Project Description:

In this project, I am developing an automated credit card approval predictor using advanced machine learning techniques. By analyzing factors like income levels, and credit inquiries, we're creating a model to accurately evaluate credit card applications.

Leveraging the Credit Card Approval dataset, My goal is to streamline and enhance the credit application process, offering faster, more consistent, and data-driven decisions.

Through this project, We aim to showcase the potential of AI-driven solutions in revolutionizing critical financial decision-making processes and contributing to the advancement of financial technology.

Why is the proposal important in today's world?

In today's world, financial institutions face the challenge of efficiently assessing credit card applications. Our proposal holds significance as it employs data analysis to predict creditworthiness, enhancing the decision-making process.

As financial transactions continue to digitalize, accurate credit predictions become crucial for risk management and customer satisfaction.

How predicting a good client is worthy for a bank?

Predicting a reliable client is invaluable for banks. It minimizes the risk of default on loans, reduces bad debts, and ensures a healthy loan portfolio.

This predictive ability streamlines the lending process, improves customer experience, and ultimately enhances the bank's financial stability and reputation.

How is it going to impact the banking sector?

Our proposal impacts the banking sector by introducing data-driven precision to credit evaluations. It revolutionizes decision-making, making it faster and more accurate.

By minimizing defaults and bad loans, banks can save resources and focus on strategic growth. Moreover, customers benefit from quicker and fairer loan approvals, fostering trust in the banking system.

If any, what is the gap in the knowledge or how my proposed method can be helpful if required in the future for any bank in India.

There exists a gap in traditional credit assessment methods, which might overlook subtle patterns in vast datasets.

My method bridges this gap by harnessing advanced analytics, uncovering hidden insights, and making more informed credit predictions.

In the future, this approach can serve as a template for other banks in India to adopt data-centric strategies for risk assessment.

Our initial hypotheses are:

Hypothesis 1: Through data analysis, we anticipate discovering significant patterns that correlate with creditworthiness.

Hypothesis 2: Machine learning models, particularly those based on ensemble methods, will outperform individual algorithms in credit prediction.

Hypothesis 3: Key features such as annual income, employment duration, and credit history will emerge as pivotal indicators for credit approval.

As we explore the data and test different models, we will refine these hypotheses and extract actionable insights to develop an efficient credit prediction model tailored for the banking sector. The model's effectiveness will be justified through relevant cost functions and visualized using graphs, showcasing its superiority over other potential models.

Importing Libraries:

We will import all of the primary packages into our python environment.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
import warnings
warnings.filterwarnings('ignore')
```

Loading Data:

We start the project by loading the dataset in our Jupyter notebook. The dataset is loaded into a pandas dataframe named data.

```
data=pd.read_csv('Credit_card[1].csv')
data.head() #looking at the dataset, we print the first five rows
using .head().
```

	Ind_ID	GENDER	Car_Owner	Propert_Owner	CHILDREN	Annual_income	\
0	5008827	M	Y	Y	0	180000.0	
1	5009744	F	Y	N	0	315000.0	
2	5009746	F	Y	N	0	315000.0	
3	5009749	F	Y	N	0	NaN	
4	5009752	F	Y	N	0	315000.0	

```
            Type_Income        EDUCATION Marital_status   
Housing_type  \
0             Pensioner  Higher education        Married  House / 
apartment   
1  Commercial associate  Higher education        Married  House / 
apartment   
2  Commercial associate  Higher education        Married  House / 
apartment   
3  Commercial associate  Higher education        Married  House / 
apartment   
4  Commercial associate  Higher education        Married  House / 
apartment   

   Birthday_count  Employed_days  Mobile_phone  Work_Phone  Phone  
EMAIL_ID  \
0        -18772.0         365243             1           0      0  
0   
1        -13557.0           -586             1           1      1  
0   
2             NaN           -586             1           1      1  
0   
3        -13557.0           -586             1           1      1  
0   
4        -13557.0           -586             1           1      1  
0   

  Type_Occupation  Family_Members  
0             NaN               2  
1             NaN               2  
2             NaN               2  
3             NaN               2  
4             NaN               2  
```

To prove or disprove our hypotheses:

We will employ an exploratory data analysis (EDA) approach. We'll begin by comprehensively examining the dataset's distribution, identifying outliers, and addressing missing values. We'll analyze the relationships between features and target variables, employing visualization techniques such as histograms and correlation matrices.

Feature engineering techniques like scaling, one-hot encoding for categorical variables, and handling missing values will be essential to prepare the data for modeling.

Our data analysis approach is justified as it enables us to uncover hidden patterns, understand the significance of features, and lay the groundwork for the subsequent machine learning phase.

Let's Start :

Knowing Our Data:

To understand our data better, we use pandas features .info(),.describe(),shape.

We are also checking if we have any null data using .isnull().sum()

```
data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1548 entries, 0 to 1547
Data columns (total 18 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Ind_ID           1548 non-null   int64  
 1   GENDER           1541 non-null   object 
 2   Car_Owner        1548 non-null   object 
 3   Propert_Owner    1548 non-null   object 
 4   CHILDREN         1548 non-null   int64  
 5   Annual_income    1525 non-null   float64
 6   Type_Income      1548 non-null   object 
 7   EDUCATION        1548 non-null   object 
 8   Marital_status   1548 non-null   object 
 9   Housing_type     1548 non-null   object 
 10  Birthday_count   1526 non-null   float64
 11  Employed_days    1548 non-null   int64  
 12  Mobile_phone     1548 non-null   int64  
 13  Work_Phone       1548 non-null   int64  
 14  Phone            1548 non-null   int64  
 15  EMAIL_ID         1548 non-null   int64  
 16  Type_Occupation  1060 non-null   object 
 17  Family_Members   1548 non-null   int64  
dtypes: float64(2), int64(8), object(8)
memory usage: 217.8+ KB
```

From the output we get the following information about the data: Data has a total of 1548 entries i.e. approval or rejection data of 1548 credit card applications with a total of 17 columns or input features.

From the output’s Dtype column, we see several features with Dtype as object (string or mixed), which we will have to convert into int64 in later stage for ML Algorithms.

```
data.shape

(1548, 18)

data.describe()
```

```
             Ind_ID     CHILDREN  Annual_income  Birthday_count  \
count  1.548000e+03  1548.000000   1.525000e+03     1526.000000
mean   5.078920e+06     0.412791   1.913993e+05   -16040.342071
std    4.171759e+04     0.776691   1.132530e+05     4229.503202
min    5.008827e+06     0.000000   3.375000e+04   -24946.000000
25%    5.045070e+06     0.000000   1.215000e+05   -19553.000000
50%    5.078842e+06     0.000000   1.665000e+05   -15661.500000
75%    5.115673e+06     1.000000   2.250000e+05   -12417.000000
max    5.150412e+06    14.000000   1.575000e+06    -7705.000000

       Employed_days  Mobile_phone   Work_Phone        Phone
EMAIL_ID  \
count    1548.000000        1548.0  1548.000000  1548.000000
1548.000000
mean    59364.689922           1.0     0.208010     0.309432
0.092377
std    137808.062701           0.0     0.406015     0.462409
0.289651
min    -14887.000000           1.0     0.000000     0.000000
0.000000
25%     -3174.500000           1.0     0.000000     0.000000
0.000000
50%     -1565.000000           1.0     0.000000     0.000000
0.000000
75%      -431.750000           1.0     0.000000     1.000000
0.000000
max    365243.000000           1.0     1.000000     1.000000
1.000000

       Family_Members
count     1548.000000
mean         2.161499
std          0.947772
min          1.000000
25%          2.000000
50%          2.000000
75%          3.000000
max         15.000000
```

```
data.isnull().sum()
```

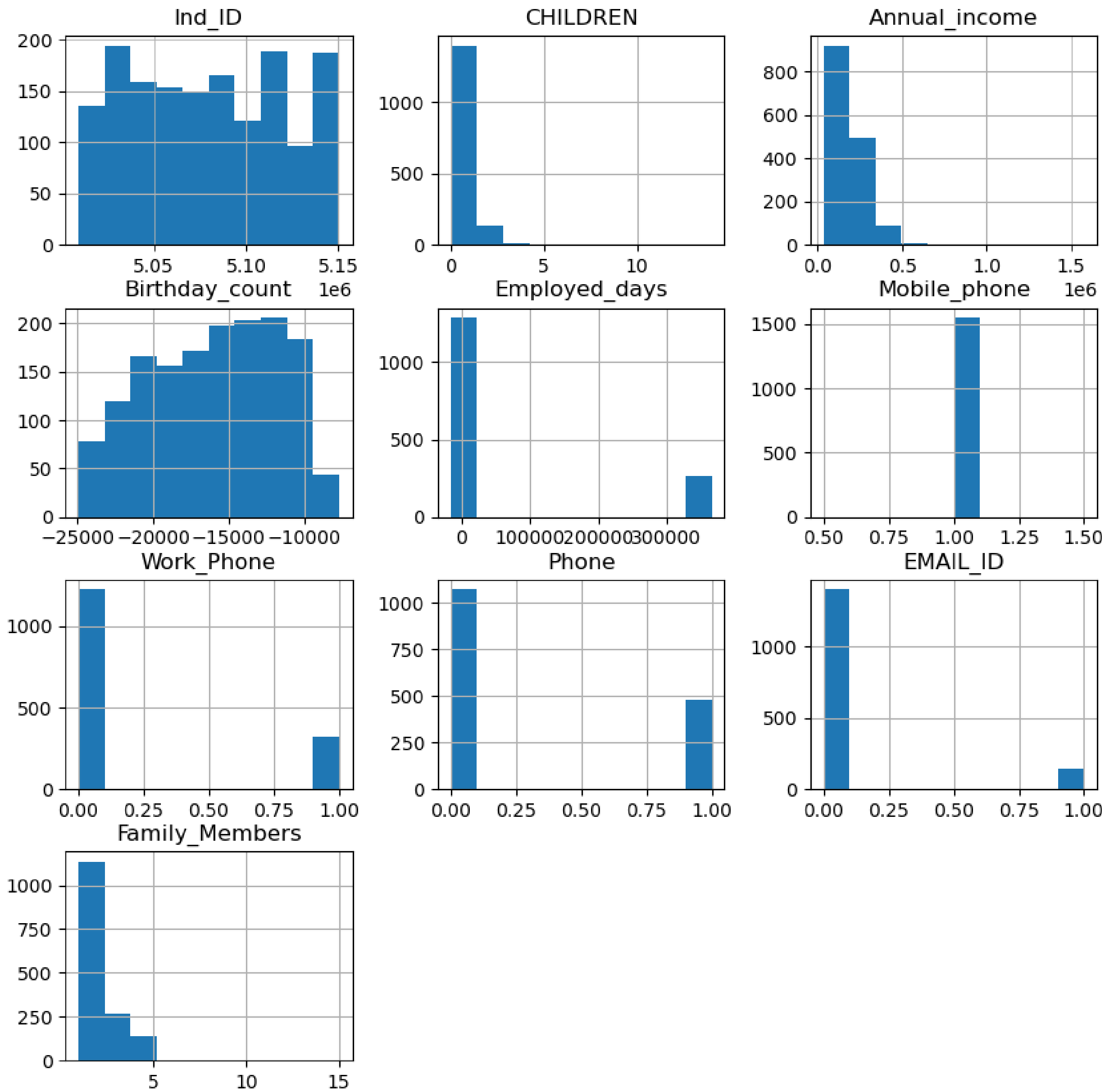
```
Ind_ID              0
GENDER              7
Car_Owner           0
Propert_Owner       0
CHILDREN            0
Annual_income      23
Type_Income         0
EDUCATION           0
Marital_status      0
```

