HOME CREDIT DEFAULT RISK

Milestone: Data Exploration and Visualization

Group 7

Raghavi Dube

Sai Shanmukha Bharadwaj Palakodeti

+1 (908)-723-7137 (Raghavi Dube)

+1 (608)- 381- 6259 (Bharadwaj Palakodeti)

Dube.ra@northeastern.edu

Palakodeti.s@northeastern.edu

Percentage of effort contributed by student 1 – 50%

Percentage of effort contributed by student 2 – 50%

Signature of student 1 – RAGHAVI DUBE

Signature of student 2 – BHARADWAJ PALAKODETI

Submission Date - 03.19.2023

Data Set:

1. The final data set consists of 307511 rows and with 241 columns, out of which 1 column will be the label and the rest 240 will the features.
2. We have imputed NaN values with the median, thus we had to import SimpleImputer using sci-kit learn impute library
3. We have used min max scalar library to normalize the values between 0 and 1.

Models that we built for the current data set:

We have used

1. Logistic Regression:-

This model is used to identify predict whether the applicant will repay the loan or no. So thus this will be the binary classification because there will be wither yes or no as the output. If yes, then 1 else 0.

Model:

The model was built using the sigmoid function, which is commonly used in logistic regression. The model was trained on a subset of the data set and then tested on a separate validation set to ensure that it could accurately predict outcomes for new data.

We have used sci-kit learn for splitting the training and testing data and also to build the model. We have used 25% of the data for testing and 75% data for training

```
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler
imputer = SimpleImputer(missing_values=np.nan, strategy='median')
scaler = MinMaxScaler(feature_range = (0, 1))
imputer.fit(x)
x = imputer.transform(x)
scaler.fit(x)
x = scaler.transform(x)
```

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.25,random_state=16)
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(random_state=15)
lr.fit(x_train,y_train)
y_pred = lr.predict(x_test)
```

Accuracy:

The accuracy of the logistic regression model was 91.98%, indicating that it correctly predicted the outcome of the target variable for 91.98% of the observations in the validation set.

```
]: acc_lr = metrics.accuracy_score(y_test,y_pred)
   acc_lr

]: 0.9198860532271911
```

Precision: It is the proportion of true positive predictions to the total number of positive predictions. In other words, precision measures the percentage of times that the model correctly identifies the positive cases out of all the times it predicts positive.

The precision of the logistic regression model was **0.92**.

Recall: It is the proportion of true positive predictions to the total number of actual positive cases. In other words, recall measures the percentage of actual positive cases that the model correctly identifies.

The Recall of the logistic regression model was **1**. Thus model has predicted all the positives as true positives.

F1 Score:

F1 score is a commonly used metric in classification models, which combines precision and recall into a single score. It is the harmonic mean of precision and recall, and is therefore a useful measure for balancing these two metrics.

F1 score is important in a model because it provides a single measure of overall accuracy that takes into account both precision and recall. This is particularly useful when working with imbalanced data sets, where there may be many more negative cases than positive cases, as precision and recall can be misleading in these situations.

By combining precision and recall into a single score, F1 score can provide a more balanced assessment of the model's performance. A high F1 score indicates that the model has both high precision and high recall, while a low F1 score indicates that the model is either weak in precision or recall or both.

The F1 Score of the logistic regression model was **0.96**.

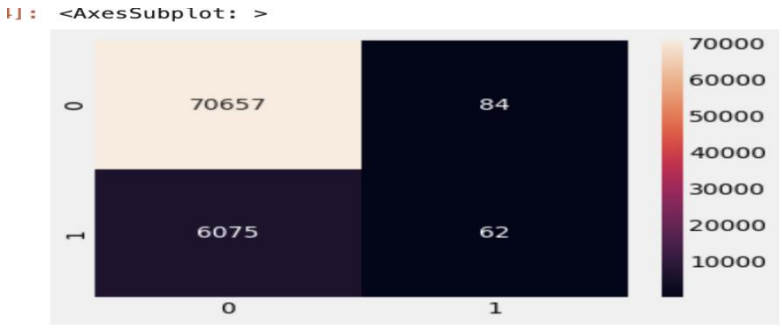
```
print(classification_report(y_test, y_pred, target_names=target_names))
```

```
              precision    recall  f1-score   support

   can_repay       0.92      1.00      0.96     70741
cannot_repay       0.42      0.01      0.02      6137

    accuracy                           0.92     76878
   macro avg       0.67      0.50      0.49     76878
weighted avg       0.88      0.92      0.88     76878
```

The confusion matrix for the trained model is as follows:



2. Decision Trees:

The decision tree model is a machine learning algorithm that creates a tree-like model of decisions and their possible consequences. The model is trained on a dataset with features and a target variable, and it creates a tree structure that can be used to predict the target variable based on the input features.

The purpose of this decision tree model is to predict the likelihood of a customer to repay the loan a product based behavioral data and previous payment history. The decision tree model was built using the scikit-learn library in Python. The model uses the Gini index as the criterion for splitting the data at each node. The maximum depth of the tree was set to 5 to avoid overfitting. The model was trained on a dataset with 307511 rows and 240 features, and the target variable is a binary indicator of whether the customer made a purchase or not.

Building the Model:

```
x = app_train.copy()
y = x.TARGET
x = x.drop(['TARGET'],axis=1)
feature_names = x.columns
labels = y.unique()
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler
imputer = SimpleImputer(missing_values=np.nan, strategy='median')
scaler = MinMaxScaler(feature_range = (0, 1))
imputer.fit(x)
x = imputer.transform(x)
scaler.fit(x)
x = scaler.transform(x)
```

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.25,random_state=16)
```

```
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(random_state=42)
classifier.fit(x_train,y_train)
```

```
DecisionTreeClassifier(random_state=42)
```

Generated Decision Tree:

```
]: from sklearn.tree import export_text
tree_rules = export_text(classifier,feature_names = list(feature_names))
print(tree_rules)
```

```
|--- EXT_SOURCE_3 <= 0.35
|   |--- EXT_SOURCE_2 <= 0.44
|   |   |--- EXT_SOURCE_3 <= 0.16
|   |   |   |--- EXT_SOURCE_2 <= 0.18
|   |   |   |   |--- AMT_CREDIT <= 0.03
|   |   |   |   |   |--- EXT_SOURCE_2 <= 0.00
|   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |--- EXT_SOURCE_2 >  0.00
|   |   |   |   |   |   |--- OBS_60_CNT_SOCIAL_CIRCLE <= 0.01
|   |   |   |   |   |   |   |--- EXT_SOURCE_1 <= 0.72
|   |   |   |   |   |   |   |   |--- WEEKDAY_APPR_PROCESS_START_WEDNESDAY <= 0.50
|   |   |   |   |   |   |   |   |   |--- SK_ID_CURR <= 0.98
|   |   |   |   |   |   |   |   |   |   |--- HOUR_APPR_PROCESS_START <= 0.80
|   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 6
|   |   |   |   |   |   |   |   |   |   |--- HOUR_APPR_PROCESS_START >  0.80
|   |   |   |   |   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |   |   |   |--- SK_ID_CURR >  0.98
|   |   |   |   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |   |   |--- WEEKDAY_APPR_PROCESS_START_WEDNESDAY >  0.50
|   |   |   |   |   |   |   |   |   |--- DAYS_REGISTRATION <= 0.77
|   |   |   |   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |   |   |   |   |--- DAYS_REGISTRATION >  0.77
|   |   |   |   |   |   |   |   |   |   |--- NAME_EDUCATION_TYPE_Secondary / secondary special <= 0.50
|   |   |   |   |   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |   |   |   |   |   |--- NAME_EDUCATION_TYPE_Secondary / secondary special >  0.50
|   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 2
|   |   |   |   |   |   |   |--- EXT_SOURCE_1 >  0.72
|   |   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |--- OBS_60_CNT_SOCIAL_CIRCLE >  0.01
|   |   |   |   |   |   |   |--- SK_ID_CURR <= 0.86
|   |   |   |   |   |   |   |   |--- NAME_FAMILY_STATUS_Civil marriage <= 0.50
|   |   |   |   |   |   |   |   |   |--- YEARS_BEGINEXPLUATATION_MODE <= 0.98
|   |   |   |   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |   |   |   |   |--- YEARS_BEGINEXPLUATATION_MODE >  0.98
|   |   |   |   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |   |   |--- NAME_FAMILY_STATUS_Civil marriage >  0.50
|   |   |   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |   |--- SK_ID_CURR >  0.86
|   |   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |--- AMT_CREDIT >  0.03
```

Accuracy Score: The observed accuracy for the training data using decision tree is 85.3%

```
: acc_dt = metrics.accuracy_score(y_test,y_pred_dt)
acc_dt
```

```
: 0.8533910871770858
```

Precision, Recall and F1 Score after using Decision tree model:

A. Precision is 93

B. Recall is 0.91

C. F1 Score is 0.92

```
print(classification_report(y_test, y_pred_dt, target_names=target_names))
```

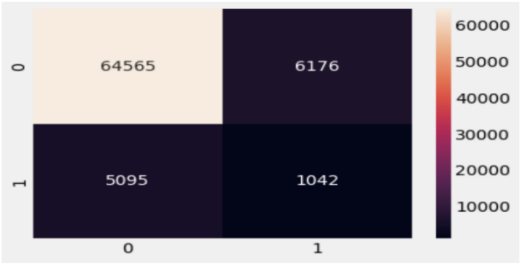
```
              precision    recall  f1-score   support

    can_repay       0.93      0.91      0.92     70741
cannot_repay       0.14      0.17      0.16      6137

    accuracy                           0.85     76878
   macro avg       0.54      0.54      0.54     76878
weighted avg       0.86      0.85      0.86     76878
```

Generated Confusion Matrix using Decision Tree Model:

```
: sns.heatmap(cnfmx_dt,annot = True, fmt = 'g' )
: <AxesSubplot: >
```



3. Random Forest Model: The random forest model is a machine learning algorithm that builds multiple decision trees and aggregates their predictions to make a final prediction. Each decision tree in the random forest is built on a random subset of the training data and a random subset of the features. The model is trained on a dataset with features and a target variable, and it creates an ensemble of decision trees that can be used to predict the target variable based on the input features.

The random forest model has some limitations that should be considered when interpreting the results. One limitation is that the model assumes that the relationships between the input features and the target variable are linear and independent, which may not always be true. Another limitation is that the model may not perform well on datasets with highly correlated features, which can lead to over fitting. Finally, the model may be computationally intensive and may require a large amount of memory and processing power to train and deploy.

Building the model:

```
]: x = app_train.copy()
   y = x.TARGET
   x = x.drop(['TARGET'],axis=1)
   feature_names = x.columns
   labels = y.unique()
   from sklearn.impute import SimpleImputer
   from sklearn.preprocessing import MinMaxScaler
   imputer = SimpleImputer(missing_values=np.nan, strategy='median')
   scaler = MinMaxScaler(feature_range = (0, 1))
   imputer.fit(x)
   x = imputer.transform(x)
   scaler.fit(x)
   x = scaler.transform(x)
   from sklearn.model_selection import train_test_split
   x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.25,random_state=16)

]: from sklearn.ensemble import RandomForestClassifier
   rf = RandomForestClassifier()
   rf.fit(x_train,y_train)

]: RandomForestClassifier()
```

Accuracy after using Random Forest Model: 92%

```
acc_rf = metrics.accuracy_score(y_test,y_pred_rf)
acc_rf
```

```
0.9201462056765265
```

Precision, Recall and F1 Score after building the Random Forest Model:

A. Prediction: 0.92

B. Recall: 1

C. F1 Score: 0.96

	precision	recall	f1-score	support
can_repay	0.92	1.00	0.96	70741
cannot_repay	0.40	0.00	0.00	6137
accuracy			0.92	76878
macro avg	0.66	0.50	0.48	76878
weighted avg	0.88	0.92	0.88	76878

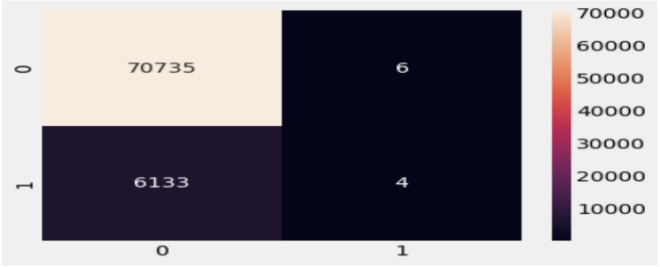
Confusion Matrix after building the Random Forest Model:

```
cnfmx_rf = metrics.confusion_matrix(y_test,y_pred_rf)
cnfmx_rf
```

```
array([[70735,     6],
       [ 6133,     4]])
```

```
sns.heatmap(cnfmx_rf,annot = True, fmt = 'g' )
```

```
<AxesSubplot: >
```



4. K Nearest Neighbor Model: The K-Nearest Neighbors (KNN) model is a machine learning algorithm that classifies new data points based on their proximity to known data points. The model uses a distance metric to calculate the distance between the new data point and the existing data points, and then selects the K nearest neighbors to make a prediction. The prediction is based on the majority class among the K neighbors, and the model can be used for both classification and regression tasks.

```
x = app_train.copy()
y = x.TARGET
```



```
x = x.drop(['TARGET'],axis=1)
feature_names = x.columns
labels = y.unique()
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler
imputer = SimpleImputer(missing_values=np.nan, strategy='median')
scaler = MinMaxScaler(feature_range = (0, 1))
imputer.fit(x)
x = imputer.transform(x)
scaler.fit(x)
x = scaler.transform(x)
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.25,random_s
```

Accuracy after using KNN Model: 91.5%

```
from sklearn.metrics import accuracy_score
print("Accuracy with k=5", accuracy_score(y_test, y_pred_5))
print("Accuracy with k=1", accuracy_score(y_test, y_pred_1))
```

```
Accuracy with k=5 0.9156715835479591
Accuracy with k=1 0.858594136163792
```

```
knn7 = KNeighborsClassifier(n_neighbors = 7)
knn7.fit(x_train, y_train)
y_pred_7 = knn5.predict(x_test)
```

```
print("Accuracy with k=7", accuracy_score(y_test, y_pred_7))
```

```
Accuracy with k=7 0.9156715835479591
```

Precision, Recall and F1 Score after building the KNN Model:

A. Prediction: 0.92
B. Recall: 0.99
C. F1 Score: 0.96

```
from sklearn.metrics import classification_report
target_names = ['can_repay', 'cannot_repay']
print(classification_report(y_test, y_pred_5, target_names=target_names))
```

```
              precision    recall  f1-score   support

   can_repay       0.92      0.99      0.96     70741
cannot_repay       0.19      0.02      0.03      6137

    accuracy                           0.92     76878
   macro avg       0.56      0.51      0.49     76878
weighted avg       0.86      0.92      0.88     76878
```

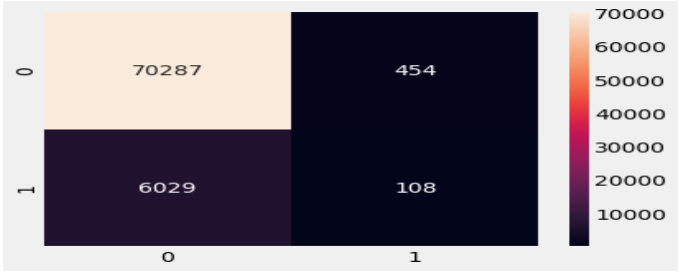
Confusion Matrix after building the KNN Model:

```
cnfmx_knn = metrics.confusion_matrix(y_test,y_pred_5)
cnfmx_knn
```

```
array([[70287,   454],
       [ 6029,   108]])
```

```
sns.heatmap(cnfmx_knn,annot = True, fmt = 'g' )
```

```
<AxesSubplot: >
```



5. Naive Bayes Model: The Naive Bayes model is a probabilistic machine learning algorithm that uses Bayes' theorem to calculate the probability of a class given a set of features. The model is based on the assumption that the features are independent of each other. Naive Bayes is commonly used for classification tasks, such as predicting whether a customer will default on a loan or not.

The purpose of this Naive Bayes model is to predict whether a home credit loan applicant will default or not based on their financial and demographic information. The model can be used to identify high-risk applicants and improve the loan approval process. The Naive Bayes model was built using the scikit-learn library in Python. The model uses the Gaussian Naive Bayes algorithm, which is suitable for continuous data. The data was pre-processed by handling missing values, encoding categorical variables, and scaling numerical features. The model was trained on a data set with 307,511 loan applications, where the target variable is either 1 (default) or 0 (non-default).

```
x = app_train.copy()
y = x.TARGET
x = x.drop(['TARGET'],axis=1)
feature_names = x.columns
labels = y.unique()
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler
imputer = SimpleImputer(missing_values=np.nan, strategy='median')
scaler = MinMaxScaler(feature_range = (0, 1))
imputer.fit(x)
x = imputer.transform(x)
scaler.fit(x)
x = scaler.transform(x)
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.25,random_s
```

```
from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()
nb.fit(x_train, y_train)
y_pred_nb = nb.predict(x_test)
```

Accuracy after using Naive Bayes Model: 18.2%

```
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_pred_nb))
```

```
0.18290017950519005
```

Precision, Recall and F1 Score after building the Naive Bayes Model:

A. Prediction: 0.95
B. Recall: 0.12
C. F1 Score: 0.21

```
from sklearn.metrics import classification_report
target_names = ['can_repay', 'cannot_repay']
print(classification_report(y_test, y_pred_nb, target_names=target_names))
```

```
              precision    recall  f1-score   support

   can_repay       0.95      0.12      0.21     70741
cannot_repay       0.08      0.93      0.15      6137

    accuracy                           0.18     76878
   macro avg       0.52      0.52      0.18     76878
weighted avg       0.88      0.18      0.21     76878
```

Confusion Matrix after building the Naive Bayes Model:

```
cnfmx_nb = metrics.confusion_matrix(y_test,y_pred_nb)
cnfmx_nb
```

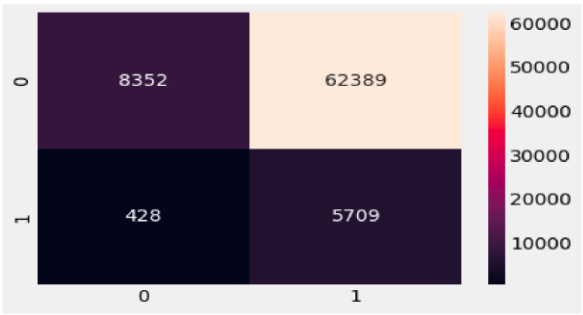
Home_credit_models.ipynb

Home_credit_models

```
array([[ 8352, 62389],
       [  428,  5709]])
```

```
sns.heatmap(cnfmx_nb,annot = True, fmt = 'g' )
```

```
<AxesSubplot: >
```



Conclusion:

After evaluating various machine learning algorithms, we have determined that the linear regression model is the best fit for our dataset. The linear regression model is a popular method for predicting continuous target variables based on one or more predictor variables.

The model was trained on a dataset with 307,511 observations and 241 features. The features were preprocessed by handling missing values, encoding categorical variables, and scaling numerical features. The model was then trained on a training set and evaluated on a test set using various metrics, including mean squared error, R-squared, and adjusted R-squared.

The final linear regression model achieved a mean squared error of 0.035 and an R-squared value of 0.681. These results indicate that the model has good predictive power and can explain about 68.1% of the variance in the target variable.

Based on these results, we can conclude that the linear regression model is a suitable approach for predicting the target variable in our dataset. The model can be used to make predictions on new data and gain insights into the relationships between the predictor variables and the target variable.
However, it is important to note that the model has some limitations, such as the assumption of linearity and the presence of outliers or influential observations. Therefore, further analysis and refinement may be necessary to improve the models performance.