# Using Machine Learning to Uncover Long-Term Investment Prospects

**by**

**Ronit Sohal**

**Abstract**

Investing refers to the act of allocating resources, usually money, with the expectation of generating an income or profit. It can take on many forms, such as buying stocks, bonds, real estate, or starting a business. However, this paper focuses on investment through the stock market but with a twist, machine learning. I go through several machine learning models to determine if determining a good investment is possible through machine learning and which model would be best at determining whether an investment is worthwhile.

**Introduction**

Using Machine Learning and fundamental approaches, it is possible to predict whether an investment is a good investment or not and which machine learning model would be best at it. In order to complete this project I gathered a dataset consisting of data from financial reports on the SEC website. Using this data I was able to calculate several financial metrics which are usually used to establish if a company is doing well. I preprocessed this dataset and then implemented five different machine learning models with three models outperforming the others.

**Data Selection & Manipulation**

I got my data from an online vendor which reflects the financial statements provided by the SEC website. I loaded the data into Google Collab to conduct my research and joined them together to form the final dataset that I will be using for my research. Below are the libraries/packages I used for my research.

```
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import csv
import seaborn as seaborn
```

Before going into the actual research, here is a quick look at how I merged the dataset

```
df1 = income.merge(balance, left_on=['Ticker', 'Fiscal Year'],
right_on=['Ticker', 'Fiscal Year']); df1.drop_duplicates()
df2 = df1.merge(cashflow, left_on=['Ticker', 'Fiscal Year'],
right_on=['Ticker', 'Fiscal Year']); df2.drop_duplicates()
DATA = df2.copy()
DATA.head()
```

The final dataset is a merged dataset of several companies and consists of data on the several features present in the annual reports of those companies; however, for the research I am conducting I only need certain values in order to calculate the features required. The features or benchmarks I will be using are the Current Ratio, LTD-to-Earnings, ROE, Debt to EBIT, and Interest Coverage Ratio. Below is how I calculated these features and added them to the dataset.

```
DATA['Current Ratio'] = DATA['Total Current Assets']/DATA['Total
Current Liabilities']
DATA['LTD-to-Earnings'] = DATA['Long Term Debt']/DATA['Net
Income']
DATA['ROE'] = DATA['Net Income']/(DATA['Total
Assets']-DATA['Total Liabilities'])
DATA['Debt-to-EBIT'] = (DATA['Long Term Debt'] + DATA['Short
Term Debt']) /DATA['Pretax Income (Loss)']
DATA['Interest Coverage Ratio'] = DATA['Net Income']/DATA['Interest
Expense  Net']
```

After adding these columns, I added a “Success” column which determined if the company met certain standardized values for each feature calculated above. If the company met the criteria for being a Successful company it was marked with a 1 and if it even missed one benchmark it was marked with a 0.

```
DATA['Success'] = np.where((DATA['Current Ratio']>1)&
(DATA['LTD-to-Earnings']<5) & (DATA['ROE']>0.12)  &
(DATA['Debt-to-EBIT']<3), 1, 0)
```

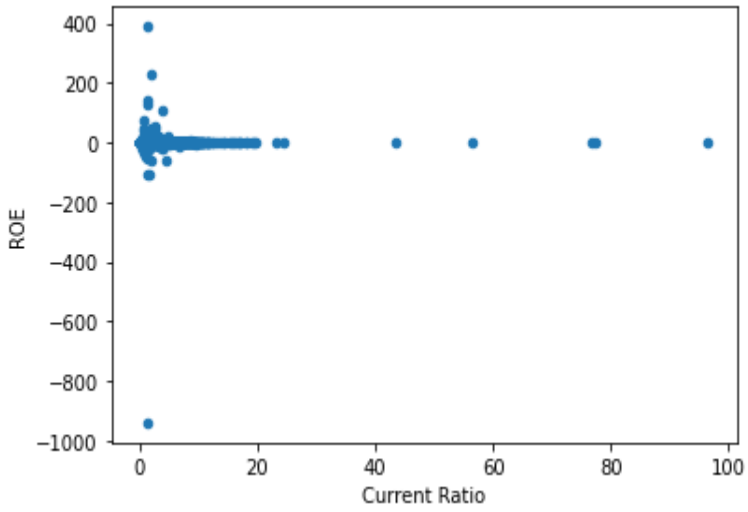
I then went on to pre-process the data by trimming the dataset down to only the five features I will be using to conduct my research and deleting any undefined or missing values.

```
DATA=DATA[['Current Ratio', 'LTD-to-Earnings', 'ROE',
'Debt-to-EBIT','Interest Coverage Ratio', 'Success']]
DATA.replace([np.inf, -np.inf], np.nan, inplace=True) # this code
DATA=DATA.dropna() #drop missing values
DATA.head()
```

