Open in Colab

Introduction

A credit card is a convinient tool that allows you to buy items now and pay for them later. Credit card is a physical payment card that allows you to get credit from a financial institution. If you buy something with credit, you are in debt. This means you owe money to the company that gave you the credit card. If you don't pay the entire amount at the end of each month, you pay a fee for the credit card called interest. .If managed correctly, credit cards can be great way to build credit and manage your money.

Importance in Today's world:

Credit score cards are a common risk control method in the financial history. It uses personal information and data submitted by credit card applicants to predict the probability of future defaults and credit card borrowings. The bank is able to decide whether to issue a credit card to the applicant. Credit scores can objectively quantify the magnitude of risk. Credit score is a number that depicts a consumer's credit worthiness.

Importance of Predicting a good client:

Credit risk as the board in banks basically centers around deciding the the probability of customer's default or credit decay and how expensice it will end up being assuming it happens. It is important to consider major factors and predict beforehand the probability of consumers defaulting given their conditions, which is where a machine learning model comes in handy and allows the bank and major finantial institutions to predict whether the customer will defaul or not. This project builds a machine learning model with the best accuracy possible.

Impact on Banking Sector:

Banks receive a lot of credit card applications. Many of the applications do not get approved for a variety of reasons, like increased loan balances or poor-income levels. Manually analysing these applications can be very time consuming and full of human errors. Hence we can automate this task with the help of machine learning.

Import Libraries

```
import pandas as pd
import numpy as np


import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px

from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import StandardScaler


from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error, r2_score
```

Importing Data set

```
a=pd.read_csv('Credit_card.csv')
aa=a
b=pd.read_csv('Credit_card_label.csv')
bb=b
```

```
aa
```

	Ind_ID	GENDER	Car_Owner	Propert_Owner	CHILDREN	Annual_income	Type_Income	EDUCATION	Marital_status	Housing_type	Birthday_
0	5008827	M	Y	Y	0	180000.0	Pensioner	Higher education	Married	House / apartment	-1
1	5009744	F	Y	N	0	315000.0	Commercial associate	Higher education	Married	House / apartment	-1
2	5009746	F	Y	N	0	315000.0	Commercial associate	Higher education	Married	House / apartment	
3	5009749	F	Y	N	0	NaN	Commercial associate	Higher education	Married	House / apartment	-1
4	5009752	F	Y	N	0	315000.0	Commercial associate	Higher education	Married	House / apartment	-1
...	...	...	...	...	...	...	...	...	...	...	
1543	5028645	F	N	Y	0	NaN	Commercial associate	Higher education	Married	House / apartment	-1
1544	5023655	F	N	N	0	225000.0	Commercial associate	Incomplete higher	Single / not married	House / apartment	-1
1545	5115992	M	Y	Y	2	180000.0	Working	Higher education	Married	House / apartment	-1
1546	5118219	M	Y	N	0	270000.0	Working	Secondary / secondary special	Civil marriage	House / apartment	-1
1547	5053790	F	Y	Y	0	225000.0	Working	Higher education	Married	House / apartment	-1

1548 rows × 18 columns

```
bb
```

	Ind_ID	label
0	5008827	1
1	5009744	1
2	5009746	1
3	5009749	1
4	5009752	1
...	...	...
1543	5028645	0
1544	5023655	0
1545	5115992	0
1546	5118219	0
1547	5053790	0

1548 rows × 2 columns

Merging the Data Set

```
cc=pd.merge(aa, bb,
how='outer', on='Ind_ID')
```

Understanding and Manupulating Data Set

- Checking the Unique value

```
cc
```

	Ind_ID	GENDER	Car_Owner	Propert_Owner	CHILDREN	Annual_income	Type_Income	EDUCATION	Marital_status	Housing_type	Birthday_
0	5008827	M	Y	Y	0	180000.0	Pensioner	Higher education	Married	House / apartment	-1
1	5009744	F	Y	N	0	315000.0	Commercial associate	Higher education	Married	House / apartment	-1
2	5009746	F	Y	N	0	315000.0	Commercial associate	Higher education	Married	House / apartment	
3	5009749	F	Y	N	0	NaN	Commercial associate	Higher education	Married	House / apartment	-1
4	5009752	F	Y	N	0	315000.0	Commercial associate	Higher education	Married	House / apartment	-1
...	...	...	...	...	...	...	...	...	...	...	
1543	5028645	F	N	Y	0	NaN	Commercial associate	Higher education	Married	House / apartment	-1
1544	5023655	F	N	N	0	225000.0	Commercial associate	Incomplete higher	Single / not married	House / apartment	-1
1545	5115992	M	Y	Y	2	180000.0	Working	Higher education	Married	House / apartment	-1
1546	5118219	M	Y	N	0	270000.0	Working	Secondary / secondary special	Civil marriage	House / apartment	-1
1547	5053790	F	Y	Y	0	225000.0	Working	Higher education	Married	House / apartment	-1

1548 rows × 19 columns

```
cc.nunique()
```

```
Ind_ID            1548
GENDER               2
Car_Owner            2
Propert_Owner        2
CHILDREN             6
Annual_income      115
Type_Income          4
EDUCATION            5
Marital_status       5
Housing_type         6
Birthday_count    1270
Employed_days      956
Mobile_phone         1
Work_Phone           2
Phone                2
EMAIL_ID             2
Type_Occupation     18
Family_Members       7
label                2
dtype: int64
```

Checking Null Value

```
cc.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
label                0
dtype: int64
```

```
cc['GENDER'].unique()
```

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

```
cc[cc['GENDER'] == 'M'].count()
```

```
Ind_ID             568
GENDER             568
Car_Owner          568
Propert_Owner      568
CHILDREN           568
Annual_income      559
Type_Income        568
EDUCATION          568
Marital_status     568
Housing_type       568
Birthday_count     558
Employed_days      568
Mobile_phone       568
Work_Phone         568
Phone              568
EMAIL_ID           568
Type_Occupation    438
Family_Members     568
label              568
dtype: int64
```

```
cc[cc['GENDER'] == 'F'].count()
```

```
Ind_ID             973
GENDER             973
Car_Owner          973
Propert_Owner      973
CHILDREN           973
Annual_income      959
Type_Income        973
EDUCATION          973
Marital_status     973
Housing_type       973
Birthday_count     961
Employed_days      973
Mobile_phone       973
Work_Phone         973
Phone              973
EMAIL_ID           973
Type_Occupation    617
Family_Members     973
label              973
dtype: int64
```

```
# The mode here in gender is females. So we can change the null values to the mode value.

cc['GENDER'] = cc['GENDER'].fillna('F')

cc['GENDER'].unique()
```

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

```
cc['Annual_income'].unique()
```

```
array([ 180000. ,  315000. ,       nan,  450000. ,   90000. ,  472500. ,
        270000. ,  126000. ,  202500. ,  157500. ,  112500. ,  540000. ,
        292500. ,  135000. ,   76500. ,  215100. ,  225000. ,   67500. ,
        171000. ,  103500. ,   99000. ,  391500. ,   65250. ,   72900. ,
        360000. ,  256500. ,  675000. ,  247500. ,   85500. ,  121500. ,
        130500. ,  211500. ,   81000. ,   72000. ,  148500. ,  162000. ,
        195750. ,  585000. ,  216000. ,  306000. ,  108000. ,   63000. ,
         45000. ,  337500. ,  131400. ,  117000. ,  445500. ,  234000. ,
       1575000. ,  144000. ,   67050. ,   73350. ,  193500. ,  900000. ,
         94500. ,  198000. ,   54000. ,  166500. ,  167400. ,  153000. ,
        423000. ,  243000. ,  283500. ,  252000. ,  495000. ,  612000. ,
         36000. ,  139500. ,  133650. ,  427500. ,  261000. ,  231750. ,
         90900. ,   45900. ,  119250. ,   58500. ,  328500. ,  787500. ,
        594000. ,  119700. ,   69372. ,   37800. ,  387000. ,  207000. ,
        189000. ,  333000. ,  105750. ,  382500. ,  141750. ,   40500. ,
        405000. ,   44550. ,  301500. ,  351000. ,  175500. ,  121900.5,
        238500. ,   33750. ,  116100. ,  297000. ,  630000. ,  418500. ,
         83250. ,  173250. ,  274500. ,  115200. ,   56250. ,   95850. ,
        185400. ,  810000. ,  184500. ,  165600. ,  114750. ,   47250. ,
         49500. ,   69750. ])
```

```
cc['Annual_income'].mean()
```

```
191399.3262295082
```

```
cc['Annual_income'] = cc['Annual_income'].fillna(cc['Annual_income'].mean())
```

```
cc['Annual_income'].unique()
```

```
array([ 180000.        ,  315000.        ,  191399.32622951,
        450000.        ,   90000.        ,  472500.        ,
        270000.        ,  126000.        ,  202500.        ,
        157500.        ,  112500.        ,  540000.        ,
        292500.        ,  135000.        ,   76500.        ,
        215100.        ,  225000.        ,   67500.        ,
        171000.        ,  103500.        ,   99000.        ,
        391500.        ,   65250.        ,   72900.        ,
        360000.        ,  256500.        ,  675000.        ,
        247500.        ,   85500.        ,  121500.        ,
        130500.        ,  211500.        ,   81000.        ,
         72000.        ,  148500.        ,  162000.        ,
        195750.        ,  585000.        ,  216000.        ,
        306000.        ,  108000.        ,   63000.        ,
         45000.        ,  337500.        ,  131400.        ,
        117000.        ,  445500.        ,  234000.        ,
       1575000.        ,  144000.        ,   67050.        ,
         73350.        ,  193500.        ,  900000.        ,
         94500.        ,  198000.        ,   54000.        ,
        166500.        ,  167400.        ,  153000.        ,
        423000.        ,  243000.        ,  283500.        ,
        252000.        ,  495000.        ,  612000.        ,
         36000.        ,  139500.        ,  133650.        ,
        427500.        ,  261000.        ,  231750.        ,
         90900.        ,   45900.        ,  119250.        ,
         58500.        ,  328500.        ,  787500.        ,
        594000.        ,  119700.        ,   69372.        ,
         37800.        ,  387000.        ,  207000.        ,
        189000.        ,  333000.        ,  105750.        ,
        382500.        ,  141750.        ,   40500.        ,
        405000.        ,   44550.        ,  301500.        ,
        351000.        ,  175500.        ,  121900.5       ,
        238500.        ,   33750.        ,  116100.        ,
        297000.        ,  630000.        ,  418500.        ,
         83250.        ,  173250.        ,  274500.        ,
        115200.        ,   56250.        ,   95850.        ,
        185400.        ,  810000.        ,  184500.        ,
        165600.        ,  114750.        ,   47250.        ,
         49500.        ,   69750.        ])
```

```
cc['Birthday_count'] = cc['Birthday_count'].fillna(cc['Birthday_count'].mean())
```

```
cc['Birthday_count'].unique()
```

```
array([-18772.        , -13557.        , -16040.34207077, ...,
       -10229.        , -15292.        , -16601.        ])
```

```python
cc['Type_Occupation'].mode()
```

```
0    Laborers
Name: Type_Occupation, dtype: object
```

```python
cc['Type_Occupation'] = cc['Type_Occupation'].fillna('Laborers')
```

```python
cc['Type_Occupation'].unique()
```

```
array(['Laborers', 'Core staff', 'Cooking staff', 'Sales staff',
       'Accountants', 'High skill tech staff', 'Managers',
       'Cleaning staff', 'Drivers', 'Low-skill Laborers', 'IT staff',
       'Waiters/barmen staff', 'Security staff', 'Medicine staff',
       'Private service staff', 'HR staff', 'Secretaries',
       'Realty agents'], dtype=object)
```

```python
cc.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
Birthday_count     0
Employed_days      0
Mobile_phone       0
Work_Phone         0
Phone              0
EMAIL_ID           0
Type_Occupation    0
Family_Members     0
label              0
dtype: int64
```

```python
for c in cc.columns:
    print("---- %s ---" % c)
    print(cc[c].value_counts())
```

```
---- Ind_ID ---
5008827    1
5142163    1
5024925    1
5143560    1
5068648    1
          ..
5148792    1
5142290    1
5095324    1
5118270    1
5053790    1
Name: Ind_ID, Length: 1548, dtype: int64
---- GENDER ---
F    980
M    568
Name: GENDER, dtype: int64
---- Car_Owner ---
N    924
Y    624
Name: Car_Owner, dtype: int64
---- Propert_Owner ---
Y    1010
N     538
Name: Propert_Owner, dtype: int64
---- CHILDREN ---
0     1091
1      305
2      134
3       16
```

```
4       1
14      1
Name: CHILDREN, dtype: int64
---- Annual_income ---
135000.0    170
112500.0    144
180000.0    137
157500.0    125
225000.0    119
           ...
119700.0      1
69372.0       1
37800.0       1
333000.0      1
69750.0       1
Name: Annual_income, Length: 116, dtype: int64
---- Type_Income ---
Working                 798
Commercial associate    365
Pensioner               269
State servant           116
Name: Type_Income, dtype: int64
---- EDUCATION ---
Secondary / secondary special    1031
Higher education                  426
Incomplete higher                  68
Lower secondary                    21
```

```
cc.dtypes
```

```
Ind_ID               int64
GENDER              object
Car_Owner           object
Propert_Owner       object
CHILDREN             int64
Annual_income      float64
Type_Income         object
EDUCATION           object
Marital_status      object
Housing_type        object
Birthday_count     float64
Employed_days        int64
Mobile_phone         int64
Work_Phone           int64
Phone                int64
EMAIL_ID             int64
Type_Occupation     object
Family_Members       int64
label                int64
dtype: object
```

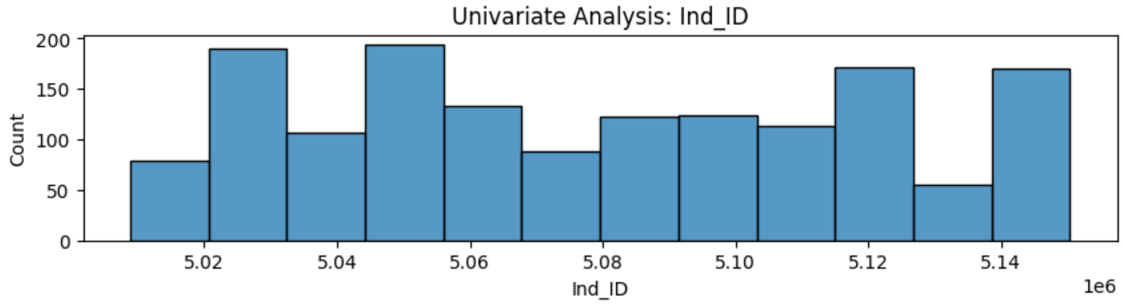
```
cc.to_csv('Cleaned_dataset')
```

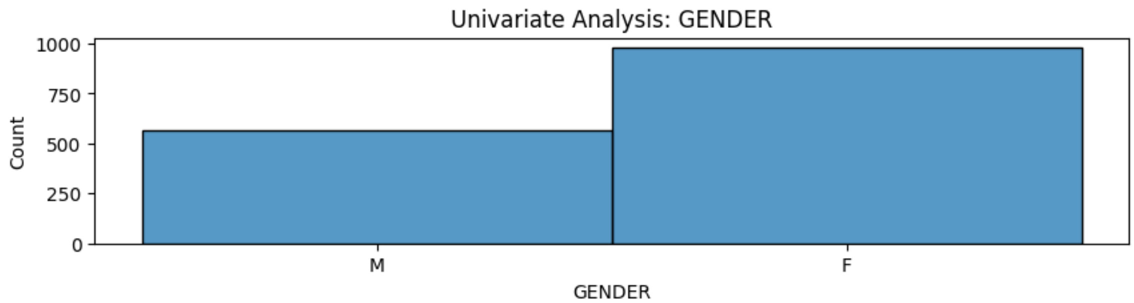
EDA

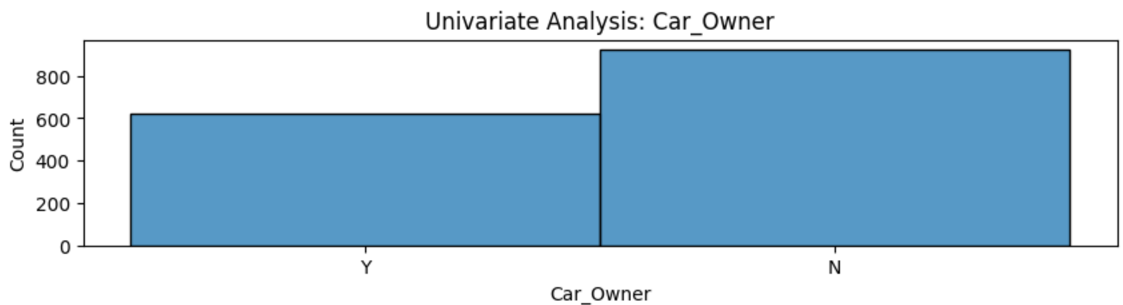
Univariate Analaysis

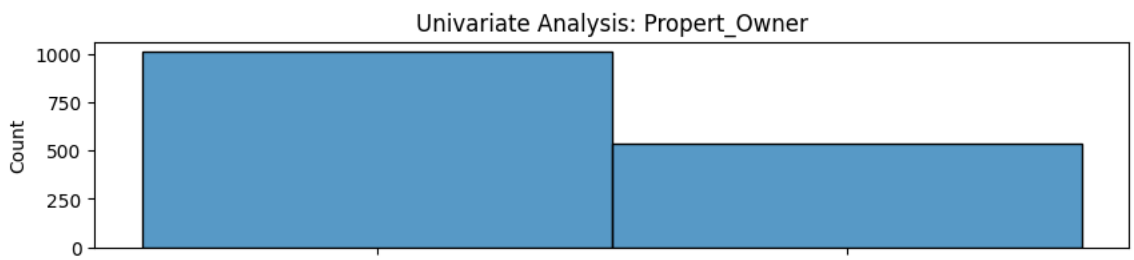
- Histogram
- Pie Chart

```
for column in cc.columns:
    plt.figure(figsize=(10,2))
    sns.histplot(cc[column])
    plt.title(f'Univariate Analysis: {column}')
    plt.xlabel(column)
    plt.ylabel('Count')
    plt.show()
```







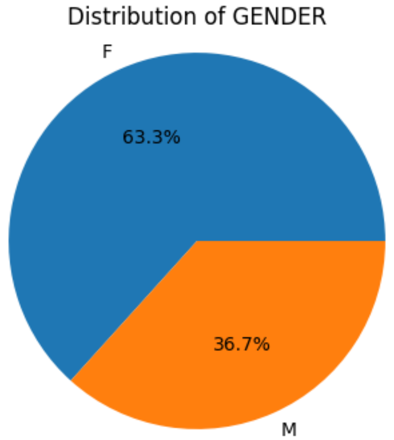


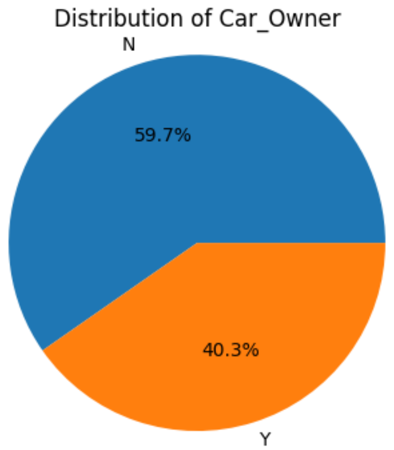
Insights:

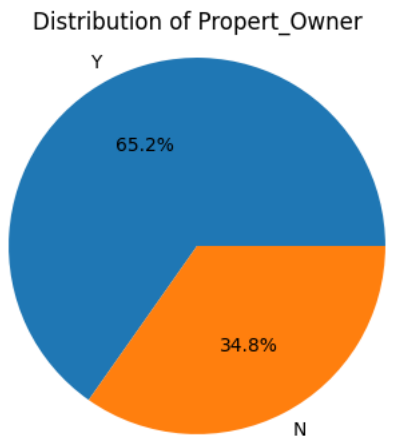
- Most of the applicants are females.
- Most of the applicants do not own car, but they own property.
- From the graph it was found that, more than 1000 applicants do not have children.
- From the Annual income histogram, the graph is right skewed whcih implies that most of the applicants are present towards right of the peak. The peak is pointed at approximate value 0.12.
- From the above histogram we can observe most of the applicants source of income is through working.
- Most of the applicants education level is secondary/secondary special.
- Most of the applicants are married.
- Most of the applicants own house/apartments.
- Birthday count values are normally distributed.
- many of the applicants have lesser employed days.
- Each and every applicant has mobile phones.
- Most of the applicants do not have work phone.
- Only few of the applicants have email-ID.
- Most of the applicants are labourer's by occupation.
- Most of the applicants have 2 members in the family.
- Most of the applicants credit card is approved.

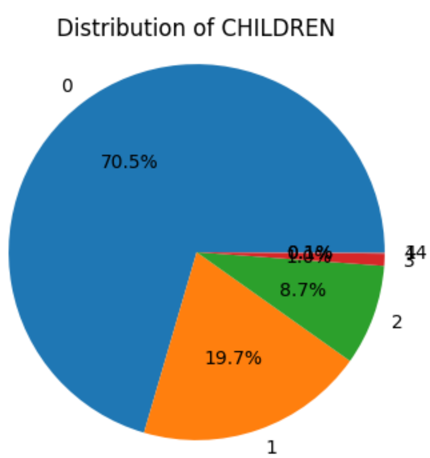
0.0 0.2 0.4 0.6 0.8 1.0 1.2 1.4 1.6

```
columns = ['GENDER', 'Car_Owner', 'Propert_Owner', 'CHILDREN', 'Type_Income', 'EDUCATION', 'Marital

for column in columns:

    category_counts = cc[column].value_counts()

    plt.figure(figsize=(4, 4))
    plt.pie(category_counts, labels=category_counts.index, autopct='%1.1f%%')
    plt.title(f'Distribution of {column}')
    plt.axis('equal')
    plt.show()
```

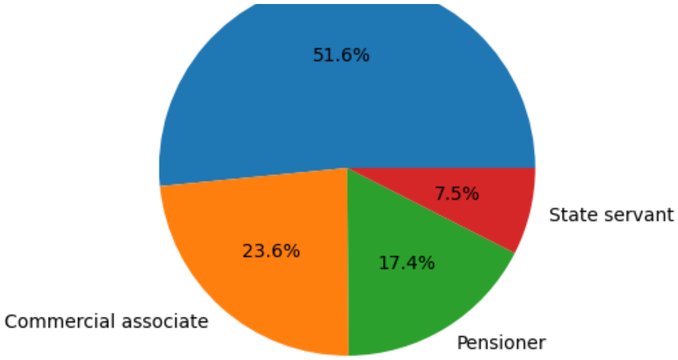

Distribution of GENDER
F
63.3%
36.7%
M


Distribution of Car_Owner
N
59.7%
40.3%
Y

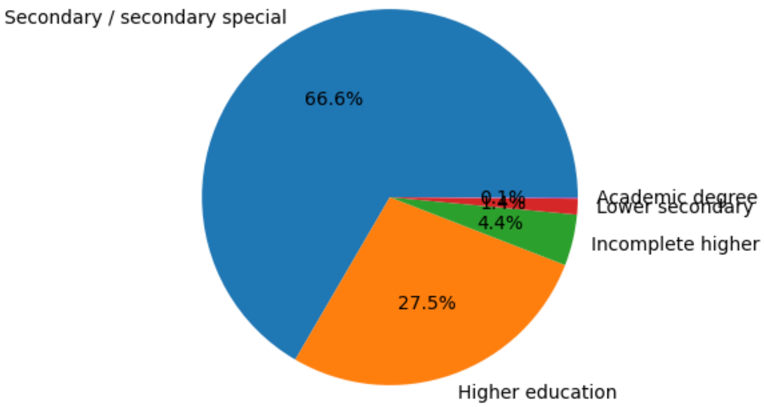

Distribution of Propert_Owner
Y
65.2%
34.8%
N


Distribution of CHILDREN
0
70.5%
8.7%
2
19.7%
1

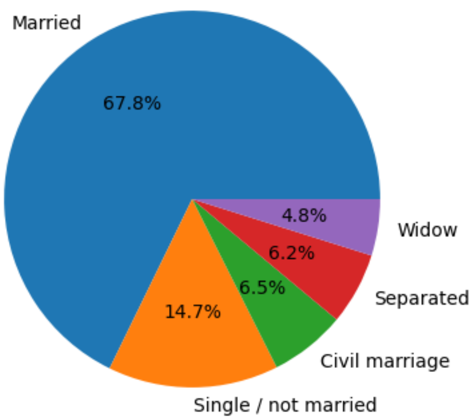

Distribution of Type_Income
Working



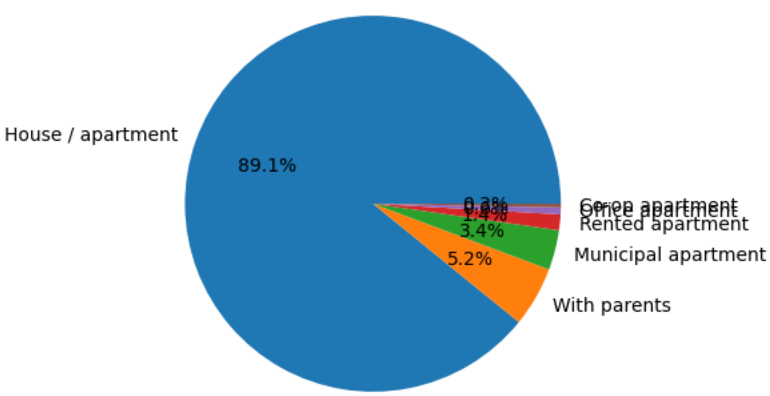
Distribution of EDUCATION



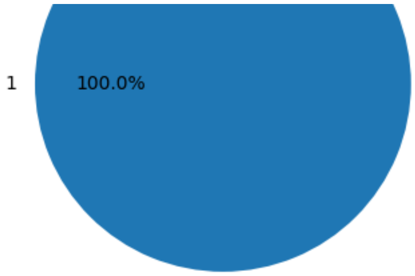
Distribution of Marital_status



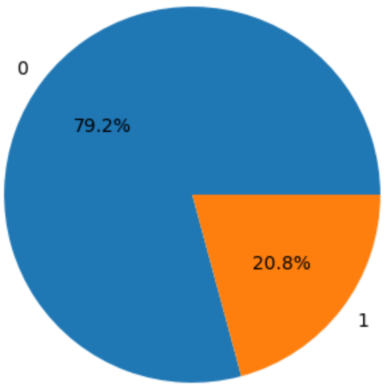
Distribution of Housing_type



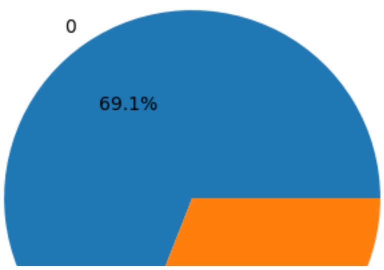
Distribution of Mobile_phone



Distribution of Work_Phone



Distribution of Phone



Insights:

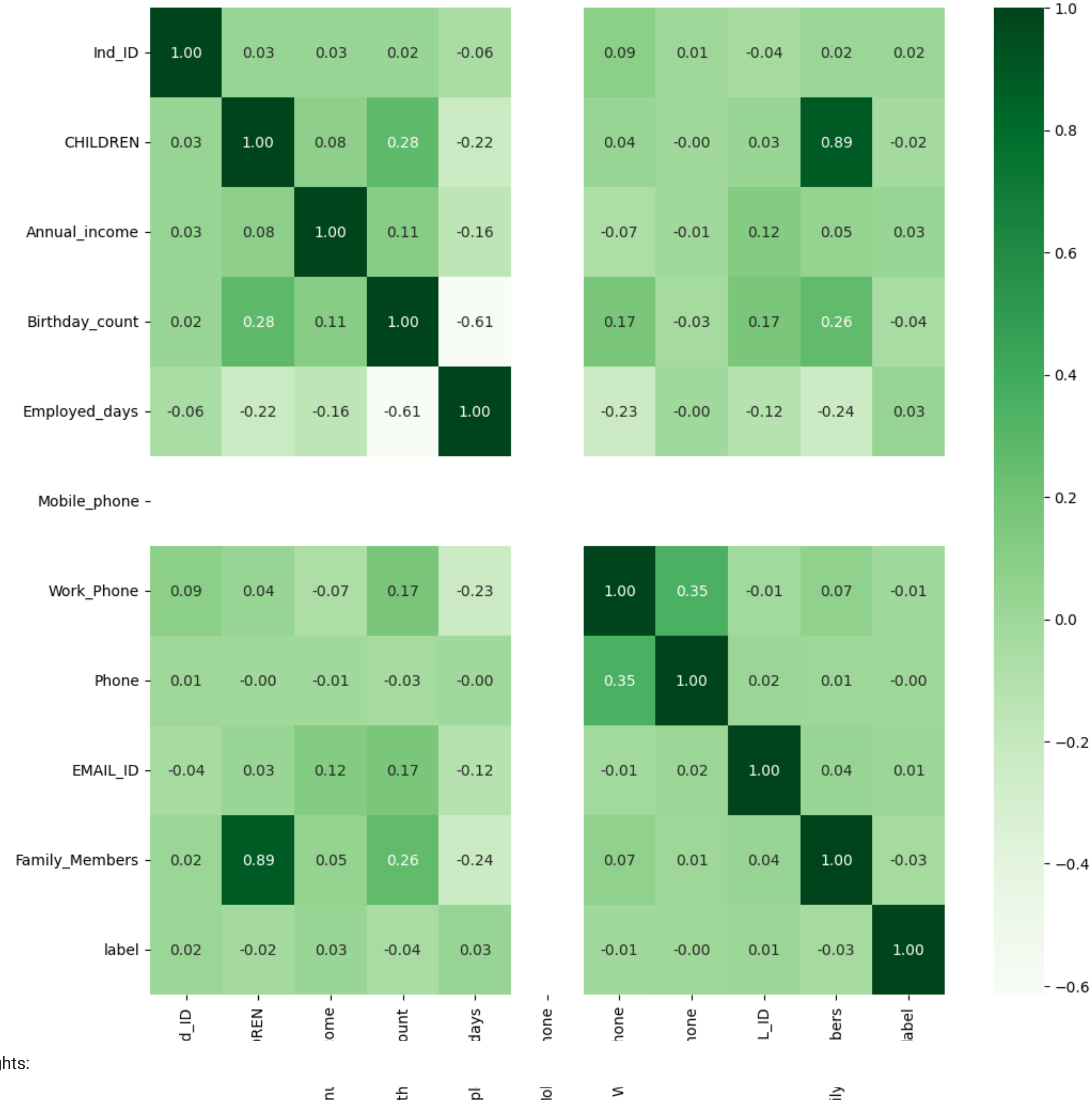
- 63.3% of the aopplicants are females and 36.7% of the applicants are males.
- 40.3% of the applicants own car, and 59.7% of the applicants do not own car.
- 65.2% of the applicants own property and 34.8% of the applicants do not own property.
- 70.5% of the applicants do not have children.
- Approximately 50% of the applicants source of income is through working.
- Approximately 10% of the applicants do not own House/apartment.

