Project Proposal: Predicting Credit Card Approval

Author:

Govind G
https://github.com/govindgopidas

Section 1: Questions to Answer

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

- In today's world, where financial institutions face increasing risks and competition, accurately predicting credit card approvals is essential. By identifying creditworthy clients, banks can reduce default rates, minimize financial losses, and make informed lending decisions.

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

- Accurate credit card approval predictions can positively impact the banking sector in several ways:
    - Reducing credit risk: Banks can minimize financial losses by approving credit cards only for customers likely to repay their debts.
    - Enhancing customer experience: Efficient approval processes can attract more customers and improve overall satisfaction.
    - Increasing profitability: Targeted lending to creditworthy clients can lead to higher interest and fee income.

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

- The proposed method can address the gap in credit risk assessment by leveraging advanced data analytics and machine learning techniques. It can help banks in India and elsewhere by:
    - Providing a more accurate assessment of creditworthiness.
    - Reducing the reliance on traditional credit scores.
    - Adapting to evolving customer behaviors and economic conditions.

Section 2: Initial Hypothesis

Hypothesis:

- We hypothesize that several factors, including annual income, employment status, education level, and family size, will significantly influence credit card approval decisions.
- We assume that machine learning models will outperform traditional rule-based credit scoring systems, improving accuracy and minimizing risk.

Section 3: Data Analysis Approach

Data Analysis Approach:

- We will start with exploratory data analysis (EDA) to:
  - Identify correlations between features and the target variable (credit card approval).
  - Visualize the distribution of key variables.
  - Detects and handles missing or outlier data.
- We will perform feature engineering to create new informative features and encode categorical variables.
- To justify our findings, we will create data visualizations, such as histograms, scatter plots, and correlation matrices.

3.1 Business problem understanding

1. Business Problem Understanding

The business problem addressed here is how to accurately predict credit card approval decisions for potential customers. This problem is crucial for banks as they need to make informed lending decisions to minimize credit risk, reduce financial losses, enhance the customer experience, and ultimately increase profitability. The project aims to leverage data analytics and machine learning to improve the credit assessment process and help banks make more precise and efficient lending decisions, ultimately benefiting both the financial institution and its customers.

3.2 Importing the required libraries

```python
# Importing all required libraries:
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import plotly.express as px
import seaborn as sns
```

3.3 Importing the collected dataset

```
# Importing the collected dataset:
credit_card = pd.read_csv("Credit_card.csv")
credit_card_label = pd.read_csv("Credit_card_label.csv")
```

```
credit_card.head()
```

	Ind_ID	GENDER	Car_Owner	Propert_Owner	CHILDREN	Annual_income	Type_Income	EDUCATION	Marital_status	Housing_type	Birthday_count	Employe
0	5008827	M	Y	Y	0	180000.0	Pensioner	Higher education	Married	House / apartment	-18772.0	
1	5009744	F	Y	N	0	315000.0	Commercial associate	Higher education	Married	House / apartment	-13557.0	
2	5009746	F	Y	N	0	315000.0	Commercial associate	Higher education	Married	House / apartment	NaN	
3	5009749	F	Y	N	0	NaN	Commercial associate	Higher education	Married	House / apartment	-13557.0	
4	5009752	F	Y	N	0	315000.0	Commercial associate	Higher education	Married	House / apartment	-13557.0	

3.4 Merged both the tables for EDA

```
# Merged both the tables for EDA
creditcard_df = credit_card.merge(credit_card_label, on='Ind_ID', how='inner')
creditcard_df
```

	Ind_ID	GENDER	Car_Owner	Propert_Owner	CHILDREN	Annual_income	Type_Income	EDUCATION	Marital_status	Housing_type	Birthday_count	Empl
0	5008827	M	Y	Y	0	180000.0	Pensioner	Higher education	Married	House / apartment	-18772.0	
1	5009744	F	Y	N	0	315000.0	Commercial associate	Higher education	Married	House / apartment	-13557.0	
2	5009746	F	Y	N	0	315000.0	Commercial associate	Higher education	Married	House / apartment	NaN	
3	5009749	F	Y	N	0	NaN	Commercial associate	Higher education	Married	House / apartment	-13557.0	
4	5009752	F	Y	N	0	315000.0	Commercial associate	Higher education	Married	House / apartment	-13557.0	
...	...	...	...	...	...	...	...	...	...	...	...	
1543	5028645	F	N	Y	0	NaN	Commercial associate	Higher education	Married	House / apartment	-11957.0	
1544	5023655	F	N	N	0	225000.0	Commercial associate	Incomplete higher	Single / not married	House / apartment	-10229.0	
1545	5115992	M	Y	Y	2	180000.0	Working	Higher education	Married	House / apartment	-13174.0	

3.5 Table info

```
creditcard_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1548 entries, 0 to 1547
Data columns (total 19 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Ind_ID           1548 non-null   int64  
 1   GENDER           1541 non-null   object 
 2   Car_Owner        1548 non-null   object 
 3   Propert_Owner    1548 non-null   object 
 4   CHILDREN         1548 non-null   int64  
 5   Annual_income    1525 non-null   float64
 6   Type_Income      1548 non-null   object 
 7   EDUCATION        1548 non-null   object 
 8   Marital_status   1548 non-null   object 
 9   Housing_type     1548 non-null   object 
 10  Birthday_count   1526 non-null   float64
 11  Employed_days    1548 non-null   int64  
 12  Mobile_phone     1548 non-null   int64  
 13  Work_Phone       1548 non-null   int64  
 14  Phone            1548 non-null   int64  
 15  EMAIL_ID         1548 non-null   int64  
 16  Type_Occupation  1060 non-null   object 
 17  Family_Members   1548 non-null   int64  
 18  label            1548 non-null   int64  
dtypes: float64(2), int64(9), object(8)
memory usage: 241.9+ KB
```

Descriptive Statistics

3.6.1

```
creditcard_df.describe(include='all')
```

	Ind_ID	GENDER	Car_Owner	Propert_Owner	CHILDREN	Annual_income	Type_Income	EDUCATION	Marital_status	Housing_type	Birthday_coun
count	1.548000e+03	1541	1548	1548	1548.000000	1.525000e+03	1548	1548	1548	1548	1526.00000
unique	NaN	2	2	2	NaN	NaN	4	5	5	6	Nal
top	NaN	F	N	Y	NaN	NaN	Working	Secondary / secondary special	Married	House / apartment	Nal
freq	NaN	973	924	1010	NaN	NaN	798	1031	1049	1380	Nal
mean	5.078920e+06	NaN	NaN	NaN	0.412791	1.913993e+05	NaN	NaN	NaN	NaN	-16040.34207
std	4.171759e+04	NaN	NaN	NaN	0.776691	1.132530e+05	NaN	NaN	NaN	NaN	4229.50320
min	5.008827e+06	NaN	NaN	NaN	0.000000	3.375000e+04	NaN	NaN	NaN	NaN	-24946.00000
25%	5.045070e+06	NaN	NaN	NaN	0.000000	1.215000e+05	NaN	NaN	NaN	NaN	-19553.00000
50%	5.078842e+06	NaN	NaN	NaN	0.000000	1.665000e+05	NaN	NaN	NaN	NaN	-15661.50000
75%	5.115673e+06	NaN	NaN	NaN	1.000000	2.250000e+05	NaN	NaN	NaN	NaN	-12417.00000
max	5.150412e+06	NaN	NaN	NaN	14.000000	1.575000e+06	NaN	NaN	NaN	NaN	-7705.00000

3.6.2

```
# Checking whether there is any values foreign values:
```

```
creditcard_df["Family_Members"].unique()
```

```
array([ 2,  3,  1,  4,  6,  5, 15], dtype=int64)
```

```
creditcard_df["CHILDREN"].unique()
```

```
array([ 0,  1,  2,  4,  3, 14], dtype=int64)
```

```
creditcard_df["label"].unique()
```

```
array([1, 0], dtype=int64)
```

```
creditcard_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',
       'label'],
      dtype='object')
```

3.7 Renaming columns properly

```
# Renaming columns properly:

creditcard_df.rename(columns = {
    'Ind_ID':'Id',
    'GENDER':'Gender',
    'CHILDREN':'Children',
    'Type_Income':'Income_Type',
    'EDUCATION':'Education',
    'EMAIL_ID':'Email_Id',
    'Type_Occupation':'Occupation_Type',
    'label':'Label'
}, inplace = True)

creditcard_df.head()
```

	Id	Gender	Car_Owner	Propert_Owner	Children	Annual_income	Income_Type	Education	Marital_status	Hou
0	5008827	M	Y	Y	0	180000.0	Pensioner	Higher education	Married	
1	5009744	F	Y	N	0	315000.0	Commercial	Higher	Married	

3.8 Checking for duplicate values

```
# Finding duplicates:
creditcard_df[creditcard_df.duplicated()]
```

Education	Marital_status	Housing_type	Birthday_count	Employed_days	Mobile_phone	Work_Phone	Phone	Email_Id	Occupation_Typ

- There are no duplicated values in the entire dataset

Treating missing values

3.9.1

```
# Checking for missing values:
creditcard_df.isnull().sum()
```

```
Id                  0
Gender              7
Car_Owner           0
Propert_Owner       0
Children            0
Annual_income       0
Income_Type         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
Occupation_Type   488
Family_Members      0
Label               0
dtype: int64
```

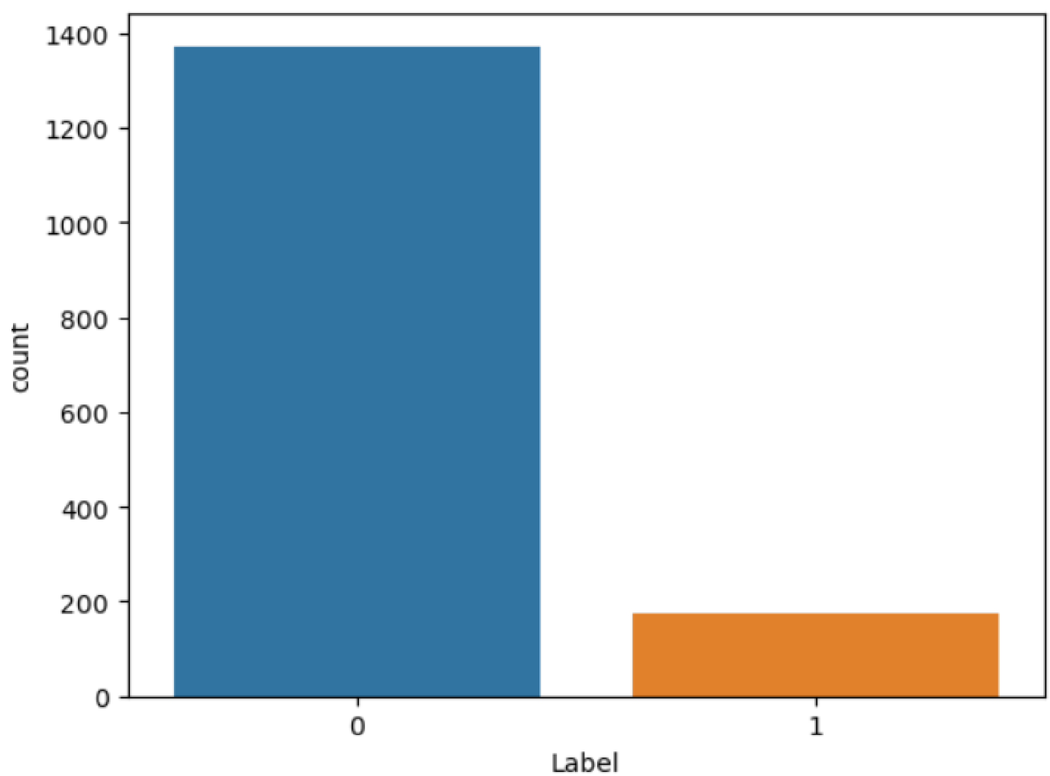
3.9.2

```
# Filling important features with median values:

median_income = creditcard_df['Annual_income'].median()
creditcard_df['Annual_income'].fillna(median_income, inplace=True)

creditcard_df['Gender'].fillna('NaN', inplace=True)
```

Data exploration using plots:

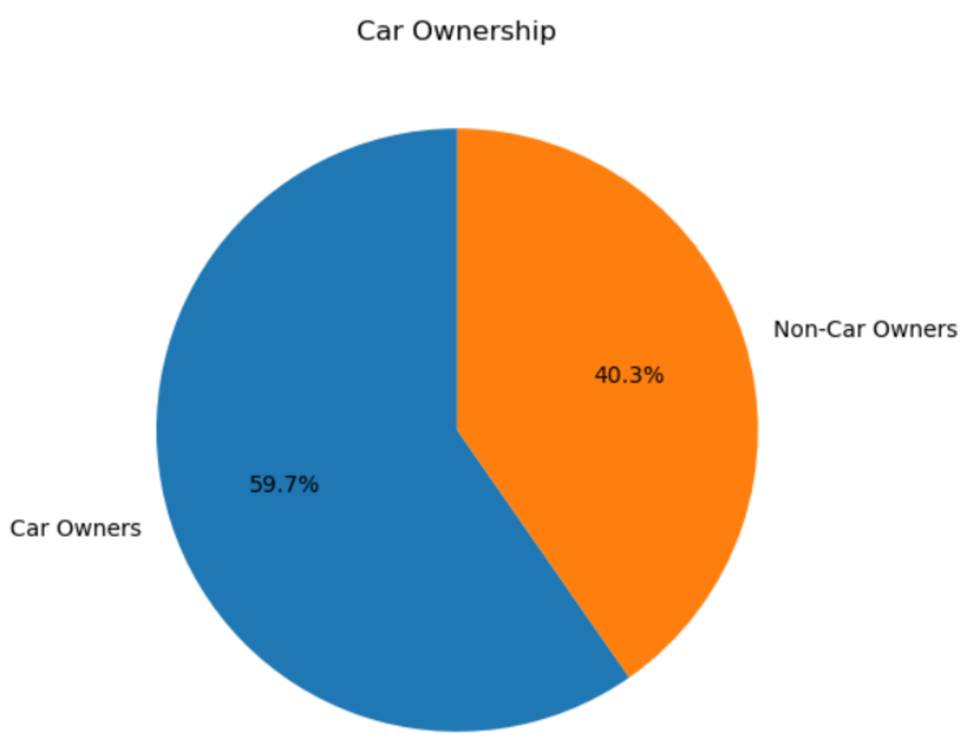
3.10.1 Count plot

```
# Comparison of applications that are accepted and rejected
import seaborn as sns
sns.countplot(creditcard_df, x="Label")
```



