****

**ANL252**

**Python for Data Analytics**

**End-of-Course Assignment**

**January 2023 Presentation**

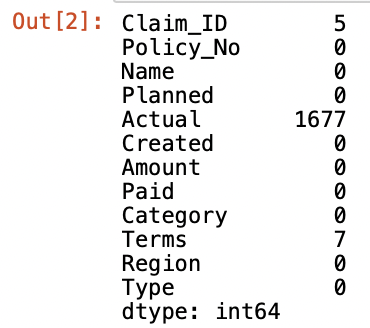
|  |  |
| --- | --- |
| Name: | PI Number: |
| CHEN JING | K1981651 |

**Submission Date: 06 March 2023**

*All Python codes expressed in text format are included in the appendix of this report.*

**Question 1.**





Based on the output [1], the ECA.csv file contains 24,213 rows of observations by 12 columns of variables. Per output [2], the variables that contain the missing datas include the “Claim\_ID '' variable, “Actual”variable and the “Terms” variable.

**Question 2.**



****

To adjust for the missing values, need to find out the number of observations (rows) with at least 1 missing value amongst the 12 variables. Based on the output [3], there are a total of 1689 observations with at least 1 missing value in “Claim\_ID”, “Actual” and “Terms'' columns. There is a sum of 12 observations with missing values in “Claim\_ID” and “Terms” per output [4].

****

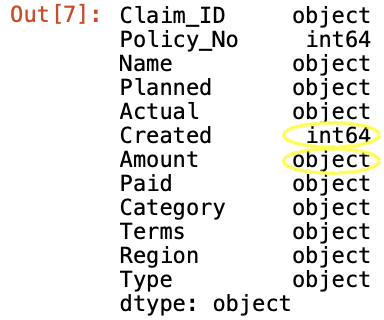
The missing values in the “Actual” column is an indication that the insured has not claimed the insurance plan yet. Such information is less meaningful for the analysis on the claim processing. As such, the observations with missing values under the “Actual” column will be dropped. As seen in output [5], after dropping the missing values in the “Actual” column, the size of the dataset was reduced to 22,536 rows by 12 columns.

****

For the observations with missing values in columns “Claim\_ID” and “Terms”, there is data under the “Planned” and “Actual” columns. The column data of “Claim\_ID” and “Terms” itself could be less useful for the analysis on the claim processing but the data from the date variables are critical for such analysis. For the claim processing analysis, missing values of these two variables will be replaced with “1234567899” and “Unknown” under the “Claim\_ID” and “Terms” columns respectively. The size of the dataset remained at 22,536 rows by 12 columns per output [6] since the 12 rows of observations with missing values under “Claim\_ID” and “Terms” were replaced instead of deleted.

**Question 3.**

Other than replacing missing values, there are other forms of data preparation to check and understand the dataset before we can proceed to use the dataset for analysis. When we use pandas to read the csv files into the Jupyter Notebook, there could be some inaccuracy in the data type. As such, we can identify areas of improvement by looking at details of the data type assigned to each of the variables.

****

Based on the output [7], the "Amount" column, which should contain integer values, is currently of data type "object." This indicates that the column includes either text or a mixture of numeric and non-numeric values. To enable further analysis, the column needs to be converted to numeric data. Given that the column includes many numbers with decimals, it is recommended to convert the data type to float.

Furthermore, the data under the ‘Amount’ column has varying decimal places (‘dp’). Such inconsistency could lead to rounding errors and causing inaccuracy in the results of the data analysis. For example, a 3dp data could be bigger than the 2dp data when rounded up but less than the 2dp data when rounded down. Thus, it is crucial to standardize the number of decimal places to ensure reliable analysis results.

Regarding the "Created" variable, it should have the same data type as "Planned" and "Actual" since they are all date variables. The difference in data type is due to the different format used for the "Created" column. Therefore, corrections need to be made to ensure that the column is in date format and that the data type matches that of the other two date columns.

