

**ANL252**

**Python for Data Analytics**

**End-of-Course Assessment**

**January 2023 Presentation**

|  |  |
| --- | --- |
| TUTORIAL GROUP | T05 |
| STUDENT | LEONA INEZ TAN SHENG GUAT |
| PI NUMBER | J2082686 |
| SUBMISSION DATE | 06/30/2023 |

|  |
| --- |
| *I hereby declare that this assignment is my own work, unless otherwise acknowledged or credited by appropriate reference, I have read and abide by the SUSS Honour Code and I am aware of the penalties associated with plagiarism and collusion listed in the SUSS Student Handbook.* |

**Question 1**

Using the dataset provided, it is assumed that the missing data in the dataset are denoted buy the strings ‘Unkn’ and ‘???’. To identify which cells in the dataframe include these values, we utilize the isin() function (GeeksforGeeks, 2018). After that, to determine whether each column has any missing values, we use the any() function (GeeksforGeeks, 2022). Lastly, we print the column names that contain missing values.

When the code is run, the output is as follows. This means that the ‘Terms’ column contain missing values.

**Output**

Index ([‘Terms’], dtype =’object’)

**Question 2**

To treat missing values in a dataset is largely dependent on the type and extent of the missing values. There are 3 common methods in which we can go about handling the missing data which are deletion, imputation, and ignoring the missing values (Yildirim, 2021). Deletion involves removing rows with missing values, but this may lead to loss of information, especially if many values are missing. Imputation involves filling in the missing values with estimated values based on other observations in the dataset.

However, in this case, the meaning of each value for the internal codes of the organization is unknown, it may be difficult to determine the appropriate treatment method. Thus, treating the variables as nominal or categorical and using imputation methods such as mode imputation or hot deck imputation can be effective.

A common way to fill in missing values for categorical variables is by using the mode, which is the most frequently occurring value in the dataset (Narang, 2023). This means that the missing values are replaced with the value that appears most often in the dataset for that particular variable. To find the mode in the “Terms” column, we can use the following code (Refer to Appendix 2). The output will be “AD23”. Thus, mode imputation will be used by replacing the missing values with “AD23” as it is often better than deleting or removing data because it helps preserve the sample size and hence, can result in a more accurate analysis of the data.

**Question 3**

**Data Cleaning**

Data cleaning is an important step in preparing data for analysis. Essentially, it involves going through the data to identify and correct any errors, inconsistencies, or irrelevant information (GeeksforGeeks, 2023). This is important because it helps to ensure that the data is accurate and of high quality, which in turn can help to produce more accurate analysis and results.

By removing duplicates, filling in missing data, and correcting errors, we can make sure that the data is complete and consistent. This can make it easier to analyze and identify patterns or insights that might not have been visible otherwise.

Ultimately, having clean data can make decision-making more effective by providing accurate and reliable information.

**Data Transformation**

In this code example (Refer to Appendix 3.2), the data is transformed by calculating the time taken to settle claims, extracting the month and year from the claim settlement date, and grouping the data by policy type and month to sum the claim amount. This transformation is useful in a corporate claims processing context because it allows for the preparation and organization of data in a way that is more conducive to analysis and decision-making (GeeksforGeeks, 2023). By performing these operations, data can be transformed from raw records into useful insights that can make informed business decisions and strategies. The resulting grouped dataframe is saved to a new CSV file called 'ECA\_transformed.csv' using the .to\_csv() method.

**Data Aggregation**

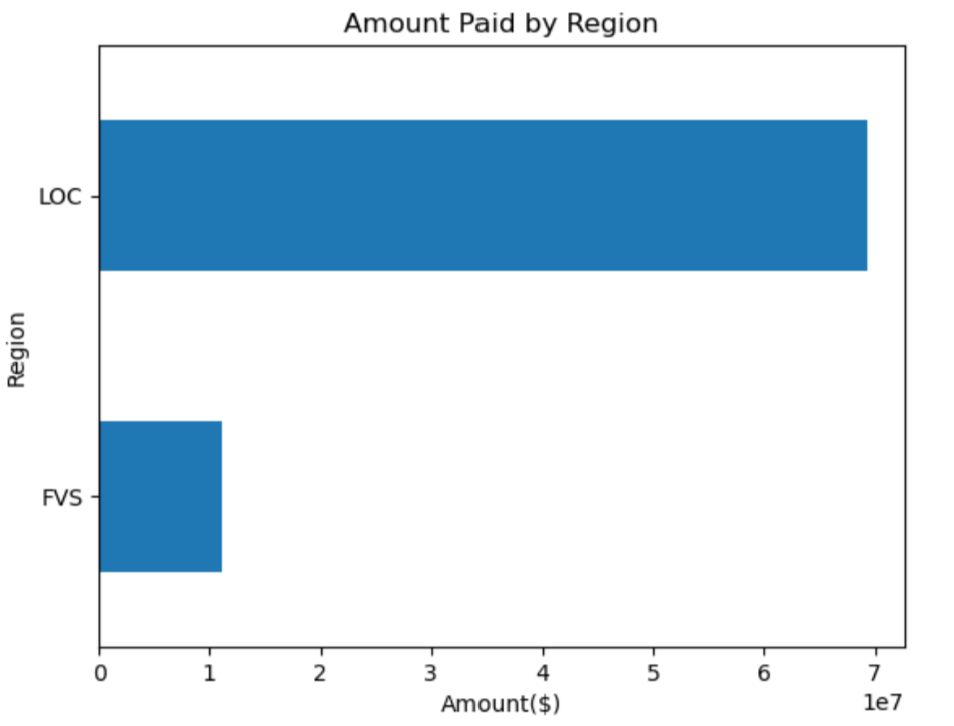
Data aggregation can also be useful for insurance corporate claims processing. Aggregation involves summarizing or grouping data together to create higher-level overviews or insights (GeeksforGeeks, 2021). For example, the insurance company may want to analyze the total number of claims filed per month or quarter (Refer to Appendix 3.3). By aggregating the data, they can see that the highest number of claims were filed in May 2021 (1915 claims), followed by April 2021 (1704 claims). The number of claims gradually decreased from June 2021 to August 2022, with only 9 claims being filed in August 2022. Then, the number of claims increased again from September 2022 to October 2022, with the highest number of claims being filed in October 2022 (156 claims). Aggregating the data can help to reveal patterns or trends that may not be apparent when looking at individual records (GeeksforGeeks, 2021).

**Question 4**

**Insight 1: Comparison of amount paid by the regions**

An interesting part of the dataset provided for corporate claims processing is the amount paid by region. This metric can help the insurance company identify which regions have higher or lower claim pay-outs, and could help in assessing risk for future policies. We can visualize the amount paid by region using a horizontal bar chart. This will show the amount paid for each region, which can help us identify which regions have higher or lower claim pay-outs. In this case, the LOC region has a higher claims pay-out as can be seen in the chart below.

**Visualization 1: Horizontal Bar Chart**

****

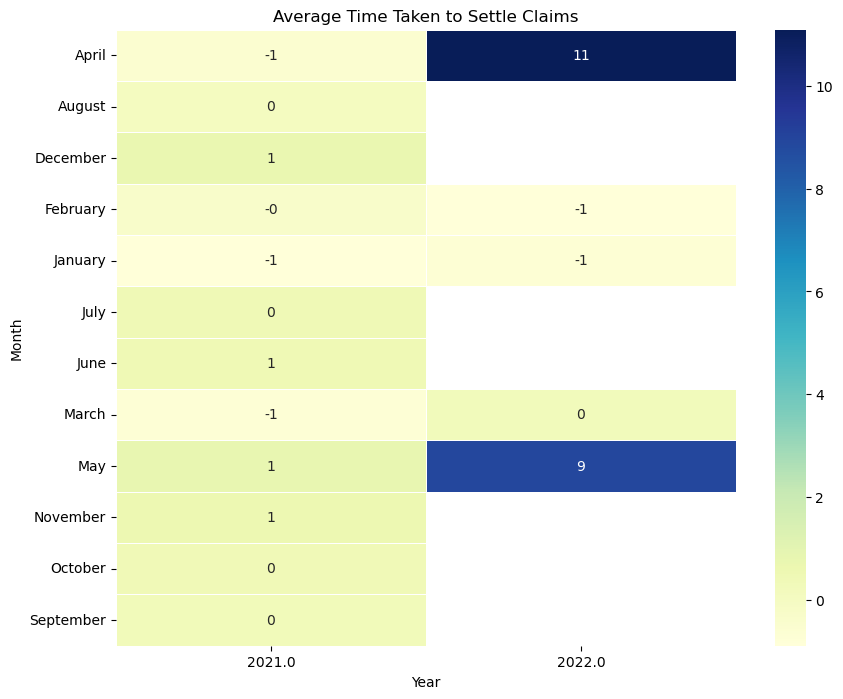
**\*See Appendix 4.1 for the code**

**Insight 2: Trends in the time taken to settle claims over the years**

The heatmap displays the average time taken to settle claims in an insurance company per month and year (GeeksforGeeks, 2020). The color of each cell represents the average time taken to settle claims for a specific month and year. The darker the color, the longer it takes to settle claims on average. This helps in identifying patterns or trends in claim settlement time. By analyzing the heatmap, we can identify the years and months with the longest average settlement time and areas that need improvement.

The color range of the heatmap represents the magnitude of the values in the matrix. In the code provided, the heatmap color range is represented by the **cmap** parameter, which is set to **"YlGnBu"** (Holtz, 2018). This color map goes from yellow (low values) to green (medium values) to blue (high values). The intensity of the color indicates the magnitude of the value. In the code, the darker the blue, the higher the **Time Taken to Settle** value is for a given month and year.

**Visualization 2: Heatmap**

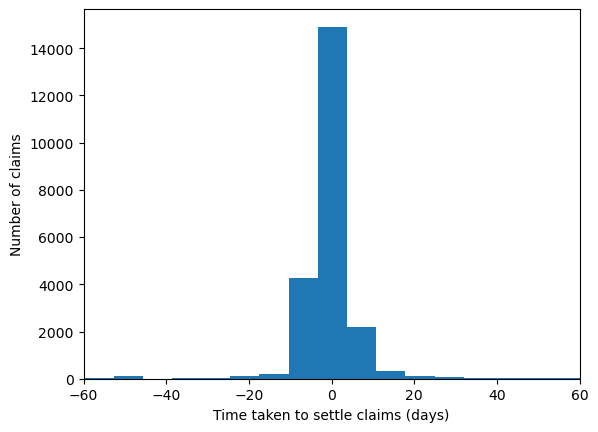


**\*See Appendix 4.2 for the code**

**Insight 3: Comparison of Time taken for claim settlements**

Histograms reveal patterns and trends in settlement times, identifying areas for improvement like the most common time to settle claims and any outliers. This data can help streamline processes, reduce delays, and address bottlenecks. Since the histogram follows a bell-curved distribution, it suggests that most claims are settled within a narrow range of time, with fewer claims taking either a very short or very long time to settle (GeeksforGeeks, 2022). If the most common settlement time is 0 days, it could indicate that a large portion of claims are settled quickly, with few delays or issues. By identifying trends, patterns and spotting outliers in settlement times, the company can improve their claims processing procedures and efficiency, reduce delays, and improve customer satisfaction.

An insight obtained from the histogram is that the time to settle a claim can be a positive or negative value, with negative values indicating efficient claims processing and positive values indicating inefficiencies or documentation challenges. Some claims had negative values, such as -14,000 days (found using the Excel function [IFERROR (DATEDIF (D2, E2,"d"),"-")], see Appendix 4.4)., which is impossible and requires data review and outlier investigation. Similarly, claims with time taken to settle value of +50 days indicate delayed settlements due to processing delays, disputes, or backlog of claims, which should be analyzed to identify areas for improvement.

**Histogram**

**\*See Appendix 4.3 for the code**

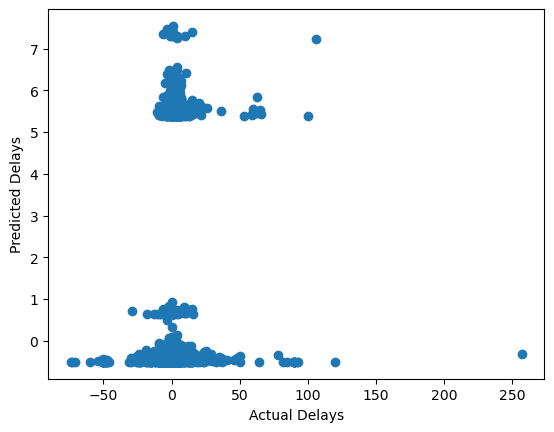
**Question 5**

The linear regression model was used to predict the delay in days between the Planned and Actual date in processing the claims. The approach taken includes data pre-processing, which involved converting the 'Amount' column to numeric values and dropping any rows with missing values. The delay in days was then calculated as the difference between the 'Actual' and 'Planned' dates.

The data was split into a training set (80% of the data) and a testing set (20% of the data). The preprocessing steps were defined for numeric and categorical data, including scaling and one-hot encoding using the ColumnTransformer. A pipeline was created to combine the preprocessing steps with the linear regression model.

The training data was utilized to fit the model, which was then employed to forecast the delays on the testing data. The mean squared error was calculated to be 92.47, indicating that the model's predicted delays were not very accurate. The R-squared value was 0.026, suggesting that the model explains only a small proportion of the variance in the data.

These results indicate that the linear regression model is not a very good fit for the data, and further analysis is needed to identify a more suitable model. It is also possible that additional data, such as information on the claimant's demographic characteristics or the type of claim being processed, could be used to improve the model's performance. (Refer to Appendix 5)



**Question 6**

The linear regression model predicts the delay in days based on several variables including the type of claim and the region of the claim.

Looking at the coefficients, we can see that the type of claim has a significant impact on the predicted delay. For example, a claim of type L002 is expected to have a delay that is 5.3304 days longer than a claim of type Q001, all else being equal. Similarly, a claim of type L005 is expected to have a delay that is 4.1712 days longer than a claim of type Q001, all else being equal.

The region of the claim also has a significant impact on the predicted delay, with both Region\_FVS and Region\_LOC having a coefficient of 0.1889. This means that if a claim is located in either of these regions, we would expect the predicted delay to be 0.1889 days longer than if the claim was located in another region, all else being equal.

Overall, the R-squared value of 0.0263 indicates that the model explains only a small portion of the variance in the data. This means that there are likely other factors that are not included in the model that also contribute to the delay in processing claims. The mean squared error of 92.47 indicates that the model has a relatively high level of error in its predictions. Therefore, while the model provides some insights into the factors that contribute to delays, its predictive power may be limited.

**Linear Regression Equation**

Delay = 0.1266(Type\_L001) - 5.3304(Type\_L002) + 2.4125(Type\_L003) + 2.7490(Type\_L004) + 4.1712(Type\_L005) - 4.1913(Type\_Q001) + 0.1889(Region\_FVS) + 0.1889(Region\_LOC) + 5.1086

Where:

* Type\_L001, Type\_L002, Type\_L003, Type\_L004, Type\_L005, and Type\_Q001 are binary variables indicating the type of claim.
* Region\_FVS and Region\_LOC are binary variables indicating the region of the claim.
* Delay is the predicted delay in days.
* The coefficients represent the amount by which the predicted delay changes for a one-unit increase in the corresponding variable. For example, if the type of claim is changed from L002 to L003, then it would be expected for the predicted delay to increase by 2.4125 days, all else being equal. The intercept term of 5.1086 represents the predicted delay for a claim that is of type O001, located in the LOC region, and has an amount of zero.

**References**

Wu, K. Y. (2022). *ANL252 Python for data analytics (study guide)*. Singapore University of Social Sciences.

GeeksforGeeks (2018) *Python: Pandas dataframe.isin()*, *GeeksforGeeks*. GeeksforGeeks. Available at: https://www.geeksforgeeks.org/python-pandas-dataframe-isin/ (Accessed: March 5, 2023).

GeeksforGeeks (2022) *Python any() function*, *GeeksforGeeks*. GeeksforGeeks. Available at: https://www.geeksforgeeks.org/python-any-function/ (Accessed: March 5, 2023).

Narang, M. (2023) *Handling missing data: Mean, Median, Mode*, *Naukri.com*. Available at: https://www.naukri.com/learning/articles/handling-missing-data-mean-median-mode/#Mean/-Median-/Mode-imputation (Accessed: March 5, 2023).

w3schools (2023) *Pandas DataFrame mode() Method*, *Pandas dataframe mode() method*. Available at: https://www.w3schools.com/python/pandas/ref\_df\_mode.asp (Accessed: March 5, 2023).

Yildirim, S. (2021) *8 methods for handling missing values with python pandas*, *Medium*. Towards Data Science. Available at: https://towardsdatascience.com/8-methods-for-handling-missing-values-with-python-pandas-842544cdf891 (Accessed: March 6, 2023).

GeeksforGeeks (2023) *ML: Overview of data cleaning*, *GeeksforGeeks*. GeeksforGeeks. Available at: https://www.geeksforgeeks.org/data-cleansing-introduction/ (Accessed: March 6, 2023).

Ebner, J. and Anusha, K. (2021) *How to use the pandas dropna method*, *Sharp Sight*. Available at: https://www.sharpsightlabs.com/blog/pandas-dropna/ (Accessed: March 5, 2023).

GeeksforGeeks (2022) *Python: Pandas dataframe.drop\_duplicates()*, *GeeksforGeeks*. GeeksforGeeks. Available at: https://www.geeksforgeeks.org/python-pandas-dataframe-drop\_duplicates/ (Accessed: March 5, 2023).

GeeksforGeeks (2018) *Python: Pandas dataframe.select\_dtypes()*, *GeeksforGeeks*. GeeksforGeeks. Available at: https://www.geeksforgeeks.org/python-pandas-dataframe-select\_dtypes/ (Accessed: March 6, 2023).

GeeksforGeeks (2023) *Data transformation in Data Mining*, *GeeksforGeeks*. GeeksforGeeks. Available at: https://www.geeksforgeeks.org/data-transformation-in-data-mining/ (Accessed: March 6, 2023).

GeeksforGeeks (2021) *Grouping and aggregating with pandas - geeksforgeeks*, *Grouping and Aggregating with Pandas*. Available at: https://www.geeksforgeeks.org/grouping-and-aggregating-with-pandas/ (Accessed: March 5, 2023).

GeeksforGeeks (2022) *Python any() function*, *GeeksforGeeks*. GeeksforGeeks. Available at: https://www.geeksforgeeks.org/python-any-function/ (Accessed: March 5, 2023).

GeeksforGeeks (2020) *Seaborn Heatmap - A comprehensive guide*, *GeeksforGeeks*. GeeksforGeeks. Available at: https://www.geeksforgeeks.org/seaborn-heatmap-a-comprehensive-guide/ (Accessed: March 6, 2023).

Mohammad Shahid (2023) *Handy Python pandas for Data Aggregation*, *Kaggle*. Kaggle. Available at: https://www.kaggle.com/code/itsmohammadshahid/handy-python-pandas-for-data-aggregation (Accessed: March 6, 2023).

Real Python (2022) *Linear regression in python*, *Real Python*. Real Python. Available at: https://realpython.com/linear-regression-in-python/ (Accessed: March 6, 2023).

Real Python (2021) *Split your dataset with scikit-learn's train\_test\_split()*, *Real Python*. Real Python. Available at: https://realpython.com/train-test-split-python-data/ (Accessed: March 6, 2023).

Holtz, Y. (2018) *Control color in Seaborn heatmaps*, *The Python Graph Gallery*. Available at: https://www.python-graph-gallery.com/92-control-color-in-seaborn-heatmaps (Accessed: March 6, 2023).

GeeksforGeeks (2022) *Interpretations of histogram*, *GeeksforGeeks*. GeeksforGeeks. Available at: https://www.geeksforgeeks.org/interpretations-of-histogram/ (Accessed: March 6, 2023).

GeeksforGeeks (2022) *Plotting histogram in python using Matplotlib*, *GeeksforGeeks*. GeeksforGeeks. Available at: https://www.geeksforgeeks.org/plotting-histogram-in-python-using-matplotlib/ (Accessed: March 6, 2023).

**Appendix**

**Appendix 1: Code for Question 1**

import pandas as pd

# Read in the dataset

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

# Print the columns that contain missing values

print(df.columns[df.isin(['Unkn', '???','blanks']).any()])

**Appendix 2: Code for Question 2**

import pandas as pd

# Read in the dataset

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

# Create a DataFrame

data = {'Terms'}

# Find the mode of the 'Terms' column

mode = df['Terms'].mode()[0]

print("The mode is:", mode)

**Codes for Question 3**

**Appendix 3.1: Code for Data Cleaning**

import pandas as pd

# Load the dataset into a Pandas dataframe

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

# Remove duplicate rows from the dataset

df.drop\_duplicates(inplace=True)

# Replace missing data with the mean of the column (numeric columns only)

num\_cols = df.select\_dtypes(include='number').columns

df[num\_cols] = df[num\_cols].fillna(df[num\_cols].mean())

# Convert non-numeric values to NaN

df = df.apply(pd.to\_numeric, errors='coerce')

# Remove rows with missing data

df.dropna(inplace=True)

**Appendix 3.2: Code for Data Transformation**

import pandas as pd

# Load the data

df = pd.read\_csv('ECA.csv', parse\_dates=['Actual', 'Planned'], infer\_datetime\_format=True)

# Calculate the time taken to settle claims

df['Time Taken to Settle'] = (df['Actual'] - df['Planned']).dt.days

# Extract month and year from the date

df['Month'] = df['Actual'].dt.month\_name()

df['Year'] = df['Actual'].dt.year

# Group the data by policy type and month, and sum the claim amount

grouped\_df = df.groupby(['Type', 'Month'])['Amount'].sum().reset\_index()

grouped\_df.to\_csv('ECA\_transformed.csv', index=False)

**Appendix 3.3: Code for Data Aggregation**

import pandas as pd

# Load the dataset into a Pandas dataframe

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

# Convert the date column to a datetime datatype

df['Actual'] = pd.to\_datetime(df['Actual'])

# Group the data by month and count the number of claims filed

monthly\_claims = df.groupby(pd.Grouper(key='Actual', freq='M')).count()

# Print the results

print(monthly\_claims['Claim\_ID'])

**Codes for Question 4**

**Appendix 4.1: Code for Visualization 1**

import pandas as pd

import matplotlib.pyplot as plt

# Load the corporate claims processing dataset

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

# Convert the amount column to numeric

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

# Group the data by region and sum the amount paid for each region

region\_totals = df.groupby('Region')['Amount'].sum()

# Create a horizontal bar chart

fig, ax = plt.subplots()

region\_totals.plot(kind='barh', ax=ax)

# Add labels and title

ax.set\_xlabel('Amount($)')

ax.set\_ylabel('Region')

ax.set\_title('Amount Paid by Region')

plt.show()

**Appendix 4.2: Code for Visualization 2**

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

# Load the data

df = pd.read\_csv('ECA.csv', parse\_dates=['Actual', 'Planned'], infer\_datetime\_format=True)

# Calculate the time taken to settle claims

df['Time Taken to Settle'] = (df['Actual'] - df['Planned']).dt.days

# Extract month and year from the date

df['Month'] = df['Actual'].dt.month\_name()

df['Year'] = df['Actual'].dt.year

# Pivot the data to create a matrix

matrix = df.pivot\_table(index='Month', columns='Year', values='Time Taken to Settle')

# Create the heatmap

plt.figure(figsize=(10, 8))

sns.heatmap(matrix, annot=True, fmt=".0f", cmap="YlGnBu", linewidths=.5)

plt.title('Average Time Taken to Settle Claims')

plt.xlabel('Year')

plt.ylabel('Month')

plt.show()

**Appendix 4.3: Code for Visualization 3**

import pandas as pd

import matplotlib.pyplot as plt

df = pd.read\_csv('ECA.csv', parse\_dates=['Actual', 'Planned'], infer\_datetime\_format=True)

# Calculate the time taken to settle claims

df['Time Taken to Settle'] = (df['Actual'] - df['Planned']).dt.days

# Create the histogram and set the axis range

plt.hist(df['Time Taken to Settle'], bins=50)

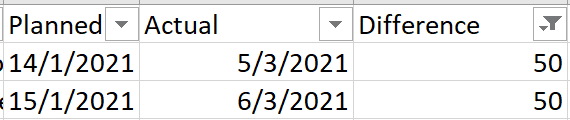
plt.xlabel('Time taken to settle claims (days)')

plt.ylabel('Number of claims')

plt.xlim(-60, 60) # set the x-axis range from 0 to 500

plt.show()

**Appendix 4.4**



**Appendix 5: Code for Question 5**

import pandas as pd

from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline

from sklearn.preprocessing import OneHotEncoder, StandardScaler

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

import matplotlib.pyplot as plt

# Read in the data

df = pd.read\_csv('ECA.csv', parse\_dates=['Actual', 'Planned'], infer\_datetime\_format=True)

# Create a new column for delay in days

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

# Convert the 'Amount' column to numeric values

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

# Drop any rows with missing values

df.dropna(inplace=True)

# Split the data into training and testing sets

train = df.sample(frac=0.8, random\_state=42)

test = df.drop(train.index)

# Define the preprocessing steps for numeric and categorical data

numeric\_transformer = Pipeline(steps=[('scaler', StandardScaler())])

categorical\_transformer = Pipeline(steps=[('onehot', OneHotEncoder(handle\_unknown='ignore'))])

# Combine the preprocessing steps using ColumnTransformer

preprocessor = ColumnTransformer(transformers=[

('num', numeric\_transformer, ['Amount']),

('cat', categorical\_transformer, ['Type', 'Region'])

])

# Create a pipeline that combines the preprocessing steps with a linear regression model

model = Pipeline(steps=[

('preprocessor', preprocessor),

('regressor', LinearRegression())

])

# Fit the model on the training data

model.fit(train[['Amount', 'Type', 'Region']], train['Delay'])

# Make predictions on the test set

y\_pred = model.predict(test[['Amount', 'Type', 'Region']])

# Calculate and print the mean squared error and r-squared value

mse = mean\_squared\_error(test['Delay'], y\_pred)

r\_squared = r2\_score(test['Delay'], y\_pred)

print('Predicted delays:', y\_pred)

print('Mean Squared Error:', mse)

print('R-Squared:', r\_squared)

# Create a scatter plot to visualize the predicted delays against the actual delays

plt.scatter(test['Delay'], y\_pred)

plt.xlabel('Actual Delays')

plt.ylabel('Predicted Delays')

plt.show()