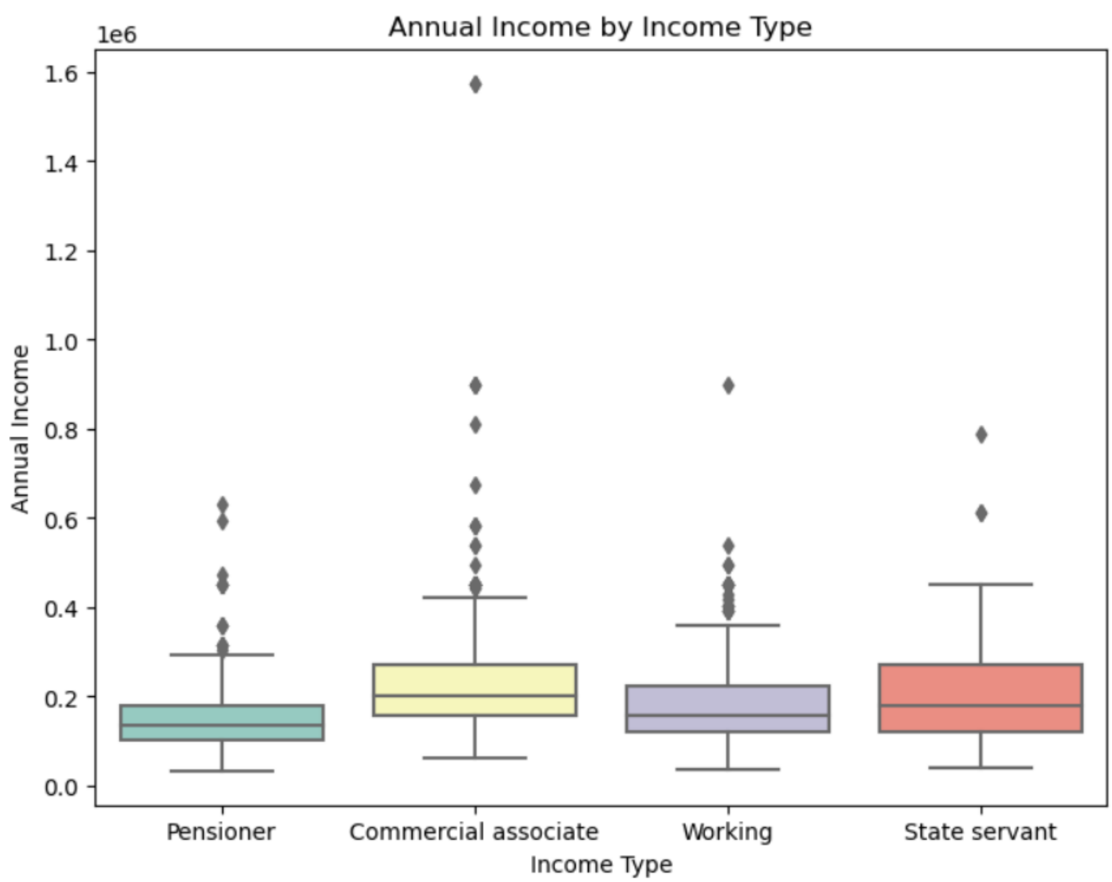
3.10.2 Pie chart

```
# Customers who owns a car and not

car_counts = creditcard_df['Car_Owner'].value_counts()
custom_labels = ['Car Owners', 'Non-Car Owners']

plt.figure(figsize=(6, 6))  # Optional: Set the figure size
plt.pie(car_counts, labels=custom_labels, autopct='%1.1f%%', startangle=90)

plt.title('Car Ownership')

plt.show()
```



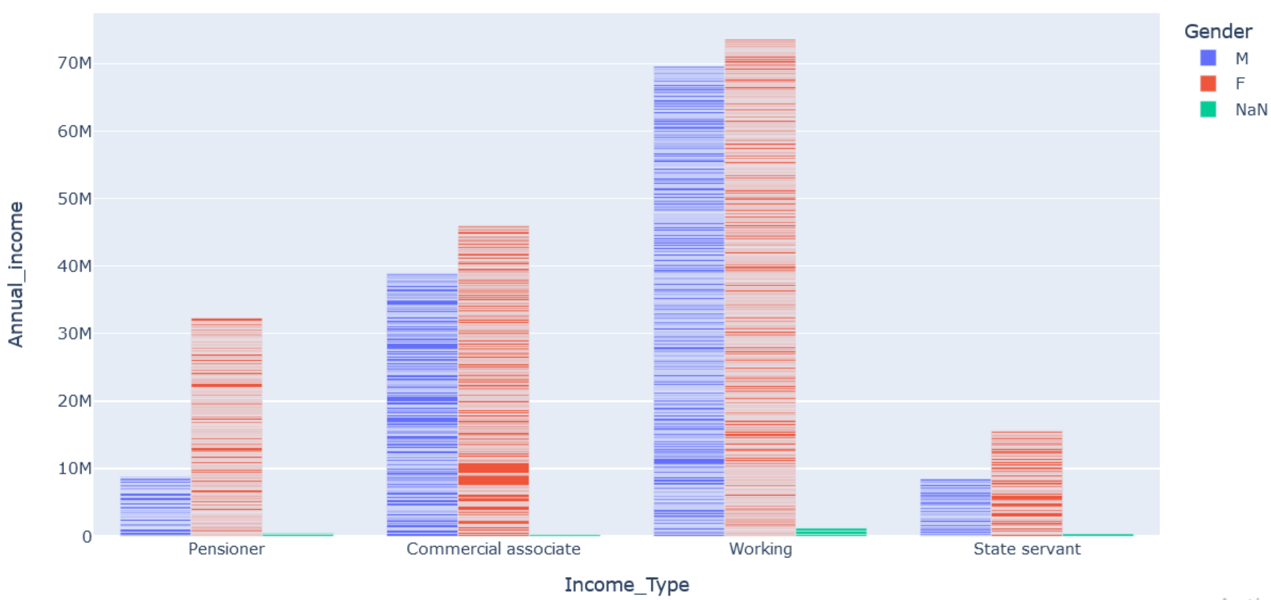
3.10.3 Boxplot

```
# Create a boxplot to visualize the relationship between Annual income and Income type

plt.figure(figsize=(8, 6))
sns.boxplot(x=creditcard_df['Income_Type'], y=creditcard_df['Annual_income'], data=df, palette='Set3')

plt.title('Annual Income by Income Type')
plt.xlabel('Income Type')
plt.ylabel('Annual Income')

plt.show()
```



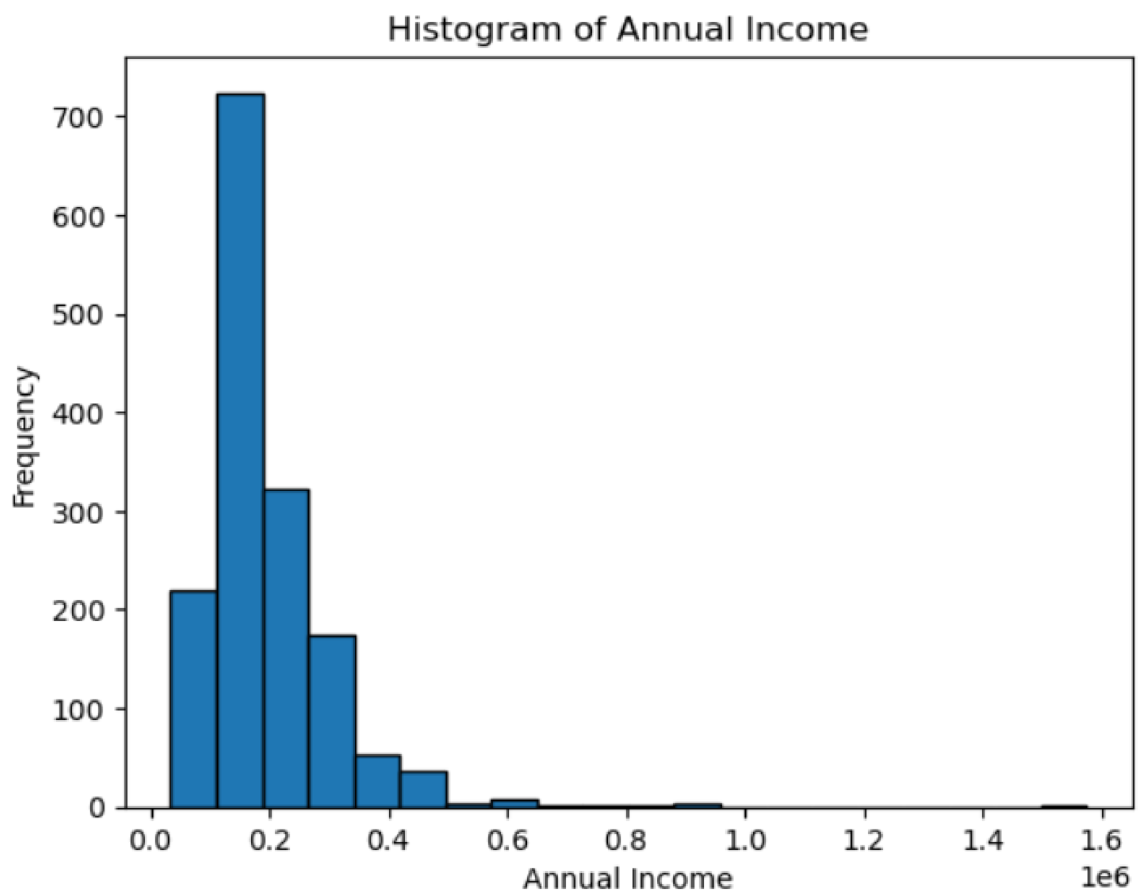
3.10.4 Bar chart

```
# Relation between Annual incom, Income type in Gender

fig=px.bar(creditcard_df, x="Income_Type", y="Annual_income", color="Gender", barmode="group")
fig.show()
```



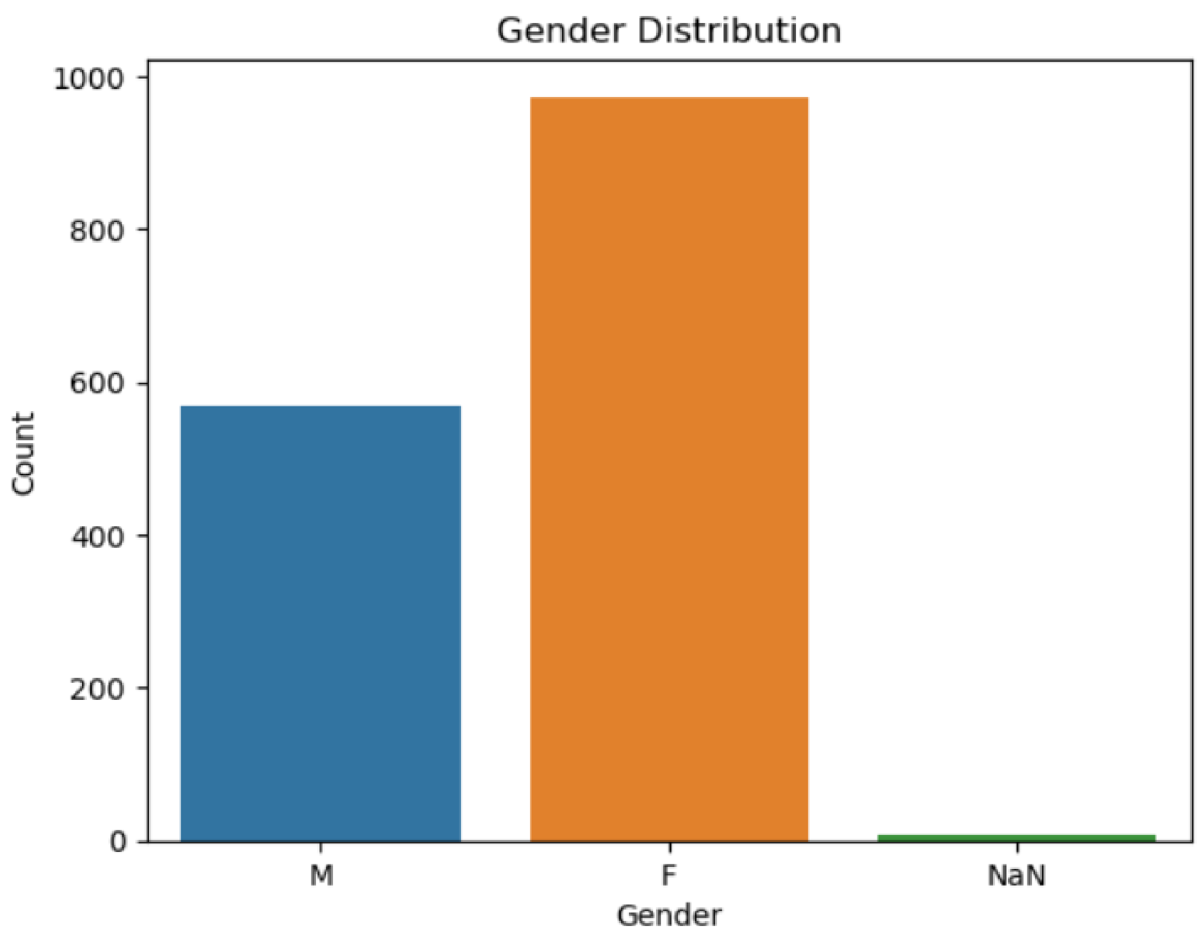
3.10.5 Histogram

```
creditcard_df['Annual_income'].plot.hist(bins=20, edgecolor='k')
plt.xlabel('Annual Income')
plt.ylabel('Frequency')
plt.title('Histogram of Annual Income')
plt.show()
```



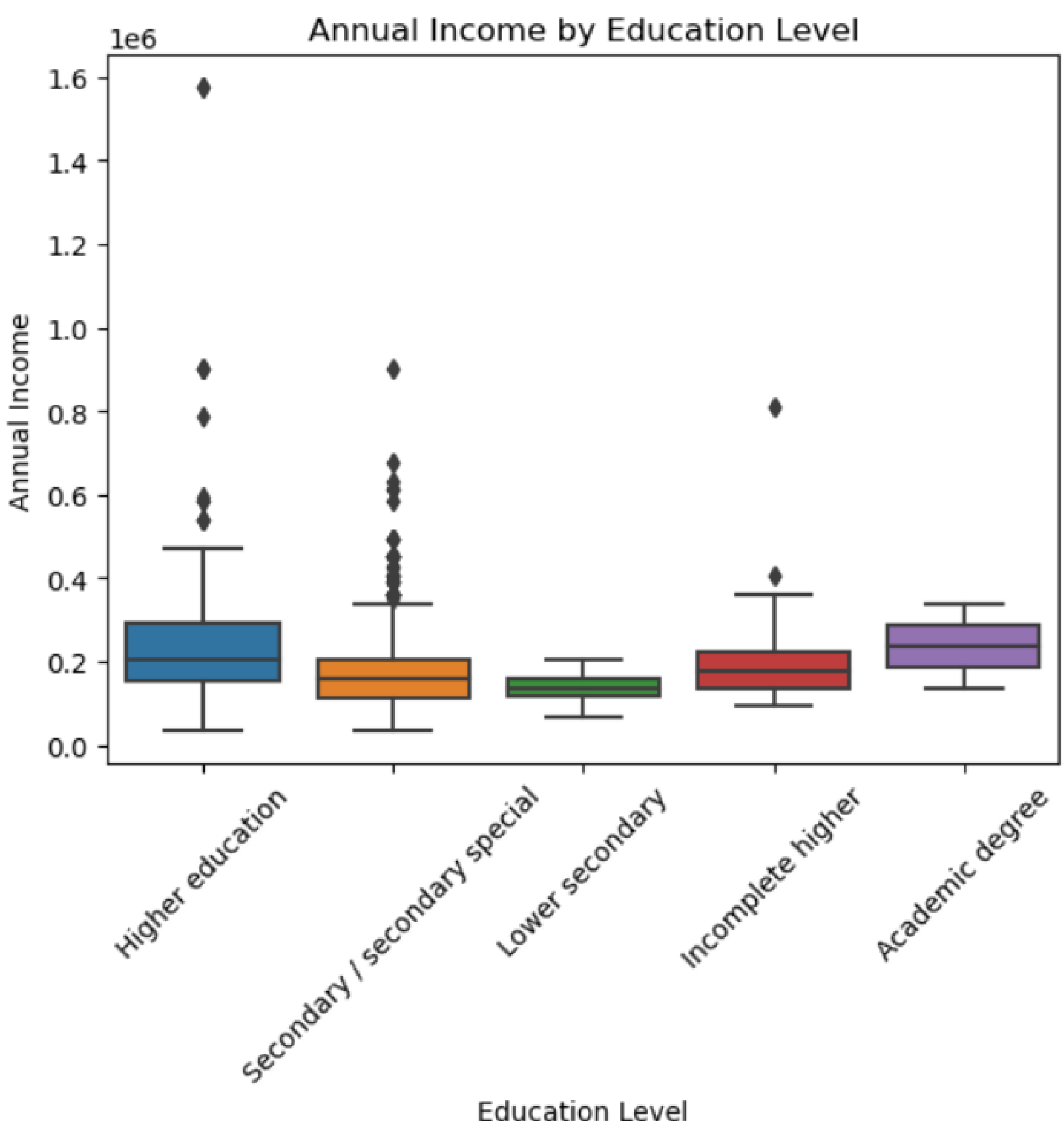
3.10.6 Count plot

```
sns.countplot(data=creditcard_df, x='Gender')
plt.xlabel('Gender')
plt.ylabel('Count')
plt.title('Gender Distribution')
plt.show()
```



