
svc = SVC(kernel='linear',gamma='scale')
svc.fit(X_train, Y_train)
y_pred = svc.predict(X_test)
```

Lİnear SVC definition

```
display_roc(y_test, y_pred)

print("Accuracy : %",accuracy_score(y_test, y_pred) * 100)

report_performance(y_test, y_pred)
```

```
Original Model
```




```
[2. 1. 0.]
Accuracy : % 97.36842105263158
              precision    recall  f1-score   support

         0.0       0.96      0.99      0.98       136
         1.0       0.99      0.95      0.97        92

    accuracy                           0.97       228
   macro avg       0.98      0.97      0.97       228
weighted avg       0.97      0.97      0.97       228
```

## PART 4

```
%%time

dataset = hour_df[["temp","atemp","humidity","windspeed","casual","registered","total_count"]]
target = hour_df['total_count']
```

```
CPU times: user 2.77 ms, sys: 0 ns, total: 2.77 ms
Wall time: 2.66 ms
```

Target and dataset definition from Hour dataset

```
dataset = np.array_split(dataset, 2)[0]
target = np.array_split(target, 2)[0]

X_train, X_test, y_train, y_test = train_test_split(dataset,target,test_size=0.3,random_state=0)
```

Dataset turn into numpy data and split the train and test

```
sc = StandardScaler()
x_train = sc.fit_transform(X_train)
x_test = sc.transform(X_test)
```

Pre Processing - Normalization

```
svr = svm.SVR(kernel='rbf', C=1e3, gamma=0.1)
svr.fit(x_train, y_train)
```

```
SVR
SVR(C=1000.0, gamma=0.1)
```

SVM fitting -Radial Basis Function

```
y_pred = svr.predict(x_test)
print ("Regression score is:",format(metrics.r2_score(y_test, y_pred),'.4f'))
```

```
Regression score is: 0.9999
```

Prediction of SVR - Regression r^2 Score

## KFold

```
%%time

dataset = hour_df[["temp","atemp","humidity","windspeed","casual","registered","total_count"]]
target = hour_df['total_count']
```

```
CPU times: user 1.66 ms, sys: 574 µs, total: 2.23 ms
Wall time: 2.75 ms
```

Test and Train splitting

```
dataset = np.array_split(dataset, 2)[0]
target = np.array_split(target, 2)[0]
```

Dataset turn into to numpy dataset

```
kf = KFold(n_splits=6, shuffle=True, random_state=0)

train_scores = []
test_scores = []


svr = svm.SVR(kernel='rbf', C=1e3, gamma=0.1)


sc = StandardScaler()
```

```
for train_index, test_index in kf.split(dataset):
    X_train, X_test = dataset.iloc[train_index], dataset.iloc[test_index]
    y_train, y_test = target.iloc[train_index], target.iloc[test_index]

    x_train = sc.fit_transform(X_train)
    x_test = sc.transform(X_test)

    svr.fit(x_train, y_train)

    y_train_pred = svr.predict(x_train)
    y_test_pred = svr.predict(x_test)

    train_score = r2_score(y_train, y_train_pred)
    test_score = r2_score(y_test, y_test_pred)

    train_scores.append(train_score)
    test_scores.append(test_score)


mean_train_score = np.mean(train_scores)
std_train_score = np.std(train_scores)
mean_test_score = np.mean(test_scores)
std_test_score = np.std(test_scores)

print('SVR Regression Score (mean) for train set:', format(mean_train_score, '.4f'), ', Standard deviation for train set:', format(std_trair
print('SVR Regression Score (mean) for test set:', format(mean_test_score, '.4f'), ', Standard deviation for test set:', format(std_test_scc
```

```
SVR Regression Score (mean) for train set: 1.0000 , Standard deviation for train set: 0.0000
SVR Regression Score (mean) for test set: 1.0000 , Standard deviation for test set: 0.0000
```

# PART 5

```
dataset = audit_class_df
```

```
cX = dataset.drop(['Risk'], axis=1)
cy = dataset['Risk']
```

```
X_train, X_test, y_train, y_test = train_test_split(cX, cy, test_size=0.3, random_state=42)
```

Split the dataset into training and testing sets

