Welcome to Colab!

(New) Try the Gemini API

- Generate a Gemini API key
- Talk to Gemini with the Speech-to-Text API
- Compare Gemini with ChatGPT
- More notebooks

If you're already familiar with Colab, check out this video to learn about interactive tables, the executed code history view, and the command palette.

Start coding or generate with AI.

What is Colab?

Colab, or "Colaboratory", allows you to write and execute Python in your browser, with

- Zero configuration required
- Access to GPUs free of charge
- Easy sharing

Whether you're a **student**, a **data scientist** or an **AI researcher**, Colab can make your work easier. Watch Introduction to Colab to learn more, or just get started below!

Getting started

The document you are reading is not a static web page, but an interactive environment called a **Colab notebook** that lets you write and execute code.

For example, here is a **code cell** with a short Python script that computes a value, stores it in a variable, and prints the result:

```
seconds_in_a_day = 24 * 60 * 60
seconds_in_a_day
```

```
86400
```

To execute the code in the above cell, select it with a click and then either press the play button to the left of the code, or use the keyboard shortcut "Command/Ctrl+Enter". To edit the code, just click the cell and start editing.

Variables that you define in one cell can later be used in other cells:

```
seconds_in_a_week = 7 * seconds_in_a_day
seconds_in_a_week
```

```
604800
```

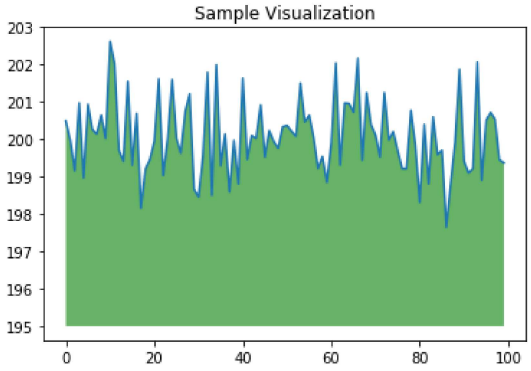
Colab notebooks allow you to combine **executable code** and **rich text** in a single document, along with **images**, **HTML**, **LaTeX** and more. When you create your own Colab notebooks, they are stored in your Google Drive account. You can easily share your Colab notebooks with co-workers or friends, allowing them to comment on your notebooks or even edit them. To learn more, see Overview of Colab. To create a new Colab notebook you can use the File menu above, or use the following link: create a new Colab notebook.

Colab notebooks are Jupyter notebooks that are hosted by Colab. To learn more about the Jupyter project, see jupyter.org.

Data science

With Colab you can harness the full power of popular Python libraries to analyze and visualize data. The code cell below uses **numpy** to generate some random data, and uses **matplotlib** to visualize it. To edit the code, just click the cell and start editing.

```
import numpy as np
from matplotlib import pyplot as plt

ys = 200 + np.random.randn(100)
x = [x for x in range(len(ys))]

plt.plot(x, ys, '-')
plt.fill_between(x, ys, 195, where=(ys > 195), facecolor='g', alpha=0.6)

plt.title("Sample Visualization")
plt.show()
```



You can import your own data into Colab notebooks from your Google Drive account, including from spreadsheets, as well as from Github and many other sources. To learn more about importing data, and how Colab can be used for data science, see the links below under Working with Data.

Machine learning

With Colab you can import an image dataset, train an image classifier on it, and evaluate the model, all in just a few lines of code. Colab notebooks execute code on Google's cloud servers, meaning you can leverage the power of Google hardware, including GPUs and TPUs, regardless of the power of your machine. All you need is a browser.

Colab is used extensively in the machine learning community with applications including:

- Getting started with TensorFlow
- Developing and training neural networks
- Experimenting with TPUs
- Disseminating AI research
- Creating tutorials

To see sample Colab notebooks that demonstrate machine learning applications, see the machine learning examples below.

More Resources

Working with Notebooks in Colab

- Overview of Colaboratory
- Guide to Markdown
- Importing libraries and installing dependencies
- Saving and loading notebooks in GitHub
- Interactive forms

- Interactive widgets

Working with Data

- Loading data: Drive, Sheets, and Google Cloud Storage
- Charts: visualizing data
- Getting started with BigQuery

Machine Learning Crash Course

These are a few of the notebooks from Google's online Machine Learning course. See the full course website for more.

- Intro to Pandas DataFrame
- Linear regression with tf.keras using synthetic data

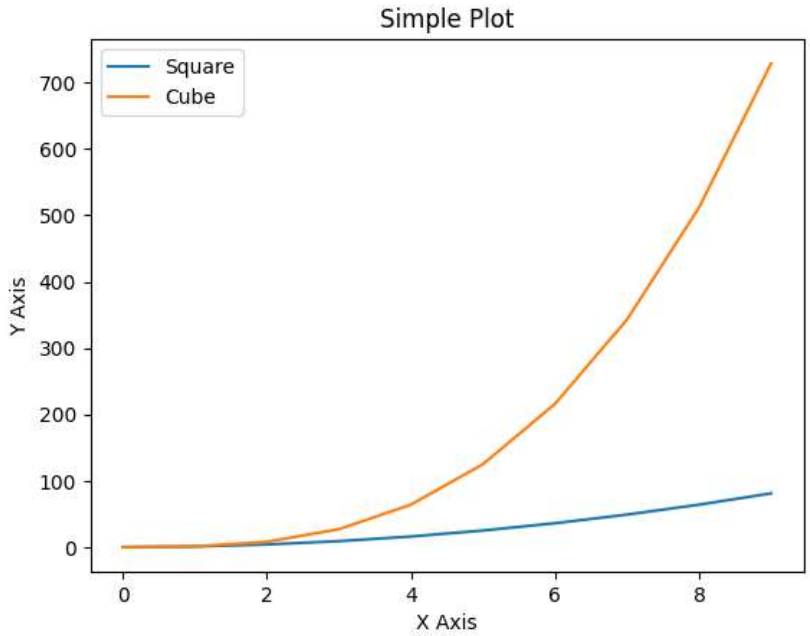
Using Accelerated Hardware

- TensorFlow with GPUs
- TensorFlow with TPUs

Featured examples

- NeMo Voice Swap: Use Nvidia's NeMo conversational AI Toolkit to swap a voice in an audio fragment with a computer generated one.
- Retraining an Image Classifier: Build a Keras model on top of a pre-trained image classifier to distinguish flowers.
- Text Classification: Classify IMDB movie reviews as either *positive* or *negative*.
- Style Transfer: Use deep learning to transfer style between images.
- Multilingual Universal Sentence Encoder Q&A: Use a machine learning model to answer questions from the SQuAD dataset.
- Video Interpolation: Predict what happened in a video between the first and the last frame.

```
# prompt:

# Create a simple plot with two lines
import matplotlib.pyplot as plt

# Create the data
x = range(10)
y = [i**2 for i in x]
z = [i**3 for i in x]

# Plot the lines
plt.plot(x, y, label="Square")
plt.plot(x, z, label="Cube")

# Add a title and axis labels
plt.title("Simple Plot")
plt.xlabel("X Axis")
plt.ylabel("Y Axis")

# Add a legend
plt.legend()

# Show the plot
plt.show()
```




```
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
```

Import the credit dataset

```
a=pd.read_csv('/Credit Card Odin.csv')
aa=a


b=pd.read_csv('/Label Credit.csv')
bb=b


aa
```

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

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

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

```
cc
```

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

1548 rows × 19 columns

Understanding and manipulating data

```
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 for null values