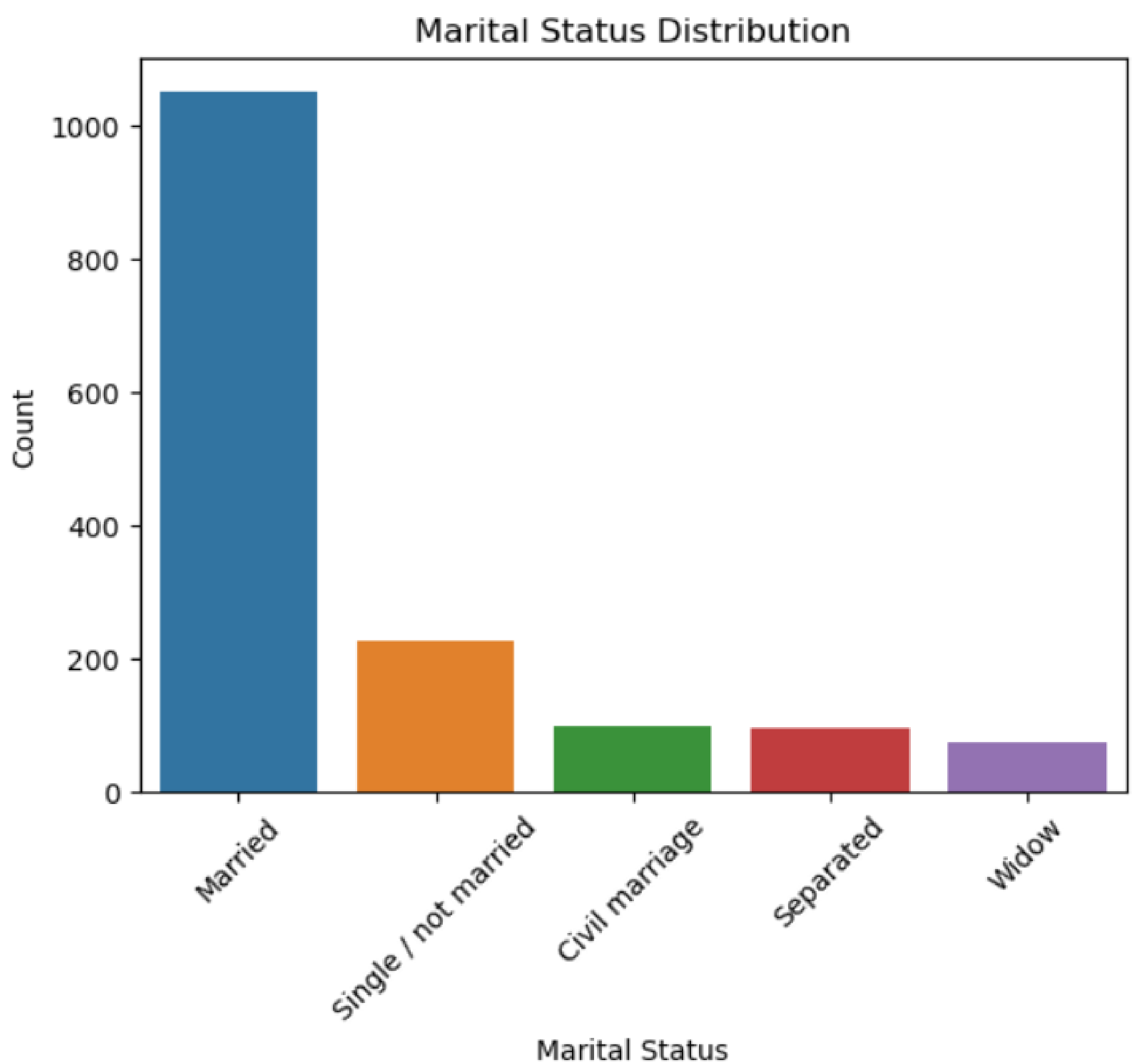
3.10.7 Box plot

```
sns.boxplot(data=creditcard_df, x='Education', y='Annual_income')
plt.xlabel('Education Level')
plt.ylabel('Annual Income')
plt.title('Annual Income by Education Level')
plt.xticks(rotation=45)
plt.show()
```



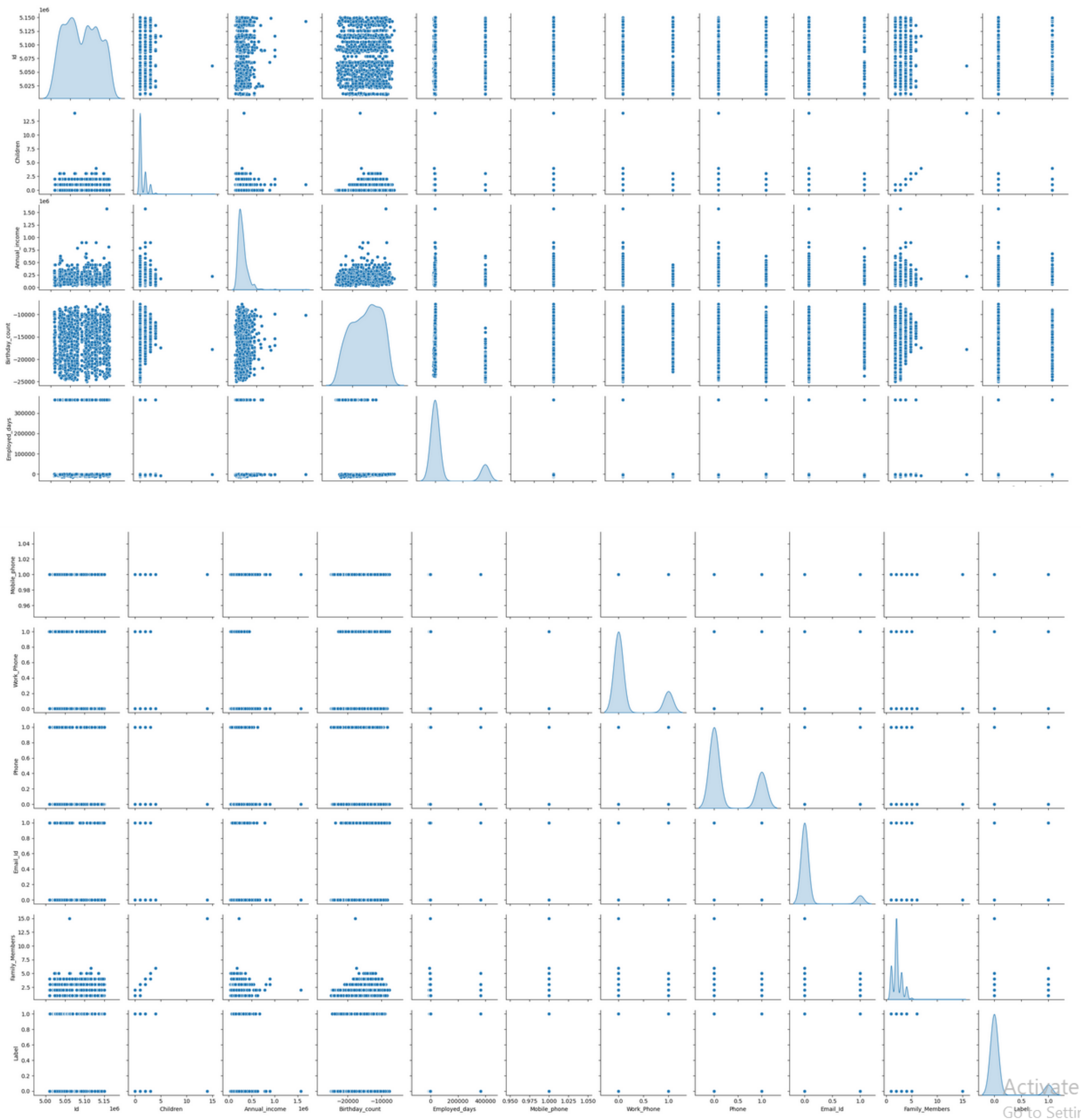
3.10.7 Cout plot

```
sns.countplot(data=creditcard_df, x='Marital_status')
plt.xlabel('Marital Status')
plt.ylabel('Count')
plt.title('Marital Status Distribution')
plt.xticks(rotation=45)
plt.show()
```



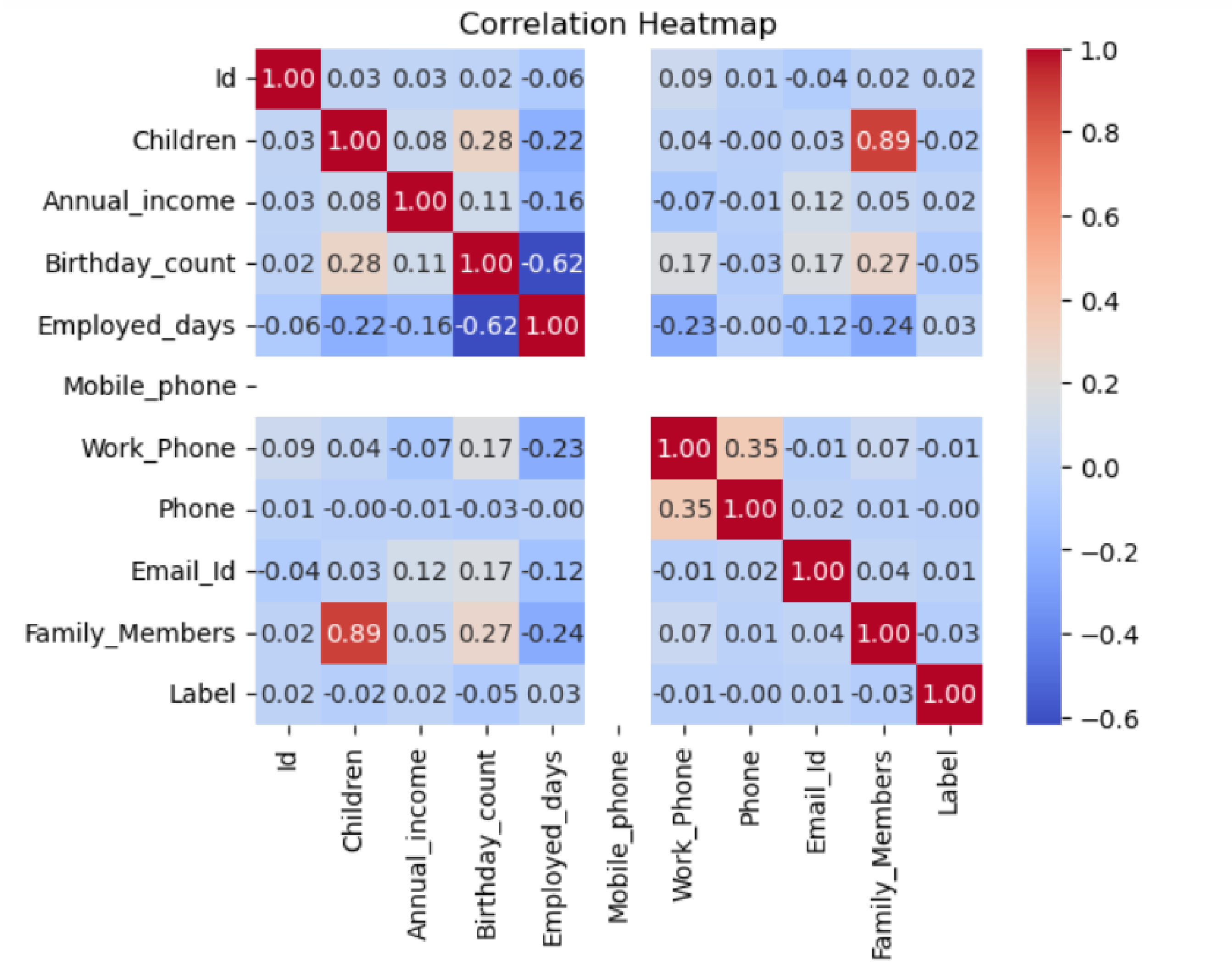
3.10.7 Pair plot

```
# Exploration using pair plot
sns.pairplot(creditcard_df, diag_kind='kde')
plt.show()
```



3.10.7 Correlation Heat map

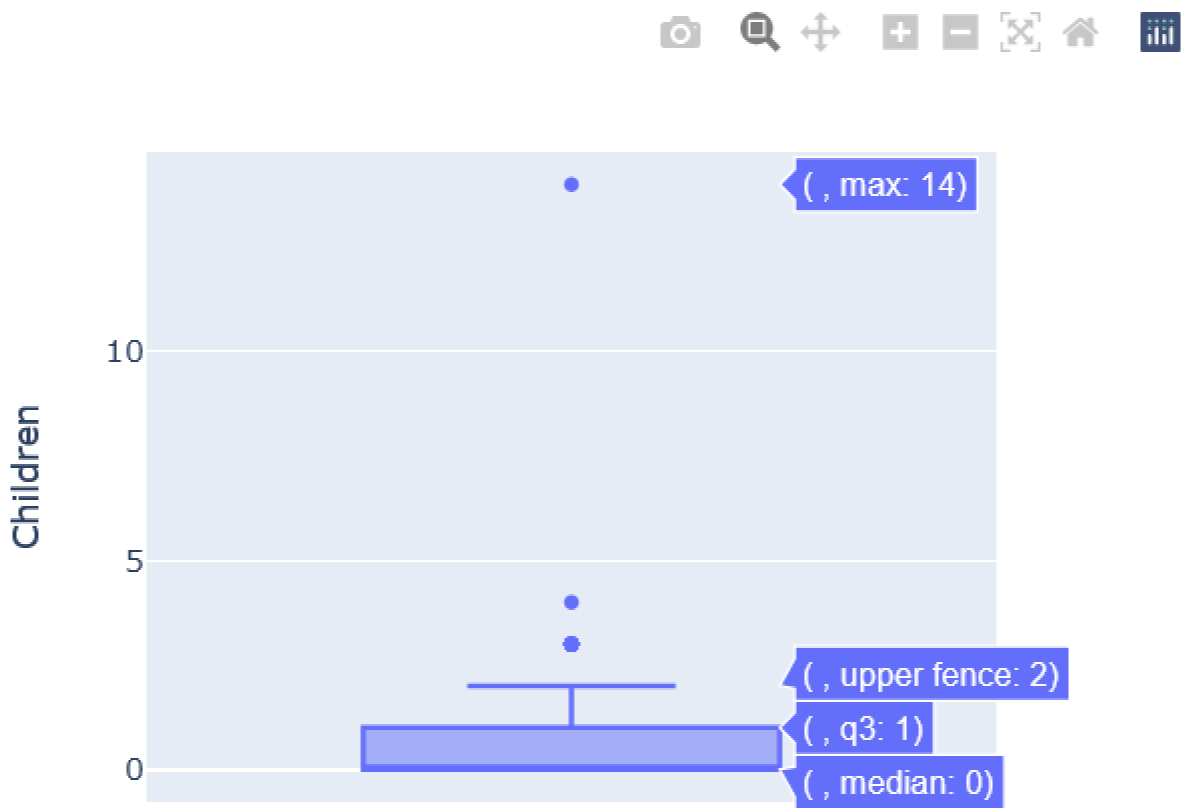
```
# Calculate the correlation matrix including all columns (both numeric and non-numeric)
correlation_matrix = creditcard_df.corr(numeric_only=True)

# Create the heatmap
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Heatmap')
plt.show()
```



Checking for outliers using boxplot

3.11.1

```
fig = px.box (creditcard, y="Children", width=500, height=400)
fig.show()
```



3.11.2

```
fig = px.box (creditcard, y="Family_Members", width=500, height=400,
              labels={"Family_Members"})
fig.show()
```

Section 4: Machine Learning Approach

Machine Learning Approach:

- We will employ various machine learning models, including but not limited to:
    1. Logistic Regression
    2. Random Forest Classifier
    3. Gradient Boosting Classifier
    4. Support Vector Machine (SVM)
- Model selection and hyperparameter tuning will be performed through cross-validation.
- We will evaluate model performance using metrics like accuracy, precision, recall, and F1-score.
- the highest accuracy and the most favorable trade-offs between precision and recall will be selected as the final model.
- We will compare the chosen model's performance to other models to demonstrate its superiority.

Encoding

4.1.1 Dummy encoding

```
# Dummy encoding:
credit = pd.get_dummies(creditcard, columns=['Car_Owner', 'Propert_Owner', 'Housing_type', 'Income_Type', 'Gender'], drop_first=T
credit
```

icipal ment	Housing_type_Office apartment	Housing_type_Rented apartment	Housing_type_With parents	Income_Type_Pensioner	Income_Type_State servant	Income_Type_Working	Gender_M	Gender_NaN
0	0	0	0	1	0	0	1	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	1	0
0	0	0	0	0	0	1	1	0
0	0	0	0	0	0	1	0	0

4.2.1 Separating the dependent and independent variables

```
columns = ["Children", "Annual_income", "Employed_days", "Family_Members","Car_Owner_Y", "Propert_Owner_Y
x = credit[columns]
y = credit["Label"]
x.head()
```

	Children	Annual_income	Employed_days	Family_Members	Car_Owner_Y	Propert_Owner_Y	Housing_type_House / apartment	Housing_type_
0	0	180000.0	365243	2	1	1	1	
1	0	315000.0	-586	2	1	0	1	
2	0	315000.0	-586	2	1	0	1	
3	0	166500.0	-586	2	1	0	1	
4	0	315000.0	-586	2	1	0	1	

4.3.1 Train Test Split

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.15, random_state = 16)
```

```
x_train.head()
```

	Children	Annual_income	Employed_days	Family_Members	Car_Owner_Y	Propert_Owner_Y	Housing_type_House / apartment	Hou
132	0	112500.0	365243	2	0	0	0	
919	0	135000.0	365243	2	0	0	1	
1388	0	247500.0	-1871	2	1	1	1	
508	0	90000.0	-793	2	0	1	1	
1420	0	247500.0	-2980	2	0	1	1	

```
x_test.head()
```

	Children	Annual_income	Employed_days	Family_Members	Car_Owner_Y	Propert_Owner_Y	Housing_type_House / apartment	Hou
89	0	202500.0	-1394	2	1	0	0	
920	1	360000.0	-2908	3	0	0	1	
1439	0	135000.0	-10762	2	0	0	1	
1464	0	180000.0	365243	2	0	1	1	
1103	1	180000.0	-5981	2	1	1	1	

4.4.1 Standardization / Feature scaling

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

Modeling:

4.5.1 Logistic regression

```
# Modeling
from sklearn.linear_model import LogisticRegression
logistic_reg = LogisticRegression(random_state = 0, max_iter=1000)
logistic_reg.fit(x_train, y_train)

# Predictions
ypred_train = logistic_reg.predict(x_train)
ypred_test  = logistic_reg.predict(x_test)


# Evaluation
from sklearn.metrics import accuracy_score
print("Train Accuracy: ", accuracy_score(y_train, ypred_train))
print("Test Accuracy: ", accuracy_score(y_test, ypred_test))

# Calculation of cross validation score:
from sklearn.model_selection import cross_val_score
print("Cross validation score: ", cross_val_score(logistic_reg, x,y, cv=5, scoring="accuracy").mean())
```

```
Train Accuracy:  0.8935361216730038
Test Accuracy:  0.8798283261802575
Cross validation score:  0.8869506211504332
```

4.5.2 KNN / K - Nearest Neighbour

```
# K-Nearest Neighbors with default parameters

# Modeling
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier()     # The default K value is 5
model.fit(x_train, y_train)

# Prediction
ypred_train = model.predict(x_train)
ypred_test  = model.predict(x_test)

# Evaluation
from sklearn.metrics import accuracy_score
print("Train Accuracy: ", accuracy_score(y_train, ypred_train))
print("Test Accuracy: ", accuracy_score(y_test, ypred_test))

from sklearn.model_selection import cross_val_score
print("Cross validation score: ", cross_val_score(model,x,y, cv=5, scoring="accuracy").mean())
```

```
Train Accuracy:  0.9064638783269962
Test Accuracy:  0.8454935622317596
Cross validation score:  0.8514082889654452
```

Hyper Parameter Tuning to improve the model's performance with best combination of hyperparameters

```
# Hyper Parameter Tuning to improve the model's performance with best combination of hyperparameters

from sklearn.model_selection import GridSearchCV

estimator = KNeighborsClassifier()

param_grid = {'n_neighbors': list(range(1,11))}

cv_classifier = GridSearchCV(estimator, param_grid, cv=5, scoring='accuracy')

cv_classifier.fit(x_train, y_train)

cv_classifier.best_params_
```

```
{'n_neighbors': 4}
```

KNN with the best combinations

```
# KNN with the best combinations
# Applying the k value obtained from Hyper parameter tuning to the KNN to get the best accuracy & CVS

# Modeling
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier(n_neighbors = 4)     # Changed the default K value to the value obtained from HPT
model.fit(x_train, y_train)

# Prediction
ypred = model.predict(x_test)
ypred_train = model.predict(x_train)
ypred_test  = model.predict(x_test)

# Evaluation
from sklearn.metrics import accuracy_score
print("Train Accuracy: ", accuracy_score(y_train, ypred_train))
print("Test Accuracy: ", accuracy_score(y_test, ypred_test))

from sklearn.model_selection import cross_val_score
print("Cross validation score: ", cross_val_score(model,x,y, cv=5, scoring="accuracy").mean())
```

```
Train Accuracy:  0.9064638783269962
Test Accuracy:  0.8669527896995708
Cross validation score:  0.8630399832967951
```

4.5.3 SVM / Support Vector Machine

```
# Support vector machine with default parameters

# Modeling
from sklearn.svm import SVC
svm_model = SVC()                    # The default kernel value is 'rbf'
svm_model.fit(x_train, y_train)

# Prediction
ypred_train = svm_model.predict(x_train)
ypred_test  = svm_model.predict(x_test)

# Evaluation
from sklearn.metrics import accuracy_score
print("Train Accuracy: ", accuracy_score(y_train, ypred_train))
print("Test Accuracy: ", accuracy_score(y_test, ypred_test))

from sklearn.model_selection import cross_val_score
print("cross validation score: ", cross_val_score(model, x, y, cv=5, scoring="accuracy").mean())
```

```
Train Accuracy:  0.8973384030418251
Test Accuracy:  0.8798283261802575
cross validation score:  0.8630399832967951
```

4.5.4 Decision tree

```
from sklearn.tree import DecisionTreeClassifier
dt_model = DecisionTreeClassifier()

dt_model.fit(x_train, y_train)
```

```
DecisionTreeClassifier()
```

```
# Decision tree with default parameters

# Modeling
from sklearn.tree import DecisionTreeClassifier
dt_model = DecisionTreeClassifier(random_state=16)      # The default parameters are criterion='gini' and
dt_model.fit(x_train, y_train)

# Prediction
ypred_train = dt_model.predict(x_train)
ypred_test  = dt_model.predict(x_test)

# Evaluation
from sklearn.metrics import accuracy_score
print("Train Accuracy: ", accuracy_score(y_train, ypred_train))
print("Test Accuracy: ", accuracy_score(y_test, ypred_test))

from sklearn.model_selection import cross_val_score
print("cross validation score: ", cross_val_score(model, x, y, cv=5, scoring="accuracy").mean())
```

```
Train Accuracy:  0.9870722433460076
Test Accuracy:  0.8412017167381974
cross validation score:  0.8630399832967951
```

Hyper Parameter Tuning to improve the model's performance with best combination of hyperparameters

```
# Hyper Parameter Tuning to improve the model's performance with best combination of hyperparameters

from sklearn.model_selection import GridSearchCV

estimator = DecisionTreeClassifier(random_state=16)

param_grid = {"criterion":["gini","entropy"], "max_depth":list(range(1,11))}

grid = GridSearchCV(estimator, param_grid, cv=5)

grid.fit(x,y)

grid.best_params_
```

```
{'criterion': 'gini', 'max_depth': 2}
```

Decision tree with the best combinations

```
# Decision tree with the best combinations

# Modeling
from sklearn.tree import DecisionTreeClassifier
dt_model = DecisionTreeClassifier(random_state=16, criterion='gini', max_depth=2)
dt_model.fit(x_train, y_train)

# Prediction
ypred_train = dt_model.predict(x_train)
ypred_test  = dt_model.predict(x_test)

# Evaluation
from sklearn.metrics import accuracy_score
print("Train Accuracy: ", accuracy_score(y_train, ypred_train))
print("Test Accuracy: ", accuracy_score(y_test, ypred_test))

from sklearn.model_selection import cross_val_score
print("cross validation score: ", cross_val_score(model, x, y, cv=5, scoring="accuracy").mean())
```

```
Train Accuracy:  0.8912547528517111
Test Accuracy:  0.8798283261802575
cross validation score:  0.8630399832967951
```

4.5.5 Random forest

```
# Random forest with default parameters

# Modeling
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(random_state=16)        # The default parameters is n_estimators=100
model.fit(x_train, y_train)

# Prediction
ypred_train = model.predict(x_train)
ypred_test  = model.predict(x_test)

# Evaluation
from sklearn.metrics import accuracy_score
print("Train Accuracy: ", accuracy_score(y_train, ypred_train))
print("Test Accuracy: ", accuracy_score(y_test, ypred_test))

from sklearn.model_selection import cross_val_score
print("cross validation score: ", cross_val_score(model, x, y, cv=5, scoring="accuracy").mean())
```

```
Train Accuracy:  0.9870722433460076
Test Accuracy:  0.8969957081545065
cross validation score:  0.8727361937571771
```

Hyper Parameter Tuning to improve the model's performance with best combination of hyperparameters

```
# Hyper Parameter Tuning to improve the model's performance with best combination of hyperparameters

from sklearn.model_selection import GridSearchCV

estimator = RandomForestClassifier (random_state=0)

param_grid = {'n_estimators' :list(range(1,101)), "max_depth" :list(range(1,11))}

grid = GridSearchCV(estimator, param_grid, scoring="accuracy",cv=5)

grid.fit(x_train,y_train)

grid.best_params_
```

```
{'max_depth': 10, 'n_estimators': 20}
```

Random forest with the best combinations

```
# Random forest with the best combinations

# Modeling
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(random_state=16, max_depth=10, n_estimators=20)
model.fit(x_train, y_train)

# Prediction
ypred_train = model.predict(x_train)
ypred_test  = model.predict(x_test)

# Evaluation
from sklearn.metrics import accuracy_score
print("Train Accuracy: ", accuracy_score(y_train, ypred_train))
print("Test Accuracy: ", accuracy_score(y_test, ypred_test))

from sklearn.model_selection import cross_val_score
print("cross validation score: ", cross_val_score(model, x, y, cv=5, scoring="accuracy").mean())
```

```
Train Accuracy:  0.9269961977186312
Test Accuracy:  0.8798283261802575
cross validation score:  0.8882409437310784
```

4.5.6 XG Boost

```
# Modeling
from xgboost import XGBClassifier
xgb_model = XGBClassifier()
xgb_model.fit(x_train, y_train)

# Prediction
ypred_train = xgb_model.predict(x_train)
ypred_test  = xgb_model.predict(x_test)

# Evaluation
from sklearn.metrics import accuracy_score
print("Train Accuracy: ", accuracy_score(y_train, ypred_train))
print("Test Accuracy: ", accuracy_score(y_test, ypred_test))

from sklearn.model_selection import cross_val_score
print("cross validation score: ", cross_val_score(model, x, y, cv=5, scoring="accuracy").mean())
```

```
Train Accuracy:  0.9771863117870723
Test Accuracy:  0.8841201716738197
cross validation score:  0.8882409437310784
```

Model Selection

Comparing all model accuracy to choose the best model:

	Model	Train Accuracy	Test Accuracy	Cross Validation Score
0	Logistic regression	0.89	0.87	0.88
1	KNN	0.90	0.86	0.86
2	SVM	0.89	0.87	0.86
3	Decission tree	0.89	0.87	0.86
4	Random forest	0.92	0.87	0.88
5	XG Boost	0.97	0.88	0.88

* Among the models XGBoost is appears to be a good one wiht highest train accuracy, decent test accuracy and decent cross validation score.

* So XGBoost is the best model here

Checking for confusion matrix and Classification report for the final model

```
from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test, ypred_test))
```

```
[[197   8]
 [ 19   9]]
```

```
from sklearn.metrics import classification_report
print(classification_report(y_test, ypred_test))
```

```
              precision    recall  f1-score   support

           0       0.91      0.96      0.94       205
           1       0.53      0.32      0.40        28

    accuracy                           0.88       233
   macro avg       0.72      0.64      0.67       233
weighted avg       0.87      0.88      0.87       233
```

SQL Queries

Used MySQL to perform the following queries:

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

```sql
# Group the customers based on their income type and find the average of their annual income.
select Type_income as Income_type, avg(Annual_Income) as Average_Annual_Income from credit_card group by Type_Income;
```

Income_type	Average_Annual_Income
Pensioner	152867.6598513011
Commercial associate	229458.08219178082
Working	178553.09774436092
State servant	211422.41379310345

2. **Find the female owners of cars and property.**

```sql
# Find the female owners of cars and property.
select Ind_ID as Applicant_ID from credit_card where
GENDER='F' and  Car_Owner='Y' and  Propert_Owner='Y';
```

Applicant_ID
5018498
5018501
5018503
5024213
5036660
5053254
5065348
5068389
5069216
5069218
5125708
5125716
5033574
5143231
5028917
5095310
5143235
5054522
5126034

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

```sql
# Find the male customers who are staying with their families.
select distinct Housing_type from credit_card;
select Ind_ID as Applicant_ID from credit_card where Housing_type ='With parents' and GENDER='M';
```

Applicant_ID
5021303
5079166
5079167
5079168
5050729
5028383
5143019
5067982
5143573
5038751
5010203
5058267
5024352
5150038
5113302
5126311

4. **List the top five people having the highest income.**

```sql
# List the top five people having the highest income.
select Ind_ID as Applicant_ID, Annual_income from credit_card order by Annual_income desc limit 5;
```

Applicant_ID	Annual_income
5058350	99000
5143009	99000
5135530	99000
5114032	99000
5089658	99000

5. **Determine how many married people are having bad credit.**

```sql
# Determine how many married people are having bad credit.
select cc.Ind_ID as Applicant_ID from credit_card cc
inner join credit_card_label ccl on cc.Ind_ID=ccl.Ind_ID
where Marital_status='Married' and label=1;
```

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

Applicant_ID
5008827
5009744
5009746
5009749
5009752
5009753
5009754
5009894
5010864
5010868
5018498
5018501
5018503

6. **Identify the highest education level and calculate the total count.**

```sql
# Identify the highest education level and calculate the total count.
select EDUCATION, count(*) from credit_card where EDUCATION='Academic degree';
```

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

EDUCATION	count(*)
Academic degree	2

7. **Compare between married males and females, determining who has more bad credit.**

```sql
# Compare between married males and females, determining who has more bad credit.
select GENDER as Gender, count(*) as Bad_credit_count from credit_card cc inner join credit_card_label ccl on cc.Ind_ID=ccl.Ind_ID
where Marital_status='Married' and label=1 group by GENDER;
```

Gender	Bad_credit_count
M	51
F	59
	4