In [8]:

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

In [9]:

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

In [10]:

```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 [11]:

```python
# 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_delinq",
    "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", "num_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_bankruptcies",
    "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 [12]:

```python
# Load the data
file_path = Path('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 values
x = {'Current': 'low_risk'}
df = df.replace(x)

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

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

df.head()
```

Out[12]:

	loan_amnt	int_rate	installment	home_ownership	annual_inc	verification_status	issue_d	loan_status	pymnt_plan	dti	...	pc
0	10500.0	0.1719	375.35	RENT	66000.0	Source Verified	Mar-2019	low_risk	n	27.24	...	
1	25000.0	0.2000	929.09	MORTGAGE	105000.0	Verified	Mar-2019	low_risk	n	20.23	...	
2	20000.0	0.2000	529.88	MORTGAGE	56000.0	Verified	Mar-2019	low_risk	n	24.26	...	
3	10000.0	0.1640	353.55	RENT	92000.0	Verified	Mar-2019	low_risk	n	31.44	...	
4	22000.0	0.1474	520.39	MORTGAGE	52000.0	Not Verified	Mar-2019	low_risk	n	18.76	...	

5 rows × 86 columns

Split the Data into Training and Testing

In [13]:

```python
# Create a DataFrame fro
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'])

binary_encoded.head()
```

Out[13]:

	loan_amnt	int_rate	installment	annual_inc	loan_status	dti	delinq_2yrs	inq_last_6mths	open_acc	pub_rec	...	issue_d_Feb 201
0	10500.0	0.1719	375.35	66000.0	low_risk	27.24	0.0	0.0	8.0	0.0	...	
1	25000.0	0.2000	929.09	105000.0	low_risk	20.23	0.0	0.0	17.0	1.0	...	
2	20000.0	0.2000	529.88	56000.0	low_risk	24.26	0.0	0.0	8.0	0.0	...	
3	10000.0	0.1640	353.55	92000.0	low_risk	31.44	0.0	1.0	10.0	1.0	...	
4	22000.0	0.1474	520.39	52000.0	low_risk	18.76	0.0	1.0	14.0	0.0	...	

5 rows × 96 columns

In [14]:

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

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

In [15]:

```python
X.describe()
```

Out[15]:

	loan_amnt	int_rate	installment	annual_inc	dti	delinq_2yrs	inq_last_6mths	open_acc	pu
count	68817.000000	68817.000000	68817.000000	6.881700e+04	68817.000000	68817.000000	68817.000000	68817.000000	68817.0
mean	16677.594562	0.127718	480.652863	8.821371e+04	21.778153	0.217766	0.497697	12.587340	0.1
std	10277.348590	0.048130	288.062432	1.155800e+05	20.199244	0.718367	0.758122	6.022869	0.3
min	1000.000000	0.060000	30.890000	4.000000e+01	0.000000	0.000000	0.000000	2.000000	0.0
25%	9000.000000	0.088100	265.730000	5.000000e+04	13.890000	0.000000	0.000000	8.000000	0.0
50%	15000.000000	0.118000	404.560000	7.300000e+04	19.760000	0.000000	0.000000	11.000000	0.0
75%	24000.000000	0.155700	648.100000	1.040000e+05	26.660000	0.000000	1.000000	16.000000	0.0
max	40000.000000	0.308400	1676.230000	8.797500e+06	999.000000	18.000000	5.000000	72.000000	4.0

8 rows × 95 columns

In [17]:

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

Out[17]:

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

In [20]:

```python
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,random_state=1,stratify=y)
```

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 [21]:

```python
# Resample the training data with the BalancedRandomForestClassifier
from imblearn.ensemble import BalancedRandomForestClassifier

clf = BalancedRandomForestClassifier(n_estimators=100, random_state=1)
brf = clf.fit(X_train, y_train)
brf
```

Out[21]:

```
BalancedRandomForestClassifier(random_state=1)
```

In [22]:

```python
# Calculated the balanced accuracy score
y_pred = clf.predict(X_test)
balanced_accuracy_score(y_test, y_pred)
```

Out[22]:

```
0.7913166620335118
```

In [23]:

```python
# Display the confusion matrix
confusion_matrix(y_test, y_pred)
```

Out[23]:

```
array([[   60,    27],
       [ 1832, 15286]])
```

In [26]:

```python
# Print the imbalanced classification report
print(classification_report_imbalanced(y_test, y_pred))
```

```
                   pre       rec       spe        f1       geo       iba       sup

  high_risk       0.03      0.69      0.89      0.06      0.78      0.60        87
   low_risk       1.00      0.89      0.69      0.94      0.78      0.63     17118

avg / total       0.99      0.89      0.69      0.94      0.78      0.63     17205
```

In [27]:

```python
# List the features sorted in descending order by feature importance
sorted(zip(brf.feature_importances_,X.columns),reverse=True)
```

Out[27]:

```
[(0.08688199001638192, 'total_rec_prncp'),
 (0.0590208937579822, 'total_rec_int'),
 (0.05788002535986676, 'last_pymnt_amnt'),
 (0.05337764050442732, 'total_pymnt'),
 (0.047244347238919335, 'total_pymnt_inv'),
 (0.027402075041283256, 'int_rate'),
 (0.019820883059206464, 'annual_inc'),
 (0.018224950995722262, 'issue_d_Jan-2019'),
 (0.018061481401240535, 'installment'),
 (0.01719329529075484, 'out_prncp'),
 (0.017032071726450125, 'dti'),
 (0.016830240433675255, 'mths_since_recent_inq'),
 (0.01646107066464477, 'bc_util'),
 (0.01631214686241231, 'avg_cur_bal'),
 (0.016023051840526978, 'mths_since_rcnt_il'),
 (0.01596431826754691, 'il_util'),
 (0.015312931786500918, 'revol_bal'),
 (0.01490345174111062, 'all_util'),
 (0.0146975483195715, 'mo_sin_old_il_acct'),
 (0.01453012965529148, 'mo_sin_old_rev_tl_op'),
 (0.01421748008244558, 'out_prncp_inv'),
 (0.013722295970305281, 'tot_cur_bal'),
 (0.013573373127185588, 'max_bal_bc'),
 (0.013330479256863426, 'total_il_high_credit_limit'),
 (0.012804100460376104, 'total_rev_hi_lim'),
 (0.012593895028521231, 'tot_hi_cred_lim'),
 (0.012359462339243668, 'issue_d_Mar-2019'),
 (0.011927056457785952, 'total_bc_limit'),
 (0.011864728925297895, 'total_bal_il'),
 (0.011757611545677843, 'mo_sin_rcnt_rev_tl_op'),
 (0.011589517904968422, 'total_bal_ex_mort'),
 (0.011465317720753614, 'num_actv_bc_tl'),
 (0.011389742038584973, 'bc_open_to_buy'),
 (0.01104544706760886, 'pct_tl_nvr_dlq'),
 (0.010990897553456154, 'num_rev_accts'),
 (0.010736827453354384, 'mths_since_recent_bc'),
 (0.010503616015888995, 'num_sats'),
 (0.009765553520470649, 'total_acc'),
 (0.009696788470479934, 'loan_amnt'),
 (0.009221192898681785, 'num_il_tl'),
 (0.009065070397062973, 'num_bc_tl'),
 (0.008691831736663552, 'total_cu_tl'),
 (0.008689239586110578, 'num_bc_sats'),
 (0.008631823090021314, 'inq_last_12m'),
 (0.00859227278606504, 'open_acc'),
 (0.008355568157480684, 'num_rev_tl_bal_gt_0'),
 (0.007909299867887156, 'acc_open_past_24mths'),
 (0.00784638612983659, 'mo_sin_rcnt_tl'),
 (0.007735416380785323, 'num_op_rev_tl'),
 (0.007370333748686457, 'percent_bc_gt_75'),
 (0.007243098053800038, 'open_il_24m'),
 (0.007187091440329451, 'num_actv_rev_tl'),
 (0.007084083947232684, 'inq_fi'),
 (0.007074932141739246, 'open_act_il'),
 (0.006516388607704907, 'num_tl_op_past_12m'),
 (0.006255031645711616, 'mort_acc'),
 (0.006141252844748262, 'open_rv_24m'),
 (0.005776966140303564 5, 'open_acc_6m'),
 (0.00552039836096O441, 'inq_last_6mths'),
 (0.004859523483830137, 'open_rv_12m'),
 (0.004629844174457991, 'open_il_12m'),
 (0.004522346413346704, 'next_pymnt_d_May-2019'),
 (0.004286897098062842, 'num_accts_ever_120_pd'),
 (0.004213747729439285, 'delinq_2yrs'),
 (0.004066256570052635, 'total_rec_late_fee'),
 (0.0036262940499469805, 'tot_coll_amt'),
 (0.003561368880423488, 'issue_d_Feb-2019'),
 (0.0032117942583482816, 'next_pymnt_d_Apr-2019'),
 (0.002589197116852898, 'verification_status_Not Verified'),
 (0.0023618981004151175, 'verification_status_Verified'),
 (0.0023264157721138053, 'home_ownership_OWN'),
 (0.0023154275555716804, 'pub_rec'),
 (0.0021439640070435315, 'home_ownership_RENT'),
 (0.0017790930453640147, 'home_ownership_MORTGAGE'),
 (0.001758783364233998, 'initial_list_status_f'),
 (0.0017511066809955652, 'verification_status_Source Verified'),
 (0.0014570448344382431, 'pub_rec_bankruptcies'),
 (0.0014387297371841158, 'application_type_Joint App'),
 (0.0014108137582575194, 'initial_list_status_w'),
 (0.001376765417322568, 'application_type_Individual'),
 (0.00048792802705261714, 'home_ownership_ANY'),
 (0.00034867442178781544, 'num_tl_90g_dpd_24m'),
 (5.967263886217593e-05, 'collections_12_mths_ex_med'),
 (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, 'chargeoff_within_12_mths'),
 (0.0, 'acc_now_delinq')]
```

Easy Ensemble AdaBoost Classifier

In [28]:

```python
# Train the EasyEnsembleClassifier
from imblearn.ensemble import EasyEnsembleClassifier

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

Out[28]:

```
EasyEnsembleClassifier(n_estimators=100, random_state=1)
```

In [29]:

```python
# Calculated the balanced accuracy score
y_pred = eec.predict(X_test)
balanced_accuracy_score(y_test, y_pred)
```

Out[29]:

```
0.9251352679776481
```

In [30]:

```python
# Display the confusion matrix
confusion_matrix(y_test, y_pred)
```

Out[30]:

```
array([[   79,     8],
       [  989, 16129]])
```

In [31]:

```python
# Print the imbalanced classification report
print(classification_report_imbalanced(y_test, y_pred))
```

```
                   pre       rec       spe        f1       geo       iba       sup

  high_risk       0.07      0.91      0.94      0.14      0.92      0.85        87
   low_risk       1.00      0.94      0.91      0.97      0.92      0.86     17118

avg / total       0.99      0.94      0.91      0.97      0.92      0.86     17205
```

In [ ]: