Introduction :

Objective : To predict whether a customer get approval for the credit card or not

Assumptions: Customers must fill out a form, either physically or online, to apply for a credit card. The application data is used to evaluate the applicant's creditworthiness. The decision is made using the application data in addition to the Credit Bureau Score, such as the FICO Score in the US or the CIBIL Score in India, and other internal information on the applicants. Additionally, the banks are rapidly taking a lot of outside data into account to enhance the caliber of credit judgements.

Brief introduction to the data: There is 2 data tables, the first one is the information of each customer, the second show the approval records for credit card.

points to know: Label: 0 is application approved and 1 is application rejected.

Steps involved

1. Data collection
2. Finding out the target variable
3. Data cleaning 3.1 Removing the duplicate values 3.2 Filling the null values 3.3 Removing the Outliers
4. Feature selection
5. Feature scaling (standardization)
6. Model training and test with different methods
7. Finding the best model
8. deploy the model
9. conclusions

Questions

1.Why is your proposal important in today’s world? How predicting a good client is worthy for a bank?

This proposal is very important for the bank sectors to predict that can customer will get an approval for the credit card or not. One can predict the good client with his annual income and expenditure and based one cibil performence and how he managing all these things

2.How is it going to impact the banking sector?

Based on the developed model one can easily predict that we can provide the credit card or not and that will ease the work for any banks in banking sector

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

This proposed method can be used for any bank and if some new data will add then there is certain changes we need to do and then it will be used

ML PROJECT_1 CREDIT CARD APPROVAL PREDICTION:

1. DATA COLLECTION

```
# importing all necessary libraries
import pandas as pd                 # importing pandas library (used for retrive data from pandas dataframes and all related records)
import numpy as np                  # importing numpy library (used for retrive data from array based)
import matplotlib.pyplot as plt     # importing matplotlib.pyplot library (used for plots between the features)
import seaborn as sns               # importing seaborn library (advanced ploting library)
```

Uploading the necessary files

```
df1=pd.read_csv('/content/Credit_card.csv')       # read_csv is a function used to read and return the normal csv file to read here
df2=pd.read_csv('/content/Credit_card_label.csv')
df1.head(2)                                        # inorder to get first few records from the given data df1

    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

            Type_Income         EDUCATION Marital_status
Housing_type  \
0             Pensioner  Higher education        Married  House / apartment
1  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

  Type_Occupation  Family_Members
0             NaN               2
1             NaN               2
```

checking both dataframes containing same records or not

```
print(df1['Ind_ID'].nunique(),df2['Ind_ID'].nunique())

1548 1548
```

we found that both the tables have same number of unique records so we are going to merge these two datasets using merge function

```
df=df1.merge(df2)                                        # merge is used to
club those above two csv files
df.head()                                                # head is used to
return top 5 records from the overall data

    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  label
0             NaN               2      1
1             NaN               2      1
```

```
2              NaN                  2      1
3              NaN                  2      1
4              NaN                  2      1
```

```
df.shape  #shape functiion used to determine the numbers rows and columns in the given dataframe
```

```
(1548, 19)
```

```
df.info()               # info function describes about the column information where there is any null values present or not
                        # and defines the datatype of each column or feature
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1548 entries, 0 to 1547
Data columns (total 19 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
 18  label            1548 non-null   int64
dtypes: float64(2), int64(9), object(8)
memory usage: 241.9+ KB
```

OBSERVATIONS:

1.Here total 19 columns and in which 12 columns are numerical type(dtype with int and float64) and 7 are categorical type (object type)

2.and total number of rows are 1548 and some columns are not having 1548 records so the remaining have filled with null values

3.we need to replace the null values with the according values

2.DATA CLEANING:

a part of data cleaning, first step is to remove the duplicate records and check those are removed or not

```
df=df.drop('Ind_ID',axis=1) # drop function basically used to delete the column(ind_id is column with unique values
                            # if we drop this then we can find the duplicate values )

df.shape                   # shape function gives the rows and columns of the data before removing duplicate records

(1548, 18)

df.duplicated().sum()       # this is used to find the number of duplicate records

162

df.drop_duplicates(inplace=True)  # this is used to delete the duplicate records

df.shape                    # after removing the duplicates we can clealy observe the record numbers

(1386, 18)

df.duplicated().sum()       # this is for checking the duplicates the removed or not and those are totally removed

0
```

second step is checking for missing values

```
df.isnull().sum()            # this is used to find out how many null values present in each

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     438
Family_Members        0
label                 0
dtype: int64
```

for this datset we have some missing or null values in some columns like Annual_income, Birthday_count,Type_occupation.So we have to fill those values

```
ms=df.isnull().mean()*100        # this is used to find out how much pecentage missing values are there in each column
ms
```

```
GENDER               0.505051
Car_Owner            0.000000
Propert_Owner        0.000000
CHILDREN             0.000000
Annual_income        1.659452
Type_Income          0.000000
EDUCATION            0.000000
Marital_status       0.000000
Housing_type         0.000000
Birthday_count       1.587302
Employed_days        0.000000
Mobile_phone         0.000000
Work_Phone           0.000000
Phone                0.000000
EMAIL_ID             0.000000
Type_Occupation     31.601732
Family_Members       0.000000
label                0.000000
dtype: float64
```

The missing values are very less percentage in case of gender,Annual_income,Birthday_count but in case of type_occupation it is more than 30 percentage and we can not impute those values and that will lead to overfit the result case so i am deleting this column and filling the missing values in other missing value columns

```
plt.figure(figsize=(20,6))
sns.barplot(x=ms.index, y=ms.values, palette='coolwarm')
```

```
<Axes: >
```

From the above plot also we can observe the there are some missing values in the data in som columns like Gender,annual_income,type_occupation..

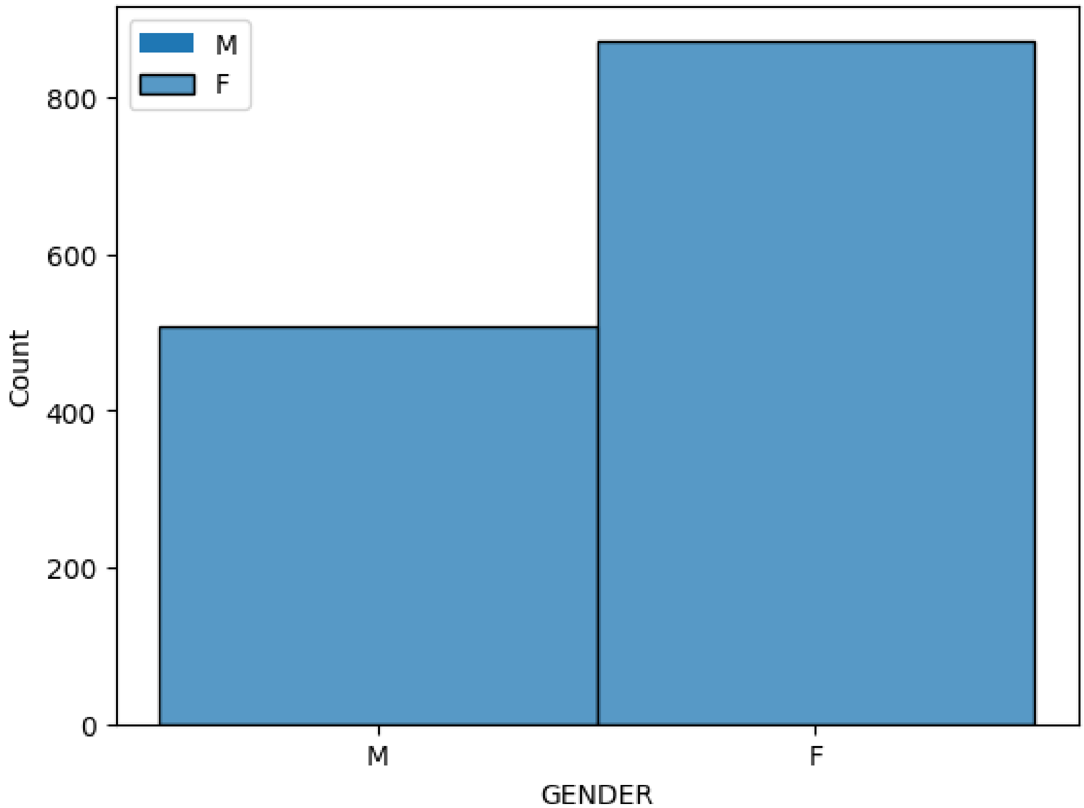
```
# Replacing the null values(missing) with some values based on imputations like mean,mode,median

y=df['GENDER'].value_counts() # gender is categorical so i am imputing the missing valueswith mode
y                              # this is used to get the values for each category to find the mode
```

```
F    872
M    507
Name: GENDER, dtype: int64
```

as the gender column has more number of female candidates,So we can treat Female F is like mode and we will the missing values with F

```
sns.histplot(df['GENDER'])
plt.legend(df.GENDER)
```

```
<matplotlib.legend.Legend at 0x78d404b437c0>
```



OBSERVATIONS:

As a plot we can clearly observe that female are more in number so that is the mode (most frequently occuring)

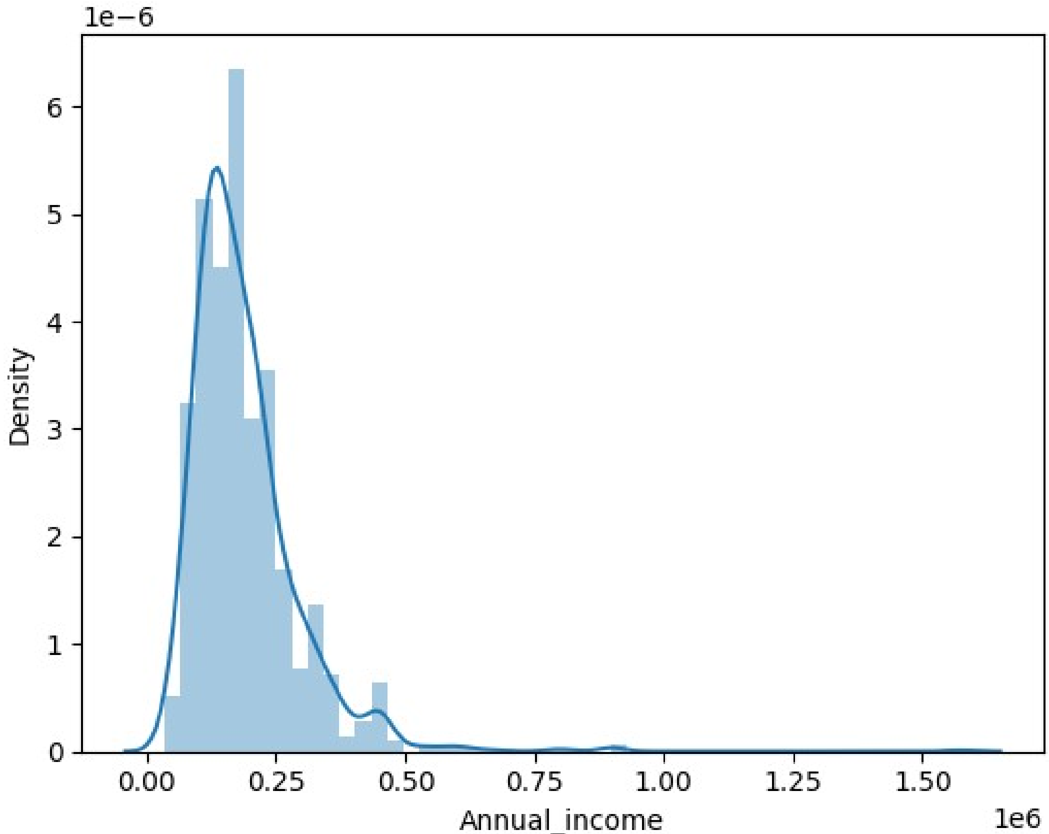
```
df['GENDER'].fillna('F',inplace=True)  # replacing the null values with F values

df['GENDER'].value_counts()   # checking those null values are filled or not with this function

F    879
M    507
Name: GENDER, dtype: int64

df.info()         # to get the information about the columns

<class 'pandas.core.frame.DataFrame'>
Int64Index: 1386 entries, 0 to 1547
Data columns (total 18 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   GENDER          1386 non-null   object
 1   Car_Owner       1386 non-null   object
 2   Propert_Owner   1386 non-null   object
```

```
 3   CHILDREN         1386 non-null   int64
 4   Annual_income    1363 non-null   float64
 5   Type_Income      1386 non-null   object
 6   EDUCATION        1386 non-null   object
 7   Marital_status   1386 non-null   object
 8   Housing_type     1386 non-null   object
 9   Birthday_count   1364 non-null   float64
 10  Employed_days    1386 non-null   int64
 11  Mobile_phone     1386 non-null   int64
 12  Work_Phone       1386 non-null   int64
 13  Phone            1386 non-null   int64
 14  EMAIL_ID         1386 non-null   int64
 15  Type_Occupation  948 non-null    object
 16  Family_Members   1386 non-null   int64
 17  label            1386 non-null   int64
dtypes: float64(2), int64(8), object(8)
memory usage: 205.7+ KB
```

for this datset we have some missing or null values in some columns like Annual_income, Birthday_count,Type_occupation.So we have to fill those values

next feature is Annual_income we will check the distribution and fill the missing values accordingly

```
sns.distplot(df['Annual_income']) # plotting annual_income column to know whether it is normally distributed or not

<ipython-input-189-c4cd223d6053>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['Annual_income']) # plotting annual_income column to know whether it is normally distributed or not

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

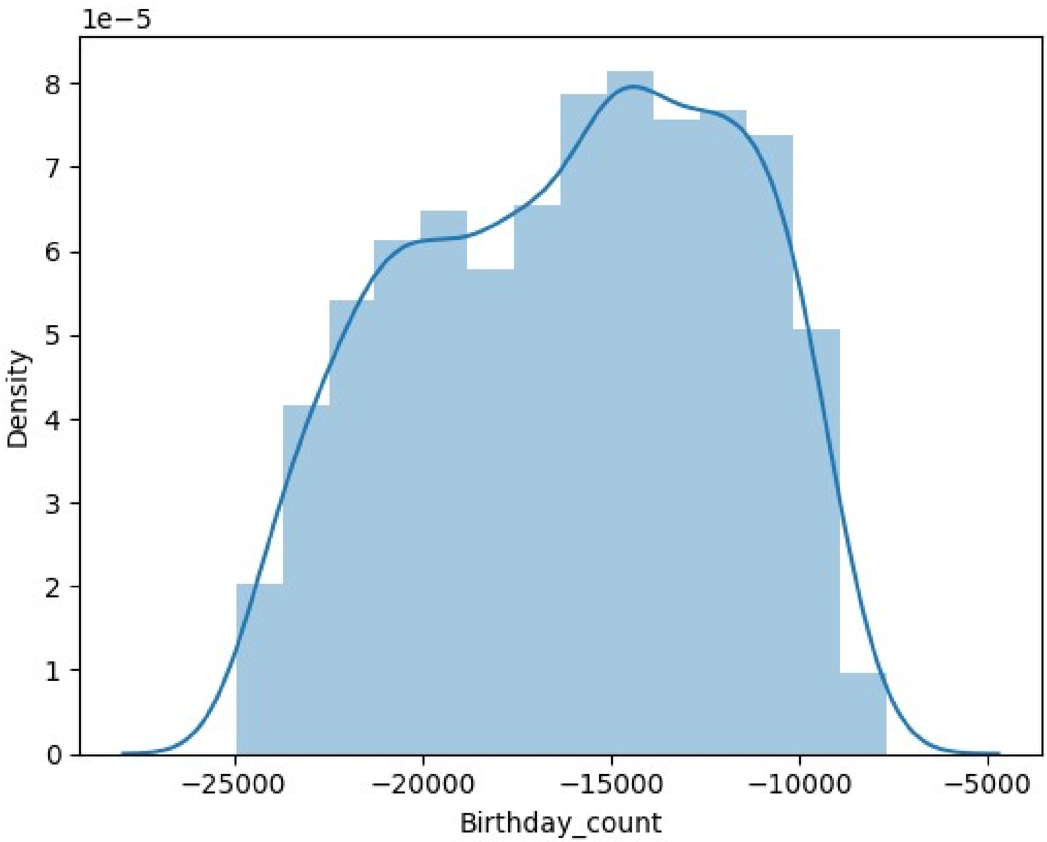


The above plot is not normal distributed one and we know that as plot Normal distributed then we will use mean for replacing the missing values and for non_normal distributed one we will use median for the missing values.So here inorder to fill the missing values we will use median.

```
df['Annual_income'].fillna(df['Annual_income'].median(),inplace=True)
# filling the annual_income null values with median value

sns.distplot(df['Birthday_count'])
```

```
<ipython-input-191-f1d3bfa4f3e0>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `histplot` (an axes-level function for
histograms).
```

```
For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['Birthday_count'])

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



The above plot is not normal distributed one and we know that as plot Normal distributed then we will use mean for replacing the missing values and for non_normal distributed one we will use median for the missing values.So here inorder to fill the missing values we will use median.

```
df['Birthday_count'].value_counts()

-18173.0    4
-24611.0    3
-10177.0    3
```

```
-21363.0    3
-13557.0    3
           ..
-21026.0    1
-11382.0    1
-10314.0    1
-19974.0    1
-16601.0    1
Name: Birthday_count, Length: 1270, dtype: int64
```

```
df['Birthday_count'].fillna(df['Birthday_count'].median(),inplace=True
)   # filling the Birthday_count column null values with median value
```

now we have a dataset with cleaned data that is without any missing values and duplicate values and we have to take care about the outliers in further steps

as data is proper we will go for the main insights from it

```
df.isnull().sum()
```

```
GENDER               0
Car_Owner            0
Propert_Owner        0
CHILDREN             0
Annual_income        0
Type_Income          0
EDUCATION            0
Marital_status       0
Housing_type         0
Birthday_count       0
Employed_days        0
Mobile_phone         0
Work_Phone           0
Phone                0
EMAIL_ID             0
Type_Occupation    438
Family_Members       0
label                0
dtype: int64
```

```
df.Type_Occupation.value_counts()
```

```
Laborers                 240
Core staff               155
Managers                 116
Sales staff              111
Drivers                   77
High skill tech staff     59
```

```
Medicine staff             43
Accountants                41
Security staff             21
Cleaning staff             20
Cooking staff              20
Private service staff      17
Low-skill Laborers          9
Secretaries                 8
Waiters/barmen staff        5
HR staff                    3
Realty agents               2
IT staff                    1
Name: Type_Occupation, dtype: int64
```

```
df.Type_Occupation.fillna('Laborers',inplace=True)        # replacing
the null values with most frequent one

df["AGE"] = ((-df["Birthday_count"])/365).apply(int)       # creating
the age column by converting the birthday_count column into years

df['AGE'].unique()                                           # to find
the unique values of age and checking for the conversion into numbers
(years)
```

```
array([51, 37, 42, 60, 49, 24, 46, 35, 32, 48, 33, 43, 59, 30, 54, 29,
55,
       57, 65, 52, 44, 63, 50, 31, 26, 28, 45, 67, 25, 64, 41, 38, 39,
34,
       47, 62, 53, 61, 58, 27, 56, 66, 40, 36, 23, 22, 68, 21])
```

```
df["YEAR_EMPLOYED"] = np.ceil(-(df["Employed_days"]/365))      #
creating the years_employed column by converting the employed_days
ie., divide by 365 inro years

df['YEAR_EMPLOYED']=np.where(df['YEAR_EMPLOYED']<0,0,df.YEAR_EMPLOYED)
# converting negitive values into zero years experience means freshers

df.drop(['Birthday_count','Employed_days'],axis=1,inplace=True)      #
we have converted the information into years format so now deleting
the those previous columns

df.head()                                           # checking the
first 5 rows whether they are converted or not
```

	GENDER	Car_Owner	Propert_Owner	CHILDREN	Annual_income	\
0	M	Y	Y	0	180000.0	
1	F	Y	N	0	315000.0	
2	F	Y	N	0	315000.0	
3	F	Y	N	0	162000.0	
5	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
5             Pensioner  Higher education        Married  House /
apartment

   Mobile_phone  Work_Phone  Phone  EMAIL_ID Type_Occupation
Family_Members  \
0             1           0      0         0        Laborers
2
1             1           1      1         0        Laborers
2
2             1           1      1         0        Laborers
2
3             1           1      1         0        Laborers
2
5             1           1      1         0        Laborers
2

   label  AGE  YEAR_EMPLOYED
0      1   51            0.0
1      1   37            2.0
2      1   42            2.0
3      1   37            2.0
5      1   37            2.0
```

```
# Define the filename for the Excel file
excel_file_name = 'cleaned.xlsx'

# Save the DataFrame to an Excel file
df.to_excel(excel_file_name, index=False)

from google.colab import files

# downloading the cleaned excel file for the use sql quaries and for
further development
files.download(excel_file_name)
```

```
<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>
```

The above dataset file is cleaned file without having any missing values,duplicates and that will help to know more information in MYSQL and that will used in MYSQL

DATA PRESENTATION:

1. This presentation section helps us to know and understand the data in a better way by visualization

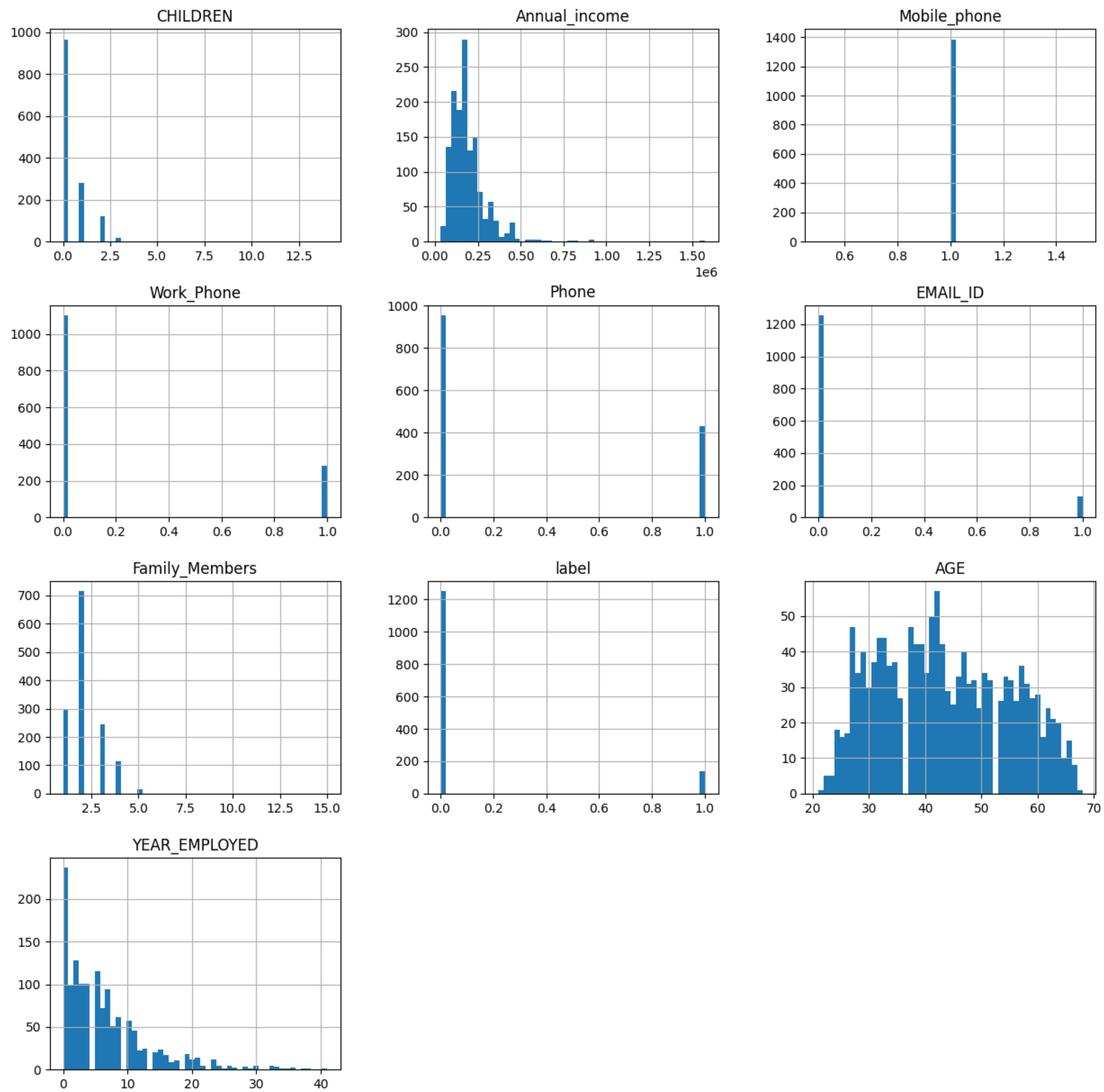
2. With the help of graphs or plots we can get the more information of data in a simple way

columns=['GENDER', 'Car_Owner', 'Propert_Owner', 'CHILDREN', 'Annual_income', 'Type_Income', 'EDUCATION', 'Marital_status', 'Housing_type', 'Birthday_count', 'Employed_days', 'Mobile_phone', 'Work_Phone', 'Phone', 'EMAIL_ID', 'Family_Members', 'label'], dtype='object'

observations of every feature in single plot:

```
df.hist(bins=50,figsize=(15,15))
```

```
array([[<Axes: title={'center': 'CHILDREN'}>,
        <Axes: title={'center': 'Annual_income'}>,
        <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: title={'center': 'label'}>,
        <Axes: title={'center': 'AGE'}>],
       [<Axes: title={'center': 'YEAR_EMPLOYED'}>, <Axes: >, <Axes:
>]],
      dtype=object)
```



observations:

1. From above plots we can clearly observe that phone,mobile_phone,work_phone,email features having the 0 or 1 and those will not have any impact on credit card approval process those data is only for communicative so we can drop those columns

2. Children and family members both are come under same category as we can see that from correlation those are highly correlated so we can drop the children column also

3. We will see the remaining all other features in following graphs with more detailed way

```
df.columns
```

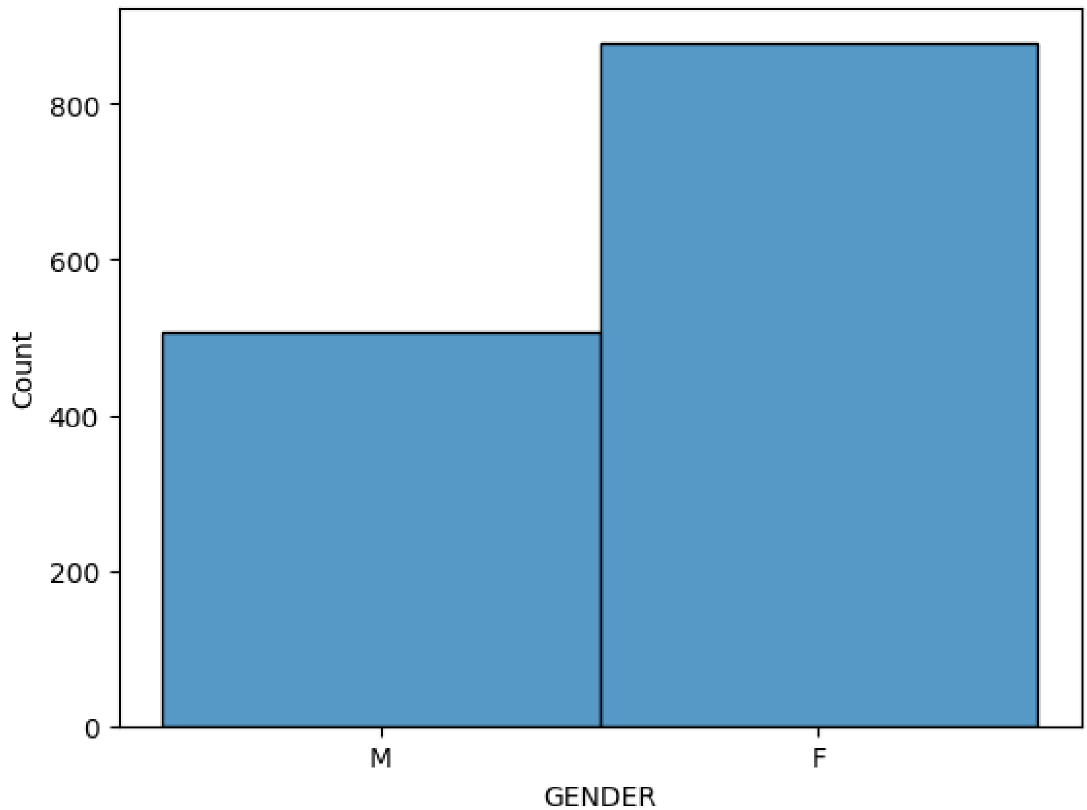
```
Index(['GENDER', 'Car_Owner', 'Propert_Owner', 'CHILDREN',
'Annual_income',
       'Type_Income', 'EDUCATION', 'Marital_status', 'Housing_type',
       'Mobile_phone', 'Work_Phone', 'Phone', 'EMAIL_ID',
'Type_Occupation',
       'Family_Members', 'label', 'AGE', 'YEAR_EMPLOYED'],
      dtype='object')
```

columns=['GENDER', 'Car_Owner', 'Propert_Owner', 'CHILDREN', 'Annual_income', 'Type_Income', 'EDUCATION', 'Marital_status', 'Housing_type', 'Birthday_count', 'Employed_days', 'Mobile_phone', 'Work_Phone', 'Phone', 'EMAIL_ID', 'Family_Members', 'label'],

GENDER

