|  |
| --- |
| **ANL252**  **Python for Data Analytics** |
| **End-of-Course Assessment**  **January 2023 Presentation** |
| **Student Name: Chia Wan Qi Felicia**  **Student PI Number: Z2210118**  **Submission Date: 5 March 2023** |

Question 1

The variables that contain the missing values is Claim\_ID, Actual and Terms.

As seen from the code shown below.

|  |
| --- |
| ## ECA – Question 1  import numpy as np  import pandas as pd  ## To define the missing values  missing\_values = ["Unkn", "???", ""]  ## Read CSV into pandas  df = pd.read\_csv("/Users/feliciachia/Desktop/ANL252 - Python For Data Analytics/ECA.csv", na\_values = missing\_values)  ## To find the blanks, "Unkn", and "???" in the variables  missings = df.isnull().any()  missings  missings[missings == True].index |

This is a picture of the output from Jupyter Notebook in Appendix 1.



Appendix 1

Question 2

A value of a variable that might be absent from empirical studies is referred to as missing data. Missing data can occur for a variety of reasons, including defective measurement tools and sensitive question refusals. Missing data made it impossible to build models or make predictions and are not desirable in data analytics.

We would need to identify the missing values with the datasets provided.

Firstly, to define the missing values other than blank in the dataset, there are values like ‘Unkn’ and ‘???’. Thus, need to locate the values from the variables.

After which, to read dataset given in a csv file format using pandas with the read\_csv() function.

And to locate the missing data in the dataset using the isnull.().any() and index functions. In which, it would show the variable that have missing values.

With the help of these functions to identify the missing data. We could choose to replace the missing values or to delete the missing values easily. And that we would be able to the spot the variables that have missing values without searching.

Question 3

Data transformation comprises transforming data into a format that is better suited for analysis. Trying to take out any duplicate data, for example. Inaccurate analysis and outcomes can result from duplicate data. In which case, the 'drop.duplicates()' function in pandas can be used to eliminate any duplication on a specified columns.

For example, as shown below is the code in Jupyter Notebook.

|  |
| --- |
| import pandas as pd  ## To load data from csv file  df = pd.read\_csv('/Users/feliciachia/Desktop/ANL252 - Python For Data Analytics/ECA.csv')  ## To drop duplicate rows based on 'Claim\_ID' and 'Name' columns  df.drop\_duplicates(subset=['Claim\_ID', 'Name'], inplace=False) |

Handling missing values. As it is common problem in dataset to have missing values which could lead to incomplete analysis. Where there is many different functions available to handle the missing values in the dataset such as ‘fillna()’, and ‘isnal()’ functions.

In which it could be seen in the table below from Jupyter Notebook.

|  |
| --- |
| import pandas as pd  ## To load data from csv file  df = pd.read\_csv('/Users/feliciachia/Desktop/ANL252 - Python For Data Analytics/ECA.csv')  ## To check for missing values  print(df.isna().sum())  ## To replace any missing values in the DataFrame with a specified value  df.fillna(value=0, inplace=True)  ## To write the updated DataFrame back to a CSV file  df.to\_csv('/Users/feliciachia/Desktop/ANL252 - Python For Data Analytics/ECA\_test.csv', index=False) |

Question 3 (continue)

Another way of handling missing values is to use the ‘dropna()’ function.

In which it could be seen in the table below from Jupyter Notebook.

|  |
| --- |
| import pandas as pd  ## To load data from csv file  df = pd.read\_csv('/Users/feliciachia/Desktop/ANL252 - Python For Data Analytics/ECA.csv')  ## To drop any rows that contain missing values  df.dropna(inplace=True)  ## To write the updated DataFrame back to a CSV file  df.to\_csv('/Users/feliciachia/Desktop/ANL252 - Python For Data Analytics/ECA\_test1.csv', index=False) |

Question 4

1. Using the code as shown below. We could know the number of cases that are paid or not paid for the 2 different regions. As we could tell the status of payment for each region.

|  |
| --- |
| df.pivot\_table(index = 'Region, columns = 'Paid', values = 'Amount', aggfunc = 'count') |

As seen in Appendix 2. It is the output from Jupyter Notebook.

Text

Description automatically generated with medium confidence

Appendix 2

For supporting visualization would be as shown in Figure 1 and 2 as a pie chart. It shows the percentage of claims paid for each region.

Chart, pie chart

Description automatically generatedChart, pie chart

Description automatically generated

Figure 1 Figure 2

Question 4 (continue)

1. From the code shown below. We would be able to know the number of claims was made per person base on their names as well as to know if the amount has been paid to them yet for each claims made.

|  |
| --- |
| df.pivot\_table(index = 'Name', columns = 'Paid', values = 'Claim\_ID', aggfunc = 'count') |

In Appendix 3, we could see the output from Jupyter Notebook. However, there is too many rows. Hence, only the first 5 and last 5 was shown.

Graphical user interface, text, application

Description automatically generated

Appendix 3

For the supporting visualization would be a stacked bar graph. As we would be able to see the number of claims made per person.

Chart, bar chart

Description automatically generated

Figure 3

Question 4 (continue)

1. From the code below, we can know the different type of claims made, and the number of claim types for each category.

|  |
| --- |
| df.pivot\_table(index = 'Type', columns = 'Paid', values = 'Amount', aggfunc = 'count') |

In Appendix 4, we could see the output from Jupyter Notebook. For the NaN, it means there is none.

Graphical user interface, text

Description automatically generated

Appendix 4

For the supporting visualization, it would be as shown in the Figure 4 as a histogram. As we would be able to tell the number of claim type made for each category.

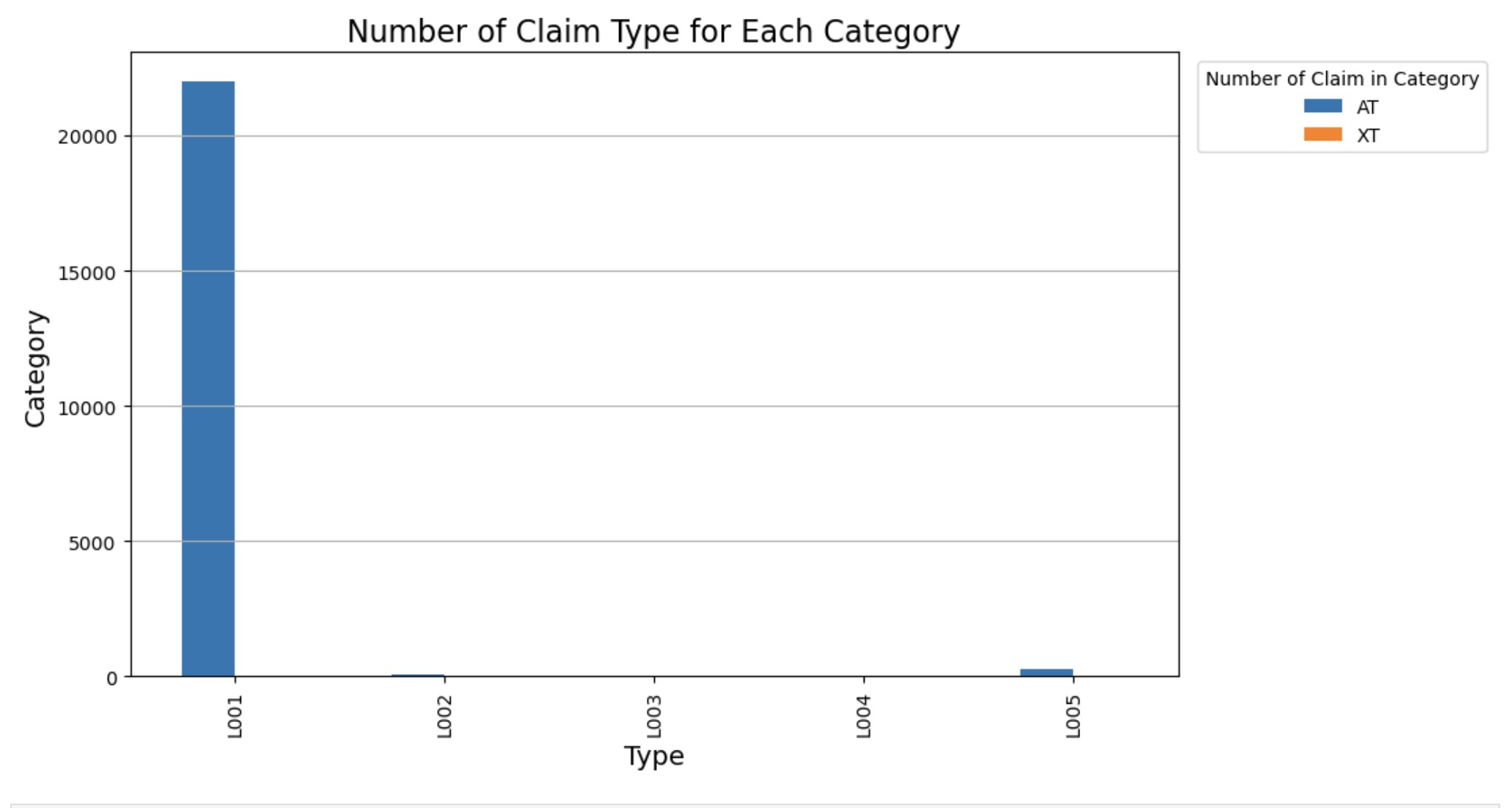


Figure 4

Question 5

To perform linear regression modelling, firstly we would need to import pandas, numpy, matplotlib and sklearn. In which, some of the functions would be used later.

To load the dataset in csv file first. After which, there is a need to identify any missing values in each variable and to remove any of the missing values inside the dataset.

As we are using Planned and Actual variable for the linear regression. The dataset has to be standardized. Hence, we would need to convert the variable to be the same before further data analysis could be done. Thus, to convert the Planned variable first, then to convert Actual variable and to remove the time inside the variable.

After that, need to define the x-axis and y=axis and to choose to fit the model for Planned and Actual data. And to print the intercept and coefficient for the model. Lastly, to plot the scatter graph. As seen below is the code in Jupyter Notebook.

|  |
