Python Data Analysis & ML

Credit Card Approval Data Analysis

Andre Tran
Jason Zhao
Justin Chen

Overview

Introduction	01
Our Dataset	02
Data Preprocessing	03
Data Analysis	04
Building an ML Model	05
Model Improvements	06
Conclusion	07
Thank you!	08

Introduction

Credit Cards are used all over the world. Especially in the United States, credits are a part of everyday life. Some of the credit cards in the industry offer exclusive perks such as lounge access, travel credits, and more!

The credit card approval process though is notoriously vague. With some people with worse “stats” getting approved over those with better “stats”. We hope through our investigation to discover which factors are the most influential in determining credit card approval.

Our Dataset

We have used Kaggle as the source of our dataset. Kaggle offers datasets on a wide variety of subjects from business to health and lifestyle. It is optimal for data science and data analytics projects

Dataset Used:

“Credit Card Eligibility Data: Determining Factors

Understanding the Dynamics of Credit Card Eligibility: Insights”

Description

by: Rohit Sharma
Source: https://www.kaggle.com/datasets/rohit265/credit-card-eligibility-data-determining-factors

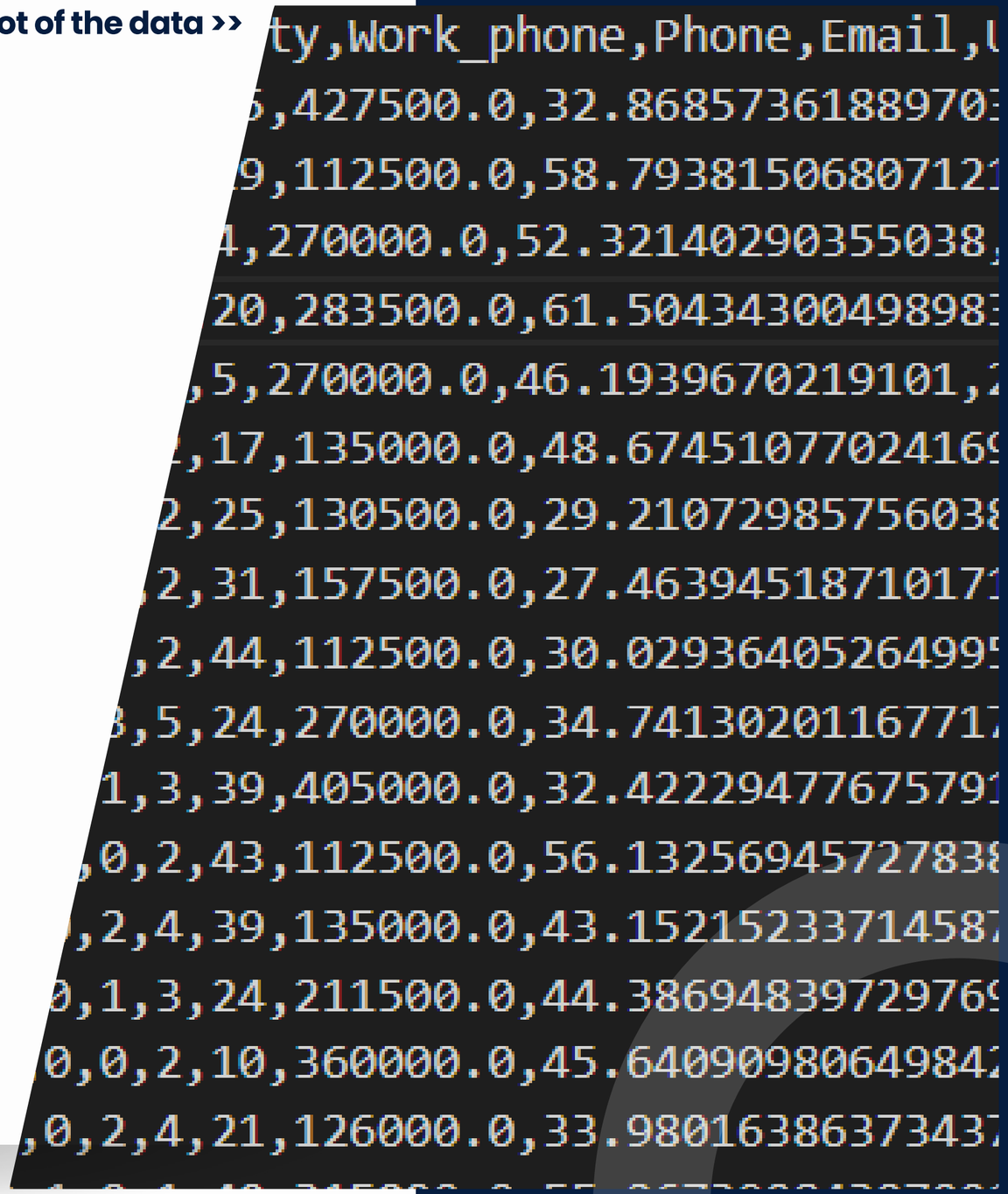
here is a snapshot of the data >>

The Data

There are 20 different data columns, representing 18 factors and 1 result column.

The factors are:

- Gender
- Own_car
- Own_Property
- Work_phone
- Phone
- Email
- Unemployed
- Num_children
- Num_family
- Account_length
- Total_income
- Age
- Years_employed
- Income_type
- Education_type:
- Family_status
- Housing_type
- Occupation_type
- Target(indicates where or not individual is eligible for cc, 1-yes, 0- no)



Data Understanding

At the beginning we try to get a general idea of the data we are working with.

This data is not the most meaningful as the code with perform computations on numbers like gender(0/1) to find the mean where that value is supposed to act more like a string.

```
        ID  Gender  Own_car  Own_property  Work_phone  Phone  Email  \
0  5008804       1        1             1           1      0      0
1  5008806       1        1             1           0      0      0
2  5008808       0        0             1           0      1      1
3  5008812       0        0             1           0      0      0
4  5008815       1        1             1           1      1      1

   Unemployed  Num_children  Num_family  Account_length  Total_income  \
0           0             0           2              15      427500.0
1           0             0           2              29      112500.0
2           0             0           1               4      270000.0
3           1             0           1              20      283500.0
4           0             0           2               5      270000.0

         Age  Years_employed           Income_type  \
0  32.868574       12.435574               Working
1  58.793815        3.104787               Working
2  52.321403        8.353354  Commercial associate
3  61.504343        0.000000             Pensioner
```

```python
import pandas as pd

# Load the dataset
df = pd.read_csv('datasetFinal.csv')

# Display the first few rows of the dataset
print(df.head())

# Get a summary of the dataset
print(df.info())
print(df.describe())
```

Data Preprocessing

We will now:

- handle missing values
- undergo data transformation via One-Hot Encoding

We can tell from our data there are no missing/null values

One Hot Encoding we convert our categorical values of Income, Education, Family, Housing and Occupation into a new binary variable 1/0

```python
# Check for missing values
print(df.isnull().sum())

# Fill missing values or drop rows/columns as appropriate
df.fillna(df.median(), inplace=True)
```

```
ID                 0
Gender             0
Own_car            0
Own_property       0
Work_phone         0
Phone              0
Email              0
Unemployed         0
Num_children       0
Num_family         0
Account_length     0
Total_income       0
Age                0
Years_employed     0
Income_type        0
Education_type     0
Family_status      0
Housing_type       0
Occupation_type    0
Target             0
dtype: int64
```

```python
# Convert categorical variables to numerical values using one-hot encoding
df = pd.get_dummies(df, columns=['Income_type', 'Education_type', 'Family_status', 'Housing_type',
                                 'Occupation_type'], drop_first=True)

# Print the first few rows to check the transformation
print(df.head())
```

```
   Occupation_type_Sales staff  Occupation_type_Secretaries  \
0                        False                        False
1                        False                        False
2                         True                        False
3                        False                        False
4                        False                        False

   Occupation_type_Security staff  Occupation_type_Waiters/barmen staff
0                           False                                False
1                            True                                False
2                           False                                False
3                           False                                False
4                           False                                False
```

Scaling Numerical Values

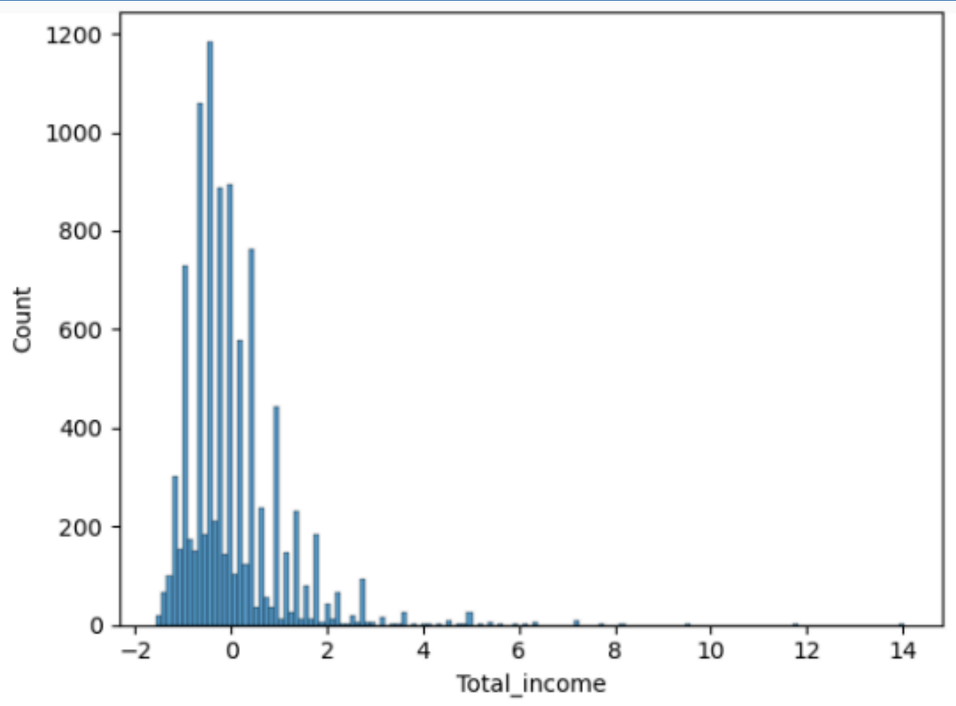
Now we scale numerical variables to ensure they all contribute equally to the model's performance