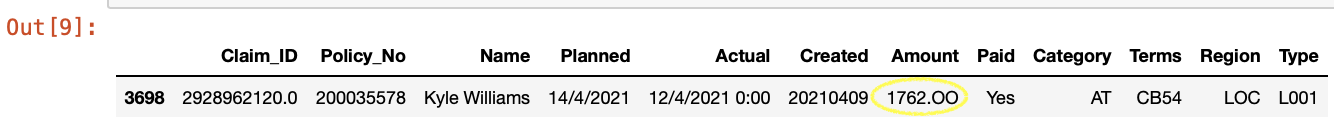
Further data preparation to ensure all data types are correct and standardized to ensure usability of the dataset:

1. **Correct data types of ‘Amount’ column**

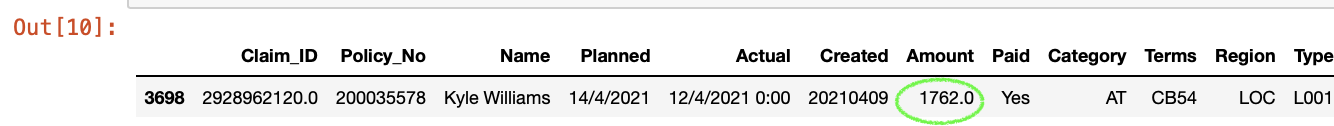
output [8]

****

The output [8] indicates that there is an anomaly, “1762.OO”, in the ‘Amount’ column which hinders the conversion of the column to float. Locate the specific row of data to better understand the anomaly.

****

From output [9], it has shown that there is a typo error where the alphabet O was used instead of the number 0.

****

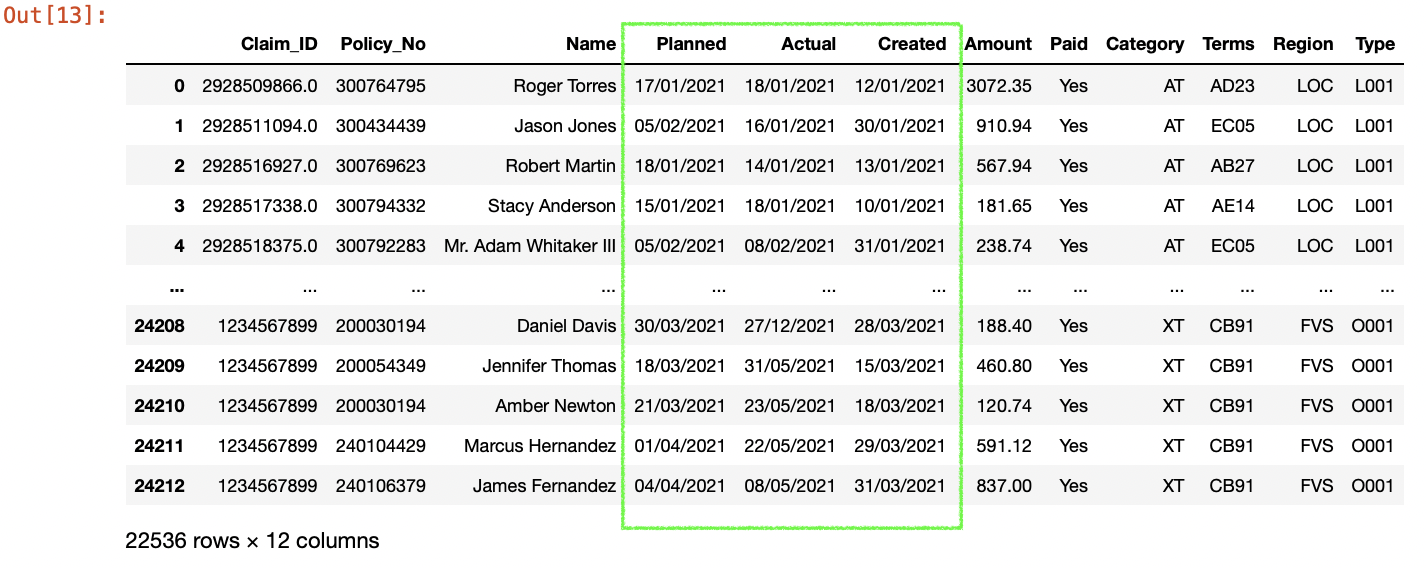
The correction was made with. replace () function and the data type was changed to float. After the correction, the anomaly has been corrected and the data are of the right type ‘float64’ as seen in output [10] and output [11].

1. **Standardize to ‘Amount’ column to 2 decimal places**



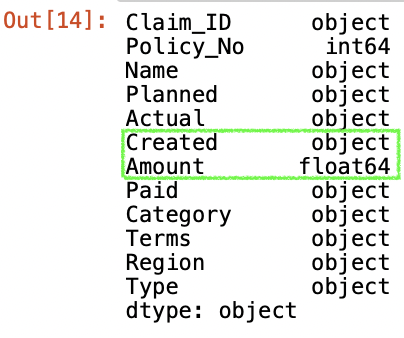
‘Amount’ column is now standardized with 2 decimal places with use of the .round() function. This format will facilitate easy analysis and interpretation by the dataset users.

1. **Standardize DATE variables - ‘Planned’, ‘Actual’ and ‘Created’**



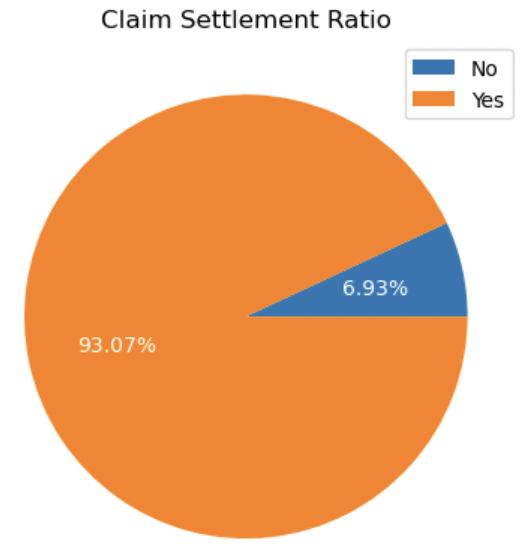
The dates in the dataset were converted to the Pandas datetime format using the pd.to\_datetime () function. The dt.strftime () function was then used to format the datetime object into a string with the desired format of ‘%d/%m/%Y’.

For the "Planned" column, which was in the format of DD/MM/YYYY, the day and month formats were converted from a single digit to two digits to match the Pandas datetime format. Similarly, the "Actual" column had the “hour:minute” removed in addition to the day and month to ensure standardization of the date variable.



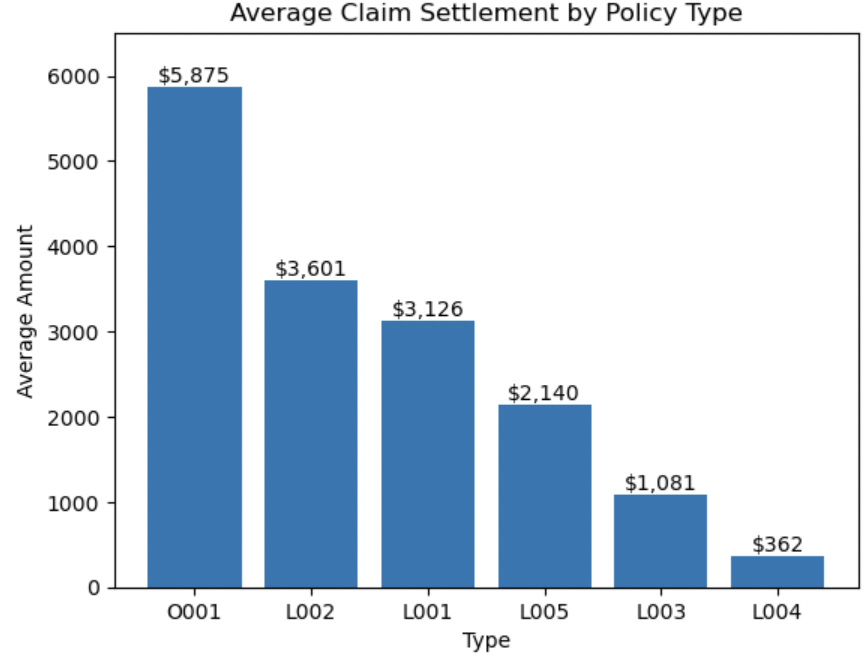
The "Created" column underwent the most transformation as it changed from an integer data type to an object data type and followed the standardized date format.

**Question 4.**

****

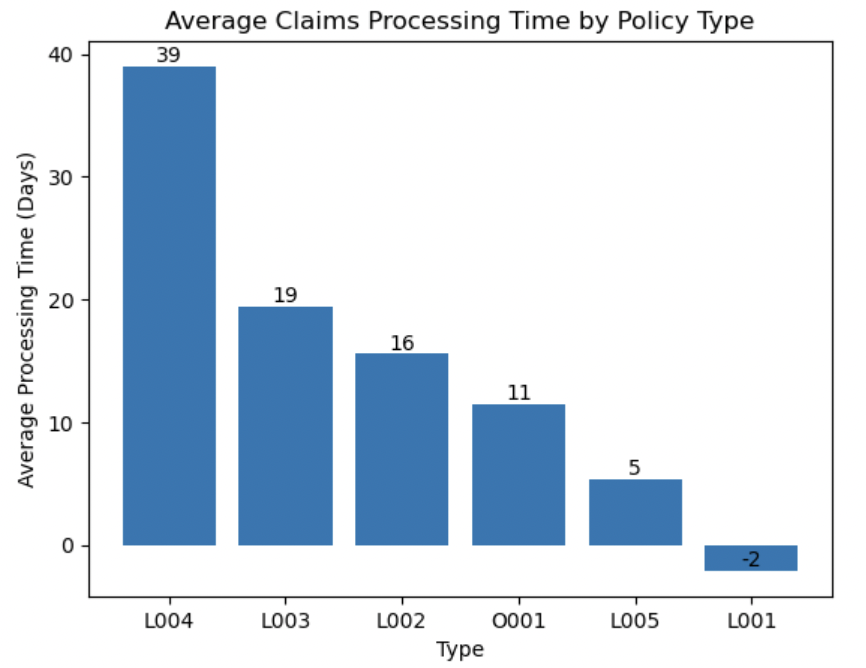
**Figure 1 | Claim Settlement Ratio**

Figure 1 illustrates the claim settlement ratio that helps insurance companies evaluate the number of claims paid out in relation to the number of claims received. This ratio reflects the insurer's dedication to fulfilling its obligations to policyholders. A high ratio of "Yes" in the figure indicates that the insurer settles claims with a higher probability, providing reassurance to the clients of this insurance company. Moreover, maintaining a favorable ratio can help the insurer attract potential clients and stay competitive in the industry.

****

**Figure 2 | Average Amount by Policy Type**

Figure 2 reveals that the Type "O001" policy has the highest average claim amount compared to the other five policy types. This suggests that the "O001" policy may be more susceptible to claims. This highlights the need for the insurance company to reassess its pricing and coverage to mitigate business risk effectively. With periodic knowledge on the average claim amounts for each policy type will help the insurance company to better manage its risk exposure, ensure its long-term financial viability, and provide comprehensive coverage to its policyholders.



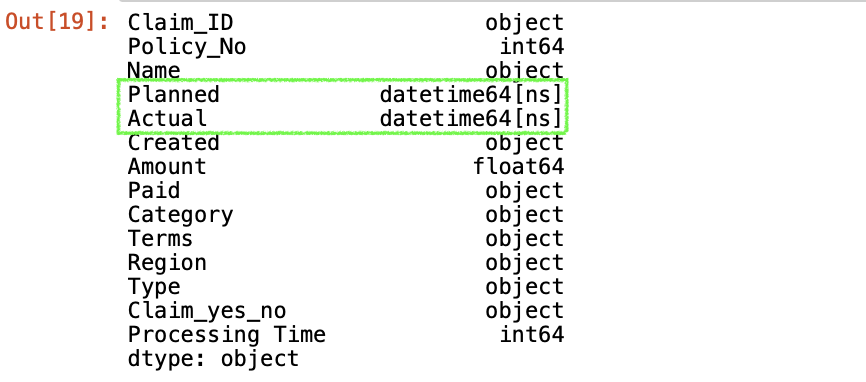
**Figure 3 | Average Claims Processing Time by Policy Type**

To calculate the average processing time, the actual date of claim settlement is subtracted from the planned date of claim settlement. Based on Figure 3, policy types "L001" to "L004" are ranked in order of fastest to slowest processing time.

By regularly evaluating and optimizing the claim processes could allow the insurance company to provide more efficient service to the policyholders. Policyholders are more likely to renew their policies with an insurer that provides prompt and efficient services. Therefore, a faster processing time for policy type "L001" can increase customer satisfaction and loyalty. Conversely, policy type "L004" with a slower processing time may erode customer confidence in the insurance company's ability to deliver claims. To remain competitive in “L004”, the insurance company could streamline the claims process to improve efficiency.

**Question 5.**

Before moving on to create the linear regression model, check through the data type of “Planned” and “Actual” columns to see if a conversion to datetime data type is required.



In this case, the “Planned” and “Actual” variables are of datetime format and thus conversion is not required before generating the “Delay” column.

First step of building the linear regression model will be to import all the relevant libraries. In this case, it will be the pandas, the matplotlib and the sklearn.linear\_model. Next, the number of days between the “Planned” and “Actual” dates that is equivalent to the number of days delayed in processing of the insurance claim is computed. To convert the difference between the dates under the “Planned” and “Actual” column, the “.days” function is used to convert the result to days. The resulting values for delay days will then be added to form the new column “Delay” in the dataframe.

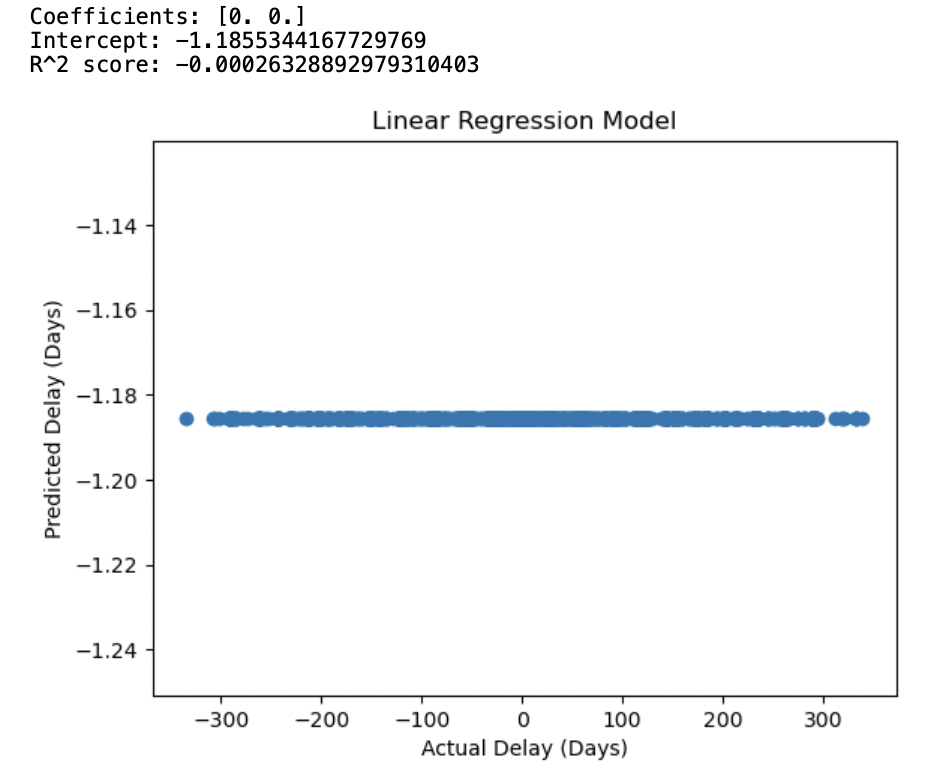
Before performing the linear regression modelling, the “Planned” and “Actual” column variables need to be converted from datetime format to integer format for the model to run. This is because the model requires numeric input variables to make predictions. This step is done by calculating the number of days between the maximum date variable index to minimum date variable index for each column variable. The purpose of this step is to get the days range for each of the data variables. The resulting values form the new columns “Planned\_days” and “Actual\_days” in the dataframe.

Moving on, the data is then split into training and testing sets, with 80% of the data randomly sampled to be used as training dataset. The remaining 20% is used as a testing set to test for the model’s accuracy. The step is done with the Pandas sample and data drop function.

After the data have been split, the X variables must be able identified to predict the Y, the targeted delay days. In this case, the x variables are the Planned days and Actual days while the y axis will be the target of predicted delay days. The linear regression model is then trained on the training data using the LinearRegression() function in the scikit-learn package.

Next, the trained model is used to predict the delay days for the testing set. The “Planned\_days” and “Actual\_days” column variables of the testing set are passed to the trained model using the .predict(X)method. The predicted result is saved as “y\_prediction”.

The corresponding model coefficients and R^2 are set to be printed out as these are useful in evaluating the model’s accuracy. The R^2 is a measure on the proportion of the variance for dependent variables (y).



**Figure 4 | Linear Regression Model**

Lastly, a scatter plot is created as seen in Figure 4. Purpose of the plot is to visualize the predicted vs actual delay for the testing dataset. The predicted values are plotted to form the different scattered data points (x,y) seen in Figure 4. The plot is also titled “Linear Regression Model”.

**Question 6.**

Coefficients = 0,0

Intercept = -1.185534416772969

Variables = Planned\_days & Actual\_days

R^2 = -0.00026328892979310403

Linear regression equation: Y = a + bX1 + cX2

Y = -1.185534416772969 + 0 Planned\_days + 0 Actual\_days

There are two inputs variables: “Planned\_days” and “Actual\_days”. With a negative R^2, it indicates the model is not explaining any variance in the testing data and not useful to predict the delay days. This could be due to the limited information provided by the two input variables alone being insufficient to explain the complex reasons behind the delays in claim processing.

**APPENDICES**

**Appendix 1 - Question 1**

|  |
| --- |
| import pandas as pd  *#Read CSV and note Missing values as blank, ‘Unkn” and ‘???’*  data = pd.read\_csv("ECA.csv",na\_values=[" ","Unkn","???"])  data |
| *#State variables (column title) that contains missing values*  data.isna().sum(axis = 0) |

**Appendix 2 - Question 2**

|  |
| --- |
| *#List out the observations (rows) with missing values*  missrow = data.isna().any(axis = 1)  data.loc[missrow[missrow == True].index] |
| *# Check for rows with missing values in specific columns*  data\_missing = data[data[['Claim\_ID', 'Terms']].isnull().any(axis=1)]  data\_missing |
| *#Drop missing values for “Actual” column*  data.dropna(subset = ['Actual'], inplace=True')  data |
| *# Filling missing values for multiple columns*  data.fillna({'Claim\_ID': '1234567899', 'Terms': 'Unkown'}, inplace=True)  data |

**Appendix 3 - Question3**

|  |  |
| --- | --- |
|  | *#Check the data type of Variables*  data.dtypes |
| *(1)* | *# Convert Amount column from string to a numeric data*  data['Amount'] = data['Amount'].astype(float) |
| *# Find row(s) where 'Amount' column contains '1762.OO'*  *data.loc[data['Amount'] == '1762.OO']* |
| *# Replace non-numeric value in 'Amount' column*  *data['Amount'] = data['Amount'].replace({'1762.OO': '1762.00'})*  *# Convert 'Amount' column to float*  *data['Amount'] = data['Amount'].astype(float)*  *# Check replacement of the non-numeric value in 'Amount' column*  *data.loc[data['Amount'] == 1762.0]* |
| *# Check the data type of 'Amount'*  *data['Amount'].dtype* |
| *(2)* | *# Convert 'Amount' column to 2 decimal place*  *data['Amount'] = data['Amount'].round(2)*  *data* |
| *(3)* | *# Convert the 'Planned' column to datetime format and then format it as DD/MM/YYYY*  *data['Planned'] = pd.to\_datetime(data['Planned'], format='%d/%m/%Y').dt.strftime('%d/%m/%Y')*  *# Convert the 'Actual' column to datetime format and then format it as DD/MM/YYYY*  *data['Actual'] = pd.to\_datetime(data['Actual'], format='%d/%m/%Y %H:%M').dt.strftime('%d/%m/%Y')*  *# Convert the 'Created' column to datetime format and then format it as DD/MM/YYYY*  *data['Created'] = pd.to\_datetime(data['Created'], format='%Y%m%d').dt.strftime('%d/%m/%Y')*  *# Print the updated DataFrame*  *data* |
| *#Check the data type of Variables*  *data.dtypes* |

**Appendix 4 - Question 4**

|  |
| --- |
| ## Figure 1  import pandas as pd  import matplotlib.pyplot as plt  *# Create a new column "Claim\_yes\_no" based on presence of data in the "Actual" column*  data['Claim\_yes\_no'] = data['Actual'].notna().map({True: 'Yes', False: 'No'})  *# Manually set the number of "No" values to 1677 and "Yes" values to 22536*  counts = {'No': 1677, 'Yes': 22536}  *# Create a pie chart with data labels as percentages and white text color*  plt.pie(counts.values(), labels=counts.keys(), autopct='%.2f%%', textprops={'color': 'white'})  *# Add a title and legend*  plt.title('Claim Settlement Ratio')  plt.legend()  *# Show the chart*  plt.show() |
| ## Figure 2  import pandas as pd  import matplotlib.pyplot as plt  *# Group the data by type and calculate the mean amount*  grouped\_data = data.groupby("Type")["Amount"].mean().sort\_values(ascending=False)  *# Create a bar chart using the grouped data*  plt.bar(grouped\_data.index, grouped\_data.values)  *# Add data labels to the chart*  for i in range(len(grouped\_data)):  plt.text(i, grouped\_data.values[i], f"${grouped\_data.values[i]:,.0f}", ha="center", va="bottom")  *# Add axis labels and a title*  plt.xlabel("Type")  plt.ylabel("Average Amount")  plt.title("Average Claim Settlement by Policy Type")  plt.ylim(0, 6500)  plt.figure(figsize=(10,8))  *# Display the chart*  plt.show() |
| ## Figure 3  import pandas as pd  import matplotlib.pyplot as plt  *# Convert the Planned, Actual, and Created columns to datetime*  data['Planned'] = pd.to\_datetime(data['Planned'])  data['Actual'] = pd.to\_datetime(data['Actual'])  *# Calculate the claims processing time by policy type*  data['Processing Time'] = (data['Actual'] - data['Planned']).dt.days  *# Calculate the average processing time by policy type*  avg\_processing\_time = data.groupby('Type')['Processing Time'].mean().sort\_values(ascending=False)  *# Create a bar chart of the average processing time by policy type*  plt.bar(avg\_processing\_time.index, avg\_processing\_time.values)  *# Add data labels to the chart*  for i in range(len(avg\_processing\_time)):  plt.text(i, avg\_processing\_time.values[i], f"{avg\_processing\_time.values[i]:,.0f}", ha="center", va="bottom")  plt.xlabel('Type')  plt.ylabel('Average Processing Time (Days)')  plt.title('Average Claims Processing Time by Policy Type')  plt.show() |

**Appendix 5 - Question5**

|  |
| --- |
| ## Building Linear Regression Model to Predict Delays - Figure 4  # Check data type of "Planned" and "Actual" columns  data.dtypes  import pandas as pd  import matplotlib.pyplot as plt  from sklearn.linear\_model import LinearRegression  *# Calculate the claims processing time by policy type*  data['Delay'] = (data['Actual'] - data['Planned']).dt.days  *# Convert the dates to integers representing the number of days between them*  data['Planned\_days'] = (data['Planned'].max() - data['Planned'].min()).days  data['Actual\_days'] = (data['Actual'].max() - data['Planned'].min()).days  *# Split the data into training and testing sets*  train\_data = data.sample(frac=0.8, random\_state=1)  test\_data = data.drop(train\_data.index)  *# Train a linear regression model on the training data where X-axis are features and y axis is target variable.*  X\_train = train\_data[['Planned\_days', 'Actual\_days']]  y\_train = train\_data['Delay']  model = LinearRegression()  model.fit(X\_train, y\_train)  *# Make predictions on the testing data*  X\_test = test\_data[['Planned\_days', 'Actual\_days']]  y\_test = test\_data['Delay']  y\_prediction = model.predict(X\_test)  *# Print the model coefficients and R^2 score*  print('Coefficients:', model.coef\_)  print('Intercept:', model.intercept\_)  print('R^2 score:', model.score(X\_test, y\_test))  *# Plot the predicted vs. actual delay values on the testing data*  plt.scatter(y\_test, y\_prediction)  plt.xlabel('Actual Delay (Days)')  plt.ylabel('Predicted Delay (Days)')  plt.title('Linear Regression Model')  plt.show() |