Credit Risk Resampling Techniques

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
```

Read the CSV into DataFrame

In [3]:
```
# Load the data
file_path = Path('Resources/lending_data.csv')
df = pd.read_csv(file_path)
df.head()
```

Out[3]:

	loan_size	interest_rate	homeowner	borrower_income	debt_to_income	num_of_accounts	c
0	10700.0	7.672	own	52800	0.431818	5	
1	8400.0	6.692	own	43600	0.311927	3	
2	9000.0	6.963	rent	46100	0.349241	3	
3	10700.0	7.664	own	52700	0.430740	5	
4	10800.0	7.698	mortgage	53000	0.433962	5	

Split the Data into Training and Testing

In [11]:
```
# Create our features
X = df.copy()
X.drop("loan_status", axis=1, inplace=True)
X = pd.get_dummies(X)

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

In [12]:
```
# X.head()
X.columns
```

Out[12]:
```
Index(['loan_size', 'interest_rate', 'borrower_income', 'debt_to_inco
me',
       'num_of_accounts', 'derogatory_marks', 'total_debt',
       'homeowner_mortgage', 'homeowner_own', 'homeowner_rent'],
      dtype='object')
```

In [13]:
```
X.describe()
```

Out[13]:

	loan_size	interest_rate	borrower_income	debt_to_income	num_of_accounts	derog
count	77536.000000	77536.000000	77536.000000	77536.000000	77536.000000	7
mean	9805.562577	7.292333	49221.949804	0.377318	3.826610	
std	2093.223153	0.889495	8371.635077	0.081519	1.904426	
min	5000.000000	5.250000	30000.000000	0.000000	0.000000	
25%	8700.000000	6.825000	44800.000000	0.330357	3.000000	
50%	9500.000000	7.172000	48100.000000	0.376299	4.000000	
75%	10400.000000	7.528000	51400.000000	0.416342	4.000000	
max	23800.000000	13.235000	105200.000000	0.714829	16.000000	

In [14]:
```
# type(y)
y.head()
```

Out[14]:

	loan_status
0	low_risk
1	low_risk
2	low_risk
3	low_risk
4	low_risk

In [15]:
```
# Check the balance of our target values
y['loan_status'].value_counts()
```

Out[15]:
```
low_risk     75036
high_risk     2500
Name: loan_status, dtype: int64
```

In [16]:
```
from sklearn.metrics import confusion_matrix, accuracy_score, classifi
from sklearn.model_selection import train_test_split
```

In [17]:
```
# Create X_train, X_test, y_train, y_test
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 [18]:
```
# Create the StandardScaler instance
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

In [19]:
```
# 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 [20]:
```
# Scale the training and testing data
X_train_scaled = X_scaler.transform(X_train)
X_test_scaled = X_scaler.transform(X_test)
```

Simple Logistic Regression

In [33]:
```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(solver='lbfgs', random_state=1)
model.fit(X_train_scaled, y_train)
```

Out[33]:
```
LogisticRegression(random_state=1)
```

In [35]:
```
# Calculated the balanced accuracy score
from sklearn.metrics import balanced_accuracy_score
y_pred = model.predict(X_test_scaled)
print(f'BAS: {balanced_accuracy_score(y_test, y_pred)}')
```

```
BAS: 0.9868772353780972
```

In [36]:
```
# Display the confusion matrix
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, y_pred)
```

Out[36]:
```
array([[  615,    13],
       [  104, 18652]])
```

In [37]:
```
# 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       0.86      0.98      0.99      0.91      0.99      0
.97       628
   low_risk       1.00      0.99      0.98      1.00      0.99      0
.98     18756

avg / total       0.99      0.99      0.98      0.99      0.99      0
.98     19384
```

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 [38]:
```
# Resample the training data with the RandomOversampler
from imblearn.over_sampling import RandomOverSampler

# View the count of target classes with Counter
oversampler = RandomOverSampler(random_state=1)
X_resampled, y_resampled = oversampler.fit_resample(X_train_scaled, y_
Counter(y_resampled)
```

Out[38]:
```
Counter({'loan_status': 1})
```

In [39]:
```
# Train the Logistic Regression model using the resampled data
# from sklearn.linear_model import LogisticRegression

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

Out[39]:
```
LogisticRegression(random_state=1)
```

In [41]:
```
# Calculated the balanced accuracy score
from sklearn.metrics import balanced_accuracy_score
y_predict = model.predict(X_test_scaled)
print(f'BAS: {balanced_accuracy_score(y_test, y_predict)}')
```

```
BAS: 0.994465804912704
```

In [42]:
```
# Display the confusion matrix
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, y_predict)
```

Out[42]:
```
array([[  625,     3],
       [  118, 18638]])
```

In [43]:
```
# Print the imbalanced classification report
from imblearn.metrics import classification_report_imbalanced

print(classification_report_imbalanced(y_test, y_predict))
```

```
                   pre       rec       spe        f1       geo       
iba       sup

  high_risk       0.84      1.00      0.99      0.91      0.99      0
.99       628
   low_risk       1.00      0.99      1.00      1.00      0.99      0
.99     18756

avg / total       0.99      0.99      1.00      0.99      0.99      0
.99     19384
```

SMOTE Oversampling

In [45]:
```
# Resample the training data with SMOTE
from imblearn.over_sampling import SMOTE

SMOTE_X_resampled, SMOTE_y_resampled = SMOTE(random_state=1).fit_resam
    X_train_scaled, y_train)

# View the count of target classes with Counter
Counter(SMOTE_y_resampled.loan_status)
```

Out[45]:
```
Counter({'low_risk': 56280, 'high_risk': 56280})
```

In [46]:
```
# Train the Logistic Regression model using the resampled data
model = LogisticRegression(solver='lbfgs', random_state=1)
model.fit(SMOTE_X_resampled, SMOTE_y_resampled)
```

Out[46]:
```
LogisticRegression(random_state=1)
```

In [48]:
```
# Calculated the balanced accuracy score
SMOTE_y_pred = model.predict(X_test_scaled)
print(f'BAS: {balanced_accuracy_score(y_test, SMOTE_y_pred)}')
```

```
BAS: 0.994492463048767
```

In [49]:
```
# Display the confusion matrix
confusion_matrix(y_test, SMOTE_y_pred)
```

Out[49]:
```
array([[  625,     3],
       [  117, 18639]])
```

In [50]:
```
# Print the imbalanced classification report
print(classification_report_imbalanced(y_test, SMOTE_y_pred))
```

```
                   pre       rec       spe        f1       geo       
iba       sup

  high_risk       0.84      1.00      0.99      0.91      0.99      0
.99       628
   low_risk       1.00      0.99      1.00      1.00      0.99      0
.99     18756

avg / total       0.99      0.99      1.00      0.99      0.99      0
.99     19384
```

Undersampling

In this section, you will test an undersampling algorithm 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 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. Display 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 [52]:
```
# Resample the data using the ClusterCentroids resampler
from imblearn.under_sampling import ClusterCentroids
X_train_ccresampled, y_train_ccresampled = ClusterCentroids(random_sta

# View the count of target classes with Counter
Counter(y_train_ccresampled.loan_status)
```

Out[52]:
```
Counter({'high_risk': 1872, 'low_risk': 1872})
```

In [53]:
```
# Train the Logistic Regression model using the resampled data
model = LogisticRegression(solver='lbfgs', random_state=1)
model.fit(X_train_ccresampled, y_train_ccresampled)
Uy_pred = model.predict(X_test_scaled)
```

In [54]:
```
# Calculate the balanced accuracy score
print(f'BAS: {balanced_accuracy_score(y_test, Uy_pred)}')
```

```
BAS: 0.9932200039936265
```

In [55]:
```
# Display the confusion matrix
confusion_matrix(y_test, Uy_pred)
```

Out[55]:
```
array([[  623,     5],
       [  105, 18651]])
```

In [56]:
```
# Print the imbalanced classification report
print(classification_report_imbalanced(y_test, Uy_pred))
```

```
                   pre       rec       spe        f1       geo       
iba       sup

  high_risk       0.86      0.99      0.99      0.92      0.99      0
.99       628
   low_risk       1.00      0.99      0.99      1.00      0.99      0
.99     18756

avg / total       1.00      0.99      0.99      0.99      0.99      0
.99     19384
```

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. Display 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 [57]:
```
# Resample the training data with SMOTEENN
from imblearn.combine import SMOTEENN
sm = SMOTEENN(random_state=1)
SNNX_resampled, SNNy_resampled = sm.fit_resample(X_train_scaled, y_tra

# View the count of target classes with Counter
Counter(SNNy_resampled.loan_status)
```

Out[57]:
```
Counter({'high_risk': 55690, 'low_risk': 55965})
```

In [58]:
```
# Train the Logistic Regression model using the resampled data
model = LogisticRegression(solver='lbfgs', random_state=1)
model.fit(SNNX_resampled, SNNy_resampled)
```

Out[58]:
```
LogisticRegression(random_state=1)
```

In [60]:
```
# Calculate the balanced accuracy score
SNNy_pred = model.predict(X_test_scaled)
balanced_accuracy_score(y_test, SNNy_pred)
```

Out[60]:
```
0.994492463048767
```

In [61]:
```
# Display the confusion matrix

confusion_matrix(y_test, SNNy_pred)
```

Out[61]:
```
array([[  625,     3],
       [  117, 18639]])
```

In [62]:
```
# Print the imbalanced classification report
print(classification_report_imbalanced(y_test, SNNy_pred))
```

```
                   pre       rec       spe        f1       geo
iba       sup

  high_risk       0.84      1.00      0.99      0.91      0.99      0
.99       628
   low_risk       1.00      0.99      1.00      1.00      0.99      0
.99     18756

avg / total       0.99      0.99      1.00      0.99      0.99      0
.99     19384
```

Final Questions

```
1. Which model had the best balanced accuracy score?

   The combination of over and undersampling had the highest score
but they were extremely close.

2. Which model had the best recall score?

    The naive random, SMOTE, and combo all had a recall score of
1.00.

3. Which model had the best geometric mean score?

    For all methods, geo score was .99 for high and low risk.
```

In [ ]: