

**ANL252**

**Python For Data Analytics**

**End-of-Course Assessment**

**July Semester 2022**

**Submitted by:**

|  |  |
| --- | --- |
| **Name** | **PI No.** |
| Shawn Tan Jin Chuan | J2172224 |

**Tutorial Group:** T09

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

**Submission Date:** 05 September 2022

**Question 1**

Categorical variables refer to a category or type. In this dataset, the categorical variables are ID, GENDER, EDUCATION, MARITAL, S(n), and RATING.

Numerical variables refer to values that can be measured, or the number holds a numerical meaning. In this dataset, the numerical variables are LIMIT, BALANCE, INCOME, AGE, B(n), and R(n).

**Question 2**

The 4 data pre-processing tasks are identifying duplicate entries, missing values, extreme values, and consistent data type.

Duplicate Entries

To do proper data cleansing, it is important to remove any duplicate entries within the dataset. The credit facility might go through multiple channels to collect data and this might lead to them collecting more than one record of a customer. Another reason for existence of duplicate entries in the dataset might simply be human error where the data entry clerk inputs the same record of someone more than once. Hence, to eliminate the possible occurrence of such scenario, duplicate entries must be identified. To identify all duplicate entries within the dataset, the code is:

ecadf.loc[ecadf.duplicated()]

This displays a 3 rows x 24 columns dataframe, showing that there are 3 duplicate entries within the dataset. It is crucial to remove these additional rows as the addition of these entries can potentially ruin the splitting of data into train and test sets, as the duplicate rows might not end up in the same set. This often leads to inaccuracy and biasness when training and testing the model. To drop the rows, the code is:

ecadf.drop\_duplicates(inplace = True)

To do a check whether the entries are correctly dropped and removed, the following code could be used to do a check:

print(len(ecadf))

print('----------')

print(ecadf.loc[ecadf.duplicated()])

The 3 rows are being dropped as the len of dataset changed from 18769 to 18766.

Missing Values

For proper data cleansing, it is important to identify missing or null values within the dataset. This might be a result of customers giving incomplete information to the credit facility. As part of data wrangling, these entries should be treated so that the datasets could be used to provide more valuable and accurate insights. For numerical values, the missing values could be replaced with the mean or if not being removed from the dataset. For categorical values, the missing values could be replaced with the mode, if not being removed from the dataset. To identify the sum of null values in each column, the following code could be used:

ecadf.isnull().sum(axis = 0)

From this, we can identify that EDUCATION has 13 entries with missing values and MARITAL has 38 entries with missing values. To prevent the absence of these values from affecting the accuracy and performance of the model, the entries could be removed from the dataset. This could be done so by the code:

ecadf = ecadf.dropna(axis = 0)

This will drop all entries within the dataset with NA values and a check could be done with this:

ecadf.isnull().sum(axis = 0)

OR

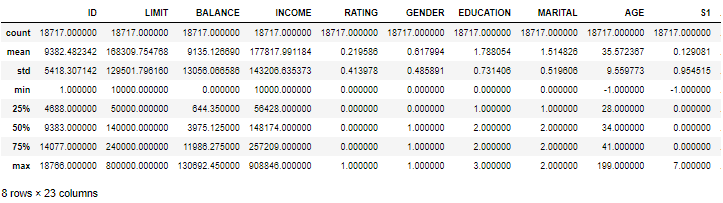
len(ecadf)

Checking the len of the dataframe returns the remaining number of rows within the dataset. This way, we would be able to tell if the correct number of rows are being dropped. The number of entries left is 18717, according to the len of the dataframe.

Extreme Values

To identify and sieve out extreme values, the following code could be used:

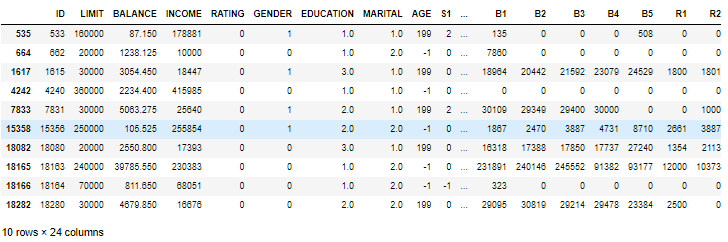
ecadf.describe()



From the output, we can see that the minimum for age is -1 and maximum is 199.

For values of age, it is impossible for someone to less than 0 years old. Based on research conducted by the University of Washington (Kim Eckart, 2021), it is very less likely that someone could live past 130 years old. Hence, based on this piece of information, 0 and 130 would be the threshold for values minimum and maximum age allowed in our dataset. First step would be to determine the presence of these extreme values by using the following code:

ecadf.loc[(ecadf['AGE'] < 1) + (ecadf['AGE'] > 130)]



There are 10 entries being identified with extreme values. As seen from the table, the impossible ages present within the dataset is either -1 or 199. This might result from customers keying in their wrong date of birth or a data entry error by one of the workers. Such extreme values or outliers can be replaced by the mean value of the age to ensure that they do not mess up with the model. The mean age could be derived from:

meanAge = round(ecadf['AGE'].mean())

print('Average Age: ', meanAge)

The average age is found out to be 36 and all the records with age -1 and 199 could be replaced with this value by using the following code:

ecadf['AGE'] = ecadf['AGE'].replace([-1,199], meanAge)

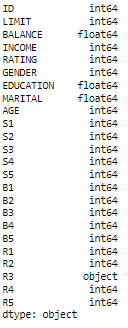
This will replace all the extreme ages in the dataset with the mean age. A check could be done with this code:

ecadf.loc[(ecadf['AGE'] < 1) + (ecadf['AGE'] > 130)]

Consistent Data Type

Before processing on the dataset, it is important that the data types are all checked through to see what type values we are going to be passing into our model. This could be done so by:

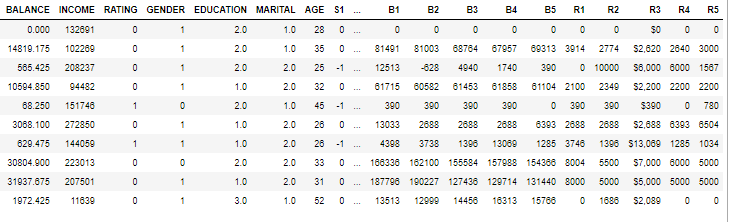
ecadf.dtypes



This is the output being produced. From here, it is evident that there is an abnormality within the dataset because the R(n) variables should produce the same data type. In this case, some were integers, while R3 is an object. To drill further in, we could display all the values of no numeric data within the column R3. This could be done so by the following code:

ecadf[ecadf['R3'].str.isnumeric().astype(str).str.contains('False')]

This creates and output of:



From the dataframe, it is evident that the presence of dollar and comma signs is what is causing the data to be stored as an object, instead of an integer. To standardize the format across all R(n) columns, the signs could be removed by using the code:

ecadf['R3'] = ecadf['R3'].str.replace('$', '')

ecadf['R3'] = ecadf['R3'].str.replace(',', '')

This essentially removes all the dollar and comma signs under R3, leaving only the numeric values behind. Next step is top convert the data type of R3 from an object to integer by using the code:

ecadf['R3'] = pd.to\_numeric(ecadf['R3'])

The absence of the dollar and comma sign will allow the successful conversion of R3 into a numerical column. The values will hence be stored as an integer.

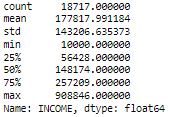
**Question 3**

Histogram

To create a histogram, the bins must be created first. To derive the bins, the minimum and maximum value should be determined. For INCOME, the data could be represented by:

ecadf['INCOME'].describe()

This will print out the output:



Hence, out minimum value is 10000 and maximum value is 908846. The bins we create should range from value 0 to 1000000 with an interval of 100000, showing all the income tiers. The bins could be created by the following code:

incomeBin = np.arange(0, 1000000, 100000)

After creating the bin for income, we could plot the histogram with the code:

plt.figure(figsize=(12,8))

plt.hist(ecadf['INCOME'], bins=incomeBin)

plt.title('Histogram of Income', fontsize = 18)

plt.xlabel('Income Amount', fontsize = 14)

plt.ylabel('Number of Customers', fontsize = 14)

plt.xticks(ticks=incomeBin)

plt.show()

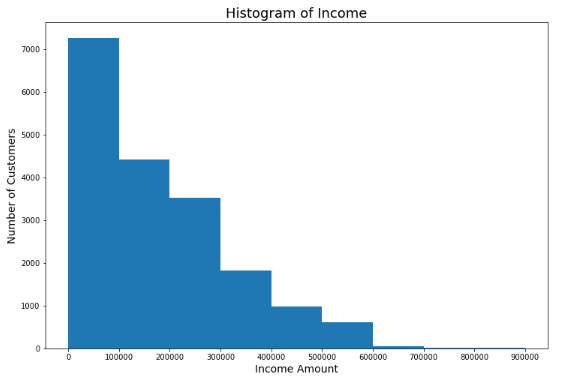


Figure : Histogram of Income

The histogram was being generated from the income of all customers within the credit facility company. It displays the summary of the annual wage of customers at a glance. From figure 1, it is evident that the income distribution is right skewed, which meant that there is a higher frequency of customers drawing a salary that is leaning towards the left side. This suggests that most of the customers earn lower salary as a large distribution is earning between $0 to $100000.

Bar Chart

A bar chart could be used to show the total number of customers of each marital status. This will allow the credit facility company to better understand the demographics of its customers. The bar chart could be plot using:

#Plot Bar Chart

Others = sum(ecadf['MARITAL']==0)

Single = sum(ecadf['MARITAL']==1)

Married = sum(ecadf['MARITAL']==2)

y\_bar = ('Others','Single','Married')

x\_bar = (Others,Single,Married)

bar = plt.figure(figsize =(8, 8))

plt.bar(y\_bar,x\_bar, edgecolor="black")

plt.xlabel("Marital Status", fontsize = 14)

plt.ylabel("Number of Customers", fontsize = 14)

plt.title("Marital Status of Customers", fontsize = 18)

#Show Plot

plt.show()

The output of bar chart produced is:

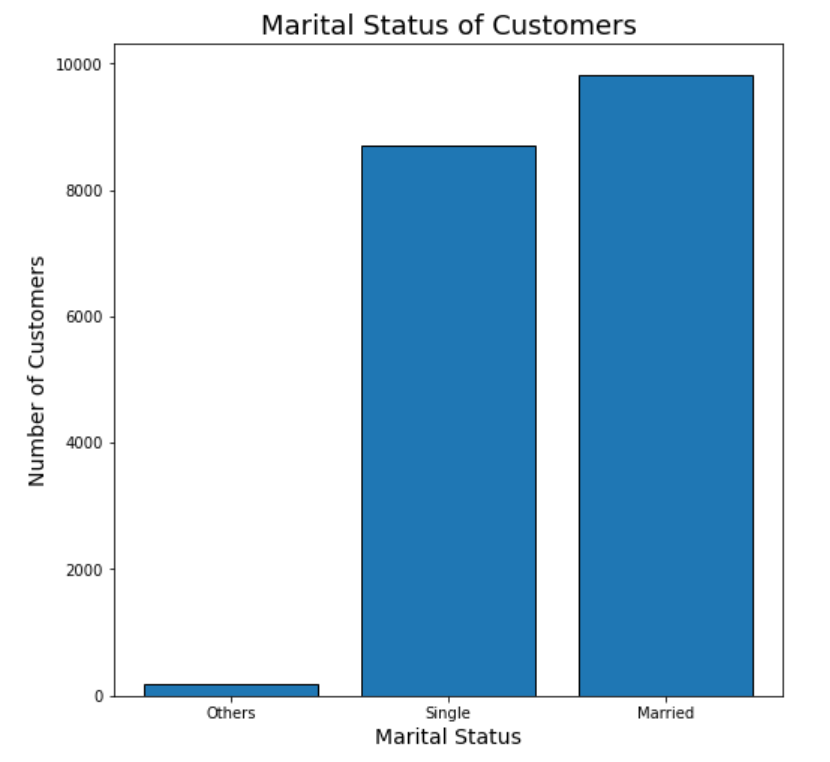


Figure : Bar Chart of Marital Status of Customers

From here, we could tell that there are more customers who are married than single. This could allow the credit facility to come with loan plans suited for married couples to cater more to their needs. Hence, knowing the credit facility pool of customers and their marital status could potentially allow the company to come up with attractive and suitable loan plans and attract more customers.

Clustered Bar Chart

A clustered bar chart could be used to display the credit rating of customers, while further breaking them down into their gender. The code to plot a clustered bar chart is:

#Plot Clustered Bar Chart

fig, ax = plt.subplots(figsize=(8,8))

ecadf\_clustered = ecadf.groupby('GENDER')['RATING'].value\_counts().unstack().fillna(0).astype(int)

ecadf\_clustered.plot(kind='bar', ax=ax, edgecolor="black")

plt.xlabel('Gender', fontsize=14)

plt.xticks(rotation=0)

plt.ylabel('Number of Customers', fontsize=14)

plt.title('Credit Rating by Gender', fontsize=18)

#Show Plot

plt.show()

The output of the clustered bar chart is:

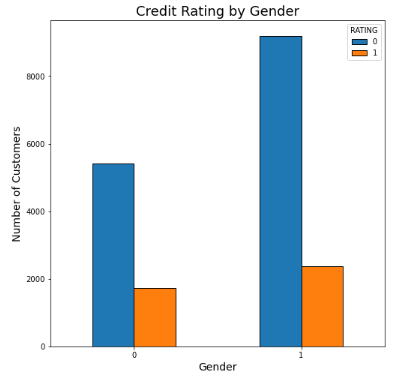


Figure : Clustered Bar Chart of Credit Rating by Gender

From figure 3, we can tell that there are more female customers than male customers within the credit facility dataset. As for males, there are more than 5000 people with a credit rating of 0 and less than 2000 people with a credit rating of 1. For females, there are more than 9000 people with a credit rating of 0 and slightly more than 2000 people with a credit rating of 1. This tells the company about the credit statuses of all the existing customers and it is being grouped by gender for further studies and possible analysis. This will essentially affect the customers’ borrowing capability and the company would be able to at one glance know how much they are able to lend in totality.

Pie Chart

A pie chart could be used to show the different education level of customers as there are 4 main categories as seen from the data dictionary. The code for this is as follows:

#Plot Pie Chart

Others = sum(ecadf['EDUCATION']==0)

Postgraduate = sum(ecadf['EDUCATION']==1)

Tertiary = sum(ecadf['EDUCATION']==2)

HighSchool = sum(ecadf['EDUCATION']==3)

y\_pie = 'Others', 'Postgraduate', 'Tertiary', 'HighSchool'

x\_pie = [Others, Postgraduate, Tertiary, HighSchool]

def function(percentage):

return "{:1.1f}%".format(percentage)

fig = plt.figure(figsize =(8,8))

plt.pie(x\_pie, labels = y\_pie, autopct=lambda percentage: function(percentage),explode=[1,0,0,0])

plt.title('Education Level of Customers', fontsize=18)

#Show Plot

plt.show()

The output of the pie chart is:

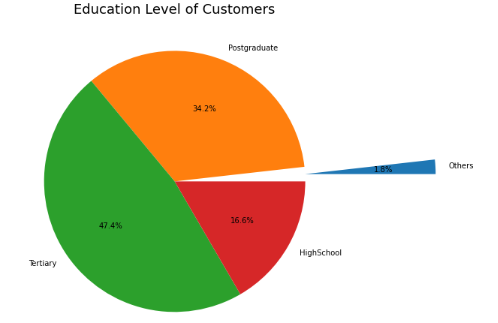


Figure : Pie Chart of Education Level of Customers

From figure 4, it is evident that a large percentage of 47.4% of the customers from the credit facility dataset attained tertiary education. 34.2% of customers are post graduates, 16.6% were from high school, while 1.8% falls in the category others. This tells us the highest attained level of education of the customers. Knowledge on this information can potentially boost overall customer experiences by providing suitable offers catering to these subsets of people.

Box Plot

A box plot could be used to show the distribution of numeric variables such as the credit limit of customers. The code to plot a box plot is:

#Plot Box Plot

boxplot = ecadf['LIMIT']

fig = plt.figure(figsize =(8, 8))

#Creating plot

plt.boxplot(boxplot)

plt.title("Credit Limit of Customers", fontsize=18)

#Show Plot

plt.show()

The output of the box plot is:



Figure : Box Plot of Credit Limit of Customersgood

20M

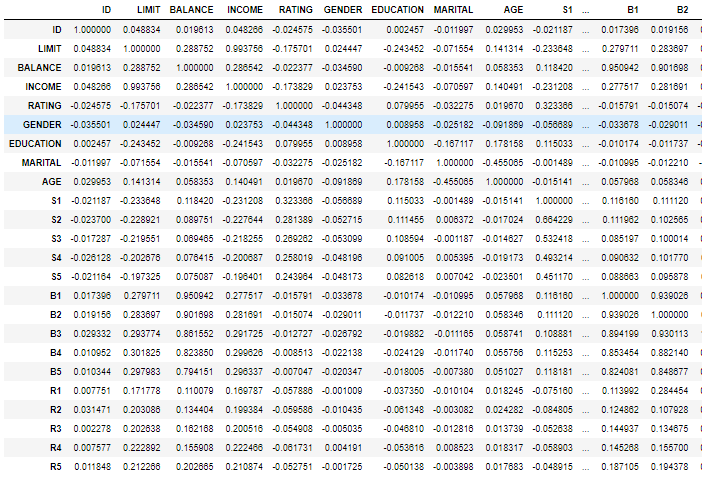
The orange line of the box plot represents the median of the dataset. The box shows the middle 50% of the credit facility dataset, which also meant that it is an indication of the interquartile range. The whiskers represent the top and bottom 25% of the data. This meant that somewhere between 100000 and 200000 is the median credit limit. Most customers would be able to get a credit limit between 50000 to 250000, but some customers would have a credit limit of 500000, and even above. This helps the company determine the limit to the amount of bad debt that the credit facility is being exposed to, in an even that customers default payment.

**Question 4**

Since the goal is to accurately predict variable B1, B1 will then be the dependent variable which is being plotted on y-axis. Firstly, we will look at the correlation between all the datasets using the code:

ecadf.corr()

The output produce is:



We will focus on the B1 column since we are supposed to predict the variable B1. From here, we can identify all the correlation between B1 and all the other variables. It is it evident that BALANCE, B2, B3, B4, and B5 all have a positive correlation with B1. A high linear correlation would increase the accuracy of the prediction model. Hence, we will be using these values as our x values to perform linear regression modelling.

We will continue to do additional data pre-processing for out datasets as it might cause overfitting if we were to feed all our datasets into our linear regression model. Hence it is important that we further split the dataset into training and testing datasets. Training datasets will be used to train our model while testing datasets will be used to test it subsequently. This could be accomplished by using the code train\_test\_split:

#Set Random State and Testing Size

#Training Size will be 70 and Testing Size will be 30

random\_state = 0

test\_size = 0.3

#Splitting of data into training and testing datasets

x\_train, x\_test, y\_train, y\_test = train\_test\_split(

x, y, test\_size = test\_size, random\_state = random\_state

)

This will split our model into 70 and 30 with 70 being the training size and 30 being the testing size. Next, we will train the linear regression model where we fit the line to our training datasets. This could be so by:

#Import Scikit-Learn model type

from sklearn.linear\_model import LinearRegression

regressor = LinearRegression()

#Fit line to x\_train & y\_train dataset

regressor.fit(x\_train, y\_train)

The line that is being formed will be derived by the features and intercept. To identify both values, we could use the code:

#Identify intercept

print(regressor.intercept\_)

#Identify slope

print(regressor.coef\_)

The output for intercept is 199.9 while the outputs for the slope for B2, B3, B4, B5 and BALANCE are 0.389, 0.0517, -0.0245, 0.039, 3.109 respectively.

A function could be created to compute B1:

#Function to calculate b1

def calc(slope, intercept, a):

return slope\*a+intercept

b1 = calc(regressor.coef\_, regressor.intercept\_, 3)

print(b1)

Either that, or we could just use the existing predict function:

b1 = regressor.predict([[0.38944096, 0.05173866, -0.02452624, 0.0391129, 3.1097069]])

Next, to conduct prediction on the testing dataset, the x\_test values will be parsed into the same method as well:

#Conduct prediction on testing dataset

y\_pred = regressor.predict(x\_test)

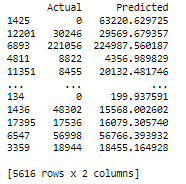
To present both actual and predicted results, a new dataframe could be created to show them side by side:

#Compare results between prediction of testing results

df\_preds = pd.DataFrame({'Actual': y\_test.squeeze(), 'Predicted': y\_pred.squeeze()})

print(df\_preds)

This is the dataframe output:



To see how well the model predicts, metrics such as Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Squared Error (RMSE) could be used. This can be computed by the following code:

#Import MAE & MSE formula

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error

#Calculating metrics

mae = mean\_absolute\_error(y\_test, y\_pred)

mse = mean\_squared\_error(y\_test, y\_pred)

rmse = np.sqrt(mse)

From this, we can compute that the MAE is 6514.72, MSE is 358270957.01, and RMSE is 18928.05.

Ultimately, the code to create the multiple linear regression model is:

#Import Seaborn for Linear Regression Model

import seaborn as sns

variables = ['B2','B3','B4','B5','BALANCE']

#Plot Regression Model

for v in variables:

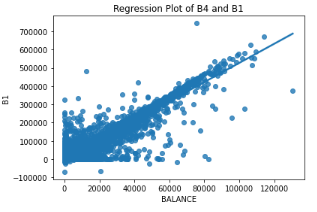
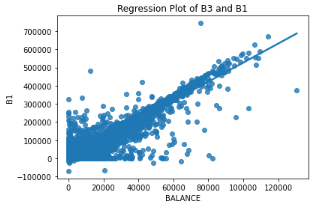
plt.figure() # Creating a rectangle (figure) for each plot

# Regression Plot also by default includes

# best-fitting regression line

# which can be turned off via `fit\_reg=False`

sns.regplot(x=var, y='B1', data=ecadf).set(title=f'Regression Plot of {v} and B1');



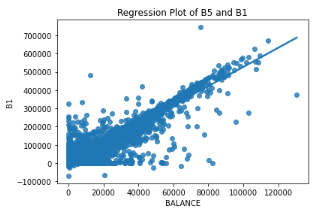




Figure : Multiple Linear Regression Models

The for loop will generate a total of 5 regression plots as there are 5 x variables being input into the loop, with all of them to be plotted with B1.

**Question 5**

Since there is 1 dependent variable, 5 independent variables used in our linear regression equation, the final equation could be expressed as:

Y = A1X1 + A2X2 + A3X3 + A4X4 + A5X5 + C where:

Y = target variable B1

A1X1 = coefficient of independent variable BALANCE

A2X2 = coefficient of independent variable B2

A3X3 = coefficient of independent variable B3

A4X4 = coefficient of independent variable B4

A5X5 = coefficient of independent variable B5

C = interception

The following code could be used to derive the above coefficients and interception values:

#Display Intercept

regressor.intercept\_

#Display Coefficients of the features

regressor.coef\_

Hence, with the respective output, the final linear regression equation is:

Y = 3.109707X1 + 0.389441X2 + 0.051739X3 – 0.024526X4 + 0.039113X5 + 199.937591

The values of the coefficient of the x values is able to determine the magnitude on how much B1 can change just by increasing the value of the independent values. For BALANCE, when it increases by 1, the target B1 will increase by 3.109707. Similarly, for B2, B3, B4, and B5, the magnitude of change affected on the target B1 will be reflected based on the coefficient. It is conclusive that having the highest coefficient, the independent variable BALANCE will affect the target B1 by a large magnitude as compared to the rest. This is followed by B2 with a coefficient of 0.389441. This meant that both of these independent variables have a stronger relationship with B1. For B4, since it has a negative correlation, it indicates that B4 and target B1 has a negative relationship.

Therefore, a key insight would be the fact that any changes in the target B1 would most likely be due to the change in BALANCE and B2, while B3, B4 and B5 are not significantly correlated with B1.

The same metrics could be computed for this case. MSE could be used as the loss function while additionally, R2 can be computed as an additional metric:

#Calculate R2

actual\_minus\_predicted = sum((y\_test - y\_pred)\*\*2)

actual\_minus\_actual\_mean = sum((y\_test - y\_test.mean())\*\*2)

r2 = 1 - actual\_minus\_predicted/actual\_minus\_actual\_mean

print('R²:', r2)

The output of R2 is 0.928. This meant that the linear regression model can explain 92.8% of the variance within the target B1. Therefore, the linear regression model can conduct an accurate prediction of the dependent variable B1.

**References**

Wu, K. Y. (2022). ANL252 Python for data analytics (study guide). Singapore University of Social Sciences.

Kim Eckart. (2021). How long can a person live? The 21st century may see a record-breaker. UW News. Retrieved September 5, 2022, from <https://www.washington.edu/news/2021/07/01/how-long-can-a-person-live-the-21st-century-may-see-a-record-breaker/#:~:text=Among%20their%20findings%3A,years%20old%20(68%25%20probability)%3B>

Saeed.M. (2022). Calculating Pearson Correlation Coefficient in Python with Numpy. StackAbuse. Retrieved from: <https://stackabuse.com/calculating-pearson-correlation-coefficient-in-python-with-numpy/>