```
svm_model = SVC(kernel='poly', degree=3)
svm_model.fit(X_train, y_train)
```

```
SVC
SVC(kernel='poly')
```

Train a polynomial kernel SVM model

```
y_pred = svm_model.predict(X_test)
```

Make predictions on the testing set

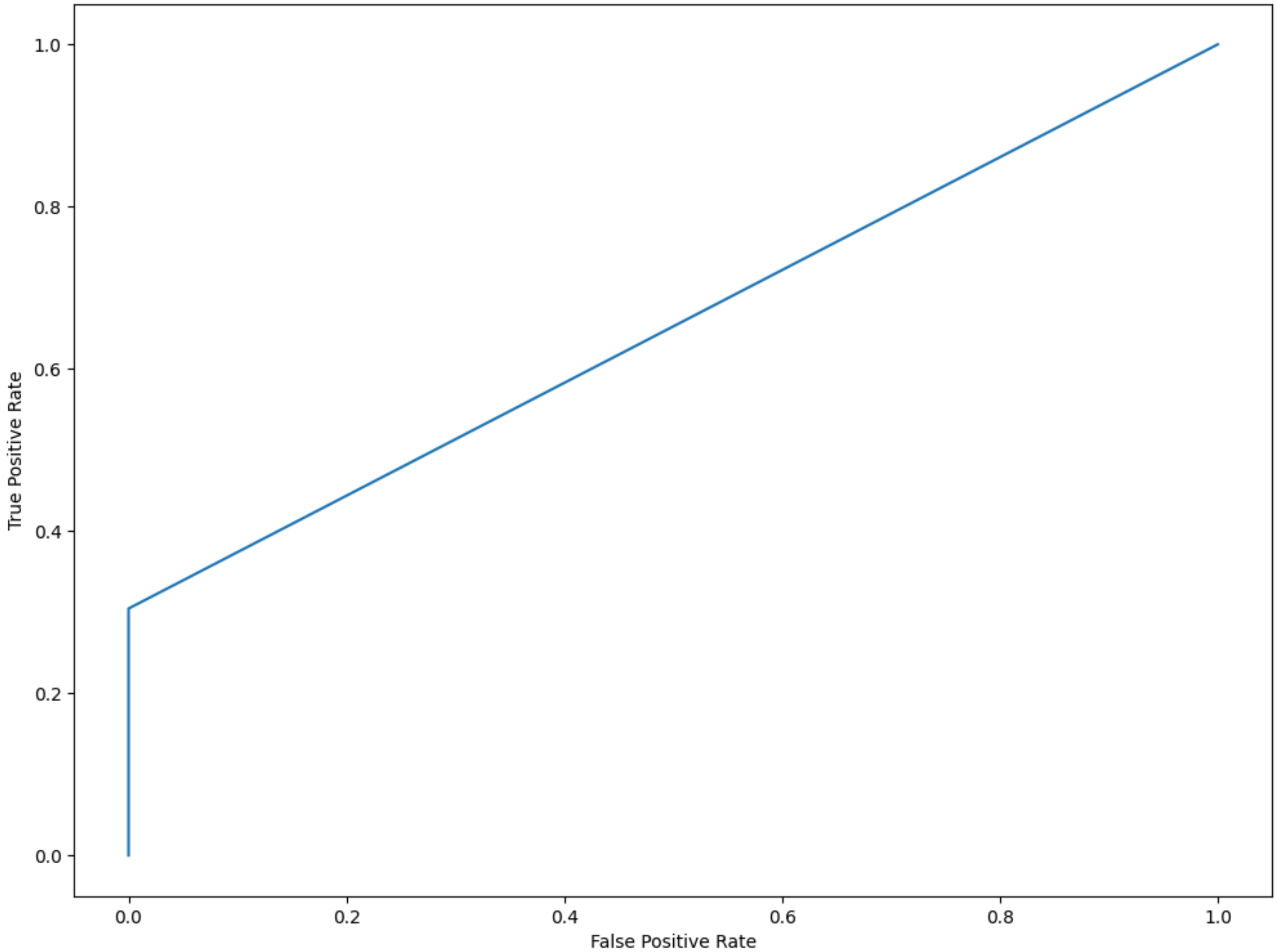
```
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.7192982456140351
```

Display Accuracy

