A picture containing food, plate

Description automatically generated

**ANL252**

**Python for Data Analytics**

**End-of-course Assessment 01**

**Koo Chi Hang**

**K2081853**

**5 September 2022**

Table of Contents

[Question 1 3](#_Toc113242004)

[Question 2 4](#_Toc113242005)

[Method 1 4](#_Toc113242006)

[Method 2 5](#_Toc113242007)

[Method 3 6](#_Toc113242008)

[Method 4 9](#_Toc113242009)

[Question 3 11](#_Toc113242010)

[1st Insight 11](#_Toc113242011)

[2nd Insight 12](#_Toc113242012)

[3rd Insight 13](#_Toc113242013)

[4th Insight 14](#_Toc113242014)

[5th Insight 16](#_Toc113242015)

[Question 4 19](#_Toc113242016)

[Question 5 23](#_Toc113242017)

[References 24](#_Toc113242018)

# Question 1

*#Importing necessary libraries in python*

import pandas as pd

*#Importing dataframe from ECA\_data file on desktop*

data = pd.read\_csv("/Users/chihang/Desktop/ECA\_data.csv")

*#Setting up a list that only consist of categorical variables*

categorical\_list = data.loc[:,['ID','RATING','GENDER','EDUCATION','MARITAL','S1','S2','S3','S4','S5']]

*#Setting up a list that only consist of numeric columns*

numeric\_list = data.loc[:,['LIMIT','BALANCE','INCOME','B1','B2','B3','B4','B5','R1','R2','R3','R4','R5']]

print(categorical\_list)

print(numeric\_list)

# Question 2

For the purpose of this question, I have split the 4 data pre-processing tasks into separate parts and methods.

## Method 1

*#Importing necessary libraries in python*

import pandas as pd

*#(1st method)Specify and remove missing values*

#Define specific strings such as str with white spaces("") as missing values during the import process

clean\_data\_one = pd.read\_csv("/Users/chihang/Desktop/ECA\_data.csv",

na\_values = "na\_string", na\_filter = True)

*#Locate missing data in the dataframe*

clean\_data\_one.isnull().sum(axis = 0)

*#Remove rows with missing values*

clean\_data\_one.dropna(axis = 0, how = "any", inplace=True)

*#check that the missing values have indeed been removed*

clean\_data\_one.isnull().sum(axis = 0)

Initially these were columns and the number of missing values that were identified by the isnull() function. There were **13 missing values** in ‘EDUCATION’ column and **38 missing values** in ‘MARITAL’ column.

Table

Description automatically generated

(Figure 1: Output of identification of missing values)

These missing values were later dropped from the data set using the drop.na() function. However this method is not recommended as it would lead to the loss of precious data.

## Method 2

*#(2nd method)Replacing missing data with mean/mode/median*

*#Importing necessary libraries in python*

import pandas as pd

*#Define specific strings such as str with white spaces("") as missing values during the import process*

clean\_data\_two = pd.read\_csv("/Users/chihang/Desktop/ECA\_data.csv",

na\_values = "na\_string", na\_filter = True)

*#Locate missing data in the dataframe*

clean\_data\_two.isnull().sum(axis = 0)

*#EDUCATION and MARITAL are categorical, hence will be using mode to replace the missing values*

for column in ['EDUCATION', 'MARITAL']:

clean\_data\_two[column].fillna(clean\_data\_two[column].mode()[0], inplace=True)

*#Check that the missing values have indeed been replaced*

clean\_data\_two.isnull().sum(axis = 0)

As identified in method 1, the missing values were from **categorical columns**. Hence, this method would make use of mode to replace the identified missing values and is the **recommended method** for data pre-processing in this report as it is able to prevent 4wastage of data.

