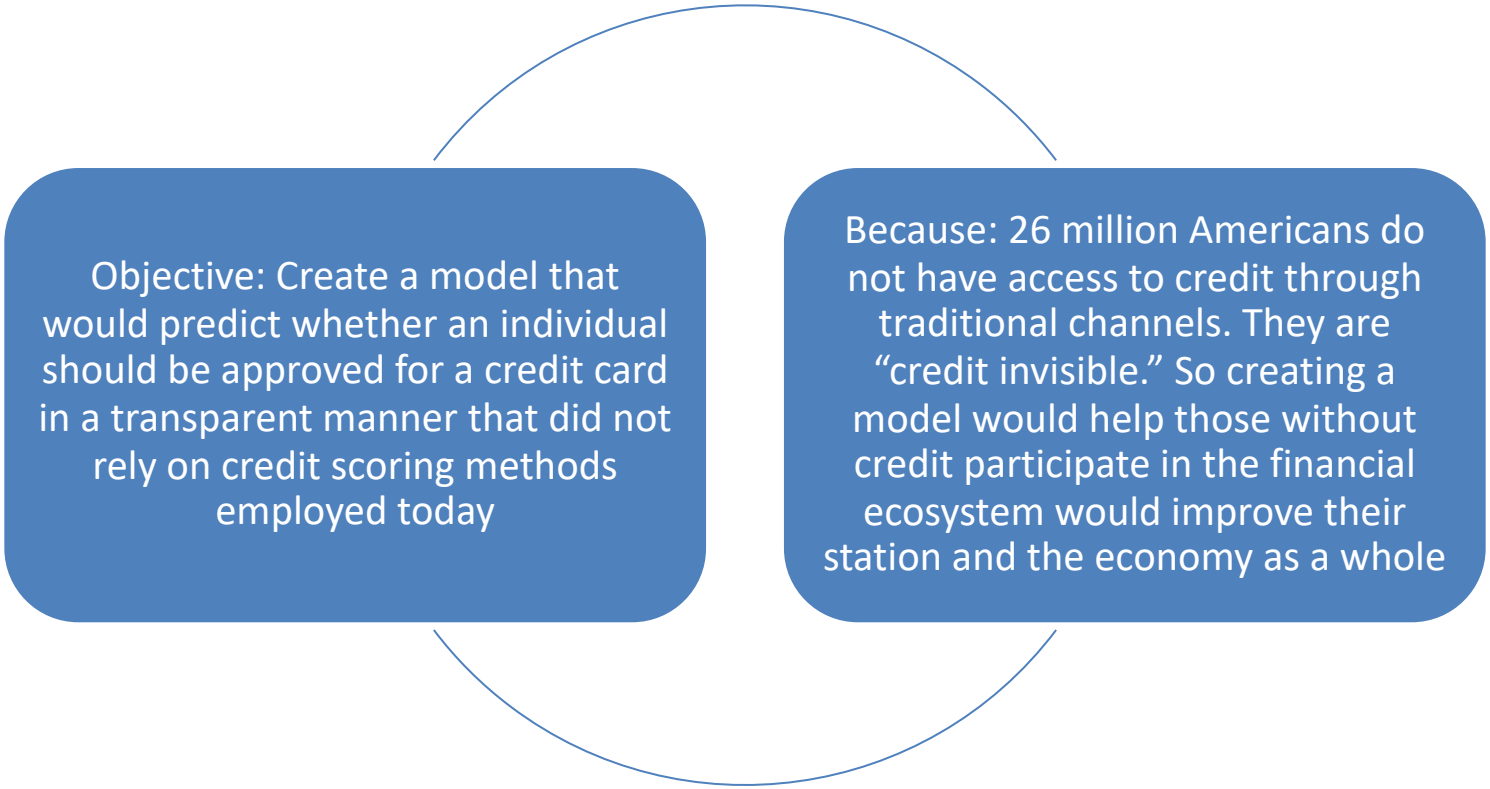
Team Members:

04/18/2024

Building a Machine Learning Model for Predicting Credit Payment Delinquency

Our Goal


Objective: Create a model that would predict whether an individual should be approved for a credit card in a transparent manner that did not rely on credit scoring methods employed today
Because: 26 million Americans do not have access to credit through traditional channels. They are “credit invisible.” So creating a model would help those without credit participate in the financial ecosystem would improve their station and the economy as a whole

Is this already being done?

- Literature:
  - Industry Shift : Research shows a trend toward incorporating non-traditional data into credit models for more inclusive financial assessments globally.
- Our Innovation:
  - Transparent Modeling: Unlike "black box" models, our approach ensures comprehensible outcomes, detailing the impact of late payments and recent financial behavior.

First things First

Credit Reporting Agency Insights:

"Black Box" Models: Current approval models lack transparency, making output from the models difficult to interpret.

Impact of Late Payments: Credit scores are significantly affected by late payments, with recentness adding to the severity.

Banking Industry Insights:

Communication Challenges: Banks face challenges in clearly explaining loan approval or denial to customers.

Credit Score Influence: Emphasized the heavy reliance on credit scores for loan decision-making.

Actionable Takeaways

- Transparent Model Development:
  - Innovate credit approval models to be transparent, ensuring consumers understand what factors affect their credit outcomes.
- Payment History Prioritization:
  - Emphasize the significance of payment timeliness in consumer education and in the criteria of credit evaluation models.

