Logo

Description automatically generated

ANL252

Python for Data Analytics

End-of-Course Assessment

Singapore University of Social Sciences

T05

Nur Jannah Binte Yazit

W1981876

Submitted On: 6 March 2023

**Table of Contents:**

[Question 1 4](#_Toc128964326)

[Output for Qn1 4](#_Toc128964327)

[Analysis for Qn1 6](#_Toc128964328)

[Question 2 7](#_Toc128964329)

[Output for Qn2 7](#_Toc128964330)

[Analysis for Qn2 10](#_Toc128964331)

[Question 3 11](#_Toc128964332)

[Output for Qn3 11](#_Toc128964333)

[Output for Data Preparation 1: Remove Duplicated Data 12](#_Toc128964334)

[Output for Data Preparation 2: Date Format 12](#_Toc128964335)

[Output for Data Preparation 3: 2 Decimal Places for Amount 14](#_Toc128964336)

[Analysis for Qn3 15](#_Toc128964337)

[Question 4 16](#_Toc128964338)

[Output for Qn4 16](#_Toc128964339)

[Output for Insight 1: 19](#_Toc128964340)

[Output for Insight 2: 20](#_Toc128964341)

[Output for Insight 3: 21](#_Toc128964342)

[Analysis for Qn4 22](#_Toc128964343)

[Analysis for Insight 1: 22](#_Toc128964344)

[Analysis for Insight 2: 23](#_Toc128964345)

[Analysis for Insight 3: 23](#_Toc128964346)

[Question 5 24](#_Toc128964347)

[Output for Qn5 24](#_Toc128964348)

[Analysis for Qn5 30](#_Toc128964349)

[Question 6 31](#_Toc128964350)

[Appendix A – Code for Qn1 32](#_Toc128964351)

[Appendix B – Code for Qn2 34](#_Toc128964352)

[Appendix C – Code for Qn3 37](#_Toc128964353)

[Appendix D – Code for Qn4 40](#_Toc128964354)

[Appendix E – Code for Qn5 45](#_Toc128964355)

# Question 1

## Output for Qn1

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

RangeIndex: 24213 entries, 0 to 24212

Data columns (total 12 columns):

# Column Non-Null Count Dtype

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

0 Claim\_ID 24208 non-null float64

1 Policy\_No 24213 non-null int64

2 Name 24213 non-null object

3 Planned 24213 non-null object

4 Actual 22536 non-null object

5 Created 24213 non-null int64

6 Amount 24213 non-null object

7 Paid 24213 non-null object

8 Category 24213 non-null object

9 Terms 24213 non-null object

10 Region 24213 non-null object

11 Type 24213 non-null object

dtypes: float64(1), int64(2), object(9)

memory usage: 2.2+ MB

True

Claim\_ID 5

Policy\_No 0

Name 0

Planned 0

Actual 1677

Created 0

Amount 0

Paid 0

Category 0

Terms 0

Region 0

Type 0

dtype: int64

1682

Table

Description automatically generated with medium confidence

3

Table

Description automatically generated

4

Count of Missing Values: 1689

## Analysis for Qn1

From the output above, the missing variables are *Claim\_ID with 5 missing values*, *Actual with 1677 missing values,* and *Terms with a total of 7 (3 + 4) missing values*. Therefore, there are **1689 missing values** in the data set.

# Question 2

## Output for Qn2

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

RangeIndex: 24213 entries, 0 to 24212

Data columns (total 12 columns):

# Column Non-Null Count Dtype

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

0 Claim\_ID 24208 non-null float64

1 Policy\_No 24213 non-null int64

2 Name 24213 non-null object

3 Planned 24213 non-null object

4 Actual 22536 non-null object

5 Created 24213 non-null int64

6 Amount 24213 non-null object

7 Paid 24213 non-null object

8 Category 24213 non-null object

9 Terms 24213 non-null object

10 Region 24213 non-null object

11 Type 24213 non-null object

dtypes: float64(1), int64(2), object(9)

memory usage: 2.2+ MB

Claim\_ID 5

Policy\_No 0

Name 0

Planned 0

Actual 1677

Created 0

Amount 0

Paid 0

Category 0

Terms 0

Region 0

Type 0

dtype: int64

Yes 22536

No 1677

Name: Paid, dtype: int64

Claim\_ID 5

Policy\_No 0

Name 0

Planned 0

Actual 0

Created 0

Amount 0

Paid 0

Category 0

Terms 0

Region 0

Type 0

dtype: int64

Table

Description automatically generated

Table

Description automatically generated

Claim\_ID 0

Policy\_No 0

Name 0

Planned 0

Actual 0

Created 0

Amount 0

Paid 0

Category 0

Terms 0

Region 0

Type 0

dtype: int64

3

4

0

0

7

## Analysis for Qn2

There are better options to treat the missing values than deleting the row that consists of missing values, as each row stores valuable information like the name of the claimant, date of payment, and amount, to name a few. Therefore, the best option is to replace the missing values. Firstly, I treated the missing values in the Actual variable. I replaced it with “Unpaid” as the number of No in Paid variable matches the number of missing values in the Actual variable of 1677. Hence, it is safe to conclude that the reason for the missing value in the Actual variable is that it is yet to be paid. Secondly, I treated the missing value in the Claim\_ID variable. Since the ID for the claim is missing, I treat it by replacing the missing values with the same number as Policy No. By doing so, creating a set of new IDs is unnecessary. Moreover, the Policy No. can be used to track the Claim\_ID, thereby creating an easy reference. Lastly, for the “???” and “Unkn” in the Terms variable, I treated them by replacing them with a new code, “ZZ00”, to indicate that this is an unknown code.

# Question 3

## Output for Qn3

Yes 22536

No 1677

Name: Paid, dtype: int64

Claim\_ID 0

Policy\_No 0

Name 0

Planned 0

Actual 0

Created 0

Amount 0

Paid 0

Category 0

Terms 0

Region 0

Type 0

dtype: int64

### Output for Data Preparation 1: Remove Duplicated Data

Table

Description automatically generated



### Output for Data Preparation 2: Date Format

Table

Description automatically generated

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

Int64Index: 24210 entries, 0 to 24212

Data columns (total 12 columns):

# Column Non-Null Count Dtype

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

0 Claim\_ID 24210 non-null object

1 Policy\_No 24210 non-null object

2 Name 24210 non-null object

3 Planned 24210 non-null object

4 Actual 24210 non-null object

5 Created 24210 non-null object

6 Amount 24210 non-null object

7 Paid 24210 non-null object

8 Category 24210 non-null object

9 Terms 24210 non-null object

10 Region 24210 non-null object

11 Type 24210 non-null object

dtypes: object(12)

memory usage: 2.4+ MB

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

Int64Index: 24210 entries, 0 to 24212

Data columns (total 12 columns):

# Column Non-Null Count Dtype

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

0 Claim\_ID 24210 non-null object

1 Policy\_No 24210 non-null object

2 Name 24210 non-null object

3 Planned 24210 non-null datetime64[ns]

4 Actual 22533 non-null datetime64[ns]

5 Created 24210 non-null datetime64[ns]

6 Amount 24210 non-null object

7 Paid 24210 non-null object

8 Category 24210 non-null object

9 Terms 24210 non-null object

10 Region 24210 non-null object

11 Type 24210 non-null object

dtypes: datetime64[ns](3), object(9)

memory usage: 2.4+ MB

Table

Description automatically generated

### Output for Data Preparation 3: 2 Decimal Places for Amount

Claim\_ID object

Policy\_No object

Name object

Planned datetime64[ns]

Actual datetime64[ns]

Created datetime64[ns]

Amount float64

Paid object

Category object

Terms object

Region object

Type object

dtype: object

Table

Description automatically generated

## Analysis for Qn3

The first step is to remove any duplicated data to prepare the data for further analysis. Duplicated data could affect any statistical findings. Secondly, it is to set a standardised date format of YYYY-mm-dd for the Planned, Actual and Created variables, ensure that their datatype is the datetime and maintain consistency. Additionally, the datetime data type would assist further data analysis by making it neater to deal with. Lastly, the Amount variable is set to two decimal places (d.p) since it is in monetary terms. In finance, two d.p provide more precision and are more compatible with financial software, which is why two d.p for monetary terms are always widely accepted.

# Question 4

## Output for Qn4

Yes 22536

No 1677

Name: Paid, dtype: int64

Claim\_ID 0

Policy\_No 0

Name 0

Planned 0

Actual 0

Created 0

Amount 0

Paid 0

Category 0

Terms 0

Region 0

Type 0

dtype: int64

Table

Description automatically generated



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

Int64Index: 24210 entries, 0 to 24212

Data columns (total 12 columns):

# Column Non-Null Count Dtype

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

0 Claim\_ID 24210 non-null object

1 Policy\_No 24210 non-null object

2 Name 24210 non-null object

3 Planned 24210 non-null datetime64[ns]

4 Actual 22533 non-null datetime64[ns]

5 Created 24210 non-null datetime64[ns]

6 Amount 24210 non-null object

7 Paid 24210 non-null object

8 Category 24210 non-null object

9 Terms 24210 non-null object

10 Region 24210 non-null object

11 Type 24210 non-null object

dtypes: datetime64[ns](3), object(9)

memory usage: 2.4+ MB

Table

Description automatically generated

Claim\_ID object

Policy\_No object

Name object

Planned datetime64[ns]

Actual datetime64[ns]

Created datetime64[ns]

Amount float64

Paid object

Category object

Terms object

Region object

Type object

dtype: object

Table

Description automatically generated

### Output for Insight 1:

Yes 22533

No 1677

Name: Paid, dtype: int64

Chart, pie chart

Description automatically generated

### Output for Insight 2:

Table

Description automatically generated

Chart

Description automatically generated

### Output for Insight 3:

Table

Description automatically generated

67 rows × 6 columns

Text(0.5, 1.0, 'Terms vs Type Code')

A picture containing chart

Description automatically generated

## Analysis for Qn4

### Analysis for Insight 1:

A **pie chart** can be created to find the percentage of claimants who have made the payment for their insurance claims. Pie charts can identify the most frequent items (the bigger slice) and analyse variables. From the pie chart above, it is apparent that most of the claimants, 93.1%, have made the payment while only a few of them, 6.9%, have yet to, despite having insurance coverage. From this pie chart, it can be assumed that 93.1% of the claimants are aware of their insurance and have experienced any damages or loss to make the claim payment. Additionally, the claimants were well-educated, understood the coverage and knew how to file a claim. It also implies that their insurance company might be a good fit for them. The best way to address the 6.9% of the claimants is to educate them unless they have yet suffered any damages or losses, about the insurance plan and ensure that the insurance provider works harder to strengthen the relationship.

### Analysis for Insight 2:

A **box plot** can be used to analyse the payment amount of the claims. A box plot is suitable for continuous quantitative data. The box plot above shows that most of the payments are less than $20,000, with a max of more than $140,000. It also indicates that the claimants of more than $150,000 are the outliners for the data. Moreover, the box plot also aids in indicating the 25th and 75th percentile of the data.

### Analysis for Insight 3:

A **heatmap** is a suitable chart to analyse the relationship between two variables based on their frequencies. The darker shade indicates most frequencies, while the lighter shade indicates lesser or 0 frequency. A benefit of using a heatmap is that it is easy to identify the trends and outliers in extensive data. From the heatmap above, most insurance plans follow the Terms code AD23 while the Type code is L001. Accordingly, this pairing of codes is the most popular among claimants.

# Question 5

## Output for Qn5

Claim\_ID 0

Policy\_No 0

Name 0

Planned 0

Actual 0

Created 0

Amount 0

Paid 0

Category 0

Terms 0

Region 0

Type 0

dtype: int64



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

Int64Index: 22521 entries, 0 to 24090

Data columns (total 12 columns):

# Column Non-Null Count Dtype

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

0 Claim\_ID 22521 non-null float64

1 Policy\_No 22521 non-null int64

2 Name 22521 non-null object

3 Planned 22521 non-null datetime64[ns]

4 Actual 22521 non-null datetime64[ns]

5 Created 22521 non-null datetime64[ns]

6 Amount 22521 non-null object

7 Paid 22521 non-null object

8 Category 22521 non-null object

9 Terms 22521 non-null object

10 Region 22521 non-null object

11 Type 22521 non-null object

dtypes: datetime64[ns](3), float64(1), int64(1), object(7)

memory usage: 2.2+ MB

Table

Description automatically generated

Chart, scatter chart

Description automatically generated

LinearRegression()

Chart, scatter chart

Description automatically generated

Chart, scatter chart

Description automatically generated

Intercept: 6.759734946166782

Coefficient: [0.57005106]

Coefficient of Determination: 0.3175647332095146

Predicted Response:

[16.4506029 9.60999023 17.02065396 ... 15.31050079 11.3201434

12.46024551]

Chart, scatter chart

Description automatically generated

Table

Description automatically generated

Notes:  
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

y = [0.57005106] x + 6.759734946166782

## Analysis for Qn5

For Linear Regression analysis, missing values data, including “???” and “Unkn”, will be removed. Additionally, as aforementioned, duplicated data will also be removed as this would help to generate a more accurate, consistent, and improved data quality for machine learning. Moreover, as part of data preparation, the Planned, Actual, and Created variables will be formatted into YYYY-mm-dd (datetime) for consistency and better data quality. Next, a scatter plot will be created to find if there is a linear relationship between the Planned and Actual dates for the prediction of delay days before proceeding with the steps to create Linear Regression using python codes. Lastly, scatter plots were created for the trained, tested and predicted data for the linear regression for better visualisation. Python codes can be used to calculate the intercept and coefficient, as shown in Appendix E.

# Question 6

From the output in Qn5 above, the linear regression equation is:

**y = [0.57005106] x + 6.759734946166782**

The Planned and Actual dates formed a positive correlation from the scatter plots created in Qn5 and the linear regression equation above. Thus, for every increase in the x-value, the y-value increases. To illustrate, for every 1-day increase in Planned (x-axis), the predicted value of Actual (y-axis) will increase by 0.57 days (to 2 d.p). Additionally, it can be concluded that the relationship between the Planned and Actual dates formed a strong relationship due to the steepness of the linear regression slope of 6.76 days (to 2 d.p). Applying it to this concept means there will be a delay in day(s) between the Planned and Actual dates. Hence the Actual date for the claim usually happens after the Planned date.

# Appendix A – Code for Qn1

# import the necessary libraries

import pandas as pd

import numpy as np

# read the dataset

df = pd.read\_csv("ECA.csv")

# print the dataset information

df.info()

# read the dataset

df = pd.read\_csv("/Users/nurjannah/Desktop/4.2/ANL252/\_ECA\_LOG353/ECA.csv")

# print the dataset information

df.info()

# to find the total count of the missing values of each variable from the dataset

df.isnull().sum()

# to find the total count of missing values from the dataset

df.isnull().sum().sum()

# print the data that has any cell contain "???" in Terms

df[df["Terms"] == "???"]

# find the total count of "???" values in Terms

(df["Terms"] == "???").sum()

# print the data that has any cell contain "Unkn" in Terms

df[df["Terms"] == "Unkn"]

# find the total count of "Unkn" values in Terms

(df["Terms"] == "Unkn").sum()

# calculate the total no. of missing values

print(f"Count of Missing Values:", (df.isnull().sum().sum())+(df["Terms"] == "???").sum()+(df["Terms"] == "Unkn").sum())

# Appendix B – Code for Qn2

# import the necessary libraries

import pandas as pd

import numpy as np

# read the dataset

df = pd.read\_csv("ECA.csv")

# print the dataset information

df.info()

# to find the total count of the missing values of each variable from the dataset

df.isnull().sum()

# to treat the missing data for the Actual, first need to found out if the number matches with unpaid claims

df["Paid"].value\_counts()

# since the number of missing data in Actual matches with the No in Paid variable of 1677

# note: Actual is an object data type

# hence, to treat the missing data in Actual, replace it with Unclaim since it is yet to be claimed or paid

df["Actual"] = df["Actual"].fillna("Unpaid")

# check again if the missing data in Actual is being treated

df.isnull().sum()

# from the dataset, the missing values in Claim\_ID variables are located at the last 5 row

df.tail()

# the variable Claim\_ID is a float data type

# hence, to treat the missing data in Claim\_ID, replace it with the number in Policy\_No for easy reference

df2 = df.fillna(method="bfill",axis=1)

df2.tail()

# check again if the missing data in Claim\_ID is being treated

df2.isnull().sum()

# lastly, treat the "???" and "Unkn" in Terms variable, by replacing them with a code

# count how many "???" in Terms

(df2["Terms"] == "???").sum()

# count how many "Unkn" in Terms

(df2["Terms"] == "Unkn").sum()

# lets create a new code "ZZ00" to represent unknown term code for the Terms variable

df2\_updated = df2.replace({"Terms": {"???":"ZZ00","Unkn":"ZZ00"}})

# check if issue for "???" in Terms has resolved

(df2\_updated["Terms"] == "???").sum()

# check if issue for "Unkn" in Terms has resolved

(df2\_updated["Terms"] == "Unkn").sum()

# check if the total count for the missing value in Terms variable matches with the findings from Qn1; should have be 7(3 from "???" + 4 from "Unkn")

(df2\_updated["Terms"] == "ZZ00").sum()

# Appendix C – Code for Qn3

# import the necessary libraries

import pandas as pd

import numpy as np

# read the dataset

df = pd.read\_csv("ECA.csv")

# to treat the missing data for the Actual, first need to found out if the number matches with unpaid claims

df["Paid"].value\_counts()

# since the number of missing data in Actual matches with the No in Paid variable of 1677

# the variable Actual is an object data type

# hence, to treat the missing data in Actual, replace it with Unclaim since it is yet to be claimed or paid

df["Actual"] = df["Actual"].fillna("Unpaid")

# the variable Claim\_ID is float data type

# hence, to treat the missing data in Claim\_ID, replace it with the number in Policy\_No

df2 = df.fillna(method="bfill",axis=1)

# check if the missing data in Actual and Claim\_ID are being treated

df2.isnull().sum()

# lastly, treat the "???" and "Unkn" in Terms variable, by replacing them with a code

# lets create a new code "ZZ00" to represent unknown term code for the Terms variable

df2\_updated = df2.replace({"Terms": {"???":"ZZ00","Unkn":"ZZ00"}})

# 1. remove any duplicated data

# print duplicated data

df2\_updated[df2\_updated.duplicated(keep=False)]

# keep the frist duplicated row

df3 = df2\_updated.drop\_duplicates()

# check if there is any duplication again; should be no more duplicated data

df3[df3.duplicated(keep=False)]

# 2. convert the date format for Planned, Actual and Created variables

# first, check the current date format for Planned and Created variables

df3.head()

# secondly, check the data type for Planned, Actual and Created variables

df3.info()

# then, convert the data type for Planned, Actual and Created variables to datetime

# switch off SettingWithCopyWarning in Pandas

pd.options.mode.chained\_assignment = None

df3["Planned"] = pd.to\_datetime(df3["Planned"], format="%d/%m/%Y")

df3["Created"] = pd.to\_datetime(df3["Created"], format="%Y%m%d")

df3["Actual"] = pd.to\_datetime(df3["Actual"], errors='coerce')

# check again the data type for Planned, Actual and Created variables; should be datetime64[ns]

df3.info()

#finally, check if the date format is the same for Planned, Actual and Created variables

df3.head()

# 3. make the Amount variable to 2 decimal places

# firstly, convert the data type for Amount from object to float

df3["Amount"] = pd.to\_numeric(df3["Amount"], errors="coerce")

# print the latest data type for the dataframe

print(df3.dtypes)

# then, around the Amount to 2 decimal places

df3["Amount"] = df3["Amount"].round(decimals = 2)

# print the first 5 sets of data to check if code run correctly

df3.head()

# Appendix D – Code for Qn4

# import the necessary libraries

import pandas as pd

import numpy as np

# read the dataset

df = pd.read\_csv("ECA.csv")

# to treat the missing data for the Actual, first need to found out if the number matches with unpaid claims

df["Paid"].value\_counts()

# since the number of missing data in Actual matches with the No. in Paid variable of 1677

# the variable Actual is an object data type

# hence, to treat the missing data in Actual, replace it with Unclaim since it is yet to be claimed or paid

df["Actual"] = df["Actual"].fillna("Unpaid")

# the variable Claim\_ID is float data type

# hence, to treat the missing data in Claim\_ID, replace it with the number in Policy\_No

df2 = df.fillna(method="bfill",axis=1)

# check if the missing data in Actual and Claim\_ID are being treated

df2.isnull().sum()

# lastly, treat the "???" and "Unkn" in Terms variable, by replacing them with a code

# lets create a new code "ZZ00" to represent unknown term code for the Terms variable

df2\_updated = df2.replace({"Terms": {"???":"ZZ00","Unkn":"ZZ00"}})

# 1. remove any duplicated data

# print duplicated data

df2\_updated[df2\_updated.duplicated(keep=False)]

# keep the frist duplicated row

df3 = df2\_updated.drop\_duplicates()

# check if there is any duplication again; should be no more duplicated data

df3[df3.duplicated(keep=False)]

# 2. convert the date format for Planned, Actual and Created variables into datetime

# switch off SettingWithCopyWarning in Pandas

pd.options.mode.chained\_assignment = None

df3["Planned"] = pd.to\_datetime(df3["Planned"], format="%d/%m/%Y")

df3["Created"] = pd.to\_datetime(df3["Created"], format="%Y%m%d")

df3["Actual"] = pd.to\_datetime(df3["Actual"], errors="coerce")

# check again the data type for Planned, Actual and Created variables; should be datetime64[ns]

df3.info()

#finally, check if the date format is the same for PPlanned, Actual and Created variables

df3.head()

# 3. make the Amount variable to 2 decimal places

# firstly, convert the data type for Amount from object to float

df3["Amount"] = pd.to\_numeric(df3["Amount"], errors="coerce")

# then, around the Amount to 2 decimal places

df3["Amount"] = df3["Amount"].round(decimals = 2)

# print the latest data type for the dataframe

print(df3.dtypes)

# print the first 5 sets of data to check if code run correctly

df3.head()

# Insight 1 (pie chart)

# retrieving data for Paid

paidData = df3[["Paid"]]

# count the frequencies of each variable

df3["Paid"].value\_counts()

# create pie chart for Insight 1

# creating pie chart using Matplotlib

import matplotlib.pyplot as plt

sizes = [1677, 22533]

labels = "No", "Yes"

explode = (0.2,0) #only exploding the "No" slice

plt.pie(sizes,

explode=explode,

labels=labels,

autopct ="%1.1f%%",

colors=["#9f86c0", "#ffb3c6"],

shadow=True)

# title and labels for the box plot

plt.title("No. of Paid Claims",fontsize=15)

plt.axis("equal")

plt.show()

# Insight 2 (box plot)

# retrieving data for Amount and analyse it

amountData = df3[["Amount"]]

amountData.describe()

# create the (horizontal) box plot for Insight 2

# creating boxplot using Seaborne

import seaborn as sns

plt.figure(figsize=(11,7))

sns.boxplot(data=amountData,

x="Amount",

orient="h",

color="#c8b6ff",

medianprops={"color": "#d90429"})

# title and labels for the box plot

plt.title("Amount Claimed",fontsize=15)

plt.xlabel("Amount in $",fontsize=12)

plt.ylabel("Frequencies",fontsize=12)

plt.show()

# Insight 3 (heatmap)

# analysing the relationship between Terms and Type codes

#fill the NaN to 0

df3.groupby("Terms").Type.value\_counts().unstack().fillna(0)

