```
#download the dataset
import pandas as pd
df=pd.read_csv("/content/Credit_card.csv")
```

```
df.head()
```

	Ind_ID	GENDER	Car_Owner	Propert_Owner	CHILDREN	Annual_income	Type_Income	EDUCATION	Marital_status	Housing_type	Birthda
0	5008827	M	Y	Y	0	180000.0	Pensioner	Higher education	Married	House / apartment	
1	5009744	F	Y	N	0	315000.0	Commercial associate	Higher education	Married	House / apartment	
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	
4	5009752	F	Y	N	0	315000.0	Commercial associate	Higher education	Married	House / apartment	

Double-click (or enter) to edit

```
df.shape
```

```
(1548, 18)
```

```
#dchecking missing values
df.isnull().sum()
```

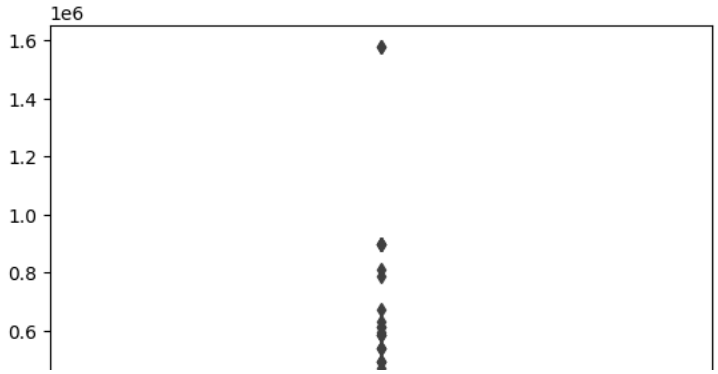
```
Ind_ID               0
GENDER               7
Car_Owner            0
Propert_Owner        0
CHILDREN             0
Annual_income       23
Type_Income          0
EDUCATION            0
Marital_status       0
Housing_type         0
Birthday_count      22
Employed_days        0
Mobile_phone        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

```
#Understand the business problem
# Data understanding(data exploration cleaning missing value outliers)
#EDA
#Feature engineering
#model training
#model evaluation
#hyper parameter tuning
#model deployment-dont have to do it
#conclusion-
```

https://scikit-learn.org/stable/auto_examples/feature_selection/plot_rfe_digits.html#sphx-glr-auto-examples-feature-selection-plot-rfe-digits-py

```
#checking for outliers
import seaborn as sns
sns.boxplot(df.Annual_income)
```

```
<Axes: >
```




```
sns.distplot(df.Annual_income)
```

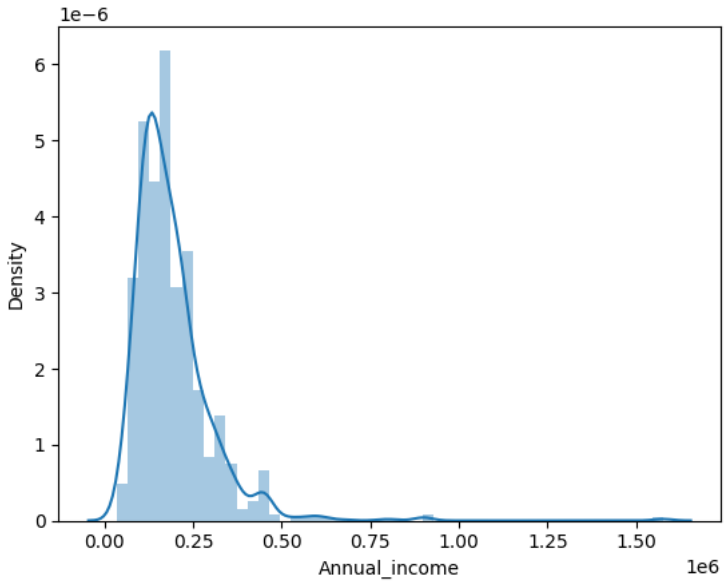
```
<ipython-input-11-468eb1990966>:1: UserWarning:

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




```
df['Annual_income'].fillna(df['Annual_income'].mean(), inplace = True)
df['Annual_income']
```

```
0       180000.00000
1       315000.00000
2       315000.00000
3       191399.32623
4       315000.00000
            ...
1543    191399.32623
1544    225000.00000
1545    180000.00000
1546    270000.00000
1547    225000.00000
Name: Annual_income, Length: 1548, dtype: float64
```

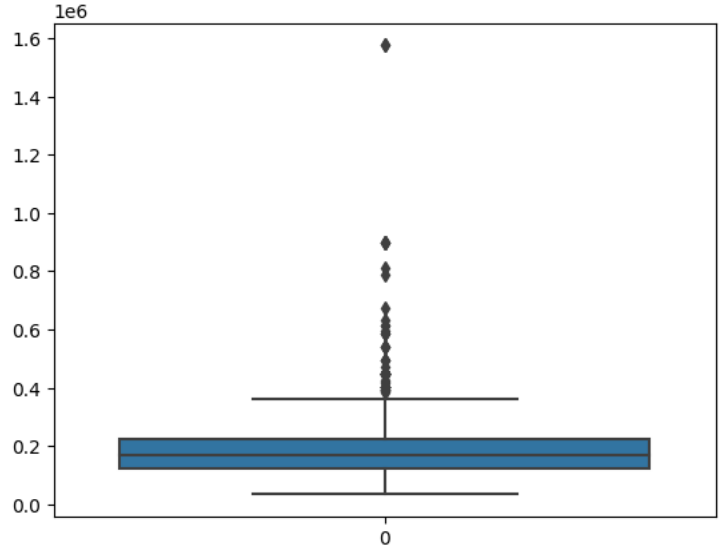
```
df.isnull().sum()
```

```
Ind_ID              0
GENDER              7
Car_Owner           0
Propert_Owner       0
CHILDREN            0
Annual_income       0
Type_Income         0
EDUCATION           0
Marital_status      0
Housing_type        0
Birthday_count     22
```

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

```
sns.boxplot(df.Annual_income)
```

```
<Axes: >
```




```
df['GENDER'] = df['GENDER'].fillna(df['GENDER'].mode()[0])
```

```
df.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      22
Employed_days        0
Mobile_phone         0
Work_Phone           0
Phone                0
EMAIL_ID             0
Type_Occupation    488
Family_Members       0
dtype: int64
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1548 entries, 0 to 1547
Data columns (total 18 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Ind_ID           1548 non-null   int64  
 1   GENDER           1548 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    1548 non-null   float64
 6   Type_Income      1548 non-null   object 
 7   EDUCATION        1548 non-null   object 
 8   Marital_status   1548 non-null   object 
 9   Housing_type     1548 non-null   object 
 10  Birthday_count   1526 non-null   float64
 11  Employed_days    1548 non-null   int64  
 12  Mobile_phone     1548 non-null   int64  
 13  Work_Phone       1548 non-null   int64  
 14  Phone            1548 non-null   int64  
 15  EMAIL_ID         1548 non-null   int64
```

```
 16  Type_Occupation  1060 non-null   object
 17  Family_Members   1548 non-null   int64
dtypes: float64(2), int64(8), object(8)
memory usage: 217.8+ KB
```

```
df['Type_Occupation'] = df['Type_Occupation'].fillna(df['Type_Occupation'].mode()[0])
```

```
df.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     22
Employed_days       0
Mobile_phone        0
Work_Phone          0
Phone               0
EMAIL_ID            0
Type_Occupation     0
Family_Members      0
dtype: int64
```

```
df['Birthday_count'].fillna(df['Birthday_count'].median(), inplace = True)
df['Birthday_count']
```

```
0      -18772.0
1      -13557.0
2      -15661.5
3      -13557.0
4      -13557.0
         ...
1543   -11957.0
1544   -10229.0
1545   -13174.0
1546   -15292.0
1547   -16601.0
Name: Birthday_count, Length: 1548, dtype: float64
```

```
df.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
dtype: int64
```

