**Capstone Project 1**

Name: **Jaswinder Singh Ahuja**

Student ID: **S5037**

Cohort ID: **DS37A**

We have been given a Machine Learning Project where we are required to analyze the data with multiple dimensions and apply machine learning algorithms.

The project that I have chosen is ***Credit Card Approval***.

Note: The file contains answers to the questions and from section 3, answers are given in the form of coding comments and highlighted texts.

Section wise answers:

Section 1: Questions to Answer

What questions do you want to answer?

1. Why is your proposal important in today’s world? How predicting a good client is worthy for a bank?   
   Ans- An ideal client for a bank is someone who consistently makes regular payments, positively influencing banking profits, investor interest, market fund turnover, and transaction charges. My proposal aims to assist banks in identifying their ideal clients by considering specific details.
2. How is it going to impact the banking sector?   
   Ans- In the banking sector, a customer who makes payments is crucial, and simultaneously, creditworthiness holds significance. This implies that an ideal client for a bank is someone who is a regular customer and possesses the ability to repay, as opposed to someone who defaults or may default due to payment capacity issues.
3. If any, what is the gap in the knowledge or how your proposed method can be helpful if required in future for any bank in India.  
   Ans- Considering the data, it is very basic since there are very few details mentioned here where it is difficult to derive ideal and non-ideal customers. For clarity and understanding we would need to consider more factors such as follows:
   * + Client’s credit history
     + Geographical credit history
     + Nature of Job (Stable/Unstable)
     + Capping limits as per Client’s income
     + Age of the Client

Section 2: Initial Hypothesis (or hypotheses)

1. Here you have to make some assumptions based on the questions you want to address based on the DA track or ML track.   
    i) If DA track please aim to identify patterns in the data and important features that may impact a ML model.  
   ii) If ML track please perform part ‘i’ as well as multiple machine learning models, perform all required steps to check if there is any assumption and justify your model. Why is your model better than any other possible model? Please justify it by relevant cost functions and if possible by any graph.  
   Ans- **a)** **My first hypothesis from the data is income and approval of credit card are corelated.  
    b) My second hypothesis is that the random forest classifier model will achieve the highest level of accuracy.**
2. From step 1, you may see some relationship that you want to explore and will develop a belief about data

Section 3: Data analysis approach

1. What approach are you going to take in order to prove or disprove your hypothesis?
2. What feature engineering techniques will be relevant to your project?
3. Please justify your data analysis approach.
4. Identify important patterns in your data using the EDA approach to justify your findings.

Answers to the section 3 questions are below:

We will start our project by downloading the data files Credit\_card and Credit\_card\_label. File Credit\_card\_label has Ind\_ID (customer id) and label column (0,1 variables i.e. 0 means approved and 1 means rejected).

We will start our project by importing necessary libraries:

# Importing necessary libraries

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.impute import SimpleImputer

from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

import seaborn as sns

import matplotlib.pyplot as plt

print("Libraries imported successfully")

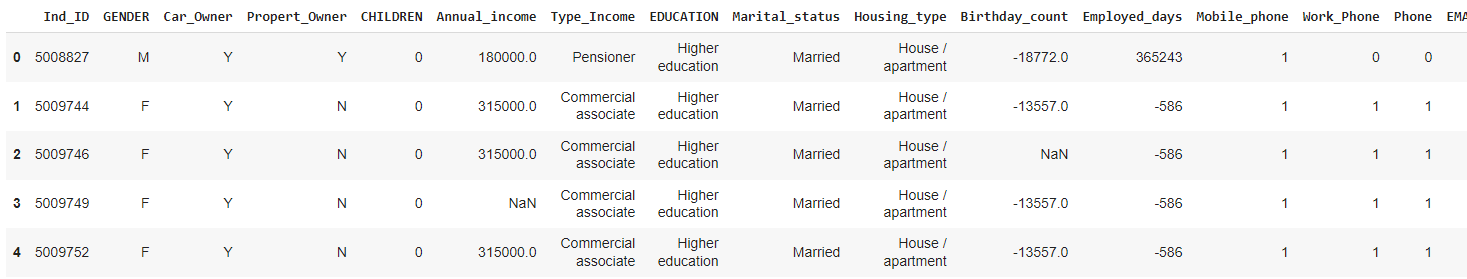
# Loading first csv file "Credit\_card.csv"

credit\_card\_data = pd.read\_csv('Credit\_card.csv')

# Showing the first few rows of the file

#print("Credit\_card.csv:")

credit\_card\_data.head()

Preview of the first few rows:  
  


# Loading the second file "Credit\_card\_label.csv"

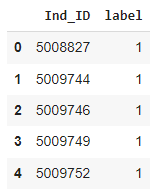
credit\_card\_label = pd.read\_csv('Credit\_card\_label.csv')

# Showing the first few rows of the file

#print("\nCredit\_card\_label.csv:")

credit\_card\_label.head()

Preview of the first few rows:



# Merging the files/dataframes with common heading 'Ind\_ID'

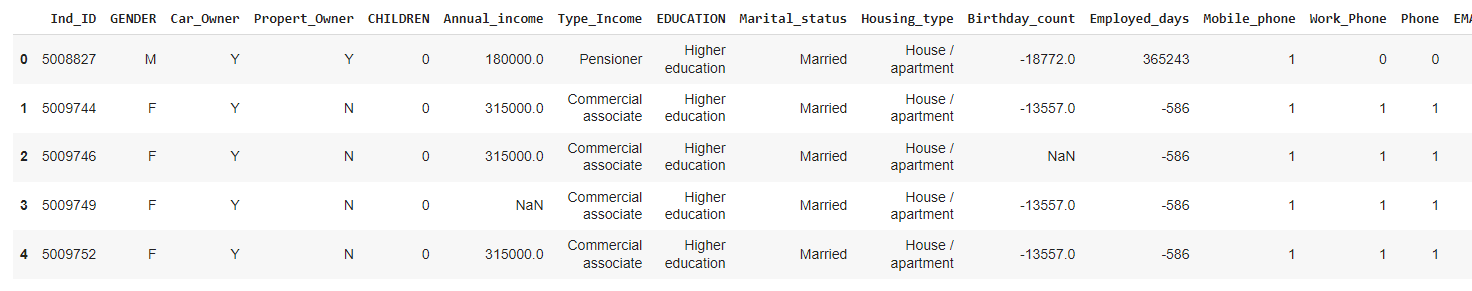
merged\_data = pd.merge(credit\_card\_data, credit\_card\_label, on='Ind\_ID')

print("Merge Successful")

# Showing the the first few rows of the merged dataframe

merged\_data.head()

Preview of the first few rows:



# Checking the variable information

merged\_data.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

# Identify missing values

missing\_values = merged\_data.isnull().sum()

# Showing missing values for the specified columns

print("Missing Values:")

print(missing\_values[['GENDER', 'Annual\_income', 'Type\_Occupation', 'Birthday\_count']])

Missing Values:

GENDER 7

Annual\_income 23

Type\_Occupation 488

Birthday\_count 22

dtype: int64

# Converting the negative 'Birthday Count' values to positive

merged\_data['Birthday\_count'] = merged\_data['Birthday\_count'].abs()

  # Check the summary statistics after converting to ensure there are no negative values

  print("Summary Statistics after Converting 'Birthday Count' to Positive:")

  print(merged\_data['Birthday\_count'].describe())

Summary Statistics after Converting 'Birthday Count' to Positive:

count 1526.000000

mean 16040.342071

std 4229.503202

min 7705.000000

25% 12417.000000

50% 15661.500000

75% 19553.000000

max 24946.000000

Name: Birthday\_count, dtype: float64

# Calculating the mean and median for numerical columns

mean\_annual\_income = merged\_data['Annual\_income'].mean()

median\_annual\_income = merged\_data['Annual\_income'].median()

mean\_birthday\_count = merged\_data['Birthday\_count'].mean()

median\_birthday\_count = merged\_data['Birthday\_count'].median()

# Print the calculated values

print("Mean Annual Income:", mean\_annual\_income)

print("Median Annual Income:", median\_annual\_income)

print("\nMean Birthday Count:", mean\_birthday\_count)

print("Median Birthday Count:", median\_birthday\_count)

Output:  
Mean Annual Income: 191399.3262295082

Median Annual Income: 166500.0

Mean Birthday Count: 16040.342070773264

Median Birthday Count: 15661.5

# Imputing the missing values

# For numerical columns like 'Annual\_income' and 'Birthday\_count', imputing values using median

merged\_data['Annual\_income'].fillna(merged\_data['Annual\_income'].median(), inplace=True)

merged\_data['Birthday\_count'].fillna(merged\_data['Birthday\_count'].median(), inplace=True)

# For categorical columns like 'GENDER' and 'Type\_Occupation', using mode imputation

merged\_data['GENDER'].fillna(merged\_data['GENDER'].mode()[0], inplace=True)

merged\_data['Type\_Occupation'].fillna(merged\_data['Type\_Occupation'].mode()[0], inplace=True)

# Now verifying that missing values have been handled

print("\nMissing Values after Imputation:")

print(merged\_data[['GENDER', 'Annual\_income', 'Type\_Occupation', 'Birthday\_count']].isnull().sum())

Missing Values after Imputation:

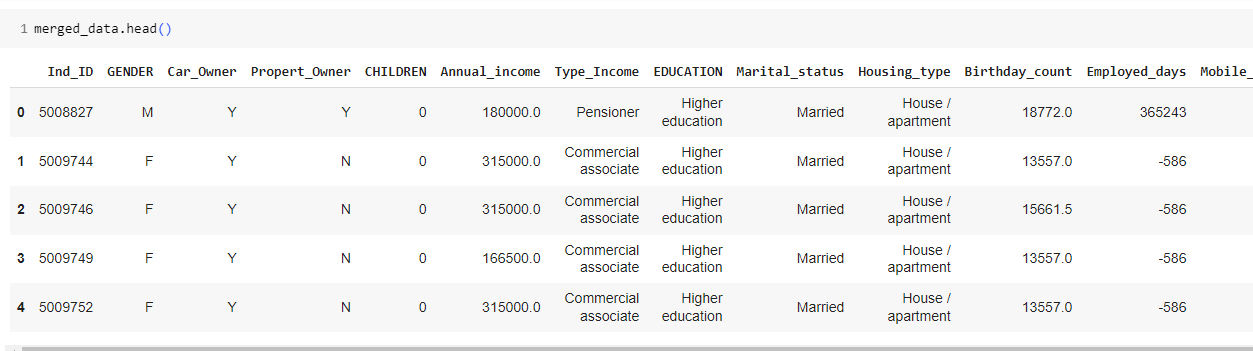
GENDER 0

Annual\_income 0

Type\_Occupation 0

Birthday\_count 0

dtype: int64



Checking the null values again:

merged\_data.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

# Converting 'Birthday\_count' to 'Age'

merged\_data['Age'] = merged\_data['Birthday\_count'] // 365

# Treating the negative values in 'Employed\_days' by converting them to positive

merged\_data['Employed\_days'] = merged\_data['Employed\_days'].abs()

# Creating a new column for 'Experience' in years

merged\_data['Experience'] = merged\_data['Employed\_days'] // 365

# Showing the first few rows to verify the changes

print(merged\_data[['Birthday\_count', 'Age', 'Employed\_days', 'Experience']].head())

Birthday\_count Age Employed\_days Experience

0 18772.0 51.0 365243 1000

1 13557.0 37.0 586 1

2 15661.5 42.0 586 1

3 13557.0 37.0 586 1

4 13557.0 37.0 586 1

# Imputing missing values for 'GENDER' using the mode (most frequent value)

merged\_data['GENDER'].fillna(merged\_data['GENDER'].mode()[0], inplace=True)

# Imputing missing values for 'Annual\_income' using the median

merged\_data['Annual\_income'].fillna(merged\_data['Annual\_income'].median(), inplace=True)

# Imputing missing values for 'Type\_Occupation' using the mode (most frequent value)

merged\_data['Type\_Occupation'].fillna(merged\_data['Type\_Occupation'].mode()[0], inplace=True)

# Verify that missing values have been handled

print("Missing Values after Imputation:")

print(merged\_data[['GENDER', 'Annual\_income', 'Type\_Occupation']].isnull().sum())

Missing Values after Imputation:

GENDER 0

Annual\_income 0

Type\_Occupation 0

dtype: int64

# Exporting the updated DataFrame to a new CSV file

merged\_data.to\_csv('updated\_merged\_data.csv', index=False)

print("Updated Merged Data exported to 'updated\_merged\_data.csv'")

Note: Downloaded file is named as ‘updated\_merged\_data.csv’ which is available in the zip folder

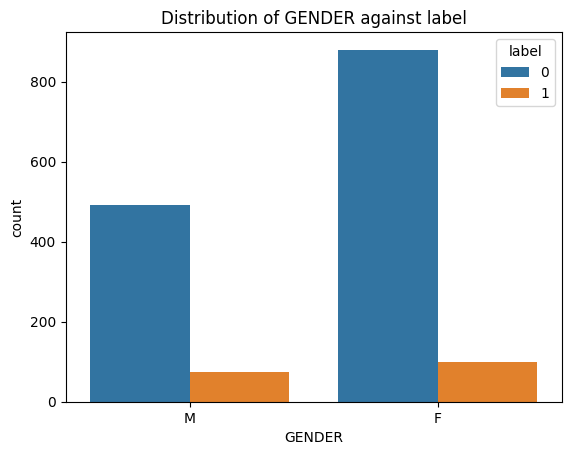
**EDA Part:**

# Visualizing the distribution of the target variable 'label'

sns.countplot(x='label', data=merged\_data)

plt.title('Distribution of Credit Card Approval')

plt.show()



# Stacked bar chart for 'Type\_Occupation' against 'label'

type\_occ\_count = pd.crosstab(merged\_data['Type\_Occupation'], merged\_data['label'])

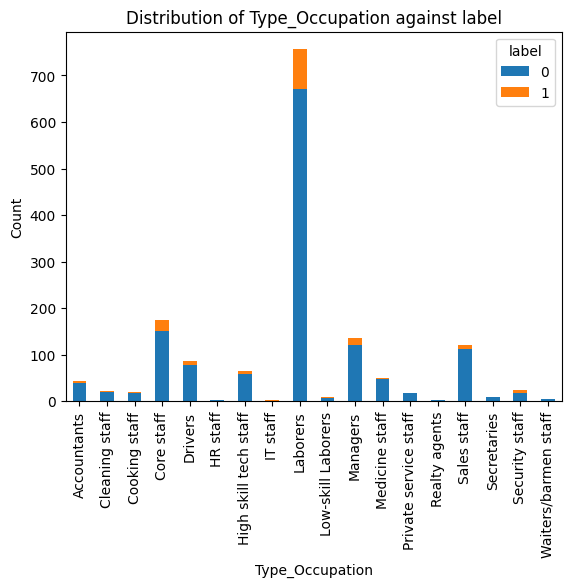
type\_occ\_count.plot(kind='bar', stacked=True)

plt.title('Distribution of Type\_Occupation against label')

plt.xlabel('Type\_Occupation')

plt.ylabel('Count')

plt.show()

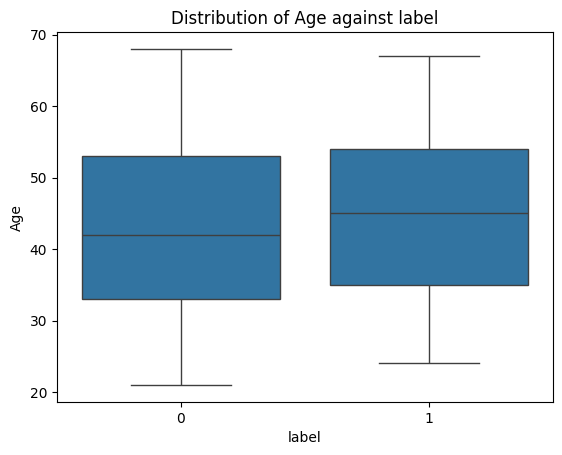


# Boxplot for 'Age' against 'label'

sns.boxplot(x='label', y='Age', data=merged\_data)

plt.title('Distribution of Age against label')

plt.show()



# Histogram for 'Experience' with different colors for each label

plt.hist([merged\_data[merged\_data['label'] == 0]['Experience'],

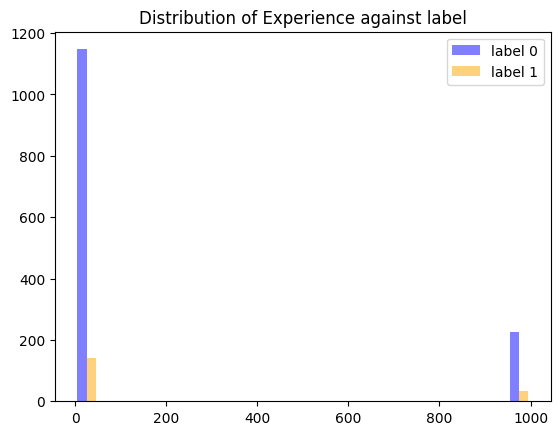
          merged\_data[merged\_data['label'] == 1]['Experience']],

          bins=20, color=['blue', 'orange'], alpha=0.5, label=['label 0', 'label 1'])

plt.title('Distribution of Experience against label')

plt.legend()

plt.show()



# Histogram for 'Annual\_income' with different colors for each label

plt.hist([merged\_data[merged\_data['label'] == 0]['Annual\_income'],

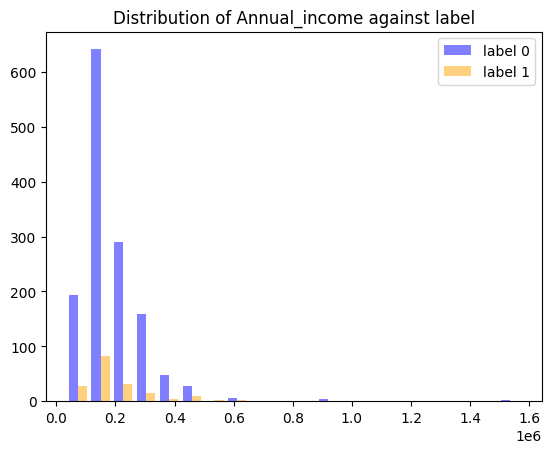
          merged\_data[merged\_data['label'] == 1]['Annual\_income']],

          bins=20, color=['blue', 'orange'], alpha=0.5, label=['label 0', 'label 1'])

plt.title('Distribution of Annual\_income against label')

plt.legend()

plt.show()



# Calculate correlation matrix

correlation\_matrix = merged\_data[['label', 'Age', 'Experience', 'Annual\_income', 'Family\_Members']].corr()

# Get correlation of 'label' with other columns

label\_correlation = correlation\_matrix['label'].sort\_values(ascending=False)

# Plotting a heatmap for the correlation matrix

plt.figure(figsize=(10, 8))

sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm', fmt=".2f")

plt.title('Correlation Matrix')

plt.show()

# Print the correlation values

print("Correlation of 'label' with other columns:")

print(label\_correlation)

# Summary of strong correlations (e.g., absolute correlation greater than 0.3)

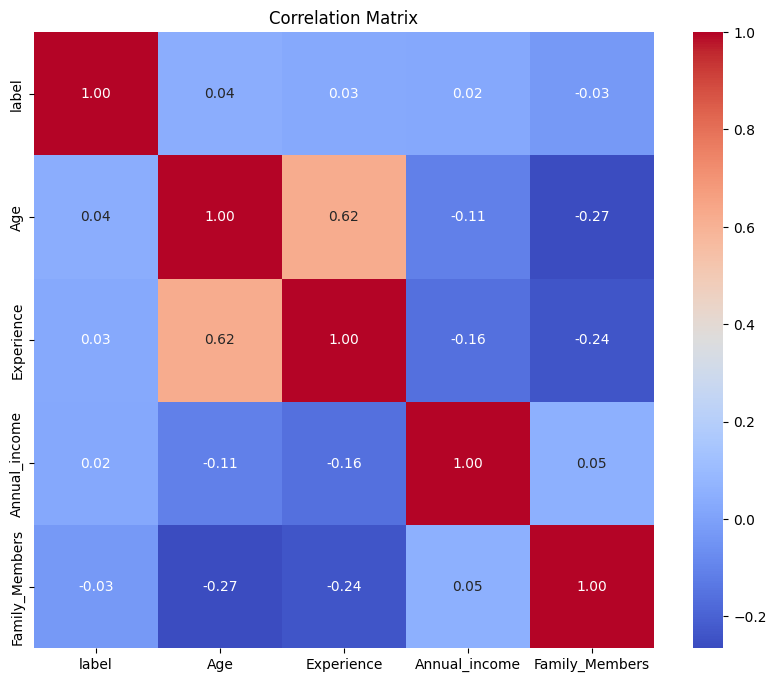
strong\_correlation = label\_correlation[abs(label\_correlation) > 0.3]

# Print summary of strong correlations

print("\nSummary of Strong Correlations:")

for column, correlation in strong\_correlation.items():

    print(f"'{column}' has a correlation of {correlation:.2f} with 'label'")



Correlation of 'label' with other columns:

label 1.000000

Age 0.044451

Experience 0.028390

Annual\_income 0.024425

Family\_Members -0.030709

Name: label, dtype: float64

Summary of Strong Correlations:

'label' has a correlation of 1.00 with 'label'

Section 4: Machine learning approach

1. What method will you use for machine learning based predictions for credit card approval?
2. Please justify the most appropriate model.
3. Please perform necessary steps required to improve the accuracy of your model.
4. Please compare all models (at least 4 models).

Performing T-Test on the Hypothesis:

Testing Annual\_income column and label column relation using T-Test.

from scipy.stats import ttest\_ind

income\_label\_0 = merged\_data[merged\_data['label'] == 0]['Annual\_income']

income\_label\_1 = merged\_data[merged\_data['label'] == 1]['Annual\_income']

t\_stat, p\_value = ttest\_ind(income\_label\_0, income\_label\_1)

print("Income for label 0:")

print(income\_label\_0.head())

print("\nIncome for label 1:")

print(income\_label\_1.head())

Income for label 0:

175 81000.0

176 225000.0

177 180000.0

178 135000.0

179 135000.0

Name: Annual\_income, dtype: float64

Income for label 1:

0 180000.0

1 315000.0

2 315000.0

3 166500.0

4 315000.0

Name: Annual\_income, dtype: float64

from scipy.stats import ttest\_ind

t\_stat, p\_value = ttest\_ind(income\_label\_0, income\_label\_1)

print("T-statistic:", t\_stat)

print("P-value:", p\_value)

T-statistic: -0.9606540345622575

P-value: 0.3368763675425366

Since, P-value is far from 0.5 we cannot say that the relation is strong enough.

**Checking another parameter Age and label performing T-Test:**

from scipy.stats import ttest\_ind

# Extract age data for each label

age\_label\_0 = merged\_data[merged\_data['label'] == 0]['Age']

age\_label\_1 = merged\_data[merged\_data['label'] == 1]['Age']

# Perform t-test

t\_stat\_age, p\_value\_age = ttest\_ind(age\_label\_0, age\_label\_1)

# Print the results

print("T-statistic:", t\_stat\_age)

print("P-value:", p\_value\_age)

T-statistic: -1.7495230357128924

P-value: 0.08039905895998459

Now, the P-value is very far from 0.5 and the relation appears weak.

**Checking another parameter Experience and label performing T-Test:**

from scipy.stats import ttest\_ind

# Extract experience data for each label

experience\_label\_0 = merged\_data[merged\_data['label'] == 0]['Experience']

experience\_label\_1 = merged\_data[merged\_data['label'] == 1]['Experience']

# Perform t-test

t\_stat\_experience, p\_value\_experience = ttest\_ind(experience\_label\_0, experience\_label\_1)

# Print the results

print("T-statistic for Experience:", t\_stat\_experience)

print("P-value for Experience:", p\_value\_experience)

T-statistic for Experience: -1.1167232500845914

P-value for Experience: 0.26428619794365327

Here, again P-value is very far from 0.5 and thus none of the parameters are corelated with label column.

**To cross check we also have checked Chi-Square method:**

from scipy.stats import chi2\_contingency

# Create a contingency table

contingency\_table = pd.crosstab(merged\_data['label'], merged\_data['Type\_Occupation'])

# Perform Chi-squared test

chi2\_stat, p\_value, dof, expected = chi2\_contingency(contingency\_table)

# Print the results

print("Chi-squared Statistic:", chi2\_stat)

print("P-value:", p\_value)

Chi-squared Statistic: 36.56584762842536

P-value: 0.003851575952868713

**Again, P-value is far from 0.5**

Machine Learning Models:

# Import necessary libraries

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler, LabelEncoder

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.svm import SVC

# Load the  updated data

merged\_data = pd.read\_csv('updated\_merged\_data.csv')

# Drop unnecessary columns

X = merged\_data.drop(['Ind\_ID', 'label'], axis=1)

y = merged\_data['label']

# Encoding the categorical variables

label\_encoder = LabelEncoder()

X['GENDER'] = label\_encoder.fit\_transform(X['GENDER'])

X['Car\_Owner'] = label\_encoder.fit\_transform(X['Car\_Owner'])

X['Propert\_Owner'] = label\_encoder.fit\_transform(X['Propert\_Owner'])