```
cc.isnull()
```

	Ind_ID	GENDER	Car_Owner	Propert_Owner	CHILDREN	Annual_income	Type_Income	EDUCATION	Marital_status	Housing
0	False	False	False	False	False	False	False	False	False	
1	False	False	False	False	False	False	False	False	False	
2	False	False	False	False	False	False	False	False	False	
3	False	False	False	False	False	True	False	False	False	
4	False	False	False	False	False	False	False	False	False	
...	...	...	...	...	...	...	...	...	...	
1543	False	False	False	False	False	True	False	False	False	
1544	False	False	False	False	False	False	False	False	False	
1545	False	False	False	False	False	False	False	False	False	
1546	False	False	False	False	False	False	False	False	False	
1547	False	False	False	False	False	False	False	False	False	

1548 rows × 19 columns

```
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 female so nan or blank values can be replaced by the same

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

this line of code ensures that any missing values in the 'GENDER' column of the DataFrame 'cc' are replaced with the string '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'].describe()
```

```
count    1.525000e+03
mean     1.913993e+05
std      1.132530e+05
min      3.375000e+04
25%      1.215000e+05
50%      1.665000e+05
75%      2.250000e+05
max      1.575000e+06
Name: Annual_income, dtype: float64
```

```
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.        ])
```

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

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

```
cc['Type_Occupation']
```

```
0               NaN
1               NaN
2               NaN
3               NaN
4               NaN
             ...
1543       Managers
1544    Accountants
1545       Managers
1546        Drivers
1547            NaN
Name: Type_Occupation, Length: 1548, dtype: object
```

```
cc['Type_Occupation'].count()
```

```
1060
```

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

```
array([nan, 'Core staff', 'Cooking staff', 'Laborers', '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)
```

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

```
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)
```

```
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
```

```
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
Academic degree                     2
```

```
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')
```

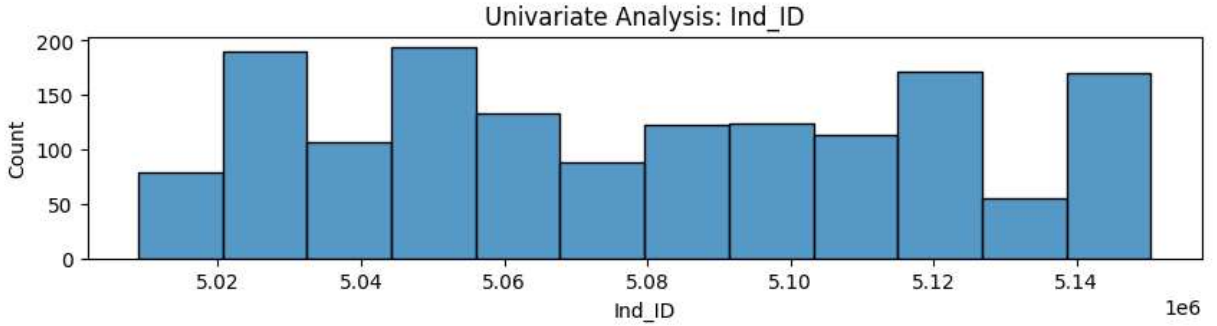
to_csv: A pandas function used to write the DataFrame to a CSV file.

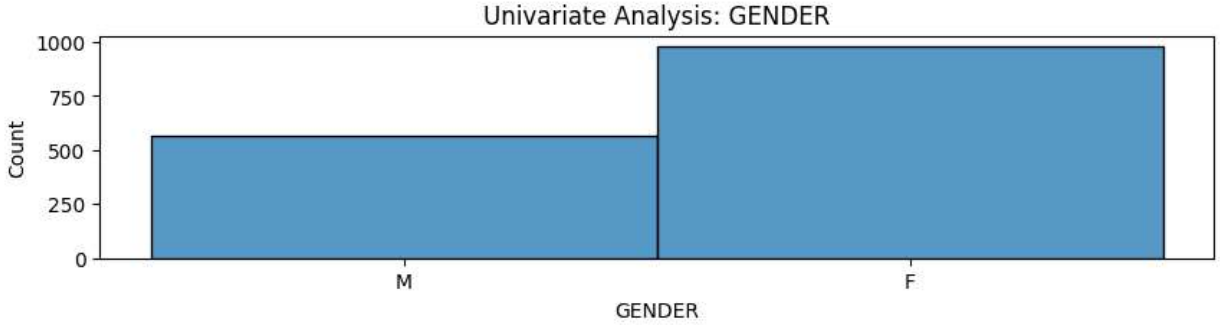
EDA Univariate Analaysis

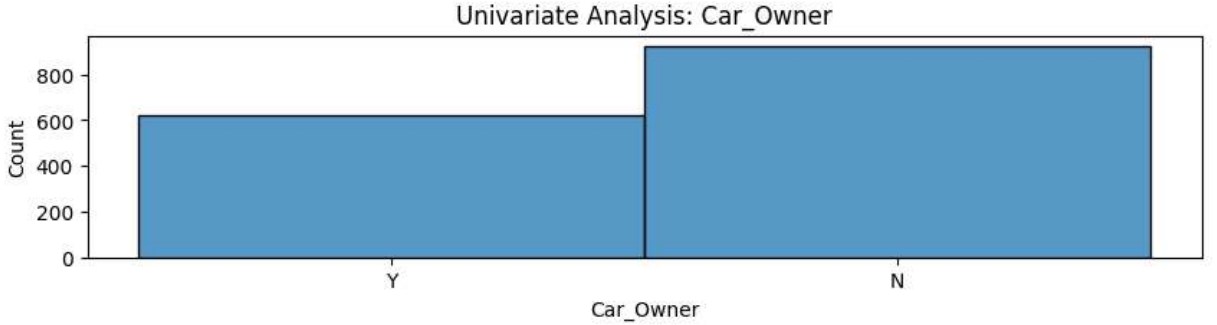
Histogram

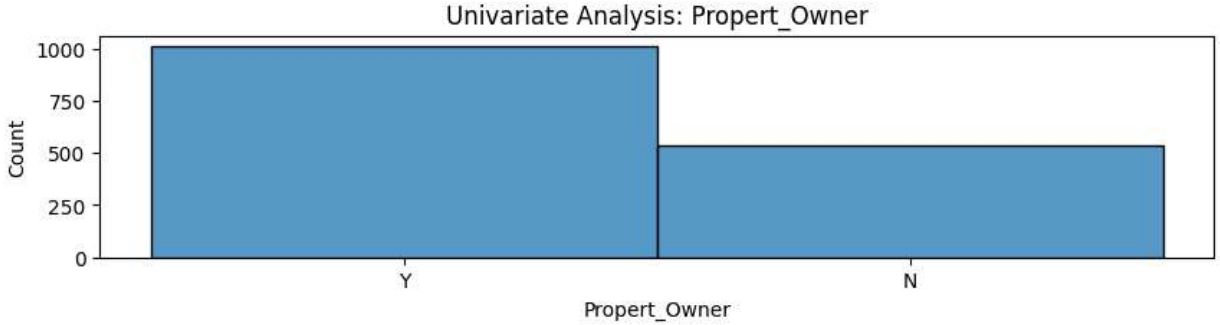
Pie Chart

```
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()
```


Univariate Analysis: Ind_ID
Count
200
150
100
50
0
5.02
5.04
5.06
5.08
5.10
5.12
5.14
Ind_ID
1e6


Univariate Analysis: GENDER
Count
1000
750
500
250
0
M
F
GENDER

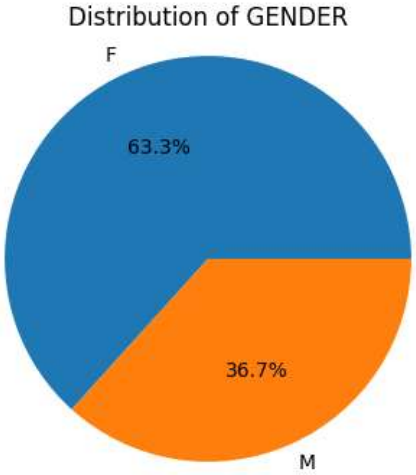

Univariate Analysis: Car_Owner
Count
800
600
400
200
0
Y
N
Car_Owner

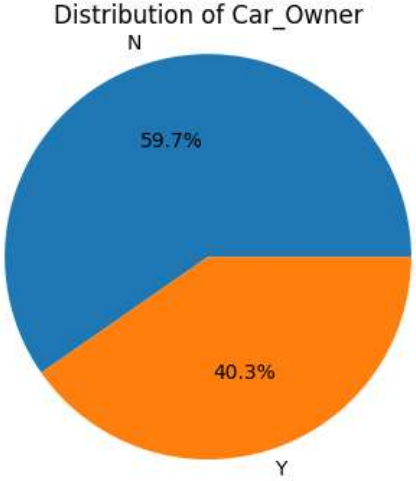

Univariate Analysis: Propert_Owner
Count
1000
750
500
250
0
Y
N
Propert_Owner

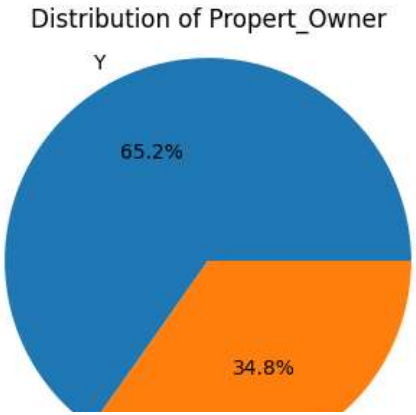

Univariate Analysis: CHILDREN

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

```
columns = ['GENDER', 'Car_Owner', 'Propert_Owner', 'CHILDREN', 'Type_Income', 'EDUCATION', 'Marital_status', 'Housing_type', 'Mobile_phone', 'Work_Phone', 'Phone', 'EMAIL_ID

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%

Insights:

1. 63.3% of the aopplicants are females and 36.7% of the applicants are males.
2. 40.3% of the applicants own car, and 59.7% of the applicants do not own car.
3. 65.2% of the applicants own property and 34.8% of the applicants do not own property.
4. 70.5% of the applicants do not have children.
5. Approximately 50% of the applicants source of income is through working.
6. Approximately 10% of the applicants do not own House/apartment.

