Higher National Diploma in Data Science.

Machine Learning.

Module leader: Eng. Chameera De Silva.

COLOMBO-HNDDS-FT-23.1

2023.05.30

M.M. Mohamed Sabath_CONHDDS23.1F-007

Acknowledgement

This undertaking could not have been completed without the assistance and direction of a large number of individuals. I would like to thank each of them from the bottom of my heart.
I appreciate my mentor, Mr. Chameera De Silva, for providing me with this opportunity.

Outstanding opportunity to oversee and direct this assignment. I would also like to thank you for providing us with the necessary information and resources to complete this project. I was able to complete this en in large part because of the wonderful cooperation and support of my friends and family, to whom I would like to express my appreciation.

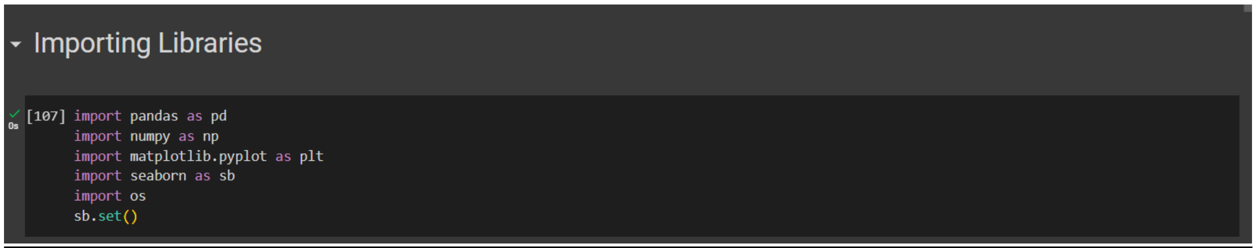
Contents

Introduction

This task involves using a secondary banking data set, preprocessing it, and then using a machine learning model to predict whether a customer will sign up for a term deposit (yes/no) based on their demographic data (age, job, marital status, education, etc.), as well as their prior banking history (number of contacts with the bank, results of previous marketing campaigns, etc.).

Step by step Code Explanation

1) **Importing the relevant libraries.**

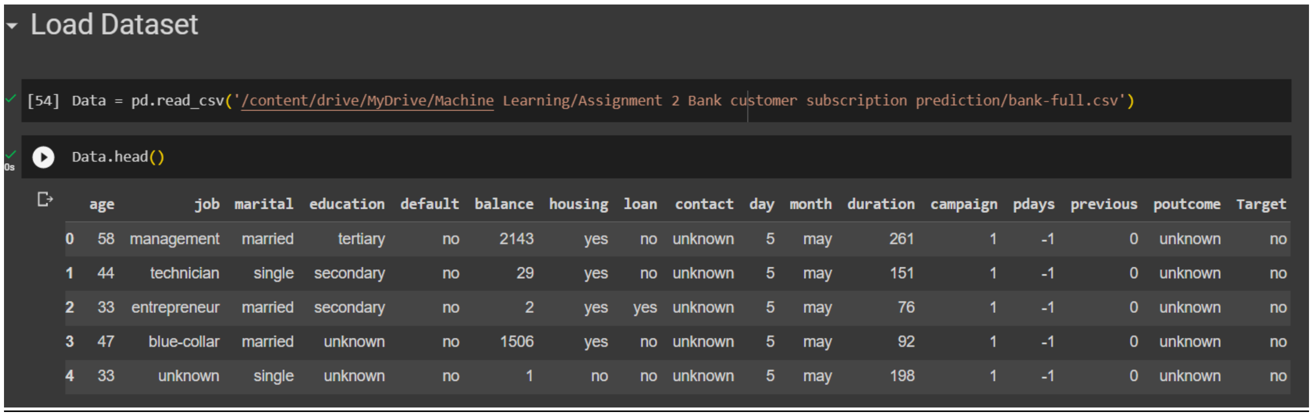


Importing Libraries

```
[107] import pandas as pd
      import numpy as np
      import matplotlib.pyplot as plt
      import seaborn as sb
      import os
      sb.set()
```

This is the first and the key important step for creating the model. In here we have to import all the relevant python libraries in order to work with the dataset.

2) Loading and printing the Data set in the notebook



Load Dataset

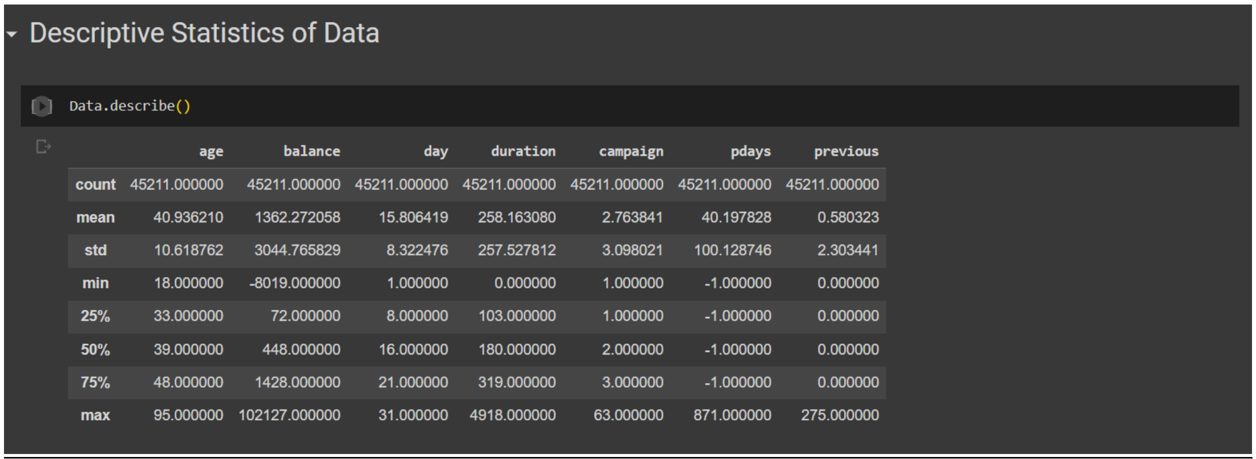
```
[54] Data = pd.read_csv('/content/drive/MyDrive/Machine Learning/Assignment 2 Bank customer subscription prediction/bank-full.csv')
```

```
Data.head()
```

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	Target
0	58	management	married	tertiary	no	2143	yes	no	unknown	5	may	261	1	-1	0	unknown	no
1	44	technician	single	secondary	no	29	yes	no	unknown	5	may	151	1	-1	0	unknown	no
2	33	entrepreneur	married	secondary	no	2	yes	yes	unknown	5	may	76	1	-1	0	unknown	no
3	47	blue-collar	married	unknown	no	1506	yes	no	unknown	5	may	92	1	-1	0	unknown	no
4	33	unknown	single	unknown	no	1	no	no	unknown	5	may	198	1	-1	0	unknown	no

In here we have to load the dataset from the google drive. For that, we have to get the location of the data set file path which is exactly located at the drive and then we have to paste it correctly in order to read the csv file.

3) Running the descriptive statistics for the Data set



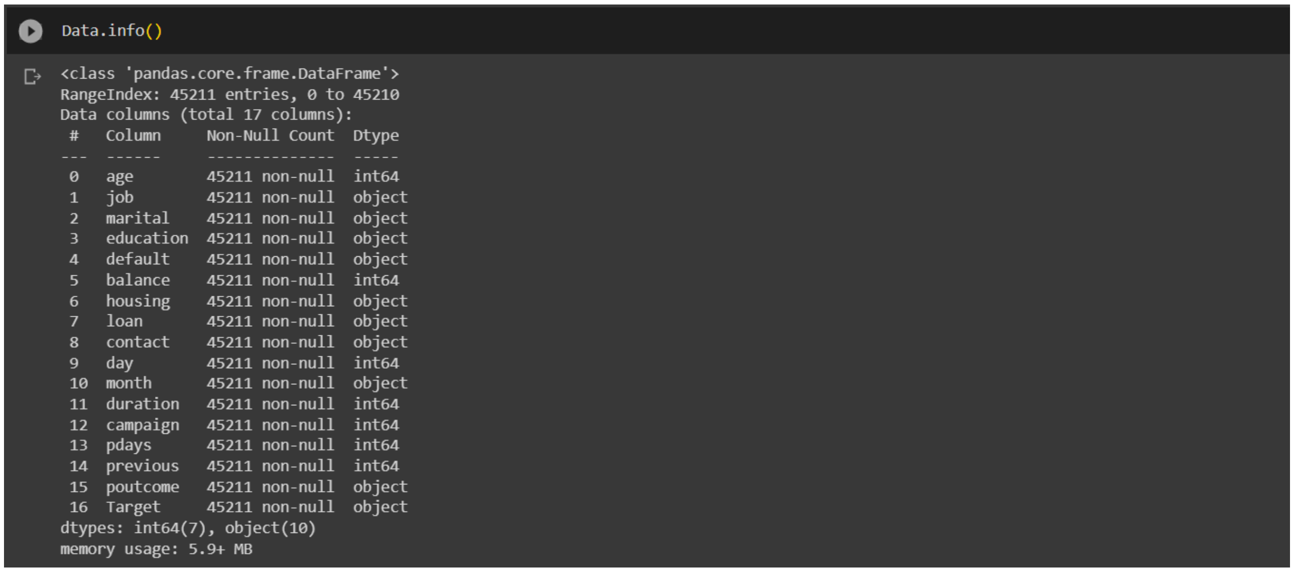
Descriptive Statistics of Data

```
Data.describe()
```

	age	balance	day	duration	campaign	pdays	previous
count	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000
mean	40.936210	1362.272058	15.806419	258.163080	2.763841	40.197828	0.580323
std	10.618762	3044.765829	8.322476	257.527812	3.098021	100.128746	2.303441
min	18.000000	-8019.000000	1.000000	0.000000	1.000000	-1.000000	0.000000
25%	33.000000	72.000000	8.000000	103.000000	1.000000	-1.000000	0.000000
50%	39.000000	448.000000	16.000000	180.000000	2.000000	-1.000000	0.000000
75%	48.000000	1428.000000	21.000000	319.000000	3.000000	-1.000000	0.000000
max	95.000000	102127.000000	31.000000	4918.000000	63.000000	871.000000	275.000000

This step is about running a descriptive analysis of the data set and in here in this step we can get the description about the data set such as count, mean, standard deviation, minimum data point, 25th percentile of the data, 50th percentile of the data which is median , 75th percentile of the data and the maximum point of the data.

4) Checking information about the columns in the dataset.