```
sns.histplot(df['GENDER'])
```

```
<Axes: xlabel='GENDER', ylabel='Count'>
```



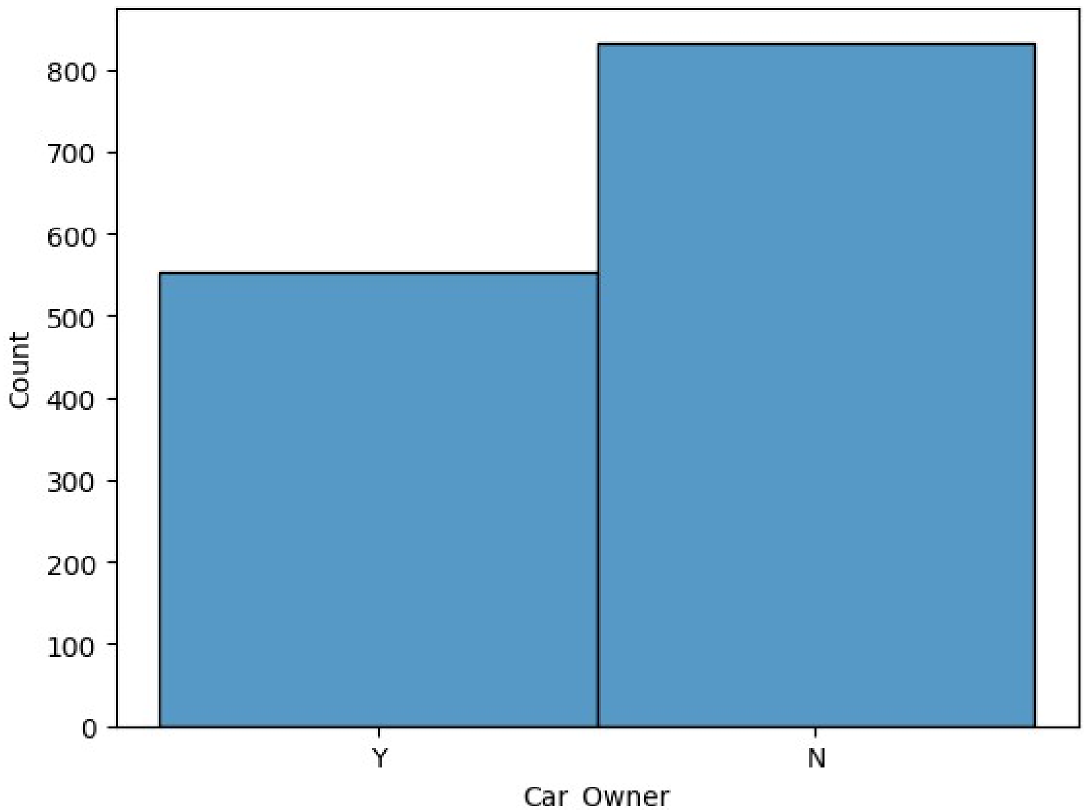
OBSERVATIONS:

1. There are more female persons are applying for credit cards than male persons

CAR OWNER

```
sns.histplot(df['Car_Owner'])
```

```
<Axes: xlabel='Car_Owner', ylabel='Count'>
```



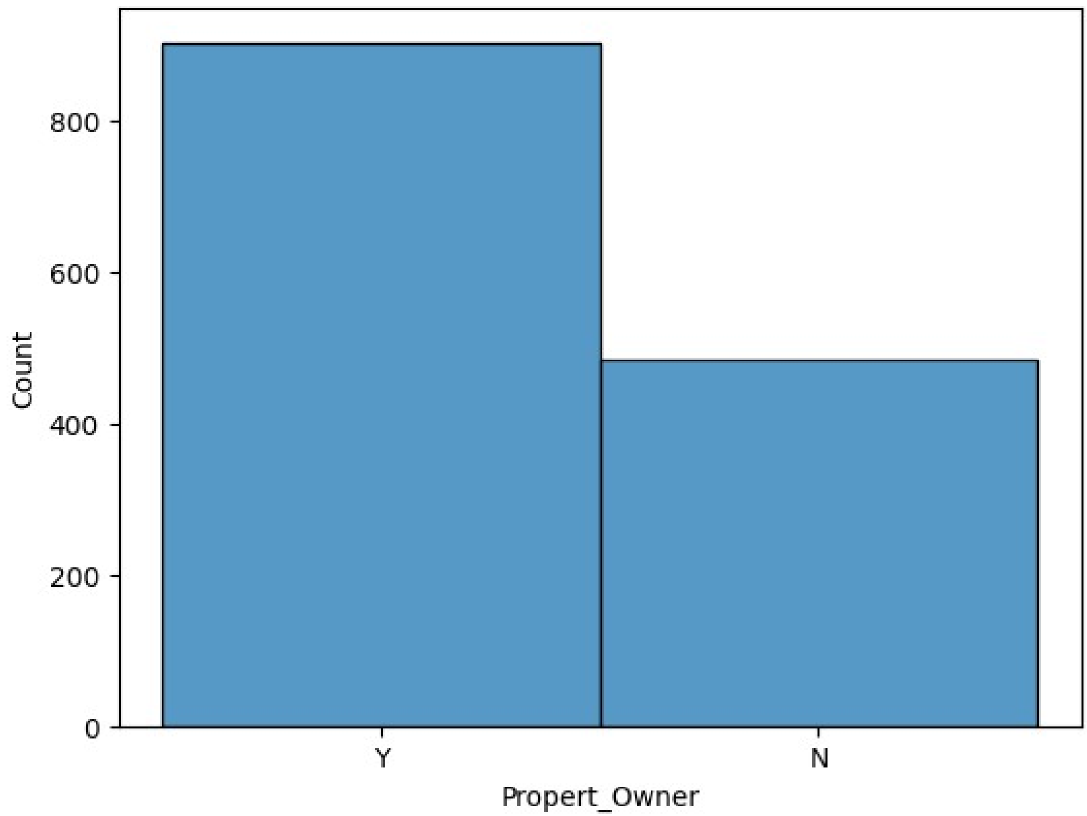
OBSERVATIONS:

From the plot we observed that there are persons without having an car are more in number compared with car persons are applying for the credit cards

Propert_Owner

```
sns.histplot(df['Propert_Owner'])
```

```
<Axes: xlabel='Propert_Owner', ylabel='Count'>
```



OBSERVATIONS:

From the plot we observed that there are persons with having property are more in number compared withot property persons are applying for the credit cards

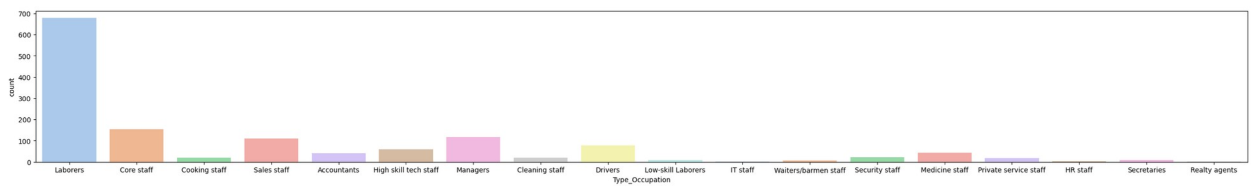
TYPE_OCCUPATION

```
plt.figure(figsize=(20,6))
sns.barplot(x=ms.index, y=ms.values, palette='coolwarm')
```

```
<Axes: >
```


```
plt.figure(figsize=(32,4))
sns.countplot(x=df['Type_Occupation'],palette='pastel')
```

```
<Axes: xlabel='Type_Occupation', ylabel='count'>
```



OBSERVATIONS:

we can clearly observe that Laborers are mostly applied for the credit card and next we have core staff and after that managers and after that sales staff are applied

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

```
Laborers                 678
Core staff               155
Managers                 116
Sales staff              111
Drivers                   77
High skill tech staff     59
Medicine staff            43
Accountants               41
Security staff            21
Cooking staff             20
Cleaning staff            20
Private service staff     17
Low-skill Laborers         9
Secretaries                8
Waiters/barmen staff       5
HR staff                   3
Realty agents              2
```

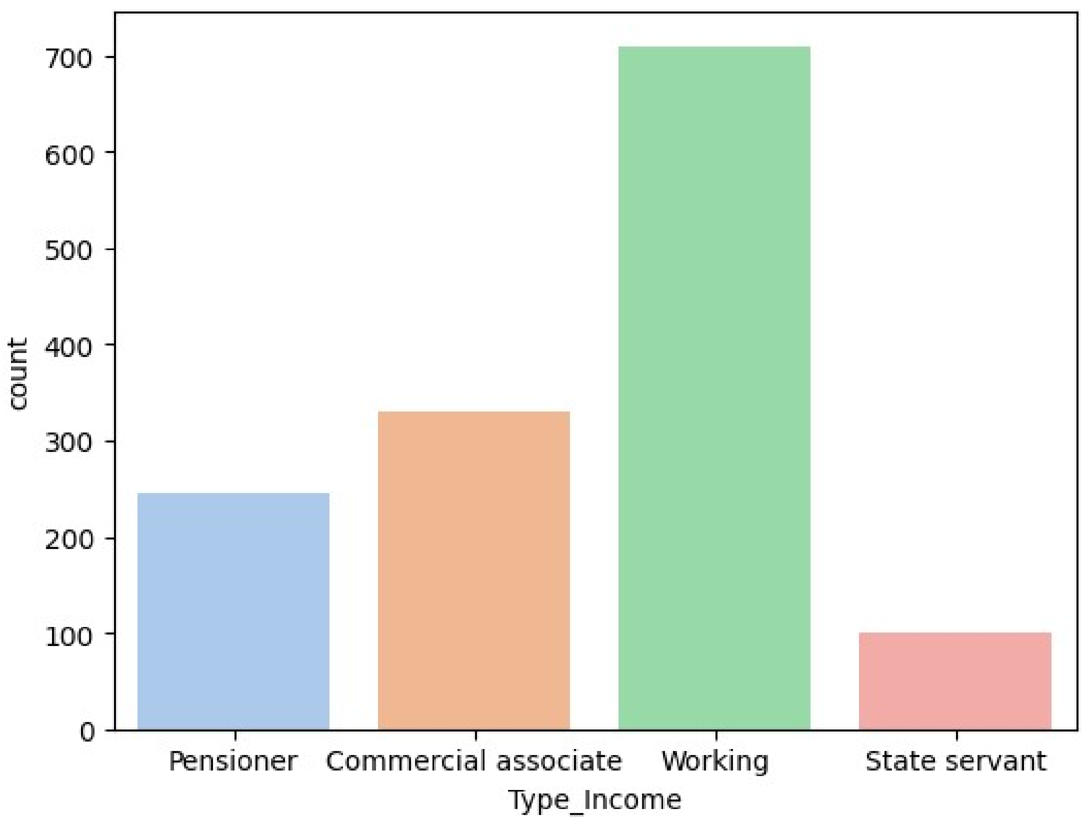
```
IT staff                   1
Name: Type_Occupation, dtype: int64
```

OBSERVATIONS:

We can observe that there are more numbera are Laborers

TYPE_INCOME:

```
df['Type_Income'].unique()

array(['Pensioner', 'Commercial associate', 'Working', 'State
servant'],
      dtype=object)

sns.countplot(x=df['Type_Income'],palette='pastel')

<Axes: xlabel='Type_Income', ylabel='count'>
```



OBSERVATIONS:

1. Mostly working pepole are applying for the credit cards and next commercial associate and next pensioner persons and last we have state servents

2.we should focus on working professional first and then commerical associates

3.Pensioners and state servents are the least percentages of applying for the creit cards

WORK_PHONE:

```
df.Work_Phone.value_counts()    # checking the number of persons having the work phone or not

0    1102
1     284
Name: Work_Phone, dtype: int64
```

OBSERVATIONS: Persons with work phone are less in number compared to persond without workphone

phone