Get Data (Kaggle)

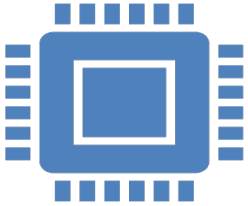
Dataset 1: Consumer Application

- Description: The dataset contains data for 430,900 applicants, represented across 18 columns, detailing demographics, financial behavior, and past credit history, which are critical for predicting credit card approval status.

Dataset 2: Credit Records

- Description : Holds a little more than 1,000,000 records regarding payment history information, including whether or not an account was paid, and when that payment was made (or not).

Challenges and Solutions



The Challenges

No label: there was no column given to act as a label for our predictive model

Lack of mission critical information: credit scores

Missing Data: Significant amounts of missing data in key columns which complicate the predictive modeling.

Outliers: Presence of extreme values in income which could skew results.

Imbalance: Unbalanced data with respect to the target variable, which may affect model performance.

The Solution

Feature Engineering: Creating a new variable that could serve as a proxy for their credit score, and serve as a reliable label for the creation of a predictive model.

Data Cleaning: Removing unnecessary columns and outliers to improve model reliability.

Feature Engineering: Enhancing data with techniques like PCA and categorical encoding to better capture the essence of the data.

Balancing our Dataset: Using techniques like oversampling and aggregation to handle imbalances in the dataset.

Building Our Model

Data Cleaning and Preprocessing

Handling Missing Values:

- First Dataset: Removed a column with 33% missing values due to low correlation with the target variable, preserving dataset integrity.
- Second Dataset: Removed rows containing missing values for the MONTHS_BALANCE column which will be used as in feature engineering for our the target variable, avoiding unreliable imputation.

Outlier Removal:

- Focused on "AMT_INCOME_TOTAL" column, removing extreme income outliers exceeding the 75th percentile to prevent model overfitting and bias, enhancing prediction accuracy and fairness.

Feature Engineering:

- Aggregation: Consolidated credit records per applicant for a clearer creditworthiness analysis.
- Binning: Reduced the granularity of the MONTHS_BALANCE_mean into meaningful categories (based on IQR due to skew); thereby reducing noise, removing outliers, and improving interoperability.
- PCA: Applied to highly correlated dummy variables to enhance model efficiency.
- Encoding: Used ordinal and one-hot encoding for categorical variables to ensure compatibility with machine learning models.

Data Transformation:

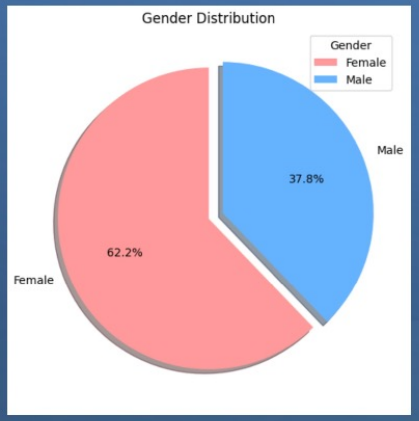
- Emphasized standardizing data over normalizing, given our normal distribution after removing outliers.
- This decision was less critical, because tree based algorithms (our model of choice) are less sensitive to scale, or distribution for the features.

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from datetime import timedelta
from scipy.stats import norm
import math
import seaborn as sns
import statsmodels.api as sm
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from scipy.stats import randint
from scipy.stats import uniform
from io import StringIO
from sklearn.impute import SimpleImputer

myDF0 = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/Project/archive/application_record (1).csv")
myDF1 = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/Project/archive/credit_record.csv")

myDF0.head()

myDF1.head()

# Example: Aggregating multiple credit records for each ID
agg_rules = {
    'MONTHS_BALANCE': ['min', 'max', 'mean'],  # Numeric column example
    'STATUS': ['max']  # Categorical column example, assuming 'STATUS' indicates delinquency level
}
credit_summary = myDF1.groupby('ID').agg(agg_rules).reset_index()
# Flatten MultiIndex in columns
credit_summary.columns = ['_'.join(col).strip() for col in credit_summary.columns.values]

# Renaming 'ID_' to 'ID' for consistency before merging
credit_summary.rename(columns={'ID_': 'ID'}, inplace=True)

credit_summary.head()

credit_summary.isnull().sum()

myDF2 = myDF0.merge(credit_summary, how = 'left', on = ['ID'])

myDF2.head()

myDF2.describe()

# Calculate IQR for AMT_INCOME_TOTAL
Q1 = myDF2['AMT_INCOME_TOTAL'].quantile(0.25)
Q3 = myDF2['AMT_INCOME_TOTAL'].quantile(0.75)
IQR = Q3 - Q1

# Define lower and upper bounds for outliers
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
```

```python
# Remove outliers and overwrite myDF2 with the filtered dataset
myDF2 = myDF2[(myDF2['AMT_INCOME_TOTAL'] >= lower_bound) & (myDF2['AMT_INCOME_TOTAL'] <= upper_bound)]

# Print the new shape of myDF2 to see how many rows were removed
print('New shape after removing outliers:', myDF2.shape)

myDF2.head()

myDF2.isnull().sum()

print('Application Record data shape: ',myDF0.shape)
print('Credit Record data shape: ',myDF1.shape)
print('Aggregated Credit Record data shape: ',credit_summary.shape)
print('Merged data shape: ',myDF2.shape)