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

```
count    1.548000e+03
mean     1.913993e+05
std      1.124080e+05
min      3.375000e+04
25%      1.215000e+05
50%      1.710000e+05
75%      2.250000e+05
max      1.575000e+06
Name: Annual_income, dtype: float64
```

```
#Handling outliers
q1 = df['Annual_income'].quantile(0.25)
q3 = df['Annual_income'].quantile(0.75)
```

```
iqr=q3-q1
iqr
```

```
103500.0
```

```
upperlimit=q3+1.5*iqr
upperlimit
```

```
380250.0
```

```
lowerlimit=q1-1.5*iqr
lowerlimit
```

```
-33750.0
```

```
df[df['Annual_income'] > upperlimit]
```

	Ind_ID	GENDER	Car_Owner	Propert_Owner	CHILDREN	Annual_income	Type_Income	EDUCATION	Marital_status	Housing_type	Birt
8	5010864	M	Y	Y	1	450000.0	Commercial associate	Secondary / secondary special	Married	House / apartment	
9	5010868	M	Y	Y	1	450000.0	Pensioner	Secondary / secondary special	Married	House / apartment	
10	5010869	M	Y	Y	1	450000.0	Commercial associate	Secondary / secondary special	Single / not married	House / apartment	
14	5021303	M	N	N	1	472500.0	Pensioner	Higher education	Married	With parents	
25	5024213	F	Y	Y	0	540000.0	Commercial associate	Higher education	Married	House / apartment	
...	...	...	...	...	...	...	...	...	...	...	
1457	5095423	M	Y	Y	0	405000.0	Working	Higher education	Married	House / apartment	
1467	5113401	M	Y	Y	0	450000.0	Commercial associate	Higher education	Single / not married	House / apartment	
1479	5126562	F	N	N	0	450000.0	Working	Higher education	Married	House / apartment	
1495	5090302	F	N	Y	0	405000.0	Commercial associate	Secondary / secondary special	Married	House / apartment	
1538	5125816	F	Y	N	0	450000.0	Pensioner	Higher education	Married	House / apartment	

73 rows × 18 columns

```
df[df['Annual_income'] > upperlimit].count()
```

```
Ind_ID             73
GENDER             73
Car_Owner          73
Propert_Owner      73
CHILDREN           73
Annual_income      73
Type_Income        73
EDUCATION          73
Marital_status     73
Housing_type       73
Birthday_count     73
Employed_days      73
Mobile_phone       73
Work_Phone         73
Phone              73
EMAIL_ID           73
Type_Occupation    73
Family_Members     73
dtype: int64
```

```
df.shape
```

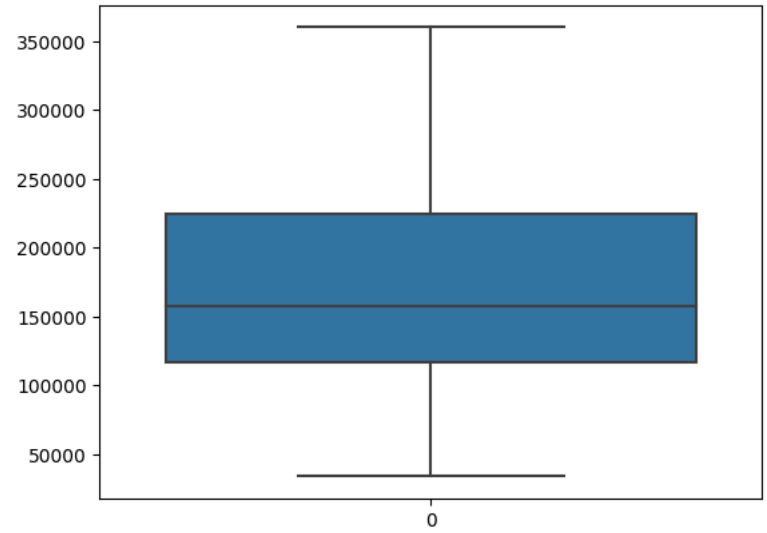
```
(1548, 18)
```

```
df1 = df[df['Annual_income'] < upperlimit]
df1.shape
```

```
(1475, 18)
```

```
sns.boxplot(df1.Annual_income)
```

```
<Axes: >
```




```
df1.columns
```

```
Index(['Ind_ID', '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', 'Type_Occupation', 'Family_Members'],
      dtype='object')
```

```
df1.drop(['Ind_ID','Mobile_phone','Work_Phone','Phone','EMAIL_ID'], axis=1, inplace=True)
```

```
<ipython-input-33-8054d7b5cdac>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  df1.drop(['Ind_ID','Mobile_phone','Work_Phone','Phone','EMAIL_ID'], axis=1, inplace=True)
```

```
df1.columns
```

```
Index(['GENDER', 'Car_Owner', 'Propert_Owner', 'CHILDREN', 'Annual_income',
       'Type_Income', 'EDUCATION', 'Marital_status', 'Housing_type',
       'Birthday_count', 'Employed_days', 'Type_Occupation', 'Family_Members'],
      dtype='object')
```

```
#feature selection
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
```

```
for col in df1.columns:
    if df1[col].dtype == 'object':
        df1[col] = le.fit_transform(df1[col])
```

```
<ipython-input-36-5241e37006ae>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  df1[col] = le.fit_transform(df1[col])
<ipython-input-36-5241e37006ae>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
```

```
  df1[col] = le.fit_transform(df1[col])
<ipython-input-36-5241e37006ae>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  df1[col] = le.fit_transform(df1[col])
<ipython-input-36-5241e37006ae>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  df1[col] = le.fit_transform(df1[col])
<ipython-input-36-5241e37006ae>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  df1[col] = le.fit_transform(df1[col])
<ipython-input-36-5241e37006ae>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  df1[col] = le.fit_transform(df1[col])
<ipython-input-36-5241e37006ae>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  df1[col] = le.fit_transform(df1[col])
<ipython-input-36-5241e37006ae>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  df1[col] = le.fit_transform(df1[col])
```

```
df1.drop_duplicates(inplace=True)
```

```
<ipython-input-37-1788250b656d>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  df1.drop_duplicates(inplace=True)
```

```
X = df1.iloc[:, :-1]
y = df1.iloc[:,-1]
```

```
#Applying ML  model to check the accuracy
from sklearn.model_selection import KFold,StratifiedKFold,train_test_split
kfold = StratifiedKFold(n_splits=8, shuffle=True, random_state=0)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

```
for train_index, test_index in kfold.split(X,y):
    # Split the data into train and test sets
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_split.py:700: UserWarning: The least populated class in y has only
  warnings.warn(
```

```
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(class_weight='balanced',max_depth=48,splitter='best',random_state=42,min_samples_s
classifier.fit(X_train, y_train)
```

```
▾                    DecisionTreeClassifier
DecisionTreeClassifier(class_weight='balanced', max_depth=48,
                       min_samples_split=48, random_state=42)
```

```
y_pred = classifier.predict(X_test)
```

```
y_pred_train= classifier.predict(X_train)
```

```
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score
```

```
acc = accuracy_score(y_test,y_pred)
confusion_mat = confusion_matrix(y_test,y_pred)



specificity_test = confusion_mat[0,0] / (confusion_mat[0,0] + confusion_mat[0,1])


print(f'Accuracy Score = {acc}\n Confusion Matrix = {confusion_mat}\n Specificity Test = {specificity_test}')
```

```
Accuracy Score = 0.9939024390243902
 Confusion Matrix = [[34  0  0  0  0]
 [ 0 86  0  0  0]
 [ 0  1 27  0  0]
 [ 0  0  0 14  0]
 [ 0  0  0  0  2]]
 Specificity Test = 1.0
```