Bivariate Analysis

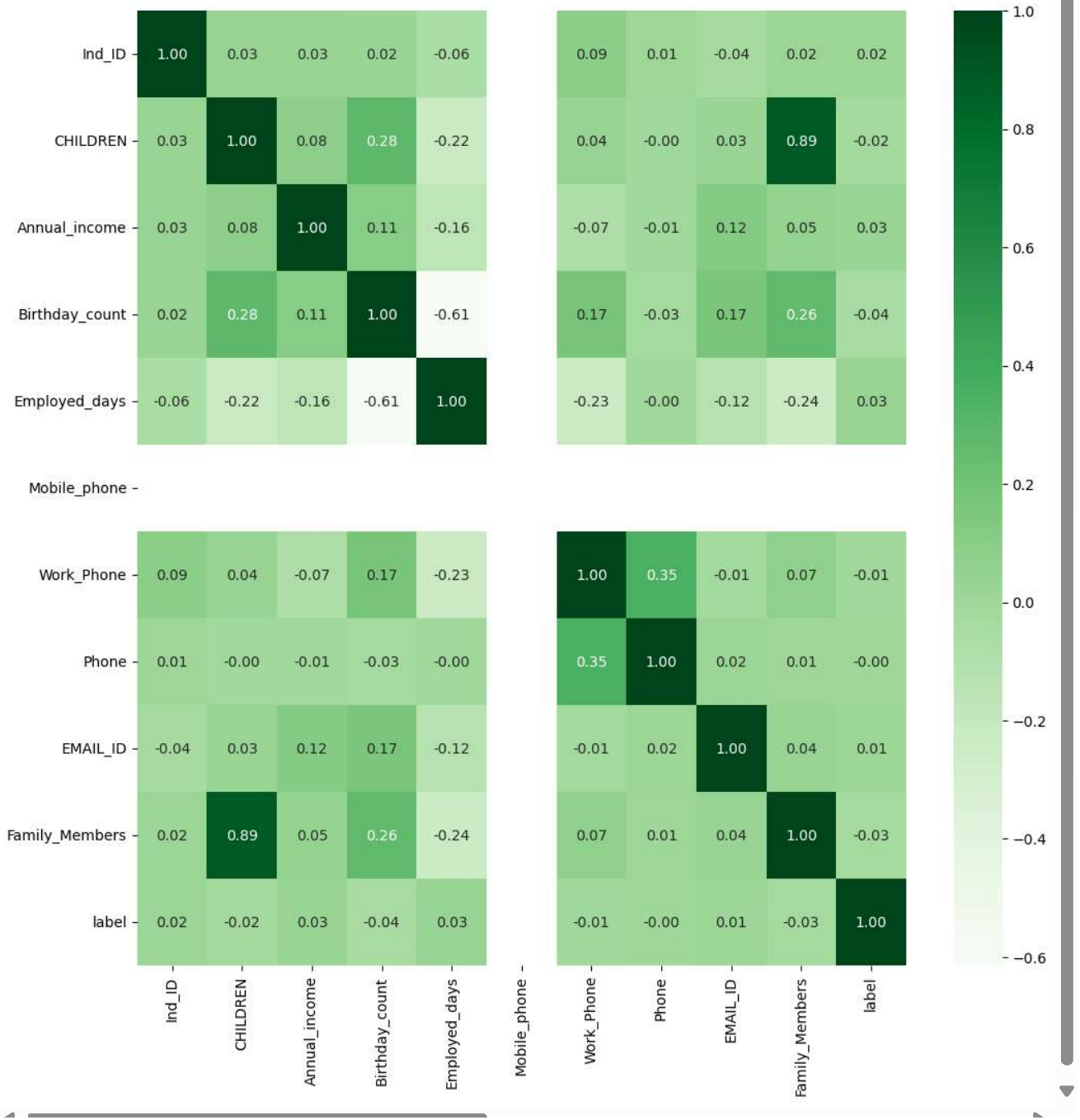
Correlation

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()
```

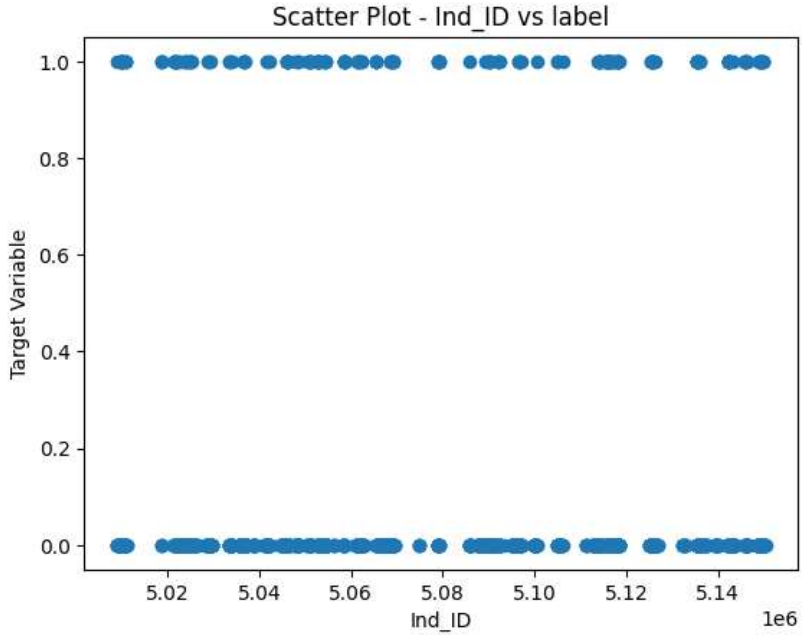
```
<ipython-input-43-8507fb91683a>:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated.
  corr_df=cc.corr()
```

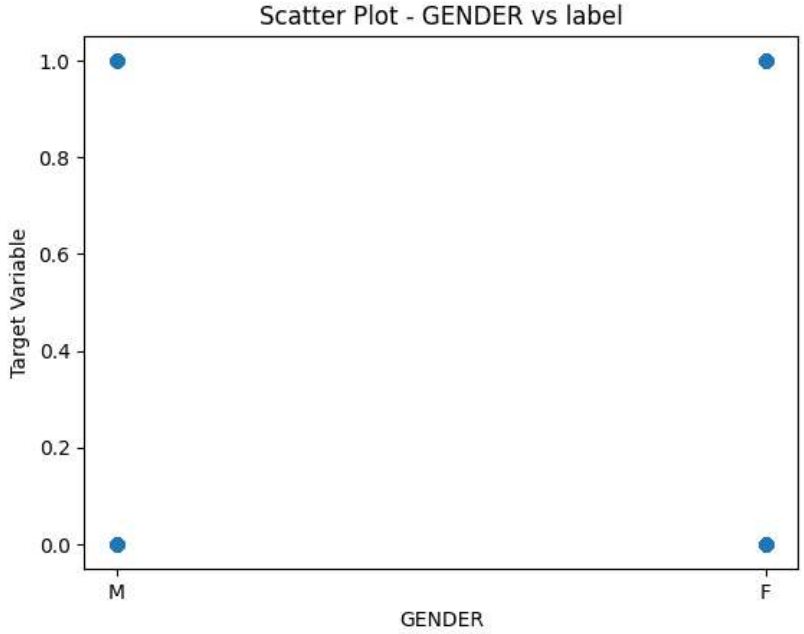


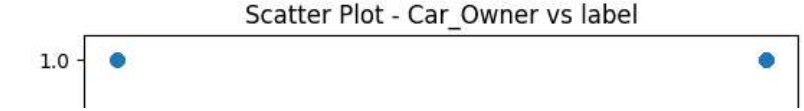
Insights:

1. The family members and children column are highly correlated.
2. 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()
```

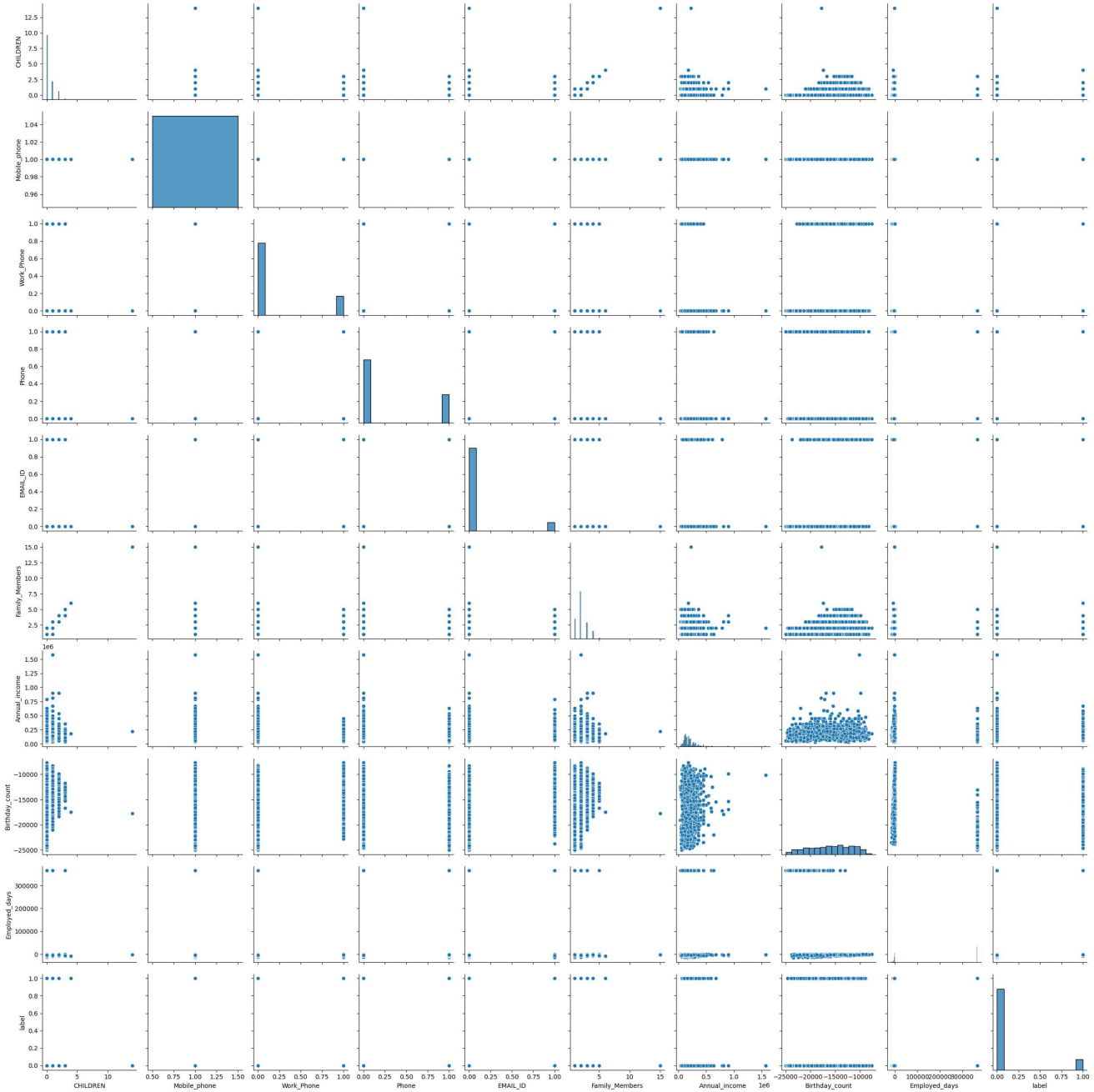

Scatter Plot - Ind_ID vs label
Target Variable
1.0
0.8
0.6
0.4
0.2
0.0
5.02
5.04
5.06
5.08
5.10
5.12
5.14
Ind_ID
1e6


Scatter Plot - GENDER vs label
Target Variable
1.0
0.8
0.6
0.4
0.2
0.0
M
F
GENDER


Scatter Plot - Car_Owner vs label
1.0

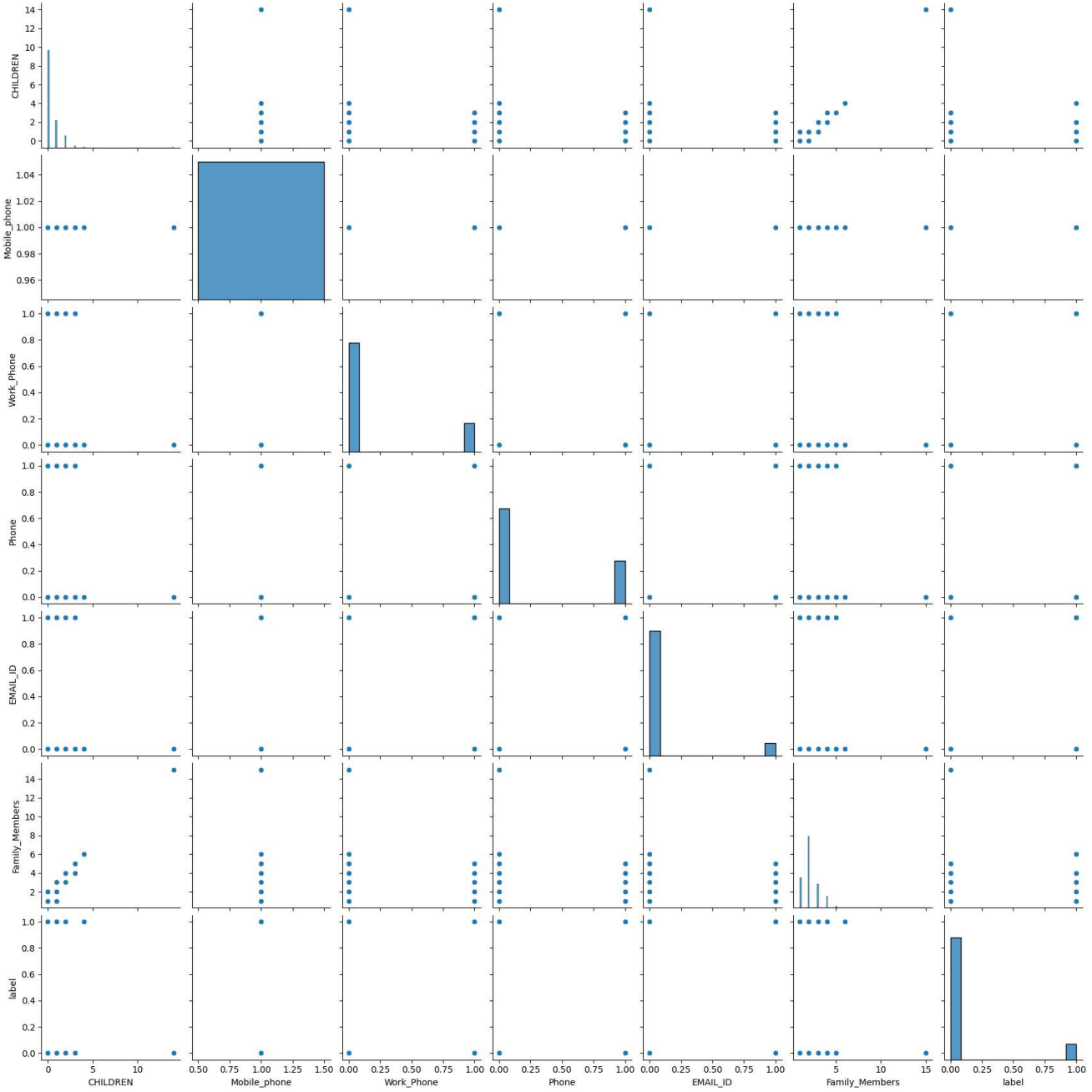
```
columns = ['GENDER', 'Car_Owner','Propert_Owner','CHILDREN','Type_Income','EDUCATION','Marital_status','Housing_type','Mobile_phone','Work_Phone','Phone','EMAIL_ID','Family_
sns.pairplot(cc[columns])
```

<seaborn.axisgrid.PairGrid at 0x7d9dd1722890>

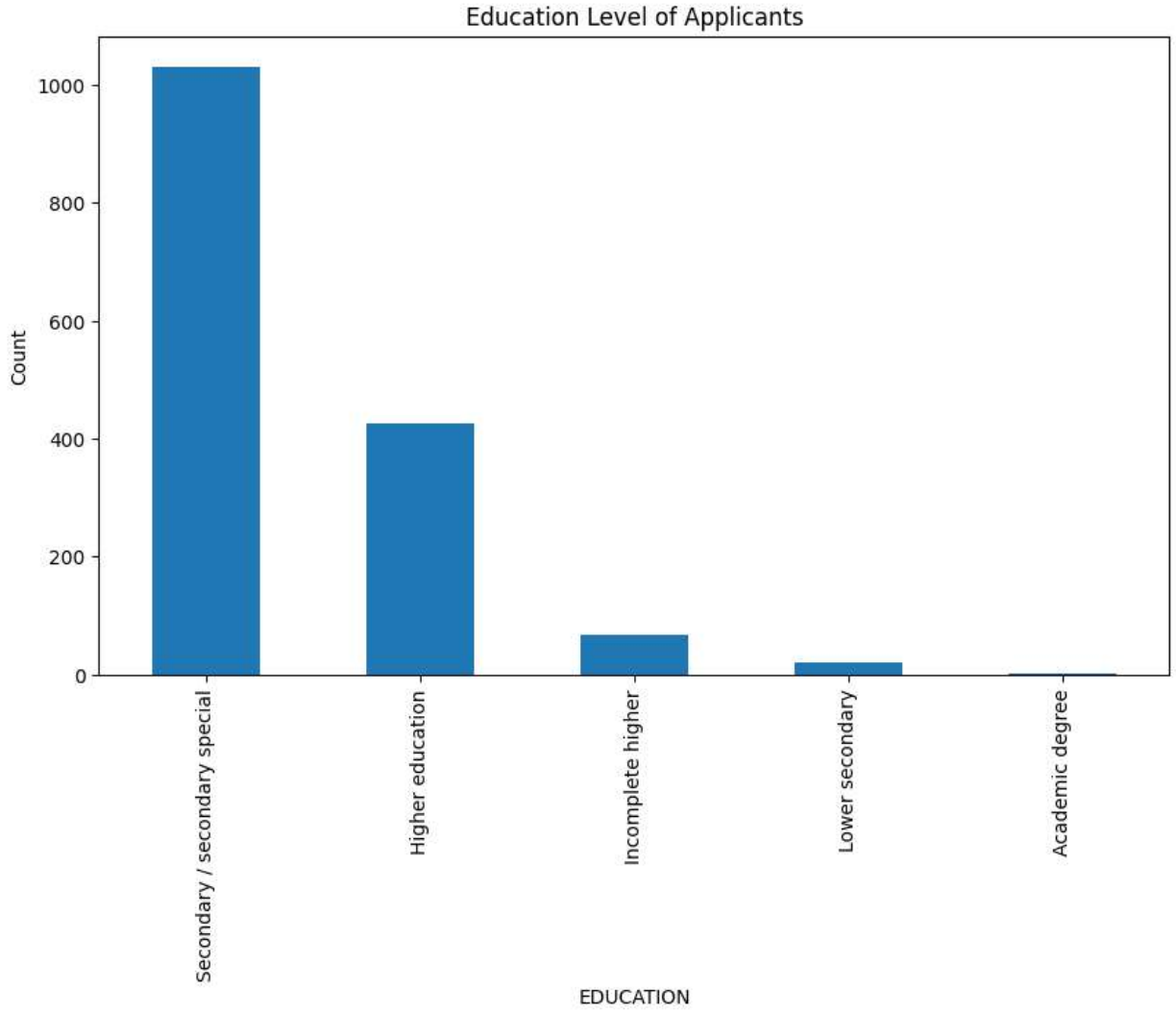


```
columns = ['GENDER', 'Car_Owner','Propert_Owner','CHILDREN','Type_Income','EDUCATION','Marital_status','Housing_type','Mobile_phone','Work_Phone','Phone','EMAIL_ID','Family_
sns.pairplot(cc[columns])
```

