

**ANL252**

**Python for Data Analytics**

**End-of-Course Assessment**

**January 2023 Presentation**

**Submitted by:**

|  |  |
| --- | --- |
| **Name** | **PI No.** |
| **Teo Meng Hee, Tiffany** | **N2082088** |

**Tutorial Group: ­­­­­­­­­­­­ T 05**

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

**Submission Date: 06/03/2023**

**Question 1**

Based on the corresponding output from the python codes, the variables that contain missing values are “Claim\_ID”, “Actual” and “Terms”, with 5, 1677 and 7 missing values, respectively.

**Question 2**

Some possibilities to treat missing date during data preparation include removing them or replacing it with another value. Overall, the proportion of missing data is 6.976% of the 24213 claims in the dataset. Although this proportion is not significant, one shall evaluate whether the missing values instead of removing all of it. The missing values under “Terms” and “Claim\_ID” can be removed as will be challenging to replace it with a value and irrational to replace with mean, mode or median values of the columns.

However, the missing values under “Actual” has a more meaning and a possibility of replacing it. Missing “Actual” suggests that the claims are not being settled by the claimant, thus the lack of date input. To further prove this assumption, these missing values have payment status of “No” under “Paid” column. These missing values can be meaningful for subsequently insight analysis. As such, I have decided to replace the missing blank values to “Unsettled”. Therefore, at this point, 12 rows are deleted from the dataframe, leaving 24201 rows and 12 columns as its dimensions.

**Question 3**

**Data preparation 1: Remove duplicates within the dataset.**

Duplicates can taint data accuracy and quality because it causes estimation biases in the dataset’s parameters. Consequently, on insights, if there are not handled via elimination prior to data analysis. Python has identified 3 cases of duplicated claim data. Specifically, ID 2928532814, 2930189310 and 2930370663. Using the codes in appendix, another 3 rows are removed from the dataset. Now, there are 24198 rows.

**Data preparation 2: Remove outliers in ‘Amount’ column.**

Biasedness in statistical estimation could arise from the existence of outliers in the dataset. Consequently, it can negatively impact the accuracy of data visualisation and insights. Thus, one should identify and eliminate outliers within the dataset. Calculating and using interquartile range, outliers were detected under the ‘Amount’ column. Specifically, pay-out amount greater than 11220 and less than -5892. A total of 1242 outliers were removed, leaving 22956 rows of dataset. Moving forward, biasedness is reduced, allowing for more accurate analysis from the visualisation and insights gained.

**Data preparation 3: New column of the time difference between Actual and Planned claim settlement dates.**

A potential useful variable that is not in the original dataframe is the time difference between the ‘Actual’ date of claim settlement and ‘Planned’ date of claim settlement. This new variable can help with visualising and potentially gaining insights like factors for delays in settlement process or characteristics of delays. To perform this data preparation, I firstly, created a second dataframe that exclude the “Unsettled” rows under “Actual”. This is because these rows do not have input dates, thus will be redundant for case involving dates. It will be more useful for “Paid” status analysis, which we can use the first cleaned up dataframe. Secondly, the 2 columns ‘Actual’ and ‘Planned’ needs to be converted to datatime format by applying pandas.to\_datetime. This step will aid the calculation of date difference. Finally, calculate the time difference, in terms of days, by “Actual” minus “Planned for a new column “Delay\_days”. Positive values under that column suggests that there was a delay in claimant’s claim settlement. Negative values of days suggests that claimant was settled their claim in advanced of their planned claim settlement date.

**Question 4**

**Insight 1: Majority of payout amount is below $4000.**

From insurance company point of view, this visualisation can give the company an understanding of the distribution of claimants’ amount to gain insight on its severity of claims, which is the amount to be paid out to the claimants. Thereafter, develop and implement strategies to better manage their reserves to cover payout amount and reduce financial risk. Since ‘Amount’ is a continuous variable, a histogram can be used to visualise the amount distribution across the 22956 claims in the cleaned and prepped dataset. The “Amount” column is selected as the plotting values.

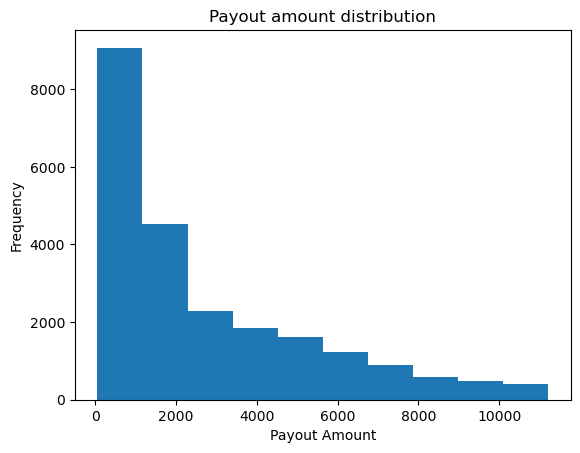


Figure 4.1 Histogram on Payout Amount Distribution

Figure 4.1 shows a right-skewed distribution of claimants’ payout amount where the distribution peaks on the lower end of the amount distribution. Looking at the heights of the histogram bars, one could infer that majority of claimants’ insurance amount is between $0 to $4000, with many being below $2000.

**Insight 2: Positive correlation between ‘Delay\_days’ and ‘Amount’**

Next, I wanted to investigate whether the claim amount has got any impact on the timeliness of claims being settled. Subsequently, the insurance company could gain insights on possible factors for late claim settlements and work on improving its claim settlement punctuality. Previously, during the data preparation step, a new column “Delay\_days” that will suggest the specific number of days regarding whether the claims are settled ahead or over the planned date. This shall be used, along with the respective ‘Amount’ column. As such, scatterplot is appropriate to find out the relationship between the 2 variables.

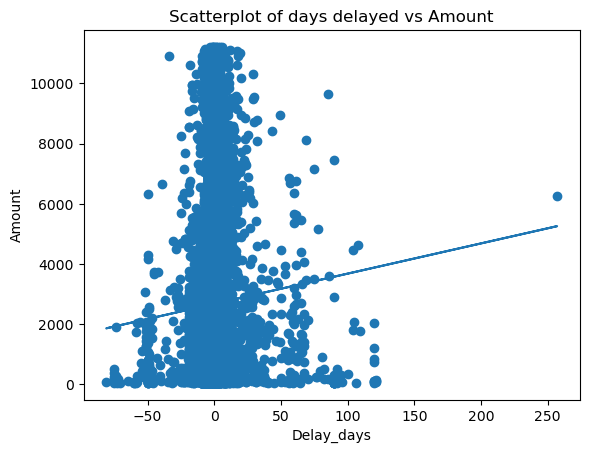


Figure 4.2 Scatterplot on ‘Delay\_days’ and ‘Amount’

Figure 4.2 shows many plots being scattered between ‘-50’ and ‘50’ days difference between ‘Actual’ and ‘Planned’ settlement dates. Some interesting plots include the one on the far right of around $6000 in payout amount, settled over 250 days after the claim’s planned settlement date. To better understand and estimate the data pattern, a trendline has been plotted to annotate the line of best fit and gain insight on the general trend of the dataset used. The line is a straight upwards line, suggesting a positive linear relationship between the 2 variables. Positive correlation suggests that the higher the payout amount, the higher the tendency of a greater delay days for claim settlement.

**Insight 3: All Type “L004” claims paid while “L003” type have the most proportion of unpaid claims.**

Finally, I wanted to investigate the characteristic of claims that are not yet paid. To do so, I have chosen the characteristic of internal type classification code to be assessed with the payment status. I used 100 percent stacked bar charts to highlight the proportions of payment status on the various Type of claims. This would allow better comparison. There are 6 types of claim classification code, each stacked with 2 different colours: blue representing unpaid status, and orange representing paid status.

Chart, bar chart

Description automatically generated

Figure 4.3 100% Stacked bar chart on ‘Paid’ and ‘Type’

Figure 4.3 shows that type “L004” claims are all fully paid while type “L003” has the highest proportion of unpaid claims. Based on the print of the chart’s data frame., the proportion of ‘No’ and ‘Yes’ for “L003” type is 28.57% and 71.43%, respectively.

