Big Data for Financial Computation

In [1]:

```
# importing libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.utils import resample
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report, log_loss, precision_recall_curve, accu
from sklearn.metrics import confusion_matrix
from imblearn.over_sampling import RandomOverSampler

import tensorflow
import keras
from keras.layers import Dense
from keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import to_categorical
```

In [2]:

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

Loading the Dataset

In [3]:

```
# loading dataset
data = pd.read_csv('financial_data.csv')
```

In [4]:

```
# checking dataset shape
data.shape
```

Out[4]:

```
(1700, 28)
```

Analyzing the Dataset

In [5]:

```
data.head(5)
```

Out[5]:

	Sales/Revenues	Gross Margin	EBITDA	EBITDA Margin	Net Income Before Extras	Total Debt	Net Debt	LT Debt	:
0	-0.005496	0.030763	0.018885	0.024515	0.146849	-0.029710	-0.019296	-0.042648	0
1	-0.005496	0.030763	0.088716	0.094733	0.146849	-0.029710	-0.019296	-0.042648	0
2	-0.007045	0.023159	0.088716	0.096440	0.108590	0.039410	0.034268	0.009059	0
3	-0.009396	0.028400	0.088716	0.099046	0.146137	0.030071	0.036938	-0.016964	0
4	-0.009009	0.027714	0.088716	0.098611	0.123500	0.024224	0.034445	-0.034132	0

5 rows × 28 columns

In [6]:

```
data.columns
```

Out[6]:

```
Index(['Sales/Revenues', 'Gross Margin', 'EBITDA', 'EBITDA Margin',
       'Net Income Before Extras', 'Total Debt', 'Net Debt', 'LT Debt',
       'ST Debt', 'Cash', 'Free Cash Flow', 'Total Debt/EBITDA',
       'Net Debt/EBITDA', 'Total MV', 'Total Debt/MV', 'Net Debt/MV',
       'CFO/Debt', 'CFO', 'Interest Coverage', 'Total Liquidity',
       'Current Liquidity', 'Current Liabilities', 'EPS Before Extras', 'P
E',
       'ROA', 'ROE', 'InvGrd', 'Rating'],
      dtype='object')
```

In [7]:

```
# checking for null values
data.isnull().sum().to_numpy()
```

Out[7]:

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0], dtype=int64)
```

In [8]:

```
# find all categorical columns in dataset
categorical_cols = [col for col in data.columns if data[col].dtype == 'object']
print('Categorical columns:', categorical_cols)
```

```
Categorical columns: ['Rating']
```

Converting all Categorical Columns into Numerical

In [9]:

```
def convert_numerical(df, col):
    le = LabelEncoder()

    # encoding the categorical columns in the DataFrame using LabelEncoder
    for col in categorical_cols:
        if df[col].dtype == 'object':
            df[col] = le.fit_transform(df[col])

    return df
```

In [10]:

```
new_data = convert_numerical(data, categorical_cols)
new_data.head(4)
```

Out[10]:

	Sales/Revenues	Gross Margin	EBITDA	EBITDA Margin	Net Income Before Extras	Total Debt	Net Debt	LT Debt	:
0	-0.005496	0.030763	0.018885	0.024515	0.146849	-0.029710	-0.019296	-0.042648	0
1	-0.005496	0.030763	0.088716	0.094733	0.146849	-0.029710	-0.019296	-0.042648	0
2	-0.007045	0.023159	0.088716	0.096440	0.108590	0.039410	0.034268	0.009059	0
3	-0.009396	0.028400	0.088716	0.099046	0.146137	0.030071	0.036938	-0.016964	0

4 rows × 28 columns

In [11]:

```
data['InvGrd'].value_counts()
```

Out[11]:

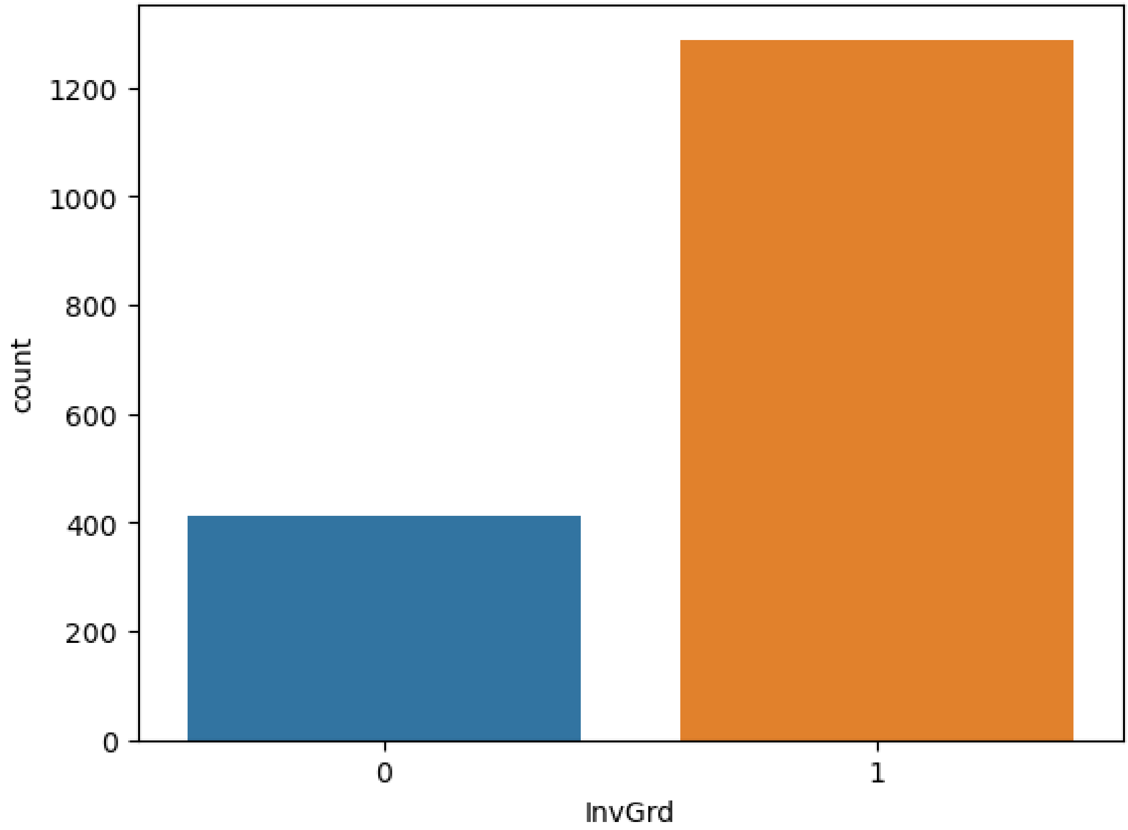
```
1    1287
0     413
Name: InvGrd, dtype: int64
```

In [12]:

```
sns.countplot(x=data['InvGrd'])
```

Out[12]:

```
<Axes: xlabel='InvGrd', ylabel='count'>
```



In [13]:

```
X, y = data.drop('InvGrd', axis=1), data['InvGrd']
X.shape, y.shape
```

Out[13]:

```
((1700, 27), (1700,))
```

Data Splicing

The dataset will be split into training & testing sets. The training data will be used for training, while testing data will be used for checking the skill of model on unseen dataset. The split for data will be 80:20, i.e., 80% will be placed in training and 20% will be placed in testijng set. This can be achieved by using sklearn's `train_test_split()` function.

In [14]:

```
# splitting the dataset into training and testing sets
test_size = 0.2
random_state = 42 # for reproducibility of the results

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

In [15]:

```
X_train.shape
```

Out[15]:

```
(1360, 27)
```

Applying Machine Learning Models

Two models of machine learning will be applied on dataset, in which Ridge and Lasso regularization techniques will be applied, which are described below:

- Logistic Regression (Ridge & Lasso)

In [17]:

```
def logistic_regression(model, X_train, X_test, y_train, y_test):
    # fit model to training data
    model.fit(X_train, y_train)

    # make predictions on testing data
    y_pred = model.predict(X_test)

    # calculate accuracy
    accuracy = accuracy_score(y_test, y_pred)

    # calculate classification report and confusion matrix
    report = classification_report(y_test, y_pred)
    matrix = confusion_matrix(y_test, y_pred)

    # calculating loss of model
    loss = log_loss(y_test, y_pred)

    # return results
    return accuracy, report, matrix, loss
```

In [18]:

```
# create models
ridge_model = LogisticRegression(penalty='l2', solver='lbfgs')
lasso_model = LogisticRegression(penalty='l1', solver='saga')

# apply logistic regression function and print results
ridge_accuracy, ridge_report, ridge_matrix, ridge_loss = logistic_regression(ridge_model,
                                                                  X_test, y_train, y_test)

lasso_accuracy, lasso_report, lasso_matrix, lasso_loss = logistic_regression(lasso_model,
                                                                  X_test, y_train, y_test)
```

In [19]:

```
print("Ridge Regression Results:")
print("Accuracy: %.2f%%" % ((ridge_accuracy)*100))
print('Loss of Ridge: %.2f' % (ridge_loss))
print("Classification Report:\n", ridge_report)
print("Confusion Matrix of Ridge Logistic Regression:\n", ridge_matrix)
```

```
Ridge Regression Results:
Accuracy: 76.76%
Loss of Ridge: 8.37
Classification Report:
               precision    recall  f1-score   support

           0       0.78      0.08      0.15        84
           1       0.77      0.99      0.87       256

    accuracy                           0.77       340
   macro avg       0.77      0.54      0.51       340
weighted avg       0.77      0.77      0.69       340

Confusion Matrix of Ridge Logistic Regression:
 [[  7  77]
 [  2 254]]
```

In [20]:

```
print("Lasso Regression Results:")
print("Accuracy: %.2f%%" % ((lasso_accuracy)*100))
print('Loss of Lasso: %.2f' % (lasso_loss))
print("Classification Report:\n", lasso_report)
print("Confusion Matrix of Lasso Logistic Regression:\n", lasso_matrix)
```

```
Lasso Regression Results:
Accuracy: 74.41%
Loss of Lasso: 9.22
Classification Report:
               precision    recall  f1-score   support

           0       0.20      0.01      0.02        84
           1       0.75      0.98      0.85       256

    accuracy                           0.74       340
   macro avg       0.48      0.50      0.44       340
weighted avg       0.62      0.74      0.65       340

Confusion Matrix of Lasso Logistic Regression:
 [[  1  83]
 [  4 252]]
```

Re-Sampling the Dataset Using Random Over Sampling

In [21]:

```
ros = RandomOverSampler(sampling_strategy='not majority', random_state=42)
X_resampled, y_resampled = ros.fit_resample(X, y)
```

In [22]:

```
# splitting dataset again on resampled data
X_train_r, X_test_r, y_train_r, y_test_r = train_test_split(X_resampled, y_resampled, tes
                                                    random_state=random_state)
```

Applying Models Again on Resampled Dataset

In [23]:

```
# applying models on sampled dataset
ridge_accuracy_r, ridge_report_r, ridge_matrix_r, ridge_loss_r = logistic_regression(ridg
                                                                 X_test_r, y_train_r, y_t

lasso_accuracy_r, lasso_report_r, lasso_matrix_r, lasso_loss_r = logistic_regression(lass
                                                                 X_test_r, y_train_r, y_t
```

In [24]:

```
print("Ridge Regression Results After Random Oversampling:")
print("Accuracy: %.2f%%" % ((ridge_accuracy_r)*100))
print('Loss of Ridge: %.2f' % (ridge_loss_r))
print("Classification Report:\n", ridge_report_r)
print("Confusion Matrix of Ridge Logistic Regression:\n", ridge_matrix_r)
```

```
Ridge Regression Results After Random Oversampling:
Accuracy: 61.55%
Loss of Ridge: 13.86
Classification Report:
               precision    recall  f1-score   support

           0       0.65      0.48      0.55       255
           1       0.60      0.75      0.66       260

    accuracy                           0.62       515
   macro avg       0.62      0.61      0.61       515
weighted avg       0.62      0.62      0.61       515

Confusion Matrix of Ridge Logistic Regression:
 [[123 132]
 [ 66 194]]
```

In [25]:

```
print("Lasso Regression Results After Random Oversampling:")
print("Accuracy: %.2f%%" % ((lasso_accuracy_r)*100))
print('Loss of Lasso: %.2f' % (lasso_loss_r))
print("Classification Report:\n", lasso_report_r)
print("Confusion Matrix of Lasso Logistic Regression:\n", lasso_matrix_r)
```

```
Lasso Regression Results After Random Oversampling:
Accuracy: 53.79%
Loss of Lasso: 16.66
Classification Report:
               precision    recall  f1-score   support

           0       0.52      0.94      0.67       255
           1       0.70      0.15      0.24       260

    accuracy                           0.54       515
   macro avg       0.61      0.54      0.45       515
weighted avg       0.61      0.54      0.45       515

Confusion Matrix of Lasso Logistic Regression:
 [[239  16]
 [222  38]]
```

Applying Artificial Neural Network

In [27]:

```
X_nn, y_nn = data.drop('Rating', axis=1), data['Rating']
X_nn.shape, y_nn.shape
```

Out[27]:

```
((1700, 27), (1700,))
```

In [28]:

```
# resampling the dataset
ros = RandomOverSampler(sampling_strategy='not majority', random_state=42)
X_resampled_nn, y_resampled_nn = ros.fit_resample(X_nn, y_nn)
```

In [29]:

```
y_new = to_categorical(y_resampled_nn)
y_new.shape
```

Out[29]:

```
(5216, 16)
```

In [30]:

```
# splitting dataset again on resampled data
X_train_nn, X_test_nn, y_train_nn, y_test_nn = train_test_split(X_resampled_nn, y_new, te
                                                    random_state=random_state)
```

In [31]:

```
X_train_nn.shape, y_train_nn.shape
```

Out[31]:

```
((4172, 27), (4172, 16))
```

In [50]:

```
def artificial_nn():
    model = Sequential()
    model.add(Dense(512, activation='relu', input_shape=(27,)))
    model.add(Dense(256, activation='relu'))
    model.add(Dense(128, activation='relu'))
    model.add(Dense(16, activation='softmax'))

    model.compile(optimizer=Adam(),
                  loss='binary_crossentropy',
                  metrics=['accuracy'])

    return model
```

In [51]:

```
nn_model = artificial_nn()
nn_model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_14 (Dense)            (None, 512)               14336

 dense_15 (Dense)            (None, 256)               131328

 dense_16 (Dense)            (None, 128)               32896

 dense_17 (Dense)            (None, 16)                2064

=================================================================
Total params: 180,624
Trainable params: 180,624
Non-trainable params: 0
_________________________________________________________________
```

In [52]:

```
callbacks = keras.callbacks.EarlyStopping(monitor='val_accuracy',
                                          patience=5,
                                          min_delta=0.01)

history = nn_model.fit(X_train_nn, y_train_nn, epochs=200,
             validation_split=0.2,
             callbacks=callbacks,
             verbose=1)
```

```
Epoch 1/200
105/105 [==============================] - 1s 5ms/step - loss: 0.3746 -
accuracy: 0.1756 - val_loss: 0.2425 - val_accuracy: 0.2802
Epoch 2/200
105/105 [==============================] - 0s 4ms/step - loss: 0.2125 -
accuracy: 0.3227 - val_loss: 0.1982 - val_accuracy: 0.3269
Epoch 3/200
105/105 [==============================] - 1s 8ms/step - loss: 0.1767 -
accuracy: 0.4243 - val_loss: 0.1711 - val_accuracy: 0.4192
Epoch 4/200
105/105 [==============================] - 0s 4ms/step - loss: 0.1531 -
accuracy: 0.4729 - val_loss: 0.1606 - val_accuracy: 0.4599
Epoch 5/200
105/105 [==============================] - 0s 4ms/step - loss: 0.1463 -
accuracy: 0.5106 - val_loss: 0.1473 - val_accuracy: 0.4790
Epoch 6/200
105/105 [==============================] - 0s 4ms/step - loss: 0.1358 -
accuracy: 0.5454 - val_loss: 0.1438 - val_accuracy: 0.5114
Epoch 7/200
```

In [62]:

```
_, accuracy_nn = nn_model.evaluate(X_test_nn, y_test_nn)
print('Accuracy of Neural Network Model: %.2f%%' % ((accuracy_nn)*100))
```

```
33/33 [==============================] - 0s 2ms/step - loss: 0.1123 - accu
racy: 0.8716
Accuracy of Neural Network Model: 87.16%
```

In [63]:

```
plt.figure(figsize=(10, 5))
plt.title('Compulative Comparison of Classifiers (Test Dataset)')
models = [' Logistic Ridge (InvGRD)', 'Logistic Lasso (InvGrd)', 'Artificial Neural Netwo
res =[ridge_accuracy_r, lasso_accuracy_r, accuracy_nn]
plt.bar(models, res)
```

Out[63]:

```
<BarContainer object of 3 artists>
```