| --- |
| import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  import sklearn  from sklearn.linear\_model import LinearRegression  ## To load the dataset  df = pd.read\_csv('/Users/feliciachia/Desktop/ANL252 - Python For Data Analytics/ECA.csv')  ## To count the number of missing values in each column  missing\_counts = df.isna().sum()  ## Drop the rows with any missing value  df = df.dropna(axis = 0, how = "any")  ## To convert to datetime of Planned variable  df['Planned'] = pd.to\_datetime(df['Planned'], format='%d/%m/%Y') |

Question 5 (continue)

|  |
| --- |
| ## To convert to yyyy-mm-dd format  df['Planned'] = df['Planned'].dt.strftime('%Y-%m-%d')  ## To convert datetime of Actual variable  df['Actual'] = pd.to\_datetime(df['Actual'], format='%d/%m/%Y %H:%M')  ## To convert to yyyy-mm-dd format and to remove the 0:00  df['Actual'] = df['Actual'].dt.strftime('%Y-%m-%d')  ## To define x-axis and y-axis  planned = pd.to\_datetime(df['Planned']).dt.day.values.reshape(-1, 1)  actual = pd.to\_datetime(df['Actual']).dt.day.values.reshape(-1, 1)  # Fit the model on the planned and actual data  model = LinearRegression()  model.fit(planned, actual)  Y\_pred = model.predict(planned)  # Print the intercept and coefficients of the model  print("Intercept: ", model.intercept\_)  print("Coefficient: ", model.coef\_)  ## To plot the scatter graph  plt.scatter(planned, actual)  plt.plot(planned, Y\_pred, color = 'orange')  plt.title("Days needed for processing claims")  plt.show() |

Question 6

Hence, the output in Jupyter Notebook is as shown in Figure 5 using the code in the table from Question 5.

Chart, scatter chart

Description automatically generated

Figure 5

The intercept value represents the predicted value of the response variable (the 'Actual' variable) when the predictor variable (the 'Planned' variable) is zero.

The coefficient value shows how the response variable's projected value changes when the predictor variable is increased by one unit. In other words, the anticipated real date will increase or drop by the coefficient value for every day that the intended date is raised.

The linear regression equation is Actual = Intercept + Coefficient x Planned

Question 6 (continue)

Hence, for instance, if the intercept is 6.77 and the coefficient is 0.57.

Actual = 6.77 x 0.57 x Planned

Thus, meaning that for every increase of each day in Planned, the delay in days of Actual would be expected to increase by 0.57 days.