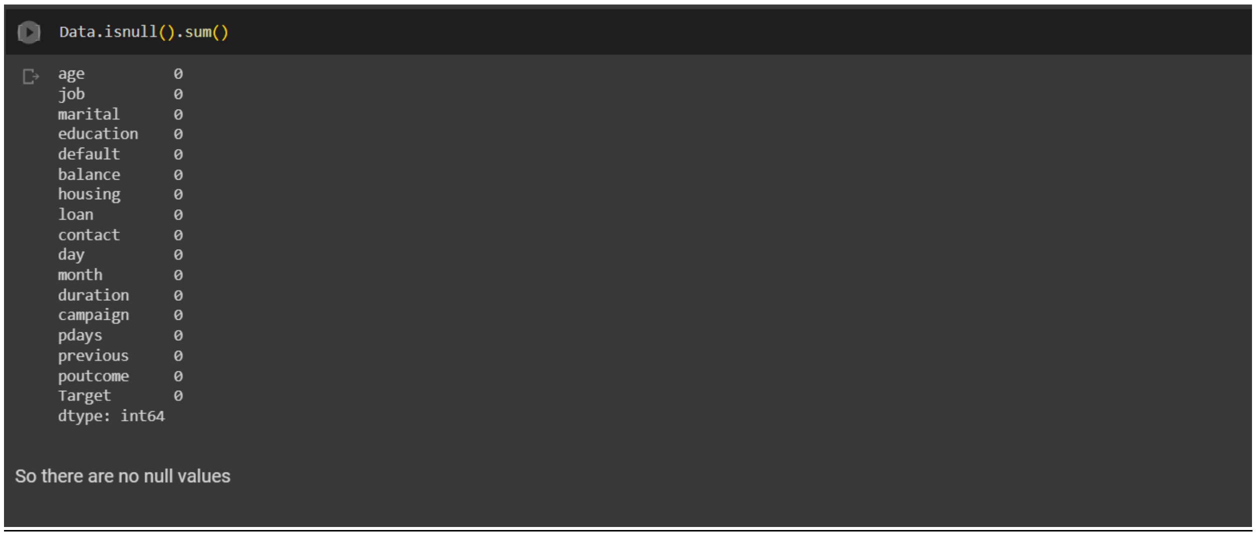


```
Data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45211 entries, 0 to 45210
Data columns (total 17 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   age        45211 non-null  int64 
 1   job        45211 non-null  object
 2   marital    45211 non-null  object
 3   education  45211 non-null  object
 4   default    45211 non-null  object
 5   balance    45211 non-null  int64 
 6   housing    45211 non-null  object
 7   loan       45211 non-null  object
 8   contact    45211 non-null  object
 9   day        45211 non-null  int64 
 10  month      45211 non-null  object
 11  duration   45211 non-null  int64 
 12  campaign   45211 non-null  int64 
 13  pdays      45211 non-null  int64 
 14  previous   45211 non-null  int64 
 15  poutcome   45211 non-null  object
 16  Target     45211 non-null  object
dtypes: int64(7), object(10)
memory usage: 5.9+ MB
```

In here we can get all the information about every column such as whether It has any null values or not and the data type of the variable and the observations or the number of data points in every particular column. So that here we can check the number of columns as well.

5) Checking the null values



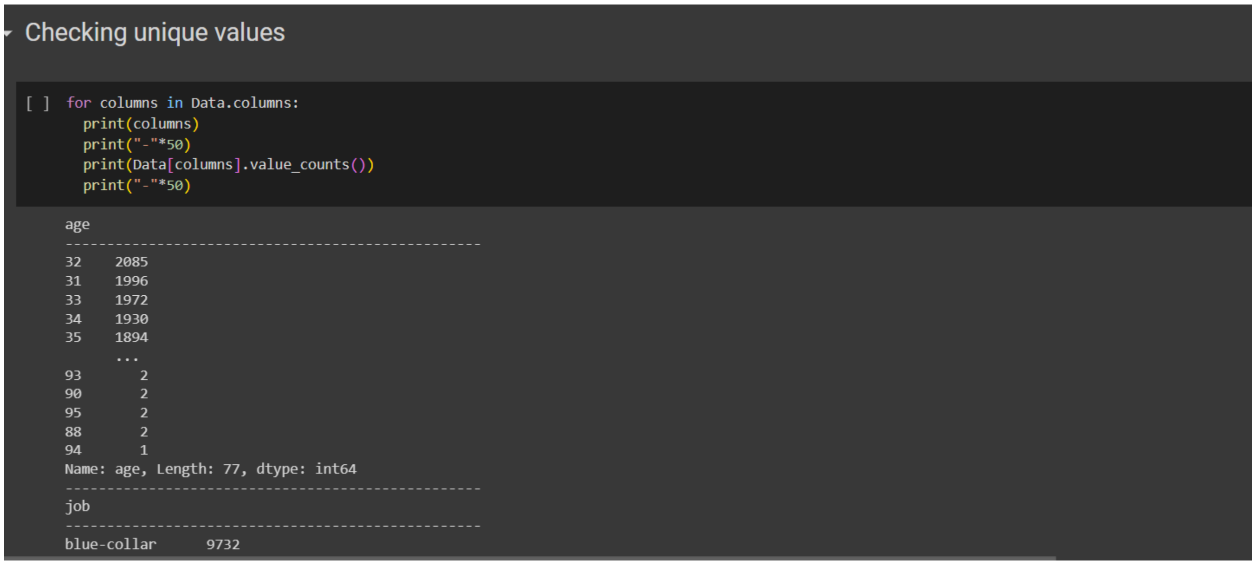
```
Data.isnull().sum()
```

```
age          0
job          0
marital      0
education    0
default      0
balance      0
housing      0
loan         0
contact      0
day          0
month        0
duration     0
campaign     0
pdays        0
previous     0
poutcome     0
Target       0
dtype: int64
```

So there are no null values

This is a part of preprocessing. In here we are checking is there any null values in the entire data set. It shows that there are no null values in the data set. So we can move further on the analysis part.

6) Checking the unique values



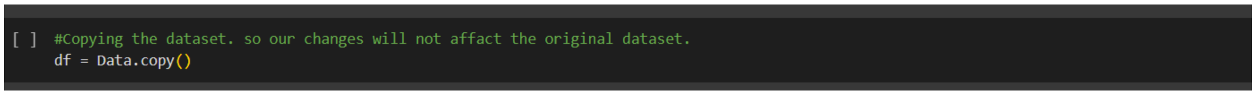
Checking unique values

```
[ ] for columns in Data.columns:
      print(columns)
      print("-"*50)
      print(Data[columns].value_counts())
      print("-"*50)
```

```
age
--------------------------------------------------
32    2085
31    1996
33    1972
34    1930
35    1894
      ...
93       2
90       2
95       2
88       2
94       1
Name: age, Length: 77, dtype: int64
--------------------------------------------------
job
--------------------------------------------------
blue-collar      9732
```

In this step we are checking every unique value of every variable. So here if there are any "unknown" values, we can find it. When we see the above code, it is about a for loop which has assigned to print column name first and then it is printing 50'-'in order to separate the column name and the variables and then it is printing the every unique values of that column and the relevant count of that unique value.

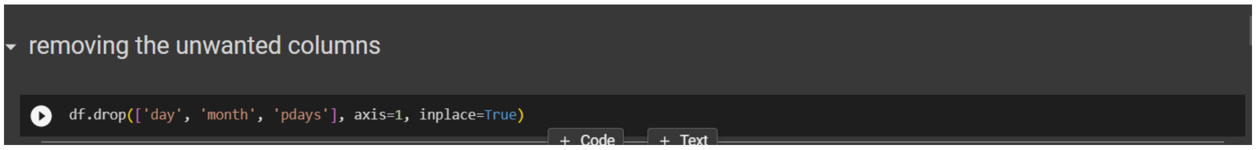
7) Copying the dataset and assigning it to another variable called df



```
[ ] #Copying the dataset. so our changes will not affact the original dataset.
    df = Data.copy()
```

This is not an essential step. This is about, for a safety purpose I am copying the data set and assigning it to another variable called "df" so for my further analysis I can use that copied data, the analysis will not affect the main data.

8) Removing unwanted columns



```
df.drop(['day', 'month', 'pdays'], axis=1, inplace=True)
```

In this step I am removing the ‘day’, ‘month’, ‘pdays’ columns. Because the month and the day is about when the person has called lastly to the bank. So it might not have that much impact on the model. The pdays also is about the number of days that passed by after the individual was last contacted from a previous campaign. So that might also will not affect the model.

9) Checking outliers

When developing a machine learning (ML) model, it is crucial to identify and eliminate data outliers. The performance and generalizability of a model might be adversely affected by outliers, which are data points that differ greatly from the overall majority of the observations. Reasons for eliminating outliers are,

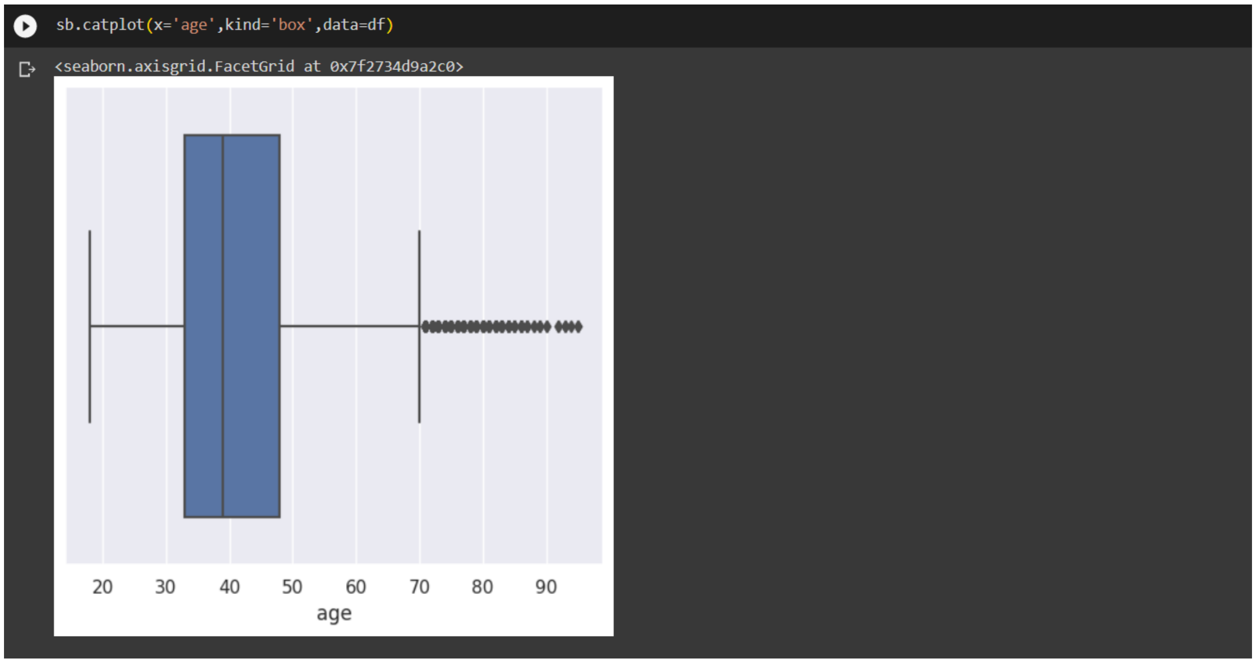
Accuracy and performance of the model: Outliers can distort data by introducing bias and noise. Machine learning (ML) algorithms are programmed to analyze large amounts of data in order to draw conclusions. Unusual or extreme results, known as outliers, can skew the data and make good prediction difficult. When outliers are eliminated, the ML model's accuracy and efficiency are both enhanced.

Many ML methods, for example, assume the data follows a normal distribution or has homoscedastic variance (constant). The model's reliability gets compromised when outliers cause it to deviate from these expectations. We can improve the confidence of the model's outcomes and predictions by removing outliers and thereby ensuring that the data conforms to the assumptions of the ML algorithm.

Generalizability and robustness are two key goals of ML models. Due to their unusual nature, outliers may not be reflective of the data the model will see in the future. We can make the model more robust and able to generalize to new cases by removing outliers so that it can concentrate on learning the regular patterns and correlations in the data.

The influence of outliers on the predicted model parameters can be substantial. Optimization approaches are frequently used by ML algorithms to estimate the model's parameters; nevertheless, outliers can have an outsized effect on these estimates, resulting in unstable and untrustworthy models. Parameter estimates from a model are more stable after outliers are removed.

