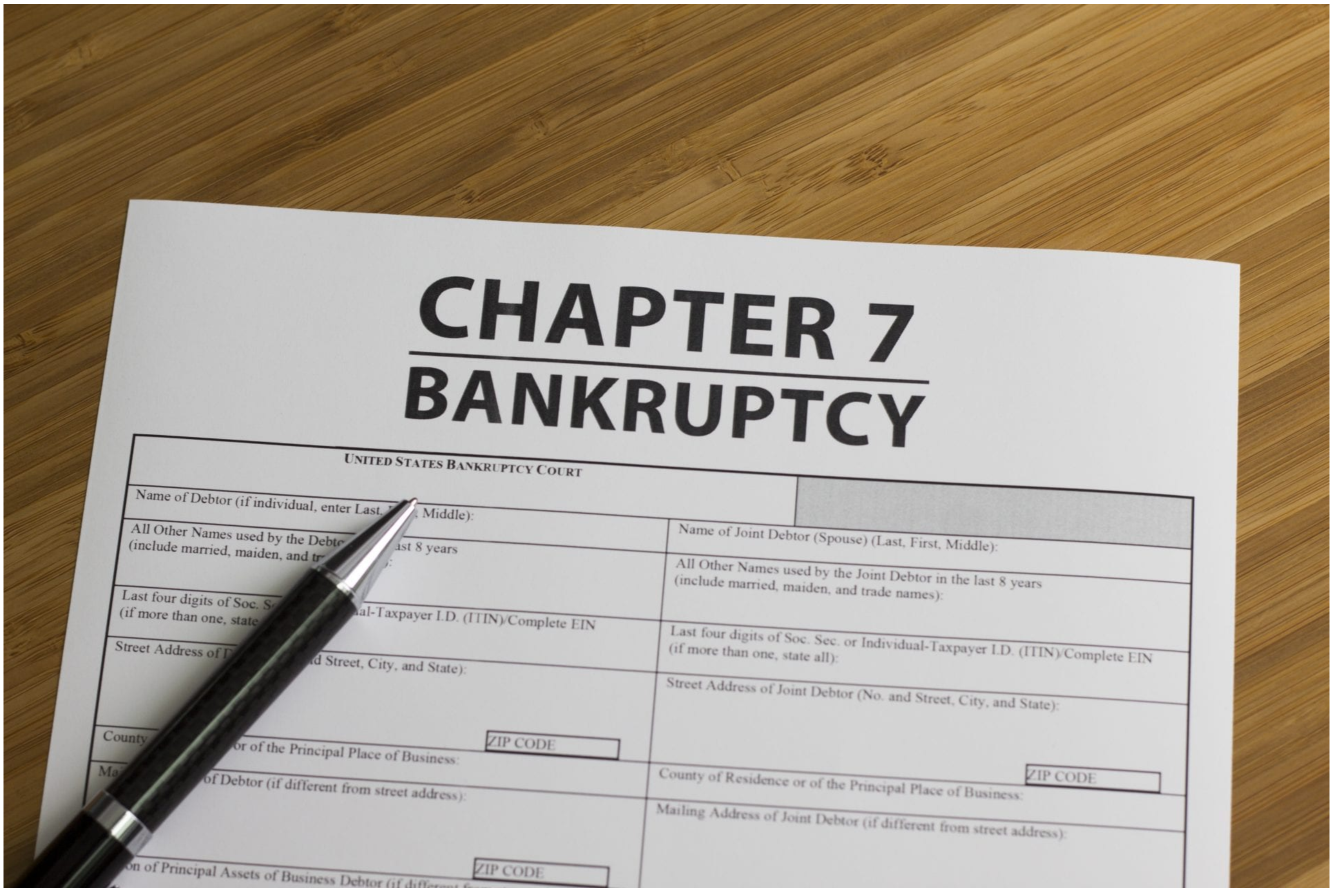
Company bankruptcy prediction based on financial metrics

Author: Andrei Mikhailov
email: am@mikhailovandrei.com
Flex Program:
Instructor name: Abhineet Kulkarni



Overview

During this project, we will be building models for prediction companies bunkruptcy based on given set of financial features. For the sake of research, we will use the following dataset:

https://www.kaggle.com/code/kaixiongf/company-bankruptcy-w-pycaret (https://www.kaggle.com/code/kaixiongf/company-bankruptcy-w-pycaret)

The data collected from the Taiwan Economic Journal for the years 1999 to 2009 and to see what insight can be generated during data exploration and which model is able to predict company bankruptcy most accurately.

We will employ:

- Logistic Regression,
- KNN,
- Decision Tree,
- SMOTE and
- Random Forest.

Business Problem

The customer, Restructuring Investment Bank, wants to increase precision of their forecast for their customers, debtors (the distressed companies) and creditors (banks, lenders) when capital structure issues arise, which primarily stem from over-leveraged companies with insufficient liquidity to meet their obligations.

We will build different models based on the given dataset of portfolio companies.

Data Understanding

During the following steps, we will load and expore the datasets.

In [1]:
```
# Import libraries
import pandas as pd
from sklearn.model_selection import train_test_split
import numpy as np
from sklearn.metrics import precision_score, recall_score, accuracy_score, f1_score
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
from sklearn.metrics import plot_confusion_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import precision_score, recall_score, accuracy_score, f1_score, classification_report
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.feature_selection import (VarianceThreshold, SelectKBest, f_regression, mutual_info_regression,
    RFE, RFECV)
from sklearn.linear_model import LinearRegression, LassoCV, Lasso
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LogisticRegression
from imblearn.over_sampling import RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import MinMaxScaler
```

```
/Users/andreim/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/xgboost/compat.py:93: FutureWarning: pandas.
Int64Index is deprecated and will be removed from pandas in a future version. Use pandas.Index with the appropriate
dtype instead.
  from pandas import MultiIndex, Int64Index
```

In [2]:
```
# Import dataset
data = pd.read_csv('/Users/andreim/Desktop/Mastering/Flatiron/Phase_III/Final_Project/Bankruptcy/data.csv')
```

In [3]:
```
data.head()
```

Out[3]:

	Bankrupt?	ROA(C) before interest and depreciation before interest	ROA(A) before interest and % after tax	ROA(B) before interest and depreciation after tax	Operating Gross Margin	Realized Sales Gross Margin	Operating Profit Rate	Pre-tax net Interest Rate	After-tax net Interest Rate	Non-industry income and expenditure/revenue	...	Net Income to Total Assets	Total assets to GNP price	No-credit Interval
0	1	0.370594	0.424389	0.405750	0.601457	0.601457	0.998969	0.796887	0.808809	0.302646	...	0.716845	0.009219	0.622879
1	1	0.464291	0.538214	0.516730	0.610235	0.610235	0.998946	0.797380	0.809301	0.303556	...	0.795297	0.008323	0.623652
2	1	0.426071	0.499019	0.472295	0.601450	0.601364	0.998857	0.796403	0.808388	0.302035	...	0.774670	0.040003	0.623841
3	1	0.399844	0.451265	0.457733	0.583541	0.583541	0.998700	0.796967	0.808966	0.303350	...	0.739555	0.003252	0.622929
4	1	0.465022	0.538432	0.522298	0.598783	0.598783	0.998973	0.797366	0.809304	0.303475	...	0.795016	0.003878	0.623521

5 rows × 96 columns

In [4]:
```
data['Bankrupt?'].value_counts()
```

Out[4]:
```
0    6599
1     220
Name: Bankrupt?, dtype: int64
```

As we can see we are dealing with severe class imbalance.

In [5]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6819 entries, 0 to 6818
Data columns (total 96 columns):
 #   Column                                                    Non-Null Count  Dtype  
---  ------                                                    --------------  -----  
 0   Bankrupt?                                                 6819 non-null   int64  
 1    ROA(C) before interest and depreciation before interest  6819 non-null   float64
 2    ROA(A) before interest and % after tax                   6819 non-null   float64
 3    ROA(B) before interest and depreciation after tax        6819 non-null   float64
 4    Operating Gross Margin                                   6819 non-null   float64
 5    Realized Sales Gross Margin                              6819 non-null   float64
 6    Operating Profit Rate                                    6819 non-null   float64
 7    Pre-tax net Interest Rate                                6819 non-null   float64
 8    After-tax net Interest Rate                              6819 non-null   float64
 9    Non-industry income and expenditure/revenue              6819 non-null   float64
 10   Continuous interest rate (after tax)                     6819 non-null   float64
 11   Operating Expense Rate                                   6819 non-null   float64
 12   Research and development expense rate                    6819 non-null   float64
 13   Cash flow rate                                           6819 non-null   float64
 14   Interest-bearing debt interest rate                      6819 non-null   float64
 15   Tax rate (A)                                             6819 non-null   float64
 16   Net Value Per Share (B)                                  6819 non-null   float64
 17   Net Value Per Share (A)                                  6819 non-null   float64
 18   Net Value Per Share (C)                                  6819 non-null   float64
 19   Persistent EPS in the Last Four Seasons                  6819 non-null   float64
 20   Cash Flow Per Share                                      6819 non-null   float64
 21   Revenue Per Share (Yuan ¥)                               6819 non-null   float64
 22   Operating Profit Per Share (Yuan ¥)                      6819 non-null   float64
 23   Per Share Net profit before tax (Yuan ¥)                 6819 non-null   float64
 24   Realized Sales Gross Profit Growth Rate                  6819 non-null   float64
 25   Operating Profit Growth Rate                             6819 non-null   float64
 26   After-tax Net Profit Growth Rate                         6819 non-null   float64
 27   Regular Net Profit Growth Rate                           6819 non-null   float64
 28   Continuous Net Profit Growth Rate                        6819 non-null   float64
 29   Total Asset Growth Rate                                  6819 non-null   float64
 30   Net Value Growth Rate                                    6819 non-null   float64
 31   Total Asset Return Growth Rate Ratio                     6819 non-null   float64
 32   Cash Reinvestment %                                      6819 non-null   float64
 33   Current Ratio                                            6819 non-null   float64
 34   Quick Ratio                                              6819 non-null   float64
 35   Interest Expense Ratio                                   6819 non-null   float64
 36   Total debt/Total net worth                               6819 non-null   float64
 37   Debt ratio %                                             6819 non-null   float64
 38   Net worth/Assets                                         6819 non-null   float64
 39   Long-term fund suitability ratio (A)                     6819 non-null   float64
 40   Borrowing dependency                                     6819 non-null   float64
 41   Contingent liabilities/Net worth                         6819 non-null   float64
 42   Operating profit/Paid-in capital                         6819 non-null   float64
 43   Net profit before tax/Paid-in capital                    6819 non-null   float64
 44   Inventory and accounts receivable/Net value              6819 non-null   float64
 45   Total Asset Turnover                                     6819 non-null   float64
 46   Accounts Receivable Turnover                             6819 non-null   float64
 47   Average Collection Days                                  6819 non-null   float64
 48   Inventory Turnover Rate (times)                          6819 non-null   float64
 49   Fixed Assets Turnover Frequency                          6819 non-null   float64
 50   Net Worth Turnover Rate (times)                          6819 non-null   float64
 51   Revenue per person                                       6819 non-null   float64
 52   Operating profit per person                              6819 non-null   float64
 53   Allocation rate per person                               6819 non-null   float64
 54   Working Capital to Total Assets                          6819 non-null   float64
 55   Quick Assets/Total Assets                                6819 non-null   float64
 56   Current Assets/Total Assets                              6819 non-null   float64
 57   Cash/Total Assets                                        6819 non-null   float64
 58   Quick Assets/Current Liability                           6819 non-null   float64
 59   Cash/Current Liability                                   6819 non-null   float64
 60   Current Liability to Assets                              6819 non-null   float64
 61   Operating Funds to Liability                             6819 non-null   float64
 62   Inventory/Working Capital                                6819 non-null   float64
 63   Inventory/Current Liability                              6819 non-null   float64
 64   Current Liabilities/Liability                            6819 non-null   float64
 65   Working Capital/Equity                                   6819 non-null   float64
 66   Current Liabilities/Equity                               6819 non-null   float64
 67   Long-term Liability to Current Assets                    6819 non-null   float64
 68   Retained Earnings to Total Assets                        6819 non-null   float64
 69   Total income/Total expense                               6819 non-null   float64
 70   Total expense/Assets                                     6819 non-null   float64
 71   Current Asset Turnover Rate                              6819 non-null   float64
 72   Quick Asset Turnover Rate                                6819 non-null   float64
 73   Working capitcal Turnover Rate                           6819 non-null   float64
 74   Cash Turnover Rate                                       6819 non-null   float64
 75   Cash Flow to Sales                                       6819 non-null   float64
 76   Fixed Assets to Assets                                   6819 non-null   float64
 77   Current Liability to Liability                           6819 non-null   float64
 78   Current Liability to Equity                              6819 non-null   float64
 79   Equity to Long-term Liability                            6819 non-null   float64
```

```
 80   Cash Flow to Total Assets                                6819 non-null   float64
 81   Cash Flow to Liability                                   6819 non-null   float64
 82   CFO to Assets                                            6819 non-null   float64
 83   Cash Flow to Equity                                      6819 non-null   float64
 84   Current Liability to Current Assets                      6819 non-null   float64
 85   Liability-Assets Flag                                    6819 non-null   int64
 86   Net Income to Total Assets                               6819 non-null   float64
 87   Total assets to GNP price                                6819 non-null   float64
 88   No-credit Interval                                       6819 non-null   float64
 89   Gross Profit to Sales                                    6819 non-null   float64
 90   Net Income to Stockholder's Equity                       6819 non-null   float64
 91   Liability to Equity                                      6819 non-null   float64
 92   Degree of Financial Leverage (DFL)                       6819 non-null   float64
 93   Interest Coverage Ratio (Interest expense to EBIT)       6819 non-null   float64
 94   Net Income Flag                                          6819 non-null   int64
 95   Equity to Liability                                      6819 non-null   float64
dtypes: float64(93), int64(3)
memory usage: 5.0 MB
```

All features in our dataset are represented as int or float. There are no missing values.

Data preparation

In [6]:
```
# As we are interested if the company is about to be bankrupt, our target feature is "Bankruptcy?".
# Therefor we drop it
X = data.drop('Bankrupt?', axis=1)
y = data['Bankrupt?']

# Apply train-test-split method
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=2001)
```

As we see most of our data is represented in values between 0 and 1, therefore we use MinMaxScaler method

In [7]:
```
# Scale the data
scaler = MinMaxScaler()
# Fit the scaled data
scaler.fit(X_train)
X_train_scaled = pd.DataFrame(scaler.transform(X_train), index=X_train.index, columns=X_train.columns)
X_train_scaled
```

Out[7]:

	ROA(C) before interest and depreciation before interest	ROA(A) before interest and % after tax	ROA(B) before interest and depreciation after tax	Operating Gross Margin	Realized Sales Gross Margin	Operating Profit Rate	Pre-tax net Interest Rate	After-tax net Interest Rate	Non-industry income and expenditure/revenue	Continuous interest rate (after tax)	...	Net Income to Total Assets	Total assets to GNP price	c Int
4238	0.523659	0.556019	0.556507	0.621348	0.621420	0.999148	0.797514	0.809397	0.303368	0.781656	...	0.759368	1.271294e-13	0.62
2937	0.467359	0.493440	0.497832	0.597933	0.597933	0.998898	0.797251	0.809181	0.303432	0.781444	...	0.705667	2.122690e-13	0.62
3538	0.531989	0.519130	0.557150	0.597645	0.597645	0.998948	0.797357	0.809285	0.303512	0.781545	...	0.730942	2.508159e-14	0.62
3879	0.544081	0.559971	0.571765	0.649072	0.649072	0.999351	0.797664	0.809487	0.303204	0.781931	...	0.758793	1.571543e-13	0.62
551	0.471925	0.514080	0.506130	0.593645	0.593645	0.998931	0.797323	0.809253	0.303488	0.781500	...	0.722608	3.892992e-13	0.61
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
1637	0.490642	0.538453	0.525778	0.597018	0.597018	0.998979	0.797383	0.809309	0.303492	0.781567	...	0.738295	9.088226e-14	0.62
5094	0.503688	0.548389	0.540554	0.602149	0.602149	0.998850	0.798024	0.809882	0.304882	0.782170	...	0.752669	1.022029e-11	0.63
6160	0.454714	0.464401	0.490069	0.592852	0.592852	0.998678	0.796751	0.808756	0.303019	0.781165	...	0.682516	1.472602e-14	0.62
2438	0.514828	0.656255	0.546389	0.603208	0.603208	0.998966	0.797745	0.809651	0.304152	0.781576	...	0.828320	3.112829e-14	0.62
1540	0.478850	0.507219	0.514321	0.592607	0.593047	0.998862	0.797239	0.809196	0.303486	0.781442	...	0.720014	4.724864e-13	0.62

5114 rows × 95 columns

Building models

Logistic Regression

In [8]:
```
# Building logistic regression
logreg = LogisticRegression(fit_intercept=False, C=1e12, solver='liblinear')
model_log = logreg.fit(X_train_scaled, y_train)
model_log
```

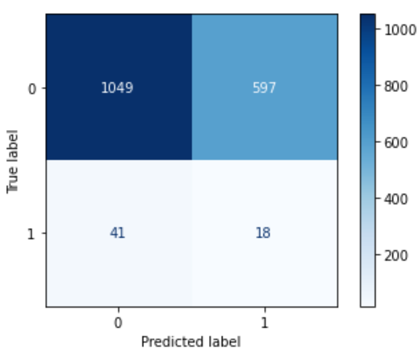
Out[8]:
```
LogisticRegression(C=1000000000000.0, fit_intercept=False, solver='liblinear')
```

In [9]:
```
# Predicting values
y_hat_train = logreg.predict(X_train)
y_hat_test = logreg.predict(X_test)
```

In [10]:
```
print('Training Precision: ', precision_score(y_train, y_hat_train, average=None))
print('Testing Precision: ', precision_score(y_test, y_hat_test, average=None))
print('\n\n')

print('Training Recall: ', recall_score(y_train, y_hat_train, average=None))
print('Testing Recall: ', recall_score(y_test, y_hat_test, average=None))
print('\n\n')

print('Training Accuracy: ', accuracy_score(y_train, y_hat_train))
print('Testing Accuracy: ', accuracy_score(y_test, y_hat_test))
print('\n\n')

print('Training F1-Score: ', f1_score(y_train, y_hat_train, average=None))
print('Testing F1-Score: ', f1_score(y_test, y_hat_test, average=None))
```

```
Training Precision:  [0.97598522 0.04448017]
Testing Precision:  [0.96238532 0.02926829]



Training Recall:  [0.64001615 0.51552795]
Testing Recall:  [0.63730255 0.30508475]



Training Accuracy:  0.6360969886585843
Testing Accuracy:  0.6258064516129033



Training F1-Score:  [0.77307645 0.08189443]
Testing F1-Score:  [0.76681287 0.05341246]
```

In [11]:
```
# Plot the confusion matrix
plot_confusion_matrix(logreg, X_test, y_test,
                      cmap=plt.cm.Blues)
plt.show()
```



As we can see, due to the significant class imbalance, the model performs very poor on prediction of companies that are true bunkrupts.

KNN

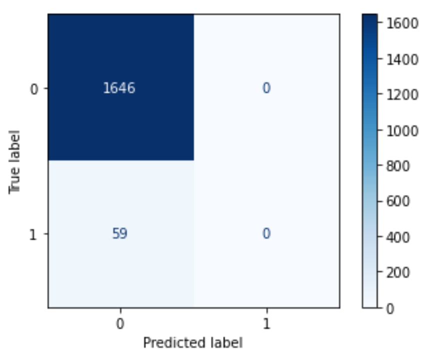
In [12]:
```
# Instantiate StandardScaler
scaler = StandardScaler()

# Transform the training and test sets
scaled_data_train = scaler.fit_transform(X_train)
scaled_data_test = scaler.transform(X_test)
```

In [13]:
```
# Import KNeighborsClassifier
from sklearn.neighbors import KNeighborsClassifier

# Instantiate KNeighborsClassifier
clf = KNeighborsClassifier()

# Fit the classifier
clf.fit(scaled_data_train, y_train)

# Predict on the test set
test_preds = clf.predict(scaled_data_test)
```

In [14]:
```
# Complete the function
def print_metrics(labels, preds):
    print("Precision Score: {}".format(precision_score(labels, preds, average=None)))
    print("Recall Score: {}".format(recall_score(labels, preds, average=None)))
    print("Accuracy Score: {}".format(accuracy_score(labels, preds)))
    print("F1 Score: {}".format(f1_score(labels, preds,average=None)))

print_metrics(y_test, test_preds)
```

```
Precision Score: [0.97156398 0.64705882]
Recall Score: [0.9963548  0.18644068]
Accuracy Score: 0.9683284457478006
F1 Score: [0.98380324 0.28947368]
```

In [15]:
```
# Plot the confusion matrix
plot_confusion_matrix(clf, X_test, y_test,
                      cmap=plt.cm.Blues)
plt.show()
```



The KNN model performs even worse, givin actually zero prediction of the true values for the bankrupt companies, even showing much less false positives

Decision tree

In [16]:
```
# Instantiate and fit a DecisionTreeClassifier
tree_clf = DecisionTreeClassifier(criterion='gini', max_depth=5)
tree_clf.fit(X_train, y_train)
```

Out[16]:
```
DecisionTreeClassifier(max_depth=5)
```

In [17]:
```
# Test set predictions
pred = tree_clf.predict(X_test)

# Confusion matrix and classification report
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))
```

```
[[1621   25]
 [  44   15]]
              precision    recall  f1-score   support

           0       0.97      0.98      0.98      1646
           1       0.38      0.25      0.30        59

    accuracy                           0.96      1705
   macro avg       0.67      0.62      0.64      1705
weighted avg       0.95      0.96      0.96      1705
```

Decision tree model performance is close to the Logistic regression but still giving us low recall and F-1 scores.

Random forest

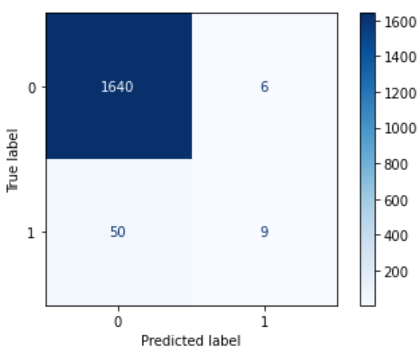
In [18]:
```
#Random forest

forest = RandomForestClassifier(n_estimators=100, max_depth= 5)
forest.fit(X_train, y_train)
```

Out[18]:
```
RandomForestClassifier(max_depth=5)
```

In [19]:
```
print("Train accuracy Score:", forest.score(X_train, y_train))
print("Test accuracy Score:", forest.score(X_test, y_test))
```

```
Train accuracy Score: 0.9792725850606179
Test accuracy Score: 0.9671554252199414
```

In [20]:
```
plot_confusion_matrix(forest, X_test, y_test,
                      cmap=plt.cm.Blues)
plt.show()
```



In [21]:
```
rfc_pred = forest.predict(X_test)

f1_score(y_test, rfc_pred)
```

Out[21]:
```
0.24324324324324323
```

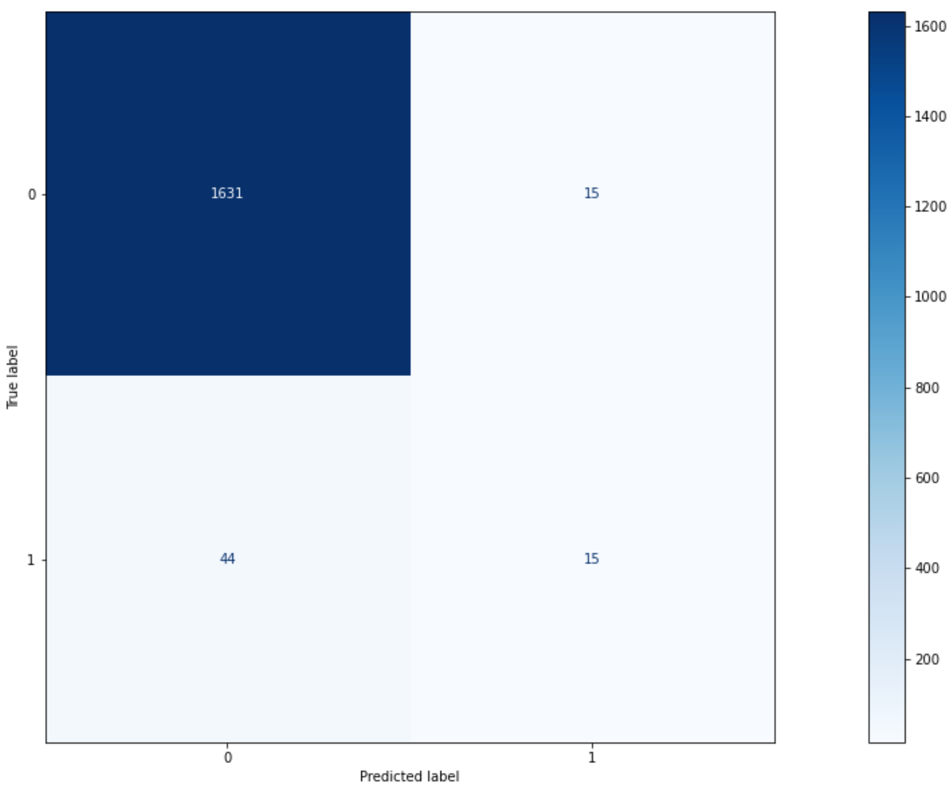
Random Forest doesnt show any significant improvements.

XGBooster

In [22]:
```
# Instantiate XGBClassifier
clf = XGBClassifier()

# Fit XGBClassifier
clf.fit(X_train, y_train)

# Predict on training and test sets
training_preds = clf.predict(X_train)
test_preds = clf.predict(X_test)

# Accuracy of training and test sets
training_accuracy = accuracy_score(y_train, training_preds)
test_accuracy = accuracy_score(y_test, test_preds)

print('Training Accuracy: {:.4}%'.format(training_accuracy * 100))
print('Validation accuracy: {:.4}%'.format(test_accuracy * 100))
```

```
/Users/andreim/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/xgboost/data.py:173: FutureWarning: pandas.I
nt64Index is deprecated and will be removed from pandas in a future version. Use pandas.Index with the appropriate d
type instead.
  from pandas import MultiIndex, Int64Index

Training Accuracy: 100.0%
Validation accuracy: 96.54%
```

In [23]:
```
fig, ax = plt.subplots(figsize=(25, 10))

plot_confusion_matrix(clf, X_test, y_test,
                     cmap=plt.cm.Blues, ax=ax)
plt.show()
```



XGBooster model performs the best compare to all previous models, even still very poor predicts true bankrupts.

Dealing with data imbalance

In this section we employ Undersampler and Oversampler methods to reduce data imbalance effect. Than we will use Decision Tree to build a model on both undersampled and oversampled datasets.

In [27]:
```
# Use under_sampler
under_sampler = RandomUnderSampler(random_state=42)
X_train_under, y_train_under = under_sampler.fit_resample(X_train,y_train)
# Print shape of the dataset
print(X_train_under.shape)
```

```
(322, 95)
```

In [28]:
```
# Use over_sampler
over_sampler = RandomOverSampler(random_state=42)
X_train_over, y_train_over = over_sampler.fit_resample(X_train,y_train)
print(X_train_over.shape)
```

```
(9906, 95)
```

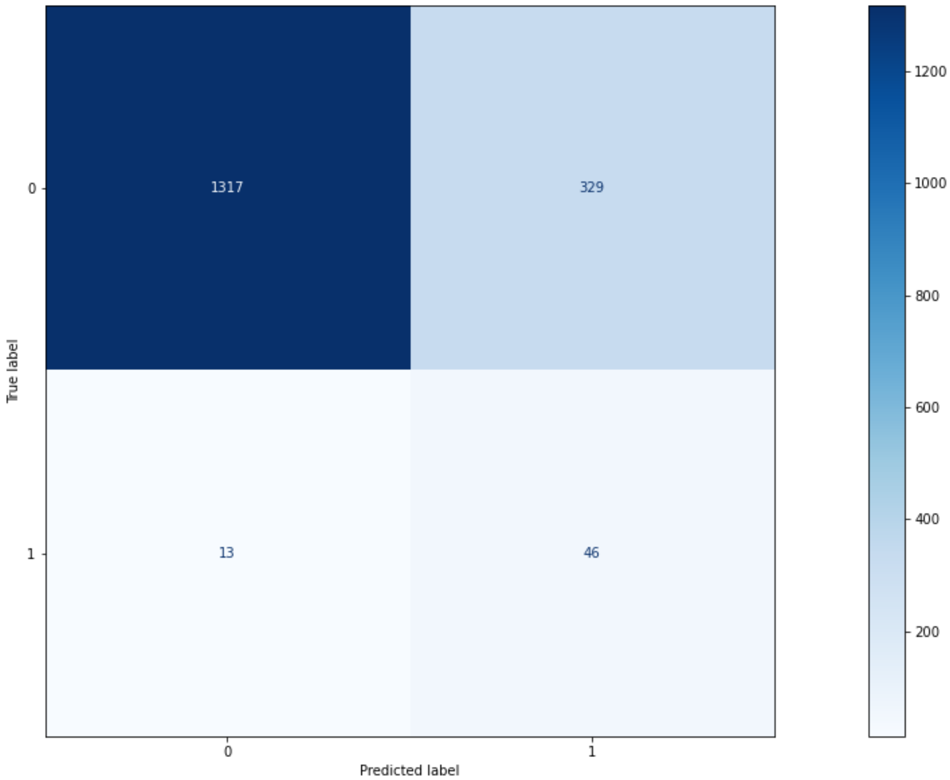
In [29]:
```
# Use DecisionTreeClassifier
model_reg = DecisionTreeClassifier(random_state=42)

model_reg.fit(X_train, y_train)

model_under = DecisionTreeClassifier(random_state=42)

model_under.fit(X_train_under, y_train_under)

model_over =  DecisionTreeClassifier(random_state=42)

model_over.fit(X_train_over, y_train_over)
```

Out[29]:
```
DecisionTreeClassifier(random_state=42)
```

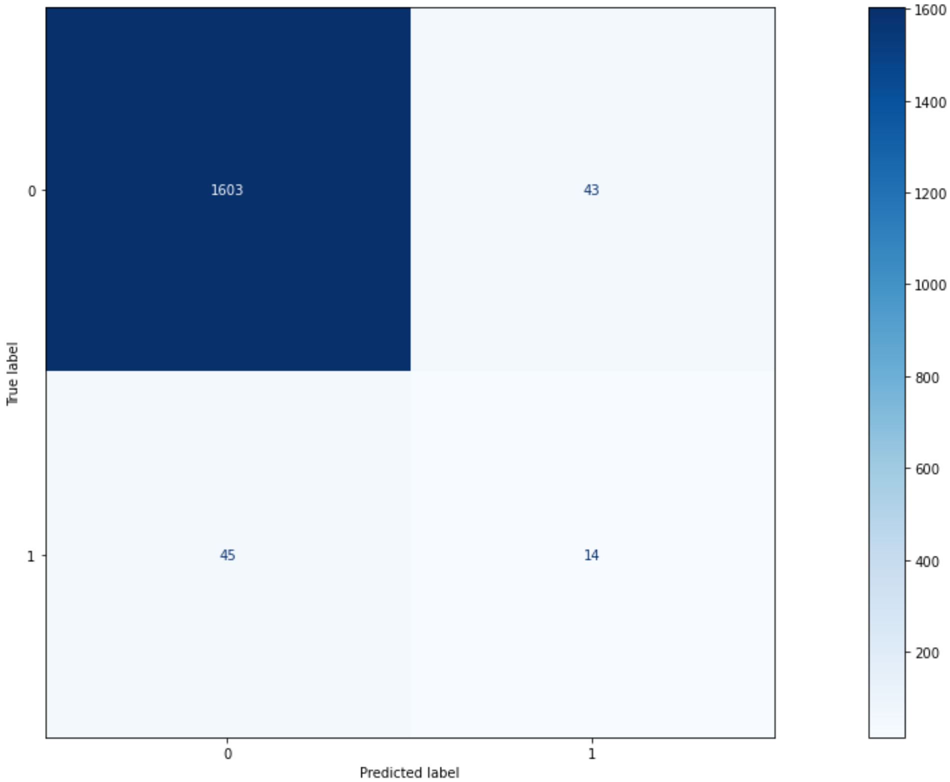
In [30]:
```
for m in [model_reg, model_under, model_over]:
    acc_train = m.score(X_train,y_train)
    acc_test = m.score(X_test,y_test)

    print("Training Accuracy:", round(acc_train, 4))
    print("Test Accuracy:", round(acc_test, 4))
```

```
Training Accuracy: 1.0
Test Accuracy: 0.946
Training Accuracy: 0.8252
Test Accuracy: 0.7994
Training Accuracy: 1.0
Test Accuracy: 0.9484
```

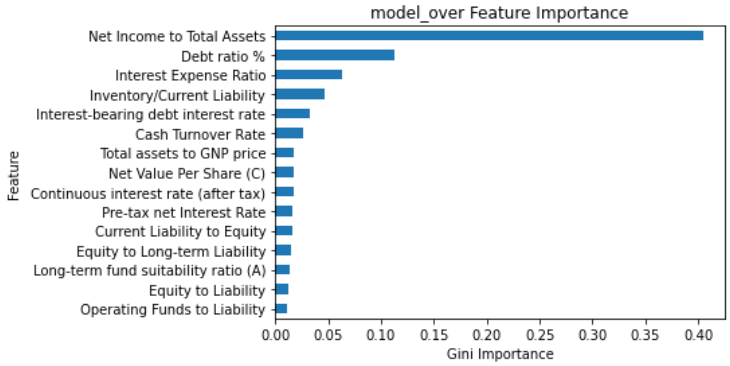
In [31]:
```
# Plot confusion matrix for undersampled data
fig, ax = plt.subplots(figsize=(25, 10))

plot_confusion_matrix(model_under, X_test, y_test,
                      cmap=plt.cm.Blues, ax=ax)
plt.show()
```



In [32]:
```
# Plot confusion matrix for oversampled data

fig, ax = plt.subplots(figsize=(25, 10))

plot_confusion_matrix(model_over, X_test, y_test,
                      cmap=plt.cm.Blues, ax=ax)
plt.show()
```



In [33]:
```
# Discover feature importance
importances = model_over.feature_importances_

feat_imp = pd.Series(importances, index=X_train_over.columns).sort_values()

# Plot series
feat_imp.tail(15).plot(kind="barh")
plt.xlabel("Gini Importance")
plt.ylabel("Feature")
plt.title("model_over Feature Importance");
```



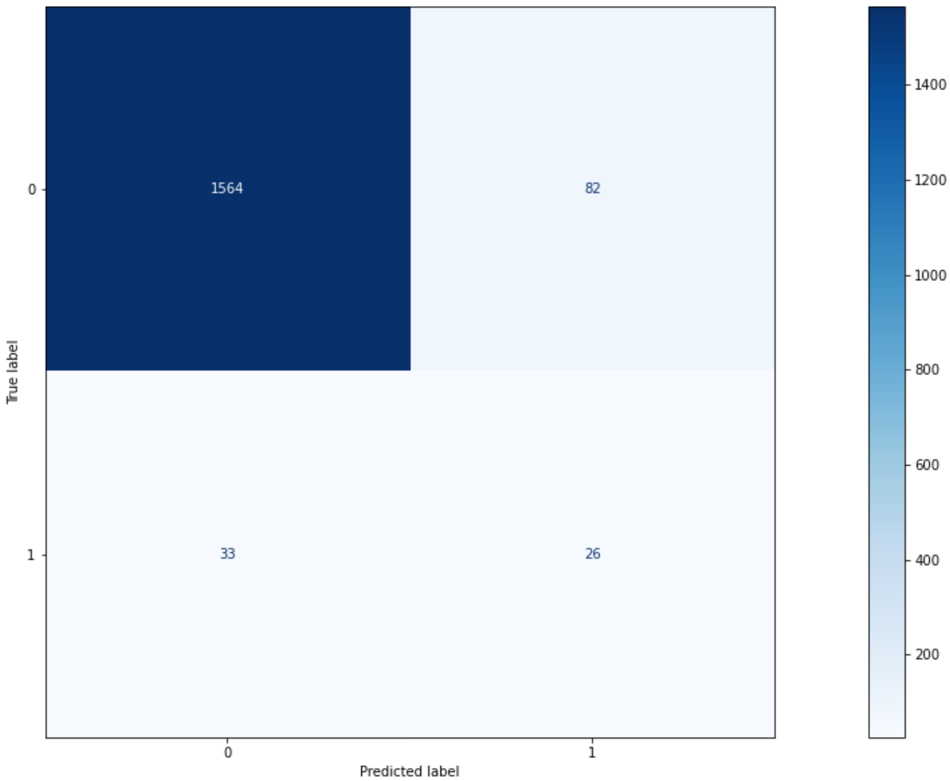
SMOTE

In [34]:
```
# Split data using SMOTE
X_train_sm, y_train_sm = SMOTE(random_state=1).fit_resample(X_train, y_train)
```

In [35]:
```
from sklearn import metrics
# Build a model
model = DecisionTreeClassifier(random_state=1)
model.fit(X_train_sm, y_train_sm)
pred = model.predict(X_test)
cm = confusion_matrix(y_test, pred)
print("Train set Accuracy: ", metrics.accuracy_score(y_train_sm, model.predict(X_train_sm)))
print("Test set Accuracy: ", metrics.accuracy_score(y_test, pred))
```

```
Train set Accuracy:  1.0
Test set Accuracy:  0.9325513196480938
```

In [36]:
```
# Plot confusion matrix
fig, ax = plt.subplots(figsize=(25, 10))

plot_confusion_matrix(model, X_test, y_test,
                      cmap=plt.cm.Blues, ax=ax)
plt.show()
```



In [37]:
```
pred = model.predict(X_test)
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           0       0.98      0.95      0.96      1646
           1       0.24      0.44      0.31        59

    accuracy                           0.93      1705
   macro avg       0.61      0.70      0.64      1705
weighted avg       0.95      0.93      0.94      1705
```

Conlusion

Due to the very strong effect of the data imbalance it's hard to build a reliable model for companies bankruptcy using giving dataset.