<seaborn.axisgrid.PairGrid at 0x7d9dc92e2da0>



```
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=90)
plt.show()
```

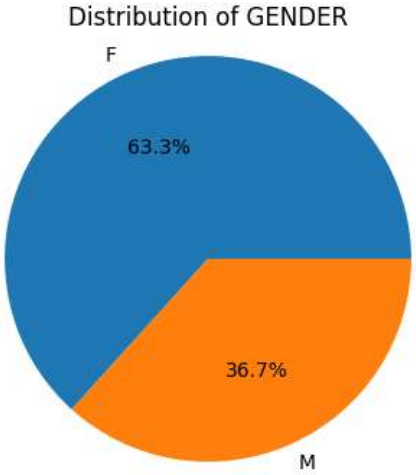


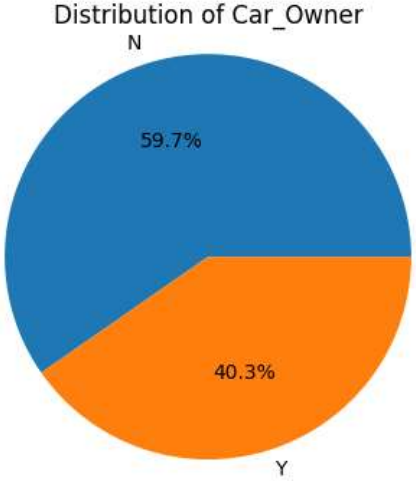
Insights :

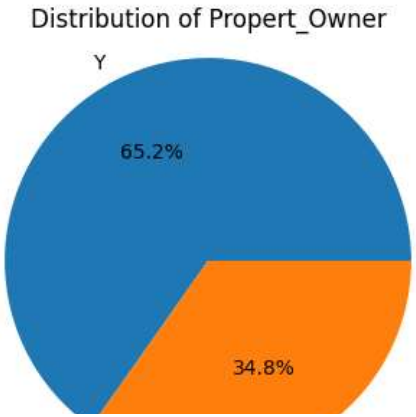
From the graph above, most of the applicants have secondary/secondary special level of Education, which is followed by Higher Education.

The remaining applicants contribution towards Education level is minimal/negligible.

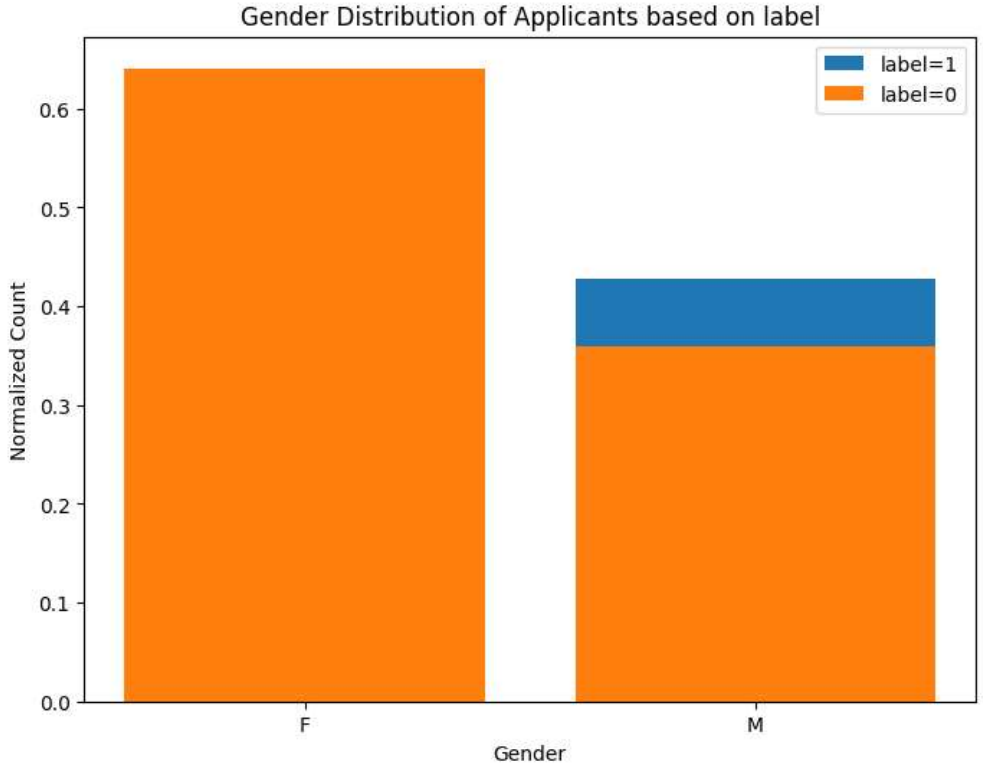
```
import pandas as pd
import matplotlib.pyplot as plt

columns = ['GENDER', 'Car_Owner', 'Propert_Owner', 'CHILDREN', 'Type_Income', 'EDUCATION', 'Marital_status', 'Housing_type', 'Mobile_phone', 'Work_Phone', 'Phone', 'EMAIL_ID

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%

```
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

```
from sklearn.preprocessing import LabelEncoder

columns = ['GENDER', 'Car_Owner', 'Propert_Owner', 'Type_Income', 'EDUCATION', 'Marital_status', 'Housing_type', 'Type_Occupation']

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]
```

```
cc.dropna(subset=['Annual_income'], inplace=True)
```

```
cc.dropna(subset=['Birthday_count'], inplace=True)
```

```
for column in cc.columns:
    if cc[column].dtype == 'object':
        print(column)
```

```
GENDER
Car_Owner
Propert_Owner
Type_Income
EDUCATION
Marital_status
Housing_type
Type_Occupation
```

```
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]
```

```
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
```

```
from sklearn.model_selection import train_test_split

X = cc[['GENDER', 'Car_Owner', 'Propert_Owner', 'CHILDREN', 'Type_Income', 'EDUCATION', 'Marital_status', 'Housing_type', 'Mobile_phone', 'Work_Phone', 'Phone', 'EMAIL_ID',
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
730	-0.761309	-0.821781	0.729845	-0.531645	0.888906	0.691398	0.576153	-0.301490	0.0
100	1.313527	-0.821781	-1.370155	0.756284	0.102156	0.691398	-0.444310	0.737946	0.0
619	-0.761309	-0.821781	-1.370155	0.756284	0.888906	0.691398	-1.464773	-0.301490	0.0
838	1.313527	-0.821781	-1.370155	-0.531645	0.102156	0.691398	-0.444310	-0.301490	0.0
1419	-0.761309	-0.821781	-1.370155	-0.531645	-0.684595	0.691398	0.576153	0.737946	0.0
...	...	...	...	...	...	...	...	...	...
1130	-0.761309	-0.821781	-1.370155	-0.531645	-1.471346	-1.533725	-0.444310	-0.301490	0.0
1294	1.313527	1.216869	0.729845	0.756284	0.888906	0.691398	-0.444310	-0.301490	0.0
860	1.313527	1.216869	-1.370155	2.044213	0.888906	0.691398	-0.444310	-0.301490	0.0
1459	-0.761309	-0.821781	0.729845	-0.531645	0.888906	-1.533725	-0.444310	-0.301490	0.0
1126	-0.761309	1.216869	0.729845	-0.531645	0.888906	-1.533725	-0.444310	-0.301490	0.0

1083 rows × 17 columns

