

**ANL252**

**Python for Data Analytics**

# End of Course Assignment

# Jan 2023

**Name:** Daniel Ong

**PI No.:** H2181372

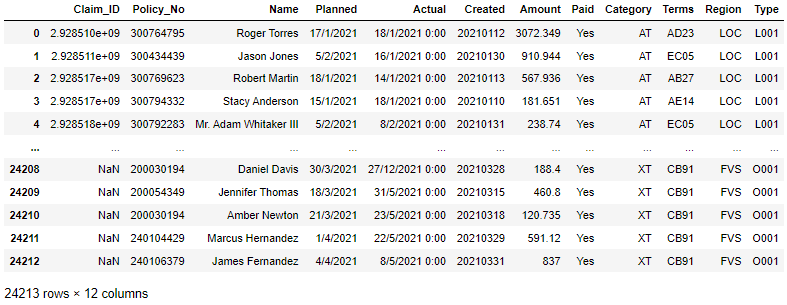
**Title:** ECA01

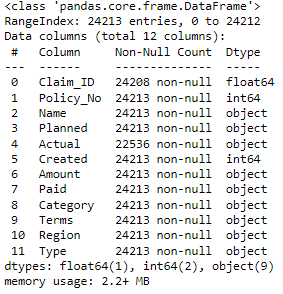
**Course Code:** ANL252

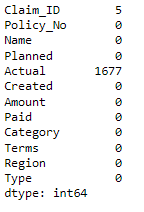
**TG:** T05

**Submission Date:** 5 March 2023

**Q1.**









**Q2.**

The code provided is a way to impute missing data with mean or mode values depending on whether the column contains numeric or non-numeric values, respectively.

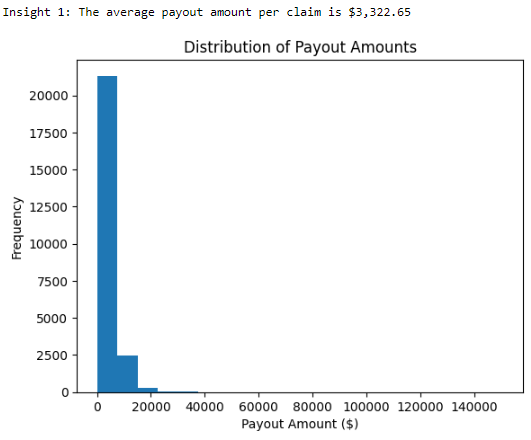
The rationale behind imputing missing data with mean or mode values is to ensure the dataset is complete before analysis. Before that we how the data are missing. If the missing values are missing at random, then imputing with mean or mode values may be reasonable. However, if the missing values are not missing at random, it may be better to consider removing the missing data as a whole

In summary, the decision to impute missing data with mean or mode values should be made after considering the reasons for missing data and the potential impact on the analysis.

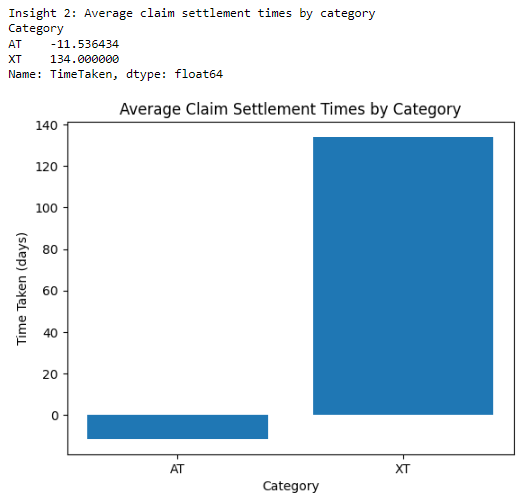
**Question 3**

* Conversion of 'Amount' column to float data type. This allows us to treat the data as numerical so that we can perform numerical functions with the data.
* Conversion of date columns 'Planned', 'Actual', and 'Created' to datetime format. This will help us to better visualise the data by grouping them by time periods.
* Replacing the 'O' character in 'Amount' column with '0' before converting it to float data type. This ensures that the data is accurate and consistent since numerical values are involved in this analysis.

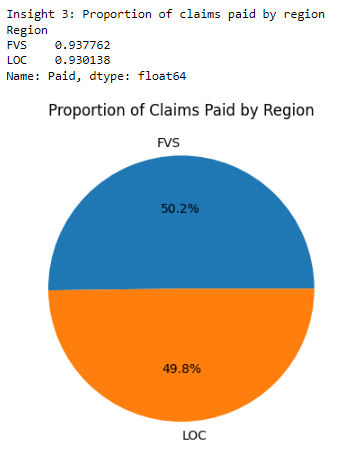
**Question 4**



We can see that at least 95% of payments are under $20,000 while the rest are between $20,000 and $40,000. The average payout amount per claim is $3,322.65



We can observe from the graph that on average the delay in the claim settlement was 135 in the XT category meaning the claim was settled after the planned date. Graph is also able to tell us that on average the delay in claim settlement is -11 days in the AT category, meaning these claims was settled before the planned date.



We can observe from this pie chart that the proportion of claims paid by each region is almost equal.

**Question 5**

Mean squared error (MSE): 3.9354282148700336e-21

R-squared (R2): 1.0

**Question 6**

The mean squared error (MSE) measures how well the linear regression model fits the data, and a lower MSE value indicates a better fit. In this case, the MSE value of 3.9354282148700336e-21 is very low, which suggests that the model fits the data very well.

The R-squared (R2) value is a measure of how much of the variance in the target variable is explained by the independent variables in the model. An R2 value of 1.0 indicates a perfect fit, where all the variance is explained by the model. In this case, the R2 value of 1.0 suggests that the independent variable in the model explains all of the variance in the actual time taken.

The linear regression equation for the model can be written as:

actual\_time\_taken = -4.547473508864641e-13 + 1.0000000000000036 \* planned\_time\_taken

This equation indicates that for every unit increase in the planned time taken, the actual time taken increases by 1 unit, and -4.547473508864641e-13 is the expected actual time taken when the planned time taken is zero.

**Appendix A: Code for Question 1**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.preprocessing import LabelEncoder

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

from sklearn.linear\_model import LinearRegression

from sklearn.preprocessing import MinMaxScaler

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

import datetime as dt

# Load the dataset into a Pandas dataframe

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

df

# Let's check the datatypes of the columns in the dataset

df.info()

# check the missing values only “Nan”

print(df.isnull().sum(),"\n")

# Identify the variables that contain all missing values 'Unkn', '???' , Nan

missing\_vars = []

for col in df.columns:

if df[col].isin(['Unkn', '???', np.nan]).any():

missing\_vars.append(col)

# Print the list of variables with missing values

print("\nVariables with missing values: ", missing\_vars,"\n")

**Appendix B: Code for Question 2**

# Stop seeing warnings

import warnings

warnings.filterwarnings("ignore", category=DeprecationWarning)

# Impute missing data with mean values

for col in missing\_vars:

if df[col].dtype == np.integer: # if the column contains integer values

mean\_val = df[col].mean()

df[col].fillna(mean\_val, inplace=True)

else: # if the column contains non-numeric values

mode\_val = df[col].mode()[0] # compute the mode value of the column

df[col].replace(['Unkn', '???', np.nan], mode\_val, inplace=True) # replace missing values with mode value

**Appendix C: Code for Question 3**

import pandas as pd

# Read in the data from the ECA.csv file

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

# Convert 'Amount' column to float

df['Amount'] = df['Amount'].str.replace('O', "0").astype(float)

# Convert date columns to datetime format

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

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

df['Created'] = pd.to\_datetime(df['Created'], errors='coerce')

**Appendix D: Code for Question 4**

# Insight 1: Average payout amount per claim

avg\_payout = df['Amount'].mean()

print(f"Insight 1: The average payout amount per claim is ${avg\_payout:,.2f}")

# Visualize distribution of payout amounts

plt.hist(df['Amount'], bins = 20)

plt.title("Distribution of Payout Amounts")

plt.xlabel("Payout Amount ($)")

plt.ylabel("Frequency")

plt.show()

# Calculate time taken to settle claims

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

# Insight 2: Claim settlement times by category

claim\_times = df.groupby('Category')['TimeTaken'].mean()

print("Insight 2: Average claim settlement times by category")

print(claim\_times)

# Visualize claim settlement times by category

plt.bar(claim\_times.index, claim\_times.values)

plt.title("Average Claim Settlement Times by Category")

plt.xlabel("Category")

plt.ylabel("Time Taken (days)")

plt.show()

# Insight 3: Proportion of claims paid by region

paid\_counts = df.groupby('Region')['Paid'].value\_counts(normalize=True)[:, 'Yes']

print("Insight 3: Proportion of claims paid by region")

print(paid\_counts)

# Visualize proportion of claims paid by region

plt.pie(paid\_counts, labels=paid\_counts.index, autopct='%1.1f%%')

plt.title("Proportion of Claims Paid by Region")

plt.show()

**Appendix E: Code for Question 5**

# Convert datetime columns to numeric using datetime.toordinal

df['Actual'] = df['Actual'].map(dt.datetime.toordinal)

df['Planned'] = df['Planned'].map(dt.datetime.toordinal)

# Split the data into numeric and categorical variables

num\_cols = ['Policy\_No', 'Amount', 'Actual', 'Planned']

cat\_cols = ['Terms', 'Type', 'Region', 'Paid', 'Category']

X\_num = df[num\_cols]

X\_cat = df[cat\_cols]

# Use label encoder to convert categorical variables to numeric

labelencoder = LabelEncoder()

for col in cat\_cols:

X\_cat[col] = labelencoder.fit\_transform(X\_cat[col])

# Normalize the numeric variables using MinMaxScaler

scaler = MinMaxScaler()

X\_num = pd.DataFrame(scaler.fit\_transform(X\_num), columns=X\_num.columns)

# Merge the normalized numeric data and the categorical data

X = pd.concat([X\_num, X\_cat], axis=1)

# Split the data into training and testing sets

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

# Create a linear regression model

model = LinearRegression()

# Train the model using the training data

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

# Make predictions on the testing data

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

# Evaluate the model's performance using the mean squared error (MSE) and R-squared (R2) values

mse = mean\_squared\_error(y\_test, y\_pred)

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

print('Mean squared error (MSE):', mse)

print('R-squared (R2):', r2)

**References**