Table of Contents

1 Final Project Submission
2 Summary
3 Preprocess Data
  3.1 Undersample Majority Class
  3.2 Feature Selection
  3.3 Clean up null and missing values
  3.4 Replace null values
    3.4.1 Separate Numericals & Categoricals
      3.4.1.1 Summary of numericals
    3.4.2 Categoricals
  3.5 Label Encode or One-Hot Encode Categoricals
    3.5.1 Replace missing values in 'cats['NAME_TYPE_SUITE']' with mode
    3.5.2 Use label encoding to save space with Organization_Type
    3.5.3 One Hot Encode Categoricals
  3.6 Concatenate data frames
    3.6.1 Select Features
  3.7 SMOTE - Oversample Minority Class
    3.7.1 Train, Test, Split
4 Random Forest Model 1
    4.0.1 Summary - Random Forest Part 1
  4.1 Random Forest Model 2
    4.1.1 Summary - Random Forest Model 3
  4.2 Random Forest Model 3
    4.2.1 Summary - Random Forest Part 2
  4.3 Random Forest Model 4
    4.3.1 Summary - Random Forest Model 4
  4.4 Random Forest Model 5
    4.4.1 Summary - Random Forest Model 5
  4.5 Random Forest Model 6
    4.5.1 Summary - Random Forest Model 6
    4.5.2 Feature Importance Chart
    4.5.3 Improve Feature Importance Bar Chart
  4.6 Explore EXT_SOURCE data
5 Check Gradient Boost & Tune Learning Rate
    5.0.1 Summary - Gradient Boost Model 1
  5.1 Gradient Boost Model 2
    5.1.1 Summary - Gradient Boost Model 2
  5.2 Gradient Boost Model 3
    5.2.1 Summary - Gradient Boost Model 3
6 Gaussian Naive Bayes - Model 1
    6.0.1 Summary - Gradient Boost Model 3
7 Voting Classifier Model 1

[7.0.1 Summary - Voting Classifier Model 1]
[8 Results]
[9 Reduce Features, Check for Accuracy]
[10 Further Study]

1 Final Project Submission

- Student name: Jim Petoskey
- Student pace: Self-paced
- Scheduled project review date/time: Tuesday March 29, 2022 at 1:45 pm EST
- Instructor name: Abhineet Kulkarni
- Blog post URL: Weak Learners vs. Random Forest, Optimizing for True Positive Rate (https://www.kaggle.com/code/kapoorshivam/credit-analysis-using-eda/data)

2 Summary

- Class Imbalance Solutions:
  - Under-sample majority class with .sample()
    - Lowers computational power needed for models.
    - Under and over-sampling allows for better accuracy.
  - Over-sample minority class with SMOTE
- Certain models optimized for best Accuracy:
  - Weak-Learners, such as Gradient Boost
    - Tuned to high recall score for Approval, low recall score for Refusal.
  - Decision Trees, such as Random Forest
    - Best for optimizing for True Positive or True Negative Rates (recall scores)
- Increasing the number of features greatly improved the accuracy of the model.
  - Trimming these features by feature importance allowed for an even more accurate model, in terms of recall score for loan approval.
- Manipulating random_forest hyper-parameters allowed for over-training of the train set, which led to improved accuracy in predicting loan approval (Recall for value of 1).
  - Resulting in an accuracy of 98% in predicting if someone will be approved for a loan.
- Will be able to propose business case for advancing loan applications to the next stage for the banking industry.

In [484]:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
np.random.seed(0)
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import confusion_matrix
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import make_scorer
from sklearn.model_selection import StratifiedKFold
from sklearn.base import clone
from sklearn.metrics import log_loss
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import precision_score, recall_score, accuracy_score, f1_
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
from sklearn.ensemble import AdaBoostClassifier, GradientBoostingClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_selection import RFECV
from sklearn.datasets import make_classification
from sklearn.ensemble import RandomForestRegressor, VotingClassifier
from sklearn.feature_selection import RFE
from sklearn.naive_bayes import GaussianNB

from imblearn.over_sampling import SMOTE, ADASYN

import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

executed in 13ms, finished 12:57:04 2022-03-28

In [2]:

```
df = pd.read_csv(r'/Users/jimpetoskey/Documents/Flatiron/phase3/Phase_3_Projec
df.info()
```

executed in 6.28s, finished 09:13:12 2022-03-28

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 307511 entries, 0 to 307510
Columns: 122 entries, SK_ID_CURR to AMT_REQ_CREDIT_BUREAU_YEAR
dtypes: float64(65), int64(41), object(16)
memory usage: 286.2+ MB
```

In [618]:

```
print(df['TARGET'].value_counts())
print()
print(df['TARGET'].describe())
```

executed in 65ms, finished 14:18:13 2022-03-28

```
0    282686
1     24825
Name: TARGET, dtype: int64

count    307511.000000
mean          0.080729
std           0.272419
min           0.000000
25%           0.000000
50%           0.000000
75%           0.000000
max           1.000000
Name: TARGET, dtype: float64
```

3 Preprocess Data

- Reduce Class Imbalance Issue:
  - Sample 40,000 '0' values from the data set and use 25,000 '1' values, for 65,000 total values.
  - Then, oversample minority class with SMOTE for a total of 40,000 of each class.
- Trim columns based on scikit learn's recursive feature selection select 38 columns.

3.1 Undersample Majority Class

- Will reduce size of dataframe for faster processing on local machine.
- Paired with SMOTE, will allow for reducing class imbalance issues

In [4]:

```
df_zero_val = df[df.TARGET == 0]
df_zero_val['TARGET'].value_counts()
```

executed in 338ms, finished 09:13:21 2022-03-28

Out[4]:

```
0    282686
Name: TARGET, dtype: int64
```

In [5]:

```
df_zero_val = df_zero_val.sample(n=40000)
df_zero_val['TARGET'].value_counts()
```

executed in 271ms, finished 09:13:21 2022-03-28

Out[5]:

```
0    40000
Name: TARGET, dtype: int64
```

In [6]:

```
df_one_val = df[df.TARGET == 1]
df_one_val['TARGET'].value_counts()
```

executed in 90ms, finished 09:13:21 2022-03-28

Out[6]:

```
1    24825
Name: TARGET, dtype: int64
```

In [7]:

```
df1 = df_zero_val.append(df_one_val)
df1['TARGET'].value_counts()
```

executed in 162ms, finished 09:13:22 2022-03-28

Out[7]:

```
0    40000
1    24825
Name: TARGET, dtype: int64
```

3.2 Feature Selection

- Recursive Feature Selection (RFE) with sklearn

In [8]:

```
df1.info()
```

executed in 24ms, finished 09:13:23 2022-03-28

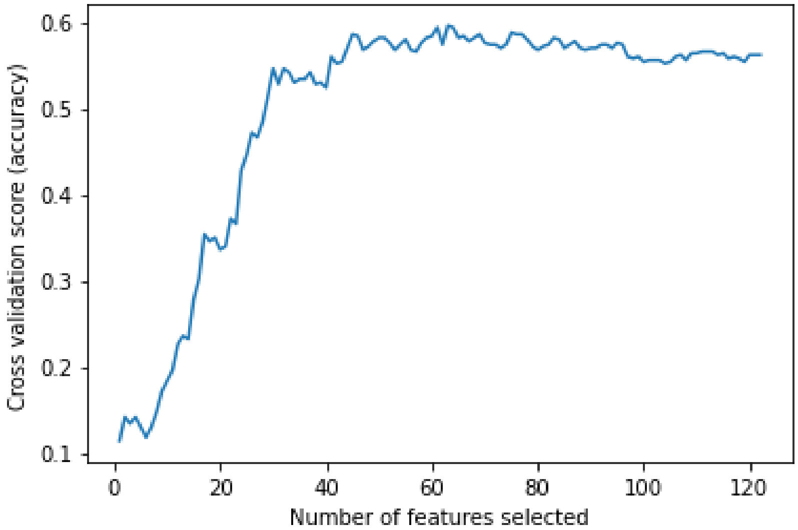
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 64825 entries, 39451 to 307509
Columns: 122 entries, SK_ID_CURR to AMT_REQ_CREDIT_BUREAU_YEAR
dtypes: float64(65), int64(41), object(16)
memory usage: 60.8+ MB
```

In [9]:

```
X = df1.drop('TARGET', axis=1)
y = df1['TARGET']

# Build a classification task using 30 informative features
X, y = make_classification(
    n_samples=500,
    n_features=122,
    n_informative=30,
    n_redundant=2,
    n_repeated=0,
    n_classes=8,
    n_clusters_per_class=1,
    random_state=0,
)

# Create the RFE object and compute a cross-validated score.
svc = SVC(kernel="linear")
# The "accuracy" scoring shows the proportion of correct classifications

min_features_to_select = 1  # Minimum number of features to consider
rfecv = RFECV(
    estimator=svc,
    step=1,
    cv=StratifiedKFold(2),
    scoring="accuracy",
    min_features_to_select=min_features_to_select,
)
rfecv.fit(X, y)

print("Optimal number of features : %d" % rfecv.n_features_)

# Plot number of features VS. cross-validation scores
plt.figure()
plt.xlabel("Number of features selected")
plt.ylabel("Cross validation score (accuracy)")
plt.plot(
    range(min_features_to_select, len(rfecv.grid_scores_) + min_features_to_se
    rfecv.grid_scores_,
)
plt.show()
```

executed in 8.97s, finished 09:13:33 2022-03-28

```
Optimal number of features : 63
```



