

**ANL252**

**Python for Data Analytics**

# **Group-Based Assignment**

**July 2021 Presentation**

**Submitted by:**

|  |  |
| --- | --- |
| **Name** | **PI No.** |
| **Yeo Ker Woon, Jan** | **E1881560** |
| **Tong Yew Tiong Alex** | **H0914367** |
| **Tia Kang Jun** | **B2070385** |
| **Danial Loke Zhi Weng** | **Y1981340** |

**Tutorial Group: T09**

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

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

|  |  |  |
| --- | --- | --- |
| Name | Contribution | Signature |
| **Yeo Ker Woon, Jan** | The team started the project individually before meeting to discussed on the details. Putting in the pieces as a team by merging the different parts. All members played a unique role, with stronger ones helping to enhance the final project with better codes. | Jan yeo |
| **Tong Yew Tiong Alex** | Alex Tong |
| **Tia Kang Jun** | Tia Kang Jun |
| **Danial Loke** | Danial Loke |

**Qn1a)**

|  |  |
| --- | --- |
| **Code:** | import numpy as np  # Create lists for X1, X2 and Y  X1 = [4, 6, 8, 4, 10, 1, 9, 5, 2, 7, 8, 2, 2, 8, 7, 9, 1, 4, 6, 9]  X2 = [0.2, 0.1, 0.3, 0.6, 0.1, 0.4, 0.6, 0.3, 0.5, 0.5, 0.1, 0.9, 0.8, 1, 0.9, 0.1, 0.2, 0.2, 0.7, 0.7]  Y = [1.16, 0.06, -1.79, 1.55, -4.88, 1.37, -1.25, -1.1, 3.23, -2.71, -0.99, 3.23, 4.55, 2.7, -1.13, -0.88, 2.08, 1.62, -0.9, 0.46]  # Create a Numpy Array with the lists above.  # The array will be according to the list, thus 3 rows and 20 columns.  # We have to swap the rows and columns by transposing the array using .T to fit the original data given.  X1\_X2\_Y = np.array([X1, X2, Y]).T  print("The values of X1, X2 and Y are: ")  display(X1\_X2\_Y)  # Checking the shape of the array to see if it matches the shape of the original data.  X1\_X2\_Y\_shape = X1\_X2\_Y.shape  print(f"The shape of the combined array is:\n{X1\_X2\_Y\_shape}") |
| **Output:** |  |

**Qn1b)**

|  |  |
| --- | --- |
| **Code:** | # The linear regression is Y\_hat = 2 - 0.5\*(X1) + 2.5\*(X2).  # The array of the predicted values will have 1 rows and 20 columns.  # We have to swap the rows and columns by transposing the array using .T to match the data given.  Y\_hat = np.array([2 - 0.5\*X1\_X2\_Y[:,0] + 2.5\*X1\_X2\_Y[:,1]]).T  print("The values of Ŷ for the observed values of X1 and X2 are: ")  display(Y\_hat)  # Checking the shape of the array, which matches the shape of the original data.  Y\_hat\_shape = Y\_hat.shape  print(f"The shape of the Ŷ array is:\n{Y\_hat\_shape}") |
| **Output:** |  |

**Qn1c)**

|  |  |
| --- | --- |
| **Code:** | # the expression is ê = Y - Ŷ  # The array of the residuals will have 1 rows and 20 columns.  # We have to swap the rows and columns by transposing the array using .T to match the data given.  e\_hat = np.array([X1\_X2\_Y[:, 2] - Y\_hat[:, 0]]).T  print("The values of ê are:")  display(e\_hat)  # Checking the shape of the array, which matches the shape of the original data.  e\_hat\_shape = e\_hat.shape  print(f"The shape of the ê array is:\n{e\_hat\_shape}") |
| **Output:** |  |

**Qn1d)**

|  |  |
| --- | --- |
| **Code:** | import matplotlib.pyplot as plt  # Plotting of histogram  # To read x-axis clearly, create a list for bins with increment of 1 using np.arange().  # Additionally, need to change the location of the xticks to fit the bins by using plt.xticks.  # Adjust edge colour to black so that the x-axis can be read clearly as well.  bin = np.arange(-3, 3.5, 1)  plt.hist(e\_hat, bins = bin, align = 'mid', edgecolor = 'black')  plt.xticks(bin)  plt.title("Histogram of residuals\n", fontsize = 20, fontweight = 'bold')  plt.xlabel("\nResiduals of the model ê", fontsize = 15)  plt.ylabel("Frequency of Values\n", fontsize = 15)  plt.show() |
| **Output:** |  |

The normality assumption with zero mean is valid based on this histogram. This is because the shape of the histogram represents somewhat of a bell shape, thus indicating a normal distribution.

**Qn1e)**

|  |  |
| --- | --- |
| **Code:** | # Plot scatter plot where the x-axis represents Ŷ and the y-axis represents the residuals ê.  # Change the location of the xticks to fit the data by using plt.xticks.  # Change the location of the yticks to fit the data by using plt.yticks.  # Plot a zero line which represents the zero mean using plt.axhline.  plt.scatter(Y\_hat, e\_hat)  plt.xticks(np.arange(-3.5, 4, 1))  plt.yticks(np.arange(-2.5, 3, 1))  plt.axhline(y = 0, color = 'black')  plt.title("Scatter Plot\nConstant Variance Assumption\n", fontsize = 20, fontweight = 'bold')  plt.xlabel("\nPredicted values Ŷ", fontsize = 15)  plt.ylabel("Residuals of the model ê\n", fontsize = 15)  plt.show() |
| **Output:** |  |

The constant variance assumption is valid based on this scatter plot. This is true because the plot above seems to be randomly scattered around the mean of zero, thus the scatter plot is not showing any patterns. Furthermore, the values of all the data points are more or less on the same level.

**Qn2a)**

|  |  |
| --- | --- |
| **Code:** | import pandas as pd  # The missing values in the data are ".", thus need to mention it under na\_values.  class\_df = pd.read\_csv("class.csv", na\_values = ".")  class\_df |
| **Output:** |  |

**Qn2b)**

|  |  |
| --- | --- |
| **Code:** | # Sorting 'Age' of the students in descending order and then by their 'Gender' in ascending order.  # After sorting, the index is re-arranged, thus use reset\_index() to drop and reset the index.  class\_df.sort\_values(by = ['Age', 'Gender'], ascending = [False, True], inplace = True)  class\_df.reset\_index(drop = True, inplace = True)  display(class\_df) |
| **Output:** |  |

**Qn2c)**

|  |  |
| --- | --- |
| **Code:** | # Use a for loop to display all the missing values at different rows and columns.  for item in class\_df.columns:  #if loop will only if there is any null item in the columns of the dataframe  if class\_df[item].isnull().any():  # Use .loc to locate the missing values to display them  display(class\_df.loc[class\_df[item].isnull()])  print(f"There is a missing value at \ncolumn: {item}\nrow/index: {class\_df[class\_df[item].isnull()].index.tolist()}") |
| **Output:** |  |

**Qn2d)**

|  |  |
| --- | --- |
| **Code:** | # Calculate the values and store them as variables.  # Gender – replace missing values by the gender with the highest frequency.  Gender\_mode = class\_df["Gender"].mode()[0]  # Age – replace missing values by the median age.  # Use int() for age because age must be an integer.  Age\_median = int(class\_df["Age"].median())  # Height – replace missing values by the mean height.  # We round the height to 1 decimal place as the values in the dataframe is in 1 decimal place  Height\_mean = round(class\_df["Height"].mean(), 1)  # Weight – replace missing values by the mean weight.  # We round the weight to 1 decimal place as the values in the dataframe is in 1 decimal place  Weight\_mean = round(class\_df["Weight"].mean(), 1)  # Replace missing values in 4 columns with 4 different values.  # Create new data frame to note the cleaned data.  class\_df\_clean = class\_df.fillna({"Gender": Gender\_mode,  "Age": Age\_median,  "Height": Height\_mean,  "Weight": Weight\_mean})  #display the cleaned dataframe  display(class\_df\_clean) |
| **Output:** |  |

**Qn2e)**

|  |  |
| --- | --- |
| **Code:** | # variable a is the first quartile and variable b is the third quartile.  a = 0.25  b= 0.75  # Create a list for the columns that might have outliers.  col\_name = ["Age", "Height", "Weight"]  # Use a for loop to loop the required columns into the outlier forumla.  for item in col\_name:  q1 = class\_df\_clean[item].quantile(q = a)  q3 = class\_df\_clean[item].quantile(q = b)  iqr = q3 - q1    y1 = round(q1 - 1.5\*iqr, 1)  y2 = round(q3 + 1.5\*iqr, 1)    # Display the outliers after each loop to show the outliers for each column.  print(f"For column {item}, the outliers are shown in the dataframe below: ")  display(class\_df\_clean[((class\_df\_clean[item] < y1) | (class\_df\_clean[item] > y2))])  print("\n"\*2)    # Rename the dataframe after the removal of the outliers  class\_df\_clean2 = class\_df\_clean[~((class\_df\_clean[item] < y1) | (class\_df\_clean[item] > y2))]  # Display the dataframe with the removed outliers: class\_df\_clean2  print("The final dataframe after removing all the outliers can be seen below: ")  display(class\_df\_clean2) |
| **Output:** |  |

**Qn3)**

The output dataset of an outer join is the union of the involved DataFrames. This means that the output dataset includes all available rows/columns across all DataFrames. Accordingly, as some of the rows/columns across the involved DataFrames would not match, there will be missing values (NaN) in the output dataset for the respective rows/columns that did not match.

On the contrary, the output dataset of an inner join is the intersection of the involved DataFrames. This means that the output dataset includes only the common rows/columns across all DataFrames. Accordingly, there will be no missing values in the output dataset as the rows/columns across the involved DataFrames matches.

To merge two or more DataFrames using inner/outer join, the pd.concat() function in the pandas package can be used. These are the most common parameters of this function: pd.concat(objs, axis, join). ‘Objs’ requires the user to specify all the DataFrames to be merged in a list format. ‘axis’ determines whether the DataFrames will merge below (axis = 0) or beside each other (axis = 1), with default 0. ‘join’ determines the type of join to take place (join = “inner” / join = “outer”), with default “outer”.

**Word count: 195 words**