# Define a function for custom binning that returns numeric codes
def custom_bin_numeric(months_balance):
    if months_balance <= -60:
        return 4  # "12+ Months Past"
    elif months_balance > -13.5 and months_balance <= -23:
        return 3  # "6-11 Months Past"
    elif months_balance > -6 and months_balance <= -13.5:
        return 2  # "1-5 Months Past"
    elif months_balance < -6:
        return 1  # "Current Month"
    else:
        return 0  # Assuming positive values are unexpected or represent a default group

# Apply custom binning
myDF2['Balance_Numeric_Category'] = myDF2['MONTHS_BALANCE_mean'].apply(custom_bin_numeric)

print('Merged data shape: ',myDF2.shape)

myDF2.isnull().sum()

myDF2.drop('OCCUPATION_TYPE',axis=1,inplace=True)

myDF2 = myDF2.dropna()
myDF2.isnull().sum()

print('DF2 data shape: ',myDF2.shape)

size_mapping = {
'5': 7,
'4': 6,
'3': 5,
'2': 4,
'1': 3,
'0': 2,
'X': 1,
'C': 0}
myDF2['STATUS_max'] = myDF2['STATUS_max'].map(size_mapping)

myDF2['Proxy_Late_status'] = (myDF2['STATUS_max'] >= 5).astype(int)
```

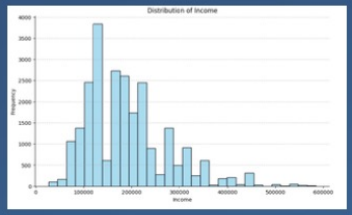
Exploratory Data Analysis (EDA)

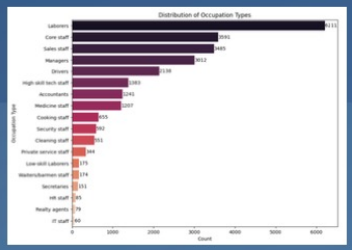
- Exploratory Data Analysis
  - Numerical Features: Analyzed income distributions using histograms and descriptive statistics.
  - Categorical Variables: Examined variables like gender and car ownership using value counts, bar graphs, and pie charts.
  - Relationships: Utilized heatmaps to assess correlations and cross-tabulations with chi-squared tests for associations; measured strength with Cramer's V.
  - Multicollinearity: Identified potential multicollinearity issues among features using the Variance Inflation Factor (VIF).
  - Feature Importance: Employed a Random Forest model to ascertain the most influential variables in predicting credit outcomes.

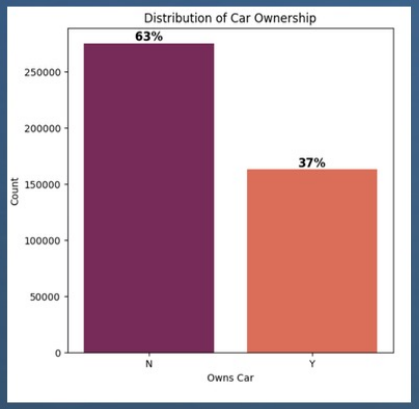


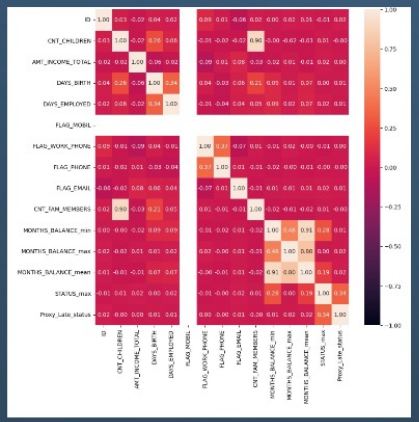

```
# Convert 'OCCUPATION_TYPE' to dummy variables
occupation_dummies = pd.get_dummies(myDF2['OCCUPATION_TYPE'])

# Choose a method of correlating these with 'STATUS', for example, taking the mean of 'STATUS' for each categ
for column in occupation_dummies.columns:
    print(f"Correlation between STATUS and {column}: {occupation_dummies[column].corr(myDF2['STATUS_max'])}")
```

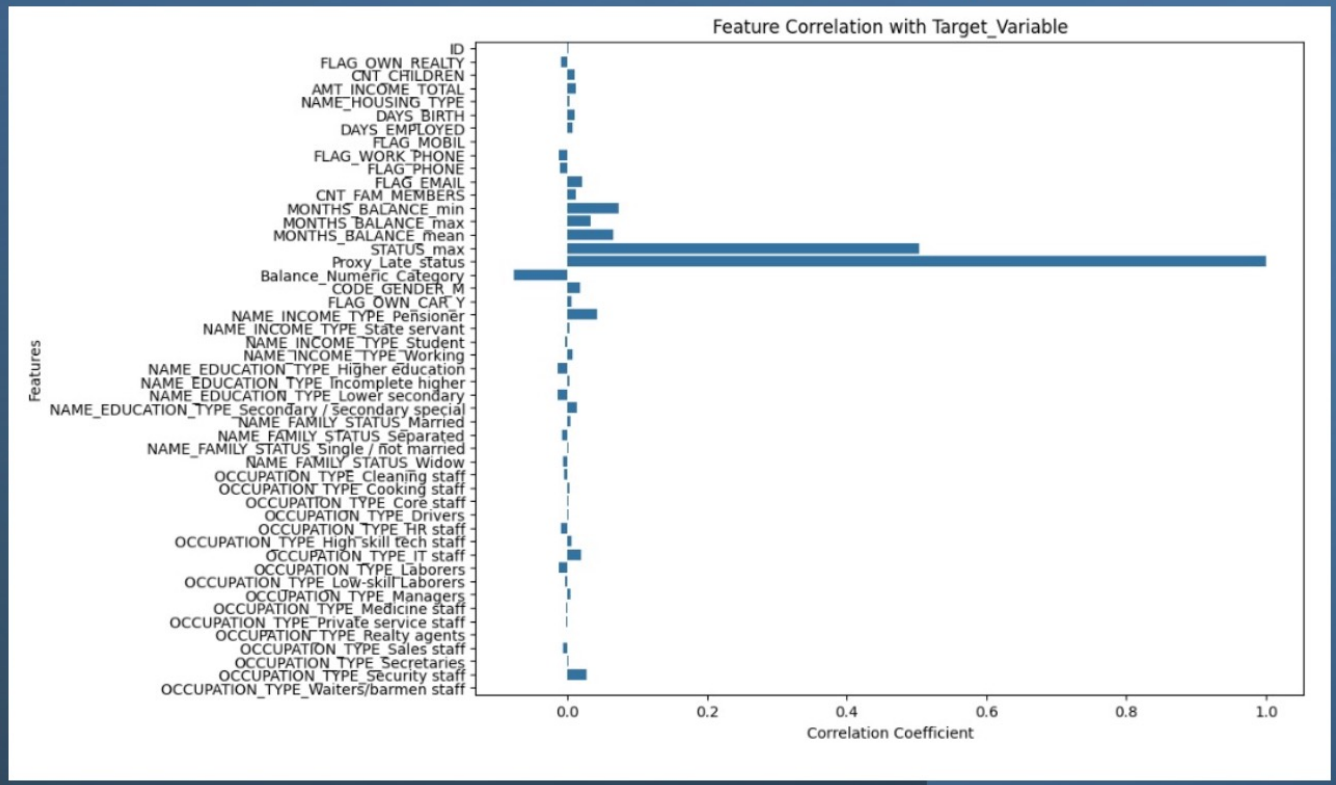
```
Correlation between STATUS and Accountants: 0.005289736184141631
Correlation between STATUS and Cleaning staff: 0.0071129547826333205
Correlation between STATUS and Cooking staff: 0.014192038974256745
Correlation between STATUS and Core staff: 0.007204558674580062
Correlation between STATUS and Drivers: 0.001968888706256736
Correlation between STATUS and HR staff: -0.008179911784773584
Correlation between STATUS and High skill tech staff: -0.0022355886825467246
Correlation between STATUS and IT staff: 0.006760086969219528
Correlation between STATUS and Laborers: -0.021799815419437055
Correlation between STATUS and Low-skill Laborers: -0.005186401147591959
Correlation between STATUS and Managers: 0.007612445358260752
Correlation between STATUS and Medicine staff: -0.021260236218097934
Correlation between STATUS and Private service staff: 0.005862050438523463
Correlation between STATUS and Realty agents: 0.007712459412395175
Correlation between STATUS and Sales staff: 0.0021835338956760282
Correlation between STATUS and Secretaries: -0.001930215480009887
Correlation between STATUS and Security staff: 0.01957301102938077
Correlation between STATUS and Waiters/barmen staff: 0.004303791630213245
```











Feature Selection

- Feature X Selection:
  - Influenced by domain knowledge,
  - Random Forest analysis, and
  - Feature correlation.
- Feature Y Selection:
  - Based on consultations with banking industry experts and credit reporting agencies.
  - Emphasis on the significance of recent late payments on credit card approval and credit score impact.
  - Recognized as a major influencer on credit scores.

Splitting the Dataset

Standard Split Ratio: 70% training, 30% test.

Training Set: Used for initial model training.

Test Set: Evaluates final model performance.

Special Consideration:

Due to the presence of an unbalanced dataset, we implemented oversampling techniques to ensure a fair representation of all classes in the training process.

```python
myDF2['Proxy_Late_status'] = (myDF2['STATUS_max'] >= 5).astype(int)

# Creating a new column based on conditions directly with pandas
myDF2['Target_Variable'] = ((myDF2['Proxy_Late_status'] == 1) & (myDF2['MONTHS_BALANCE_mean'] <= 2)).astype(int)

# Count the frequency of rows that meet both conditions
frequency = sum((myDF2['Proxy_Late_status'] == 1) & (myDF2['MONTHS_BALANCE_mean'] <= 2))

print("Frequency of rows that meet both conditions:", frequency)

# Count the frequency of rows that do not meet both conditions
frequency = sum(~((myDF2['Proxy_Late_status'] == 1) & (myDF2['MONTHS_BALANCE_mean'] <= 2)))

print("Frequency of rows that do not meet both conditions:", frequency)

myDF2.Target_Variable

X_categorical = pd.get_dummies(myDF2[['CODE_GENDER', 'FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'NAME_FAMILY_STATUS']])
X_numerical = myDF2[['CNT_CHILDREN', 'AMT_INCOME_TOTAL']]

X = pd.concat([X_numerical, X_categorical], axis=1)
y = myDF2.Target_Variable

X = X.drop(['CODE_GENDER_M', 'FLAG_OWN_CAR_Y', 'FLAG_OWN_REALTY_Y'], axis=1)

from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# Selecting the family status columns from X
family_status_features = X[['NAME_FAMILY_STATUS_Civil marriage',
                            'NAME_FAMILY_STATUS_Married',
                            'NAME_FAMILY_STATUS_Separated',
                            'NAME_FAMILY_STATUS_Single / not married',
                            'NAME_FAMILY_STATUS_Widow']]
```

```python
# Standardizing the family status features
scaler = StandardScaler()
family_status_std = scaler.fit_transform(family_status_features)

# Applying PCA to the family status features
pca = PCA(n_components=1)  # Change this if you want more components
family_status_pca = pca.fit_transform(family_status_std)

# Adding the PCA component back to the X DataFrame as a new column
X['Family_Status_PCA'] = family_status_pca

# Now drop the original family status columns from X
X.drop(columns=family_status_features.columns, inplace=True)

X.head()

print('Predictor data shape: ',X.shape)

print('Target data shape: ',y.shape)

from sklearn.model_selection import train_test_split
from imblearn.over_sampling import RandomOverSampler

# Split the original dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)

# Apply oversampling to the training set only
oversampler = RandomOverSampler(random_state=0)
X_train, y_train = oversampler.fit_resample(X_train, y_train)  # Reassign the oversampled data back

# Now, X_train and y_train contain the oversampled training data

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)
```

Model Selection Based on Problem

Start with simpler models for beginners:

Logistic Regression: Effective for binary outcomes; provides probability scores for observations.

Decision Trees: Easy to interpret; handles both numerical and categorical data.

Progress to more complex models for improved accuracy:

Random Forest: Reduces overfitting by averaging multiple decision trees.

Gradient Boosting: Sequentially builds small trees to correct errors; highly effective for varied datasets.

XGBoost: Optimized gradient boosting with faster performance and high efficiency.

Support Vector Machines (SVM): Performs well in high-dimensional spaces; suitable for complex classification boundaries.

Our Models

Our diverse approach in model selection ensures that we cater to varying aspects of predictive analytics, from robustness and accuracy to interpretability and regulatory compliance. This strategic choice aids in delivering a more reliable and effective predictive system.

Logistic Regression:

Purpose: Establish a baseline for model performance with a simple, interpretable model.

Key Benefits: Provides probability scores for outcomes, useful for threshold tuning.

Decision Trees:

Purpose: Segment data into homogenous groups to understand influential features.

Key Benefits: Easy to interpret; handles non-linear data effectively.

Random Forest:

Purpose: Improve accuracy over single decision trees by ensemble learning.

Key Benefits: Reduces overfitting, enhances stability and accuracy.

GBM:

Purpose: Sequentially build models to correct previous errors and boost performance.

Key Benefits: Often achieves high accuracy, good for handling imbalanced datasets.

Support Vector Machines (SVM):

Purpose: Classify using hyperplane in multidimensional space that maximally separates classes.

Key Benefits: Effective in high-dimensional spaces, robust against overfitting.

XGBoost:

Purpose: Implement a scalable and efficient boost algorithm.

Key Benefits: Handles missing data, offers built-in regularization to avoid overfitting.

Decision Tree

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

# Fitting the model
tree = DecisionTreeClassifier(criterion='gini', max_depth=None, min_samples_leaf = 1, min_samples_split = 2)
tree.fit(X_train_std, y_train)

# Making predictions and evaluating the model
y_pred_train = tree.predict(X_train_std)
y_pred_test = tree.predict(X_test_std)

train_accuracy = accuracy_score(y_train, y_pred_train)
test_accuracy = accuracy_score(y_test, y_pred_test)

print(f'Training Accuracy: {train_accuracy:.2f}')
print(f'Testing Accuracy: {test_accuracy:.2f}')
```

```
Training Accuracy: 0.96
Testing Accuracy: 0.93
```

XGBoost

```
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score

# Initialize and Train XGBoost Model
model = XGBClassifier(use_label_encoder=False, eval_metric='mlogloss')
# Note: y_train.values.ravel() is used to convert y_train from DataFrame to a 1D array, which XGBClassifier expects.
model.fit(X_train_std, y_train.values.ravel())

# Make predictions and evaluate the model
y_pred = model.predict(X_test_std)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

```
Accuracy: 92.66%
```

Gradient Boosting

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score

# Initialize the Gradient Boosting classifier
gb_clf = GradientBoostingClassifier(n_estimators=100,  # Number of trees
                                    learning_rate=0.1,  # Contribution of each tree
                                    max_depth=3,  # Depth of each tree
                                    random_state=0)

# Fit the model on the training data
gb_clf.fit(X_train_std, y_train)

# Predict on the training set and the test set
y_train_pred = gb_clf.predict(X_train_std)
y_test_pred = gb_clf.predict(X_test_std)

# Calculate and print the accuracy on the training set and the test set
train_accuracy = accuracy_score(y_train, y_train_pred)
test_accuracy = accuracy_score(y_test, y_test_pred)

print(f"Training Accuracy: {train_accuracy:.4f}")
print(f"Test Accuracy: {test_accuracy:.4f}")
```

```
Training Accuracy: 0.8853
Test Accuracy: 0.8258
```

SVM

```
from sklearn.svm import LinearSVC

# Initialize LinearSVC with max_iter parameter
svm = LinearSVC(C=1000.0, random_state=0, max_iter=10)

# Fit the model on the standardized training data
svm.fit(X_train_std, y_train.values.ravel())  # Ensure y_train is 1D

# Predict on the standardized test data
y_pred = svm.predict(X_test_std)

#SVM model evaluation including; Accuracy score, confusion matrix, classification report
from sklearn.metrics import accuracy_score

print('Accuracy: %.2f' % accuracy_score(y_test, y_pred))

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