## Method 3

*#(3rd method)Removing outliers through inter quartile range*

*#Importing necessary libraries in python*

import pandas as pd

*#Importing dataframe from ECA\_data file on desktop*

data = pd.read\_csv("/Users/chihang/Desktop/ECA\_data.csv")

*#Setting up a list that only consist of numeric columns*

*#Only using numeric columns as there are only a fixed possible outcomes for categorical columns*

*#Removed R3 column due to index error that cant be solved*

numeric\_list\_two = data.loc[:,['LIMIT','BALANCE','INCOME','B1','B2','B3','B4','B5','R1','R2','R4','R5']]

*#Check number of values in the dataframe before cleaning*

numeric\_list\_two.info()

*#Setting of interquartile range according to 0.25 and 0.75 percentile*

Q1 = numeric\_list\_two.quantile(0.25)

Q3 = numeric\_list\_two.quantile(0.75)

IQR = Q3 - Q1

*#Removing of outliers that exceeds the interquartile range*

cleaned\_numeric\_list = numeric\_list\_two[

~((numeric\_list\_two < (Q1 - 1.5 \* IQR))

|(numeric\_list\_two > (Q3 + 1.5 \* IQR))).any(axis=1)]

*#Check number of values in the dataframe after cleaning*

cleaned\_numeric\_list.info()

**Before the reduction of outliers:**

Table

Description automatically generated

(Figure 2: Output before the reduction of outliers)

**After the reduction of outliers that exceeds the IQR range:**

Table

Description automatically generated

(Figure 3: Output after the reduction of outliers)

As we can observe from the 2 figures, the process of removing rows with outliers have resulted in **a reduction of 5662 rows**. This reduction is fairly significant as it had previously represented 30% of the original set. Therefore, this would method would only be recommended for analysis which requires an extreme level of accuracy.

## Method 4

*#(4th method)Discretisation for income*

*#Importing necessary libraries in python*

import pandas as pd

import numpy as np

*#Importing dataframe from ECA\_data file on desktop*

data = pd.read\_csv("/Users/chihang/Desktop/ECA\_data.csv")

*#Using cut() function to bin the income data into 5 different categories for easier identification*

data['INCOME'] = pd.cut(np.array(data['INCOME']), 5,

labels=["Very Low Income",

"Low Income", "Middle Income", "High Income",

"Very High Income"])

*#Shows dataframe*

data

Table

Description automatically generated

(Figure 4: First 5 rows of the dataset after discretisation)

As the credit limit is already generated based on the income, we believe categorising the income level would make it easier for the management to understand the income demographics of their customers.

# Question 3

For clearer articulation of this question, the 5 relevant insights of the data set will be explained and analyses individually.

## 1st Insight

*#(1st Insight)Descriptive statistics for numerical columns*

*#Importing necessary libraries in python*

import matplotlib.pyplot as plt

import pandas as pd

*#Coverting numeric columns into a new dataframe for easier processing*

Numeric\_df = pd.DataFrame(clean\_data\_two,columns

=['LIMIT','BALANCE','INCOME',

'B1','B2','B3','B4','B5','R1','R2','R3','R4','R5'])

*#Use describe function to showcase the descriptive statistics of the data rounded to up 2 decimal place*

Numeric\_df.describe().round(2)

Table

Description automatically generated

(Figure 5: Output of descriptive statistics of numeric data sets)

Figure 5 is useful in providing an overall view of the features in the data set. It is able to show the count, mean, standard deviation, minimum value, maximum value and the percentile values of data across multiple columns. An example of an interesting observation would be that the minimum amount of billables ranges from -$209051 to -$69777 across the 5 months.

## 2nd Insight

*#(2st Insight)Histogram of credit limits*

*#Importing necessary libraries in python*

import matplotlib.pyplot as plt

*#Will be using the cleaned data set w/ missing value replaced (2nd method)*

*#Convert the "LIMIT" column of the dataframe into a list*

limit\_list = list(clean\_data\_two["LIMIT"])

*#rmin and rmax are the minimum and maximum value of the list*

rmin = min(limit\_list)

rmax = max(limit\_list)

*#Allocating of areas for the figure*

plt.figure(figsize=(6,6), dpi=80)

*#Histogram with 30 bins, ranging from rmin to rmax, bins are centered between the bin edges, vertical orientation, relative width 0f 60%*

plt.hist(limit\_list, bins=30, range=(rmin, rmax), align="mid", orientation="vertical", rwidth=0.6)

plt.title("Histogram of Credit Limit")

plt.xlabel("Dollars ($)")

plt.ylabel("Frequency")

plt.show()

Chart, histogram

Description automatically generated

(Figure 6: Histogram of credit limit)

We can observe from figure 6 that the highest frequencies of credit limit lies between the short end of the range, with occasional spikes of frequencies at $200000 and $500000. Moreover, the frequencies drastically decrease from $500000 onwards, with only a small handful of customers being entitled to a larger than $500000 worth of credit limit.

