

**ANL252 (Online)**

**Python for Data Analytics**

# **End-of-Course Assessment**

**January 2023 Presentation**

**Submitted by:**

|  |  |
| --- | --- |
| **Name** | **PI No.** |
| **Jansen Edmund Joseph Junior** | **H2070894** |

**Tutorial Group: ­­­­­­­­­­­­ T 05**

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

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

**Question 1**

Firstly, we import the csv for it to be displayed as a Pandas dataframe:

*import os*

*import pandas as pd*

*employee\_data=pd.read\_csv('ECA.csv')*

*employee\_data*

After reading the dataset (employee\_data) as a Pandas data frame, we print the data frame to get the info of it’s shape.

*24213 rows × 12 columns*

Using the command info(), we are able to display all the variables in the data frame and identify its type as well as the Non-null count.

<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

In a perfect data set with no null entries, the non-null count should match the total number of rows in the full data frame. From the info displayed using the code line employee\_data.info(), we can see that the following variables contains missing values:

1. Claim\_ID
2. Actual

Another method to identify the null entries is using isnull(), where axis=0.

employee\_data.isnull().sum(axis = 0)

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

This supports the previous method, showing that Claim\_ID and Actual has null values present. The total number of lines with null entries are 5 from Claim\_ID and 1677 from Actual.

**Question 2**

The solutions that we may adopt to deal with the null values are to remove or to fill them up. Since we have determined that the two variables of Claim\_ID and Actual has null values, we should treat them appropriately.

For Actual, the total number of entries with null entries is 1677, which is a large amount of entries, approximately 6.9% of the total dataframe. Due to this, it needs to be treated in order to not disrupt the use for analysis. For this, we may use the following codes:

employee\_data\_2=employee\_data.fillna(method='pad', axis=1)

By doing so, instead of inputting dummy dates, we use the dates found in ‘Planned’, to replace the null values found in ‘Amount’. By doing so, we assumed that the entries for the actual date of pay-outs were the same dates that were planned. This way, we are able to keep the other important data such as the amount paid out and the policies it was claimed under.

For Claim\_ID, due to the small number of entries that we affected, removing them from the dateframe should not result in any skewed results. Furthermore, when using dropna() on the original data set, the result is a shape is:

22531 rows × 12 columns

This shows that the null entries found in Claim\_ID and Actuals lies on different lines, and would not affect the lines that were previously treated. Therefore, we use the code:

employee\_data\_3=employee\_data\_2.dropna()

This would result in the shape of:

24208 rows × 12 columns

Lastly, we perform a check on the dataframe again to ensure that there is no null value present:

employee\_data\_3.isnull().sum()

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

**Question 3**

1. **Formatting of data types**

Firstly, we display the dtypes of the current dataframe:

Claim\_ID object

Policy\_No object

Name object

Planned object

Actual object

Created object

Amount object

Paid object

Category object

Terms object

Region object

Type object

dtype: object

We can see that the above highlighted variables are of the wrong type and would affect the use of it for analysis.

Firstly we treat the variables of ‘Planned’, ‘Actual’ and ‘Created’ which are supposed to be datetime64[ns].

employee\_data\_3.Planned=pd.to\_datetime(employee\_data\_3.Planned)

employee\_data\_3.Actual=pd.to\_datetime(employee\_data\_3.Actual)

employee\_data\_3.Created=pd.to\_datetime(employee\_data\_3.Created)

Next, we treat the variables ‘Paid’ to change it to a Float64. We use float instead of int due to it being in currency and would require higher accuracy in terms of cents, thus the decimal points are crucial.

employee\_data\_3['Amount']=pd.to\_numeric(employee\_data\_3['Amount'])

When using this code, we would encounter the error of

ValueError: Unable to parse string "1762.OO" at position 3698

This error occurs because in the dataframe, the cell value was keyed in as 1762.OO instead of 1762.00, and therefore python was unable to change the dtype to a float. We rectify it with the following codes:

employee\_data\_3.at[3698, 'Amount']=1762.00

employee\_data\_3['Amount']=pd.to\_numeric(employee\_data\_3['Amount'])

lastly, we treat the categorical variable and change their type accordingly:

employee\_data\_3['Paid']=employee\_data\_3['Paid'].astype('category')

employee\_data\_3['Category']=employee\_data\_3['Category'].astype('category')

employee\_data\_3['Terms']=employee\_data\_3['Terms'].astype('category')

employee\_data\_3['Region']=employee\_data\_3['Region'].astype('category')

employee\_data\_3['Type']=employee\_data\_3['Type'].astype('category')

This would result in the dtype as shown:

Claim\_ID object

Policy\_No object

Name object

Planned datetime64[ns]

Actual datetime64[ns]

Created datetime64[ns]

Amount float64

Paid category

Category category

Terms category

Region category

Type category

dtype: object

1. **Removing outliers**

We may use the dataprep module to help us to visualise the other variables that need treating.

from dataprep.eda import create\_report

create\_report(employee\_data\_3)

Through the report, we may get some insights on the variables individually. For ‘Amount’, by looking at the box plot generated by the report (**Figure 1**), we can see that there are many outliers present in the dataframe and should be removed.

