**ANL252**

**Python for Data Analytics**

**Group-based Assignment (GBA01)**

**July 2021 Presentation**

**T09**

**Group 8**

|  |  |
| --- | --- |
| Brandon Tan Xuan Ming | Y1970855 |
| Cheong Jia En, Sandras | N1981099 |
| Muhammad Syairazi Bin Mahdhar | Z1970078 |

Submission Date:

27 August 2021

**Question 1**

**(a)**

# Importing Numpy and Matplotlib

import numpy as np

import matplotlib.pyplot as plt

# Creating array\_1 to store the observed data.

array\_1 = 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],

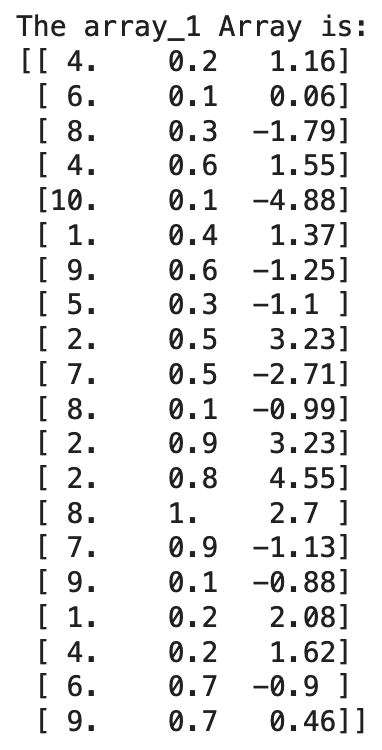
[4, 0.2, 1.62],

[6, 0.7, -0.9],

[9, 0.7, 0.46]])

# Printing array\_1

print(f'The array\_1 Array is: \n{array\_1}')



**(b)**

# Let y\_hat be the predicted value of y.

# Let y\_hatarr be an initialised 1 x 20 array to store all the y\_hat values.

y\_hatarr = np.zeros(20)

# The loop here calculates the y\_hat value from each row in array\_1, and stores it in y\_array.

t\_1 = 0

while t\_1 < 20:

for a in array\_1:

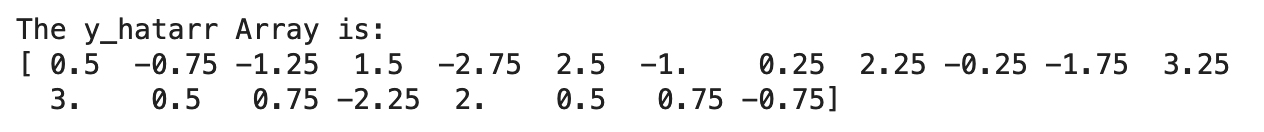
y\_hat = 2 - 0.5\*a[0] + 2.5\*a[1]

y\_hatarr[t\_1] = y\_hat

t\_1 += 1

# Printing y\_hatarr

print(f'The y\_hatarr Array is: \n{y\_hatarr}')



**(c)**

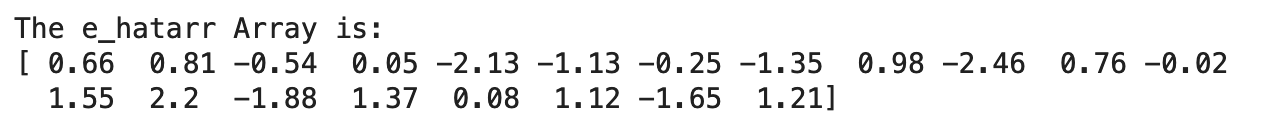
# Let e\_hat be the value when the y column in array\_1 subtracts each y\_hatarr value.

# Let e\_hatarr be the array to store each e\_hat value.

e\_hatarr = array\_1[0:, 2] - y\_hatarr

# Printing e\_hatarr

print(f'The e\_hatarr Array is: \n{e\_hatarr}')



**(d)**

# Generating the Histogram using e\_hatarr for the Residual Distribution.

plt.hist(e\_hatarr, range = (-2.5, 2.5), bins = 10, rwidth = 0.8, color = 'darkblue', align = 'mid')

plt.xlabel('Residual Values')

plt.ylabel('Frequencies')

plt.title('Residual Distribution')

plt.xticks(ticks = [-2.5, -2.0, -1.5, -1.0, -0.5, 0, 0.5, 1.0, 1.5, 2.0, 2.5],

labels = [-2.5, -2.0, -1.5, -1.0, -0.5, 0, 0.5, 1.0, 1.5, 2.0, 2.5])

# Displaying the Histogram

plt.show()

# Finding the mean of the Residual Distribution via the Numpy function.

# We then compare the mean to 0, and print our findings.

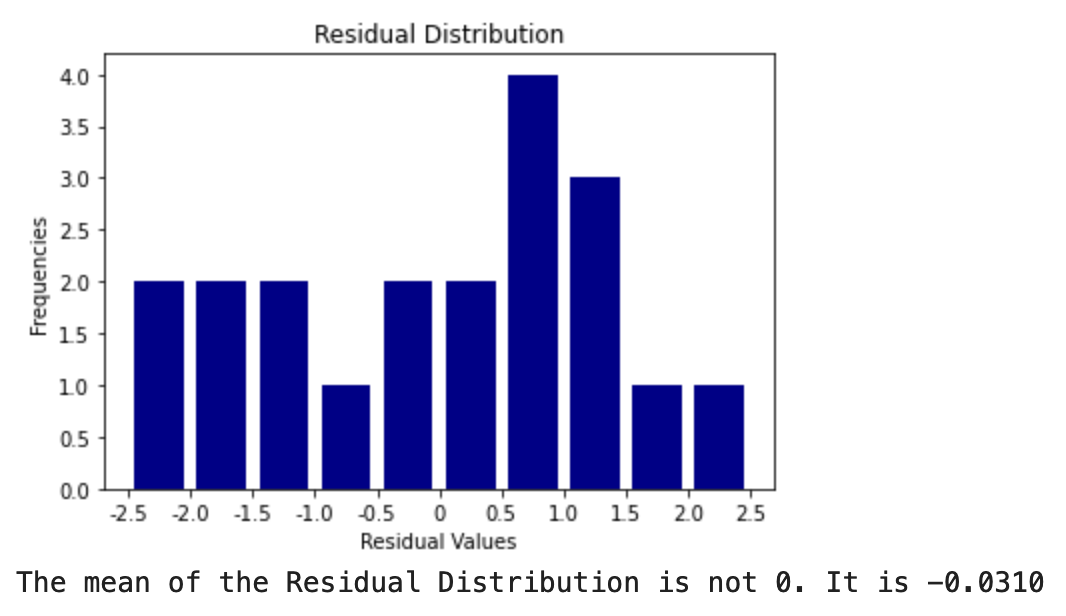
mean = np.mean(e\_hatarr)

if mean == 0:

print('The mean of the Residual Distribution is 0.')

else:

print(f'The mean of the Residual Distribution is not 0. It is {mean:.4f}')



**(e)**

# Generating a Scatter Plot using values from y\_hatarr as the x-axis and the values from e\_hatarr as the y-axis.

plt.scatter(y\_hatarr, e\_hatarr, color = 'red', marker = 'o', edgecolor = 'black')

plt.xlabel('Predicted Values, Ŷ')

plt.ylabel('Residual Values, ê')

plt.title("Correlation between Predicted Values Ŷ and Residual Values ê")

plt.xticks(ticks = [-4.0, -3.0, -2.0, -1.0, 0, 1.0, 2.0, 3.0, 4.0],

labels = [-4.0, -3.0, -2.0, -1.0, 0, 1.0, 2.0, 3.0, 4.0])

