Exercise 1.1 Pandas

Importing the required libraries

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import scipy.linalg
import warnings
warnings.filterwarnings('ignore')
```

Load data

To load the data into dataframe from the .csv file

In [2]:

```
data = pd.read_csv("GasPrices.csv")
data.head()
```

Out[2]:

	Unnamed: 0	ID	Name	Price	Pumps	Interior	Restaurant	CarWash	Highway	Intersection
0	1	1	Shell	1.79	4	Y	N	N	N	Y
1	2	2	Valero	1.83	4	Y	N	N	N	Y
2	3	3	7-Eleven	1.88	4	Y	N	N	N	Y
3	4	4	Texaco	1.88	4	Y	N	Y	N	Y
4	5	5	Shell	1.84	6	Y	N	N	N	Y

Summarize numeric values

This cell summarizes the sum, mean, standard deviation of Price, pumps, gasolines and income

In [22]:

```
Price_mean = round(np.mean(data["Price"]),3)
Price_sum = np.sum(data["Price"])
Price_sd = np.std(data["Price"])
Price_min = np.min(data["Price"])
Price_max = np.max(data["Price"])
print("Price:\nMean:",round(Price_mean,3),"\t\tSum:",round(Price_sum,3),"\tStandard Deviati
Pumps_mean = np.mean(data["Pumps"])
Pumps_sum = np.sum(data["Pumps"])
Pumps_sd = np.std(data["Pumps"])
Pumps_min = np.min(data["Pumps"])
Pumps_max = np.max(data["Pumps"])
print("\nPumps:\nMean:",round(Pumps_mean,3),"\t\tSum:",round(Pumps_sum,3),"\tStandard Devia
Income_mean = np.mean(data["Income"])
Income_sum = np.sum(data["Income"])
Income_sd = np.std(data["Income"])
Income_min = np.min(data["Income"])
Income_max = np.max(data["Income"])
print("\nIncome:\nMean:",round(Income_mean,3),"\tSum:",round(Income_sum,3),"\tStandard Devi
Gasoline_mean = np.mean(data["Gasolines"])
Gasoline_sum = np.sum(data["Gasolines"])
Gasoline_sd = np.std(data["Gasolines"])
Gasoline_min = np.min(data["Gasolines"])
Gasoline_max = np.max(data["Gasolines"])
print("\nGasolines:\nMean:",round(Gasoline_mean,3),"\t\tSum:",round(Gasoline_sum,3),"\tStar
```

```
Price:
Mean: 1.864             Sum: 188.29     Standard Deviation: 0.081
Min: 1.73               Max: 2.09

Pumps:
Mean: 6.95              Sum: 702        Standard Deviation: 3.906
Min: 2          Max: 24

Income:
Mean: 56727.218         Sum: 5729449    Standard Deviation: 25739.98
Min: 12786              Max: 128556

Gasolines:
Mean: 3.465             Sum: 350        Standard Deviation: 0.555
Min: 1          Max: 4
```

Group field by name

This cell sorts the data in ascending order inorder to compute the mean of price of income per brand

In [9]:

```
Data_sorted = data.sort_values('Name')
Data_sorted = Data_sorted.reset_index(drop = True)
```

In [10]:

```
Data_sorted.head()
```

Out[10]:

	Unnamed: 0	ID	Name	Price	Pumps	Interior	Restaurant	CarWash	Highway	Intersection
0	101	101	7-Eleven	1.95	4	Y	N	N	N	Y
1	14	14	7-Eleven	1.95	4	Y	N	N	N	Y
2	50	50	7-Eleven	1.79	4	Y	N	N	N	Y
3	12	12	7-Eleven	1.97	4	Y	N	N	N	Y
4	25	25	7-Eleven	1.84	6	Y	N	N	N	Y

Find average price, average pumps and average income

To find the average price, average number of pumps and average income of the brands

In [11]:

```
meandata = pd.DataFrame(columns = ['Name','AvgPrice','AvgPumps','AvgIncome'],index=range(0,
meandata.AvgPrice = meandata.AvgPrice.astype(float)
meandata.AvgPumps = meandata.AvgPumps.astype(float)
meandata.AvgIncome = meandata.AvgIncome.astype(float)
meandata = meandata.fillna(0)
oldname = None
index = 0
for i in range(0,len(Data_sorted)):
    name = Data_sorted.Name[i]
    if oldname != name:
        meandata.Name[index] = name
        meandata.AvgPrice[index] = round(np.mean(Data_sorted.loc[Data_sorted.Name == name].
        meandata.AvgPumps[index] = round(np.mean(Data_sorted.loc[Data_sorted.Name == name].
        meandata.AvgIncome[index] = round(np.mean(Data_sorted.loc[Data_sorted.Name == name]
        oldname = name
        index+=1
```

In [12]:

```
meandata.head()
```

Out[12]:

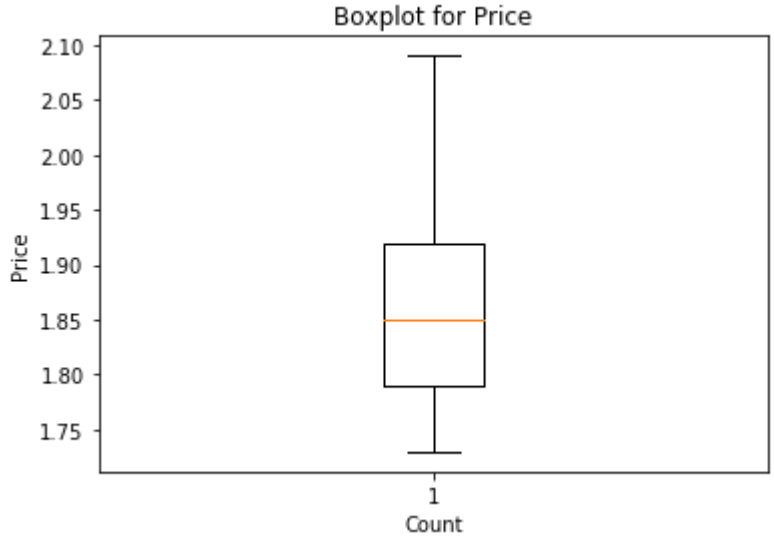
	Name	AvgPrice	AvgPumps	AvgIncome
0	7-Eleven	1.89	4.67	53432.33
1	Around the Corner Store	1.94	2.00	63750.00
2	Chevron	1.87	8.73	61754.64
3	Citgo	1.84	4.00	49387.00
4	Conoco	1.89	4.00	43545.50

Boxplot data

To plot the boxplot for visualizing statistical information between the variables

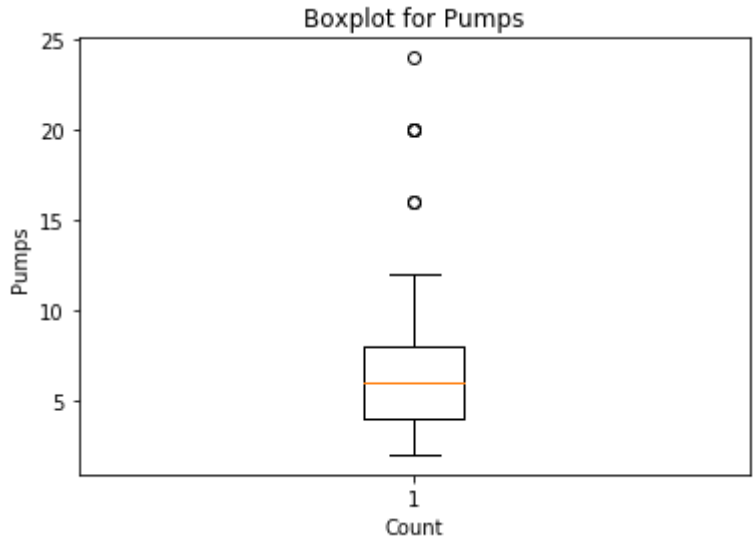
In [13]:

```
plt.xlabel("Count")
plt.ylabel("Price")
plt.title("Boxplot for Price")
plt.boxplot(Data_sorted.Price)
plt.show()
```



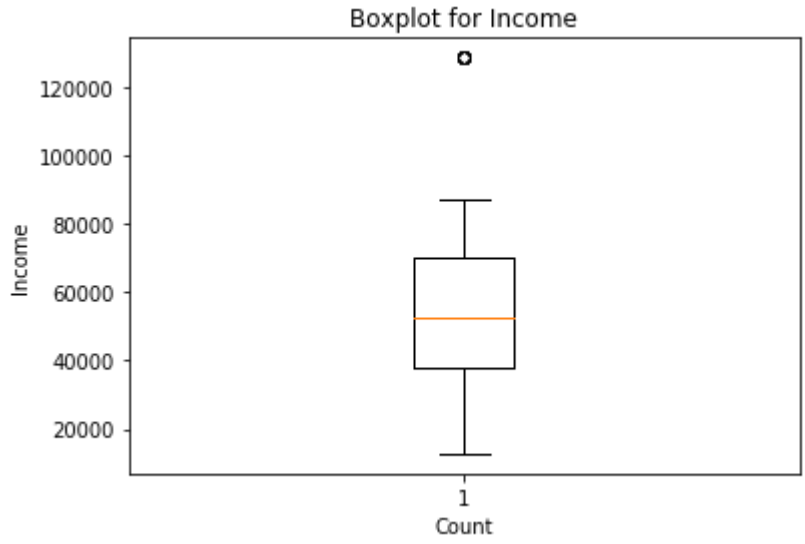
In [14]:

```
plt.xlabel("Count")
plt.ylabel("Pumps")
plt.title("Boxplot for Pumps")
plt.boxplot(Data_sorted.Pumps)
plt.show()
```



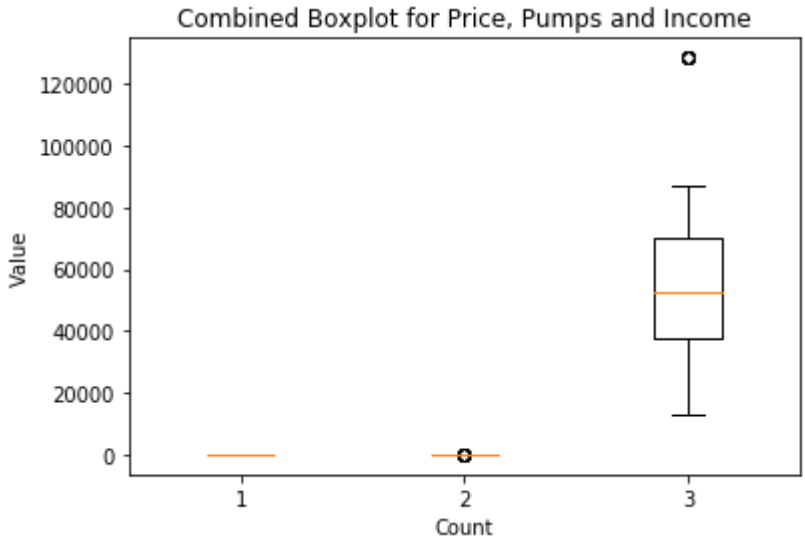
In [15]:

```
plt.xlabel("Count")
plt.ylabel("Income")
plt.title("Boxplot for Income")
plt.boxplot(Data_sorted.Income)
plt.show()
```



In [16]:

```
plt.xlabel("Count")
plt.ylabel("Value")
plt.title("Combined Boxplot for Price, Pumps and Income")
boxdata = [Data_sorted.Price,Data_sorted.Pumps,Data_sorted.Income]
plt.boxplot(boxdata)
plt.show()
```



Predicting line

Predicting the target value from the set of parameters learned using least squares method

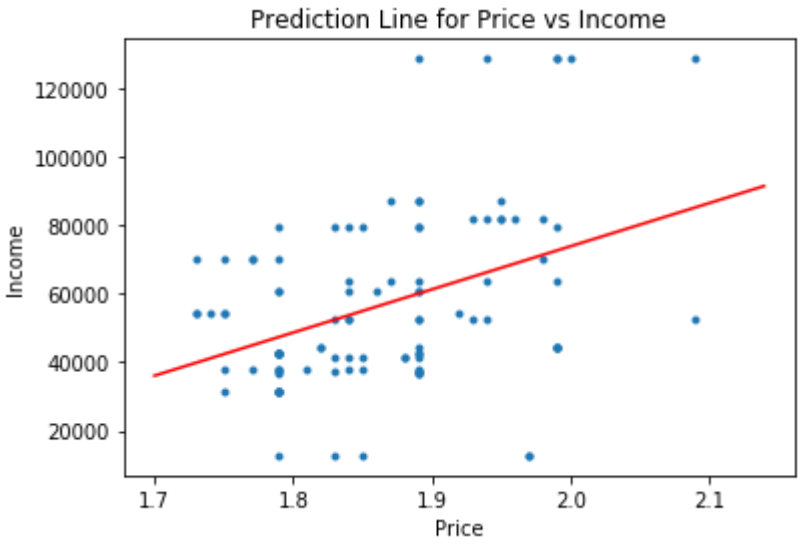
In [17]:

```
A = np.vstack([Data_sorted.Price, np.ones(len(Data_sorted.Price))]).T
y = np.ones((len(Data_sorted),1))
y[:,0] = Data_sorted.Income
b1,b0 = np.linalg.lstsq(A,y)[0]
print("Parameters:\n\nβ0:",round(b0[0],3),"\tβ1:",round(b1[0],3),"\n")
```

```
Parameters:

β0: -177642.807         β1: 125717.63
```

In [18]:

```
plt.plot(A[:,0],y,".")
t = np.arange(1.70,2.15,0.02)
plt.plot(t,b0+b1*t,'r')
plt.xlabel("Price")
plt.ylabel("Income")
plt.title("Prediction Line for Price vs Income")
plt.show()
```



Normalize

Inorder to overcome noises, we normalize the income and predict the target values again

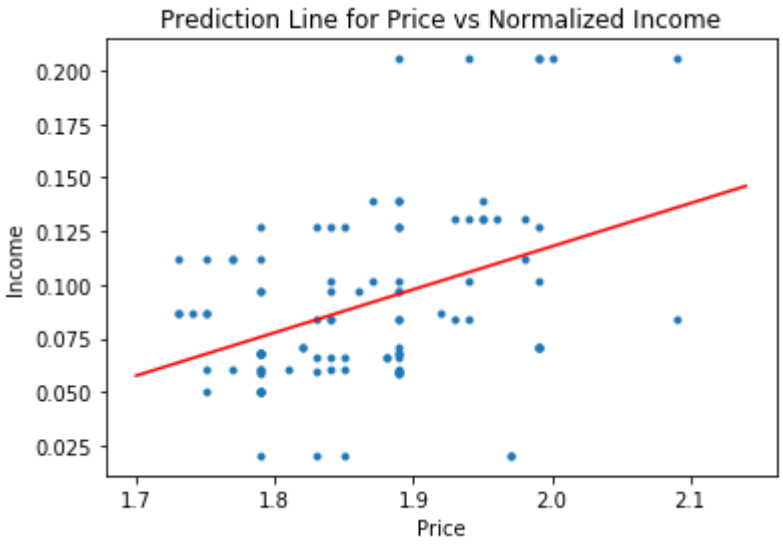
In [19]:

```
total = 0
for i in range (0,len(y)):
    total+= pow(y[i],2)
norm_val = pow(total,0.5)
y = y/norm_val
b1,b0 = np.linalg.lstsq(A,y)[0]
print("\n\nParameters:\n\nβ0:",round(b0[0],3),"\tβ1:",round(b1[0],3),"\n")
```

```
Parameters:

β0: -0.284      β1: 0.201
```

In [20]:

```
plt.plot(A[:,0],y,".")
t = np.arange(1.70,2.15,0.02)
plt.plot(t,b0+b1*t,'r')
plt.xlabel("Price")
plt.ylabel("Income")
plt.title("Prediction Line for Price vs Normalized Income")
plt.show()
```



In [ ]:

Exercise 1.2 Linear Regression via Normal Equations

Importing the required libraries

In [300]:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import scipy.linalg
import random
import math
import warnings
warnings.filterwarnings('ignore')
```

Function Name: Create_data

Parameter: Filename

This function reads the data, shuffles them and split it to input variables and target values

After analysing the Mean Square error with differernt input variables, price,income,gasolines and pumps are taken into account. Also, the variables namely CarWash, Competitors, Zipcode, Address, ID, Restaurant does not play a major role as it can be seen in the data file. And when we analyse the top 10 incomes, we get to see that location of the station i.e., highways, intersection influences the income but as the error increases when we use them, they are also neglected

In [301]:

```
def create_data(filename):
    Data = pd.read_csv(filename)
    Req_Data = Data.drop(columns = ['Unnamed: 0','ID','Restaurant',
                                    'CarWash','Competitors','Zipcode','Address'
                                     ,'Highway','IntersectionStoplight','Name','Stoplight'
                                    ,'Interior','Intersection','Brand'])
    Shuffle_data = Req_Data.sample(frac = 1)
    Shuffle_data = pd.get_dummies(Shuffle_data)
    Y_data = Shuffle_data.Income/np.linalg.norm(Shuffle_data.Income)
    #Y_data = Shuffle_data.Income
    X_data = Shuffle_data.drop(columns = ['Income'])
    X_data = X_data.values
    return X_data,Y_data
```

Function Name: create_Test_Train_data

Parameter: Data of input variables and target

This function splits the input variables and target data to Train and Test data with 80% and 20% of the data respectively

In [302]:

```
def create_Test_Train_data(X_data,Y_data):
    Y_train = Y_data[:math.ceil(0.8*len(Y_data))]
    Y_test = Y_data[math.ceil(0.8*len(Y_data)):]
    X_train = X_data[:math.ceil(0.8*len(X_data))]
    X_test = X_data[math.ceil(0.8*len(X_data)):]
    X = np.ones((X_train.shape[0],X_train.shape[1]+1))
    X[:,1:] = X_train
    A = np.dot(X.T,X)
    B = np.dot(X.T,Y_train)
    return Y_train,Y_test,X_train,X_test,A,B
```

Function Name: slv_beta

Parameter: Input variables matrix and Target vector

This function returns the set of parameters trained from the input variables

In [303]:

```
def slv_beta(A,B):
    b = np.linalg.solve(A,B)
    return b
```

Function Name: Predict

Parameter: Parameter vector

This function predicts the target value for the test data and returns the same

In [304]:

```
def Predict(b):
    Test = np.ones((X_test.shape[0],X_test.shape[1]+1))
    Test[:,1:] = X_test
    Pred_Y_test = np.matmul(Test,b)
    return Pred_Y_test
```

Function Name: MSE_Plot

Parameter: Predicted vector

This function calculates the Root Mean Square Error for the data predicted by the model

In [359]:

```
def RMSE_Plot(Pred_Y_test):
    total = 0
    for i in range(0,len(Y_test)):
        total+=pow(Pred_Y_test[i]-Y_test.values[i],2)
    print("\nMSE:     ",total/len(Y_test))
    plt.plot(Y_test.values,"bo")
    plt.plot(Pred_Y_test,"ro")
    plt.xlabel("Value")
    plt.ylabel("Test and Prediction")
    plt.show()
    return RMSE
```

Function Name: cholesky_matrix

Parameter: Input variables matrix

This function does operations within the matrix and returns the Lower Triangular Matrix to perform backward substitution process for getting the set of parameters

In [346]:

```
def cholesky_matrix(A):

    if A.shape[0] == A.shape[1]:
        A = A.astype(float)
        L = np.zeros_like(A)
        size = np.size(A, 0)
        L[0, 0] = pow(A[0, 0],0.5)
        for i in range(1, size):
            L[i:, i-1] = (A[i:, i-1] - L[i:, :i-1].dot(L[i-1, :i-1])) / L[i-1, i-1]
            L[i, i] = pow((A[i, i] - L[i, :i].dot(L[i, :i])),0.5)
        return L

    else:
        print ("Invalid Matrix")
```

Function Name: gauss_beta

Parameter: Input variables matrix and Target vector

This function solves the set of linear equation returns the set of parameters trained from the input variables using Gaussian elimination method

In [347]:

```
def gauss_beta(A,B):
    if A.shape[0] == A.shape[1]:
        Gauss_mat = np.ones((A.shape[0],A.shape[1]+1))
        Gauss_mat[:,:A.shape[0]] = A[:,:A.shape[0]]
        Gauss_mat[:,A.shape[0]] = B
        A = Gauss_mat
        n = len(A)
        for i in range(0, n):
            # Search for maximum in this column
            maxEl = abs(A[i][i])
            maxRow = i
            for k in range(i+1, n):
                if abs(A[k][i]) > maxEl:
                    maxEl = abs(A[k][i])
                    maxRow = k
            # Swap maximum row with current row (column by column)
            for k in range(i, n+1):
                tmp = A[maxRow][k]
                A[maxRow][k] = A[i][k]
                A[i][k] = tmp

            # Make all rows below this one 0 in current column
            for k in range(i+1, n):
                c = -A[k][i]/A[i][i]
                for j in range(i, n+1):
                    if i == j:
                        A[k][j] = 0
                    else:
                        A[k][j] += c * A[i][j]

        # Solve equation Ax=b for an upper triangular matrix A
        x = [0 for i in range(n)]
        for i in range(n-1, -1, -1):
            x[i] = A[i][n]/A[i][i]
            for k in range(i-1, -1, -1):
                A[k][n] -= A[k][i] * x[i]
        return x

    else:
        print ("Invalid Matrix")
```

Function Name: forward_substitution and backward_substitution

Parameter: Input variables matrix and Target vector

These functions performs the forward and backward substitutions for the lower triangle and upper triangle matrix respectively

In [348]:

```
def forward_substitution(A, B):

    if A.shape[0] == A.shape[1]:
        x = np.zeros_like(B)
        for i in range(np.size(x, 0)):
            x[i] = (B[i] - A[i, :i].dot(x[:i])) / A[i, i]
        return x

    else:
        print ("Invalid Matrix")

def backward_substitution(A, B):

    if A.shape[0] == A.shape[1]:
        x = np.zeros_like(B)
        for i in range(np.size(x, 0)).__reversed__():
            x[i] = (B[i] - A[i, i+1:].dot(x[i+1:])) / A[i, i]
        return x

    else:
        print ("Invalid Matrix")
```

Function Name: QR_beta

Parameter: Input variables matrix and Target vector

This function solves the set of linear equation returns the set of parameters trained from the input variables using QR decomposition method

In [349]:

```
def QR_beta(A,B):
    Q,R = np.linalg.qr(A)
    dot_QB = np.dot(Q.T,B)
    b = backward_substitution(R,dot_QB)
    return b
```

Function Name: linalg_slv

Parameter: Input variables matrix and Target vector

This function solves the set of linear equation using normal equations. From the learned set of parameters, the target data is predicted from the model and the error is computed

In [350]:

```
def linalg_slv(A,B):
    if A.shape[0] == A.shape[1]:
        b = slv_beta(A,B)
        print("Solving linear equations using Normal equations:\n")
        print("Set of Parameters:\n",b)
        Pred_Y_test = Predict(b)
        RMSE = RMSE_Plot(Pred_Y_test)
        return Pred_Y_test

    else:
        print ("Invalid Matrix")
```

Function Name: gaussian_elimination

Parameter: Input variables matrix and Target vector

This function solves the set of linear equation using Gaussian elimination method. From the learned set of parameters, the target data is predicted from the model and the error is computed

In [351]:

```
def gaussian_elimination(A,B):
    b = gauss_beta(A,B)
    print("Solving linear equations using Gaussian elimination:\n")
    print("Set of Parameters:\n",b)
    Pred_Y_test = Predict(b)
    RMSE = RMSE_Plot(Pred_Y_test)
    return Pred_Y_test
```

Function Name: cholesky_decomposition

Parameter: Input variables matrix and Target vector

This function solves the set of linear equation using Cholesky decomposition method. From the learned set of parameters, the target data is predicted from the model and the error is computed

In [352]:

```
def cholesky_decomposition(A,B):
    mat_cholesky= cholesky_matrix(A)
    y = forward_substitution(mat_cholesky,B)
    y=np.array(y)
    b = backward_substitution(mat_cholesky.T,y)
    print("Solving linear equations using Cholesky Decomposition:\n")
    print("Set of Parameters: \n",b)
    Pred_Y_test = Predict(b)
    RMSE = RMSE_Plot(Pred_Y_test)
    return Pred_Y_test
```

Function Name: QR_decomposition

Parameter: Input variables matrix and Target vector

This function solves the set of linear equation using QR decomposition method. From the learned set of parameters, the target data is predicted from the model and the error is computed

In [353]:

```
def QR_decomposition(A,B):
    b = QR_beta(A,B)
    print("Solving linear equations using QR Decomposition:\n")
    print("Set of Parameters:\n",b)
    Pred_Y_test = Predict(b)
    RMSE = RMSE_Plot(Pred_Y_test)
    return Pred_Y_test
```

In [354]:

```
def RMSE(Predicted,Y_test):
    return np.sqrt(np.sum(pow(Y_test - np.mean(Predicted),2))/len(Y_test))
```

Main function to call other functions

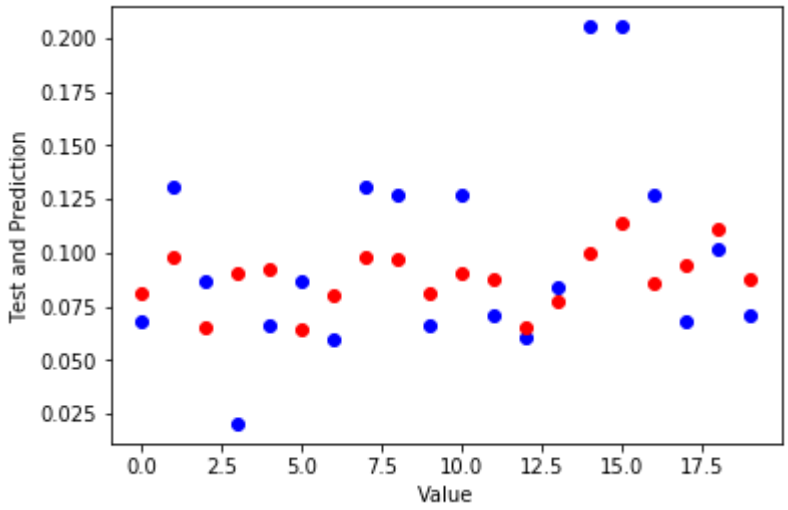
In [355]:

```
if __name__ == "__main__":

    filename = "GasPrices.csv"
    X_data,Y_data = create_data(filename)
    Y_train,Y_test,X_train,X_test,A,B = create_Test_Train_data(X_data,Y_data)
```

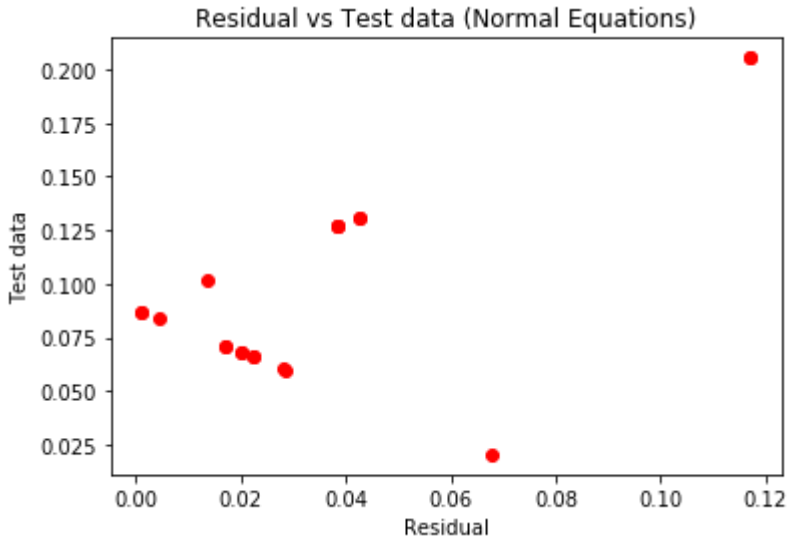
In [361]:

```
Pred_slv = linalg_slv(A,B)
residual_slv  = abs(Y_test.values - np.mean(Pred_slv))
print("\nAverage Residual: ",np.average(residual_slv))
print("\nRMSE : ",round(RMSE(residual_slv,Y_test),5))
plt.plot(residual_slv ,Y_test,'ro')
plt.xlabel("Residual")
plt.ylabel("Test data")
plt.title("Residual vs Test data (Normal Equations)")
plt.show()
```

```
Solving linear equations using Normal equations:

Set of Parameters:
 [-0.25705143  0.17101638  0.00075655  0.00623746]

MSE:      0.0017161062057884425
```



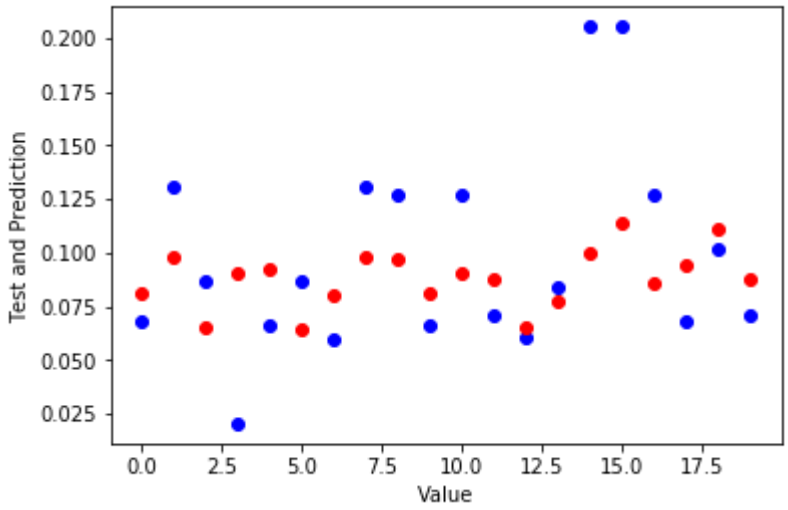

```
Average Residual:  0.03493582456177754

RMSE :  0.07817
```



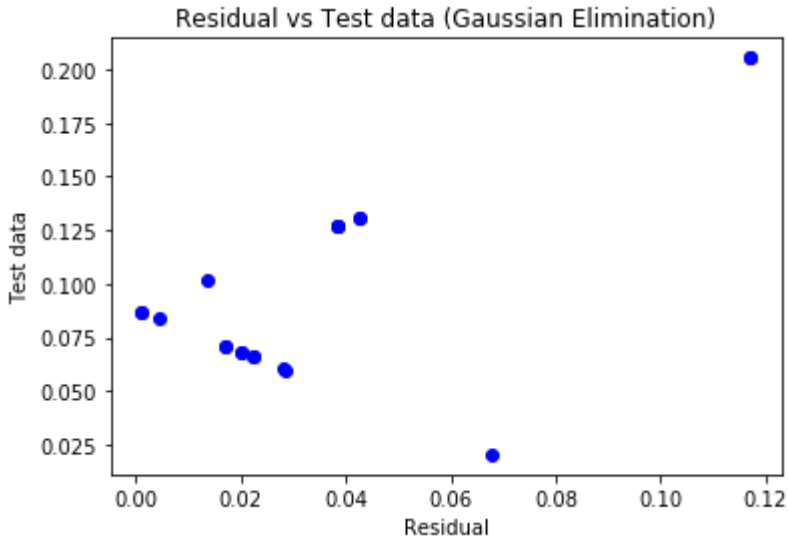
In [365]:

```
Pred_ge = gaussian_elimination(A,B)
residual_ge = abs(Y_test.values - np.mean(Pred_ge))
print("\nAverage Residual: ",np.average(residual_ge))
print("\nRMSE : ",round(RMSE(residual_ge,Y_test),5))
plt.plot(residual_ge ,Y_test,'bo')
plt.xlabel("Residual")
plt.ylabel("Test data")
plt.title("Residual vs Test data (Gaussian Elimination)")
plt.show()
```

```
Solving linear equations using Gaussian elimination:

Set of Parameters:
 [-0.2570514254119503, 0.1710163814561609, 0.0007565529462504973, 0.00623746
23675145995]

MSE:      0.0017161062057884386
```



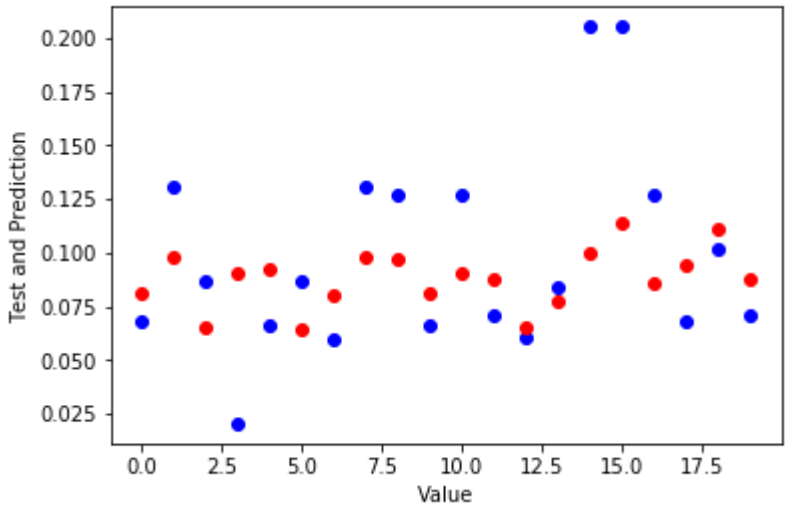

```
Average Residual:  0.03493582456177753

RMSE :  0.07817
```



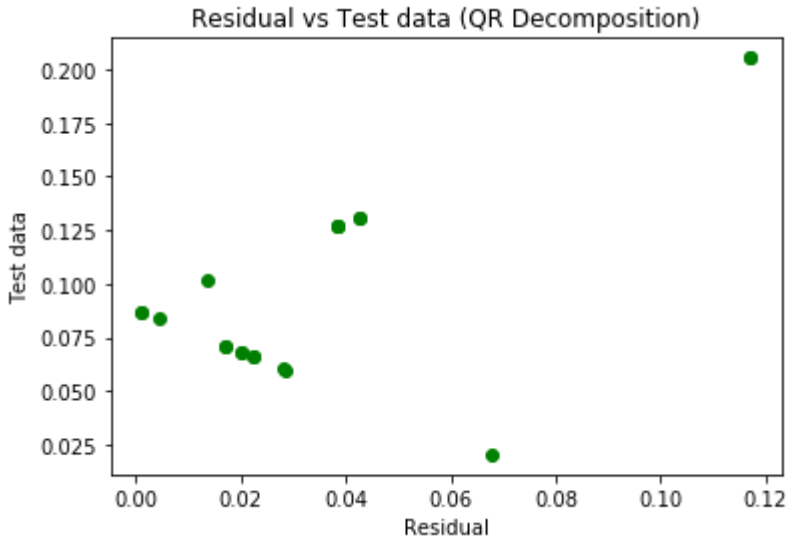
In [366]:

```
Pred_qr = QR_decomposition(A,B)
residual_qr = abs(Y_test.values - np.mean(Pred_qr))
print("\nAverage Residual: ",np.average(residual_qr))
print("\nRMSE : ",round(RMSE(residual_qr,Y_test),5))
plt.plot(residual_qr ,Y_test,'go')
plt.xlabel("Residual")
plt.ylabel("Test data")
plt.title("Residual vs Test data (QR Decomposition)")
plt.show()
```

```
Solving linear equations using QR Decomposition:

Set of Parameters:
 [-0.25705143  0.17101638  0.00075655  0.00623746]

MSE:      0.0017161062057885843
```



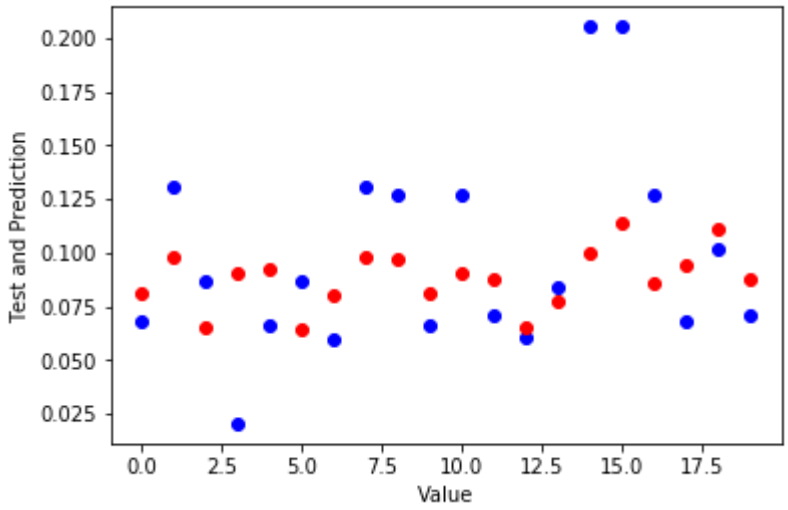

```
Average Residual:  0.03493582456177764

RMSE :  0.07817
```



In [367]:

```
Pred_cholesky = cholesky_decomposition(A,B)
residual_cholesky = abs(Y_test.values - np.mean(Pred_cholesky))
print("\nAverage Residual: ",np.average(residual_cholesky))
print("\nRMSE : ",round(RMSE(residual_cholesky,Y_test),5))
plt.plot(residual_cholesky ,Y_test,'yo')
plt.xlabel("Residual")
plt.ylabel("Test data")
plt.title("Residual vs Test data (Cholesky Decomposition)")
plt.show()
```

```
Solving linear equations using Cholesky Decomposition:

Set of Parameters:
 [-0.25705143  0.17101638  0.00075655  0.00623746]

MSE:      0.0017161062057884102
```




```
Average Residual:  0.03493582456177752

RMSE :  0.07817
```