

**ANL252**

**PYTHON FOR DATA ANALYTICS**

# **Group-Based Assignment**

**July 2021 Presentation**

**Submitted by Group 6:**

|  |  |
| --- | --- |
| **Name** | **PI No.** |
| **MUHAMMAD FARHAN BIN SAAD** | **H1771425** |
| **JIA LEONG TEE** | **M2110660** |
| **YEOW HWEE GLEN ONG** | **W2110804** |
| **MICHELLE TAN MING HUI** | **J2110299** |
| **EE HAI OON** | **Q2011729** |

**Tutorial Group: ­­­­­­­­­­­­ T09**

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

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

**Question 1(a)**

To answer this question, we have to import NumPy into the Jupyter notebook in order to execute the program.

|  |  |
| --- | --- |
| Python Program Code | Annex A- Results of output of data |
|  |  |
| import numpy as np  data = 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]]) |  |

The result under Annex A corresponds to the data in the GBA question showing 20 rows and 3 columns of X1, X2 and Y.

**Question 1(b)**

***Ŷ* = 2 - 0.5*X*1 + 2.5*X*2**

In order to solve the above equation, we have to input a 2-dimensional slicing for data X1, X2 and Ŷ so that the program could identify which data it belongs to.

# x1, x2, y

# data[:,0] -> x1

# data[:,1] -> x2

# data[:,2] -> y

# Find predicted

predicted = 2 - (0.5 \* data[:,0]) + (2.5 \* data[:,1])

predicted

In addition, we input another NumPy function ‘np.vstack’ to stack the results vertically in response to the original dataset arrangement. Results are shown under Annex B.

predicted\_vert = np.vstack(predicted)

predicted\_vert

|  |
| --- |
| Annex B - Results of output of data for predicted (or expected) value |
|  |
|  |

**Question 1(c)**

Similar to question 1(b), we have to input the 2-dimentional slicing for data Y and insert in the equation.

# x1, x2, y

# data[:,0] -> x1

# data[:,1] -> x2

# data[:,2] -> y

# Find Residual

residual = data[:,2] - predicted

residual

After getting the result in the program, we input NumPy ‘np.vstack’ to get the vertical data to correspond to the data. Results are shown under Annex C.

residual\_vert = np.vstack(residual)

residual\_vert

|  |
| --- |
| Annex C - Results of output of data for residuals |
|  |
|  |

**Question 1(d)**

The histogram shows that the residuals under Annex D indicates that the assumption of normality distribution with zero mean and constant variance is deemed valid given that the chart has a shape that that is similar to a bell-curve with a slight left skew (Zheng, 2018), with the peak at values ranging between 0 to 1 and looks symmetrically distributed with the left tail of the histogram between values of -3 to -2 having a frequency of 2 and the right tail of the histogram between values of 2 and 3 having a value of 1.

To begin plotting the histogram, we import the *pyplot* function from matplotlib which is intended to have to generate graphs and charts. We then refer to the maximum and minimum values of the residuals as a guiding principle to plot the graph. Since the rounded minimum would show -3 and the rounded maximum would be 3, we select the bins to -4 and -4 to distinguish clearly there are no values between (+/-) 3 and (+/-) 4. Afterwards, we plot the histogram by using plt.hist function along with the label axis respectively and ticks to be set between -4 and 5 (since the end-point is 4) and a step value of 1.

import matplotlib.pyplot as plt

residual\_vert.min()

residual\_vert.max()

bins\_select = np.arange(-4.0, 4.0, 1)

plt.hist(residual\_vert, bins = (bins\_select) , rwidth = 0.90, color='orange', edgecolor='black', linewidth=1)

plt.xlabel("Residuals")

plt.ylabel("Frequencies")

plt.title("Residual Chart")

plt.xticks(ticks = range(-4, 5, 1), labels = range(-4, 5, 1))

plt.show()

|  |  |  |
| --- | --- | --- |
| Annex D - Histogram of the frequencies of the Residuals and Workings of Max and Min  |  |  | | --- | --- | |  |  | |

**Question 1(e)**

The scatter plot shown under Annex E indicates that the constant variance assumption between is deemed valid since it does not show any obvious pattern in the residuals.

Since the pyplot function was imported from 1(d), we continue off by plotting the scatter plot by using the plt.scatter function with the labels in the x and y-axis. Additionally, we have added in a red line in the zero value for under the y-axis as indicated in the lesson recording (Zheng, 2018) to further assist us to look for indications of any patterns in the scatter plot. We have established for y-axis to be set between -4 and 5 (since the end-point is 4) and a step value of 1 as explained from 1(d). For x-axis, the predicted values can be set as -3 to 4 after deriving the maximum and minimum values. However, since there is not a huge increase in the figures, we can set it as -4 and 5 instead (since the end-point is 4).

predicted\_vert.max()

predicted\_vert.min()

plt.scatter(predicted\_vert, residual\_vert, color = "orange", marker = "o", edgecolor = "black")

plt.axhline(y = 0, color = 'red', linestyle = '-')

plt.xlabel("Predicted Value")

plt.ylabel("Residual")

plt.xticks(ticks = range(-4, 5, 1), labels = range(-4, 5, 1))

plt.yticks(ticks = range(-4, 5, 1), labels = range(-4, 5, 1))

plt.title("Scatter Plot Pattern of Predicted vs Residual for Constant Variance Assumption (Checks)")

plt.show()

|  |  |  |
| --- | --- | --- |
| Annex E – Scatter Plot of Predicted Value against Residuals Workings of Max and Min  |  |  | | --- | --- | |  |  | |

**Question 2(a)**

**Description:**

We read in the raw dataset as a csv file, and display the dataset to analyse the values of the raw csv file. We have observed that the missing values in the dataset is populated by “.”. Thus, we adjust pandas read-in configuration by determining the na values as “.”.

**Script:**

import pandas as pd

# import raw .csv to analyse

df\_class\_raw = pd.read\_csv("class.csv")

display(df\_class\_raw)

# import raw .csv with na key value

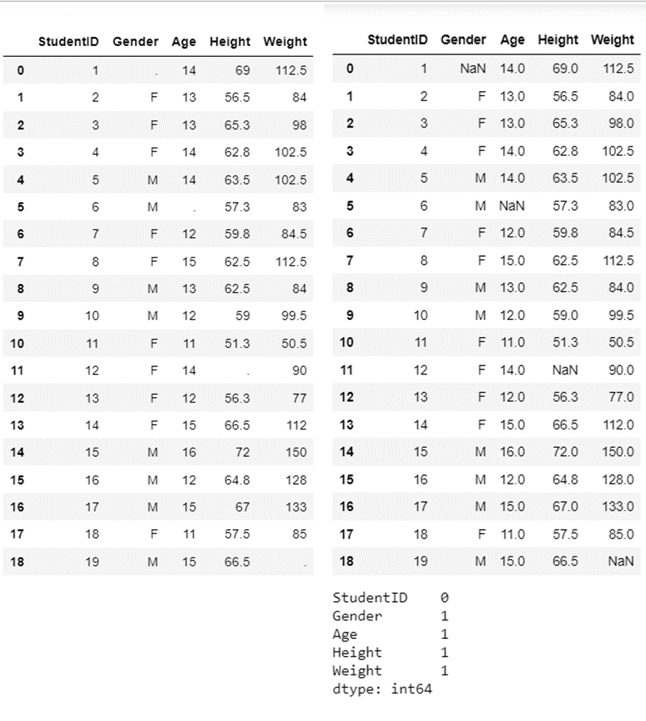
df\_class = pd.read\_csv("class.csv", na\_values = ".")

display(df\_class)

# summary of the cleaned up .csv

display(df\_class.isnull().sum())

**Output:**



**Question 2(b)  
  
  
Description:**

We leveraged on a group-by parameter to group the values of age and gender together. This function is then enclosed within a sort value function with ascending equals to false and true respectively. This is to sort age by descending order and gender by ascending order. Lastly, an inplace parameter is used to ensure that the variables are sorted at a record level.

**Script:**

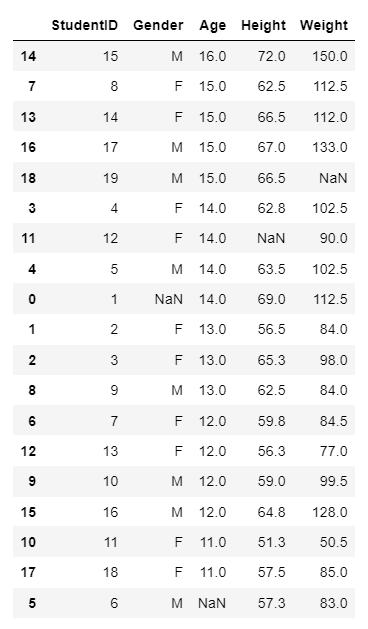
# multi-field sort on the cleaned up .csv

# Age descending followed by Gender ascending

df\_class.sort\_values(by = ["Age", "Gender"], ascending = [False, True], inplace =True)

display(df\_class)

**Output:**



**Question 2(c)**

**Description:**

We leverage on a null function from pandas and a sum function to identify the total number of na values in each field. We then independently report the record ID of the na values for each field that has na count more than 0. Finally, we sliced the dataset to display all of the records containing na values.

**Script:**

# Summary of the total count of na values

display(df\_class.isnull().sum())

# Record ID of Gender na values

print("\n")

print("Record ID and na value under Gender")

df\_class\_GenderNull = df\_class.Gender.loc[df\_class.Gender.isnull()]

display(df\_class\_GenderNull)

# Record ID of Age na values

print("\n")

print("Record ID and na value under Age")

df\_class\_AgeNull = df\_class.Age.loc[df\_class.Age.isnull()]

display(df\_class\_AgeNull)

# Record ID of Height na values

print("\n")

print("Record ID and na value under Height")

df\_class\_HeightNull = df\_class.Height.loc[df\_class.Height.isnull()]

display(df\_class\_HeightNull)

# Record ID of Weight na values

print("\n")

print("Record ID and na value under Weight")

df\_class\_WeightNull = df\_class.Weight.loc[df\_class.Weight.isnull()]

display(df\_class\_WeightNull)

# Location of the na values

print("\n")

display(df\_class[df\_class.isnull().any(axis=1)])

# Create a new function:

def null\_count(df\_class):

return sum(df\_class.isnull())

print("\n")

print("\n")

# Applying per column:

print("Summary of column null(s):")

print("\n")

print(df\_class.apply(null\_count, axis=0))

print("\n")

print("\n")

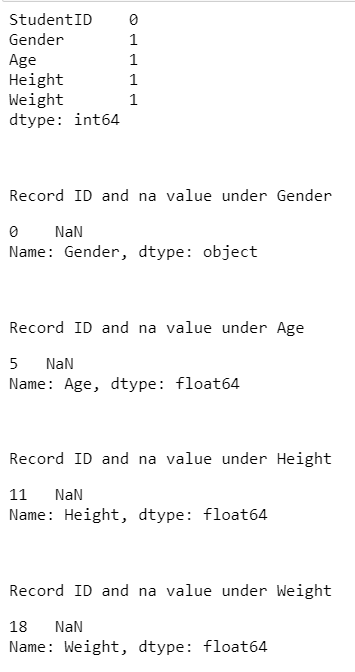
# Applying per row:

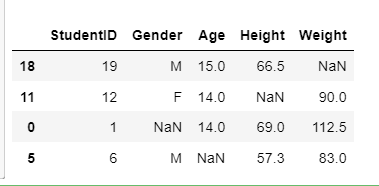
print("Summary of row null(s):")

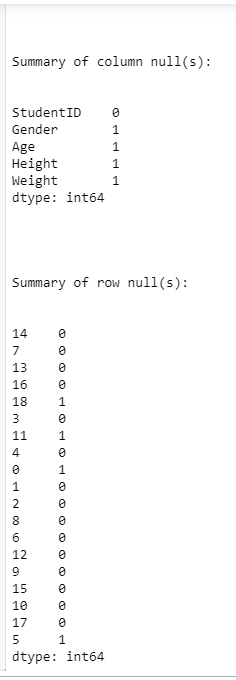
print("\n")

print(df\_class.apply(null\_count, axis=1))

**Output:**







**Question 2(d)**

**Description:**

We leverage a mode, median and mean function from pandas to determine the mode of the gender values, the median of the age values, and the mean of the height values and the weight values. This is enclosed within a fill na function from pandas to replace the na values of the respective fields with the corresponding derived values.

Finally, the respective mode, median and mean for the corresponding fields are printed out and the records that previously contained na values were sliced out from the dataset after the replacement. These two reports are visually analysed to ensure that the na values have been replaced accordingly.

**Script:**

# Deriving mean, median and mode for the respective fields

df\_class\_Gendermode = df\_class["Gender"].mode().at[0]

df\_class\_Agemedian = df\_class["Age"].median()

df\_class\_Heightmean = df\_class["Height"].mean()

df\_class\_Weightmean = df\_class["Weight"].mean()

# Replacing na values to the respective required values

df\_class[["Gender"]] = df\_class[["Gender"]].fillna(df\_class.mode())

df\_class[["Age"]] = df\_class[["Age"]].fillna(df\_class.median())

df\_class[["Height"]] = round(df\_class[["Height"]].fillna(df\_class.mean()),1)

df\_class[["Weight"]] = round(df\_class[["Weight"]].fillna(df\_class.mean()),1)

display(df\_class)

# Printing mean, median and mode for the respective fields

print("\n")

print(f"The mode of genders is {df\_class\_Gendermode}")

print(f"The median of the ages is {df\_class\_Agemedian}")

print(f"The mean of the heights is {df\_class\_Heightmean:.1f}")

print(f"The mean of the weights is {df\_class\_Weightmean:.1f}")

# Selecting previous fields with na values for comparision

display(df\_class.loc[[18, 11, 0, 5], :])

**Output:**



**Question 2(e)**

**Description:**

The bar chart from the plot function is leveraged from pandas to visualise categorical values such as gender. This is to detect and identify outliers that does not belong to either the “M” or “F” category.

On the other hand, the interquartile is leveraged to detect and identify numerical values such as age, weight and height. Additionally, the upper bound and lower bound is determined for each of the numerical type fields to identify individual field outliers that are found beyond the upper bound and lower bound for each of the numerical type fields. Finally, the outliers are excluded by returning the values found within the upper and lower bound range.

**Script:**

# Plotting a bar chart to detect outlier for gender

df\_class['Gender'].value\_counts().plot(kind='bar', xlabel="Gender", ylabel="Frequency", title="Count by gender", color="turquoise", rot=0)

# stating q1 and q3

q1 = 0.25

q3 = 0.75

# determining 1st and 3rd quartiles of Age

Age\_q1 = df\_class["Age"].quantile(q1)

Age\_q3 = df\_class["Age"].quantile(q3)

Age\_iqr = Age\_q3 - Age\_q1

# determining outliers for Age

Age\_y1 = Age\_q1 - 1.5 \* Age\_iqr

Age\_y2 = Age\_q3 - 1.5 \* Age\_iqr

# determining 1st and 3rd quartiles of Height

Height\_q1 = df\_class["Height"].quantile(q1)

Height\_q3 = df\_class["Height"].quantile(q3)

Height\_iqr = Height\_q3 - Height\_q1

# determining outliers for Height

Height\_y1 = Height\_q1 - 1.5 \* Height\_iqr

Height\_y2 = Height\_q3 - 1.5 \* Height\_iqr

# determining 1st and 3rd quartiles of Weight

Weight\_q1 = df\_class["Weight"].quantile(q1)

Weight\_q3 = df\_class["Weight"].quantile(q3)

Weight\_iqr = Weight\_q3 - Weight\_q1

# determining outliers for Weight

Weight\_y1 = Weight\_q1 - 1.5 \* Weight\_iqr

Weight\_y2 = Weight\_q3 - 1.5 \*Weight\_iqr

# displaying outliers for age field

df\_Age\_outlier = df\_class[~(

(df\_class["Age"] < Age\_y1) | (df\_class["Age"] > Age\_y2)

)]

# displaying outliers for height field

df\_Height\_outlier = df\_class[~(

(df\_class["Height"] < Height\_y1) | (df\_class["Height"] > Height\_y2)

)]

# displaying outliers for weight field

df\_Weight\_outlier = df\_class[~(

(df\_class["Weight"] < Weight\_y1) | (df\_class["Weight"] > Weight\_y2)

)]

print("Outliers for age field:")

display(df\_Age\_outlier)

print("\n")

print("\n")

print("Outliers for height field:")

display(df\_Height\_outlier)

print("\n")

print("\n")

print("Outliers for weight field:")

display(df\_Weight\_outlier)

print("\n")

print("\n")

# excluding outliers from the dataset for Age

df\_class\_nooutlier = df\_class[~(

(df\_class["Age"] > Age\_y1) & (df\_class["Age"] < Age\_y2)

)]

# excluding outliers from the dataset for Height

df\_class\_nooutlier = df\_class[~(

(df\_class["Height"] > Height\_y1) & (df\_class["Height"] < Height\_y2)

)]

# excluding outliers from the dataset for Weight

df\_class\_nooutlier = df\_class[~(

(df\_class["Weight"] > Weight\_y1) & (df\_class["Weight"] < Weight\_y2)

)]

# displaying non-outliers within the dataset

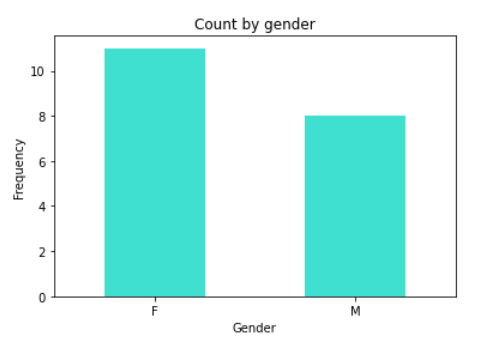
print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")

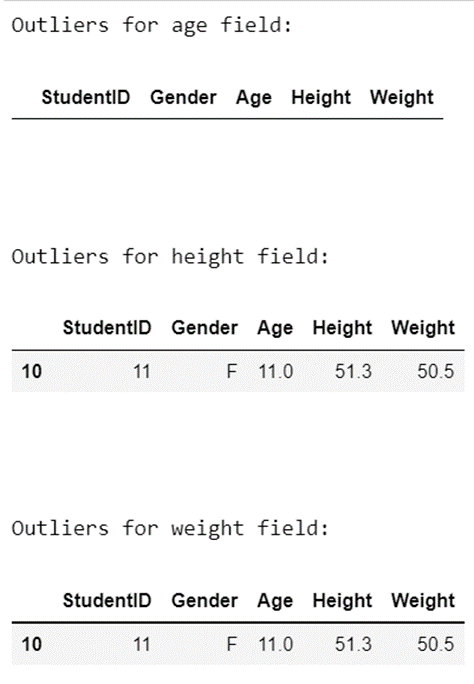
print("\n")

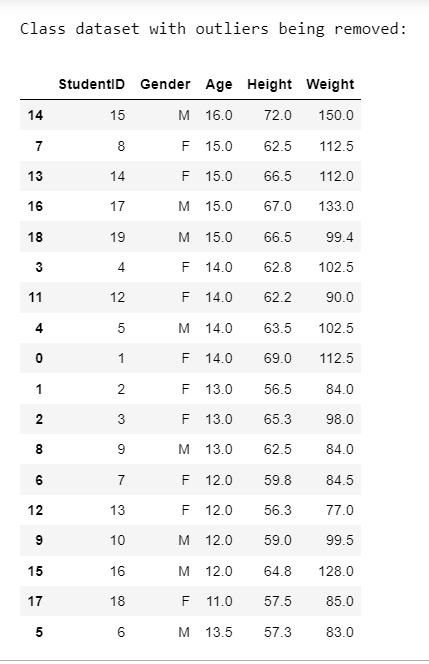
print("Class dataset with outliers being removed:")

display(df\_class\_nooutlier)

**Output:**



****



**Question 3**

An Inner join combines DataFrames based on a join key and only return a new DataFrame that only keeps row that have matching values in the original DataFrames while the outer join does the opposite and returns all the combination of rows from the original DataFrames, NaN will be resulted in fill where data is missing on the original DataFrames however this join type is rarely used. There is also left and right outer join. Left join will not lose information from its DataFrames and returns all rows from the left DataFrame, the non-matching rows from the other DataFrames will contain NaN for the result. For the right join, it does the opposite and returns all rows from the right DataFrames. The pandas function for performing joins is called merge. For inner join, we will have to pass inner in the how argument of the merge() function to perform the join. For outer join, we will have to pass outer in the how argument of the merge() function to perform the join. For left or right join, we will still use the merge() function but we pass either left or right in the how argument.

*(195 words)*

**References:**

1. Zheng, F.(2018).Assumptions for Linear Regression [Video]. Singapore: Singapore University of Social Sciences.

<https://d2jifwt31jjehd.cloudfront.net/BUS105/LessonRecording/BUS105_SU03CH02T03_V2_0/presentation_html5.html>

1. Combining DataFrames with Pandas – Data Analysis and Visualization in Python for Ecologists. (2021). Retrieved 26 August 2021, from <https://datacarpentry.org/python-ecology-lesson/05-merging-data/>

**Declaration**

|  |  |  |
| --- | --- | --- |
| Name | Contribution | Signature |
| **MUHAMMAD FARHAN BIN SAAD** | I did Q1 | Text, letter  Description automatically generated |
| **JIA LEONG TEE** | I did Q1 | Text, letter  Description automatically generated |
| **YEOW HWEE GLEN ONG** | I did Q2 | A black and white drawing of a pair of glasses  Description automatically generated with low confidence |
| **MICHELLE TAN MING HUI** | I did Q2 | A picture containing arthropod, invertebrate  Description automatically generated |
| **EE HAI OON** | I did Q3 |  |