X['Type\_Income'] = label\_encoder.fit\_transform(X['Type\_Income'])

X['EDUCATION'] = label\_encoder.fit\_transform(X['EDUCATION'])

X['Marital\_status'] = label\_encoder.fit\_transform(X['Marital\_status'])

X['Housing\_type'] = label\_encoder.fit\_transform(X['Housing\_type'])

X['Phone'] = label\_encoder.fit\_transform(X['Phone'])

X['EMAIL\_ID'] = label\_encoder.fit\_transform(X['EMAIL\_ID'])

X['Type\_Occupation'] = label\_encoder.fit\_transform(X['Type\_Occupation'])

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Standardize numerical features

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

# Define models

models = {

    'Random Forest': RandomForestClassifier(random\_state=42),

    'Logistic Regression': LogisticRegression(random\_state=42),

    'Support Vector Machine': SVC(random\_state=42),

    'Gradient Boosting': GradientBoostingClassifier(random\_state=42)

}

# Train and evaluate models

for model\_name, model in models.items():

    print(f"Training {model\_name}...")

    model.fit(X\_train, y\_train)

    # Make predictions on the test set

    y\_pred = model.predict(X\_test)

    # Evaluate the model

    accuracy = accuracy\_score(y\_test, y\_pred)

    print(f"\n{model\_name} Performance:")

    print(f"Accuracy: {accuracy:.4f}")

    print("Confusion Matrix:")

    print(confusion\_matrix(y\_test, y\_pred))

    print("Classification Report:")

    print(classification\_report(y\_test, y\_pred))

    print("="\*50)

Training Random Forest...

Random Forest Performance:

Accuracy: 0.9419

Confusion Matrix:

[[280 0]

[ 18 12]]

Classification Report:

precision recall f1-score support

0 0.94 1.00 0.97 280

1 1.00 0.40 0.57 30

accuracy 0.94 310

macro avg 0.97 0.70 0.77 310

weighted avg 0.95 0.94 0.93 310

==================================================

Training Logistic Regression...

Logistic Regression Performance:

Accuracy: 0.9032

Confusion Matrix:

[[280 0]

[ 30 0]]

Classification Report:

precision recall f1-score support

0 0.90 1.00 0.95 280

1 0.00 0.00 0.00 30

accuracy 0.90 310

macro avg 0.45 0.50 0.47 310

weighted avg 0.82 0.90 0.86 310

==================================================

Training Support Vector Machine...

Support Vector Machine Performance:

Accuracy: 0.9032

Confusion Matrix:

[[280 0]

[ 30 0]]

Classification Report:

precision recall f1-score support

0 0.90 1.00 0.95 280

1 0.00 0.00 0.00 30

accuracy 0.90 310

macro avg 0.45 0.50 0.47 310

weighted avg 0.82 0.90 0.86 310

==================================================

Training Gradient Boosting...

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/\_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero\_division` parameter to control this behavior.

\_warn\_prf(average, modifier, msg\_start, len(result))

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/\_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero\_division` parameter to control this behavior.

\_warn\_prf(average, modifier, msg\_start, len(result))

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/\_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero\_division` parameter to control this behavior.

\_warn\_prf(average, modifier, msg\_start, len(result))

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/\_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero\_division` parameter to control this behavior.

\_warn\_prf(average, modifier, msg\_start, len(result))

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/\_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero\_division` parameter to control this behavior.

\_warn\_prf(average, modifier, msg\_start, len(result))

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/\_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero\_division` parameter to control this behavior.

\_warn\_prf(average, modifier, msg\_start, len(result))

Gradient Boosting Performance:

Accuracy: 0.9161

Confusion Matrix:

[[279 1]

[ 25 5]]

Classification Report:

precision recall f1-score support

0 0.92 1.00 0.96 280

1 0.83 0.17 0.28 30

accuracy 0.92 310

macro avg 0.88 0.58 0.62 310

weighted avg 0.91 0.92 0.89 310

==================================================

Random Forest:

Accuracy: 94.19% Precision for label 0: 94% Precision for label 1: 100% Recall for label 0: 100% Recall for label 1: 40% F1-score for label 0: 97% F1-score for label 1: 57%

Logistic Regression:

Accuracy: 90.32% Precision for label 0: 90% Precision for label 1: 0% Recall for label 0: 100% Recall for label 1: 0% F1-score for label 0: 95% F1-score for label 1: 0%

Support Vector Machine:

Accuracy: 90.32% Precision for label 0: 90% Precision for label 1: 0% Recall for label 0: 100% Recall for label 1: 0% F1-score for label 0: 95% F1-score for label 1: 0%

Gradient Boosting:

Accuracy: 91.61% Precision for label 0: 92% Precision for label 1: 83% Recall for label 0: 100% Recall for label 1: 17% F1-score for label 0: 96% F1-score for label 1: 28%

Hypertuning Parameters

from sklearn.model\_selection import GridSearchCV

# Define the parameter grid

param\_grid = {

    'n\_estimators': [50, 100, 150],

    'max\_depth': [None, 10, 20],

    'min\_samples\_split': [2, 5, 10],

    'min\_samples\_leaf': [1, 2, 4]

}

# Create the Random Forest model

rf\_classifier = RandomForestClassifier(random\_state=42)

# Perform grid search with 5-fold cross-validation

grid\_search = GridSearchCV(rf\_classifier, param\_grid, cv=5, scoring='accuracy')

grid\_search.fit(X\_train, y\_train)

GridSearchCV

estimator: RandomForestClassifier

RandomForestClassifier

# Print the best parameters

print("Best Hyperparameters:", grid\_search.best\_params\_)

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

# Evaluate the model with the best hyperparameters

best\_rf\_model = grid\_search.best\_estimator\_

y\_pred\_rf = best\_rf\_model.predict(X\_test)

# Calculate accuracy

accuracy = accuracy\_score(y\_test, y\_pred\_rf)

print("Accuracy:", accuracy)

# Confusion Matrix

conf\_matrix = confusion\_matrix(y\_test, y\_pred\_rf)

print("\nConfusion Matrix:")

print(conf\_matrix)

# Classification Report

class\_report = classification\_report(y\_test, y\_pred\_rf)

print("\nClassification Report:")

print(class\_report)

Accuracy: 0.9387096774193548

Confusion Matrix:

[[280 0]

[ 19 11]]

Classification Report:

precision recall f1-score support

0 0.94 1.00 0.97 280

1 1.00 0.37 0.54 30

accuracy 0.94 310

macro avg 0.97 0.68 0.75 310

weighted avg 0.94 0.94 0.93 310

Conclusion:

Of the 2 Hypothesis:

- Income and label (approval/rejection) are not strongly corelated due to less evidence of data. It may be possible if high amount of data is available. However, at this stage it is difficult to consider this hypothesis. Additionally, if there are more features like credit score, payment history and geographical locations this could have supported the hypothesis and we could have proved this.

- Random forest classifier has achieved highest accuracy at 94.19% of the other models. This proves the second hypothesis correct.

SQL Queries:

1. Group the customers based on their income type and find the average of their annual income.
2. Find the female owners of cars and property.
3. Find the male customers who are staying with their families.
4. Please list the top five people having the highest income.
5. How many married people are having bad credit?
6. What is the highest education level and what is the total count?
7. Between married males and females, who is having more bad credit?

We have initialized by creating the table as per headers available in the updated cleaned file which was exported during the Machine Learning Project.

CREATE TABLE credit\_card\_approval.updated\_merged\_data (

Ind\_ID INT,

GENDER VARCHAR(10),

Car\_Owner VARCHAR(10),

Propert\_Owner VARCHAR(10),

CHILDREN INT,

Annual\_income INT,

Type\_Income VARCHAR(20),

EDUCATION VARCHAR(50), -- Corrected to VARCHAR(50) since text length was high

Marital\_status VARCHAR(20),

Housing\_type VARCHAR(20),

Birthday\_count INT,

Employed\_days INT,

Mobile\_phone VARCHAR(15),

Work\_Phone VARCHAR(15),

Phone VARCHAR(15),

EMAIL\_ID VARCHAR(50),

Type\_Occupation VARCHAR(50),

Family\_Members INT,

label VARCHAR(10),

Age INT,

Experience INT

);

Loading the CSV file in the project:

LOAD DATA INFILE 'C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\updated\_merged\_data.csv'

INTO TABLE credit\_card\_approval.updated\_merged\_data

FIELDS TERMINATED BY ','

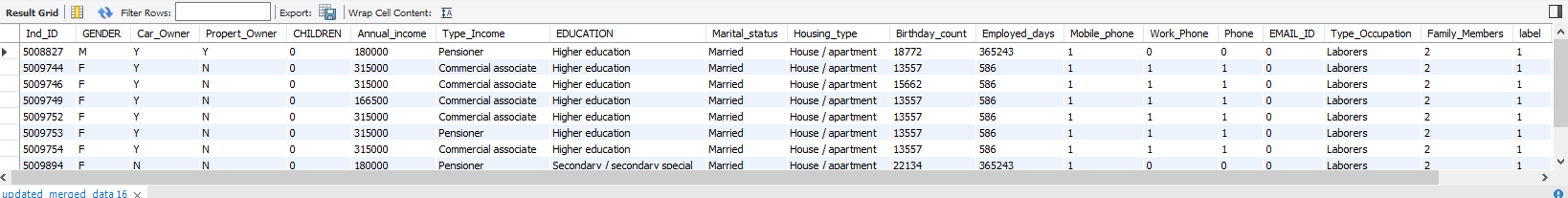
ENCLOSED BY '"'

LINES TERMINATED BY '\r\n'

IGNORE 1 ROWS; -- Use this if your CSV file has a header row, to skip it during import

Checking whether the data is imported in the database:

SELECT \* FROM credit\_card\_approval.updated\_merged\_data LIMIT 10;

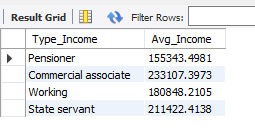


-- 1. Group the customers based on their income type and find the average of their annual income.

SELECT Type\_Income, AVG(Annual\_income) AS Avg\_Income

FROM credit\_card\_approval.updated\_merged\_data

GROUP BY Type\_Income;

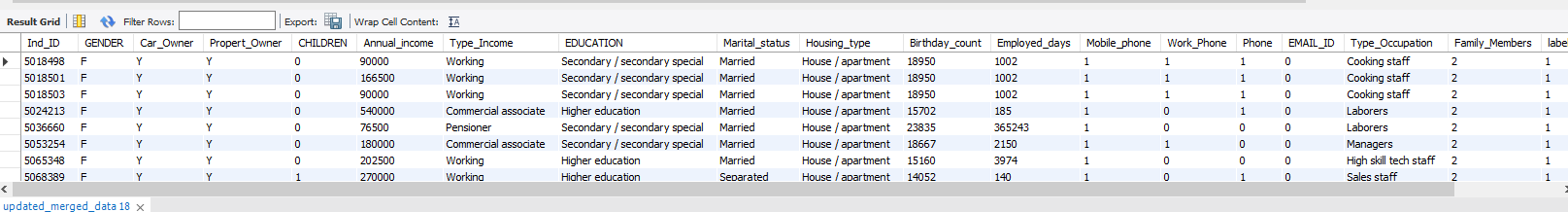


-- 2. Find the female owners of cars and property.

SELECT \*

FROM credit\_card\_approval.updated\_merged\_data

WHERE GENDER = 'F' AND (Car\_Owner = 'Y' AND Propert\_Owner = 'Y');

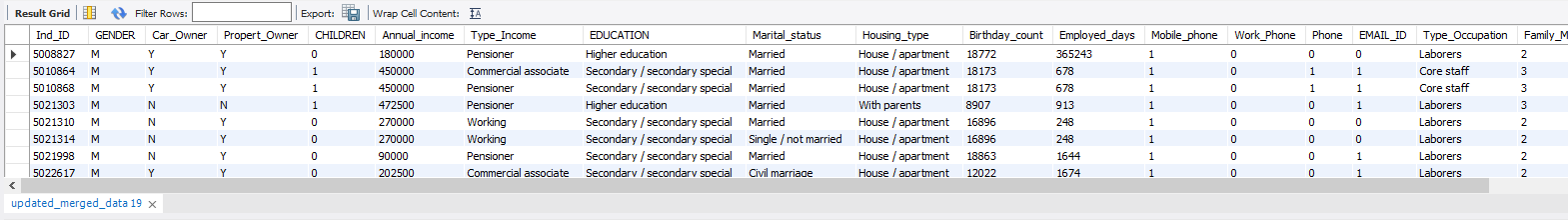


-- 3. Find the male customers who are staying with their families.

SELECT \*

FROM credit\_card\_approval.updated\_merged\_data

WHERE GENDER = 'M' AND Family\_Members > 1;



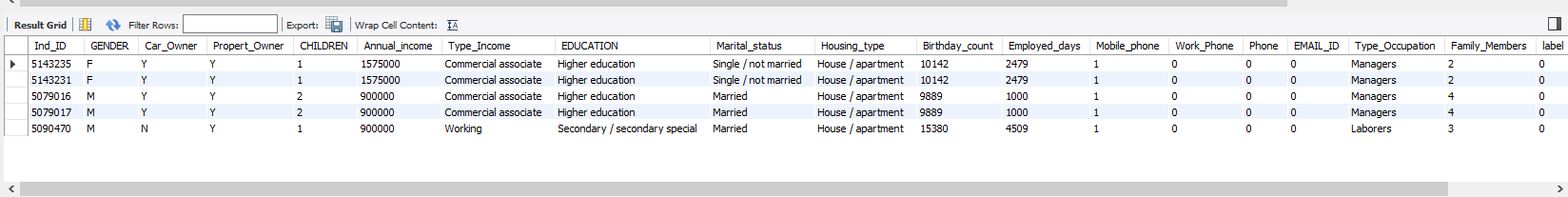
-- 4. List the top five people having the highest income.

SELECT \*

FROM credit\_card\_approval.updated\_merged\_data

ORDER BY Annual\_income DESC

LIMIT 5;

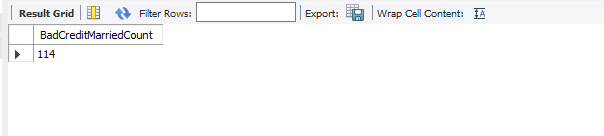


-- 5. How many married people are having bad credit?

SELECT COUNT(\*) AS BadCreditMarriedCount

FROM credit\_card\_approval.updated\_merged\_data

WHERE Marital\_status = 'Married' AND label = '1';



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

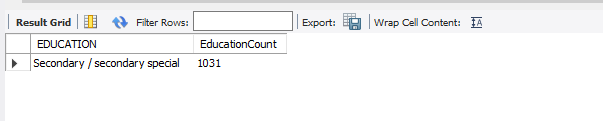
SELECT EDUCATION, COUNT(\*) AS EducationCount

FROM credit\_card\_approval.updated\_merged\_data

GROUP BY EDUCATION

ORDER BY COUNT(\*) DESC

LIMIT 1;



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

SELECT Marital\_status, GENDER, COUNT(\*) AS BadCreditCount

FROM credit\_card\_approval.updated\_merged\_data

WHERE Marital\_status = 'Married' AND label = '1'

GROUP BY Marital\_status, GENDER;