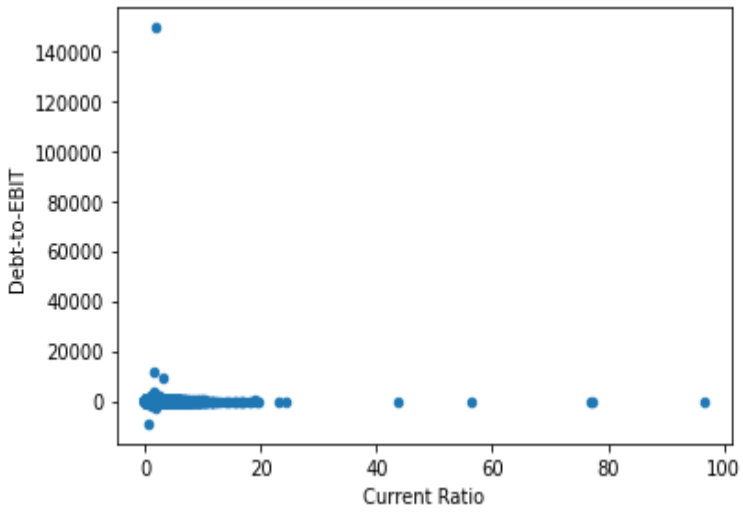
| | Current Ratio | LTD-to-Earnings | ROE | Debt-to-EBIT | Interest Coverage Ratio | Success |
|---|---|---|---|---|---|---|
| 0 | 3.846561 | 4.121212 | 0.108808 | 3.500000 | -7.573770 | 0 |
| 1 | 3.300871 | 2.633041 | 0.141468 | 2.504359 | -12.000000 | 1 |
| 2 | 3.286080 | 5.693038 | 0.069131 | 1.901691 | -8.540541 | 0 |
| 3 | 1.533173 | 1.672269 | 0.225569 | 2.619151 | -28.184211 | 1 |
| 4 | 2.327880 | 3.176634 | 0.147548 | 2.801663 | -10.271429 | 1 |

**Exploratory Data Analysis**

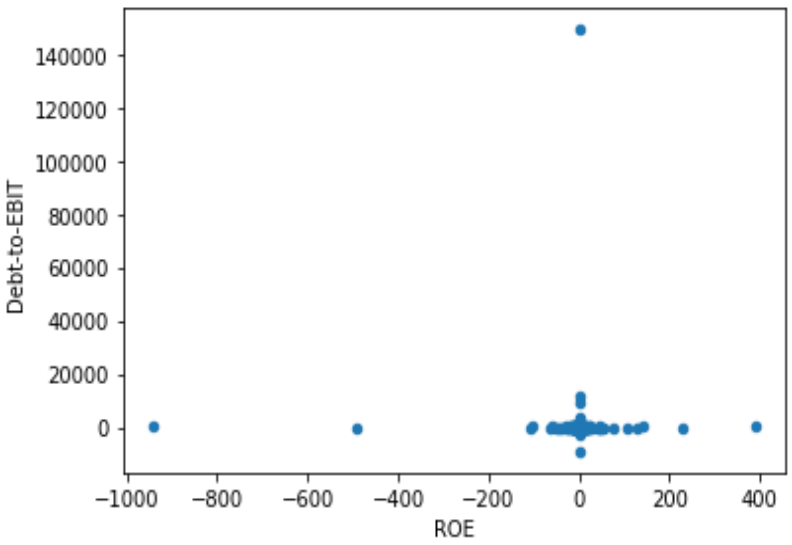
Before diving into the experiment, it is important to understand the features being used and the relationship between the features. Plotting graphs such as Scatterplots, Histograms, Box Plots, and Correlation heatmaps will give better insight into the data.



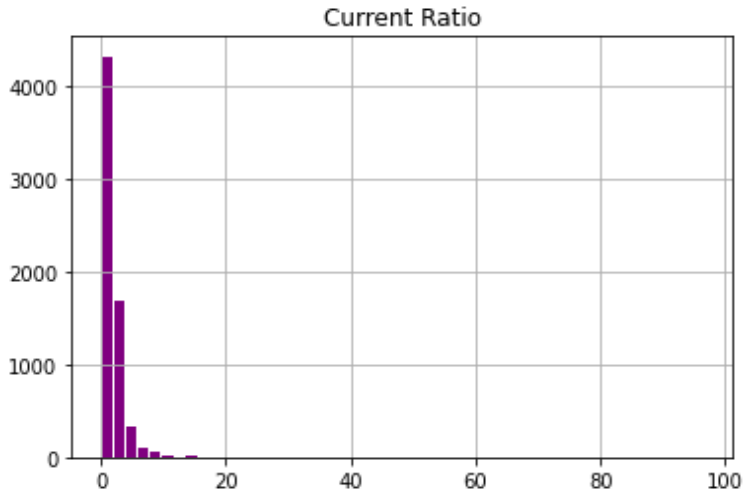
As can be seen from the Scatterplot to the left, as Current Ratio changes ROE doesn't change and vice versa. This shows there is no correlation between the two features and it is safe to use both in the experiment. The ability to manage debt has no effect on how efficiently a company generates profit.



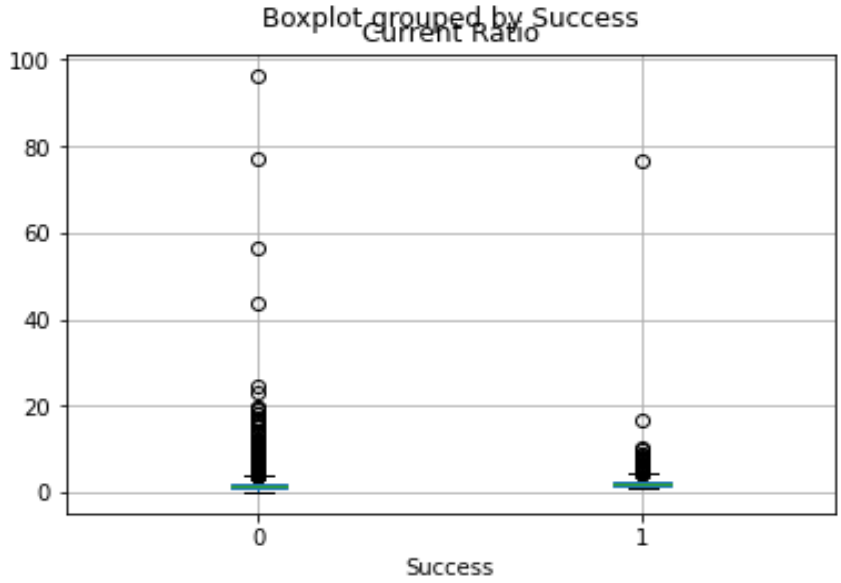

As can be seen from the Scatterplot to the left, as Current Ratio changes Debt-to-EBIT doesn't change and vice versa. This shows there is no correlation between the two features and it is safe to use both in the experiment. A company good at managing debt requires less income to put towards debt.



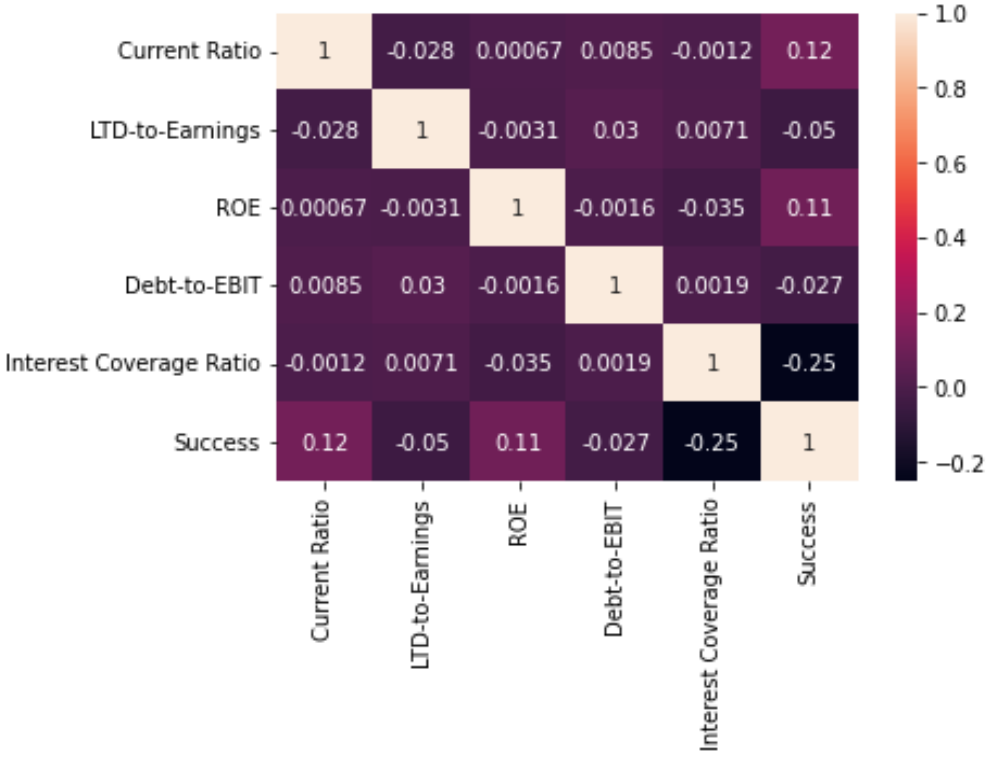
The two values can go in any direction positive to negative as shown by the plot to the right. Along with that we can once again see no correlation between the two values indicating that it is safe to use both features in our experiment.




The Current Ratio Histogram shows a right skew meaning there are some extremes that exist within the dataset. In those years those companies were very good at managing their debt as their Current Ratio was well above 1. Additionally, even though these extremes exist within the data set they are not a majority within the data meaning there shouldn't be an issue in that regard.



This Current Ratio boxplot shows the existence of potential outliers, points that are way above the performance of a normal company. This shows the need to remove them from the data as they can potentially skew predictions and which will lead to a poor machine learning model.



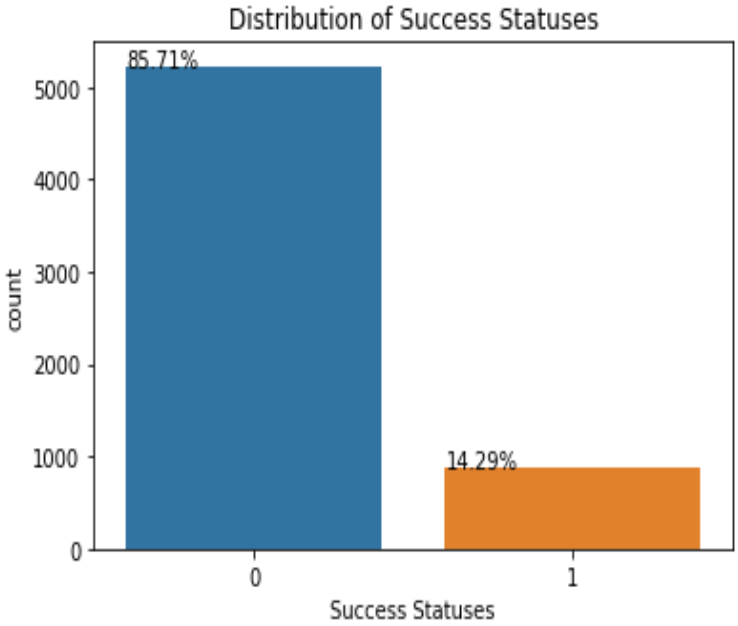

This is a Correlation Heatmap between all of the five features I will be using to make my predictions. As it can be seen there is no strong correlation between any one of the features indicating that using all five of these features in our prediction model is ok

**Data Pre-Processing**

After finding out about the existence of outliers which could skew the results of the prediction model and the experiment as a whole, I figured a little pre-processing was needed. I went on to eliminate extreme outliers which would be any company above the 99th percentile of any feature and below the 1st percentile of any feature. The code I used can be found below:

```
FEATURES = ['Current Ratio', 'LTD-to-Earnings', 'ROE',
'Debt-to-EBIT','Interest Coverage Ratio']
RESPONSE = 'Success'
newDATA = DATA.copy()
for f in FEATURES:
  q_low = newDATA[f].quantile(0.01) # identifying threshold for the
bottom 1%
  q_hi  = newDATA[f].quantile(0.99) # identifying threshold for the top 1%

  df_filtered = newDATA[(newDATA[f] < q_hi) & (newDATA[f] > q_low)]
```

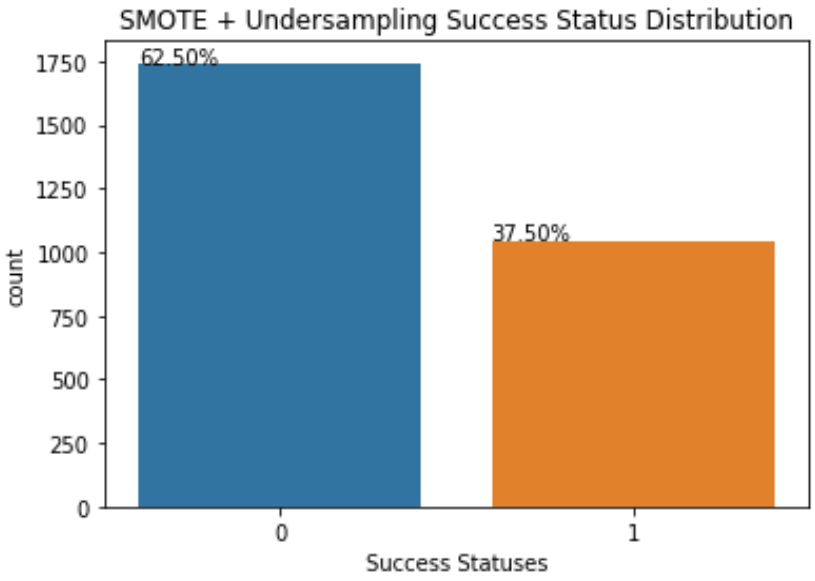
Following the outlier problem, another problem that was already present in the dataset was Label Imbalancing:




As it can be seen from the Distribution of Success Statuses the minority in this case, which are the 1's or the companies labeled as successful, only makeup 14.29 percent of the data set. As it is, the machine learning model won't have enough data to learn the decision boundary. To fix this problem I implemented Synthetic Minority Oversampling Technique or SMOTE for short.

```
from collections import Counter # counts hashable objects
from imblearn.over_sampling import SMOTE # Synthetic Minority
Over-sampling Technique.
# smote balances class distribution by randomly increasing minority
class examples + replicating them

from imblearn.under_sampling import RandomUnderSampler #
# difference between random undersampling vs smote: smote creates
new artificial training examples to increase size of dataset. under
sampler
# does the exact opposite

from imblearn.pipeline import Pipeline # sequence of data processing
mechanisms.
from numpy import where
FEATURES = df_filtered.columns[:-1]; RESPONSE =
DATA.columns[-1]
over = SMOTE(sampling_strategy=0.2)
under = RandomUnderSampler(sampling_strategy=0.6) # increase the
observations for class 1 so that they are closer to class 0. So it will be a
fair prediction.
# we are trying to increase the amount of companies that are successful
(duplicating the samples, redrawing from them) so we can

# increased the successful
steps = [('o', over), ('u', under)]
pipeline = Pipeline(steps=steps)
# transform the dataset

X_SMOTE, y_SMOTE = pipeline.fit_resample(df_filtered.loc[:,
FEATURES], df_filtered.loc[:, RESPONSE])

# plotting label distribution after SMOTE
```

```
ax = sns.countplot(x='Success', data = pd.DataFrame(y_SMOTE))
plt.title('SMOTE + Undersampling Success Status Distribution')
plt.xlabel('Success Statuses')
total = len(y_SMOTE)
for p in ax.patches:
    percentage = '{:.2f}%'.format(100 * p.get_height()/total)
    x_coord = p.get_x()
    y_coord = p.get_y() + p.get_height()+0.02
    ax.annotate(percentage, (x_coord, y_coord))
```




As it can be seen through the help of SMOTE the minority now makes up 37.50% of the dataset which is enough data for the machine learning model to make accurate decision boundaries.

The last step of my Data pre-processing was splitting the data into two data sets one for training and one for testing. Using a 80/20 split, 80% for training and 20% for testing, the data was now ready for the classification models.

