**Introduction**

The following project most likely is referring to a credit card system, where what is borrowed in the month will be repaid in the subsequent month. If no one defaulted then it would be normal that B(n)

**Question 1**

The following code was used to be able to identify data types for each variable (column) in the dataset of customers:

ECA\_data.info()

The results show that there are 3 float (numeric), 20 integer types (numeric) and 1 object (string) variables. In total 23 numeric and 1 categorical. Highlighted in yellow is the summary values, highlighted in green is the numeric variables, and finally highlighted in red is the categorical variables.

Below is a table detailing the variable (column) name and its respective data type (categorical, numeric):

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 18769 entries, 0 to 18768

Data columns (total 24 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 ID 18769 non-null int64 🡪 numeric

1 LIMIT 18769 non-null int64 🡪 numeric

2 BALANCE 18769 non-null float64🡪 numeric

3 INCOME 18769 non-null int64 🡪 numeric

4 RATING 18769 non-null int64 🡪 numeric

5 GENDER 18769 non-null int64 🡪 numeric

6 EDUCATION 18756 non-null float64🡪 numeric

7 MARITAL 18731 non-null float64🡪 numeric

8 AGE 18769 non-null int64 🡪 numeric

9 S1 18769 non-null int64 🡪 numeric

10 S2 18769 non-null int64 🡪 numeric

11 S3 18769 non-null int64 🡪 numeric

12 S4 18769 non-null int64 🡪 numeric

13 S5 18769 non-null int64 🡪 numeric

14 B1 18769 non-null int64 🡪 numeric

15 B2 18769 non-null int64 🡪 numeric

16 B3 18769 non-null int64 🡪 numeric

17 B4 18769 non-null int64 🡪 numeric

18 B5 18769 non-null int64 🡪 numeric

19 R1 18769 non-null int64 🡪 numeric

20 R2 18769 non-null int64 🡪 numeric

21 R3 18769 non-null object 🡪 categorical

22 R4 18769 non-null int64 🡪 numeric

23 R5 18769 non-null int64 🡪 numeric

dtypes: float64(3), int64(20), object(1)

memory usage: 3.4+ MB

**Question 2**

**Rationale for sequence of steps**

The following steps for data preparation are executed in the following sequence. This sequence should be adhered to as it prioritizes data replacement before data removal. This is a logical sequence as it optimizes data retention. The steps that come first replace erroneous cells or observations and give them a fair “chance” to be deemed useful relative to the other observations. This way the later steps where data is dropped is done in as objective a way possible by carefully considering each record on a level playing field.

**Data Issue 1**

In order to do regression analysis, one needs all the data types to be numeric. In this case we can observe that R3 is listed as categorical in question 1. This could be due to the fact that the column is not homogenous in its data type. There may be a string variable or special character in that column that is causing what should be an integer variable to be an object instead.

After further exploration of the data file we find that there are some values with the ‘$’ and ‘,’ characters in the cell. These need to be removed in order for the data in R3 to be fully processed as an integer type.

**Data Preparation 1**

Using the code below we are able to replace the ‘$’ and ‘,’ signs with an empty space allowing for the data to be fully numeric and therefore be read as an integer type.

ECA\_data['R3'] = ECA\_data['R3'].str.replace(',','', regex=True)

ECA\_data['R3'] = ECA\_data['R3'].str.replace('$','', regex=True)

ECA\_data['R3'] = ECA\_data['R3'].astype(int)

The success of our codes ability to replace these special characters and thus convert the variable to an integer type can be assessed using the following code:

ECA\_data.info()

The following shows the output reflecting the successful transition to integer (highlighted in green)

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 18769 entries, 0 to 18768

Data columns (total 24 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 ID 18769 non-null int64

1 LIMIT 18769 non-null int64

2 BALANCE 18769 non-null float64

3 INCOME 18769 non-null int64

4 RATING 18769 non-null int64

5 GENDER 18769 non-null int64

6 EDUCATION 18756 non-null float64

7 MARITAL 18731 non-null float64

8 AGE 18769 non-null int64

9 S1 18769 non-null int64

10 S2 18769 non-null int64

11 S3 18769 non-null int64

12 S4 18769 non-null int64

13 S5 18769 non-null int64

14 B1 18769 non-null int64

15 B2 18769 non-null int64

16 B3 18769 non-null int64

17 B4 18769 non-null int64

18 B5 18769 non-null int64

19 R1 18769 non-null int64

20 R2 18769 non-null int64

21 R3 18769 non-null int32

22 R4 18769 non-null int64

23 R5 18769 non-null int64

dtypes: float64(3), int32(1), int64(20)

memory usage: 3.4 MB

Note that the difference between int32 and int64 is just the length of the numbers allowed in that particular cell of the observation.

**Data Issue 2**

It is important to check for missing or blank data in any dataset before coding. Blanks or missing data are incomplete records with missing observations in any of its respective cells. These are understood as NaN values in Jupyter Notebook. Generally, the import process can define various types of characters synonymous with missing or blank data such as ‘-‘ or ‘?’ to name a few. They can be defined in the import process such that Jupyter Notebook recognises it as a NaN value.

NaN values can be treated in multiple ways however, it is only as a last resort that one should drop the record altogether. A better alternative especially for ordinal and ratio type numeric data would be to replace the blank with the mean value of the variable. On the other hand, for categorical variables that are nominal in nature one could use the mode instead. This will prevent data loss especially if the proportion of missing data is large relative to the total population of observations.

**Data Preparation 2**

The following code allows one to identify which variables contain missing values and the frequency of the missing values within that variable.

ECA\_data.isnull().sum()

The output is as follows:

ID 0

LIMIT 0

BALANCE 0

INCOME 0

RATING 0

GENDER 0

EDUCATION 13

MARITAL 38

AGE 0

S1 0

S2 0

S3 0

S4 0

S5 0

B1 0

B2 0

B3 0

B4 0

B5 0

R1 0

R2 0

R3 0

R4 0

R5 0

dtype: int64

Highlighted in red are the variables containing NaN (null) values. Here Education and Marital have 13 and 38 missing values respectively. (Highlighted in red) Because Education and Marital are nominal categorical data we should replace these values with the mode of those variables. While it might skew the data the effect is preferable to losing these records which are otherwise properly filled.

The following is the code needed to replace the blanks with the mode of the respective variables:

ECA\_data['EDUCATION'] = ECA\_data['EDUCATION'].fillna(ECA\_data['EDUCATION'].mode()[0])

ECA\_data['MARITAL'] = ECA\_data['MARITAL'].fillna(ECA\_data['MARITAL'].mode()[0])

The assessment of the above is done with the following:

ECA\_data.isnull().sum()

The output is:

ID 0

LIMIT 0

BALANCE 0

INCOME 0

RATING 0

GENDER 0

EDUCATION 0

MARITAL 0

AGE 0

S1 0

S2 0

S3 0

S4 0

S5 0

B1 0

B2 0

B3 0

B4 0

B5 0

R1 0

R2 0

R3 0

R4 0

R5 0

dtype: int64

Notice how EDUCATION and MARITAL columns now have 0 (highlighted in green) as an attributed value. Relative to before we can thus conclude that the blanks have been successfully replaced with the modes of these columns thereby preventing data loss.

**Data Issue 3**

After replacing and filling up the data set as far as possible there is a chance that there are more duplicates in addition to existing duplicate observations. Duplicates are repeats of observations from the same individual in the context of this question. The result is an increase in the bias based on that individual’s profile. Insights generated would thus be too influenced and therefore defined by these repeated observations; reducing the quality of the results of the regression model. It is therefore important to minimise such bias by removing these duplicates. In this case the data loss is desired to distil the data to have as many unique observations as possible such that bias in the results are minimised.

**Data Preparation 3**

To check for duplicate values the following is used:

ECA\_data.duplicated().sum()

By running this code, we get the output value of 3. From this we know that there are 3 duplicated observations(rows) to be removed.

To remove the rows the following is used:

ECA\_data = ECA\_data.drop\_duplicates(keep='first')

The line keep = ‘first’ will prevent the originals from being removed with the duplicates. To assess if all duplicated records have been removed we can run the following once more:

ECA\_data.duplicated().sum()

In doing so we get the values of 0 which thus means the all observations are now unique to each other.

**Data Issue 4**

**Data Preparation 4**

**Question 3**

Before visualisations a number of additional columns were created with the string versions of many of the variables such that during visualisation the viewer has the ease of understanding labelling of the values. (ie. Customer repayment reflected status in nth month. (-1; Prompt payment, 0: Minimum sum payment, Grouping [1,2,3,4,5,6,7,8,9 ect.]:Delayed payment)

Moreover, the age column was binned so as to be able to create aggregations within the visualisations to view some of the trends with more clarity.

The order of the visualisations is also important as it follows the depth of investigation into the credit card company’s customer activities in terms of demographics to payment punctuality respectively.

**Visualisation 1**

**Analysis 1**

To view the general demographics of the customers of the bank. One can see the break down by age group (these were binned in the visualisation preparation stage) and respective gender within that age group. This is the most general chart that gives a sense of what customer segment the bank tends to attract.

The clustered bar chart shows that the bank is generally more popular with the younger generation of individuals aged in their 20s and 30s. Moreover, its clients tend to be women more than men.

**Visualisation 2**

**Analysis 2**

To view the distribution of income across the population dataset. This is useful as it gives a general idea of how wealthy the bank’s current customers tend to be. We will assume that the INCOME variable refers to the annual income of the customer given the scale of the salaries being measured.

As a result of the visual one can see that most of the population have relatively low income. In fact, given that 50,000 annual income is a recognised benchmark that demarcates poverty, we can see that most of the bank’s customers earn very low income.

As calculated, for greater precision, we can see that this constitutes 4039 individuals or 22% of the population of customers.

In addition, the median is useful in showing how much half of the population tends to earn. In this case the 50th percentile of INCOME is $148210.50.

**Visualisation 3**

**Analysis 3**

This is a cumulative histogram of the BALANCE of the customers of this bank. What is of interest to the bank would be 2 things.

Firstly, it shows the general strength of the customer’s balance values. Generally, we would want this visual to have a stronger diagonal slope but in this case we can see that the slope is gentle for the most part with only a sharp increase towards the end. This tells us that the customer base have generally poor bank balances with small sums.

Secondly, by mousing over, and viewing the corresponding table to the visual, the bank can track the number of accounts with a BALANCE of 0. This is important to a bank as these are potentially dormant accounts and, therefore, would need further investigation to determine if they should be removed.

Lastly the median allows us to see that the general amount of BALANCE in an account of a customer trends around $3959.55. This figure is expected given the number of low incomes customers but concerning for a bank as it indicates poorly with regards to its capital and lending power.

**Visualisation 4**

**Analysis 4**

A pie chart (2A) was created to view the status distribution of S1\_str (or the most recent month of customer debt repayment status). We assume that S1\_str is a cumulative (or carried over) status from the previous months (see correlation chart in Question 4, note the high correlation between S1,S2,S3,S4 and S5 which shows dependency). Hence based on this we can see the population of customer’s current debt status.

In the pie chart we can observe that 69.5% of customers have minimum payment status (note that in the visualisation preparation phase we created a string version of S1 for this reason).

Moreover, we can observe that 14.9% have delayed payments. Both of these statuses are important in showing the ability of customers to repay their loans on time. Minimum sum payment is generally indicative of customers who cannot repay the full amount as they cannot afford to. In general, it is understood that customers will repay the balance of their debt they moment they are able to as they would end up paying out additional interest in the long run. This means that the 69.5% of customers who fall under minimum payment status are more likely to default and join the 14.9% of those who have delayed payment status in the future should nothing be done by the bank. The conclusion is that the bank may have loose lending practices and may need to be more discerning in terms of the LIMIT value they assign to each customer.

A second related Pie chart was created with a more filtered view specifically to those customers with 0 as a BALANCE value. This was done as further investigation from Analysis 3 as one method to increase the amount of information regarding the level of dormancy out of all the empty bank accounts. Here using the same variable of S1\_str we can see that fortunately, on one hand, most of the accounts are likely to be active still. This is because there is only 0.1% (1 record) of delayed payments showing that the vast majority of owners of the empty bank accounts are still actively using their credit cards. What is concerning however, is the majority of these empty bank account owners are under minimum sum payment status. Assuming that most of these customers only own this one bank account, they are most at risk of defaulting as all they earn is used to pay back their debts.

**Visualisation 5**

**Analysis 5**

**Question 4**

Before pre-processing it is generally good practice to view a correlation heatmap of all the variables. This would give an idea of the general independence of the variables, particularly the explanatory (input) variables.

As we can see in this diagram, based on color most of the variables are independent and those that have a high correlation are expected as they are cumulative of each other (e.g, B(n) and S(n)).

The overall approach is to do a standardized and non-standardized multi-variate regression model and evaluate their performance with standard error statistics. Generally, both variations of the models follow the same steps but with the difference that the standardized model scales the input and target variables to the same plane where the mean is as close to 0 as possible for all variables.

The standardized model is useful in bringing all variables to the same unit of measurement for a more even treatment.

**Data Pre-processing (General)**

In this phase, we will need to encode the some of the nominal variables where the range of values are not binary. For instance, EDUCATION values associated to the level of education are not binary and furthermore are nominal. On the other hand, GENDER is binary. Hence, we do not need to encode GENDER, but we do need to encode variables like EDUCATION. This includes MARITAL. This was each status value (EDUCATION: 0,1,2,3) is listed as its own separate binary column making the values binary values which the linear regression model can analyse correctly. This was done for the MARITAL column as well.

**Data Pre-processing (General) CODE**

#TO set the encoding parameters for EDUCATION and MARITAL

#Prefix dictionaries

{'EDUCATION':'EDU'}.items()

{'MARITAL':'MAR'}.items()

def onehot\_encode(df, column\_dict):

df = df.copy()

#Allows mapping of prefixes to each value column

for column, prefix in column\_dict.items():

dummies = pd.get\_dummies(df[column], prefix=prefix)

df = pd.concat([df, dummies], axis=1)

#To drop the old EDUCATION and MARITAL columns

df = df.drop(column, axis=1)

return df

**Data Pre-processing (Non-Standardized)**

In this phase we pre-process the data for multi-variate regression model creation without standardization. The code executes the necessary parameters required to create a copy of the ECA\_data dataset, drop the unnecessary ‘ID’ column, pass encoding parameters, and finally split the dataset into x (input) and y (target) variables to be identified in the regression model.

We can verify the success of our model as shown in the Jupiter file by checking the new data frames x and y separately. Note the number of rows and columns as 18,766 rows and 27 columns for x and 18,766 rows and 1 column for y.

**Data Pre-processing (Non-Standardized) CODE**

#Pre-processing code including addition of encoding parameters

#Here we split

def preprocess\_inputs(df):

df = df.copy()

#Drop ID

df = df.drop('ID', axis=1)

#Encode parameters from above included

df = onehot\_encode(

df,

{

'EDUCATION': 'EDU',

'MARITAL': 'MAR'

}

)

#Split x (input) and y (target) variables

y = df[['B1']].copy()

x = df.drop('B1', axis=1).copy()

return x,y

#Passes through our ECA\_data pandas dataframe through the above parameters

x, y = preprocess\_inputs(ECA\_data)

**Train-Test Split Creation (Non-Standardized)**

In this phase we set the parameters for splitting the data further into 4 groups: x\_train, x\_test and y\_train, y\_test. These segmented datasets are determined by a 70% to 30% train test ratio respectively. The random state of 101 is to ensure that the model returns the same result at state 101 each time it is run. If one does not determine random state each time the program is run, one will get different model results in terms of predictions and quality of said predictions.

**Train-Test Split Creation (Non-Standardized) CODE**

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = 0.3, random\_state = 101)

print('(x) population data totals:', x.shape)

print('(y) population data totals:',y.shape)

print('(x\_train) training data totals:', x\_train.shape)

print('(y\_train) training data totals:', y\_train.shape)

print('(x\_test) testing data totals:', x\_test.shape)

print('(y\_test) testing data totals:', y\_test.shape)

By running the above code we get the following output

(x) population data totals: (18766, 27)

(y) population data totals: (18766, 1)

(x\_train) training data totals: (13136, 27)

(y\_train) training data totals: (13136, 1)

(x\_test) testing data totals: (5630, 27)

(y\_test) testing data totals: (5630, 1)

Note that 5630 is 30% of the total population count which is indicative of the success of the splitting.

**Model Creation (Non-Standardized)**

Next is the model creation phase which uses sklearn LinearRegression() to create a multi-variate regression model. Here the nominated x\_train y\_train values are used to run the iterations of training the model on ‘seen’ data. Finally, the ‘predictions’ column is generated to show the predicted values of the target variable y\_test by running the x\_test input variables using the trained model.

**Model Creation (Non-Standardized) CODE**

#Model creation parameters using Sklearn LinearRegression

model = LinearRegression()

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

#Runs the trained program using the unseen test datasets for both input and target

predictions = pd.DataFrame(model.predict(x\_test), columns = y.columns)

#Generates the predictions dataframe for preview and also to verify that model outputs the testing target variable.

predictions

**Model Evaluation (Non-Standardized)**

Lastly evaluative statistics are generated to view the model’s performance and ability to predict. Here the predicted values are compared to the actual values where ‘predictions’ variable (Predicted values) is compared to the y\_test variable (Actual values).

Based on the output the model seems to have performed relatively well with a model accuracy deemed at 95%. Model accuracy is the R^2 value in percentage. However, the model seems to have overly inflated MSE, MAE, RMSE and MAPE which is expected given that many of the overall data frame variables are huge. Moreover, it is expected because all the variables use different units of measurement. Hence, in this case it would be difficult to determine the model’s predictive ability on these measures. What is good about the non-standardized model is that we would not need to rescale the predicted values.

mean\_squared\_error : 235095720.5511057

mean\_absolute\_error : 5867.383956430697

mean\_absolute\_percentage\_error: 1.930679578795263e+18

model\_accuracy: 95.49848646852139

**Data Pre-processing, Train-Test split, Model Creation & Evaluation (Standardized)**

Same approach, rationale as non-standardized. But code has different naming conventions for x and y. Variables: x is now x\_scaled and y is now y\_scaled.

The only code that is different is in the pre-processing phase as shown below where we use the StandardScaler() option from sklearn. This brings all the variables down to the same unit of measurement by standardisation allowing for more even comparison and treatment of variables. Note that both input and target are scaled in order to get a more accurate representation of model performance. The additions to the original non standardised codes are highlighted in yellow.

**Data Pre-processing (Standardized) CODE**

def preprocess\_inputs(df):

df = df.copy()

scaler = StandardScaler()

# Drop ID

df = df.drop('ID', axis=1)

df = onehot\_encode(

df,

{

'EDUCATION': 'EDU',

'MARITAL': 'MAR'

}

)

#Split and rescale x and y variables

y\_scaled = df[['B1']].copy()

y\_scaled = pd.DataFrame(scaler.fit\_transform(y\_scaled), columns = y\_scaled.columns)

x\_scaled = df.drop('B1', axis=1).copy()

x\_scaled = pd.DataFrame(scaler.fit\_transform(x\_scaled), columns = x\_scaled.columns)

return x\_scaled,y\_scaled

x\_scaled, y\_scaled = preprocess\_inputs(ECA\_data)

**Model Evaluation (Standardized) CODE**

Here we can see that the standardised model generates more consistent performance scores where the model accuracy is now consistent with the MSE, MAE and MAPE.

mean\_squared\_error : 0.0454535073889768

mean\_absolute\_error : 0.08158416023021545

mean\_absolute\_percentage\_error: 1.7958677295830767

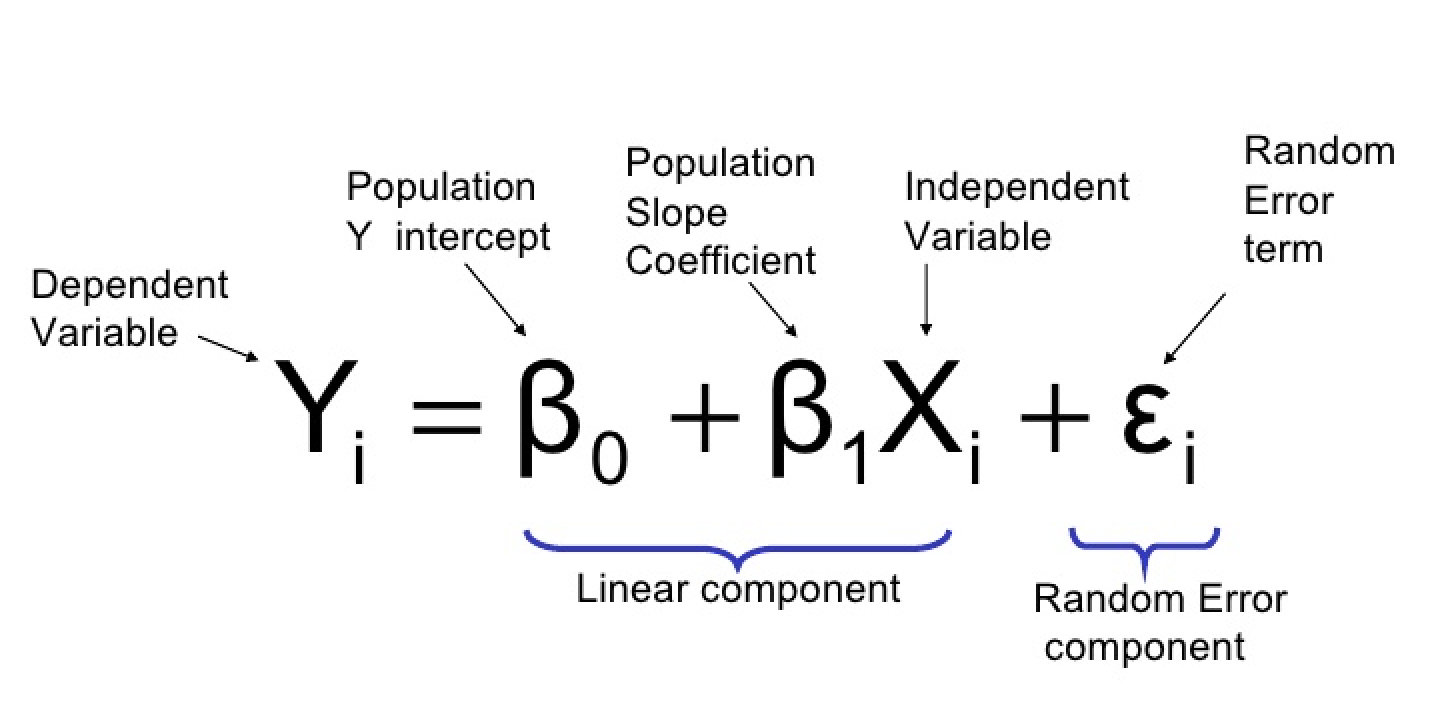
model\_accuracy: 95.49848646852134

However, the main drawback of the standardized model is that the target variable predicted results are all normalised as well. Hence, both models should be used together, the non-standardized for prediction results generation and the standardized version for tracking model performance over time.

**Question 5**

To get the coefficient equation one needs the corresponding coefficients to each variable Xi.

Regression equations follow the template below:



For the non-standardized model, the co-efficients listed in the output of the code

#To print the coefficients and constant of the regression equation

r\_sq=model.score(x\_test, y\_test)

print(f"coefficient of determination: {r\_sq}")

print(f"intercept: {model.intercept\_}")

print(f"coefficients: {model.coef\_}")

They are re-arranged in a pandas dataframe in the following code

#To re-arrange the list into a dataframe with the corresponding variables to co-efficient

Coefficients = pd.DataFrame(model.coef\_, columns = x.columns)

Coefficients

Coefficients.round(decimals=3)