Module 2 Homework

ISE-529 Predictive Analytics

Chinmay Gherde

Note: In answering the following questions, you may only use functionality from Base Python, NumPy, Pandas, or Seaborn

In [1]:

```
#importing libraries

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

1. Evaluating regression functions

1A) The file "HW 2 Problem 1 Data.xlsx" contains two datasets - training and test in two different spreadsheet tabs. Read the tables into two dataframes (training_data and test_data) and display the first 10 rows of each dataframe. Hint - look up the Pandas function for reading in Excel files.

In [2]:

```
training_data = pd.read_excel(r'C:\Users\Chinmay\Downloads\HW 2 Problem 1 Data.xlsx', sheet
test_data = pd.read_excel(r'C:\Users\Chinmay\Downloads\HW 2 Problem 1 Data.xlsx', sheet_nam
```

In [3]:

```
#displaying first 10 rows of each
training_data.head(10)
```

Out[3]:

	X	Y
0	61	17661.067682
1	87	15482.455058
2	38	17444.767982
3	6	-4270.225550
4	54	8075.733045
5	29	16820.129406
6	77	23921.367914
7	70	16541.631267
8	21	4425.240897
9	76	30681.044509

In [4]:

```
test_data.head(10)
```

Out[4]:

	X	Y
0	36	15986.579536
1	64	-2761.487999
2	66	9752.039830
3	88	20038.697197
4	4	-13421.192312
5	80	6644.121448
6	22	-10124.906855
7	85	21502.561217
8	63	12105.189261
9	8	-12161.603294

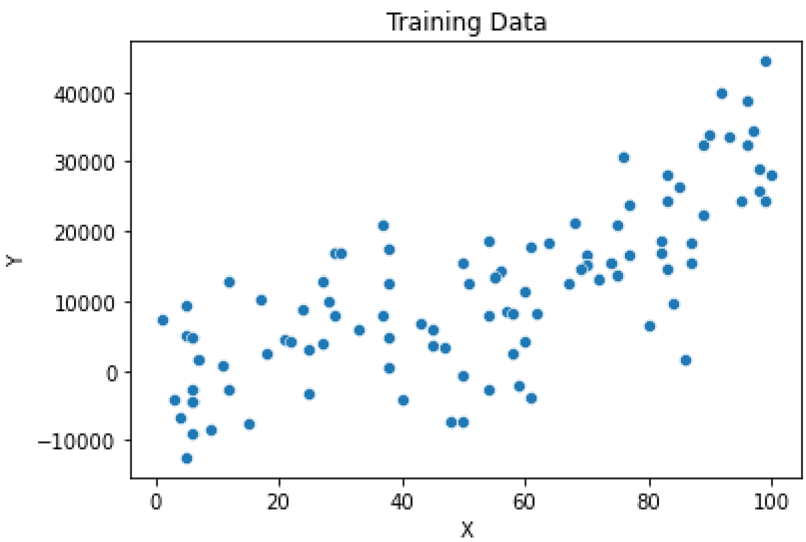
1b) Using Seaborn, create scatterplots of the two dataframes. For full credit, include a title ("Training Data" and "Test Data") for the two scatterplots.

In [5]:

```
sns.scatterplot(data=training_data, x="X", y="Y").set(title="Training Data")
```

Out[5]:

```
[Text(0.5, 1.0, 'Training Data')]
```



In [6]:

```
sns.scatterplot(data=test_data, x="X", y="Y").set(title="Test Data")
```

Out[6]:

```
[Text(0.5, 1.0, 'Test Data')]
```

1c) Now, we are going to calculate the test and training MSEs for three different candidate regression models:

- F1: $300x - 4280$
- F2: $3.37x^2 - 36.8x + 1220$
- F3: $0.1x^3 - 12x^2 + 590x - 3600$

Create three functions for these three candidate models - f1(), f2(), and f3(). They should take as an input x and return the predicted value of y corresponding to that input:

In [7]:

```
def f1(x):

    return 300*x-4280
```

In [8]:

```
def f2(x):

    return 3.37*(x**2) - 36.8*x + 1220
```

In [9]:

```
def f3(x):

    return 0.1*(x**3) - 12*(x**2) + 590*x - 3600
```

In [10]:

```
##testing
x1 = test_data['X']
y1 = test_data['Y']

a1 = f1(x1)
```

1d) Now, write a function calc_mse() that takes three parameter inputs:

- x - an array (or Pandas series) of predictor X values
- y - an array (or Pandas series) of response Y values
- f - a function to be called for each value of the x and y arrays

The function should calculate the MSE for the function f using the x and y data arrays

In [12]:

```
def calc_mse(x,y,f):
    y_pred = f(x)
    mse = np.square(np.subtract(y, y_pred)).mean()
    return mse
```

1e) Call this calc_mse function six times to calculate the training and test MSE for each of the three models:

In [13]:

