Question 1

The variables or column names are.

1. Claim\_ID
2. Actual
3. Terms

Question 2

Blanks or null values are found in Claim\_ID and Actual columns.

Missing values of ‘???’ and ‘Unkn’ are found in Terms column.

Claim\_ID is a unique ID for each applicant. The solution is to generate a new unique ID for these applicants.

Values ‘???’ and ‘Unkn’ in Terms column will be changed to a standardized string value of ‘Unknown’. The reason is the meaning of each value for the internal codes of the organization is unknown therefore it is not right to assign a random code.

Empty values from Actual column will be replaced with 'Not Paid Yet' because the claimer has not paid. Instead of leaving it blank, it is better to view and read it as 'Not Paid Yet' for clearer understanding.

Question 3

1. Replacing values from Amount Column to be integer dtype.
   1. Since the values for Amount are numbers, it makes sense to convert the data type to integer for better analysis.
2. Convert dates from Planned, Actual and Created column to be standardized and consistent.
   1. These 3 columns have values that shows dates. Therefore, it is better to change the values to datetime and standardize them to the same format.
3. Select variables to show only data where Paid = Yes
   1. It does not make sense to analyze data of payments that are yet to be paid.
   2. Therefore, it is better to create a new dataframe to show all data that already made payment.

Question 4

1. Total claims paid for 2021 and 2022.
2. Total claims for each month in 2021 and 2022.
3. All unique Claimer names and their total claims for 2021 and 2022

The total claims paid in 2021 is $65,285,881.35 and the total claims paid in 2022 is $9,535,922.39.

The total claims for each month in 2021 are as follows:

Actual\_Date

1 $2,346,527.41

2 $4,724,802.44

3 $5,262,125.60

4 $6,375,096.23

5 $5,934,714.38

6 $5,330,343.69

7 $5,919,255.16

8 $5,584,635.96

9 $5,436,065.66

10 $6,222,134.06

11 $5,921,856.62

12 $6,228,324.13

The total amount of claims paid are at least $2 million which is a huge sum.

The insurance company is having a lot of business and a lot of work. This data can be shown

to the Board

of Directors to assist with request of more manpower.

The total claims for each month in 2022 are as follows:

Actual\_Date

1 $2,911,529.33

2 $4,265,660.03

3 $2,205,088.42

4 $134,345.97

5 $19,298.64

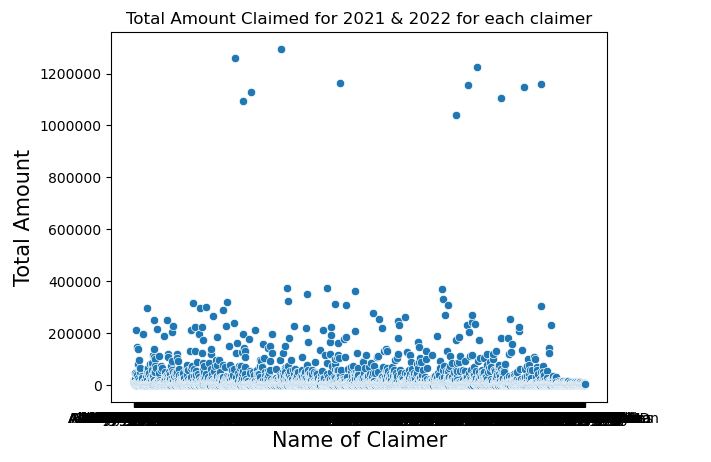
The data only shows total claim from January to May. Maybe their financial year starts in

May each year.

In the month of April and May, the sum of claim is surprisingly low where it did not reach theusual $2 Million per month.

Scatterplot data for all unique Claimer names and their total claims for 2021

and 2022



This shows that most of the total claims are between 0 to $400,000. Only 11 claimers have a total claim of more than $1 million.

There are too many names or clients that are engaging with the insurance company

therefore the x-axis which is supposed to show all names, are not visible.

Question 5

The dates from Actual and Planned are filtered according to 2021 and 2022. There will be 2 different scatterplots for 2021 and 2022.

Data preparation of calculating delay in dates will be performed.

Actual Dates minus Planned dates will show the delay in dates of the claimer to make payment. There will be negative dates which will be replaced with 0 because there is no such thing as negative date. The claimer just managed to pay ahead of the planned date. Therefore, there is no delay in payment.