```
Accuracy: 0.55
[[5791 4677]
 [   3    8]]
              precision    recall  f1-score   support

           0       1.00      0.55      0.71     10468
           1       0.00      0.73      0.00        11

    accuracy                           0.55     10479
   macro avg       0.50      0.64      0.36     10479
weighted avg       1.00      0.55      0.71     10479
```

Logistic

```
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression(C=100000.0, random_state=0)
lr.fit(X_train_std, y_train)

y_pred = lr.predict(X_test_std)

from sklearn.metrics import accuracy_score
print('Accuracy: %.2f' % accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.60
[[6301 4167]
 [   9    2]]
              precision    recall  f1-score   support

           0       1.00      0.60      0.75     10468
           1       0.00      0.18      0.00        11

    accuracy                           0.60     10479
   macro avg       0.50      0.39      0.38     10479
weighted avg       1.00      0.60      0.75     10479
```

Model Evaluation

- Evaluation Metrics:
  - For Classification Tasks:
    - Accuracy: Measures the overall correctness of the model.
    - Precision: Assesses the model's ability to identify only relevant instances.
    - Recall: Evaluates the model's capacity to identify all relevant instances.
  - For Regression Tasks:
    - Mean Squared Error (MSE): Quantifies the average squared difference between the estimated values and actual value.
- Checks Performed:
  - Multicollinearity: Ensure no excessive correlation between independent variables which can affect model reliability.
  - Accuracy, Precision, and Recall: Confirm the model performs well across these metrics to balance the trade-off between catching as many positives as possible and minimizing false positives.
  - K-Fold Cross-Validation: Conducted to evaluate model performance by dividing the data into K equally (or nearly equally) sized segments, or "folds". This method ensures that the model generalizes well across different data sets and is not overly fitted to a single subset of data.

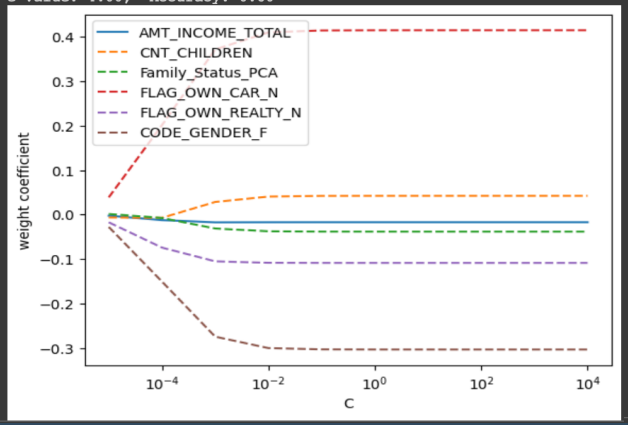
```python
from sklearn.model_selection import cross_val_score
scores = cross_val_score(estimator=tree,
                         X=X_train,
                         y=y_train,
                         cv=10,
                         n_jobs=1)
print('CV accuracy scores: %s' % scores)
print('CV accuracy: %.3f +/- %.3f' % (np.mean(scores), np.std(scores)))
print(f'Average cross-validation score: {np.mean(scores):.4f} ± {np.std(scores):.4f}')
```

```
CV accuracy scores: [0.96353953 0.96292503 0.95903318 0.9645637  0.96210569 0.96190086
 0.96251536 0.9639492  0.96312986 0.96005735]
CV accuracy: 0.962 +/- 0.002
Average cross-validation score: 0.9624 ± 0.0016
```

```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Fitting the model
tree = DecisionTreeClassifier(criterion='gini', max_depth=None, min_samples_leaf = 1, min_samples_split = 2)
tree.fit(X_train_std, y_train)

# Making predictions and evaluating the model
y_pred_train = tree.predict(X_train_std)
y_pred_test = tree.predict(X_test_std)

train_accuracy = accuracy_score(y_train, y_pred_train)
test_accuracy = accuracy_score(y_test, y_pred_test)

print(f'Training Accuracy: {train_accuracy:.2f}')
print(f'Testing Accuracy: {test_accuracy:.2f}')
```

```
Training Accuracy: 0.96
Testing Accuracy: 0.93
```

```
Accuracy: 0.60
[[6301 4167]
 [   9    2]]
              precision    recall  f1-score   support

           0       1.00      0.60      0.75     10468
           1       0.00      0.18      0.00        11

    accuracy                           0.60     10479
   macro avg       0.50      0.39      0.38     10479
weighted avg       1.00      0.60      0.75     10479

 1) AMT_INCOME_TOTAL               0.553716
 2) CNT_CHILDREN                   0.144007
 3) Family_Status_PCA              0.103209
 4) FLAG_OWN_CAR_N                 0.071938
 5) FLAG_OWN_REALTY_N              0.064538
 6) CODE_GENDER_F                  0.062593
e value: -5.00,  Accuracy: 0.59
e value: -4.00,  Accuracy: 0.60
e value: -3.00,  Accuracy: 0.60
e value: -2.00,  Accuracy: 0.60
e value: -1.00,  Accuracy: 0.60
e value: 0.00,  Accuracy: 0.60
e value: 1.00,  Accuracy: 0.60
e value: 2.00,  Accuracy: 0.60
e value: 3.00,  Accuracy: 0.60
e value: 4.00,  Accuracy: 0.60
```




```python
import pandas as pd
import numpy as np
from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.tools.tools import add_constant

# Ensure all columns are numeric and convert them explicitly to float to avoid data type issues
X = pd.get_dummies(myDF2.drop(['Target_Variable'], axis=1), drop_first=True).astype(float)

# Handling any infinite or NaN values in X
X.replace([np.inf, -np.inf], np.nan, inplace=True)
X.dropna(inplace=True)

# Add a constant term for the intercept (required for VIF calculation)
X_with_constant = add_constant(X)

# Initialize DataFrame to store VIF
vif_data = pd.DataFrame()
vif_data["Feature"] = X_with_constant.columns

# Calculate VIF for each feature using a list comprehension to ensure all values are finite
vif_data["VIF"] = [variance_inflation_factor(X_with_constant.values, i)
                   for i in range(X_with_constant.shape[1])]

# Display the VIF
print(vif_data.sort_values(by='VIF', ascending=False))
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/stats/outliers_influence.py:198: RuntimeWarning: divide by zero encountered in scalar divide
  vif = 1. / (1. - r_squared_i)
                                    Feature           VIF
12                          MONTHS_BALANCE_min           inf
13                          MONTHS_BALANCE_max           inf
14                         MONTHS_BALANCE_mean           inf
7                                  FLAG_MOBIL  1.880145e+04
11                           CNT_FAM_MEMBERS  4.363143e+02
27  NAME_EDUCATION_TYPE_Secondary / secondary special  4.004627e+02
24          NAME_EDUCATION_TYPE_Higher education  3.677366e+02
2                             CNT_CHILDREN  3.024172e+02
25        NAME_EDUCATION_TYPE_Incomplete higher  6.951074e+01
30      NAME_FAMILY_STATUS_Single / not married  5.916962e+01
29           NAME_FAMILY_STATUS_Separated  2.832932e+01
26          NAME_EDUCATION_TYPE_Lower secondary  1.434805e+01
31                NAME_FAMILY_STATUS_Widow  1.226814e+01
39                 OCCUPATION_TYPE_Laborers  5.199307e+00
34               OCCUPATION_TYPE_Core staff  3.458945e+00
45              OCCUPATION_TYPE_Sales staff  3.437019e+00
41                 OCCUPATION_TYPE_Managers  3.203794e+00
35                  OCCUPATION_TYPE_Drivers  3.016448e+00
28               NAME_FAMILY_STATUS_Married  2.874688e+00
37        OCCUPATION_TYPE_High skill tech staff  2.058859e+00
42              OCCUPATION_TYPE_Medicine staff  2.011539e+00
17                    Balance_Numeric_Category  2.004152e+00
18                              CODE_GENDER_M  1.678278e+00
47              OCCUPATION_TYPE_Security staff  1.571457e+00
33               OCCUPATION_TYPE_Cooking staff  1.554026e+00
```

Hyperparameter Tuning

Methods:

Manual Tuning: Adjust parameters based on intuition and iterative testing.

Automated Tuning:

- Grid Search: Systematically test combinations of parameters to find the best set.
- Random Search: Randomly sample parameter spaces for a faster, less exhaustive search.

Other Areas:

Regularization Strength: Experiment with different levels to see how they influence model performance and feature weights.

- Analysis: Plotting results to visually understand the impact of regularization on feature relevance and model accuracy.

```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Fitting the model
tree = DecisionTreeClassifier(criterion='gini', max_depth=None, random_state=0)
tree.fit(X_train_std, y_train)

# Making predictions and evaluating the model
y_pred_train = tree.predict(X_train_std)
y_pred_test = tree.predict(X_test_std)

train_accuracy = accuracy_score(y_train, y_pred_train)
test_accuracy = accuracy_score(y_test, y_pred_test)

print(f'Training Accuracy: {train_accuracy:.4f}')
print(f'Testing Accuracy: {test_accuracy:.4f}')
```

```
Training Accuracy: 0.9625
Testing Accuracy: 0.9272
```

```python
from sklearn.model_selection import GridSearchCV

# Parameters to tune
param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 10, 20, 30, 50],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Initialize the DecisionTreeClassifier
dt_classifier = DecisionTreeClassifier(random_state=0)

# Initialize the GridSearchCV object
grid_search = GridSearchCV(estimator=dt_classifier, param_grid=param_grid, cv=5, scoring='accuracy', n_jobs=-1, verbose=1)

# Fit the grid search to the data
grid_search.fit(X_train_std, y_train)

# Print the best parameters and the best score
print("Best parameters found: ", grid_search.best_params_)
print("Best cross-validation score: {:.4}".format(grid_search.best_score_))

# Use the best estimator to make predictions
y_pred = grid_search.best_estimator_.predict(X_test_std)

# Evaluate the predictions
test_accuracy = accuracy_score(y_test, y_pred)
print("Accuracy on the test set: {:.4}".format(test_accuracy))
```

```
Fitting 5 folds for each of 90 candidates, totalling 450 fits
Best parameters found:  {'criterion': 'gini', 'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2}
Best cross-validation score: 0.9623
Accuracy on the test set: 0.9272
```

Sample Results

```
X.loc[100]
```

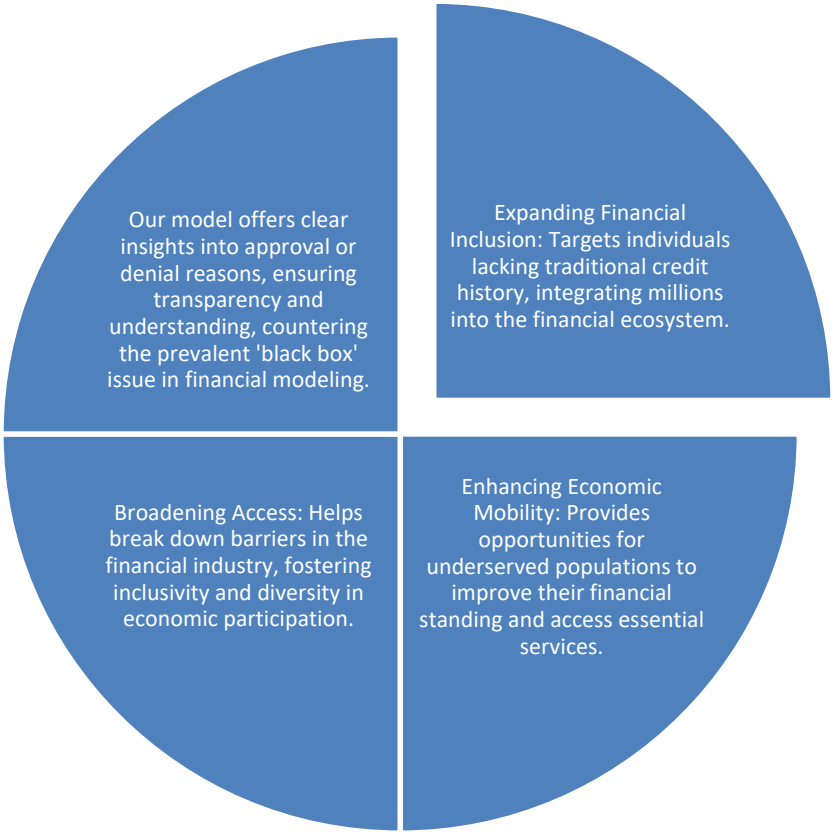
```
CNT_CHILDREN              0.000000
AMT_INCOME_TOTAL     297000.000000
CODE_GENDER_F             1.000000
FLAG_OWN_CAR_N            1.000000
FLAG_OWN_REALTY_N         0.000000
Family_Status_PCA         2.096417
Name: 100, dtype: float64
```

```
from sklearn.linear_model import LogisticRegression

# Assuming X is your feature matrix and y is your target vector
# Fit the classifier model on the entire dataset
classifier = LogisticRegression()
classifier.fit(X, y)

# Sample one row from your dataset (e.g., the first row)
sample_row = X.iloc[[100]]  # Adjust the index as needed

# Get the coefficients
coefficients = classifier.coef_

# Multiply the coefficients by the features of the sampled row
importance = sample_row.values * coefficients

# Print the importance of each feature
print("Importance of each feature towards classification:")
for i, feature in enumerate(X.columns):
    print(f"{feature}: {importance[0][i]}")
```

```
Importance of each feature towards classification:
CNT_CHILDREN: -0.0
AMT_INCOME_TOTAL: -16.082142496259006
CODE_GENDER_F: -1.0852576098667662e-09
FLAG_OWN_CAR_N: -4.277843009031648e-12
FLAG_OWN_REALTY_N: -0.0
Family_Status_PCA: -2.76555225129503e-10
```

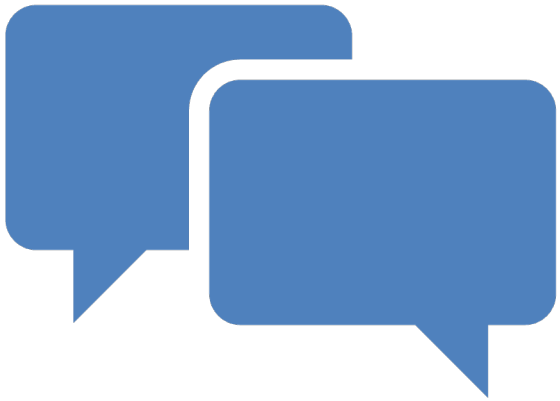
The feature importance values indicate how much each variable sways the model's credit card approval prediction for a specific sample. A zero importance for CNT_CHILDREN and FLAG_OWN_REALTY_N means they don't affect this prediction. A negative value for AMT_INCOME_TOTAL and CODE_GENDER_F suggests these lower the approval odds, while a positive value for Family_Status_PCA increases them. Notably, not owning a car is the most significant negative predictor for this case. This example highlights our model’s transparency, providing clear rationales for each decision.

Our Model


Our model offers clear insights into approval or denial reasons, ensuring transparency and understanding, countering the prevalent 'black box' issue in financial modeling.
Expanding Financial Inclusion: Targets individuals lacking traditional credit history, integrating millions into the financial ecosystem.
Broadening Access: Helps break down barriers in the financial industry, fostering inclusivity and diversity in economic participation.
Enhancing Economic Mobility: Provides opportunities for underserved populations to improve their financial standing and access essential services.

Our Recommendations

- Managerial Actions:
  - Expand Outreach: Focus on demographics typically underserved by traditional credit systems.
  - Enhance Transparency: Implement clear communication strategies to explain credit decisions to applicants.
  - Refine Products: Tailor credit products to align with the predictive factors driving approval, such as offering starter cards to those without a car.



Open Floor for Questions

Q&A