

**ANL252**

**PYTHON FOR DATA ANALYTICS**

**ECA**

# **End-of-Course Assessment**

**January 2023 Presentation**

**Submitted by:**

|  |  |
| --- | --- |
| **Name** | **PI No.** |
| **LIM WEI QUAN** | **K2110655** |

**Tutorial Group: ­­­­­­­­­­­­ T 05**

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

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

**Question 1**

Text

Description automatically generated with low confidence

*Result from the python code as per Appendix 1*

I use read\_csv() function to read a dataset as a Pandas dataframe. After that I use the isnull() method that requires Python to verify each dataframe cell and return True if it is a NaN. Each “True” values in each row or column are summed up using the. sum() method. By counting missing values, it corresponds to count the number of missing values. Next, each column's values will be added together when axis = 0.

In the end, I managed to find out that Claim\_ID and Actual variable contain missing values. Claim\_ID variable has 5 numbers of the missing values, whereas Actual variable contains 1677 number of the missing values.

**Question 2**

Table

Description automatically generated with medium confidence

*Result from python code as per Appendix 2*

For those blanks in the Claim\_ID and Actual column, it cannot replace with the mean of the Claim\_ID and Actual column. This is because claim id and actual date of claim settlement is non-numeric value so I unable to compute the mean accordingly.

Claim\_ID is a unique identifier for each claim, and each ID represents a distinct claim. Replacing Claim\_ID with the mean would result in a loss of important information and could potentially introduce errors in the analysis.

For the actual date of claim settlement, it represents a date variable, which is not a numeric variable in the sense that we can not to calculate the mean. Additionally, the missing values in this column could indicate a variety of reasons why a claim has not been settled yet and replacing them with the mean could lead to inaccurate and potentially misleading results.

If there are missing values in other columns that are numerical in nature, I will consider replacing those missing values with the mean of the column. However, this should only be done if it makes sense for the specific variable and context. For example, it may not be appropriate to replace missing values in a column that represents binary variables (e.g., 0 or 1) with the mean, as this could create non-binary values and lead to misleading results.

In the end, python managed to filter out those entire rows which contain missing values and show the non-null values only. I only need to analyze those complete and useful rows to do further analysis.

**Question 3**

|  |  |
| --- | --- |
|  |  |
| Before applying pd.to datetime () function | After applying pd.to datetime () function |

*Result from first data preparation task as per appendix 3*

The first data preparation tasks implemented by me is using pd.to datetime () function. The purpose of using pd.to datetime () function is converting ‘Actual’ column to the Pandas datetime object. After that, I uses the ‘dt.strftime method to format datetime object as a string by showing the date component only. It also assigns the resulting string back to the ‘Actual’ column. Therefore, you can see the result from the respective code which only shows the date in DD/MM/YEAR. The reason for removing the time from the ‘Actual’ column because I think that time does not provide any useful information for further analysis. In the end, I used dataframe.to\_csv to save the updated changed into csv file.

|  |  |
| --- | --- |
|  |  |
| Before applying the pd. To datetime () function & dt.strtime method | After applying the pd. To datetime () function & dt.strtime method |

*Result from second data preparation task as per appendix 3*

Secondly, I will use pd.to datetime () function and dt.strftime method to split the value ‘YYYYMMDD’ date format under’ ‘ Created’ column into component parts (day, month, year) and reformat it as DD-MM-YYYY. Firstly, I uses pd.read\_csv function to read the CSV file into a Pandas dataframe called ‘df’ then use pd.to datetime () function to convert the ‘Created’ column to Pandas datetime object by indicating the input format using the ‘format’ parameter. The datetime object is then formatted as a string with the desired format (%d-%m-%Y) using the dt.strftime() method, and the resulting string is assigned back to the date column. In the end, I can get the desired result such as change 20210112 to be 12/01/2021. After applying the pd. To datetime () function and dt.strtime method, I can easily analysis the ‘created’ column with the clear date format. In the end, I used dataframe.to\_csv to save the updated changed into csv file.

|  |  |
| --- | --- |
|  |  |
| Before | After |

*Result from third data preparation task as per appendix 3*

The third data preparation will be cleaning the invalid value occurs in the ‘Amount’ column.

I managed to find out that 1,762. OO under row 3700 and it will affect further analysis of the data later. I used the ‘replace’ method of pandas to replace the string 'OO' in the 'Amount' column of the DataFrame (df) with the string '00'. The regex=True parameter tells the method to use regular expressions to find and overwrite the 'OO' pattern. After that I uses ‘pd.to\_numeric() function to convert the values of a dataframe column to a numeric data type and then write the updated data frame to a new CSV file using the to\_csv() function.

In the end, I managed to clean by removing time from the ‘Actual’ column, convert the date format for the ‘Created’ column and correct the 1762.OO value in the ‘Amount’ column. I can use the clean csv dataset to do further analysis later.

**Question 4**

**Chart, histogram

Description automatically generated**

*Histogram for first insight as per appendix 4*

I used the histogram to create first insight about the number of claims settled between 0 to 30 days. The reason of creating this histogram chart is helping the insurance company to understand the efficiency of settling the claims for their customer.

Firstly, I import pandas and matplotlib.pyplot and then use pd.read\_csv () function to read the csv file into pandas dataframe. Next, I define a dataframe call df[‘Day Taken’] by using ‘Actual’ column to minus the ‘Created’ column to get the number of the days taken to settle each claim. I also use pd.to\_datetime() to format the date in the ‘Actual’ and ‘Created’ column.

After I calculated number of the days taken to settle each claim, I use df['Days\_Taken'] > =1 to find out the ‘Days\_Taken’ column is greater than or equal to 1 and use df['Days\_Taken'] <=30 to find out the ‘Days\_Taken’ column is less than or equal to 30. The & is an operator to combine both conditions where the ‘Day\_Takens’ value is between 1 to 30.

Last but not least, I use dataframe.groupby() function to group those value between 1 to 30 and created different bin to classify the number of claims settled in each category. Then I use .plot.hist(bins=30,align='left') to create 30 bins and the bars aligned to the left edge of each bin. Other than that, I label the title and axis label by using plt.title, plt.xlabel and plt.ylabel.

Lastly, I type plt.show() function to show the bar chart. From the bar chart, I can observe that most of the claims settled between 0 -10 days. The histogram considers as right skewed because it has a “tail” on the right side of the distribution. Based on this result, it is considered quite efficient for an insurance company to process and approve the claims to their customer. This is because customer will feel that your insurance company is more reliable and efficient compared to your competitor because most of the claims able to approve within 10 days.

Chart, bar chart

Description automatically generated

*Bar chart for second insight as per appendix 4*

The second insight for the corporate claims processing of the insurance company is the average payout amount by region. Based on the datatset, there are only 2 regions available in the ‘Region’ column which is FVS and LOC. Therefore, I created a bar chart to find out the average payout amount by region to better understand the medical cost and repair cost on different region.

At first, I import pandas and matplotlib.pyplot and then use pd.read\_csv () function to read the csv file into pandas DataFrame. Next, I created a new DataFrame called avg\_payout\_by region with two columns which are ‘Region’ and ‘Amount’. I also use mean () method to calculate the mean value of the ‘Amount’ column for each group and then use reset\_index() method to convert the grouped result into a new DataFrame.

After I calculated the average payout amount by region, I use plot.bar() to plot a bar chart of the average payout amount by region. Lastly, I label the title and axis label by using plt.title, plt.xlabel and plt.ylabel. By using plt.show ()function, the completed bar chart will be showm.

From the histogram, I observe that the average payout amount of the ‘FVS’ region is higher than ‘LOC’. Since the ‘FVC’ and ‘LOC’ region is internal region classification code, I only can guess that medical cost and repair cost at ‘FVC’ region is high and the population size of the ‘FVC’ region is big. ‘FVC’ region might be a developed city. Thus, an insurance company might consider adjusting insurance pricing and policies for ‘FVC’ region accordingly.

**Chart, pie chart

Description automatically generated**

*Pie chart for third insight as per appendix 4*

The last insight for the corporate claims processing of the insurance company is creating pie chart to find out the proportion of paid and unpaid claims. The purpose of creating this insight is finding out whether a claim has been paid or not. It is possible to use this information to analyze the efficiency of the claims processing system.

Firstly, I import pandas and matplotlib.pyplot and then use pd.read\_csv () function to read the csv file into pandas DataFrame. Next, I use value\_counts() function to count the number of occurrences of each value in the ‘Paid’ column of the DataFrame ‘df’. After that, I use the. plot() method to create a visualization of the data by using a kind parameter (kind-‘pie’) to create a pie chart and autopct parameter to display a float showing the percentage of each count.

Lastly, I use plt.title to label the plot title and plt.show ()function to show the completed pie chart. Thus, pie charts can be used to display the proportions of claims paid and unpaid. According to the pie chart, 93.1% of the claim had been paid whereas 6.9% of the claims are still pending to pay. For those 6.9% of the unpaid claim, I believed that it might have some reason behind on it.

**Question 5**

Chart, scatter chart

Description automatically generated

*Result from python code as per appendix 5*

Firstly, I import pandas, matplotlib.pyplot, estimator linear regression from the sklearn.linear\_model and train\_test\_split() function from the model\_selection as well. After that data will be load into a pandas dataframe in order to use linear regression modeling to forecast the delay in days. Next, I use pd.read\_csv () function to read the csv file into pandas DataFrame and use dropna() method to remove all rows which contain any missing value.

By using pd.to\_datetime() format, date format in ‘Actual’ and ‘Planned’ column will convert into specified format as DD/MM/YYYY. Subsequently, I created a new column call ‘Delay’ by using data from ‘Actual’ column minus off the data from ‘Planned’. In this scenario, the ‘Planned’ column from the data dataframe serves as the representation of the features needed for prediction. Double square brackets are used to pick a single column and assign the value as a pandas dataframe. Y will be representing the value to be predicted or target variable which is ‘Delay’ column in this case. A training set and a testing set were created by randomly dividing the dataset. For the test\_size parameter is set to 0.2, It means that 20% of the data will be used for testing and the balance of the data will be used for training. By setting random state=42, we are

fixing the random seed to a certain number (42 in this example) to ensure that the train-test split is consistent across executions of the function. This helps to ensure consistency in the outcomes and reproducibility between multiple runs.

I train a linear regression model by using LinearRegression(). For the fit() method, it will train the model on training set. Dates are represented as integer values in the ‘Planned’ column by converting it to int64 format, which allows the LinearRegression() model to use those values as input features. Afterwards, I use predict () function to generate predictions for the delay values in the testing set using the trained linear regression model. Since that time the dates in the ‘Planned’ column were already transformed to int64 format, the X test data must also be converted using astype (“int64”) before it can be used as input for the forecast function.

In addition, I use model.score() method to calculate the coefficient of determination (R-squared) of the linear regression model. In a linear regression model, the amount of variance in the dependent variable (y) that can be predicted from the independent variable (X) is expressed statistically as R-squared. In this case, the R-squared score will be calculating by using ‘X\_test’ and ‘Y\_test’.

Lastly, I use plt.scatter() method to create a scatter plot with the values of the y\_test data on the y-axis and the values of the X\_test data on the x-axis. In this scenario, the color option is set to black by using color parameter. By using plt.plot, a line graph with the values of the X\_test data on the x-axis and the anticipated values of the y\_test data on the y-axis will be plotted. The color of the lines is set to blue, and the line thickness of the line set as 3mm. In the end, I use plt.show ()function to show the completed pie chart.

In the nutshell, I never apply any further data pre-processing for modelling because data pre-processing such as remove outlier values does not given any effect for the current result. Even though I tried to remove outlier by removing those delay more than 50 and less than -50, the result also the same with the current and the shape of the scatter plot does not have any interesting point for further analysis. Thus, those outlier values do not need to remove.

**Question 6**

Based on the result from Question 5, R-squared score is 0 which means that the model does not explain any of the variability in the target variable. In this scenario, the correlation between two variables is no linear relationship as the R-squared score does not close to -1 or 1.

For the linear regression equation, please refer to the below linear regression equation for your reference. In order to calculate intercept and slope coefficient, I use model.intercept and model coef attributes to calculate the value.

Original Linear Regression Equation:

Delay = slope\*Planned + intercept

m = model.coef\_[0]

b = model.intercept\_

print(f"Linear regression equation: Delay = {m:.2f} \* Planned + {b:.2f}")

Revised Linear Regression Equation after apply model.intercept and model.coef.

**Delay = -0.00\*Planned + 84**

It seems like the equation does not provide any meaningful information to predict delay accurately.

**Appendix:**

1. **Python code for question 1**

import pandas as pd

# Read the dataset into a dataframe

df = pd.read\_csv(r"C:\Users\Shawn Lim\Desktop\ECA.csv")

*# Check for missing values in each variable*

null\_counts = df.isnull().sum(axis=0)

null\_counts2 = df.isnull().any()

print(null\_counts)

print(null\_counts2)

1. **Python code for question 2**

import pandas as pd

import numpy as np

*# Read the dataset into a dataframe*

df = pd.read\_csv(r"C:\Users\Shawn Lim\Desktop\ECA.csv")

*# Remove those blank (whitespace) from the dataframe*

df\_clean = df.dropna(axis=0, how = 'any')

*# Verify that there are no more blank values in the dataframe*

df\_clean

1. **Python code for question 3 – first data preparation task**

import pandas as pd

*# Read the dataset into a dataframe*

df = pd.read\_csv(r"C:\Users\Shawn Lim\Desktop\ECA.csv")

df\_clean = df.dropna(axis=0, how='any').copy()

*# Convert the date column to a Pandas datetime object*

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

*# Format the datetime object as a string with only the date component*

df['Actual'] = df['Actual'].dt.strftime('%d/%m/%Y')

*# Save the updated DataFrame to a current CSV file*

df.to\_csv(r"C:\Users\Shawn Lim\Desktop\ECA.csv", index=False)

df['Actual']

**Python code for question 3 – second data preparation task**

import pandas as pd

*# Read the dataset into a dataframe*

df = pd.read\_csv(r"C:\Users\Shawn Lim\Desktop\ECA.csv")

df\_clean = df.dropna(axis=0, how='any').copy()

*# Convert the date column to a Pandas datetime object*

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

*# Format the datetime object as a string with the desired format*

df['Created'] = df['Created'].dt.strftime('%d/%m/%Y')

*# Save the updated DataFrame to a current CSV file*

df.to\_csv(r"C:\Users\Shawn Lim\Desktop\ECA.csv", index=False)

df['Created']

**Python code for question 3 – third data preparation task**

import pandas as pd

*# Read CSV file*

df = pd.read\_csv(r"C:\Users\Shawn Lim\Desktop\ECA.csv")

*# Identify cells with 'OO' and replace with '00'*

df = df.replace({'Amount': {'OO': '00'}}, regex=True)

*# Convert 'Amount' column to numeric data type*

df['Amount'] = pd.to\_numeric(df['Amount'])

*# Write updated data frame to current CSV file*

df.to\_csv(r"C:\Users\Shawn Lim\Desktop\ECA.csv", index=False)

df['Amount']

1. **Python code for question 4 – first insight**

import pandas as pd

import matplotlib.pyplot as plt

*# Read the CSV file into a pandas dataframe*

df = pd.read\_csv(r"C:\Users\Shawn Lim\Desktop\ECA.csv", parse\_dates=['Planned', 'Actual', 'Created'], dayfirst=True)

*# Calculate the number of days taken to settle each claim based on the difference between Actual and Created columns*

df['Days\_Taken'] = (pd.to\_datetime(df['Actual']) - pd.to\_datetime(df['Created'])).dt.days

*# Filter the data to only include claims that were settled within 30 days*

df\_30days = df[(df['Days\_Taken'] >= 0) & (df['Days\_Taken'] <= 30)]

*# Plot the histogram of days taken to settle claims*

ax=df\_30days['Days\_Taken'].plot.hist(bins=30,align='left')

*# Set the plot title and axis labels*

plt.title('Distribution of Days Taken to Settle Claims')

plt.xlabel('Days Taken to Settle Claim')

plt.ylabel('Number of Claims')

*# Set the lower limit of the x-axis to 0*

ax.set\_xlim(left=0)

*# Show the plot*

plt.show()

**Python code for question 4 – second insight**

import pandas as pd

import matplotlib.pyplot as plt

*# Read the CSV file into a pandas dataframe*

df = pd.read\_csv(r"C:\Users\Shawn Lim\Desktop\ECA.csv")

*# Calculate the average payout amount by region*

avg\_payout\_by\_region = df.groupby('Region')['Amount'].mean().reset\_index()

*# Plot a bar chart of the average payout amount by region*

plt.bar(avg\_payout\_by\_region['Region'], avg\_payout\_by\_region['Amount'])

*# Set the plot title and axis labels*

plt.title('Average Payout Amount by Region')

plt.xlabel('Region')

plt.ylabel('Average Payout Amount')

*# Show the plot*

plt.show()

**Python code for question 4 – third insight**

import pandas as pd

import matplotlib.pyplot as plt

*# Read the CSV file into a pandas dataframe*

df = pd.read\_csv(r"C:\Users\Shawn Lim\Desktop\ECA.csv")

*# Create a pie chart of the payment status*

df['Paid'].value\_counts().plot(kind='pie', autopct='%1.1f%%')

*# Set the plot title*

plt.title('Proportion of Paid vs Unpaid Claims')

*# Show the plot*

plt.show()

1. **Python code for question 5**

import pandas as pd

import matplotlib.pyplot as plt

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

from sklearn.linear\_model import LinearRegression

*# Load the data into a pandas dataframe*

data = pd.read\_csv(r"C:\Users\Shawn Lim\Desktop\ECA.csv")

*# Drop any rows with missing values in the 'Actual' column*

data = data.dropna(subset=['Actual'])

*# Convert the Planned and Actual columns to datetime format with specified format*

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

data['Actual'] = pd.to\_datetime(data['Actual'], format='%d/%m/%Y')

*# Create a new column for the delay between Planned and Actual dates*

data['Delay'] = (data['Actual'] - data['Planned']).dt.days

*# Split the data into training and testing sets*

X = data[['Planned']]

y = data['Delay']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

*# Train a linear regression model on the training set*

model = LinearRegression()

model.fit(X\_train.astype('int64'), y\_train)

*# Predict the delay for the testing set*

y\_pred = model.predict(X\_test.astype('int64'))

*# Calculate the R-squared score for the model*

score = model.score(X\_test.astype('int64'), y\_test)

print(f"R-squared score: {score:.2f}")

*# Plot the actual vs predicted values*

plt.scatter(X\_test, y\_test, color='black')

plt.plot(X\_test, y\_pred, color='blue', linewidth=3)

plt.xticks(())

plt.yticks(())

plt.show()