```
y_train
```

```
730     0
100     1
619     0
838     0
1419    0
       ..
1130    0
1294    0
860     0
1459    0
1126    0
Name: label, Length: 1083, dtype: int64
```

```
X_test
```

	GENDER	Car_Owner	Propert_Owner	CHILDREN	Type_Income	EDUCATION	Marital_status	Housing_type	Mobile_phone
30	-0.761309	-0.821781	0.729845	-0.531645	0.888906	-0.050310	1.596616	2.816818	0.0
1514	-0.761309	-0.821781	-1.370155	-0.531645	-0.684595	0.691398	2.617078	0.737946	0.0
1182	-0.761309	-0.821781	-1.370155	2.044213	0.888906	-0.050310	1.596616	-0.301490	0.0
1205	-0.761309	-0.821781	0.729845	-0.531645	-0.684595	0.691398	2.617078	-0.301490	0.0
528	1.313527	-0.821781	0.729845	-0.531645	0.888906	0.691398	-1.464773	-0.301490	0.0
...	...	...	...	...	...	...	...	...	...
312	1.313527	-0.821781	0.729845	-0.531645	0.888906	0.691398	1.596616	-0.301490	0.0
978	-0.761309	1.216869	0.729845	-0.531645	0.888906	0.691398	-0.444310	-0.301490	0.0
892	-0.761309	-0.821781	-1.370155	-0.531645	-0.684595	-1.533725	-0.444310	0.737946	0.0
109	1.313527	-0.821781	-1.370155	-0.531645	0.888906	0.691398	2.617078	0.737946	0.0
952	1.313527	-0.821781	-1.370155	-0.531645	-1.471346	0.691398	-0.444310	-0.301490	0.0

465 rows × 17 columns

```
y_test
```

```
30      1
1514    0
1182    0
1205    0
528     0
       ..
312     0
978     0
892     0
109     1
952     0
Name: label, Length: 465, dtype: int64
```

```
missing_values = X_train.isnull().sum() + y_train.isnull().sum()

if missing_values.any():
    # Drop rows with missing values
    X_train = X_train.dropna()
    y_train = y_train.dropna()
```

```
y_test
```

```
30      1
1514    0
1182    0
1205    0
```

```
528    0
       ..
312    0
978    0
892    0
109    1
952    0
Name: label, Length: 465, dtype: int64
```

Machine Learning

Logistic Regression

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

```
y_train = y_train.apply(lambda x: 0 if x == 'F' else 1).astype(int)
```

```
# Create an instance of the LogisticRegression class
logistic_model = LogisticRegression()
```

```
logistic_model.fit(X_train,y_train)
```

▾ LogisticRegression
LogisticRegression()

KNN Classifier

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.ensemble import HistGradientBoostingClassifier
```

```
# Create an instance of the KNeighborsClassifier class
knn_model = KNeighborsClassifier(n_neighbors=5)

# Train the KNN model on the training data
knn_model.fit(X_train, y_train)
```

```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

Decision Tree Classifier

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score


tree_model = DecisionTreeClassifier()
```

```
tree_model.fit(X_train,y_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score


# Create an instance of the RandomForestClassifier class
forest_model = RandomForestClassifier()

# Train the random forest model on the training data
forest_model.fit(X_train, y_train)
```

```
▾ RandomForestClassifier
RandomForestClassifier()
```

SVM Alogorithm

```
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

```
# Create an instance of the SVC class
svm_model = SVC()

# Train the SVM model on the training data
svm_model.fit(X_train, y_train)
```

SVC
SVC()

Prediction

Logistic Regression

```
lr_pred = logistic_model.predict(X_test)
```

KNN PRediction

```
knn_pred = knn_model.predict(X_test)
```

Decision Tree

```
dt_pred = tree_model.predict(X_test)
```

Random Forest

```
random_pred = forest_model.predict(X_test)
```

SVM

```
svm_pred = svm_model.predict(X_test)
```

Evaluating the Prediction

Logistic Regression

```
accuracy_lr = accuracy_score(y_test, lr_pred)
print("Accuracy:", accuracy_lr)
```

```
Accuracy: 0.9075268817204301
```

```
from sklearn.metrics import precision_score, confusion_matrix, classification_report
```

```
lr_precision = precision_score(y_test, lr_pred,zero_division=0)
print("Precision Score",lr_precision)
```

```
Precision Score 0.0
```

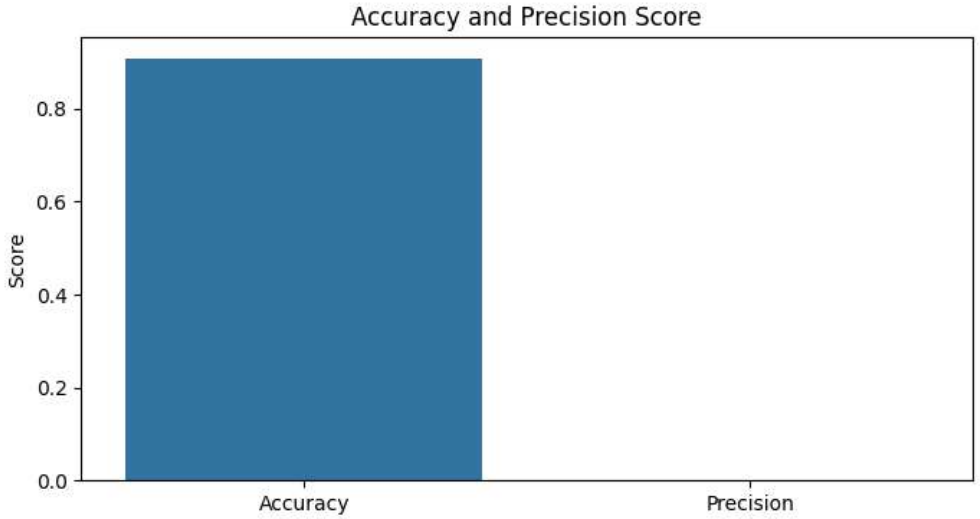
```
lr_confusion_matrix = confusion_matrix(y_test, lr_pred)
print("Score Of Confusion Matrix",lr_confusion_matrix)
```

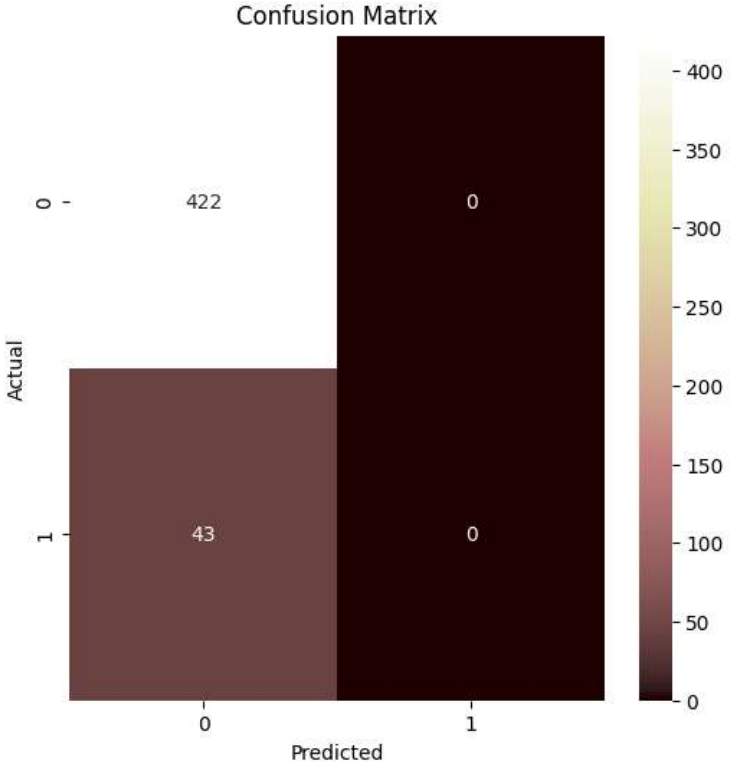
```
Score Of Confusion Matrix [[422   0]
 [ 43   0]]
```

