

**ANL252**

**End-of-Course Assessment - January Semester 2023**

# Python for Data Analytics

**SUBMITTED BY:**

NAME: ANGEL LING YONG ENG

PI No.: K2211248

TUTORIAL GROUP: T05

INSTRUCTOR’S NAME: MUNISH KUMAR

SUBMISSION DATE: 6 MARCH 2023

## Question 1

After reading the dataset, Claim ID and Actual column contain missing values. The Claim ID column has a total of 5 rows of missing values, whereas the Actual column has a total of 1677 rows of missing values. Additionally, the Term column has a total of 7 rows of unique expressions (“Unkn”, “???”) which I derived them using the codes below.

In addition, there are 1677 rows of missing values in the Actual column due to their policy claims not being paid yet, which can be justified by the status of payment, Paid column indicating “No”. Hence, there is no actual date of claim settlement being filled in.

**Code (missing values):**

import pandas as pd

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

*# identify columns with missing values*

missing\_values\_col = df.columns[df.isna().any()]

print(missing\_values\_col)

*# create a list of column names to check*

columns\_to\_check = ["Claim\_ID", "Actual"]

*# count the number of empty cells in the columns*

num\_empty\_cells = df[columns\_to\_check].isna().sum()

print(num\_empty\_cells)

**Output:**

Index(['Claim\_ID', 'Actual'], dtype='object')

Claim\_ID 5

Actual 1677

dtype: int64

**Code (unique expressions):**

*# identify the columns that contain specific unique expressions*

specific\_values = df.isin([' ', 'Unkn', '???']).sum()

print(specific\_values)

**Output:**

Claim\_ID 0

Policy\_No 0

Name 0

Planned 0

Actual 0

Created 0

Amount 0

Paid 0

Category 0

**Terms 7**

Region 0

Type 0

dtype: int64

## Question 2

To treat the missing data, there are several ways to prepare. Below are the two ways that are commonly used and that I can prepare which are dropping the missing values and imputing the missing values.

1. **Drop the missing values:**

The rationale is that missing values can add bias to analysis since it can be challenging to effectively estimate a variable's true value when a significant number of its values are missing. This may occasionally result in inaccurate estimations of other variables in the dataset. In question 1, we have identified the specific columns with missing values. Therefore, we can use the .dropna() function, to drop the rows with missing values in these two columns, Claim\_ID and Actual.

As mentioned earlier in Q1, the missing values in Actual columns are due to insurance claims, not payout yet, hence for efficiency, we will be removing the entire missing values rows and focus on just those policy claims that are already paid out and have an Actual date of claim settlement in this data analysis.

**Code (Drop the missing values + Sanity check):**

*# drop the rows with missing values in the specific columns*

df = df.dropna(subset=["Claim\_ID", "Actual"])

print(f'\nAfter dropping the missing value')

*# count the number of empty cells in the columns*

num\_empty\_cells = df[columns\_to\_check].isna().sum()

print(num\_empty\_cells)

**Output:**

After dropping the missing value

Claim\_ID 0

Actual 0

dtype: int64

Similarly, we have identified the specific rows with unique expressions in question 1. Hence, we will use the .drop() function with the index parameter to remove the specific rows containing unique expressions in the Terms column.

**Code (Drop the unique expressions):**

*# filter the rows w/ unique expression*

specific\_rows = df[df['Terms'].isin([' ','Unkn', '???'])]

*# Drop the specific rows*

df = df.drop(specific\_rows.index)

check\_column = df.isin([' ','Unkn', '???']).sum()

print(f'After dropping the unique expression:', check\_column)

**Output:**

After dropping the unique expression:

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

\*I will be generating this as a CSV file for the cleaned data after using the drop method so that I have a fresh new dataset file and I can use read\_csv() for the relevant subsequent data preparation.

*#Export the cleaned df to CSV*

df.to\_csv('DROP\_VALUES.csv', index=False)

1. **Impute the missing values:**

Firstly, I will input new values into empty cells in Claim ID column by generating random numbers with the common starting prefix of “29285”. It is important to have a unique identifier that is tagged to each individual as we can make the dataset more complete and accurate. Furthermore, the company can easily identify the policy owner if there are any issues with their policy claims.

**Code:**

import random

#importing data file

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

*#treat Claim\_ID column*

def generate\_claim\_id():

random\_num = random.randint(10\*\*4, 10\*\*5 - 1)

claim\_id = "29285" + str(random\_num)

return claim\_id

*# Fill in empty cells in the claim\_ID column with random values*

df['Claim\_ID'] = df['Claim\_ID'].apply(lambda x: generate\_claim\_id() if pd.isna(x) else x)

*# check for missing values in 'claim\_id' column*

missing\_values = df['Claim\_ID'].isnull().any()

*# print result*

if missing\_values.any():

print("There are missing values in the following columns:")

print(missing\_values[missing\_values == True].index)

else:

print("There are no missing values in claim\_ID columns.")

**Output:**

There are no missing values in claim\_ID columns.

Secondly, assuming that for those we identified earlier that the policy amount is paid out, I will input new values into the missing actual date of claim settlement. Missing values may result in incomplete or inaccurate data visualization hence I will randomly generate dates in a range. With this, we can ensure that the data visualization is accurate and reflects the true nature of the dataset in which if we were to calculate the duration to settle a claim, we will have the necessary data.

**Code:**

import numpy as np

*# Generate random dates from 2021-01-01 to 2022-12-31 with daily frequency and timestamp of 0:00*

dates = pd.date\_range(start='2021-01-01 0:00', end='2022-12-31 0:00', freq='D').strftime('%d/%m/%Y %H:%M')

*# Fill up missing values in 'Actual' column with random dates generated*

df['Actual'] = df['Actual'].fillna(pd.Series(np.random.choice(dates, size=len(df.index))))

*# check for missing values in 'Actual' column*

missing\_values = df['Actual'].isnull().any()

*# print result*

if missing\_values.any():

print("There are missing values in the following columns:")

print(missing\_values[missing\_values == True].index)

else:

print("There are no missing values in Actual columns.")

**Output:**

There are no missing values in Actual columns.

Thirdly, replace the unique expression ‘???’ and “Unkn” with “Missing” in the Terms column. This is to facilitate when the company is identifying which internal terms and conditions code the policy is under, the moment they see that it shows Missing, the company can seek out more information using the policy number for example. It is also a form of standardizing the format of data so that when we are generating visualization tables, we will not have multiple different unknown variables to handle as it can affect the accuracy of the analysis and conclusions drawn from the dataset.

**Code:**

*# replace ??? and Unkn values in Term column with Missing*

df['Terms'] = df['Terms'].replace(['???', 'Unkn'], 'Missing')

*# identify the sum of cells with following values*

values\_with = df.isin(['', 'Unkn', '???']).sum()

*# print message indicating all values are treated*

if values\_with.sum() == 0:

print("All unique values are treated and replaced with 'Missing'")

**Output:**

All unique values are treated and replaced with 'Missing'

\*I will be generating this as a CSV file for the cleaned data after using the impute method so that I have a fresh new dataset file and I can use read\_csv() for the relevant subsequent data preparation.

*# Export the cleaned df to CSV*

df.to\_csv('IMPUTE\_VALUES.csv', index=False)

## Question 3

From here on, I will import DROP\_VALUES.csv to perform further data analysis. (22,524 rows x 12 columns)

1. **Data Normalisation**

Data normalization involves rescaling data to have values between 0 and 1 or -1 and 1, which is helpful when the data has different scales or measurement units. This prevents domination of analysis by a feature with a higher value and allows for equal contribution from each feature. Techniques such as min-max scaling, z-score normalization, and robust scaling can be used. Normalized data can be analyzed using histograms or box plots to spot patterns or outliers.

Meanwhile, during file preparation using the MinMaxScaler(), it returned this line of error “ValueError: could not convert string to float: '1762.OO' ”. It means that it spotted an amount that was keyed in a string format (1762.OO), hence we need to convert it to the correct float format (1762.00).

**Code:**

*#import relevant library*

from sklearn.preprocessing import MinMaxScaler

*#importing data file*

df = pd.read\_csv("DROP\_VALUES.csv")

*# Replace the wrong value in the column with new value*

*df['Amount'].replace('1762.OO', 1762.00, inplace=True)*

*# Scale the ‘Amount’ column to the range [0, 1]*

scaler = MinMaxScaler()

df[‘Amount\_scaled’] = scaler.fit\_transform(df[['Amount']])

df

**Output:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Claim\_ID** | **Policy\_No** | **Name** | **Planned** | **Actual** | **Created** | **Amount** | **Paid** | **Category** | **Terms** | **Region** | **Type** | **Amount\_scaled** |
| 0 | 2928509866 | 300764795 | Roger Torres | 17/1/2021 | 18/1/2021 0:00 | 20210112 | 3072.349 | Yes | AT | AD23 | LOC | L001 | 0.020021 |
| 1 | 2928511094 | 300434439 | Jason Jones | 5/2/2021 | 16/1/2021 0:00 | 20210130 | 910.944 | Yes | AT | EC05 | LOC | L001 | 0.005675 |
| 2 | 2928516927 | 300769623 | Robert Martin | 18/1/2021 | 14/1/2021 0:00 | 20210113 | 567.936 | Yes | AT | AB27 | LOC | L001 | 0.003399 |
| 3 | 2928517338 | 300794332 | Stacy Anderson | 15/1/2021 | 18/1/2021 0:00 | 20210110 | 181.651 | Yes | AT | AE14 | LOC | L001 | 0.000835 |
| 4 | 2928518375 | 300792283 | Mr. Adam Whitaker III | 5/2/2021 | 8/2/2021 0:00 | 20210131 | 238.74 | Yes | AT | EC05 | LOC | L001 | 0.001214 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 22519 | 3960616217 | 240104423 | Mary Taylor | 8/3/2022 | 6/3/2022 0:00 | 20220303 | 6126.018 | Yes | AT | CB91 | FVS | O001 | 0.040288 |
| 22520 | 3960616229 | 240104423 | Sarah Holland | 7/3/2022 | 9/3/2022 0:00 | 20220302 | 6288.599 | Yes | AT | CB91 | FVS | O001 | 0.041367 |
| 22521 | 3960616328 | 240104429 | Sean Foster | 15/3/2022 | 23/3/2022 0:00 | 20220310 | 3164.472 | Yes | AT | CB91 | FVS | O001 | 0.020632 |
| 22522 | 3960616355 | 240104429 | Tammy Duncan | 8/3/2022 | 23/3/2022 0:00 | 20220304 | 1150.452 | Yes | AT | CB91 | FVS | O001 | 0.007265 |
| 22523 | 3960617066 | 240105686 | Mr. Robert Rivera | 10/3/2022 | 12/3/2022 0:00 | 20220305 | 603.68 | Yes | AT | CB91 | FVS | O001 | 0.003636 |

22524 rows × 13 columns

As we can see additional column ‘Amount\_scaled’ is added to the dataset and with this, we can proceed to plot the necessary charts such as box plot or histogram to analyze the distribution of the data.

1. **Data Standardization**

Data Standardization is a step in data preparation that involves converting data from one format or structure to another. In this case, the date columns in the raw dataset such as “Planned”, “Actual”, and “Created” are not in the same format. To ensure consistency and accuracy in date-related calculations or analyses, these columns will be standardized and converted to YYYY-MM-DD format. This step will improve the accuracy of any subsequent studies that involve the dates and make the data more valuable due to the correct standardization and normalization.

**Code (For “Created” column):**

*# Convert to datetime format and then to the desired format*

df['Created'] = pd.to\_datetime(df['Created'], format='%Y%m%d').dt.strftime('%Y-%m-%d')

print(df)

\*\*E.g. Changed 20210112 to 2021-01-12

**Code (For “Actual” column):**

*# Convert the ‘Actual’ column to datetime format*

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

*# Format the ‘Actual’ column as a string in the desired format*

df['Actual'] = df['Actual'].dt.strftime("%Y-%m-%d")

print(df)

\*\*E.g. Changed 18/1/2021 0:00 to 2021-01-18

**Code (For “Planned” column):***# Convert the 'Planned' column to datetime format*

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

*# Format the 'Planned' column as a string in the desired format*

df['Planned'] = df['Planned'].dt.strftime("%Y-%m-%d")

print(df)

\*\*E.g. Changed 17/1/2021 to 2021-01-17

**Output:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Claim\_ID** | **Policy\_No** | **Name** | **Planned** | **Actual** | **Created** | **Amount** | **Paid** | **Category** | **Terms** | **Region** | **Type** | **Amount\_scaled** |
| 0 | 2928509866 | 300764795 | Roger Torres | 2021-01-17 | 2021-01-18 | 2021-01-12 | 3072.349 | Yes | AT | AD23 | LOC | L001 | 0.020021 |
| 1 | 2928511094 | 300434439 | Jason Jones | 2021-02-05 | 2021-01-16 | 2021-01-30 | 910.944 | Yes | AT | EC05 | LOC | L001 | 0.005675 |
| 2 | 2928516927 | 300769623 | Robert Martin | 2021-01-18 | 2021-01-14 | 2021-01-13 | 567.936 | Yes | AT | AB27 | LOC | L001 | 0.003399 |
| 3 | 2928517338 | 300794332 | Stacy Anderson | 2021-01-15 | 2021-01-18 | 2021-01-10 | 181.651 | Yes | AT | AE14 | LOC | L001 | 0.000835 |
| 4 | 2928518375 | 300792283 | Mr. Adam Whitaker III | 2021-02-05 | 2021-08-02 | 2021-01-31 | 238.74 | Yes | AT | EC05 | LOC | L001 | 0.001214 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 22519 | 3960616217 | 240104423 | Mary Taylor | 2022-03-08 | 2022-06-03 | 2022-03-03 | 6126.018 | Yes | AT | CB91 | FVS | O001 | 0.040288 |
| 22520 | 3960616229 | 240104423 | Sarah Holland | 2022-03-07 | 2022-09-03 | 2022-03-02 | 6288.599 | Yes | AT | CB91 | FVS | O001 | 0.041367 |
| 22521 | 3960616328 | 240104429 | Sean Foster | 2022-03-15 | 2022-03-23 | 2022-03-10 | 3164.472 | Yes | AT | CB91 | FVS | O001 | 0.020632 |
| 22522 | 3960616355 | 240104429 | Tammy Duncan | 2022-03-08 | 2022-03-23 | 2022-03-04 | 1150.452 | Yes | AT | CB91 | FVS | O001 | 0.007265 |
| 22523 | 3960617066 | 240105686 | Mr. Robert Rivera | 2022-03-10 | 2022-12-03 | 2022-03-05 | 603.68 | Yes | AT | CB91 | FVS | O001 | 0.003636 |

22524 rows × 13 columns

1. Identifying and/or removing outliers

Outliers can have a significant impact on statistical analyses because they can skew the data and lead to incorrect conclusions. For example, if we are analyzing the average payout amount for a particular insurance type and there are a few extreme outliers with extremely high payout amounts, it can significantly inflate the average payout amount and make it appear that the policy is much more costly than it actually is for the majority of policyholders.

Identifying and handling outliers is essential in ensuring accurate data analysis. Removing outliers can improve the accuracy of statistical analyses by reducing the impact of extreme values on the overall data. However, it is important to consider the context and purpose of the analysis before removing outliers. In some cases, outliers may be valid data points that represent unusual but still relevant cases, and removing them may lead to a loss of important information.

**Code:**

*#identifying the outliers using Amount\_scaled column*

q1 = df['Amount\_scaled'].quantile(0.25)

q3 = df['Amount\_scaled'].quantile(0.75)

iqr = q3 - q1

lower\_bound = q1 - 1.5 \* iqr

upper\_bound = q3 + 1.5 \* iqr

outliers = df[(df['Amount\_scaled'] < lower\_bound) | (df['Amount\_scaled'] > upper\_bound)]

outliers

**Output:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Claim\_ID** | **Policy\_No** | **Name** | **Planned** | **Actual** | **Created** | **Amount** | **Paid** | **Category** | **Terms** | **Region** | **Type** | **Amount\_scaled** |
| **41** | 2928540195 | 300792293 | Brian Espinoza | 15/1/2021 | 15/1/2021 | 10/1/2021 | 15395.87 | Yes | AT | AD23 | LOC | L001 | 0.101813 |
| **63** | 2928543306 | 300689451 | Kristen Webb | 17/1/2021 | 17/1/2021 | 12/1/2021 | 13541.04 | Yes | AT | AD23 | LOC | L001 | 0.089502 |
| **102** | 2928546732 | 300707822 | Nicholas Banks Jr. | 15/1/2021 | 15/1/2021 | 10/1/2021 | 15540.848 | Yes | AT | AD23 | LOC | L001 | 0.102775 |
| **132** | 2928551881 | 300799367 | Daniel Mcdonald | 18/1/2021 | 22/1/2021 | 13/1/2021 | 15865.722 | Yes | AT | AD23 | LOC | L001 | 0.104932 |
| **272** | 2928576815 | 300875006 | Rebecca Sosa | 19/1/2021 | 22/1/2021 | 14/1/2021 | 12522.755 | Yes | AT | AD23 | LOC | L001 | 0.082744 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **22479** | 3960613993 | 240104440 | Amy Powell | 24/2/2022 | 3/2/2022 | 19/2/2022 | 12056.894 | Yes | AT | CB91 | FVS | O001 | 0.079652 |
| **22481** | 3960614084 | 240104440 | Sarah Cortez | 28/2/2022 | 3/4/2022 | 23/2/2022 | 16970.981 | Yes | AT | CB91 | FVS | O001 | 0.112267 |
| **22486** | 3960614363 | 240106181 | Brian Phillips | 28/2/2022 | 30/4/2022 | 25/2/2022 | 12158.28 | Yes | AT | CB91 | FVS | O001 | 0.080325 |
| **22491** | 3960614483 | 240106408 | Brian Yang | 4/3/2022 | 3/10/2022 | 27/2/2022 | 15771.196 | Yes | AT | CB91 | FVS | O001 | 0.104304 |
| **22493** | 3960614620 | 240104409 | Jennifer Adams | 6/3/2022 | 3/6/2022 | 1/3/2022 | 12823.369 | Yes | AT | CB91 | FVS | O001 | 0.084739 |

1160 rows × 13 columns

The generated output shows there are 1,160 rows of outliers, indicating that the payout amounts for these claims are higher than the average payout amounts. This information can be used to inform further analysis or to identify patterns in the data that may be useful for improving insurance policies. However, it is important to consider the potential impact of these outliers on any statistical analyses and to handle them appropriately based on the specific context and purpose of the analysis.

## Question 4

1. Claims processing efficiency [Planned vs Actual]

After standardizing the dates column, the "Settlement\_duration" column was created in a new data frame. This column calculates the difference between the actual date of claim settlement and the planned date of claim settlement. The duration of settlement is a useful metric to assess the efficiency of the claims processing department. A delay between the planned and actual settlement dates could indicate issues within the department, such as understaffing. The data will be visualized using a histogram with 5 bins to show the distribution of settlement durations.

**Code:**

import matplotlib.pyplot as plt

*# convert the date columns to datetime objects*

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

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

*# Calculate the difference between the two date columns*

df['Settlement\_duration'] = df['Actual'] - df['Planned']

*# Plot a histogram of the duration of settlement*

plt.hist(df['Settlement\_duration'].dt.days, bins=5)

plt.xlabel('Duration of Settlement (days)')

plt.ylabel('Number of Claims')

plt.title('Distribution of Duration of Insurance Claim Settlement')

plt.show()

**Output:**

Settlement\_duration

0 1 days

1 -20 days

2 -4 days

3 3 days

4 178 days

... ...

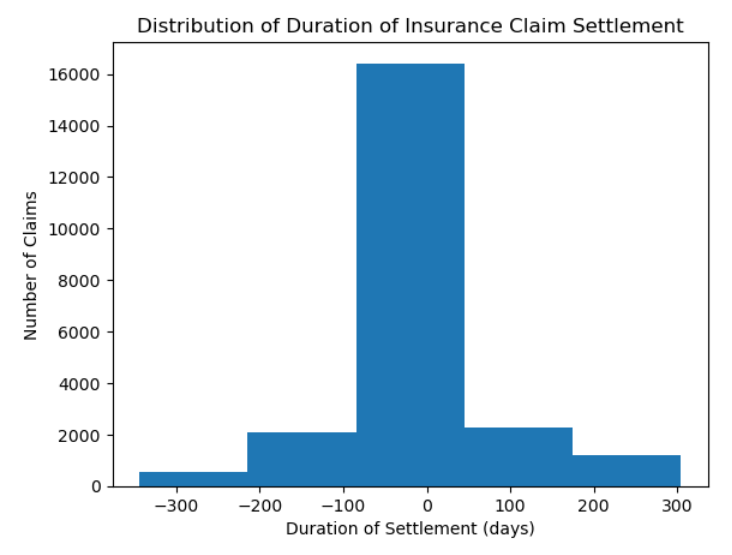
22519 87 days

22520 180 days

22521 8 days

22522 15 days

22523 268 days

The histogram indicates that most insurance claims are resolved quickly, but some claims take longer than expected to settle. Positive numbers of days are used to represent the length of time it takes to settle a claim from the time it is recorded to the time it is paid out. Negative numbers of days, on the other hand, may represent the length of time it takes to resolve a claim in reverse, starting with the settlement date and ending with the reporting date. Negative days can also indicate how long it took to resolve a claim compared to the expected time or how much longer it took than usual.

1. **Policy Performance [Type and Amount]**

First, we will do a count() in the Type column to show the frequency of each Type.

**Code:**

count\_df = pd.DataFrame(df['Type'].value\_counts())

count\_df

**Table:**

Type

L001 20432

O001 1788

L005 231

L002 67

L003 5

L004 1

The table provides an overview of the number of policies associated with each insurance type. Type L001 has the highest count of 20,432, while Type L004 has the lowest count of 1. The higher count of Type L001 policies may suggest that individuals consider it more valuable and essential, possibly due to the extent of protection offered, ease of acquiring insurance, perceived risk of the covered event, or insurance price. The lower count of Type L004 policies owned by only one policyholder may indicate that it is considered less essential and valuable, possibly due to the risk perception or legal requirements. However, other factors like personal preferences, availability, and pricing of insurance may also impact the popularity of each type.

**Code:**

*# group the dataframe by Insurance Type and sum the Amount*

grouped\_type = df.groupby('Type').sum()

*# create a bar chart using Matplotlib*

plt.bar(grouped\_type.index, grouped\_type['Amount'])

*# set chart title and axis labels*

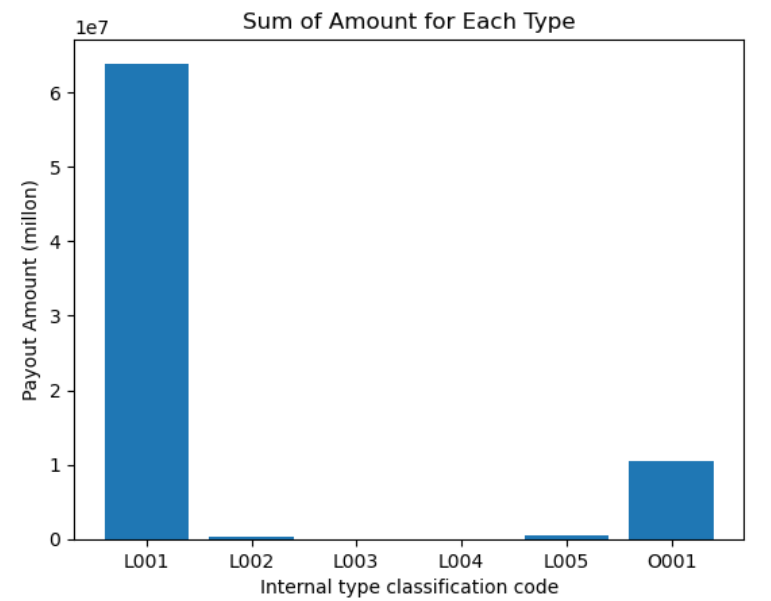
plt.title('Sum of Amount for Each Type')

plt.xlabel('Internal type classification code')

plt.ylabel('Payout amount (millon)')

plt.show()

**Output**:



By analyzing the number of claims and payout amounts associated with each type of policy, we can assess the overall performance of the policies. Type L001 has a higher-than-average number of payouts of the total sum of over 6 million, which may indicate that the policy-insured event is more likely to occur or that the policy is covering a higher-risk group of individuals. This could suggest that the insured individuals may be engaging in higher-risk activities or live in an area that poses greater risks to the policyholder, which entails a more comprehensive and expensive policy.

On the other hand, Type L002, L003, L004, and L005 have a lower-than-average number of claims or payouts, indicating that the insured event is less likely to occur or that the policy covers a lower-risk group of people. This could suggest that the insured individuals are taking preventive measures or live in a low-risk area for certain events. For instance, a lower-than-average number of claims on a health insurance policy may indicate that the insured individuals are taking good care of their health, reducing the likelihood of illness or injury.

In general, the popularity of a particular insurance type may reveal how people perceive the risks they face and the importance they attach to safeguarding their possessions. By analyzing claims data, insurance companies can better understand the risks associated with each policy and adjust their offerings to ensure that they are adequately covering policyholders' needs.

1. **Payment Status [Paid]**

In this analysis, I am using the raw dataset (ECA.csv) so to analyze the initial payment status for the policy.

The "Paid" column in insurance claims data can be used to assess the promptness of claim payments. A high percentage of claims marked as "Not Paid" may suggest that the insurance company is experiencing payment delays or denying many claims.

**Code:**

*# Load data into a DataFrame*

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

*# Count the number of 'yes' and 'no' values in the Paid column*

counts = df['Paid'].value\_counts()

*# Create a pie chart using the counts*

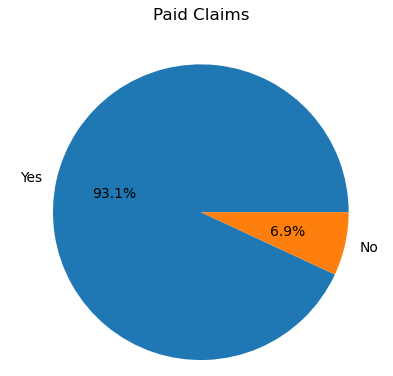
plt.pie(counts, labels=counts.index, autopct='%1.1f%%')

*# Add a title to the chart*

plt.title('Paid Claims')

plt.show()

**Output:**



After analyzing the pie chart, we can infer that the insurance company has a high rate of claim payment, as 93.1% of claims were paid. This indicates that the company is financially stable and has a good record of fulfilling its policyholders' obligations. The low percentage of denied claims, only 6.9%, may also indicate that the company has effective risk management strategies in place to minimize losses and prevent claims from being denied. However, it's important to note that this information alone is not sufficient to fully assess the insurance company's performance. Other factors, such as the number and types of claims processed, the company's financial strength and stability, and customer satisfaction, need to be considered to form a more comprehensive evaluation.

By analyzing these data points, insurance companies can improve their claims processing procedures, identify potential inefficiencies or issues, and enhance their policy offerings. This can ultimately lead to better customer experiences, higher retention rates, and improved financial performance.

## Question 5

To perform linear regression modeling, we need to collect data on planned and actual processing dates and the delay in days. After cleaning the data and removing missing values, we will use simple linear regression as the appropriate model for one dependent and one independent variable. We will split the data into training and testing sets, train the model on the training set, and evaluate its performance using metrics such as RMSE, MAE, and R-squared. The goal is to predict the delay in days by using the trained model to make predictions on new data.

To improve model performance, some useful data pre-processing techniques include handling missing values by either removing them, imputing them statistically or using algorithms that handle missing values. Outlier detection and removal can be performed using methods like Z-score or Interquartile Range. Categorical variables need to be encoded numerically using methods like one-hot encoding or label encoding. Feature engineering involves creating new features from existing variables to improve model performance.

In summary, to predict the delay in days for processing claims using linear regression, we need to collect and pre-process the data, select relevant features, choose an appropriate model, train and test the model, evaluate its performance, and make predictions on new data. The approach will vary based on the dataset's characteristics and the analysis objectives. The below codes are how I perform the linear regression modeling.

**Code:**

import pandas as pd

import numpy as np

from sklearn.linear\_model import LinearRegression

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

from sklearn.preprocessing import StandardScaler

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

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

*# Drop rows with any missing value*

df = df.dropna(axis=0, how="any")

*# Drop columns that are not needed in the analysis*

df = df.drop(['Category', 'Terms', 'Region', 'Type', 'Created', 'Paid'], axis=1)

*# Convert dates to datetime format*

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

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

*# Extract day of the month from dates*

df['Planned\_day'] = df['Planned'].dt.day

df['Actual\_day'] = df['Actual'].dt.day

*# Split the data into training and testing sets*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df[['Planned\_day']], df['Actual\_day'], test\_size=0.3, random\_state=33)

*# Scale the features*

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

*# Fit the model on the training data*

model = LinearRegression()

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

*# Make predictions on the testing data*

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

*# Evaluate the performance of the model*

print("Mean squared error: {:.2f}".format(mean\_squared\_error(y\_test, y\_pred)))

print("Mean absolute error: {:.2f}".format(mean\_absolute\_error(y\_test, y\_pred)))

print("R-squared: {:.2f}".format(r2\_score(y\_test, y\_pred)))

*# Fit the model on the training data*

model = LinearRegression()

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

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

*# Fit the model on the planned and actual data*

model.fit(planned, actual)

*# Print the intercept and coefficients of the model*

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

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

**Output:**

Mean squared error: 52.77

Mean absolute error: 5.15

R-squared: 0.31

Intercept: [6.771663]

Coefficient: [[0.57301543]]

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Claim\_ID** | **Policy\_No** | **Name** | **Planned** | **Actual** | **Amount** | **Planned\_day** | **Actual\_day** | **Settlement\_duration** |
| **0** | 2.93E+09 | 300764795 | Roger Torres | 2021-01-17 | 2021-01-18 | 3072.349 | 17 | 18 | 1 |
| **1** | 2.93E+09 | 300434439 | Jason Jones | 2021-02-05 | 2021-01-16 | 910.944 | 5 | 16 | 11 |
| **2** | 2.93E+09 | 300769623 | Robert Martin | 2021-01-18 | 2021-01-14 | 567.936 | 18 | 14 | -4 |
| **3** | 2.93E+09 | 300794332 | Stacy Anderson | 2021-01-15 | 2021-01-18 | 181.651 | 15 | 18 | 3 |
| **4** | 2.93E+09 | 300792283 | Mr. Adam Whitaker III | 2021-02-05 | 2021-02-08 | 238.74 | 5 | 8 | 3 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **24085** | 3.96E+09 | 240104423 | Mary Taylor | 2022-03-08 | 2022-03-06 | 6126.018 | 8 | 6 | -2 |
| **24086** | 3.96E+09 | 240104423 | Sarah Holland | 2022-03-07 | 2022-03-09 | 6288.599 | 7 | 9 | 2 |
| **24087** | 3.96E+09 | 240104429 | Sean Foster | 2022-03-15 | 2022-03-23 | 3164.472 | 15 | 23 | 8 |
| **24088** | 3.96E+09 | 240104429 | Tammy Duncan | 2022-03-08 | 2022-03-23 | 1150.452 | 8 | 23 | 15 |
| **24090** | 3.96E+09 | 240105686 | Mr. Robert Rivera | 2022-03-10 | 2022-03-12 | 603.68 | 10 | 12 | 2 |

22524 rows × 11 columns

## Question 6 - 10 marks

The model's mean squared error (MSE) is 52.77, which is the average squared difference between predicted and actual values. This means that the model's predictions are off by the square root of the MSE or about 7.31 days on average. The model's performance degrades as the MSE increases.

The model's mean absolute error (MAE) is 5.15, which represents the average absolute difference between predicted and actual values. This means that the model's predictions are on average 5.16 days off. The lower the MAE, the better the performance of the model.

The model's R-squared value is 0.31, which measures the proportion of variance in the actual values explained by the model. This means that the model explains 31% of the data's variance. The higher the R-squared, the better the performance of the model.

Overall, these metrics indicate that the model's performance is inadequate, there is room for improvement. The MSE and MAE values are high, indicating that the model is not making accurate predictions. The R-squared value is also quite low, implying that other factors may be influencing the relationship between the Planned and Actual days. More research may be required to identify these factors and improve the model's performance.

**Linear Equation:**

y = 6.771663 + 0.57301543 \* x

In the above equation suggests that y is the predicted target variable (actual day) and x is the predictor variable (planned day). This equation tells us how the predicted target variable changes as the predictor variable changes.

**----- END OF ECA PAPER ----**