```
Housing_type          0
Birthday_count       22
Employed_days         0
Mobile_phone          0
Work_Phone            0
Phone                 0
EMAIL_ID              0
Type_Occupation     488
Family_Members        0
dtype: int64
```

Visualizing the Data:

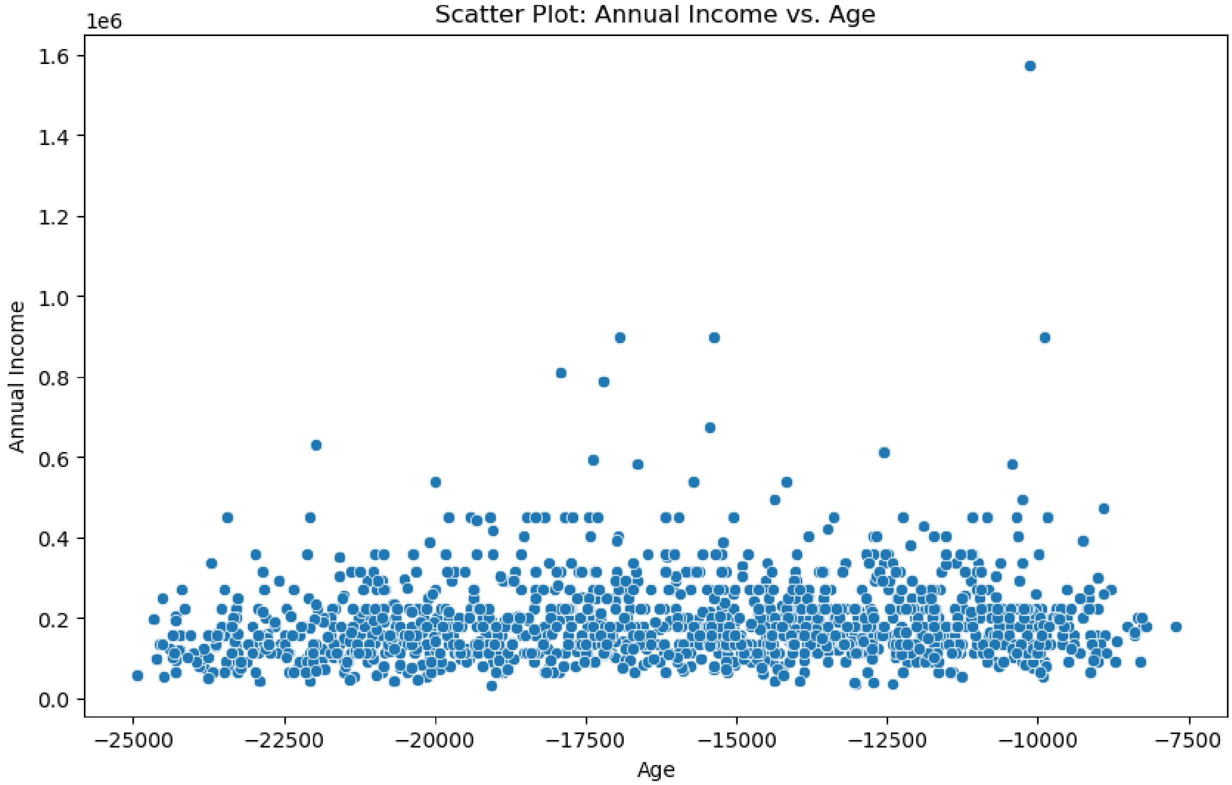
```
data.hist(figsize=(10,10))
```

```
array([[<Axes: title={'center': 'Ind_ID'}>,
        <Axes: title={'center': 'CHILDREN'}>,
        <Axes: title={'center': 'Annual_income'}>],
       [<Axes: title={'center': 'Birthday_count'}>,
        <Axes: title={'center': 'Employed_days'}>,
        <Axes: title={'center': 'Mobile_phone'}>],
       [<Axes: title={'center': 'Work_Phone'}>,
        <Axes: title={'center': 'Phone'}>,
        <Axes: title={'center': 'EMAIL_ID'}>],
       [<Axes: title={'center': 'Family_Members'}>, <Axes: >, <Axes:
>]],
      dtype=object)
```



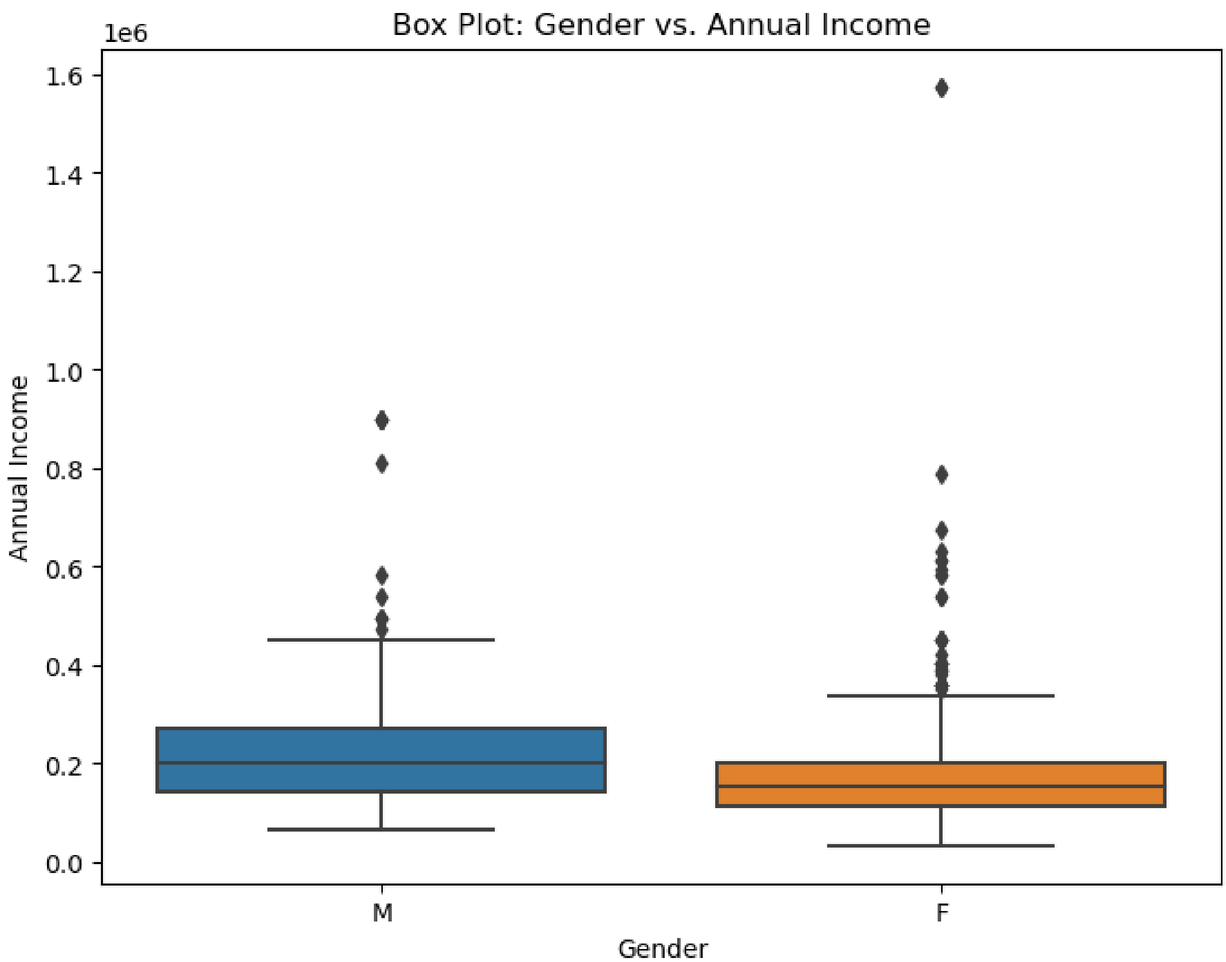

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns



# Drop rows with missing values for the relevant columns
data_cleaned = data.dropna(subset=['Annual_income', 'Birthday_count'])

# Scatter plot: Annual Income vs. Age
plt.figure(figsize=(10, 6))
sns.scatterplot(data=data_cleaned, x='Birthday_count',
y='Annual_income')
```

```
plt.title('Scatter Plot: Annual Income vs. Age')
plt.xlabel('Age')
plt.ylabel('Annual Income')
plt.show()
```




```
data.head()

    Ind_ID GENDER Car_Owner Propert_Owner  CHILDREN  Annual_income  \
0  5008827      M         Y             Y         0       180000.0
1  5009744      F         Y             N         0       315000.0
2  5009746      F         Y             N         0       315000.0
3  5009749      F         Y             N         0            NaN
4  5009752      F         Y             N         0       315000.0

            Type_Income         EDUCATION Marital_status
Housing_type  \
0             Pensioner  Higher education        Married   House /
apartment
1  Commercial associate  Higher education        Married   House /
apartment
2  Commercial associate  Higher education        Married   House /
apartment
3  Commercial associate  Higher education        Married   House /
apartment
4  Commercial associate  Higher education        Married   House /
```

```
apartment

   Birthday_count  Employed_days  Mobile_phone  Work_Phone  Phone  
EMAIL_ID  \
0        -18772.0         365243             1           0      0   
0   
1        -13557.0           -586             1           1      1   
0   
2             NaN           -586             1           1      1   
0   
3        -13557.0           -586             1           1      1   
0   
4        -13557.0           -586             1           1      1   
0   

  Type_Occupation  Family_Members  
0             NaN               2  
1             NaN               2  
2             NaN               2  
3             NaN               2  
4             NaN               2  

# Drop rows with missing values for the relevant columns
data_cleaned_gender = data.dropna(subset=['GENDER', 'Annual_income'])

# Box plot: Gender vs. Annual Income
plt.figure(figsize=(8, 6))
sns.boxplot(data=data_cleaned_gender, x='GENDER', y='Annual_income')
plt.title('Box Plot: Gender vs. Annual Income')
plt.xlabel('Gender')
plt.ylabel('Annual Income')
plt.show()
```




```
import pandas as pd
from scipy.stats import pearsonr



# Drop rows with missing values for the relevant columns
data_cleaned = data.dropna(subset=['Annual_income', 'Birthday_count'])

# Calculate Pearson correlation and p-value
correlation, p_value = pearsonr(data_cleaned['Annual_income'],
data_cleaned['Birthday_count'])

print("Pearson Correlation:", correlation)
print("P-value:", p_value)

Pearson Correlation: 0.11163819215291984
P-value: 1.4377485176990563e-05
```

The analysis revealed a statistically significant positive correlation (correlation coefficient ≈ 0.112) between 'Annual Income' and 'Age.'

The low p-value (≈ 1.44e-05) indicates that as individuals' age increases, their annual income tends to rise. However, the correlation is weak, suggesting that age explains only a small portion of the income variation.

While the results show a meaningful link, it's important to remember that correlation doesn't imply causation, and other unexplored factors might influence this relationship."

```
data['GENDER'].value_counts()
```

```
F    973
M    568
Name: GENDER, dtype: int64
```

```
from scipy.stats import ttest_ind

# Drop rows with missing values for the relevant columns
data_cleaned_gender = data.dropna(subset=['GENDER', 'Annual_income'])

# Separate data by gender
female_income = data_cleaned_gender[data_cleaned_gender['GENDER'] ==
'F']['Annual_income']
male_income = data_cleaned_gender[data_cleaned_gender['GENDER'] ==
'M']['Annual_income']

# Perform independent samples t-test
t_statistic, p_value = ttest_ind(female_income, male_income)

print("T-statistic:", t_statistic)
print("P-value:", p_value)
```

```
T-statistic: -8.580214521268209
P-value: 2.3046545601423273e-17
```

The conducted independent samples t-test revealed a substantial and statistically significant difference in 'Annual Income' based on gender.

The negative t-statistic (≈ -8.58) and very low p-value (≈ 2.30e-17) indicate that, on average, one gender's annual income is significantly lower than the other.

This underscores the role of gender in influencing income disparities within the dataset. It's important to consider broader factors that might contribute to these observed differences.

```
data.duplicated()
```

```
0       False
1       False
2       False
3       False
4       False
        ...
1543    False
```

```
1544    False
1545    False
1546    False
1547    False
Length: 1548, dtype: bool
```

```
data.corr()
```

```
                  Ind_ID  CHILDREN  Annual_income  Birthday_count  \
Ind_ID          1.000000  0.032535       0.030147        0.022909
CHILDREN        0.032535  1.000000       0.078497        0.279716
Annual_income   0.030147  0.078497       1.000000        0.111638
Birthday_count  0.022909  0.279716       0.111638        1.000000
Employed_days  -0.055396 -0.219095      -0.160175       -0.619039
Mobile_phone         NaN       NaN            NaN             NaN
Work_Phone      0.085794  0.035014      -0.071171        0.174687
Phone           0.008403 -0.004908      -0.006439       -0.029215
EMAIL_ID       -0.037923  0.025776       0.122320        0.166749
Family_Members  0.016950  0.890248       0.050957        0.266527

                Employed_days  Mobile_phone  Work_Phone     Phone
EMAIL_ID  \
Ind_ID              -0.055396           NaN    0.085794  0.008403 -
0.037923
CHILDREN            -0.219095           NaN    0.035014 -0.004908
0.025776
Annual_income       -0.160175           NaN   -0.071171 -0.006439
0.122320
Birthday_count      -0.619039           NaN    0.174687 -0.029215
0.166749
Employed_days        1.000000           NaN   -0.231184 -0.003403 -
0.118268
Mobile_phone              NaN           NaN         NaN       NaN
NaN
Work_Phone          -0.231184           NaN    1.000000  0.352439 -
0.009594
Phone               -0.003403           NaN    0.352439  1.000000
0.018105
EMAIL_ID            -0.118268           NaN   -0.009594  0.018105
1.000000
Family_Members      -0.238705           NaN    0.072228  0.005372
0.035098

                Family_Members
Ind_ID                0.016950
CHILDREN              0.890248
Annual_income         0.050957
Birthday_count        0.266527
Employed_days        -0.238705
Mobile_phone               NaN
```

```
Work_Phone           0.072228
Phone                0.005372
EMAIL_ID             0.035098
Family_Members       1.000000
```

Data.corr() is used to find the pairwise correlation of all columns in the Pandas Dataframe in Python.

Any NaN values are automatically excluded. Any non-numeric data type or columns in the Dataframe, it is ignored.

```
data.isnull().sum()
```

```
Ind_ID              0
GENDER              7
Car_Owner           0
Propert_Owner       0
CHILDREN            0
Annual_income      23
Type_Income         0
EDUCATION           0
Marital_status      0
Housing_type        0
Birthday_count     22
Employed_days       0
Mobile_phone        0
Work_Phone          0
Phone               0
EMAIL_ID            0
Type_Occupation   488
Family_Members      0
dtype: int64
```

Data.isnull().sum() provides us with missing values in the dataset , with above data we can see that there are missing data in 'Gender','Annual_income','Birthday_count','Type_Occupation'.

We can either remove or replace the missing values with any imputation method in later stage.

Data Cleaning:

In the world of data analysis, dealing with messy data is a reality we can't escape. Every dataset, without exception, may contain missing values across various columns, each corresponding to a data entry. Before diving into data analysis and drawing conclusions.

It's crucial to recognize the existence of these missing values in our dataset. The way missing values are represented can vary, such as using symbols like ?, NaN. When a column's data type is numeric (int or float), missing values are often indicated using NaN. On the other hand, for columns with categorical data types, we display the distinct values present. As we inspect the dataset, we notice the presence of missing values, marked with the label '?'.

Also there are several columns in the raw data which does not add any value to the data, we are dropping all such columns which has very less or no impact on our output.

```
data.drop('Mobile_phone',inplace=True,axis=1)
data.drop('Work_Phone',inplace=True,axis=1)
data.drop('Phone',inplace=True,axis=1)
data.drop('EMAIL_ID',inplace=True,axis=1)
```

Here, we dropped columns like 'Mobile_phone','Work_Phone','Phone','EMAIL_ID' as they have no impact on the Output.

```
data['Age'] = data['Birthday_count']/365*(-1)  #Changing the
Birthdaycount column into Age
```

Here, We change the 'Birthday_count' column into 'Age' so that our data works and looks better.

```
data['Experience'] = data['Employed_days']/365*(-1) #Changing the
Employed days column into Experience

data['Experience'] = data['Experience'].apply(lambda x: round(x, 2))
```

As we changed for 'Age' column likewise We are converting the 'Employed_days' column into 'Experience' so that our data works and looks better.

```
data.drop('Birthday_count',inplace=True,axis=1)
data.drop('Employed_days',inplace=True,axis=1)
```

We dropped both the columns 'Birthday_count' and 'Employed_days' from the original data as we have changed the data into 'Age' and 'Experience'.

Finding and Handling Missing Number:

Missing data is probably one of the most common issues when working with real datasets. Data can be missing for a multitude of reasons, including sensor failure, data vintage, improper data management, and even human error. Missing data can occur as single values, multiple values within one feature, or entire features may be missing.

It is important that missing data is identified and handled appropriately prior to further data analysis or machine learning. Many machine learning algorithms can’t handle missing data and require entire rows, where a single missing value is present, to be deleted or replaced (imputed) with a new value.

Median Imputation on 'Annual_income' column:

Median imputation is a technique used to replace missing values in a dataset with the median value of the available data. In the context of annual income, median imputation involves replacing missing income values with the median income of the individuals or cases for which income data is available.

This method is often used to handle missing data in a way that avoids extreme outliers and maintains the overall distribution of income.

```
data['Annual_income'].fillna(data['Annual_income'].median())
```

```
0       180000.0
1       315000.0
2       315000.0
3       166500.0
4       315000.0
          ...
1543    166500.0
1544    225000.0
1545    180000.0
1546    270000.0
1547    225000.0
Name: Annual_income, Length: 1548, dtype: float64
```

Imputed the 23 missing annual income values with the median income to mitigate the influence of outliers

```
data.dropna(inplace=True)
```

'Data.dropna(inplace=True)' will remove all rows from the data DataFrame that contain at least one missing value and update the DataFrame itself.

This can be a useful step when you want to clean your data by getting rid of rows that have incomplete information.

Just make sure to use this method with caution, as removing rows with missing data might lead to loss of valuable information, and it's important to consider the impact on your analysis or model.

```
data = data.astype({'Age':'int'})
```

We are changing the datatype into 'int' for Age column from 'float'.

```
!pip install missingno
```

```
Requirement already satisfied: missingno in c:\users\nishita singh\
anaconda3\lib\site-packages (0.5.2)
Requirement already satisfied: numpy in c:\users\nishita singh\
anaconda3\lib\site-packages (from missingno) (1.24.3)
Requirement already satisfied: matplotlib in c:\users\nishita singh\
anaconda3\lib\site-packages (from missingno) (3.7.1)
Requirement already satisfied: scipy in c:\users\nishita singh\
anaconda3\lib\site-packages (from missingno) (1.10.1)
```

```
Requirement already satisfied: seaborn in c:\users\nishita singh\
anaconda3\lib\site-packages (from missingno) (0.12.2)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\nishita
singh\anaconda3\lib\site-packages (from matplotlib->missingno) (1.0.5)
Requirement already satisfied: cycler>=0.10 in c:\users\nishita singh\
anaconda3\lib\site-packages (from matplotlib->missingno) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\nishita
singh\anaconda3\lib\site-packages (from matplotlib->missingno)
(4.25.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\nishita
singh\anaconda3\lib\site-packages (from matplotlib->missingno) (1.4.4)
Requirement already satisfied: packaging>=20.0 in c:\users\nishita
singh\anaconda3\lib\site-packages (from matplotlib->missingno) (23.0)
Requirement already satisfied: pillow>=6.2.0 in c:\users\nishita
singh\anaconda3\lib\site-packages (from matplotlib->missingno) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\nishita
singh\anaconda3\lib\site-packages (from matplotlib->missingno) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\
nishita singh\anaconda3\lib\site-packages (from matplotlib->missingno)
(2.8.2)
Requirement already satisfied: pandas>=0.25 in c:\users\nishita singh\
anaconda3\lib\site-packages (from seaborn->missingno) (1.5.3)
Requirement already satisfied: pytz>=2020.1 in c:\users\nishita singh\
anaconda3\lib\site-packages (from pandas>=0.25->seaborn->missingno)
(2022.7)
Requirement already satisfied: six>=1.5 in c:\users\nishita singh\
anaconda3\lib\site-packages (from python-dateutil>=2.7->matplotlib-
>missingno) (1.16.0)
```

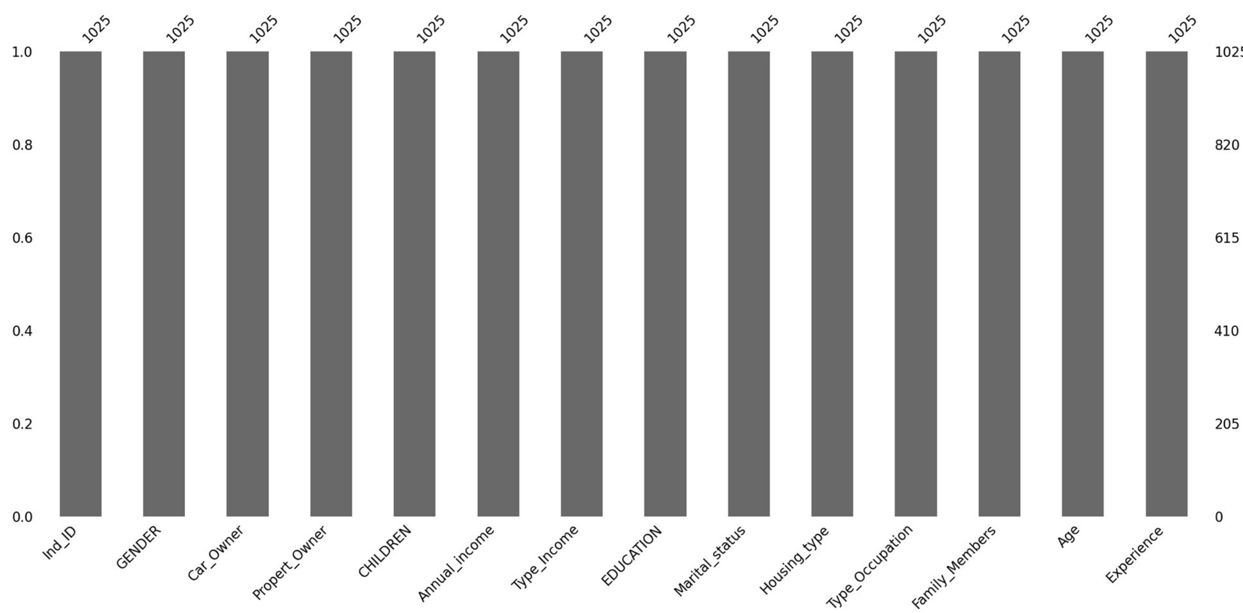
```
data.head()
```

```
     Ind_ID GENDER Car_Owner Propert_Owner  CHILDREN  Annual_income  \
8   5010864      M         Y             Y         1       450000.0
9   5010868      M         Y             Y         1       450000.0
10  5010869      M         Y             Y         1       450000.0
11  5018498      F         Y             Y         0        90000.0
13  5018503      F         Y             Y         0        90000.0

             Type_Income                      EDUCATION
Marital_status  \
8   Commercial associate  Secondary / secondary special
Married
9              Pensioner  Secondary / secondary special
Married
10  Commercial associate  Secondary / secondary special  Single / not
married
11               Working  Secondary / secondary special
Married
13               Working  Secondary / secondary special
Married
```

```
         Housing_type Type_Occupation  Family_Members  Age  Experience
8   House / apartment      Core staff               3   49        1.86
9   House / apartment      Core staff               3   49        1.86
10  House / apartment      Core staff               1   49        1.86
11  House / apartment   Cooking staff               2   51        2.75
13  House / apartment   Cooking staff               2   51        2.75
```

```python
import missingno as msno
msno.bar(data) # shows how much data is missing
#The amount of empty spaces shows missing data
```

```
<Axes: >
```



We can see that there is no missing values as we have already dealt with the missing values earlier and dropped all the na value in the dataset i.e., the missing values were removed from the original dataset.

Though we should always be cautious before removing any missing values , either we can handle the missing values by imputation as we did earlier for 'Annual_ income column' and for others we removed the missing data by 'data.dropna'.

```python
data.shape
```

```
(1025, 14)
```

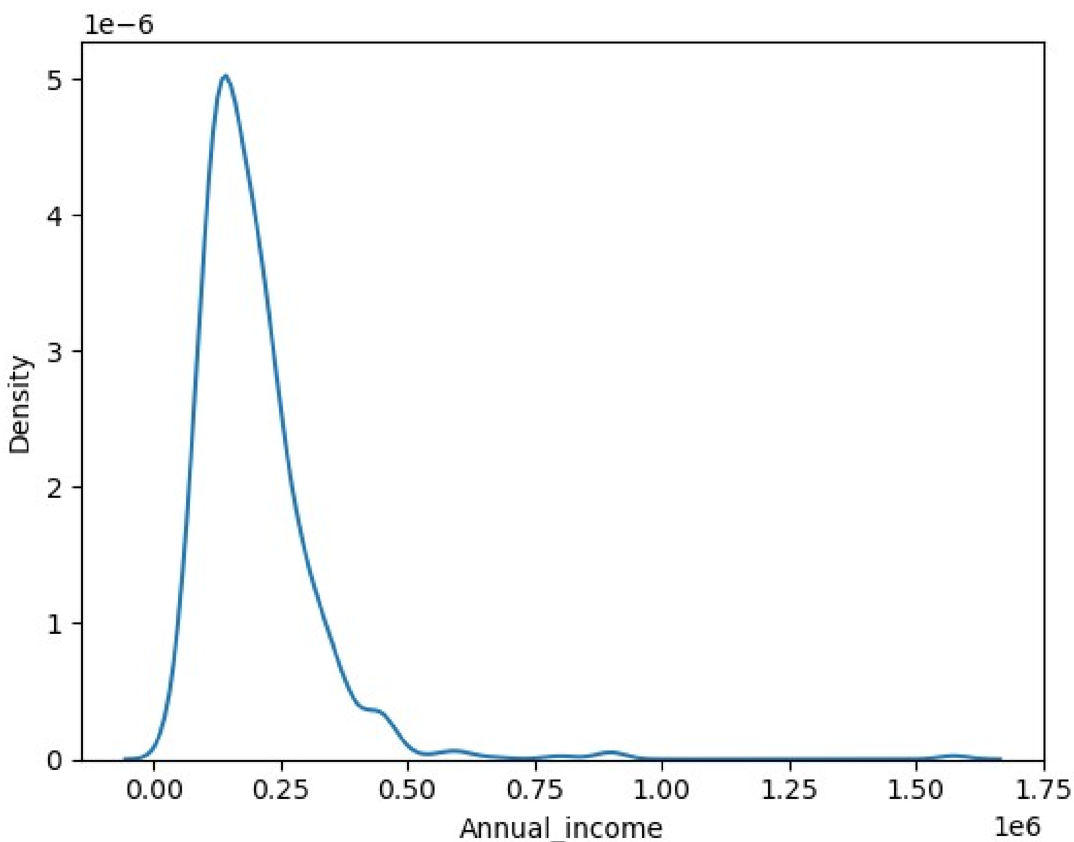
Outlier Treatment:

Outlier treatment involves identifying and addressing data points in a dataset that deviate significantly from the overall pattern or distribution of the data.

Outliers can be caused by measurement errors, data entry mistakes, or genuine anomalies in the data. Managing outliers is important because they can skew statistical analysis, model performance, and the overall understanding of the data.

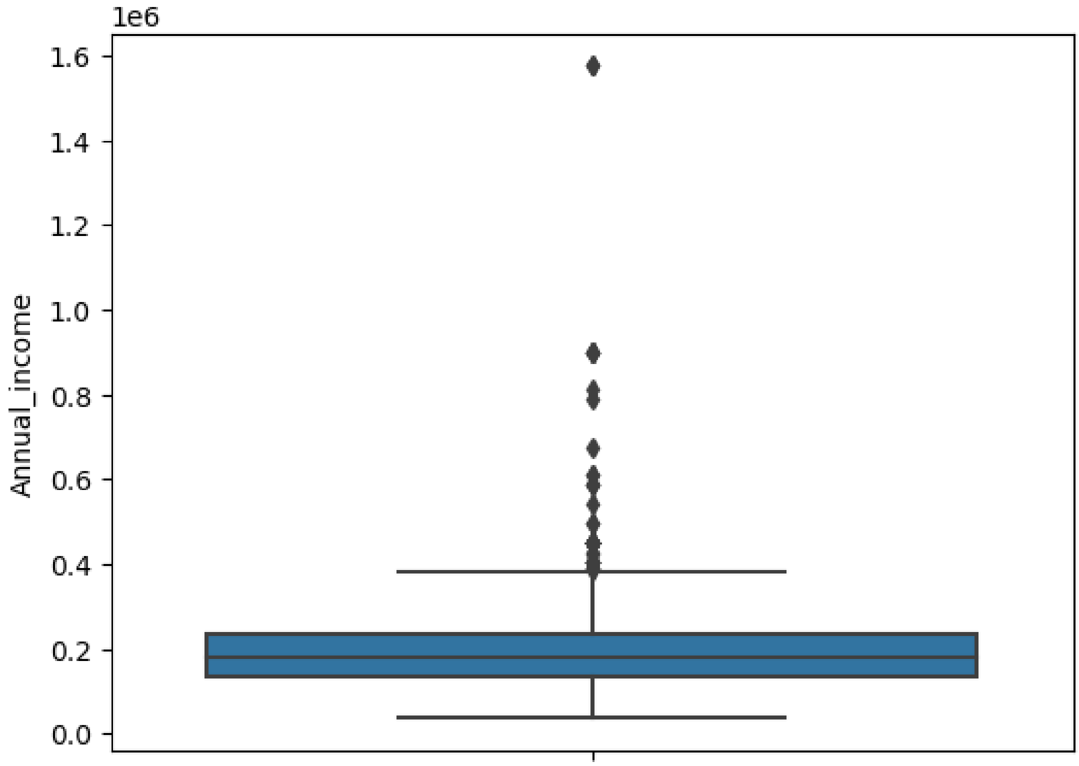
```
sns.kdeplot(data['Annual_income'])
```

```
<Axes: xlabel='Annual_income', ylabel='Density'>
```




```
sns.boxplot(y=data['Annual_income'])
```

```
<Axes: ylabel='Annual_income'>
```



We can easily see the presence of outliers in the given data.

Data points that fall significantly above or below certain thresholds based on these methods are often flagged as outliers. we are going to use IQR outlier treatment to deal with the outliers in our data.

```
data.describe()

             Ind_ID     CHILDREN  Annual_income  Family_Members
Age  \
count  1.025000e+03  1025.000000   1.025000e+03     1025.000000
1025.000000
mean   5.081019e+06     0.494634   2.001105e+05        2.272195
40.202927
std    4.193412e+04     0.844456   1.215105e+05        0.996604
9.515088
min    5.008865e+06     0.000000   3.600000e+04        1.000000
21.000000
25%    5.045380e+06     0.000000   1.350000e+05        2.000000
32.000000
50%    5.088503e+06     0.000000   1.800000e+05        2.000000
40.000000
75%    5.116478e+06     1.000000   2.340000e+05        3.000000
48.000000
max    5.150221e+06    14.000000   1.575000e+06       15.000000
65.000000
```

```
       Experience
count  1025.000000
mean      7.387688
std       6.575290
min       0.200000
25%       2.680000
50%       5.390000
75%       9.980000
max      40.790000
```

Outlier detection and Treatment using IQR:

-Outlier detection using the Interquartile Range (IQR) is a practical method for spotting potential outliers in a dataset.

-Compute the IQR by finding the range between the third quartile (Q3) and the first quartile (Q1).

-Multiply the IQR by a chosen factor (commonly 1.5 or 3) to determine lower and upper boundaries.

-Data points falling below the lower bound or above the upper bound are considered possible outliers.

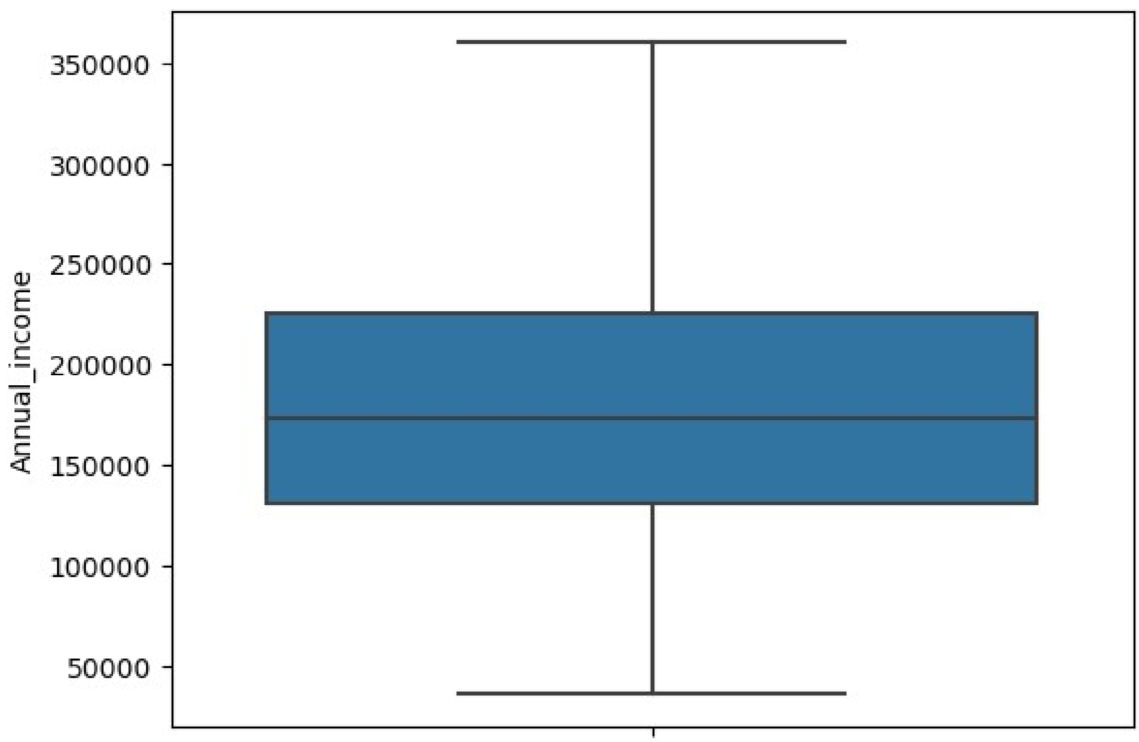
-Visualize these outliers through a box plot, where points outside the whiskers indicate potential outliers.

-Examine flagged data points, considering their context and whether further investigation or treatment is necessary.

-Adjust the multiplier factor based on the data's characteristics and sensitivity requirements.

```
Q1= data['Annual_income'].quantile(0.25)
Q3= data['Annual_income'].quantile(0.75)

Q1

135000.0

Q3

234000.0

IQR= Q3 - Q1
IQR  # quantative value which shows variation b/w values.
#The smaller the iqr the better data

99000.0

low_lim =Q1-1.5 * IQR
high_lim= Q3 + 1.5 * IQR
```

```
low_lim
high_lim

382500.0

data = data[(data['Annual_income']> low_lim) &
(data['Annual_income']<high_lim)]

outlier =[]
def detect_outlier(column):
    for x in data[column]:
        if (x > high_lim) or (x<low_lim):
            outlier.append(x)

detect_outlier('Annual_income')

outlier

[]

detect_outlier('Age')
outlier

[]

sns.boxplot(y=data['Annual_income'])

<Axes: ylabel='Annual_income'>
```



We observe that the outliers have been removed by the IQR treatment.

```
data.head()

     Ind_ID GENDER Car_Owner Propert_Owner  CHILDREN  Annual_income  \
11  5018498      F         Y             Y         0        90000.0
13  5018503      F         Y             Y         0        90000.0
15  5021310      M         N             Y         0       270000.0
16  5021314      M         N             Y         0       270000.0
17  5021430      F         N             Y         0       126000.0

             Type_Income                      EDUCATION
Marital_status  \
11               Working  Secondary / secondary special
Married
13               Working  Secondary / secondary special
Married
15               Working  Secondary / secondary special
Married
16               Working  Secondary / secondary special  Single / not
married
17  Commercial associate               Higher education  Single / not
married

         Housing_type Type_Occupation  Family_Members  Age  Experience

11  House / apartment   Cooking staff               2   51        2.75

13  House / apartment   Cooking staff               2   51        2.75

15  House / apartment        Laborers               2   46        0.68

16  House / apartment        Laborers               2   46        0.68

17  House / apartment     Sales staff               1   51        6.77
```

Encoding:

Encoding is a process used in data preprocessing and machine learning to convert categorical data (non-numeric values) into a numerical format that can be understood by algorithms.

Categorical data includes variables like colors, types of products, or geographic regions.

```
!pip install category_encoders

Requirement already satisfied: category_encoders in c:\users\nishita
singh\anaconda3\lib\site-packages (2.6.2)
Requirement already satisfied: numpy>=1.14.0 in c:\users\nishita
singh\anaconda3\lib\site-packages (from category_encoders) (1.24.3)
```

```
Requirement already satisfied: scikit-learn>=0.20.0 in c:\users\
nishita singh\anaconda3\lib\site-packages (from category_encoders)
(1.3.0)
Requirement already satisfied: scipy>=1.0.0 in c:\users\nishita singh\
anaconda3\lib\site-packages (from category_encoders) (1.10.1)
Requirement already satisfied: statsmodels>=0.9.0 in c:\users\nishita
singh\anaconda3\lib\site-packages (from category_encoders) (0.14.0)
Requirement already satisfied: pandas>=1.0.5 in c:\users\nishita
singh\anaconda3\lib\site-packages (from category_encoders) (1.5.3)
Requirement already satisfied: patsy>=0.5.1 in c:\users\nishita singh\
anaconda3\lib\site-packages (from category_encoders) (0.5.3)
Requirement already satisfied: python-dateutil>=2.8.1 in c:\users\
nishita singh\anaconda3\lib\site-packages (from pandas>=1.0.5-
>category_encoders) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\nishita singh\
anaconda3\lib\site-packages (from pandas>=1.0.5->category_encoders)
(2022.7)
Requirement already satisfied: six in c:\users\nishita singh\
anaconda3\lib\site-packages (from patsy>=0.5.1->category_encoders)
(1.16.0)
Requirement already satisfied: joblib>=1.1.1 in c:\users\nishita
singh\anaconda3\lib\site-packages (from scikit-learn>=0.20.0-
>category_encoders) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\
nishita singh\anaconda3\lib\site-packages (from scikit-learn>=0.20.0-
>category_encoders) (2.2.0)
Requirement already satisfied: packaging>=21.3 in c:\users\nishita
singh\anaconda3\lib\site-packages (from statsmodels>=0.9.0-
>category_encoders) (23.0)
```

```python
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import OrdinalEncoder
import category_encoders as ce
```

Encoded Education column using Ordinal Endcoding:

Ordinal encoding is a method used to convert categorical variables with an inherent order or ranking into numerical values. This technique is particularly useful when dealing with data where the categories have a meaningful sequence, but the actual numeric values don't hold significant meaning.

```python
data['EDUCATION'].unique()
```

```
array(['Secondary / secondary special', 'Higher education',
       'Lower secondary', 'Incomplete higher'], dtype=object)
```

```
data['EDUCATION']=data['EDUCATION'].map({'Higher
education':4,'Secondary / secondary special':3,'Lower
secondary':2,'Incomplete higher':1})

data.head()

    Ind_ID GENDER Car_Owner Propert_Owner  CHILDREN  Annual_income  \
11  5018498      F         Y             Y         0        90000.0
13  5018503      F         Y             Y         0        90000.0
15  5021310      M         N             Y         0       270000.0
16  5021314      M         N             Y         0       270000.0
17  5021430      F         N             Y         0       126000.0

             Type_Income  EDUCATION       Marital_status
Housing_type  \
11               Working          3              Married  House /
apartment
13               Working          3              Married  House /
apartment
15               Working          3              Married  House /
apartment
16               Working          3  Single / not married  House /
apartment
17  Commercial associate          4  Single / not married  House /
apartment

   Type_Occupation  Family_Members  Age  Experience
11   Cooking staff               2   51        2.75
13   Cooking staff               2   51        2.75
15        Laborers               2   46        0.68
16        Laborers               2   46        0.68
17     Sales staff               1   51        6.77
```

Encoding with Get_dummies:

```
df_dummy=pd.get_dummies(data[['GENDER','Type_Income','Car_Owner','Prop
ert_Owner']],drop_first=True)

data=pd.concat([data,df_dummy],axis=1)

data.shape

(973, 20)

data.isnull().sum()

Ind_ID                        0
GENDER                        0
Car_Owner                     0
Propert_Owner                 0
CHILDREN                      0
```

```
Annual_income                     0
Type_Income                       0
EDUCATION                         0
Marital_status                    0
Housing_type                      0
Type_Occupation                   0
Family_Members                    0
Age                               0
Experience                        0
GENDER_M                          0
Type_Income_Pensioner             0
Type_Income_State servant         0
Type_Income_Working               0
Car_Owner_Y                       0
Propert_Owner_Y                   0
dtype: int64
```

```
data.drop(columns
=['GENDER','Type_Income','Car_Owner','Propert_Owner'],inplace=True)
```

Encoding with OneHotEncoder:

```
data['Marital_status'].value_counts()
```

```
Married                   680
Single / not married      142
Civil marriage             73
Separated                  54
Widow                      24
Name: Marital_status, dtype: int64
```

```
OHE=OneHotEncoder(handle_unknown='ignore',sparse=False)
```

```
OHE.fit(data[['Marital_status']])
```

```
OneHotEncoder(handle_unknown='ignore', sparse=False,
sparse_output=False)
```

```
encoded=OHE.transform(data[['Marital_status']])
```

```
pd.DataFrame(encoded)
```

```
       0    1    2    3    4
0    0.0  1.0  0.0  0.0  0.0
1    0.0  1.0  0.0  0.0  0.0
2    0.0  1.0  0.0  0.0  0.0
3    0.0  0.0  0.0  1.0  0.0
4    0.0  0.0  0.0  1.0  0.0
..   ...  ...  ...  ...  ...
968  0.0  1.0  0.0  0.0  0.0
969  0.0  1.0  0.0  0.0  0.0
```

```
970  0.0  0.0  0.0  1.0  0.0
971  0.0  1.0  0.0  0.0  0.0
972  1.0  0.0  0.0  0.0  0.0

[973 rows x 5 columns]
```

```
encoded
```

```
array([[0., 1., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       ...,
       [0., 0., 0., 1., 0.],
       [0., 1., 0., 0., 0.],
       [1., 0., 0., 0., 0.]])
```

```
labels_marital=pd.DataFrame()
labels_marital['Marital_status_married']=encoded[:,0]
labels_marital['Marital_status_Single']=encoded[:,1]
labels_marital['Marital_status_Civil_marriage']=encoded[:,2]
labels_marital['Marital_status_Separated']=encoded[:,3]
labels_marital['Marital_status_Widow']=encoded[:,4]
```

```
labels_marital.isnull().sum()
```

```
Marital_status_married           0
Marital_status_Single            0
Marital_status_Civil_marriage    0
Marital_status_Separated         0
Marital_status_Widow             0
dtype: int64
```

```
data=pd.concat([data.reset_index(drop=True),labels_marital.reset_index
(drop=True)], axis=1)
```

```
data.drop('Marital_status',inplace=True,axis=1)
```

```
data.isnull().sum()
```

```
Ind_ID                           0
CHILDREN                         0
Annual_income                    0
EDUCATION                        0
Housing_type                     0
Type_Occupation                  0
Family_Members                   0
Age                              0
Experience                       0
GENDER_M                         0
Type_Income_Pensioner            0
Type_Income_State servant        0
```

```
Type_Income_Working                 0
Car_Owner_Y                         0
Propert_Owner_Y                     0
Marital_status_married              0
Marital_status_Single               0
Marital_status_Civil_marriage       0
Marital_status_Separated            0
Marital_status_Widow                0
dtype: int64
```

```
data['Housing_type'].value_counts()
```

```
House / apartment      859
With parents            63
Municipal apartment     28
Rented apartment        11
Office apartment         7
Co-op apartment          5
Name: Housing_type, dtype: int64
```

```
OHE_house=OneHotEncoder(handle_unknown='ignore',sparse=False)

OHE_house.fit(data[['Housing_type']])
```

```
OneHotEncoder(handle_unknown='ignore', sparse=False,
sparse_output=False)
```

```
encoded_house=OHE_house.transform(data[['Housing_type']])

labels_house=pd.DataFrame()
labels_house['housing_type_house']=encoded_house[:,0]
labels_house['housing_type_with_parents']=encoded_house[:,1]
labels_house['housing_type_Municipal_apartment']=encoded_house[:,2]
labels_house['housing_type_Rented_apartment']=encoded_house[:,3]
labels_house['housing_type_Office_apartment']=encoded_house[:,4]
labels_house['housing_type_Office_Co_op_apartment']=encoded_house[:,5]

data=
pd.concat([data.reset_index(drop=True),labels_house.reset_index(drop=T
rue)], axis=1)

data.drop('Housing_type',inplace=True,axis=1)

data['Type_Occupation'].value_counts()
```

```
Laborers                   256
Core staff                 161
Managers                   111
Sales staff                111
Drivers                     80
High skill tech staff       60
Medicine staff              49
```

```
Accountants                 40
Security staff              22
Cleaning staff              20
Cooking staff               18
Private service staff       15
Low-skill Laborers           9
Secretaries                  9
Waiters/barmen staff         5
HR staff                     3
IT staff                     2
Realty agents                2
Name: Type_Occupation, dtype: int64
```

```
df_dummy_occ=pd.get_dummies(data[['Type_Occupation']],drop_first=True)

data=pd.concat([data,df_dummy_occ],axis=1)

data.head()
```

```
    Ind_ID  CHILDREN  Annual_income  EDUCATION Type_Occupation  \
0  5018498         0        90000.0          3   Cooking staff
1  5018503         0        90000.0          3   Cooking staff
2  5021310         0       270000.0          3        Laborers
3  5021314         0       270000.0          3        Laborers
4  5021430         0       126000.0          4     Sales staff

   Family_Members  Age  Experience  GENDER_M
Type_Income_Pensioner  ...  \
0               2   51        2.75         0
0  ...
1               2   51        2.75         0
0  ...
2               2   46        0.68         1
0  ...
3               2   46        0.68         1
0  ...
4               1   51        6.77         0
0  ...

   Type_Occupation_Laborers  Type_Occupation_Low-skill Laborers  \
0                         0                                   0
1                         0                                   0
2                         1                                   0
3                         1                                   0
4                         0                                   0

   Type_Occupation_Managers  Type_Occupation_Medicine staff  \
0                         0                               0
1                         0                               0
2                         0                               0
```

```
3                              0                                     0
4                              0                                     0

   Type_Occupation_Private service staff  Type_Occupation_Realty
agents  \
0                                      0
0
1                                      0
0
2                                      0
0
3                                      0
0
4                                      0
0

   Type_Occupation_Sales staff  Type_Occupation_Secretaries  \
0                            0                            0
1                            0                            0
2                            0                            0
3                            0                            0
4                            1                            0

   Type_Occupation_Security staff  Type_Occupation_Waiters/barmen
staff
0                               0
0
1                               0
0
2                               0
0
3                               0
0
4                               0
0

[5 rows x 42 columns]
```

```
data.drop('Type_Occupation',inplace=True,axis=1)

data.head()
```

```
    Ind_ID  CHILDREN  Annual_income  EDUCATION  Family_Members  Age  \
0  5018498         0        90000.0          3               2   51
1  5018503         0        90000.0          3               2   51
2  5021310         0       270000.0          3               2   46
3  5021314         0       270000.0          3               2   46
4  5021430         0       126000.0          4               1   51

   Experience  GENDER_M  Type_Income_Pensioner  Type_Income_State
```

```
servant  \
0        2.75         0                       0
0
1        2.75         0                       0
0
2        0.68         1                       0
0
3        0.68         1                       0
0
4        6.77         0                       0
0

   ...  Type_Occupation_Laborers  Type_Occupation_Low-skill
Laborers  \
0  ...                         0                                    0

1  ...                         0                                    0

2  ...                         1                                    0

3  ...                         1                                    0

4  ...                         0                                    0


   Type_Occupation_Managers  Type_Occupation_Medicine staff  \
0                         0                               0
1                         0                               0
2                         0                               0
3                         0                               0
4                         0                               0

   Type_Occupation_Private service staff  Type_Occupation_Realty
agents  \
0                                      0
0
1                                      0
0
2                                      0
0
3                                      0
0
4                                      0
0

   Type_Occupation_Sales staff  Type_Occupation_Secretaries  \
0                            0                            0
1                            0                            0
2                            0                            0
3                            0                            0
```

```
4                            1                            0

   Type_Occupation_Security staff  Type_Occupation_Waiters/barmen
staff
0                                0
0
1                                0
0
2                                0
0
3                                0
0
4                                0
0

[5 rows x 41 columns]
```

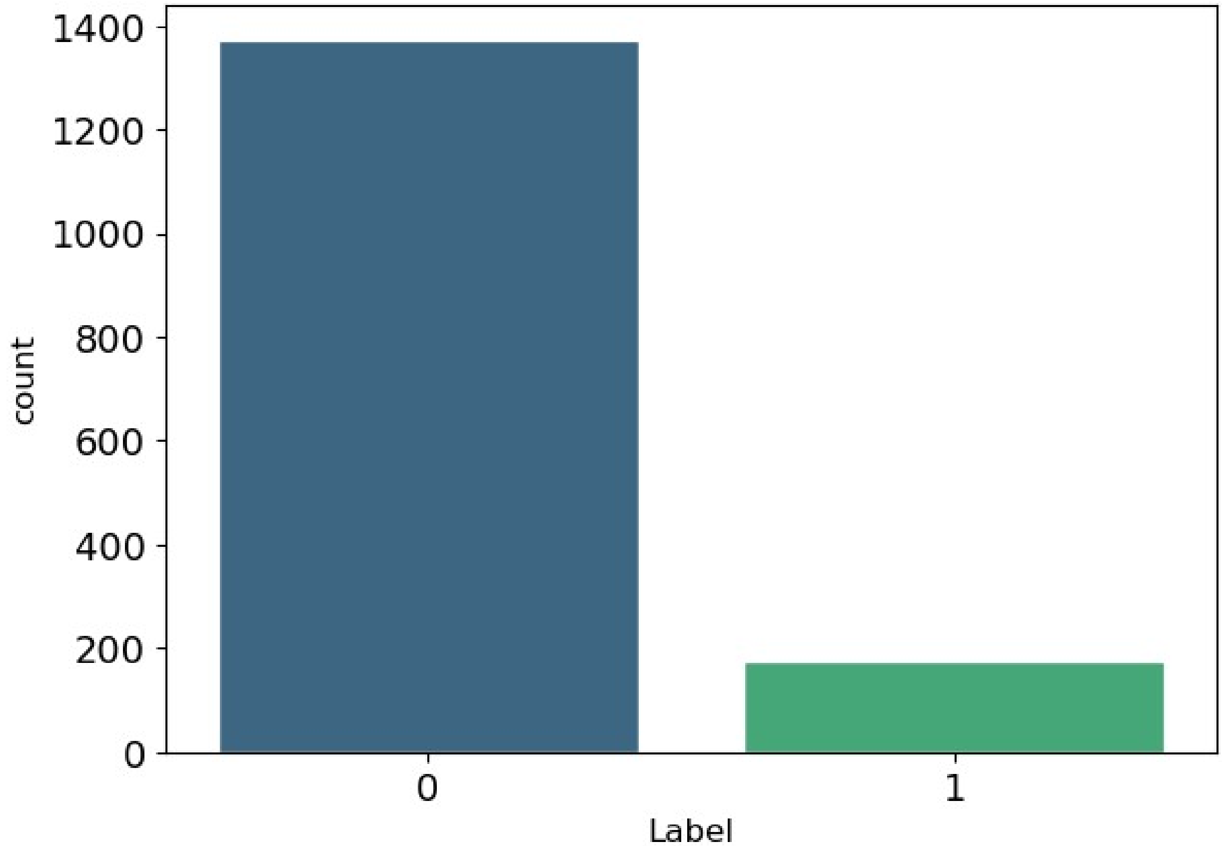
Loading Target Variable Dataset:

We will load the target table and merge both input and output table for further analysis to split and train our machine learning model.

```
data_op= pd.read_csv('Credit_card_label[1].csv')

data_op.head()

    Ind_ID  label
0  5008827      1
1  5009744      1
2  5009746      1
3  5009749      1
4  5009752      1
```

Visualizing target Variable:

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots(1, 1, figsize=(7,5), sharex=True)
sns.countplot(data=data_op,
x='label',edgecolor="white",palette="viridis",order=data_op["label"].v
alue_counts().index)
total = data_op['label'].value_counts().sum()
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.xlabel('Label', fontsize=12)
plt.ylabel('count', fontsize=12)

plt.show()
```



Merge the target data

```
M_data=pd.merge(data,data_op,on= 'Ind_ID')

M_data

      Ind_ID  CHILDREN  Annual_income  EDUCATION  Family_Members  Age  \
0    5018498         0        90000.0          3               2   51
1    5018503         0        90000.0          3               2   51
2    5021310         0       270000.0          3               2   46
3    5021314         0       270000.0          3               2   46
4    5021430         0       126000.0          4               1   51
..       ...       ...            ...        ...             ...  ...
968  5024049         1       144000.0          4               3   35
969  5118268         1       360000.0          3               3   30
```

```
970  5023655         0       225000.0          1                1   28

971  5115992         2       180000.0          4                4   36

972  5118219         0       270000.0          3                2   41


     Experience  GENDER_M  Type_Income_Pensioner  Type_Income_State
servant  \
0          2.75         0                      0
0
1          2.75         0                      0
0
2          0.68         1                      0
0
3          0.68         1                      0
0
4          6.77         0                      0
0
..          ...       ...                    ...
...
968        8.01         0                      0
0
969        9.69         1                      0
1
970        3.31         0                      0
0
971        6.79         1                      0
0
972        1.77         1                      0
0

     ...  Type_Occupation_Low-skill Laborers  Type_Occupation_Managers
\
0    ...                                   0                         0

1    ...                                   0                         0

2    ...                                   0                         0

3    ...                                   0                         0

4    ...                                   0                         0

..   ...                                 ...                       ...

968  ...                                   0                         0

969  ...                                   0                         0
```

```
970  ...                                    0                         0

971  ...                                    0                         1

972  ...                                    0                         0


     Type_Occupation_Medicine staff  Type_Occupation_Private service 
staff  \
0                                 0                                  
0  
1                                 0                                  
0  
2                                 0                                  
0  
3                                 0                                  
0  
4                                 0                                  
0  
..                              ...                                  
...  
968                               0                                  
0  
969                               0                                  
0  
970                               0                                  
0  
971                               0                                  
0  
972                               0                                  
0  

     Type_Occupation_Realty agents  Type_Occupation_Sales staff  \
0                                0                            0   
1                                0                            0   
2                                0                            0   
3                                0                            0   
4                                0                            1   
..                             ...                          ...   
968                              0                            0   
969                              0                            0   
970                              0                            0   
971                              0                            0   
972                              0                            0   

     Type_Occupation_Secretaries  Type_Occupation_Security staff  \
0                              0                               0   
1                              0                               0   
2                              0                               0   
3                              0                               0   
```

```
4                            0                               0
..                         ...                             ...
968                          0                               0
969                          0                               0
970                          0                               0
971                          0                               0
972                          0                               0

     Type_Occupation_Waiters/barmen staff  label
0                                       0      1
1                                       0      1
2                                       0      1
3                                       0      1
4                                       0      1
..                                    ...    ...
968                                     0      0
969                                     0      0
970                                     0      0
971                                     0      0
972                                     0      0

[973 rows x 42 columns]
```

```
M_data.drop('Ind_ID',inplace=True,axis=1)

M_data
```

```
     CHILDREN  Annual_income  EDUCATION  Family_Members  Age
Experience  \
0           0        90000.0          3               2   51
2.75
1           0        90000.0          3               2   51
2.75
2           0       270000.0          3               2   46
0.68
3           0       270000.0          3               2   46
0.68
4           0       126000.0          4               1   51
6.77
..        ...            ...        ...             ...  ...          .
..
968         1       144000.0          4               3   35
8.01
969         1       360000.0          3               3   30
9.69
970         0       225000.0          1               1   28
3.31
971         2       180000.0          4               4   36
6.79
972         0       270000.0          3               2   41
```

```
1.77

     GENDER_M  Type_Income_Pensioner  Type_Income_State servant  \
0           0                      0                          0
1           0                      0                          0
2           1                      0                          0
3           1                      0                          0
4           0                      0                          0
..        ...                    ...                        ...
968         0                      0                          0
969         1                      0                          1
970         0                      0                          0
971         1                      0                          0
972         1                      0                          0

     Type_Income_Working  ...  Type_Occupation_Low-skill Laborers  \
0                      1  ...                                  0
1                      1  ...                                  0
2                      1  ...                                  0
3                      1  ...                                  0
4                      0  ...                                  0
..                   ...  ...                                ...
968                    1  ...                                  0
969                    0  ...                                  0
970                    0  ...                                  0
971                    1  ...                                  0
972                    1  ...                                  0

     Type_Occupation_Managers  Type_Occupation_Medicine staff  \
0                           0                               0
1                           0                               0
2                           0                               0
3                           0                               0
4                           0                               0
..                        ...                             ...
968                         0                               0
969                         0                               0
970                         0                               0
971                         1                               0
972                         0                               0

     Type_Occupation_Private service staff  Type_Occupation_Realty
agents  \
0                                        0
0
1                                        0
0
2                                        0
0
3                                        0
```

```
0
4                                         0
0
..                                      ...
...
968                                       0
0
969                                       0
0
970                                       0
0
971                                       0
0
972                                       0
0

     Type_Occupation_Sales staff  Type_Occupation_Secretaries  \
0                              0                            0
1                              0                            0
2                              0                            0
3                              0                            0
4                              1                            0
..                           ...                          ...
968                            0                            0
969                            0                            0
970                            0                            0
971                            0                            0
972                            0                            0

     Type_Occupation_Security staff  Type_Occupation_Waiters/barmen
staff  \
0                                 0
0
1                                 0
0
2                                 0
0
3                                 0
0
4                                 0
0
..                              ...
...
968                               0
0
969                               0
0
970                               0
0
```

```
971                                 0
0
972                                 0
0

     label
0        1
1        1
2        1
3        1
4        1
..     ...
968      0
969      0
970      0
971      0
972      0

[973 rows x 41 columns]
```

Train test split

In our project, we performed a train-test split, a common practice in machine learning.

This involved dividing our dataset into a training set and a testing set. The training set was used to teach our machine learning model the patterns and relationships in the data.

After training, we evaluated the model's performance using the testing set, which contained new, unseen data. This approach helped us ensure that our model could generalize well to real-world situations and avoid overfitting."

```
#Training and Testing
from sklearn.model_selection import train_test_split

df_train,df_test=train_test_split(M_data,test_size=0.2,train_size=0.8)
```

Feature Scaling:

```
from sklearn.preprocessing import StandardScaler

scaler_std=StandardScaler()

numvars=['Annual_income','Age','Experience','Family_Members','CHILDREN
'] #only columns which were numerical in start not encoded.

df_train[numvars] = scaler_std.fit_transform(df_train[numvars])

numvars=['Annual_income','Age','Experience','Family_Members','CHILDREN
']
```

```
df_test[numvars] = scaler_std.transform(df_test[numvars])

#splitting the data for testing.
X_test=df_test.drop('label',axis=1)      #Input Testing
Y_test=df_test['label']                 #Output Testing

#splitting the data for training.
X_train=df_train.drop('label',axis=1) #Input  Training
Y_train=df_train['label']
```

ML Algorithms:

1.)Logistic Regression:

We use logistic regression, a statistical method, to analyze and model relationships between variables. Suited for binary classification tasks, it estimates the probability of an outcome based on input features.

By fitting a logistic curve to the data, it classifies instances into two classes. Logistic regression aids in understanding factors influencing outcomes and predicting future events

```
# Import MinMaxScaler
from sklearn.preprocessing import MinMaxScaler
# Instantiate MinMaxScaler and use it to rescale X_train and X_test
scaler = MinMaxScaler(feature_range=(0,1))
rescaledxTrain = scaler.fit_transform(X_train)
rescaledxTest = scaler.transform(X_test)

# Import LogisticRegression
from sklearn.linear_model import LogisticRegression
# Instantiate a LogisticRegression classifier with default parameter
values
logreg = LogisticRegression()

# Fit logreg to the train set
logreg.fit(rescaledxTrain, Y_train)

LogisticRegression()

# Import confusion_matrix
from sklearn.metrics import confusion_matrix
# Use logreg to predict instances from the test set and store it
y_pred1 = logreg.predict(rescaledxTest)
y_pred2 = logreg.predict(rescaledxTrain)

# Get the accuracy score of logreg model and print it
print("Test: Accuracy = ", logreg.score(rescaledxTest,Y_test))
print("Train: Accuracy = ", logreg.score(rescaledxTrain,Y_train))
```

```
# Print the confusion matrix of the logreg model
confusion_matrix(Y_test,y_pred1)
```

```
Test: Accuracy =  0.9128205128205128
Train: Accuracy =  0.8984575835475579
```

```
array([[178,   0],
       [ 17,   0]], dtype=int64)
```

```
Y_test.shape,y_pred1.shape
```

```
((195,), (195,))
```

2.)Decision Tree:

Implemented decision tree algorithm, a machine learning technique for classification and regression tasks. The tree-like model makes decisions based on input features, branching to different outcomes. It recursively splits data to maximize information gain and minimize impurity, resulting in a predictive model.

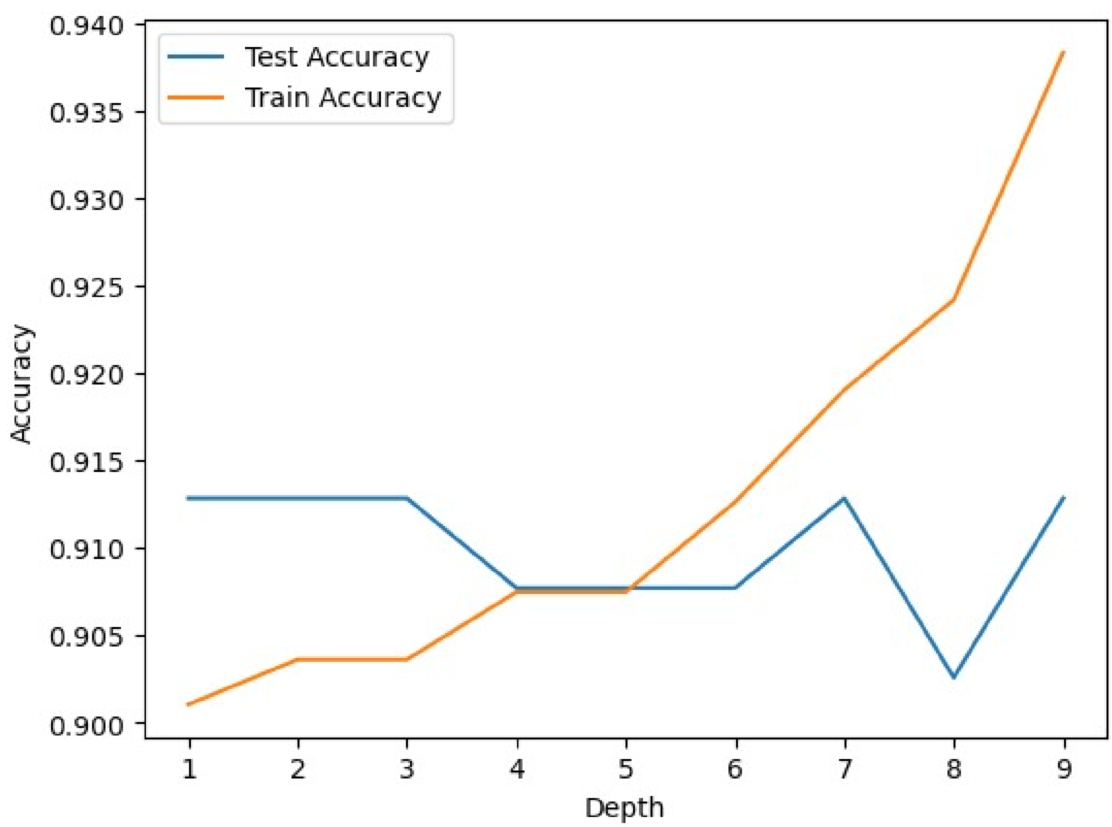
Decision trees are interpretable, useful for feature selection, and can handle nonlinear relationships. They can be prone to overfitting but are often part of ensemble methods like Random Forests.

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import GradientBoostingClassifier
from pandas import DataFrame
import matplotlib.pyplot as plt

train_acc=[]
test_acc=[]
list_score=[]
p=[]
from sklearn import tree
for i in range(1, 10):

    dtc = tree.DecisionTreeClassifier(max_depth = i ,random_state = 0)
    dtc.fit(X_train,Y_train)

    train_pred = dtc.predict(X_train)
    #train_acc.append(score(train_pred, yTrain))

    test_pred = dtc.predict(X_test)
    #test_acc.append(score(test_pred, yTest))
    test_acc = accuracy_score(Y_test, test_pred)
    train_acc = accuracy_score(Y_train, train_pred)
    print(i,'Train score:',train_acc,'Test score:',test_acc)

    list_score.append([i,accuracy_score(train_pred,
Y_train),accuracy_score(test_pred, Y_test)])
```

```
df2 = DataFrame(list_score,columns=['Depth','Train Accuracy','Test
Accuracy'])
plt.plot(df2['Depth'],df2['Test Accuracy'],label='Test Accuracy')
plt.plot(df2['Depth'],df2['Train Accuracy'],label='Train Accuracy')
plt.xlabel('Depth')
plt.ylabel('Accuracy')
plt.legend()
```

```
1 Train score: 0.9010282776349614 Test score: 0.9128205128205128
2 Train score: 0.903598971722365 Test score: 0.9128205128205128
3 Train score: 0.903598971722365 Test score: 0.9128205128205128
4 Train score: 0.9074550128534704 Test score: 0.9076923076923077
5 Train score: 0.9074550128534704 Test score: 0.9076923076923077
6 Train score: 0.9125964010282777 Test score: 0.9076923076923077
7 Train score: 0.9190231362467867 Test score: 0.9128205128205128
8 Train score: 0.9241645244215938 Test score: 0.9025641025641026
9 Train score: 0.9383033419023136 Test score: 0.9128205128205128
```

```
<matplotlib.legend.Legend at 0x213dc69c490>
```




```
 dtc = tree.DecisionTreeClassifier(max_depth = 4 ,random_state = 0)
dtc.fit(X_train,Y_train)
```

```
train_pred = dtc.predict(X_train)
    #train_acc.append(score(train_pred, yTrain))

test_pred = dtc.predict(X_test)
    #test_acc.append(score(test_pred, yTest))
test_acc = accuracy_score(Y_test, test_pred)
train_acc = accuracy_score(Y_train, train_pred)
print('Train score:',train_acc,'Test score:',test_acc)

Train score: 0.9074550128534704 Test score: 0.9076923076923077
```

3.) Gradient Boost:

Applied gradient boosting, an ensemble learning technique, to improve model performance. It combines multiple weak learners sequentially, each correcting errors of its predecessor. During training, it assigns higher weights to misclassified instances, focusing on difficult cases.

By aggregating predictions, it creates a strong model that excels in predictive accuracy. Gradient boosting is widely used due to its ability to handle complex relationships and reduce overfitting.

```
clf = GradientBoostingClassifier(random_state=0)
clf.fit(X_train, Y_train)

train_predict = clf.predict(X_train)
test_predict = clf.predict(X_test)


test_acc_grad = accuracy_score(Y_test, test_predict)
train_acc_grad = accuracy_score(Y_train, train_predict)
print('Train score:',train_acc_grad,'Test score:',test_acc_grad)

Train score: 0.9434447300771208 Test score: 0.9128205128205128
```

4.) Random forest:

Utilized random forest, an ensemble learning algorithm, for robust predictions. Comprising multiple decision trees, it reduces overfitting by aggregating their outputs. Each tree is trained on a random subset of data and features, enhancing diversity.

By averaging or voting over individual tree predictions, random forest provides accurate results, handles noisy data, and identifies important features. It's suitable for classification and regression tasks and is resistant to outliers

```
from sklearn.ensemble import RandomForestClassifier
classifier= RandomForestClassifier(n_estimators= 10,
criterion="entropy")
classifier.fit(X_train, Y_train)
```

```
RandomForestClassifier(criterion='entropy', n_estimators=10)

#Predicting the test set result
Y_Pred_random= classifier.predict(X_test)

train_predict_random = classifier.predict(X_train)


#Creating the Confusion matrix
from sklearn.metrics import confusion_matrix
cm= confusion_matrix(Y_test, Y_Pred_random)
cm

array([[177,   1],
       [  9,   8]], dtype=int64)

test_acc_random= accuracy_score(Y_test, Y_Pred_random)
train_acc_random = accuracy_score(Y_train, train_predict_random)
print('Train score:',train_acc_random,'Test score:',test_acc_random)

Train score: 0.9781491002570694 Test score: 0.9487179487179487
```

Conclusion:

```
Algo_data=pd.DataFrame()
Algo_data['Model']=['Logistic Regression','Decision Tree','Gradient Boost','Random Forest']
Algo_data['Train_Accuracy']=[logreg.score(rescaledxTrain,Y_train),train_acc,train_acc_grad,train_acc_random]
Algo_data['Test_Accuracy']=[logreg.score(rescaledxTest,Y_test),test_acc,test_acc_grad ,test_acc_random]
Algo_data
```

	Model	Train_Accuracy	Test_Accuracy
0	Logistic Regression	0.898458	0.912821
1	Decision Tree	0.907455	0.907692
2	Gradient Boost	0.943445	0.912821
3	Random Forest	0.978149	0.948718

In our analysis, we evaluated four different models on the given dataset.

-The Logistic Regression model showed solid performance with a train accuracy of 89.8% and a test accuracy of 91.3%.

-The Decision Tree model achieved a train accuracy of 90.7% and a test accuracy of 90.8%, indicating its effectiveness.

-Gradient Boost exhibited strong predictive capabilities, achieving a train accuracy of 94.3% and a test accuracy of 91.3%.

-However, the Random Forest model emerged as the top performer, attaining a train accuracy of 97.8% and a test accuracy of 94.9%.

Based on these results, we conclude that the Random Forest model is the most suitable choice for this dataset, offering both high training accuracy and strong generalization to unseen data.