With this insight, the insurance company can investigate the reason for a greater proportion of “L003” claim type not being paid. As well, see whether there is a better claiming process for “L004” type that other type’s processes can follow.

**Question 5**

To perform linear regression (LR) modelling in Python, the first step is to install additional model functions relating to scikit-learn, besides the standard required packages like pandas as pd. Some relevant modules and functions from scikit-learn applicable for linear regression analytical task includes importing LinearRegression, pre-processing, metrics, train\_test\_split and mean\_squared\_error.

**Pre-processing steps:**

Before carrying out the analytics algorithm, data preparation and pre-processing steps are required. Foremost, missing data, which has already been handle in question 2. The next step taken was to create dataframe for LR modelling with selected columns. The y dependent variable is “Delay\_days” which is the difference between “Planned” and “Actual”, calculated previously in earlier questions. For independent, x-variables, I have selected “Amount” and “Region”.

Secondly, since “Region” is a categorical variable, it needs to be converted into dummy variables for it to be evaluated and included in scikit-learn algorithm computation. Dummy variables are binary with 2 values, 0 and 1.

Thirdly, “Amount” has a larger value range compared to “Delay\_days” with smaller range values. This means that “Amount” can outweigh “Delay\_days” variables that will cause inaccurate modelling results Hence, it is necessary to perform the data transformation method of normalisation to scale down “Amount” to match “Delay\_days”. After identifying these variables to normalise and normalising them, I concatenated the normalised variables with the initial dataframe with added suffices so the labels will not be duplicated or overwritten.

Finally, splitting the modelling dataframe for training and testing. This can help to measure the predictive model’s performance through determining whether the model can predict unseen data accurately. X will be the normalised “Amount” and the 2 dummy variables “Region\_FVS” and “Region\_LOC”. Y is the normalised “Delay\_days”.

**Linear Regression Modelling:**

After the data have been preprocessed and prepared, we can now perform the linear regression modelling with “LinearRegression()”. Additional codes are added to get outputs for attributes like coefficient and intercept for linear regression equation. As well, the regression score (R-square) which measures how strong the relationship is between predicted and predictor variables in the modelling. Furthermore, calculating the root mean square error (RMSE) is the square root of mean square errors between actual and predicted values. RMSE can help to evaluate whether the linear regression model’s performance is accurate. A lower and closer the RMSE value to 0 suggests a more accurate model.

**Question 6**

**Linear regression equation:**

Y = mx + c

* m referring to x-variable’s coefficient
* c referring to intercept

Y (Delay\_days\_norm) = 0.5257 – 0.5235 X1 (Amount\_norm) + 0.0054 X2 (Region\_FVS) – 0.0054 X3 (Region\_LOC)

**Other modelling result:**

Regression score resulted to 0.09037.

RSME is 0.0402.

**Discussion on results:**

The coefficients for X1 and X3 are negative, suggesting that “Delay\_days\_norm” decreases as “Amount\_norm” and “Region\_LOC” increases. On the other hand, X2’s coefficient is positive, meaning that “Delay\_days\_norm” increases as “Region\_FVS” increases. Since this is a multiple linear regression, the value of the coefficient suggests how much the y-variable will be expected to increase when the respective independent variable increases. Means that “Amount\_norm” holds a much significant proportion and impact on “Delay\_days\_norm” compared to the Region-related variables.

The regression score suggests that the predictor variables of “Amount” and “Region” only explains about 9% of the variance in the dependent variable is explained by the independent variables. The RSME value is small, indicating that the model has suggested good and accurate predictors with small residuals.

**Appendices**

**Appendix 1: Python code in text format for question 1**

# installing libraries for Q1 to Q4

import numpy as np

import pandas as pd

import sqlite3

import matplotlib as mpl

import matplotlib.pyplot as plt

# QUESTION 1: Locating Missing Values

# using pandas to read dataset

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

# review the dimensions of the dataset's array

print(df.shape)

# identifying missing data

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

# locating missing data. 1 = True, 0 = False. Value is sum of number of rows missing data

any\_missing = df.isnull().sum(axis=0/1)

print(any\_missing)

# identify columns with missing data based on missing data location

missing\_col = df.columns[df.isna().any()].tolist()

print(missing\_col)

# select & show rows with missing data in the df

missing\_rows = df.isnull().any(axis=1)

df.loc[missing\_rows[missing\_rows == True].index]

**Appendix 2: Python codes in text format for question 2**

# QUESTION 2: Dealing with Missing Values

# delete rows with missing values under "Claim\_ID" & "Terms"

df\_del\_miss = df.dropna(subset = ["Claim\_ID", "Terms"], axis = 0, how = "any")

# check deletion has been done

print(df\_del\_miss.isnull().sum())

print(df\_del\_miss)

# check that "Actual" missing data means unpaid claims

missing\_rows = df\_del\_miss[["Actual"]].isnull().any(axis=1)

df\_del\_miss.loc[missing\_rows[missing\_rows == True].index]

# replace missing data under "Actual" with 'Unsettled'

values = {"Actual": "Unsettled"}

df\_rep\_miss = df\_del\_miss.fillna(value = values)

print(df\_rep\_miss)

# check that missing values have been resolved

print(df\_rep\_miss.isnull().sum())

**Appendix 3: Python codes in text format for question 3**

# QUESTION 3: DATA PREPARATION

# DATA PREPARATION 1: Removing duplicates

df\_rep\_miss.drop\_duplicates(inplace=True)

# check duplicates

print(df\_rep\_miss.shape)

# DATA PREPARATION 2: Identifying & removing outliers in 'Amount'

# remove impurities of non-numeric values in numeric variable column

df\_rep\_miss["Amount"] = pd.to\_numeric(df\_rep\_miss.Amount.astype(str).str.replace('OO', '00'), errors = 'coerce').fillna(0).astype(int)

# check type is consistent

print(df\_rep\_miss["Amount"])

# computing criteria to detect outliers in payout 'Amount' variable

q1 = df\_rep\_miss["Amount"].quantile(q = 0.25)

q3 = df\_rep\_miss["Amount"].quantile(q = 0.75)

iqr = q3-q1

low\_bound = q1 - 1.5\*iqr

upp\_bound = q3 + 1.5\*iqr

print(f"q1: {q1}\nq3: {q3}\ninterquartile range: {iqr}\nlower bound: {low\_bound}\nupper bound: {upp\_bound}")

# removing outliers in 'Amount' column

df\_no\_outliers = df\_rep\_miss[~((df\_rep\_miss["Amount"]< low\_bound) | (df\_rep\_miss["Amount"]> upp\_bound))]

print(df\_no\_outliers)

# DATA PREPARATION 3: Create new column of "Delay\_days"

# new df for this data prep

# remove "Unsettled" rows as will not be relevant for 'Delay' calculation

df\_2 = df\_no\_outliers.loc[df\_no\_outliers["Actual"] != "Unsettled"]

# convert 'Planned' column into datetime format

df\_2["Planned"] = pd.to\_datetime(df\_2["Planned"], format='%d/%m/%Y')

# convert 'Actual' column into datetime format

df\_2["Actual"] = pd.to\_datetime(df\_2["Actual"], format='%d/%m/%Y %H:%M')

# calculate difference between 'Actual' and 'Planned' dates

df\_2["Delay\_days"] = (df\_2["Actual"] - df\_2["Planned"]).dt.days

print(df\_2)

**Appendix 4: Python codes in text format for question 4**

# QUESTION 4: Visualisation for Insights

# VISUALISATION 4.1: Histogram on payout amount distribution

## select 'Amount' pay column in the dataset

amount = df\_no\_outliers["Amount"]

print(amount)

## plot histogram 'Amount' with x-axis values, histtype, bin ranges

plt.hist(amount, histtype = 'bar', bins=10)

## apply plot options for chart’s title, and axes labels

plt.title("Payout Amount Histogram Distribution")

plt.xlabel("Payout Amount")

plt.ylabel("Frequency")

# VISUALISATION 4.2: Scatterplot on "Delay\_days" and "Amount"

# selecting columns for chart 2's dataframe

chart2\_df = df\_2[["Delay\_days", "Amount"]]

print(chart2\_df)

# plot scatterplot with defined x and y values

x = df\_2["Delay\_days"]

y = df\_2["Amount"]

plt.scatter(x, y)

# add trendline in scatterplot

z = np.polyfit(x, y, 1)

p = np.poly1d(z)

plt.plot(x, p(x))

# apply plot options to add in chart title, and axes labels

plt.title("Scatterplot of Delay\_days vs Amount")

plt.xlabel("Delay\_days")

plt.ylabel("Amount")

# Visualisation 4.3: 100% stacked bar chart on "Type" and "Paid"

# dataframe and parameters for 100% stacked bar

df\_chart3 = (df\_no\_outliers

.groupby("Type")["Paid"]

.value\_counts(normalize=True)

.mul(100)

.round(2)

.unstack())

df\_chart3.plot(kind= 'bar', stacked = True)

# apply plot options to add chart title, and axes labels

plt.title("100% stacked of 'Paid' & 'Type' unpaid claims")

plt.xlabel("Type")

plt.ylabel("Frequency of Payment Status")

# print df to see the data statistics for the visualisation

print(df\_chart3)

**Appendix 5: Python codes in text format for question 5 & 6 for Linear Regression modelling**

# QUESTION 5: Further pre-processing & Linear Regression Modelling

# import model functions from scikit-learn

import numpy as np

import pandas as pd

import sklearn

from sklearn.linear\_model import LinearRegression

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

from sklearn import metrics

from sklearn import preprocessing

from sklearn.metrics import mean\_squared\_error

# create dataframe for modelling

X\_var = ["Amount", "Region"]

y\_var = ["Delay\_days"]

df\_model = df\_2[X\_var + y\_var]

print(df\_model)

# creating dummy variables for categorical variables "Category" & "Region"

df\_2\_wDummy = pd.get\_dummies(df\_model)

print(df\_2\_wDummy)

# Data transformation: Normalisation

## selecting variables to normalised

normvar\_list = ["Amount", "Delay\_days"]

df\_model\_toNorm = df\_2\_wDummy[normvar\_list]

df\_model\_toNorm\_col = list(df\_model\_toNorm.columns.values)

df\_model\_toNorm\_row = list(df\_model\_toNorm.index)

print(df\_model\_toNorm)

## normalising the numeric variables

df\_model\_normArray = preprocessing.normalize(df\_model\_toNorm)

df\_model\_scaled = pd.DataFrame(df\_model\_normArray, columns = df\_model\_toNorm\_col, index = df\_model\_toNorm\_row)

print(df\_model\_scaled)

## concatenate normalised variables with initial dataframe

df\_model\_scaled = df\_model\_scaled.add\_suffix("\_norm")

numnormvar\_list = df\_model\_scaled.columns.values

df\_model\_final = pd.concat([df\_2\_wDummy, df\_model\_scaled], axis = 1)

print(df\_model\_final)

# Training & Testing data

x = df\_model\_final[["Amount\_norm", "Region\_FVS", "Region\_LOC"]]

y = df\_model\_final["Delay\_days\_norm"]

## splitting training & testing dataframes

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = 0.25, random\_state = 42)

# Performing Linear Regression modelling and results

reg = LinearRegression()

reg.fit(x\_train, y\_train)

y\_pred = reg.predict(x\_test)

df\_predict = pd.DataFrame({'Actual':y\_test,'Predicted':y\_pred})

print(df\_predict)

## identify additional attributes for modelling results & evaluation

reg.score(x, y)

print(f"The regression score is {reg.score(x,y)}")

reg.coef\_

print(f"The coefficient is {reg.coef\_}")

reg.intercept\_

print(f"The intercept is {reg.intercept\_}")

# evaluating LR performance using RSME

RSME = (np.sqrt(mean\_squared\_error(y\_test, y\_pred)))

print(RSME)