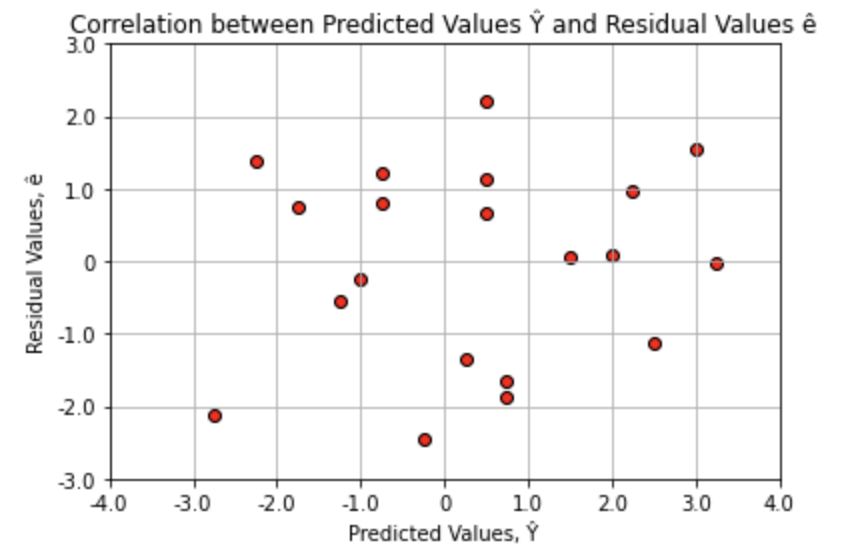
plt.yticks(ticks = [-3.0, -2.0, -1.0, 0, 1.0, 2.0, 3.0],

labels = [-3.0, -2.0, -1.0, 0, 1.0, 2.0, 3.0])

plt.grid(True)

# Displaying the Scatter Plot

plt.show()



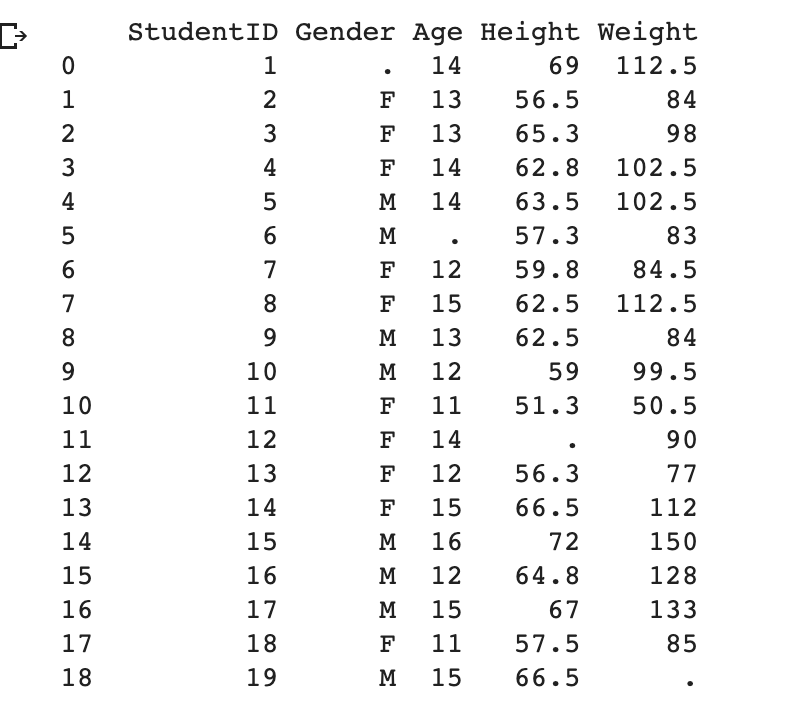
The scatter plot does not have any regular pattern, and neither are all the points lying on the same level. To the right, the extreme point is near (3, 0). To the left, the extreme point is near (-3, -2). To the top, the extreme point is near (0.5, 2). And to the bottom, the extreme point is near (0, -2.5). In between, the other points are scattered without any bias toward a specific direction. Therefore, we can conclude that there is no constant variance.

**Question 2**

import pandas as pd

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

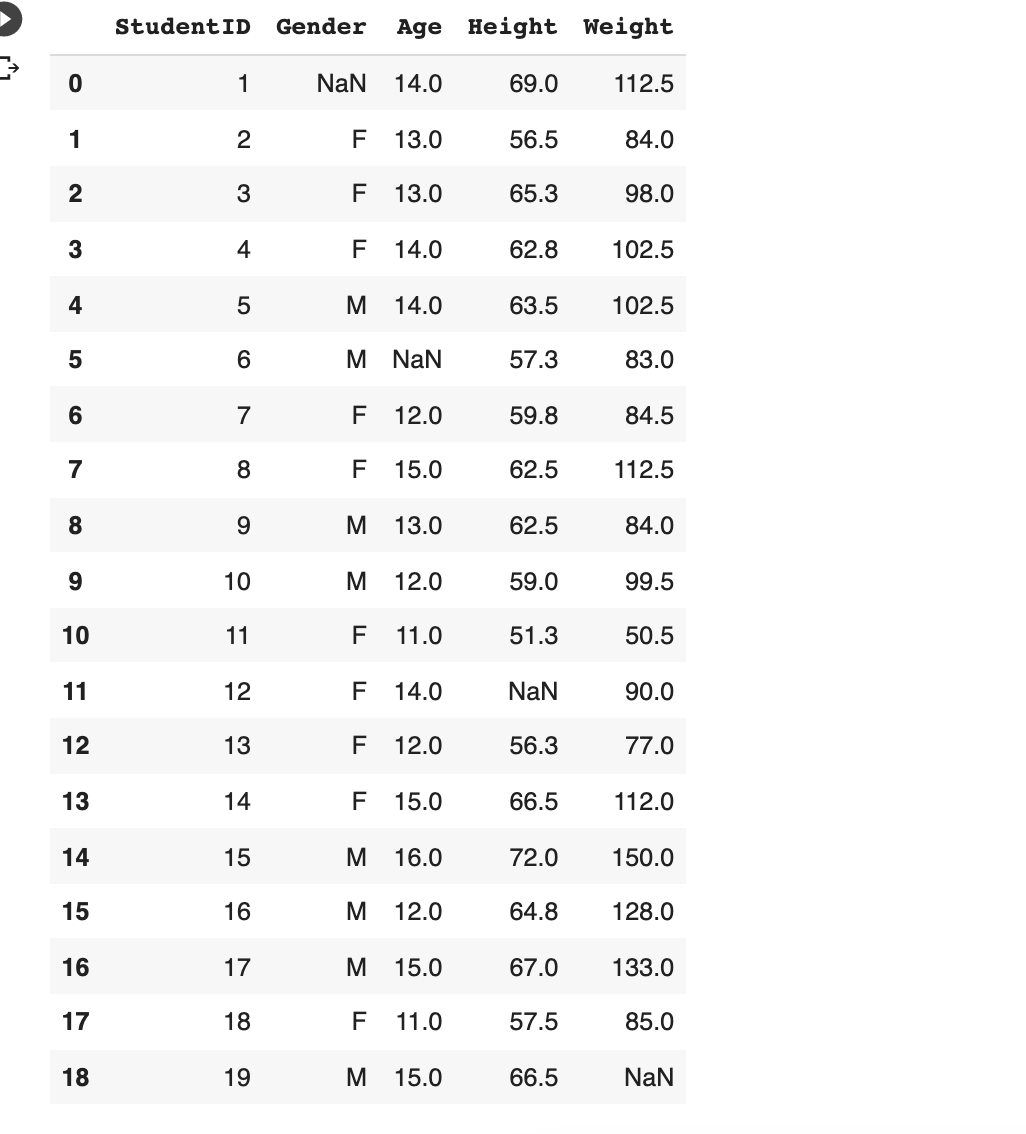
df\_class



# Specify missing values that were "." to NaN

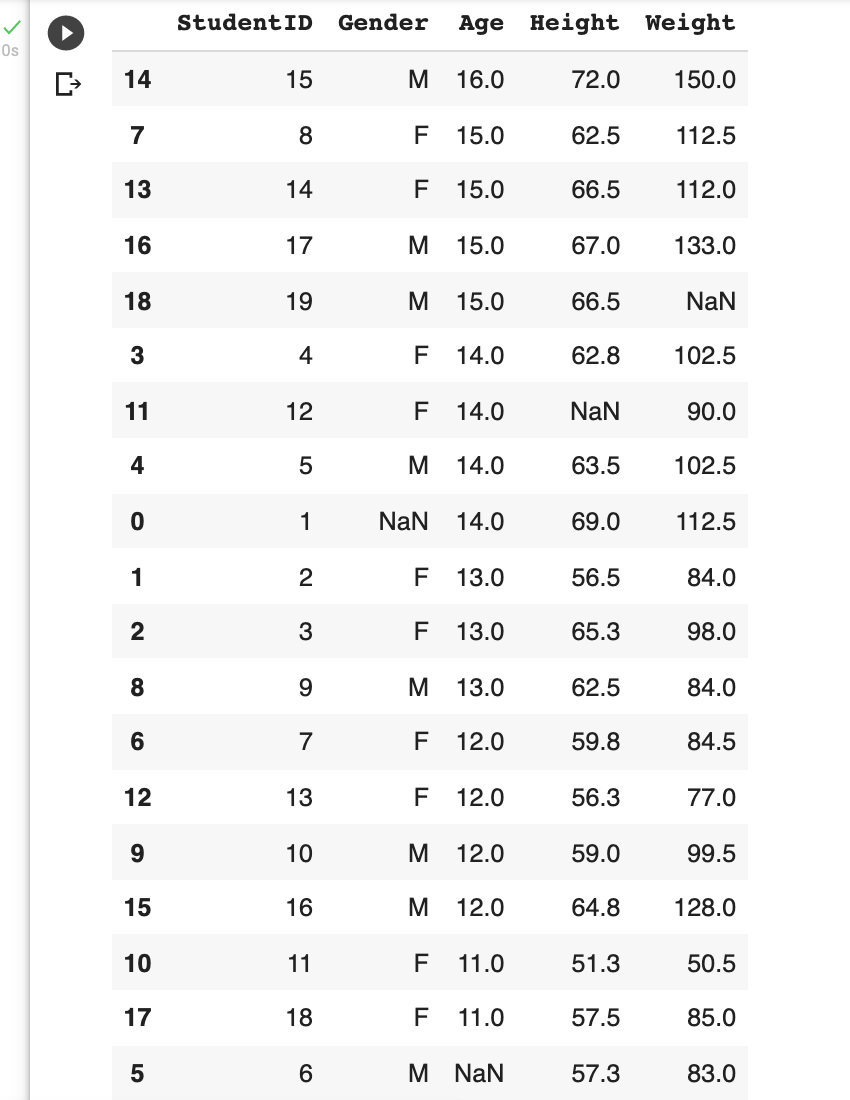
df\_class = pd.read\_csv("class.csv", na\_values = ".")

display(df\_class)



# Sorting Age in descending order and Gender in ascending order (alphabetical order)

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



import numpy as np

# Check every column and row and return the sum of Nan for each column and each row

column = df\_class.isnull().sum(axis = 0)

row = df\_class.isnull().any(axis = 1)

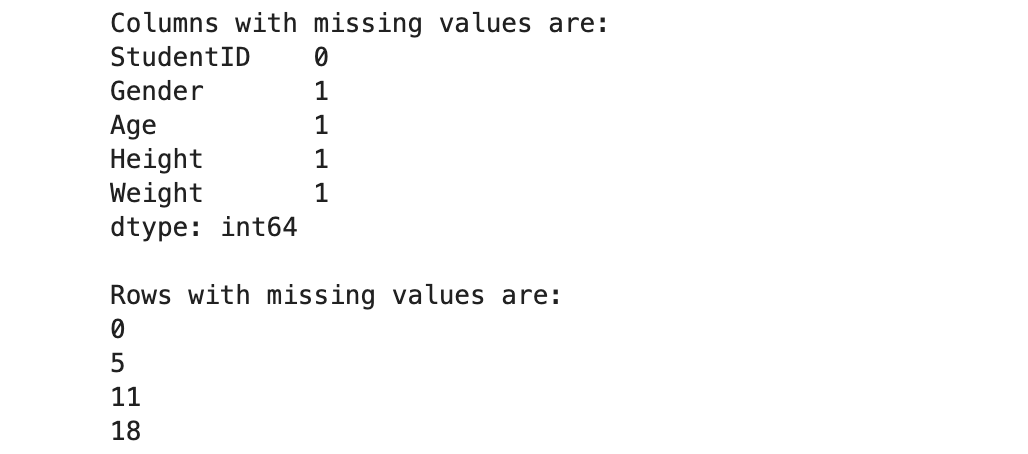
missingrow = np.where(row)[0]

print(f"Columns with missing values are: \n{column}\n")

print(f"Rows with missing values are: ")

for i in missingrow:

print(i)



The output above shows the columns and rows with missing values. Therefore, missing data are found in the columns “Gender”, “Age”, “Height” and “Weight”, and in rows 0, 5, 11 and 18.

# Replace missing values by gender with highest frequency

df\_class["Gender"].fillna(df\_class["Gender"].mode()[0], inplace = True)

# Replace missing values by median age

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

# Replace missing values by mean height

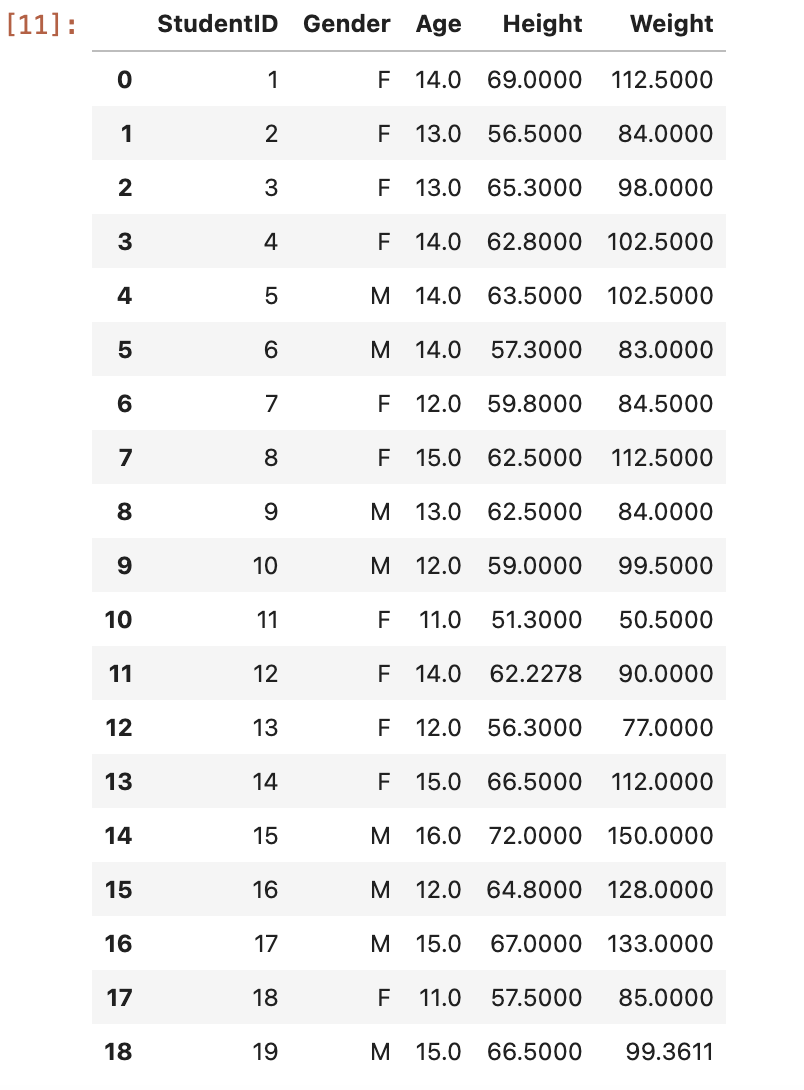
df\_class["Height"] = df\_class["Height"].fillna(df\_class["Height"].mean())

# Replace missing values by mean weight

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

# Rounding

df\_class.round({"Age": 0, "Height": 4, "Weight": 4})



# Detecting outliers with Interquartile Range (IQR)

# Calculating IQR for each column

q1 = df\_class.quantile(0.25)

q3 = df\_class.quantile(0.75)

iqr = q3 - q1

print(f"{iqr}\n")

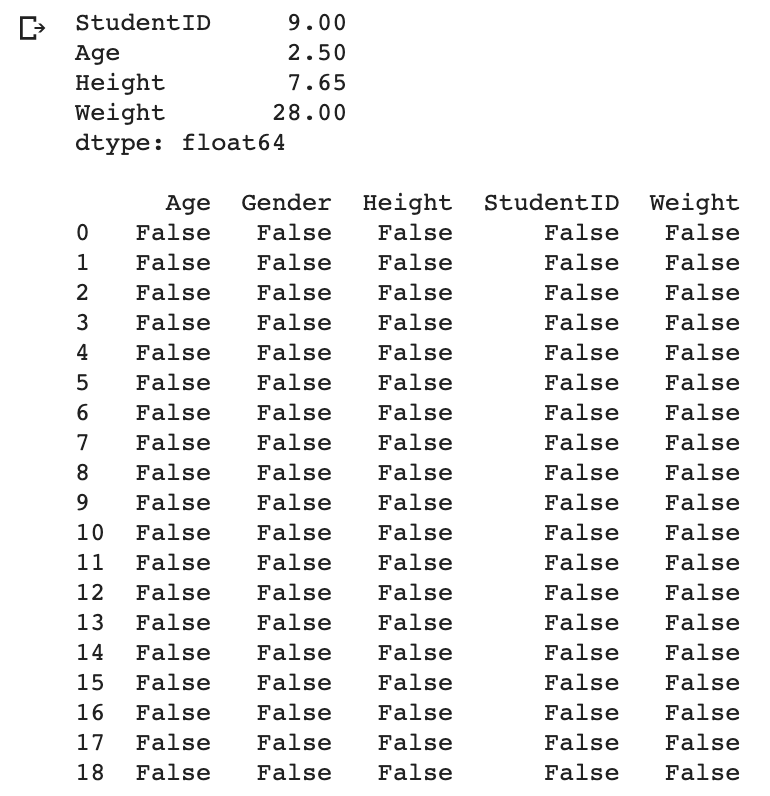
# True indicates presence of an outlier

# False indicates the values are valid

lowerbound = q1 - 1.5 \* iqr

upperbound = q3 + 1.5 \* iqr

print((df\_class < lowerbound) | (df\_class > upperbound))



The returned output indicates that there are no outliers detected as evident where all values are valid and represented by “False”. Therefore, there is no need to delete any rows.

**Question 3**

An **inner join** between two dataframes involves intersecting their datasets while an **outer join** involves the union of the two dataframes.Inner join returns a dataframe with only rows that have matching keys in the joint table (DataScience Made Simple, 2020). A full outer join returns all rows and joins records of matching keys in the two tables where Nan is returned when there is no match (DataScience Made Simple, 2020). Likewise, a **left join** will return all rows from the left table and any rows with matching keys from the right table where Nan will be returned if there is no match, and vice versa for a **right join** (DataScience Made Simple, 2020).

The various joins can be performed with a **merge() function** which is part of the pandas package, taking both dataframes as arguments (Sharma, 2020). It does an inner join by default where other joins i.e., outer, left, right can be specified with the ‘how’ parameter, and users have control over which columns to combine on with the ‘on’ parameter (Stratis, n.d.). ‘left\_on’ and ‘right\_on’ are arguments applicable when there is a need to specify different column names in the two dataframes (Sharma, 2020).

(197 words)

**References**

DataScience Made Simple. (2020, December 24). *Join in Pandas: Merge data frames (inner, outer, right, left join) in pandas python*. Retrieved from<https://www.datasciencemadesimple.com/join-merge-data-frames-pandas-python/>

Sharma, A. (2020, February 27). *Joins in Pandas: Master the Different Types of Joins in Python*. Retrieved from<https://www.analyticsvidhya.com/blog/2020/02/joins-in-pandas-master-the-different-types-of-joins-in-python/>

Stratis, K. (n.d.). *Combining Data in Pandas With merge(), .join(), and concat()*. Retrieved from<https://realpython.com/pandas-merge-join-and-concat/>