```
display_roc(y_test, y_pred)
```

```
Original Model
```




```
[2. 1. 0.]
```

Print Roc Curve

```
report_performance(y_test, y_pred)
```

```
              precision    recall  f1-score   support

         0.0       0.68      1.00      0.81       136
         1.0       1.00      0.30      0.47        92

    accuracy                           0.72       228
   macro avg       0.84      0.65      0.64       228
weighted avg       0.81      0.72      0.67       228
```

precision - recall - f1-score - support

```
confusion_matrix(y_test, y_pred)
```

```
array([[136,   0],
       [ 64,  28]])
```

## PART 6

```
def tree_to_rules(tree, feature_names):

    rules = []
    def traverse(node_id, current_rule):
        feature_index = tree.tree_.feature[node_id]
        threshold = tree.tree_.threshold[node_id]
        feature_name = feature_names[feature_index]

        if tree.tree_.children_left[node_id] == tree.tree_.children_right[node_id]:
            class_label = np.argmax(tree.tree_.value[node_id][0])
            rules.append((current_rule, class_label))
            return

        left_rule = current_rule + [(feature_name, "<=", threshold)]
        traverse(tree.tree_.children_left[node_id], left_rule)
        right_rule = current_rule + [(feature_name, ">", threshold)]
        traverse(tree.tree_.children_right[node_id], right_rule)

    traverse(0, [])
    return rules


dataset = audit_class_df


cX = dataset.drop(['Risk'], axis=1)
cy = dataset['Risk']
```

Delete "Risk" column from dataset

```
X_train, X_test, y_train, y_test = train_test_split(cX, cy, test_size=0.2, random_state=42)

print()
```

Train and test splitting

## Pre Prunning

```
clf_pre = DecisionTreeClassifier(max_depth=3, min_samples_leaf=5, random_state=42)
clf_pre.fit(X_train, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=3, min_samples_leaf=5, random_state=42)
```

Decision Tree fitting - Pre - prunning

```
y_pred = clf_pre.predict(X_test)
```

Prediction of fitted data

```
print("Accuracy:", accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.9078947368421053
```

```
print("Precision:", precision_score(y_test, y_pred))
```

```
Precision: 0.9615384615384616
```

```
print("Recall:", recall_score(y_test, y_pred))
```

```
Recall: 0.8064516129032258
```

```
print("F1 score:", f1_score(y_test, y_pred))
```

```
F1 score: 0.8771929824561403
```

```
confusion_matrix(y_test.astype(int), y_pred.astype(int))
```

```
print()
```

```
print()
```

## Post Prunning

```
clf_post = DecisionTreeClassifier(random_state=42)
clf_post.fit(X_train, y_train)

path = clf_post.cost_complexity_pruning_path(X_train, y_train)
```

Decision Tree fitting - Post - prunning

```
ccp_alphas, impurities = path.ccp_alphas, path.impurities
clfs = []
for ccp_alpha in ccp_alphas:
    clf = DecisionTreeClassifier(random_state=42, ccp_alpha=ccp_alpha)
    clf.fit(X_train, y_train)
    clfs.append(clf)
y_pred_post = clfs[-1].predict(X_test)
```

Decision Tree classifier fitting

```
print("Accuracy:", accuracy_score(y_test, y_pred_post))
```

```
Accuracy: 0.5921052631578947
```

```
print("Precision:", precision_score(y_test, y_pred_post))
```

```
Precision: 0.0
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined and be
  _warn_prf(average, modifier, msg_start, len(result))
```

```
print("Recall:", recall_score(y_test, y_pred_post))
```

```
Recall: 0.0
```

```
print("F1 score:", f1_score(y_test, y_pred_post))
```

```
F1 score: 0.0
```

```
confusion_matrix(y_test.astype(int), y_pred_post.astype(int))
```

```
array([[90,  0],
       [62,  0]])
```

```
dataset = audit_class_df
```

```
X = dataset.drop(['Risk'], axis=1)
y = dataset['Risk']
```

```
feature_names = X.columns

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