```
df.Phone.unique()         # checking any other values except 0 or 1 in phone column

array([0, 1])
```

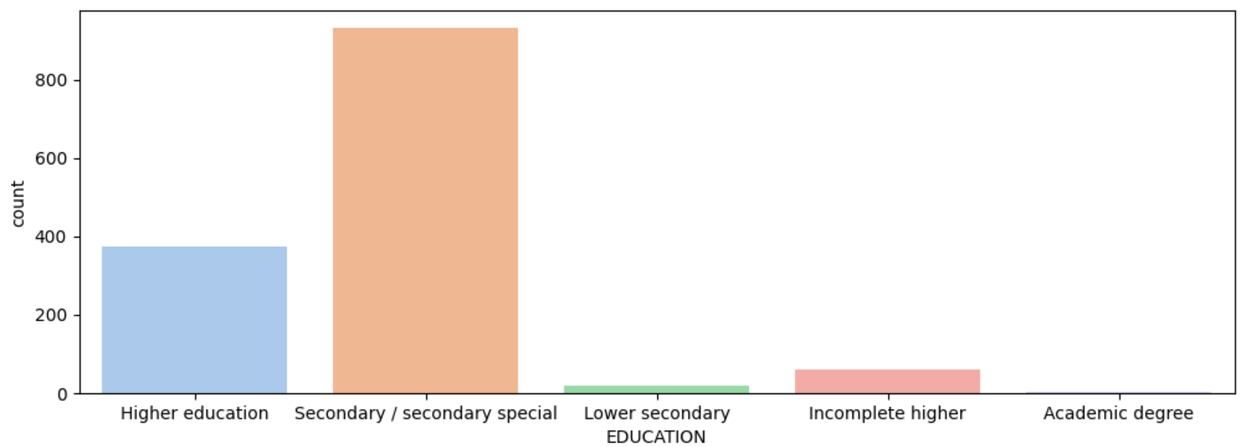
Dropping the unnecessary features from the dataset

```
df.drop(['Mobile_phone','EMAIL_ID','Work_Phone','Phone'],axis=1)  # dropping the  phone and mail columns because there is no use in data analysis part
df.head(2)

  GENDER Car_Owner Propert_Owner  CHILDREN  Annual_income  \
0      M         Y             Y         0       180000.0
1      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

   Mobile_phone  Work_Phone  Phone  EMAIL_ID Type_Occupation
```

```
Family_Members  \
0               1            0       0          0         Laborers
2
1               1            1       1          0         Laborers
2

   label  AGE  YEAR_EMPLOYED
0      1   51            0.0
1      1   37            2.0
```

EDUCATION:

```
plt.figure(figsize=(12,4))
sns.countplot(x=df['EDUCATION'],palette='pastel')

<Axes: xlabel='EDUCATION', ylabel='count'>
```



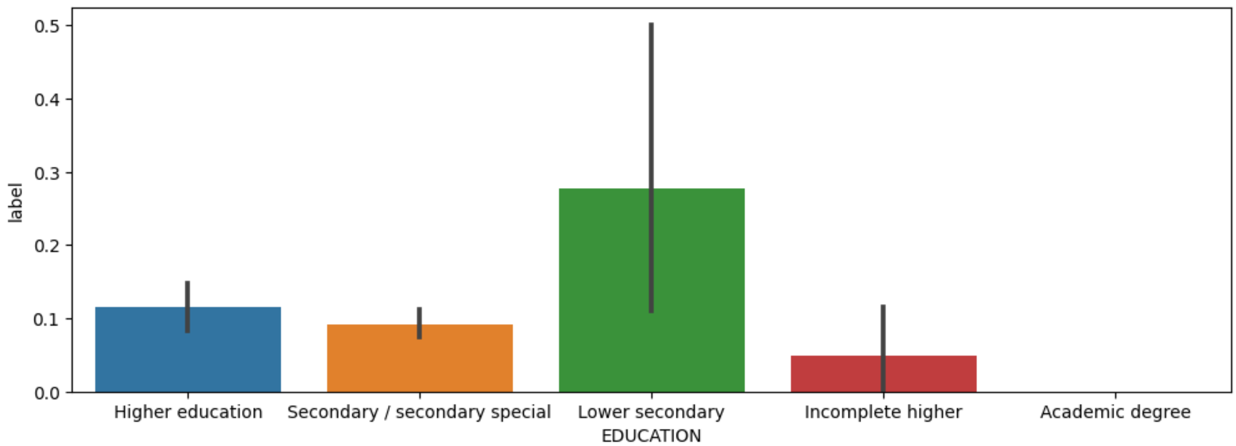
OBSERVATIONS:

1. We can observe that Secondary or secondary special are the most people that applying for the credit cards and then we have higher education persons

2. we have least bother about the lower education and academic degree persons

Another plot representing Education feature

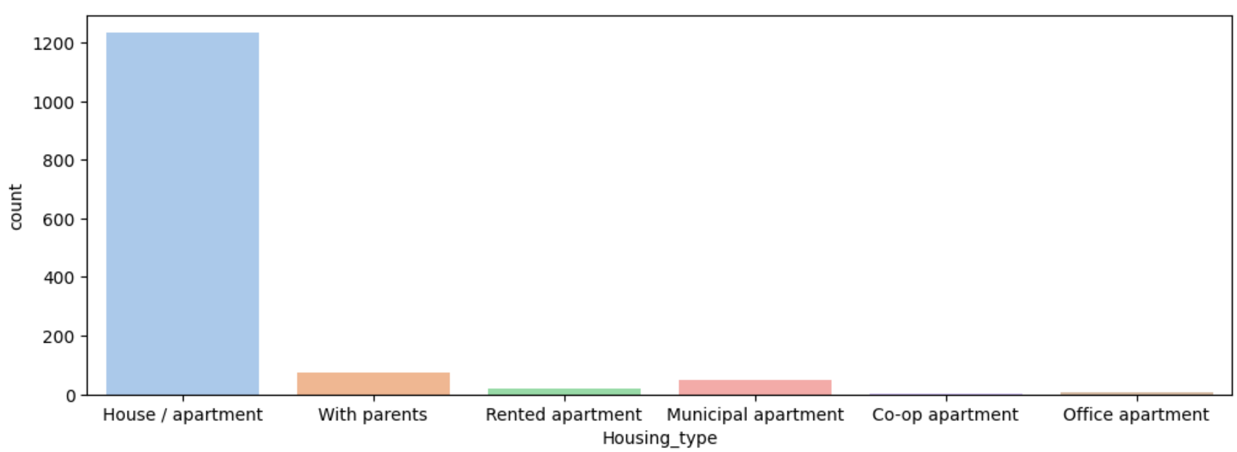
```
plt.figure(figsize=(12,4))
sns.barplot(x=df['EDUCATION'],y=df['label'])

<Axes: xlabel='EDUCATION', ylabel='label'>
```



HOUSING_TYPE

```
plt.figure(figsize=(12,4))
sns.countplot(x=df['Housing_type'],palette='pastel')
```

```
<Axes: xlabel='Housing_type', ylabel='count'>
```




```
df.Housing_type.value_counts()
```

```
House / apartment      1233
With parents             75
Municipal apartment      47
Rented apartment         20
Office apartment          8
Co-op apartment           3
Name: Housing_type, dtype: int64
```

Observations:

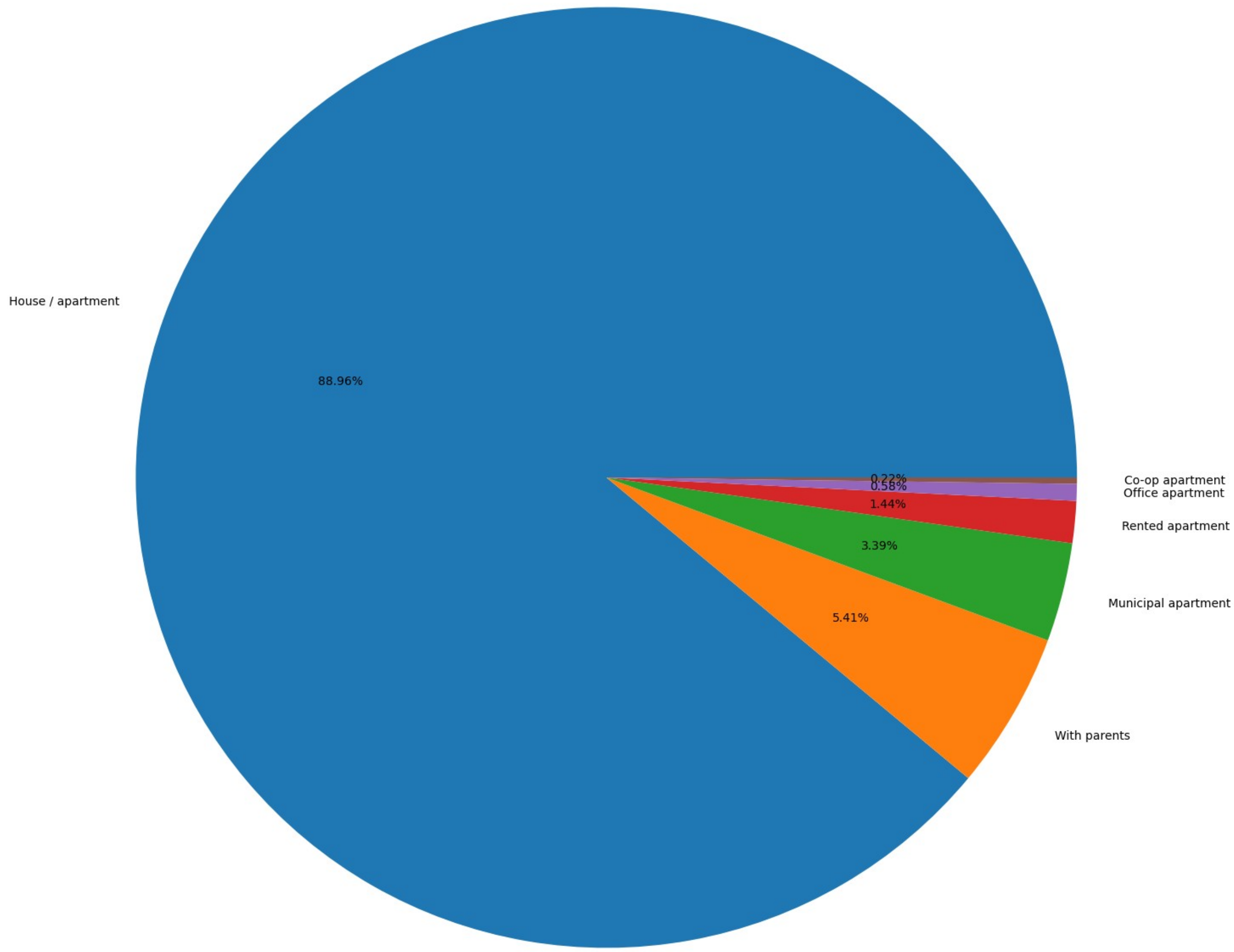
###1. Majoritively the house or apartment pesons are applying for the credit cards

###2. we least bother about the all other categories

Showing the percentage of different categories of housing type in pie chart

```
h_types=df.Housing_type.value_counts().index

h_num=df.Housing_type.value_counts().values

plt.figure(figsize=(18,18))
plt.pie(h_num,labels=h_types,autopct='%1.2f%%')
```

```
([<matplotlib.patches.Wedge at 0x78d40352eb90>,
  <matplotlib.patches.Wedge at 0x78d40352ea70>,
  <matplotlib.patches.Wedge at 0x78d40352f760>,
  <matplotlib.patches.Wedge at 0x78d40352fdf0>,
  <matplotlib.patches.Wedge at 0x78d4035584c0>,
  <matplotlib.patches.Wedge at 0x78d403558b50>],
 [Text(-1.0345120186549353, 0.3738781663302783, 'House / apartment'),
  Text(0.9526280128224967, -0.5499998810780429, 'With parents'),
  Text(1.0665975572488933, -0.26901607920474524, 'Municipal
apartment'),
  Text(1.0950191286065154, -0.10456150336442104, 'Rented apartment'),
  Text(1.0994462012176471, -0.03490058205940704, 'Office apartment'),
  Text(1.099974569201803, -0.007479780030710484, 'Co-op apartment')],
 [Text(-0.5642792829026918, 0.20393354527106086, '88.96%'),
  Text(0.5196152797213618, -0.2999999351334779, '5.41%'),
  Text(0.5817804857721236, -0.1467360432025883, '3.39%'),
  Text(0.5972831610580992, -0.0570335472896842, '1.44%'),
  Text(0.5996979279368984, -0.01903668112331293, '0.58%'),
  Text(0.5999861286555289, -0.004079880016751173, '0.22%')])
```

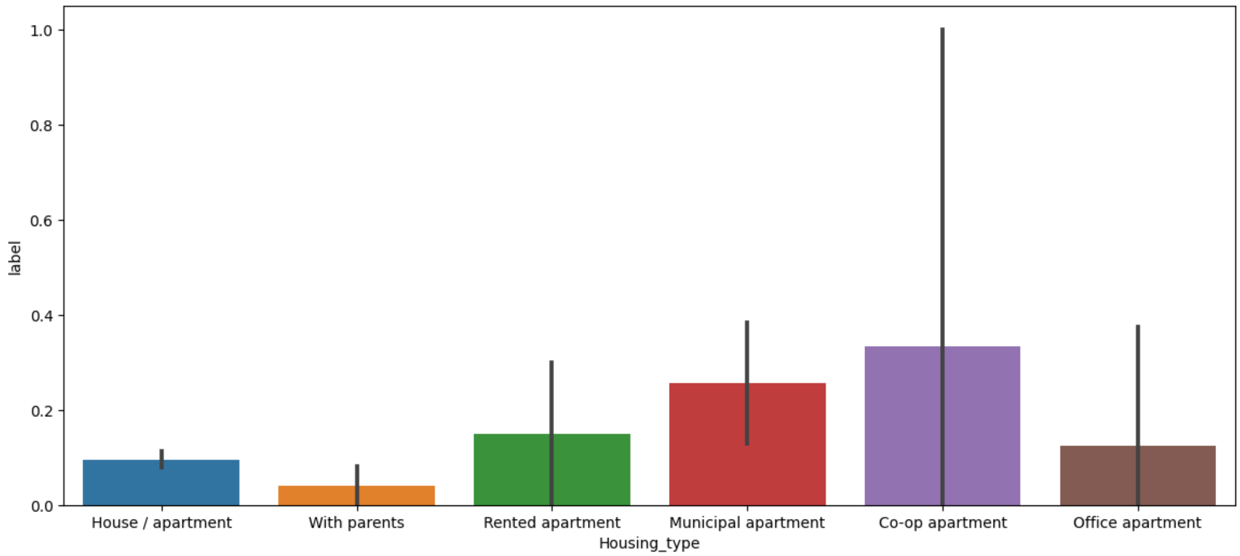


Observations:

###1. Majority of house or apartment pesons are applying for the credit cards

```
plt.figure(figsize=(14,6))
sns.barplot(x=df['Housing_type'],y=df['label'])
```

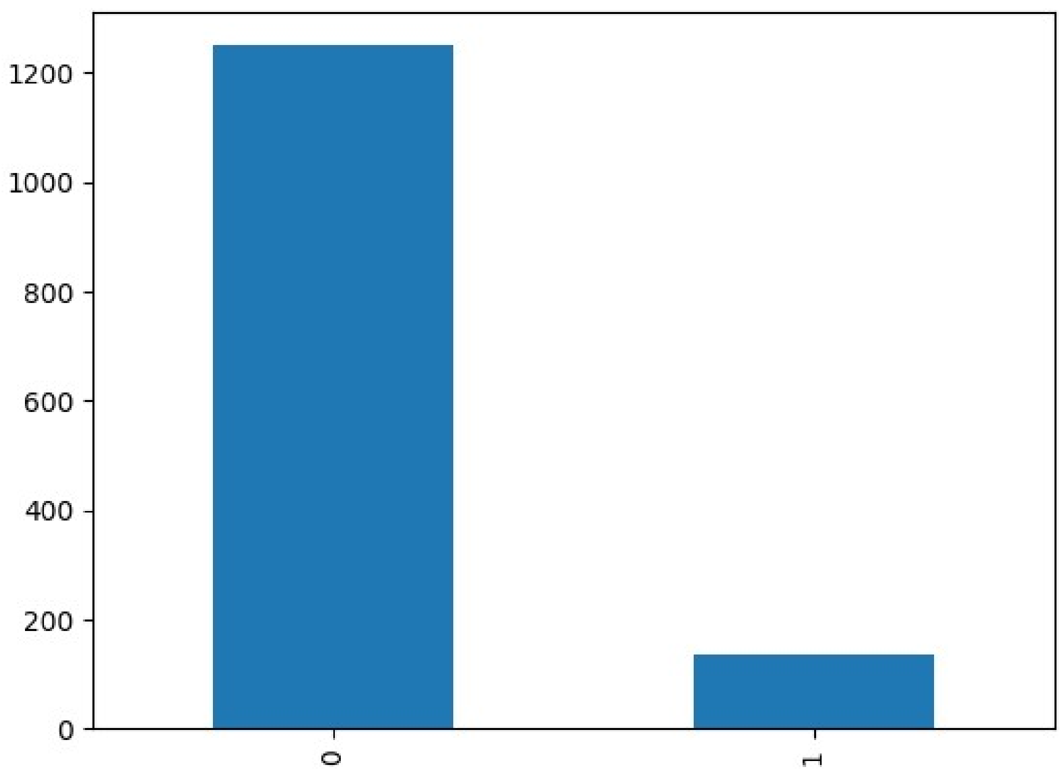
```
<Axes: xlabel='Housing_type', ylabel='label'>
```



LABEL(TARGET FEATURE)

```
df['label'].value_counts().plot(kind='bar')
```

```
<Axes: >
```



OBSERVATIONS:

0represents card approved and 1 represents not approved

we can say that the most of the persons got approval for thier applying the credit cards

```

```

OBSERVATIONS FROM PLOTS:

1. There are some columns like children,family_members there is no use

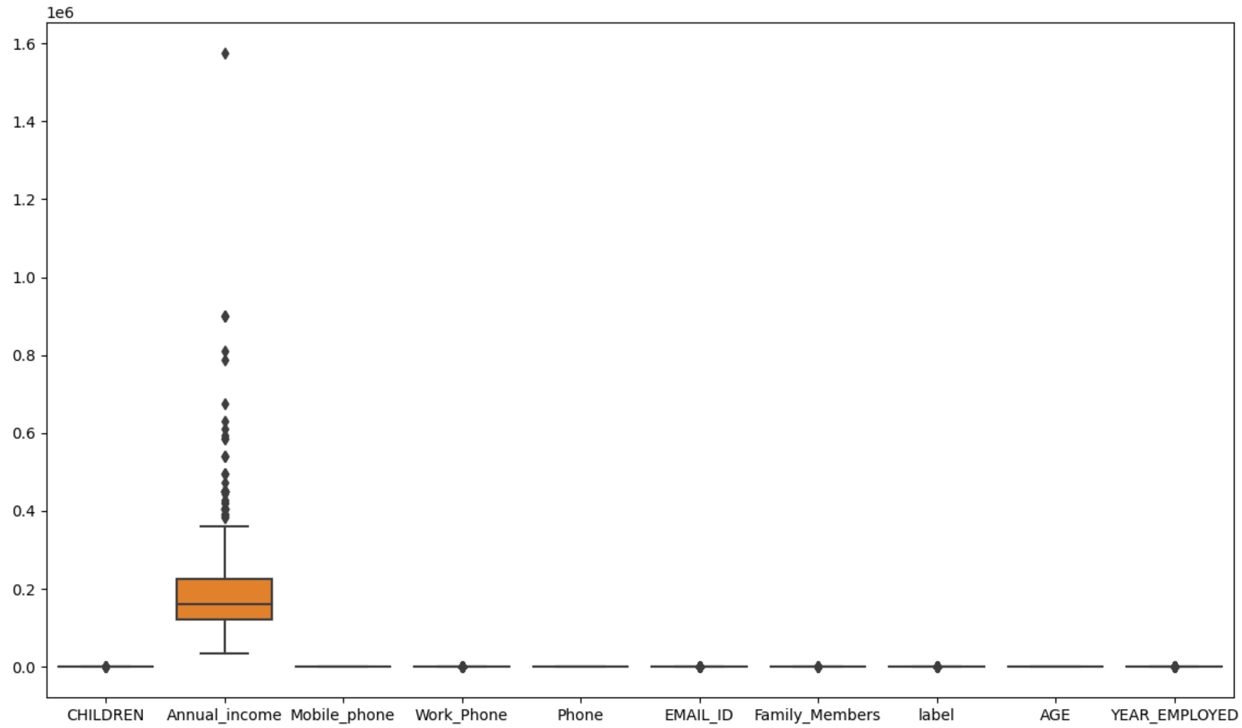
For the modelling the data, every feature should be numerical if the feature is category type ,we have to convert into numerical by using map or label_encoding or dummies method and ensure thar data should not have any outliers.

Next step is dealing with outliers

Outliers are values that far extreme from the mean zone and if they are present that will lead to change the entire mean or median, so we need to remove those values otherwise those are affect the other necessary values.

we can identify theoutliers using boxplot and we can remove the outliers with IQR (interquartile range) method

```
## boxplot to find out the outliers
plt.figure(figsize=(14,8))
sns.boxplot(df)

<Axes: >
```



OBSERVARTIONS:

It is evident that there are more number of outliers(extreme values) present in the annual_income column and Anhual_income is one of the most affecting parameter for the predicting the creditcard approval. So, we have to deal this issue and have to remove the outliers before proceeding into modelling

```
df['Annual_income'].describe()    # describe function gives the stastical summery of given data

count    1.386000e+03
mean     1.890221e+05
std      1.060995e+05
min      3.375000e+04
25%      1.215000e+05
50%      1.620000e+05
75%      2.250000e+05
max      1.575000e+06
Name: Annual_income, dtype: float64
```

Outliers are present in the Annual_income feature so we are removing those with the help of IQR method

```
q3= 2.250000e+05     # 75% is consideras q3
q1= 1.215000e+05     # 25% is consider as q1
```

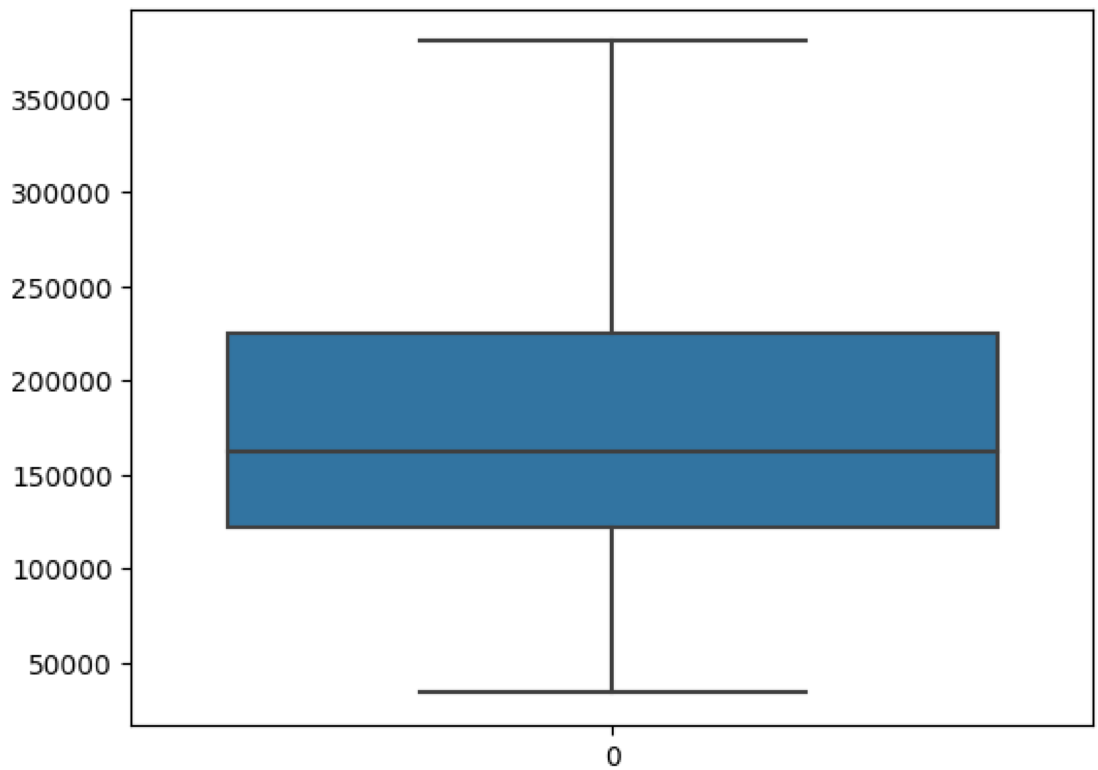
```
iqr=q3-q1            # IQR is difference between q3 and q1
uc=q3+1.5*iqr        # upper boundery for this data
uc
```

```
380250.0
```

```
df['Annual_income']=np.where(df['Annual_income']>uc,uc,df['Annual_income'])   # above upper boundary data we are fitting into at the uc level
```

```
sns.boxplot(df['Annual_income'])           # after replacing the outliers values to upper boundary
```

```
<Axes: >
```



With IQR method, we can achieve that there is no outliers present in the Annual_Income feature and now the data is clean and so we can go head with modelling section

```
df['Annual_income'].describe()
```

```
count      1386.000000
mean     183116.250000
std       81610.338803
min       33750.000000
```

```
25%       121500.000000
50%       162000.000000
75%       225000.000000
max       380250.000000
Name: Annual_income, dtype: float64
```

We are done with data cleaning steps that is now the data is clean without having the outliers also

next step is feature enginnering

converting categorical into numerical

```
df.info()      # to find the category (object) columns
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1386 entries, 0 to 1547
Data columns (total 18 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   GENDER           1386 non-null   object
 1   Car_Owner        1386 non-null   object
 2   Propert_Owner    1386 non-null   object
 3   CHILDREN         1386 non-null   int64
 4   Annual_income    1386 non-null   float64
 5   Type_Income      1386 non-null   object
 6   EDUCATION        1386 non-null   object
 7   Marital_status   1386 non-null   object
 8   Housing_type     1386 non-null   object
 9   Mobile_phone     1386 non-null   int64
 10  Work_Phone       1386 non-null   int64
 11  Phone            1386 non-null   int64
 12  EMAIL_ID         1386 non-null   int64
 13  Type_Occupation  1386 non-null   object
 14  Family_Members   1386 non-null   int64
 15  label            1386 non-null   int64
 16  AGE              1386 non-null   int64
 17  YEAR_EMPLOYED    1386 non-null   float64
dtypes: float64(2), int64(8), object(8)
memory usage: 238.0+ KB
```

```
catg_data=df.select_dtypes(['object']).columns
catg_data
```

```
Index(['GENDER', 'Car_Owner', 'Propert_Owner', 'Type_Income',
'EDUCATION',
```

```
       'Marital_status', 'Housing_type', 'Type_Occupation'],
      dtype='object')
```

```
df.GENDER.unique()
```

```
array(['M', 'F'], dtype=object)
```

GENDER : ['M' 'F'] Car_Owner : ['Y' 'N'] Propert_Owner : ['Y' 'N'] Type_Income : ['Pensioner' 'Commercial associate' 'Working' 'State servant'] EDUCATION : ['Higher education' 'Secondary / secondary special' 'Lower secondary' 'Incomplete higher' 'Academic degree'] Marital_status : ['Married' 'Single / not married' 'Civil marriage' 'Separated' 'Widow'] Housing_type : ['House / apartment' 'With parents' 'Rented apartment' 'Municipal apartment' 'Co-op apartment' 'Office apartment']

To find out howmany different categories are present in each column

1. if the categories are 2 then we can go with map method

2. else (>2) we go with other methods called label encoder or one hot encoding

```
# converting gender, car_owner,property_owner columns into numerical
type using map function ie.., 2 sub catgories

df['GENDER']=df['GENDER'].map({'F':0,'M':1})
df['Car_Owner']=df['Car_Owner'].map({'N':0,'Y':1})
df['Propert_Owner']=df['Propert_Owner'].map({'N':0,'Y':1})
```

using label encoder method, converting remaining category columns into numeric type more than 2 categories

```
from sklearn.preprocessing import LabelEncoder                        #
importing the labelencoder method from sklearn library
LE=LabelEncoder()
df['Marital_status']=LE.fit_transform(df['Marital_status'])           #
fiting the marital_status column into numeric
df['Type_Income']=LE.fit_transform(df['Type_Income'])
df['EDUCATION']=LE.fit_transform(df['EDUCATION'])
df['Housing_type']=LE.fit_transform(df['Housing_type'])
```

All the category features are converted into numerical except Type_occupation

As type_occupations has more number of null values so we will drop that column

```
df.head()

   GENDER  Car_Owner  Propert_Owner  CHILDREN  Annual_income
Type_Income  \
0       1          1              1         0       180000.0
1
1       0          1              0         0       315000.0
0
2       0          1              0         0       315000.0
0
3       0          1              0         0       162000.0
0
5       0          1              0         0       315000.0
1

   EDUCATION  Marital_status  Housing_type  Mobile_phone  Work_Phone
Phone  \
0          1               1             1             1           0
0
1          1               1             1             1           1
1
2          1               1             1             1           1
1
3          1               1             1             1           1
1
5          1               1             1             1           1
1

   EMAIL_ID Type_Occupation  Family_Members  label  AGE  YEAR_EMPLOYED

0         0        Laborers               2      1   51            0.0

1         0        Laborers               2      1   37            2.0

2         0        Laborers               2      1   42            2.0

3         0        Laborers               2      1   37            2.0

5         0        Laborers               2      1   37            2.0
```

Dropping type_occupation column

```
df.drop('Type_Occupation',axis=1)
df.head(2)
```

```
   GENDER  Car_Owner  Propert_Owner  CHILDREN  Annual_income
Type_Income  \
0       1          1              1         0       180000.0
1
1       0          1              0         0       315000.0
0

   EDUCATION  Marital_status  Housing_type  Mobile_phone  Work_Phone
Phone  \
0          1               1             1             1           0
0
1          1               1             1             1           1
1

   EMAIL_ID Type_Occupation  Family_Members  label  AGE  YEAR_EMPLOYED

0         0        Laborers               2      1   51            0.0

1         0        Laborers               2      1   37            2.0
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1386 entries, 0 to 1547
Data columns (total 18 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   GENDER           1386 non-null   int64
 1   Car_Owner        1386 non-null   int64
 2   Propert_Owner    1386 non-null   int64
 3   CHILDREN         1386 non-null   int64
 4   Annual_income    1386 non-null   float64
 5   Type_Income      1386 non-null   int64
 6   EDUCATION        1386 non-null   int64
 7   Marital_status   1386 non-null   int64
 8   Housing_type     1386 non-null   int64
 9   Mobile_phone     1386 non-null   int64
 10  Work_Phone       1386 non-null   int64
 11  Phone            1386 non-null   int64
 12  EMAIL_ID         1386 non-null   int64
 13  Type_Occupation  1386 non-null   object
 14  Family_Members   1386 non-null   int64
 15  label            1386 non-null   int64
 16  AGE              1386 non-null   int64
 17  YEAR_EMPLOYED    1386 non-null   float64
dtypes: float64(2), int64(15), object(1)
memory usage: 238.0+ KB
```

```
# dropping unnecessary columns like
Mobile_phone,phone,work_phone,mail_id columns
df.drop(['Mobile_phone','Work_Phone','Phone','EMAIL_ID'],axis=1)
df.head(2)
```

```
   GENDER  Car_Owner  Propert_Owner  CHILDREN  Annual_income
Type_Income  \
0       1          1              1         0       180000.0
1
1       0          1              0         0       315000.0
0

   EDUCATION  Marital_status  Housing_type  Mobile_phone  Work_Phone
Phone  \
0          1               1             1             1           0
0
1          1               1             1             1           1
1

   EMAIL_ID Type_Occupation  Family_Members  label  AGE  YEAR_EMPLOYED

0         0        Laborers               2      1   51            0.0

1         0        Laborers               2      1   37            2.0
```

Feature enginnering:

Selecting the best features for the data modelling training part

the selection of features done with two different ways 1. selecting most affecting features and dropping the others for approval of credit card 2. Selecting all the least affecting features without dropping them

Compare those above two

```
df.drop(['Car_Owner','Propert_Owner','Type_Income','EDUCATION',
         'Marital_status','Housing_type'],axis=1)
```

```
      GENDER  CHILDREN  Annual_income  Mobile_phone  Work_Phone  Phone
\
0          1         0       180000.0             1           0      0

1          0         0       315000.0             1           1      1

2          0         0       315000.0             1           1      1

3          0         0       162000.0             1           1      1

5          0         0       315000.0             1           1      1
```

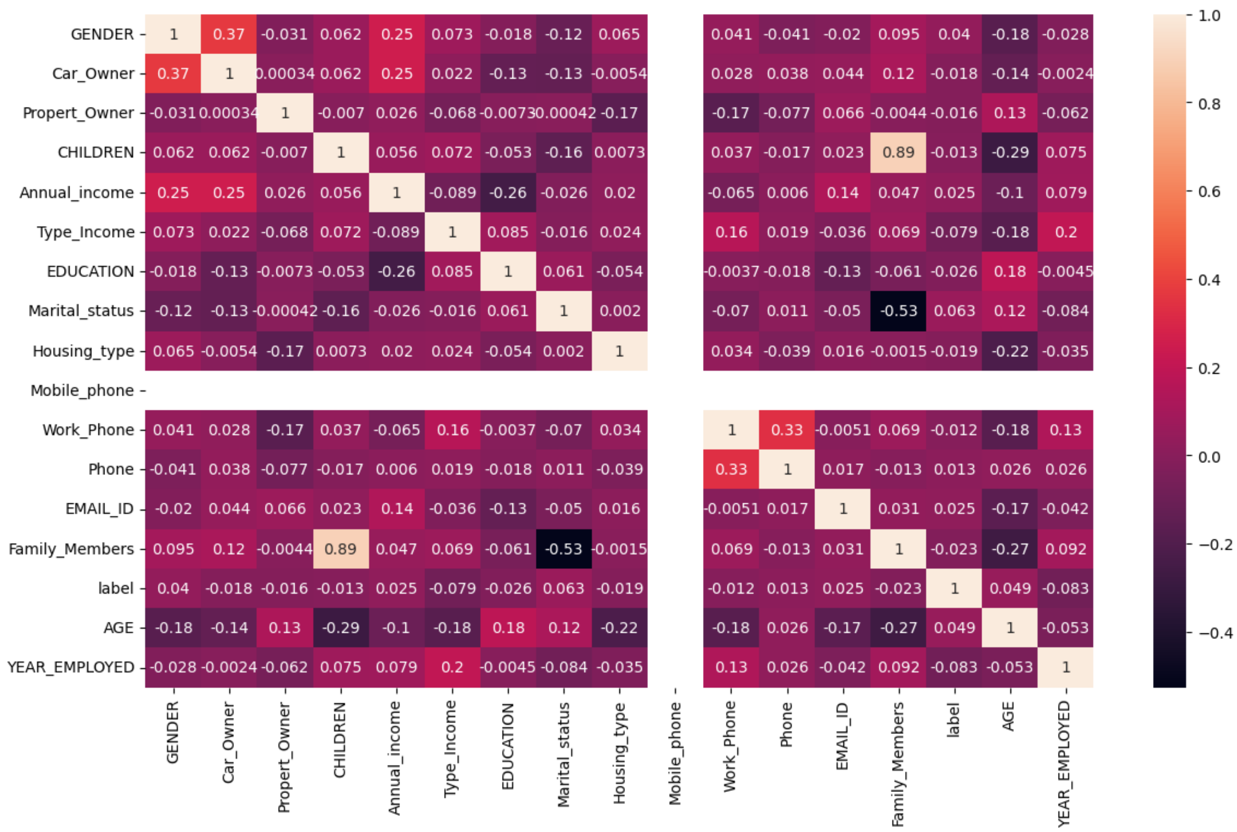
```
...       ...        ...              ...           ...          ...    ...
1542        1          1        360000.0              1            0      1
1543        0          0        162000.0              1            0      0
1544        0          0        225000.0              1            0      0
1546        1          0        270000.0              1            1      1
1547        0          0        225000.0              1            0      0

      EMAIL_ID Type_Occupation  Family_Members  label  AGE
YEAR_EMPLOYED
0            0        Laborers               2      1   51
0.0
1            0        Laborers               2      1   37
2.0
2            0        Laborers               2      1   42
2.0
3            0        Laborers               2      1   37
2.0
5            0        Laborers               2      1   37
2.0
...        ...             ...             ...    ...  ...
...
1542         0         Drivers               3      0   30
10.0
1543         0        Managers               2      0   32
6.0
1544         0     Accountants               1      0   28
4.0
1546         0         Drivers               2      0   41
2.0
1547         0        Laborers               2      0   45
8.0

[1386 rows x 12 columns]
```

```
df.drop(['Mobile_phone','Work_Phone','Phone','EMAIL_ID','Type_Occupati
on'],axis=1)
```

```
      GENDER  Car_Owner  Propert_Owner  CHILDREN  Annual_income
Type_Income  \
0          1          1              1         0       180000.0
1
1          0          1              0         0       315000.0
0
2          0          1              0         0       315000.0
```

```
0
3           0          1              0         0       162000.0
0
5           0          1              0         0       315000.0
1
...       ...        ...            ...       ...            ...
...
1542        1          1              0         1       360000.0
2
1543        0          0              1         0       162000.0
0
1544        0          0              0         0       225000.0
0
1546        1          1              0         0       270000.0
3
1547        0          1              1         0       225000.0
3

      EDUCATION  Marital_status  Housing_type  Family_Members  label
AGE  \
0             1               1             1               2      1
51
1             1               1             1               2      1
37
2             1               1             1               2      1
42
3             1               1             1               2      1
37
5             1               1             1               2      1
37
...         ...             ...           ...             ...    ...
...
1542          4               1             1               3      0
30
1543          1               1             1               2      0
32
1544          2               3             1               1      0
28
1546          4               0             1               2      0
41
1547          1               1             1               2      0
45

      YEAR_EMPLOYED
0               0.0
1               2.0
2               2.0
3               2.0
5               2.0
```

```
...                ...
1542              10.0
1543               6.0
1544               4.0
1546               2.0
1547               8.0

[1386 rows x 13 columns]

KC=df.corr()
plt.figure(figsize=(14,8))
sns.heatmap(KC,annot=True)

<ipython-input-246-be6108c453b8>:1: FutureWarning: The default value
of numeric_only in DataFrame.corr is deprecated. In a future version,
it will default to False. Select only valid columns or specify the
value of numeric_only to silence this warning.
  KC=df.corr()

<Axes: >
```



The plot of number of family member and number of children and correlation table confirm the correlation. As the number of family member cover the number of children, we chose to drop the number of children feature.

```
df.drop(['CHILDREN'],axis=1)
```

```
      GENDER  Car_Owner  Propert_Owner  Annual_income  Type_Income  
EDUCATION  \
0          1          1              1       180000.0            1   
1
1          0          1              0       315000.0            0   
1
2          0          1              0       315000.0            0   
1
3          0          1              0       162000.0            0   
1
5          0          1              0       315000.0            1   
1
...      ...        ...            ...            ...          ...   
...
1542       1          1              0       360000.0            2   
4
1543       0          0              1       162000.0            0   
1
1544       0          0              0       225000.0            0   
2
1546       1          1              0       270000.0            3   
4
1547       0          1              1       225000.0            3   
1

      Marital_status  Housing_type  Mobile_phone  Work_Phone  Phone  
EMAIL_ID  \
0                  1             1             1           0      0   
0
1                  1             1             1           1      1   
0
2                  1             1             1           1      1   
0
3                  1             1             1           1      1   
0
5                  1             1             1           1      1   
0
...              ...           ...           ...         ...    ...   
...
1542               1             1             1           0      1   
0
1543               1             1             1           0      0   
0
1544               3             1             1           0      0   
0
1546               0             1             1           1      1   
0
1547               1             1             1           0      0   
```

```
0

      Type_Occupation  Family_Members  label  AGE  YEAR_EMPLOYED
0            Laborers               2      1   51            0.0
1            Laborers               2      1   37            2.0
2            Laborers               2      1   42            2.0
3            Laborers               2      1   37            2.0
5            Laborers               2      1   37            2.0
...               ...             ...    ...  ...            ...
1542          Drivers               3      0   30           10.0
1543         Managers               2      0   32            6.0
1544      Accountants               1      0   28            4.0
1546          Drivers               2      0   41            2.0
1547         Laborers               2      0   45            8.0

[1386 rows x 17 columns]
```

1. selecting most affecting features and dropping the others for approval of credit card

```
tdf=df.drop(['Car_Owner', 'Propert_Owner', 'CHILDREN',
        'Type_Income', 'EDUCATION', 'Marital_status',
       'Housing_type',  'Mobile_phone',
       'Work_Phone', 'Phone', 'EMAIL_ID', 'Type_Occupation',
'Family_Members',
       ],axis=1)
tdf.head(2)

   GENDER  Annual_income  label  AGE  YEAR_EMPLOYED
0       1       180000.0      1   51            0.0
1       0       315000.0      1   37            2.0
```

Feature Scaling

```
X=tdf.drop('label',axis=1)                              # Split the data into
features (X) and target variable (Y)
Y=tdf.label

from sklearn.preprocessing import StandardScaler   # converting all
the features into same scale using Standardscalar method by importing
standardscalar
sc=StandardScaler()
X=sc.fit_transform(X)

X=pd.DataFrame(data=X,columns=['GENDER','Annual_income','AGE','YEAR_EM
PLOYED'])  # converting into dataframe
X.head()

     GENDER  Annual_income       AGE  YEAR_EMPLOYED
0  1.316711      -0.038198  0.663800      -0.971845
```

```
1 -0.759468       1.616601 -0.558131      -0.669120
2 -0.759468       1.616601 -0.121727      -0.669120
3 -0.759468      -0.258838 -0.558131      -0.669120
4 -0.759468       1.616601 -0.558131      -0.669120
```

```
from sklearn.model_selection import train_test_split,GridSearchCV
# using traintestsplit we can split the entire data into training and
testing
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
# importing LogisticRegression model
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.2,rando
m_state=42)
```

A model can be train and test with many methods and Here the target variable is categorical so we can use any classification methods

1.Logistic Regression

```
 # using logistic regression model
lr=LogisticRegression()

# train the model with training data
lr.fit(X_train,Y_train)

# printing the training accuracy score how accurately the model is
trained
print(lr.score(X_train,Y_train))

#predicting the unknown testing data with trained model
pred=lr.predict(X_test)

#printing the accuracy_score that how the testing data is correct with
actual results
print(accuracy_score(Y_test,pred))

0.9007220216606499
0.9028776978417267
```

1.By the above result we can say that both accuracies are similar.

2.therefore we can conclude that this model is perfect and predicting the unknown data well

3.Both trained accuracy and predicted accuracy are same so that no overfitting happens

Hyper parameter tuning for better results

```
param_grid = {
    'penalty': ['l1', 'l2'],             # Regularization type
    'C': [0.001, 0.01, 0.1, 1, 10],     # Inverse of regularization
strength
    'solver': ['liblinear', 'saga'],    # Algorithm to use in
optimization
    'max_iter': [100, 200, 300],        # Maximum number of iterations
}

# Create a grid search object with cross-validation
grid_search = GridSearchCV(lr, param_grid, cv=5, scoring='accuracy')

# Fit the grid search to the training data
grid_search.fit(X_train, Y_train)



# Get the best hyperparameters from the grid search
best_params = grid_search.best_params_


0.9007220216606499
Best Hyperparameters: {'C': 0.001, 'max_iter': 100, 'penalty': 'l1',
'solver': 'liblinear'}
Test Accuracy: 0.9028776978417267
```

Train the model with Hyper parameter results

```
# Use the best hyperparameters to create the final logistic regression
model
best_logistic_reg = LogisticRegression(**best_params)

# Fit the final model on the training data
best_logistic_reg.fit(X_train, Y_train)

print(best_logistic_reg.score(X_train,Y_train))

# Make predictions on the test data
y_pred = best_logistic_reg.predict(X_test)

# Calculate accuracy on the test data
accuracy = accuracy_score(Y_test, y_pred)

print("Best Hyperparameters:", best_params)
print("Test Accuracy:", accuracy)

0.9007220216606499
Best Hyperparameters: {'C': 0.001, 'max_iter': 100, 'penalty': 'l1',
'solver': 'liblinear'}
Test Accuracy: 0.9028776978417267
```

It is evident that the model is performing well even with hyper parameter tuning

without performing the hyper parameter tuning also the model is performing good

so, The logisticRegression model is well fitted for this data

1. Kneighbours Classification:

```
# importing Kneighboursclassifier model
from sklearn.neighbors import KNeighborsClassifier

# using KneighboursClassifier model
knc1=KNeighborsClassifier()

# train the model with training data
knc1.fit(X_train,Y_train)

# printing the training accuracy score how accurately the model is
trained
print(knc1.score(X_train,Y_train))

#predicting the unknown testing data with trained model
pred=knc1.predict(X_test)

#printing the accuracy_score that how the testing data is correct with
actual results
print(accuracy_score(Y_test,pred))

0.9061371841155235
0.8920863309352518
```

1.Both training and testing accuracies are almost similar

2.this model is also giving best results

3. there is no or very minute chance of overfitting

1. we will change the k number and checking the accuracy scores

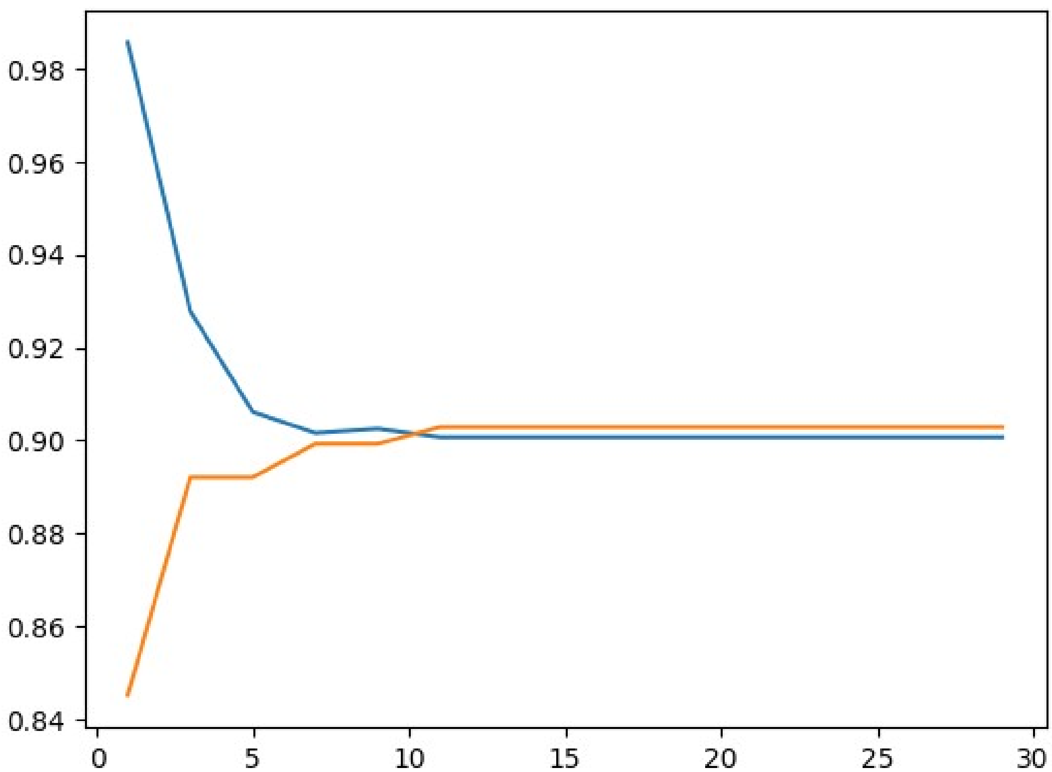
```
scores=[]
a_scores=[]
k_range=range(1,31,2)
for i in k_range :
  knn=KNeighborsClassifier(n_neighbors=i)
  knn.fit(X_train,Y_train)
  scores.append(knn.score(X_train,Y_train))
```

```
  pred=knn.predict(X_test)
  a_scores.append(accuracy_score(Y_test,pred))
print(scores)
print(a_scores)
```

```
[0.9855595667870036, 0.927797833935018, 0.9061371841155235,
0.9016245487364621, 0.9025270758122743, 0.9007220216606499,
0.9007220216606499, 0.9007220216606499, 0.9007220216606499,
0.9007220216606499, 0.9007220216606499, 0.9007220216606499,
0.9007220216606499, 0.9007220216606499, 0.9007220216606499]
[0.8453237410071942, 0.8920863309352518, 0.8920863309352518,
0.8992805755395683, 0.8992805755395683, 0.9028776978417267,
0.9028776978417267, 0.9028776978417267, 0.9028776978417267,
0.9028776978417267, 0.9028776978417267, 0.9028776978417267,
0.9028776978417267, 0.9028776978417267, 0.9028776978417267]
```

```
plt.plot(k_range,scores)
plt.plot(k_range,a_scores)
```

```
[<matplotlib.lines.Line2D at 0x78d402695300>]
```



when Kvalue >= 11, the testing accuracy is more than the training accuracy

And,Therfore no chance of overfitting

3.Random Forest classifier (as Decision Tree can only perform one certain route but Random Forest considered all outcomes so i am applying Random Forest here)

```
# importing the RandomForestClassifier Model
from sklearn.ensemble import RandomForestClassifier

# use the RandomForest Classifer model
RF=RandomForestClassifier()

# train the model with training data
RF.fit(X_train,Y_train)

# printing the training accuracy score how accurately the model is trained
print(RF.score(X_train,Y_train))

#predicting the unknown testing data with trained model
pred=RF.predict(X_test)

#printing the accuracy_score that how the testing data is correct with actual results
print(accuracy_score(Y_test,pred))
```

```
0.9873646209386282
0.8884892086330936
```

From the results we can clearly observe that the training accuracy is more compared to testing accuracy. So, it is evident that overfitting is happened. So we will do cross validation and then hyperparameter tuning to get batter results

```
from sklearn.model_selection import cross_val_score
cvscore=cross_val_score(RF,X,Y,cv=5)
cvscore.mean()
```

```
0.8744539386541309
```

By doing the cross validation, the accurcacy is not similar with the training accuracy in the model.

so in order to get best results we will go for hyper parameter tuning

Hyper parameter tuning for Random Forest model

```python
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    # Add more hyperparameters as needed
}
grid_search = GridSearchCV(estimator=RandomForestClassifier(),
param_grid=param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, Y_train)

best_params = grid_search.best_params_
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)

accuracy = accuracy_score(Y_test, y_pred)


print(f"Best Hyperparameters: {best_params}")
print(f"Accuracy: {accuracy}")
```

```
Best Hyperparameters: {'max_depth': 10, 'min_samples_split': 5,
'n_estimators': 300}
Accuracy: 0.8992805755395683
```

With hyperparameter Tuning we get some best results but these are not so accurate comapred to other methods

1. Boosting:

```python
# importing XGBOOST model
import xgboost as xgb

# use xgboost model
model=xgb.XGBClassifier(objective='binary:logistic',n_estimators=3)

# # train the model with training data
model.fit(X_train,Y_train)

# printing the training accuracy score how accurately the model is
trained
print(model.score(X_train,Y_train))

#predicting the unknown testing data with trained model
pred=model.predict(X_test)

#printing the accuracy_score that how the testing data is correct with
actual results
print(accuracy_score(Y_test,pred))
```

```
0.907942238267148
0.89568345323741
```

The both accuracies are very similar so with this model also we can test the unknown data effectively

```
param_grid = {
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 4, 5],
    'n_estimators': [100, 200, 300],
    'min_child_weight': [1, 2, 3],
    'gamma': [0, 0.1, 0.2],

}

grid_search = GridSearchCV(estimator=xgb.XGBClassifier(),
param_grid=param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, Y_train)
best_params = grid_search.best_params_
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)
accuracy = accuracy_score(Y_test, y_pred)
print(f"Best Hyperparameters: {best_params}")
print(f"Accuracy: {accuracy}")

Best Hyperparameters: {'gamma': 0, 'learning_rate': 0.1, 'max_depth':
4, 'min_child_weight': 1, 'n_estimators': 200}
Accuracy: 0.89568345323741
```

With the best parameters also we aregetting same accuracy so, this model is predicting good results withoout haperparameter tuning also

1.We have tested and trained the model with all different types models

2. Except RandomForest Every model is giving better results

3. Comparitively Logistic Regression model is giving very good results than Knieghbours and Xgboosting models

Finally concluded that LogisticRegressor is best fitted model for this case

2. Selecting all the least affecting features without dropping them

```
tdf2=df.drop(['Work_Phone','Marital_status','Mobile_phone','Phone',
'EMAIL_ID','Type_Occupation','Family_Members'],axis=1)

tdf2.head()

   GENDER  Car_Owner  Propert_Owner  CHILDREN  Annual_income
Type_Income  \
```

```
0       1          1              1         0       180000.0
1
1       0          1              0         0       315000.0
0
2       0          1              0         0       315000.0
0
3       0          1              0         0       162000.0
0
5       0          1              0         0       315000.0
1

   EDUCATION  Housing_type  label  AGE  YEAR_EMPLOYED
0          1             1      1   51            0.0
1          1             1      1   37            2.0
2          1             1      1   42            2.0
3          1             1      1   37            2.0
5          1             1      1   37            2.0
```

```
X1=tdf2.drop('label',axis=1)
Y1=tdf2.label
```

```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
X1=sc.fit_transform(X1)
```

```
X1=pd.DataFrame(data=df,columns=['GENDER','Car_Owner','Propert_Owner',
'CHILDREN','Annual_income','Type_Income','Housing_type','AGE','YEAR_EM
PLOYED'])
X1.head()
```

```
   GENDER  Car_Owner  Propert_Owner  CHILDREN  Annual_income
Type_Income  \
0       1          1              1         0       180000.0
1
1       0          1              0         0       315000.0
0
2       0          1              0         0       315000.0
0
3       0          1              0         0       162000.0
0
5       0          1              0         0       315000.0
1

   Housing_type  AGE  YEAR_EMPLOYED
0             1   51            0.0
1             1   37            2.0
2             1   42            2.0
3             1   37            2.0
5             1   37            2.0
```

```
from sklearn.model_selection import train_test_split,GridSearchCV
X1_train,X1_test,Y1_train,Y1_test=train_test_split(X1,Y1,test_size=0.2
,random_state=42)
```

1. Linear Regression Model

```
 # using logistic regression model
lr1=LogisticRegression()

# train the model with training data
lr1.fit(X_train,Y_train)

# printing the training accuracy score how accurately the model is
trained
print(lr1.score(X_train,Y_train))

#predicting the unknown testing data with trained model
pred=lr1.predict(X_test)

#printing the accuracy_score that how the testing data is correct with
actual results
print(accuracy_score(Y_test,pred))
```

```
0.9007220216606499
0.9028776978417267
```

CONCLUSIONS:

It is evident that even with other features also we are getting same accuracy what we are getting without unnecessary columns

Therefore it is concluded that LogisticRegression model with most affected columns or features is the method is giving best results

Live_project : SQL part

Here we have used claened csv file and that was used in mysql workbench and i have done work in the MYSQL only therefore i am inserting the images of sql quaries and outcomes with this file

1.Group the customers based on their income type and find the average of their annual income.

