

**ANL252**

**Python for Data Analytics**

# **Group-Based Assignment**

**July 2021 Presentation**

**Submitted by:**

|  |  |
| --- | --- |
| **Name** | **PI No.** |
| **CHUNG WEI TAT** | **Y2070638** |
| **TAN SHI HAO** | **Z2011226** |
| **MICHAEL CHENG LEE BENG** | **B1981656** |
| **HO QI-WEI** | **B1972075** |

**Tutorial Group: T 09**

**Submission Date: 29/08/2021**

|  |  |  |
| --- | --- | --- |
| Name | Contribution | Signature |
| **CHUNG WEI TAT** | I did Q2 |  |
| **TAN SHI HAO** | I did Q1a,b,c |  |
| **MICHAEL CHENG LEE BENG** | I did Q1d & e |  |
| **HO QI-WEI** | I did Q3 |  |

**Question 1**

**#(a)**

import numpy as np

array\_list = 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],

])

**#(b)**

def linear\_regression(array\_list): # this function will accept the table

result = []

for row in array\_list: # indicate each row in the table

x1 = row[0] # the first value for each row is X1

x2 = row[1] # the second evaluet for each row is X2

BigY = 2 - (0.5 \* x1) + (2.5 \* x2) # the formula stated in assignment

result.append(BigY) # adding the BigY to the result list

result = np.array(result) # converting result list to a numpy array

return result # returning the numpy array

**#(c)**

def residuals(table,BigY\_array):

result = [] # making an empty list to add our results

y\_array = [] # making an empty list to every Y from the table

𝑒\_array = [] # making an empty list to add every 𝑒

for row in table: # for each row in the table

y = row[2] # the third element for each row has the y value

y\_array.append(y)

for i in range(0,len(BigY\_array)):

𝑒 = y\_array[i] - BigY\_array[i]

𝑒\_array.append(𝑒) # add result to the table

result = np.array(𝑒\_array) # converting our 𝑒̂\_array list to a numpy array

return result # returning the numpy array

BigY\_array = linear\_regression(array\_list)

print(BigY\_array)

𝑒\_array = residuals(array\_list,BigY\_array)

print(𝑒\_array)

**#(d)**

import matplotlib.pyplot as plt # for the graphs

fs = 18

plt.style.use('fivethirtyeight')

plt.hist(𝑒\_array, edgecolor='black')

plt.title('Histogram of residuals', fontsize = fs)

plt.xlabel('Residuals', fontsize = fs)

plt.ylabel('Residual frequencies', fontsize = fs)

plt.show()

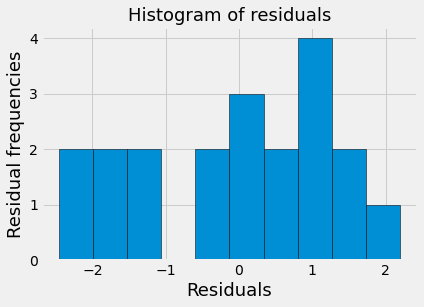
#The assumption states that if we collect many individual random samples from a

#population and calculate sample mean and then create a histogram to visualize

#the distribution of sample means, a perfect bell curve should appear.

#The histogram it does ressemble a bell as it shows that there is a curve

#going up to the peak and drop.(around residuals = 1 and 2).



**#(e)**

plt.scatter(x=BigY\_array,y=𝑒\_array)

plt.title("Scatter plot of residuals vs predicted Y")

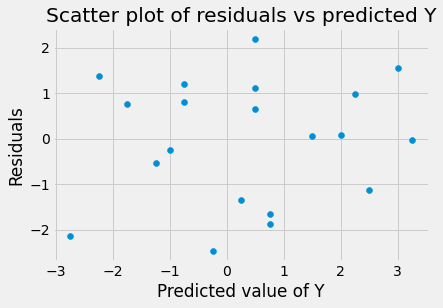
plt.xlabel("Predicted value of Y")

plt.ylabel("Residuals")

plt.show()

#It can be conclude that the constant variance assumption is met since the residuals will

#be randomly scattered around with no obvious shape/pattern.



**Question 2**

**#(a)**

**import pandas as pd**

**missing\_fields = []**

**# detect "." as na\_values and na\_filter = true convert to Nan**

**data = pd.read\_csv("class.csv", na\_values = ".", na\_filter = True)**

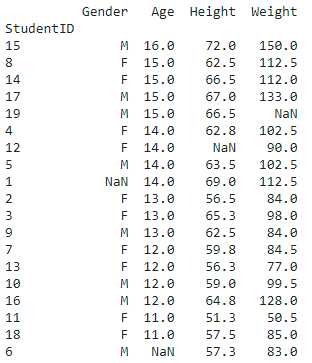
**# set StudentID as index**

**data.set\_index("StudentID", inplace = True)**

**#(b)**

**sorted\_data = data.sort\_values(by = ["Age", "Gender"], ascending = [False, True])**

**print (sorted\_data)**

****

**#(c)**

**# get all columns that has NaN**

**null\_col = sorted\_data.isnull().any(axis = 0)**

**null\_col\_index = null\_col[null\_col == True].index**

**# get all rows that has NaN**

**null\_row = sorted\_data.isnull().any(axis = 1)**

**null\_row\_index = null\_row[null\_row == True].index**

**for x in null\_row\_index:**

**for y in null\_col\_index:**

**xy\_data = sorted\_data.loc[x, y]**

**# check if value is NaN. if true store value**

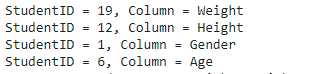
**if (pd.isna(xy\_data)):**

**print ('StudentID = {0}, Column = {1}'.format (x, y))**

**# store in global array as obj {StudentID: Column}**

**obj = {x: y}**

**missing\_fields.append(obj)**

****

**#(d)**

**# to remove NaN from each column because it affects the calculation**

**# print (sorted\_data)**

**for z in missing\_fields:**

**# key = row, value = column**

**for key, value in z.items():**

**# declare use for global**

**replace\_value = "0"**

**# if height or weight calculate mean**

**if value == "Height" or value == "Weight":**

**replace\_value = sorted\_data[value].mean()**

**# round of to 2 decimal places**

**replace\_value = round(replace\_value, 2)**

**elif (value == "Gender"):**

**# run through loop to check Gender**

**count\_m = 0**

**count\_f = 0**

**for data in sorted\_data[value]:**

**if (data == 'M'): count\_m += 1**

**if (data == 'F'): count\_f += 1**

**# replace gender with the highest frequency**

**if (count\_m > count\_f): replace\_value = "M" # set m if higher**

**if (count\_f > count\_m): replace\_value = "F" # set f if higher**

**elif (value == "Age"):**

**# replace value by the median age**

**# sort by age ascending**

**sorted\_data\_age = sorted\_data.sort\_values(by = ["Age"], ascending = [True])**

**# get age median**

**replace\_value = sorted\_data\_age['Age'].median()**

**else:**

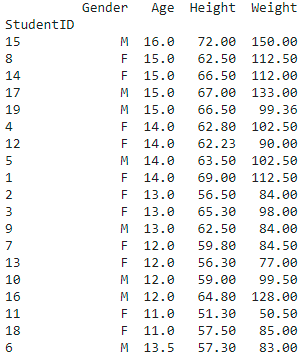
**# do nothing**

**print ("")**

**# update field**

**sorted\_data.loc[[key], [value]] = replace\_value**

**print (sorted\_data)**

****

**#(e)**

**# detect outliers**

**outlier\_studentid = []**

**outlier\_col = ['Age', 'Height', 'Weight']**

**# for each col check q1 and q3**

**for col in outlier\_col:**

**q1 = sorted\_data[col].quantile(q = .25) # 1st quantile**

**q3 = sorted\_data[col].quantile(q = .75) # 3rd quantile**

**# run through loop to see if exceed quantile**

**for key, data in sorted\_data[col].items():**

**if (data > q3 or data < q1): outlier\_studentid.append(key)**

**# remove duplicated value**

**outlier\_studentid = list(set(outlier\_studentid))**

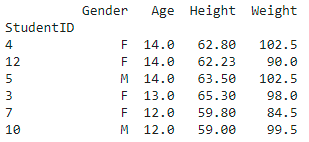
**# print (outlier\_studentid)**

**for studentid in outlier\_studentid:**

**# delete the corresponding rows**

**sorted\_data = sorted\_data.drop(studentid)**

**print (sorted\_data)**

****

**Question 3**

The merging of multiple DataFrames can be carried out using the functions concat(), merge() or join(). For example, after we import pandas as pd, we can call the function combine\_DataFrames = pd.concat(objs, axis, join) to combine DataFrames. There are 3 parameters - *objs*, *axis* and *join*. The *objs* parameter is a list with all the names of DataFrames to be concatenated. The *axis* parameter can be values 0 or 1, where the DataFrames are to be concatenated underneath one another or sideways respectively. Lastly, the *join* parameter can be a string that is either inner or outer (outer by default), thereby specifying whether it is an inner or outer join.

Inner join is the merging of multiple datasets that only keeps the matching variables; while outer join is that keeps all available variables and its rows and columns. The inner join is akin to the intersection of sets, as it entails each row to have common column values across the merging DataFrames. The outer join is akin to the union of sets, as it takes in all available data, leaving 0 or NaN where there is missing data or rows without some of its corresponding column value.