```
lr_classification_report = classification_report(y_test, lr_pred,zero_division=0)
print("Score Of Classification report",lr_classification_report)
```

```
Score Of Classification report               precision    recall  f1-score   support

           0       0.91      1.00      0.95       422
           1       0.00      0.00      0.00        43

    accuracy                           0.91       465
   macro avg       0.45      0.50      0.48       465
weighted avg       0.82      0.91      0.86       465
```

```
plt.figure(figsize=(8, 4))
scores = {'Accuracy': accuracy_lr, 'Precision': lr_precision}
sns.barplot(x=list(scores.keys()), y=list(scores.values()))
plt.title('Accuracy and Precision Score')
plt.ylabel('Score')
plt.show()

plt.figure(figsize=(6, 6))
sns.heatmap(lr_confusion_matrix, annot=True, cmap='pink', fmt='g')
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

print("Classification Report:")
print(lr_classification_report)
```






```
Classification Report:
              precision    recall  f1-score   support

           0       0.91      1.00      0.95       422
```

```
accuracy_knn = accuracy_score(y_test, knn_pred)
print("Accuracy:", accuracy_knn)
```

```
Accuracy: 0.8924731182795699
```

```
knn_precision = precision_score(y_test, knn_pred,zero_division=0)
print("Precision Score",knn_precision)
```

```
Precision Score 0.18181818181818182
```

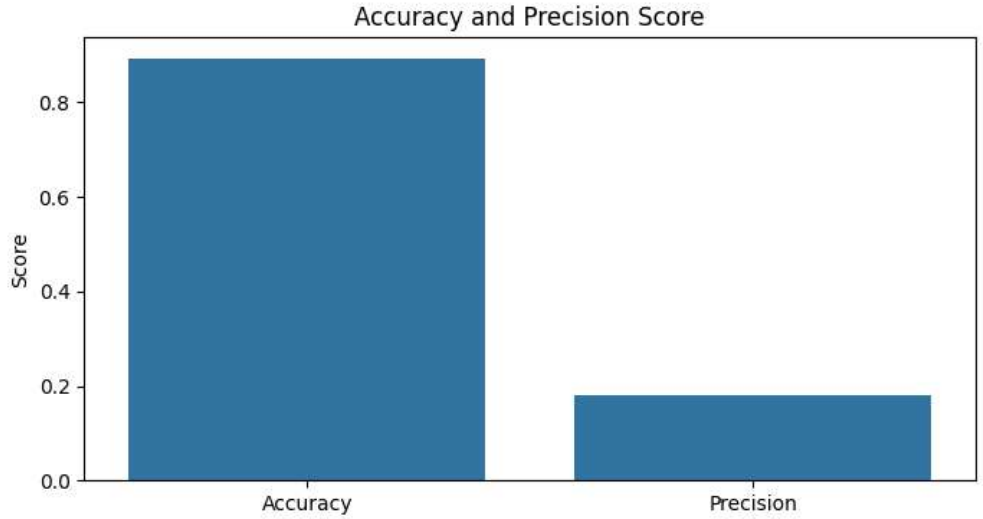
```
knn_confusion_matrix = confusion_matrix(y_test, knn_pred)
print("Score Of Confusion Matrix",knn_confusion_matrix)
```

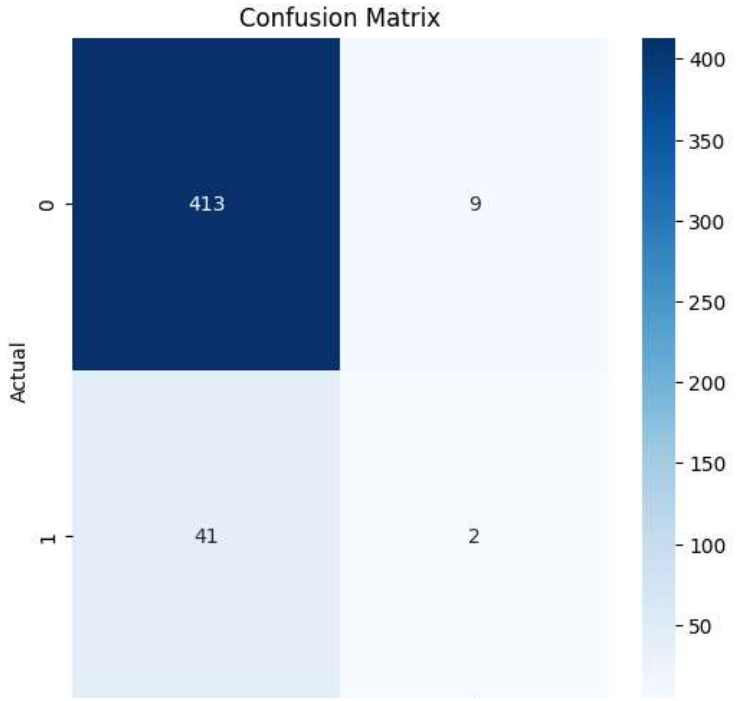
```
Score Of Confusion Matrix [[413   9]
 [ 41   2]]
```

```
knn_classification_report = classification_report(y_test, knn_pred,zero_division=0)
print("Score Of Classification report",knn_classification_report)
```

```
Score Of Classification report               precision    recall  f1-score   support

           0       0.91      0.98      0.94       422
           1       0.18      0.05      0.07        43

    accuracy                           0.89       465
   macro avg       0.55      0.51      0.51       465
weighted avg       0.84      0.89      0.86       465
```

```
plt.figure(figsize=(8, 4))
scores = {'Accuracy': accuracy_knn, 'Precision': knn_precision}
sns.barplot(x=list(scores.keys()), y=list(scores.values()))
plt.title('Accuracy and Precision Score')
plt.ylabel('Score')
plt.show()

plt.figure(figsize=(6, 6))
sns.heatmap(knn_confusion_matrix, annot=True, cmap='Blues', fmt='g')
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

print("Classification Report:")
print(knn_classification_report)
```





#DEcision tree