```
from google.colab import files
from IPython.display import Image

uploaded=files.upload()
```

```
<IPython.core.display.HTML object>
```

```
Saving Screenshot 2023-09-26 124125.png to Screenshot 2023-09-26
124125.png
```

```
Image('Screenshot 2023-09-26 124125.png',width=1000)
```

```sql
select * from new.cleaned;

select type_income,avg(annual_income) as average from new.cleaned
group by type_income
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

type_income	average
Pensioner	155343.4981
Commercial associate	233107.3973
Working	180848.2105
State servant	211422.4138

1. i have downloaded the cleaned excel from this data and uploaded the csv file into MYSQL by creating the new database and with table called new.cleaned and so i am using the same table here

2. I have called type_income and thier averages by grouping them with the help of Group by

3.The average of Commercial associates are heigher comapared to others then we have State Servents

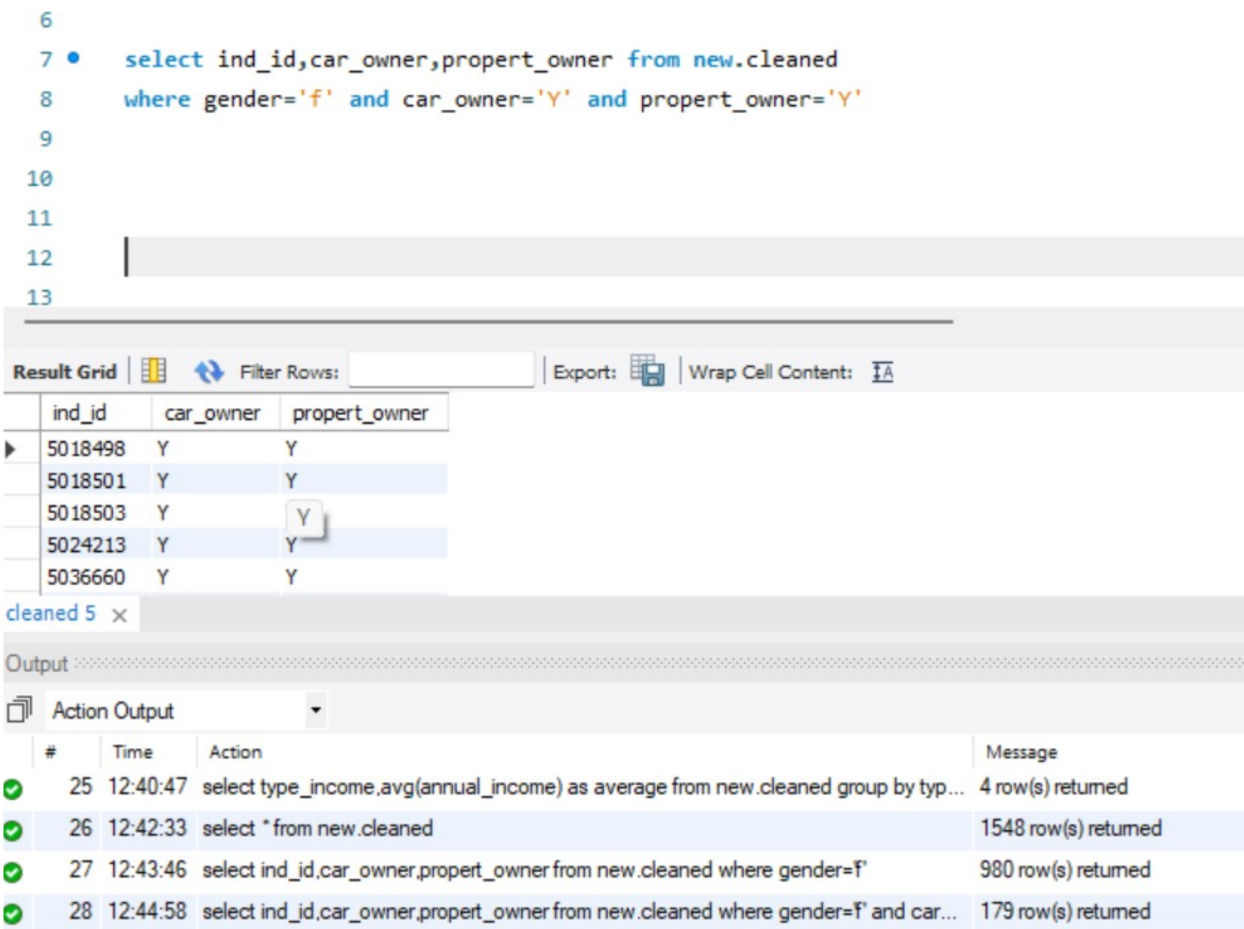
2.Find the female owners of cars and property.

```python
uploaded=files.upload()
```

```
<IPython.core.display.HTML object>
```

```
Saving Screenshot 2023-09-26 124541.png to Screenshot 2023-09-26 124541.png
```

```python
Image('Screenshot 2023-09-26 124541.png',width=1000)
```



They asked to find out list of female persons having both car and property

The result table describes the Ind_id of each femae persons having car and property

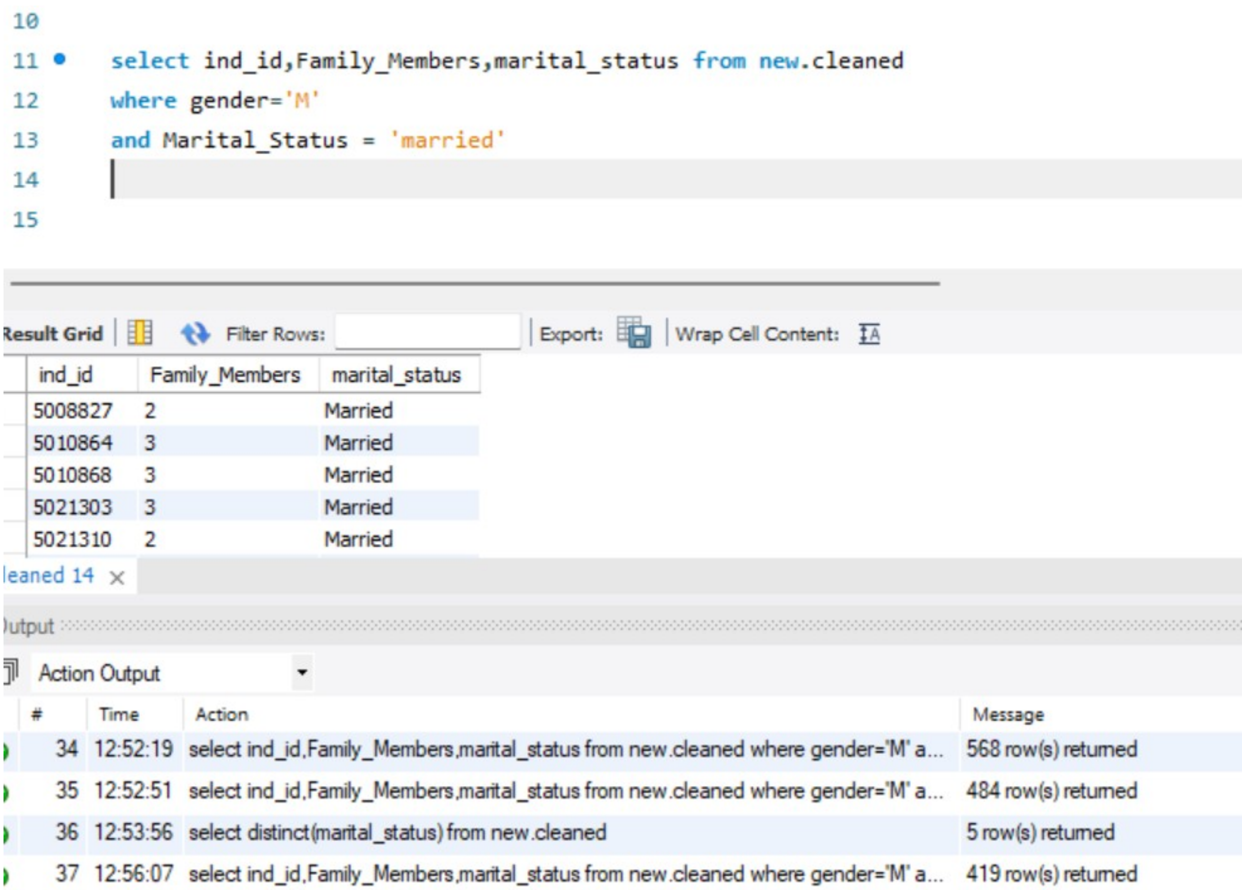
3.Find the male customers who are staying with their families.

```
uploaded=files.upload()
```

```
<IPython.core.display.HTML object>
```

```
Saving Screenshot 2023-09-26 125648.png to Screenshot 2023-09-26 125648.png
```

```
Image('Screenshot 2023-09-26 125648.png',width=1000)
```



```sql
select ind_id,Family_Members,marital_status from new.cleaned
where gender='M'
and Marital_Status = 'married'
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

ind_id	Family_Members	marital_status
5008827	2	Married
5010864	3	Married
5010868	3	Married
5021303	3	Married
5021310	2	Married

leaned 14 ×

utput

Action Output

#	Time	Action	Message
34	12:52:19	select ind_id,Family_Members,marital_status from new.cleaned where gender='M' a...	568 row(s) returned
35	12:52:51	select ind_id,Family_Members,marital_status from new.cleaned where gender='M' a...	484 row(s) returned
36	12:53:56	select distinct(marital_status) from new.cleaned	5 row(s) returned
37	12:56:07	select ind_id,Family_Members,marital_status from new.cleaned where gender='M' a...	419 row(s) returned

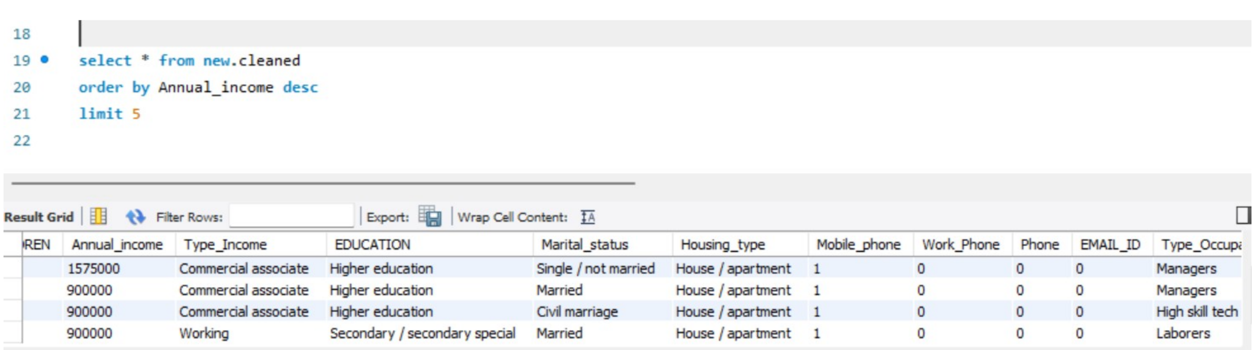
4. Please list the top five people having the highest income.

```
uploaded=files.upload()
```

```
<IPython.core.display.HTML object>
```

```
Saving Screenshot 2023-09-26 130018.png to Screenshot 2023-09-26 130018.png
```

```
Image('Screenshot 2023-09-26 130018.png',width=1000)
```



```sql
select * from new.cleaned
order by Annual_income desc
limit 5
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

REN	Annual_income	Type_Income	EDUCATION	Marital_status	Housing_type	Mobile_phone	Work_Phone	Phone	EMAIL_ID	Type_Occupa
	1575000	Commercial associate	Higher education	Single / not married	House / apartment	1	0	0	0	Managers
	900000	Commercial associate	Higher education	Married	House / apartment	1	0	0	0	Managers
	900000	Commercial associate	Higher education	Civil marriage	House / apartment	1	0	0	0	High skill tech
	900000	Working	Secondary / secondary special	Married	House / apartment	1	0	0	0	Laborers

The result table tell us that the top 4 are working as commercial associates and these are highly paid

5.How many married people are having bad credit?

```
uploaded=files.upload()
```

```
<IPython.core.display.HTML object>
```

```
Saving Screenshot 2023-09-26 132152.png to Screenshot 2023-09-26 132152.png
```

```
Image('Screenshot 2023-09-26 132152.png',width=1000)
```

```sql
select count(*) as bad_customers from new.cleaned
where marital_status ='Married' and label=1;
```

bad_customers
114

the result tells us that there are 114 bad customers in our data and here label=1 represents credit card not approved (rejected) persons we call it as bad customers

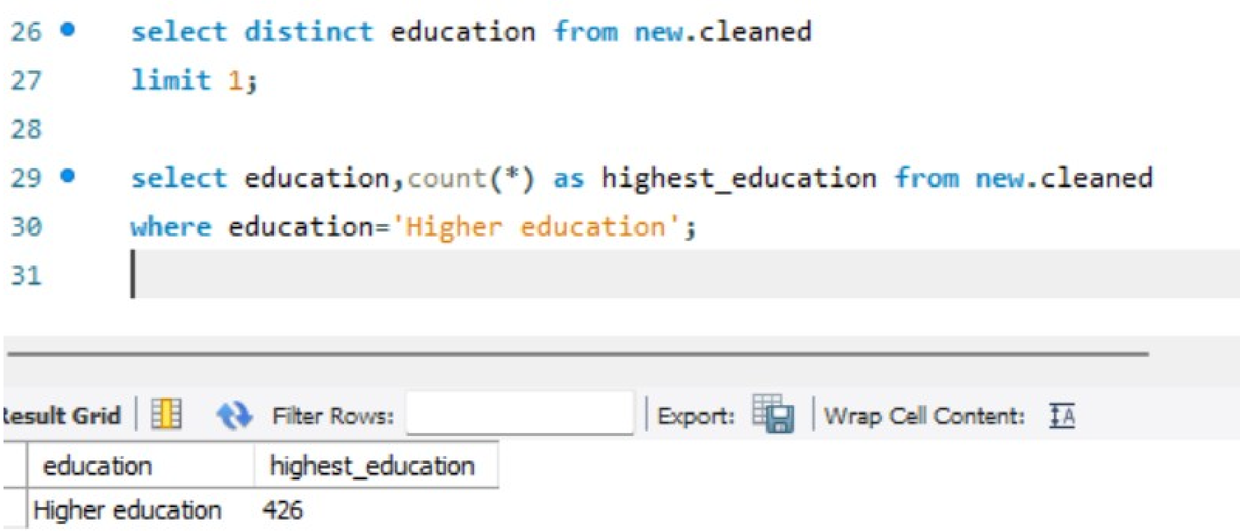
6.What is the highest education level and what is the total count?

```
uploaded=files.upload()
```

```
<IPython.core.display.HTML object>
```

```
Saving Screenshot 2023-09-26 131702.png to Screenshot 2023-09-26 131702.png
```

```
Image('Screenshot 2023-09-26 131702.png',width=1000)
```



Highest education is Higher Education and the count is 426 persons

7.Between married males and females, who is having more bad credit?

```
uploaded=files.upload()
```

```
<IPython.core.display.HTML object>
```

```
Saving Screenshot 2023-09-26 133132.png to Screenshot 2023-09-26
133132.png
```

```
Image('Screenshot 2023-09-26 133132.png',width=1000)
```

```sql
select gender,count(*) as bad_customers from new.cleaned
where label=1 and marital_status='married'
group by gender
order by count(*) desc
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

gender	bad_customers
F	63
M	51

There are more female customers as bad customers thann male