# create the heatmap for Insight 3

# creating heatmap using Seaborne

plt.figure(figsize=(11,7))

codeData = df3.groupby("Terms").Type.value\_counts().unstack().fillna(0)

sns.heatmap(codeData, cmap=sns.cubehelix\_palette(as\_cmap=True), linewidth=0.5)

# title and labels for the heatmap

plt.title("Terms vs Type Code", fontsize=15)

# Appendix E – Code for Qn5

# import the necessary libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

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

# read the dataset

df = pd.read\_csv("ECA.csv", na\_values=['Unkn', '???'])

# treat the missing values in Actual and Claim\_ID by deleting them, as this will affect the Linear Regression

df = df.dropna(axis = 0, how = "any")

# delete any duplicated datas and keep the frist duplicated row

df2 = df.drop\_duplicates()

# check if all missing data being treated

df2.isnull().sum()

# check if there is any duplication again; should be no more duplicated data

df2[df2.duplicated(keep=False)]

# convert the date format for Planned, Actual and Created variables into datetime (as done in Qn3)

# switch off SettingWithCopyWarning in Pandas

pd.options.mode.chained\_assignment = None

df2["Planned"] = pd.to\_datetime(df2["Planned"], format="%d/%m/%Y")

df2["Created"] = pd.to\_datetime(df2["Created"], format="%Y%m%d")

df2["Actual"] = pd.to\_datetime(df2["Actual"], errors="coerce")

# check again the data type for Planned, Actual and Created variables; should be datetime64[ns]

df2.info()

#finally, check if the date format is the same for Planned, Actual and Created variables

df2.head()

# Planned for x-axis and Actual for y-axis

import datetime as dt

x = pd.to\_datetime(df2["Planned"]).dt.day.values.reshape(-1, 1)

y = pd.to\_datetime(df2["Actual"]).dt.day.values

# plot the scatter

df2.plot(kind="scatter",x="Planned",y="Actual",color="#b79ced")

plt.title("Planned vs Actual Scatter Plot", fontsize=15)

plt.xlabel("Planned Dates",fontsize=12)

plt.ylabel("Actual Dates",fontsize=12)

plt.show()

# split dataset for training and set

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

# fit the linear regression into training set

model = LinearRegression().fit(X\_train, y\_train)

model

# predicting Test set results

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

# train set results - chart

plt.scatter(X\_train, y\_train, color ="#ff686b")

plt.plot(X\_train, model.predict(X\_train), color="#001427")

plt.title("Delayed (Train)", fontsize=15)

plt.xlabel("Planned Dates",fontsize=12)

plt.ylabel("Actual Dates",fontsize=12)

plt.show()

# test set results - chart

plt.scatter(X\_test, y\_test, color ="#ff686b")

plt.plot(X\_train, model.predict(X\_train), color="#001427")

plt.title("Delayed (Test)", fontsize=15)

plt.xlabel("Planned Dates",fontsize=12)

plt.ylabel("Actual Dates",fontsize=12)

plt.show()

# fit the model on the planned and actual data

model = LinearRegression().fit(x, y)

# print the intercept coefficient and Coefficient of Determination of the model

print("Intercept: ", model.intercept\_)

print("Coefficient: ", model.coef\_)

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

print(f"Coefficient of Determination: {r\_sq}")

# print the prediction response

y\_pred = model.predict(x)

print(f"Predicted Response:\n{y\_pred}")

# plot Prediction chart

plt.scatter(x, y, color="#669bbc")

plt.plot(x, y\_pred, color="#780000")

plt.title("Prediction", fontsize=15)

plt.xlabel("Planned Dates",fontsize=12)

plt.ylabel("Actual Dates",fontsize=12)

plt.show()

# view regression Results

import statsmodels.api as sm

X\_stat = sm.add\_constant(X\_train)

regsummary = sm.OLS(y\_train, X\_stat).fit()

regsummary.summary()

# print Linear Regression equation

print(f"y =", model.coef\_,"x", "+", model.intercept\_)