Outliers are removed earlier using upper and lower quantile. The scatterplot will then be created as shown below with Linear Regression to make predictions of delay in dates.

Question 6

The scatterplot below shows the delay in dates to make payment for 2021.

However, the Linear Regression shows a bad fit, which means Linear Regression is not the best method to provide predictions of future values for the year 2021.

Unlike for the year 2021, there is a bit of linear regression, but it is not the best. It can still be used to predict future values.

Question 6

The equation for Linear regression is Y = mX + a, where X is the independent variable and Y is the dependent variable. The slope of the line is m while a is the intercept.

Appendix

import random

from datetime import datetime

import numpy as np

import pandas as pd

import seaborn as sb

import matplotlib.pyplot as plt

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

from sklearn import preprocessing

from sklearn.linear\_model import LinearRegression

from sklearn import linear\_model

# Read csv file and store into a dataframe

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

# To find out the dtype of each columns

result = df.dtypes

result

# To locate the column that has "???" value and the total sum

search\_QnMarks = df.isin(['???']).sum()

search\_QnMarks

# To locate the column that has "Unkn" value and the total sum

search\_Unkn = df.isin(['Unkn']).sum()

search\_Unkn

# To locate the column that has "Blanks" value and the total sum

search\_Blanks = df.isnull().sum()

search\_Blanks

# To fill up all the blank values with 0

df['Claim\_ID'] = df['Claim\_ID'].fillna(0)

# Print the index and value of Claim\_ID that has 0

for index, col in df.iterrows():

if col['Claim\_ID'] == 0:

print(index, col['Claim\_ID'])

# Get all the numbers of ID from Claim\_ID into a list

claimID\_list = list(df['Claim\_ID'])

# To generate new Claim\_IDs for the Claim\_IDs which has 0 value

# Create a list to store the newly generated Claim\_IDs2

random\_ClaimIDs = []

# Generate 5 IDs

while len(random\_ClaimIDs) < 5:

i = random.randrange(3960000000, 3960999999)

# If the random generated ID is already in the Claim\_ID, regenerate a new one

if i in claimID\_list:

i = random.randrange(3960000000, 3960999999)

# Add the ID into the list

else:

random\_ClaimIDs.append(i)

# Replace the Claim\_ID that has 0 value with the newly generated ID

df.iloc[[24208, 24209, 24210, 24211, 24212],[0]] = random\_ClaimIDs[0:5]

# For loop through the column and index to locate the index of the data that

# contains "???" and "Unkn" in the Terms column

for index, col in df.iterrows():

if col['Terms'] == '???' or col['Terms'] == 'Unkn':

print(index, col['Terms'])

# Replacing the value from "???" and "Unkn" in Terms column to "Unknown" to be standardised

df.iloc[[2729,2730,2731,2732,3492,6177,6329],[9]] = "Unknown"

print(df.loc[[2729,2730,2731,2732,3492,6177,6329], ['Terms']])

# Convert all amount from string to numeric type

df[['Amount']] = df[['Amount']].apply(pd.to\_numeric)

# Data Preparation: Replacing values from Amount Column to be integer dtype

# Locate the index of the value "1762.OO"

for index, col in df.iterrows():

if col['Amount'] == "1762.OO":

print(index, col['Amount'])

# Locate the index of the value and change value from "1762.OO" to "1762.00"

df.iloc[[3698],[6]] = 1762.00

# Convert all amount from string to numeric type

df[['Amount']] = df[['Amount']].apply(pd.to\_numeric)

df.dtypes

# Data Preparation: Convert dates from Planned, Actual and Created column to be standardized and consistent.

# Create a new list to store all values from "Created" column

new\_Planned = list(df["Planned"])

# Create a new list for appending new datetime values

new\_Date\_Planned = []

# Convert all the int values in the list to string and then to datetime format

for x in new\_Planned:

# Convert to date time

datetime\_object = datetime.strptime(x, "%d/%m/%Y")

# Append to new list

new\_Date\_Planned.append(datetime\_object)

# Create a new column from the Dataframe from the new list

df['new\_Date\_Planned'] = new\_Date\_Planned

# Data Preparation: Replace values of missing dates from Actual column

# Fill NA of blanks Actual date to "Not Paid Yet"

df["Actual"] = df["Actual"].fillna('Not Paid Yet')

# Create a new list to store all values from "Created" column

new\_Actual = list(df["Actual"])

# Create a new list for appending new datetime values

new\_Date\_Actual = []

# Convert all the int values in the list to string and then to datetime format

for x in new\_Actual:

# If value is not "Not Paid Yet"

if x != "Not Paid Yet":

# Convert to string

y = str(x)

# Convert to date time

datetime\_object = datetime.strptime(y, "%d/%m/%Y %H:%M")

# Append to new list

new\_Date\_Actual.append(datetime\_object.date())

else:

# Append "Not Paid Yet" string value

new\_Date\_Actual.append(x)

# Create a new column from the Dataframe from the new list

df['new\_Date\_Actual'] = new\_Date\_Actual

df

# Create a new list to store all values from "Created" column

new\_Created = list(df["Created"])

# Create a new list for appending new datetime values

new\_Date\_Created = []

# Convert all the int values in the list to string and then to datetime format

for x in new\_Created:

#print(x)

y = str(x)

#type(x)

#print(type(y))

# Slice the string to convert to a datetime string format

z = y[:4] + '-' + y[4:6] + '-' + y[6:]

#print(z)

# Convert string to datetime type

datetime\_object = datetime.strptime(z, "%Y-%m-%d")

#print(type(datetime\_object))

#Append the value into the new list

new\_Date\_Created.append(datetime\_object)

# Create a new column from the Dataframe from the new list

df['new\_Date\_Created'] = new\_Date\_Created

df

# Drop old columns named "Actual", "Created" and "Planned"

new\_df = df.drop(["Actual","Created","Planned"], axis=1)

new\_df

# Data Preparation: Select Variables to show only data where Paid = Yes

# New Dataframe created to only show data where Paid = 'Yes'

new\_df = new\_df.loc[new\_df['Paid'] == 'Yes']

# Get the dtypes for all colummns

results = new\_df.dtypes

results

# Data Preparation to convert "new\_Date\_Actual" to datetime64

date\_Actual = list(new\_df["new\_Date\_Actual"])

# Create a new list for appending new datetime values

Actual\_Date = []

# Convert all the int values in the list to string and then to datetime format

for x in date\_Actual:

# Convert to string

y = str(x)

# Convert to date time

datetime\_object = datetime.strptime(y, "%Y-%m-%d")

# Append to new list

Actual\_Date.append(datetime\_object)

#Actual\_Date

# Drop old columns named "new\_Date\_Actual"

new\_df = df.drop("new\_Date\_Actual", axis=1)

new\_df

# Data Preparation: Select Variables to show only data where Paid = Yes

# New Dataframe created to only show data where Paid = 'Yes'

new\_df = new\_df.loc[new\_df['Paid'] == 'Yes']

# Create a new column from the Dataframe from the new list

new\_df['Actual\_Date'] = Actual\_Date

# Rename columns to make it more understandable

new\_df.rename(columns = {'new\_Date\_Planned':'Planned\_Date', 'new\_Date\_Created':'Created\_Date'}, inplace = True)

new\_df

# Review columns dtypes

results = new\_df.dtypes

results

# Create new dataframe for dates in 2021 only

filtered\_df\_2021 = new\_df.loc[(new\_df['Actual\_Date'] >= '2021-01-01')

& (new\_df['Actual\_Date'] < '2021-12-31')]

# Remove unnecessary columns and create new dataframe from these columns for 2021

new\_df\_filtered\_2021 = filtered\_df\_2021[['Claim\_ID','Policy\_No','Name','Actual\_Date','Amount']]

new\_df\_filtered\_2021

# Create new dataframe for dates in 2022 only

filtered\_df\_2022 = new\_df.loc[(new\_df['Actual\_Date'] >= '2022-01-01')

& (new\_df['Actual\_Date'] < '2022-12-31')]

# Remove unnecessary columns and create new dataframe from these columns for 2022

new\_df\_filtered\_2022 = filtered\_df\_2022[['Claim\_ID','Policy\_No','Name','Actual\_Date','Amount']]

new\_df\_filtered\_2022

# Create new Dataframe for filtered\_df\_2021 Table

new\_df\_filtered\_2021\_table = new\_df\_filtered\_2021.pivot\_table(index='Name', values='Amount', aggfunc='sum')

# Reset index for the dataframe

# Total claims for each claimer in 2021

new\_df\_filtered\_2021\_table = new\_df\_filtered\_2021\_table.reset\_index()

new\_df\_filtered\_2021\_table.rename(columns = {'Amount':'Amount for 2021'}, inplace = True)

new\_df\_filtered\_2021\_table

# Calculate total claim amount using for loop

total\_claims\_2021 = 0

for i in new\_df\_filtered\_2021\_table['Amount for 2021']:

total\_claims\_2021 = total\_claims\_2021 + i

print(f"The total claims paid in 2021 is ${total\_claims\_2021:,.2f}")

# Create new Dataframe for filtered\_df\_2021 Table

new\_df\_filtered\_2022\_table = new\_df\_filtered\_2022.pivot\_table(index='Name', values='Amount', aggfunc='sum')

# Reset index for the dataframe

# Total claims for each claimer in 2022

new\_df\_filtered\_2022\_table = new\_df\_filtered\_2022\_table.reset\_index()

new\_df\_filtered\_2022\_table.rename(columns = {'Amount':'Amount for 2022'}, inplace = True)

new\_df\_filtered\_2022\_table

# Calculate total claim amount using for loop

total\_claims\_2022 = 0

for i in new\_df\_filtered\_2022\_table['Amount for 2022']:

total\_claims\_2022 = total\_claims\_2022 + i

print(f"The total claims paid in 2022 is ${total\_claims\_2022:,.2f}")

#calculate sum of sales grouped by month in 2021

new\_df\_filtered\_2021\_month = "$" + new\_df\_filtered\_2021.groupby(new\_df\_filtered\_2021.Actual\_Date.dt.month)['Amount'].sum().map('{:,.2f}'.format)

new\_df\_filtered\_2021\_month

#calculate sum of sales grouped by month in 2021

new\_df\_filtered\_2022\_month = "$" + new\_df\_filtered\_2022.groupby(new\_df\_filtered\_2022.Actual\_Date.dt.month)['Amount'].sum().map('{:,.2f}'.format)

new\_df\_filtered\_2022\_month

# Create new Dataframe for claimers using merge function

filtered\_name\_amt = pd.merge(new\_df\_filtered\_2021\_table, new\_df\_filtered\_2022\_table, on = "Name", how = "outer")

# Fill NA to 0

filtered\_name\_amt = filtered\_name\_amt.fillna(0)

# Calculate total amount claimed for both 2021 and 2022 for all claimers and create a new column to show the total sum

filtered\_name\_amt['Total Claimed for 2021 & 2022'] = filtered\_name\_amt['Amount for 2021'] + filtered\_name\_amt['Amount for 2022']

filtered\_name\_amt

# Get a list of data from Name column for x axis

axis\_x = list(filtered\_name\_amt['Name'])

# Get a list of data from Total Claimed for 2021 & 2022 column for y axis

axis\_y = list(filtered\_name\_amt['Total Claimed for 2021 & 2022'])

# Use seaborn library to visualise random data distribution with scatter plot graph

#sb.scatterplot(filtered\_name\_amt['Name'], filtered\_name\_amt['Total Claimed for 2021 & 2022'])

sb.scatterplot(axis\_x, axis\_y)

# Define title label

plt.title("Total Amount Claimed for 2021 & 2022 for each claimer", fontsize=12)

# Define x-axis label

plt.ylabel("Total Amount", fontsize=15)

# Define y-axis label

plt.xlabel("Name of Claimer", fontsize=15)

# Remove Scientific notation for total amount

plt.ticklabel\_format(style='plain', axis='y')

plt.show()

# Create new Dataframe with columns from 'Amount','Planned\_Date' & 'Actual\_Date' from 2021

estimated\_Delay\_2021 = filtered\_df\_2021[['Amount','Planned\_Date','Actual\_Date']]

estimated\_Delay\_2021

# Create new list to store all the dates from 'Planned\_Date' column in 2021

all\_planned\_dates\_2021 = list(estimated\_Delay\_2021['Planned\_Date'])

#all\_planned\_dates\_2021

# Create new list to store all the dates from 'Actual\_Date' column in 2021

all\_actual\_dates\_2021 = list(estimated\_Delay\_2021['Actual\_Date'])

#all\_actual\_dates\_2021

# Create new list to store delay days for 2021 claims

delay\_dates\_2021 = []

# iterates over 2 lists in 2021 to calculate delay days

for (a, b) in zip(all\_planned\_dates\_2021, all\_actual\_dates\_2021):

c = b - a

# If number of days is negative, meaning no delay

if c.days < 0:

# It will be stored as 0 days of delay

delay\_dates\_2021.append(0)

else:

# Else, number of days delayed in payment will be recorded in delay\_dates\_2021 list

delay\_dates\_2021.append(c.days)

# Create new column 'Number of days delayed' and show all the number of days delayed from 2021 claim payments

estimated\_Delay\_2021['Number of days delayed'] = delay\_dates\_2021

#estimated\_Delay\_2021

# drop these row indexes where there are no delays in payment made

estimated\_Delay\_2021 = estimated\_Delay\_2021[estimated\_Delay\_2021['Number of days delayed'] != 0]

#estimated\_Delay\_2021

# Set x and y axis into a variable

delay\_xAxis\_2021 = estimated\_Delay\_2021['Amount']

delay\_yAxis\_2021 = estimated\_Delay\_2021['Number of days delayed']

# Scatterplot of the dataframe

plt.scatter(delay\_xAxis\_2021, delay\_yAxis\_2021)

# Define title label

plt.title("Days Delayed for Claim Payments in 2021", fontsize=15)

# Define x-axis label

plt.ylabel("Number of Days", fontsize=15)

# Define y-axis label

plt.xlabel("Amount", fontsize=15)

# Review columns dtypes

results = estimated\_Delay\_2021.dtypes

results

# Convert data type to int for Amount and Number of days delayed

estimated\_Delay\_2021["Amount"] = estimated\_Delay\_2021["Amount"].astype(int)

estimated\_Delay\_2021["Number of days delayed"] = estimated\_Delay\_2021["Number of days delayed"].astype(int)

# Calculate q1 and q3 quantile

q1\_amt\_2021 = estimated\_Delay\_2021["Number of days delayed"].quantile(q = .25)

q3\_amt\_2021 = estimated\_Delay\_2021["Number of days delayed"].quantile(q = .75)

# Calculate IQR

iqr\_amt\_2021 = q3\_amt\_2021 - q1\_amt\_2021

# Upper bound

upper\_2021= q3\_amt\_2021+1.5\*iqr\_amt\_2021

# Lower bound

lower\_2021= q1\_amt\_2021-1.5\*iqr\_amt\_2021

# To remove outliers

estimated\_Delay\_2021\_outlier = estimated\_Delay\_2021[~((estimated\_Delay\_2021["Number of days delayed"] < lower\_2021) | (estimated\_Delay\_2021["Number of days delayed"] > upper\_2021))]

estimated\_Delay\_2021\_outlier

# Set x and y axis into a variable

delay\_xAxis\_2021 = estimated\_Delay\_2021\_outlier["Amount"]

delay\_yAxis\_2021 = estimated\_Delay\_2021\_outlier["Number of days delayed"]

# Test train split

x\_train, x\_test, y\_train, y\_test = train\_test\_split(delay\_xAxis\_2021, delay\_yAxis\_2021)

# Scatter plot visualisation

plt.scatter(x\_train, y\_train, label='Training Data', color='g')

plt.scatter(x\_test, y\_test, label='Testing Data', color='b')

plt.legend()

plt.title("Test Train 2021")

# Define x-axis label

plt.ylabel("Number of Days", fontsize=15)

# Define y-axis label

plt.xlabel("Amount", fontsize=15)

plt.show()

# Draw a linear regression model and train it

reg = LinearRegression()

reg.fit(x\_train.values.reshape(-1,1), y\_train.values)

# Scatter plot visualisation with linear regression model

prediction = reg.predict(x\_test.values.reshape(-1,1))

plt.plot(x\_test, prediction, label='Linear Regression', color='r')

plt.scatter(x\_test, y\_test, label = "Actual Test Data", color='b')

# Define title label

plt.title("Days Delayed for Claim Payments in 2021", fontsize=15)

# Define x-axis label

plt.ylabel("Number of Days", fontsize=15)

# Define y-axis label

plt.xlabel("Amount", fontsize=15)

plt.legend(loc='upper center', bbox\_to\_anchor=(0.5, -0.13), ncol=2)

plt.show()

# Create new Dataframe with columns from 'Amount','Planned\_Date' & 'Actual\_Date' from 2022

estimated\_Delay\_2022 = filtered\_df\_2022[['Amount','Planned\_Date','Actual\_Date']]

# Create new list to store all the dates from 'Planned\_Date' column in 2022

all\_planned\_dates\_2022 = list(estimated\_Delay\_2022['Planned\_Date'])

# Create new list to store all the dates from 'Actual\_Date' column in 2022

all\_actual\_dates\_2022 = list(estimated\_Delay\_2022['Actual\_Date'])

# Create new list to store delay days for 2021 claims

delay\_dates\_2022 = []

# iterates over 2 lists in 2021 to calculate delay days

for (a, b) in zip(all\_planned\_dates\_2022, all\_actual\_dates\_2022):

c = b - a

# If number of days is negative, meaning no delay

if c.days < 0:

# It will be stored as 0 days of delay

delay\_dates\_2022.append(0)

else:

# Else, number of days delayed in payment will be recorded in delay\_dates\_2021 list

delay\_dates\_2022.append(c.days)

# Create new column 'Number of days delayed' and show all the number of days delayed from 2022 claim payments

estimated\_Delay\_2022['Number of days delayed'] = delay\_dates\_2022

# drop these row indexes where there are no delays in payment made

estimated\_Delay\_2022 = estimated\_Delay\_2022[estimated\_Delay\_2022['Number of days delayed'] != 0]

# Set x and y axis into a variable

delay\_xAxis\_2022 = estimated\_Delay\_2022['Amount']

delay\_yAxis\_2022 = estimated\_Delay\_2022['Number of days delayed']

# Scatterplot of the dataframe

plt.scatter(delay\_xAxis\_2022, delay\_yAxis\_2022)

# Define title label

plt.title("Days Delayed for Claim Payments in 2022", fontsize=15)

# Define x-axis label

plt.ylabel("Number of Days", fontsize=15)

# Define y-axis label

plt.xlabel("Amount", fontsize=15)

# Review columns dtypes

results = estimated\_Delay\_2022.dtypes

results

# Convert data type to int for Amount and Number of days delayed

estimated\_Delay\_2022["Amount"] = estimated\_Delay\_2022["Amount"].astype(int)

estimated\_Delay\_2022["Number of days delayed"] = estimated\_Delay\_2022["Number of days delayed"].astype(int)

# Calculate q1 and q3 quantile

q1\_amt\_2022 = estimated\_Delay\_2022["Number of days delayed"].quantile(q = .25)

q3\_amt\_2022 = estimated\_Delay\_2022["Number of days delayed"].quantile(q = .75)

# Calculate IQR

iqr\_amt\_2022 = q3\_amt\_2022 - q1\_amt\_2022

# Upper bound

upper\_2022= q3\_amt\_2022+1.5\*iqr\_amt\_2022

# Lower bound

lower\_2022= q1\_amt\_2022-1.5\*iqr\_amt\_2022

# To remove outliers

estimated\_Delay\_2022\_outlier = estimated\_Delay\_2022[~((estimated\_Delay\_2022["Number of days delayed"] < lower\_2022) | (estimated\_Delay\_2022["Number of days delayed"] > upper\_2022))]

estimated\_Delay\_2022\_outlier

# Set x and y axis into a variable

delay\_xAxis\_2022 = estimated\_Delay\_2022\_outlier["Amount"]

delay\_yAxis\_2022 = estimated\_Delay\_2022\_outlier["Number of days delayed"]

# Test train split

x\_train2, x\_test2, y\_train2, y\_test2 = train\_test\_split(delay\_xAxis\_2022, delay\_yAxis\_2022)

# Scatter plot visualisation

plt.scatter(x\_train2, y\_train2, label='Training Data', color='g')

plt.scatter(x\_test2, y\_test2, label='Testing Data', color='b')

plt.legend()

plt.title("Test Train 2022")

# Define x-axis label

plt.ylabel("Number of Days", fontsize=15)

# Define y-axis label

plt.xlabel("Amount", fontsize=15)

plt.show()

# Draw a linear regression model and train it

reg = LinearRegression()

reg.fit(x\_train2.values.reshape(-1,1), y\_train2.values)

# Scatter plot visualisation with linear regression model

prediction = reg.predict(x\_test2.values.reshape(-1,1))

plt.plot(x\_test2, prediction, label='Linear Regression', color='r')

plt.scatter(x\_test2, y\_test2, label = "Actual Test Data", color='b')

# Define title label

plt.title("Days Delayed for Claim Payments in 2022", fontsize=15)

# Define x-axis label

plt.ylabel("Number of Days", fontsize=15)

# Define y-axis label

plt.xlabel("Amount", fontsize=15)

plt.legend(loc='upper center', bbox\_to\_anchor=(0.5, -0.13), ncol=2)

plt.show()