**Course Code : ANL252**

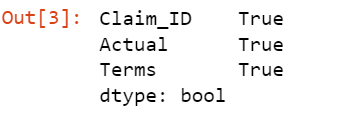
**Assignment No. : ECA**

|  |  |
| --- | --- |
| **Name** | **SUSS PI No.** |
| TAY BAN CHIN DAVE | Q2172629 |

**Submission Date : 6 March 2023**

**Answer to Question 1**

The variables that contain missing values are ‘Claim\_ID’, ‘Actual’, and ‘Terms’.



**Answer to Question 2**

In this analysis, I applied several cleaning steps to prepare the dataset for subsequent analysis. Firstly, I removed the 'Claim\_ID' column since it contains unique identifiers assigned to each row that are not relevant to the analysis. Similarly, I dropped the 'Terms' column since the meaning of each value for the internal codes of the organization is unknown, and thus does not contribute to our understanding of the data.

In addition, I removed rows with null values in the 'Actual' variable because these records may be irrelevant to the analysis. 6.93% of the dataset was affected, but most of the data was still preserved to generate meaningful insights. I also created a separate file containing only the records with null values in 'Actual' to provide feedback to the database team to provide more complete data in the future.

Overall, these cleaning steps ensure that the dataset is more accurate and relevant for our analysis, avoiding unnecessary clutter and potentially irrelevant information.

**Answer to Question 3**

Additional pre-processing steps taken in this analysis are aimed at improving the quality of the data for accurate analysis.

The first task was to remove the irrelevant columns 'Category,' 'Region,' and 'Type'. Like 'Terms,' which I removed earlier, the meaning of each value of the organization's internal codes is unknown, and thus does not contribute to our understanding of the data. This step assists in reducing data complexity and improving data visualisation.

The second task involved determining the processing time for insurance claims by calculating the time difference between the 'Created' and 'Actual' columns. Each record should have had a 'Actual' claim settlement date, regardless of whether the claim was approved or denied. Erroneous records with null values in 'Actual' that resulted in negative time difference values were found and removed, ensuring that only accurate and meaningful data is used for analysis. They are incorrect because it is not possible to approve claims before they are even created. A separate file containing only the incorrect records was also created to provide feedback to the database team to improve data accuracy in the future. 6.14% of the dataset was affected, and most of the data was still preserved to generate meaningful insights.

It is not advisable to replace null values in the 'Actual' column with mean or median values. We don't know whether the actual dates will deviate significantly from the mean or median. The presence of null values in this column indicates that the processing time for those claims is unknown or unrecorded, and using mean or median will result in inaccurate and potentially biased conclusions about claim processing time.

In the third task, I prepped the 'Amount' variable. It was converted from string to numeric datatype, and since only one invalid value was identified it was easy to amend directly in the source file.

Outliers in the data were also identified and removed, as they can significantly skew the analysis results and provide misleading insights. Saving a checkpoint file before removing outliers allows for a comparison of the data with and without outliers for a more comprehensive analysis.

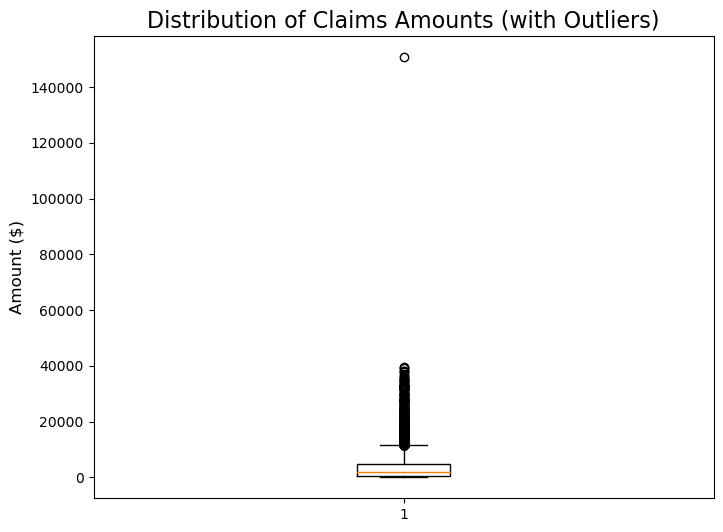
The pre-processing steps taken in this analysis have improved the quality of the data for accurate analysis. The removal of irrelevant columns, erroneous records, and outliers ensures that only accurate and meaningful data is used for analysis.

**Answer to Question 4**

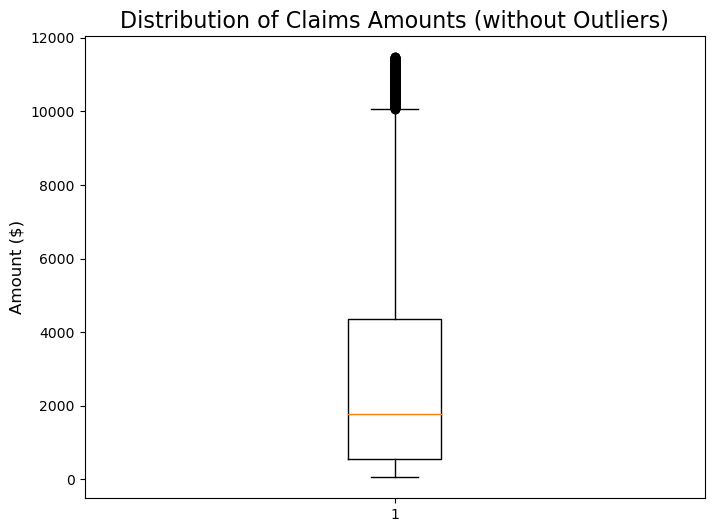
**Insight One - Comparing average claim amounts**

Understanding the average amount paid out per claim can help the insurance company better predict and manage their financial risks. Additionally, it may highlight areas where the company can improve its underwriting or claims handling practices to reduce the likelihood of high-cost claims. I went back to the initial 'ECA' source file and saw that there were only 2 claims records in December 2021. Instead of a huge jump in claims amount after year 2020, the better insight should be that the records are only gathered from late 2020 onwards. The best way to verify this is to check with the data team that provided the source file. Assuming that is the case, based on the analysis, we can see that there is not a significant difference in the average claims amount between 2021 and 2022. This could suggest that there may not have been any major changes in the nature or severity of claims filed between those two years.

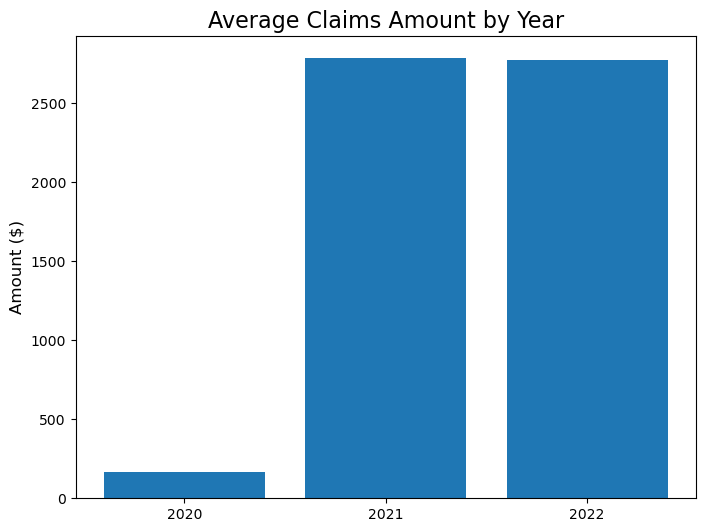
To visualize the insight, I first created box plots for both the skewed data and data with outliers removed. The first chart with outliers shows that the initial data is very skewed. The part of the box to the bottom of the median is much shorter than the part to the top of the median with many extreme values on top. This means the smaller claim amounts are much closer together than the larger amounts.



A more accurate claims average to derive for most claims, would be to use the pre-processed data without outliers. This box plot shows a relatively more symmetric shape with less skewness of claims amounts distribution.



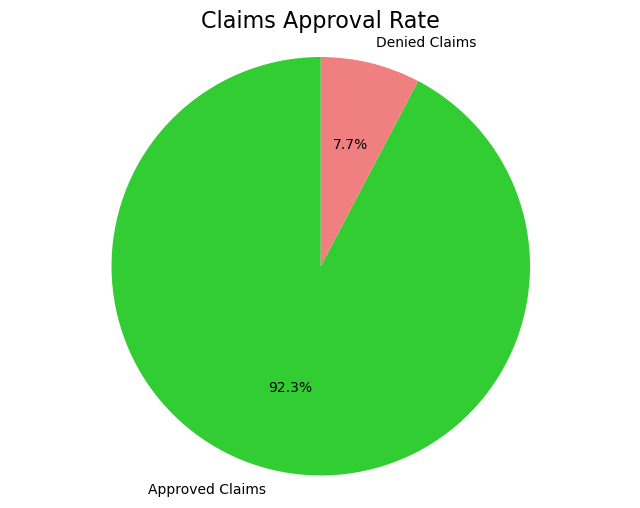
Subsequently, a bar chart showing the average claims amount for each year side-by-side.



**Insight Two - Comparing claim approval rates**

Claims approval rates: It would be useful to understand how many claims are approved and how many are denied, and the reasons for these outcomes. This information could help identify areas where the claims process can be improved, such as by increasing the clarity of policy terms or providing additional training to claims assessors.

To illustrate this, I computed the number of approved and denied claims, and generated a pie chart.



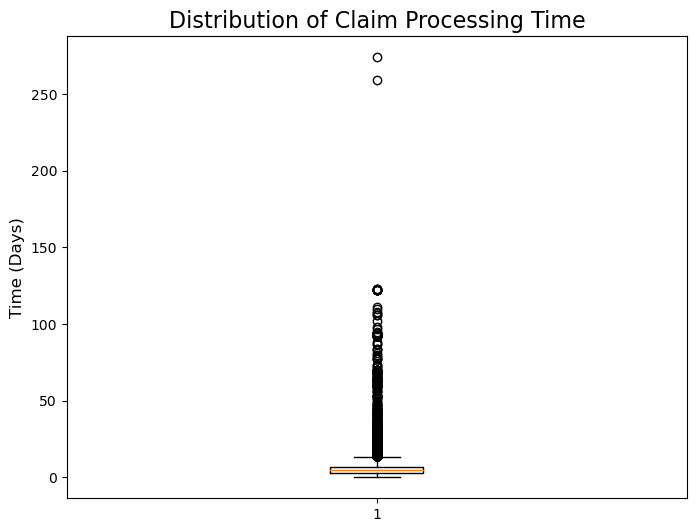
Based on the analysis of the claims dataset, it was observed that most of the claims (92.3%) were approved, while only a small proportion (7.7%) were denied. This insight may suggest that the claims review process is generally successful in identifying valid claims, and that claims are being processed in a timely and efficient manner.

**Insight Three - Claim processing time**

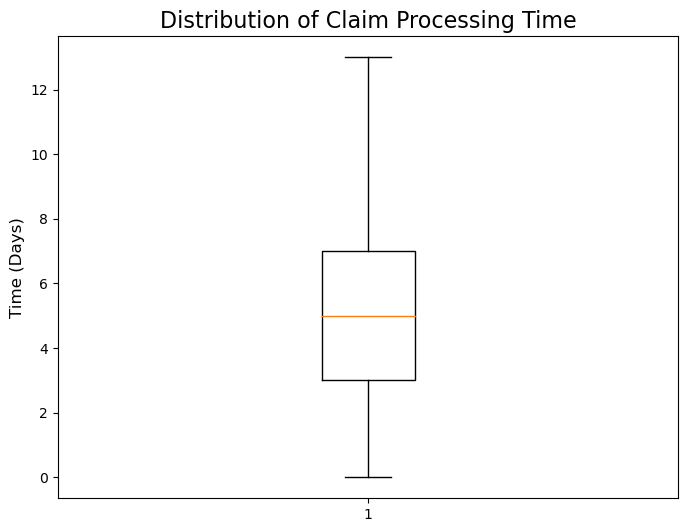
The time it takes to process and settle claims is an important factor in customer satisfaction. If claims take too long to process, it can lead to frustration and even loss of customers. Analysing claim processing times can help identify bottlenecks or inefficiencies in the claims process and help the insurance company take steps to streamline the process and improve customer satisfaction.

Analysing these insights could help the insurance company make data-driven decisions to improve their claims processing practices and ultimately increase customer satisfaction.

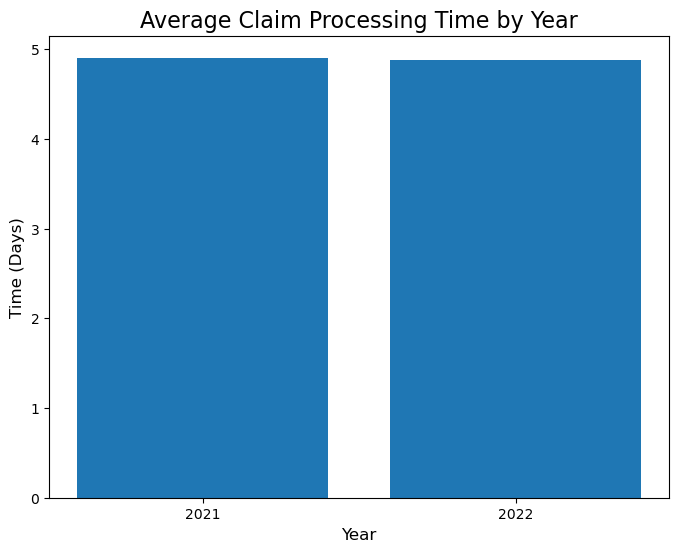
To process the data, I computed the time difference between claims created and settled, and had the outliers removed. Initially I observed very skewed data for the time difference between claims created and settled, by using a box plot. The part of the box to the bottom of the median is much shorter than the part to the top of the median, with many extreme values on top. This means the shorter time differences are much closer together than the longer time differences.



Next step was to remove the outliers. This box plot shows a relatively more symmetric shape with less skewness of time difference distribution.



The outliers include all the 2 records in year 2020, which results in no average figure to derive for 2020. I then used a bar chart to visualise the average processing time for each year.



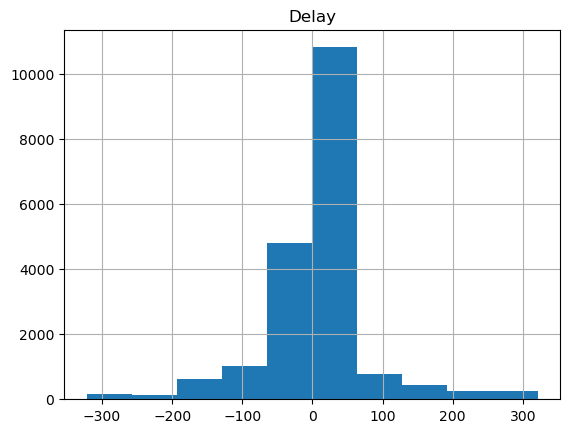
The average claim processing time for both 2021 and 2022 is relatively low at 4.91 and 4.88 days, respectively. This suggests that the insurance company has efficient claims processing procedures in place. However, to get a more complete understanding of the company's performance, it would be valuable to benchmark these results against industry standards. By comparing the average claim processing times to those of other insurance companies in the industry, the company can better assess its strengths and weaknesses and identify opportunities for improvement. Consistency in claim processing time can be an important factor in customer satisfaction and can contribute to building trust and loyalty with customers. If the insurance company can maintain a similar processing time in the future, it can help to build a positive reputation for the company and attract new customers.

**Answer to Question 5**

All relevant python libraries from scikit-learn and seaborn are imported to aid the analysis.

Earlier to answer Question 4, I changed ‘Created’, ‘Planned’, and ‘Actual’ that were initially object datatype to datetime64[ns] datatype. This helps facilitate the addition of a new column called 'Delay' by deducting planned claim settlement dates actual planned dates. ‘Delay’ will be the target variable “y”.

A history is generated to see how the values are distributed for ‘Delay’



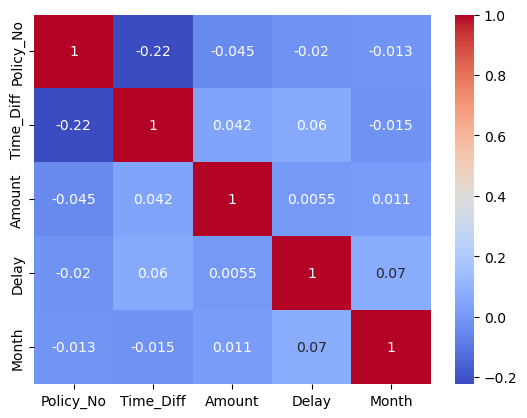
‘Policy\_No’ and ‘Name’ are unique identifiers which will not be used as features. These variables do not add any meaningful information to the model. Including unique identifiers as features may lead to overfitting, as the model may try to fit the noise present in these variables instead of the true signal in the data.

I will only be using data with processing time recorded for approved claims, because explained in an earlier question, the data did not record the time taken to process claims that were denied. Since all the values for 'Paid' will be "Yes', this will not be useful as a feature and will be dropped as well.

After checking for and removing null values, I converted the 'Created' into a float64 representing the months of the year, to use as one of the features to potentially capture any seasonality of claims volume that may exist in the data. ‘Month’ and ‘Amount’ are selected as the numeric variable features “x”, to be used in the linear regression model.

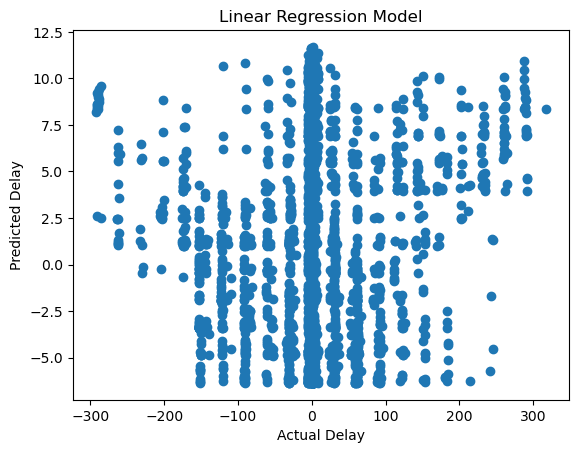
I then scaled the features using MinMaxScaler from scikit-learn, as good practice to prevent biased results, since the ‘Amount’ values contain a much larger maximum value and numeric range than the months of the year.

Next was to create a correlation matrix for all features. From there we can foretell that the model will be accurate at predicting claims processing delays, since there are no features that are closely related (close to 1) to ‘Delay’. They are closer to 0 instead.



The features and target variables are then split into training and testing sets, using 80% of data as test size and 20% as testing size. After training, testing, and fitting, the model is created. Accuracy of prediction is poor, as indicated by a R-squared score of close to zero.