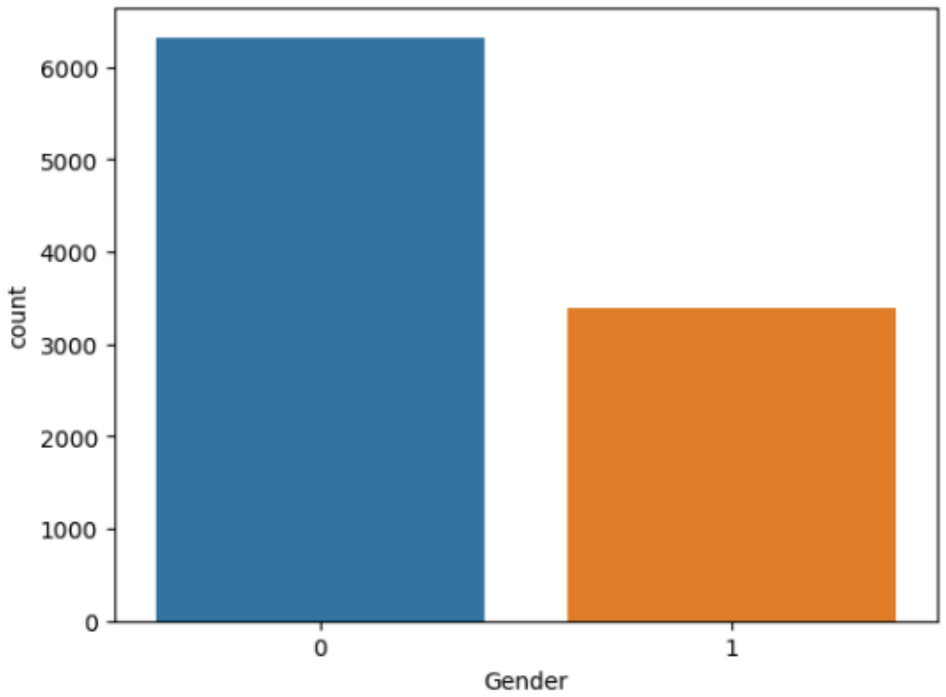
```
from sklearn.preprocessing import StandardScaler

# List of numerical columns to scale
num_cols = ['Total_income', 'Age', 'Years_employed', 'Num_children', 'Num_family', 'Account_length']

# Initialize the scaler
scaler = StandardScaler()

# Scale the numerical columns
df[num_cols] = scaler.fit_transform(df[num_cols])
```

Exploratory Data Analysis(EDA)

We first do univariate analysis to see the distribution of income among all the credit card applicants. Then we can see the distribution of applicants based on gender

```
import matplotlib.pyplot as plt
import seaborn as sns

# Plot the distribution of 'Total_income'
sns.histplot(df['Total_income'])
plt.show()

# Plot the count of 'Gender'
sns.countplot(x='Gender', data=df)
plt.show()
```





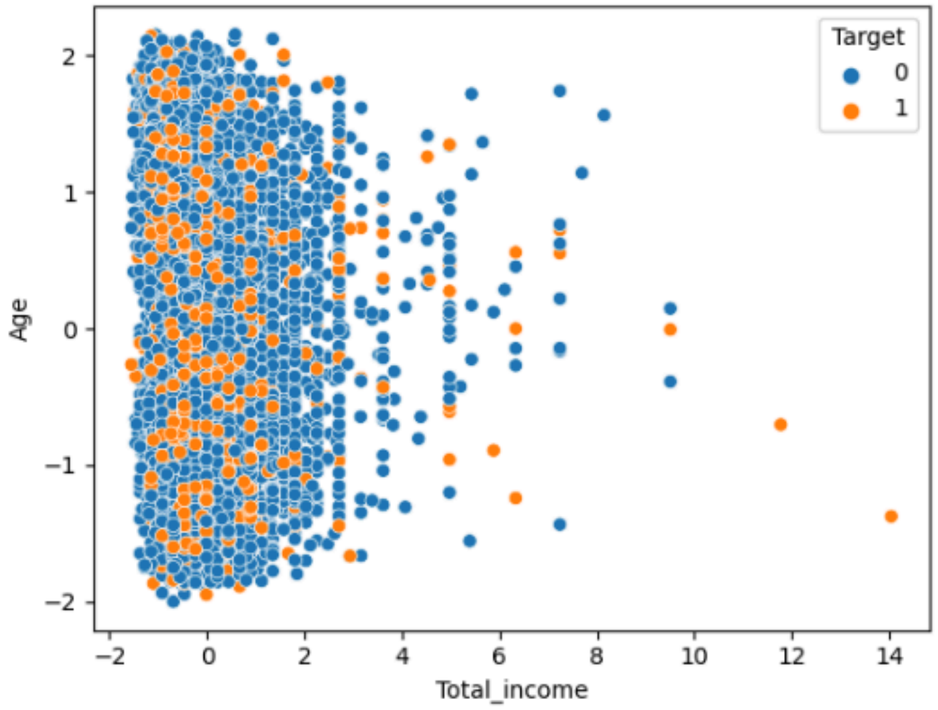
Bivariate Analysis

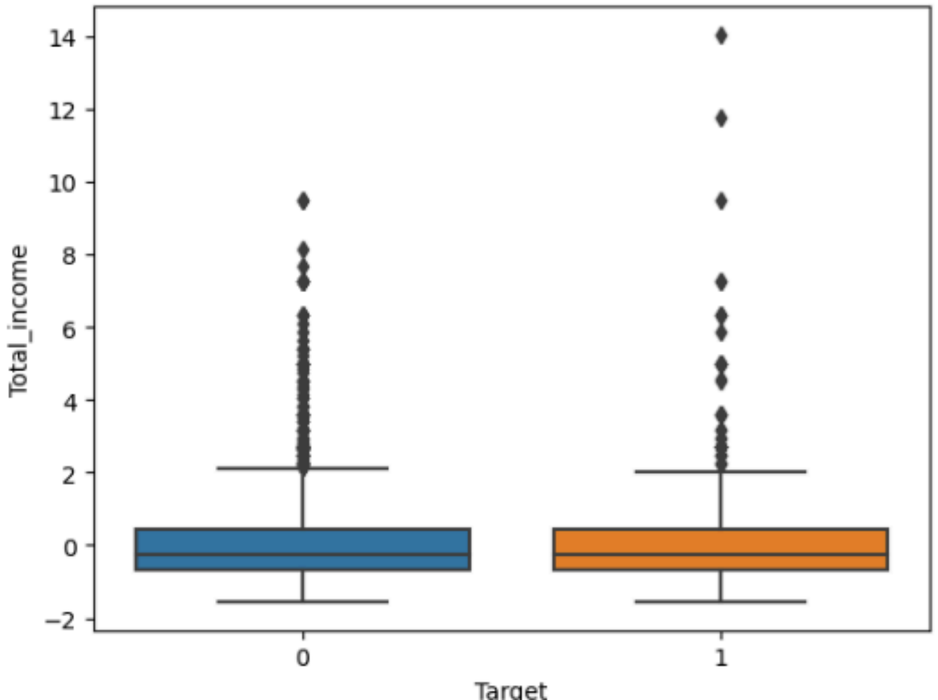
Now we do bivariate data analysis to see the relation between a pair of variables.

- Total income vs. Age
- Target vs. Total Income

*Remember target determines whether or not an applicant was accepted. 1 means the applicant was approved for the credit card. As we can see from the histogram, on average, those with higher incomes will tend to be accepted at higher rates than those with lower incomes.

```
Plot the relationship between 'Total_income' and 'Age'
ns.scatterplot(x='Total_income', y='Age', hue='Target', data=df)
lt.show()

Plot the relationship between 'Total_income' and 'Target'
ns.boxplot(x='Target', y='Total_income', data=df)
lt.show()
```





Model Building

For building a Machine Learning Model, we need to go through two steps:

- Feature Selection
- Model Selection and Training

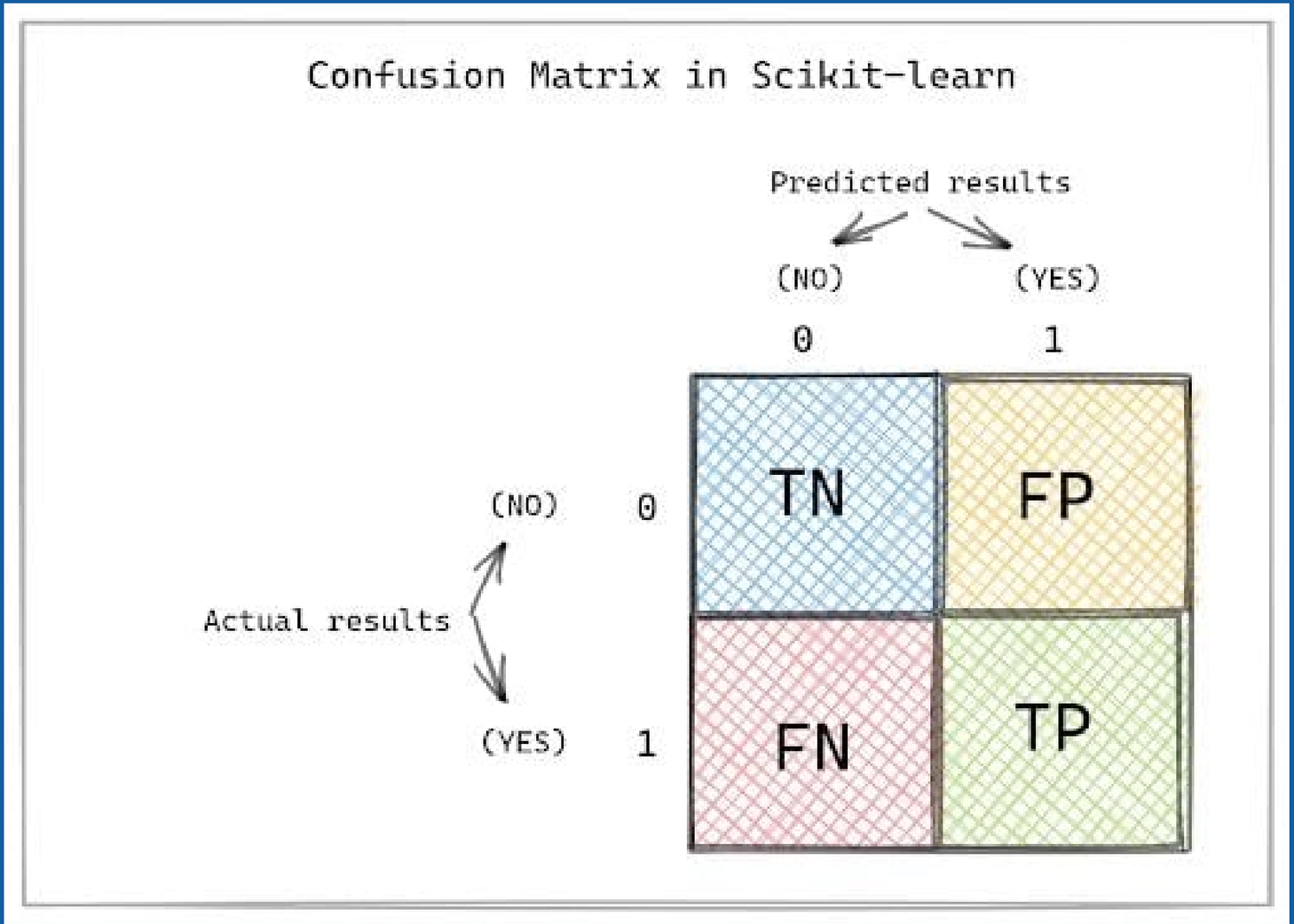
Feature Selection

```
# Plot the relationship between 'Total_income' and 'Target'
sns.boxplot(x='Target', y='Total_income', data=df)
plt.show()

# Select features and target variable
X = df.drop('Target', axis=1)
y = df['Target']
```

