

**ANL252**

**PYTHON FOR DATA ANALYTICS**

**End-of-Course Assessment**

**January 2023 Presentation**

**Submitted by:**

|  |  |
| --- | --- |
| **Name** | **PI No.** |
| **Janelle Lim Sock Hui** | **E2181690** |

**Tutorial Group: ­­­­­­­­­­­­ T05**

**Instructor’s Name: Prof. Munish Kumar**

**Submission Date: 05/03/2022**

Table of Contents

[**Question 1** 2](#_Toc128947134)

[**Question 2** 2](#_Toc128947135)

[**Question 3** 4](#_Toc128947136)

[**Question 4** 5](#_Toc128947137)

[**Question 5** 8](#_Toc128947138)

[**Question 6** 9](#_Toc128947139)

[**Appendix** 12](#_Toc128947140)

[**Question 1 Python code** 12](#_Toc128947141)

[**Question 2 Python code** 12](#_Toc128947142)

[**Question 3 Python code** 13](#_Toc128947143)

[**Question 4 Python code** 15](#_Toc128947144)

[**Question 5 Python code** 18](#_Toc128947145)

[**References** 21](#_Toc128947146)

# **Question 1**

The corresponding Python code is in Appendix. To read “Unkn” and “???” as missing values in the dataset as a Pandas dataframe, the following code could be written:

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

To print the number of missing values in each column, the following code is used:

* df.isna().sum(axis=0)

The “Claim\_ID”, “Actual” and “Terms” variables contain missing values.

“Claim\_ID” variable contains 5 missing values.

“Actual” variable contains 1,677 missing values.

“Terms” variable contains 7 missing values.

# **Question 2**

The corresponding Python code is in Appendix. To read “Unkn” and “???” as missing values in the dataset as a Pandas dataframe, the following code could be written:

* df = pd.read\_csv('ECA.csv', na\_values=['Unkn','???'], na\_filter=True)
* The “na\_values” parameter reads “Unkn” and “???” as missing values, while the “na\_filter” parameter is set to “True” to fill all missing values with NaN values.

This also allows format standardisation; all missing values are replaced to NaN values for easier data manipulation using libraries like NumPy and Pandas.

After reading the dataset as a Pandas dataframe, it is crucial to check for missing values in each column to ensure that we have complete data set. We can use the “df.isna.sum()” method to do so. The following code could be written:

* display(df.isna().sum(axis=0))
  + The “df.isna()” method returns “True” if the cell contains a missing value and “False” otherwise. Treating “True” values as 1 and “False” values as 0, the “sum(axis=0)” method then computes the sum of “True” values along the vertical axis of the dataframe.

This would return the number of missing values in each column. In this case, we have 1,689 missing values in the “Actual” column, 5 missing values in the “Claim\_ID” column and 7 missing values in the “Terms” column.

Given that every records in the dataset might be important, it might not be appropriate to remove any rows or columns that contain missing values. This approach may lead to a reduced sample size and the loss of valuable information if the missing values are not missing at random. Furthermore, this approach would lead to the removal of 1,689 rows due to missing values.

The “Claim\_ID” and “Terms” variables contain 12 missing values altogether, possibly caused by incomplete data entry or the failure to gather the necessary information. Despite this missing values, it is important to note that the corresponding rows still hold relevant information that is vital for analysing the corporate claims processing of the insurance company. On the other hand, the missing values under the “Actual” column are a result of entries marked as “No” in the “Paid” column. In such cases, the “Actual” entry cannot be input as there has been no payment made yet for the claim. It is essential to recognise this source of missing data, as it differs from the missing values resulting from incomplete data entry. Understanding the nature and causes of missing values is crucial in accurately interpreting and utilising the available data.

In the case of the “Actual” column, we can remove the rows with missing values as they correspond to claims that have not been paid yet. Removing these rows will give us a more accurate picture of the processed claims of the insurance company. This would lead to the dataset consisting of only “Yes” entries under the “Paid” column, which corresponds to the processed claims. Subsequently, a total of 1,677 rows will be removed from the dataframe.

To drop the rows with missing values under “Actual” column, the “dropna()” method could be used.

* df.dropna(subset=['Actual'], inplace=True)
  + The “dropna()” method drop all rows containing any missing values in the dataframe, while the “subset=['Actual']” parameter specifies the “Actual” column to check for missing values and “inplace=True” parameter updates the dataframe in place instead of returning a copy.

I would also recommend filling the missing values under “Claim\_ID” and “Terms” variables with the string “Unknown”. The “fillna()” method could be used to do so.

* df.fillna({'Claim\_ID': 'Unknown', 'Terms': 'Unknown'}, inplace=True)
  + A dictionary was passed to the “fillna()” method, with the 2 column names as keys and “Unknown” as the value to fill the missing values with. The “inplace=True” parameter updates the dataframe in place instead of returning a copy.

In conclusion, treating missing data is a critical step in data preparation. I handled missing data by treating “Unkn” and “???” as missing values, removing rows with missing values in the “Actual” column, and filling missing values in the “Claim\_ID” and “Terms” columns with the string “Unknown”. These steps ensure a complete dataset for analysis with losing any valuable information.

# **Question 3**

The corresponding Python code is in Appendix. The following are the 3 data preparation tasks required for further analysis of the data.

1. Check for duplicates using the “duplicated()” method. This is to ensure that each row represents a unique claim and remove any unnecessary redundancy in the dataset. This would reduce the chances of errors and improve data accuracy and reliability of data analysis. The following code could be written:
   * df[df.duplicated(keep=False)]
   * The “duplicated()” method checks each row of the dataframe for duplicates, where it returns ‘True” if the row is a duplicate of a previous row and “False” if the row is not a duplicate. The “keep=False” parameter specifies all duplicates to be considered as duplicates, not just the second and subsequent occurrences.
   * “df[]” would then return a subset of the original dataframe “df” that contains all rows that are duplicates.

This gave an output of 6 rows, consisting of 3 duplicates, where all its columns have the same values. To drop the duplicates, “drop\_duplicates()” method could be used. The following code could be written:

* df = df.drop\_duplicates(keep='first')
  + This will drop all duplicate rows in the dataframe and keep only the first occurrence of each unique row. The “keep=‘first’” parameter specifies the first occurrence of each duplicated row to be retained, and all subsequent occurrences to be dropped.

1. Convert date columns to datetime format. This ensures data standardisation and facilitates data manipulation, sorting, time-based analysis, and visualisation. The “Planned”, “Actual”, and “Created” columns contain dates in a string format with different date formats in each column. The “Actual” column contains “0:00” for each row, which is redundant. The “Created” column displays an unclear date format of “yyyymmdd”. Thereby, all three columns could be converted to a datetime object using “pandas.to\_datetime” function, where its default format is “YYYY-MM-DD”. The following code could be written:

* df['Planned'] = pd.to\_datetime(df['Planned'], format='%d/%m/%Y')

df['Actual'] = pd.to\_datetime(df['Actual'], format='%d/%m/%Y %H:%M')

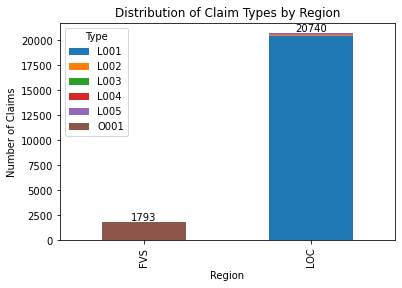
df['Created'] = pd.to\_datetime(df['Created'], format='%Y%m%d')

1. Convert “Amount” column to numeric format. The column represents the amount of money associated with each claim. This would ensure data standardisation and facilitates data manipulation, sorting and mathematical operations such as calculating the sum or average. Any invalid values in the “Amount” column would prompt a “ValueError” upon conversion. Hence, we need to identify any invalid values in the column that cannot be converted to float. In this case, row index 3698 has an invalid value of “1762.OO”. To fix this invalid value, we can replace it with the correct format of “1762.00”, before converting the “Amount” column to float values.

# **Question 4**

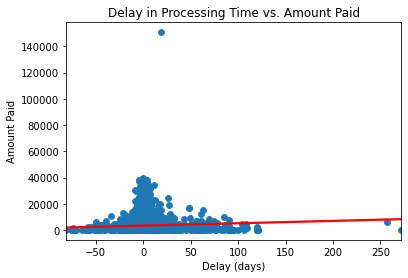
The corresponding Python code is in Appendix. The three insights after data preparation and analysis are as follows.

Firstly, the majority of processed claims fall under the “LOC” region, while the “FVS” region has a smaller number of claims. Specifically, there are 20,740 claims under the “LOC” region and 1,793 claims under the “FVS” region. This indicates that the company may have a larger customer base in the “LOC” region and a smaller customer base in the “FVS” region. Furthermore, the majority of the claims in the “LOC” region fall under the “L001” type. A stacked bar chart of distribution of claim types by region is used to illustrate this (Refer to Figure 1).

 **Figure 1: Distribution of Claim Types by Region**

This insight could help the company allocate more resources to regions with higher claim counts to ensure efficient claims processing. Additionally, the higher number of claims in a region could also indicate a higher risk to accidents or other types of incidents, which could impact the insurance company’s risk management strategies. Hence, the company could analyse the types of claims that are most common in each region to better understand the customers’ needs in each area.

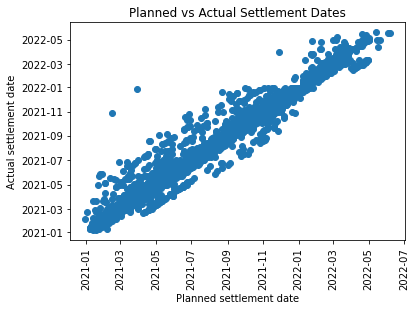
Secondly, most claims are processed on time or within a few days of their planned settlement date. A scatter plot of the delay in claim processing time against the amount paid out is created to illustrate this (Refer to Figure 2). Furthermore, the points on the scatter plot are more concentrated around the centre approximately on the ‘0’ data point. This also suggest that the amount of money paid out may not have a significant impact on the delay in processing time. The regression line of the scatter plot also showed an increasing yet slightly flat slope, suggesting a positive yet weak correlation between the amount of money paid out and the delay in processing time. The claims that took longer to process might tend to have higher pay-outs, however this is based on a minimal correlation between the two variables.



**Figure 2: Delay in Process Time vs. Amount Paid**

There are 3 outliers evident in the scatter plot which corresponds to these claims – claim amounts of “$150723.864”, “$6267.117” and “$188.400”, which took 19 days, 257 days, and 272 days respectively to process. The company could use this insight to allocate resources accordingly. Additionally, the company could investigate whether there are specific reasons for the correlation and the outliers, such as more complex claims requiring more time to process.

Lastly, the scatter plot of the planned settlement date against actual settlement date was illustrated, showing a positive correlation between the two variables, with most data points forming a roughly 45-degree angle indicating direct proportionality (Refer to Figure 3). This shows that for a given planned settlement date, the actual settlement date tends to be relatively close.



**Figure 3: Planned vs Actual Settlement Dates**

However, there is considerable variability in settlement times, with some claims settling earlier and others settling much later. Notably, three data points stand out as outliers with unusually long delays. These claims had planned settlement dates of “2021-02-14”, “2021-03-30” and “2021-11-29”, but actual settlement dates of “2021-10-29”, “2021-12-27” and “2022-03-30” respectively, resulting in delay days of 257, 272 and 121 days. Two of the outliers are identical to part of what was found from the insight in Figure 2. These outliers suggest that the settlement delay could be due to several factors, such as the complexity of the claim or the availability of information needed to process the claim. The company could use this insight to identify areas for improvement in its claims processing efficiency. Despite the variability in settlement times, the scatter plot’s overall trend indicates that the company is generally meeting its planned settlement dates.

In summary, these insights provide a better understanding of the corporate claims processing of the insurance company and can be used to identify areas for improvement to increase efficiency, customer satisfaction, and overall business success.

# **Question 5**

The corresponding Python code is in Appendix. The first step is to create a new column in the dataframe called “Delay” that calculates the delay in days between the planned and actual date of processing the claims. This is done by subtracting the “Planned” column from the “Actual” column and converting the resulting timedelta object to the number of days.

The next step is to clip the “Delay” column to a minimum of zero, such that any negative values in the column will be set the zero. The rationale behind this is to remove the advanced days in the claims processing, due to the actual claim date being way earlier than planned. With the negative delay days, it will only cause variability and inaccuracy in the linear regression model. Therefore, they are set to 0, to show that there are no delays for the corresponding claims.

The “X” variables represent the independent variables that will be used to predict “y” variable, which is the “Delay” column. The independent variables used are “Amount”, “Category”, “Type”, and “Policy\_No” variables. I chose these independent variables as upon trying different models, this model gave the lowest mean squared error.

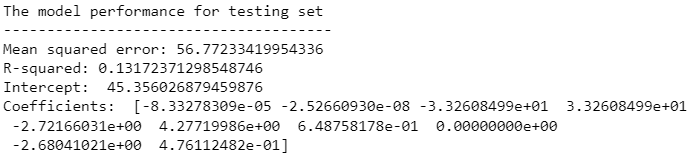
The categorical variables “Category” and “Type” are one-hot coded using the “pd.get\_dummies()” method, which creates new binary columns for each category and type and replaces the original columns in the “X” dataframe.