The model is visualised as a scatter plot.



**Answer to Question 6**

As anticipated by the heatmap, the linear regression model does not accurately predict the processing time delay. The R-squared score is close to zero rather than close to one for greater accuracy. The scatter plot points are far from being considered as close to a diagonal line, indicating that the predicted values are not close to the actual values at all.

Nonetheless, we learned about the relationships between the target variable and the dataset's features. We now know that, in addition to the other variables, claims amount and claims volume are not good predictors of claims processing time delays. This suggests that there are other factors influencing delays that are not captured in the current dataset.

There are other factors that could influence claim processing time and help generate a better model to predict delays. Here are some examples:

Claim type - Different types of claims may necessitate different types of documentation or be processed differently, resulting in varying processing times.

Claim complexity - Complex claims may take longer to process because they require additional investigation or analysis.

Experience of claim adjusters - More experienced claim adjusters may be able to process claims more quickly and efficiently.

A decision tree model or other models may provide more meaningful insights because they are better suited to capture complex relationships between variables and identify important variables in a non-linear manner that a linear model cannot. Decision trees, for example, can divide data into categories based on the most important variables, making it easier to interpret the results and gain insights into the underlying factors that influence claims processing time.

*(2044 words)*

**Appendix**

**Code for all questions, copied from python file as pasted as text as follows:**

#!/usr/bin/env python

# coding: utf-8

# In[ ]:

**# Qn 1**

import pandas as pd

# Load the ECA dataset with null values defined

claims = pd.read\_csv('ECA.csv', na\_values=['Unkn', '???'])

# In[ ]:

claims.shape

# In[ ]:

# Find out which variables have missing values

missing\_mask = claims.isnull()

has\_missing = missing\_mask.any()

has\_missing[has\_missing == True]

# In[ ]:

**# Qn 2**

# Derive percentage of records with null values in the 'Actual' variable

null\_count = claims['Actual'].isnull().sum()

total\_count = len(claims)

percentage = (null\_count / total\_count) \* 100

print(f"Percentage of null values in 'Actual' variable: {percentage:.2f}%")

# In[ ]:

# Save a .csv file containing only the records with null values in 'Actual' variable

null\_records = claims[claims['Actual'].isnull()]

null\_records.to\_csv('null\_records.csv', index=False)

# Verify that the CSV file was created successfully

null\_records\_check = pd.read\_csv('null\_records.csv')

null\_records\_check.head()

# In[ ]:

# Clean the 'claims' dataframe by dropping rows with null values in 'Actual' variable, removing 'Claim\_ID' and 'Terms' columns

claims\_clean1 = claims.dropna(subset=['Actual']).drop(['Claim\_ID', 'Terms'], axis=1)

# In[ ]:

claims\_clean1.shape

# In[ ]:

# 1677 records (24213-22536 rows) with null values for 'Actual' removed from data

# 'Claim\_ID' and 'Terms' columns removed

# In[ ]:

# Find out if cleaned dataset have missing values now

missing\_mask = claims\_clean1.isnull()

has\_missing = missing\_mask.any()

has\_missing[has\_missing == True]

# In[ ]:

**# Qn 3**

# Task 1: remove irrelevant columns 'Category', 'Region', and 'Type'

claims\_clean2 = claims\_clean1.drop(['Category', 'Region', 'Type'], axis=1)

claims\_clean2

# In[ ]:

# irrelevant columns removed

# In[ ]:

# Task 2: Calculate time difference in claims created and claims settled

# Convert 'Created' and 'Actual' columns to same datetime format

claims\_clean2['Created'] = pd.to\_datetime(claims\_clean2['Created'], format='%Y%m%d')

claims\_clean2['Actual'] = pd.to\_datetime(claims\_clean2['Actual'], infer\_datetime\_format=True)

# Insert a new column called 'Time\_Diff' and calculate time difference between 'Actual' and 'Created' columns in days

claims\_clean2.insert(3, 'Time\_Diff', (claims\_clean2['Actual'] - claims\_clean2['Created']).dt.days)

# In[ ]:

claims\_clean2

# In[ ]:

# Some 'time\_diff' values are negative. Finding out how many records

# Percentage of records with negative values in the 'Time\_Diff'

negative\_time\_diff = claims\_clean2[claims\_clean2['Time\_Diff'] < 0]

num\_neg\_time\_diff = len(negative\_time\_diff)

percentage\_neg\_time\_diff = (num\_neg\_time\_diff / len(claims\_clean2)) \* 100

num\_neg\_time\_diff

# In[ ]:

print("Percentage of records with negative time difference values: {:.2f}%".format(percentage\_neg\_time\_diff))

# In[ ]:

# Save a .csv file containing only the records with negative values in 'Time\_Diff' variable

negative\_time\_diff.to\_csv('inaccurate\_dates.csv', index=False)

# Verify that the CSV file was created successfully

inaccurate\_dates\_check = pd.read\_csv('inaccurate\_dates.csv')

inaccurate\_dates\_check.head()

# In[ ]:

# Filter out records with negative time differences

claims\_clean3 = claims\_clean2[claims\_clean2['Time\_Diff'] >= 0]

claims\_clean3

# In[ ]:

# erroneous records removed

# In[ ]:

# Task 3: Prepping the 'Amount column'

# clean invalid values and identify outliers

# find out the data types

claims\_clean3.dtypes

# In[ ]:

# convert 'Amount' from string to numeric datatype

# create a boolean mask

mask = claims\_clean3['Amount'].apply(lambda x: not str(x).replace('.','',1).isdigit())

# use .loc with the boolean mask to update the 'Amount' column

claims\_clean3.loc[mask, 'Amount'] = pd.to\_numeric(claims\_clean3.loc[mask, 'Amount'], errors='coerce')

# In[ ]:

null\_mask = pd.to\_numeric(claims\_clean3['Amount'], errors='coerce').isnull()

null\_values = claims\_clean3[null\_mask]

null\_values

# In[ ]:

# cross-referenced the dataset for this entry

# found the invalid value of 1792.OO; Letters 'O' instead of numbers '0'

# amended the invalid value in the source file claims\_clean3 directly,

# and checkpoint saved it as claims\_clean4.csv

# another dataset saved and specified with outliers included

claims\_clean4 = pd.read\_csv('claimsclean4.csv')

claims\_clean4.to\_csv('claimsclean4\_outliers\_included.csv', index=False)

claims\_clean4.head()

# In[ ]:

#double-check to make sure all null values removed

null\_mask = pd.to\_numeric(claims\_clean4['Amount'], errors='coerce').isnull()

null\_values = claims\_clean4[null\_mask]

null\_values

# In[ ]:

# find out the data types again. Should show as float64

claims\_clean4.dtypes

# In[ ]:

# identify and remove outliers in the 'Amount' variable

q1 = claims\_clean4['Amount'].quantile(0.25)

q3 = claims\_clean4['Amount'].quantile(0.75)

iqr = q3 - q1

upper\_bound = q3 + 1.5\*iqr

lower\_bound = q1 - 1.5\*iqr

# identify and remove outliers in the 'Amount' variable

q1 = claims\_clean4['Amount'].quantile(0.25)

q3 = claims\_clean4['Amount'].quantile(0.75)

iqr = q3 - q1

upper\_bound = q3 + 1.5\*iqr

lower\_bound = q1 - 1.5\*iqr

# Save the final dataset, with final tasks implemented, as 'claims\_prepped'

claims\_prepped = claims\_clean4[(claims\_clean4['Amount'] >= lower\_bound) & (claims\_clean4['Amount'] <= upper\_bound)] = claims\_clean4[(claims\_clean4['Amount'] >= lower\_bound) & (claims\_clean4['Amount'] <= upper\_bound)]

# check for shape after outliers are removed

claims\_prepped.shape

# In[ ]:

# Checkpoint. Save dataset with outliers removed

claims\_prepped.to\_csv('claims\_prepped.csv', index=False)

# In[ ]:

**# Qn 4**

# Insight 1: Average claims amount, with and without outliers

import matplotlib.pyplot as plt

# In[ ]:

# Load the data with outliers included

claims\_with\_outliers = pd.read\_csv('claimsclean4\_outliers\_included.csv')

# Create a box plot to visualize the distribution of claims amounts

fig, ax = plt.subplots(figsize=(8, 6))

ax.boxplot(claims\_with\_outliers['Amount'])

ax.set\_title('Distribution of Claims Amounts (with Outliers)', fontsize=16)

ax.set\_ylabel('Amount ($)', fontsize=12)

plt.show()

# In[ ]:

# Calculate and print the mean and median claims amount from the claims\_prepped dataset

mean\_amount = claims\_with\_outliers['Amount'].mean()

median\_amount = claims\_with\_outliers['Amount'].median()

print(f"Mean claims amount (with outliers): ${mean\_amount:.2f}")

print(f"Median claims amount (with outliers): ${median\_amount:.2f}")

# In[ ]:

# Comparing that to preprocessed data with outliers removed

# Create a box plot to visualize the distribution of claims amounts in the claims\_prepped dataset

fig, ax = plt.subplots(figsize=(8, 6))

ax.boxplot(claims\_prepped['Amount'])

ax.set\_title('Distribution of Claims Amounts (without Outliers)', fontsize=16)

ax.set\_ylabel('Amount ($)', fontsize=12)

plt.show()

# In[ ]:

# Calculate and print the mean and median claims amount from the claims\_prepped dataset without outliers

mean\_amount\_without\_outliers = claims\_prepped['Amount'].mean()

median\_amount\_without\_outliers = claims\_prepped['Amount'].median()

print(f"Mean claims amount (without outliers): ${mean\_amount\_without\_outliers:.2f}")

print(f"Median claims amount (without outliers): ${median\_amount\_without\_outliers:.2f}")

# In[ ]:

claims\_prepped.dtypes

# In[ ]:

# Convert 'Created', 'Planned', and 'Actual' to datetime formats

mask = claims\_prepped['Created'].apply(lambda x: not isinstance(x, pd.Timestamp))

claims\_prepped.loc[mask, 'Created'] = claims\_prepped.loc[mask, 'Created'].apply(lambda x: pd.to\_datetime(x))

mask = claims\_prepped['Planned'].apply(lambda x: not isinstance(x, pd.Timestamp))

claims\_prepped.loc[mask, 'Planned'] = claims\_prepped.loc[mask, 'Planned'].apply(lambda x: pd.to\_datetime(x))

mask = claims\_prepped['Actual'].apply(lambda x: not isinstance(x, pd.Timestamp))

claims\_prepped.loc[mask, 'Actual'] = claims\_prepped.loc[mask, 'Actual'].apply(lambda x: pd.to\_datetime(x))

# In[ ]:

claims\_prepped.dtypes

# In[ ]:

# Calculate the average claims amount for each year

avg\_amount\_2020 = claims\_prepped[claims\_prepped['Created'].dt.year == 2020]['Amount'].mean()

avg\_amount\_2021 = claims\_prepped[claims\_prepped['Created'].dt.year == 2021]['Amount'].mean()

avg\_amount\_2022 = claims\_prepped[claims\_prepped['Created'].dt.year == 2022]['Amount'].mean()

# Create a bar chart showing the average claims amount for each year

fig, ax = plt.subplots(figsize=(8, 6))

ax.bar(['2020', '2021', '2022'], [avg\_amount\_2020, avg\_amount\_2021, avg\_amount\_2022])

ax.set\_title('Average Claims Amount by Year', fontsize=16)

ax.set\_ylabel('Amount ($)', fontsize=12)

plt.show()

# In[ ]:

# Insight 2: Claims approval rate

# Count the number of approved claims in the prepped dataset

approved\_claims = len(claims\_prepped[claims\_prepped['Paid'] == 'Yes'])

# Count the number of denied claims in the null records dataset

denied\_claims = len(null\_records[null\_records['Paid'] == 'No'])

# Ensure that the number of approved claims equals the number of 'Yes' values in the 'Paid' column of the prepped dataset

assert approved\_claims == len(claims\_prepped[claims\_prepped['Paid'] == 'Yes']), "Number of approved claims does not match the number of 'Yes' values in the 'Paid' column of the prepped dataset."

# Ensure that the number of denied claims equals the number of 'No' values in the 'Paid' column of the null records dataset

assert denied\_claims == len(null\_records[null\_records['Paid'] == 'No']), "Number of denied claims does not match the number of 'No' values in the 'Paid' column of the null records dataset."

# Print the number of approved and denied claims

print(f"Number of approved claims: {approved\_claims}")

print(f"Number of denied claims: {denied\_claims}")

# In[ ]:

import matplotlib.pyplot as plt

# create a list of labels and values for approved and denied claims

labels = ['Approved Claims', 'Denied Claims']

values = approved\_claims, denied\_claims

# create a pie chart with percentages and colors for each segment

fig, ax = plt.subplots(figsize=(8, 6))

colors = ['#32CD32', '#F08080'] # lime green and light coral colors

ax.pie(values, labels=labels, autopct='%1.1f%%', startangle=90, colors=colors)

ax.axis('equal') # Equal aspect ratio ensures that pie is drawn as a circle.

ax.set\_title('Claims Approval Rate', fontsize=16)

plt.show()

# In[ ]:

# Insight 3. Claim Processing Time.

import matplotlib.pyplot as plt

# Create a box plot of the claim processing time

fig, ax = plt.subplots(figsize=(8, 6))

ax.boxplot(claims\_prepped['Time\_Diff'])

ax.set\_title('Distribution of Claim Processing Time', fontsize=16)

ax.set\_ylabel('Time (Days)', fontsize=12)

plt.show()

# In[ ]:

# Data is very skewed. Removing outliers

# Calculate the interquartile range

Q1 = claims\_prepped['Time\_Diff'].quantile(0.25)

Q3 = claims\_prepped['Time\_Diff'].quantile(0.75)

IQR = Q3 - Q1

# Filter out outliers

claims\_prepped2 = claims\_prepped[(claims\_prepped['Time\_Diff'] >= Q1 - 1.5\*IQR) & (claims\_prepped['Time\_Diff'] <= Q3 + 1.5\*IQR)]

# Checkpoint

claims\_prepped2.to\_csv('claims\_prepped2.csv', index=False)

# In[ ]:

import matplotlib.pyplot as plt

# Create a box plot of the claim processing time, with outliers removed

fig, ax = plt.subplots(figsize=(8, 6))

ax.boxplot(claims\_prepped2['Time\_Diff'])

ax.set\_title('Distribution of Claim Processing Time', fontsize=16)

ax.set\_ylabel('Time (Days)', fontsize=12)

plt.show()

# In[ ]:

# Calculate the average claim processing time for each year

avg\_processing\_time\_2020 = claims\_prepped2[claims\_prepped2['Created'].dt.year == 2020]['Time\_Diff'].mean()

avg\_processing\_time\_2021 = claims\_prepped2[claims\_prepped2['Created'].dt.year == 2021]['Time\_Diff'].mean()

avg\_processing\_time\_2022 = claims\_prepped2[claims\_prepped2['Created'].dt.year == 2022]['Time\_Diff'].mean()

# Print the results

print("Average claim processing time for 2020: {:.2f} days".format(avg\_processing\_time\_2020))

print("Average claim processing time for 2021: {:.2f} days".format(avg\_processing\_time\_2021))

print("Average claim processing time for 2022: {:.2f} days".format(avg\_processing\_time\_2022))

# In[ ]:

import matplotlib.pyplot as plt

# Create a bar chart of the average claim processing time for each year

x = ['2020', '2021', '2022']

y = [avg\_processing\_time\_2020, avg\_processing\_time\_2021, avg\_processing\_time\_2022]

fig, ax = plt.subplots(figsize=(8, 6))

ax.bar(x, y)

ax.set\_title('Average Claim Processing Time by Year', fontsize=16)

ax.set\_xlabel('Year', fontsize=12)

ax.set\_ylabel('Time (Days)', fontsize=12)

plt.show()

# In[ ]:

**# Qn 5.** linear regression modelling to predict the delay in days

# between the 'Planned' and 'Actual' dates

# import relevant libraries first

from sklearn.preprocessing import MinMaxScaler

from sklearn.linear\_model import LinearRegression

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

from sklearn.metrics import r2\_score

import seaborn as sns

# In[ ]:

claims\_prepped2.dtypes

# In[ ]:

# create a new column called 'Delay' in the 'claims\_prepped2' dataframe,

# which will be the difference in days between the 'Actual' and 'Planned' dates

# we will use the 'dt.days' attribute of the datetime object to get the difference in days

claims\_prepped2.loc['Delay'] = (claims\_prepped2['Actual'] - claims\_prepped2['Planned']).dt.days

# check if the 'Delay' column has been added

claims\_prepped2.head()

# In[ ]:

claims\_prepped2.dtypes

# In[ ]:

# create histogram to see the distribution of 'Delay'

claims\_prepped2.hist(column='Delay')

# In[ ]:

# check for any rows with missing values in the Delay column

claims\_prepped2['Delay'].isnull().values.any()

# In[ ]:

# removed missing values and created a copy to continue processing

claims\_prepped\_copy = claims\_prepped2.copy()

claims\_prepped\_copy.dropna(subset=['Delay'], inplace=True)

# In[ ]:

# Create a 'Month' column from the months 1-12 in 'Created' column

# checkpoint before further processing

claims\_prepped\_copy['Month'] = claims\_prepped\_copy['Created'].dt.month

claims\_prepped\_copy.to\_csv('claims\_prepped\_copy.csv', index=False)

# select the 'Amount' and 'Month' columns for scaling

cols\_to\_scale = ['Amount', 'Month']

data\_to\_scale = claims\_prepped\_copy[cols\_to\_scale]

# scale the selected columns using MinMaxScaler

scaler = MinMaxScaler()

scaled\_data = scaler.fit\_transform(data\_to\_scale)

# replace the original columns with the scaled data

claims\_prepped\_copy[['Amount', 'Month']] = scaled\_data

# In[ ]:

claims\_prepped\_copy.head()

# In[ ]:

# create a correlation matrix for all features

corr\_matrix = claims\_prepped\_copy.corr()

# plot the correlation matrix as a heatmap

sns.heatmap(corr\_matrix, annot=True, cmap='coolwarm')

# In[ ]:

# Split the data into training and testing sets

X = claims\_prepped\_copy[['Amount', 'Month']]

y = claims\_prepped\_copy['Delay']

# Training, testing, adn fitting the data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=0)

lr = LinearRegression()

lr.fit(X\_train, y\_train)

# Make predictions on the testing set and calculate R-squared score

y\_pred = lr.predict(X\_test)

score = r2\_score(y\_test, y\_pred)

print('R-squared score:', score)

# In[ ]:

# visualize the linear regression model

# plot the predicted values against the actual values

plt.scatter(y\_test, y\_pred)

plt.xlabel('Actual Delay')

plt.ylabel('Predicted Delay')

plt.title('Linear Regression Model')

plt.show()