## 3rd Insight

*#(3rd Insight)Line graph of customer's billable according months*

*#Importing necessary libraries in python*

import matplotlib.pyplot as plt

*#Will be using the cleaned data set w/ missing value replaced (2nd method)*

*#Creating a list of the total sum of billables divided by 10000*

Billable\_list = clean\_data\_two.loc[:,['B1','B2','B3','B4','B5']].sum()/10000

*#Line graph of billables with y-axis range from 600000 to 1000000, font size 12, with markers and gridlines*

plt.plot(Billable\_list,marker='o')

plt.title('Total amount of billables according to months', fontsize=12)

plt.xlabel('Months', fontsize=12)

plt.ylabel('Total Billables in $(`0000)', fontsize=12)

plt.ylim(60000,100000)

plt.grid(True)

plt.show()

Chart, line chart

Description automatically generated

(Figure 7: Line graph of total amount of billables)

From figure 7, we can observe that there is a decreasing trend of billables owed across the months. This could be due to factors such as customers repaying earlier or that the company is doing a better job in recovering its receivables.

## 4th Insight

*#(4th Insight)Pie chart of customer's ratings*

*#Importing necessary libraries in python*

import matplotlib.pyplot as plt

import pandas as pd

*#Replace variables 0 and 0 with Good and Bad respectively*

Ratings\_df = clean\_data\_two['RATING'].replace(0, "Good").replace(1, "Bad")

*#Check the value counts of each unique variable*

Ratings\_df.value\_counts()

*#Creating the index list*

Ratings\_index = ['Good', 'Bad']

*#Creating the data list*

Ratings\_data = [14654, 4115]

*#Setting the format of the percentage*

def my\_fmt(x):

print(x)

return '{:.4f}%\n({:.0f})'.format(x, 18769\*x/100)

*#Creating pie chart of credit ratings*

plt.pie(Ratings\_data, labels = Ratings\_index, autopct=my\_fmt, shadow=True)

plt.title("Percentage of credit ratings amongst customers")

*#Show pie chart*

plt.show()

Chart, pie chart

Description automatically generated

(Figure 8: Pie chart of credit ratings percentages)

From figure 7, we can observe that 78% of the company`s customers have a good credit rating while the remaining 22% of customers are tagged with bad credit rating. This data is important to the company as customers with bad credit ratings are more likely to default their payment which may lead to bad debts. Hence, this would help the company to maintain a healthy balance of customers and reduce having too many customers with bad credit ratings.

## 5th Insight

*#(5th Insight)repayment status across months*

*#Importing necessary libraries in python*

import matplotlib.pyplot as plt

import pandas as pd

*#Coverting numeric columns into a new dataframe for easier processing*

Status\_df = pd.DataFrame(clean\_data\_two,columns =['S1','S2','S3','S4','S5'])

*#Function to show all columns and names in the dataframe*

pd.options.display.max\_columns = None

pd.options.display.max\_rows = None

*#Show total counts of each variables in the dataframe*

*#Realised that that there were NaN(missing)values*

Status\_df\_two = Status\_df.apply(pd.Series.value\_counts)

*#Check for number off missing values*

Status\_df\_two.isnull().sum(axis = 0)

*#Use mode to replace categorical missing values*

for column in ['S1','S2','S3','S4','S5']:

Status\_df\_two[column].fillna(Status\_df\_two[column].mode()[0], inplace=True)

*#check that the missing values have indeed been replaced*

Status\_df\_two.isnull().sum(axis = 0)

*#Show final table*

print(Status\_df\_two)

Table

Description automatically generated

(Figure 9: Screenshot of repayment status data frame)

Figure 9 acts a time series data frame that allows us to observe repayment data across the 5 months. Generally, majority of payments are prompt and paid in minimum sum across these 5 months. However, we can observe that there are some unusual spikes in repayment status. For example, there were 9 customers in S2 which had a delayed payment of 7 months. However in S3, the number increased to 41 customers which may suggest a need for the company to investigate the reason behind this abnormal increase.

# Question 4

*#Importing neccessary libraries in python*

import matplotlib.pyplot as plt

import numpy as np

from sklearn import linear\_model, metrics

from sklearn.linear\_model import LinearRegression

*#Coverting numeric columns into a new dataframe for easier processing*

X\_df = clean\_data\_two.loc[:,['LIMIT','BALANCE','RATING','GENDER','EDUCATION','MARITAL']]

Y\_df = clean\_data\_two.loc[:,['B1']]

*#Define X variables and response vector Y*

X = X\_df

Y = Y\_df

**Explanation:**

Up till this part, what we have done is to create features variables in order to model the data. The X variable contains independent variables (E.g data from LIMIT, BALANCE, etc columns) while the Y variable only contain one dependent variables (E.g billable data from B1 columns).

*#Split X and Y into training and testing sets (60% training data & 40% test data)*

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.4,

random\_state=1)

**Explanation:**

In this part, train\_test\_split method is used to create training and testing sets for the model. Test size was set as 0.4 which means that 60% of the data will be used for **training sets** while 40% of the remaining data will be used for **testing sets**. This will help the model to improve its accuracy.

*#Create linear regression object*

reg = linear\_model.LinearRegression()

**Explanation:**

LinearRegression() function was used to create the regression model for our data set.

*#Train the model using the training sets*

reg.fit(X\_train, Y\_train)

**Explanation:**

After creating the model, fit() function was applied to allow the model to gain knowledge of the statistics of the training sets.

*#Make predictions using the testing set*

Y\_pred = reg.predict(X\_test)

*#Predict the Y values for the first 5 row in the testing set*

Y\_pred[0:5,0]

**Explanation:**

Predict() function was applied to make predictions based on the testing set. This can be further applied to rows that are beyond the current data set.

*#Rounding off parametres from regression model to 2 decimal places*

r\_square = np.round(reg.score(X\_test, Y\_test),2)

Coefficients = np.round(reg.coef\_, 2)

Intercept\_c = np.round(reg.intercept\_, 2)

*#Variance score*

print("Coefficient of determination (R^2):",r\_square)

*#Regression coefficients*

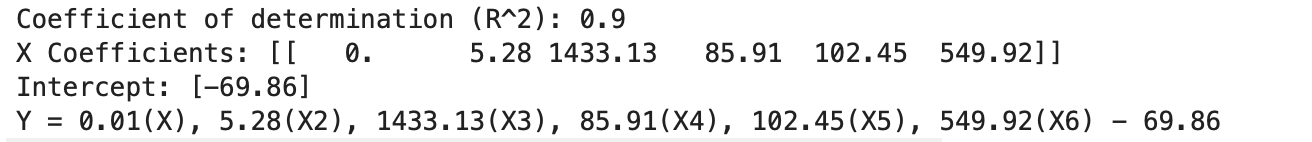
print("X Coefficients:",Coefficients)

*#Intercept*

print("Intercept:",Intercept\_c)

*#After obtaining the values, plot regression equation*

print("Y = 0.01(X), 5.28(X2), 1433.13(X3), 85.91(X4), 102.45(X5), 549.92(X6) - 69.86")



(Figure 10: Screenshot of regression equation output)

**Explanation:**

Score(), coef\_() and intercept() functions were utilised in this part to generate values of the coefficients and intercept in order to form the linear regression equation which will be explained more in Question 5. The important factor to highlight is this part would be the **R-squared value** which indicates the accuracy or how well fitted the model is. The higher the R-squared value, the better the model fits the data set.

*#Plot for residual error*

*#Setting plot style*

plt.style.use('fivethirtyeight')

*#Plotting residual errors in training data*

plt.scatter(reg.predict(X\_train), Y\_train - reg.predict(X\_train),

color = "green", s = 2, label = 'Train data')

*#Plotting residual errors in test data*

plt.scatter(reg.predict(X\_test), Y\_test - Y\_pred,

color = "blue", s = 2, label = 'Test data')

*#Plotting line for zero residual error*

plt.hlines(y = 0, xmin = 0, xmax = 700000, linewidth = 2)

*#Plotting legend*

plt.legend(loc = 'upper right')

*#Plot title*

plt.title("Residual errors")

*#show plot*

plt.show()

**Explanation:**

Residual errors is calculated by subtracting the predicted value by the expected value. As the current linear regression model is one with multiple variables, we believe that a residual plot would be more efficient in helping to visualise the fit of the model as compared to 6 individual regression plot.

Chart, scatter chart

Description automatically generated

(Figure 11: Residual plot of regression model)

# Question 5

As shown in figure 10, the regression equation is

**Y = 0.01+ 5.28+ 1433.13+ 85.91+102.45+ 549.92 - 69.86**

Where,

**Y** = Billable amount in month B1

= credit limit in $

= Account balance in $

= Rating (0: Good, 1:Bad)

= Gender (0: Male, 1: Female)

= Education Status (0: Others, 1: Postgraduate, 2: Tertiary, 3: High School)

= Marital Status (0: Others, 1: Singles, 2: Married)

The regression equation can be inferred as such,

1. For every $ of credit limit that the customer is entitled to, his/her billable amount in B1 will increase by **$0.01**

2. For every $ in the customer’s account balance, his/her billable amount in B1 will increase by **$5.28**.

3. If the customer has a bad rating, his/her billable amount in B1 will increase by **$1433.13**.

4. If the customer is a female, her billable amount in B1 will increase by **$85.91**.

5. For each tier of decrease in the customer`s education level, his/her billable amount in B1 will increase by **$102.45.**

6. If the customer is single, his/her billable amount in B1 will increase by **$549.92**.

7. If the customer is married, his/her billable amount in B1 will increase by **$1099.84**.

The R-squared (Coefficient of determination) is 0.9 which shows that 90% of the data can be explained by the regression model which indicates that the predicted values are a very good fit for this data set. It can also be inferred that 90% of the variation in billable amounts in B1 can be explained by the variation of independent variables (to **)**

With reference to figure 11, we can observe that majority of the residual error are located near to the zero residual line, which supports our findings that there the predicted values are highly accurate.

# References

Dan,N. (n.d.). *Multiple linear Regression with Python*. <https://stackabuse.com/multiple-linear-regression-with-python/>

GeeksforGeeks. (2022, August 22). *Linear Regression (Python Implementation)*. <https://www.geeksforgeeks.org/linear-regression-python-implementation/>

GeeksforGeeks. (2022, February 21). *How to Create a Residual Plot in Python.* <https://www.geeksforgeeks.org/how-to-create-a-residual-plot-in-python/>

GeeksforGeeks. (2022, July 11). *Multiple Linear Regression With scikit-learn.* <https://www.geeksforgeeks.org/multiple-linear-regression-with-scikit-learn/>

Jason, B. (2017, January 11). *How to Model Residual Erros to Correct Time Series Forecasts with Python.* <https://machinelearningmastery.com/how-to-connect-model-input-data-with-predictions-for-machine-learning/>

Jason, B. (2019, November 15). *How to Connect Model Input Data With Predictions for Machine Learning*. <https://machinelearningmastery.com/how-to-connect-model-input-data-with-predictions-for-machine-learning/>

Nagesh, S, C. (2019, March 29). *A Beginner’s Guide to Linear Regression in Python with Scikit-Learn*. <https://www.kdnuggets.com/2019/03/beginners-guide-linear-regression-python-scikit-learn.html>

Sckitlearn. (n.d.). *Linear Regression Example.* <https://scikit-learn.org/stable/auto_examples/linear_model/plot_ols.html>