In [1]:
```python
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```python
import numpy as np
import pandas as pd
from pathlib import Path
from collections import Counter
```

In [4]:
```python
!pip install imblearn
```

```
Collecting imblearn
  Downloading imblearn-0.0-py2.py3-none-any.whl (1.9 kB)
Collecting imbalanced-learn
  Downloading imbalanced_learn-0.8.0-py3-none-any.whl (206 kB)
Requirement already satisfied: scipy>=0.19.1 in c:\users\emman\anaconda3\li
b\site-packages (from imbalanced-learn->imblearn) (1.6.1)
Requirement already satisfied: scikit-learn>=0.24 in c:\users\emman\anacond
a3\lib\site-packages (from imbalanced-learn->imblearn) (0.24.1)
Requirement already satisfied: joblib>=0.11 in c:\users\emman\anaconda3\lib
\site-packages (from imbalanced-learn->imblearn) (1.0.1)
Requirement already satisfied: numpy>=1.13.3 in c:\users\emman\anaconda3\li
b\site-packages (from imbalanced-learn->imblearn) (1.19.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\emman\anaco
nda3\lib\site-packages (from scikit-learn>=0.24->imbalanced-learn->imblear
n) (2.1.0)
Installing collected packages: imbalanced-learn, imblearn
Successfully installed imbalanced-learn-0.8.0 imblearn-0.0
```

In [5]:
```python
from sklearn.metrics import balanced_accuracy_score
from sklearn.metrics import confusion_matrix
from imblearn.metrics import classification_report_imbalanced
```

Read the CSV and Perform Basic Data Cleaning

In [6]:

```
# https://help.lendingclub.com/hc/en-us/articles/215488038-What-do-the-differ

columns = [
    "loan_amnt", "int_rate", "installment", "home_ownership",
    "annual_inc", "verification_status", "issue_d", "loan_status",
    "pymnt_plan", "dti", "delinq_2yrs", "inq_last_6mths",
    "open_acc", "pub_rec", "revol_bal", "total_acc",
    "initial_list_status", "out_prncp", "out_prncp_inv", "total_pymnt",
    "total_pymnt_inv", "total_rec_prncp", "total_rec_int", "total_rec_late_fe
    "recoveries", "collection_recovery_fee", "last_pymnt_amnt", "next_pymnt_c
    "collections_12_mths_ex_med", "policy_code", "application_type", "acc_now
    "tot_coll_amt", "tot_cur_bal", "open_acc_6m", "open_act_il",
    "open_il_12m", "open_il_24m", "mths_since_rcnt_il", "total_bal_il",
    "il_util", "open_rv_12m", "open_rv_24m", "max_bal_bc",
    "all_util", "total_rev_hi_lim", "inq_fi", "total_cu_tl",
    "inq_last_12m", "acc_open_past_24mths", "avg_cur_bal", "bc_open_to_buy",
    "bc_util", "chargeoff_within_12_mths", "delinq_amnt", "mo_sin_old_il_acct
    "mo_sin_old_rev_tl_op", "mo_sin_rcnt_rev_tl_op", "mo_sin_rcnt_tl", "mort_
    "mths_since_recent_bc", "mths_since_recent_inq", "num_accts_ever_120_pd",
    "num_actv_rev_tl", "num_bc_sats", "num_bc_tl", "num_il_tl",
    "num_op_rev_tl", "num_rev_accts", "num_rev_tl_bal_gt_0",
    "num_sats", "num_tl_120dpd_2m", "num_tl_30dpd", "num_tl_90g_dpd_24m",
    "num_tl_op_past_12m", "pct_tl_nvr_dlq", "percent_bc_gt_75", "pub_rec_bank
    "tax_liens", "tot_hi_cred_lim", "total_bal_ex_mort", "total_bc_limit",
    "total_il_high_credit_limit", "hardship_flag", "debt_settlement_flag"
]

target = ["loan_status"]
```

In [18]:

```
# Load the data
file_path = Path('./Resources/LoanStats_2019Q1.csv')
df = pd.read_csv(file_path, skiprows=1)[:-2]
df = df.loc[:, columns].copy()

# Drop the null columns where all values are null
df = df.dropna(axis='columns', how='all')

# Drop the null rows
df = df.dropna()

# Remove the `Issued` loan status
issued_mask = df['loan_status'] != 'Issued'
df = df.loc[issued_mask]

# convert interest rate to numerical
df['int_rate'] = df['int_rate'].str.replace('%', '')
df['int_rate'] = df['int_rate'].astype('float') / 100


# Convert the target column values to low_risk and high_risk based on their v
x = {'Current': 'low_risk'}
df = df.replace(x)

x = dict.fromkeys(['Late (31-120 days)', 'Late (16-30 days)', 'Default', 'In
df = df.replace(x)

df.reset_index(inplace=True, drop=True)

df.head()
```

Out[18]:

	loan_amnt	int_rate	installment	home_ownership	annual_inc	verification_status	issue_d
0	10500.0	0.1719	375.35	RENT	66000.0	Source Verified	Mar-2019
1	25000.0	0.2000	929.09	MORTGAGE	105000.0	Verified	Mar-2019
2	20000.0	0.2000	529.88	MORTGAGE	56000.0	Verified	Mar-2019
3	10000.0	0.1640	353.55	RENT	92000.0	Verified	Mar-2019
4	22000.0	0.1474	520.39	MORTGAGE	52000.0	Not Verified	Mar-2019

5 rows × 86 columns

Split the Data into Training and Testing

In [20]:

```
# Create a DataFrame fro
df_binary_encoded = pd.get_dummies(df, columns=[
    'initial_list_status',
    'home_ownership',
    'verification_status',
    'issue_d',
    'pymnt_plan',
    'next_pymnt_d',
    'application_type',
    'hardship_flag',
    'debt_settlement_flag'])

df_binary_encoded.head()
```

Out[20]:

	loan_amnt	int_rate	installment	annual_inc	loan_status	dti	delinq_2yrs	inq_last_6mths
0	10500.0	0.1719	375.35	66000.0	low_risk	27.24	0.0	0.(
1	25000.0	0.2000	929.09	105000.0	low_risk	20.23	0.0	0.(
2	20000.0	0.2000	529.88	56000.0	low_risk	24.26	0.0	0.(
3	10000.0	0.1640	353.55	92000.0	low_risk	31.44	0.0	1.(
4	22000.0	0.1474	520.39	52000.0	low_risk	18.76	0.0	1.(

5 rows × 96 columns

In [24]:

```
# Create our features
X = df_binary_encoded.drop(columns="loan_status", axis=1)

# Create our target
y = df["loan_status"]
y
```

In [25]:
```
X.describe()
```

Out[25]:

	loan_amnt	int_rate	installment	annual_inc	dti	delinq_2yrs	i
count	68817.000000	68817.000000	68817.000000	6.881700e+04	68817.000000	68817.000000	
mean	16677.594562	0.127718	480.652863	8.821371e+04	21.778153	0.217766	
std	10277.348590	0.048130	288.062432	1.155800e+05	20.199244	0.718367	
min	1000.000000	0.060000	30.890000	4.000000e+01	0.000000	0.000000	
25%	9000.000000	0.088100	265.730000	5.000000e+04	13.890000	0.000000	
50%	15000.000000	0.118000	404.560000	7.300000e+04	19.760000	0.000000	
75%	24000.000000	0.155700	648.100000	1.040000e+05	26.660000	0.000000	
max	40000.000000	0.308400	1676.230000	8.797500e+06	999.000000	18.000000	

8 rows × 95 columns

In [27]:
```
# Check the balance of our target values
y.value_counts()
```

Out[27]:
```
low_risk     68470
high_risk      347
Name: loan_status, dtype: int64
```

In [28]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)

X_train.shape
```

Out[28]:
```
(51612, 95)
```

Ensemble Learners

In this section, you will compare two ensemble algorithms to determine which algorithm results in the best performance. You will train a Balanced Random Forest Classifier and an Easy Ensemble AdaBoost classifier . For each algorithm, be sure to complete the folliowing steps:

1. Train the model using the training data.
2. Calculate the balanced accuracy score from sklearn.metrics.
3. Print the confusion matrix from sklearn.metrics.
4. Generate a classication report using the `imbalanced_classification_report` from imbalanced-learn.
5. For the Balanced Random Forest Classifier onely, print the feature importance sorted in descending order (most important feature to least important) along with the feature score

Note: Use a random state of 1 for each algorithm to ensure consistency between tests

Balanced Random Forest Classifier

Balanced Random Forest Classifier

In [30]:
```
# Resample the training data with the BalancedRandomForestClassifier
from imblearn.ensemble import BalancedRandomForestClassifier

X_train_scaled = X_scaler.transform(X_train)
X_test_scaled = X_scaler.transform(X_test)

X_train_scaled.shape

# Create a random forest classifier
brf_model = BalancedRandomForestClassifier(n_estimators=500, random_state=1)

# Fitting the model
brf_model = rf_model.fit(X_train_scaled, y_train)
```

In [ ]:
```
# Making predictions using the testing data.
predictions = brf_model.predict(X_test_scaled)
predictions
```

In [11]:
```
# Calculated the balanced accuracy score
from sklearn.metrics import accuracy_score

acc_score = accuracy_score(y_test, predictions)
print(acc_score)
```

Out[11]:
```
0.7885466545953005
```

In [12]:
```
# Display the confusion matrix
from sklearn.metrics import confusion_matrix, classification_report

matrix = confusion_matrix(y_test, predictions)
cm_df = pd.DataFrame(
    matrix, index=["Actual High-Risk", "Actual Low-Risk"], columns=["Predicte
cm_df
```

Out[12]:
```
array([[   71,    30],
       [ 2153, 14951]])
```

In [13]:
```
# Print the imbalanced classification report
from imblearn.metrics import classification_report_imbalanced
print(classification_report_imbalanced(y_test, predictions))
```

```
                   pre       rec       spe        f1       geo       iba
sup

  high_risk       0.03      0.70      0.87      0.06      0.78      0.60
101
   low_risk       1.00      0.87      0.70      0.93      0.78      0.62
17104

avg / total       0.99      0.87      0.70      0.93      0.78      0.62
17205
```

In [14]:

```python
# List the features sorted in descending order by feature importance

#importance_df = pd.DataFrame(sorted(zip(rf_model.feature_importances_, X.col
#importance_df.columns = ['Feature_Importance', 'Feature']
sorted(zip(rf_model.feature_importances_, X.columns), reverse=True)
```

```
loan_amnt: (0.07876809003486353)
int_rate: (0.05883806887524815)
installment: (0.05625613759225244)
annual_inc: (0.05355513093134745)
dti: (0.0500331813446525)
delinq_2yrs: (0.02966959508700077)
inq_last_6mths: (0.021129125328012987)
open_acc: (0.01980242888931366)
pub_rec: (0.01747062730041245)
revol_bal: (0.016858293184471483)
total_acc: (0.01641297102011915)
out_prncp: (0.015220714904737209)
out_prncp_inv: (0.015115240704562424)
total_pymnt: (0.014926655663448373)
total_pymnt_inv: (0.014899352873994727)
total_rec_prncp: (0.014881069023035237)
total_rec_int: (0.014859446582326507)
total_rec_late_fee: (0.014832564501144122)
recoveries: (0.014613819728800227)
collection_recovery_fee: (0.014487685026878092)
last_pymnt_amnt: (0.013921085423763812)
collections_12_mths_ex_med: (0.013534131593418711)
policy_code: (0.013364759441576994)
acc_now_delinq: (0.013322898824752 25)
tot_coll_amt: (0.013265926832893358)
tot_cur_bal: (0.013115450898 13887)
open_acc_6m: (0.013045300628985 67)
open_act_il: (0.0130446065288952)
open_il_12m: (0.013030046723135838)
open_il_24m: (0.012855901280381887)
mths_since_rcnt_il: (0.012799085067590 16)
total_bal_il: (0.012773576514405109)
il_util: (0.011968994260747247)
open_rv_12m: (0.010982948025240226)
open_rv_24m: (0.010579906006851516)
max_bal_bc: (0.010575363106694519)
all_util: (0.010320067009550682)
total_rev_hi_lim: (0.010209212170253059)
inq_fi: (0.009753839399393215)
total_cu_tl: (0.009662050208879065)
inq_last_12m: (0.009632472481996241)
acc_open_past_24mths: (0.009393346012674945)
avg_cur_bal: (0.00872448189550355)
bc_open_to_buy: (0.008628938824946404)
bc_util: (0.008330966254402506)
chargeoff_within_12_mths: (0.007570544824579072)
delinq_amnt: (0.007548811505974241)
mo_sin_old_il_acct: (0.007489717491934961)
mo_sin_old_rev_tl_op: (0.007382231721841728)
mo_sin_rcnt_rev_tl_op: (0.007272665006598051)
```

```
mo_sin_rcnt_tl: (0.006998827313196186)
mort_acc: (0.006866662924995743)
mths_since_recent_bc: (0.006714495620628373)
mths_since_recent_inq: (0.006561432872333855)
num_accts_ever_120_pd: (0.006240598451492287)
num_actv_bc_tl: (0.006216409633238659)
num_actv_rev_tl: (0.0061708920490257954)
num_bc_sats: (0.006083218608279307)
num_bc_tl: (0.005640206440873574)
num_il_tl: (0.005634546230136711)
num_op_rev_tl: (0.005131046989565006)
num_rev_accts: (0.005106000423451099)
num_rev_tl_bal_gt_0: (0.005036652777545191)
num_sats: (0.004860024796675963)
num_tl_120dpd_2m: (0.004198582835532627)
num_tl_30dpd: (0.004018916067963884)
num_tl_90g_dpd_24m: (0.0037571920083085985)
num_tl_op_past_12m: (0.003082852259926947)
pct_tl_nvr_dlq: (0.0029133221443170495)
percent_bc_gt_75: (0.002824523629114469)
pub_rec_bankruptcies: (0.002204946377565813)
tax_liens: (0.0020912385738361574)
tot_hi_cred_lim: (0.002015258269512615)
total_bal_ex_mort: (0.0019325773153555006)
total_bc_limit: (0.001901604006185586)
total_il_high_credit_limit: (0.0015046400907840708)
home_ownership_ANY: (0.0014589723334940362)
home_ownership_MORTGAGE: (0.0013727925120781853)
home_ownership_OWN: (0.0011520703643731528)
home_ownership_RENT: (0.0011005704165634263)
verification_status_Not Verified: (0.0009956935704327383)
verification_status_Source Verified: (0.0007150315534652695)
verification_status_Verified: (0.0004955956183545533)
issue_d_Feb-2019: (0.0002730803587770788)
issue_d_Jan-2019: (0.0)
issue_d_Mar-2019: (0.0)
pymnt_plan_n: (0.0)
initial_list_status_f: (0.0)
initial_list_status_w: (0.0)
next_pymnt_d_Apr-2019: (0.0)
next_pymnt_d_May-2019: (0.0)
application_type_Individual: (0.0)
application_type_Joint App: (0.0)
hardship_flag_N: (0.0)
debt_settlement_flag_N: (0.0)
```

Easy Ensemble AdaBoost Classifier

In [15]:
```python
# Train the EasyEnsembleClassifier
from imblearn.ensemble import EasyEnsembleClassifier

eec = EasyEnsembleClassifier(n_estimators=100, random_state=1)
eec.fit(X_train, y_train)
```

Out[15]:
```
EasyEnsembleClassifier(n_estimators=100, random_state=1)
```

In [16]:
```python
# Calculated the balanced accuracy score
from sklearn.metrics import accuracy_score

acc_score2 = accuracy_score(y_test, y_pred)
print(acc_score2)
```

Out[16]:
```
0.9316600714093861
```

In [17]:
```python
# Display the confusion matrix
from sklearn.metrics import confusion_matrix, classification_report

cm_df = pd.DataFrame(
    matrix, index=["Actual High-Risk", "Actual Low-Risk"], columns=["Predicte
cm_df
```

Out[17]:
```
array([[   93,     8],
       [  983, 16121]])
```

In [18]:
```python
# Print the imbalanced classification report
from imblearn.metrics import classification_report_imbalanced

print(classification_report_imbalanced(y_test, y_pred))
```

```
                   pre       rec       spe        f1       geo       iba
sup

  high_risk       0.09      0.92      0.94      0.16      0.93      0.87
101
   low_risk       1.00      0.94      0.92      0.97      0.93      0.87
17104

avg / total       0.99      0.94      0.92      0.97      0.93      0.87
17205
```

In [31]:
```python
# By Emmanuel Martinez
```