Model Selection and Training

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Initialize and train the model
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)
```

Model Evaluation



Model Evaluation

Now we evaluate the effectiveness of our model. From the matrix we can tell:

- True Negatives (TN): 2498
- False Positives (FP): 1
- False Negatives (FN): 412
- True Positives (TP): 2

This matrix indicates:

- The model correctly identified 2498 out of 2499 individuals who are not eligible for a credit card (very high true negatives).
- The model incorrectly identified 1 individual as eligible when they are not (very low false positives).
- The model incorrectly identified 412 individuals as not eligible when they are (high false negatives).
- The model correctly identified only 2 out of 414 individuals who are eligible for a credit card (very low true positives).

```
[[2498    1]
 [ 412    2]]
              precision    recall  f1-score   support

           0       0.86      1.00      0.92      2499
           1       0.67      0.00      0.01       414

    accuracy                           0.86      2913
   macro avg       0.76      0.50      0.47      2913
weighted avg       0.83      0.86      0.79      2913
```

```python
# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

Improving the Model

The confusion matrix displays some imbalance with a high ability to determine true negatives but poorly able to determine true positives.

To adjust the imbalance we use Gradient Boost which raises the number of accurately diagnosed true positives.

The number of false negatives decreased and true positives increased.

```python
from sklearn.ensemble import GradientBoostingClassifier

# Training the GradientBoostingClassifier on the balanced data
gb_clf = GradientBoostingClassifier(random_state=42)
gb_clf.fit(X_train, y_train)

# Predicting on the test set
y_pred_gb = gb_clf.predict(X_test)

# Evaluating the model
print(confusion_matrix(y_test, y_pred_gb))
print(classification_report(y_test, y_pred_gb))
```

```
[[2492    7]
 [ 408    6]]
              precision    recall  f1-score   support

           0       0.86      1.00      0.92      2499
           1       0.46      0.01      0.03       414

    accuracy                           0.86      2913
   macro avg       0.66      0.51      0.48      2913
weighted avg       0.80      0.86      0.80      2913
```

Hyperparameter Tuning

We can further improve the model by using hyperparameter tuning with GridSearchCV

(1) Grid Search involves specifying a grid of hyperparameters (a list of hyperparameter values to try for each hyperparameter). The method then creates a combination of all the possible hyperparameter values.

(2) For each combination of hyperparameters, the model is then trained and validated using cross-validation (CV)

The model trained with the optimal hyperparameters is outputted. You can see from the new confusion matrix that by using hyperparameter tuning, we've produced an even more optimized Gradient Boosting model

```python
#%% Hyperparameter Tuning with GridSearchCV
from sklearn.model_selection import GridSearchCV

param_grid = {
    'n_estimators': [100, 200],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 4, 5]
}

grid_search = GridSearchCV(GradientBoostingClassifier(random_state=42), param_grid, cv=5, scoring='f1')
grid_search.fit(X_train, y_train)
best_gb_model = grid_search.best_estimator_
```

```
[[2465   34]
 [ 394   20]]
              precision    recall  f1-score   support

           0       0.86      0.99      0.92      2499
           1       0.37      0.05      0.09       414

    accuracy                           0.85      2913
   macro avg       0.62      0.52      0.50      2913
weighted avg       0.79      0.85      0.80      2913
```

Evaluating other Models

We played around with some other models to determine if these models could produce better outputs. The other models we tested were:

1. Logistic Regression (fourth place)
2. k_NN (second place)
3. Gaussian Naive Bayes (last place)
4. Decision Tree (best results)
5. XGBoost (third place)

SVC was ruled out as the SVM training took too long, and was only (practically) runnable after changes were done such as reducing dimensionality

Multinomial Naive Bayes was not used as the classifier was not designed to receive datasets with negative values

Should Gradient Boosting be included in the rankings, it would have replaced XGBoost for third place

#1

```
Logistic Regression Model

Best score found by GridSearchCV for LogisticRegression:
0.05286311347925569

[[2499    0]
 [ 406    8]]
              precision    recall  f1-score   support

           0       0.86      1.00      0.92      2499
           1       1.00      0.02      0.04       414

    accuracy                           0.86      2913
   macro avg       0.93      0.51      0.48      2913
weighted avg       0.88      0.86      0.80      2913
```

#2

```
Best score found by GridSearchCV for KNeighborsClassifier:
0.10678138180882069

[[2316  183]
 [ 367   47]]
              precision    recall  f1-score   support

           0       0.86      0.93      0.89      2499
           1       0.20      0.11      0.15       414

    accuracy                           0.81      2913
   macro avg       0.53      0.52      0.52      2913
weighted avg       0.77      0.81      0.79      2913
```

#3

```
Gaussian Naive Bayes Model

Best score found by GridSearchCV for GaussianNB:
0.004571428571428571

[[2498    1]
 [ 413    1]]
              precision    recall  f1-score   support

           0       0.86      1.00      0.92      2499
           1       0.50      0.00      0.00       414

    accuracy                           0.86      2913
   macro avg       0.68      0.50      0.46      2913
weighted avg       0.81      0.86      0.79      2913
```

#4

```
Decision Tree Model

Best score found by GridSearchCV for DecisionTreeClassifier:
0.16545094761091117

[[2162  337]
 [ 335   79]]
              precision    recall  f1-score   support

           0       0.87      0.87      0.87      2499
           1       0.19      0.19      0.19       414

    accuracy                           0.77      2913
   macro avg       0.53      0.53      0.53      2913
weighted avg       0.77      0.77      0.77      2913
```

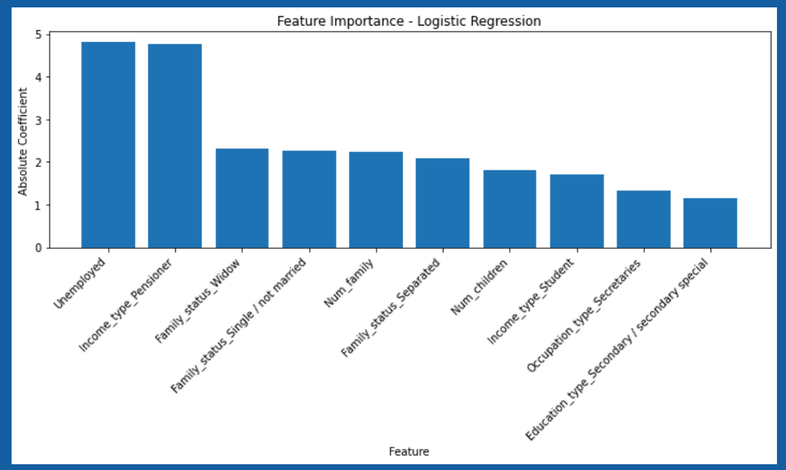
#5

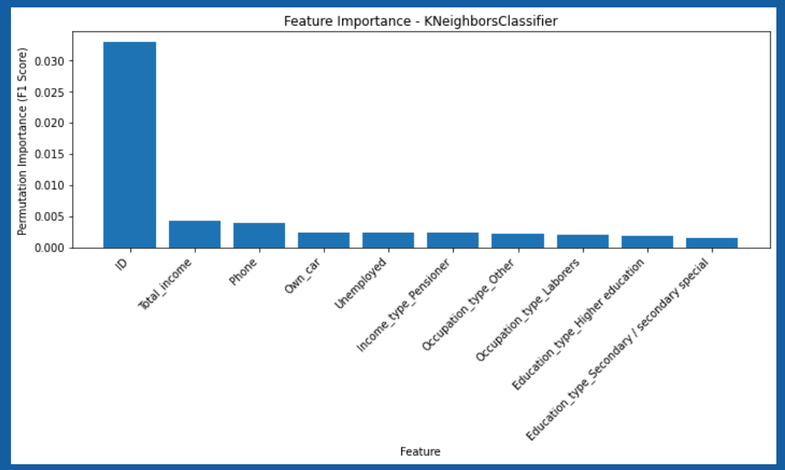
```
XGBoost Model

Best score found by GridSearchCV for XGBoost:
0.08008587329095276

[[2480   19]
 [ 404   10]]
              precision    recall  f1-score   support

           0       0.86      0.99      0.92      2499
           1       0.34      0.02      0.05       414

    accuracy                           0.85      2913
   macro avg       0.60      0.51      0.48      2913
weighted avg       0.79      0.85      0.80      2913
```

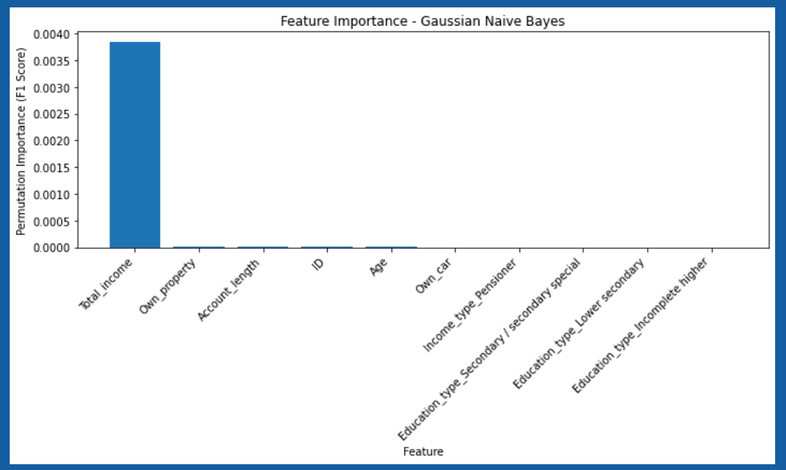
```
Best score found by GridSearchCV for GradientBoostingClassifier:
0.08334584667488885

[[2465   34]
 [ 394   20]]
              precision    recall  f1-score   support

           0       0.86      0.99      0.92      2499
           1       0.37      0.05      0.09       414

    accuracy                           0.85      2913
   macro avg       0.62      0.52      0.50      2913
weighted avg       0.79      0.85      0.80      2913
```

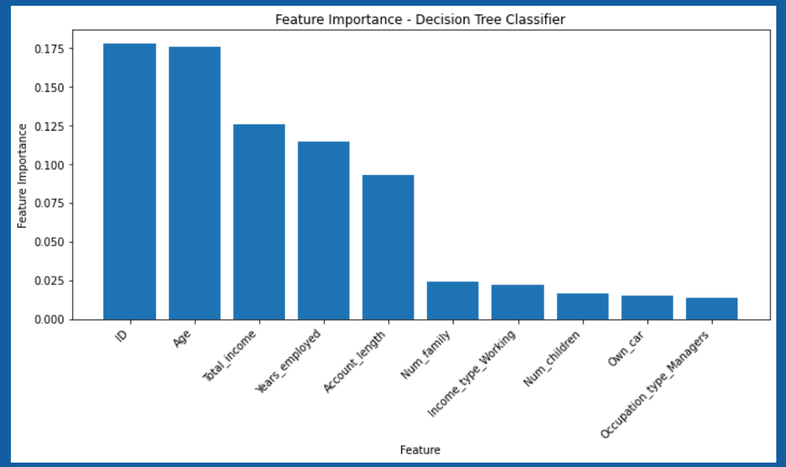
Feature Importance Analysis

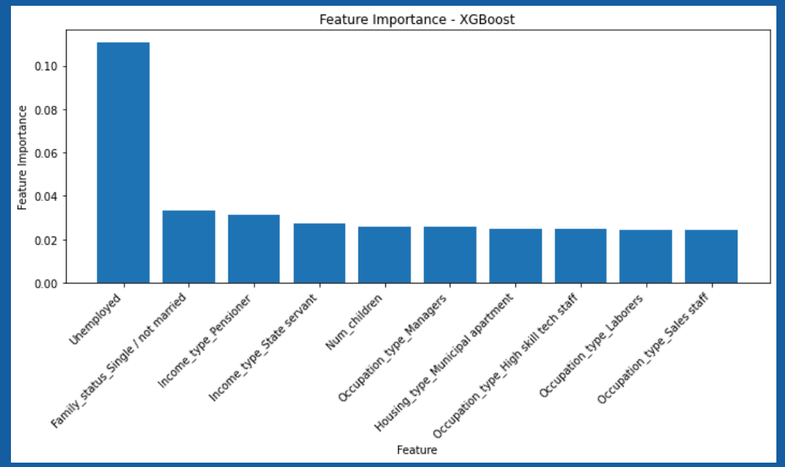
Using each of the models, we used feature importance analysis determine which features are most influential in predicting credit card approval.

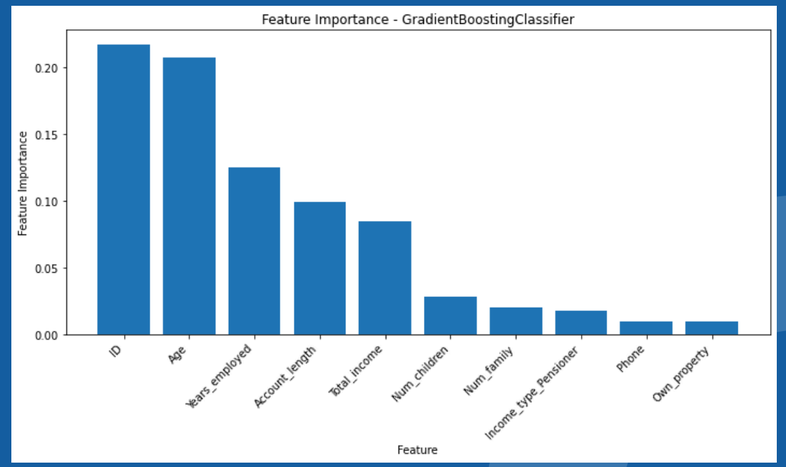












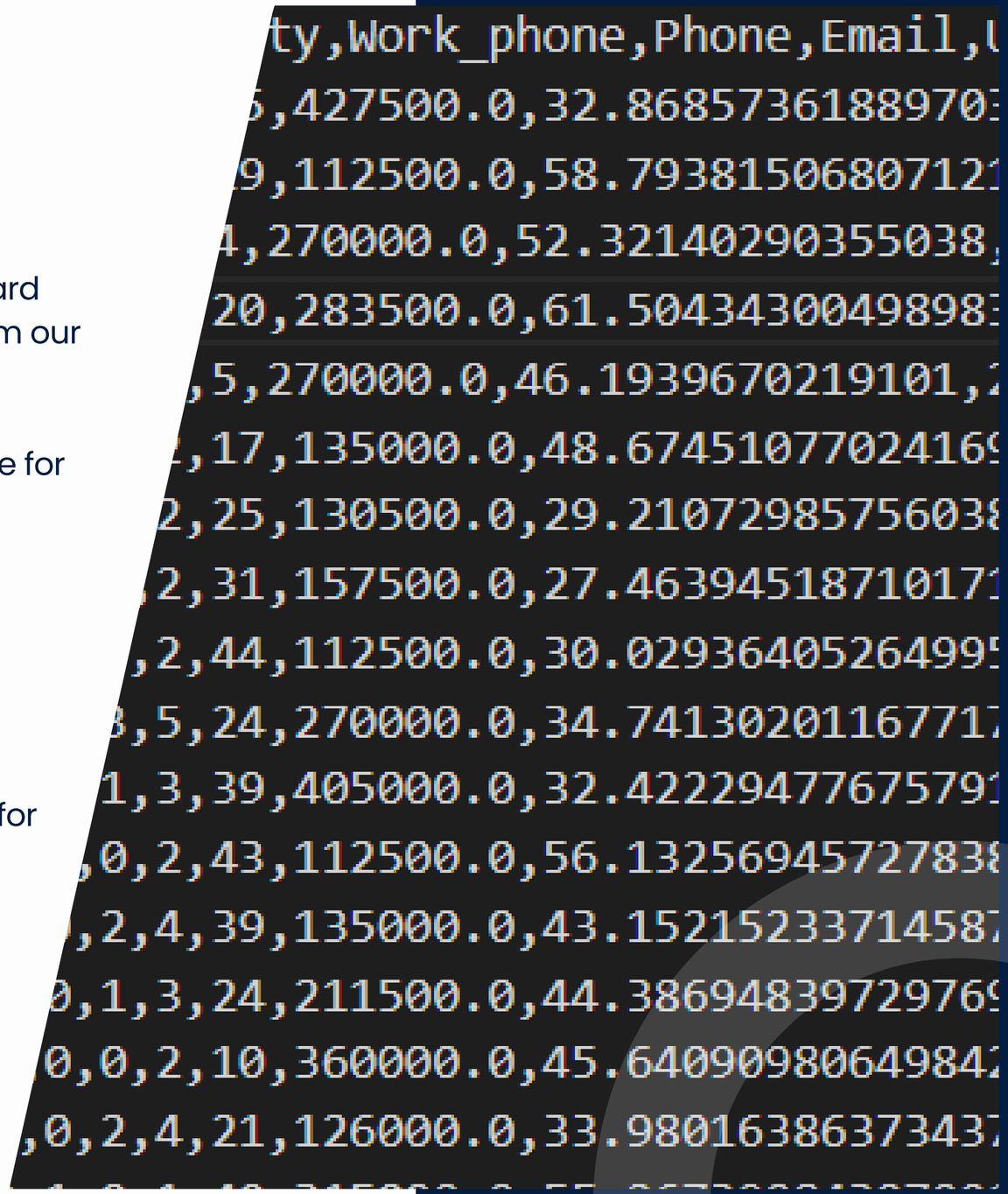
Takeways from the Model

We have identified a model that is better at identifying the credit card application result of an applicant based on various data points from our observations.

These factors increase the likelihood of the target being true(eligible for card):

- The higher the account length the better
- The better the total income the better
- The higher the years employed the better
- The higher the education type the better
- Being employed

These are potential considerations for those interested in applying for and getting approved for credit cards in the future.



Hypothesis Testing

Because we were curious, we decided to do some hypothesis testing.

Using f_oneway we were able to find the p-values comparing the rejected credit card applicants with the accepted credit card applicants.

We concluded that there is no statistically significant difference in credit card eligibility for gender, but there is a statistically significant difference in credit card eligibility when considering bank account length.

```
# Hypothesis Testing
# Null Hypothesis: There is no difference in credit card eligibity between genders.
# Alternative Hypothesis: There is a difference in credit card eligibility between genders.
```

```
from scipy.stats import f_oneway

zero_target = df[df['Target'] == 0]
one_target = df[df['Target'] == 1]

def one_way_anova(col):
    f_value, p_value = f_oneway(zero_target[col], one_target[col])
    return p_value
```

```
one_way_anova('Gender')
```

```
0.13857045412274227
```

```
#Since the p-value is greater than 0.05, we fail to reject the null hypothesis, meaning there is statistically no difference
#in credit card eligibility between genders.
```

```
# Null Hypothesis: There is no difference in credit card eligibility when considering account length.
# Alternative Hypothesis: There is a difference in credit card eligibility when considering account length.
```

```
one_way_anova('Account_length')
```

```
2.0714630879665047e-14
```

```
#Since the p-value is less than 0.05, we can reject the null hypothesis, meaning there is a statistically significant difference
#in credit card eligibility when considering applicants' account length.
```

Arowwai
Industries

THANK YOU!