Graphical user interface, chart, box and whisker chart

Description automatically generated

**Figure 1**: Dataprep Report for ‘Amount’

We can proceed to remove the outliers with the following codes:

q1=employee\_data\_3['Amount'].quantile(q = 0.25)

q3=employee\_data\_3['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 threshold: {low\_bound}\nupper threshold: {upp\_bound}')

employee\_data\_nooutlier=employee\_data\_3[~((employee\_data\_3['Amount'] < low\_bound) | (employee\_data\_3['Amount'] > upp\_bound))]

employee\_data\_nooutlier

By removing the outliers, its results in a dataframe with the shape:

22966 rows × 12 columns

1. **Removing of redundant variables**

Lastly, we may remove insignificant columns in the dataframe that have no purpose or use for analysis. We may drop the columns ‘Claim\_ID’ and ‘Category’. After the data preparation, ‘Claim\_ID’ shows 100% unique values while ‘Category’ shows 0% unique values according to the Dataprep report. This would mean that it would not have any significant value when doing analysis and should be removed. Other variable that can be removed is ‘Created’, ‘Terms’, ‘Region’ and ‘Type’, as they would not be used in the analysis. This would result in a cleaner dataframe with less columns, making it easier to view for the user.

employee\_data\_clean=employee\_data\_nooutlier[['Policy\_No', 'Name', 'Planned', 'Actual', 'Amount', 'Paid']]

employee\_data\_clean

**Question 4**

1. **Insights on claim personnel**

In this, we look into the frequencies of which the claims are made by each person. To provide comparison, we can split the dataframe by the years.

import datetime as dt

df2021=employee\_data\_clean[employee\_data\_clean['Actual'].dt.year==2021]

df2022=employee\_data\_clean[employee\_data\_clean['Actual'].dt.year==2022]

Using Dataprep, we create the Dataprep report for both 2021 and 2022. We look at the variable of ‘Name’.

Chart, bar chart

Description automatically generated

**Figure 2**: Dataprep Report for 2021 ‘Name’

From **Figure 2,** we see that the top 10 names that appear in the dataframe, with the Troy Phillips having 472 entries and Taylor Campbell appearing 433 times. This shows that the top 10 entries within the dataframe had did more than 400 claims for the year of 2021. This should be alarming as they had successfully filed a large amount of claims within a year. We compare the results in 2022 (**Figure 3**).

Chart, bar chart

Description automatically generated

**Figure 3**: Dataprep Report for 2022 ‘Name’

Although the dataframe for 2022 does not cover the entire year, it shows repeated names within the top 10 names with most claims. The company should look into these clients, and do an audit on the type of claims they have been making. This is to reduce the risk of them exploiting their claim policies or to prevent fraud. This would help the company relook into their systems to approve their claims and may reduce the claim expenditure.

1. **Claim patterns**

When looking at the Dataset Insights for the ‘Amount” variable is skewed. This would make sense as there are a wide array of policies that could be claimed. The results are generally skewed due to the lower claims being a lot more frequent than the higher claims. However when we look at the comparison data the amount claimed for the two years, they both show the same surge in claim amount around the $2000 mark (**Figure 4**). Once again the company can look into the reason on why the higher claim amount experiences a surge and what rectifications can be made to tighten their claim process.

Chart, histogram

Description automatically generated

**Figure 4**: Frequency of Claims by amount

1. **Frequency of delays**

Lastly, we can look at the frequencies of delay that the company has based on their actual and planned dates of payout. Figure 5 shows the planned dates that were forecasted to do a payout while Figure 6 shows the actual dates that payments were actually made.

Chart, line chart, histogram

Description automatically generated

Figure 5: Planned dates for Claim Pay-outs in 2022

Chart, line chart, histogram

Description automatically generated

Figure 5: Actual dates for Claim Pay-outs in 2022

Looking at the two line charts, we can see that the shape of both are fairly similar in the peaks and drops. This would mean that the company is doing well in forecasting their payout dates for the clients.

**Question 5**

The approached taken to this linear regression is that the amount paid (Y independant variable) would affect the number of days delayed (X dependant variable) for claims to be processed, where the assumption is that the higher the claim value, the more likely it would be delayed. Firstly we need to prepare the data type for ‘Actual’ and ‘Planned’, as it is currently in datetime format and needs to be changed to values so that the new coloumn ‘Delay’ could be stored as an integer.

from sklearn.linear\_model import LinearRegression

model=LinearRegression()

planned=pd.to\_datetime(employee\_data\_clean['Planned']).dt.day.values.reshape(-1, 1)

actual=pd.to\_datetime(employee\_data\_clean['Actual']).dt.day.values.reshape(-1, 1)

We then store the two variables that would be used to fit in to the linear regression

delay=actual-planned

amount=employee\_data\_clean['Amount']

Subsequently we fit the variables into the linear regression.

# Fit the model on the planned and actual data

model.fit(delay, amount)

By doing the above steps, we are able to generate the intercept and coeffiecent of the linear regression.

# Print the intercept and coefficients of the model

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

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

Intercept: 2672.285868750985

Coefficient: [3.17751582]

**Question 6**

The regression equation is: Y= 3.17751582 + 2672.29(X).

The intercept is the of equation is known as the constant, which represents the mean value while the coefficient shows the relation on how the dependant variable would react to a change in the independent variable.

In this linear regression, the constant is the amount the claim is entitled to while the dependant is the number of days that the claim would be delayed.

When looking at the intercept of 2672.29, it shows the mean of the entries that were in the variable ‘Amount’. The coefficient, being positive shows that an increase in the independent variable, would lead to a decrease in the dependant variable. This means that an increase in the claim amount would lead to a increase in the number of days in delay. This shows that there is a positive relation between the two variables. This would make sense in the real world, as with a higher claim value, there is often times more processes and approval to seek before the claims is given to the customer.