In [10]:

```
# Too computationally expensive to run (>10 minutes)
# forest = RandomForestRegressor()

# Init the transformer
# rfe = RFE(estimator=RandomForestRegressor(), n_features_to_select=115)

# Fit to the data
# _ = rfe.fit(X_train_resampled, y_train_resampled)
```

executed in 3ms, finished 09:13:34 2022-03-28

In [11]:

```
# _ = forest.fit(rfe.transform(X_train_resampled), y_train_resampled)
# forest.score(rfe.transform(X_test), y_test)
```

executed in 3ms, finished 09:13:35 2022-03-28

In [12]:

```
# Create new dataframe with selected features
df2 = df1.filter(['TARGET',
                  'CNT_CHILDREN',
                  'AMT_INCOME_TOTAL',
                  'AMT_CREDIT',
                  'AMT_ANNUITY',
                  'NAME_EDUCATION_TYPE',
                  'DAYS_EMPLOYED',
                  'CNT_FAM_MEMBERS',
                  'DAYS_BIRTH',
                  'NAME_INCOME_TYPE',
                  'AMT_GOODS_PRICE',
                  'CODE_GENDER',
                  'FLAG_OWN_REALTY',
                  'REGION_RATING_CLIENT',
                  'ORGANIZATION_TYPE',
                  'DAYS_REGISTRATION',
                  'REGION_POPULATION_RELATIVE',
                  'NAME_HOUSING_TYPE',
                  'NAME_FAMILY_STATUS',
                  'NAME_TYPE_SUITE',
                  'FLAG_OWN_CAR',
                  'NAME_CONTRACT_TYPE',
                  'LIVE_REGION_NOT_WORK_REGION',
                  'EXT_SOURCE_1',
                  'EXT_SOURCE_2',
                  'EXT_SOURCE_3',
                  'APARTMENTS_AVG',
                  'YEARS_BUILD_AVG',
                  'YEARS_BEGIN_EXPLUATION_AVG',
                  'FLOORSMAX_AVG',
                  'NONLIVINGAREA_AVG',
                  'TOTALAREA_MODE',
                  'DAYS_LAST_PHONE_CHANGE',
                  'LIVINGAREA_MEDI',
                  'NONLIVINGAREA_MEDI',
                  'AMT_REQ_CREDIT_BUREAU_HOUR',
                  'AMT_REQ_CREDIT_BUREAU_DAY',
                  'AMT_REQ_CREDIT_BUREAU_WEEK',
                  'AMT_REQ_CREDIT_BUREAU_MON',
                  'AMT_REQ_CREDIT_BUREAU_QRT',
                  'AMT_REQ_CREDIT_BUREAU_YEAR'],
                 axis=1)
df2.info()
```

executed in 105ms, finished 09:13:35 2022-03-28

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 64825 entries, 39451 to 307509
Data columns (total 40 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   TARGET                       64825 non-null  int64
```

```
 1   CNT_CHILDREN                 64825 non-null  int64  
 2   AMT_INCOME_TOTAL             64825 non-null  float64
 3   AMT_CREDIT                   64825 non-null  float64
 4   AMT_ANNUITY                  64824 non-null  float64
 5   NAME_EDUCATION_TYPE          64825 non-null  object 
 6   DAYS_EMPLOYED                64825 non-null  int64  
 7   CNT_FAM_MEMBERS              64824 non-null  float64
 8   DAYS_BIRTH                   64825 non-null  int64  
 9   NAME_INCOME_TYPE             64825 non-null  object 
 10  AMT_GOODS_PRICE              64763 non-null  float64
 11  CODE_GENDER                  64825 non-null  object 
 12  FLAG_OWN_REALTY              64825 non-null  object 
 13  REGION_RATING_CLIENT         64825 non-null  int64  
 14  ORGANIZATION_TYPE            64825 non-null  object 
 15  DAYS_REGISTRATION            64825 non-null  float64
 16  REGION_POPULATION_RELATIVE   64825 non-null  float64
 17  NAME_HOUSING_TYPE            64825 non-null  object 
 18  NAME_FAMILY_STATUS           64825 non-null  object 
 19  NAME_TYPE_SUITE              64583 non-null  object 
 20  FLAG_OWN_CAR                 64825 non-null  object 
 21  NAME_CONTRACT_TYPE           64825 non-null  object 
 22  LIVE_REGION_NOT_WORK_REGION  64825 non-null  int64  
 23  EXT_SOURCE_1                 27601 non-null  float64
 24  EXT_SOURCE_2                 64680 non-null  float64
 25  EXT_SOURCE_3                 51247 non-null  float64
 26  APARTMENTS_AVG               30640 non-null  float64
 27  YEARS_BUILD_AVG              20864 non-null  float64
 28  FLOORSMAX_AVG                31239 non-null  float64
 29  NONLIVINGAREA_AVG            27818 non-null  float64
 30  TOTALAREA_MODE               32196 non-null  float64
 31  DAYS_LAST_PHONE_CHANGE       64825 non-null  float64
 32  LIVINGAREA_MEDI              31047 non-null  float64
 33  NONLIVINGAREA_MEDI           27818 non-null  float64
 34  AMT_REQ_CREDIT_BUREAU_HOUR   55232 non-null  float64
 35  AMT_REQ_CREDIT_BUREAU_DAY    55232 non-null  float64
 36  AMT_REQ_CREDIT_BUREAU_WEEK   55232 non-null  float64
 37  AMT_REQ_CREDIT_BUREAU_MON    55232 non-null  float64
 38  AMT_REQ_CREDIT_BUREAU_QRT    55232 non-null  float64
 39  AMT_REQ_CREDIT_BUREAU_YEAR   55232 non-null  float64
dtypes: float64(24), int64(6), object(10)
memory usage: 20.3+ MB
```

3.3 Clean up null and missing values

In [13]:

```
df2.isnull().sum()
```

executed in 53ms, finished 09:13:36 2022-03-28

Out[13]:

```
TARGET                            0
CNT_CHILDREN                      0
AMT_INCOME_TOTAL                  0
AMT_CREDIT                        0
AMT_ANNUITY                       1
NAME_EDUCATION_TYPE               0
DAYS_EMPLOYED                     0
CNT_FAM_MEMBERS                   1
DAYS_BIRTH                        0
NAME_INCOME_TYPE                  0
AMT_GOODS_PRICE                  62
CODE_GENDER                       0
FLAG_OWN_REALTY                   0
REGION_RATING_CLIENT              0
ORGANIZATION_TYPE                 0
DAYS_REGISTRATION                 0
REGION_POPULATION_RELATIVE        0
NAME_HOUSING_TYPE                 0
NAME_FAMILY_STATUS                0
NAME_TYPE_SUITE                 242
FLAG_OWN_CAR                      0
NAME_CONTRACT_TYPE                0
LIVE_REGION_NOT_WORK_REGION       0
EXT_SOURCE_1                  37224
EXT_SOURCE_2                    145
EXT_SOURCE_3                  13578
APARTMENTS_AVG                34185
YEARS_BUILD_AVG               43961
FLOORSMAX_AVG                 33586
NONLIVINGAREA_AVG             37007
TOTALAREA_MODE                32629
DAYS_LAST_PHONE_CHANGE            0
LIVINGAREA_MEDI               33778
NONLIVINGAREA_MEDI            37007
AMT_REQ_CREDIT_BUREAU_HOUR     9593
AMT_REQ_CREDIT_BUREAU_DAY      9593
AMT_REQ_CREDIT_BUREAU_WEEK     9593
AMT_REQ_CREDIT_BUREAU_MON      9593
AMT_REQ_CREDIT_BUREAU_QRT      9593
AMT_REQ_CREDIT_BUREAU_YEAR     9593
dtype: int64
```

3.4 Replace null values

- Categoricals: Replace with mode

- Numericals: Replace with mean

3.4.1 Separate Numericals & Categoricals

- Complete further investigation of features

In [14]:

```
# Select only numerical features
num = df2.select_dtypes(include=np.number)
num.info()
```

executed in 52ms, finished 09:13:38 2022-03-28

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 64825 entries, 39451 to 307509
Data columns (total 30 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   TARGET                       64825 non-null  int64  
 1   CNT_CHILDREN                 64825 non-null  int64  
 2   AMT_INCOME_TOTAL             64825 non-null  float64
 3   AMT_CREDIT                   64825 non-null  float64
 4   AMT_ANNUITY                  64824 non-null  float64
 5   DAYS_EMPLOYED                64825 non-null  int64  
 6   CNT_FAM_MEMBERS              64824 non-null  float64
 7   DAYS_BIRTH                   64825 non-null  int64  
 8   AMT_GOODS_PRICE              64763 non-null  float64
 9   REGION_RATING_CLIENT         64825 non-null  int64  
 10  DAYS_REGISTRATION            64825 non-null  float64
 11  REGION_POPULATION_RELATIVE   64825 non-null  float64
 12  LIVE_REGION_NOT_WORK_REGION  64825 non-null  int64  
 13  EXT_SOURCE_1                 27601 non-null  float64
 14  EXT_SOURCE_2                 64680 non-null  float64
 15  EXT_SOURCE_3                 51247 non-null  float64
 16  APARTMENTS_AVG               30640 non-null  float64
 17  YEARS_BUILD_AVG              20864 non-null  float64
 18  FLOORSMAX_AVG                31239 non-null  float64
 19  NONLIVINGAREA_AVG            27818 non-null  float64
 20  TOTALAREA_MODE               32196 non-null  float64
 21  DAYS_LAST_PHONE_CHANGE       64825 non-null  float64
 22  LIVINGAREA_MEDI              31047 non-null  float64
 23  NONLIVINGAREA_MEDI           27818 non-null  float64
 24  AMT_REQ_CREDIT_BUREAU_HOUR   55232 non-null  float64
 25  AMT_REQ_CREDIT_BUREAU_DAY    55232 non-null  float64
 26  AMT_REQ_CREDIT_BUREAU_WEEK   55232 non-null  float64
 27  AMT_REQ_CREDIT_BUREAU_MON    55232 non-null  float64
 28  AMT_REQ_CREDIT_BUREAU_QRT    55232 non-null  float64
 29  AMT_REQ_CREDIT_BUREAU_YEAR   55232 non-null  float64
dtypes: float64(24), int64(6)
memory usage: 15.3 MB
```

In [15]:

```
# Replace null values in numerical columns with mean
for i in num.columns[num.isnull().any(axis=0)]:     #---Applying Only on varia
    num[i].fillna(num[i].mean(),inplace=True)

print(num.isnull().sum())
```

executed in 58ms, finished 09:13:38 2022-03-28

```
TARGET                        0
CNT_CHILDREN                  0
AMT_INCOME_TOTAL              0
AMT_CREDIT                    0
AMT_ANNUITY                   0
DAYS_EMPLOYED                 0
CNT_FAM_MEMBERS               0
DAYS_BIRTH                    0
AMT_GOODS_PRICE               0
REGION_RATING_CLIENT          0
DAYS_REGISTRATION             0
REGION_POPULATION_RELATIVE    0
LIVE_REGION_NOT_WORK_REGION   0
EXT_SOURCE_1                  0
EXT_SOURCE_2                  0
EXT_SOURCE_3                  0
APARTMENTS_AVG                0
YEARS_BUILD_AVG               0
FLOORSMAX_AVG                 0
NONLIVINGAREA_AVG             0
TOTALAREA_MODE                0
DAYS_LAST_PHONE_CHANGE        0
LIVINGAREA_MEDI               0
NONLIVINGAREA_MEDI            0
AMT_REQ_CREDIT_BUREAU_HOUR    0
AMT_REQ_CREDIT_BUREAU_DAY     0
AMT_REQ_CREDIT_BUREAU_WEEK    0
AMT_REQ_CREDIT_BUREAU_MON     0
AMT_REQ_CREDIT_BUREAU_QRT     0
AMT_REQ_CREDIT_BUREAU_YEAR    0
dtype: int64
```

In [16]:

```
num.info()
```

executed in 22ms, finished 09:13:39 2022-03-28

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 64825 entries, 39451 to 307509
Data columns (total 30 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   TARGET                       64825 non-null  int64  
 1   CNT_CHILDREN                 64825 non-null  int64  
 2   AMT_INCOME_TOTAL             64825 non-null  float64
 3   AMT_CREDIT                   64825 non-null  float64
 4   AMT_ANNUITY                  64825 non-null  float64
 5   DAYS_EMPLOYED                64825 non-null  int64  
 6   CNT_FAM_MEMBERS              64825 non-null  float64
 7   DAYS_BIRTH                   64825 non-null  int64  
 8   AMT_GOODS_PRICE              64825 non-null  float64
 9   REGION_RATING_CLIENT         64825 non-null  int64  
 10  DAYS_REGISTRATION            64825 non-null  float64
 11  REGION_POPULATION_RELATIVE   64825 non-null  float64
 12  LIVE_REGION_NOT_WORK_REGION  64825 non-null  int64  
 13  EXT_SOURCE_1                 64825 non-null  float64
 14  EXT_SOURCE_2                 64825 non-null  float64
 15  EXT_SOURCE_3                 64825 non-null  float64
 16  APARTMENTS_AVG               64825 non-null  float64
 17  YEARS_BUILD_AVG              64825 non-null  float64
 18  FLOORSMAX_AVG                64825 non-null  float64
 19  NONLIVINGAREA_AVG            64825 non-null  float64
 20  TOTALAREA_MODE               64825 non-null  float64
 21  DAYS_LAST_PHONE_CHANGE       64825 non-null  float64
 22  LIVINGAREA_MEDI              64825 non-null  float64
 23  NONLIVINGAREA_MEDI           64825 non-null  float64
 24  AMT_REQ_CREDIT_BUREAU_HOUR   64825 non-null  float64
 25  AMT_REQ_CREDIT_BUREAU_DAY    64825 non-null  float64
 26  AMT_REQ_CREDIT_BUREAU_WEEK   64825 non-null  float64
 27  AMT_REQ_CREDIT_BUREAU_MON    64825 non-null  float64
 28  AMT_REQ_CREDIT_BUREAU_QRT    64825 non-null  float64
 29  AMT_REQ_CREDIT_BUREAU_YEAR   64825 non-null  float64
dtypes: float64(24), int64(6)
memory usage: 15.3 MB
```

3.4.1.1 Summary of numericals

- AMT_INCOME_TOTAL has one or many outliers that may need to be trimmed.
- Widely varying magnitudes - normalization will be important

3.4.2 Categoricals

In [628]:

```
cats = df2.select_dtypes(include='object')
cats.info()
```

executed in 62ms, finished 14:42:06 2022-03-28

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 64825 entries, 39451 to 307509
Data columns (total 10 columns):
 #   Column               Non-Null Count  Dtype 
---  ------               --------------  ----- 
 0   NAME_EDUCATION_TYPE  64825 non-null  object
 1   NAME_INCOME_TYPE     64825 non-null  object
 2   CODE_GENDER          64825 non-null  object
 3   FLAG_OWN_REALTY      64825 non-null  object
 4   ORGANIZATION_TYPE    64825 non-null  object
 5   NAME_HOUSING_TYPE    64825 non-null  object
 6   NAME_FAMILY_STATUS   64825 non-null  object
 7   NAME_TYPE_SUITE      64583 non-null  object
 8   FLAG_OWN_CAR         64825 non-null  object
 9   NAME_CONTRACT_TYPE   64825 non-null  object
dtypes: object(10)
memory usage: 5.4+ MB
```

3.5 Label Encode or One-Hot Encode Categoricals

- Explore to figure out number of unique values
    - If unique values >5, then label encode

In [629]:

```
cats['NAME_INCOME_TYPE'].value_counts()
```

executed in 23ms, finished 14:42:07 2022-03-28

Out[629]:

```
Working                 35570
Commercial associate    14616
Pensioner               10443
State servant            4179
Unemployed                 10
Student                     4
Maternity leave             2
Businessman                 1
Name: NAME_INCOME_TYPE, dtype: int64
```

In [634]:

```
cats['NAME_TYPE_SUITE'].value_counts(dropna=False)
```

executed in 21ms, finished 14:43:27 2022-03-28

Out[634]:

```
Unaccompanied      52634
Family              8204
Spouse, partner     2391
Children             669
Other_B              431
NaN                  242
Other_A              194
Group of people       60
Name: NAME_TYPE_SUITE, dtype: int64
```

3.5.1 Replace missing values in 'cats['NAME_TYPE_SUITE']' with mode

In [635]:

```
# Replace missing values in 'cats['NAME_TYPE_SUITE']' with mode
cats['NAME_TYPE_SUITE'].fillna(value='Unaccompanied', inplace=True)
cats['NAME_TYPE_SUITE'].value_counts(dropna=False)
```

executed in 24ms, finished 14:43:44 2022-03-28

Out[635]:

```
Unaccompanied      52876
Family              8204
Spouse, partner     2391
Children             669
Other_B              431
Other_A              194
Group of people       60
Name: NAME_TYPE_SUITE, dtype: int64
```

In [636]:

```
cats['FLAG_OWN_CAR'].value_counts()
```

executed in 16ms, finished 14:43:54 2022-03-28

Out[636]:

```
N    43493
Y    21332
Name: FLAG_OWN_CAR, dtype: int64
```

In [637]:

```
cats['NAME_FAMILY_STATUS'].value_counts()
```

executed in 22ms, finished 14:43:54 2022-03-28

Out[637]:

```
Married                  40564
Single / not married     10253
Civil marriage            6744
Separated                 4185
Widow                     3078
Unknown                      1
Name: NAME_FAMILY_STATUS, dtype: int64
```

In [638]:

```
cats['FLAG_OWN_REALTY'].value_counts()
```

executed in 17ms, finished 14:43:55 2022-03-28

Out[638]:

```
Y    44739
N    20086
Name: FLAG_OWN_REALTY, dtype: int64
```

In [639]:

```
cats['NAME_HOUSING_TYPE'].value_counts()
```

executed in 29ms, finished 14:43:55 2022-03-28

Out[639]:

```
House / apartment      56875
With parents            3599
Municipal apartment     2370
Rented apartment        1239
Office apartment         529
Co-op apartment          213
Name: NAME_HOUSING_TYPE, dtype: int64
```

In [640]:

```
cats['NAME_EDUCATION_TYPE'].value_counts()
```

executed in 21ms, finished 14:43:55 2022-03-28

Out[640]:

```
Secondary / secondary special    47616
Higher education                 14062
Incomplete higher                 2205
Lower secondary                    914
Academic degree                     28
Name: NAME_EDUCATION_TYPE, dtype: int64
```

In [641]:

```
cats['NAME_CONTRACT_TYPE'].value_counts()
```

executed in 22ms, finished 14:43:56 2022-03-28

Out[641]:

```
Cash loans         59367
Revolving loans     5458
Name: NAME_CONTRACT_TYPE, dtype: int64
```

In [642]:

```
cats['ORGANIZATION_TYPE'].value_counts()
```

executed in 18ms, finished 14:43:57 2022-03-28

Out[642]:

```
Business Entity Type 3    15088
XNA                       10451
Self-employed              8816
Other                      3398
Business Entity Type 2     2284
Medicine                   2190
Government                 2085
School                     1750
Trade: type 7              1726
Construction               1601
Kindergarten               1396
Business Entity Type 1     1257
Transport: type 4          1210
Trade: type 3               818
Industry: type 3            758
Security                    744
Industry: type 9            672
Industry: type 11           631
Housing                     599
Agriculture                 572
Military                    474
Bank                        456
Transport: type 2           448
Postal                      442
Restaurant                  439
Police                      428
Trade: type 2               376
Security Ministries         365
Transport: type 3           336
Services                    306
Industry: type 7            264
Industry: type 1            239
University                  238
Industry: type 4            200
Hotel                       199
Electricity                 181
Industry: type 5            121
Telecom                     121
Insurance                   120
Emergency                   109
Advertising                 103
Realtor                      99
Trade: type 6                97
Industry: type 2             92
Trade: type 1                72
Culture                      70
Mobile                       70
```

```
Legal Services             66
Industry: type 12          59
Cleaning                   57
Transport: type 1          38
Industry: type 6           19
Religion                   17
Industry: type 10          16
Industry: type 13          14
Trade: type 4              13
Trade: type 5              10
Industry: type 8            5
Name: ORGANIZATION_TYPE, dtype: int64
```

3.5.2 Use label encoding to save space with Organization_Type

- Interpreting Organization Type won't be as important as building a predictive model.

In [643]:

```
# Instantiate LabelEncoder
label_encoder = LabelEncoder()

# create new row for Organization_type with label encoder
cats['ORGANIZATION_TYPE_CODE'] = label_encoder.fit_transform(cats['ORGANIZATIO

cats.head()
```

executed in 51ms, finished 14:43:58 2022-03-28

Out[643]:

	NAME_EDUCATION_TYPE	NAME_INCOME_TYPE	CODE_GENDER	FLAG_OWN_REALTY
39451	Secondary / secondary special	Pensioner	F	Y
80296	Higher education	Commercial associate	M	Y
34978	Secondary / secondary special	State servant	F	Y
214576	Secondary / secondary special	Working	M	N
169952	Secondary / secondary special	Working	M	N

In [644]:

```
# drop Organization_Type column
cats = cats.drop('ORGANIZATION_TYPE', axis=1)

cats.head()
```

executed in 48ms, finished 14:43:59 2022-03-28

Out[644]:

	NAME_EDUCATION_TYPE	NAME_INCOME_TYPE	CODE_GENDER	FLAG_OWN_REALTY
39451	Secondary / secondary special	Pensioner	F	Y
80296	Higher education	Commercial associate	M	Y
34978	Secondary / secondary special	State servant	F	Y
214576	Secondary / secondary special	Working	M	N
169952	Secondary / secondary special	Working	M	N

3.5.3 One Hot Encode Categoricals

In [645]:

```
# Separate Label Encoded Column
label_encoded = cats['ORGANIZATION_TYPE_CODE']

cats = cats.drop('ORGANIZATION_TYPE_CODE', axis=1)

cats.head()
```

executed in 45ms, finished 14:44:00 2022-03-28

Out[645]:

	NAME_EDUCATION_TYPE	NAME_INCOME_TYPE	CODE_GENDER	FLAG_OWN_REALTY
39451	Secondary / secondary special	Pensioner	F	Y
80296	Higher education	Commercial associate	M	Y
34978	Secondary / secondary special	State servant	F	Y
214576	Secondary / secondary special	Working	M	N
169952	Secondary / secondary special	Working	M	N

In [646]:

```
# copy categoricals
cat_dummies = cats.copy()

# get dummies for categoricals from cat_dummies
cat_dummies = pd.get_dummies(cat_dummies, drop_first=True)

cat_dummies.head()
```

executed in 161ms, finished 14:44:01 2022-03-28

Out[646]:

	NAME_EDUCATION_TYPE_Higher education	NAME_EDUCATION_TYPE_Incomplete higher	NAME_EDUCAT
39451	0	0	
80296	1	0	
34978	0	0	
214576	0	0	
169952	0	0	

5 rows × 31 columns

In [647]:

```
# print shape - column, row attributes
print(cat_dummies.shape)
```

executed in 12ms, finished 14:44:02 2022-03-28

```
(64825, 31)
```

In [648]:

```
# replace with mode for categorical variables
# df5['NAME_TYPE_SUITE'].fillna(df5['NAME_TYPE_SUITE'].mode()[0], inplace=True
# print('NAME_TYPE_SUITE:', df5['NAME_TYPE_SUITE'].describe())
# print()

```

executed in 4ms, finished 14:44:03 2022-03-28

3.6 Concatenate data frames

- Did not remove outliers

In [649]:

```
# concatenate cat_dummies and numericals

df3 = pd.concat([num,
                 cat_dummies,
                label_encoded], axis=1)
df3.head()
```

executed in 181ms, finished 14:44:04 2022-03-28

Out[649]:

	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	DAYS_
39451	0	0	270000.0	573408.0	29407.5	
80296	0	1	90000.0	225000.0	10039.5	
34978	0	1	180000.0	715095.0	48109.5	
214576	0	0	247500.0	765000.0	22365.0	
169952	0	0	202500.0	528633.0	38817.0	

5 rows × 62 columns

In [650]:

```
df3.isnull().sum()
```

executed in 74ms, finished 14:44:05 2022-03-28

Out[650]:

```
TARGET                              0
CNT_CHILDREN                        0
AMT_INCOME_TOTAL                    0
AMT_CREDIT                          0
AMT_ANNUITY                         0
                                   ..
NAME_TYPE_SUITE_Spouse, partner     0
NAME_TYPE_SUITE_Unaccompanied       0
FLAG_OWN_CAR_Y                      0
NAME_CONTRACT_TYPE_Revolving loans  0
ORGANIZATION_TYPE_CODE              0
Length: 62, dtype: int64
```

In [651]:

```
# Check value counts and Dtypes
df3.info()
```

executed in 89ms, finished 14:44:06 2022-03-28

```
ull  float64
 24  AMT_REQ_CREDIT_BUREAU_HOUR                         64825 non-n
ull  float64
 25  AMT_REQ_CREDIT_BUREAU_DAY                          64825 non-n
ull  float64
 26  AMT_REQ_CREDIT_BUREAU_WEEK                         64825 non-n
ull  float64
 27  AMT_REQ_CREDIT_BUREAU_MON                          64825 non-n
ull  float64
 28  AMT_REQ_CREDIT_BUREAU_QRT                          64825 non-n
ull  float64
 29  AMT_REQ_CREDIT_BUREAU_YEAR                         64825 non-n
ull  float64
 30  NAME_EDUCATION_TYPE_Higher education               64825 non-n
ull  uint8
 31  NAME_EDUCATION_TYPE_Incomplete higher              64825 non-n
ull  uint8
 32  NAME_EDUCATION_TYPE_Lower secondary                64825 non-n
ull  uint8
 33  NAME_EDUCATION_TYPE_Secondary / secondary special  64825 non-n
```

3.6.1 Select Features

3.7 SMOTE - Oversample Minority Class

3.7.1 Train, Test, Split

In [249]:

```
# Define X and y
y = df3['TARGET']
X = df3.drop('TARGET', axis=1)

# Perform train-test split with random_state=42 and stratify=y
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, rando
```

executed in 278ms, finished 10:03:45 2022-03-28

In [250]:

```
# Previous original class distribution
print(y_train.value_counts())

# Fit SMOTE to training data
X_train_resampled, y_train_resampled = SMOTE().fit_resample(X_train, y_train)

# Preview synthetic sample class distribution
print('\n')
print(pd.Series(y_train_resampled).value_counts())

# Note, if you get an Attribute Error: 'SMOTE' object has no attribute
# '_validate_data', then downgrade your version of imblearn to 0.6.2
# or upgrade your version of sklearn to 0.23
```

executed in 2.13s, finished 10:03:47 2022-03-28

```
0    23963
1    14932
Name: TARGET, dtype: int64


1    23963
0    23963
Name: TARGET, dtype: int64
```

4 Random Forest Model 1

- Prioritize Accuracy for Loan Approval

In [527]:

```
# Scale train and test sets with StandardScaler
X_train_std = StandardScaler().fit_transform(X_train_resampled)
X_test_std = StandardScaler().fit_transform(X_test)
```

executed in 434ms, finished 13:09:42 2022-03-28

In [528]:

```
# Instantiate and fit a RandomForestClassifier
forest1 = RandomForestClassifier()
forest1.fit(X_train_std, y_train_resampled)
```

executed in 18.3s, finished 13:10:00 2022-03-28

Out[528]:

```
RandomForestClassifier()
```

In [529]:

```
# Training accuracy score
forest1.score(X_train_std, y_train_resampled)
```

executed in 2.18s, finished 13:10:03 2022-03-28

Out[529]:

```
1.0
```

In [530]:

```
# Test accuracy score
forest1.score(X_test_std, y_test)
```

executed in 640ms, finished 13:10:03 2022-03-28

Out[530]:

```
0.3870420362514462
```

In [531]:

```
# Train set predictions
train_pred = forest1.predict(X_train_std)

# Test set predictions
pred = forest1.predict(X_test_std)
```

executed in 2.60s, finished 13:10:06 2022-03-28

In [532]:

```
# Confusion matrix and classification report
print("Training Set")
print(confusion_matrix(y_train_resampled, train_pred))
print(classification_report(y_train_resampled, train_pred))
print()
print("Test Set")
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))
```

executed in 349ms, finished 13:10:06 2022-03-28

```
Training Set
[[23963     0]
 [    0 23963]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     23963
           1       1.00      1.00      1.00     23963

    accuracy                           1.00     47926
   macro avg       1.00      1.00      1.00     47926
weighted avg       1.00      1.00      1.00     47926


Test Set
[[  162 15875]
 [   19  9874]]
              precision    recall  f1-score   support

           0       0.90      0.01      0.02     16037
           1       0.38      1.00      0.55      9893

    accuracy                           0.39     25930
   macro avg       0.64      0.50      0.29     25930
weighted avg       0.70      0.39      0.22     25930
```

4.0.1 Summary - Random Forest Part 1

- Model is fairly proportional and has a fairly high recall score for Approval, which is desirable for my business case.
- I want the model to have a recall score for approval that is above 85%, so I will work in that direction.

4.1 Random Forest Model 2

- Improve Accuracy for Loan Approval

In [533]:

```
# Scale train and test sets with StandardScaler
X_train_std = StandardScaler().fit_transform(X_train_resampled)
X_test_std = StandardScaler().fit_transform(X_test)
```

executed in 366ms, finished 13:10:43 2022-03-28

In [534]:

```
# Instantiate and fit a RandomForestClassifier
forest2 = RandomForestClassifier(n_estimators=20, max_depth= 10)
forest2.fit(X_train_std, y_train_resampled)
```

executed in 2.10s, finished 13:10:45 2022-03-28

Out[534]:

```
RandomForestClassifier(max_depth=10, n_estimators=20)
```

In [535]:

```
# Training accuracy score
forest2.score(X_train_std, y_train_resampled)
```

executed in 276ms, finished 13:10:45 2022-03-28

Out[535]:

```
0.7767182740057589
```

In [536]:

```
# Test accuracy score
forest2.score(X_test_std, y_test)
```

executed in 138ms, finished 13:10:45 2022-03-28

Out[536]:

```
0.39332819128422675
```

In [537]:

```
# Train set predictions
train_pred = forest2.predict(X_train_std)

# Test set predictions
pred = forest.predict(X_test_std)
```

executed in 922ms, finished 13:10:46 2022-03-28

In [538]:

```python
# Confusion matrix and classification report
print("Training Set")
print(confusion_matrix(y_train_resampled, train_pred))
print(classification_report(y_train_resampled, train_pred))
print()
print("Test Set")
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))
```

executed in 325ms, finished 13:10:46 2022-03-28

```
Training Set
[[18895  5068]
 [ 5633 18330]]
              precision    recall  f1-score   support

           0       0.77      0.79      0.78     23963
           1       0.78      0.76      0.77     23963

    accuracy                           0.78     47926
   macro avg       0.78      0.78      0.78     47926
weighted avg       0.78      0.78      0.78     47926


Test Set
[[6401 9636]
 [1598 8295]]
              precision    recall  f1-score   support

           0       0.80      0.40      0.53     16037
           1       0.46      0.84      0.60      9893

    accuracy                           0.57     25930
   macro avg       0.63      0.62      0.56     25930
weighted avg       0.67      0.57      0.56     25930
```

In [263]:

```
# Feature importance
forest2.feature_importances_
```

executed in 12ms, finished 10:04:07 2022-03-28

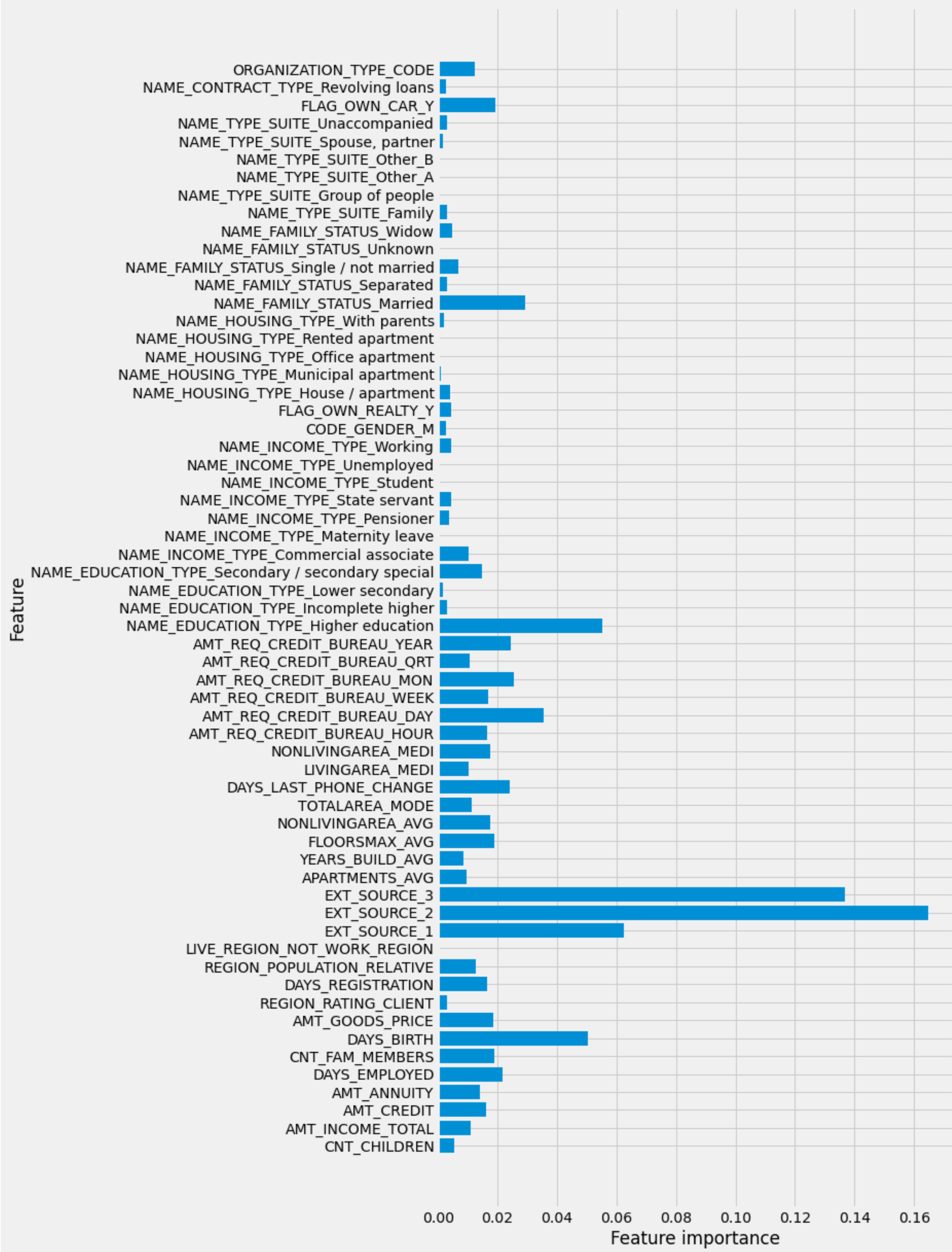
Out[263]:

```
array([5.35380974e-03, 1.09003607e-02, 1.60531531e-02, 1.42051689e-0
2,
       2.16308836e-02, 1.90955362e-02, 5.02991533e-02, 1.85075935e-0
2,
       2.93613644e-03, 1.65623085e-02, 1.26029516e-02, 7.65647224e-0
4,
       6.23557756e-02, 1.64916165e-01, 1.36652702e-01, 9.65744044e-0
3,
       8.55606933e-03, 1.88041795e-02, 1.76657200e-02, 1.14072587e-0
2,
       2.42227780e-02, 1.02140684e-02, 1.76260954e-02, 1.65582583e-0
2,
       3.55447583e-02, 1.69499030e-02, 2.53770956e-02, 1.05432531e-0
2,
       2.43893231e-02, 5.51166479e-02, 3.04160478e-03, 1.62320209e-0
3,
       1.46568602e-02, 1.03434341e-02, 0.00000000e+00, 3.85068891e-0
3,
       4.27839328e-03, 0.00000000e+00, 1.13096311e-05, 4.51139986e-0
3,
       2.73567220e-03, 4.48012922e-03, 3.99029371e-03, 9.74626201e-0
4,
       6.21118520e-04, 3.28847618e-04, 1.99635035e-03, 2.94173835e-0
2,
       3.13429944e-03, 6.71944772e-03, 0.00000000e+00, 4.85091101e-0
3,
       3.02718709e-03, 0.00000000e+00, 1.90660154e-04, 3.71205625e-0
4,
       1.52878934e-03, 3.14043895e-03, 1.94537999e-02, 2.82657779e-0
3,
       1.24251744e-02])
```

In [264]:

```
def plot_feature_importances(model):
    n_features = X_train.shape[1]
    plt.figure(figsize=(8,20))
    plt.barh(range(n_features), model.feature_importances_, align='center')
    plt.yticks(np.arange(n_features), X_train.columns.values)
    plt.xlabel('Feature importance')
    plt.ylabel('Feature')

plot_feature_importances(forest2)
```

executed in 1.90s, finished 10:04:10 2022-03-28



4.1.1 Summary - Random Forest Model 3

- Model almost always predicts approval - probably related to large value for max_depth
- Not a good enough model to recommend for business case as almost every application would be passed on to a human, nullifying the benefit of this model.
- Neat to see the feature importances as this will be important for building a business case and recommending which features to collect data on in the future.

4.2 Random Forest Model 3

- Improve Accuracy for Loan Approval by using standard scaler

In [539]:

```
# Scale train and test sets with StandardScaler
X_train_std = StandardScaler().fit_transform(X_train_resampled)
X_test_std = StandardScaler().fit_transform(X_test)
```

executed in 277ms, finished 13:11:20 2022-03-28

In [540]:

```
# Instantiate and fit a RandomForestClassifier
forest3 = RandomForestClassifier(criterion='gini',n_estimators=3, max_features
forest3.fit(X_train_std, y_train_resampled)
```

executed in 137ms, finished 13:11:21 2022-03-28

Out[540]:

```
RandomForestClassifier(max_depth=4, max_features=2, n_estimators=3)
```

In [541]:

```
# Training accuracy score
forest3.score(X_train_std, y_train_resampled)
```

executed in 49ms, finished 13:11:21 2022-03-28

Out[541]:

```
0.6397779910695656
```

In [542]:

```
# Test accuracy score
forest3.score(X_test_std, y_test)
```

executed in 29ms, finished 13:11:22 2022-03-28

Out[542]:

```
0.4876205167759352
```

In [543]:

```
# Train set predictions
train_pred = forest3.predict(X_train_std)

# Test set predictions
pred = forest3.predict(X_test_std)
```

executed in 49ms, finished 13:11:22 2022-03-28

In [544]:

```
# Confusion matrix and classification report
print("Training Set")
print(confusion_matrix(y_train_resampled, train_pred))
print(classification_report(y_train_resampled, train_pred))
print()
print("Test Set")
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))
```

executed in 297ms, finished 13:11:23 2022-03-28

```
Training Set
[[15029  8934]
 [ 8330 15633]]
              precision    recall  f1-score   support

           0       0.64      0.63      0.64     23963
           1       0.64      0.65      0.64     23963

    accuracy                           0.64     47926
   macro avg       0.64      0.64      0.64     47926
weighted avg       0.64      0.64      0.64     47926


Test Set
[[ 4055 11982]
 [ 1304  8589]]
              precision    recall  f1-score   support

           0       0.76      0.25      0.38     16037
           1       0.42      0.87      0.56      9893

    accuracy                           0.49     25930
   macro avg       0.59      0.56      0.47     25930
weighted avg       0.63      0.49      0.45     25930
```

4.2.1 Summary - Random Forest Part 2

- Model almost always predicts approval - don't know why exactly
- Not a good enough model to recommend for business cas as almost every application would be passed on to a human, nullifying the benefit of this model.

4.3 Random Forest Model 4

- Try without standard scalar

In [545]:

```
# Instantiate and fit a RandomForestClassifier
forest4 = RandomForestClassifier(criterion='gini',n_estimators=8, max_features
forest4.fit(X_train_resampled, y_train_resampled)
```

executed in 269ms, finished 13:11:56 2022-03-28

Out[545]:

```
RandomForestClassifier(max_depth=2, max_features=2, n_estimators=8)
```

In [546]:

```
# Training accuracy score
forest4.score(X_train_resampled, y_train_resampled)
```

executed in 83ms, finished 13:11:56 2022-03-28

Out[546]:

```
0.6506280515795184
```

In [547]:

```
# Test accuracy score
forest4.score(X_test, y_test)
```

executed in 41ms, finished 13:11:57 2022-03-28

Out[547]:

```
0.6104126494408022
```

In [548]:

```
# Train set predictions
train_pred = forest4.predict(X_train_std)

# Test set predictions
pred = forest4.predict(X_test_std)
```

executed in 60ms, finished 13:11:57 2022-03-28

In [549]:

```
# Confusion matrix and classification report
print("Training Set")
print(confusion_matrix(y_train_resampled, train_pred))
print(classification_report(y_train_resampled, train_pred))
print()
print("Test Set")
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))
```

executed in 258ms, finished 13:11:58 2022-03-28

```
Training Set
[[13648 10315]
 [ 7380 16583]]
              precision    recall  f1-score   support

           0       0.65      0.57      0.61     23963
           1       0.62      0.69      0.65     23963

    accuracy                           0.63     47926
   macro avg       0.63      0.63      0.63     47926
weighted avg       0.63      0.63      0.63     47926


Test Set
[[6821 9216]
 [2211 7682]]
              precision    recall  f1-score   support

           0       0.76      0.43      0.54     16037
           1       0.45      0.78      0.57      9893

    accuracy                           0.56     25930
   macro avg       0.60      0.60      0.56     25930
weighted avg       0.64      0.56      0.56     25930
```

4.3.1 Summary - Random Forest Model 4

- This model is the most balanced of the models I have been able to create, with a recall score that is similar for approval and refusal.

4.4 Random Forest Model 5

- Try without standard scaler and higher n_estimators

In [550]:

```
# Instantiate and fit a RandomForestClassifier
forest5 = RandomForestClassifier(criterion='gini',n_estimators=100, max_featur
forest5.fit(X_train_resampled, y_train_resampled)
```

executed in 2.67s, finished 13:12:18 2022-03-28

Out[550]:

```
RandomForestClassifier(max_depth=2, max_features=8)
```

In [551]:

```
# Training accuracy score
forest5.score(X_train_resampled, y_train_resampled)
```

executed in 509ms, finished 13:12:19 2022-03-28

Out[551]:

```
0.6988273588448859
```

In [552]:

```
# Test accuracy score
forest5.score(X_test, y_test)
```

executed in 298ms, finished 13:12:19 2022-03-28

Out[552]:

```
0.6612032394909372
```

In [601]:

```
# Train set predictions
train_pred = forest5.predict(X_train_std)

# Test set predictions
pred = forest5.predict(X_test_std)
```

executed in 571ms, finished 13:35:21 2022-03-28

In [602]:

```
# Confusion matrix and classification report
print("Training Set")
print(confusion_matrix(y_train_resampled, train_pred))
print(classification_report(y_train_resampled, train_pred))
print()
print("Test Set")
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))
```

executed in 270ms, finished 13:35:21 2022-03-28

```
Training Set
[[ 8857 15106]
 [ 2697 21266]]
              precision    recall  f1-score   support

           0       0.77      0.37      0.50     23963
           1       0.58      0.89      0.70     23963

    accuracy                           0.63     47926
   macro avg       0.68      0.63      0.60     47926
weighted avg       0.68      0.63      0.60     47926


Test Set
[[ 5582 10455]
 [ 1317  8576]]
              precision    recall  f1-score   support

           0       0.81      0.35      0.49     16037
           1       0.45      0.87      0.59      9893

    accuracy                           0.55     25930
   macro avg       0.63      0.61      0.54     25930
weighted avg       0.67      0.55      0.53     25930
```

4.4.1 Summary - Random Forest Model 5

- Higher n_estimators, means the model optimizes for overall accuracy, which reduces the recall score for loan approval.
- Reducing max_depth to 2, lead to higher recall score for loan approval (.85) and allows me to filter ~25% of the applications out as refusals.

4.5 Random Forest Model 6

- Optimize for high accuracy with rejection to add to voting classifier

In [583]:

```
# Instantiate and fit a RandomForestClassifier
forest6 = RandomForestClassifier()
forest6.fit(X_train, y_train)
```

executed in 12.6s, finished 13:21:54 2022-03-28

Out[583]:

```
RandomForestClassifier()
```

In [584]:

```
# Training accuracy score
forest6.score(X_train, y_train)
```

executed in 1.92s, finished 13:21:56 2022-03-28

Out[584]:

```
1.0
```

In [588]:

```
# Test accuracy score
forest6.score(X_test, y_test)
```

executed in 2.26s, finished 13:23:45 2022-03-28

Out[588]:

```
0.6968762051677594
```

In [589]:

```
# Train set predictions
train_pred = forest6.predict(X_train)

# Test set predictions
pred = forest6.predict(X_test)
```

executed in 5.32s, finished 13:23:52 2022-03-28

In [591]:

```
# Confusion matrix and classification report
print("Training Set")
print(confusion_matrix(y_train, train_pred))
print(classification_report(y_train, train_pred))
print()
print("Test Set")
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))
```

executed in 318ms, finished 13:24:22 2022-03-28

```
Training Set
[[23963     0]
 [    0 14932]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     23963
           1       1.00      1.00      1.00     14932

    accuracy                           1.00     38895
   macro avg       1.00      1.00      1.00     38895
weighted avg       1.00      1.00      1.00     38895


Test Set
[[13659  2378]
 [ 5482  4411]]
              precision    recall  f1-score   support

           0       0.71      0.85      0.78     16037
           1       0.65      0.45      0.53      9893

    accuracy                           0.70     25930
   macro avg       0.68      0.65      0.65     25930
weighted avg       0.69      0.70      0.68     25930
```

4.5.1 Summary - Random Forest Model 6

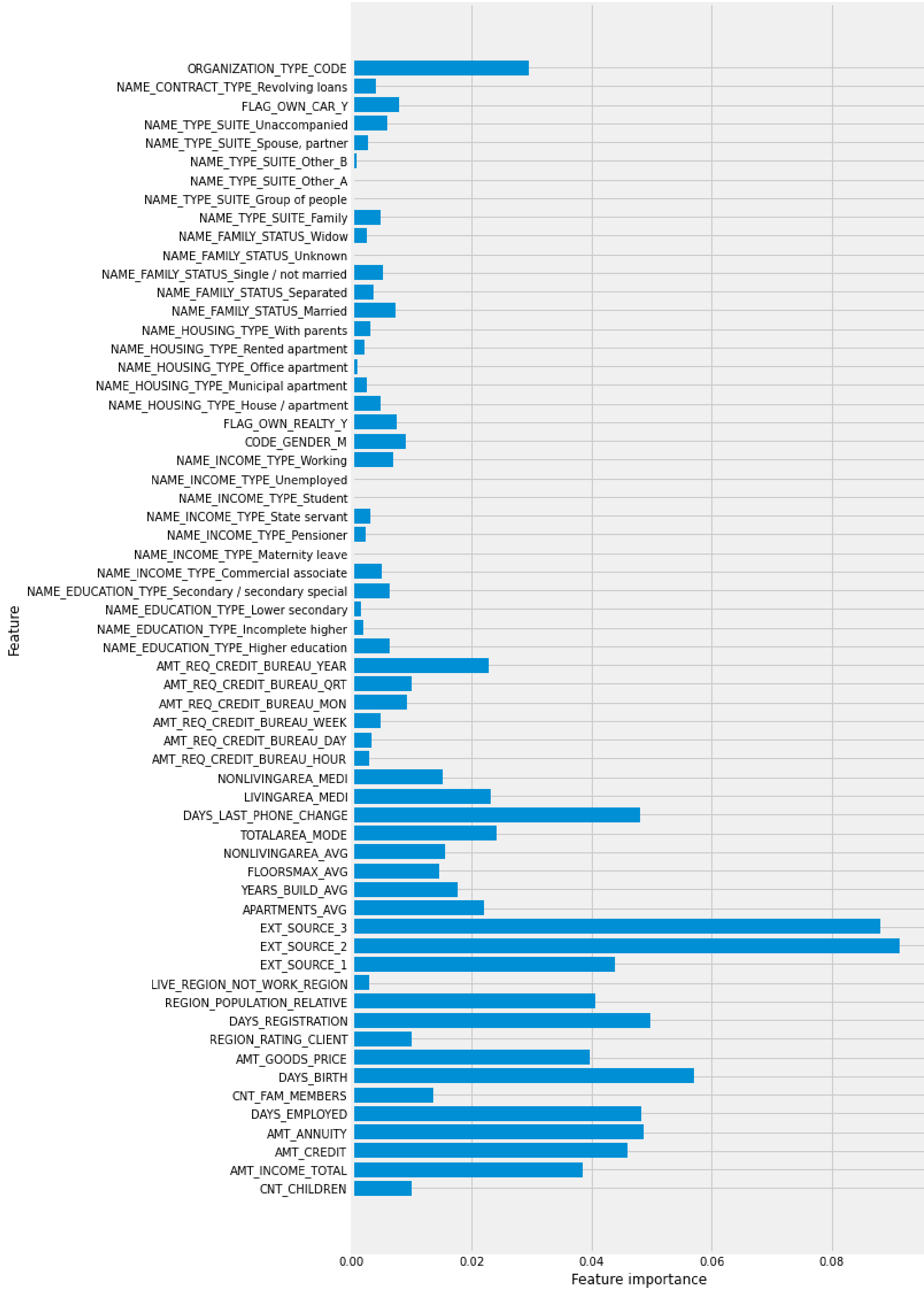
- Higher n_estimators, means the model optimizes for overall accuracy, which reduces the recall score for loan approval.
- Reducing max_depth to 2, lead to higher recall score for loan approval (.85) and allows me to filter ~25% of the applications out as refusals.

4.5.2 Feature Importance Chart

In [610]:

```
def plot_feature_importances(model):
    n_features = X_train.shape[1]
    plt.figure(figsize=(8,20))
    plt.barh(range(n_features), model.feature_importances_, align='center')
    plt.yticks(np.arange(n_features), X_train.columns.values)
    plt.xlabel('Feature importance')
    plt.ylabel('Feature')

plot_feature_importances(forest6)
```

executed in 3.46s, finished 14:08:23 2022-03-28



4.5.3 Improve Feature Importance Bar Chart

- Ascending order for top 10 values

In [611]:

```
# Feature importance
features = pd.DataFrame(forest6.feature_importances_)
features['Feature'] = X_train.columns.values
features['Feature Importance'] = features[0]
features = features.drop(0, axis=1)
features = features.sort_values(by=['Feature Importance'], ascending=True)
features = features.nlargest(n=10, columns=['Feature Importance'])
features
```

executed in 194ms, finished 14:08:24 2022-03-28

Out[611]:

	Feature	Feature Importance
13	EXT_SOURCE_2	0.091410
14	EXT_SOURCE_3	0.088133
6	DAYS_BIRTH	0.057027
9	DAYS_REGISTRATION	0.049775
3	AMT_ANNUITY	0.048751
4	DAYS_EMPLOYED	0.048270
20	DAYS_LAST_PHONE_CHANGE	0.048093
2	AMT_CREDIT	0.045905
12	EXT_SOURCE_1	0.043791
10	REGION_POPULATION_RELATIVE	0.040657

In [612]:

```
import matplotlib.style as style
# style.available
style.use('fivethirtyeight')
```

executed in 8ms, finished 14:08:24 2022-03-28

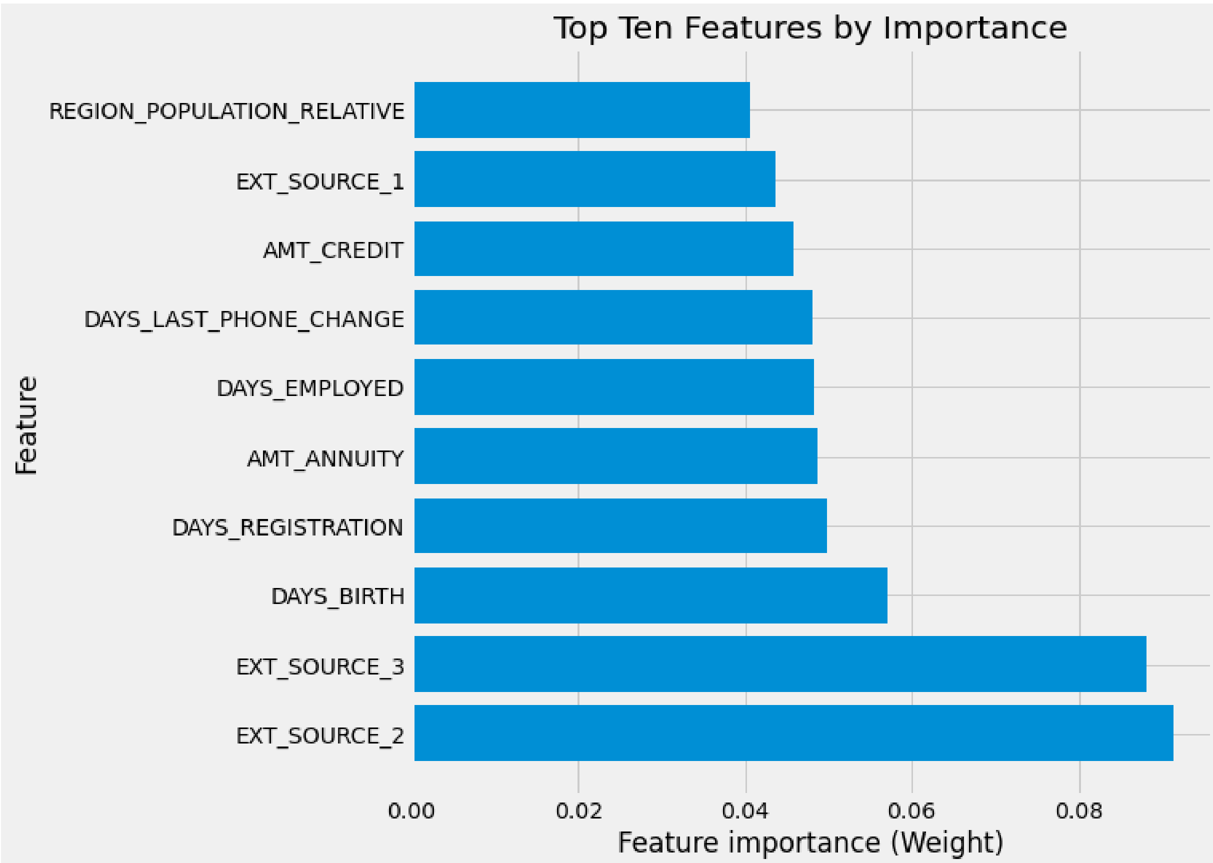
In [613]:

```
plt.figure(figsize=(8,8))
plt.barh(range(10), features['Feature Importance'], align='center')
plt.yticks(np.arange(10), features['Feature'])
plt.xlabel('Feature importance (Weight)')
plt.ylabel('Feature')
plt.title('Top Ten Features by Importance')
```

executed in 1.27s, finished 14:08:25 2022-03-28

Out[613]:

```
Text(0.5, 1.0, 'Top Ten Features by Importance')
```



4.6 Explore EXT_SOURCE data

In [455]:

```
df3['EXT_SOURCE_1'].describe()
```

executed in 30ms, finished 10:46:23 2022-03-28

Out[455]:

```
count    64825.000000
mean         0.465946
std          0.141130
min          0.014568
25%          0.465946
50%          0.465946
75%          0.465946
max          0.942680
Name: EXT_SOURCE_1, dtype: float64
```

5 Check Gradient Boost & Tune Learning Rate

- Iterated on this many times -
  - Default learning rate (near 0.1) is the best.
    - Learning rate should be in range of 0.0 - 1.0

In [438]:

```
# Scale train and test sets with StandardScaler from SMOTE
# X_train_std = StandardScaler().fit_transform(X_train_resampled)
# X_test_std = StandardScaler().fit_transform(X_test)
```

executed in 7ms, finished 10:37:16 2022-03-28

In [560]:

```
# Instantiate an GradientBoostingClassifier
gbt_clf1 = GradientBoostingClassifier(learning_rate=0.1, random_state=42)
```

executed in 15ms, finished 13:13:55 2022-03-28

In [561]:

```
# Fit GradientBoostingClassifier
gbt_clf1.fit(X_train_resampled, y_train_resampled)
```

executed in 41.5s, finished 13:14:36 2022-03-28

Out[561]:

```
GradientBoostingClassifier(random_state=42)
```

In [562]:

```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=3,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=100,
                           n_iter_no_change=None, presort='auto',
                           random_state=42, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
```

executed in 13ms, finished 13:14:36 2022-03-28

Out[562]:

```
GradientBoostingClassifier(presort='auto', random_state=42)
```

In [563]:

```
# GradientBoosting model predictions
gbt_clf_train_preds = gbt_clf1.predict(X_train_resampled)
gbt_clf_test_preds = gbt_clf1.predict(X_test_std)
```

executed in 528ms, finished 13:14:37 2022-03-28

In [564]:

```
# Confusion matrix and classification report for training set
print("Training Set")
print(confusion_matrix(y_train_resampled, gbt_clf_train_preds))
print(classification_report(y_train_resampled, gbt_clf_train_preds))

print("Test Set")
# Confusion matrix and classification report for test set
print(confusion_matrix(y_test, gbt_clf_test_preds))
print(classification_report(y_test, gbt_clf_test_preds))
```

executed in 284ms, finished 13:14:37 2022-03-28

```
Training Set
[[18597  5366]
 [ 6426 17537]]
              precision    recall  f1-score   support

           0       0.74      0.78      0.76     23963
           1       0.77      0.73      0.75     23963

    accuracy                           0.75     47926
   macro avg       0.75      0.75      0.75     47926
weighted avg       0.75      0.75      0.75     47926

Test Set
[[ 3598 12439]
 [  718  9175]]
              precision    recall  f1-score   support

           0       0.83      0.22      0.35     16037
           1       0.42      0.93      0.58      9893

    accuracy                           0.49     25930
   macro avg       0.63      0.58      0.47     25930
weighted avg       0.68      0.49      0.44     25930
```

5.0.1 Summary - Gradient Boost Model 1

- This model is also over-trained - leading to an approval of 79% of applications.
- However, the model is 90% accurate at predicting approvals, so it is more accurate than other models and would allow me to remove 21% of the applications, which categorize as refusals, with a higher degree of confidence.

5.1 Gradient Boost Model 2

- Tune for more balanced test distribution

In [565]:

```
# Instantiate an GradientBoostingClassifier
gbt_clf2 = GradientBoostingClassifier(learning_rate=0.05, max_depth=2, min_sam
```

executed in 18ms, finished 13:14:37 2022-03-28

In [566]:

```
# Fit GradientBoostingClassifier
gbt_clf2.fit(X_train_resampled, y_train_resampled)
```

executed in 24.8s, finished 13:15:02 2022-03-28

Out[566]:

```
GradientBoostingClassifier(learning_rate=0.05, max_depth=2, min_sampl
es_leaf=20,
                           random_state=42)
```

In [567]:

```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=3,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=100,
                           n_iter_no_change=None, presort='auto',
                           random_state=42, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
```

executed in 28ms, finished 13:15:02 2022-03-28

Out[567]:

```
GradientBoostingClassifier(presort='auto', random_state=42)
```

In [595]:

```
# GradientBoosting model predictions
gbt_clf_train_preds = gbt_clf2.predict(X_train_resampled)
gbt_clf_test_preds = gbt_clf2.predict(X_test)
```

executed in 403ms, finished 13:31:11 2022-03-28

In [596]:

```
# Confusion matrix and classification report for training set
print("Training Set")
print(confusion_matrix(y_train_resampled, gbt_clf_train_preds))
print(classification_report(y_train_resampled, gbt_clf_train_preds))

print("Test Set")
# Confusion matrix and classification report for test set
print(confusion_matrix(y_test, gbt_clf_test_preds))
print(classification_report(y_test, gbt_clf_test_preds))
```

executed in 269ms, finished 13:31:12 2022-03-28

```
Training Set
[[16989  6974]
 [ 6511 17452]]
              precision    recall  f1-score   support

           0       0.72      0.71      0.72     23963
           1       0.71      0.73      0.72     23963

    accuracy                           0.72     47926
   macro avg       0.72      0.72      0.72     47926
weighted avg       0.72      0.72      0.72     47926

Test Set
[[11287  4750]
 [ 3788  6105]]
              precision    recall  f1-score   support

           0       0.75      0.70      0.73     16037
           1       0.56      0.62      0.59      9893

    accuracy                           0.67     25930
   macro avg       0.66      0.66      0.66     25930
weighted avg       0.68      0.67      0.67     25930
```

5.1.1 Summary - Gradient Boost Model 2

- This model is over-trained - leading to an approval of 84% of applications.
    - It will recommend the removal of ~22% of the applications as refusals.

5.2 Gradient Boost Model 3

In [570]:

```
# Instantiate an GradientBoostingClassifier
gbt_clf3 = GradientBoostingClassifier(random_state=42)
```

executed in 13ms, finished 13:15:04 2022-03-28

In [571]:

```
# Fit GradientBoostingClassifier
gbt_clf3.fit(X_train_resampled, y_train_resampled)
```

executed in 34.0s, finished 13:15:38 2022-03-28

Out[571]:

```
GradientBoostingClassifier(random_state=42)
```

In [572]:

```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=3,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=100,
                           n_iter_no_change=None, presort='auto',
                           random_state=42, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
```

executed in 14ms, finished 13:15:38 2022-03-28

Out[572]:

```
GradientBoostingClassifier(presort='auto', random_state=42)
```

In [593]:

```
# GradientBoosting model predictions
gbt_clf_train_preds = gbt_clf3.predict(X_train_resampled)
gbt_clf_test_preds = gbt_clf3.predict(X_test)
```

executed in 471ms, finished 13:30:39 2022-03-28

In [594]:

```
# Confusion matrix and classification report for training set
print("Training Set")
print(confusion_matrix(y_train_resampled, gbt_clf_train_preds))
print(classification_report(y_train_resampled, gbt_clf_train_preds))

print("Test Set")
# Confusion matrix and classification report for test set
print(confusion_matrix(y_test, gbt_clf_test_preds))
print(classification_report(y_test, gbt_clf_test_preds))
```

executed in 292ms, finished 13:30:39 2022-03-28

```
Training Set
[[18597  5366]
 [ 6426 17537]]
              precision    recall  f1-score   support

           0       0.74      0.78      0.76     23963
           1       0.77      0.73      0.75     23963

    accuracy                           0.75     47926
   macro avg       0.75      0.75      0.75     47926
weighted avg       0.75      0.75      0.75     47926

Test Set
[[12324  3713]
 [ 4270  5623]]
              precision    recall  f1-score   support

           0       0.74      0.77      0.76     16037
           1       0.60      0.57      0.58      9893

    accuracy                           0.69     25930
   macro avg       0.67      0.67      0.67     25930
weighted avg       0.69      0.69      0.69     25930
```

5.2.1 Summary - Gradient Boost Model 3

- This model is also over-trained - leading to an approval of 83% of applications when only 38% were actually approved.
  - However, it is 93% accurate at indicating when an application was approved. Meaning we can predict with a 93% likelihood whether an application will be approved.

6 Gaussian Naive Bayes - Model 1

In [478]:

```
from sklearn.naive_bayes import GaussianNB
clf1 = GaussianNB()
```

executed in 249ms, finished 12:50:45 2022-03-28

In [479]:

```
clf1.fit(X_train_resampled, y_train_resampled)
```

executed in 394ms, finished 12:50:46 2022-03-28

Out[479]:

```
GaussianNB()
```

In [481]:

```
# GradientBoosting model predictions
clf1_train_preds = clf1.predict(X_train_resampled)
clf1_test_preds = clf1.predict(X_test_std)
```

executed in 257ms, finished 12:50:51 2022-03-28

In [482]:

```
# Confusion matrix and classification report for training set
print("Training Set")
print(confusion_matrix(y_train_resampled, clf1_train_preds))
print(classification_report(y_train_resampled, clf1_train_preds))

print("Test Set")
# Confusion matrix and classification report for test set
print(confusion_matrix(y_test, clf1_test_preds))
print(classification_report(y_test, clf1_test_preds))
```

executed in 271ms, finished 12:50:55 2022-03-28

```
Training Set
[[ 9474 14489]
 [ 5995 17968]]
              precision    recall  f1-score   support

           0       0.61      0.40      0.48     23963
           1       0.55      0.75      0.64     23963

    accuracy                           0.57     47926
   macro avg       0.58      0.57      0.56     47926
weighted avg       0.58      0.57      0.56     47926

Test Set
[[16037     0]
 [ 9893     0]]
              precision    recall  f1-score   support

           0       0.62      1.00      0.76     16037
           1       0.00      0.00      0.00      9893

    accuracy                           0.62     25930
   macro avg       0.31      0.50      0.38     25930
weighted avg       0.38      0.62      0.47     25930
```

6.0.1 Summary - Gradient Boost Model 3

- This model is also over-trained - leading to an approval of 83% of applications when only 38% were actually approved.
  - However, it is 93% accurate at indicating when an application was approved. Meaning we can predict with a 93% likelihood whether an application will be approved.

7 Voting Classifier Model 1

- Improve model by combining models that accurately predict each class.

In [607]:

```
clf1 = forest5
clf2 = gbt_clf3
clf3 = forest6
X = X_train_resampled
y = y_train_resampled
eclf1 = VotingClassifier(estimators=[
    ('lr', clf1), ('rf', clf2), ('gnb', clf3)], voting='hard', weights=[2,1,1]
eclf1 = eclf1.fit(X, y)
```

executed in 52.7s, finished 13:38:10 2022-03-28

In [608]:

```
# GradientBoosting model predictions
eclf1_train_preds = eclf1.predict(X_train_resampled)
eclf1_test_preds = eclf1.predict(X_test_std)
```

executed in 4.69s, finished 13:38:14 2022-03-28

In [609]:

```
# Confusion matrix and classification report for training set
print("Training Set")
print(confusion_matrix(y_train_resampled, eclf1_train_preds))
print(classification_report(y_train_resampled, eclf1_train_preds))

print("Test Set")
# Confusion matrix and classification report for test set
print(confusion_matrix(y_test, eclf1_test_preds))
print(classification_report(y_test, eclf1_test_preds))
```

executed in 350ms, finished 13:38:15 2022-03-28

```
Training Set
[[19393  4570]
 [ 7214 16749]]
              precision    recall  f1-score   support

           0       0.73      0.81      0.77     23963
           1       0.79      0.70      0.74     23963

    accuracy                           0.75     47926
   macro avg       0.76      0.75      0.75     47926
weighted avg       0.76      0.75      0.75     47926

Test Set
[[6343 9694]
 [1614 8279]]
              precision    recall  f1-score   support

           0       0.80      0.40      0.53     16037
           1       0.46      0.84      0.59      9893

    accuracy                           0.56     25930
   macro avg       0.63      0.62      0.56     25930
weighted avg       0.67      0.56      0.55     25930
```

7.0.1 Summary - Voting Classifier Model 1

- This is the best model so far that has an approval recall score or True Positive Rate near 0.85 because it also has a True Negative Rate of 0.40 or 40%.

8 Results

- Certain models optimized for best Accuracy (want True Positive Rate > 85%):
  - Weak-Learners, such as Gradient Boost
    - Tuned to high recall score for Approval, low recall score for Refusal. Or, vice-versa. Difficult to balance recall scores for approval and refusal.
  - Decision Trees, such as Random Forest

        - Best for optimizing for True Positive or True Negative Rates (recall scores)
            - Attained 98% True Positive or True Negative rate, but not both in the same model.
    - Ensemble Methods, such as Voting Classifier
        - Find compromise between models, leading to best performance within desired outcome.
- Increasing the number of features greatly improved the accuracy of the model.
    - Trimming these features by feature importance allowed for an even more accurate model, in terms of recall score for loan approval.
- Using standard scaler before fit optimized for increased True Positive Rate, but decreased True Negative Rate.
- Manipulating random_forest hyper-parameters allowed for over-training of the train set, which led to improved accuracy in predicting loan approval (Recall for value of 1).
    - Resulting in an accuracy of 98% in predicting if someone will be approved for a loan.
- Will be able to propose business case for advancing loan applications to the next stage for the banking industry.

9 Reduce Features, Check for Accuracy

- Optimize for accuracy of Loan Approval
- This code is too computationally expensive to run on this data set for my computer(>20 min), but I am curious whether a model with fewer features would produce similar results.

In [100]:

```
# Init the transformer
# rfe = RFE(estimator=RandomForestRegressor(), n_features_to_select=10)

# Fit to the training data
# _ = rfe.fit(X_train_std, y_train_resampled)
```

executed in 5ms, finished 09:15:08 2022-03-25

In [101]:

```
# make predictions for test set based on model
# pred = rfe.predict(X_test_std)
```

executed in 5ms, finished 09:15:13 2022-03-25

In [102]:

```
# Confusion matrix and classification report
# print(confusion_matrix(y_test, pred))
# print(classification_report(y_test, pred))
```

executed in 5ms, finished 09:15:20 2022-03-25

10 Further Study

- Need to better understand the existing features and gather more relevant features.

        - What do EXT_SOURCE_1, 2, and 3 values signify?
        - What do the AMT_REQ_CREDIT_BUREAU values signify?
        - Can we gather credit scores, amount of loan request, amount of cash/market-value of assets?
- Need to test this model on a sample of actual market data to examine how it performs and perform tuning or further iterations of the model.
- Determine which features or data are more accessible.
    - Can we collect these features from customers, or do we need to pay credit rating agencies for the customers' credit scores?
        - Determine how much it would cost to gather data (i.e. credit scores).
        - Determine which data is most valid to collect in an online questionnaire.

In [ ]:

```

```