|  |  |
| --- | --- |
| **Name** | **Contribution** |
| Fok Jiajun Samuel | I did question 1-3 with Isabel & Bryan |
| Isabel Loh Li Jun | I did question 1-3 with Bryan & Samuel |
| Lee Min Kang | I did question 1-3 with Isabel & Samuel |



**ANL252 (Online)**

**PYTHON FOR DATA ANALYTICS**

# **Group-Based Assignment**

**July 2021 Presentation**

**Submitted by:**

|  |  |
| --- | --- |
| **Name** | **PI No.** |
| **Fok Jiajun Samuel** | **J2111086** |
| **Isabel Loh Li Jun** | **Z2070330** |
| **Lee Min Kang Bryan** | **E1771573** |
|  |  |

**Tutorial Group: T 09**

**Instructor’s Name: Dr. Munish Kumar**

**Submission Date: 28/08/2021**

**Question 1**

1. *#Numpy, Math Library*

import matplotlib.pyplot as plt *#for part 1d & 1e: Histogram & Scatter Diagram*

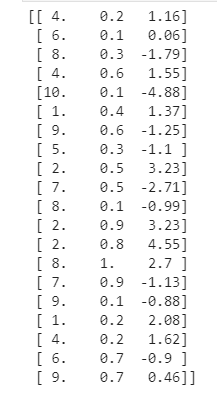
import numpy as np *#for numpy arrays*

import math

arr = np.array([[4, 0.2, 1.16], [6, 0.1, 0.06], [8, 0.3, -1.79], [4, 0.6, 1.55], [10, 0.1, -4.88], [1, 0.4, 1.37], [9, 0.6, -1.25], [5, 0.3, -1.1], [2, 0.5, 3.23], [7, 0.5, -2.71], [8, 0.1, -0.99], [2, 0.9, 3.23], [2, 0.8, 4.55], [8, 1, 2.7], [7, 0.9, -1.13], [9, 0.1, -0.88], [1, 0.2, 2.08], [4, 0.2, 1.62], [6, 0.7, -0.9], [9, 0.7, 0.46]]) *#Store X1, X2 and Y into a Numpy array*

newarr = np.resize(arr,(20,3)) *#Resize numpy array to have 3 columns, 20 rows*

print(newarr) *#Newarr array*

****

1. *#Variables X1 & X2 will represent X1 & X2 respectively.*

*#obtain X1, X2, Y array from newarr numpy array.*

X1 = newarr[:,[0]] *#Stored as a separate array called X1, obtain 1st column in newarr as X1 array*

X2 = newarr[:,[1]] *#Stored as a separate array called X2, obtain 2nd column in newarr as X2 array*

Y = newarr[:,[2]] *#Stored as a separate array called Y, , obtain 3rd column in newarr as Y array*

*#Variable PY is used to represent Y^*

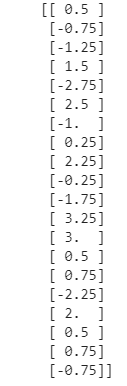
e1 = 0.5

e2 = 2.5

PY = 2 - (np.multiply(e1, X1)) + (np.multiply(e2,X2)) *#Y^ = 2 - 0.5X1 + 2.5X2.*

*#np.multiply used here to multiply an array to a number*

print (PY)

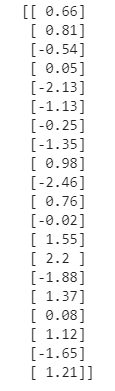


**(c)**

*#Var Re is used to represent e^, and Variable Y represents Y.*

Re = Y - PY *#Residual e^ = Y - Y^.*

print (Re) *#Re array*



**(d)** *#Histogram: Re (Residuals e)*

Y = np.rint(Y) *#Round off the values in array Y*

Y = Y.astype(int) *#Convert values in array Y, into integers*

PY = np.rint(PY) *#Round off the values in array PY*

PY = PY.astype(int) *#Convert values in array PY, into integers*

Re = np.rint(Re) *#Round off the values in array Re*

Re = Re.astype(int) *#Convert values in array Re, into integers*

*#Use “Re = Y - PY”, as Re has no x value, for the histogram.*

*#Xticks, histogram, chart title, x and y axis required*

plt.figure(figsize = (10, 10)) *#adjust the size of histogram*

plt.hist(Y-PY, bins = 20, label="Re", edgecolor ='lightblue') *#Size of the columns to be 20, colour: light blue. Data taken to be “Y - PY”*

plt.xticks(ticks = np.arange(-3, 4, 0.5), labels = np.arange(-3, 4, 0.5)) *#Range for X axis: -3 to 3.5, increased by 0.5 each*

plt.yticks(ticks = np.arange(-3, 7, 0.5), labels = np.arange(-3, 7, 0.5)) *#Range for Y axis: -3 to 7, increased by 0.5 each*

plt.margins(0.1) *#Margins for X and Y axis*

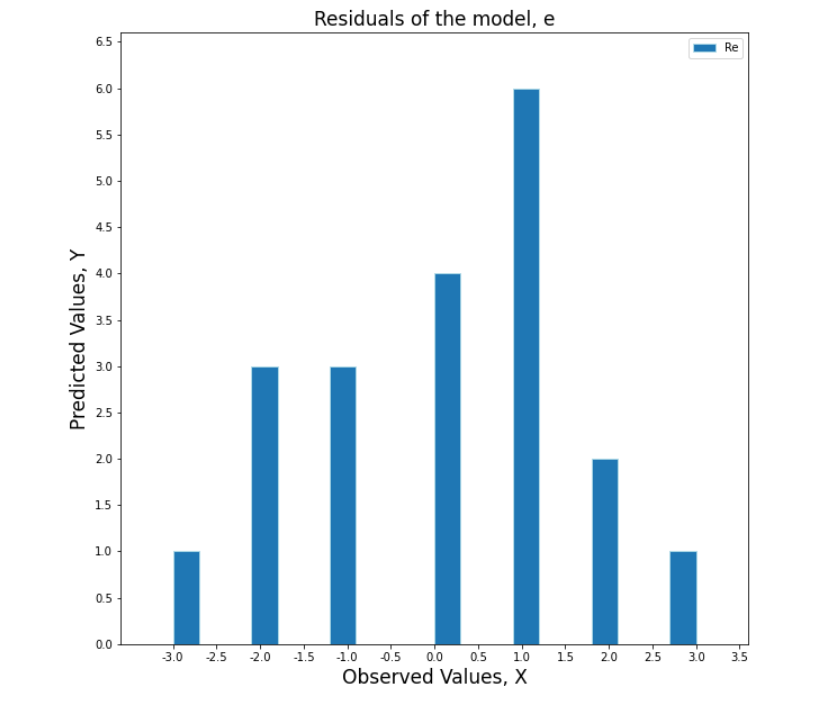
plt.xlabel ("Observed Values, X", size = 17) *#X axis label, font size: 17*

plt.ylabel ("Predicted Values, Y", size = 17) *#Y axis label, font size: 17*

plt.title ("Residuals of the model, e", size = 17) *#Chart title, font* size: 17

plt.legend (loc = 'upper right') *#Chart legend on top right*

**(d)**

****

*#Statement to answer 1d*

I agree that the normality assumption with zero mean is valid based on the histogram. From the histogram, the data seems evenly spread out and the mean seems to be 0.

**(e)** *#X axis: Predicted Y^ values (PY)*

*#Y axis: Residuals e (Re)*

*#Chart title: Correlation between PY and Re*

*#Xticks and Yticks*

plt.figure(figsize = (10, 10)) *#adjust the size of scatter diagram*

plt.scatter(PY, Re, color = "blue", marker = "o", edgecolor = "blue") *#use blue dot for scatter diagram, with reference to PY data*

plt.xlabel("Predicted Y values", size = 15) *#X axis label, font size:15*

plt.ylabel("Residuals e", size = 15) *#Y axis label, font size:15*

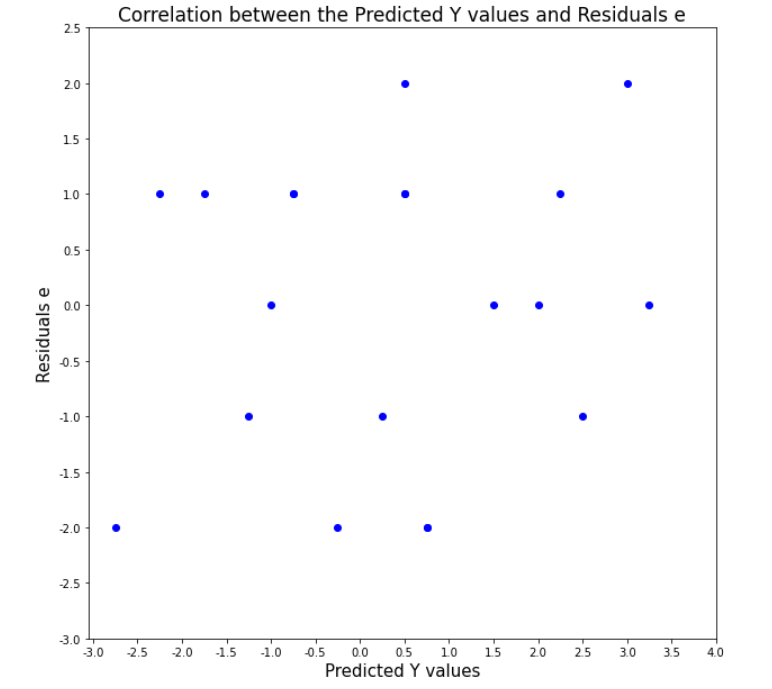
plt.xticks(ticks = np.arange(-3.0, 4.5, 0.5), labels = np.arange(-3.0, 4.5, 0.5)) *#Range for X axis: -3 to 4, increased by 0.5 each*

plt.yticks(ticks = np.arange(-3, 3.0, 0.5), labels = np.arange(-3, 3.0, 0.5)) *#Range for Y axis: -3 to 2.5, increased by 0.5 each*

plt.title("Correlation between the Predicted Y values and Residuals e", size = 17) *#Chart title, font size: 17*

plt.show()

**(e)**



*#Statement to answer 1e*

X axis: Predicted Y values

Y axis: Residuals e

Based on the scatter plot, I disagree that the constant variance assumption is valid. This is because not all points on the Y axis consist of readings of 2.0

For example, there is only 1 point at X = -3, and Y = 2, there are no other points where X = -3, hence we cant determine the difference.

Another example is at the point when X = 0, and Y = -2 , and X = 0, Y = 2. The difference between both points is 4.

Lastly, there is a point when x = 2, y = -1 , and another point where x = 2, y = 0 , the difference between both points is 1.

The constant variance assumption ensures that the furthest x value plotted against the y value should be constant. Since all 3 examples from the scatter plot were not constant, the constant variance assumption was not valid.

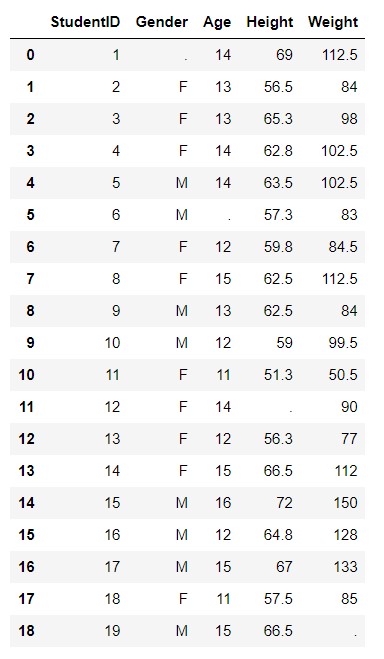
**Question 2**

**(a)** *#Question 2a*

import pandas as pd

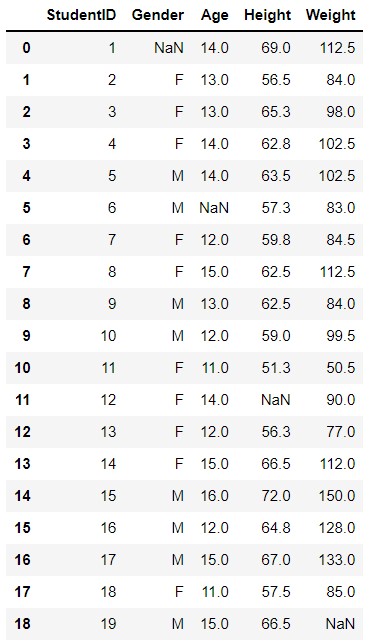
class\_data=pd.read\_csv("class.csv") *#import .csv file to pandas DataFrame*

display(class\_data) *#to display output and check for missing values*



class\_data=pd.read\_csv("class.csv",na\_values=".") *#to identify "." as a missing value*

display(class\_data) *#to see the final table with "." identified as missing value*

****

**(b)** *#Question 2b*

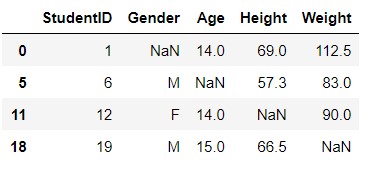
class\_data.sort\_values(by = ["Age","Gender"],ascending=[False,True]) *#sort by age in descending and gender in ascending*



**(c)** *#Question 2c*

missrow=class\_data.isnull().any(axis=1)

class\_data.loc[missrow[missrow==True].index] *#to identify rows and columns with missing values*

****

**(d)** *#Question 2d*

gender\_mode = class\_data['Gender'].mode() *#to identify mode of gender*

class\_data['Gender'].fillna(gender\_mode,inplace=True) *#to replace missing gender with the mode*

age\_median=class\_data['Age'].median() *#to calculate median of age*

class\_data['Age'].fillna(age\_median,inplace=True) *#to replace missing age with the median*

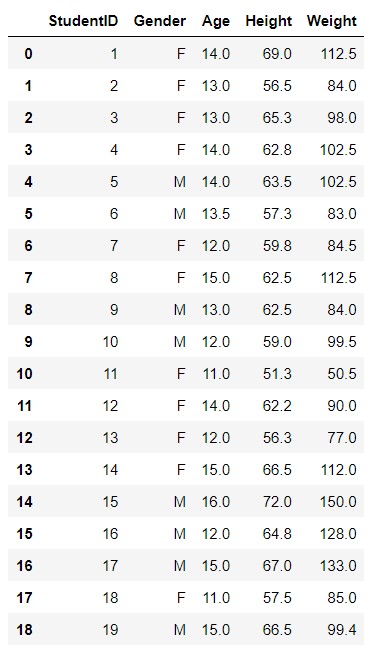
height\_mean = class\_data['Height'].mean() *#to calculate mean of height*

class\_data['Height'].fillna(height\_mean,inplace=True) *#to replace missing height with the mean*

weight\_mean=class\_data['Weight'].mean() *#to calculate mean of weight*

class\_data['Weight'].fillna(weight\_mean,inplace=True) *#to replace missing weight with the mean*

class\_data.round(1) *#to create a new df with rounding to 1 dp*



**(e)** *#Question 2e*

age\_q1 = round(class\_data["Age"].quantile(q=0.25), 2) *#to calculate q1*

age\_q3 = round(class\_data["Age"].quantile(q=0.75), 2) *#to calculate q3*

age\_iqr = round(age\_q3 - age\_q1, 2) *#to calculate interquartile range*

age\_low = round(age\_q1 - 1.5 \* age\_iqr, 2) *#to identify lower bound*

age\_upp = round(age\_q3 + 1.5 \* age\_iqr, 2) *#to identify upper bound*

print(f"AGE\nq1:{age\_q1}\nq3: {age\_q3}\ninterquartile range: {age\_iqr}\nlower: {age\_low}\nupper: {age\_upp}") *#to print results of calculation*

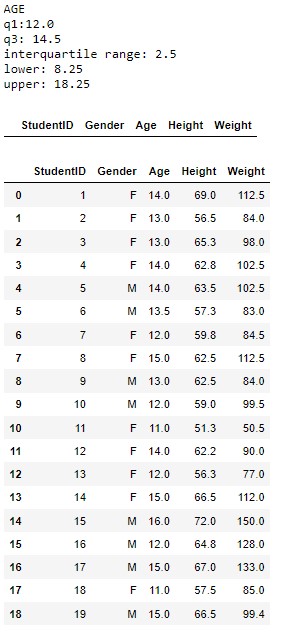
class\_data\_age\_outlier = class\_data[(class\_data["Age"] < age\_low) | (class\_data["Age"] > age\_upp)] *#to view the outliers*

class\_data\_1 = class\_data[~(class\_data["Age"] < age\_low) | (class\_data["Age"] > age\_upp)].round(1) *#to detect and remove outliers if any*

display(class\_data\_age\_outlier) *#to display the outliers if any*

display(class\_data\_1) #*to display the new dataframe without outliers from age*

**(e)** *#Age*



height\_q1 = round(class\_data\_1["Height"].quantile(q=0.25), 2) *#to calculate q1*

height\_q3 = round(class\_data\_1["Height"].quantile(q=0.75), 2) *#to calculate q3*

height\_iqr = round (height\_q3 - height\_q1, 2) *#to calculate interquartile range*

height\_low = round(height\_q1 - 1.5 \* height\_iqr, 2) *#to identify lower bound*

height\_upp = round(height\_q3 + 1.5 \* height\_iqr, 2) *#to identify upper bound*

print(f"HEIGHT\nq1:{height\_q1}\nq3: {height\_q3}\ninterquartile range: {height\_iqr}\nlower: {height\_low}\nupper: {height\_upp}") *#to print results of calculation*

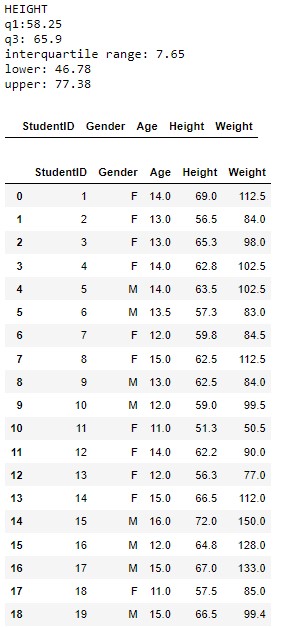
class\_data\_height\_outliers = class\_data\_1[(class\_data\_1["Height"] < height\_low) | (class\_data\_1["Height"] > height\_upp)].round(2) *#to view the outliers if any*

class\_data\_2 = class\_data\_1[~(class\_data\_1["Height"] < height\_low) | (class\_data\_1["Height"] > height\_upp)].round(2) *#to detect and remove outliers if any*

display(class\_data\_height\_outliers) *#to display the outliers*

display(class\_data\_2) *#to display the new dataframe without outliers from age and height*

**(e)** *#Height*



weight\_q1 = round(class\_data\_2["Weight"].quantile(q=0.25), 2) *#to calculate q1*

weight\_q3 = round(class\_data\_2["Weight"].quantile(q=0.75), 2) *#to calculate q3*

weight\_iqr = round(weight\_q3 - weight\_q1, 2) *#to calculate interquartile range*

weight\_low = round(weight\_q1 - 1.5 \* weight\_iqr, 2) *#to identify lower bound*

weight\_upp = round(weight\_q3 + 1.5 \* weight\_iqr, 2) *#to identify upper bound*

print(f"WEIGHT\nq1:{weight\_q1}\nq3: {weight\_q3}\ninterquartile range: {weight\_iqr}\nlower: {weight\_low}\nupper: {weight\_upp}") *#to print results of calculation*

class\_data\_weight\_outliers = class\_data\_2[(class\_data\_2["Weight"] < weight\_low) | (class\_data\_2["Weight"] > weight\_upp)] *#to view the outliers if any*

class\_data\_no\_outliers = class\_data\_2[~(class\_data\_2["Weight"] < weight\_low) | (class\_data\_2["Weight"] > weight\_upp)] *#to detect and remove outliers if any*

display(class\_data\_weight\_outliers) *#to display the outliers*

display(class\_data\_no\_outliers) *#to display the new dataframe without outliers from age, height and weight*

*#This would be the final dataframe after detecting and removing the outliers from age, height and weight*

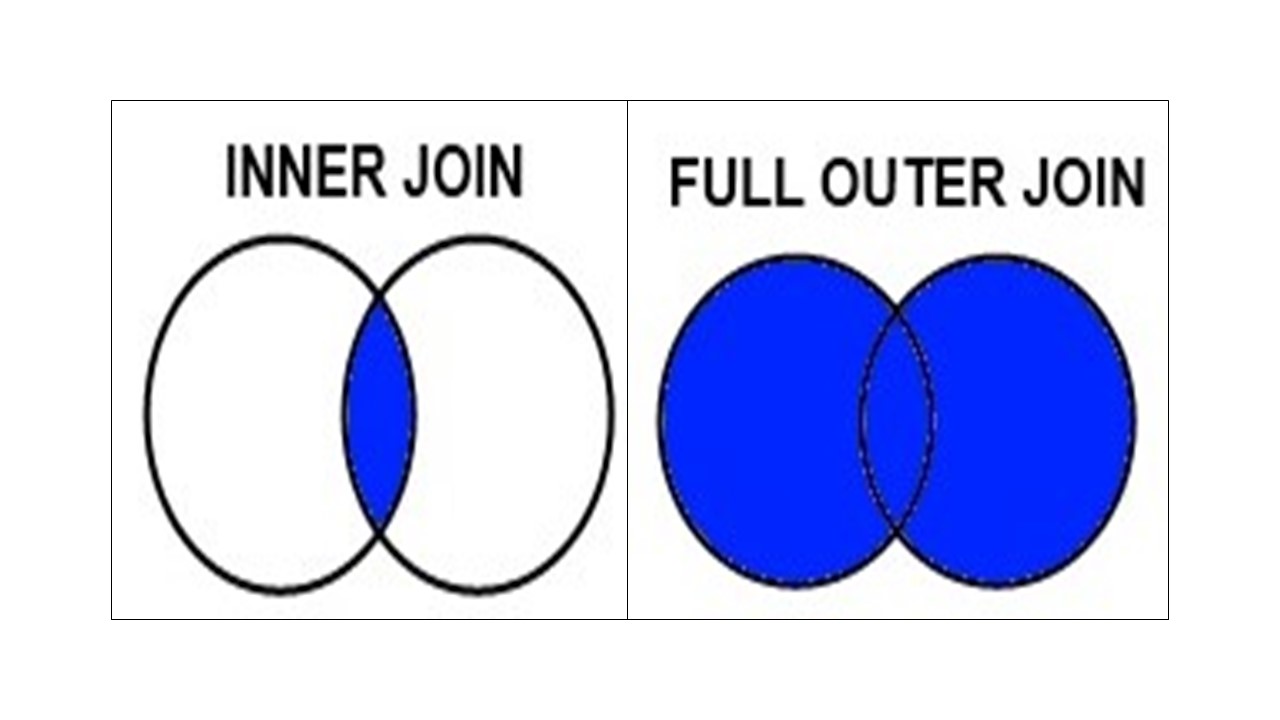
**(e)** *#Weight*



**Question 3**

***Figure 1***

*Venn Diagram*

**

*Note: Venn Diagram showing inner and outer join*

The above Venn diagrams shows the difference between the inner and outer join when merging two or more DataFrames, where the inner join shows the blue intersecting area between both circles while outer join highlights both circles in blue.

This shows that inner join returns rows that have common values from the DataFrames. While outer join returns all rows that have common values including those that do not have values in the DataFrames.

Inner join requires each row from the DataFrames to have matching columns; rows that do not have these common values are excluded in the results. While outer join returns all rows, rows that do not have values, like “NaN” in the columns, are returned.

To merge 2 or more DataFrames using pandas package, the following code can be used:

“ df\_name = pd.concat (object, axis, join) ”

“object” refers to the names of the DataFrames that will be joining. “axis” refers to the direction the DataFrames will join. “axis = 0” joins the objects vertically below the DataFrames and “axis = 1” joins the objects horizontally beside the DataFrames, with the default set to “axis = 0”.

“join” refers to the type of join used, inner or outer join, with the default as “outer”.

**(196 words)**

**References**

Wu, K. Y. (2021). ANL252 Python for data analytics (Study guide).

Singapore University of Social Sciences.