## Modeling

I went through several classification models to see which one would be the best classifier in terms of reflecting real life, accuracy, and complexity. In other words, a model that is more accurate and less complicated than another model would be classified as better. Additionally, the model should have as low of a false positive rate as possible as it could be catastrophic if someone invests in a company that essentially is indicated to fail. Therefore, a good indicator for the accuracy of the model would be the F1-Score rather than simply using just the Precision value or just the Recall value. The precision value only tells us about the true-positives relative to all the positive predictions which doesn't give much information about the accuracy of the model. The Recall value is a reflection of the true positives relative to all positive predictions, this value is often traded for a better Precision score. Essentially, choosing a high precision model would be the best in terms of minimizing loss through investment, but completely neglecting the Recall value could result in an investment opportunity being missed. Therefore, F1-Score, which takes both metrics into account, would be the best metric to determine the best classification model.

Here are the classification models I tested include:

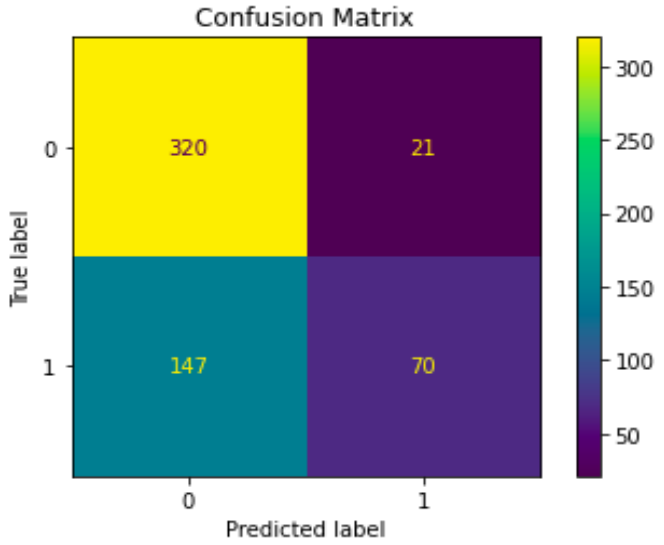
- Logistic Regression Model
- K Nearest Neighbors Model
- Random Forest Model
- Decision Tree Model
- Boosting Model

1. **Logistic Regression Model**

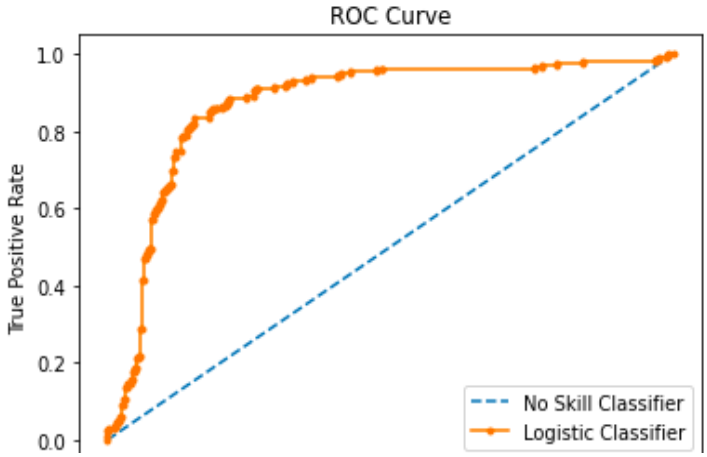
   The model estimates the probability of the binary outcome by learning the relationship from the input features. It works by using an equation that describes the relationship between the input features and the binary outcome. When the equation is graphed it takes the form of an S-Shaped curve, known as the logistic function. The output of this function represents the probability of the binary outcome being 1.

Implementation of the Logistic Regression Model:

```
from sklearn.linear_model import LogisticRegression
lr_model = LogisticRegression(random_state =
0).fit(X_train,y_train)
lr_prediction = lr_model.predict(X_test)
```




As seen through the confusion matrix of the model, while the model classifies a majority of the success and failures correctly the model still classifies some of the companies incorrectly. This is reflected in the accuracy score of 70%. Additionally, this model has a F1-Score of 0.45 for true values and 0.79 for false values. This model doesn't seem too good at classifying good investments as good investments but seems to be good at marking bad companies as bad companies.



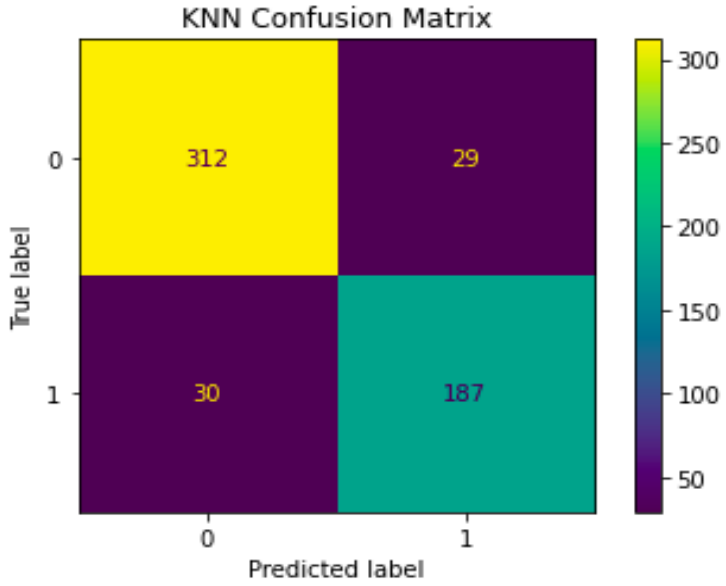
The ROC curve is close to the top left corner indicating that it is decently good at correctly identifying positive cases while minimizing false positives. The AUC score is 0.871 which is about 0.13 off of a perfect classifier. Additionally, both the shape of the curve and the area under the curve tell us that this model is better than just blind guessing which essentially means that this model can be used to invest with some insight. While it might not be the best model as it still classifies incorrectly it is better than blind guessing.

## 2. K-Nearest Neighbors Model

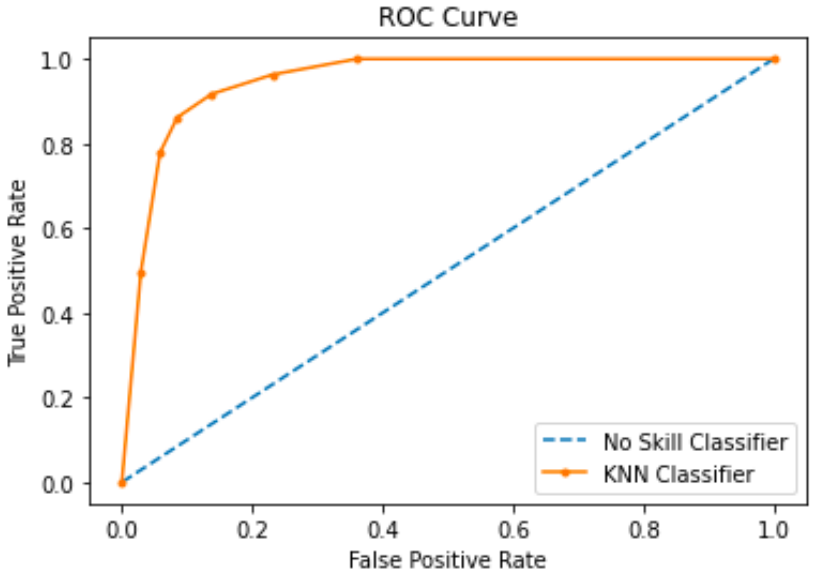
The k-nearest neighbors (KNN) model is a type of supervised machine learning algorithm used for classification and regression. The model is trained on a labeled dataset, where each example has one or more features and a label. To make a prediction for a new example, the KNN algorithm finds the k labeled examples in the training dataset that are closest to the new example (based on some distance metric, such as Euclidean distance), and then assigns the label that is most common among those k nearest neighbors.

Implementation of K-Nearest Neighbors Model:

```
# importing KNN algorithm
from sklearn.neighbors import KNeighborsClassifier
# K value set to be 6
knn_model = KNeighborsClassifier(n_neighbors=6)
# model training
knn_model.fit(X_train,y_train)
# testing the model
KNN_Prediction = knn_model.predict(X_test)
```




As seen through the confusion matrix of the model, while the model classifies a majority of the success and failures correctly the model still classifies a couple of the companies incorrectly. This is reflected in the accuracy score of 89%. Additionally, this model has an F1-Score of 0.91 for true values and 0.86 for false values. This model is fairly accurate and precise when it comes to classification; however, it isn't perfect and still has a small chance of misleading an investment.



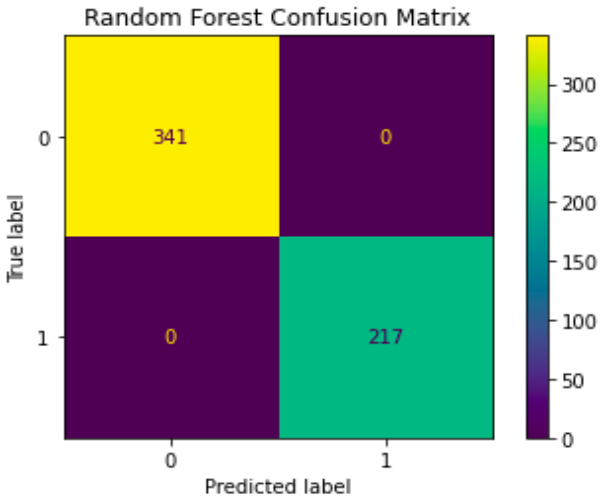
The ROC curve is very close to the top left corner indicating that it is decently good at correctly identifying positive cases while minimizing false positives. The AUC score is 0.949 which is about 0.05 off of a perfect classifier. Additionally, both the shape of the curve and the area under the curve tell us that this model is better than just blind guessing which essentially means that this model can be used to invest with some insight.

## 3. Random Forest Model

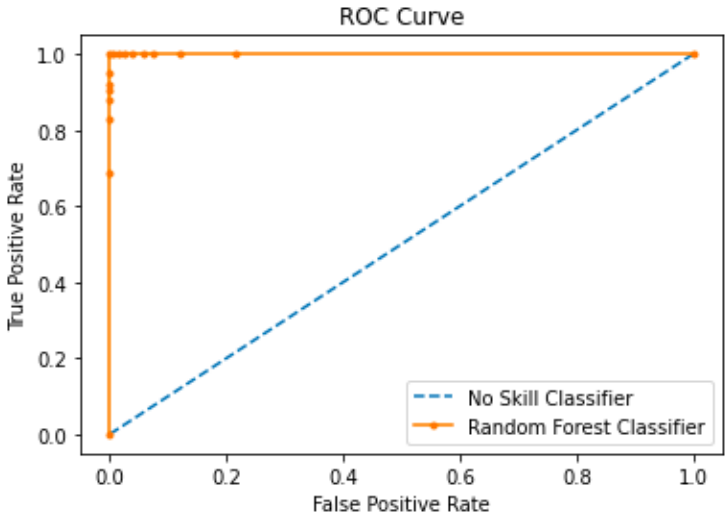
The random forest model is an ensemble learning method for classification, regression and other tasks that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. In other words, it is a collection of decision trees (forest) where each tree is grown randomly from a bootstrap sample (randomly drawn with replacement) of the original training data. It functions by randomly training multiple decision trees on different subsets of the training data and then averaging the predictions made by each tree, the subsets of the training data used to train each tree are chosen randomly (hence the word “random”). Each tree in the forest makes a prediction, and the final prediction is made by taking the majority vote among all the trees (in classification) or averaging the predictions (in regression).

Random Forest Model Implementation:

```
# import Random Forest classifier
from sklearn.ensemble import RandomForestClassifier
# instantiate the classifier
rf_model = RandomForestClassifier()
# fit the model
rf_model.fit(X_train, y_train)
# testing the model
RF_Prediction = rf_model.predict(X_test)
```




As seen through the confusion matrix of the model, the model classifies the entire test data set correctly which is reflected in the accuracy score of 100%. Additionally, this model has a F1-Score of 1.00 for true values and 1.00 for false values, as it has an accuracy of 100%. While this model is both accurate and precise when it comes to classification it could potentially result in a failed investment since we only know it is 100% accurate over the data we have, which is very limited.



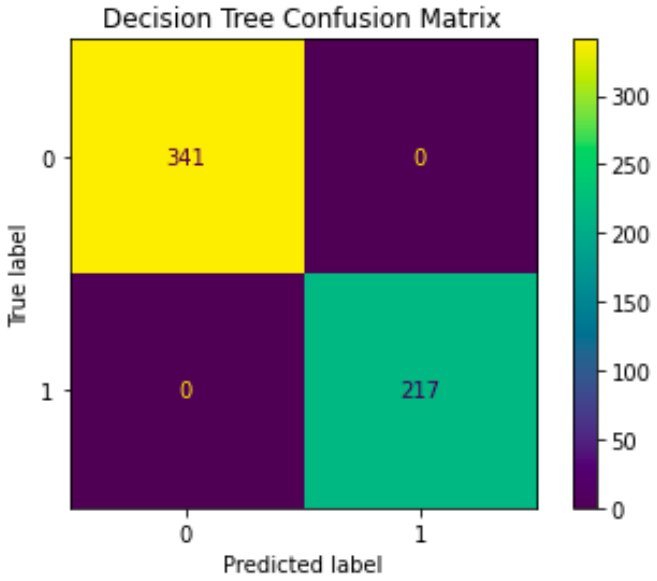
The ROC curve goes through the top left corner indicating that it is very good at correctly identifying positive cases while completely minimizing false positives. The AUC score is 1.00 which is a perfect classifier. Additionally, both the shape of the curve and the AUC score tell us that this model is better than just blind guessing which essentially means that this model can be used to invest with some insight. Especially since it succeeded at classifying all companies in the test data correctly.

4. **Decision Tree Model**

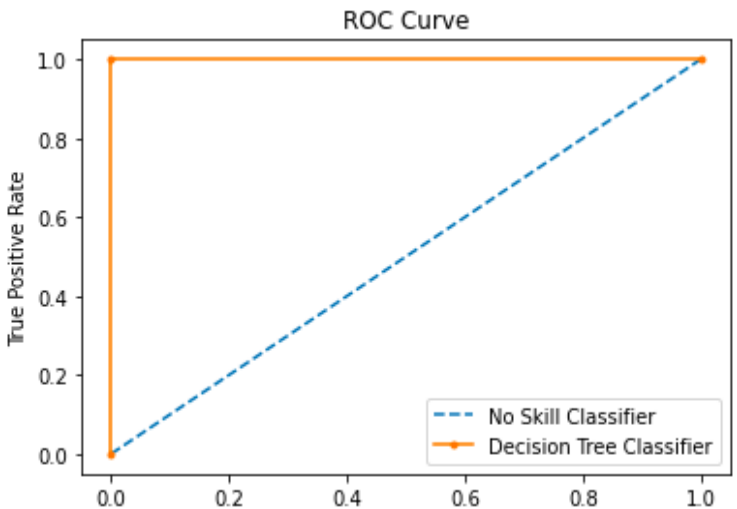
   A decision tree is a supervised machine learning algorithm used for classification and regression. It is a tree-like model that represents decisions and their possible consequences, represented graphically. The algorithm starts with the root node, which represents the entire dataset. It then splits the dataset based on the feature that results in the greatest reduction in impurity. The process continues recursively until a stopping criterion is met. The final outcome of a decision tree model is a set of rules that can be used to make predictions on new, unseen examples by traversing the tree from the root to a leaf node. Decision trees are simple to understand, interpret and visualize, but are prone to overfitting which can be addressed by techniques such as pruning, ensemble methods, and setting the depth of the tree.

   Decision Tree Model Implementation:

```
from sklearn.tree import DecisionTreeClassifier
dt_model = DecisionTreeClassifier()
dt_model = dt_model.fit(X_train,y_train)
dt_pred = dt_model.predict(X_test)
```




As seen through the confusion matrix of the model, the model classifies the entire test data set correctly which is reflected in the accuracy score of 100%. Additionally, this model has a F1-Score of  1.00 for true values and 1.00 for false values, as it has an accuracy of 100%. While this model is both accurate and precise when it comes to classification it could potentially result in a failed investment since we only know it is 100% accurate over the data we have, which is very limited.



The ROC curve goes through the top left corner indicating that it is very good at correctly identifying positive cases while completely minimizing false positives. The AUC score is 1.00 which is a perfect classifier. Additionally, both the shape of the curve and the AUC score tell us that this model is better than just blind guessing which essentially means that this model can be used to invest with some insight. Especially since it succeeded at classifying all companies in the test data correctly.

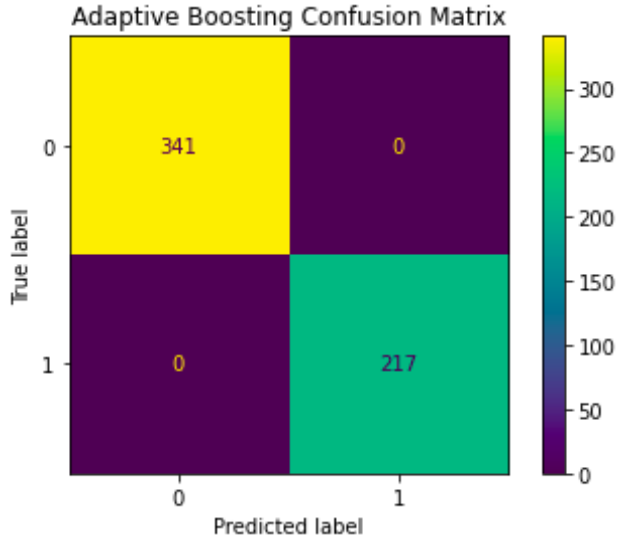
5. **Boosting Model (Adaboost)**

Boosting is an ensemble learning method that combines multiple weak models to create a strong model. It is used to improve the accuracy of a model by reducing the bias and variance of the model. Boosting algorithms work by iteratively training weak models (such as decision trees) and combining their predictions. At each iteration, the algorithm focuses on the examples that the current ensemble of weak models is struggling to correctly classify, by giving more weight to misclassified examples. The final prediction is made by taking a weighted majority vote among all the weak models. The most popular boosting algorithm is the AdaBoost, it works by iteratively training a weak model and adjusting the weight of the examples based on the model's accuracy. The goal is to focus on the examples that are hard to classify and give more weight to them on the next iteration.

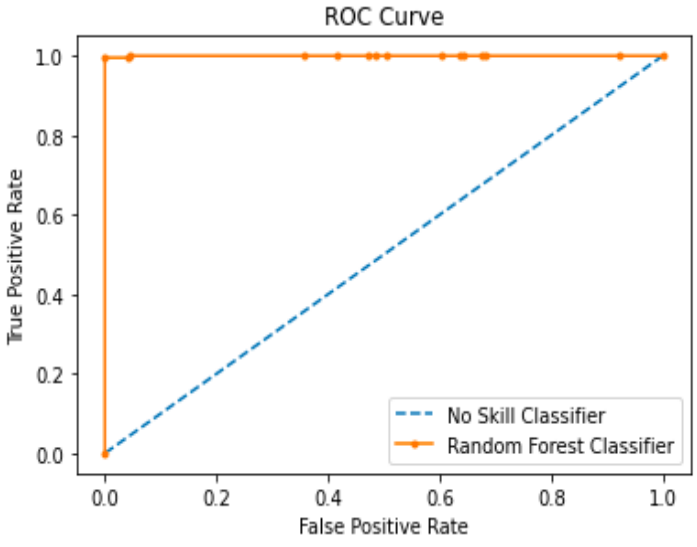
Adaboost implementation:

```
from sklearn.ensemble import AdaBoostClassifier
adaboost = AdaBoostClassifier()
adaboost.fit(X_train,y_train)
```

```
adaboost_pred = adaboost.predict(X_test)
```




As seen through the confusion matrix of the model, the model classifies the entire test data set correctly which is reflected in the accuracy score of 100%. Additionally, this model has a F1-Score of  1.00 for true values and 1.00 for false values, as it has an accuracy of 100%. While this model is both accurate and precise when it comes to classification it could potentially result in a failed investment since we only know it is 100% accurate over the data we have, which is very limited.



The ROC curve goes through the top left corner indicating that it is very good at correctly identifying positive cases while completely minimizing false positives. The AUC score is 1.00 which is a perfect classifier. Additionally, both the shape of the curve and the AUC score tell us that this model is better than just blind guessing which essentially means that this model can be used to invest with some insight. Especially since it succeeded at classifying all companies in the test data correctly.

**Results**

To determine the best machine learning model we will be looking directly at the F1-Score of each model. Out of the five models I tried, three had a 100% accuracy and thus had a F1-Score of 1.00 (which is the highest score possible). In practical use all three models would serve the purpose of accurately classifying a good investment while also minimizing the potential of a significant loss through the investment; however, since both the Random Forest Model and Adaboost Model implement decision trees in their models they are considered more complicated. Therefore the optimal model for this problem would be the Decision Tree model as it is 100% accurate over the test data and parsimonious.

**Conclusion**

Determining a worthwhile investment is possible through machine learning and with the help of fundamental metrics. The best model that I discovered to work is the Decision Tree model which creates a tree-like structure of decisions and their possible consequences, where each internal node represents a test on an attribute, each branch represents an outcome of that test and each leaf node represents a class label. This model was determined to be better due to it being parsimonious and its ability to classify every company in the test data set correctly. To take this project

to the next level I want to incorporate more approaches, such as technical indicators, to determine if an investment is worthwhile. This will make the model better and more versatile which will only serve to make investment easier and more accessible