```
# For training dataset

x_train = training_data['X']
y_train = training_data['Y']

mse_train_1 = calc_mse(x_train, y_train, f1)
print(f'MSE for training data model 1: {mse_train_1}')
```

```
MSE for training data model 1: 65116445.47500964
```

In [14]:

```
mse_train_2 = calc_mse(x_train, y_train, f2)
print(f'MSE for training data model 2: {mse_train_2}')
```

```
MSE for training data model 2: 57922250.15746113
```

In [15]:

```
mse_train_3 = calc_mse(x_train, y_train, f3)
print(f'MSE for training data model 3: {mse_train_3}')
```

```
MSE for training data model 3: 54103936.344325066
```

In [16]:

```
x_test_data = test_data['X']
y_test_data = test_data['Y']

mse_test_1 = calc_mse(x_test_data, y_test_data, f1)
print(f'MSE for test data model 1: {mse_test_1}')
```

```
MSE for test data model 1: 67828868.62264524
```

In [17]:

```
mse_test_2 = calc_mse(x_test_data, y_test_data, f2)
print(f'MSE for test data model 2: {mse_test_2}')
```

```
MSE for test data model 2: 67805434.17472021
```

In [18]:

```
mse_test_3 = calc_mse(x_test_data, y_test_data, f3)
print(f'MSE for test data model 3: {mse_test_3}')
```

```
MSE for test data model 3: 71191752.34205785
```

1f) Which of the three models would you select for use and why?

I will choose function f2 because it has consistently lower mse for training and testing dataframes. Plus its quadratic in nature and seems to capture dataset well.

Type *Markdown* and LaTeX: α^2

1g) Insted of writing functions, write a single line of Python code to calculate the test MSE for function F1. Hint: use the Python map and lambda functions.

In [80]:

```
mse_ = test_data.apply(lambda x: (x['Y'] - f1(x['X']))**2, axis=1).mean()
mse_
```

Out[80]:

```
67828868.62264524
```

2. KNN and Calculate Misclassification Rates

2a) Read the file "HW 2 Problem 2 Data.csv" into a dataframe called knn_data and display its first 10 rows

In [23]:

```
knn_data = pd.read_csv(r'C:\Users\Chinmay\Downloads\HW 2 Problem 2 Data.csv')
knn_data.head(10)
```

Out[23]:

	X1	X2	Category
0	59	43	Yellow
1	60	24	Yellow
2	54	75	Blue
3	27	5	Red
4	96	73	Blue
5	51	43	Yellow
6	21	94	Red
7	100	19	Yellow
8	73	68	Blue
9	62	9	Yellow

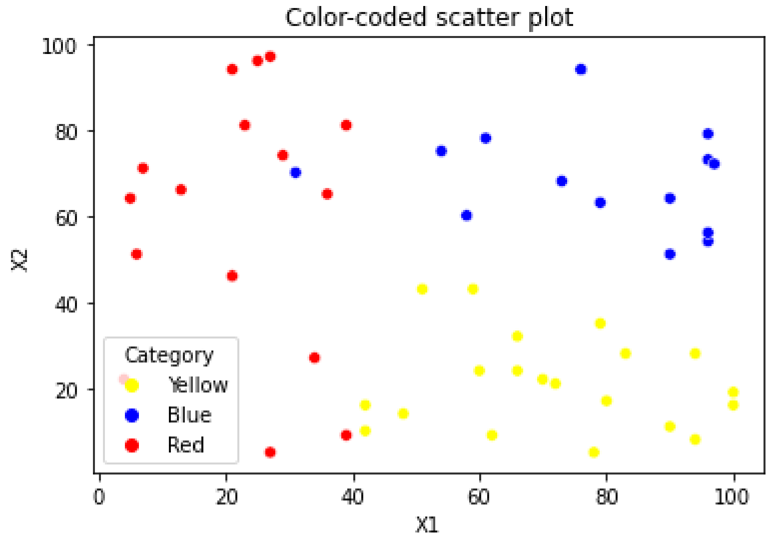
2b) Using Seaborn, create a color-coded scatterplot of the data (where each point is colored with its category color). For full credit, be sure to color the points correctly with yellow, blue, and red colors.

In [24]:

```
x1 = knn_data["X1"]
x2 = knn_data["X2"]
sns.scatterplot(data=knn_data, x=x1, y=x2, hue="Category", palette=["yellow","blue","red"]
```

Out[24]:

```
[Text(0.5, 1.0, 'Color-coded scatter plot')]
```



Now, we are going to create and assess predictions for the category of each observation using KNN-1 and KNN-3 algorithms. This means for each observation we are going to determine a prediction for its category color as if we didn't know what it was and then we will be checking to see if the prediction matches its actual category color.

The problem sub-parts below will step you through the process of doing this.

2c) Create a two-dimensional numpy array of size 50x50 that has the distance of each observation to every other observation. Use a Euclidean formula to calculate the distances. Display the first row of the distance table (this row will show the distance of each observation from the first observation)

In [113]:

```
distance_matrix = np.zeros((50,50))
for i in range(50):

    distance_matrix[i] = (np.sqrt(np.sum((knn_data[['X1','X2']].values - knn_data[['X1','X2
```

In [114]:

```
distance_matrix[0]
```

Out[114]:

```
array([ 0.        , 19.02629759, 32.38826948, 49.67896939, 47.63402146,
        8.        , 63.60031446, 47.50789408, 28.65309756, 34.13209633,
       32.01562119, 62.76941931, 57.93962375, 17.02938637, 38.07886553,
       33.42154993, 37.44329045, 39.44616585, 51.623638  , 28.3019434 ,
       42.48529157, 58.87274412, 38.60051813, 28.28427125, 31.82766093,
       43.13930922, 21.54065923, 44.55333882, 23.70653918, 38.11823711,
       31.01612484, 59.05929224, 38.89730068, 35.05709629, 29.68164416,
       13.03840481, 20.24845673, 42.94182111, 49.09175083, 39.2173431 ,
       51.42956348, 53.60037313, 62.96824597, 52.34500931, 31.90611227,
       25.55386468, 49.49747468, 37.12142239, 47.80167361, 53.75872022])
```

2d) Now create a dataframe with 50 rows (one per observation) and 3 columns labeled 'nn1_cat', 'nn2_cat', and 'nn3_cat'. Populate the dataframe with the category color for the first, second, and third nearest neighbor for each observation.

Display the first ten rows of this dataframe

In [35]:

```
indices = np.argsort(distance_matrix,1)

nn1_cat = []
nn2_cat = []
nn3_cat = []


nn1_idx = indices[:, 1]
nn2_idx = indices[:, 2]
nn3_idx = indices[:, 3]

for idx in nn1_idx:
    nn1_cat.append(knn_data['Category'].iloc[idx])

for idx in nn2_idx:
    nn2_cat.append(knn_data['Category'].iloc[idx])

for idx in nn3_idx:
    nn3_cat.append(knn_data['Category'].iloc[idx])


nn =pd.DataFrame(data=list(zip(nn1_cat, nn2_cat, nn3_cat)), columns=['nn1', 'nn2', 'nn3'])
nn.head(10)
```

Out[35]:

	nn1	nn2	nn3
0	Yellow	Yellow	Blue
1	Yellow	Yellow	Yellow
2	Blue	Blue	Red
3	Red	Yellow	Yellow
4	Blue	Blue	Blue
5	Yellow	Blue	Yellow
6	Red	Red	Red
7	Yellow	Yellow	Yellow
8	Blue	Blue	Blue
9	Yellow	Yellow	Yellow

2e) Create a Pandas series nn1_preds that has the prediction for each observation using a KNN1 algorithm. Display the first 10 rows of the series.

In [136]:

```
nn1_preds = nn['nn1']
nn1_preds.head(10)
```

Out[136]:

```
0    Yellow
1    Yellow
2      Blue
3       Red
4      Blue
5    Yellow
6       Red
7    Yellow
8      Blue
9    Yellow
Name: nn1, dtype: object
```

2f) Calculate the misclassification rate for the KNN1 algorithm on this dataset

In [149]:

```
misclassification_rate_knn1 = np.where(knn_data['Category'] != nn['nn1'])[0].size/50
misclassification_rate_knn1
```

Out[149]:

```
0.12
```

2g) Create a Pandas series nn3_preds that has the prediction for each observation using a KNN3 algorithm. Display the first 10 rows of the series. (If the three nearest neighbors all have different color categories, use the first nearest neighbor category as the prediction)

In [150]:

```
nn3_preds = []
for i in range(50):
    a,a_ct = np.unique(nn.values[i],return_counts=True)
    max_index = np.argmax(a_ct)
    if max(a_ct) == 1:
        nn3_preds.append(nn['nn1'][i])
    else:
        nn3_preds.append(a[max_index])
```

In [151]:

```
nn3_preds = pd.Series(nn3_preds)
nn3_preds.head(10)
```

Out[151]:

```
0    Yellow
1    Yellow
2      Blue
3    Yellow
4      Blue
5    Yellow
6       Red
7    Yellow
8      Blue
9    Yellow
dtype: object
```

2h) Calculate the misclassification rate for the KNN3 algorithm on this dataset

In [152]:

```
misclassification_rate_knn3 = np.where(knn_data['Category'] != nn3_preds)[0].size/50
misclassification_rate_knn3
```

Out[152]:

```
0.08
```

In [ ]:

```

```