PREDICTIONS ON MERGERS AND ACQUISITIONS DEAL COMPLETION with Machine Learning and Natural Language Processing methodologies

by

Patience Buxton

A project

submitted in fulfillment

of the requirements for the degree of

Master of Science in Data Science

University of London

March 2023

DEDICATION

I dedicate this project to my family, especially to my twin sister who has been a great source of strength, love and support in these past two years. She has pushed me to do my best and this achievement is also thanks to her, being in my life. Thank you Pri.

ACKNOWLEDGMENTS

I would like to acknowledge and thank the committee members and the tutors that helped my work on this project. I appreciate your support.

ABSTRACT

The prediction of M&A (Mergers and Acquisitions) is part of the so-called risk arbitrage which is the investment strategy that speculates on the successful completion of takeover deals (Buxton, 2022). It is of great importance to be able to correctly predict these deals completion because, when a deal is mistakenly classified as a completed deal, the arbitrageur may incur in enormous losses as a result of investing in shares of companies, target of the M&A.

In this paper, mergers and acquisitions successful deal completion was predicted with the use of Machine Learning algorithms and Natural Language Processing techniques. In more detail, after an accurate variable selection step on the numerical financial dataset, different ML techniques (NN, Random Forest, SVM and Decision Tree) were implemented to make the predictions. A logistic regression, was used as a baseline against which the performance of the selected classifiers was contrasted and compared, in order to determine the best performing model.

Sentiment analysis on a textual dataset formed by news and headlines related to M&A activities, before the deal announcement, was then performed. The results were firstly evaluated together with the share prices (to assess their predictive powers) and secondly, with the previously determined numerical financial datasets through the use of the chosen classifiers, in order to assess if the overall models' prediction power could be improved. The preliminary results indicated that the baseline model is outperformed by the Random Forest, Decision Tree and SVM models with a perfect 100% on Recall and Precision. Sentiment scores yielded excellent results when analyzed with the share prices, with 4 out of 5 models registering perfect scores. However, the results of ML and NLP combined predictive power of the models were not sufficient to prove a net improvement of the results.

The paper aims to help arbitrageurs and shareholders to better predict the deal status and to improve their investment strategy, but also, it is an excellent example of application of state-of-the-art machine learning techniques in the financial world.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

M&A: Mergers and Acquisitions

ML: Machine Learning

SVM: Support Vector Machine

NN: Neural Network

NLP: Natural Language Processing

BERT: Bidirectional Encoder Representations from Transformers

LSTM: Long Short-Term Memory Networks

FFNN: Feedforward Neural Network

VADER: Valence Aware Dictionary and Sentiment Reasoner

FinBert: Financial corpus trained BERT

WNN: Wavelet Neural Network

GNN: Graph neural networks

CHAPTER 1 – INTRODUCTION

Introduction

Nowadays, access to an unprecedented amount of data from numerical to textual to cloud computing, together with the raise of innovative technologies, have increased the application of machine learning algorithms and artificial intelligence techniques in finance and other industries.

Historically, M&A activities coincide with the existence of companies. In 1708, for example, the East India Company merged with a competitor to restore its monopoly over the Indian trade. Another example is the Italian Monte dei Paschi and Monte Pio banks which in 1784, were united as the Monti Reuniti (Wikipedia Contributors, 2023).

Technically, a merger is a legal consolidation of two business entities into one, whereas an acquisition occurs when a company purchase another and takes ownership of the company's share capital, equity interests or assets (Wikipedia Contributors, 2023).

There are different types of mergers and acquisitions, for example, we have horizontal mergers, which happens when two companies in direct competition which share the same product lines and markets, decide to merge; or vertical merger: when a company and its supplier, merge (Investopedia, 2023).

On the other hand, acquisitions can be acquisition/takeover, which is the purchase of one business or company by another company or other business entity; or a consolidation/amalgamation which occurs when two companies combine to form a new enterprise altogether, and neither of the previous companies remains independently. Acquisitions are also divided into "private" and "public" acquisitions, depending on whether the acquiree or merging company (also termed as target company) is or is not listed on a public stock market. A last and additional acquisition categorization consists of whether an acquisition is friendly or hostile: whether a purchase is perceived as being a "friendly" one or "hostile" depends significantly on how the proposed acquisition is communicated to and perceived by the target company's board of directors, employees and shareholders (Wikipedia Contributors, 2023).

Research rationale, aims and objectives

This paper aims to:

- Predict the success of M&A deal completion, using ML classification models.
- Identify the best performing ML classification model for the prediction of M&A successful deal completion.
- Predict if there will be a deal announcement, through sentiment analysis (using NLP methodologies) on news and headlines released before a M&A deal announcement.

According to (Refinitiv, 2022) the logistic regression classification model appears to be the best performing machine learning algorithm for M&A prediction - this is confirmed also in previous studies, for example in Moriarty, Ly, Lanv and McIntosh (2019) - as it generalizes better and provides higher accuracy.

A logistic regression model predicts P(Y=1) as a function of X; it is perfect for numerical databases, is easy to implement and very efficient to train, however it is quite sensitive to outliers. It is one of the most efficient machine learning classifiers for binary classification problems. The target of the algorithm (or the dependent variable) has only two possible classes, having data coded as either 1 or 0 (Buxton, 2021).

With this information in mind, the first research question is:

- Using the logistic regression as a baseline model, what is the next best performing classification model that can predict the success of M&A deal completion? Can this model outperform the logistic regression?

As reported by Karatas and Hirsa (2021), to construct a useful prediction framework, size of the target, management attitude, termination fee, competing bids and bid premium are the best variables to be taken into consideration.

Several studies have indicated that there are different measures which are statistically significant predictors of a company becoming an acquisition target. In regards to private companies, it is well known that acquisition target companies share some of these characteristics, for example: they have higher growth than non-targets; are more profitable than private non-targets, are significantly more leveraged than private non-targets and, finally, have lower levels of liquidity than non-targets (City, 2016).

So based on this project identified datasets, the second research question is:

- Which variables are the most important and impactful in making M&A prediction?

Pound and Zeckhauser (1990) have analyzed how takeover rumors and news (issued before an M&A deal announcement) can affect the stock prices.

The authors discovered that there is an approximately 7% run-up (increase in the rumored targets' company stock prices) over the 20 trading days before rumor publication.

In an estimation window of 3 months before a deal announcement, Karatas and Hirsa (2021) showed in Figure 1. below, the effect of takeover rumors and news on the share prices. These rumors or headlines are either released from the inside of the companies or developed from studying the changes in the company's shares prices itself.

The Figure shows that, depending on the market's belief on the rumor, there is a possible increase in the share price of the target company. The figure also shows how there is a marked and consistent run-up in the days preceding an official announcement.

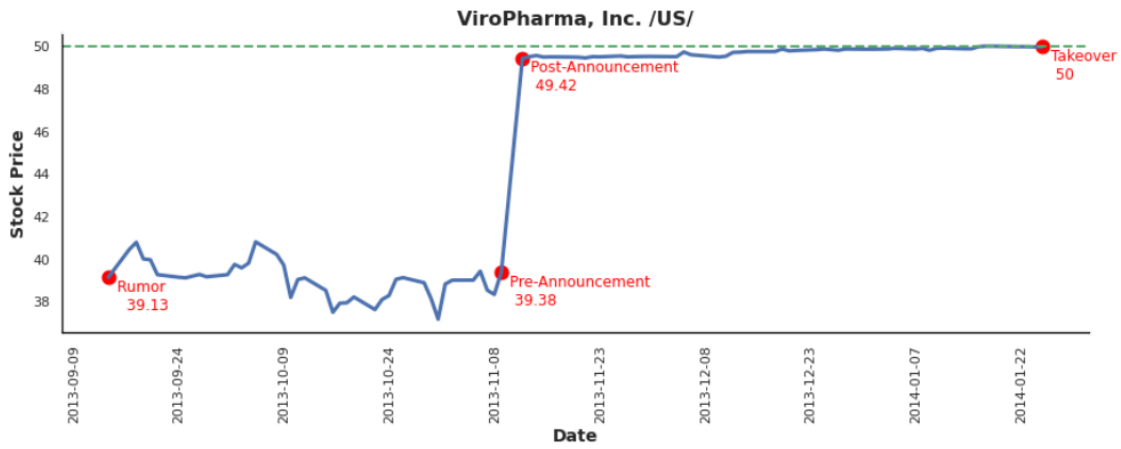

Figure 1. A Real-life example of stock price movement for Target company during M&A period (Source: Karatas and Hirsa 2021, p. 2)

According to these facts, analyzing rumors, news or headlines preceding a deal announcement and following these rumors with price share changes, is of great importance as these elements are very useful for deal prediction. These changes, can also considerably help the investment strategy of arbitrageurs.

To capture these predictive powers of these rumors in a form that can be utilized and analyzed in machine learning algorithms, sentiment analysis was performed on these rumors and on the basis of the above facts, the paper's third research question is:

- Can applying NLP sentiment analysis on rumors and news preceding an M&A announcement (while also taking into consideration the share prices change) actually help us predict an imminent deal announcement?

The hypothesis here is that:

- High share prices, together with no or very low positive news sentiment, is an indication of an imminent M&A announcement.

The fourth and last research question will try to investigate the below: can a combined approach with NLP sentiment scores, calculated on these headlines and rumors, together with ML methodologies on a full financial dataset (so not only share prices) improve or outperform the results of a model calculated solely on financial variables? So, without sentiment scores?

This question aims to identify if sentiment scores have the power to improve or even outperform models based on financial numerical datasets and is a good basis for further work to be developed.

Dissertation structure

The project starts with some background literature on M&A predictions.

Datasets description, exploration and pre-processing was then undertaken on both the financial and the textual datasets. The study then progresses with dimensionality reduction on the financial dataset: the output, a dataset only inclusive of the best performing variables, not only was evaluated and used as the default financial dataset for the rest of the study in all the model applications, but also provided an answer to the second research question, which reads: which variables are the most important and impactful in making M&A prediction?

The project then undertakes ML model application on the financial dataset: logistic regression was used as a baseline model against which Random Forest, Neural Network (NN), Decision Tree and Support Vector Machine (SVM) were confronted. The results of the analysis of this section, provided an answer to the first research question: using the logistic regression as a baseline model, what is the next best performing classification model that can predict the success of M&A deal completion? Can this model outperform the logistic regression?

Next, sentiment analysis with NLP was performed on the cleaned textual (headlines) dataset. The baseline against which the performance of VADER was compared to, was Textbolb sentiment analyzer.

The results from the sentiment analysis were evaluated and the sentiment scores added to the share prices variable only. In this approach, all the chosen algorithms were re-applied to this dataset; the approached helped us determine the best performing sentiment model and respond to the third research question: can applying NLP sentiment analysis on news prior to an announcement (while also taking into consideration the share prices change) actually help us predict an imminent M&A deal? Here, it was also possible to confirm or disprove the hypothesis.

Finally, the results of the sentiment analysis were applied to the full financial dataset (inclusive of the share prices) and all of the chosen algorithms re-applied again, in order to further compare and evaluate the results and gain more insights on if and how, sentiment analysis could improve the models' performances.

This approach allowed us to answer the fourth and last research question: can NLP sentiment scores with ML methodologies (together), improve or outperform the results of a methodology without sentiment scores?

This last evaluation provided a good understanding of the effect of the sentiment variables on the results.

CHAPTER 2 - BACKGROUND LITERATURE

The prediction of the likelihood of mergers or acquisitions successful deal completion, is a subject which has been studied in various related works.

Karatas and Hirsa (2021), for example, present a Machine Learning and Deep Learning based methodology for a takeover success prediction problem. Various ML techniques are firstly applied for data pre-processing purposes on the financial numerical dataset, successively, a classification model is built using Feedforward Neural Network (FFNN) in order to predict the success/failure of M&A (Mergers and Acquisitions) deal, after there has been an announcement. As a baseline, the authors calculate and integrate (to the previously calculated FFNN) sentiment scores on a categorical dataset with rumors and news on the announcement of the deal (so, before the deal), using LSTM autoencoder.

The paper is very interesting; however, it is limited as it presents only a few models for the prediction of the deal and dwells on the financial aspects of M&A. Moreover, the results are underwhelming: the model yields a 62% accuracy and it is shown that the use of sentiment scores in the classification framework does not improve the overall performance. The project would benefit of a more variety of ML models to compare and contrast the results, as implemented in this study, and the sentiment analysis would benefit a more in depth and sophisticated NLP approach. For this purpose, this study applies rule-based Sentiment Analysis using VADER (Valence Aware Dictionary and Sentiment Reasoner) on news and headers preceding the announcement. Finally, as they appear to be able to help predict a deal announcement, the results will be analyzed together with the stock price variables.

The concept that takeover rumors and news affect the stock prices and can be used to help predict an imminent announcement, as already mentioned, has been first been investigated in financial and economic studies. The article written by Pound and Zeckhauser (1990) is relevant to this study because it explains in details, and through real-life economic examples, how takeover rumors and news affect the stock prices.

The first important finding is that the market reacts to published takeover rumors efficiently, correctly assessing the average probability that the rumor will be followed ultimately by a takeover bid.

The authors also examined the effects of takeover rumors on the share price of some target companies: it is discovered that there has been approximately 7% run-up (increase in the rumored targets' stock prices) over the 20 trading days before rumor publication. Additionally, it is calculated that 18 out of 42 firms announced a deal within a year of the rumor publication, but only two of them had the announcement within 50 days of the rumor publication. This evidence is consistent with the hypothesis that those close to the market, such as arbitrageurs, may make short-term profits trading on rumors as they unfold.

An additional work which analyses mergers and acquisitions from a financial and economic perspective is the one from Jarrell and Poulsen (1989). The authors confirmed, again, that the presence of rumors is the strongest variable explaining pre-announcement run-ups.

Although both these articles are from a financial background, the concepts are essential for this study which aims to apply modern ML and NLP techniques to confirm these important findings and to gain additional insights from the results.

Another data science paper relevant to this study, is the one published in (Refinitiv, 2022). The article investigates and evaluates the return for investors (which is the unanticipated profits - or losses - on investments, throughout a period) pre and post M&A announcement, reinforcing the concept that these can be used to predict an imminent announcement and to identify companies target of an M&A.

First of all, the author tests different methodologies to build a predictive model for Mergers and Acquisitions (M&A) target company identification. These methodologies include SVM, Decision tree and Random Forest algorithms, but the author is able to identify that logistic regression generalizes better and provides higher accuracy. After a variable's selection, the study then uses the logistic regression model to classify target versus non-target companies (clustering on the data, is also used to optimize the model accuracy). The return for investor variable, for both target and non-target companies, is then calculated and evaluated (it appears that there is an increase).

In regards to the results, the model prediction is able to identify only 65% of the targets company and 60% of non-targets and the results from the estimation of investment returns are consistent with the abnormal return. Given the underwhelming results of the predictions, the article would benefit more model's consideration.

The article is very interesting and very well executed and it focuses on both pre and post announcement. This project, on the other hand, focuses on the pre-announcement phase only, due to this being the more useful phase for risk arbitrage: the act of purchase of stock in a corporation that appears to be the target of an imminent takeover in the hope of making large profits if the takeover occurs.

There are additional studies already confirming that logistic regression is the best performing model for this type of numerical binary classification, so contrary to this article, this study uses the logistic regression as baseline against the other predictive models, as the aim is to improve the models' accuracy, against what we already know is the best performing model.

The article's analysis on returns of investment concludes that the returns are also dependent on other factors, like individual stock returns, market timing, and the stock holding period, making the abnormal return generation, with the aim of identifying target companies, an extremely challenging job. In this regard, this study analyzes in more detail, what other specific variables can help predict an imminent announcement. Through a more sophisticated variable selection (a triangulation of results between, Pearson's correlation coefficient, t-test analysis on Pearson's (r) results and Spearman's rank coefficient, plus filter methods) the variables with largest impact will be selected and analyzed.

Finally, the study lacks an integration of NLP methodologies to support the results, as opposed to this project where sentiment analysis on news articles generated before an announcement, together with the share prices will be analyses to hopefully maximize the predictive power of the models.

To overcome the lack of sentiment analysis, a second article is written by the same author where, to extend on the Mergers and Acquisitions (M&A) predictive modelling previously calculated, a Natural Language Processing (NLP) based news sentiment variable, is calculated. The author wants to test if news sentiment derived by NLP has any significant contribution to M&A predictive modelling. The main hypothesis behind the news sentiment variable was the intuition that abnormal returns amid no or lower positive news sentiment environment could indicate an M&A announcement (Refinitiv, 2022).

For the news sentiment analysis, two BERT-based models, including FinBert and BERT-RNA, are used and the significance of variables derived by both of the models

through logistic regression, random forest, and XGBoost ML techniques, are compared. Each model is evaluated with and without news sentiment variable, to understand the effect of the variable on the results. The dataset isn't large enough to claim the robustness of the model's predictive power, and the best performing models, with an underwhelming 64% of accuracy are both the Random Forest and the XGBoost model, both calculated with sentiment variables (the first with BERT, the second with FinBert method).

The study would benefit of the training of more robust models on a larger dataset in order to achieve higher accuracies values and to be able to confirm the project hypothesis. Contrary to the article, the study performs VADER and TextBlob sentiment analysis on headlines and rumors before deal announcement, and evaluates the results together with the share prices variable with the hope of gaining more robust results to disprove or prove the same hypothesis.

The paper published in DeepAI (2021) is another example of merger and acquisitions predictions where machine learning techniques are used; the study utilizes the GraphSAGE machine learning algorithm to predict mergers and acquisitions specifically of enterprise companies.

Graphs are defined as a set of nodes and edges representing relationships between entities. Given the abundance of data sources and algorithmic decision making within financial data science, graph-based machine learning offers a performant, yet non-traditional approach to generating alpha, hence the decision of the authors to choose this model (DeepAI, 2021).

The model outputs a very good 81.79% accuracy on the validation dataset. Although this result is very promising, the paper is a simple binary prediction model, it lacks of alternative approaches and will benefit for some more analysis and the use of multiple algorithms to compare and contrast the results.

Still in the application of graphs for the study of M&A predictions, Ning and Chen (2009) constructed a coal enterprises M&A efficiency assessment system for the Chinese coal enterprises and built a M&A efficiency prediction model based on wavelet neural network. Apparently, many coal enterprises in China have implemented merger and reorganization, in the hope they can be bigger and stronger to withstand market risk.

So, the paper investigates the data in the interval in between 2004-2008 of coal enterprises M&A, applying the WNN model, which organically combines the wavelet analysis and neural network theories: the nice time frequency localization nature of wavelet, together with strong nonlinearity, fault-tolerance, self-adaptability and self-learning features of the neural network (Ning and Chen, 2009). The model yields a very small error between the actual value and the wavelet neural network: the biggest error was registered within 3%.

Overall, the model application, while limited to only one model, is quite interesting. However, there is not as of now, a general accepted framework for applying WNs: they are a new class of networks and the literature around this model is also quite limited, making the results quite unreliable.

Another paper which investigates the subject through the use of graphs, is the one implemented by Li, Shou, Treleaven and Wang (2021): the authors develop a graph neural networks (GNN) model for Mergers and Acquisitions (M&A) prediction, which aims to quantify the relationship between companies, their founders, and investors.

The utilized GNN model, takes full advantage of the relationship data to expand feature dimension and improve the prediction result. Interestingly, the dataset used for the analysis is formed of textual data (information on company, person, financial organization, product, service-provider and the article data of companies); so this M&A prediction solution integrates with the topic model for text analysis, advanced feature engineering (which is the process of using domain knowledge to extract features from raw data via data mining techniques) and uses 5-fold Cross-Validation to find the optimal parameters to boost the GNN (Li, Shou, Treleaven and Wang, 2021). The research uses Logistic Regression (LR), XGBoost and Support Vector Machine (SVM) as baseline models. The approach achieves a true positive rate of 83% with a low false positive rate 7.8%, which performance is better than the previous benchmark record 70.9%/10.6%.

The use of textual analysis to analyze and predict M&A activity is also expanded by Moriarty, Ly, Lanv and McIntosh (2019): by collecting data from a corpus of historical Business Description and Management Discussion and Analysis (MD&A) from 1994 to 2018, the authors use natural language processing (NLP) techniques to vectorize and explore each filing’s textual data, in order to cluster firms by industry and identify keywords suggestive of upcoming M&A activity.

In more detail, a logistic regression classifier was trained to distinguish between firms that are most likely be targets or acquirers.

The classification models achieve an accuracy of 72% and 77% for targets and acquirers respectively. With a precision rate of 8% for targets and 79% for acquirers.

Clustering is then performed to identify subgroups within the categories of targets and acquirers, this then allows the authors to understand which topics and terms are most indicative of each category and identify relations across subgroups.

Lastly, a website that enables users to input a company or industry and receive useful information such as the probability of that company being a target or acquirer as well as potential partners in a deal, was deployed (Moriarty, Ly, Lanv and McIntosh, 2019). The authors also provided a variety of different ways for investors to visualize the results.

In a similar direction, Jiang (2021) takes advantage of the large amount of publicly available filings through the Securities Exchange and uses text data to try and predict merger activity. After transforming the text into arrays of words, the author further reduces the dimensionality and minimize over-fitting, by using LASSO techniques to select words that are promising prediction variables. To select the best model and the best performing regressor variables, cross-validation techniques are used.

The author came up with a model that has 85% accuracy compared to a 35% accuracy using the "bag-of-words" method to predict a company's likelihood of merging from words alone on the same period's test data set.

While all these three projects take an interesting approach on analyzing M&A prediction through the use of textual data, generally it is quite hard to predict takeovers accurately only on the basis of textual data.

Moreover, textual dataset on takeovers are not easily retrievable, financial data, on the other hand, is more immediately available. Furthermore, the interpretability of the variables is different between numerical and text regression and while, individually, they both offer different insight on the problem, text supplementing financial variables is the more powerful combination and this is exactly what this study is aiming to do.

DATASETS

The datasets used in this project were retrieved from Github and published by Refinitiv-API-Samples (2022). These were previously used in another M&A prediction problem (Refinitiv, 2022) and built by the author, using an M&A Advanced Search system of Refinitiv Workspace. Because the data contained in the datasets (both numerical financial and textual) is of appropriate size and format, they were deemed fit for the purpose of this project. In particular, the financial datasets presented financial variables which described debt and liquidity: variables like 'Total Debt' or 'Free Cash Flow' are essential in the determination of the likelihood of a deal completion, and therefore the identification of M&A Target and Non-Target companies.

Additionally, the textual dataset was in a very practical format: the headlines were summarized into a singular column and, overall, the text was brief but informative.

The datasets were provided in the format of 3 Excel files. Table 1. summarizes these files' details.

Dataset Name	Details
target_data	209 x 22 Dataset with financial numerical variables of US and UK companies, target of M&A deals in the period from 2020-09-15 to 2021-11-15.
peer_data	7394 x 21 Dataset with financial numerical variables of US and UK companies, which have not been target of M&A deals in the period from the beginning of 2021 until the dataset creation date (December 2021). Compared to the previous dataset, there is one less column as there is no column for the data end date collection.
headlines	298134 x 5 Dataset with news headlines for both target and non-target companies. These headlines were collected from 30 to 5 days before the deal announcement. Column 'RIC' corresponds to column 'Instrument' of the target_data and peer_target datasets.

Table 1. Datasets details

In regards to the Target and Non-Target datasets, a description of the financial variables, which later will support the understanding of the results, is presented in the summary Table 2. below.

Column/Variable Title	Description
Instrument	Referencing code system used to identify the company.
AD-30	Initial date of collection of the values.
AD	End date of collection of the values.
Gross profit margin	Is the ratio that indicates a company's sales performance: the percentage of revenues left after the cost of goods sold is deducted.
Net income after minority interest	Is the net income after the minority interest has been deducted. The minority interest is the portion of a company not held by the parent company: a company which owns more than 50% of the shares (Investopedia, 2023).
Total Capital	Is the money available for the business itself to pay daily interactions and fund its own expansion.
Operating Margin - %, TTM	It measures the business's profit on a dollar of sales, after costs of production (wages, material etc.) deduction, but before Taxes or Interests deduction. It is evaluated in the past 12 consecutive months of a company's performance data (Trailing 12 months - TTM) (Investopedia, 2023).
Free Cash Flow:	The money a business generates from its normal business operations before interest payments and after subtracting any money spent on capital expenditures (such as rent, equipment etc.).
Revenue from Business Activities - Total	Is the operating revenue a business creates from its primary business activities. The same definition also applies to: **Revenue from Business Activities - Total. 1, 2 & 3**.
Current Ratio	It measures a company's ability to pay short-term obligations or those due within one year.
Price to book value per share (Daily Time Series Ration)	Is a method used to calculate the per-share book value of a company (which is the company's total assets minus the total liabilities), based on common shareholders' equity in the company. This variable, together with the next, are considered our share price.
Price to sales per share (Daily Time Series Ration)	Is the financial ratio that measures the total revenue earned per share over a specific time period.
Earnings before interest Taxes Depreciation & Amortization	Also called EBITDAR, adjusts net income by removing certain costs. Is a measure used to capture a company's financial performance.
Enterprise value (Daily Time series)	Is a measure of the value of a stock that compares a company's enterprise value to its revenue.
Total Shareholders' Equity incl Minority Intr.& Hybrid Debt	The shareholder equity ratio measures how much of a company's assets have been generated by issuing equity shares rather than by taking on additional debt. The lower the ratio result, the more debt a company has used to pay for its assets.
Debt - Total	It defines how much debt a company owns, compared to its assets.
Net Debt per Share	It measures the value of a government's debt expressed in terms of the amount attributable to each citizen, under the government's jurisdiction (Investopedia, 2023).
Cash & Cash Equivalents - Total	It refers to financial liquidity (cash) and to how easily assets can be converted into cash.

Table 2. Target and Non-Target datasets details

These financial variables can be divided in 5 main macro financial categories:

- Liquidity: which can be described as to how efficiently, cash can be obtained to pay bills and other short-term obligations. In the dataset, the variables that describe liquidity are: 'Current Ratio', 'Cash & Cash Equivalents – Total' and 'Total Capital.

- Leverage: refers to the use of debt (borrowed funds) to amplify returns from an investment or project. The variables that describe leverage are: ‘Net Debt per Share’ and ‘Debt – Total’.

- Company market value: key category to compare the financial position of companies against their competitors or the market overall. It included: ‘Total Shareholders' Equity incl Minority Intr & Hybrid Debt’, ‘Enterprise Value (Daily Time Series)’, ‘Price to book value per share (Daily Time Series Ration)’ and ‘Price to sales per share (Daily Time Series Ration)’.

- Company business efficiency: ‘Earnings before interest Taxes Depreciation & Amortization’, ‘Revenue from Business Activities - Total, 1, 2 and 3’.

- Growth-resource ratios: ‘Free Cash Flow’, ‘Operating Margin - %, TTM’, ‘Gross profit margin and Net income after minority interest’.

Data Pre-Processing and Cleaning

The 3 datasets went through extensive pre-processing and cleaning, in order to make them fit for purpose. In more detail, for the Target dataset, rows containing value zero were removed and an additional Target/Non-Target column labelled with values 1, was added not only for the purpose of differentiating the dataset (from the Non-Target dataset), but also for the purpose of making it classification ready. The pre-processing and cleaning steps, created a reduced size dataset of 202 x 23.

Similarly, for the Non-Target dataset (called peer_data), rows containing value zero were removed and an additional Target/Non-Target column, labelled 0 was added, making it of reduced size 7000 x 22.

These 2 numerical financial datasets were then merged together, forming a unique, but imbalanced (towards Non-Targets), numerical financial dataset of size 7202 x 23, which was used as the unique financial dataset in the rest of the project.

Lastly, the textual data in the headlines dataset was cleaned of punctuation and special characters; to obtain homogeneity all the characters were also lowered, and the stop words removed. Figure 2. shows the frequency distribution of the most frequent words in the dataset. Finally, word tokenization was performed on the dataset and, to make it ready for sentiment analysis, unnecessary columns deleted as well.

There were multiple company referencing code duplications in column ‘RIC’, so the dataset was grouped by this particular column and the text merged together. Following these steps, the dataset was reduced to size 1057 x 1.

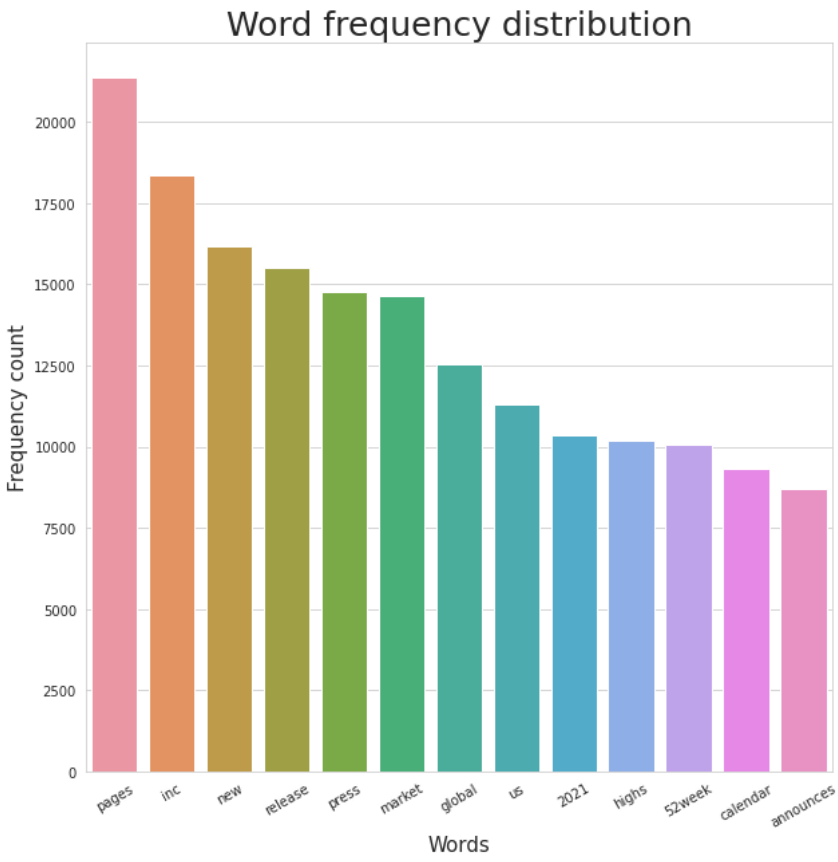

Figure 2. Most frequent words in the Headlines dataset

Variables Selection

The more significant and impactful variables in the cleaned 7202 x 23 financial dataset, were selected following a very detailed variable selection process.

The results of this process, not only were used in the application of the various machine learning models, they also allowed us to answer the second research question: which variables are the most important and impactful in making M&A prediction?

Of the 23 columns in the dataset, all the textual columns (‘Target/Non-Target’, ‘Instrument’, ‘AD-30’, ‘AD’) together with the index were removed, leaving a total of 18 strictly numerical financial variables.

Pearson correlation coefficient (r) was then used to identify and remove highly correlated variables, in order to avoid overfitting and generally improve the models’ performance.

Pearson is a measure of linear correlation between two variables, it measures a normalized measurement of covariance (a value between -1 and 1) that shows the strength and direction of the relationship between two variables. Pearson is a good measure when the variables are quantitative, normally distributed and the relationship is linear.

The scatter matrix depicted in Figure 3., however, shows the normalized dataset variables: while some of the relationship between variables are linear, overall, the variables did not appear to be normally distributed.

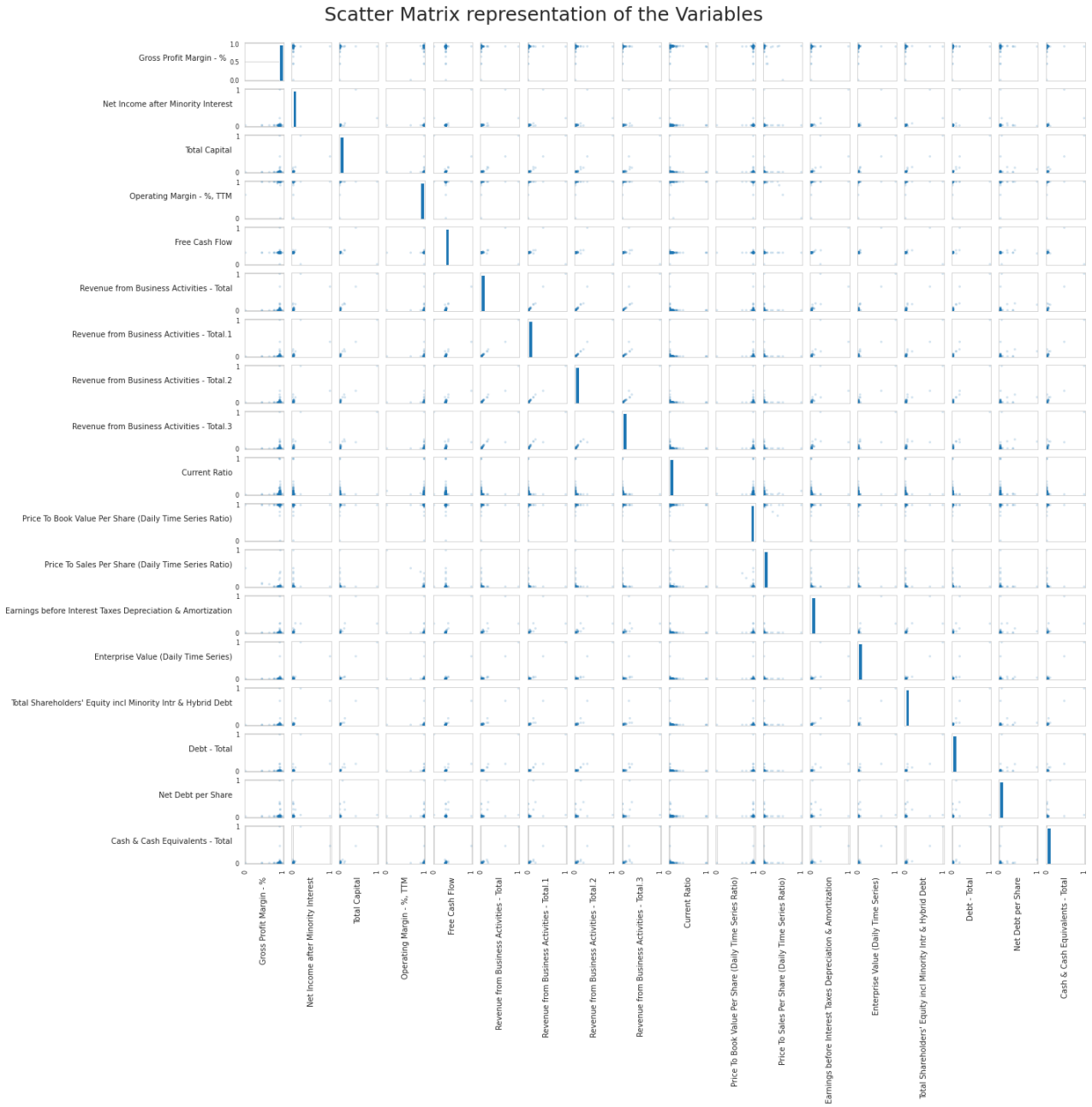

Figure 3. Scatter Matrix representation of the financial variables

To perform this dimensionality reduction task, it was then decided to perform a triangulation of results from Pearson, Spearman's rank correlation coefficients and t-test analysis on Pearson's (r) results.

Spearman's rank correlation coefficient is another statistical test that examines the degree to which two data sets are correlated; it is a good choice when the variables are not

normally distributed (the case for our dataset) and the relationship between the variables is non-linear and monotonic.

On the other hand, Pearson correlation coefficient is also an inferential statistic: it can be used to test statistical hypotheses (Turney, 2022). Specifically for this study, it was used to test whether there was a significant relationship between two variables.

Statistical significance (indicated by the probability p) indicates whether there can be confidence in a relationship between two variables (Turney, 2022).

The t statistic is calculated in the equation that follows:

$$t = \frac{r}{\sqrt{\frac{1 - r^2}{n - 2}}} \qquad \textit{(Eq. 1)}$$

Where the value r is the Pearson correlation coefficient, while n is the sample size.

On top of this triangulation evaluation, filter methods were applied to select the most performing variables. Filter methods are a type of feature selection method that works by selecting features based on some criteria prior to building the model. They do not involve the use of a model and are therefore, computationally inexpensive and flexible to use before the application of any type of machine learning algorithm.

More specifically for this project, the dataset was checked for constant features and duplicated features filter methods.

RESULTS & EVALUATION - Second research question answered

No constant features nor duplicated features were identified in the dataset. Figure 4. shows the results of Pearson's correlation coefficients on the 18 variables, represented in a Heatmap: the most correlated variables appear to be positively correlated.

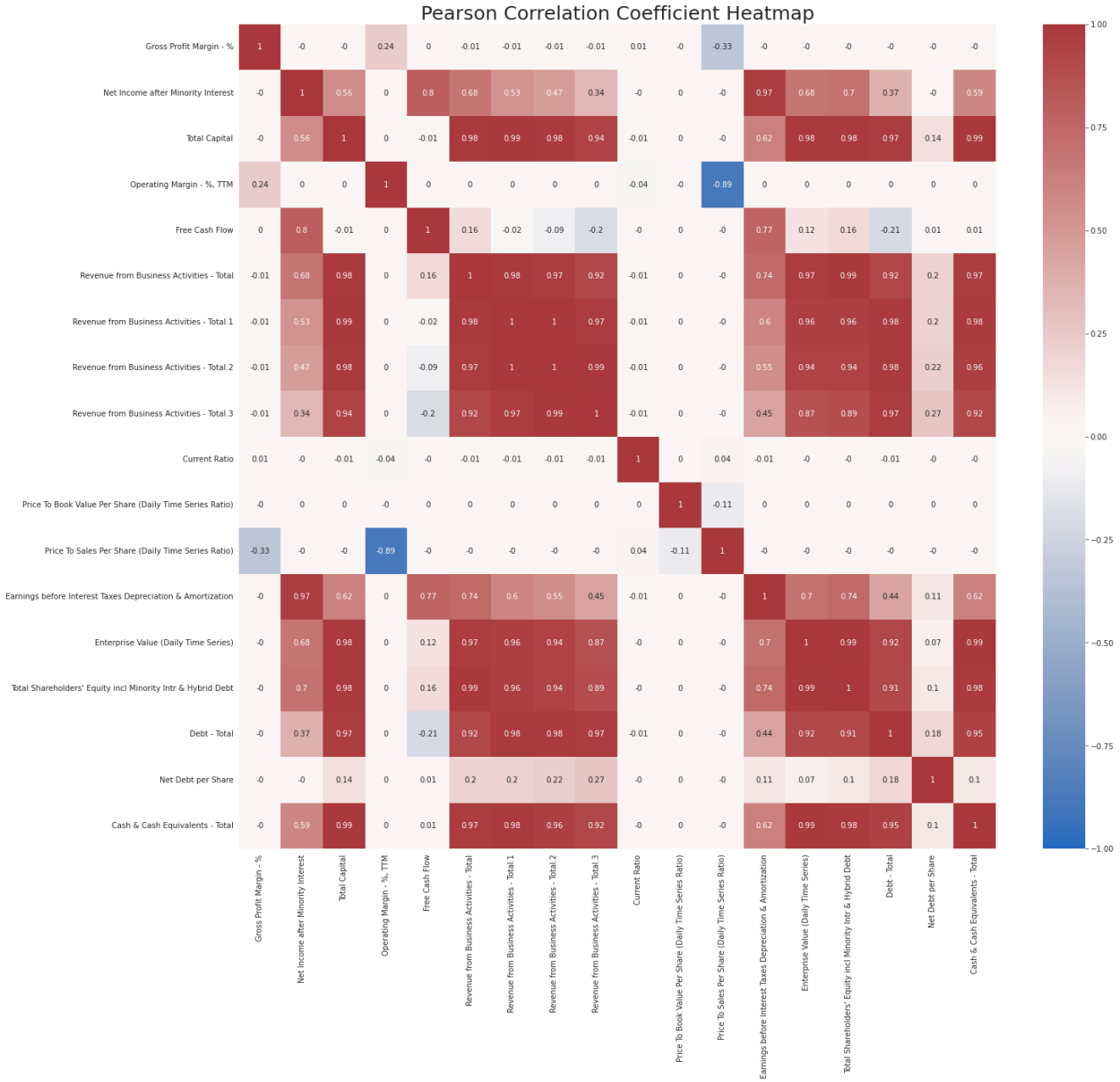

Figure 4. Pearson's correlation coefficients of the financial variables in a Heatmap

Similarly, Figure 5. below shows Spearman's correlation coefficients in a Heatmap.

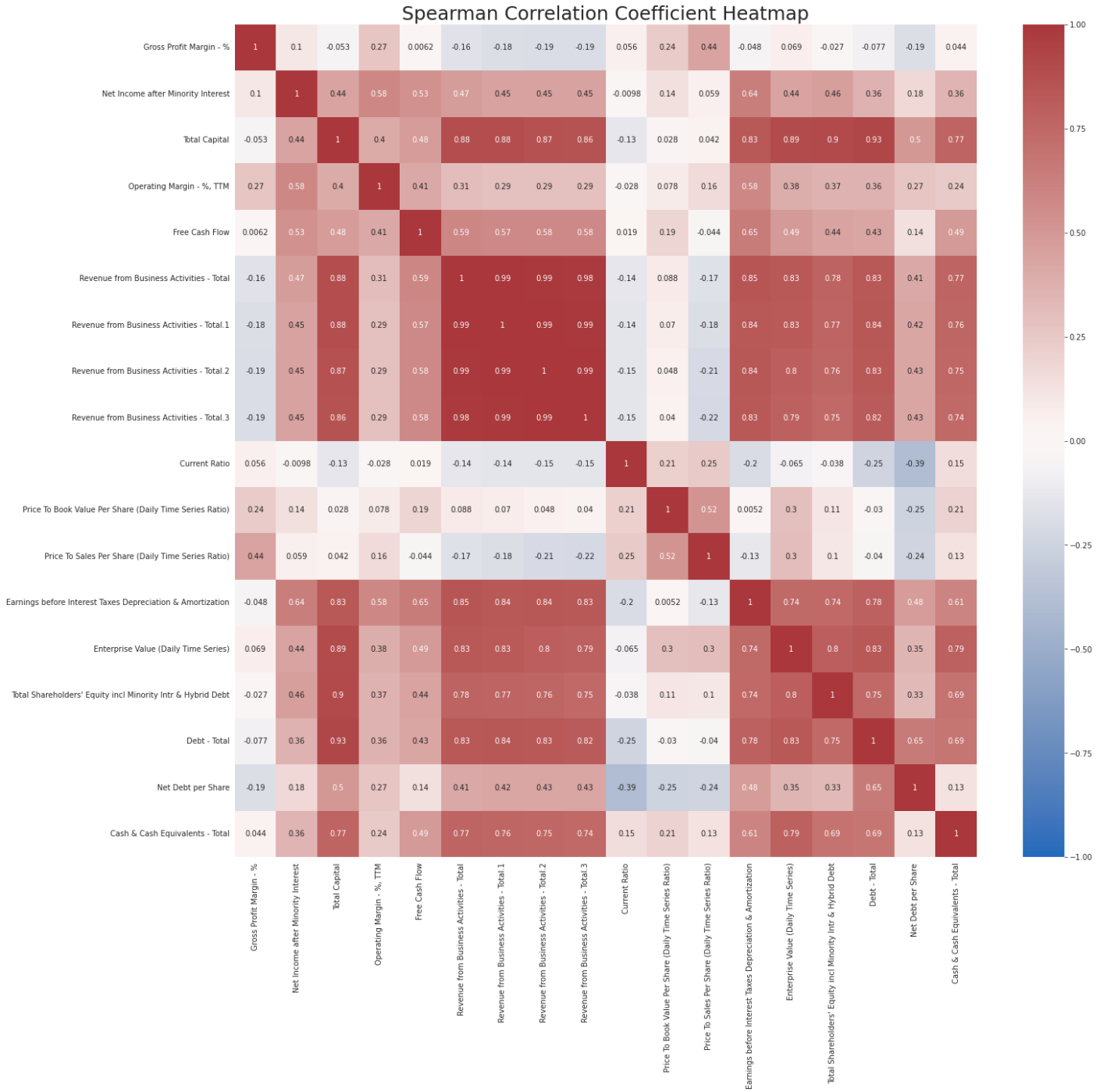

Figure 5. Spearman's correlation coefficients of the financial variables in a Heatmap

Again, the most correlated variables appear to be positively correlated. In more detail, Table 3. below, summarizes the results for both Pearson and Spearman: it shows the most positively or negatively correlated variables (positively correlated variables with a score of > 0.70 are in red; negatively correlated variables with a score of < -0.70 are in blue).

Pearson	Spearman
Cash & Cash Equivalents - Total	Cash & Cash Equivalents - Total
Total Shareholders' Equity incl Minority Intr & Hybrid Debt	Total Shareholders' Equity incl Minority Intr & Hybrid Debt
Enterprise Value (Daily Time Series)	Enterprise Value (Daily Time Series)
Debt - Total	Debt - Total
Revenue from Business Activities - Total.2	Revenue from Business Activities - Total.2
Revenue from Business Activities - Total.1	Revenue from Business Activities - Total.1
Earnings before Interest Taxes Depreciation & Amortization	Earnings before Interest Taxes Depreciation & Amortization
Revenue from Business Activities - Total.3	Revenue from Business Activities - Total.3
Revenue from Business Activities - Total	Revenue from Business Activities - Total
Free Cash Flow	/
Price To Sales Per Share (Daily Time Series Ratio)	/
Total Capital	Total Capital
Operating Margin - %, TTM	/

Table 3. Spearman and Pearson most highly correlated variables

What is interesting to point out is that Spearman's did not identify any negative correlated variables, while Pearson, managed to find that both the 'Price To Sales Per Share (Daily Time Series Ratio)' and 'Operating Margin - %, TTM' were negatively correlated.

Pearson was also able to identify another highly correlated variable that Spearman did not identify: 'Free Cash Flow'.

These 3 variables, not presented in Spearman, appeared to be the less correlated with other variables. Due to this discovery, these 3 variables were kept in the dataset.

Moreover, to decide which of the rest of the variables to actually keep, the t-test scores were used to establish if the Pearson's r statistic differed significantly from zero.

Table 4. shows the t-test results:

	Target/Non-Target	Total Capital	Revenue from Business Activities - Total	Revenue from Business Activities - Total.1	Revenue from Business Activities - Total.2	Revenue from Business Activities - Total.3	Earnings before Interest Taxes Depreciation & Amortization	Enterprise Value (Daily Time Series)	Total Shareholders' Equity incl Minority Intr & Hybrid Debt	Debt - Total	Cash & Cash Equivalents - Total
T_test	inf	-0.356457	-0.474604	-0.457526	-0.468369	-0.518286	-0.423969	-0.340538	-0.365563	-0.328449	-0.357354
p-value	0.0	0.721509	0.635084	0.647307	0.639535	0.604275	0.671601	0.733461	0.714702	0.742582	0.720837

Table 4. T-test results with p-value on the most highly correlated variables identified with both Pearson and Spearman

From this test, the only variable which was kept in the original dataset, was ‘Debt-Total’: with a p value of .7425, there was a high 74.25% chance that the relationship (the high correlation), occurred by chance.

Since this p-value was not less than .05, we failed to reject the null hypothesis: with a high p value and lower T-test, the relationship was deemed not statistically significant, so no confidence in the relationship of the variables was shown and the null hypothesis could not be rejected and the alternative hypothesis not supported. The decision to keep the variable was then made.

The threshold for the p-value was set at .7425. All other variables were deemed statistically significant (their score was below this level) and therefore deemed highly correlated and so removed from the dataset.

The results of the triangulation had a total of 9 variables removed from the final dataset, and a total of 1 variable kept in the original dataset which was now made up of 9 variables.

The second research question: which variables are the most important and impactful in making M&A prediction? was now then answered: as shown in Table 5., these were the most significant and impactful variables which could help making M&A predictions.

Most significant variables
Gross Profit Margin - %
Net Income after Minority Interest
Operating Margin - %, TTM
Free Cash Flow
Current Ratio
Price To Book Value Per Share (Daily Time Series Ratio)
Price To Sales Per Share (Daily Time Series Ratio)
Debt - Total
Net Debt per Share

Table 5. Variable selection outcome: the most significant variables

We then ended up with variables which describes:

- Leverage: 'Net Debt per Share', 'Debt - Total'
- Growth-resource ratios: 'Gross Profit Margin - %', 'Free Cash Flow', 'Operating Margin - %, TTM', 'Net Income after Minority Interest'

- Liquidity: 'Current Ratio'
- Company market value: 'Price To Book Value Per Share (Daily Time Series Ratio)', 'Price To Sales Per Share (Daily Time Series Ratio)'

This final, feature-selected financial dataset, inclusive of only the most significant variables, was used in the following methodologies. More specifically, in Methodology I and in Methodology II Approach B.

CHAPTER 3 – METHODOLOGIES

METHODOLOGY I – Models with No sentiment scores

Baseline model: Logistic Regression

To meet the aim of this project, to predict the success of M&A deal completion and identify the best performing ML model, a binary classification task (Target vs. Non-Target) was implemented on the feature selected financial dataset.

The baseline against which the performance of the chosen algorithms was going to be compared to, was the logistic regression, as it appeared to be the best performing model, also in previous studies. The logistic regression model is well suited for numerical databases, being it is easy to implement and very efficient to train.

Before applying the model, the financial dataset was divided into a test and train split; the test size was set at 30% of the full dataset and the random state set at 11.

Other classification models

Against the baseline models, the following algorithms were implemented: Random Forest, Neural Network (NN), Decision Tree and Support Vector Machine (SVM).

Optimization of each parameter was achieved, by setting: for the Random Forest the number of trees to be used in the forest at 100; for the NN, the number of neurons in the ith hidden layer was set at size 13 for all parameters and the maximum number of iterations at 500.

For the decision tree, entropy was the criteria used in the classifier as it measures the impurity in each attribute by specifying randomness in the data.

Finally, in regards to the Support Vector Machine (SVM) the LinearSVC (Linear Support Vector Classification) classification model was implemented. LinearSVC is written in liblin which supports the loss functions and penalties; it can handle a larger number of samples better and runs quicker (Simplilearn, 2022).

Models' evaluation metrics

As mentioned earlier, the financial dataset is highly imbalanced: only around 2.88% of the rows refers to target companies.

Highly imbalanced datasets usually output high accuracy. For this reason, this binary classification problem was evaluated also with other evaluation metrics: F-Score, Precision and Recall.

Below, the equations 2,3,4 and 5 to calculate these metrics.

TP = True Positive

TN = True Negative

FP = False Positive

FN = False Negative

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + TP} \quad \textit{(Eq.2)}$$

$$\text{Precision} = \frac{TP}{TP + FP} \quad \textit{(Eq. 3)}$$

$$\text{Recall} = \frac{TP}{TP + FN} \quad \textit{(Eq.4)}$$

$$\text{F1-score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \quad \textit{(Eq. 5)}$$

Moreover, in order to improve the models' results, and help with overfit, nested cross-validation was performed.

Finally, a confusion matrix (depicting text label vs. predicted labels) was also created to evaluate the performance of each classification model.

RESULTS & EVALUATION – First research question answered

On a financial dataset with a test and train split of 30% to 70%, the models were firstly applied to the training data and then predicted on the test data.
Table 6. summarizes the results.

	Accuracy (%) - mean, sd	F-score (%) - mean, sd	Recall (%) - mean, sd	Precision (%) - mean, sd	Cross validation
Logistic Regression	94.20 % ± 0.0125	97.23 % ± 0.033	97.75% ± 0.028	96.73 % ± 0.040	Yes
Random Forest	100.00 % ± 1.000	100.00 % ± 1.000	100.00 % ± 1.000	100.00 % ± 1.000	No
NN	95.82 % ± 0.011	98.18 % ± 0.000	99.71 % ± 0.000	96.70 % ± 0.000	Yes
Decision Tree	100.00 % ± 1.000	100.00 % ± 1.000	100.00 % ± 1.000	100.00 % ± 1.000	No
SVM	100.00 % ± 1.000	100.00 % ± 1.000	100.00 % ± 1.000	100.00 % ± 1.000	No

Table 6. Models results on the financial dataset (no sentiment scores)

With a F-score, Recall, Precision and Accuracy of a perfect 100%, the Random Forest, Decision Tree and SVM models were the best performing models. The logistic regression, although with a Precision of 96.73 % was the least accurate model, even after cross validation, followed by the NN model with a Recall and F-score of 99.71 % & 98.18 % respectively.

The results in this table allowed us to answer the first research question: using the logistic regression as a baseline model, what is the next best performing classification model that can predict the success of M&A deal completion? Can this model outperform the logistic regression?

Using the logistic regression as a baseline model the next best performing classification models that can predict the success of M&A deal completion are the Random Forest, Decision Tree and SVM models. They outperformed the logistic regression by + 2.77 % on F-score, + 2.25 % on Recall, + 3.27 % on Precision and + 5.80 % on Accuracy.

METHODOLOGY II – Models with sentiment score

Sentiment analysis with TextBlob: Baseline Model

On the 1057 x 1 pre-processed headlines textual dataset, sentiment analysis was performed with the chosen baseline model, TextBlox.

TextBlob uses semantic labels that help with fine-grained analysis; it returns the polarity and subjectivity scores of a sentence.

Polarity lies between [-1,1]; -1 defines a negative sentiment while 1 defines a positive sentiment; subjectivity, on the other hand, lies between [0,1] and it quantifies the amount of personal opinion and factual information contained in the text. A high subjectivity means that the text contains personal opinion rather than factual information (Shah, 2020).

The results of TextBlob sentiment analysis on the headline dataset are depicted in Figure 6. below:

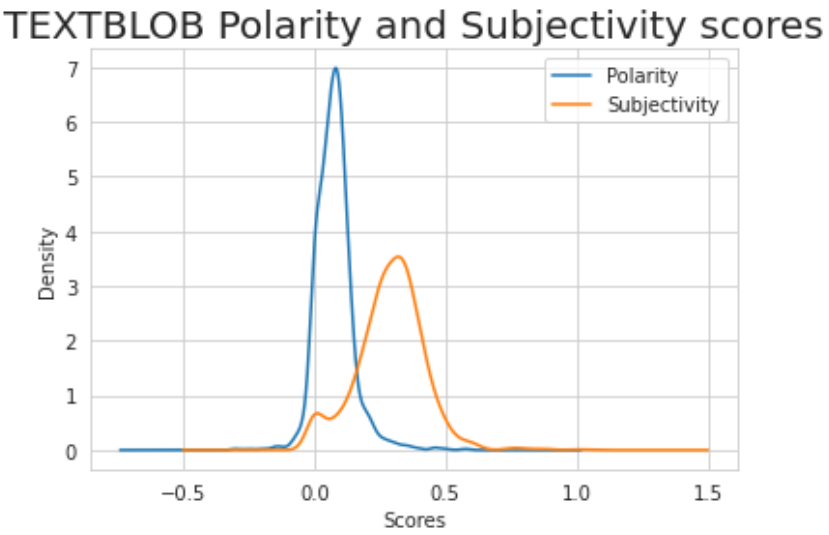

Figure 6. TextBlob Polarity and Subjectivity scores on the headline's dataset

The polarity seems to be majorly distributed around neutrality, 0 while the subjectivity seems relatively high for scores around neutrality and towards positivity.

Sentiment analysis with VADER

Sentiment analysis with VADER (Valence Aware Dictionary and Sentiment Reasoner) was executed as a model against which comparing the baseline model. VADER is a lexicon-based technique that specifies the polarity of the sentiment; it also provides scores which detects the intensity of a particular emotion. Most notably, the library provides a compound polarity score which is a metric that calculates the sum of all the lexicon ratings and normalizes them between -1 and 1; with -1 being extremely negative and 1 being extremely positive (Buxton, 2022).

The results of VADER sentiment analysis on the headline dataset, are depicted in Figure 7. below: neutrality scores seemed to be prevalent, followed by positive and then negative sentiment scores.

Figure 7. VADER Sentiment scores distribution

The results can be visualized better also in Figure 8. below, which shows the counts of the sentiment occurrences.

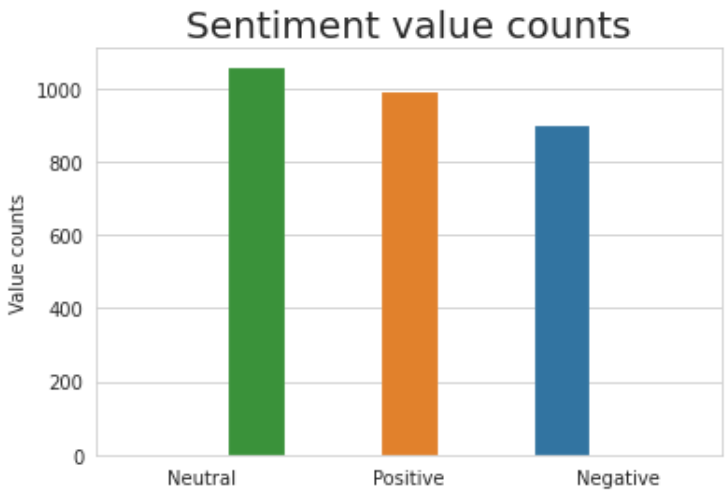

Figure 8. VADER Sentiment value counts

Again, neutrality and positivity seemed to be the sentiments majorly detected on this dataset. In regards to the compound scores, Figure 9. below depicts the scores in a line graph.

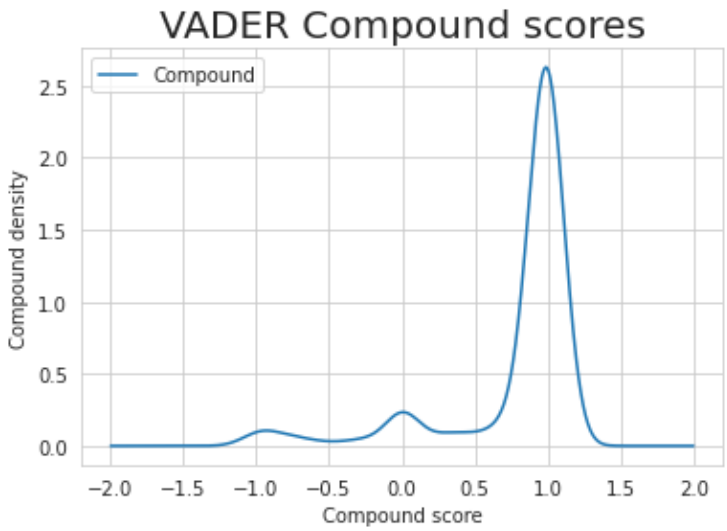

Figure 9. VADER Compound Scores

The compound scores seem to be at its highest, around value 1, towards positives, with a lesser pronounced peak around neutrality at 0.

Approach A: Share price with sentiment scores

In this approach, in order to predict the success of M&A deal completion, and to answer the third research question (can applying NLP sentiment analysis on rumors and news preceding an M&A announcement - while also taking into consideration the share prices change – actually help us predict an imminent deal announcement?), a binary classification task (Target vs. Non-Target) was implemented on the share prices together with the retrieved sentiment scores.

As previously mentioned, in our dataset, the share prices variables, correspond to the two variables: 'Price To Book Value Per Share (Daily Time Series Ratio)' and 'Price To Sales Per Share (Daily Time Series Ratio)'.

The 7202 rows in the variable selected financial dataset were regrouped with the share prices by the company name column 'RIC'; duplicated rows were also deleted and all unnecessary columns dropped.

The outcome contained only the share prices, which were merged with the Textblob Polarity and Subjectivity scores first and then, separately, with the VADER compound sentiment scores, creating a reduced, 1883 rows long dataset.

On these 2 datasets, all the machine learning models were re-applied.

RESULTS & EVALUATION – Third research question answered

Table 7. below, summarizes the results of this binary classification task.

	Price shares with VADER compound scores				Price shares with TextBlob Polarity and Subjectivity scores				Cross-validation
	Accuracy (%) - mean, sd	F-score (%) – mean, sd	Recall (%) – mean, sd	Precision (%) - mean, sd	Accuracy (%) - mean, sd	F-score (%) – mean, sd	Recall (%) - mean, sd	Precision (%) – mean, sd	
Logistic Regression	100.00 % ± 1.000	100.00 % ± 1.000	100.00 % ± 1.000	100.00 % ± 1.000	100.00 % ± 1.000	100.00 % ± 1.000	100.00 % ± 1.000	100.00 % ± 1.000	No
Random Forest	100.00 % ± 1.000	100.00 % ± 1.000	100.00 % ± 1.000	100.00 % ± 1.000	100.00 % ± 1.000	100.00 % ± 1.000	100.00 % ± 1.000	100.00 % ± 1.000	No
NN	99.73 % ± 0.0027	99.32 % ± 0.929	100.00 % ± 0.867	98.65 % ± 1.000	99.63 % ± 0.003	99.70 % ± 0.970	100.00 % ± 0.943	99.41 % ± 1.000	Yes
Decision Tree	100.00 % ± 1.000	100.00 % ± 1.000	100.00 % ± 1.000	100.00 % ± 1.000	100.00 % ± 1.000	100.00 % ± 1.000	100.00 % ± 1.000	100.00 % ± 1.000	No
SVM	100.00 % ± 1.000	100.00 % ± 1.000	100.00 % ± 1.000	100.00 % ± 1.000	100.00 % ± 1.000	100.00 % ± 1.000	100.00 % ± 1.000	100.00 % ± 1.000	No

Table 7. Models results for price share with VADER Compound scores and price share with TextBlob Polarity and Subjectivity scores

Logistic regression, Random Forest and Decision Tree and the SVM classifier all performed a perfect 100% score in all metrics for both datasets. For the NN model, the dataset with TextBlob appears to be slightly better performing than the dataset with VADER compound scores included; in fact, TextBlob surpasses VADER results on both F-score and Precision.

On the basis of this discovery, the third research question can be answered: applying NLP sentiment analysis on headlines collected before a M&A announcement (while also taking into consideration the share prices change) can actually help predict an imminent deal announcement.

The hypothesis that:

- High share prices, together with no or very low positive news sentiment, is an indication of an imminent M&A announcement.

Can be analysed in Figures 10. and 11., below, where only the normalized Target companies' data was taken into consideration to create these line graphs.

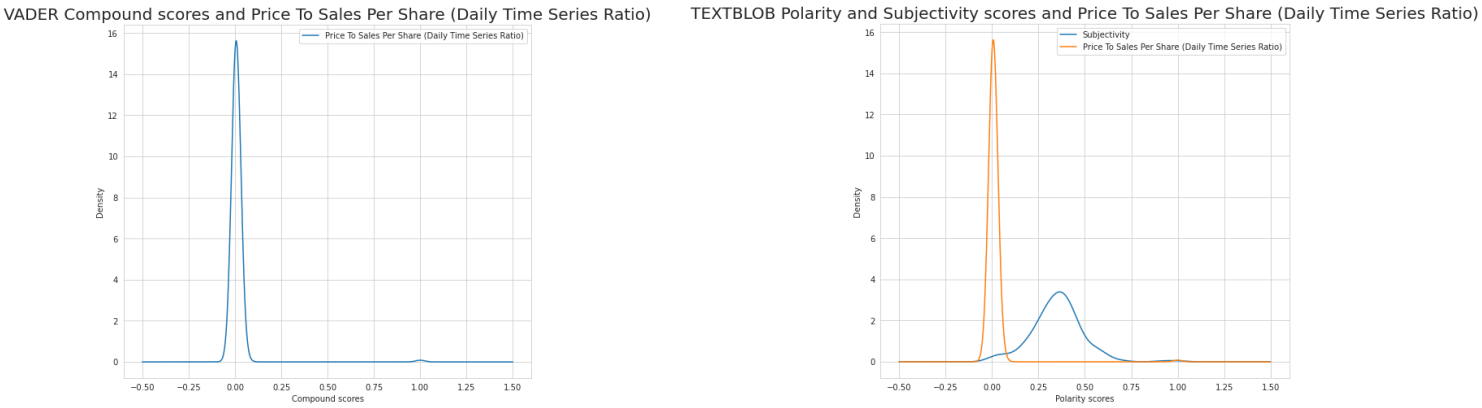

Figure 10. TextBlob and VADER scores analysed with variable Price to Sales per Share (Daily Time Series Ratio)

The two price share variables were analysed separately. The sentiment scores analysed both with TextBlob and VADER on the Price to Sales per Share (Daily Time Series Ration) for Target companies, are shown in Figure 10. As we can see, for both graphs, the price share values are distributed around Compound scores of neutralities 0 and Polarity scores of neutralities 0. The TextBlob figure also shows relatively low positive Subjectivity scores around 0.

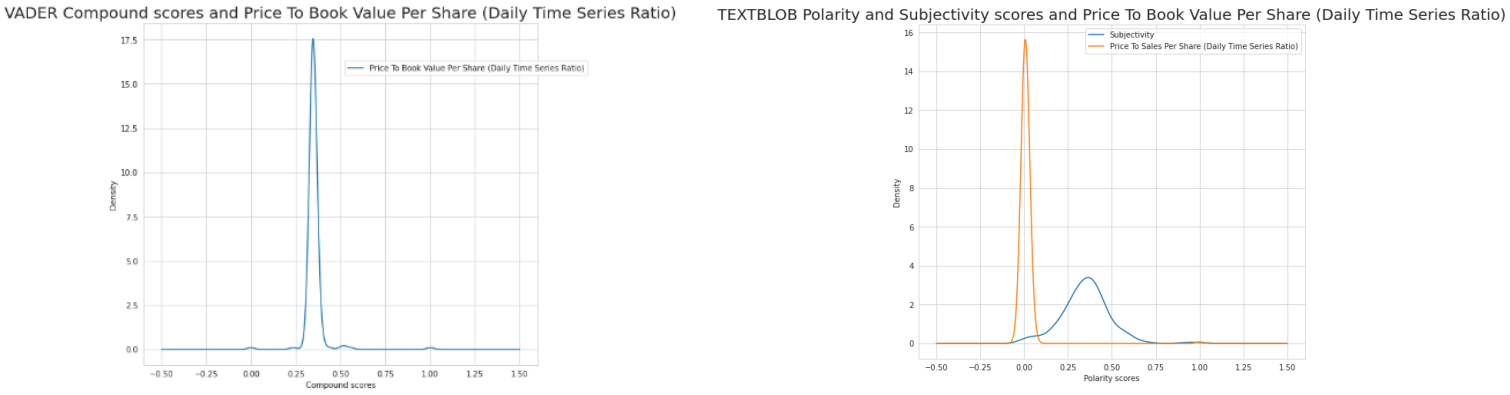

Figure 11. TextBlob and VADER scores analysed with variable Price to Book Value per Share (Daily Time Series Ratio)

In regards to the second variable, Price to Book Value per Share (Daily Time Series Ratio) the behaviour is similar: the Compound scores with VADER, shows the price share

distributed around lower positive scores; similarly, in the TextBlob Figure, the share price is distributed around Polarity 0 and lower positive Subjectivity scores.

According to these results, the hypothesis that high share prices, together with no or very low positive news sentiment, is an indication of an imminent M&A announcement can be confirmed.

Approach B: Financial dataset with sentiment scores

Lastly, in this approach, in order to answer the final research question, the variable selected full financial dataset together with the previously calculated sentiment scores (first with VADER then, separately, with TextBlob) were used for a binary classification task (Target vs. Non-Target) and all the machine learning models re-applied to further compare and evaluate the results and gain more insights on if and how, sentiment analysis could improve the results.

Similarly to what was achieved before, the 7202 rows variable selected dataset was merged with the Textblob Polarity and Subjectivity scores first and then, separately, with the VADER compound sentiment scores, by column 'RIC'. All duplicated rows were also deleted and all unnecessary columns dropped, creating two separate 1018 rows long dataset. On these 2 datasets, all the machine learning models were re-applied.

RESULTS & EVALUATION - Fourth and last research question answered

Table 8. below summarizes the results of this approach: the Logistic Regression scores, do not seem to be affected by the dataset used as the results match for both sentiment analyzer (Precision, Recall and F-score of 85.17 %, 94.27 % and 89.49 %).

	Full financial dataset with VADER compound scores				Full financial dataset with TextBlob Polarity and Subjectivity scores				Cross validation
	Accuracy (%) - mean, sd	**F-score (%) - mean, sd**	**Recall (%) - mean, sd**	**Precision (%) - mean, sd**	**Accuracy (%) - mean, sd**	**F-score (%) - mean, sd**	**Recall (%) - mean, sd**	**Precision (%) - mean, sd**	
Logistic Regression	79.86 % ± 0.022	89.49 % ± 0.033	94.27 % ± 0.022	85.17 % ± 0.062	79.86 % ± 0.022	89.49 % ± 0.033	94.27 % ± 0.022	85.17 % ± 0.062	Yes
Random Forest	100.00 % ± 1.000	100.00 % ± 1.000	100.00 % ± 1.000	100.00 % ± 1.000	100.00 % ± 1.000	100.00 % ± 1.000	100.00 % ± 1.000	100.00 % ± 1.000	No
NN	78.10 % ± 0.052	89.17 % ± 0.119	92.74 % ± 0.090	85.86 % ± 0.173	79.96 % ± 0.019	88.47 % ± 0.162	90.83 % ± 0.136	86.23 % ± 0.200	Yes
Decision Tree	100.00 % ± 1.000	100.00 % ± 1.000	100.00 % ± 1.000	100.00 % ± 1.000	100.00 % ± 1.000	100.00 % ± 1.000	100.00 % ± 1.000	100.00 % ± 1.000	No
SVM	100.00 % ± 1.000	99.80 % ± 0.988	99.61 % ± 1.000	100.00 % ± 0.977	100.00 % ± 1.000	100.00 % ± 1.000	100.00% ± 1.000	100.00 % ± 1.000	No

Table 8. Models results for financial dataset with VADER Compound scores and financial dataset with TextBlob Polarity and Subjectivity scores

Both Random Forest and Decision Tree models performed excellently with perfect 100% scores for both datasets; while the SVM model yielded perfect scores for the TextBlob analyzer, it scored slightly less in F-score and Recall with the VADER analyzer, precisely 99.80 % and 99.61 % respectively.

The Neural Network model seemed to be performing slightly better with VADER, with F-score and Recall of 89.17 % and 92.74 %, however, Precision and Accuracy on the TextBlob analyzer were slightly higher than in the VADER dataset: 86.23 % against 85.86 % and 79.96 % against 78.10 %.

To provide an answer to the fourth and last research question: can NLP sentiment scores with ML methodologies together, improve or outperform the results of the first methodology (models calculated without sentiment scores) ? - all the results (including the results of the first methodology - models with financial dataset only), were compared.

Table 9. below, summarizes these results.

	Full financial dataset with VADER compound scores				Full financial dataset with TextBlob Polarity and Subjectivity scores				Full financial dataset without sentiment scores			
	Accuracy (%) - mean, sd	**F-score (%) - mean, sd**	**Recall (%) - mean, sd**	**Precision (%) - mean, sd**	**Accuracy (%) - mean, sd**	**F-score (%) - mean, sd**	**Recall (%) - mean, sd**	**Precision (%) - mean, sd**	**Accuracy (%) - mean, sd**	**F-score (%) - mean, sd**	**Recall (%) - mean, sd**	**Precision (%) - mean, sd**
Logistic Regression	79.86% ± 0.022	89.49% ± 0.033	94.27% ± 0.022	85.17% ± 0.062	79.86% ± 0.022	89.49 % ± 0.033	94.27% ± 0.022	85.17% ± 0.062	94.20% ± 0.0125	97.23% ± 0.033	97.75% ± 0.028	96.73% ± 0.040
Random Forest	100.00% ± 1.000	100.00% ± 1.000	100.00% ± 1.000	100.00% ± 1.000	100.00% ± 1.000	100.00% ± 1.000	100.00% ± 1.000	100.00% ± 1.000	100.00% ± 1.000	100.00% ± 1.000	100.00% ± 1.000	100.00% ± 1.000
NN	78.10% ± 0.052	89.17% ± 0.119	92.74% ± 0.090	85.86% ± 0.173	79.96% ± 0.019	88.47 % ± 0.162	90.83% ± 0.136	86.23% ± 0.200	95.82% ± 0.011	98.18% ± 0.000	99.71 % ± 0.000	96.70% ± 0.000
Decision Tree	100.00% ± 1.000	100.00% ± 1.000	100.00% ± 1.000	100.00% ± 1.000	100.00% ± 1.000	100.00% ± 1.000	100.00% ± 1.000	100.00% ± 1.000	100.00% ± 1.000	100.00% ± 1.000	100.00 % ± 1.000	100.00% ± 1.000
SVM	100.00% ± 1.000	99.80% ± 0.988	99.61% ± 1.000	100.00% ± 0.977	100.00% ± 1.000	100.00% ± 1.000	100.00% ± 1.000	100.00% ± 1.000	100.00% ± 1.000	100.00% ± 1.000	100.00% ± 1.000	100.00% ± 1.000

Table 9. Models results compared: financial dataset with VADER and TextBlob vs financial dataset without sentiment scores

The Logistic regression seemed to be performing better without the sentiment scores. The models which scored a perfect 100% (Random Forest and Decision tree) constantly kept scoring a perfect score, nevertheless the dataset used: no score reduction was registered. The SVM model yielded near to perfect results, with TextBlob and the dataset without sentiment scores, reaching a full 100% in all metrics. Finally, for the NN model, the dataset without sentiment scores seemed to perform better, compared to the other 2 datasets: with a Recall of 99.71 % compared to 90.83 % in Textblob and 92.74 % in VADER.

Although both the Logistic Regression and the NN model scored better on the dataset without sentiment score, Random Forest and Decision tree scored always the same perfect results, so we can say that the best performing models did not lose any predictive powers. The answer to the last research question is that we cannot disprove or prove that NLP sentiment scores together with ML methodologies can improve or outperform the results of the first methodology (models calculated without sentiment scores).

CONCLUSIONS

In this project, we aimed to predict the success of M&A deal completion, using ML classification models and NLP methodologies.

After a successful dimensionality reduction task, were a triangulation of results between Pearson's correlation coefficient, t-test analysis on Pearson's (r) results and Spearman's rank coefficient, plus filter methods were used, the financial variables with major impact in the dataset, more useful in making M&A prediction were selected.

On this dataset, Logistic Regression was used as a baseline model against which Random Forest, Neural Network (NN), Decision Tree and Support Vector Machine (SVM) algorithms were confronted. The models which resulted to be best performing in the prediction of successful M&A deal completion were: Random Forest, Decision Tree and SVM model. With a perfect score of 100% in all the metrics, these models all outperformed the logistic regression by + 2.77 % on F-score, + 2.25 % on Recall, + 3.27 % on Precision and + 5.80 % on Accuracy.

To discern the predictive power of the news and headlines released before a M&A deal announcement, sentiment analysis with NLP was performed on the headline's textual dataset. The baseline against which the performance of VADER was compared to, was Textbolb sentiment analyzer.

The sentiment scores were firstly evaluated with the share prices variable, secondly with the variable-selected financial dataset (inclusive of the share prices), and all of the chosen algorithms re-applied again.

In the first approach the best performing models were the Logistic regression, Random Forest and Decision Tree and the SVM classifier all performed a perfect 100% score in all metrics for both datasets. Here, we were then able to confirm that applying NLP sentiment analysis on news prior to an announcement (while also taking into consideration the share prices changes) can actually help us predict an imminent M&A deal.

The hypothesis that high share prices, together with no or very low positive news sentiment, were an indication of an imminent M&A announcement could be also confirmed, thanks to the use of some data visualizations, were the price shares variables

were depicted together with VADER compound scores and TextBlob Polarity and Subjectivity scores.

In the second approach, the sentiment scores applied to the financial dataset yielded excellent results with both the Random Forest and Decision Tree models, performing a perfect 100% scores for both VADER and TextBlob datasets.

While the SVM model yielded perfect scores for the TextBlob analyzer, it scored slightly less in F-score and Recall with the VADER analyzer, precisely 99.80 % and 99.61 % respectively. The Logistic Regression scored the same results in both datasets.

Lastly, to assess the predictive power of the sentiment scores and to determine whether NLP sentiment scores could improve or outperform the results of the models calculated without sentiment scores, all the results were compared: although both the Logistic Regression and the NN model scored better on the dataset without sentiment score (perfect 100.00% on all metrics for the first and Recall of 99.71 % for the second), the Random Forest and Decision tree scored always the same perfect results in all datasets. According to these results, we could not disprove or prove that NLP sentiment scores together with ML methodologies could improve or outperform the results of the first methodology (models calculated without sentiment scores); the only fact we could confirm is that the best performing models did not lose any predictive powers in the dataset with sentiment scores.

FURTHER WORK

Further work can be done on the project to better assess the predictive power of the share price variable and to disprove or prove that NLP sentiment scores together with ML methodologies could improve or outperform the results of the first methodology (models calculated without sentiment scores).

To achieve these results, a more in-depth analysis of the share price variable could be performed on a bigger dataset with price share movements of a particular Target company, to better capture the effects during M&A pre ad post announcement.

Moreover, different ML models could be utilized, together with different NLP methodologies on a bigger dataset, for sentiment scores calculation to assess the predictive power of the sentiment scores.

This approach may provide more insights and more robust results on benefits or no benefit of the combined approach: sentiment scores together with financial dataset evaluation.

Word count [8,457]

REFERENCE

- Buxton, P. (2022). Data science research topic (DSM060-2022-APR) *COURSEWORK 2: SOCIAL MEDIA HATE SPEECH DETECTION AND ANALYSIS* (University of London). [online] Available at: https://learn.london.ac.uk/course/view.php?id=734&section=2 [Accessed 10 Dec. 2022].

- Buxton, P. (2021). NLP (DSM140-2021-APR) *NLP COURSEWORK 1* (University of London) [online] Available at: https://learn.london.ac.uk/mod/coursework/view.php?id=47251 [Accessed 7 Sep. 2022].

- City.ac.uk. (2016). *What makes a company an attractive M&A target? | Bayes Business School*. [online] Available at: https://www.bayes.city.ac.uk/faculties-and-research/research/bayes-knowledge/2016/november/what-makes-a-company-an-attractive-m-and-a-target [Accessed 26 Feb. 2023].

- DeepAI (2021). *Predicting Mergers and Acquisitions using Graph-based Deep Learning.* [online] DeepAI. Available at: https://deepai.org/publication/predicting-mergers-and-acquisitions-using-graph-based-deep-learning [Accessed 7 Nov. 2022].

- FactSet Research Systems Inc (2021). *FactSet Mergers*. [online] Factset.com. Available at: https://go.factset.com/marketplace/catalog/product/factset-mergers [Accessed 14 Dec. 2022].

- Gregg A Jarrell and Annette B Poulsen (1989). *Stock Trading before the Announcement of Tender Offers: Insider Trading or Market Anticipation?* The Journal of Law, Economics, and Organization, 5, (2), Pp 225-48.

- Investopedia. (2023). *Corporate Finance*. [online] Available at: https://www.investopedia.com/corporate-finance-and-accounting-4689821 [Accessed 21 Feb. 2023].

- Jiang, T. (2021). *Using Machine Learning to Analyze Merger Activity*. *Frontiers in Applied Mathematics and Statistics.* [online] 7. doi:10.3389/fams.2021.649501.

- John Pound and Richard Zeckhauser (1990). *"Clearly Heard on the Street: The Effect of Takeover Rumors on Stock Prices."* Journal of Business, 63, 3, Pp. 291-308.

- Karatas, T. and Hirsa, A. (2021). *Predicting Status of Pre and Post M&A Deals Using Machine Learning and Deep Learning Techniques*. arXiv.org. [online] doi:10.48550/arXiv.2110.09315.

- Li, Y., Shou, J., Treleaven, P. and Wang, J. (2021). *Graph neural network for merger and acquisition prediction.* [online] ResearchGate. Available at: https://www.researchgate.net/publication/360392911_Graph_neural_network_for_merger_and_acquisition_prediction [Accessed 7 Nov. 2022].

- Moriarty, R., Ly, H., Lan, E. and McIntosh, S. (2019). *Deal or No Deal: Predicting Mergers and Acquisitions at Scale*. [online] undefined. Available at: https://www.semanticscholar.org/paper/Deal-or-No-Deal%3A-Predicting-Mergers-and-at-Scale-Moriarty-Ly/f71ad238d4b118d84a69a252e40c5d7fcf0d8094 [Accessed 8 Nov. 2022].

- Netlify.app. (2019). *Wrapper methods - Michael Fuchs Python*. [online] Available at: https://michael-fuchs-python.netlify.app/2019/09/27/wrapper-methods/ [Accessed 22 Feb. 2023].

- Ning, Y. and Chen, X. (2009). *Coal mine enterprises merger and acquisition prediction based on wavelet neural network.* [online] ResearchGate. Available at:

https://www.researchgate.net/publication/241166203_Coal_mine_enterprises_merger_and_acquisition_prediction_based_on_wavelet_neural_network [Accessed 7 Nov. 2022].

- Refinitiv.com. (2022). *Prediction of M&A targets to generate portfolio returns.* [online] Available at: https://developers.refinitiv.com/en/article-catalog/article/prediction-of-m-and-a-targets-to-generate-portfolio-returns#TheoreticalBackground [Accessed 22 Oct. 2022].

- Refinitiv.com. (2022). *Predicting M&A Targets Using ML: Unlocking the potential of NLP based variables*. [online] Available at: https://developers.refinitiv.com/en/article-catalog/article/predicting-MnA-targets-using-ML-Unlocking-the-potential-of-NLP-variables [Accessed 5 Nov. 2022].

- Refinitiv-API-Samples (2022). *Article.RDP.Python.PredictingM-ATargetsUsingML UnlockingThePotentialOfNLPBasedVariables/Datasets at main · Refinitiv-API-Samples/Article.RDP.Python.PredictingM-ATargetsUsingML-UnlockingThePotentialOfNLPBasedVariables.* [online] GitHub. Available at: https://github.com/Refinitiv-API-Samples/Article.RDP.Python.PredictingM-ATargetsUsingML-UnlockingThePotentialOfNLPBasedVariables/tree/main/Datasets [Accessed 8 Dec. 2022].

- Refinitiv (2022). *MarketPsych*. [online] @Refinitiv. Available at: https://www.refinitiv.com/en/financial-data/financial-news-coverage/marketpsych [Accessed 14 Dec. 2022].

