In [1]:

```
import warnings
warnings.filterwarnings('ignore')
```

In [2]:

```
import numpy as np
import pandas as pd
from pathlib import Path
from collections import Counter
```

In [44]:

```
from sklearn.metrics import balanced_accuracy_score
from sklearn.metrics import confusion_matrix
from imblearn.metrics import classification_report_imbalanced
from imblearn.ensemble import BalancedRandomForestClassifier
from imblearn.ensemble import EasyEnsembleClassifier
```

Read the CSV and Perform Basic Data Cleaning

In [28]:

```
# https://help.lendingclub.com/hc/en-us/articles/215488038-What-do-the-different
-Note-statuses-mean-

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_fee",
    "recoveries", "collection_recovery_fee", "last_pymnt_amnt", "next_pymnt_d",
    "collections_12_mths_ex_med", "policy_code", "application_type", "acc_now_de
linq",
    "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_acc
",
    "mths_since_recent_bc", "mths_since_recent_inq", "num_accts_ever_120_pd", "n
um_actv_bc_tl",
    "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_bankrup
tcies",
    "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 [29]:

```
# Load the data
file_path = Path('../Resources/LoanStats_2019Q1.csv.zip')
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 valu
es
x = {'Current': 'low_risk'}
df = df.replace(x)

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

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

df.head()
```

Out[29]:

	loan_amnt	int_rate	installment	home_ownership	annual_inc	verification_status	issue_d	l
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 [30]:

```
# Create our features
# YOUR CODE HERE# YOUR CODE HERE
X = pd.get_dummies(df.drop(columns='loan_status', axis ='columns'))

# Create our target
# YOUR CODE HERE
y = df['loan_status']
```

In [31]:

```
X.describe()
```

Out[31]:

	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 [32]:

```
# Check the balance of our target values
y.value_counts()
```

Out[32]:

```
low_risk     68470
high_risk      347
Name: loan_status, dtype: int64
```

In [33]:

```
# Split the X and y into X_train, X_test, y_train, y_test
# YOUR CODE HERE
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[33]:

```
(51612, 95)
```

In [34]:

```
Counter(y_train)
```

Out[34]:

```
Counter({'low_risk': 51366, 'high_risk': 246})
```

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

In [45]:

```
# Resample the training data with the RandomOversampler
# YOUR CODE HERE
model = BalancedRandomForestClassifier(random_state=1, n_estimators =100, n_jobs
=1)
model.fit(X_train, y_train)
```

Out[45]:

```
BalancedRandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_
weight=None,
                               criterion='gini', max_depth=None,
                               max_features='auto', max_leaf_nodes=N
one,
                               max_samples=None, min_impurity_decrea
se=0.0,
                               min_samples_leaf=2, min_samples_split
=2,
                               min_weight_fraction_leaf=0.0, n_estim
ators=100,
                               n_jobs=1, oob_score=False, random_sta
te=1,
                               replacement=False, sampling_strategy=
'auto',
                               verbose=0, warm_start=False)
```

In [46]:

```
y_pred = model.predict(X_test)
balanced_accuracy_score(y_test, y_pred)
```

Out[46]:

```
0.7855052723466922
```

In [47]:

```
# Display the confusion matrix
# YOUR CODE HERE
confusion_matrix(y_test, y_pred)
```

Out[47]:

```
array([[   68,    33],
       [ 1749, 15355]])
```

In [50]:

```
# Print the imbalanced classification report
# YOUR CODE HERE
print(classification_report_imbalanced(y_test, y_pred))
```

```
                   pre       rec       spe        f1       geo
iba       sup

  high_risk       0.04      0.67      0.90      0.07      0.78
0.59       101
   low_risk       1.00      0.90      0.67      0.95      0.78
0.62     17104

avg / total       0.99      0.90      0.67      0.94      0.78
0.62     17205
```

In [51]:

```
# List the features sorted in descending order by feature importance
# YOUR CODE HERE
importances = rf_model.feature_importances_
sorted(zip(rf_model.feature_importances_, X.columns), reverse=True)
```

Out[51]:

```
[(0.0727639593858621, 'total_pymnt_inv'),
 (0.06834306477244799, 'last_pymnt_amnt'),
 (0.06782335661147988, 'total_rec_prncp'),
 (0.05739610678680039, 'total_pymnt'),
 (0.05720777169836709, 'total_rec_int'),
 (0.020688645276999505, 'out_prncp'),
 (0.02036466881582304, 'installment'),
 (0.018656936727152965, 'out_prncp_inv'),
 (0.017857917342116038, 'dti'),
 (0.01601229614899384, 'max_bal_bc'),
 (0.01574247260803747, 'loan_amnt'),
 (0.015731883401048953, 'mo_sin_old_rev_tl_op'),
 (0.015367475924442476, 'avg_cur_bal'),
 (0.014456619795496657, 'bc_open_to_buy'),
 (0.014130125299418065, 'revol_bal'),
 (0.013983344975541495, 'tot_hi_cred_lim'),
 (0.013963065564768017, 'mo_sin_old_il_acct'),
 (0.013865867169455131, 'total_bal_il'),
 (0.0138371824884218, 'tot_cur_bal'),
 (0.013826528564285607, 'bc_util'),
 (0.013665149058710115, 'total_bal_ex_mort'),
 (0.01360526986087437, 'total_il_high_credit_limit'),
 (0.013327468915488852, 'int_rate'),
```

```
(0.01299461620448167, 'total_rev_hi_lim'),
(0.01296670900656197, 'total_rec_late_fee'),
(0.012854935917059981, 'total_bc_limit'),
(0.01274799427043216, 'annual_inc'),
(0.01254789532658696, 'il_util'),
(0.012176291943558636, 'mths_since_rcnt_il'),
(0.011646746340057237, 'total_acc'),
(0.011236380029634419, 'all_util'),
(0.010913758481415506, 'mths_since_recent_inq'),
(0.010502759503267109, 'mths_since_recent_bc'),
(0.010422834496249488, 'num_rev_accts'),
(0.010295418393953577, 'num_il_tl'),
(0.009840897719249599, 'issue_d_Mar-2019'),
(0.009702446816290717, 'total_cu_tl'),
(0.009556508253824142, 'inq_last_12m'),
(0.00932897675241124, 'pct_tl_nvr_dlq'),
(0.009245165682122608, 'acc_open_past_24mths'),
(0.00914479850189322, 'open_acc'),
(0.008766860276414469, 'mo_sin_rcnt_tl'),
(0.008753590831531008, 'num_sats'),
(0.008409991730010362, 'mo_sin_rcnt_rev_tl_op'),
(0.008085080645470996, 'num_bc_tl'),
(0.007451426611443136, 'mort_acc'),
(0.007304122049674221, 'inq_fi'),
(0.007257310709766473, 'num_rev_tl_bal_gt_0'),
(0.00719659726923894, 'num_actv_rev_tl'),
(0.007196049774246929, 'num_tl_op_past_12m'),
(0.007175562861472097, 'open_rv_24m'),
(0.006820955611389246, 'num_op_rev_tl'),
(0.006711197850121663, 'num_actv_bc_tl'),
(0.006661581381759349, 'num_bc_sats'),
(0.006579428715326342, 'open_act_il'),
(0.00650718339074092, 'open_il_24m'),
(0.006295338646049471, 'open_acc_6m'),
(0.006067601744597526, 'next_pymnt_d_May-2019'),
(0.005866170950514825, 'open_rv_12m'),
(0.005624316360809707, 'issue_d_Feb-2019'),
(0.005470623296757692, 'inq_last_6mths'),
(0.005411210759588003, 'tot_coll_amt'),
(0.005360471715600 4065, 'percent_bc_gt_75'),
(0.005344898832340 7465, 'issue_d_Jan-2019'),
(0.004361349323141985, 'next_pymnt_d_Apr-2019'),
(0.004083363836033224, 'delinq_2yrs'),
(0.003919185265970917, 'num_accts_ever_120_pd'),
(0.003629448728616 5082, 'open_il_12m'),
(0.002746569917173 1655, 'pub_rec_bankruptcies'),
(0.002465324168150 4346, 'pub_rec'),
(0.002428028128649017, 'application_type_Joint App'),
(0.002099869389753846, 'application_type_Individual'),
(0.002011319828740 4085, 'verification_status_Not Verified'),
```

```
 (0.0017405870577954924, 'home_ownership_MORTGAGE'),
 (0.0017154649882675569, 'num_tl_90g_dpd_24m'),
 (0.0016741164455101065, 'home_ownership_OWN'),
 (0.0016645294355860287, 'verification_status_Source Verified'),
 (0.0016189406831765398, 'verification_status_Verified'),
 (0.0015748055405672386, 'collections_12_mths_ex_med'),
 (0.0015687370004989923, 'home_ownership_RENT'),
 (0.0014473828406065014, 'initial_list_status_w'),
 (0.0013810022342606511, 'initial_list_status_f'),
 (0.0004531475457369744, 'home_ownership_ANY'),
 (0.0003569447958177142, 'chargeoff_within_12_mths'),
 (0.0, 'tax_liens'),
 (0.0, 'recoveries'),
 (0.0, 'pymnt_plan_n'),
 (0.0, 'policy_code'),
 (0.0, 'num_tl_30dpd'),
 (0.0, 'num_tl_120dpd_2m'),
 (0.0, 'hardship_flag_N'),
 (0.0, 'delinq_amnt'),
 (0.0, 'debt_settlement_flag_N'),
 (0.0, 'collection_recovery_fee'),
 (0.0, 'acc_now_delinq')]
```

Easy Ensemble AdaBoost Classifier

In [52]:

```
# Train the Classifier
# YOUR CODE HERE
model = EasyEnsembleClassifier(random_state=1, n_estimators =100, n_jobs=1)
model.fit(X_train, y_train)
```

Out[52]:

```
EasyEnsembleClassifier(base_estimator=None, n_estimators=100, n_jobs
=1,
                       random_state=1, replacement=False,
                       sampling_strategy='auto', verbose=0, warm_sta
rt=False)
```

In [53]:

```
# Calculated the balanced accuracy score
# YOUR CODE HERE
y_pred = model.predict(X_test)
balanced_accuracy_score(y_test, y_pred)
```

Out[53]:

```
0.9316600714093861
```

In [17]:

```
# Display the confusion matrix
# YOUR CODE HERE
```

Out[17]:

```
array([[   93,     8],
       [  983, 16121]])
```

In [54]:

```
# Print the imbalanced classification report
# YOUR CODE HERE
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 [ ]:

```

```