Ensemble Learning

Initial Imports

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 [3]:
```
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 [25]:
```
# Load the data
pd.set_option('max_columns', None)
file_path = Path('Resources/LoanStats_2019Q1.csv')
df = pd.read_csv(file_path)
```

Out[25]:

	loan_amnt	int_rate	installment	home_ownership	annual_inc	verification_status	issue_
23702	20000.0	0.0819	628.49	RENT	98000.0	Not Verified	Fel 201
46065	6600.0	0.1180	218.59	RENT	35000.0	Source Verified	Jar 201
65412	13000.0	0.1298	295.66	MORTGAGE	132000.0	Verified	Jar 201
41498	20000.0	0.0702	617.73	MORTGAGE	135000.0	Verified	Jar 201
57017	6500.0	0.1691	231.46	MORTGAGE	34776.0	Not Verified	Jar 201
46779	18500.0	0.0819	581.35	RENT	45000.0	Not Verified	Jar 201
4590	8000.0	0.2250	307.60	MORTGAGE	105000.0	Source Verified	Ma 201
67612	18000.0	0.1447	423.23	MORTGAGE	75000.0	Not Verified	Jar 201
54328	9000.0	0.1180	298.07	MORTGAGE	159000.0	Not Verified	Jar 201
50305	6500.0	0.1614	228.98	MORTGAGE	100000.0	Source Verified	Jar 201

In [32]:
```
# Preview the data
df.sample(10)
```

Out[32]:

	loan_amnt	int_rate	installment	home_ownership	annual_inc	verification_status	issue_
39646	4200.0	0.1131	138.13	MORTGAGE	33258.0	Not Verified	Feb 201
22733	7500.0	0.0819	235.69	MORTGAGE	61000.0	Source Verified	Feb 201
10846	16000.0	0.1557	385.45	MORTGAGE	125000.0	Source Verified	Mar 201
22623	6250.0	0.1797	225.86	RENT	80000.0	Not Verified	Feb 201
64723	17625.0	0.2235	490.30	MORTGAGE	77000.0	Not Verified	Jan 201
49105	28000.0	0.0819	570.29	OWN	80000.0	Not Verified	Jan 201
44994	3300.0	0.1691	117.51	MORTGAGE	52000.0	Source Verified	Jan 201
61912	12000.0	0.0702	370.64	MORTGAGE	10000.0	Verified	Jan 201
48638	15000.0	0.1691	534.12	RENT	35000.0	Source Verified	Jan 201
68522	22000.0	0.0819	691.33	RENT	55000.0	Not Verified	Jan 201

In [30]:
```
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_
    "recoveries", "collection_recovery_fee", "last_pymnt_amnt", "next_
    "collections_12_mths_ex_med", "policy_code", "application_type", "
    "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
    "bc_util", "chargeoff_within_12_mths", "delinq_amnt", "mo_sin_old_
    "mo_sin_old_rev_tl_op", "mo_sin_rcnt_rev_tl_op", "mo_sin_rcnt_tl",
    "mths_since_recent_bc", "mths_since_recent_inq", "num_accts_ever_1
    "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_24
    "num_tl_op_past_12m", "pct_tl_nvr_dlq", "percent_bc_gt_75", "pub_r
    "tax_liens", "tot_hi_cred_lim", "total_bal_ex_mort", "total_bc_lim
    "total_il_high_credit_limit", "hardship_flag", "debt_settlement_fl
]

target = ["loan_status"]
```

In [6]:
```
df.shape
```

Out[6]: (68817, 86)

In [7]:
```
df = df.dropna(axis='columns', how='all')
```

In [8]:
```
df.shape
```

Out[8]: (68817, 86)

In [9]:
```
df = df.dropna()
```

In [10]:
```
df.shape
```

Out[10]: (68817, 86)

Split the Data into Training and Testing

In [33]:
```
# Create our features
# X_scaler wasn't working because home_ownership had 3 values - couldn
# X = df.drop(columns='loan_status')
X = pd.get_dummies(df.drop(columns='loan_status'))

# Create our target
targ = df['loan_status'].copy()
y = targ
```

In [34]:
```
X.describe()
```

Out[34]:

	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	

In [35]:
```
# Check the balance of our target values
y.value_counts()
```

Out[35]:
```
low_risk     68470
high_risk      347
Name: loan_status, dtype: int64
```

In [36]:
```
# Split the X and y into X_train, X_test, y_train, y_test
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state
```

Data Pre-Processing

Scale the training and testing data using the `StandardScaler` from `sklearn`. Remember that when scaling the data, you only scale the features data (`X_train` and `X_testing`).

In [37]:
```
# Create the StandardScaler instance
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

In [38]:
```
# Fit the Standard Scaler with the training data
# When fitting scaling functions, only train on the training dataset
X_scaler = scaler.fit(X_train)
```

In [39]:
```
# Scale the training and testing data
X_train_scaled = X_scaler.transform(X_train)
X_test_scaled = X_scaler.transform(X_test)
```

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 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. Display 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 only, 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 BalancedRandomForestClassifier
from imblearn.ensemble import BalancedRandomForestClassifier, EasyEnse
model = BalancedRandomForestClassifier(n_estimators=100, random_state=

#fit model
model.fit(X_train_scaled, y_train)

#predict
y_predict = model.predict(X_test_scaled)
```

In [46]:
```
# Calculated the balanced accuracy score
print(f'BAS: {balanced_accuracy_score(y_test,y_predict)}')
```

```
BAS: 0.7887512850910909
```

In [47]:
```
# Display the confusion matrix
confusion_matrix(y_test, y_predict)
```

Out[47]:
```
array([[   71,    30],
       [ 2146, 14958]])
```

In [48]:
```
# Print the imbalanced classification report
print(classification_report_imbalanced(y_test, y_predict))
```

```
                   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
.63     17104

avg / total       0.99      0.87      0.70      0.93      0.78      0
.63     17205
```

In [50]:
```
# List the features sorted in descending order by feature importance
# sorted(zip(rf_model.feature_importances_, X.columns), reverse=True)
sorted(zip(model.feature_importances_, X.columns), reverse=True)[:10]
```

Out[50]:
```
[(0.07876809003486353, 'total_rec_prncp'),
 (0.05883806887524815, 'total_pymnt'),
 (0.05625613759225244, 'total_pymnt_inv'),
 (0.05355513093134745, 'total_rec_int'),
 (0.0500331813446525, 'last_pymnt_amnt'),
 (0.02966959508700077, 'int_rate'),
 (0.021129125328012987, 'issue_d_Jan-2019'),
 (0.01980242888931366, 'installment'),
 (0.01747062730041245, 'dti'),
 (0.016858293184471483, 'out_prncp_inv')]
```

Easy Ensemble Classifier

In [56]:
```
# Train the Classifier
easy_model = EasyEnsembleClassifier(n_estimators=100, random_state=1)

# Fit the model
easy_model.fit(X_train_scaled, y_train)

# Make the predictions
eas_y_pred = easy_model.predict(X_test_scaled)
```

In [57]:
```
# Calculated the balanced accuracy score
print(f'Easy E BAS: {balanced_accuracy_score(y_test, eas_y_pred)}')
```

```
Easy E BAS: 0.931601605553446
```

In [58]:
```
# Display the confusion matrix
confusion_matrix(y_test, eas_y_pred)
```

Out[58]:
```
array([[   93,     8],
       [  985, 16119]])
```

In [59]:
```
# Print the imbalanced classification report
print(classification_report_imbalanced(y_test, eas_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 [ ]:
```

```

Final Questions

1. Which model had the best balanced accuracy score?

   Easy Ensemble had the best balanced accuracy score.
2. Which model had the best recall score?

   The Easy Ensemble model had a high risk recall of .92 and low risk recall score of .94, which is much better than the Random Forrest.
3. Which model had the best geometric mean score?

   Easy Enemble again was much better than the Random Forrest GM score.
4. What are the top three features?

   Total received principle, total payment, and total payment inv, which probably means total payments invoiced.

In [ ]:

```

```