

**ANL252**

**Python for Data Analytics**

**End-of-Course Assessment**

**July 2022 Presentation**

**Submitted by:**

|  |  |
| --- | --- |
| **Name** | **PI No.** |
| **Shun Pei Ling Felicia** | **H1882537** |

**Tutorial Group: ­­­­­­­­­­­­ T 09**

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

**Submission Date: 05/09/2022**

**Question 1**

|  |  |
| --- | --- |
| **Variable** | **Type** |
| ID | Numeric |
| LIMIT | Numeric |
| BALANCE | Numeric |
| INCOME | Numeric |
| GENDER | Categorical |
| EDUCATION | Categorical |
| MARITAL | Categorical |
| AGE | Numeric |
| S(n) | Categorical |
| B(n) | Numeric |
| R(n) | Numeric |
| RATING | Categorical |

**Question 2**

|  |  |
| --- | --- |
| In [1] | # import relevant modules with pd as an alias for pandas, plt as an alias for matplotlib.pyplot and np as an alias for numpy  import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  # read csv file "GBA\_Data.csv" into a dataframe name employee\_data  credit\_facility = pd.read\_csv("/Users/feliciashun/Desktop/SUSS/Y5S1/ANL252/ECA\_data.csv")  # capture dataframe  df = pd.DataFrame(credit\_facility)  df |

1. Incorrect data type for variable R3

|  |  |
| --- | --- |
| In [2] | # obtain an overview of DataFrame  df.info() |
| Out [2] |  |
| In [3] | # remove "$" and "," under R3  df["R3"] = df["R3"].str.replace(",", "", regex = True)  df["R3"] = df["R3"].str.replace("$", "", regex = True)  # convert datatype for R3 to integer  df["R3"] = df["R3"].astype(int) |
| In [4] | # check output with df.info()  df.info() |
| Out [4] |  |

An overview of the dataframe using the .info() method has revealed that the variable R3 has been classified wrongly as an ‘object’ under the Dtype column. The .info() method is used to provide a basic overview of the DataFrame. Further analysis of the values under the variable R3 column has detected the presence of 2 symbols, namely, ‘,’ and ‘$’. In this case, the .replace() method is used to remove the symbols by replacing them with no space in between as denoted by “”. In general, it removes the symbols. The datatype for R3 is then converted to an integer using the .astype method. ‘int’ is placed within the round brackets of the .astype() method to change the datatype to integer. By casting the R3 column to the .astype(int) method, Python would transform the type of all its values under the R3 column to integer.

The outcome is then verified using the .info() method to ensure that it has been changed accurately. Based on the output, the Dtype column for variable R3 under data #21 has been modified to ‘int64’. In other words, the datatype for R3 has been changed appropriately to integer. ‘int’ of ‘int64’ represents integer while ‘64’ represents 64 bit, which is the memory assigned to store data, or in other words, the number of digits, in each cell. Since variable R3 pertains to the customer’s repayment amount, it is a numeric variable. It should not be classified as a string as it would be inconsistent with the quantitative nature of the variable. If these data were to be stored as string, it would not be possible to conduct arithmetic operations for statistical analysis or graph plotting where numeric variables are involved.

1. Null values under variables EDUCATION and MARITAL

|  |  |
| --- | --- |
| In [5] | # count the number of null values in each column of DataFrame  df.isnull().sum(axis = 0) |
| Out [5] |  |
| In [6] | # remove rows containing null values under EDUCATION and MARITAL columns using dropna. method  df.dropna(subset = ["EDUCATION", "MARITAL"], inplace = True) |
| In [7] | # verify count of null values in each column of DataFrame  df.isnull().sum(axis = 0) |
| Out [7] |  |

The .isnull().sum() method is used to examine for the presence and count the number of null values for each cell in the DataFrame. In this case, the axis = 0 is added to obtain the sum of the number of missing values under the variable columns. Based on the output, it is found that there are empty fields under the EDUCATION and MARITAL columns. It states that the EDUCATION and MARITAL variable columns contain 13 and 38 null values respectively. In this case, the .dropna() method is applied on both of these columns to eliminate records containing empty fields.

The output is then verified using the .isnull().sum() method. Since records with empty fields have been dropped, the output has returned a ‘0’ for all variable columns to indicate the absence of null values for all variable columns. Dropping these null values would be a feasible option. Null values under these columns could imply a system error or the customers’ intention to not disclose this information. However, an in-depth analysis on the data have prompted that it may have occurred at random and carried no trend as to the reason why these values are missing. Since the dataset is huge and where the null values only accounts for a small proportion of only 51 records, omitting these records would not result in a significant information loss or biasness. Instead, it would help minimize the distortion of the validity of the results and conclusions.

1. Duplicated rows

|  |  |
| --- | --- |
| In [8] | # check for any duplications in data  df[df.duplicated()] |
| Out [8] |  |
| In [9] | # drop duplicated rows from DataFrame  df = df.drop\_duplicates() |
| In [10] | # verify for any duplications in data  df[df.duplicated()] |
| Out [10] |  |
| In [11] | # verify output  df.info() |
| Out [11] |  |

The .duplicated() method is used to identify for any repeated records of the exact values. Based on the output, Python has identified 3 duplicated records. No record should be identical to one another as each customer record is unique in nature assigned with its own unique identifier. As such, an approach to this would be to remove these duplicated data since a copy of each of these records have already been stored and can be found in the dataset. This is achieved by using the .drop\_duplicates() method. In this method, Python would remove duplicate rows based on all columns.

The result is then verified using the .duplicated() method. The output is presented in a table format with the respective variable columns and a statement, ‘0 rows x 24 columns’, below the table. This means that there are no duplicated rows based on all the 24 variable columns. In addition, using the .info() method has revealed that there are 18,717 entries. This suggests that there has been a reduction by 3 entries from 18,720 to 18,717. With that, we are certain that the duplicated rows have been removed appropriately.

1. Unusual values for AGE

|  |  |
| --- | --- |
| In [12] | # obtain summary of statistics for DataFrame using .describe()  df.describe() |
| Out [12] | \*image is cropped to focus on subject matter |
| In [13] | # drop rows containing for AGE < 0 (i.e. signify negative values for age)  df = df.drop(df.index[df["AGE"] < 0])  # drop rows containing for AGE equal to 199  df = df.drop(df.index[df["AGE"] == 199]) |
| In [14] | # verify summary of statistics for DataFrame using .describe()  df.describe() |
| Out [14] | \*image is cropped to focus on subject matter |

The .describe() function generates a summary statistic of the DataFrame columns. Based on the output, we discovered that AGE contain minimum and maximum values of an unusual nature. The minimum value under the AGE variable column is -1. This is inconsistent with the nature of age as age does not run backwards. Considering that the dataset is in the context of a credit facility, it is unusual for a person who has not born yet to be legally and physically capable of signing for a credit facility. On the other hand, maximum value under AGE is 199. It is highly impossible for a person to live with such a long life expectancy of 199 years old since the world’s oldest person to have ever live died at the age of 122 years old. In this case, the .drop() method is applied to remove records with age less than 0 as well as those containing the value of 199. In order to identify these records containing these elements, the .index() method is used to search for the row indexes containing values of less than 0, as well as values equal to 199 under the AGE column.

The outcome is then verified using the .describe() function. Based on the output, the “min” and “max” under the AGE column shows a value of 21 and 80 respectively. This means that the AGE column currently has a minimum and maximum values of 21 and 80 years old respectively. With that, we can conclude that the unusual values of age have been dropped appropriately and the current age range is consistent with its nature.

**Question 3**

Histogram

|  |  |
| --- | --- |
| In [15] | # capture INCOME column of the DataFrame  df\_income = pd.DataFrame(df, columns = ["INCOME"])  df\_income |
| In [16] | # assigning income values to income group by creating a list of conditions  conditions = [  (df\_income["INCOME"] >= 10000) & (df\_income["INCOME"] < 60000), (df\_income["INCOME"] >= 60000) & (df\_income["INCOME"] < 110000),  (df\_income["INCOME"] >= 110000) & (df\_income["INCOME"] < 160000), (df\_income["INCOME"] >= 160000) & (df\_income["INCOME"] < 210000),  (df\_income["INCOME"] >= 210000) & (df\_income["INCOME"] < 260000), (df\_income["INCOME"] >= 260000) & (df\_income["INCOME"] < 310000),  (df\_income["INCOME"] >= 310000) & (df\_income["INCOME"] < 360000), (df\_income["INCOME"] >= 360000) & (df\_income["INCOME"] < 410000),  (df\_income["INCOME"] >= 410000) & (df\_income["INCOME"] < 460000), (df\_income["INCOME"] >= 460000) & (df\_income["INCOME"] < 510000),  (df\_income["INCOME"] >= 510000) & (df\_income["INCOME"] < 560000), (df\_income["INCOME"] >= 560000) & (df\_income["INCOME"] < 610000),  (df\_income["INCOME"] >= 610000) & (df\_income["INCOME"] < 660000), (df\_income["INCOME"] >= 660000) & (df\_income["INCOME"] < 710000),  (df\_income["INCOME"] >= 710000) & (df\_income["INCOME"] < 760000), (df\_income["INCOME"] >= 760000) & (df\_income["INCOME"] < 810000),  (df\_income["INCOME"] >= 810000) & (df\_income["INCOME"] < 860000), (df\_income["INCOME"] >= 860000) & (df\_income["INCOME"] < 910000),  (df\_income["INCOME"] >= 910000) & (df\_income["INCOME"] <= 960000)  ]  # creating corresponding values to fill  income\_bins = ["10000-59999", "60000-109999", "110000-159999",  "159999-209999", "210000-259999", "260000-309999",  "310000-359999", "360000-409999", "410000-459999",  "460000-509999", "510000-559999", "560000-609999",  "610000-659999", "660000-709999", "710000-759999",  "760000-809999", "810000-859999", "860000-909999",  "910000-960000",  ]    # using np.select() to set values using multiple conditions  df\_income["INCOME RANGE"] = np.select(conditions, income\_bins)  # create pivot table for income  income\_pivot = pd.pivot\_table(df\_income, index = "INCOME RANGE", values = "INCOME", aggfunc = "count")  income\_pivot |
| In [17] | # cast "INCOME" column of the DataFrame into a list  income\_list = list(df\_income["INCOME"])  # income\_min and income\_max are the minimum and maximum value respectively of the income list  income\_min = min(income\_list)  income\_max = max(income\_list) |
| In [18] | # figure assigns an area for the chart. Figsize represents the figure size in inches.  plt.figure(figsize = (20,10))  # histogram with range from 0 to 960000, bins are centered between the bin edges, vertical orientation  plt.hist(income\_list, bins = range(income\_min, income\_max, 50000), color = "orange", align = "mid", orientation = "vertical", edgecolor = "black")  plt.title("Distribution of income", fontsize = 30, weight = "bold")  plt.xticks(ticks = range(10000, 960000, 50000), labels = range(10000, 960000, 50000), fontsize = 15, rotation = 45)  plt.yticks(ticks = range(0, 6000, 500), labels = range(0, 6000, 500), fontsize = 15)  plt.xlabel("Income ($)", fontsize = 20)  plt.ylabel("Number of customers", fontsize = 20)  plt.show() |

Table

Description automatically generated

*Figure 1*

Chart, histogram

Description automatically generated

*Figure 2: Count of customers by income range*

The histogram in Figure 1 illustrates the frequency distribution of the income of the customers of a credit facility. The values for income in the histogram are categorized to decrease the number of bins and the height of each bin represents the number of customers that belong in each bin.

Based on the above histogram, we can infer that the distribution of the income is not symmetrical. It is skewed to the left and is hence said to be positively skewed. With that, the mean would be larger than the mode of medium. From the above graph, it can be seen that majority of customers earn an income between $10,000 and $59,999. They form the largest group of 5077 customers as observed from the pivot table in Figure 1, accounting for the highest proportion of 27% of the total customers.

On the other hand, we have noticed that there are outliers and extreme values. According to Figure 2, there are 23 staffs earning an income falling within the range of $610,000 and $659,000, as well as 20 staffs earning an income range between $660,000 and $709,000. They form the outliers. There are 11, 10, 7 and 3 high earners earning an income ranging from $710,000 to $759,000, $760,000 to $809,000, $810,000 to $859,999 and $860,000 to $909,000 respectively. These high earners are the source of the positively skewed distribution as the extreme income value that are much greater than the mean outweighs the fewer number of income value which are much lower than the mean.

Bar Chart

|  |  |
| --- | --- |
| In [19] | # capture GENDER and EDUCATION column of DataFrame  df\_edu = pd.DataFrame(df, columns = ["GENDER", "EDUCATION"])  df\_edu |
| In [20] | # create frequency columns based on gender  df\_edu["FREQUENCY"] = df.groupby(["GENDER"])["GENDER"].transform("count")  # create pivot table for education levels  edu\_pivot = df\_edu.pivot\_table(index = "EDUCATION", values = "FREQUENCY", columns = "GENDER", aggfunc = "count")  # rename index and column values for EDUCATION  edu\_pivot.rename(index = {0.0:"Others", 1.0:"Postgraduate", 2.0 :"Tertiary", 3.0:"High School"}, inplace = True)  edu\_pivot.rename(columns = {0:"Male", 1:"Female"}, inplace = True)  edu\_pivot |
| In [21] | # plot bar chart using pivot table  bar\_chart = edu\_pivot.plot.bar(stacked = True, color = ["blue", "red"], figsize = (10,10), rot = 1)  plt.title("Gender proportion of customers by education", fontsize = 20, weight = "bold")  plt.xticks(fontsize = 15)  plt.yticks(ticks = range(0, 10000, 1000), labels = range(0, 10000, 1000), fontsize = 15)  plt.xlabel("Education", fontsize = 18)  plt.ylabel("Number of customers", fontsize = 18) |

Table

Description automatically generated

*Figure 3*

*Chart, bar chart

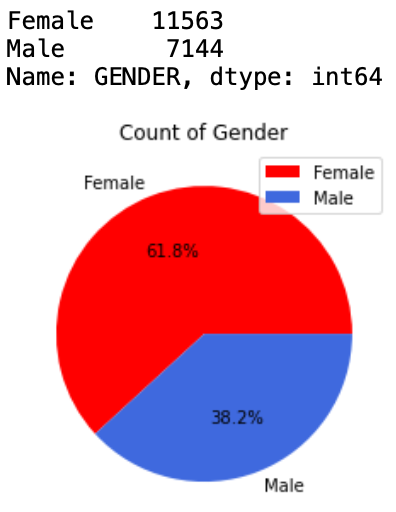
Description automatically generated*

*Figure 4: Count of customers by education*

The bar chart illustrates the demographics of the customers by their education qualification. With reference to the bar chart, we noticed that the largest group belongs to customers who have completed their tertiary education. It comprises of 8863 customers, accounting for 47% of the total population. This data is consistent with the results observed in Figure 2. In conjunction with the data of customers who completed their high school education, these two groups may be the source of the positively skewed distribution of income observed in Figure 2, earning the lowest income range of $10,000 and $59,999. This is attributed to the fact that high school leavers often earn the lowest income due to their lack of expertise and poor career prospects, with tertiary school leavers earning the next lowest since it forms the next level of education qualification. With a proportion of 47% of the population, it might have suggested the largest group of earners in the lowest income range.

Pie Chart

|  |  |
| --- | --- |
| In [22] | # capture GENDER column of DataFrame  df\_gender = pd.DataFrame(df, columns = ["GENDER"])  df\_gender |
| In [23] | # rename 0 and 1 to "Male" and "Female" respectively  df\_gender.replace(1, "Female", inplace = True)  df\_gender.replace(0, "Male", inplace = True) |
| In [24] | # set colours to the pie chart  colours = ["red", "royalblue"]  # storing the gender types in x  x = list(df\_gender["GENDER"].value\_counts().keys())  # storing the number of customers in y  y = list(df\_gender["GENDER"].value\_counts())  # plotting of pie chart with respect to the assigned colours, add in percentage  plt.pie(y, labels = x, autopct = "%.1f%%", colors = colours)  # set the pie chart's title and legend  plt.title("Count of Gender")  plt.legend()  # count of gender and displaying its value  values = df\_gender["GENDER"].value\_counts()  print(values) |



*Figure 5: Gender proportion*

Based on Figure 4, we noticed that there are significant differences in gender. In this case, an easily-interpretable form such as a pie chart would be used to better illustrate the segment sizes of the respective gender.

With reference to the pie chart above, it is noted that the customers of the credit facility are predominantly female. Out of the 18,707 customers, 11,563 or 61.8% of them are female. In addition, it is interesting to note that majority of these female customers form part of largest group of customers who have completed their tertiary education as shown in Figure 4. Out of the 11,563 female customers, 5589 or 48% of them are females as seen in Figure 3. The second education level for which there is a higher female proportion belongs to the postgraduate level. Out of the 11,563 female customers, 3871 or 33% of them are females. The fact that there is a higher proportion of female customers who have pursued these two higher education levels may have suggested the overall higher proportion of female customers that have taken loans from the credit facility, possibly due to their finance needs to fund their tertiary or postgraduate education.

Bar Chart

|  |  |
| --- | --- |
| In [25] | # capture AGE of DataFrame  df\_age = pd.DataFrame(df, columns = ["AGE"])  df\_age |
| In [26] | # assigning ages to age group by creating a list of conditions  conditions = [  (df\_age["AGE"] >= 20) & (df\_age["AGE"] < 30), (df\_age["AGE"] >= 30) & (df\_age["AGE"] < 40),  (df\_age["AGE"] >= 40) & (df\_age["AGE"] < 50), (df\_age["AGE"] >= 50) & (df\_age["AGE"] < 60),  (df\_age["AGE"] >= 60) & (df\_age["AGE"] < 70), (df\_age["AGE"] >= 70) & (df\_age["AGE"] <= 80)  ]  # creating corresponding range to fill  age\_group = ["20-29",  "30-39",  "40-49",  "50-59",  "60-69",  "70-80",  ]    # using np.select() to set values using multiple conditions  df\_age["AGE GROUP"] = np.select(conditions, age\_group)  # create pivot table for age  age\_pivot = df\_age.pivot\_table(index = "AGE GROUP", values = "AGE", aggfunc = "count")  age\_pivot |
| In [27] | # plot bar chart based on pivot table  bar\_chart = age\_pivot.plot.bar(color = "purple", figsize = (10,10), rot = 1)  plt.title("Count of customers by age group", fontsize = 20, weight = "bold")  plt.xticks(fontsize = 15)  plt.yticks(ticks = range(0, 8000, 1000), labels = range(0, 8000, 1000), fontsize = 15)  plt.xlabel("Age group", fontsize = 18)  plt.ylabel("Number of customers", fontsize = 18) |

Table

Description automatically generated

*Figure 6*

Chart, bar chart

Description automatically generated

*Figure 7: Count of customers by age group*

The bar chart in Figure 7 illustrates the demographics of the customers by age group. Based on the bar chart, we can see that majority of the customers are between the ages of 30 and 39 years old. This largest age group amount to 7042 customers, accounting for 38% of the total customers. The second largest age group belongs to the age group of 20 to 29 years old, comprising of 5923 customers or 32% of the total customers. This data is consistent with our analysis of the education level of customers in Figure 4 above where majority of the customers have attained tertiary education. Since it forms the next level of education path one often takes after high school, their age would follow accordingly and falls in the young adult age range. It further supports the results observed in Figure 4 as customers in this age group are often associated with being fresh school leavers, where they are in the phase of achieving financial stability while paying off their student loans.

Scatter Plot

|  |  |
| --- | --- |
| In [28] | # plot scatter plot  plt.figure(figsize = (14,8))  plt.scatter(df["INCOME"], df["LIMIT"], c = "green", s = 40)  plt.xticks(fontsize = 15)  plt.yticks(fontsize = 15)  plt.xlabel("Income ($)".title(), fontsize = 20)  plt.ylabel("Credit limit ($)".title(), fontsize = 20)  plt.title("Relationship between income and credit limit".title(), fontsize = 20, weight = "bold") |

Chart

Description automatically generated

*Figure 8: Relationship between Income and Credit Limit*

A scatter plot displays the values of one numeric variable plotted against the values of another numeric variable. In this case, the scatter plot in Figure 8 is used to determine whether there is any relationship or trend between income and credit limit. From the graph, we can infer that there is a positive correlation among the data. The credit limit increases with an increase in income. The amount of money the customers make impacts the amount they can afford to pay. By granting customers earning a high income with a high credit limit, the credit facility lending the money is able to feel assured and confident that the customers have the ability to repay their debt obligations. On the other hand, by granting customers with low income with a low credit limit, it helps prevent customers from defaulting on their obligations and thereby, increasing the company’s losses.

**Question 4**

1. Import relevant modules

Modules that are relevant for data preprocessing and the creation of the linear regression model are first imported as shown above.

|  |  |
| --- | --- |
| In [29] | # import relevant modules  from sklearn.linear\_model import LinearRegression  from sklearn.model\_selection import train\_test\_split  from sklearn import preprocessing  import statsmodels.api as sm  from sklearn.metrics import mean\_absolute\_error |

1. Data pre-processing

Data pre-processing are sorted into the various categories namely, data cleaning, data transformation and data reduction.

Data reduction

As part of the data reduction process, irrelevant variables are identified using the .corr() function. In this case, the .corr() function searches for any correlations between each of the variable column with our dependent variable, B1. A positive correlation value indicates a positive relationship while a negative correlation value indicates a negative relationship. A value close to 1 indicates a strong correlation while a value close to 0 indicates a weak correlation.

Based on the output, it has revealed that the numerical variables such as BALANCE and B2 are positively associated with the dependent variable, B1. In other words, as the respective values of BALANCE or B2 increases, the value of B1 increases as well. This is consistent with its nature, considering that when a customer’s credit balance increases, the customer owes a higher amount and would therefore correspondingly be billed at a higher amount.

Although variables such as B3, B4 and B5 show a strong association, they are not included in the predictive analysis, especially since their values lead up to the next, with B2 containing the latest updated amount needed to predict B1. In this case, it would only be sensible to drop them. On the other, variables such as AGE, ID, LIMIT, INCOME, RATING, EDUCATION, MARITAL, S1, S2, S3, S4 AND S5 should be dropped. The output has shown extremely small correlation values for these variables, which may suggest its poor association with B1. This is consistent with its nature considering that these categorical features do not affect or determine how much of an amount is billed to the customer.

|  |  |
| --- | --- |
| In [30] | # identify correlations among the individual columns with B1  df.corr()["B1"] |
| Out [30] |  |

The irrelevant variable columns are dropped using the .drop() method. By stating axis = 1, Python drops the specified columns. The output is then verified typing “df”, to display the rows and columns of the DataFrame.

|  |  |
| --- | --- |
| In [31] | # drop irrelevant variable columns  df.drop(["ID", "RATING", "GENDER", "LIMIT", "MARITAL", "INCOME", "EDUCATION", "AGE", "S1", "S2", "S3", "S4", "S5", "B3", "B4", "B5", "R1", "R2", "R3", "R4", "R5"], axis = 1, inplace = True)  # verify output  df |
| Out [31] |  |

Data cleaning

Next, data cleaning is performed to remove records containing 0 values under the variable columns, BALANCE and B1 and B2.

The .describe() method is used to obtain a summary statistics of the DataFrame. After that, an analysis of B1 and B2 columns is conducted by using the .loc() method to access the rows containing 0 values under these specified columns. Based on its output, it can be seen that there are a number of rows containing 0 values under those columns. An approach to this would be to eliminate these records. Considering of the fact that there are no amounts billed in these 2 months and where there is no balance due, these data do not carry any meaning and may not provide meaningful results in the later part of the prediction analysis. This is especially so, since our objective is to predict B1, the billable amount in the most recent month. The inclusion of these data will only distort the validity of the results. Since the original dataset is huge and where there are only a small sample of 717 records containing these 0 values, eliminating these rows would be a feasible option since it would not lead to a significant information loss or biasness.

|  |  |
| --- | --- |
| In [32] | # obtain summary statistics of DataFrame  df.describe() |
| Out [32] |  |
| In [33] | # analysis of individual columns  df.loc[(df['B1'] == 0) & (df['B2'] == 0) & (df['BALANCE'] == 0)] |
| Out [33] |  |

By using the .index() method, Python is able to identify the row indexes for which it contains 0 values under the specified columns. The .drop() method is used within the same line of code to eliminate these rows identified. The result is then verified using the .loc() method. Based on its output, the intended rows have been removed appropriately as there are no rows displayed under the variable columns.

|  |  |
| --- | --- |
| In [34] | # drop rows with 0 values under B1 and B2 columns  df.drop(df[(df['B1'] == 0) & (df['B2'] == 0) & (df['BALANCE'] == 0)].index, inplace = True)  # check output  df.loc[(df['B1'] == 0) & (df['B2'] == 0) & (df['BALANCE'] == 0)] |
| Out [34] |  |

Data transformation

Before transforming the data, the dependent variable, denoted as y, and independent variables, denoted as X, are extracted from the dataset. Since our objective is to predict variable, B1, the extraction can be simply performed by selecting the “B1” column, which represents our dependent variable, and save it as Y. On the other hand, since the dataset only comprises of dependent and independent variables, the .drop() method is used to drop the dependent variable, B1, to obtain the remaining independent variables. This data is stored in X. This is illustrated in Cell 35 below.

Based on the summary statistics generated from the .describe() method in Cell 32 above, it is observed that there are significant differences in the range of values between the minimum and maximum present in the numerical variables. Data transformation is necessary to prevent the larger variables from outweighing variables with smaller ranges. In this case, we perform data transformation by standardization. The StandardScaler from the preprocessing module shall be initiated to transform X as shown below. This feature basically rescales and centers the values around its mean with a standard deviation unit.

|  |  |
| --- | --- |
| In [35] | # slicing dataset into dependent variable and independent variables  y = df["B1"]  X = df.drop(columns = "B1", axis = 1) |
| In [36] | # initiate estimator and transform data  scaler = preprocessing.StandardScaler()  standardized\_data = scaler.fit\_transform(X)  standardized\_data |
| Out [36] |  |

Next, each of the y and X dataset are randomly partitioned into two subsets namely, the training set and the testing set using the train\_test\_split method. A train\_test\_split is needed to prevent overfitting. Overfitting occurs when the model is too specialized to a particular dataset such that it is unable to fit well with unseen data. In linear regression analysis, the performance of a model is dependent on its prediction accuracy on the unseen data. However, in most cases, such data are not easily found. As such, a train\_test\_split is used, where the model is first constructed based on the training dataset. However, it does not tell us on how well the model is generalized to the unseen data. In this case, the testing set is applied to measure the model’s performance on its unseen data.

We apply the golden rule of thumb to split the dataset into an 80:20 ratio. This is achieved by specifying the test size of 0.2. The training and testing dataset for the independent variable are displayed as shown in the output of Cell 38 below. Based on the output, we can see that 14,392 rows, or 80% of the dataset, have been assigned to the training set while the remaining 20%, or 3598 rows have been assigned to the testing set.

|  |  |
| --- | --- |
| In [37] | # split DataFrame into training and testing dataset  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.2, random\_state = 0) |
| In [38] | # find shape of training and testing datasets  print(y\_train.shape, y\_test.shape, X\_train.shape, X\_test.shape) |
| Out [38] |  |

1. Train model

To perform linear regression modelling, a linear regression model named model is created by initiating LinearRegression(). After that, the model is trained by fitting the training sets namely, X\_train and y\_train, into the model.

|  |  |
| --- | --- |
| In [39] | # train model using training sets  model = LinearRegression()  model.fit(X\_train, y\_train) |

This model is then now ready to be used for prediction on the unseen testing data. In this case, the .predict() function, on the basis of the trained model, can be applied on the unseen testing data, X\_test, to predict the target B1 values, denoted as y\_predict.

A comparison table is presented in the output of Cell 40 for reference to compare the predicted values of the testing set with its actual values.

|  |  |
| --- | --- |
| In [40] | # make predictions using the testing set  y\_pred = model.predict(X\_test)  # comparison of actual and predicted values  pred\_y\_df = pd.DataFrame({'Actual value':y\_test, 'Predicted value':y\_pred, 'Difference': y\_test-y\_pred})  pred\_y\_df |
| Out [40] |  |

**Question 5**

By implementing the .OLS() method, or Ordinary Least Squares method, of the statsmodel module, a summary of the regression statistics is generated as seen in the output of Cell 41. The regression statistics provide an overview of the relevant coefficient values needed to form our linear regression equation. On the other hand, in order to retrieve our intercept value, the .intercept\_ method is applied as shown in Cell 42.

With that, our linear regression equation is as follows:

B1 = 469.0394 + 3.1187(BALANCE) + 0.4419(B2)

|  |  |
| --- | --- |
| In [41] | # obtain summary of regression statistics  results = sm.OLS(y\_train, X\_train).fit()  print(results.summary()) |
| Out [41] |  |
| In [42] | # obtain intercept value  print(model.intercept\_) |
| Out [42] |  |

Based on the results obtained in Question 4, we can see that there are differences between the predicted and actual values. An in-depth analysis of the data may provide an insight that non-qualitative factors may play a part in these differences such as the period in which the credit facility bills their customers. Some customers have balance due but were not billed in the most recent month and these may have resulted in the difference in values.

In terms of quantitative analysis, the coefficient values do provide an insight in terms of the strength of the relationship between the independent and the dependant variables. A positive coefficient value indicates a positive correlation while a negative coefficient value indicates a negative correlation. A positive coefficient can be interpreted as when these independent variable increases, the dependent variable increases as well. The higher the positive or negative value, the stronger the correlation.

With reference to the linear regression equation, it is noted that both the independent variables, BALANCE and B2, have a positive coefficient value of 3.1187 and 0.4419 respectively. However, BALANCE has a stronger positive correlation with its target variable than B2 as it has a higher positive value of 3.1187 as compared to 0.4419. In other words, for every additional unit increase in BALANCE, the estimated mean B1, or billable amount in the recent month, increase by $312, while holding the other independent variable constant. On the other hand, for every additional unit increase in B2, or the billable amount in the previous month, the estimated mean B1, or billable amount in the recent month, will increase by $44, while holding the other independent variable constant. With that, we can infer that the BALANCE variable has a greater influence on the target variable B1 as compared to B2.