Q775 Assignment 1

Name: Ahana Malhotra

Student ID:400380436

Date of Submission: 10 March 2023

1. **Develop the CER prediction model used in class to force all the coefficients of the function to be non-negative. Compare the resulting residuals with those found in class and comment on the results.**

   CER prediction model is developed and two cases are performed. In the first case, all the coefficients of the functions to be non-negative are enforced and in second case all the coefficients of the functions can be non-negative. When we analyzed the residuals, we found when non-negative coefficients are not enforced, then the value of residuals have decreased, which indicates better accuracy of prediction of total cost.

   Below sheet stating 'LP-non negative' means non-negative coefficients are enforced and in others they are not.

Chart 1

	A	B	C	D	E	F	G	H	I	J
2	Job	x1	x2	x3	x4	x5	TC			
3		# of poles	wire (100 ft)	# ofcrossarms	# of insulators	# of guy wires and gurads	Total cost ($)			
4	1	1	4	1	2	1	560			
5	2	3	10	3	6	1	3080			
6	3	4	24	8	12	2	2520			
7	4	1	5	2	3	0	630			
8	5	3	12	3	12	1	1750			
9	6	3	12	3	8	1	1890			
10	7	2	10	4	6	0	1120			
11	8	4	12	8	12	0	1960			
12										
13	Variables									
14	a1+	a1-	a2+	a2-	a3+	a3-	a4+	a4-	a5+	a5-
15	618.3333333	0	46.6666667	0	0	81.66666667	0	35	0	0
16	eta1	eta2	eta3	eta4	eta5	eta6	eta7	eta8		
17	93.33333333	0	0	0	0	0	46.6666667	0		
18	rho1	rho2	rho3	rho4	rho5	rho6	rho7	rho8		
19	0	1213.33333	0	46.6666667	0	0	0	0		
20	Constraints									
21	-4.26326E-14									
22	0						residuals	LP	Least Square	
23	0						1	93.3333	291.380701	
24	6.39488E-14						2	1213.33	418.740304	
25	2.27374E-13						3	0	97.1269004	
26	0						4	46.6667	23.1957803	
27	1.42109E-14						5	0	160.806702	
28	2.27374E-13						6	0	482.420105	
29							7	46.6667	93.6084393	
30	Objective						8	0	56.1216258	
31	1400									

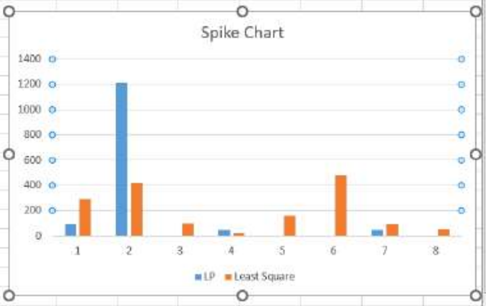
$$\min \sum_{i=1}^{m} (\eta_i + \rho_i)$$

no fixed cost

$$\text{s/t} : \quad c_i - (a_1 x_i - a_2 x_i + \underbrace{b_1 - b_2}_{=0 \; =0}) + \eta_i - \rho_i = 0; \;\; \forall i$$

$$a_1, a_2, b_1, b_2 \geq 0$$

$$\rho_i, \eta_i \geq 0$$



LP_non-negative | LP_zero | Regression_non-negative | Regression_zero | Correlation | DATA

M22

SUMMARY OUTPUT

Regression Statistics	
Multiple R	0.99039
R Square	0.98088
Adjusted F	0.62205
Standard I	424.546
Observati	8

ANOVA

	df	*SS*	*MS*	*F*	*gnificance F*
Regressior	5	2.8E+07	5547436	30.7782	0.03177
Residual	3	540718	180239		
Total	8	2.8E+07			

	Coefficients	*andard Err*	*t Stat*	*P-value*	*Lower 95%*	*Upper 95%*	*ower 95.0%*	*pper 95.0%*
Intercept	0	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A	#N/A
# of poles	1347.49	349.129	3.85959	0.03074	236.411	2458.58	236.411	2458.58
wire (100	51.3869	88.6669	0.57955	0.60287	-230.79	333.565	-230.79	333.565
# ofcrossa	-205.1	164.13	-1.2496	0.30004	-727.44	317.231	-727.44	317.231
# of insula	-195.81	102.728	-1.9061	0.15272	-522.73	131.121	-522.73	131.121
# of guy w	-104.95	467.106	-0.2247	0.83667	-1591.5	1381.59	-1591.5	1381.59

RESIDUAL OUTPUT

Observatio	*ted Total c*	*Residuals*
1	851.381	-291.38
2	2661.26	418.74
3	2422.87	97.1269
4	606.804	23.1958
5	1589.19	160.807
6	2372.42	-482.42
7	1213.61	-93.608
8	2016.12	-56.122

LP_non-negative | LP_zero | **Regression_non-negative** | Regression_zero | Correlation | DATA

	# of poles	wire (100 ft)	# ofcrossarms	# of insulators	# of guy wires and gurads	Total cost ($)
1	1	4	1	2	1	560
2	3	10	3	6	1	3080
3	4	24	8	12	2	2520
4	1	5	2	3	0	630
5	3	12	3	12	1	1750
6	3	12	3	8	1	1890
7	2	10	4	6	0	1120
8	4	12	8	12	0	1960

$$\min \sum_{i=1} (\eta_i + \rho_i)$$

no fi cost

$=0$ $=0$

$$\text{s/t}: \quad c_i - (a_1 x_i - a_2 x_i + b_1 - b_2) + \eta_i - \rho_i = 0; \ \forall i$$

$$a_1, a_2, b_1, b_2 \geq 0$$

$$\rho_i, \eta_i \geq 0$$

Variables

a1+	a1-	a2+	a2-	a3+	a3-	a4+	a4-	a5+	a5-
560	0	0	0	0	0	0	0	0	0
eta1	eta2	eta3	eta4	eta5	eta6	eta7	eta8		
0	0	0	0	0	0	0	0		
rho1	rho2	rho3	rho4	rho5	rho6	rho7	rho8		
0	1400	280	70	70	210	0	-280		

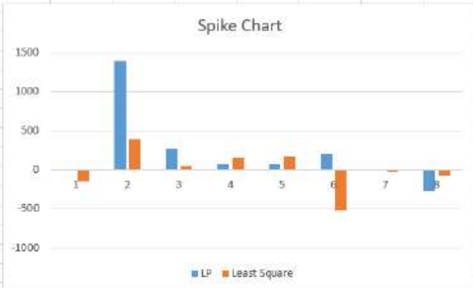
Constraints

0
0
0
0
0
0
0
0

residuals	LP	Least Square
1	0	-149.876497
2	1400	389.610778
3	280	49.9588323
4	70	154.906437
5	70	169.825973
6	210	-509.477919
7	0	-32.5243263
8	-280	-72.4232784

Objective

1750



LP_non-negative | LP_zero | Regression_non-negative | Regression_zero | Correlation | DATA

L18

SUMMARY OUTPUT

Regression Statistics	
Multiple R	0.953574
R Square	0.909304
Adjusted R	0.682563
Standard E	497.7263
Observatic	8

ANOVA

	df	*SS*	*MS*	*F*	*ignificance F*
Regressior	5	4967425	993484.9	4.010329	0.211553
Residual	2	495463	247731.5		
Total	7	5462888			

	Coefficients	*tandard Erro*	*t Stat*	*P-value*	*Lower 95%*	*Upper 95%*	*ower 95.0%*	*pper 95.0%*
Intercept	-202.337	473.4054	-0.42741	0.7107	-2239.24	1834.562	-2239.24	1834.562
# of poles	1448.538	472.6765	3.064543	0.092019	-585.225	3482.3	-585.225	3482.3
wire (100 f	59.37032	105.6155	0.562136	0.630621	-395.057	513.7972	-395.057	513.7972
# ofcrossa	-226.74	198.9694	-1.13957	0.372555	-1082.84	629.356	-1082.84	629.356
# of insula	-204.826	122.2707	-1.67519	0.235883	-730.914	321.2622	-730.914	321.2622
# of guy w	-137.413	552.8655	-0.24855	0.826904	-2516.2	2241.375	-2516.2	2241.375

RESIDUAL OUTPUT

Observatior	*ted Total c*	*Residuals*
1	709.8765	-149.876
2	2690.389	389.6108
3	2470.041	49.95883
4	475.0936	154.9064
5	1580.174	169.826
6	2399.478	-509.478
7	1152.524	-32.5243
8	2032.423	-72.4233

LP_non-negative | LP_zero | Regression_non-negative | **Regression_zero** | Correlation | DATA

	# of poles	wire (100 ft)	# ofcrossarms	# of insulators	# of guy wires and gurads	Total cost ($)
# of poles	1					
wire (100 ft)	0.818322323	1				
# ofcrossarms	0.826767382	0.789436128	1			
# of insulators	0.912732554	0.787398955	0.77770072	1		
# of guy wires and gurads	0.38271893	0.639516355	0.15430335	0.310349551	1	
Total cost ($)	0.823202005	0.65870618	0.510091751	0.599221419	0.500268216	1

LP_non-negative | LP_zero | Regression_non-negative | Regression_zero | **Correlation** | DATA

2. **For this question you will use the data 'general_classification_data.csv'**

   (a) Use the Linear Discriminant Analysis approach (e.g., from Python SKLearn package or code it based on equations discussed in class) to develop a discriminant line. Comment on the results achieved and compare them to those achieved by the linear programming method we discussed in class (and implemented in Python).

   Developed discriminant line from Linear Discriminant Analysis approach cannot distinguish two classes as these classes because LDA assume that the covariance matrix is same and it tries to find the maximum linear separation between classes while minimizing their class variance. So, in this case LDA could not find linear combination of features which leads to distinguishing between two classes.

   However, linear programming could separate the two classes successfully, because it was able to separate the boundaries by finding optimal linear boundary. As both the techniques have different assumptions and result varies according to the input data. But linear programming can solve wide range of optimization problems as compared to LDA.

Code

Linear Discriminant Analysis

In 66

```python
import pandas as pd
import numpy as np
```

In 67

```python
data = pd.read_csv("./general_classification_data.csv", index_col=0)
```

In 68

```python
data.head()
```

Out 68

5 rows 5 rows × 3 columns

Object	Score1	Score2	Class
1	0.2	0.5	1
2	0.2	0.8	1
3	0.3	0.4	1
4	0.3	0.7	1
5	0.4	0.3	1

In 69

```python
#import os
#cwd = os.getcwd()
#print(cwd)

```

In 71

```python
X = data[['Score1', 'Score2']]
```

In 72

```python
X.head()
```

Out 72

5 rows × 2 columns

Object	Score1	Score2
1	0.2	0.5
2	0.2	0.8
3	0.3	0.4
4	0.3	0.7
5	0.4	0.3

In 73

```python
print(type(X))
print(X.shape)
```

```
<class 'pandas.core.frame.DataFrame'>
(20, 2)
```

In 74

```python
y = data.Class
y.head()
```

Out 74

5 rows Length: 5, dtype: int64

Object	Class
1	1
2	1
3	1
4	1
5	1

In 75

```
print(type(y))
print(y.shape)
```

```
<class 'pandas.core.series.Series'>
(20,)
```

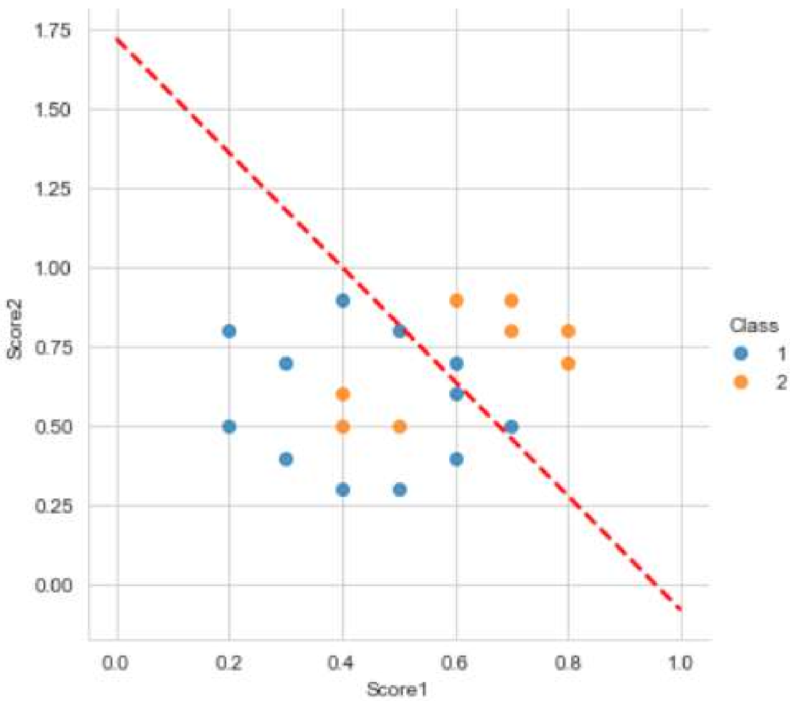
In 76

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

lda = LinearDiscriminantAnalysis()
lda.fit(X, y)

coef = lda.coef_[0]
intercept = lda.intercept_

```

In 77

```
import seaborn as sns
import matplotlib.pyplot as plt
```

```python
sns.lmplot(x='Score1', y='Score2', data=data, hue='Class', fit_reg=False)

x_vals = np.array([0, 1])
y_vals = -(coef[0] * x_vals + intercept) / coef[1]

plt.plot(x_vals, y_vals, 'r--', linewidth=2)