Bivariate Analysis

- Correlation
- Scatter plot
- Pair plot

```
corr_df=cc.corr()

plt.figure(figsize = (12,12))
sns.heatmap(data = corr_df, annot = True, cmap = "Greens", cbar = True, fmt='.2f')
plt.show()
```

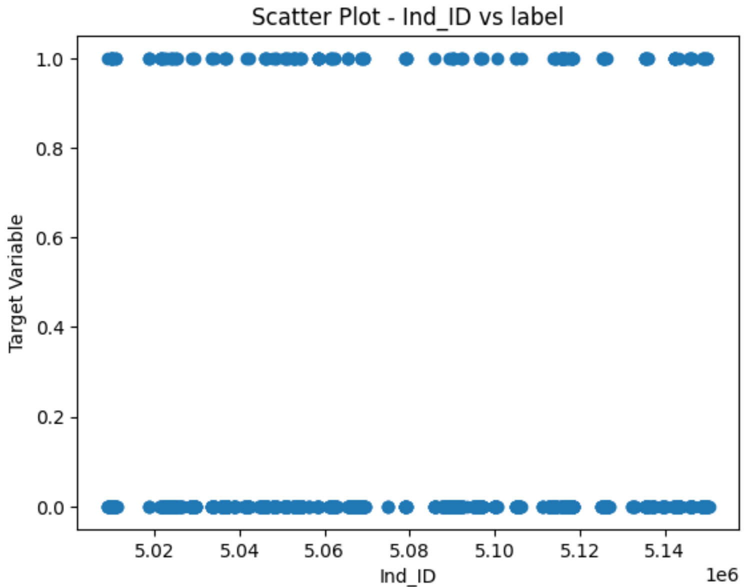
```
<ipython-input-133-8507fb91683a>:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future versic
  corr_df=cc.corr()
```



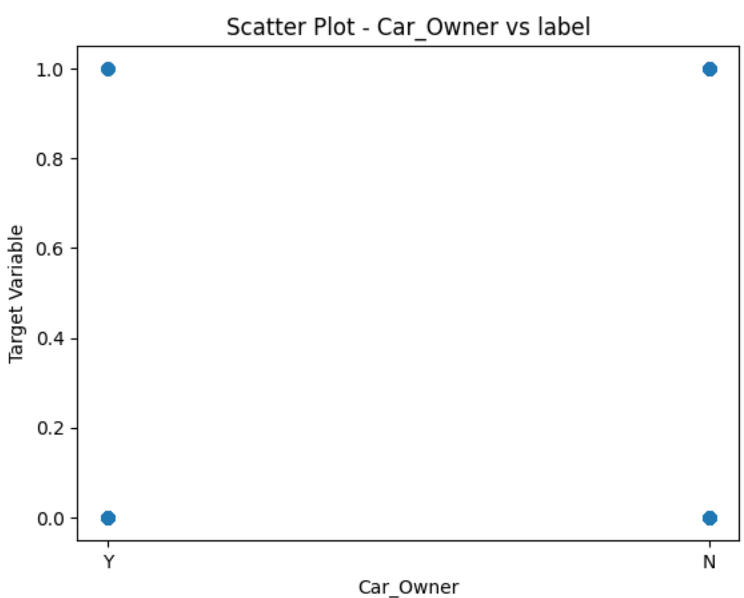
Insights:

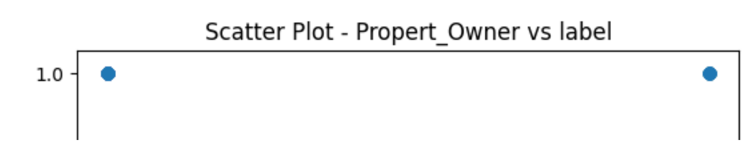
- The family members and children column are highly correlated.
- The annual income and Employed days are more correlated than any other columns.

```
target=['label']
for column in cc.columns:
    # Create a scatter plot between the target variable and the current column

    plt.figure()
    plt.scatter(cc[column], cc['label'])
    plt.xlabel(column)
    plt.ylabel('Target Variable')
    plt.title(f'Scatter Plot - {column} vs label')
    plt.show()
```

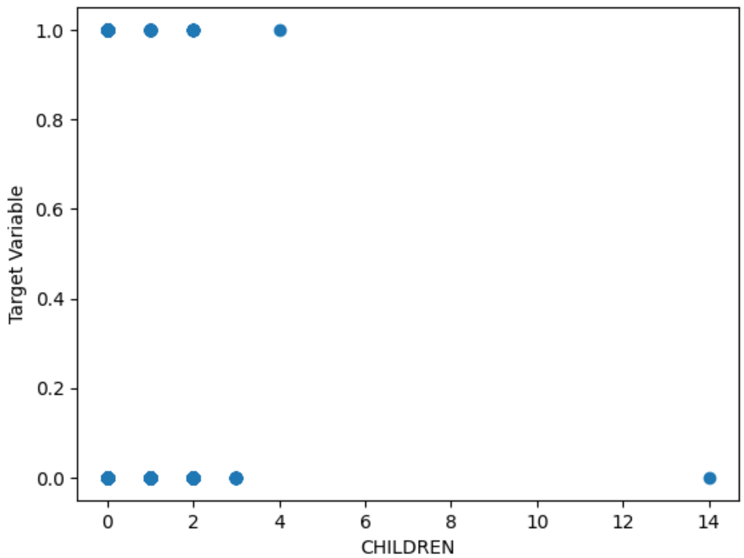

Scatter Plot - Ind_ID vs label
Target Variable
Ind_ID
1e6


Scatter Plot - GENDER vs label
Target Variable
M
F
GENDER

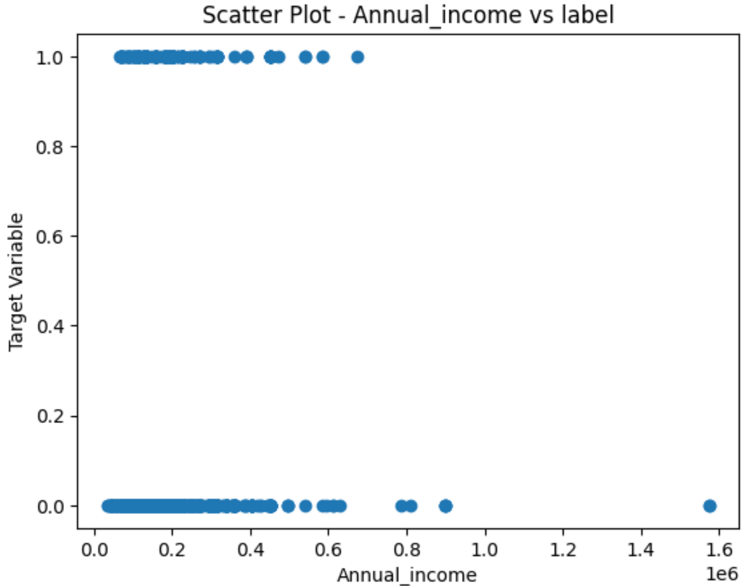

Scatter Plot - Car_Owner vs label
Target Variable
Y
N
Car_Owner


Scatter Plot - Propert_Owner vs label

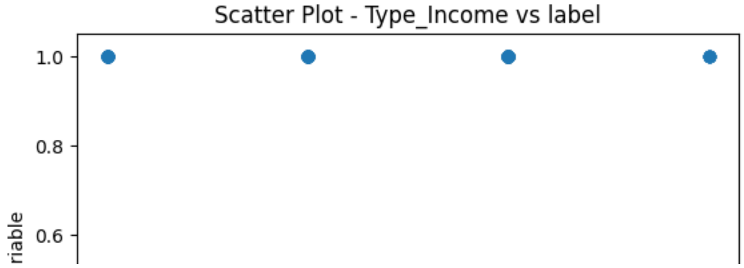
Scatter Plot - CHILDREN vs label

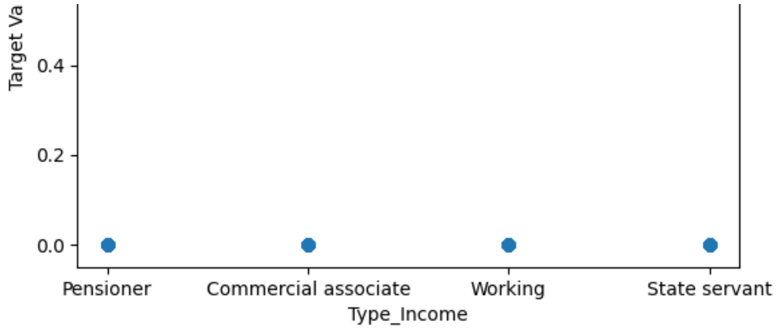


Scatter Plot - Annual_income vs label

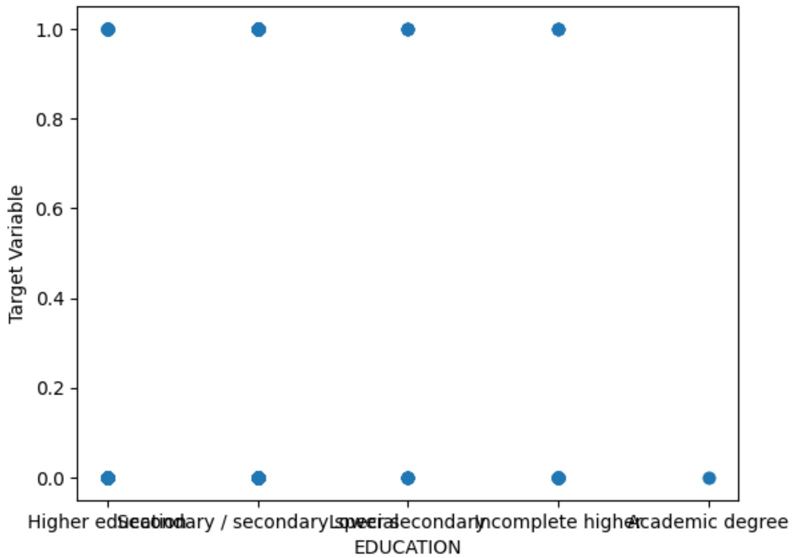


Scatter Plot - Type_Income vs label

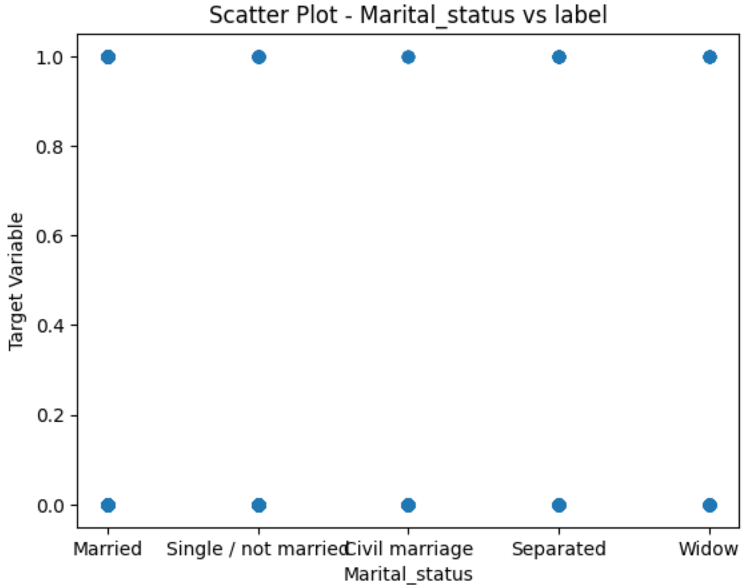




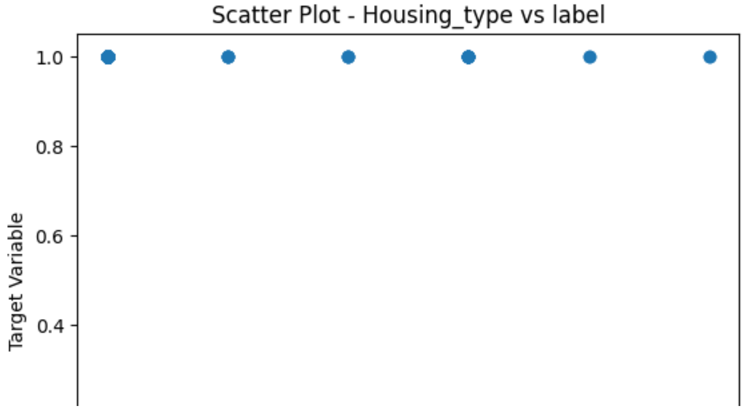
Scatter Plot - EDUCATION vs label

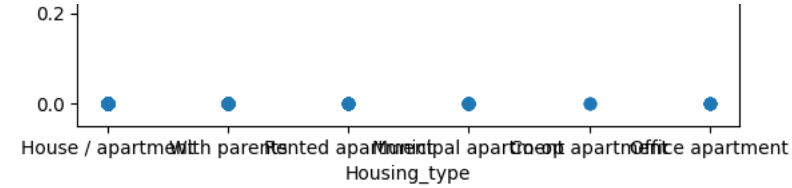


Scatter Plot - Marital_status vs label

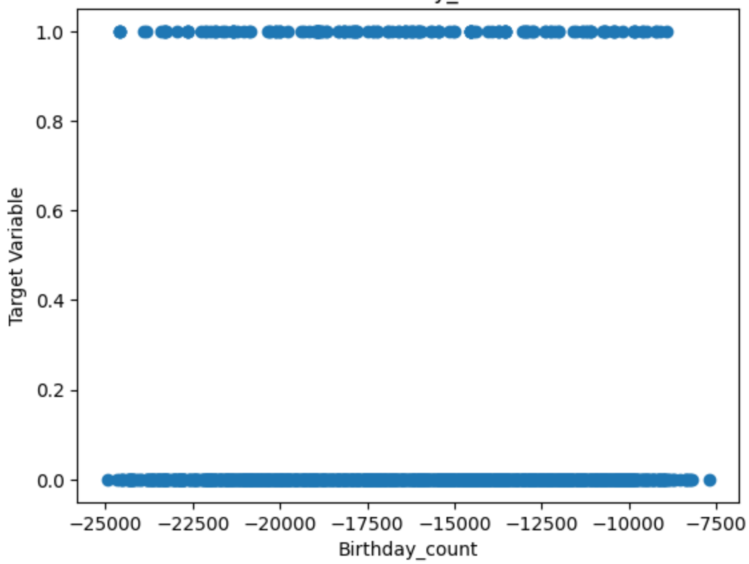


Scatter Plot - Housing_type vs label

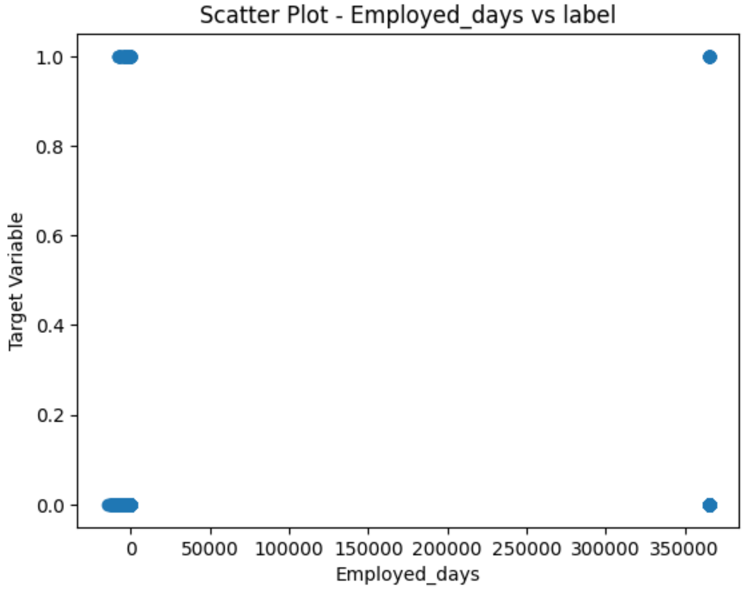




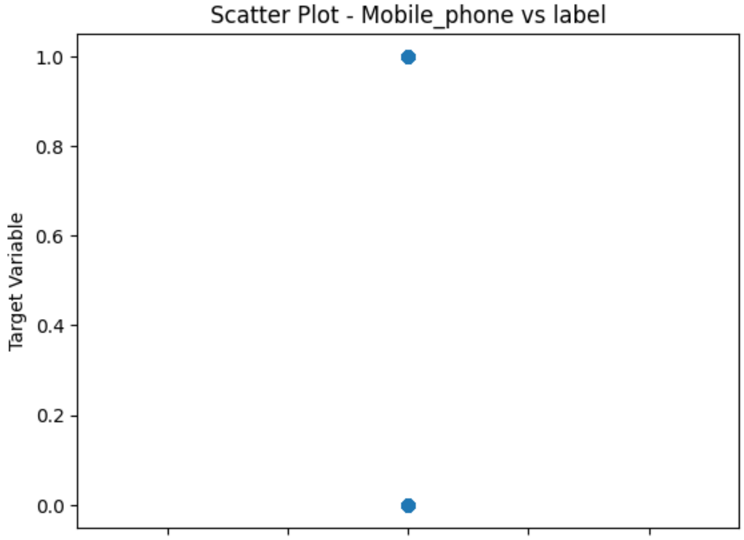
Scatter Plot - Birthday_count vs label

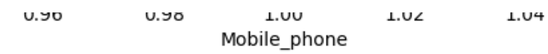


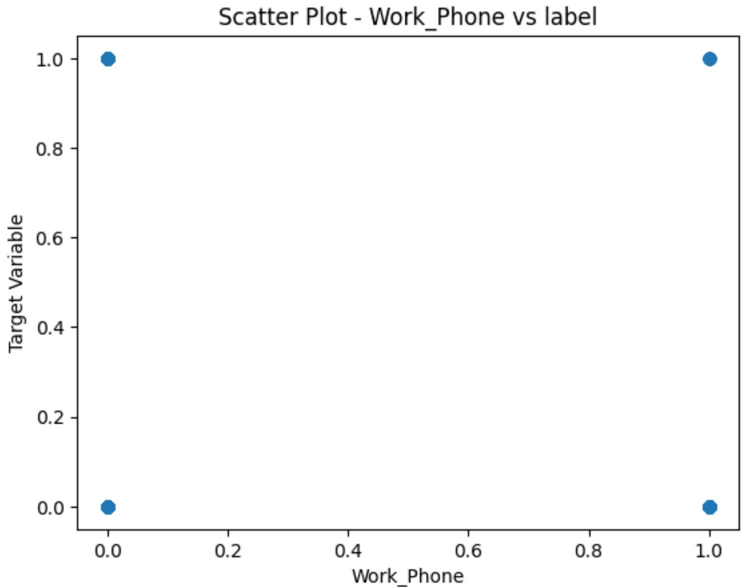
Scatter Plot - Employed_days vs label

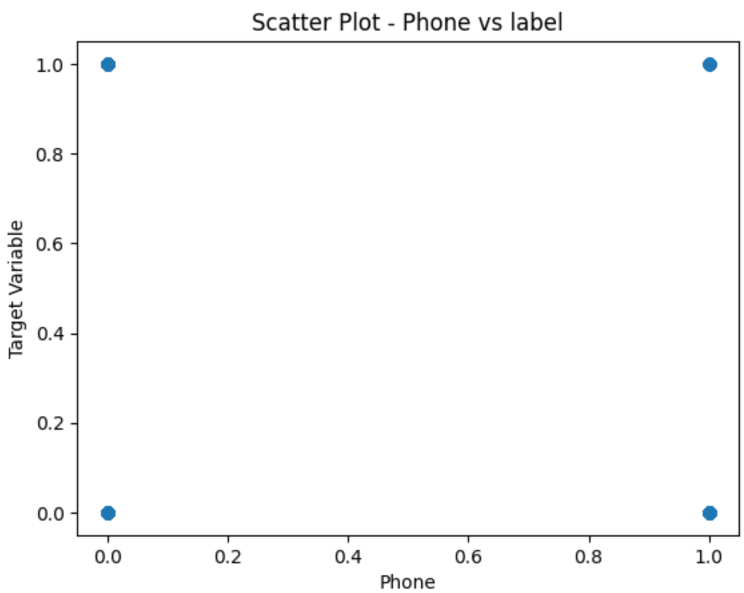


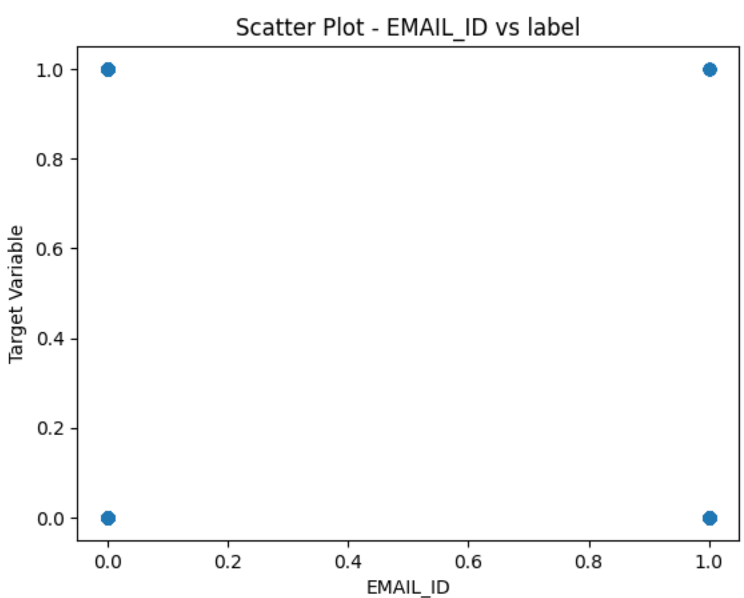
Scatter Plot - Mobile_phone vs label

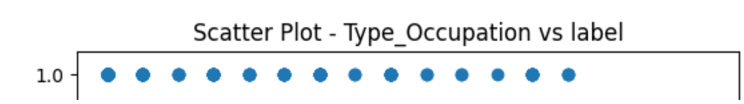


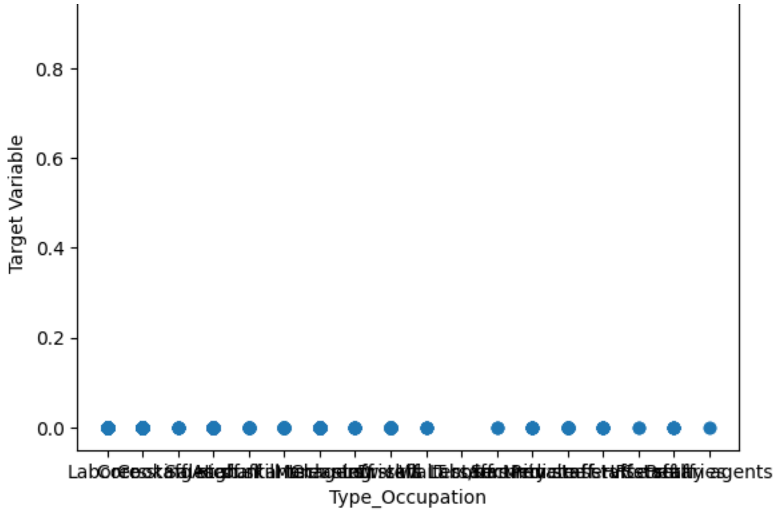

0.96
0.98
1.00
1.02
1.04
Mobile_phone

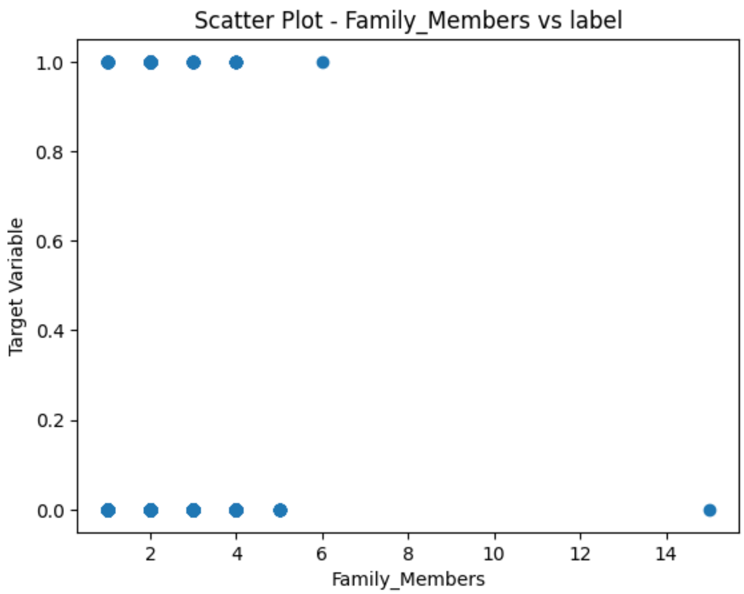

Scatter Plot - Work_Phone vs label
Target Variable
Work_Phone
0.0
0.2
0.4
0.6
0.8
1.0

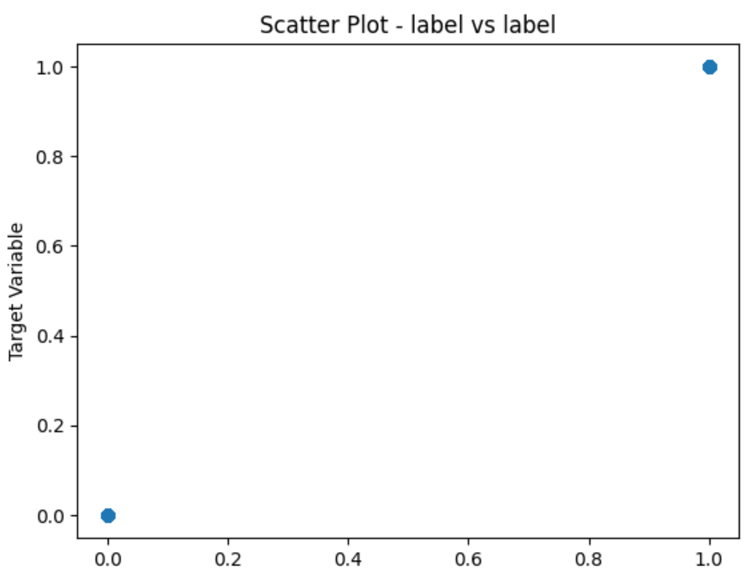

Scatter Plot - Phone vs label
Target Variable
Phone
0.0
0.2
0.4
0.6
0.8
1.0


Scatter Plot - EMAIL_ID vs label
Target Variable
EMAIL_ID
0.0
0.2
0.4
0.6
0.8
1.0


Scatter Plot - Type_Occupation vs label
1.0

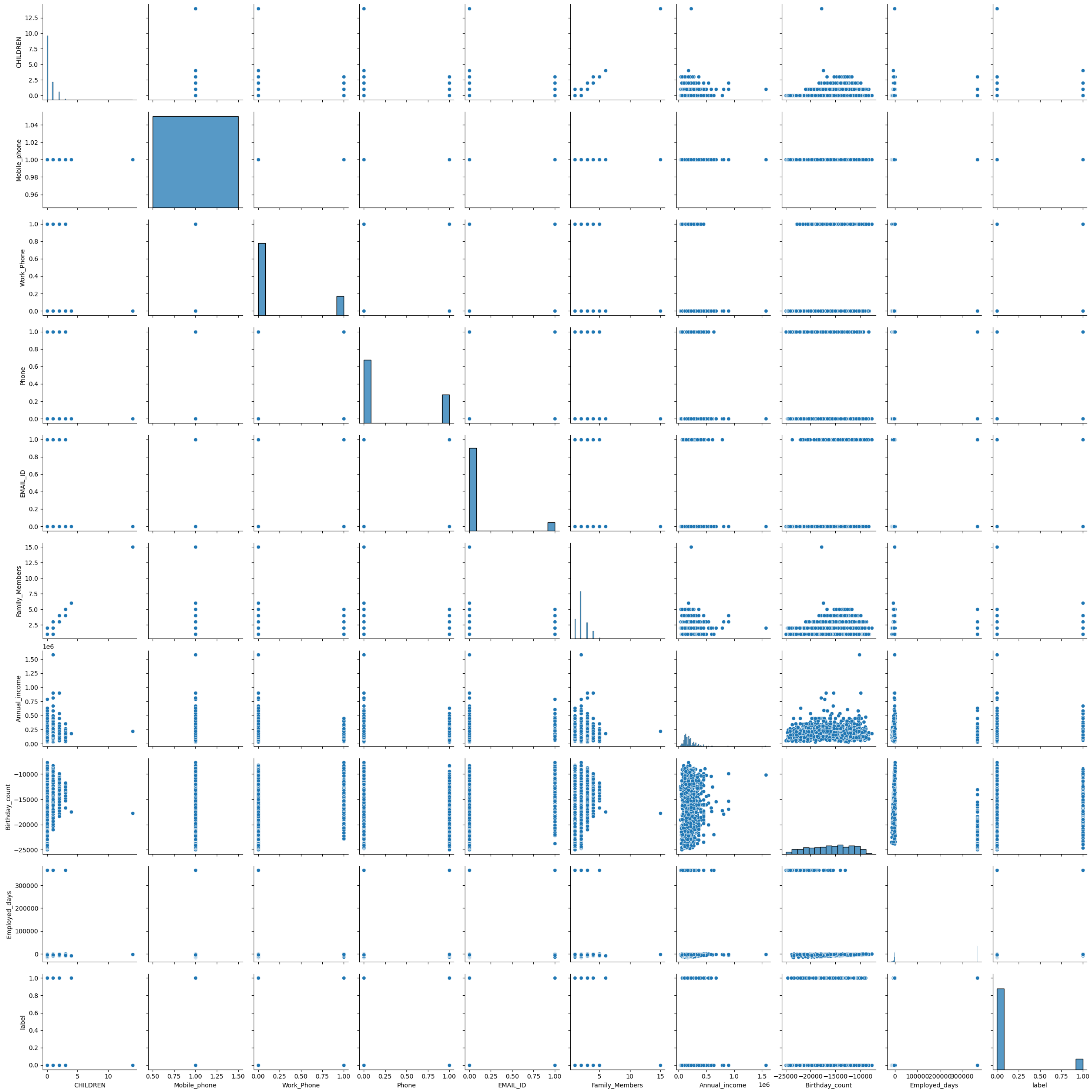






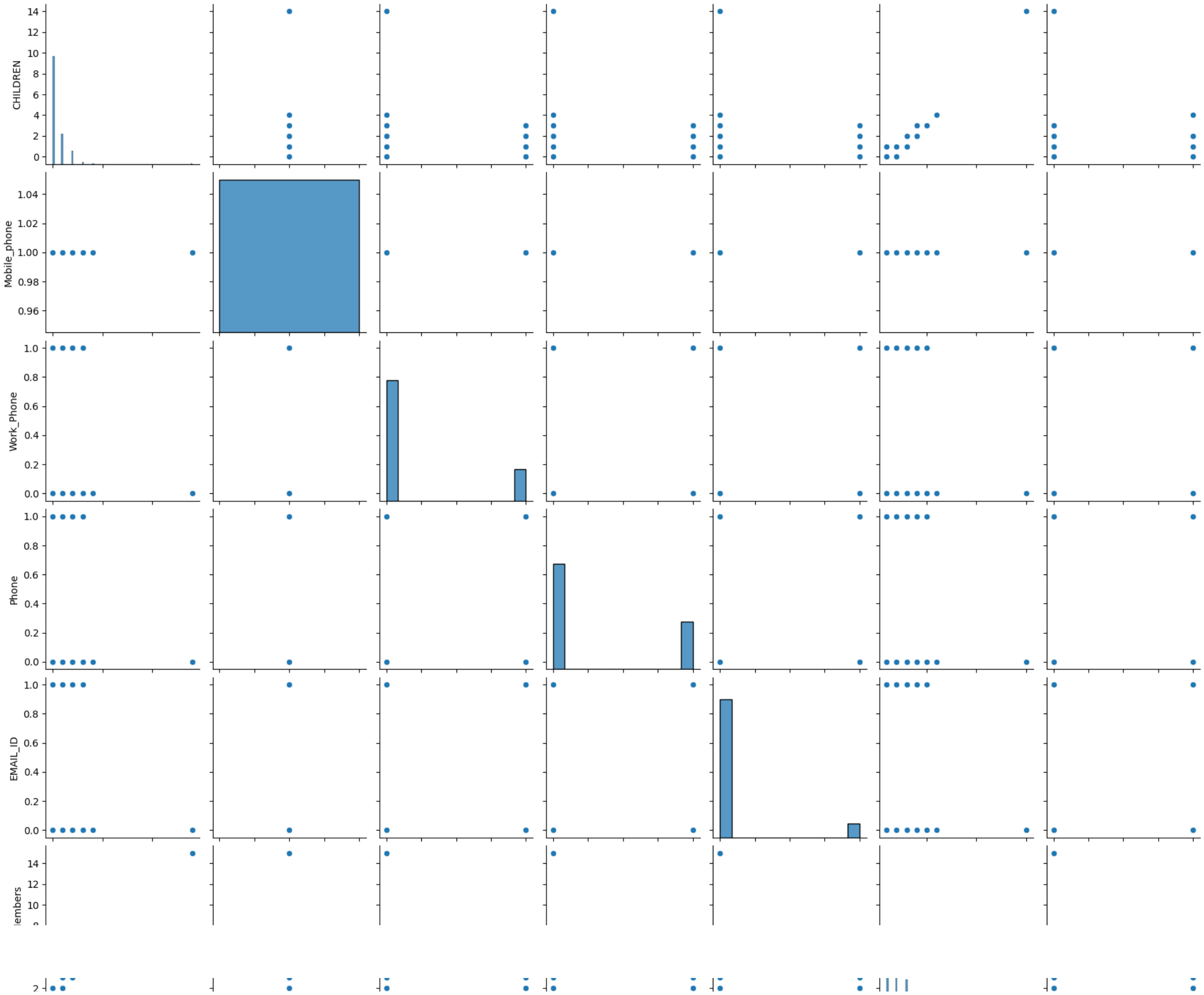

```
columns = ['GENDER', 'Car_Owner','Propert_Owner','CHILDREN','Type_Income','EDUCATION','Marital_stat
sns.pairplot(cc[columns])
```

<seaborn.axisgrid.PairGrid at 0x7fe14bd01570>

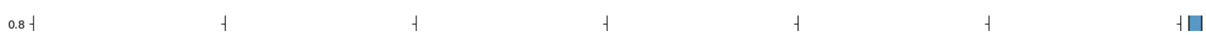



```
columns = ['GENDER', 'Car_Owner','Propert_Owner','CHILDREN','Type_Income','EDUCATION','Marital_stat
sns.pairplot(cc[columns])
```

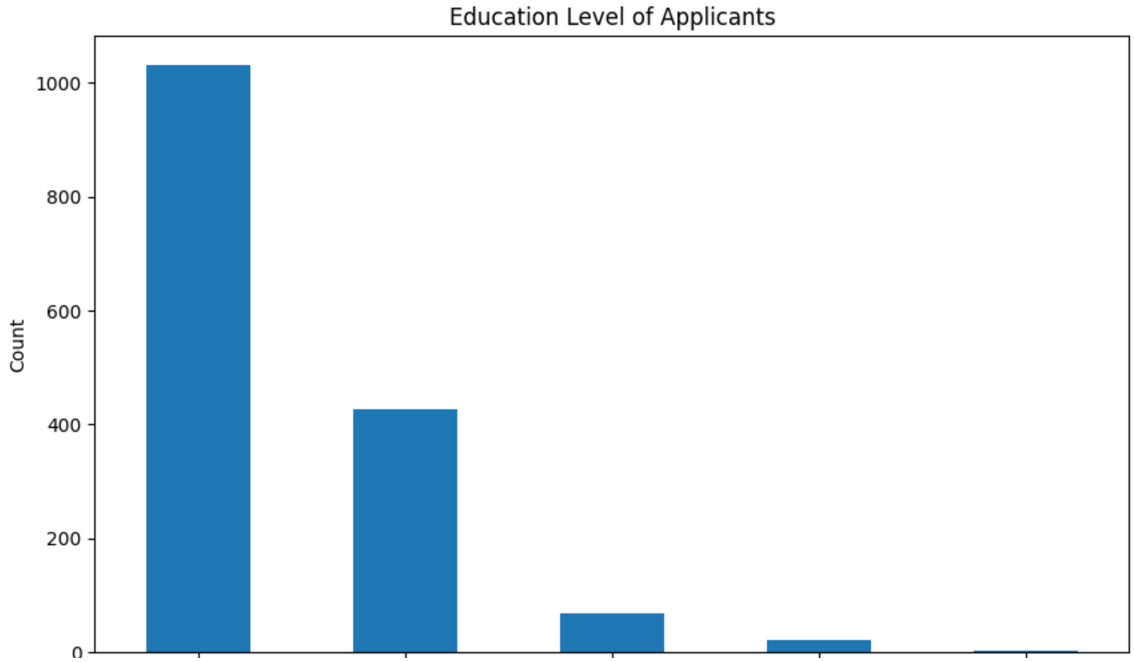
```
<seaborn.axisgrid.PairGrid at 0x7fe1425a1c00>
```



Education Level of applicants in Bar chart



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

education_counts = cc['EDUCATION'].value_counts()

plt.figure(figsize=(10, 6))
education_counts.plot(kind='bar')
plt.xlabel('EDUCATION')
plt.ylabel('Count')
plt.title('Education Level of Applicants')
plt.xticks(rotation=45)
plt.show()
```



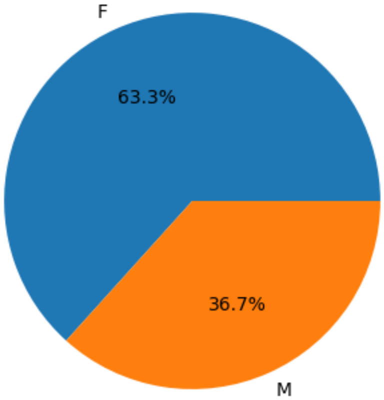
Insights :

- From the graph above, most of the applicants have secondary/secondary special level of Education, which is followed by Higher Education. The remaining applicants contribution towards Education level is minimal/negligible.

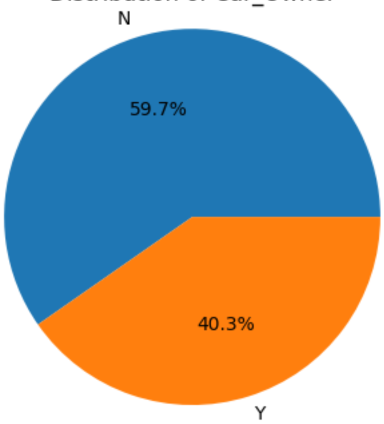
Distribution of Gender in Pie Chart

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

columns = ['GENDER', 'Car_Owner', 'Propert_Owner', 'CHILDREN', 'Type_Income', 'EDUCATION', 'Marital

for column in columns:

    category_counts = cc[column].value_counts()

    plt.figure(figsize=(4, 4))
    plt.pie(category_counts, labels=category_counts.index, autopct='%1.1f%%')
    plt.title(f'Distribution of {column}')
    plt.axis('equal')
    plt.show()
```

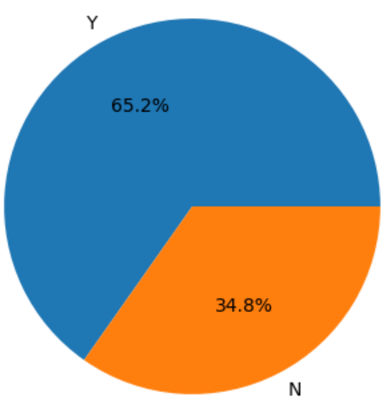
Distribution of GENDER



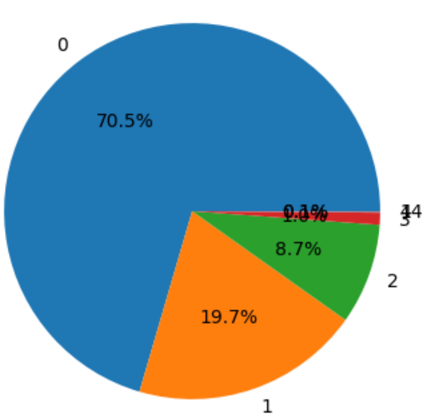
Distribution of Car_Owner



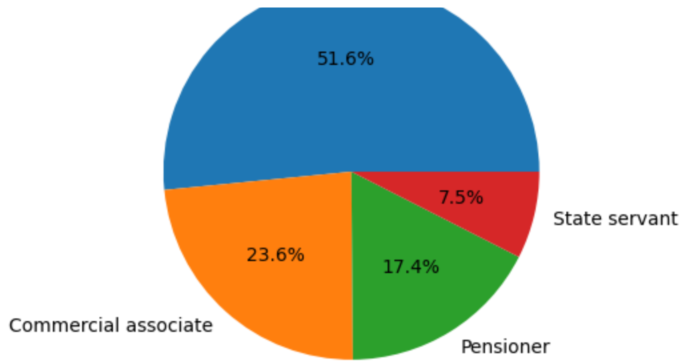
Distribution of Propert_Owner



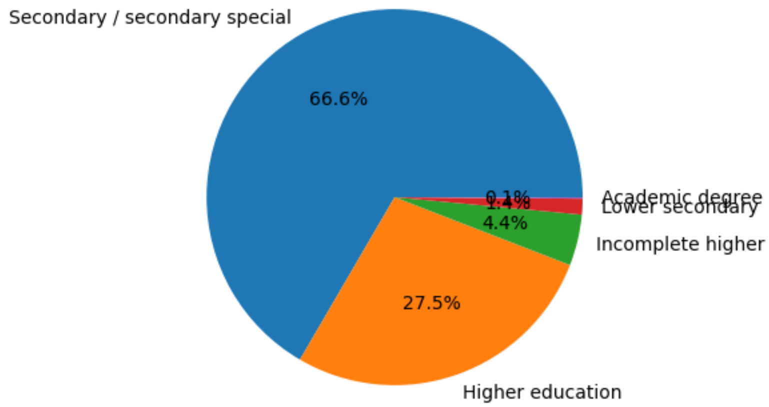
Distribution of CHILDREN



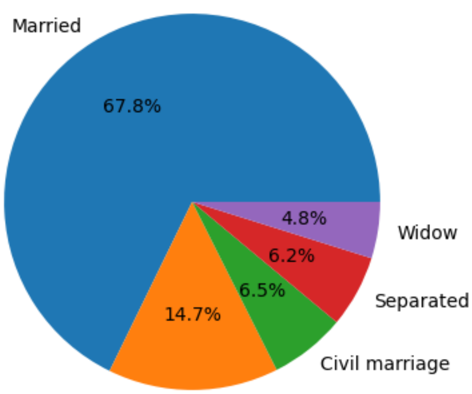
Distribution of Type_Income



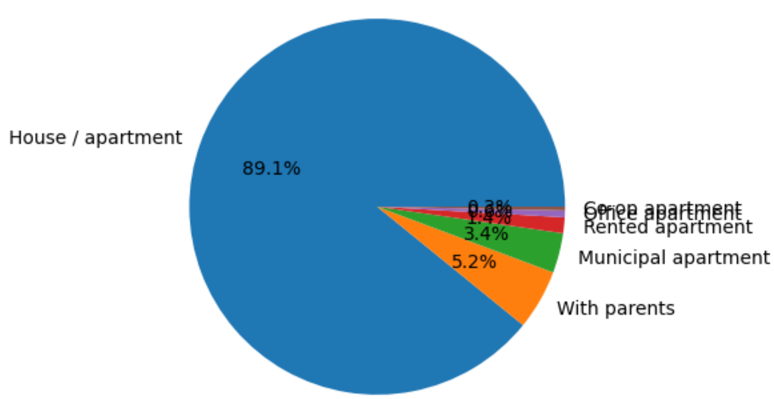
Distribution of EDUCATION



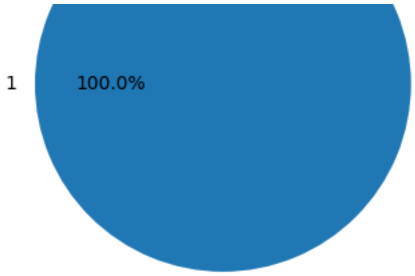
Distribution of Marital_status



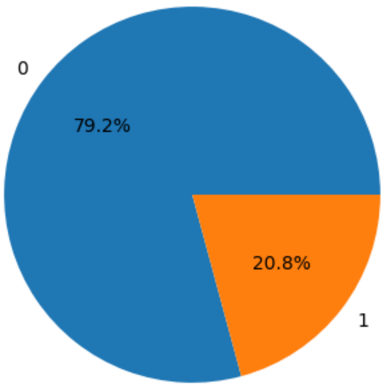
Distribution of Housing_type



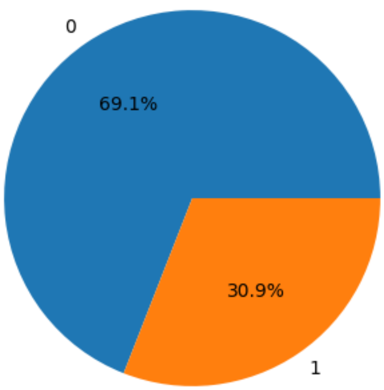
Distribution of Mobile_phone



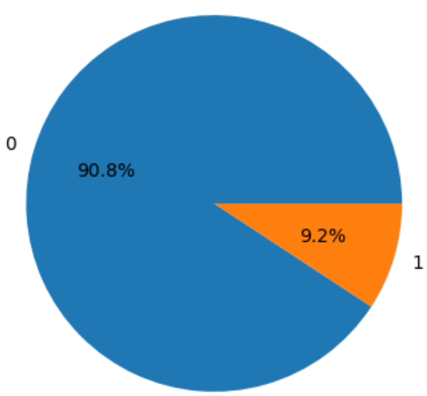
Distribution of Work_Phone



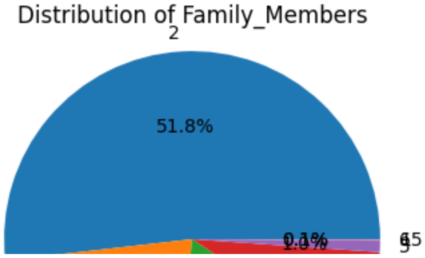
Distribution of Phone

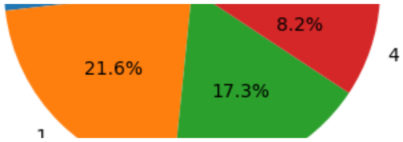


Distribution of EMAIL_ID



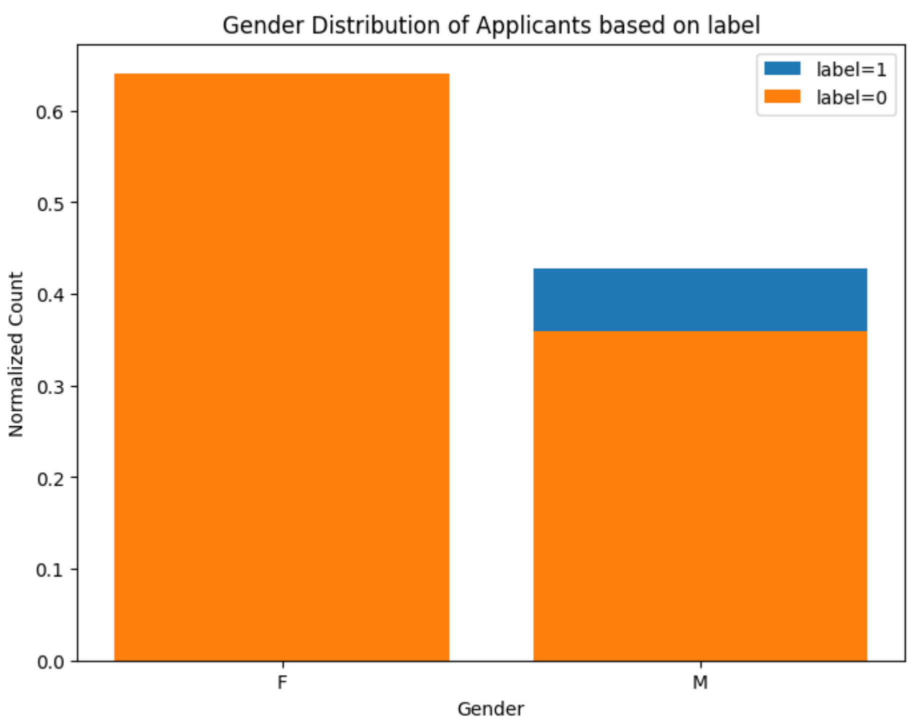
Distribution of Family_Members





Gender Distribution of Applicants in Stacked Bar Chart

```
gender_column = 'GENDER'
target = 'label'

categories = cc[target].unique()

plt.figure(figsize=(8, 6))
for category in categories:
    filtered_data = cc[cc[target] == category]
    gender_counts = filtered_data[gender_column].value_counts(normalize=True)
    plt.bar(gender_counts.index, gender_counts.values, label=f'{target}={category}')

plt.xlabel('Gender')
plt.ylabel('Normalized Count')
plt.title('Gender Distribution of Applicants based on label')
plt.legend()
plt.show()
```



Insights:

- Every female applicant's credit card is approved.
- Few of the male applicant's credit is rejected.

Feature Engineering

Imputation

Changing the categorical variables into numerical columns.

Label encoding :-

- gender
- Car owner
- property owner
- Type income
- Education
- Marital status
- Housing type
- Type occupation

```
from sklearn.preprocessing import LabelEncoder

columns = ['GENDER', 'Car_Owner', 'Propert_Owner', 'Type_Income', 'EDUCATION', 'Marital_status', 'H

encoder = LabelEncoder()

for column in columns:
    if column in cc.columns:
        cc[column] = encoder.fit_transform(cc[column])
print(cc)
```

```
       Ind_ID  GENDER  Car_Owner  Propert_Owner  CHILDREN  Annual_income  \
0     5008827       1          1              1         0   180000.00000
1     5009744       0          1              0         0   315000.00000
2     5009746       0          1              0         0   315000.00000
3     5009749       0          1              0         0   191399.32623
4     5009752       0          1              0         0   315000.00000
...       ...     ...        ...            ...       ...            ...
1543  5028645       0          0              1         0   191399.32623
1544  5023655       0          0              0         0   225000.00000
1545  5115992       1          1              1         2   180000.00000
1546  5118219       1          1              0         0   270000.00000
1547  5053790       0          1              1         0   225000.00000

      Type_Income  EDUCATION  Marital_status  Housing_type  Birthday_count  \
0               1          1               1             1   -18772.000000
1               0          1               1             1   -13557.000000
2               0          1               1             1   -16040.342071
3               0          1               1             1   -13557.000000
4               0          1               1             1   -13557.000000
...           ...        ...             ...           ...             ...
1543            0          1               1             1   -11957.000000
1544            0          2               3             1   -10229.000000
1545            3          1               1             1   -13174.000000
1546            3          4               0             1   -15292.000000
1547            3          1               1             1   -16601.000000

      Employed_days  Mobile_phone  Work_Phone  Phone  EMAIL_ID  \
0            365243             1           0      0         0
1              -586             1           1      1         0
2              -586             1           1      1         0
3              -586             1           1      1         0
4              -586             1           1      1         0
...             ...           ...         ...    ...       ...
1543          -2182             1           0      0         0
1544          -1209             1           0      0         0
1545          -2477             1           0      0         0
1546           -645             1           1      1         0
1547          -2859             1           0      0         0

      Type_Occupation  Family_Members  label
0                   8               2      1
1                   8               2      1
2                   8               2      1
3                   8               2      1
4                   8               2      1
...               ...             ...    ...
1543               10               2      0
1544                0               1      0
1545               10               4      0
1546                4               2      0
1547                8               2      0

[1548 rows x 19 columns]
```

Standard Scaling

```
from sklearn.preprocessing import StandardScaler

# Assuming your dataset is stored in a variable named 'data'
# where 'data' is a 2D array or pandas DataFrame

# Extract the target variable column
target_variable = cc['label']

# Remove the target variable column from the dataset
features = cc.drop('label', axis=1)

# Create an instance of the StandardScaler class
scaler = StandardScaler()

# Apply standard scaling to the features
scaled_features = scaler.fit_transform(features)

# Create a new DataFrame or array combining the scaled features and target variable
cc = pd.DataFrame(scaled_features, columns=features.columns)
cc['label'] = target_variable
```

Test/Train Split

```
from sklearn.model_selection import train_test_split

X = cc[['GENDER', 'Car_Owner', 'Propert_Owner', 'CHILDREN', 'Type_Income', 'EDUCATION', 'Marital_st
y = cc['label']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
X_train
```

	GENDER	Car_Owner	Propert_Owner	CHILDREN	Type_Income	EDUCATION	Marital_status	Housing_type	Mobile_phone	Work_Phone	P
730	-0.761309	-0.821781	0.729845	-0.531645	0.888906	0.691398	0.576153	-0.301490	0.0	-0.512487	-0.66
100	1.313527	-0.821781	-1.370155	0.756284	0.102156	0.691398	-0.444310	0.737946	0.0	-0.512487	-0.66
619	-0.761309	-0.821781	-1.370155	0.756284	0.888906	0.691398	-1.464773	-0.301490	0.0	1.951270	-0.66
838	1.313527	-0.821781	-1.370155	-0.531645	0.102156	0.691398	-0.444310	-0.301490	0.0	-0.512487	-0.66
1419	-0.761309	-0.821781	-1.370155	-0.531645	-0.684595	0.691398	0.576153	0.737946	0.0	-0.512487	-0.66
...	...	...	...	...	...	...	...	...	...	...	
1130	-0.761309	-0.821781	-1.370155	-0.531645	-1.471346	-1.533725	-0.444310	-0.301490	0.0	1.951270	1.49
1294	1.313527	1.216869	0.729845	0.756284	0.888906	0.691398	-0.444310	-0.301490	0.0	1.951270	-0.66
860	1.313527	1.216869	-1.370155	2.044213	0.888906	0.691398	-0.444310	-0.301490	0.0	-0.512487	1.49
1459	-0.761309	-0.821781	0.729845	-0.531645	0.888906	-1.533725	-0.444310	-0.301490	0.0	-0.512487	-0.66
1126	-0.761309	1.216869	0.729845	-0.531645	0.888906	-1.533725	-0.444310	-0.301490	0.0	1.951270	1.49

1083 rows × 17 columns

```
y_train
```