Credit Risk Resampling Techniques

In [20]:

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

In [21]:

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

Read the CSV and Perform Basic Data Cleaning

In [22]:

```
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
    "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 [23]:

```
# 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'
df = df.replace(x)

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

df.head()
```

Out[23]:

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

5 rows × 86 columns

Split the Data into Training and Testing

In [24]:

```
loans_encoding = pd.get_dummies(df, columns=['home_ownership','verification_status','issue
loans_encoding.head()
```

Out[24]:

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

5 rows × 96 columns

In [25]:

```
# Create our features
X = loans_encoding.drop('loan_status', axis=1)


# Create our target
y = loans_encoding['loan_status']
```

In [26]:

```
X.describe()
```

Out[26]:

	loan_amnt	int_rate	installment	annual_inc	dti	delinq_2yrs	inq_last_6mt
count	68817.000000	68817.000000	68817.000000	6.881700e+04	68817.000000	68817.000000	68817.0000
mean	16677.594562	0.127718	480.652863	8.821371e+04	21.778153	0.217766	0.4976
std	10277.348590	0.048130	288.062432	1.155800e+05	20.199244	0.718367	0.7581
min	1000.000000	0.060000	30.890000	4.000000e+01	0.000000	0.000000	0.0000
25%	9000.000000	0.088100	265.730000	5.000000e+04	13.890000	0.000000	0.0000
50%	15000.000000	0.118000	404.560000	7.300000e+04	19.760000	0.000000	0.0000
75%	24000.000000	0.155700	648.100000	1.040000e+05	26.660000	0.000000	1.0000
max	40000.000000	0.308400	1676.230000	8.797500e+06	999.000000	18.000000	5.0000

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,stratify=y)
```

Oversampling

In this section, you will compare two oversampling algorithms to determine which algorithm results in the best performance. You will oversample the data using the naive random oversampling algorithm and the SMOTE algorithm. For each algorithm, be sure to complete the folliowing steps:

1. View the count of the target classes using `Counter` from the collections library.
2. Use the resampled data to train a logistic regression model.
3. Calculate the balanced accuracy score from sklearn.metrics.
4. Print the confusion matrix from sklearn.metrics.
5. Generate a classication report using the `imbalanced_classification_report` from imbalanced-learn.

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

Naive Random Oversampling

In [29]:

```
# Resample the training data with the RandomOversampler
from imblearn.over_sampling import RandomOverSampler

RoS = RandomOverSampler(random_state=1)

X_resampled, y_resampled = RoS.fit_resample(X_train, y_train)
```

In [30]:

```
# Train the Logistic Regression model using the resampled data
```

```python
from sklearn.linear_model import LogisticRegression

model = LogisticRegression(random_state=1)
model.fit(X_resampled, y_resampled)
```

Out[30]:
```
LogisticRegression(random_state=1)
```

In [31]:
```python
# Calculated the balanced accuracy score
from sklearn.metrics import accuracy_score

y_pred = model.predict(X_test)

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

```
0.6816623074687591
```

In [32]:
```python
# Display the confusion matrix
from sklearn.metrics import confusion_matrix, classification_report

matrix = confusion_matrix(y_test, y_pred)
cm_df = pd.DataFrame(
    matrix, index=["Actual_High_Risk", "Actua_Low_Risk"], columns=["Predicted_High_Risk",
cm_df
```

Out[32]:

	Predicted_High_Risk	Predicted_Low_Risk
Actual_High_Risk	53	34
Actua_Low_Risk	5443	11675

In [33]:
```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       0.01      0.61      0.68      0.02      0.64      0.41        87
   low_risk       1.00      0.68      0.61      0.81      0.64      0.42     17118

avg / total       0.99      0.68      0.61      0.81      0.64      0.42     17205
```

SMOTE Oversampling

In [49]:
```python
# Resample the training data with SMOTE
from imblearn.over_sampling import SMOTE

X_resampledSMOTE, y_resampleSMOTE = SMOTE(random_state=1, sampling_strategy='auto').fit_re
```

In [50]:
```python
# Train the Logistic Regression model using the resampled data
model = LogisticRegression(random_state=1)

model.fit(X_resampledSMOTE, y_resampleSMOTE)
y_pred_SMOTE = model.predict(X_test)
```

In [51]:
```python
# Calculated the balanced accuracy score
from sklearn.metrics import balanced_accuracy_score
```

```
acc_scoreSMOTE = balanced_accuracy_score(y_test, y_pred_SMOTE)
acc_scoreSMOTE
```

Out[51]: 0.6234433606890912

In [52]:

```
# Display the confusion matrix
matrix_SMOTE = confusion_matrix(y_test, y_pred_SMOTE)

cm_SMOTE_df = pd.DataFrame(
    matrix_SMOTE, index=["Actual_High_Risk", "Actual_Low_Risk"], columns=["Predicted_High_
cm_SMOTE_df
```

Out[52]:

	Predicted_High_Risk	Predicted_Low_Risk
Actual_High_Risk	53	34
Actual_Low_Risk	6202	10916

In [53]:

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

```
                   pre       rec       spe        f1       geo       iba       sup

  high_risk       0.01      0.61      0.64      0.02      0.62      0.39        87
   low_risk       1.00      0.64      0.61      0.78      0.62      0.39     17118

avg / total       0.99      0.64      0.61      0.77      0.62      0.39     17205
```

Undersampling

In this section, you will test an undersampling algorithms to determine which algorithm results in the best performance compared to the oversampling algorithms above. You will undersample the data using the Cluster Centroids algorithm and complete the folliwing steps:

1. View the count of the target classes using `Counter` from the collections library.
2. Use the resampled data to train a logistic regression model.
3. Calculate the balanced accuracy score from sklearn.metrics.
4. Print the confusion matrix from sklearn.metrics.
5. Generate a classication report using the `imbalanced_classification_report` from imbalanced-learn.

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

In [54]:

```
# Resample the data using the ClusterCentroids resampler
# Warning: This is a large dataset, and this step may take some time to complete
from imblearn.under_sampling import ClusterCentroids

cc_resampler = ClusterCentroids(random_state=1)
X_resample_cc, y_resample_cc = cc_resampler.fit_resample(X_train, y_train)
Counter(y_resample_cc)
```

Out[54]: Counter({'high_risk': 260, 'low_risk': 260})

In [55]:
```
# Train the Logistic Regression model using the resampled data

model_cc = LogisticRegression(random_state=1)
model_cc.fit(X_resample_cc, y_resample_cc)
y_pred_cc = model_cc.predict(X_test)
```

In [56]:
```
# Calculated the balanced accuracy score
acc_score_cc = balanced_accuracy_score(y_test, y_pred_cc)
acc_score_cc
```

Out[56]:
```
0.5293611080894884
```

In [57]:
```
# Display the confusion matrix
matrix_cc = confusion_matrix(y_test, y_pred_cc)

cm_cc_df = pd.DataFrame(
    matrix_cc, index=["Actual_High_Risk", "Actual_Low_Risk"], columns=["Predicted_High_Ris
cm_cc_df
```

Out[57]:

	Predicted_High_Risk	Predicted_Low_Risk
Actual_High_Risk	53	34
Actual_Low_Risk	9423	7695

In [58]:
```
# Print the imbalanced classification report
print(classification_report_imbalanced(y_test, y_pred_cc))
```

```
                   pre       rec       spe        f1       geo       iba       sup

  high_risk       0.01      0.61      0.45      0.01      0.52      0.28        87
   low_risk       1.00      0.45      0.61      0.62      0.52      0.27     17118

avg / total       0.99      0.45      0.61      0.62      0.52      0.27     17205
```

Combination (Over and Under) Sampling

In this section, you will test a combination over- and under-sampling algorithm to determine if the algorithm results in the best performance compared to the other sampling algorithms above. You will resample the data using the SMOTEENN algorithm and complete the folliowing steps:

1. View the count of the target classes using `Counter` from the collections library.
2. Use the resampled data to train a logistic regression model.
3. Calculate the balanced accuracy score from sklearn.metrics.
4. Print the confusion matrix from sklearn.metrics.
5. Generate a classication report using the `imbalanced_classification_report` from imbalanced-learn.

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

In [60]:
```
# Resample the training data with SMOTEENN
# Warning: This is a large dataset, and this step may take some time to complete
from imblearn.combine import SMOTEENN
```

```python
smote_resampler = SMOTEENN(random_state=1)
X_resample_smote, y_resample_smote = smote_resampler.fit_resample(X, y)
```

In [61]:

```python
# Train the Logistic Regression model using the resampled data
model_smoteenn = LogisticRegression(random_state=1)

model_smoteenn.fit(X_resample_smote, y_resample_smote)
y_pred_smoteenn = model_smoteenn.predict(X_test)
```

In [62]:

```python
# Calculated the balanced accuracy score
acc_score_smoteenn = balanced_accuracy_score(y_test, y_pred_smoteenn)
acc_score_smoteenn
```

Out[62]:

```
0.6531287896185101
```

In [63]:

```python
# Display the confusion matrix
matrix_smoteenn = confusion_matrix(y_test, y_pred_smoteenn)

cm_smoteenn_df = pd.DataFrame(matrix_smoteenn, index=["Actual_High_Risk", "Actual_Low_Risk
cm_smoteenn_df
```

Out[63]:

	Predicted_High_Risk	Predicted_Low_Risk
Actual_High_Risk	60	27
Actual_Low_Risk	6563	10555

In [64]:

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

```
                   pre       rec       spe        f1       geo       iba       sup

  high_risk       0.01      0.69      0.62      0.02      0.65      0.43        87
   low_risk       1.00      0.62      0.69      0.76      0.65      0.42     17118

avg / total       0.99      0.62      0.69      0.76      0.65      0.42     17205
```

In [ ]: