

**ANL252 (Online)**

**Python for Data Analytics**

# **End-of-Course Assessment**

**July 2022 Presentation**

**Submitted by:**

| **Name** | **PI No.** |
| --- | --- |
| Ng Wei Ling | B2210595 |

**Tutorial Group:** T09

**Instructor’s Name:** Kurmar Munish

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

**Question 1**

**This credit facility dataset to be analyzed comprises records of customers’**

**demographics, amount owed, repayment history/status etc.**

**The data dictionary of this dataset is depicted in Appendix 1.**

**------------------**

**List the categorical and numeric variables in this dataset.**

In Jupyter Notebook:

import pandas as pd

import numpy as np

df = pd.read\_csv('ECA\_data.csv')

print(df)

print(df.dtypes.value\_counts())

df.dtypes == 'object'

# Seperate Numerical and variables and categorical variable

num\_vars = df.columns[df.dtypes != 'object']

cat\_vars = df.columns[df.dtypes == 'object']

print(num\_vars)

print(cat\_vars)

Plain text:

**Numerical variables** are 'ID', 'LIMIT', 'BALANCE', 'INCOME', 'RATING', 'GENDER', 'EDUCATION', 'MARITAL', 'AGE', 'S1', 'S2', 'S3', 'S4', 'S5', 'B1', 'B2', 'B3', 'B4', 'B5', 'R1', 'R2', 'R4', 'R5']

**Categorical variable** is only ‘R3’.

**Question 2**

**Conduct four (4) data pre-processing tasks for the analysis of the data, explaining**

**results obtained.**

# Importing of dataset

# To remove unwanted datas, records which isn't required which can also be refered as meaningless data.

**1.Data Cleaning**

In Jupyter Notebook:

# To find the missing values and fill the missing values

df[num\_vars].isnull().sum().sort\_values(ascending = False)

# Filling empty space with NaN

for col in df.columns:

print(f"{col}:\t\t\t{len(df[df[col] == ''])}")

df[col].replace('', np.nan, inplace=True)

df.dropna(inplace=True)

# To check if empty space has been replaced

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

# All missing values in the dataset has been replaced by NaN and droppped

**2.Noisy Data**

In Jupyter Notebook:

#To find corrupted data / incorrect datas from column

gender\_list = df['GENDER'].tolist()

for i in range(len(gender\_list)):

if gender\_list[i] >= 2 :

print("Incorrect")

else:

print("Correct")

Education\_list = df['EDUCATION'].tolist()

for i in range(len(Education\_list)):

if Education\_list[i] >= 4 :

print("Incorrect")

else:

print("Correct")

Marital\_list = df['MARITAL'].tolist()

for i in range(len(Marital\_list)):

if Marital\_list[i] >= 3 :

print("Incorrect")

else:

print("Correct")

Rating\_list = df['RATING'].tolist()

for i in range(len(Marital\_list)):

if Rating\_list[i] >= 2 :

print("Incorrect")

else:

print("Correct")

Age\_list = df['AGE'].tolist()

for i in range(len(Age\_list)):

if Age\_list[i] <= -1:

print(i)

print(Age\_list[i])

Plain text:

With the above output, i have checked 'Education', 'Gender', 'Marital' and 'Ratings' columns and i don't find any incorrect data that needs to be remove.Hence, there are no corrupted data / incorrect data in those columns.

However, in column 'AGE', there are values which is less than 0 hence there is incorrect data recorded as age must be a postive value. Values less than 0 has been replaced with 'Nan'

**3. Spliting Dataframe by Position**

In Jupyter Notebook:

half\_df = len(df) // 2

first\_half = df.iloc[:half\_df,]

print(first\_half)

# From the above graph, i have split the dataframe into half by spliting row-wise at the midpoint.First i find the midpoint by finding the lenght and diving it by two and futher spliting it by using .iloc

def split\_dataframe\_by\_position (df, splits):

dataframes = []

index\_to\_split = len(df) // splits

start = 0

end = 1

for split in range (splits):

temporary\_df = df.iloc[start:end, :]

dataframes.append (temporary\_df)

start += index\_to\_split

end += index\_to\_split

return dataframes

split\_dataframes = split\_dataframe\_by\_position(df,3)

print(split\_dataframes[1])

# from the above data, i have split the dataframe into multiple sections

### 4. Data reduction

In Jupyter Notebook:

# To increase storage efficiency and reduce data storage and analysis costs

.describe give us a number Standard Statistic for each numerical. It ignores the non-numerical dataset

df.describe(exclude='number')

Plain text:

In column R3, it contains little varience. From the 18,769 oberservation, there are only 5,218 unqiue with 0 on the top 3909 times.

**Question 3**

**Articulate five (5) relevant insights of the data, with supporting visualization for each insight.**

In Jupyter Notebook:

#Create pivot table for the result of "RATING" by "GENDER"

pd4 = pd.pivot\_table(data=df,

index=['GENDER'],

values=['RATING'],

aggfunc='count')

#Show table

pd4

#Plot bar graph

pd4.plot(kind="bar", title = "Result of Rating by Gender")

Plain text:

Interesting Insight 1

From the above chart with respect to the Gender and customer's rating, we can conclude that there are more Females who provided a customer rating as compared to the number of Males. '0' representing Males, '1' representing Females, we can conclude that are more willingly to rate the service

In Jupyter Notebook:

#Create pivot table for the number of "INCOME by "EDUCATION"

pd5 = pd.pivot\_table(data=df,

index=['EDUCATION'],

values=['INCOME'],

aggfunc='mean')

#Show table

pd5

#Plot bar graph

pd5.plot(kind="bar", title = "average INCOME according to EDUCATION")

Plain text:

Interesting Insight 2

From the above chart with respect to the income and customer's highest education, we can say that '1' which is people with Postgraduate education level has the highest average income followed by '0' (Others), '2' (Tertiary) and '3' (High School).

This shows that education level might have a relation with income.

In Jupyter Notebook:

#Create pivot table for the "BALANCE" value by "MARITAL" status

pd6 = pd.pivot\_table(data=df,

index=['MARITAL'],

values=['BALANCE'],

aggfunc='mean')

#Show table

pd6

#Plot bar graph

pd6.plot(kind="bar", title = "average BALANCE value by MARITAL status"))

Plain text:

Interesting Insight 3

From the above chart of average balance according to marital status, we can say that '1' (Single) which has the highest average Balance followed by '2' (Married) then '0' (Others).

It can be inferred that Singles usually have higher credit balance and could be a higher spender which is important information for banks and credit card companies. Higher spender is equals to better potential customers.

In Jupyter Notebook:

#Create pivot table for the total "LIMIT" value by "GENDER"

pd7 = pd.pivot\_table(data=df,

index=['GENDER'],

values=['LIMIT'],

aggfunc='count')

#Show table

pd7

#Plot bar graph

pd7.plot(kind="bar", title = "total LIMIT value by GENDER")

Plain text:

Interesting Insight 4

From the above chart with respect to the customer's total card limit and gender, Female('1') has the higher limit value than the Male('0'). From this, i can conclude that females tend to set a higher limit values as they tend to spend more for shopping, groceries and daily expense.

**Question 4**

**Perform linear regression modelling to predict the variable, B1, explaining the approach taken, including any further data pre-processing.**

In Jupyter Notebook:

import seaborn as sb

from sklearn.linear\_model import LinearRegression

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

import matplotlib.pyplot as plt

sb.set()

from sklearn import linear\_model

**Using INCOME to predict B1 values**

linreg = LinearRegression()

b1 = pd.DataFrame(df['B1'])

income = pd.DataFrame(df['INCOME'])

X\_train, X\_test, y\_train, y\_test = train\_test\_split(income, b1, test\_size = 0.25)

linreg.fit(X\_train, y\_train)

linreg = LinearRegression()

print('Intercept of Regression \t: b = ', linreg.intercept\_)

print('Coefficients of Regression \t: a = ', linreg.coef\_)

y\_train\_pred = linreg.predict(X\_train)

y\_test\_pred = linreg.predict(X\_test)

regline\_x = X\_train

regline\_y = linreg.intercept\_ + linreg.coef\_ \* X\_train

f = plt.figure(figsize=(16, 8))

plt.scatter(X\_train, y\_train)

plt.plot(regline\_x, regline\_y, 'r-', linewidth = 3)

plt.xlabel('Income', fontsize=18)

plt.ylabel('B1', fontsize=18)

plt.title('Train Dataset', fontsize=25)

plt.show()

# Explained Variance (R^2)

print("Explained Variance (R^2) \t:", linreg.score(X\_train, y\_train))

# Mean Squared Error (MSE)

def mean\_sq\_err(actual, predicted):

'''Returns the Mean Squared Error of actual and predicted values'''

return np.mean(np.square(np.array(actual) - np.array(predicted)))

mse = mean\_sq\_err(y\_train, y\_train\_pred)

print("Mean Squared Error (MSE) \t:", mse)

print("Root Mean Squared Error (RMSE) \t:", np.sqrt(mse))

# Predict according to the test dataset, to test how accurate the linear regression model is

y\_test\_pred = linreg.predict(X\_test)

# Plot the Linear Regression line

f = plt.figure(figsize=(16, 8))

plt.scatter(X\_test, y\_test)

plt.scatter(X\_test, y\_test\_pred, color = "r")

plt.xlabel('Income', fontsize=18)

plt.ylabel('B1', fontsize=18)

plt.title('Test Dataset', fontsize=25)

plt.show()

# Explained Variance (R^2)

print("Explained Variance (R^2) \t:", linreg.score(X\_test, y\_test))

# Mean Squared Error (MSE)

def mean\_sq\_err(actual, predicted):

'''Returns the Mean Squared Error of actual and predicted values'''

return np.mean(np.square(np.array(actual) - np.array(predicted)))

mse = mean\_sq\_err(y\_test, y\_test\_pred)

print("Mean Squared Error (MSE) \t:", mse)

print("Root Mean Squared Error (RMSE) \t:", np.sqrt(mse))

### Using B2 to predict B1

linreg2 = LinearRegression()

b1 = pd.DataFrame(df['B1'])

b2 = pd.DataFrame(df['B2'])

X\_train, X\_test, y\_train, y\_test = train\_test\_split(b2, b1, test\_size = 0.25)

linreg2.fit(X\_train, y\_train)

print('Intercept of Regression \t: b = ', linreg2.intercept\_)

print('Coefficients of Regression \t: a = ', linreg2.coef\_)

y\_train\_pred = linreg2.predict(X\_train)

y\_test\_pred = linreg2.predict(X\_test)

regline2\_x = X\_train

regline2\_y = linreg2.intercept\_ + linreg2.coef\_ \* X\_train

f = plt.figure(figsize=(16, 8))

plt.scatter(X\_train, y\_train)

plt.plot(regline2\_x, regline2\_y, 'r-', linewidth = 3)

plt.xlabel('B2', fontsize=18)

plt.ylabel('B1', fontsize=18)

plt.title('Train Dataset', fontsize=25)

plt.show()

# Explained Variance (R^2)

print("Explained Variance (R^2) \t:", linreg2.score(X\_train, y\_train))

# Mean Squared Error (MSE)

def mean\_sq\_err(actual, predicted):

'''Returns the Mean Squared Error of actual and predicted values'''

return np.mean(np.square(np.array(actual) - np.array(predicted)))

mse = mean\_sq\_err(y\_train, y\_train\_pred)

print("Mean Squared Error (MSE) \t:", mse)

print("Root Mean Squared Error (RMSE) \t:", np.sqrt(mse))

# Predict according to the test dataset, to test how accurate the linear regression model is

y\_test\_pred = linreg2.predict(X\_test)

# Plot the Linear Regression line

f = plt.figure(figsize=(16, 8))

plt.scatter(X\_test, y\_test)

plt.scatter(X\_test, y\_test\_pred, color = "r")

plt.xlabel('B2', fontsize=18)

plt.ylabel('B1', fontsize=18)

plt.title('Test Dataset', fontsize=25)

plt.show()

# Explained Variance (R^2)

print("Explained Variance (R^2) \t:", linreg.score(X\_test, y\_test))

# Mean Squared Error (MSE)

def mean\_sq\_err(actual, predicted):

'''Returns the Mean Squared Error of actual and predicted values'''

return np.mean(np.square(np.array(actual) - np.array(predicted)))

mse = mean\_sq\_err(y\_test, y\_test\_pred)

print("Mean Squared Error (MSE) \t:", mse)

print("Root Mean Squared Error (RMSE) \t:", np.sqrt(mse))

### Using R1 to predict B1

linreg3 = LinearRegression()

b1 = pd.DataFrame(df['B1'])

r1 = pd.DataFrame(df['R1'])

X\_train, X\_test, y\_train, y\_test = train\_test\_split(r1, b1, test\_size = 0.25)

linreg3.fit(X\_train, y\_train)

print('Intercept of Regression \t: b = ', linreg3.intercept\_)

print('Coefficients of Regression \t: a = ', linreg3.coef\_)

y\_train\_pred = linreg3.predict(X\_train)

y\_test\_pred = linreg3.predict(X\_test)

regline3\_x = X\_train

regline3\_y = linreg3.intercept\_ + linreg3.coef\_ \* X\_train

f = plt.figure(figsize=(16, 8))

plt.scatter(X\_train, y\_train)

plt.plot(regline3\_x, regline3\_y, 'r-', linewidth = 3)

plt.xlabel('R1', fontsize=18)

plt.ylabel('B1', fontsize=18)

plt.title('Train Dataset', fontsize=25)

plt.show()

# Explained Variance (R^2)

print("Explained Variance (R^2) \t:", linreg3.score(X\_train, y\_train))

# Mean Squared Error (MSE)

def mean\_sq\_err(actual, predicted):

'''Returns the Mean Squared Error of actual and predicted values'''

return np.mean(np.square(np.array(actual) - np.array(predicted)))

mse = mean\_sq\_err(y\_train, y\_train\_pred)

print("Mean Squared Error (MSE) \t:", mse)

print("Root Mean Squared Error (RMSE) \t:", np.sqrt(mse))

# Predict according to the test dataset, to test how accurate the linear regression model is

y\_test\_pred = linreg3.predict(X\_test)

# Plot the Linear Regression line

f = plt.figure(figsize=(16, 8))

plt.scatter(X\_test, y\_test)

plt.scatter(X\_test, y\_test\_pred, color = "r")

plt.xlabel('R1', fontsize=18)

plt.ylabel('B1', fontsize=18)

plt.title('Test Dataset', fontsize=25)

plt.show()

# Explained Variance (R^2)

print("Explained Variance (R^2) \t:", linreg3.score(X\_test, y\_test))

# Mean Squared Error (MSE)

def mean\_sq\_err(actual, predicted):

'''Returns the Mean Squared Error of actual and predicted values'''

return np.mean(np.square(np.array(actual) - np.array(predicted)))

mse = mean\_sq\_err(y\_test, y\_test\_pred)

print("Mean Squared Error (MSE) \t:", mse)

print("Root Mean Squared Error (RMSE) \t:", np.sqrt(mse))

### Using S1 to predict B1

linreg4 = LinearRegression()

b1 = pd.DataFrame(df['B1'])

s1 = pd.DataFrame(df['S1'])

X\_train, X\_test, y\_train, y\_test = train\_test\_split(s1, b1, test\_size = 0.25)

linreg4.fit(X\_train, y\_train)

print('Intercept of Regression \t: b = ', linreg4.intercept\_)

print('Coefficients of Regression \t: a = ', linreg4.coef\_)

y\_train\_pred = linreg4.predict(X\_train)

y\_test\_pred = linreg4.predict(X\_test)

regline4\_x = X\_train

regline4\_y = linreg4.intercept\_ + linreg4.coef\_ \* X\_train

f = plt.figure(figsize=(16, 8))

plt.scatter(X\_train, y\_train)

plt.plot(regline4\_x, regline4\_y, 'r-', linewidth = 3)

plt.xlabel('S1', fontsize=18)

plt.ylabel('B1', fontsize=18)

plt.title('Train Dataset', fontsize=25)

plt.show()

# Explained Variance (R^2)

print("Explained Variance (R^2) \t:", linreg4.score(X\_train, y\_train))

# Mean Squared Error (MSE)

def mean\_sq\_err(actual, predicted):

'''Returns the Mean Squared Error of actual and predicted values'''

return np.mean(np.square(np.array(actual) - np.array(predicted)))

mse = mean\_sq\_err(y\_train, y\_train\_pred)

print("Mean Squared Error (MSE) \t:", mse)

print("Root Mean Squared Error (RMSE) \t:", np.sqrt(mse))

# Predict according to the test dataset, to test how accurate the linear regression model is

y\_test\_pred = linreg4.predict(X\_test)

# Plot the Linear Regression line

f = plt.figure(figsize=(16, 8))

plt.scatter(X\_test, y\_test)

plt.scatter(X\_test, y\_test\_pred, color = "r")

plt.xlabel('S1', fontsize=18)

plt.ylabel('B1', fontsize=18)

plt.title('Test Dataset', fontsize=25)

plt.show()

# Explained Variance (R^2)

print("Explained Variance (R^2) \t:", linreg4.score(X\_test, y\_test))

# Mean Squared Error (MSE)

def mean\_sq\_err(actual, predicted):

'''Returns the Mean Squared Error of actual and predicted values'''

return np.mean(np.square(np.array(actual) - np.array(predicted)))

mse = mean\_sq\_err(y\_test, y\_test\_pred)

print("Mean Squared Error (MSE) \t:", mse)

print("Root Mean Squared Error (RMSE) \t:", np.sqrt(mse))

**Question 5**

**State the linear regression equation and explain key insights from the results obtained.**

Plain text:

Linear equation : Y = a + bX where X is the independent variable plotted along the x-axis, Y is the dependent variable, plotted along y-axis. a is the interception point and b is the slope.

It basically measure the extend of relationship between 2 variables known as the correlation coefficient that lies between -1 to +1, showing the strenght of the observed data of 2 variables

The key insights from the result obtained would be the use of R(n),B(n) and S(n) to predit the variable B1. According to the Dataframe,to predict variable B1, we can leave out the other variable from the dataframe and focus on the variables R1-R5,B2-B5 and S2-S5.

B(n) is in relation to S(n) and R(n) as the anount billable for the B2 is actually the amount in R1 which is reflected as the repayment amount from the previous month.

And S(n) refers to the repayment reflected status in nth month, hence we can also conclude that if customer were to make a prompt payment on B2, it would also means that the customer would pay on time, while having a delayed in payment in B2 would results in 0 for R1 as they have not made payment in B2.

From the linear regression that was done above, INCOME and B2 was used separately to predict B1 values, it can be seen that B2 was a better predictor compared to INCOME.

The RMSE for B2 as a predictor is much lower compared to INCOME as a predictor

If we compare using R1, S1 and B2 as predictor, B2 is still a better predictor just by looking at the RMSE calculated.