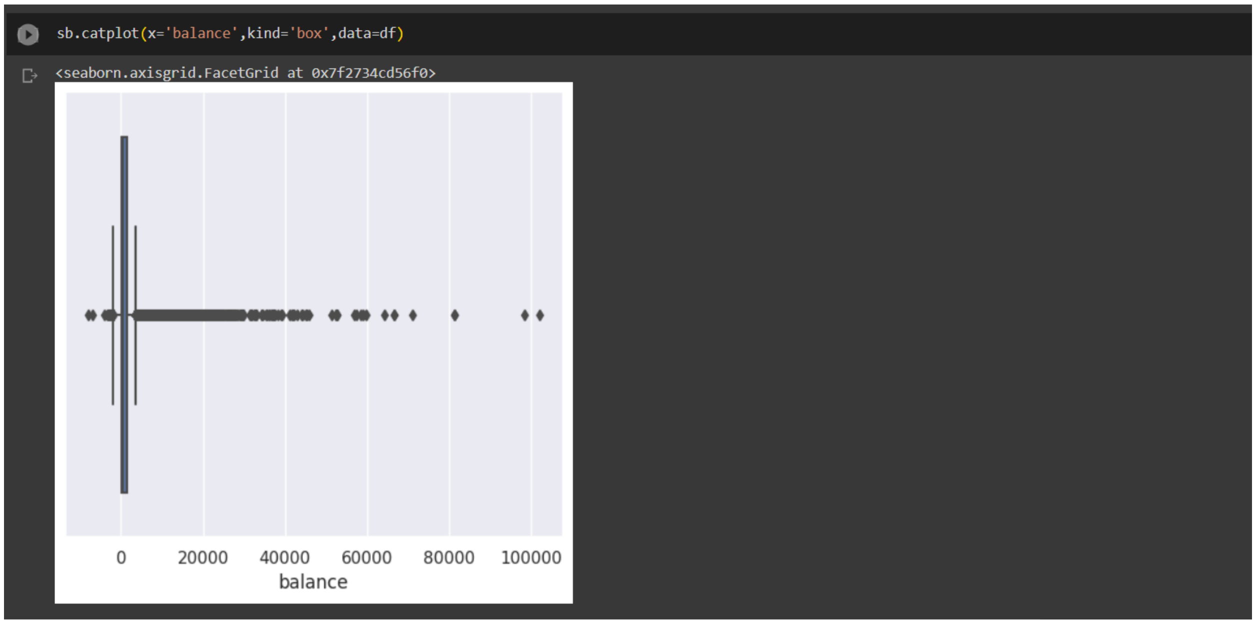
In terms of interpretability, outliers can introduce false conclusions. Finding important connections and insights in the data is a common use of ML models. These associations can be skewed by outliers and lead to incorrect conclusions if they aren't addressed. The model's interpretability and the reliability of its insights are both enhanced by the exclusion of outliers.



```
sb.catplot(x='age',kind='box',data=df)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f2734d9a2c0>
```

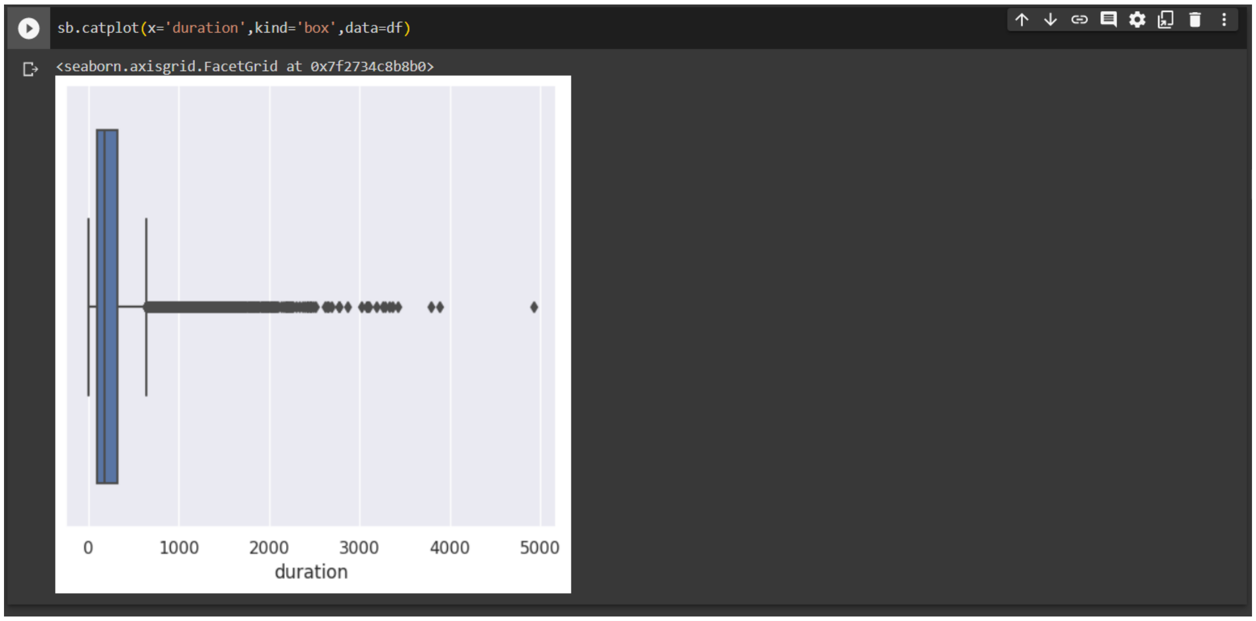
In the above code and the image, I checked whether is there any outliers in the “age” variable or not using cat plot. In the above image we can clearly say that there are outliers in the age variables. So to handle that we have to remove those outliers.



```
sb.catplot(x='balance',kind='box',data=df)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f2734cd56f0>
```

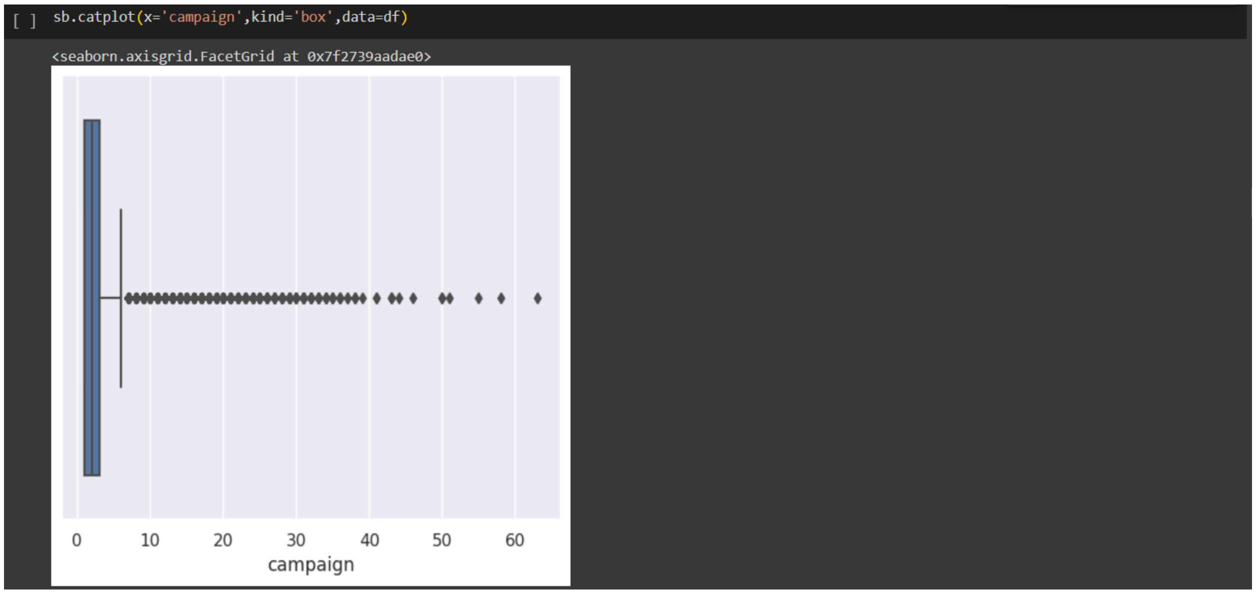
In the above I mage and the code I have checked Is there any outliers in the ‘balance’ variable or not. In this variable also we can clearly see that there are so much of outliers in that variable. So, in order to deal that we have to remove those outliers.



```
sb.catplot(x='duration',kind='box',data=df)
```

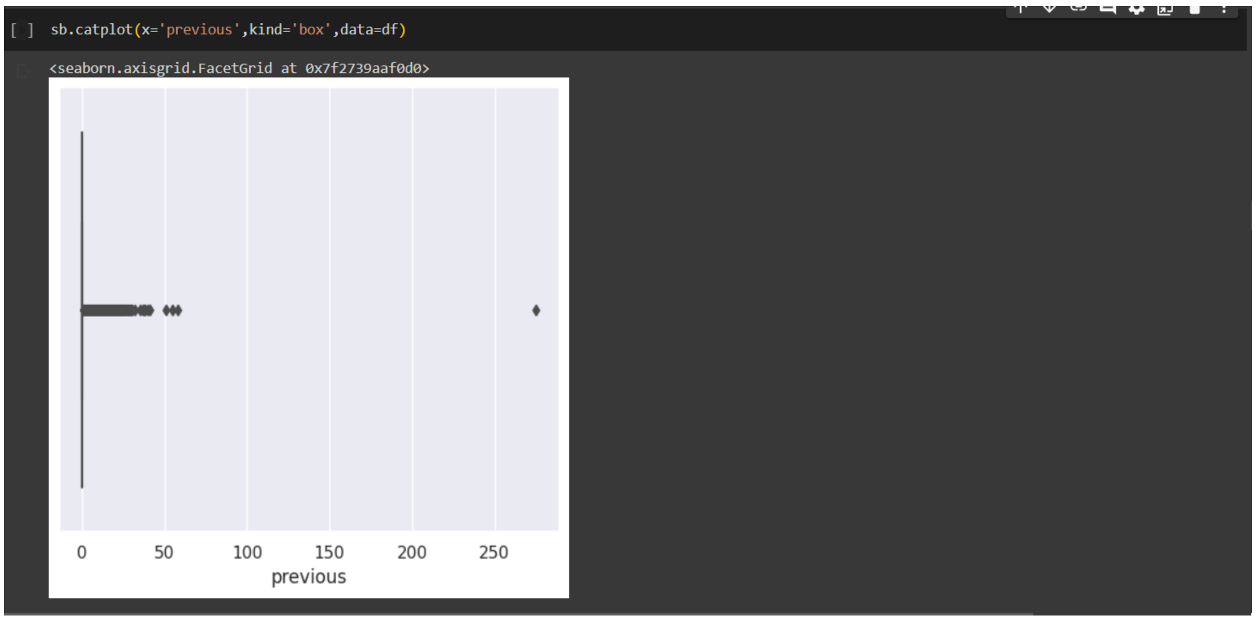
```
<seaborn.axisgrid.FacetGrid at 0x7f2734c8b8b0>
```

In the above code and the image I have checked Is there any outliers in the ‘duration’ variable or not. In this variable also we can clearly see that there are so much of outlier value in that variable. So, in order to deal that we have to remove those outliers too.



```
[ ]  sb.catplot(x='campaign',kind='box',data=df)

<seaborn.axisgrid.FacetGrid at 0x7f2739aadae0>
```

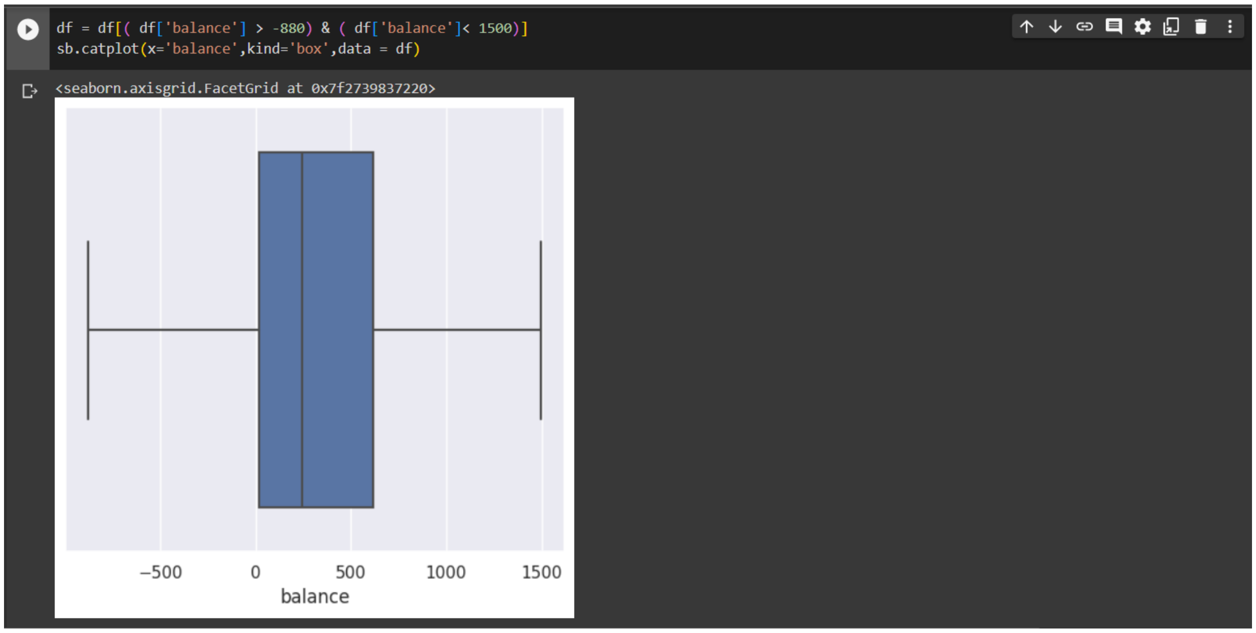
In the above code and the image I have checked Is there any outliers in the ‘campaign’ variable or not. In this variable also we can clearly see that there are so much of outlier value in that variable. So, in order to deal that we have to remove those outliers too.



```
[ ]  sb.catplot(x='previous',kind='box',data=df)

<seaborn.axisgrid.FacetGrid at 0x7f2739aaf0d0>
```

In the above code and the image I have checked whether is there any outliers value in the dataset for that ‘previous’ variable that I am going to use to create the model. So I have checked the outliers for the variables such as ‘age’, ‘balance’, ’duration’, ’campaign’, ‘previous’ variables. So I found outliers for those variables.

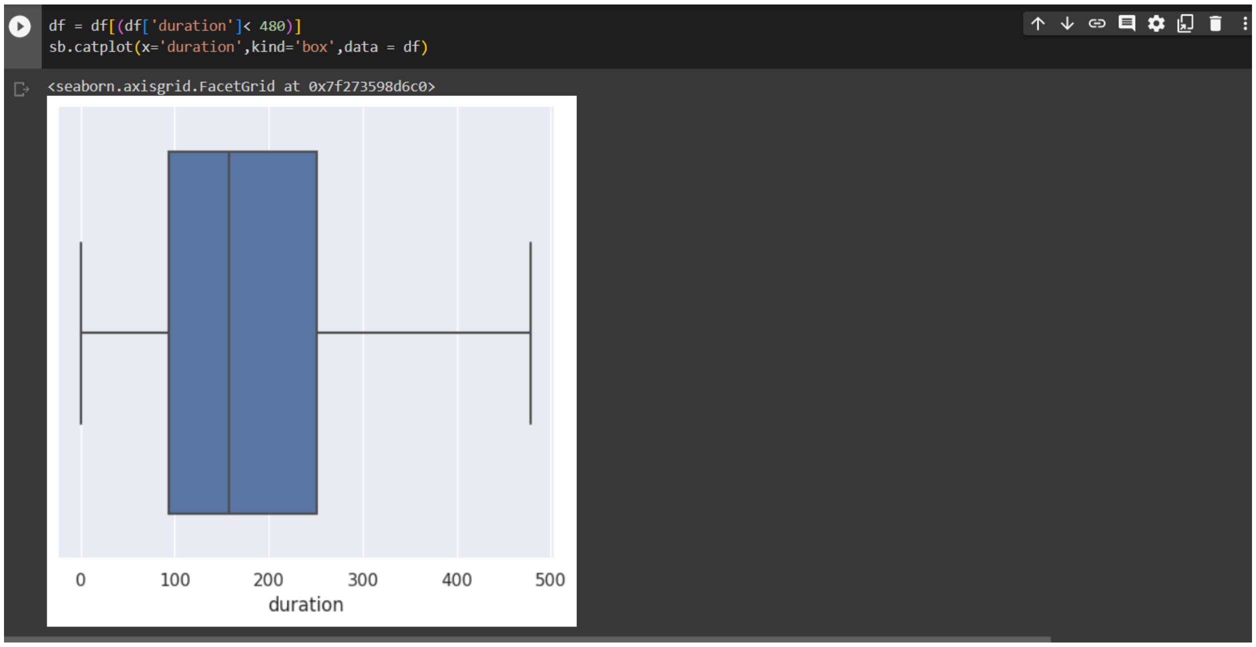
10) Removing outliers



```
df = df[( df['balance'] > -880) & ( df['balance']< 1500)]
sb.catplot(x='balance',kind='box',data = df)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f2739837220>
```

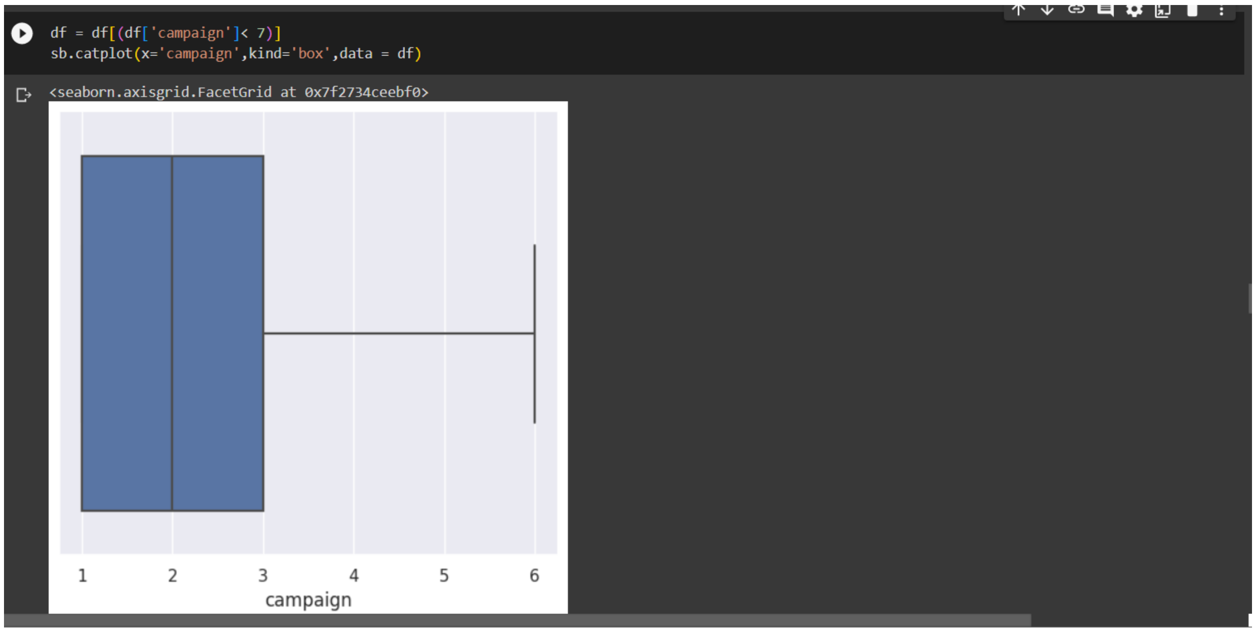
So here in this above code and the image I have removed the outliers which can affect our ML model that I found in the previous step for the balance variable



```
df = df[(df['duration']< 480)]
sb.catplot(x='duration',kind='box',data = df)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f273598d6c0>
```

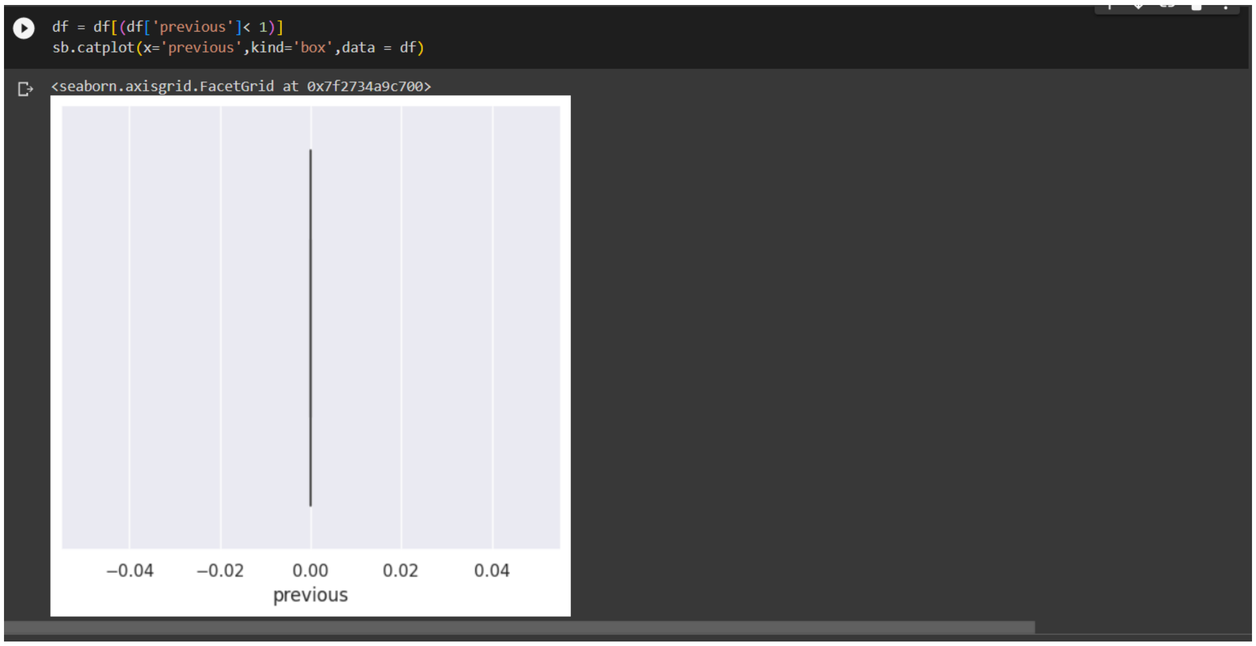
So here in this above code and the image I have removed the outliers which can affect our ML model that I found in the 'duration' variable.



```
df = df[(df['campaign']< 7)]
sb.catplot(x='campaign',kind='box',data = df)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f2734ceebf0>
```

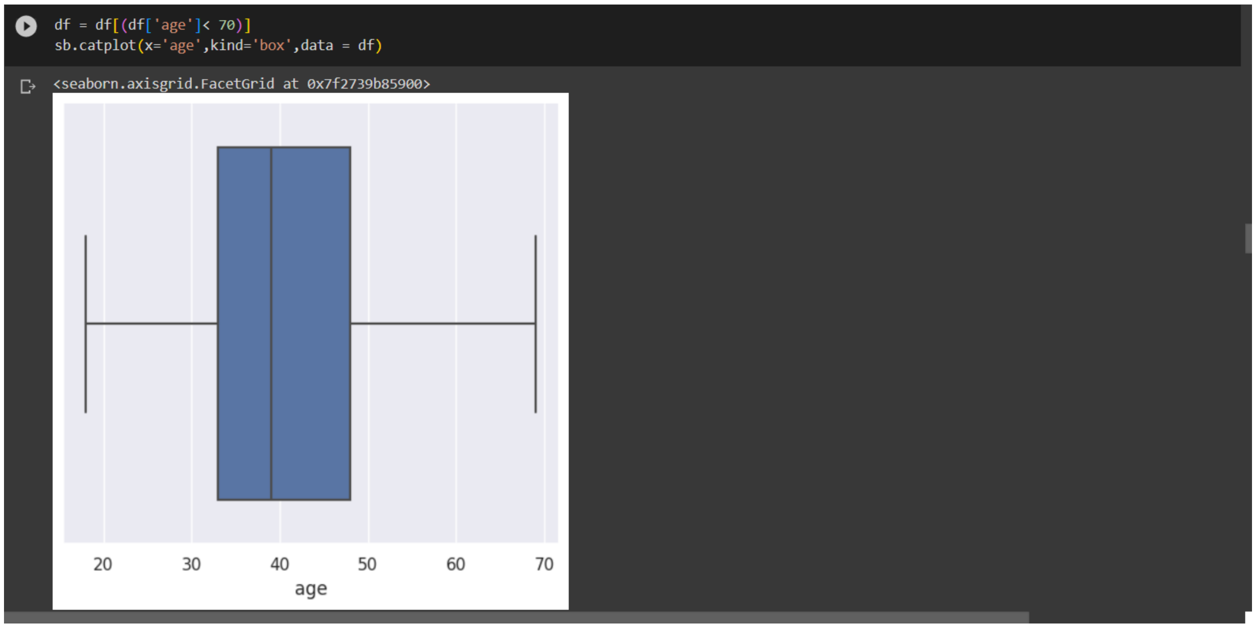
So here in this above code and the image I have removed the outliers which can affect our ML model that I found in the ‘campaign’ variable.



```
df = df[(df['previous']< 1)]
sb.catplot(x='previous',kind='box',data = df)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f2734a9c700>
```

So here in this above code and the image I have removed the outliers which can affect our ML model that I found in the ‘previous’ variable.

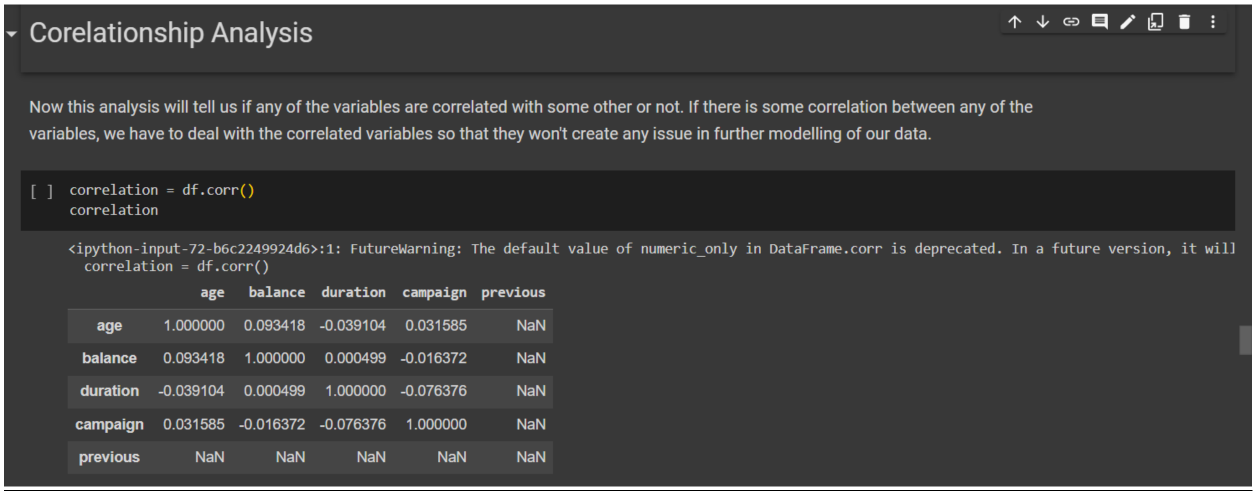


```
df = df[(df['age']< 70)]
sb.catplot(x='age',kind='box',data = df)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f2739b85900>
```

In this above code and the image I have removed the outliers which can affect our ML model that I found in the 'age' variable.

11) Correlation analysis



Corelationship Analysis

Now this analysis will tell us if any of the variables are correlated with some other or not. If there is some correlation between any of the variables, we have to deal with the correlated variables so that they won't create any issue in further modelling of our data.

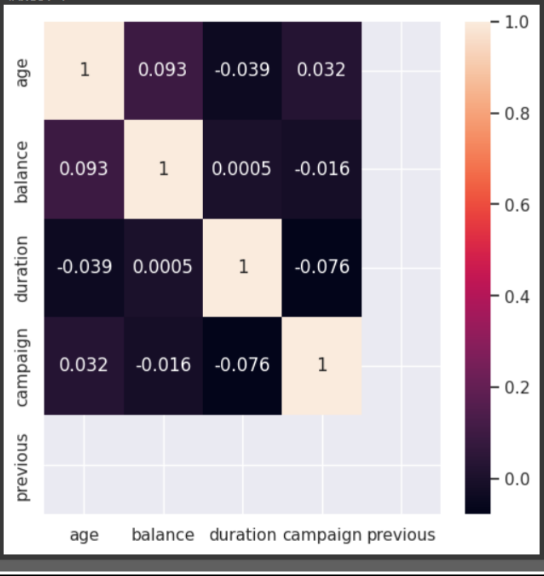
```
[ ]  correlation = df.corr()
     correlation
```

```
<ipython-input-72-b6c2249924d6>:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will
  correlation = df.corr()
```

	age	balance	duration	campaign	previous
age	1.000000	0.093418	-0.039104	0.031585	NaN
balance	0.093418	1.000000	0.000499	-0.016372	NaN
duration	-0.039104	0.000499	1.000000	-0.076376	NaN
campaign	0.031585	-0.016372	-0.076376	1.000000	NaN
previous	NaN	NaN	NaN	NaN	NaN

```
sb.heatmap(correlation, xticklabels = correlation.columns, yticklabels=correlation.columns, annot=True)
```

```
<Axes: >
```



In this step, I have checked the relationship between the numerical variables, and visualised the relationship of them using a table and a heatmap. The correlation coefficients, which in this case fall on a scale from -1 to 1, show how strongly and in what direction two variables are linearly related to one another. Coefficients of 1 and -1 show perfectly positive and negative correlation, respectively, whereas a coefficient of 0 shows no linear association. By visually representing the correlation between variables in a dataset with a heatmap, we can learn more about the interrelationships between the data points. A heatmap displays the correlation matrix graphically, with colors denoting the strength and direction of the associations between variables.

Using a heatmap to represent the correlation matrix helps researchers spot trends and connections between variables. The heatmap's color-coding makes it easy to see at a glance which factors are associated with one another, whether positively or negatively.

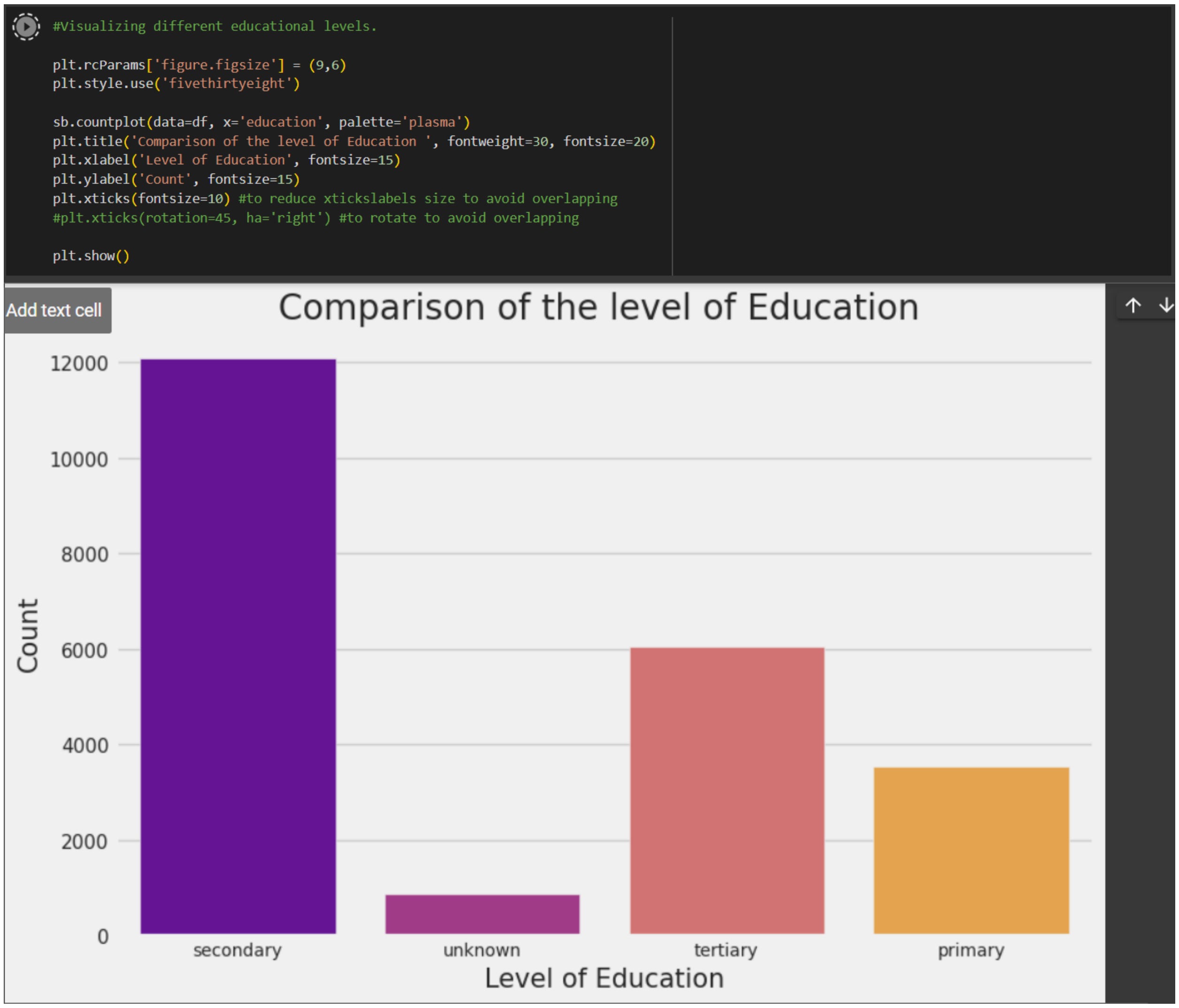
Heatmaps are very useful when working with huge datasets that contain several variables. They offer a clear and simple approach to analyzing interrelationships, which is crucial for spotting emerging patterns and insights.

We look for color patterns when interpreting a heatmap. A significant correlation is shown by a black color, whereas a lack of correlation is shown by a white color.

Overall, data analysts and researchers can benefit from heatmapping correlations since it is a straightforward and simple way to examine and comprehend the links between variables in a dataset.

In this case, if we see any highly correlated variable, we have to remove one of them. But we can clearly see that there are no that much correlation between any variables that can affect the model.

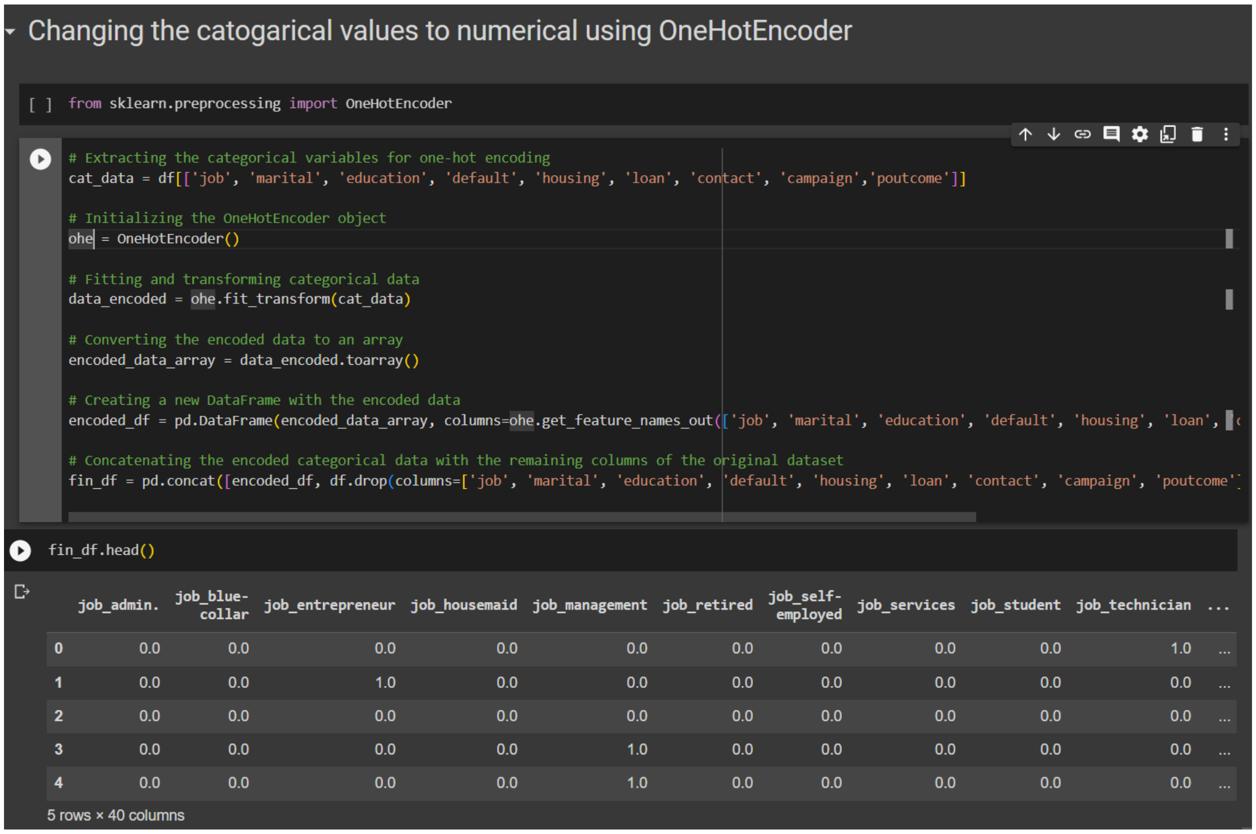
12) Visualizing the variables



```
#Visualizing different educational levels.

plt.rcParams['figure.figsize'] = (9,6)
plt.style.use('fivethirtyeight')

sb.countplot(data=df, x='education', palette='plasma')
plt.title('Comparison of the level of Education ', fontweight=30, fontsize=20)
plt.xlabel('Level of Education', fontsize=15)
plt.ylabel('Count', fontsize=15)
plt.xticks(fontsize=10) #to reduce xtickslabels size to avoid overlapping
#plt.xticks(rotation=45, ha='right') #to rotate to avoid overlapping

plt.show()
```

In this step I have visualized the level of education of the customers. So for this I have used Bar plot. In this above image we can clearly see that most of the people are from secondary level of education.

13) Changing the categorical variable as numerical using using OneHotEncoder



Changing the catogarical values to numerical using OneHotEncoder

```
from sklearn.preprocessing import OneHotEncoder
```

```
# Extracting the categorical variables for one-hot encoding
cat_data = df[['job', 'marital', 'education', 'default', 'housing', 'loan', 'contact', 'campaign','poutcome']]

# Initializing the OneHotEncoder object
ohe = OneHotEncoder()

# Fitting and transforming categorical data
data_encoded = ohe.fit_transform(cat_data)

# Converting the encoded data to an array
encoded_data_array = data_encoded.toarray()

# Creating a new DataFrame with the encoded data
encoded_df = pd.DataFrame(encoded_data_array, columns=ohe.get_feature_names_out(['job', 'marital', 'education', 'default', 'housing', 'loan',

# Concatenating the encoded categorical data with the remaining columns of the original dataset
fin_df = pd.concat([encoded_df, df.drop(columns=['job', 'marital', 'education', 'default', 'housing', 'loan', 'contact', 'campaign', 'poutcome'
```

```
fin_df.head()
```

	job_admin.	job_blue-collar	job_entrepreneur	job_housemaid	job_management	job_retired	job_self-employed	job_services	job_student	job_technician	...
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	...
1	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...
3	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...
4	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...

5 rows × 40 columns

In the above code, i have used the OneHotEncoder, a popular method for processing categorical input in machine learning models, to translate categorical variables into numerical values.In this step I converted all the categorical variables as numerical values using OneHotEncoder in order to create a logistic model.
In the above coding, I have created a new DataFrame called "cat_data" that contains only the relevant categorical variables (such as "job," "marital," "education," "default," "housing," "loan," "contact," "campaign," and "poutcome").

Then, I have created a 'ohe' object of the OneHotEncoder class. To make advantage of machine learning methods, it is usual practice to transform categorical variables into a binary matrix representation using the OneHotEncoder.

The OneHotEncoder is used to fit and transform the categorical data in 'cat_data' using the 'ohe.fit_transform(cat_data)' command. In this process, a sparse matrix representation is created by giving integer values to each distinct category within each variable.

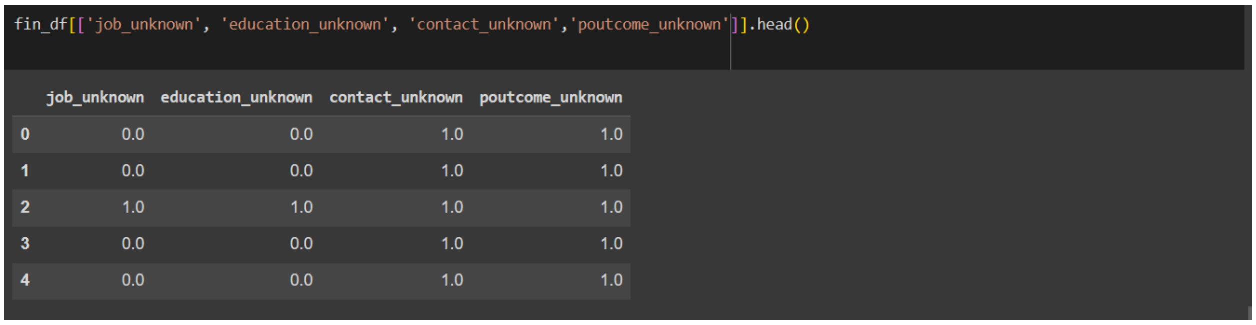
'data_encoded.toarray()' is used to transform the encoded information from a sparse matrix into a dense array format. This generates a matrix with two dimensions, where each row represents a data point and each column represents a certain category inside a variable.

To do this, we use the 'encoded_data_array' to build a new DataFrame called 'encoded_df' and then call 'ohe.get_feature_names_out()' on the OneHotEncoder object to get the names of the columns to use in the new DataFrame. Each column in this DataFrame corresponds to an encoded categorical variable.

Finally, I have used 'pd.concat()' to join the remaining columns of the original DataFrame 'df' (without the categorical variables) with the encoded categorical data in 'encoded_df'. If we want to analyze the data further or train a logistic model, we can use the resultant DataFrame ('fin_df') that contains both the original columns and the encoded categorical variables.

i have made it easier for machine learning algorithms to analyze and learn from categorical information by encoding it with OneHotEncoder. In order for the logistic model to gain useful insights from the dataset or make precise predictions, this preprocessing step is necessary.

After that, I have checked the dataset by calling the "head" function to check whether It has worked or not. So we can see that I have separated every unique value of every single column as one column. So that I can remove those "unknown"' values easily. So accordingly, now I have 40 columns.

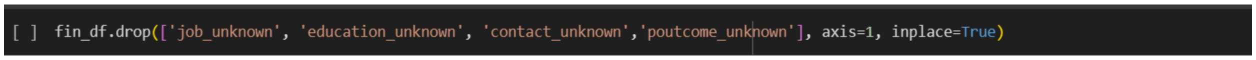


```
fin_df[['job_unknown', 'education_unknown', 'contact_unknown','poutcome_unknown']].head()
```

	job_unknown	education_unknown	contact_unknown	poutcome_unknown
0	0.0	0.0	1.0	1.0
1	0.0	0.0	1.0	1.0
2	1.0	1.0	1.0	1.0
3	0.0	0.0	1.0	1.0
4	0.0	0.0	1.0	1.0

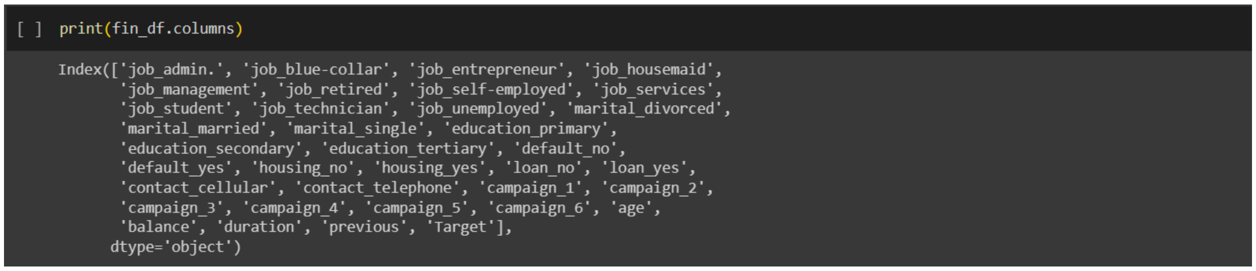
In here, I have checked the columns that are 'Unknown'.

14) Dropping out unknown columns



```
[ ]  fin_df.drop(['job_unknown', 'education_unknown', 'contact_unknown','poutcome_unknown'], axis=1, inplace=True)
```

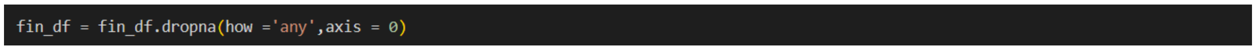
Here I have dropped out the columns which are ‘unknown’ because they might affect our model.



```
[ ]  print(fin_df.columns)

     Index(['job_admin.', 'job_blue-collar', 'job_entrepreneur', 'job_housemaid',
            'job_management', 'job_retired', 'job_self-employed', 'job_services',
            'job_student', 'job_technician', 'job_unemployed', 'marital_divorced',
            'marital_married', 'marital_single', 'education_primary',
            'education_secondary', 'education_tertiary', 'default_no',
            'default_yes', 'housing_no', 'housing_yes', 'loan_no', 'loan_yes',
            'contact_cellular', 'contact_telephone', 'campaign_1', 'campaign_2',
            'campaign_3', 'campaign_4', 'campaign_5', 'campaign_6', 'age',
            'balance', 'duration', 'previous', 'Target'],
           dtype='object')
```

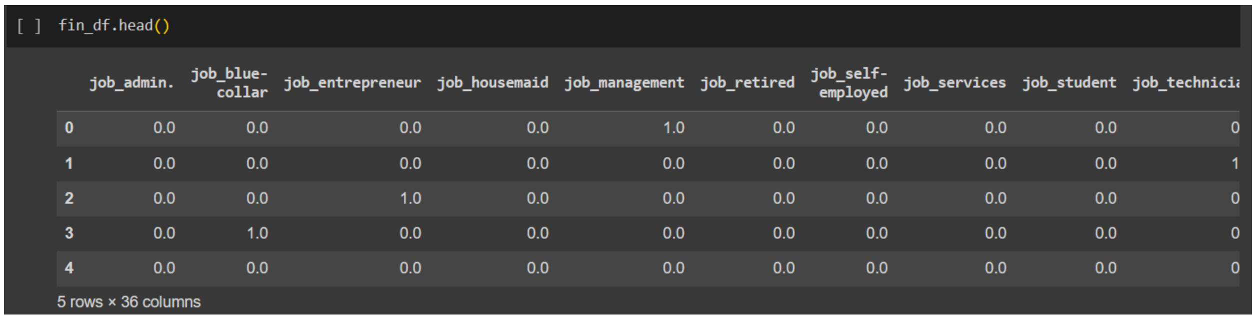
So after dropping out the ‘unknown’columns i checked out the columns to check whether those have been removed or not. For tthat,i have printed all the columns that are in the data set.

15) Removing null values



```
fin_df = fin_df.dropna(how ='any',axis = 0)
```

Eventhough i removed unknown values, i found some null values in the dataset. So in this step i have removed thos null values using this ‘dropna’ function.



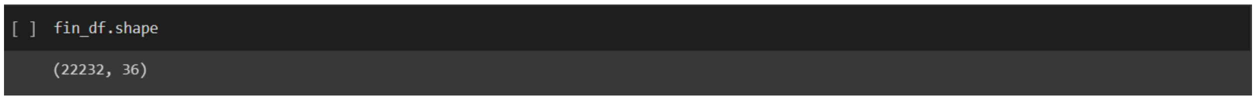
```
[ ]  fin_df.head()
```

	job_admin.	job_blue-collar	job_entrepreneur	job_housemaid	job_management	job_retired	job_self-employed	job_services	job_student	job_technicia
0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1
2	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0
3	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0

5 rows × 36 columns

And then i checked whether those null values are still available or not. but i did not see any null values in that after i removing them.

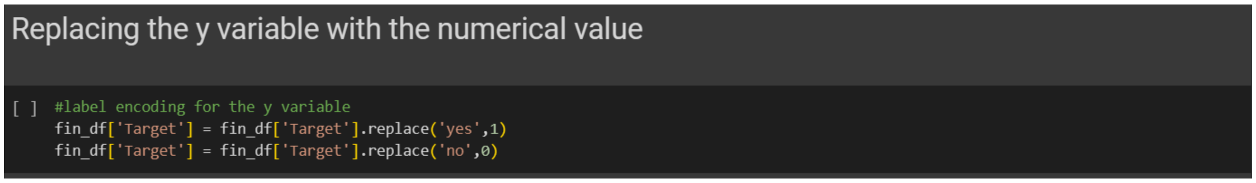
16) Checking the shape of the dataset.



```
[ ]  fin_df.shape

     (22232, 36)
```

In this step after preprocessing the data, i have checked the shape of the data set . So here it shows the number of rows and columns of the dataset.

17) Replacing the y Variable with the numerical values.



Replacing the y variable with the numerical value

```
[ ]  #label encoding for the y variable
     fin_df['Target'] = fin_df['Target'].replace('yes',1)
     fin_df['Target'] = fin_df['Target'].replace('no',0)
```

In this step i have changed the y variable (Target) to the numerical values. I have used label encoding for this. In label encoding, each target variable's category class is represented by a distinct integer label. This is necessary because the logistic regression algorithm expects a numeric value for the dependent variable as input.

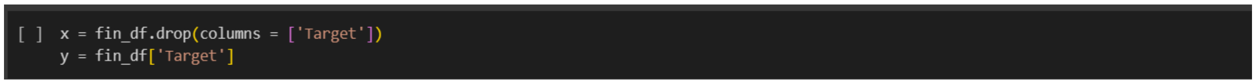
The 'Target' column of the DataFrame 'fin_df' is modified to include numerical values in place of the string labels 'yes' and 'no'. Yes is represented by the value 1, indicating the positive class, and no is represented by the value 0 indicating the negative class.

To make the target variable usable by the logistic regression technique, we execute label encoding on it, which converts the categorical classes to numeric values. Because of this, the model can easily predict the binary outcome by learning the connection between the characteristics (independent variables) and the target variable.

When working with binary classification problems, such as determining whether a consumer would subscribe to a product or not, label encoding is especially helpful. In order to estimate the coefficients and generate predictions based on the estimated probabilities, the logistic regression process requires numerical inputs.

For use with the logistic ML model, the provided code includes a label encoding step that is applied to the target variable and transforms the categorical labels 'yes' and 'no' into numerical representations (1 and 0). By making this adjustment, our ML model can learn from the encoded input and reliably anticipate the binary conclusion.

18) Defining x variables columns and y variable column in order to split the data.



```
[ ]  x = fin_df.drop(columns = ['Target'])
     y = fin_df['Target']
```

In here, The x variables and y variable are being defined in the DataFrame 'fin_df' in preparation for training a logistic ML model. This stage entails decomposing the information into its constituent parts: the independent variables ('x') and the dependent variable or goal ('y').

The target column is dropped in the code, 'x = fin_df.drop(columns=['Target']). replaces the 'Target' column in 'fin_df' with a blank one, producing the new DataFrame 'x'. The 'x' DataFrame represents the features or independent variables contributing to a model's prediction of the 'y' variable. By design, the 'drop' function contains everything from 'fin_df' except the 'Target' column.

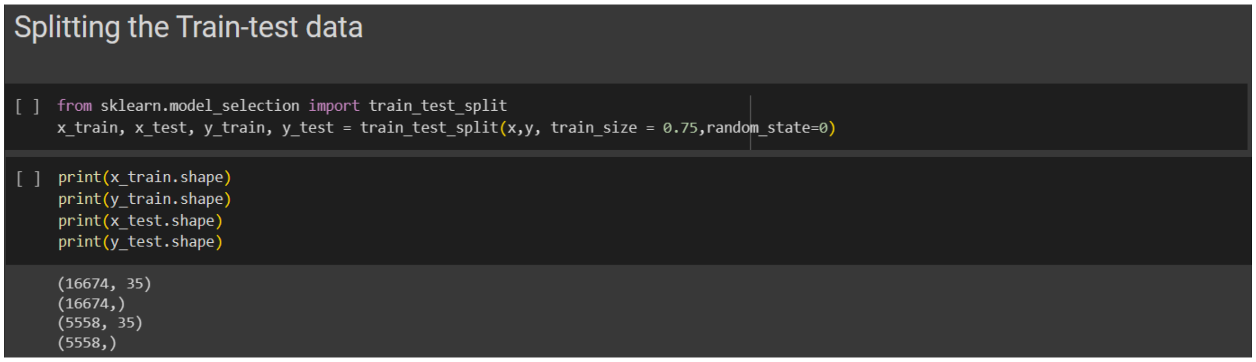
in the code 'y = fin_df['Target']', a new Series named 'y' is generated by reading the 'Target' column out of the original 'fin_df'. The 'y' variable represents the outcome or dependent variable used in the prediction process. The 'Target' values from the 'fin_df' table are included.

The dataset can be divided into a training set and a test set once the 'x' and 'y' variables have been extracted. The 'y' variable is the target against which the model's predictions will be evaluated, while the 'x' variables will be utilized to train the logistic ML model.

This is a necessary action for developing the logistic model and testing its efficacy. The 'y' variables store the right labels or outcomes, while the 'x' variables store the information necessary to generate predictions. We guarantee that the model is trained on the independent variables and learns to accurately predict the target variable by separating the data into 'x' and 'y'.

So, the 'x' and 'y' variables are defined in the code by extracting the independent and dependent variables from the 'fin_df' DataFrame. The data for the logistic ML model can be separated in this way, allowing it to learn from the independent variables and make predictions based on the related dependent variable during training and testing.

19) Splitting the train and the test data for the ML model



Splitting the Train-test data

```
[ ]  from sklearn.model_selection import train_test_split
     x_train, x_test, y_train, y_test = train_test_split(x,y, train_size = 0.75,random_state=0)
```

```
[ ]  print(x_train.shape)
     print(y_train.shape)
     print(x_test.shape)
     print(y_test.shape)
```

```
     (16674, 35)
     (16674,)
     (5558, 35)
     (5558,)
```

The 'train_test_split' method in the'sklearn.model_selection' module is being used to divide the data set into a training set and a testing set. In this section, we test how well the logistic ML model performs and how well it generalizes.

The 'train_test_split' function accepts the variables 'x' and 'y' as input, where 'x' and 'y' are the independent and dependent ones, respectively. It mixes up the information and divides it at random into four groups (named 'x_train,' 'x_test,' 'y_train,' and 'y_test'). Assigning 75% of the data to the training sets and 25% to the testing sets is the result of setting the 'train_size' option to 0.75.

We train the logistic model on one set of data ('x_train' and 'y_train'), and then test it on a second set of data ('x_test' and 'y_test') to see how well it performs on novel information. To ensure that the random shuffling and division of data is reproducible, the 'random_state' parameter has been set to 0.

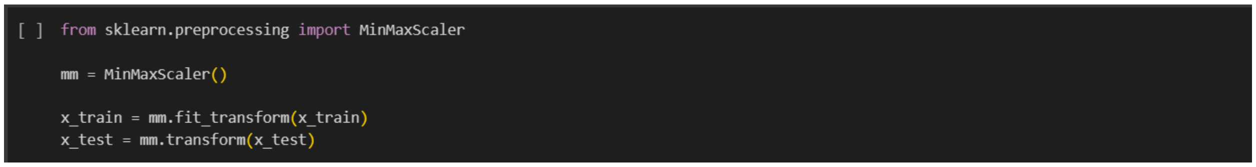
The resulting training and testing set structures are displayed using 'print' statements once the data has been divided. The 'x_train.shape' and 'y_train.shape' files display the rows and columns that make up the 'x_train' and 'y_train' sets, respectively. The dimensions of the 'x_test' and 'y_test' sets are given in the files 'x_test.shape' and 'y_test.shape', respectively.

In order to evaluate the efficacy of the logistic ML model, it is necessary to divide the dataset into training and testing sets. The model is "trained" by feeding it the training set, which contains inputs and expected outputs. This paves the way for the model to discover the hidden structures and associations in the data. The model's ability to generalize and generate predictions on new occurrences is tested on the testing set, which includes data it has never seen before.

We can estimate the model's performance on unseen data and identify problems like overfitting and underfitting by dividing the data into training and testing sets. When the model does well on the training set but poorly on the testing set, it has overfit because it has focused on the specifics of the training set rather than learning how to generalize. Instead, underfitting happens when the model has a low performance on both the training and testing sets, indicating that it has not caught the underlying patterns well enough.

In conclusion, the performance and generalization ability of the logistic ML model can be evaluated by dividing the dataset into training and testing sets with the help of the 'train_test_split' function. This lets us determine if the model is effective and suitable for practical use by training it on a subset of the data and then testing it on unseen examples.

20) Scaling the values

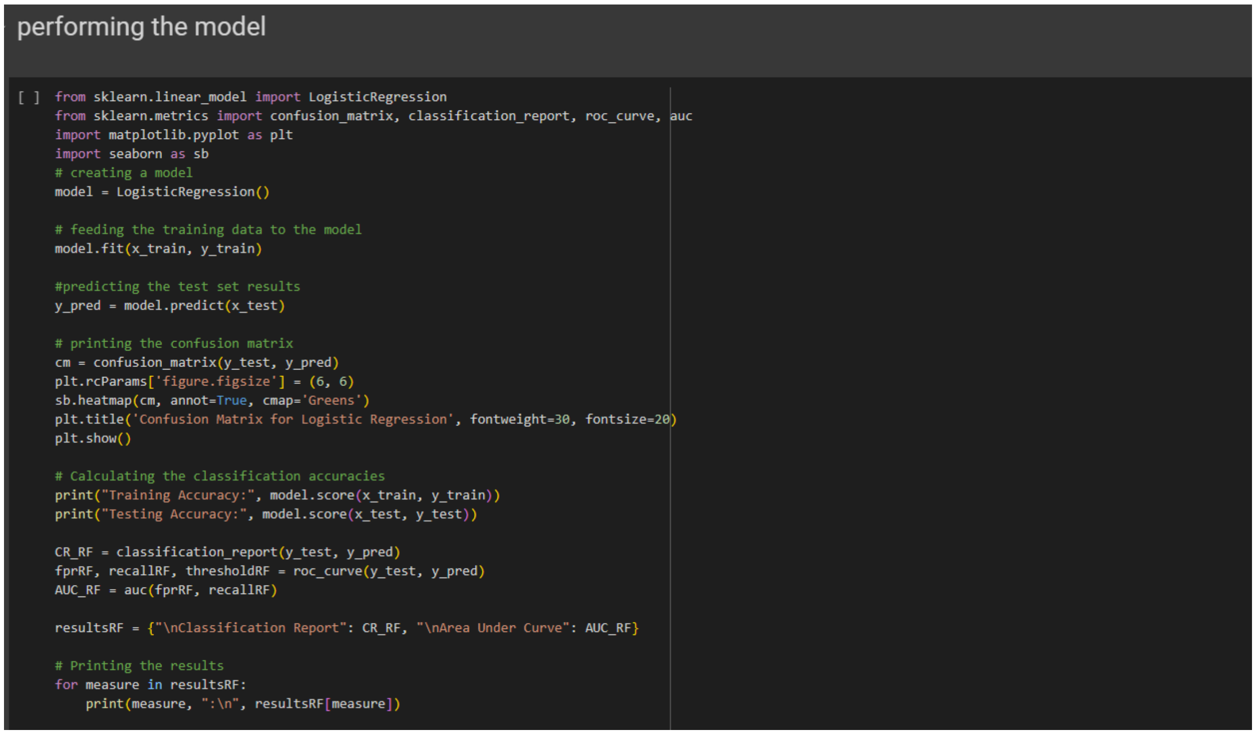

```
[ ]  from sklearn.preprocessing import MinMaxScaler

     mm = MinMaxScaler()

     x_train = mm.fit_transform(x_train)
     x_test = mm.transform(x_test)
```

I have used this MinMaxScaler to normalize the values of the independent variables in the logistic ML model. This is a crucial stage in the process since it ensures that all the variables are comparable and can be correctly read by the logistic model.

Each feature's value is transformed into a range between 0 and 1 by the MinMaxScaler. To do this, it takes the difference between the feature's maximum and minimum values and divides it by the minimum value.

MinMaxScaler is a crucial tool for scaling the independent variables' values in logistic ML models. This eliminates the possibility of errors in the coefficient interpretation caused by variances in the magnitudes of the variables and assures that they are all on the same scale. In addition to improving efficiency and integration, scaling helps the optimization algorithm used in the logistic model.

21) Performing the Logistic regression model


performing the model

```
[ ]  from sklearn.linear_model import LogisticRegression
     from sklearn.metrics import confusion_matrix, classification_report, roc_curve, auc
     import matplotlib.pyplot as plt
     import seaborn as sb
     # creating a model
     model = LogisticRegression()

     # feeding the training data to the model
     model.fit(x_train, y_train)

     #predicting the test set results
     y_pred = model.predict(x_test)

     # printing the confusion matrix
     cm = confusion_matrix(y_test, y_pred)
     plt.rcParams['figure.figsize'] = (6, 6)
     sb.heatmap(cm, annot=True, cmap='Greens')
     plt.title('Confusion Matrix for Logistic Regression', fontweight=30, fontsize=20)
     plt.show()

     # Calculating the classification accuracies
     print("Training Accuracy:", model.score(x_train, y_train))
     print("Testing Accuracy:", model.score(x_test, y_test))

     CR_RF = classification_report(y_test, y_pred)
     fprRF, recallRF, thresholdRF = roc_curve(y_test, y_pred)
     AUC_RF = auc(fprRF, recallRF)

     resultsRF = {"\nClassification Report": CR_RF, "\nArea Under Curve": AUC_RF}

     # Printing the results
     for measure in resultsRF:
         print(measure, ":\n", resultsRF[measure])
```

In here, I have performed a logistic regression model and I have visualized its results using a confusion matrix. Prediction of the dependent variable from the independent variables is being done in the above model. LogisticRegression, a class in the sklearn.linear_model module, is used to build the logistic regression model. When we see the code briefly,

A new instance of the LogisticRegression class is created by calling model = LogisticRegression().

The model is then trained using the training data by using the fit() method and passing in the x_train and y_train independent variables and the y_train target variable, respectively.

Following training, the model is used to make predictions about the test set's dependent variable using the predict() method: y_pred = model.predict(x_test).

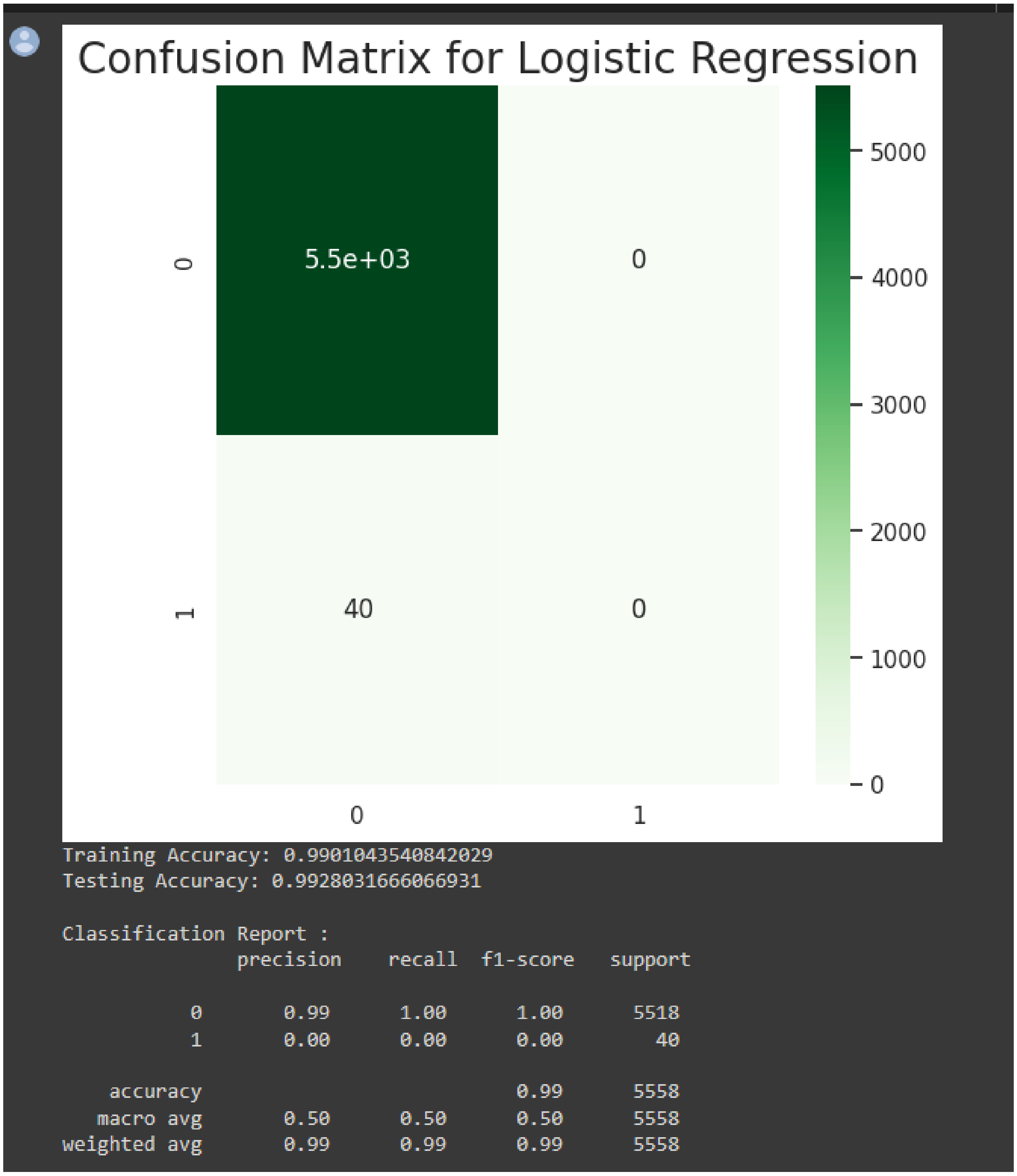
The confusion matrix, a table that summarizes the classification model's performance, is then computed by the code and displayed. The confusion matrix is calculated with the help of the sklearn.metrics module's confusion_matrix() function, and then shown as a heatmap with the help of the seaborn library.

In addition, the score() method is used to compute and display the logistic regression model's accuracy during training and testing. How well a model does on the known training set is measured by the training accuracy, whereas how well it does on the unknown test set is measured by the testing accuracy.

The classification report is then generated, which includes an in-depth analysis of the model's efficiency. The overall accuracy, as well as various macro and weighted averages, are included, in addition to measures like precision, recall, and F1-score for each class (both 0 and 1). In addition, the discriminatory power of the model is evaluated by the Area Under Curve (AUC) of the Receiver Operating Characteristic (ROC) curve.

In short, the program executes logistic regression by training a model, generating predictions, and measuring its efficacy with a number of criteria. The classification report provides in-depth insights into precision, recall, and other metrics for each class, while the confusion matrix aids in evaluating the model's forecast accuracy. The AUC quantifies the model's discriminatory power, while the training and testing accuracies provide insight into its overall performance.

Results of the Model.



```
Training Accuracy: 0.9901043540842029
Testing Accuracy: 0.9928031666066931

Classification Report :
              precision    recall  f1-score   support

           0       0.99      1.00      1.00      5518
           1       0.00      0.00      0.00        40

    accuracy                           0.99      5558
   macro avg       0.50      0.50      0.50      5558
weighted avg       0.99      0.99      0.99      5558
```

When we look at the results of this model, In both training and testing, the model performed exceptionally well, with accuracies of 99.01% and 99.28%, respectively. This means that the model can reliably predict the target variable given the independent variables. However, the classification report reveals that the model has trouble reliably identifying occurrences of the positive class (1). The model was unable to accurately classify any cases into the positive class, as indicated by the precision and recall values of 0. This may be because the dataset is skewed in favour of the negative class (0) rather than the positive class (1). The reported value of 0.5 for the area Under the Curve (AUC) indicates that the model is no more accurate than chance in distinguishing between the positive and negative classes.

These findings underline the significance of conducting additional research and developing the logistic regression model. Using data-balancing methods or modifying the classification threshold are two suggested solutions for dealing with the class imbalance problem. If the model's performance isn't optimal, using different modeling approaches or feature engineering might help. The best strategy for a particular dataset can be gleaned by comparing and contrasting the results of several models. In the end, further research and development work is required to create a more solid and accurate predictive model.

Summary

In conclusion, I have preprocessed the provided bank dataset and i have created a logistic regression ML model in order to predict the subscription of the bank customer for a fixed deposit plan. so, before that i have visualized the variables and i removed all the unwanted column and data points which can affect the mode. Then finally i got the training accuracy level of 99.01% and the testing accuracy level of 99.28%. But the confusion matrix tells that The model was unable to accurately classify any cases into the positive class, as indicated by the precision and recall values of 0. This may be because the dataset is skewed in favor of the negative class (0) rather than the positive class (1). So in order to correct and make this more good, we can perform the data balancing techniques.