The “train\_test\_split()” function is used to split the data into training and testing sets. The training set will be used to fit the model, while the testing set will be used to evaluate its performance. The split is done with a ratio of 0.75 to 0.25, with the random state set to 0 for reproducibility.

A linear regression model is first created using “LinearRegression()” function, before fitting the model to the training data using “fit()” method. The model is then used to make predictions on the testing data using the “predict()” method. The model performance is evaluated using the mean squared error and R-squared. The intercept and coefficients are also calculated to provide a better understanding of the linear regression model.

# **Question 6**

Based on the model's results, it appears that its accuracy in predicting the delay in days between planned and actual processing dates of claims may be questionable. This is evident from the high mean squared error (MSE) of 56.772 (Refer to Figure 4). The MSE represents the average of the squared differences between predicted and actual values, which is significantly high and explains the model's inaccuracy in predicting delays. The R-squared value of 0.1317 indicates that only 13.17% of the variance in the delay can be explained by the independent variables included in the model. This suggests that there may be other important factors that are not captured by the model or that the independent variables chosen are not strongly related to the outcome variable.



**Figure 4: Output for Model Performance for Testing Set**

The linear regression equation based on this model would be:

Delay = 45.36 – (8.33278309\*10-5) \* (Amount) – (2.52660930\*10-8) \* (Policy\_No) – 33.2608499 \* (Category\_AT) + 33.2608499 \* (Category\_XT) – 2.72166031 \* (Type\_L001) + 4.27719986 \* (Type\_L002) + 0.648758178 \* (Type\_L003) – 2.68041021\* (Type\_L005) + 0.476112482 \* (Type\_O001)

The intercept of 45.36 suggests that when all independent variables are equal to zero, the predicted delay would be 45.36 days. The coefficients for the independent variables show how much each variable is associated with the predicted delay.

Upon examining the coefficients, the “Amount” variable has a small negative coefficient, suggesting that an increase in claim amount may lead to a decrease in the delay, but the effect is likely to be negligible. The “Policy\_No” and the one-hot encoded categorical variables have very small coefficients that may not be meaningful in practice. The category of “AT” has a large negative coefficient, while claims in the “XT” category have a large positive coefficient, indicating that claims in the “AT” category are processed faster than those in the “XT” category. The type of claim also appears to have a significant effect, with types “L001” and “L002” having negative coefficients suggesting they are processed faster, while types “L003”, “L005”, and "O001" have positive coefficients suggesting they are processed more slowly.

However, it is important to note that this model has limited predictive power, as indicated by the low R-squared and high MSE values. Therefore, the predictions made by this model should be interpreted with caution and further investigation may be necessary to identify other important predictors of delay in claim processing.

# **Appendix**

## **Question 1 Python code**

|  |
| --- |
| # to import libraries  import pandas as pd  import numpy as np  # to read the dataset  # treating "Unkn" and "???" as missing values  # and replace missing values with NaN values  df = pd.read\_csv('ECA.csv', na\_values=['Unkn','???'], na\_filter=True)  # print the number of missing values in each column  display(df.isna().sum(axis=0)) |

* Output:

Claim\_ID 5

Policy\_No 0

Name 0

Planned 0

Actual 1677

Created 0

Amount 0

Paid 0

Category 0

Terms 7

Region 0

Type 0

dtype: int64

## **Question 2 Python code**

|  |
| --- |
| # to import libraries  import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  import seaborn as sns  # to read the dataset  # treating "Unkn" and "???" as missing values  # and replace missing values with NaN values  df = pd.read\_csv('ECA.csv', na\_values=['Unkn','???'], na\_filter=True)  # print the number of missing values in each column  display(df.isna().sum(axis=0))  # drop rows with missing values under 'Actual' column  df.dropna(subset=['Actual'], inplace=True)  # fill remaining missing values with the string "Unknown"  df.fillna({'Claim\_ID': 'Unknown', 'Terms': 'Unknown'}, inplace=True)  # display the new dataframe  display(df) |

## **Question 3 Python code**

(Continuation from Question 2 Python code)

|  |
| --- |
| # to identify duplicates  df[df.duplicated(keep=False)]  # drop duplicates  df = df.drop\_duplicates(keep='first')  # convert 'Planned', 'Actual', and 'Created' columns to a datetime object  df['Planned'] = pd.to\_datetime(df['Planned'], format='%d/%m/%Y')  df['Actual'] = pd.to\_datetime(df['Actual'], format='%d/%m/%Y %H:%M')  df['Created'] = pd.to\_datetime(df['Created'], format='%Y%m%d')  # convert 'Amount' column to numeric and check for invalid values  # set invalid\_values as True for any row where 'Amount' cannot be converted to float  invalid\_values = pd.to\_numeric(df['Amount'], errors='coerce').isna()  if invalid\_values.any():  print(f"The following values in 'Amount' cannot be converted to float: {df['Amount'][invalid\_values]}")  else:  print("All values in 'Amount' are valid") |

* Output:

The following values in 'Amount' cannot be converted to float: 3698 1762.OO

Name: Amount, dtype: object

(Continuation of Question 3 Python code)

|  |
| --- |
| # replace invalid value at row index 3698 with corrected format  df.at[3698, 'Amount'] = '1762.00'  # convert 'Amount' column to float values  df['Amount'] = pd.to\_numeric(df['Amount'])  display(df) |

## **Question 4 Python code**

(Continuation from Question 3 Python code)

|  |
| --- |
| ## chart 1  # group data by region and type  # and get count of claims  region\_type\_counts = df.groupby(['Region', 'Type']).size().reset\_index(name='Counts')  # pivot data to create a table with 'Region' as rows and 'Type' as columns  pivoted\_counts = region\_type\_counts.pivot(index='Region', columns='Type', values='Counts')  # create stacked bar chart  pivoted\_counts.plot(kind='bar', stacked=True)  # set chart title and axis labels  plt.title("Distribution of Claim Types by Region")  plt.xlabel("Region")  plt.ylabel("Number of Claims")  # calculate total counts for each region  total\_counts = df.groupby('Region').size()  # add text labels for the total counts of each region  for i, v in enumerate(total\_counts.values):  plt.text(i, v, str(v), ha='center', va='bottom')  plt.show()  ## chart 2  # creates new 'Delay' column in dataframe  # converting resulting timedelta object to number of days  df['Delay']= (df['Actual'] - df['Planned']).dt.days  # create scatter plot with 'Delay' as x-axis and 'Amount' as y-axis  sns.scatterplot(x='Delay', y='Amount', data=df)  # add regression line  sns.regplot(x='Delay', y='Amount', data=df, ci=None, line\_kws={'color': 'red'})  # set chart title and axis labels  plt.title("Delay in Processing Time vs. Amount Paid")  plt.xlabel("Delay (days)")  plt.ylabel("Amount Paid")  plt.show()  ## chart 3  # create scatter plot with 'Planned' as x-axis and 'Actual' as y-axis  plt.scatter(df['Planned'], df['Actual'])  # set chart title and axis labels  plt.title("Planned vs Actual Settlement Dates")  plt.xlabel("Planned settlement date")  plt.ylabel("Actual settlement date")  # rotate x-axis tick labels  plt.xticks(rotation=90)  plt.show() |

## **Question 5 Python code**

(Continuation from Question 4 Python code)

|  |
| --- |
| # to import libraries  from sklearn import preprocessing  from sklearn.linear\_model import LinearRegression  from sklearn.model\_selection import train\_test\_split  from sklearn.metrics import mean\_squared\_error, r2\_score  # creates new 'Delay' column in dataframe  # converting resulting timedelta object to number of days  df['Delay']= (df['Actual'] - df['Planned']).dt.days  # set negative delays to zero  df['Delay'] = df['Delay'].clip(lower=0)  # select X and y variables  X = df[['Amount','Category','Type','Policy\_No']]  y = df['Delay']  # one-hot encode categorical variables  X = pd.get\_dummies(X, columns=['Category','Type'])  # splitting of training and test data 0.75 to 0.25  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state=0)  # create linear regression model  model = LinearRegression()  # fit model to training data  model.fit(X\_train, y\_train)  # make predictions on testing data  y\_pred = model.predict(X\_test)  # evaluate model for testing set  mse = mean\_squared\_error(y\_test, y\_pred)  r2 = r2\_score(y\_test, y\_pred)  print("The model performance for testing set")  print("--------------------------------------")  print('Mean squared error: {}'.format(mse))  print('R-squared: {}'.format(r2))  # intercept of regression model  print('Intercept: ', model.intercept\_)  # coefficient of regression model  print('Coefficients: ', model.coef\_) |

* Output:

The model performance for testing set

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

Mean squared error: 56.77233419954336

R-squared: 0.13172371298548746

Intercept: 45.356026879459876

Coefficients: [-8.33278309e-05 -2.52660930e-08 -3.32608499e+01 3.32608499e+01

-2.72166031e+00 4.27719986e+00 6.48758178e-01 0.00000000e+00

-2.68041021e+00 4.76112482e-01]

# **References**

Wu, K. Y. (2022). ANL252 Python for data analytics (study guide). Singapore University of Social Sciences.