plt.show()
```



(b) Use the appropriate Neural Network code (MATLAB or Python) to obtain a neural network classifier for the same data set. Compare this result to that obtained in part 2(a).

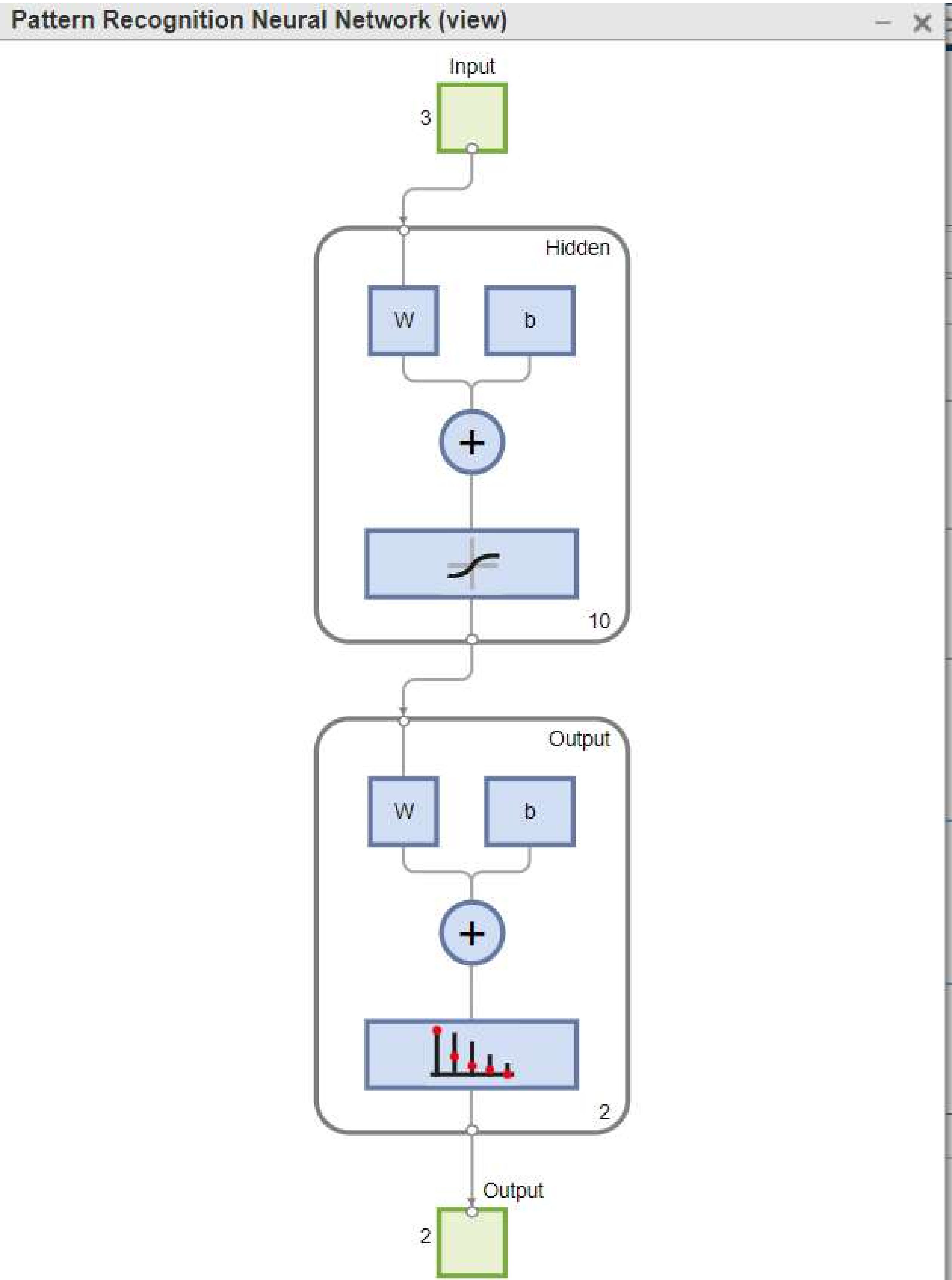
Feedforward artificial neural network code has been developed in MATLAB. It classifies the dataset into two classes with accuracy of 100%. Confusion matrix has been analyzed to check the accuracy. In the confusion matrix, we can analyze that in training data, it predicted the data with 100% accuracy and so in validation and test data.

In part 2a, accuracy of prediction of classes using linear discrimination analysis was extremely low, but linear programming method helped us getting better classification among classes because it was able to find optimal linear boundary. But if data would have been non-linear, it would be difficult for linear programming as well to separate the classes and find optimal solution. So in linear and non-linear input cases, neural network gives us best accuracy and optimal results.

In case data becomes more complex, then neural network classifier will work the best among the three methods discussed in this question.

```matlab
%% Artificial Neural Network (ANN)
clear; clc;


%% Loading the general classification data, with the classes at the last column

A = load('general_classification_data.txt');

% C is the matrix of the feature vectors
C = A(1:end,1:end-1)';

%% Create matrix for classes (targets)
T=zeros(size(unique(A(:, end)),1),size(A,1));

for i=1:size(A,1)
    T(A(i,end),i)=1;   %create identity vectors to reflect class type
end

%% Initializing a neural network 'net'
net = feedforwardnet;
net = configure(net, C, T);    % net - NN, C-feature vector, T-target

%% Setting hidden layer
hiddenLayerSize = 10; % Setting the number of hidden layers to 10
net = patternnet(hiddenLayerSize); % Pattern recognition network

%% Split data for training, validation and testing
net.divideParam.trainRatio = 0.7; % Ratio of training data is 70%
net.divideParam.valRatio = 0.2; % Ratio of validation data is 20%
net.divideParam.testRatio = 0.1; % Ratio of testing data is 10%
```

```
%% Training the network
[net, tr] = train(net, C, T);    % the resultingmodel is the output net

outputs = net(C);

%% Computing the classification errors
errors = gsubtract(T, outputs);
performance = perform(net, T, outputs);
view(net)
```



Pattern Recognition Neural Network (view)
Input
3
Hidden
W
b
10
Output
W
b
2
Output
2

Neural Network Training (09-Mar-2023 17:35:19)

Network Diagram

Training Results

Training finished: Reached minimum gradient

Training Progress

Unit	Initial Value	Stopped Value	Target Value
Epoch	0	24	1000
Elapsed Time	-	00:00:00	-
Performance	0.206	2.74e-07	0
Gradient	0.347	6.86e-07	1e-06
Validation Checks	0	1	6

Training Algorithms

Data Division: Random dividerand

Training: Scaled Conjugate Gradient trainscg

Performance: Cross Entropy crossentropy

Calculations: MEX

Training Plots

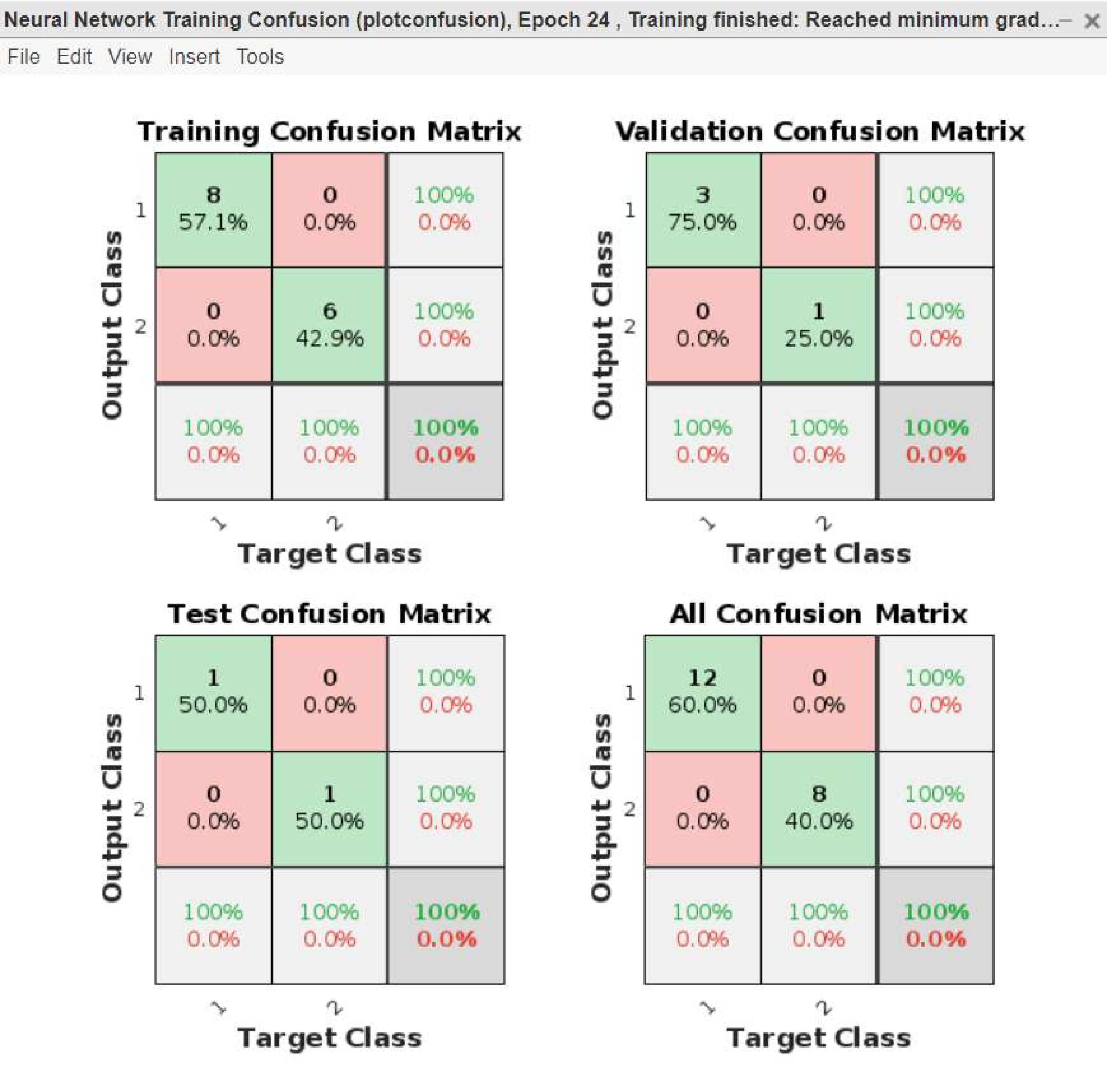
Performance

Training State

Error Histogram

Confusion

Receiver Operating Characteristic



Neural Network Training Confusion (plotconfusion), Epoch 24 , Training finished: Reached minimum grad...
File Edit View Insert Tools
Training Confusion Matrix
Validation Confusion Matrix
Test Confusion Matrix
All Confusion Matrix
Output Class
Target Class
8 57.1%
0 0.0%
6 42.9%
3 75.0%
1 25.0%
1 50.0%
12 60.0%
8 40.0%
100% 0.0%

3. **Consider the data in Table 1: ‘dataforquestion03assignment01.csv‘ (uploaded in Avenue’s Assignment 01 folder).**

   (a) Use the linear programming classification method discussed in class to develop a neural network classifier that can be used for training the data in Table 1. Comment on the regions formed by the supermasks and foreign-object masks.

   This data is highly non-linear. Therefore, we could not classify data using linear programming classification properly. As we can see, it could only predict data with 50% accuracy. Supermask could only classify one set. As there was no foreign object or no overlap in classes, foreign object class could not classify properly.

Neural Network Design and Training with LP

In 866
```python
from pulp import *
```

In 867
```python
prob = LpProblem("Neural Network Design and Training with LP", LpMinimize)
probf = LpProblem("Neural Network Design and Training with LP", LpMinimize)
```

```
C:\Users\malhoa25\Desktop\Q775 Machine Learning\venv\lib\site-packages\pulp\p
the name. Converted to '_'
  warnings.warn("Spaces are not permitted in the name. Converted to '_'")
```

LP Supermask for Class A

In 868
```python
# Variables
rho1 = LpVariable("rho1", 0)
rho2 = LpVariable("rho2", 0)
rho3 = LpVariable("rho3", 0)
rho4 = LpVariable("rho4", 0)
rho5 = LpVariable("rho5", 0)
rho6 = LpVariable("rho6", 0)
```

```
rho7 = LpVariable("rho7", 0)
rho8 = LpVariable("rho8", 0)
rho9 = LpVariable("rho9", 0)
rho10 = LpVariable("rho10", 0)
v1 = LpVariable("v1")
v2 = LpVariable("v2")
v3 = LpVariable("v3")
v4 = LpVariable("v4")
v5 = LpVariable("v5")
v6 = LpVariable("v6")
```

In 869

```
# Objective
prob +=  rho1+rho2+rho3+rho4+rho5+rho6+rho7+rho8+rho9+rho10
```

In 870

```
# Constraints
prob += 0.04*v1 +1.00*v2 + 0.20*v3 + 0.20*v4 +1.00*v5 + v6 - rho1 ==0
prob += 0.04*v1 +0.64*v2 + 0.16*v3 + 0.20*v4 +0.80*v5 + v6 - rho2 ==0
prob += 0.09*v1 +0.81*v2 + 0.27*v3 + 0.30*v4 +0.90*v5 + v6 - rho3 ==0
prob += 0.16*v1 +0.64*v2 + 0.32*v3 + 0.40*v4 +0.80*v5 + v6 - rho4 ==0
prob += 0.16*v1 +1.00*v2 + 0.40*v3 + 0.40*v4 +1.00*v5 + v6 - rho5 ==0
prob += 0.64*v1 +0.16*v2 + 0.32*v3 + 0.80*v4 +0.40*v5 + v6 - rho6 ==0
prob += 0.64*v1 +0.04*v2 + 0.16*v3 + 0.80*v4 +0.20*v5 + v6 - rho7 ==0
prob += 0.81*v1 +0.09*v2 + 0.27*v3 + 0.90*v4 +0.30*v5 + v6 - rho8 ==0
prob += 1.00*v1 +0.04*v2 + 0.20*v3 + 1.00*v4 +0.20*v5 + v6 - rho9 ==0
prob += 1.00*v1 +0.16*v2 + 0.40*v3 + 1.00*v4 +0.40*v5 + v6 - rho10 ==0
prob += v1 + v2  == -1
```

In 871

```
prob.solve()
```

Out 871

```
1
```

In 872

```
# The status of the solution is printed to the screen
print("Status:", LpStatus[prob.status])
```

```
Status: Optimal
```

Solution

In 873

```
# Each of the variables is printed with it's resolved optimum value
for v in prob.variables():
    print(v.name, "=", v.varValue)
```

```
rho5 = 0.0
rho6 = 0.03
rho7 = 0.0
rho8 = 0.0175
rho9 = 0.0
v1 = -0.5
v2 = -0.5
```

```
v3 = -0.75
v4 = 1.05
v5 = 1.05
v6 = -0.59
```

Normalized

In 874
```python
from numpy import linalg as LA
```

In 875
```python
vs=[]
for v in range(10,16):
    vs.append(prob.variables()[v].varValue)
vs
```

Out 875
```
[-0.5, -0.5, -0.75, 1.05, 1.05, -0.59]
```

In 876
```python
vsnorm=[]
for v in range(10,16):
    vsnorm.append(prob.variables()[v].varValue/LA.norm(vs))
vsnorm
```

Out 876
```
[-0.26295402051409067,
 -0.26295402051409067,
```

```
 -0.3944310307711306,
 0.552203443079590,
 0.552203443079590,
 -0.310285744206627]
```

Foreign Object Mask for Class A

In 877

```python
# Variables
eta1 = LpVariable("eta1", 0)
eta2 = LpVariable("eta2", 0)
eta3 = LpVariable("eta3", 0)
rhof1 = LpVariable("rhof1", 0)
rhof2 = LpVariable("rhof2", 0)
rhof3 = LpVariable("rhof3", 0)

vf1 = LpVariable("vf1")
vf2 = LpVariable("vf2")
vf3 = LpVariable("vf3")
vf4 = LpVariable("vf4")
vf5 = LpVariable("vf5")
vf6 = LpVariable("vf6")
```

In 878

```python
# Objective
probf += eta2 #+ eta2 + eta3
```

In 879

```python

# Constraints
probf += 0.04 * vf1 + 1.00 * vf2 + 0.2 * vf3 + 0.20 * vf4 + 1.00 * vf5 + vf6 <= 0
probf += 0.04 * vf1 + 0.64 * vf2 + 0.16 * vf3 + 0.20 * vf4 + 0.80 * vf5 + vf6 <= 0
probf += 0.09 * vf1 + 0.81 * vf2 + 0.27 * vf3 + 0.30 * vf4 + 0.90 * vf5 + vf6 <= 0
probf += 0.16 * vf1 + 0.64 * vf2 + 0.32 * vf3 + 0.40 * vf4 + 0.80 * vf5 + vf6 <= 0
probf += 0.16 * vf1 + 1.00 * vf2 + 0.40 * vf3 + 0.40 * vf4 + 1.00 * vf5 + vf6 <= 0
probf += 0.64 * vf1 + 0.16 * vf2 + 0.32 * vf3 + 0.80 * vf4 + 0.40 * vf5 + vf6 <= 0
probf += 0.64 * vf1 + 0.04 * vf2 + 0.16 * vf3 + 0.80 * vf4 + 0.20 * vf5 + vf6 <= 0
probf += 0.81 * vf1 + 0.09 * vf2 + 0.27 * vf3 + 0.90 * vf4 + 0.30 * vf5 + vf6 <= 0
probf += 1.00 * vf1 + 0.04 * vf2 + 0.20 * vf3 + 1.00 * vf4 + 0.20 * vf5 + vf6 <= 0
probf += 1.00 * vf1 + 0.16 * vf2 + 0.40 * vf3 + 1.00 * vf4 + 0.40 * vf5 + vf6 <= 0
probf += 0.04 * vf1 + 0.04 * vf2 + 0.04 * vf3 + 0.20 * vf4 + 0.20 * vf5 + eta1 - rhof1 == 0
probf += 0.04 * vf1 + 0.16 * vf2 + 0.08 * vf3 + 0.20 * vf4 + 0.40 * vf5 + eta2 - rhof2 == 0
probf += 0.09 * vf1 + 0.09 * vf2 + 0.09 * vf3 + 0.30 * vf4 + 0.30 * vf5 + eta3 - rhof3 == 0
probf += rhof2 + rhof2 + rhof3 == 1

```

In 880

```python
probf.solve()
```

Out 880

```
1
```

In 881

```python
# The status of the solution is printed to the screen
print("Status_foreign:", LpStatus[probf.status])
```

In 882
```
#### Solution
# Each of the variables is printed with it's resolved optimum value
for vf in probf.variables():
    print(vf.name, "=", vf.varValue)
```

```
eta2 = 0.0
eta3 = 1.0
rhof1 = 0.0
rhof2 = 0.0
rhof3 = 1.0
vf1 = 0.0
vf2 = 0.0
vf3 = 0.0
vf4 = 0.0
vf5 = 0.0
vf6 = 0.0
```

In 883
```
from numpy import linalg as LA
```

In 884
```
vsf = []
for vf in range(2, 8):
    vsf.append(probf.variables()[vf].varValue)
```

```
vsf
vsfnorm = []
for vf in range(2, 8):
    vsfnorm.append(probf.variables()[vf].varValue / LA.norm(vsf))
vsfnorm
```

Out 108

```
[0.0, 0.0, 0.08454898264477458, 0.0, -0.7045748610097204, 0.7045748610097204]
```

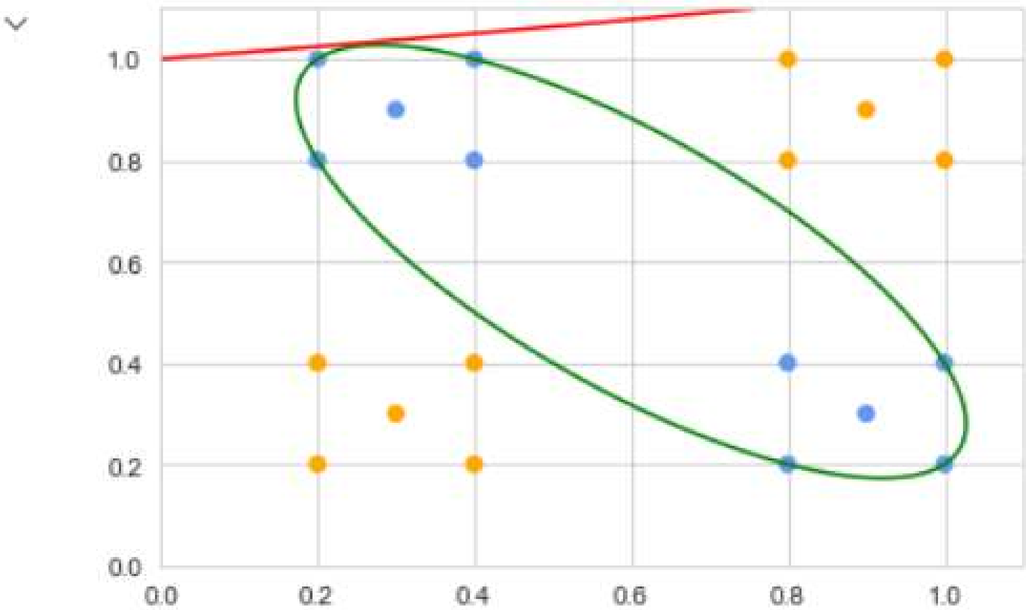
Plotting Supermask and Foreign Object Mask Class A

In 109

```
import seaborn as sns
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
def axes():
    plt.axhline(0, alpha=.1)
    plt.axvline(0, alpha=.1)

x, y = np.linspace(0,1.1,100), np.linspace(0,1.1,100)
X, Y = np.meshgrid(x,y)
Z1 = vsnorm[0]*X**2 + vsnorm[1]*Y**2 + vsnorm[2]*X*Y + vsnorm[3]*X + vsnorm[4]*Y + vsnorm[5]
Z2 = vsfnorm[0]*X**2 + vsfnorm[1]*Y**2 + vsfnorm[2]*X*Y + vsfnorm[3]*X + vsfnorm[4]*Y + vsfnorm[5]
plt.figure()
cp1 = plt.contour(X, Y, Z1,[0],colors='g')
```

```python
cp2 = plt.contour(X, Y, Z2,[0],colors='r')
score1=[]
score2=[]
color=[]
import pandas as pd
data = pd.read_csv("./data_for_question_03_assignment_01.csv", index_col=0)
for i in range(1,21):
    score1.append(data.Score1[i])
    score2.append(data.Score2[i])
    if i< 11:
        color.append(1)
    else:
        color.append(2)
import matplotlib.colors as cl
cmap = cl.LinearSegmentedColormap.from_list("", ["cornflowerblue","orange"])
plt.scatter(score1,score2,c=color, cmap=cmap)
```

```
Out 109        <matplotlib.collections.PathCollection at 0x1e75f3de400>
```



1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

(b) Graph (plot) the data set and comment on the appropriateness of the use of linear discriminant function for the separation of the two classes.

Given data is non linear and complex data, therefore it is inappropriate to use linear discriminant function. As this will not separate the classes properly.

Linear Discriminant Analysis

In 13
```
import pandas as pd
import numpy as np
```

In 14
```
data = pd.read_csv("./data_for_question_03_assignment_01.csv", index_col=0)
```

In 15
```
data.head()
```

Out 15

5 rows 5 rows × 3 columns

Object	Score1	Score2	Class
1	0.2	1.0	A
2	0.2	0.8	A
3	0.3	0.9	A
4	0.4	0.8	A
5	0.4	1.0	A

In 17

```python
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
```

In 18

```python
X = data[['Score1', 'Score2']]
```

In 19

```python
X.head()
```

Out 19

5 rows 5 rows × 2 columns

Object	Score1	Score2
1	0.2	1.0
2	0.2	0.8
3	0.3	0.9
4	0.4	0.8
5	0.4	1.0

In 20

```python
print(type(X))
print(X.shape)
```

```
<class 'pandas.core.frame.DataFrame'>
(20, 2)
```

In 21

```python
y = data.Class
y.head()
```

Out 21

5 rows   Length: 5, dtype: object

Object	Class
1	A
2	A
3	A
4	A
5	A

In 22

```
print(type(y))
print(y.shape)
```

```
<class 'pandas.core.series.Series'>
(20,)
```

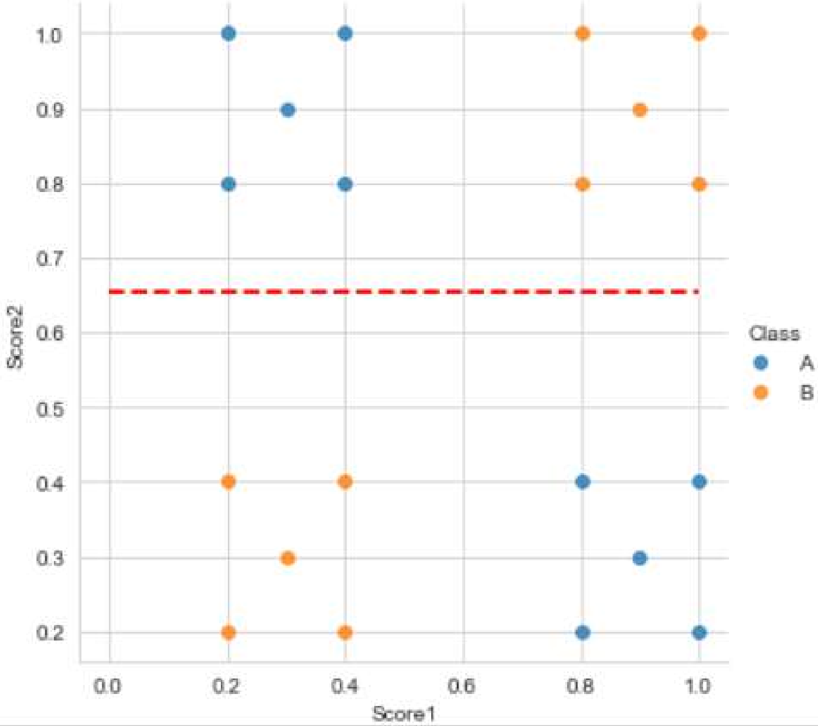
In 23

```
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

In 24

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

lda = LinearDiscriminantAnalysis()
lda.fit(X, y)

coef = lda.coef_[0]
intercept = lda.intercept_
```

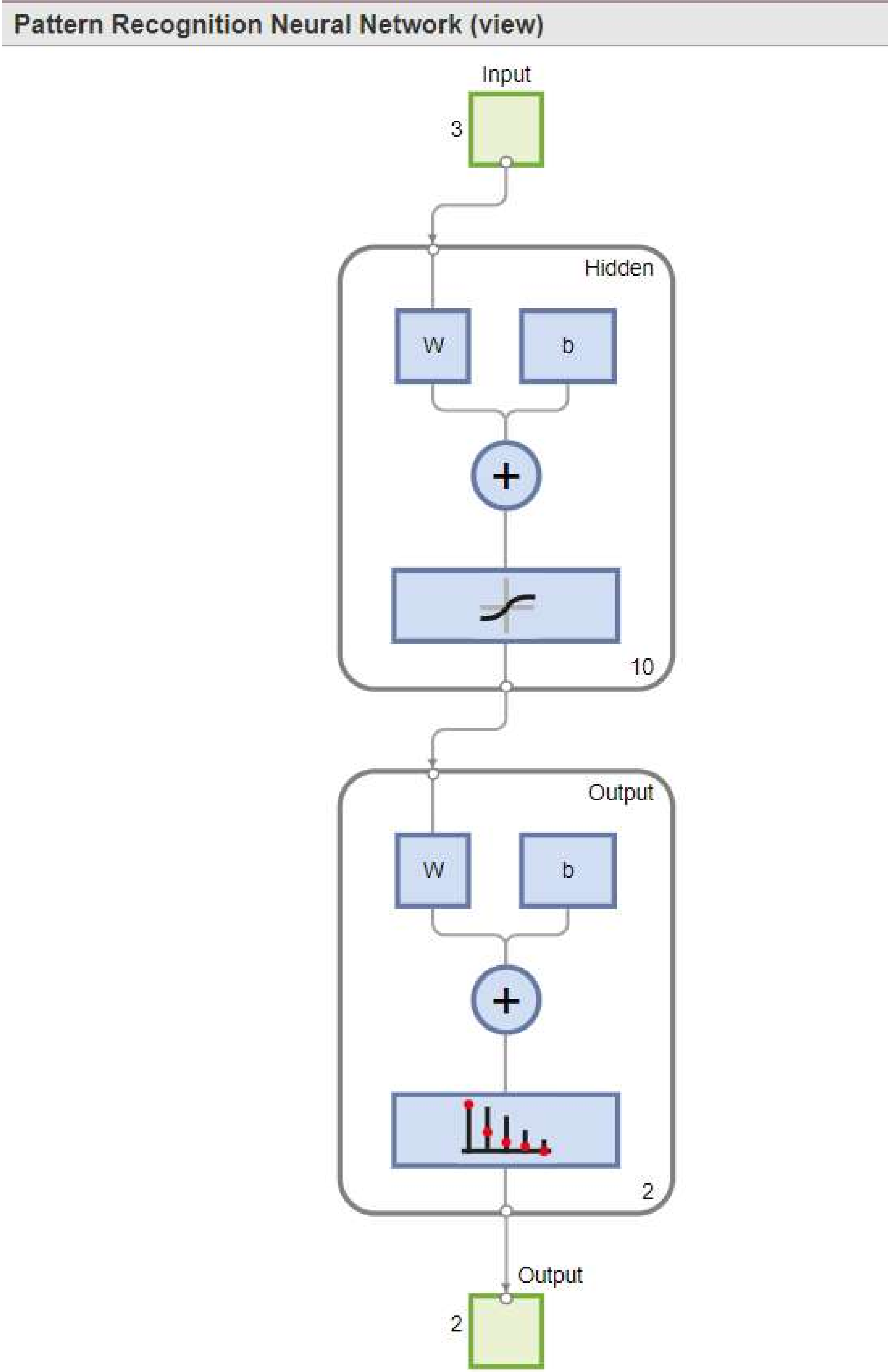
```python
import matplotlib.pyplot as plt

sns.lmplot(x='Score1', y='Score2', data=data, hue='Class', fit_reg=False)

x_vals = np.array([0, 1])
y_vals = -(coef[0] * x_vals + intercept) / coef[1]

plt.plot(x_vals, y_vals, 'r--', linewidth=2)

plt.show()
```



(c)Use the appropriate Neural Network code (MATLAB or Python) to obtain a neural network classifier for the same data set. Compare this result to that obtained in part 3(a).

Feedforward artificial neural network is used to predict the classification of the data. Referring to the confusion matrix, it shows that it has predicted the data with 100% accuracy. But when classification is done using linear programming or linear discrimination method, the these methods could not classify the objects with proper boundaries.

```matlab
clear; clc;
%% Loading the data_for_question_03_assignment_01 data, with the classes at the last column

A = load('ans3c_data.txt');

% C is the matrix of the feature vectors
C = A(1:end,1:end-1)';
%% Create matrix for classes (targets)
T=zeros(size(unique(A(:, end)),1),size(A,1));

for i=1:size(A,1)
    T(A(i,end),i)=1;   %create identity vectors to reflect class type
end
%% Initializing a neural network 'net'
net = feedforwardnet;
net = configure(net, C, T);    % net - NN, C-feature vector, T-target
%% Setting hidden layer
hiddenLayerSize = 10; % Setting the number of hidden layers to 10
net = patternnet(hiddenLayerSize); % Pattern recognition network
%% Split data for training, validation and testing
net.divideParam.trainRatio = 0.7; % Ratio of training data is 70%
net.divideParam.valRatio = 0.2; % Ratio of validation data is 20%
net.divideParam.testRatio = 0.1; % Ratio of testing data is 10%
%% Training the network
[net, tr] = train(net, C, T);    % the resultingmodel is the output net

outputs = net(C);
%% Computing the classification errors
errors = gsubtract(T, outputs);
performance = perform(net, T, outputs);
view(net)
```


Pattern Recognition Neural Network (view)
Input
3
Hidden
W
b
10
Output
W
b
2
Output
2

Neural Network Training (09-Mar-2023 19:18:15)

Network Diagram

Training Results

Training finished: Reached minimum gradient

Training Progress

Unit	Initial Value	Stopped Value	Target Value
Epoch	0	25	1000
Elapsed Time	-	00:00:00	-
Performance	0.828	3.03e-07	0
Gradient	0.593	6.95e-07	1e-06
Validation Checks	0	0	6

Training Algorithms

Data Division: Random dividerand

Training: Scaled Conjugate Gradient trainscg

Performance: Cross Entropy crossentropy

Calculations: MEX

Training Plots

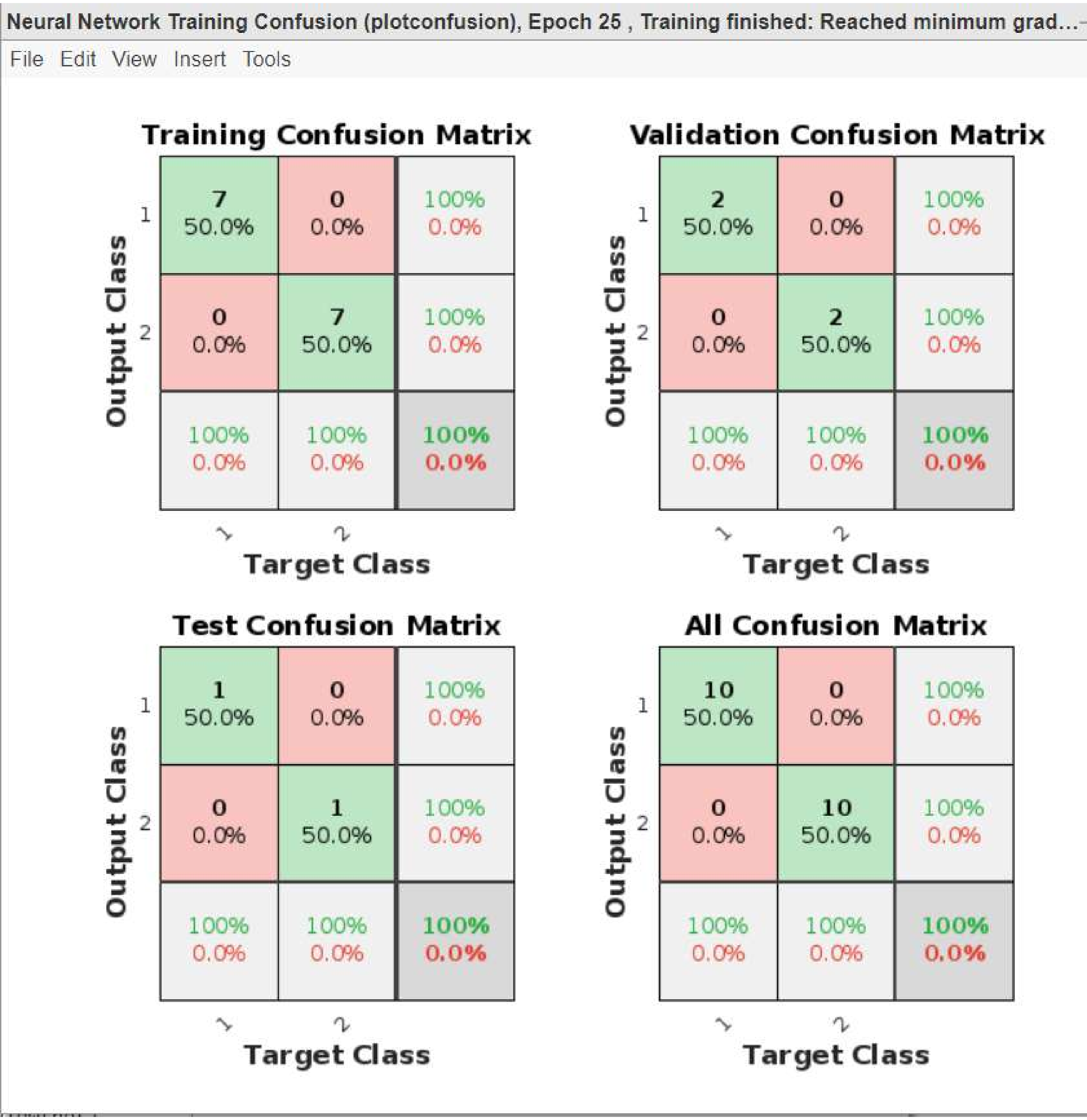
Performance

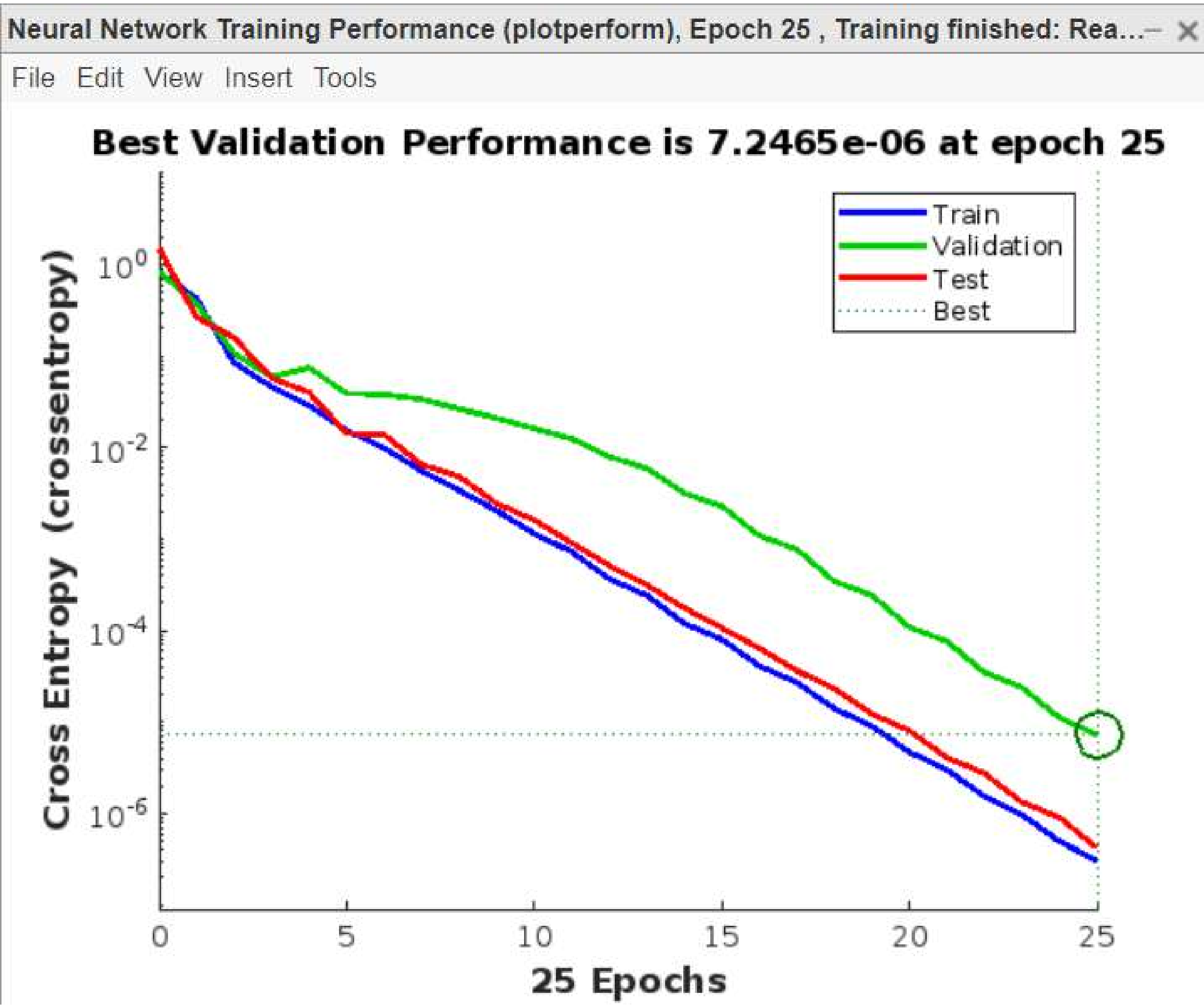
Training State

Error Histogram

Confusion

Receiver Operating Characteristic


Neural Network Training Confusion (plotconfusion), Epoch 25 , Training finished: Reached minimum grad...
File Edit View Insert Tools
Training Confusion Matrix
Validation Confusion Matrix
Test Confusion Matrix
All Confusion Matrix
7 50.0%
0 0.0%
0 0.0%
7 50.0%
2 50.0%
2 50.0%
1 50.0%
1 50.0%
10 50.0%
10 50.0%
100% 0.0%
Output Class
Target Class
1
2


Neural Network Training Performance (plotperform), Epoch 25 , Training finished: Rea...
File Edit View Insert Tools
Best Validation Performance is 7.2465e-06 at epoch 25
Train
Validation
Test
Best
10^{0}
10^{-2}
10^{-4}
10^{-6}
Cross Entropy (crossentropy)
0
5
10
15
20
25
25 Epochs

(d) Assuming that we do not know the data classes and assignments, using the graph in part 3(b) how many clusters would you suspect existed and for which objects? What is the basis of your clustering?

When we do not know the data classes and assignments, then we can use elbow method which is a graphical tool to estimate clusters. Using kmeans model, elbow curve is formed. And we can see, the elbow is formed at $k = 2$, showing that given data is clustered in total two groups.

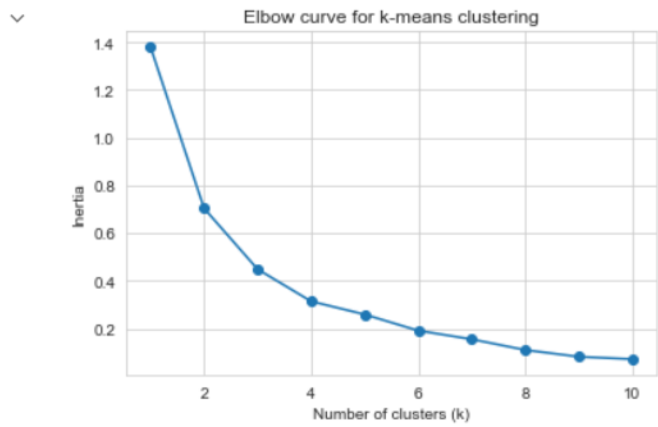


Figure 1: Elbow Curve showing k=2

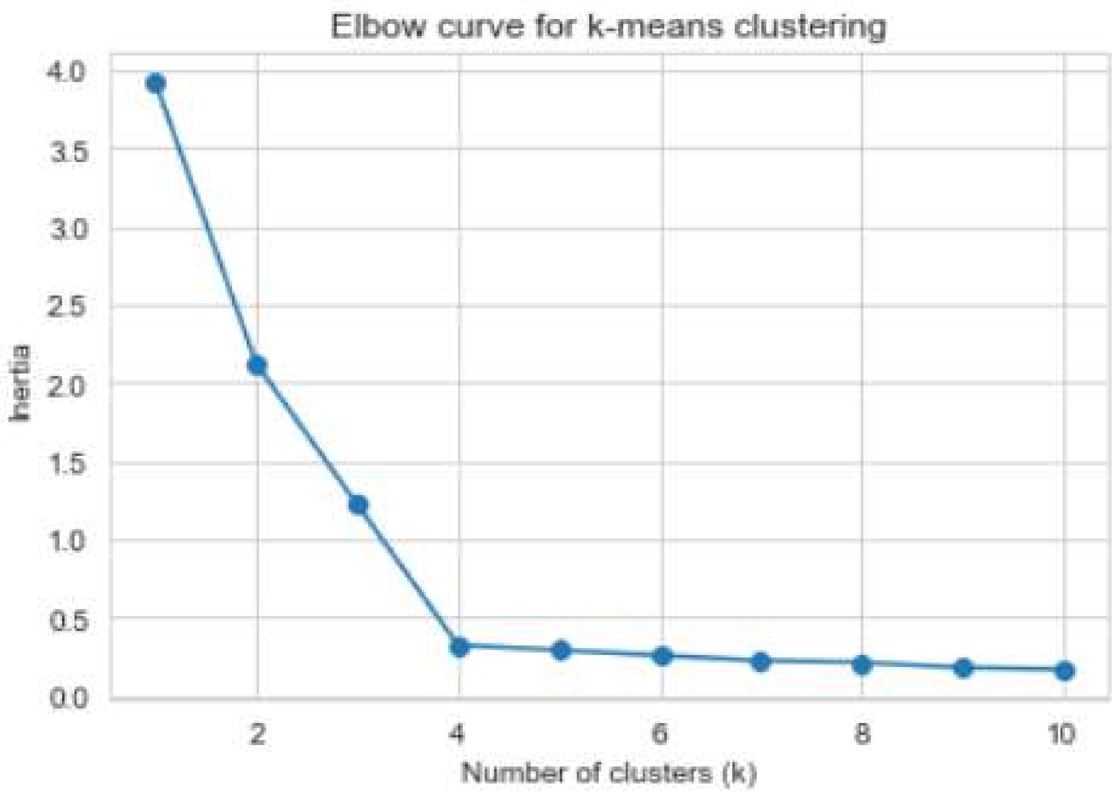
(e) Develop a clustering model and use it to find the optimal number of clusters and their objects.

Using elbow method, optimal number of clusters and objects are formed.

In 3

```python
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# read data from CSV file
data = pd.read_csv('data_for_question_03_assignment_01.csv', index_col='Object')

# drop the Class column as it's not needed for clustering
X = data.drop('Class', axis=1)

# perform clustering for different values of k`
inertias = []
k_values = range(1, 11)
for k in k_values:
    model = KMeans(n_clusters=k)
    model.fit(X)
    inertias.append(model.inertia_)

# plot the elbow curve to find the optimal number of clusters
plt.plot(k_values, inertias, '-o')
plt.xlabel('Number of clusters (k)')
plt.ylabel('Inertia')
plt.title('Elbow curve for k-means clustering')
plt.show()
```

```
warnings.warn(
```



In 4

```python
# fit the KMeans model with k=2
model = KMeans(n_clusters=2)
model.fit(X)

# print the cluster labels
labels = pd.Series(model.labels_, index=X.index)
print(labels)
```

```
Object
1     1
2     1
3     1
4     1
5     1
6     0
7     0
8     0
9     0
10    0
11    1
12    1
13    1
14    1
15    1
16    0
17    0
18    0
19    0
20    0
dtype: int32
```

4. **In signed network/graph, each edge (link) is labelled as either positive or negative to represent similarity or dissimilarity between connecting nodes (vertices), respectively. Correlation Clustering (CC) problem (Bansal et al., 2004) is a well-known graph/network partitioning problem that finds clusters in given signed network. The integer linear programming programing (ILP) formulation of the CC problem for a general weighted signed network** $G(V,E)$ **can be defined as follows (Demaine et al., 2006):**

$$\min \sum_{(i,j)\in E^+} w_{ij}x_{ij} + \sum_{(i,j)\in E^-} w_{ij}(1-x_{ij})$$

$$\begin{aligned} s.t. \quad & x_{ij} + x_{jk} \geq x_{ik} && ;\forall\, i,j,k \in V \\ & x_{ij} = x_{jk}; && \forall\, i,j \in V \\ & x_{ij} \in \{0,1\}; && \forall\, i,j \in V \end{aligned}$$

(a) Implement the above ILP formulation using any language in your choice and package (not in excel).

(b) Use your ILP implementation to identify clusters in the Slovene Parliamentary Political party's mutual relationship data set ‘slovenepoliticalparties.csv‘ (Avenue's Assignment 01 folder). The description of this data can be found in here:

ILP formulation is done in Python and part2 a and b answer is shown below.

http://networkdata.ics.uci.edu/netdata/html/Stranke94.html .

```
In 121  from docplex.mp.model import Model
        import sys
        sys.path.insert(1, './network_importer')
        import network_importer as nimp
```

Read Network

```
In 122  G = nimp.networkx_read_weighted_network_from_csv('./test_net_assgn1.csv')
```

```
In 123  print(G.nodes)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
In 124  print(G.edges)
```

```
[(1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (1, 7), (1, 8), (1, 9), (1, 10), (2, 3), (2, 4), (2, 5), (2, 6), (2, 7), (2, 8), (2, 9), (2, 10), (3, 4), (3, 5), (3, 6), (3, 7), (3, 8), (3, 9), (3, 10), (4, 5), (4, 6), (4, 7), (4, 8), (4, 9), (4, 10), (5, 6), (5, 7), (5, 8), (5, 9), (5, 10), (6, 7), (6, 8), (6, 9), (6, 10), (7, 8), (7, 9), (7, 10), (8, 9), (8, 10), (9, 10)]
```

ILP Model : Decision variables

In 125
```python
# Index range
N = range(1, len(G.nodes) + 1)

# Model
ce = Model(name='clustering_editing')

# Decision variable
x = {(i, j): ce.binary_var(name='x_{0}_{1}'.format(i, j)) for i in N for j in N}
```

ILP Model: Objective Function

In 126
```python
# Objective function
LDC: int = 0        # total edge deletion cost
LIC: int = 0        # total edge insertion cost
for i in N:
    for j in N:
        if i < j:
            if G.has_edge(i, j)>0:
                w_ij = G.edges[i, j]['weight']

                LDC_this = ce.sum(x[i, j] * w_ij)
```

```
                LDC = LDC + LDC_this
            else:
                del_ij = 1

                LIC_this = ce.sum((1 - x[i, j]) * del_ij)
                LIC = LIC + LIC_this
```

In 127

```
# Total Cost (Objective function) = Link Deletion Cost (LDC) + Link Insertion Cost (LIC)
total_cost = LDC + LIC

 # Minimize all cost
ce.minimize(total_cost)
```

Constraint

In 128

```
# Constrain: Triangle inequality
for i in N:
    for j in N:
        for k in N:
            if i != j != k:
                # linear expression for constraint
                c1_this = ce.sum(x[i, j] + x[j, k] - x[i, k])

                # set constraint
                ce.add_constraint(c1_this >= 0)

# Constraint: Undirected link
```

```python
# Constraint: Undirected link
for i in N:
    for j in N:
        if i != j !=k:
            # linear expression for constraint
            c2_this = ce.sum(x[i, j] - x[j, k])

            # set constraint
            ce.add_constraint(c2_this == 0)
```

Solve Model

In 129

```python
# Solve model
solution = ce.solve()

if solution is None:
    print('- Model is infeasible')
else:
    print('Model found an optimal solution')

assert solution
```

```
Model found an optimal solution
```

In 130

```
print('Objective value: ', solution.objective_value)
```

```
Objective value:  -1881.0
```

Returing Clusters

In 131

```
sys.path.insert(1, './partitioning_for_ce')
import partitioning_for_ce as part
```

In 132

```
# Get clusters
P = []  # clusters partition set

for i in N:
    for j in N:
        if i < j:
            if solution.get_value(x[i, j]) >= 0:
                S = []
                S.append(i)
                S.append(j)
                P = part.get_partitions(P, S)

# Merging any two partitions that have common element(s)
P_final: list = []
```

```
for L in P:
    P_final = part.get_partitions(P_final, L)

```

In 133

```
print("Final Clusters:" , P_final)
```

```
Final Clusters: [[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]]
```

The given model second constraint should be $x_{ji} = x_{ij}$. So I have re-coded to get better results and form clusters.

Clustering Editing

In 46

```
import ...

sys.path.insert(1, './network_importer')
import network_importer as nimp


```

Read Network

In 47

```
### Read Network
G = nimp.networkx_read_weighted_network_from_csv('./test_net_assgn1.csv')
```

In 48

```
print(G.nodes)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

In 49

```
print(G.edges)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

In 49

```python
print(G.edges)
```

```
[(1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (1, 7), (1, 8), (1, 9), (1, 10), (2, 3), (2, 4), (2, 5), (2, 6), (2, 7), (2, 8), (2, 9), (2,
10), (3, 4), (3, 5), (3, 6), (3, 7), (3, 8), (3, 9), (3, 10), (4, 5), (4, 6), (4, 7), (4, 8), (4, 9), (4, 10), (5, 6), (5, 7), (5,
8), (5, 9), (5, 10), (6, 7), (6, 8), (6, 9), (6, 10), (7, 8), (7, 9), (7, 10), (8, 9), (8, 10), (9, 10)]
```

ILP Model : Decision variables

In 50

```python
# Index range
N = range(1, len(G.nodes) + 1)

# Model
ce = Model(name='clustering_editing')

# Decision variable
x = {(i, j): ce.binary_var(name='x_{0}_{1}'.format(i, j)) for i in N for j in N}
```

ILP Model: Objective Function

In 51

```python
# Objective function
LDC: int = 0        # total edge deletion cost
LIC: int = 0        # total edge insertion cost
for i in N:
    for j in N:
        if i < j:
            if G.has_edge(i, j)>0:
                w_ij = G.edges[i, j]['weight']

                LDC_this = ce.sum(x[i, j] * w_ij)
                LDC = LDC + LDC_this
            else:
                del_ij = 1

                LIC_this = ce.sum((1 - x[i, j]) * del_ij)
                LIC = LIC + LIC_this
```

In 52

```python
# Total Cost (Objective function) = Link Deletion Cost (LDC) + Link Insertion Cost (LIC)
total_cost = LDC + LIC

 # Minimize all cost
ce.minimize(total_cost)
```

Constraint

In 53

```python
# Constrain: Triangle inequality
for i in N:
    for j in N:
        for k in N:
            if i != j != k:
                # linear expression for constraint
                c1_this = ce.sum(x[i, j] + x[j, k] - x[i, k])

                # set constraint
                ce.add_constraint(c1_this >= 0)

# Constraint: Undirected link
for i in N:
    for j in N:
        if i != j:
            # linear expression for constraint
            c2_this = ce.sum(x[i, j] - x[j, i])

            # set constraint
            ce.add_constraint(c2_this == 0)
```

Solve Model

In 54

```
# Solve model
solution = ce.solve()

if solution is None:
    print('- Model is infeasible')
else:
    print('Model found an optimal solution')

assert solution
```

```
Model found an optimal solution
```

In 55

```
print('Objective value: ', solution.objective_value)
```

```
Objective value:  -4193.0
```

Returing Clusters

Add Code Cell | Add Markdown Cell

In 56

```
sys.path.insert(1, './week05/partitioning_for_ce')
import partitioning_for_ce as part
```

In 57

```
# Get clusters
P = []  # clusters partition set

for i in N:
    for j in N:
        if i < j:
            if solution.get_value(x[i, j]) == 0:
                S = []
                S.append(i)
                S.append(j)
                P = part.get_partitions(P, S)

# Merging any two partitions that have common element(s)
P_final: list = []
for L in P:
    P_final = part.get_partitions(P_final, L)

```

In 58

```
print("Final Clusters:" , P_final)
```

```
Final Clusters: [[2, 4, 5, 7, 10], [1, 3, 6, 8, 9]]
```

(c) Compare and discuss your result with the clusters reported in Figure 10 in Wu et al. (2016). URL: https://www.sciencedirect.com/science/article/abs/pii/S037843711500802X

Formulation 1: As per given in homework ILP formulation given in question is implemented in Python and only one cluster $[1,2,3,4,5,6,7,8,9,10]$ is formed. However in Figure 10 in Wu et al. (2016) clusters $[2,4,5,7,10]$ and $[1,3,6,8,9]$ are formed. Even though signed link network is formed in both the ILP formulation but in paper second constraint is $x_i j = x_j i$ instead of $x_i j = x_j k$ which is given in the question. To obtain clusters for the given question we should be either changing the objective or to add more constraint as a penalty if no clusters are formed. Currently, objective is just minimizing the cost of cluster editing and is not asking to form number of clusters. So its finding an optimal result with single cluster only.

Formulation 2: Where second constraint is corrected Cluster is obtained similar to the given in paper.

5. **For this question, use the BigQuery public data 'bigquery.public data.new.york.tlc.yellow.trips.2012' on the Google Cloud Platform**

   (a) Use the 'RAND()' to pull 1 in 100,000 records. This will result in approximately 1,793 records (1 in 100,000 records). Print the first 10 entries of the extracted data.

```
In 1372
CREDENTIAL_PATH = './q775-379622-e6f9f62f9691.json'
credentials = service_account.Credentials.from_service_account_file(CREDENTIAL_PATH,)
project_id = 'q775-379622'
```

```
## Querying New York data from BigQuery
```

```
In 1373
sql_query = \
'''SELECT *
FROM bigquery-public-data.new_york.tlc_yellow_trips_2012
where rand() < 1/100000;
'''
```

```
In 1374
trips = pd.read_gbq(sql_query, project_id=project_id, credentials=credentials)
```

```
In 1375
trips.shape
```

```
Out 1375
(1833, 19)
```

Figure 2: Used SQL query to pull 1 in 100,000 records.'

I got 1833 records and 19 rows.

(b) Visualize the trip distance versus fare amount. Do you observe any anomalies? If yes, clean the data by removing the invalid data and running a new BigQuery or SQL query. Then plot the data again.

Here we see few outliers. Now we will draw box plot to understand its quartile range and then remove them and then finally drawing scatter plot again to check if all the outliers have been removed.Finally in last figure all the outliers has been removed.

In 1377

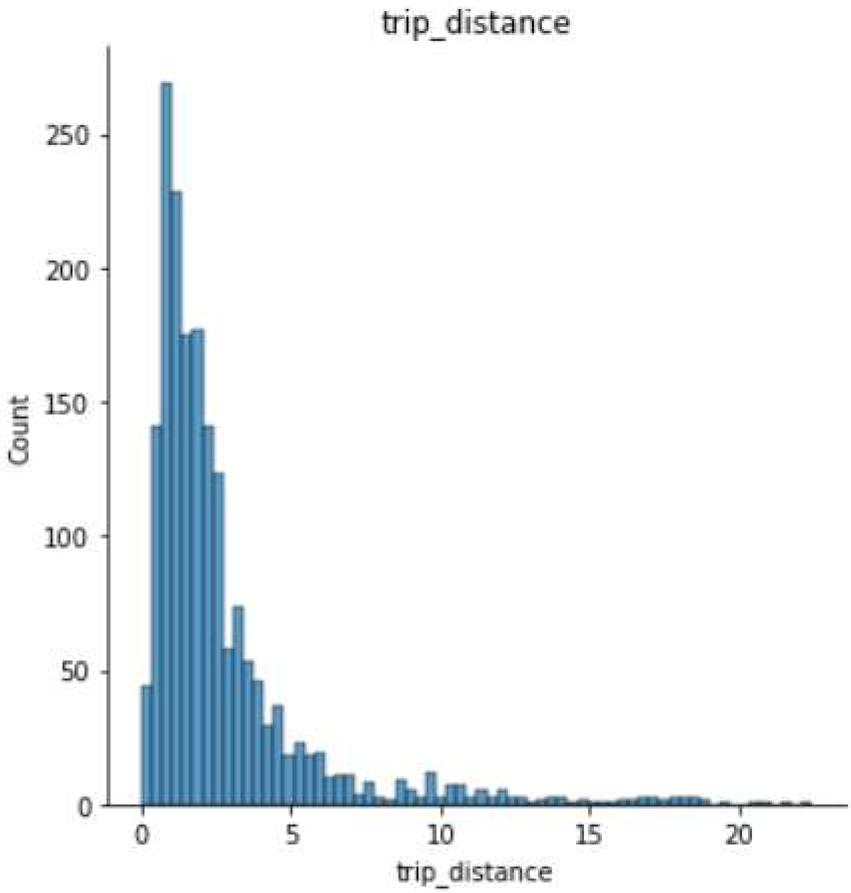
```python
# Read dataframe from local CSV file
trips['trip_distance'] = trips['trip_distance'].astype(float)
trips['fare_amount'] = trips['fare_amount'].astype(float)
```

In 1378

```python
sns.displot(trips['trip_distance']).set(title='trip_distance')

```

Out 1378

```
<seaborn.axisgrid.FacetGrid at 0x1dfcc2833d0>
```



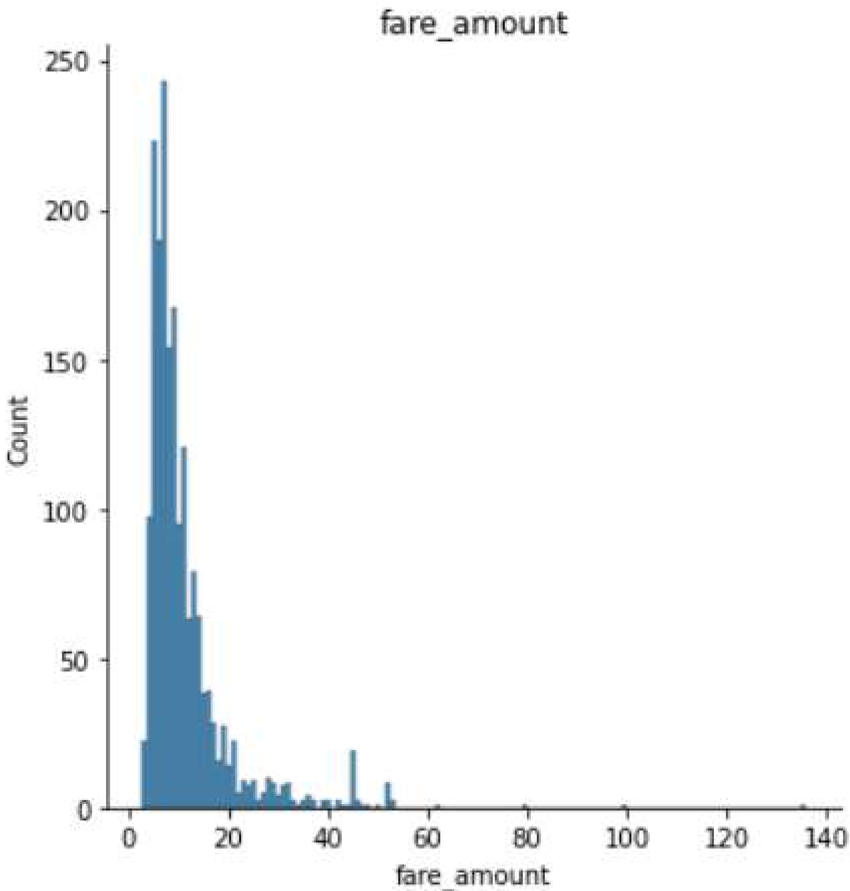
Add Code

In 1379