```
acc_train = accuracy_score(y_train,y_pred_train)
confusion_mat_train = confusion_matrix(y_train,y_pred_train)

specificity_train = confusion_mat_train[0,0] / (confusion_mat_train[0,0] + confusion_mat_train[0,1])


print(f'Accuracy Score = {acc_train}\n Confusion Matrix = {confusion_mat_train}\n Specificity Test = {specificity_train}')
```

```
Accuracy Score = 0.9878260869565217
 Confusion Matrix = [[242   0   1   0   0   0   0]
 [  7 596   0   0   0   0   0]
 [  0   3 189   0   0   0   0]
 [  0   0   1  95   1   0   0]
 [  0   0   0   0  13   0   0]
 [  0   0   0   0   0   1   0]
 [  0   0   0   0   0   1   0]]
 Specificity Test = 1.0
```

```
import sqlite3



conn=sqlite3.connect("capstone.db")


cursor=conn.cursor()


df.to_sql('creditcardInfo', conn, if_exists='replace', index=False)
```

```
1548
```

```
df.columns
```

```
Index(['Ind_ID', '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', 'Type_Occupation', 'Family_Members'],
      dtype='object')
```

```
#Group the customers based on their income type and find the average of their annual income.
cursor.execute('select avg(Annual_income) from creditcardinfo group by Type_Income').fetchall()
```

```
[(233653.1359173591,),
 (155713.74648668416,),
 (211422.41379310345,),
 (181191.4343214594,)]
```

```
#Find the female owners of cars and property.
cursor.execute('select Car_owner,Propert_owner,Gender from creditcardinfo where Gender="F" and Car_Owner="Y" and Propert_owner="Y" ').fet
```

```
[('Y', 'Y', 'F'),
 ('Y', 'Y', 'F'),
 ('Y', 'Y', 'F'),
 ('Y', 'Y', 'F'),
 ('Y', 'Y', 'F'),
 ('Y', 'Y', 'F'),
 ('Y', 'Y', 'F'),
 ('Y', 'Y', 'F'),
 ('Y', 'Y', 'F'),
 ('Y', 'Y', 'F'),
 ('Y', 'Y', 'F'),
 ('Y', 'Y', 'F'),
 ('Y', 'Y', 'F'),
```

```
 ('Y', 'Y', 'F'),
 ('Y', 'Y', 'F'),
 ('Y', 'Y', 'F'),
 ('Y', 'Y', 'F'),
 ('Y', 'Y', 'F'),
 ('Y', 'Y', 'F'),
 ('Y', 'Y', 'F'),
 ('Y', 'Y', 'F'),
 ('Y', 'Y', 'F'),
 ('Y', 'Y', 'F'),
 ('Y', 'Y', 'F'),
 ('Y', 'Y', 'F'),
 ('Y', 'Y', 'F'),
 ('Y', 'Y', 'F'),
 ('Y', 'Y', 'F'),
 ('Y', 'Y', 'F'),
 ('Y', 'Y', 'F'),
 ('Y', 'Y', 'F'),
 ('Y', 'Y', 'F'),
 ('Y', 'Y', 'F'),
 ('Y', 'Y', 'F'),
 ('Y', 'Y', 'F'),
 ('Y', 'Y', 'F'),
 ('Y', 'Y', 'F'),
 ('Y', 'Y', 'F'),
 ('Y', 'Y', 'F'),
 ('Y', 'Y', 'F'),
 ('Y', 'Y', 'F'),
 ('Y', 'Y', 'F'),
 ('Y', 'Y', 'F'),
 ('Y', 'Y', 'F'),
 ('Y', 'Y', 'F'),
 ('Y', 'Y', 'F'),
 ('Y', 'Y', 'F'),
 ('Y', 'Y', 'F'),
 ('Y', 'Y', 'F'),
 ('Y', 'Y', 'F'),
 ('Y', 'Y', 'F'),
 ('Y', 'Y', 'F'),
 ('Y', 'Y', 'F'),
 ('Y', 'Y', 'F'),
 ('Y', 'Y', 'F'),
 ('Y', 'Y', 'F'),
 ('Y', 'Y', 'F'),
```

```
#Find the male customers who are staying with their families.
df1.columns
```

```
Index(['GENDER', 'Car_Owner', 'Propert_Owner', 'CHILDREN', 'Annual_income',
       'Type_Income', 'EDUCATION', 'Marital_status', 'Housing_type',
       'Birthday_count', 'Employed_days', 'Type_Occupation', 'Family_Members'],
      dtype='object')
```

```
cursor.execute('select Family_Members,Gender from creditcardinfo where Gender="M" and Family_Members>1').fetchall()
```

```
[(2, 'M'),
 (3, 'M'),
 (3, 'M'),
 (3, 'M'),
 (2, 'M'),
 (2, 'M'),
 (2, 'M'),
 (2, 'M'),
 (2, 'M'),
 (2, 'M'),
 (2, 'M'),
 (2, 'M'),
 (2, 'M'),
 (2, 'M'),
 (3, 'M'),
 (3, 'M'),
 (2, 'M'),
 (2, 'M'),
 (3, 'M'),
 (3, 'M'),
 (2, 'M'),
 (2, 'M'),
 (2, 'M'),
 (3, 'M'),
 (2, 'M'),
 (2, 'M'),
 (2, 'M'),
 (2, 'M'),
 (2, 'M'),
 (2, 'M'),
 (2, 'M'),
 (2, 'M'),
 (2, 'M'),
```

```
 (3, 'M'),
 (2, 'M'),
 (2, 'M'),
 (4, 'M'),
 (3, 'M'),
 (3, 'M'),
 (3, 'M'),
 (2, 'M'),
 (2, 'M'),
 (3, 'M'),
 (2, 'M'),
 (2, 'M'),
 (2, 'M'),
 (2, 'M'),
 (4, 'M'),
 (4, 'M'),
 (4, 'M'),
 (3, 'M'),
 (3, 'M'),
 (3, 'M'),
 (2, 'M'),
 (2, 'M'),
 (3, 'M'),
 (3, 'M'),
 (2, 'M')]
```

```
#Please list the top five people having the highest income.
df.columns
```

```
Index(['Ind_ID', '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', 'Type_Occupation', 'Family_Members'],
      dtype='object')
```

```
cursor.execute('select Ind_ID,Annual_income, dense_rank() over(order by Annual_income desc) highest from creditcardinfo limit 5').fetchal
```

```
[(5143231, 1575000.0, 1),
 (5143235, 1575000.0, 1),
 (5090470, 900000.0, 2),
 (5079016, 900000.0, 2),
 (5079017, 900000.0, 2)]
```

```
#How many married people are having bad credit?
cursor.execute('select Marital_status,Annual_income from creditcardinfo where Marital_status="Married" and Annual_income<50000.0 ').fetcl
```

```
[('Married', 45000.0),
 ('Married', 36000.0),
 ('Married', 37800.0),
 ('Married', 40500.0),
 ('Married', 45000.0),
 ('Married', 44550.0),
 ('Married', 33750.0),
 ('Married', 40500.0),
 ('Married', 45000.0),
 ('Married', 47250.0),
 ('Married', 49500.0),
 ('Married', 45000.0)]
```

```
#What is the highest education level and what is the total count?
cursor.execute('select count(Education),Education from creditcardinfo where Education="Higher education"').fetchall()
```

```
[(426, 'Higher education')]
```

Double-click (or enter) to edit

```
#Between married males and females, who is having more bad credit?

cursor.execute('select count(Gender),Gender,Annual_income from creditcardinfo where Marital_status="Married" and Annual_income<70000.0 gr
```

```
[(36, 'F', 67500.0), (4, 'M', 65250.0)]
```

```
Females are having more bad credit than males
```

Females are having more bad credit than males

Double-click (or enter) to edit