```
sns.displot(trips['fare_amount']).set(title='fare_amount')
```

Out 1379

<seaborn.axisgrid.FacetGrid at 0x1dfcc340550>

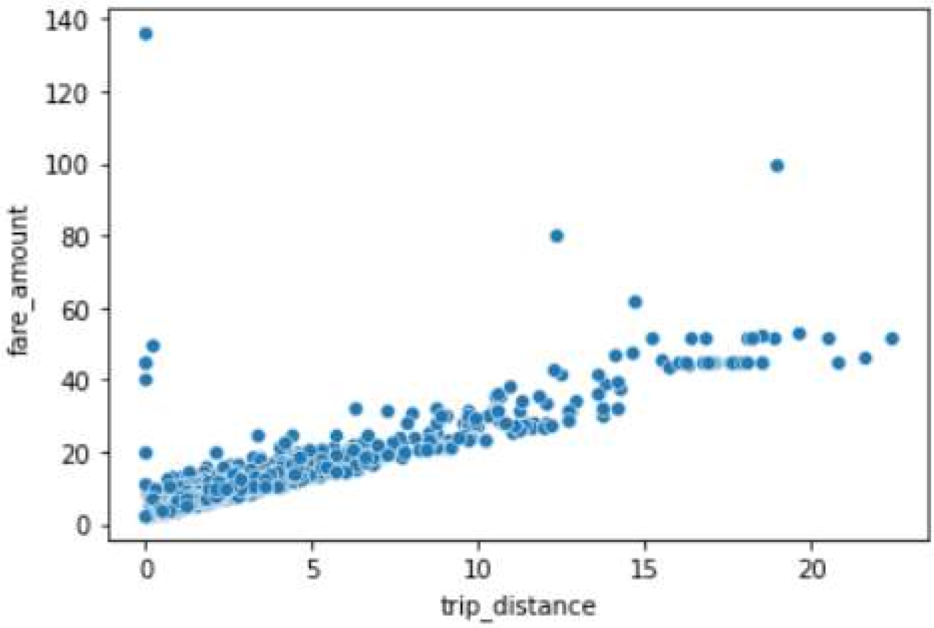


In 1380

```
sns.scatterplot(data=trips, x='trip_distance', y='fare_amount')
```

Out 1380

<AxesSubplot:xlabel='trip_distance', ylabel='fare_amount'>



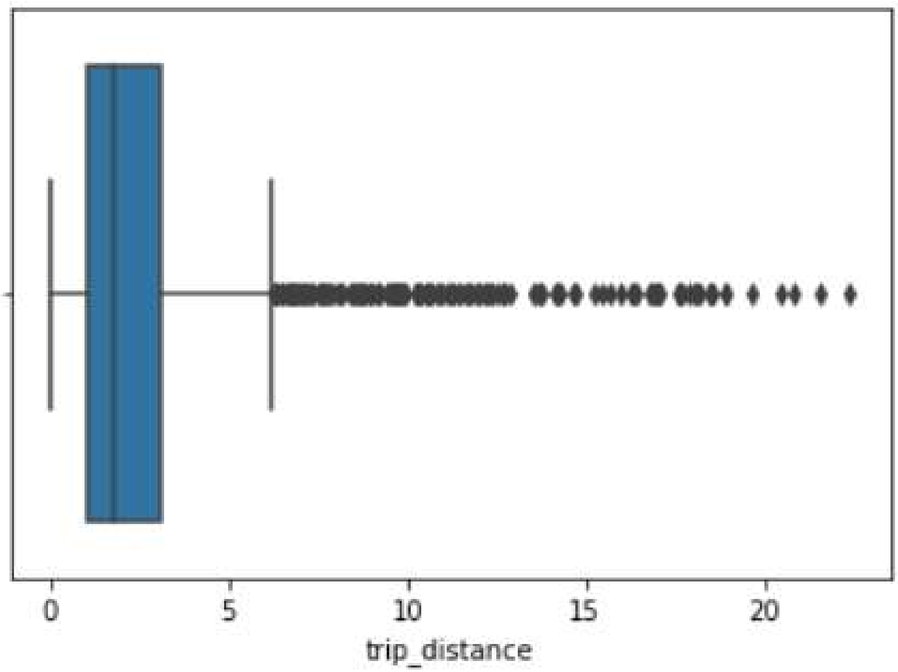
In 1381

```python
sns.boxplot(trips['trip_distance'])
```

```
C:\Users\malhoa25\Anaconda3\lib\site-packages\seaborn\_deco
arg: x. From version 0.12, the only valid positional argume
keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out 1381

```
<AxesSubplot:xlabel='trip_distance'>
```



In 1382

```
#Step-5: Finding the IQR

percentile25 = trips['trip_distance'].quantile(0.25)
percentile75 = trips['trip_distance'].quantile(0.75)

print("75th quartile: ",percentile75)
print("25th quartile: ",percentile25)
```

```
75th quartile:  3.12
25th quartile:  1.03
```

In 1383

```
iqr = percentile75 - percentile25
print ("IQR: ",iqr)
```

```
IQR:  2.09
```

In 1384

```python

#Step-6: Finding the upper and lower limits

upper_limit = percentile75 + 1.5 * iqr
lower_limit = percentile25 - 1.5 * iqr
#Step-7: Finding outliers

trips[trips['trip_distance'] > upper_limit]
trips[trips['trip_distance'] < lower_limit]

#Step-8: Trimming outliers

trips[trips['trip_distance'] < upper_limit]
new_trips = trips[trips['trip_distance'] < upper_limit]
new_trips.shape
```

Out 1384

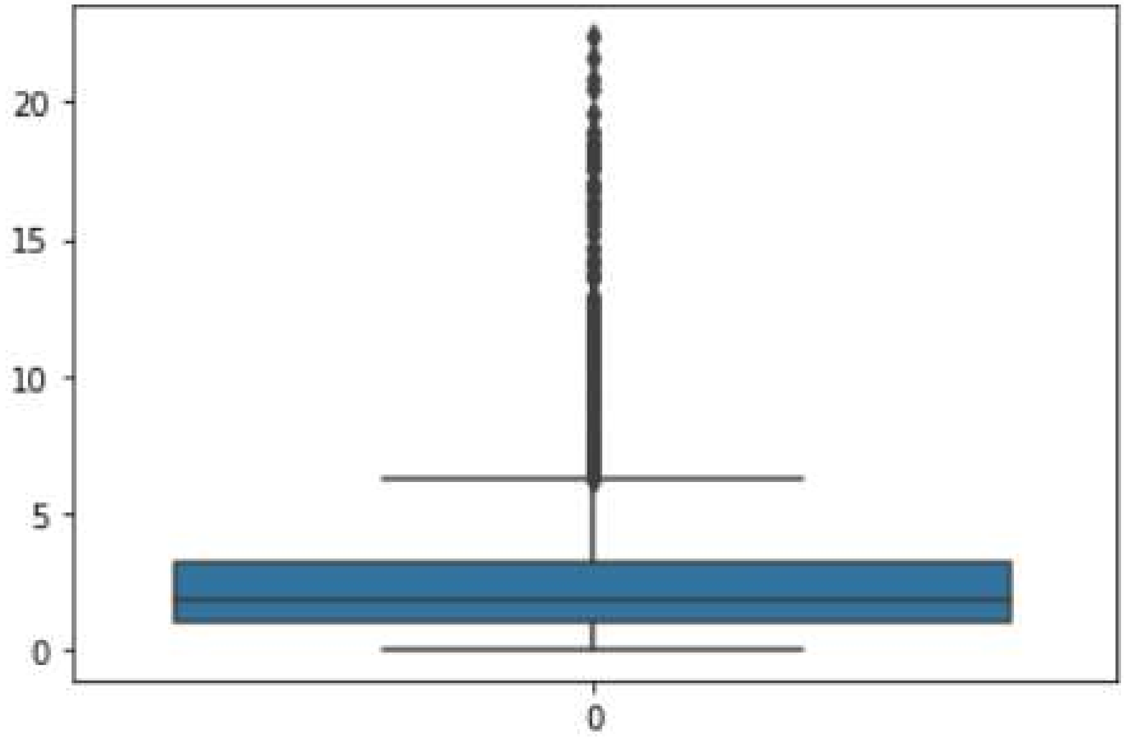
```
(1677, 19)
```

In 1385

```python
sns.boxplot(data=trips['trip_distance'])
```

Out 1385

<AxesSubplot:>

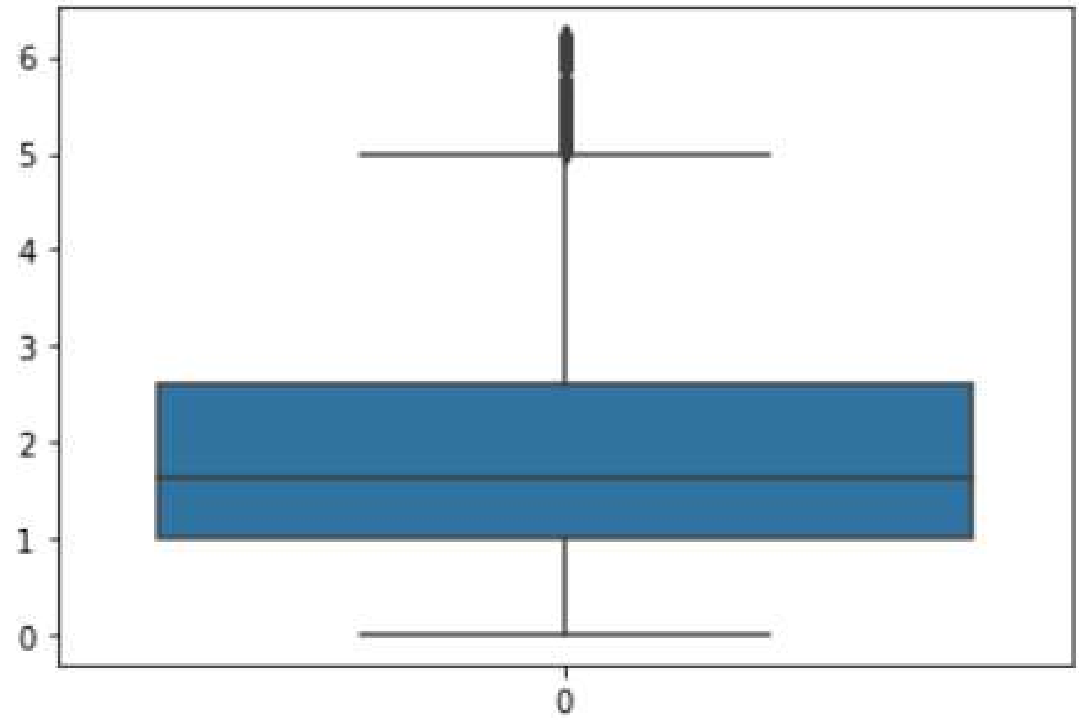


In 1386

```
#Step-9: Compare the plots after trimming
sns.boxplot(data=new_trips['trip_distance'])
```

Out 1386

<AxesSubplot:>



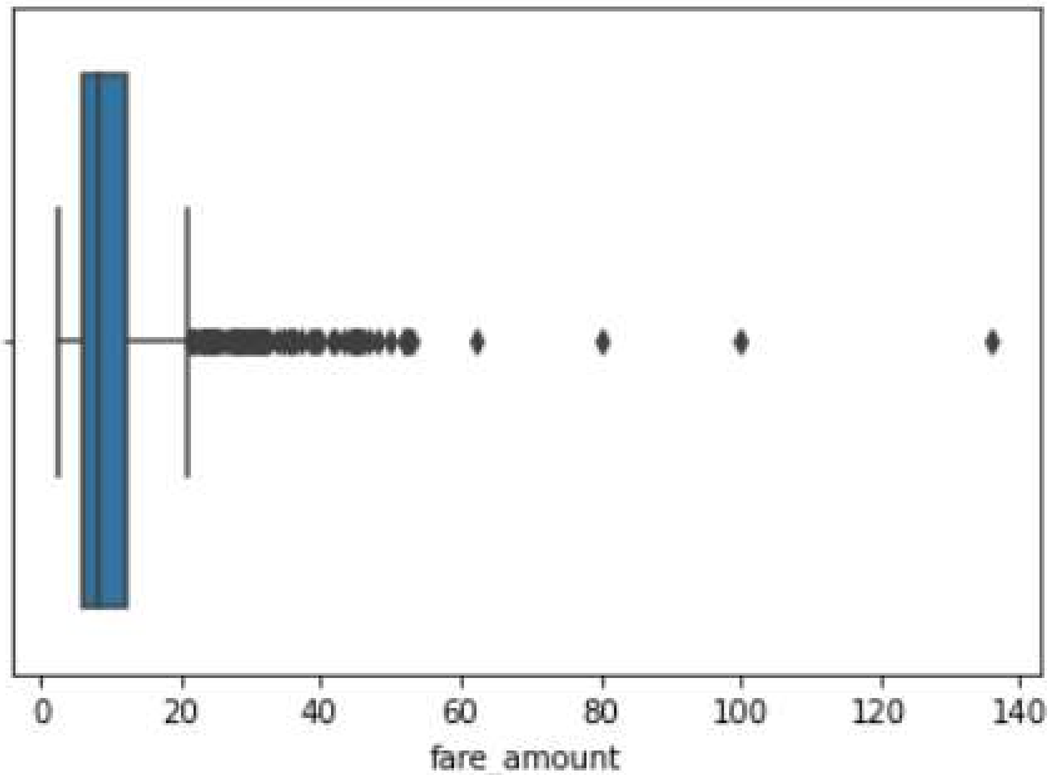
In 1387

```
sns.boxplot(trips['fare_amount'])
```

```
C:\Users\malhoa25\Anaconda3\lib\site-packages\se
arg: x. From version 0.12, the only valid positi
keyword will result in an error or misinterpreta
  warnings.warn(
```

Out 1387

```
<AxesSubplot:xlabel='fare_amount'>
```



In 1388

```python
#Step-5: Finding the IQR

percentile25 = trips['fare_amount'].quantile(0.25)
percentile75 = trips['fare_amount'].quantile(0.75)

print("75th quartile: ",percentile75)
print("25th quartile: ",percentile25)
```

```
75th quartile:  12.1
25th quartile:  6.0
```

In 1389

```python
iqr = percentile75 - percentile25
print ("IQR: ",iqr)
```

```
IQR:  6.1
```

In 1390

```
upper_limit = percentile75 + 1.5 * iqr
lower_limit = percentile25 - 1.5 * iqr
#Step-7: Finding outliers

trips[trips['fare_amount'] > upper_limit]
trips[trips['fare_amount'] < lower_limit]

#Step-8: Trimming outliers

trips[trips['fare_amount'] < upper_limit]
new_trips = trips[trips['fare_amount'] < upper_limit]
new_trips.shape
```

Out 1390

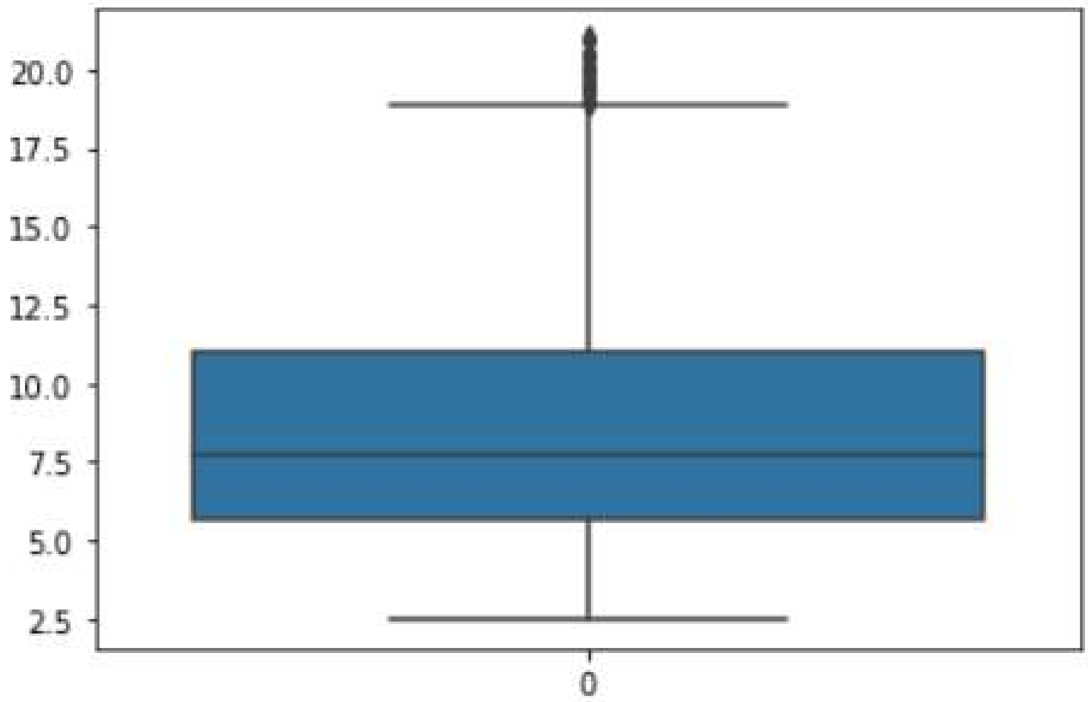
```
(1695, 19)
```

In 1391

```
sns.boxplot(data=new_trips['fare_amount'])
```

Out 1391

<AxesSubplot:>

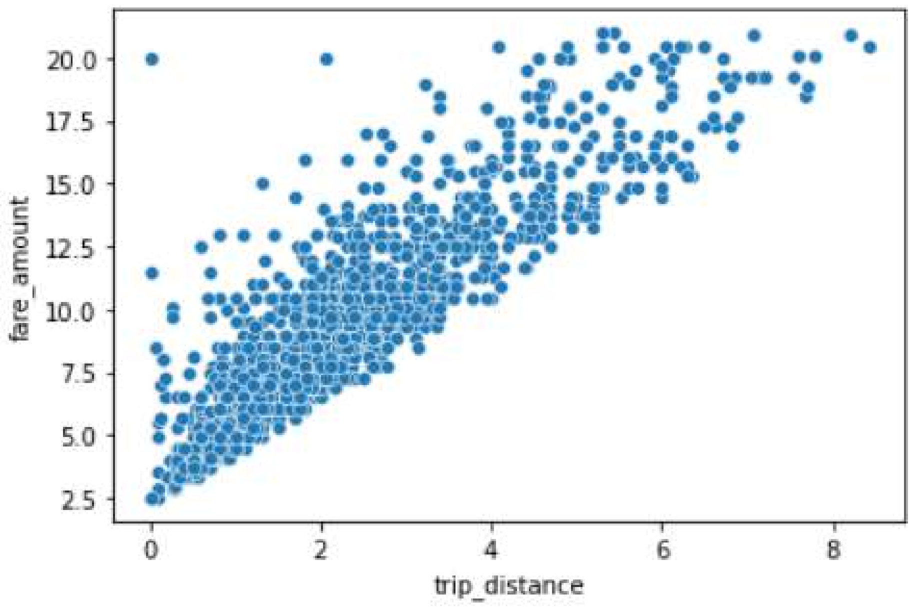


In 1392

```
sns.scatterplot(data=new_trips, x='trip_distance', y='fare_amount')
```

Out 1392

```
<AxesSubplot:xlabel='trip_distance', ylabel='fare_amount'>
```



In 1393

```
new_trips.shape
```

Out 1393

```
(1695, 19)
```

(c) Examine whether the toll amount is captured in the total amount by looking at records that have positive tolls for pick-up time 2012-09-05 15:45:00. What do you notice?

In 1394

```
sql_query2 = \
'''SELECT fare_amount, extra, mta_tax, tip_amount, tolls_amount, total_amount
FROM bigquery-public-data.new_york.tlc_yellow_trips_2012
where (pickup_datetime = '2012-09-05 15:45:00') and (tolls_amount != 0) ;
'''
```

In 1395

```
trips2 = pd.read_gbq(sql_query2, project_id=project_id, credentials=credentials)
trips2.shape
```

Out 1395

```
(13, 6)
```

In 1396

```
print(trips2)
```

	fare_amount	extra	mta_tax	tip_amount	tolls_amount	total_amount
0	38.0	0.0	0.5	0.0	4.8	43.3
1	32.0	0.0	0.5	9.6	4.8	46.9
2	68.5	0.0	0.0	0.0	9.5	78.0
3	32.5	0.0	0.5	0.0	4.8	37.8
4	52.0	0.0	0.5	10.4	4.8	67.7
5	27.0	0.0	0.5	5.4	4.8	37.7
6	28.0	0.0	0.5	5.6	4.8	38.9
7	34.0	0.0	0.5	6.8	4.8	46.1
8	32.5	0.0	0.5	0.0	4.8	37.8
9	53.0	0.0	0.5	10.6	4.8	68.9

Figure 3: Examining if toll amount is captures in the total amount

After examining, I found that toll amount is captured in the total amount. We can analyse in the figure.

(d) Report the descriptive statistics on the selected data: count, mean, std, min, $25\%, 50\%, 75\%$ **and max. What do you notice about the longitude values?**

In 1397

```
new_trips.describe()
```

Out 1397

8 rows 8 rows × 14 columns CSV

	passenger_count	trip_distance	pickup_longitude	pickup_latitude	rate_code	dropoff_longitude	dropoff_latitude
count	1695.000000	1695.000000	1695.000000	1695.000000	1695.000000	1695.000000	1695.0000
mean	1.719764	2.058490	-72.195847	39.790207	1.004720	-72.232519	39.7910
std	1.340472	1.453646	11.342457	6.187833	0.113863	11.232923	6.1879
min	0.000000	0.000000	-74.017884	0.000000	1.000000	-74.017936	0.0000
25%	1.000000	1.000000	-73.992848	40.735292	1.000000	-73.991480	40.7359
50%	1.000000	1.690000	-73.982555	40.751894	1.000000	-73.980987	40.7532
75%	2.000000	2.655000	-73.969731	40.766415	1.000000	-73.967162	40.7686
max	6.000000	8.430000	0.000000	40.847300	4.000000	0.000000	40.8473

Figure 4: Descriptive Statistics on the data

After looking at the descriptive data longitudinal values, I analyzed that even though this data has New York longitude i.e. values around -74 but few of the cells have discrepancy as maximum value shown is zero.

(e) Plot the start and end of a few (10) trips and their corresponding tolls. What do you notice?

After analyzing the scatter plot, we noticed there is no relationship between the distance travelled along with tolls amount. It may be happening that we have small data or may be the toll is in the city only. And if anyone taking the highway, needs to pay the toll.

In 10

```python
sql_query3 = \
    '''SELECT pickup_longitude, pickup_latitude, dropoff_latitude, dropoff_longitude, tolls_amount
    FROM bigquery-public-data.new_york.tlc_yellow_trips_2012
  where (tolls_amount != 0)
   LIMIT 10;
    '''
trips3 = pd.read_gbq(sql_query3, project_id=project_id, credentials=credentials)
trips3.shape
```

Out 10

```
(10, 5)
```

In 11

```python
print(trips3)
```

```
   pickup_longitude  pickup_latitude  dropoff_latitude  dropoff_longitude  \
0        -73.776883        40.645851         40.795818         -73.962699
1        -73.948316        40.829597         40.803214         -73.948467
2        -73.976826        40.759201         40.751768         -73.991501
3        -74.006153        40.748597         40.760968         -73.973491
4        -73.984777        40.721880         40.732299         -74.044324
5        -74.182947        40.628290         40.627548         -74.181620
6        -74.039399        40.730083         40.730106         -74.039349
7        -73.938656        40.752091         40.752091         -73.938656
8        -73.982093        40.762020         40.737940         -74.059900
9        -73.987108        40.769708         40.743072         -73.915164
```

In 23

```python
pickup_coords = trips3[['pickup_longitude', 'pickup_latitude']]
dropoff_coords = trips3[['dropoff_longitude', 'dropoff_latitude']]
tolls = trips3['tolls_amount']
```

In 30

```python
plt.scatter(pickup_coords['pickup_longitude'], pickup_coords['pickup_latitude'], s=tolls*20, c='r', label='Pickup')
plt.scatter(dropoff_coords['dropoff_longitude'], dropoff_coords['dropoff_latitude'], s=tolls*20, c='b', label='Dropof

# Set the x and y limits to zoom in on a particular area
plt.xlim([-73.77, -74.2])
plt.ylim([40.6, 40.9])
# Plot the route for each trip
for i in range(len(trips3)):
    pickup = (trips3['pickup_longitude'][i], trips3['pickup_latitude'][i])
    dropoff = (trips3['dropoff_longitude'][i], trips3['dropoff_latitude'][i])
    plt.plot([pickup[0], dropoff[0]], [pickup[1], dropoff[1]], 'k-', alpha=0.3)
    plt.annotate('$'+str(trips3['tolls_amount'][i]), xy=(dropoff[0], dropoff[1]), xytext=(-20, 20), textcoords='offse
     ha='right', va='bottom', bbox=dict(boxstyle='round,pad=0.5', fc='white', alpha=0.5))

# Display the plot
plt.show()
```

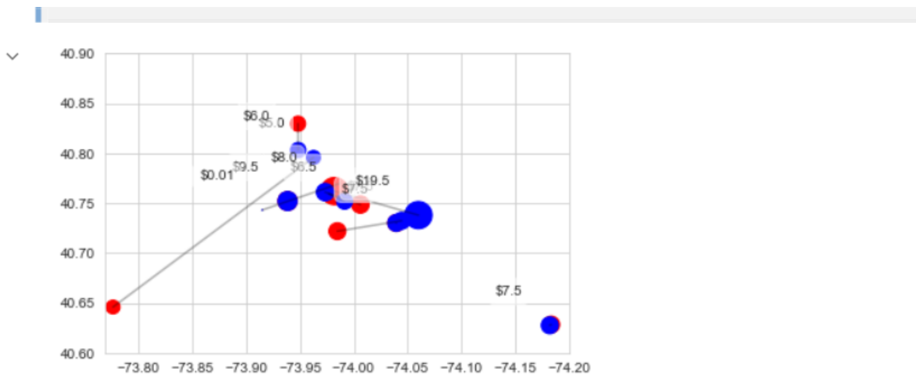


Figure 5: Scatter plot between start and end of 10 trips with corresponding tolls

(f) To prepare the data for prediction (machine learning, ML), we need to further clean it by observing the following:

- **New York city's longitudes are around –74, and latitudes are around 41.**
- **We should not have zero passengers.**
- **Clean up the total amount column to reflect only the fare and toll amounts, then remove those two columns.**
- **Before the ride starts, we will know the pickup and drop-off locations but not the trip distance (that depends on the route taken), so remove it from the ML dataset**
- **Discard the time stamp.**

The above process is sometimes referred to as the data quality control step. You may use BigQuery or Python to do the above operations. Once done, return the data descriptors as in (d) of this question. How many rows were removed?

In 1401

```
new_trips = new_trips[new_trips['pickup_latitude'] != 0]
new_trips = new_trips[new_trips['pickup_latitude'] > 1]
new_trips = new_trips[new_trips['passenger_count'] != 0]
```

In 1402

```
new_trips.shape
```

Out 1402

```
(1647, 19)
```

In 1403

```
print(new_trips)
```

```
2               NaN        17.50
4               NaN        19.44
5               NaN         9.38
...             ...          ...
1825            NaN        10.70
1826            NaN        12.24
1827            NaN        13.62
1831            NaN        11.70
1832            NaN         4.20

[1647 rows x 19 columns]
```

In 1404

```
#Replacing new values in total_amount by emptying it first and then finally adding fare and toll amount (partf)
new_trips.total_amount = "-"
new_trips.total_amount = new_trips.fare_amount + new_trips.tolls_amount
```

26 14 36

In 1405

```
new_trips.head()
```

Out 1405

5 rows 5 rows × 19 columns CSV

	vendor_id	pickup_datetime	dropoff_datetime	passenger_count	trip_distance	pickup_longitude	pick
0	CMT	2012-11-14 22:58:30+00:00	2012-11-14 23:20:28+00:00	1	5.50	-74.011095	
1	VTS	2012-09-07 12:22:00+00:00	2012-09-07 12:47:00+00:00	1	2.52	-73.963615	
2	VTS	2012-10-17 00:25:00+00:00	2012-10-17 00:45:00+00:00	4	3.76	-73.987095	
4	CMT	2012-07-24 07:32:18+00:00	2012-07-24 07:45:48+00:00	1	6.00	-74.016593	
5	VTS	2012-05-24 00:47:00+00:00	2012-05-24 00:53:00+00:00	1	1.75	-74.000945	

In 1406

```
#dropping column fare amount and toll amount
new_trips = new_trips.drop(['fare_amount'], axis=1)
new_trips
```

Out 1406

11 rows 1647 rows × 18 columns CSV

longitude	dropoff_latitude	payment_type	extra	mta_tax	tip_amount	tolls_amount	imp_surcharge	total_amount
-73.984400	40.691852	CRD	0.5	0.5	3.30	0.0	NaN	15.5
-74.005605	40.745132	CSH	0.0	0.5	0.00	0.0	NaN	17.0
-74.006200	40.706322	CSH	0.5	0.5	0.00	0.0	NaN	16.5
-73.972623	40.754156	CRD	0.0	0.5	3.24	0.0	NaN	15.7
-73.984243	40.743245	CRD	0.5	0.5	1.48	0.0	NaN	6.9
...	...	...	...	...	...	...	...	...
-74.000485	40.737380	CSH	0.5	0.5	0.00	0.0	NaN	9.7

In 1407

```
new_trips = new_trips.drop(['tolls_amount'], axis=1)
new_trips = new_trips.drop(['rate_code'], axis=1)
new_trips = new_trips.drop(['store_and_fwd_flag'], axis=1)
new_trips = new_trips.drop(['payment_type'], axis=1)
new_trips = new_trips.drop(['imp_surcharge'], axis=1)
new_trips
```

Out 1407

11 rows 1647 rows × 13 columns CSV

	vendor_id	pickup_datetime	dropoff_datetime	passenger_count	trip_distance	pickup_longitude	pic
4	CMT	2012-07-24 07:32:18+00:00	2012-07-24 07:45:48+00:00	1	6.00	-74.016593	
5	VTS	2012-05-24 00:47:00+00:00	2012-05-24 00:53:00+00:00	1	1.75	-74.000945	
...	...	...	...	...	...	...	
1825	VTS	2012-05-31 01:55:00+00:00	2012-05-31 02:07:00+00:00	3	2.84	-73.970242	
1826	CMT	2012-03-05 10:37:12+00:00	2012-03-05 10:51:37+00:00	1	2.10	-73.979540	
1827	VTS	2012-09-22 18:07:00+00:00	2012-09-22 18:18:00+00:00	1	2.55	-73.952943	
1831	CMT	2012-06-21 23:38:42+00:00	2012-06-21 23:52:54+00:00	1	2.40	-73.984838	
1832	CMT	2012-07-10 12:08:36+00:00	2012-07-10 12:11:25+00:00	1	0.50	-74.004139	

In 1408

```
new_trips = new_trips.drop(['trip_distance'], axis=1)
new_trips
```

Out 1408

11 rows 1647 rows × 12 columns

	vendor_id	pickup_datetime	dropoff_datetime	passenger_count	pickup_longitude	pickup_latitude	d
0	CMT	2012-11-14 22:58:30+00:00	2012-11-14 23:20:28+00:00	1	-74.011095	40.708404	
1	VTS	2012-09-07 12:22:00+00:00	2012-09-07 12:47:00+00:00	1	-73.963615	40.808487	
2	VTS	2012-10-17 00:25:00+00:00	2012-10-17 00:45:00+00:00	4	-73.987095	40.748255	
4	CMT	2012-07-24 07:32:18+00:00	2012-07-24 07:45:48+00:00	1	-74.016593	40.709927	
5	VTS	2012-05-24 00:47:00+00:00	2012-05-24 00:53:00+00:00	1	-74.000945	40.737432	
...	...	...	...	...	...	...	
1825	VTS	2012-05-31 01:55:00+00:00	2012-05-31 02:07:00+00:00	3	-73.970242	40.762157	
1826	CMT	2012-03-05 10:37:12+00:00	2012-03-05 10:51:37+00:00	1	-73.979540	40.771234	

In 1409

```python
#Discarding time stamp
new_trips['pickup_datetime'] = pd.to_datetime(new_trips['pickup_datetime'])
new_trips['pickup_datetime'] = new_trips['pickup_datetime'].apply(lambda x: x.timestamp()).astype(float)
new_trips['dropoff_datetime'] = pd.to_datetime(new_trips['dropoff_datetime'])
new_trips['dropoff_datetime'] = new_trips['dropoff_datetime'].apply(lambda x: x.timestamp()).astype(float)
```

In 1410

```python
#new_trips['dropoff_datetime'] = pd.to_datetime(new_trips['dropoff_datetime']).dt.date
```

In 1411

```python
new_trips = pd.get_dummies(new_trips, columns=['vendor_id'])
```

In 1412

```python
#new_trips['pickup_datetime'] = pd.to_datetime(new_trips['pickup_datetime'])
#new_trips['pickup_datetime'] = new_trips['pickup_datetime'].astype(int)
```

26 14 36

In 1413 1

```
new_trips
```

Out 1413

11 rows | 1647 rows × 13 columns | CSV

	pickup_datetime	dropoff_datetime	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff
0	1.352934e+09	1.352935e+09	1	-74.011095	40.708404	-73.984400	
1	1.347021e+09	1.347022e+09	1	-73.963615	40.808487	-74.005605	
2	1.350434e+09	1.350435e+09	4	-73.987095	40.748255	-74.006200	
4	1.343115e+09	1.343116e+09	1	-74.016593	40.709927	-73.972623	
5	1.337820e+09	1.337821e+09	1	-74.000945	40.737432	-73.984243	
...	...	...	...	...	...	...	
1825	1.338429e+09	1.338430e+09	3	-73.970242	40.762157	-74.000485	
1826	1.330944e+09	1.330945e+09	1	-73.979540	40.771234	-73.971538	
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	

In 1414 1

```
new_trips.describe()
```

Out 1414

8 rows | 8 rows × 13 columns | CSV

	pickup_datetime	dropoff_datetime	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropof
count	1.647000e+03	1.647000e+03	1647.000000	1647.000000	1647.000000	1647.000000	
mean	1.341067e+09	1.341068e+09	1.729812	-73.940562	40.751926	-73.978336	
std	9.044042e+06	9.044051e+06	1.334823	1.636223	0.024091	0.022756	
min	1.325387e+09	1.325388e+09	1.000000	-74.017884	40.641342	-74.017936	
25%	1.333459e+09	1.333460e+09	1.000000	-73.993161	40.737179	-73.991677	
50%	1.340829e+09	1.340830e+09	1.000000	-73.983017	40.752625	-73.981428	
75%	1.348764e+09	1.348765e+09	2.000000	-73.971293	40.766622	-73.968218	
max	1.356998e+09	1.356998e+09	6.000000	-7.583332	40.847300	-73.702002	

In _

```
new_trips.shape
```

: 1437

```
(1647, 13)
```

Before cleaning data we had 1833 rows 19 columns, but after cleaning we are left with 1647 rows and 13 columns. So total 186 rows have been removed.

(g) Split the quality-controlled data randomly into training (75%), validation (15%) and test (15%) sets. Show the descriptors of all these sets as in (d) of this question.

In 1416
```python
X = new_trips.copy()
y = new_trips['total_amount']
```

In 1417
```python
X_train, X_rem, y_train, y_rem = train_test_split(X,y, train_size=0.7)
```

In 1418
```python
# Now since we want the valid and test size to be equal (15% each of overall data).
# we have to define valid_size=0.5 (that is 50% of remaining data)
test_size = 0.5
X_valid, X_test, y_valid, y_test = train_test_split(X_rem,y_rem, test_size=0.5)

print(X_train.shape), print(y_train.shape)
print(X_valid.shape), print(y_valid.shape)
print(X_test.shape), print(y_test.shape)
```

```
(1152, 13)
(1152,)
(247, 13)
(247,)
(248, 13)
(248,)
```

Out 1418
```
(None, None)
```

In 1419
```python
X_train
```

26 14 36

In 1443

```
X_train.describe()
```

Out 1443

8 rows 8 rows x 13 columns CSV

	pickup_datetime	dropoff_datetime	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff
count	1.152000e+03	1.152000e+03	1152.000000	1152.000000	1152.000000	1152.000000	
mean	1.341043e+09	1.341043e+09	1.710938	-73.980761	40.751551	-73.978351	
std	9.077560e+06	9.077575e+06	1.317114	0.022346	0.023537	0.022801	
min	1.325387e+09	1.325388e+09	1.000000	-74.017884	40.641342	-74.017936	
25%	1.333350e+09	1.333350e+09	1.000000	-73.994070	40.737451	-73.991761	
50%	1.340698e+09	1.340699e+09	1.000000	-73.982891	40.752387	-73.981254	
75%	1.348924e+09	1.348925e+09	2.000000	-73.970906	40.766531	-73.967800	
max	1.356998e+09	1.356998e+09	6.000000	-73.776705	40.839140	-73.702002	

In 1444

```
y_train.describe()
```

Out 1444

```
count    1152.000000
mean        8.858854
std         3.995139
min         2.500000
25%         5.700000
50%         7.700000
75%        11.000000
max        22.300000
Name: total_amount, dtype: float64
```

In 1445

```
X_valid.describe()
```

Out 1445

8 rows 8 rows x 13 columns CSV

	pickup_datetime	dropoff_datetime	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropof
count	2.470000e+02	2.470000e+02	247.000000	247.000000	247.000000	247.000000	
mean	1.340772e+09	1.340773e+09	1.842105	-73.980509	40.754850	-73.978192	
std	8.946191e+06	8.946197e+06	1.441039	0.017916	0.025613	0.022955	
min	1.325424e+09	1.325424e+09	1.000000	-74.017242	40.676129	-74.017012	
25%	1.333767e+09	1.333767e+09	1.000000	-73.991485	40.737580	-73.990928	
50%	1.340286e+09	1.340286e+09	1.000000	-73.983356	40.756167	-73.981683	
75%	1.348430e+09	1.348431e+09	2.000000	-73.970472	40.770292	-73.969877	
max	1.356685e+09	1.356686e+09	6.000000	-73.868980	40.847300	-73.785772	

In 1446

```
y_valid.describe()
```

Out 1446

```
count    247.000000
mean       8.821457
std        3.873601
min        2.500000
25%        6.000000
50%        8.000000
75%       11.150000
max       20.900000
Name: total_amount, dtype: float64
```

26 14 36

In 1447

```
X_test.describe()
```

Out 1447

8 rows 8 rows × 13 columns CSV

	pickup_datetime	dropoff_datetime	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff
count	2.480000e+02	2.480000e+02	248.000000	248.000000	248.000000	248.000000	
mean	1.341477e+09	1.341478e+09	1.705645	-73.714045	40.750759	-73.978413	
std	9.006710e+06	9.006701e+06	1.306341	4.216356	0.024944	0.022437	
min	1.325709e+09	1.325710e+09	1.000000	-74.017342	40.670012	-74.016106	
25%	1.333595e+09	1.333596e+09	1.000000	-73.993082	40.734846	-73.991897	
50%	1.341816e+09	1.341816e+09	1.000000	-73.983620	40.753335	-73.981730	
75%	1.348373e+09	1.348374e+09	2.000000	-73.973623	40.764276	-73.969567	
max	1.356872e+09	1.356872e+09	6.000000	-7.583332	40.840615	-73.862370	

In 1448

```
y_test.describe
```

Out 1448

```
<bound method NDFrame.describe of 84      5.7
398     6.5
519     9.0
781     5.3
990     5.0
       ...
840     6.1
1773    6.5
538     9.7
332     6.1
784     6.1
Name: total_amount, Length: 248, dtype: float64>
```

(h) Save the three data sets to CSV files and verify they were created and show some of their headers.

```
Name: total_amount, Length: 240, dtype: float64
```

In 1420
```python
X_train.to_csv('mytraindata.csv', index=False)
```

In 1421
```python
from IPython.display import FileLink
FileLink('mytraindata.csv')
```

Out 1421

mytraindata.csv

In _
```python
X_train.head()
```

Out 1449

5 rows ⌄ 5 rows × 13 columns CSV ⌄

	pickup_datetime	dropoff_datetime	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff
1083	1.341239e+09	1.341239e+09	1	-74.000480	40.727296	-74.015109	
965	1.341219e+09	1.341219e+09	1	-73.960295	40.817830	-73.964845	
791	1.345055e+09	1.345055e+09	1	-73.983412	40.758132	-73.983412	
394	1.327341e+09	1.327343e+09	1	-73.989730	40.744926	-73.919194	
837	1.339354e+09	1.339356e+09	1	-73.966096	40.765254	-73.978648	

In 1422
```python
y_train.to_csv('mytraindata_y.csv', index=False)
from IPython.display import FileLink

FileLink('mytraindata_y.csv')
```

Out 1422

mytraindata_y.csv

26 14 36

In 1450

```
y_train.head()
```

Out 1450

```
1083     7.7
965      4.9
791      3.3
394     18.1
837     11.3
Name: total_amount, dtype: float64
```

In 1423

```
X_valid.to_csv('myvaliddata.csv', index=False)
from IPython.display import FileLink

FileLink('myvaliddata.csv')
```

Out 1423

myvaliddata.csv

In _

```
X_valid.head()
```

Out 1451

5 rows 5 rows × 13 columns CSV

	pickup_datetime	dropoff_datetime	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff
322	1.347906e+09	1.347906e+09	6	-73.963795	40.808062	-73.958015	
509	1.341763e+09	1.341764e+09	2	-73.975935	40.765490	-73.982147	
787	1.345653e+09	1.345654e+09	1	-73.956349	40.771674	-73.972545	
1616	1.355126e+09	1.355126e+09	1	-74.003900	40.738187	-73.994686	
1463	1.334752e+09	1.334753e+09	1	-73.981592	40.732582	-73.967221	

In 1424

```
y_valid.to_csv('myvaliddata_y.csv', index=False)
from IPython.display import FileLink

FileLink('myvaliddata_y.csv')
```

Out 1424

myvaliddata_y.csv

In 1452

```
y_valid.head()
```

Out 1452

```
322      4.5
509      6.1
787      8.5
1616     5.0
1443    13.7
Name: total_amount, dtype: float64
```

In 1425

```
X_test.to_csv('mytestdata.csv', index=False)
from IPython.display import FileLink

FileLink('mytestdata.csv')
```

Out 1425

mytestdata.csv

26 14 36

In 1453

```
X_test.head()
```

Out 1453

5 rows × 13 columns

	pickup_datetime	dropoff_datetime	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_
84	1.327087e+09	1.327088e+09	4	-73.965140	40.755128	-73.969797	
398	1.353549e+09	1.353550e+09	2	-73.999690	40.738677	-73.998082	
519	1.350695e+09	1.350696e+09	1	-74.001615	40.740797	-73.982072	
781	1.326004e+09	1.326005e+09	1	-73.979458	40.735492	-73.987908	
990	[illegible]	[illegible]	1	[illegible]	[illegible]	[illegible]	

In 1426

```
y_test.to_csv('mytestdata_y.csv', index=False)
from IPython.display import FileLink

FileLink('mytestdata_y.csv')
```

Out 1426

mytestdata_y.csv

In _

```
y_test.head()
```

Out 1454

```
84     5.7
398    6.5
519    9.0
781    5.3
990    5.0
Name: total_amount, dtype: float64
```

(i) Use linear programming as well as other predictive models (such as regression or custom models) to predict the fare rate. Think of this machine learning model as a fare-estimation tool that could be put in a phone app to suggest a likely fare to New York Taxi riders so that they are not surprised and so that they can protest if the charge is much higher than expected.

In 1427

```
# load the training, validation, and test data
train_data = pd.read_csv('mytraindata.csv')
train_targets = pd.read_csv('mytraindata_y.csv')
valid_data = pd.read_csv('myvaliddata.csv')
valid_targets = pd.read_csv('myvaliddata_y.csv')
test_data = pd.read_csv('mytestdata.csv')
test_targets = pd.read_csv('mytestdata_y.csv')
```

In 1428

```
# fit the linear regression model to the training data
model = LinearRegression()
model.fit(train_data, train_targets)
```

Out 1428

```
LinearRegression()
```

In 1429

```
# evaluate the model on the validation set
valid_preds = model.predict(valid_data)
valid_mse = mean_squared_error(valid_targets, valid_preds)
print('Validation MSE:', valid_mse)
```

```
Validation MSE: 2.35310945942012e-21
```

In 1430

```
# predict on the test set and calculate the test MSE
test_preds = model.predict(test_data)
test_mse = mean_squared_error(test_targets, test_preds)
print('Test MSE:', test_mse)
```

```
Test MSE: 2.3909759632081948e-21
```

In 1431

```
print(test_preds)
```

```
[ 4.5]
[ 8. ]
[ 5.7]
[ 5.5]
[11.7]
[10. ]
[ 9.3]
[13.5]
[ 8.5]
[10.1]
```

In 1435

```python
test_preds_df = pd.DataFrame(test_preds, columns=['predicted_total_amount'])

# Save the DataFrame to a CSV file
test_preds_df.to_csv('mypredicteddata.csv', index=False)

# Display a link to download the CSV file
from IPython.display import FileLink
FileLink('mypredicteddata.csv')
```

Out 1435

mypredicteddata.csv

I have used linear regression to predict data and attached all the training, validating, testing and predicted data csv files as well as python code for the verification purposes.

As we can see that mean square error is very small, so it shows that predicted total amount is very accurate. We can also compare in predicted data file and actual data file that total amount is very near to the actual amount.

NOTE: Linear Programming is generally not used to predict total amount for taxi fare as we need to predict continuous variable here. LP is used to optimize linear objective not for prediction. Therefore, linear regression works better for this problem.