- Shah, P. (2020). *Sentiment Analysis using TextBlob - Towards Data Science*. [online] Medium. Available at: https://towardsdatascience.com/my-absolute-go-to-for-sentiment-analysis-textblob-3ac3a11d524 [Accessed 24 Feb. 2023].

- Simplilearn (2022). *Sklearn SVM | In-depth Lesson On Support Vector Machines*. [online] Simplilearn.com. Available at: https://www.simplilearn.com/tutorials/scikit-learn-tutorial/sklearn-svm-support-vector-machines [Accessed 13 Mar. 2023].

- Turney, S. (2022). *Pearson Correlation Coefficient (r) | Guide & Examples*. [online] Scribbr. Available at: https://www.scribbr.com/statistics/pearson-correlation-coefficient/ [Accessed 22 Feb. 2023].

- Wikipedia Contributors (2023). *Mergers and acquisitions*. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Mergers_and_acquisitions [Accessed 26 Feb. 2023].

- Wisconsin School of Business.(2022). *Machine Learning and Mergers and Acquisitions.* [online]. Available at: https://business.wisc.edu/centers/nicholas/blog/nicholas-centers-first-ever-machine-learning-consulting-project-research-report-on-predicting-ma-targets/ [Accessed 7 Nov. 2022].

APPENDICES

Appendix A: Datasets Processing & Cleaning Code

Appendix A: Datasets Processing & Cleaning Code

In [ ]:

```
# Packages import and installation

import pandas as pd
import numpy as np

%pip install matplotlib
import matplotlib.pyplot as plt

%pip install seaborn
import seaborn as sns
sns.set_style("whitegrid")

from pandas.plotting import scatter_matrix

import scipy.stats as stats

%pip install nltk
import nltk
nltk.download('stopwords')
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk import FreqDist
import re
nltk.download('punkt')

nltk.download('wordnet')
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer

pd.set_option('display.max_rows', 25)
```

Data Pre-processing and Cleaning on the Target Dataset

In [ ]:
```
# Reading the Target dataset, stored as an Excel file

target_data = pd.read_excel("target_data.xlsx")
target_data.head()
```

In [ ]:
```
# Uploading the dataset into a data frame, for additional processing

df = pd.read_excel("target_data.xlsx")
```

In [ ]:
```
# Checking the dataset column types

print(df.dtypes)
```

In [ ]:
```
# Renaming the first columns as 'Index' instead of unnamed:

df.rename(columns = {"Unnamed: 0": "Index"}, inplace = True)

print(df.columns)
```

In [ ]:
```
# Getting some statistics on the dataset

df.describe()
```

In [ ]:
```
# Are there any cells with NaN/Na value or any empty cells?

df.isnull().values.any() # This checks for NaN/Na and empty cells in the entire DataFrame
```

RESULTS: There are no cells with Nan or Na value or empty cells in this dataset, next, the dataset is going to be checked for any other String values, cells with value 0 and for duplicated rows.

In [ ]:
```
# Are all the columns, numeric columns?

df.apply(lambda s: pd.to_numeric(s, errors='coerce').notnull().all())
```

RESULTS: Column 'Instrument' appears to be a string column. This is correct because this column refers to the referencing code system used to identify the company. All the other columns have numerical values.

```
# Are there cells with value 0?

df.eq(0).sum()
```

RESULTS: It appears that there are 3 columns with cells with value 0. These columns are 'Operating Margin - %, TTM' , 'Revenue from Business Activities - Total.3' and 'Debt - Total'. A total of 7 rows with value 0 overall in these column have been identified. To create a complete and comprehensive dataset, these rows are dropped as the outcome is to avoid null values to affect the results. Finally, no duplicated rows are found in the dataset.

In [ ]:
```
# In column 'Operating Margin - %, TTM' which rows exactly contain value 0?

Operating_margin = df['Operating Margin - %, TTM'] == 0

Operating_margin_True = df[Operating_margin]
```

In [ ]:
```
# In column 'Revenue from Business Activities - Total.3' which rows exactly contain value 0?

Revenue_business = df['Revenue from Business Activities - Total.3'] == 0

Revenue_business_True = df[Revenue_business]
```

In [ ]:
```
# In column 'Debt - Total' which rows exactly contain value 0?

Debt_total = df['Debt - Total'] == 0

Debt_total_True = df[Debt_total]
```

In [ ]:
```
# Creating a new dataframe which excludes these rows with value 0, by dropping the
# not needed rows and creating a new cleaned df

new_df = df.drop([109, 155, 71, 2, 14, 26, 189], axis=0)
new_df.head()
```

In [ ]:
```
# Double checking that no 0 zalues are left in the new dataframe

new_df.eq(0).sum() # Cleaned!
```

```
# Are there duplicated rows across all columns in the new dataframe?

Duplicated_Rows = new_df[new_df.duplicated()]

Duplicated_Rows # No!
```

In [ ]:
```
# Adding a Column Target/Non-Target, with number 1 to indicate the row belongs to the
# Target dataset and for later on classification task

new_df.insert(0, 'Target/Non-Target', '1')
```

In [ ]:
```
new_df
```

Data Pre-processing and Cleaning on the Non-Target Dataset

In [ ]:
```
# Uploading the Non-Target dataset into a data fram

df2 = pd.read_excel("peer_data.xlsx")

df2.head()
```

In [ ]:
```
# Checking the dataset column names and types:

print(df2.dtypes)
```

In [ ]:
```
# Renaming the first columns as 'Index' instead of unnamed:

df2.rename(columns = {"Unnamed: 0": "Index"}, inplace = True)

print(df2.columns)
```

In [ ]:
```
# Checking here if the 2 initial dataframes (Target and Non-Target) have actually the same columns:

set(df2.columns).intersection(set(new_df.columns))
```

RESULTS: It appears there are 21 columns instead of 22. This is due to the fact that the date limit colum 'AD' is not present as the dataset comprises data from until the time of extraction (December 2021).

In [ ]:

```
# Getting some statistics on the dataset

df2.describe()
```

In [ ]:

```
# Are there cells with NaN/Na value or any empty cells?

df2.isnull().values.any() # This checks for NaN/Na and empty cells in the entire DataFrame
```

In [ ]:

```
# Are all the columns, numerical columns?

df2.apply(lambda s: pd.to_numeric(s, errors='coerce').notnull().all())
```

RESULTS: Again, column Instrument appears to be a string column. Now, also column 'AD-30, is not considered numerical. Let's have a look at the column type.

In [ ]:

```
# Checking the dataset columns types

print(df2.dtypes)
```

RESULTS: The 'AD-30' Column seems to be cathegoriesed as object. We will change it to the more accurate type: date-time

In [ ]:

```
# Converting the AD-30 column into column type datetime64

df2['AD-30']= pd.to_datetime(df2['AD-30'])
```

In [ ]:

```
# Checking again the dataset columns types to see if everything is in order here

print(df2.dtypes)
```

In [ ]:

```
# Running again the numeric column code. Are the columns all numeric now? ( Except for column Instrument)

df2.apply(lambda s: pd.to_numeric(s, errors='coerce').notnull().all())
```

RESULTS: All good now!

In [ ]:

```
# Are there cells with value 0?

df2.eq(0).sum()
```

RESULTS: It appears that there are 6 columns with cells with value 0. These columns are 'Index' which is due to the fact that it starts at 0 instead of 1, which is okay. Next, we have the 'Revenue from Business Activities - Total.1' , 'Revenue from Business Activities - Total.2', 'Revenue from Business Activities - Total.3', 'Debt - Total' and 'Cash & Cash Equivalents - Total' columns.

A total of 414 rows with value 0 overall in these column have been identified. Again, to create a complete and comprehensive dataset, these rows are dropped. Finally, no duplicated rows are found in the dataset.

In [ ]:

```
# In column 'Revenue from Business Activities - Total.1' which rows exactly contain value 0?

Revenue_1 = df2['Revenue from Business Activities - Total.1'] == 0

Revenue_1_True = df2[Revenue_1]

# In column 'Revenue from Business Activities - Total.2' which rows exactly contain value 0?

Revenue_2 = df2['Revenue from Business Activities - Total.2'] == 0

Revenue_2_True = df2[Revenue_2]

# In column 'Revenue from Business Activities - Total.3' which rows exactly contain value 0?

Revenue_3 = df2['Revenue from Business Activities - Total.3'] == 0

Revenue_3_True = df2[Revenue_3]

# In column 'Debt - Total' which rows exactly contain value 0?

Debt = df2['Debt - Total'] == 0

Debt_True = df2[Debt]

# In column 'Cash & Cash Equivalents - Total' which rows exactly contain value 0?

Cash = df2['Cash & Cash Equivalents - Total'] == 0

Cash_True = df2[Cash]

# Dropping all the above rows with values 0 and creating a new dataframe

df2_new = pd.concat([df2, Revenue_1_True, Revenue_2_True, Revenue_3_True, Debt_True, Cash_True]).drop_duplicates(ke

# Checking the number of rows and columns

df2_new.shape
```

```
# Are there cells with value 0 in this new dataset (except for the Index)?

df2_new.eq(0).sum()
```

In [ ]:
```
# Are there duplicated rows across all columns in the new dataframe?

Duplicated_Rows_2 = df2_new[df2_new.duplicated()]

Duplicated_Rows_2 # No!
```

In [ ]:
```
# Adding a Column Target/Non-Target, with number 0 to indicate the row belongs to the Non Target dataset

df2_new.insert(0, 'Target/Non-Target', '0')
df2_new.shape
```

Now merging the 2 datasets (Target and Non-Target) together

In [ ]:
```
# Merging the target dataset new_df, with the non target df2_new + fixing the column type of the Terget column as i

num_dataset = new_df.append(df2_new)
num_dataset["Target/Non-Target"] = num_dataset["Target/Non-Target"].astype(str).astype(int)
num_dataset.head()
```

In [ ]:
```
# Checking the columnn and rows numbers
num_dataset.shape
```

In [ ]:
```
# Exporting the outcome into pickle, so we can use this dataset later, for further processing

num_dataset.to_pickle("Merged numerical financial dataset")
```

Data Pre-processing and Cleaning on the Headlines Dataset

In [ ]:
```
# Uploading the dataset into a data frame, let's have a look:

df3 = pd.read_excel("headlines.xlsx")

# Let's have a look at all the columns

df3.head()
```

In [ ]:
```
# Are there missing values in each of the columns?

df3.isnull().any()
```

In [ ]:
```
# Checking the dataset column types

print(df3.dtypes)

# Renaming the first columns as 'Index' instead of unnamed:

df3.rename(columns = {"Unnamed: 0": "Index"}, inplace = True)

print(df3.columns)
```

In [ ]:
```
# Removing all special characters using regex, from column 'Headlines'.

df3['Headlines'] = df3.Headlines.replace(r'[^\w\s]|_', '', regex=True)

# Adding homogenity by lowering all the letters

df3['Headlines'] = df3.Headlines.str.lower()
```

```
# Removing stop words using the function defined below

# Getting all the english stopwords list first

stop_words = set(stopwords.words('english'))

# Defining my cleaning function, by first tokenizing the text in column Headlines:

def Cleaning_function(Headlines):
    '''Tokenizing the text, removing any stop words and returning
    the cleaned tokenized sentence'''
    word_tokens = word_tokenize(Headlines)

    Cleaned_sentence = []
    for word_token in word_tokens:
        if word_token not in stop_words:
            Cleaned_sentence.append(word_token)

# Joining the cleaned sentences together and returning my clean text:

    text = (' '.join(Cleaned_sentence))
    return text

# Now applying this newly created clean function into my dataframe with apply() to column Headlines

df3['Headlines'] = df3['Headlines'].apply(Cleaning_function)

# Printing the results:

print(df3['Headlines'])
```

In [ ]:

```
# Transforming the headlines column into strings so I can perform Frequency distribution:

Text = df3['Headlines']

txt_string = Text.to_string()

# But first, counting the vocabularies in my Text in string format

print(len(txt_string)) # 17.888.039 items counted
```

```
# Plotting the Frequency distribution of the text keywords.
# But first, removing the punctuation again.

txt_string_nopunct = re.sub(r'[^\w\s]', '', txt_string)
txt_string_nopunct

# Separating the words with a space

text_list = txt_string.split(" ")
```

```
# Now plotting

# Getting the frequency distribution list

freqDist = FreqDist(text_list)

# Printing the 15 most common keywords

print(freqDist.most_common(15))

# Creating FreqDist for these 15 most common keywords

freqDist = FreqDist(text_list).most_common(15)

# Converting the above to Pandas series via Python Dictionary for easier plotting

freqDist = pd.Series(dict(freqDist))

# Removing the empty space character'' + the 'dj' character

freqDist.pop('')
freqDist.pop('dj')

# Setting figure and ax into variables, for plotting

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

# Seaborn plotting using Pandas attributes + xtick rotation for ease of viewing:

FreqDist_plot = sns.barplot(x=freqDist.index, y=freqDist.values, ax=ax)

# Displaying the titles
plt.xlabel('Words', fontsize=15)
plt.ylabel('Frequency count', fontsize=15)
plt.xticks(rotation=30)
plt.title("Word frequency distribution", fontsize=25)
```

RESULTS: There were multiple company referencing code duplications in column ‘RIC’, so the dataset was grouped by this particular column and the text merged together. Unneccessary columns were also dropped in the script below, to make it sentiment analysis ready.

```
# Dropping unneccessary columns

df3_dropped = df3.drop(['sdate'], axis=1)
```

In [ ]:
```
# Dropping unneccessary columns

df3_dropped_final = df3_dropped.drop(['edate'], axis=1)
```

In [ ]:
```
# Visualizing the outcome

df3_dropped_final.head()
```

In [ ]:
```
# Sorting the dataset by column RIC, as the enties are repeted

headlines_sorted = df3_dropped_final.groupby(['RIC'])['Headlines'].apply(list)
```

In [ ]:
```
# Visualizing the sorded Headlines column by RIC column

headlines_sorted
```

In [ ]:
```
# Placing the result in a dataframe df

df = pd.DataFrame(headlines_sorted)
```

In [ ]:
```
df # Done
```

In [ ]:
```
# Checking the dataframe type

print(type(df['Headlines'] ))
```

In [ ]:
```
# Adjusting the headlines type to string, to make it sentiment analysis ready

df['Headlines']  = df['Headlines'].astype(str)
```

In [ ]:
```
# Exporting the outcome into pickle, so we can use this dataset later, for further processing

df.to_pickle("Cleaned headlines dataset")
```

Appendix B: Variables Selection

Appendix B: Variables selection

In [ ]:
```
# Packages import and installation

import pandas as pd
import numpy as np

%pip install matplotlib
import matplotlib.pyplot as plt

%pip install seaborn
import seaborn as sns
sns.set_style("whitegrid")

from sklearn.preprocessing import MinMaxScaler
from pandas.plotting import scatter_matrix

from sklearn.model_selection import train_test_split
from sklearn.feature_selection import VarianceThreshold

import scipy.stats as stats

pd.set_option('display.max_rows', 25)
```

In [ ]:
```
# Reading the cleaned and merged (Targe+Non-target) dataset into a datafre.
# This dataset was pickled and named 'Merged numerical financial dataset' in Appendix A

num_df = pd.read_pickle("Merged numerical financial dataset")

# Visualizing the DataFrame
num_df.head()
num_df.shape
```

In [ ]:
```
# Dropping the unneccessary columns, to make my dataset ready for the filter methods

num_df_filter = num_df.drop(['Index','Instrument','AD', 'AD-30'], axis=1)
num_df_filter.shape
```

APPLYING FILTER METHODS FOR VARIABLES SELECTION

```
# Getting the dataset ready, by splitting into test and train

x = num_df_filter
y = num_df_filter['Target/Non-Target']

trainX, testX, trainY, testY = train_test_split(x, y, test_size = 0.2, random_state=45)

# Next we will use VarianceThreshold function to remove quasi constant features.

constant_filter = VarianceThreshold(threshold=0)
constant_filter.fit(trainX)

# Similarly, to find the number of constant features the following code was used:

constant_columns = [column for column in trainX.columns
                    if column not in trainX.columns[constant_filter.get_support()]]

print(len(constant_columns))
```

In [ ]:

```
# Now we want to get all the features that are not constant (features we want to keep):

len(trainX.columns[constant_filter.get_support()]) # All columns appear to be non constant
```

In [ ]:

```
# Removing Quasi-Constant features

qconstant_filter = VarianceThreshold(threshold=0.01)
qconstant_filter.fit(trainX)
```

In [ ]:

```
# Identify the Quasi constant features

qconstant_columns = [column for column in trainX.columns
                    if column not in trainX.columns[qconstant_filter.get_support()]]

print(len(qconstant_columns)) # 0
```

RESULTS: No constant features nor duplicated features were identified in the dataset.

VARIABLE SELECTION WITH A TRIANGULATION METHOD (PEARSON, SPEARMAN & t-test)

In [ ]:
```
# Dropping the unneccessary columns

num_df_noindex = num_df.drop(['Index', 'Target/Non-Target','Instrument','AD', 'AD-30'], axis=1)
```

In [ ]:
```
# Using Scikit-learn to transform the dataset with maximum absolute scaling, for better visualization

scaler = MinMaxScaler()
scaler.fit(num_df_noindex)
scaled = scaler.transform(num_df_noindex)
scaled_df = pd.DataFrame(scaled, columns=num_df_noindex.columns)

scaled_df.head()
scaled_df.shape
```

In [ ]:
```
# Crating a Scatter Matrix plot of the dataset

axes = pd.plotting.scatter_matrix(scaled_df, figsize  = [20, 20], alpha=0.2)
for ax in axes.flatten():
    ax.xaxis.label.set_rotation(90)
    ax.yaxis.label.set_rotation(0)
    ax.yaxis.label.set_ha('right')

plt.suptitle('Scatter Matrix representation of the Variables', y=1, fontsize=25)

# Setting the layout
plt.tight_layout()
plt.gcf().subplots_adjust(wspace=0.2, hspace=0.2)

# Displaying the title

plt.show()
```

In [ ]:
```
# Calculating Pearson's correlation coefficient

num_df_noindex.corr()
```

```
# Plotting the results into a Heatmap

# Define figure size

plt.figure(figsize=(22,20))

# Define title and dimensions of the words
plt.title("Pearson Correlation Coefficient Heatmap", y=1, fontsize=25)

# Printing a heat map of the correlation

matrix = num_df_noindex.corr().round(2)
sns.heatmap(matrix, annot=True, vmax=1, vmin=-1, center=0, cmap='vlag')
```

In [ ]:
```
# Selecting Only Strong Correlations in the Correlation Matrix (Checking also for negative correlation)

Matrix_strong = num_df_noindex.corr()
Matrix_strong = Matrix_strong.unstack()
Matrix_strong = Matrix_strong[abs(Matrix_strong) >= 0.7]


Most_correlated = Matrix_strong.sort_values(ascending=False)
Most_correlated
pd.set_option('display.max_rows', 10000)


print(Most_correlated)
```

In [ ]:
```
# Most correlated variables with score greater than 0.7

correlated_features = set()
threshold = 0.70

for i in range(len(matrix.columns)):
    for j in range(i):
        if abs(matrix.iloc[i, j]) >= threshold:
            colname = matrix.columns[i]
            correlated_features.add(colname)
```

In [ ]:
```
print(correlated_features)
```

RESULTS: Most correlated variables according to Pearson results, by number of variables correlation:

9 Total Capital / 9 Total Shareholders' Equity incl Minority Intr & Hybrid Debt / 8 Revenue from Business Activities - Total.2 / 8 Revenue from Business Activities - Total.1 / 8 Enterprise Value (Daily Time Series) / 8 Cash & Cash Equivalents - Total / 8 Revenue from Business Activities - Total.3 / 8 Revenue from Business Activities - Total / 8 Debt - Total / 4 Earnings before Interest Taxes Depreciation & Amortization / 2 Free Cash

In [ ]:
```
# Calculating Spearman's correlation

Matrix_Spearman = num_df_noindex.corr(method="spearman")
```

In [ ]:
```
# Showing Spearman's coefficients on a Heatmap

# Defining figure size

plt.figure(figsize=(22,20))

# Defining title

plt.title("Spearman Correlation Coefficient Heatmap", y=1, fontsize=25)

# Printing a heat map of the correlation

sns.heatmap(Matrix_Spearman, annot=True, vmax=1, vmin=-1, center=0, cmap='vlag')
plt.show()
```

In [ ]:
```
# Selecting only the most correlated variables (negatively and positively)

Matrix_P_strong = num_df_noindex.corr(method="spearman")
Matrix_P_strong = Matrix_P_strong.unstack()
Matrix_P_strong = Matrix_P_strong[abs(Matrix_P_strong) >= 0.7]


Most_correlated_P = Matrix_P_strong.sort_values(ascending=False)
Most_correlated_P

print(Most_correlated_P)
```

```
# Check with Spearman as well! The most correlated values

Spearman_correlated_features = set()
threshold = 0.70

for i in range(len(Matrix_Spearman.columns)):
    for j in range(i):
        if abs(Matrix_Spearman.iloc[i, j]) >= threshold:
            colname = Matrix_Spearman.columns[i]
            Spearman_correlated_features.add(colname)
```

In [ ]:
```
print(Spearman_correlated_features)
```

RESULTS: Most correlated variables according to Spearman results, by number of variables correlation:

9 Total Capital / 9 Revenue from Business Activities - Total / 9 Revenue from Business Activities - Total.3 / 9 Revenue from Business Activities - Total.1 / 9 Enterprise Value (Daily Time Series) / 8 Earnings before Interest Taxes Depreciation & Amortization / 8 Total Shareholders' Equity incl Minority Intr & Hybrid Debt / 8 Debt - Total / 6 Cash & Cash Equivalents - Total

Calculating t-test, to be used on Pearsman coefficients

In [ ]:
```
# Getting the dataset with the Terget column included,so I can apply
# t-test on each column, comparing target and non target values

num_df_target = num_df.drop(['Index', 'Instrument','AD-30','AD'], axis=1)
num_df_target.head()
```

In [ ]:
```
# Getting the dataset with only the highly correlated variables

Correlated_T_test = num_df_target.drop(num_df_target.columns[[1, 2, 4, 5, 10, 11, 12, 17]], axis=1)
# With Target and no index
```

In [ ]:
```
Correlated_T_test.head()
```

In [ ]:
```
# Running the T-test to help me decide which variable to remove


T_test = stats.ttest_ind(Correlated_T_test.loc[Correlated_T_test['Target/Non-Target']==1], Correlated_T_test.loc[C
```

In [ ]:
```
# Storing the calculated results in a dataframe, so we can visualize it:

T_test_data = pd.DataFrame()
T_test_data["Variable"] = Correlated_T_test.columns # Setting the Variables columns
T_test_data["T_test"] = T_test[0] # Setting the T-test values values per column
T_test_data["p-value"] = T_test[1] # Setting the p-value values per column

# Creating the acuat dataframe:
T_test_data = T_test_data.T
T_test_data.rename(columns = T_test_data.iloc[0], inplace = True)
T_test_data = T_test_data.iloc[1:]

# Printing the outcome:
T_test_data
```

RESULTS: The only variable which was kept in the original dataset and not dropped, was 'Debt-Total': with a p value of .7425, there was a high 74.25% chance that the relationship, and therefore high correlation, occurred by chance. Since this p-value was not less than .05, we failed to reject the null hypothesis, so in this case, with a high p value and lower T-test the relationship was deemed as not statistically significant as the data didn't allow to reject the null hypothesis and didn't provide support for the alternative hypothesis. The decision to keep the variable was then made. The threshold for the p-value was set at .7425, so all the other variables were deemed statistically significant and dropped.

In [ ]:
```
# Dropping unneccessary columns

num_df_features = num_df.drop(num_df.columns[[3,4,7,10,11,12,13,17,18,19,22]], axis=1) # With Target
num_df_features.head()
```

In [ ]:
```
# Checking the columns

num_df_features.columns
```

In [ ]:
```
# Cheching the dataframe shape

num_df_features.shape
```

In [ ]:
```
# Exporting the final, variable selected dataframe with pickle, so we can use this dataset later for further proces

num_df_features.to_pickle("Variable selected financial dataset")
```

Appendix C: METHODOLOGY I – Models with No sentiment scores

Appendix C: METHODOLOGY I – Models with No sentiment scores

In [ ]:
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.svm import LinearSVC

from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score

import seaborn as sns
import matplotlib
from matplotlib import pyplot as plt
import numpy as np
import pandas as pd
```

In [ ]:
```
# Read the pickle file of the variable selected dataset created in Appendix B, into a dataframe

num_df_features = pd.read_pickle("Variable selected financial dataset")

# View the DataFrame
num_df_features
```

In [ ]:
```
# Dropping unneccessary columns

num_df_features.drop(['Index','Instrument'],inplace = True, axis=1)
```

In [ ]:
```
num_df_features['Target/Non-Target'].value_counts()
```

```
  Choosing a subset of the data to split in between train and test:

X = num_df_features
y = num_df_features['Target/Non-Target']

X_train, X_test, y_train, y_test = train_test_split(X, y, shuffle=True, test_size=0.3, random_state=11)
```

BASELINE MODEL: Logistic Regression

In [ ]:

```
# Baseline performance: Logistic regression classifier

lr = LogisticRegression()

# Applying the model to the training data:

lr.fit(X_train,y_train)

# Predict the test model:

labels_lr = lr.predict(X_test)
labels_lr
```

In [ ]:

```
# Let's evaluate the results with accuracy:

print('Logistic Regression Test Accuracy:', accuracy_score(y_test, labels_lr))
print('Logistic Regression Train Accuracy:', accuracy_score(y_train, lr.predict(X_train)))

# Recall - but also precision, f1-score and support:

print(classification_report(y_test, labels_lr))
print(classification_report(y_train, lr.predict(X_train)))

# Confusion matrix:

mat_lr = confusion_matrix(y_test,labels_lr)
sns.heatmap(mat_lr, square=True, annot=True, fmt="d", cbar=False,
           xticklabels=['Non-Target', 'Target'], yticklabels=['Non-Target', 'Target'])

plt.xlabel('Predicted Label')
plt.ylabel('Label')
```

```
  Precision

precision = precision_score(y_test, labels_lr, average=None)
print(precision)

# Recall

recall = recall_score(y_test, labels_lr, average=None)
print(recall)

# F-score
f_score = f1_score(y_test, labels_lr, average=None)
print(f_score)
```

In [ ]:

```
# Applying nested cross-validation check:

scores = cross_val_score(lr, X, y, cv=10, scoring='accuracy')
print(scores)
print("%0.4f accuracy with a standard deviation of %0.4f" % (scores.mean(), scores.std()))
```

THE OTHER CLASSIFICATION MODELS

In [ ]:

```
# Random Forest model

# Creating the regressor object

regressor = RandomForestRegressor(n_estimators = 100, random_state = 0)

# Applying the model to the training data:

regressor.fit(X_train,y_train)

# Predict the test model:

labels_regressor = regressor.predict(X_test)
labels_regressor
```

```
  Let's evalueate the results with accuracy:

print('Random Forest Test Accuracy:', accuracy_score(y_test, labels_regressor))
print('Random Forest Train Accuracy:', accuracy_score(y_train, regressor.predict(X_train)))

# Recall - but also precision, f1-score and support:

print(classification_report(y_test, labels_regressor))
print(classification_report(y_train, regressor.predict(X_train)))

# Confusion matrix:

mat_regressor = confusion_matrix(y_test,labels_regressor)
sns.heatmap(mat_regressor, square=True, annot=True, fmt="d", cbar=False,
           xticklabels=['Non-Target', 'Target'], yticklabels=['Non-Target', 'Target'])

plt.xlabel('Predicted Label')
plt.ylabel('Label')
```

In [ ]:

```
# Precision

precision = precision_score(y_test, labels_regressor, average=None)
print(precision)

# Recall

recall = recall_score(y_test, labels_regressor, average=None)
print(recall)

# F-score
f_score = f1_score(y_test, labels_regressor, average=None)
print(f_score)
```

```
  Neural Network (NN)

# Building the classifier

mlp = MLPClassifier(hidden_layer_sizes=(13,13,13),max_iter=500)

# Applying the model to the training data:

mlp.fit(X_train,y_train)

# Predict the test model:

labels_mlp = mlp.predict(X_test)
labels_mlp
```

In [ ]:

```
# Let's evalueate the results with accuracy:

print('NN Test Accuracy:', accuracy_score(y_test, labels_mlp))
print('NN Train Accuracy:', accuracy_score(y_train, mlp.predict(X_train)))

# Recall - but also precision, f1-score and support:

print(classification_report(y_test, labels_mlp))
print(classification_report(y_train, mlp.predict(X_train)))

# Confusion matrix:

mat_mlp = confusion_matrix(y_test,labels_mlp)
sns.heatmap(mat_mlp, square=True, annot=True, fmt="d", cbar=False,
           xticklabels=['Non-Target', 'Target'], yticklabels=['Non-Target', 'Target'])

plt.xlabel('Predicted Label')
plt.ylabel('Label')
```

```
  Precision

precision = precision_score(y_test, labels_mlp, average=None)
print(precision)

# Recall

recall = recall_score(y_test, labels_mlp, average=None)
print(recall)

# F-score
f_score = f1_score(y_test, labels_mlp, average=None)
print(f_score)
```

In [ ]:
```
# Applying nested cross-validation check:
scores_mlp = cross_val_score(mlp, X, y, cv=10, scoring='accuracy')
print(scores_mlp)
print("%0.4f accuracy with a standard deviation of %0.4f" % (scores_mlp.mean(), scores_mlp.std()))
```

In [ ]:
```
# Decision Tree

# Building the model

dt = DecisionTreeClassifier(criterion='entropy')

# Applying the model to the training data:

dt.fit(X_train,y_train)

# Predict the test model:

labels_dt = dt.predict(X_test)
labels_dt
```

```
  Let's evalueate the results with accuracy:

print('Decision Tree Test Accuracy:', accuracy_score(y_test, labels_dt))
print('Decision Tree Train Accuracy:', accuracy_score(y_train, dt.predict(X_train)))

# Recall - but also precision, f1-score and support:

print(classification_report(y_test, labels_dt))
print(classification_report(y_train, dt.predict(X_train)))

# Confusion matrix:

mat_dt = confusion_matrix(y_test,labels_dt)
sns.heatmap(mat_dt, square=True, annot=True, fmt="d", cbar=False,
            xticklabels=['Non-Target', 'Target'], yticklabels=['Non-Target', 'Target'])

plt.xlabel('Predicted Label')
plt.ylabel('Label')
```

In [ ]:
```
# Precision

precision = precision_score(y_test, labels_dt, average=None)
print(precision)

# Recall

recall = recall_score(y_test, labels_dt, average=None)
print(recall)

# F-score
f_score = f1_score(y_test, labels_dt, average=None)
print(f_score)
```

```
  Support Vector Machine (SVM)

# Building the linear Support Vector Machine Classifier

Svm = LinearSVC(dual = False, random_state = 0, penalty = 'l1',tol = 1e-5)

Svm.fit(X_train,y_train)

# Predict the test model:

labels_svm = Svm.predict(X_test)
labels_svm
```

In [ ]:

```
# Let's evalueate the results with accuracy:

print('SVM Test Accuracy:', accuracy_score(y_test, labels_svm))
print('SVM Train Accuracy:', accuracy_score(y_train, Svm.predict(X_train)))

# Recall - but also precision, f1-score and support:

print(classification_report(y_test, labels_svm))
print(classification_report(y_train, Svm.predict(X_train)))

# Confusion matrix:

mat_svm = confusion_matrix(y_test,labels_svm)
sns.heatmap(mat_svm, square=True, annot=True, fmt="d", cbar=False,
           xticklabels=['Non-Terget', 'Terget'], yticklabels=['Non-Terget', 'Terget'])

plt.xlabel('Predicted Label')
plt.ylabel('Label')
```

```
  Precision

precision = precision_score(y_test, labels_svm, average=None)
print(precision)

# Recall

recall = recall_score(y_test, labels_svm, average=None)
print(recall)

# F-score
f_score = f1_score(y_test, labels_svm, average=None)
print(f_score)
```

Appendix D: METHODOLOGY II – Models with sentiment score

Appendix D: METHODOLOGY II – Models with sentiment score

In [ ]:
```
import pandas as pd
import numpy as np

%pip install matplotlib
import matplotlib.pyplot as plt

%pip install seaborn
import seaborn as sns
sns.set_style("whitegrid")

import nltk
nltk.download('vader_lexicon')
from nltk.sentiment.vader import SentimentIntensityAnalyzer
import regex as re
from textblob import TextBlob

from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC
from sklearn.svm import LinearSVC

from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.preprocessing import MinMaxScaler

pd.set_option('display.max_rows', 25)
```

```
  Read the pickle file of the cleaned headlines dataset created in Appendix A, into a dataframe

df = pd.read_pickle("Cleaned headlines dataset")

# View the DataFrame
df
```

In [ ]:
```

```

Sentiment analysis using VADER

In [ ]:
```
# Applying sentiment analysis using VADER on the cleaned Headlines column

analyzer = SentimentIntensityAnalyzer()
df['Scores'] = df['Headlines'].apply(analyzer.polarity_scores)
```

In [ ]:
```
# Converting the scores to string for further processing

scores_str = df['Scores'].to_string()
```

In [ ]:
```
# Splitting the results

x = scores_str.split(",")
```

In [ ]:
```
# Removing neg, neu, pos, compound from the dataset

stopwords = ['neg', 'neu', 'pos', 'compound']

new_words = [word for word in x if word not in stopwords]
```

In [ ]:
```
# Placing the results into a dataframe

df_1 = pd.DataFrame(new_words)
```

```
  Clearing the column from words and leaving only numbers

for col in df_1:
    df_1[0] = [''.join(re.findall("\d+\.\d+", item)) for item in df_1[0]]
```

In [ ]:
```
# Adding a sentiment columm to clarify the numbers above

L = ['Negative', 'Neutral', 'Positive']

df_1['Sentiment'] =  (df_1.index % len(L)).map(dict(enumerate(L)))
```

In [ ]:
```
# Adding an index

df_1.reset_index(inplace = True)
```

In [ ]:
```
# Creating an ad hoc index, I can merge the 2 df together later

l = [i for i in range(0,1057) for _ in range(3)]
```

In [ ]:
```
# Placing the index into a dataframe

l = pd.DataFrame(l)
```

In [ ]:
```
# Placing the index into index list object

Index_list = l[0]
```

In [ ]:
```
# Concatenating index and sentiment scores

Concat_df = pd.concat([Index_list, (df_1.apply(pd.Series))], axis=1) # Working
```

In [ ]:
```
# Renaming columns

Concate_New = Concat_df.rename({ 0 : 'Index'}, axis='columns')
```

In [ ]:
```
# Dropping the second index column

Concate_dropped = Concate_New.drop(['index'], axis=1)
```

```
  Renaming the columns

Concate_dropped.columns = ['Index', 'Scores','Sentiment']
```

In [ ]:
```
# Finalizing my sentiment df

Sentiment_df = Concate_dropped.pivot(index='Index', columns='Sentiment', values= 'Scores')

print(Sentiment_df)
```

In [ ]:
```
# Placing the sentiment scores into a dataframe

Sentiment_df = pd.DataFrame(Sentiment_df)
```

In [ ]:
```
# Filtering sentiment scores for the columns needed

Sentiment_df = Sentiment_df.iloc[1:]
```

In [ ]:
```
# Counting the scores for better visualization

Counted = Sentiment_df.value_counts()
```

In [ ]:
```
# Setting the correct column type

Sentiment_df["Negative"] = Sentiment_df["Negative"].astype(str).astype(float)
```

In [ ]:
```
Sentiment_df["Neutral"] = Sentiment_df["Neutral"].astype(str).astype(float)
```

In [ ]:
```
Sentiment_df["Positive"] = Sentiment_df["Positive"].astype(str).astype(float)
```

In [ ]:
```
# Plotting a bar chart of the sentiment distribution

Sentiment_melted = Sentiment_df.melt(var_name='cols', value_name='vals')

Sentiment_melted
```

In [ ]:
```
# Adjusting the column type

Sentiment_melted["vals"] = Sentiment_melted["vals"].astype(str).astype(float)
```

```
  Dropp    zero's

Sentiment_melted_cleaned = Sentiment_melted[Sentiment_melted.vals != 0]
```

```
# Plotting the sentiment scores distribution

g = sns.barplot(x=Sentiment_melted_cleaned.index, y="vals", hue='cols', data=Sentiment_melted_cleaned, linewidth =
plt.legend(title = 'Sentiment', bbox_to_anchor = (1, 1))
plt.xlabel('Sentiment', fontsize=10)
plt.ylabel('Sentiment Scores', fontsize=10)
plt.title('Sentiment scores distribution', fontsize=20)
g.set(xticklabels=[])
g.tick_params(bottom=False)
```

In [ ]:
```
# Getting a count of the sentiment values

Data = Sentiment_melted_cleaned['cols'].value_counts()
```

In [ ]:
```
# Placing the result into a dataframe

Data = pd.DataFrame(Data)
```

In [ ]:
```
# Plotting a bar graph of the sentiment counts

g = sns.barplot(x=Data.index, y='cols', hue='cols', data=Data, linewidth = 0.1)
g.get_legend().remove()
plt.ylabel('Value counts', fontsize=10)
plt.title('Sentiment value counts', fontsize=20)
```

In [ ]:
```
# Dropping the scores column from the initial dataframe for further processing

df_dropped = df.drop(['Scores'], axis=1)
```

In [ ]:
```
# Resetting the index

df_dropped.reset_index(inplace = True)
```

```
  Merging this newly created Sentiment table to the initial dataset

Headline_Sentiment = pd.concat([df_dropped, Sentiment_df], axis=1)
Headline_Sentiment.head()
```

In [ ]:
```
# Getting the VADER coumpound scores, which will be utilized later when training the ML algorithms

scores = []

# Declaring the variable for the compound scores

compound_list = []

for i in range(df['Headlines'].shape[0]):

# Creating the scores and appending the results into a separate dataframa columns

    compound = analyzer.polarity_scores(df['Headlines'][i])["compound"]

    scores.append({"Compound": compound})
```

In [ ]:
```
# Results of the above:

Compound_score = pd.DataFrame.from_dict(scores)
Compound_score
```

In [ ]:
```
# This is the sentiment dataset, inclusive of the compound scores

df_comp = Headline_Sentiment.join(Compound_score)
df_comp
```

In [ ]:
```
# Dropping Negative column as I will not need it

Headlines_Vader = df_comp.drop(['Negative'], axis=1)
```

In [ ]:
```
# Dropping Neutral column as I will not need it

Headlines_Vader_1 = Headlines_Vader.drop(['Neutral'], axis=1)
```

```
  Dropp    Positive column as I will not need it

Headlines_Vader_ok = Headlines_Vader_1.drop(['Positive'], axis=1)
```

```
# Final VADER sentiment Compound scores dataset, ready for further analysis

Headlines_Vader_ok
```

In [ ]:
```
# Plotting the Compound score fo further visualization

Compound_score.plot(kind='kde')
plt.xlabel('Compound score', fontsize=10)
plt.ylabel('Compound density', fontsize=10)
plt.title('VADER Compound scores', fontsize=20)
```

Sentiment analysis using TEXTBLOB

In [ ]:
```
# Adjusting the Headlines column type

df['Headlines']  = df['Headlines'].astype(str)
```

In [ ]:
```
# Calcolating TextBlob sentiment scores. Placing the results into a dataframe

df['Polarity'] = np.nan
df['Subjectivity'] = np.nan

pd.options.mode.chained_assignment = None

for idx, Headlines in enumerate(df['Headlines'].values):  # for each row in our df dataFrame
        if Headlines:
            sentA = TextBlob(Headlines) # pass the text only article to TextBlob to analyse
            df['Polarity'].iloc[idx] = sentA.sentiment.polarity # write sentiment polarity back to df
            df['Subjectivity'].iloc[idx] = sentA.sentiment.subjectivity # write sentiment subjectivity score back t

df.head()
```

```
  Removing the Scores column

df.drop(['Scores'], axis=1, inplace = True)
```

In [ ]:
```
# Plotting the TextBlob Polarity and Subjectivity scores

df.plot(kind='kde')
plt.xlabel('Scores', fontsize=10)
plt.ylabel('Density', fontsize=10)
plt.title('TEXTBLOB Polarity and Subjectivity scores', fontsize=20)
```

In [ ]:
```
# Dropping the unneccessary Headlines column

df.drop(['Headlines'], axis=1, inplace = True)
```

In [ ]:
```
df.reset_index(inplace = True)
df
```

APPROACH A: Share price with sentiment scores

VADER

In [ ]:
```
# Reading the pickle file from Appendix A (full financial dataset)

num_RIC_features = pd.read_pickle("Merged numerical financial dataset")

#view DataFrame
num_RIC_features
```

In [ ]:
```
# Renaming column Instrument to RIC, for further processing

num_RIC_features.rename(columns = { 'Instrument' :'RIC'}, inplace = True)
```

```
  Remov    duplicated in column RIC

num_RIC_features.drop_duplicates(subset=['RIC'])
```

```
# Merging the VADER scores dataset together with the financial datasite on column RIC

Vader_num_df = num_RIC_features.merge(Headlines_Vader_ok[['Compound', 'RIC']], on = 'RIC')
```

In [ ]:
```
# Final dataset merged on the single values of the RIC column. Is a 1883 rows × 24 columns dataframe

Vader_num_df
```

In [ ]:
```
# Preparing share price dataset
# Remove all columns between column index 1 to 3

Vader_share_price = Vader_num_df.drop(Vader_num_df.iloc[:, 2:11], axis=1)
```

In [ ]:
```
# Dropping uneccessary columns

Vader_share_price.drop(Vader_share_price.iloc[:, 2:5], inplace = True, axis=1)
```

In [ ]:
```
# Dropping uneccessary columns

Vader_share_price.drop(Vader_share_price.iloc[:, 5:10], inplace = True, axis=1)
```

In [ ]:
```
# Dropping uneccessary columns

Vader_share_price.drop(['Index','Current Ratio','Cash & Cash Equivalents - Total'], inplace = True, axis=1)
```

In [ ]:
```
# Dropping uneccessary columns

Vader_share_price
```

Applying ML on Share price dataset with VADER compound scores

```
  Choosing a subset of the data to split in between train and test:

X = Vader_share_price
y = Vader_share_price['Target/Non-Target']

X_train, X_test, y_train, y_test = train_test_split(X, y, shuffle=True, test_size=0.3, random_state=11)
```

In [ ]:
```
# BASELINE MODEL logistic regression

# Baseline performance 2: Logistic regression classifier

lr = LogisticRegression()

# Applying the model to the training data:

lr.fit(X_train,y_train)


# Predict the test model:
labels_lr = lr.predict(X_test)
```

In [ ]:
```
# Let's evaluate the results with accuracy:

print('Logistic Regression Test Accuracy:', accuracy_score(y_test, labels_lr))
print('Logistic Regression Train Accuracy:', accuracy_score(y_train, lr.predict(X_train)))

# Recall - but also precision, f1-score and support:

print(classification_report(y_test, labels_lr))
print(classification_report(y_train, lr.predict(X_train)))

# Confusion matrix:

mat_lr = confusion_matrix(y_test,labels_lr)
sns.heatmap(mat_lr, square=True, annot=True, fmt="d", cbar=False,
            xticklabels=['Non-Target', 'Target'], yticklabels=['Non-Target', 'Target'])

plt.xlabel('Predicted Label')
plt.ylabel('Label')
```

```
  Precision

precision = precision_score(y_test, labels_lr, average=None)
print(precision)

# Recall

recall = recall_score(y_test, labels_lr, average=None)
print(recall)

# F-score
f_score = f1_score(y_test, labels_lr, average=None)
print(f_score)
```

In [ ]:
```
# Random Forest on feature selected dataset with Target

# Create regressor object

regressor = RandomForestRegressor(n_estimators = 100, random_state = 0)

# Applying the model to the training data:

regressor.fit(X_train,y_train)

# Predict the test model:

labels_regressor = regressor.predict(X_test)
```

```
  Let's evalueate the results with accuracy:

print('Random Forest Test Accuracy:', accuracy_score(y_test, labels_regressor))
print('Random Forest Train Accuracy:', accuracy_score(y_train, regressor.predict(X_train)))

# Recall - but also precision, f1-score and support:

print(classification_report(y_test, labels_regressor))
print(classification_report(y_train, regressor.predict(X_train)))

# Confusion matrix:

mat_regressor = confusion_matrix(y_test,labels_regressor)
sns.heatmap(mat_regressor, square=True, annot=True, fmt="d", cbar=False,
           xticklabels=['Non-Target', 'Target'], yticklabels=['Non-Target', 'Target'])

plt.xlabel('Predicted Label')
plt.ylabel('Label')
```

In [ ]:
```
# Precision

precision = precision_score(y_test, labels_regressor, average=None)
print(precision)

# Recall

recall = recall_score(y_test, labels_regressor, average=None)
print(recall)

# F-score
f_score = f1_score(y_test, labels_regressor, average=None)
print(f_score)
```

```
  Neural Network NN
# Building the classifier

mlp = MLPClassifier(hidden_layer_sizes=(13,13,13),max_iter=500)

# Applying the model to the training data:

mlp.fit(X_train,y_train)

# Predict the test model:

labels_mlp = mlp.predict(X_test)
```

In [ ]:
```
# Let's evalueate the results with accuracy:

print('NN Test Accuracy:', accuracy_score(y_test, labels_mlp))
print('NN Train Accuracy:', accuracy_score(y_train, mlp.predict(X_train)))

# Recall - but also precision, f1-score and support:

print(classification_report(y_test, labels_mlp))
print(classification_report(y_train, mlp.predict(X_train)))

# Confusion matrix:

mat_mlp = confusion_matrix(y_test,labels_mlp)
sns.heatmap(mat_mlp, square=True, annot=True, fmt="d", cbar=False,
           xticklabels=['Non-Target', 'Target'], yticklabels=['Non-Target', 'Target'])

plt.xlabel('Predicted Label')
plt.ylabel('Label')
```

In [ ]:
```
# Applying nested cross-validation check:
scores_mlp = cross_val_score(mlp, X, y, cv=10, scoring='accuracy')
print(scores_mlp)
print("%0.4f accuracy with a standard deviation of %0.4f" % (scores_mlp.mean(), scores_mlp.std()))
```

```
  Precision

precision = precision_score(y_test, labels_mlp, average=None)
print(precision)

# Recall

recall = recall_score(y_test, labels_mlp, average=None)
print(recall)

# F-score
f_score = f1_score(y_test, labels_mlp, average=None)
print(f_score)
```

In [ ]:

```
# Decision Tree

dt = DecisionTreeClassifier(criterion='entropy')

# Applying the model to the training data:

dt.fit(X_train,y_train)

# Predict the test model:

labels_dt = dt.predict(X_test)
```

```
  Let's evalueate the results with accuracy:

print('Decision Tree Test Accuracy:', accuracy_score(y_test, labels_dt))
print('Decision Tree Train Accuracy:', accuracy_score(y_train, dt.predict(X_train)))

# Recall - but also precision, f1-score and support:

print(classification_report(y_test, labels_dt))
print(classification_report(y_train, dt.predict(X_train)))

# Confusion matrix:

mat_dt = confusion_matrix(y_test,labels_dt)
sns.heatmap(mat_dt, square=True, annot=True, fmt="d", cbar=False,
           xticklabels=['Non-Target', 'Target'], yticklabels=['Non-Target', 'Target'])

plt.xlabel('Predicted Label')
plt.ylabel('Label')
```

In [ ]:

```
# Precision

precision = precision_score(y_test, labels_dt, average=None)
print(precision)

# Recall

recall = recall_score(y_test, labels_dt, average=None)
print(recall)

# F-score
f_score = f1_score(y_test, labels_dt, average=None)
print(f_score)
```

```
  Support Vector Machine (SVM)

# Building the linear Support Vector Machine Classifier

Svm = LinearSVC(dual = False, random_state = 0, penalty = 'l1',tol = 1e-5)

Svm.fit(X_train,y_train)

# Predict the test model:

labels_svm = Svm.predict(X_test)
```

In [ ]:
```
# Let's evalueate the results with accuracy:

print('SVM Test Accuracy:', accuracy_score(y_test, labels_svm))
print('SVM Train Accuracy:', accuracy_score(y_train, Svm.predict(X_train)))

# Recall - but also precision, f1-score and support:

print(classification_report(y_test, labels_svm))
print(classification_report(y_train, Svm.predict(X_train)))

# Confusion matrix:

mat_svm = confusion_matrix(y_test,labels_svm)
sns.heatmap(mat_svm, square=True, annot=True, fmt="d", cbar=False,
            xticklabels=['Non-Terget', 'Terget'], yticklabels=['Non-Terget', 'Terget'])

plt.xlabel('Predicted Label')
plt.ylabel('Label')
```

```
  Precision

precision = precision_score(y_test, labels_svm, average=None)
print(precision)

# Recall

recall = recall_score(y_test, labels_svm, average=None)
print(recall)

# F-score
f_score = f1_score(y_test, labels_svm, average=None)
print(f_score)
```

TEXTBLOB

In [ ]:
```
# Merging the TEXTBLOB scores dataset together with the financial datasite on column RIC

Textblob_num_df = num_RIC_features.merge(df[['Polarity','Subjectivity','RIC']], on = 'RIC')
```

In [ ]:
```
Textblob_num_df
```

In [ ]:
```
# Preparing share price dataset
# Remove all columns between column index 1 to 3

Textblob_share_price = Textblob_num_df.drop(Textblob_num_df.iloc[:, 2:11], axis=1)
```

In [ ]:
```
# Dropping uneccessary columns

Textblob_share_price.drop(Textblob_share_price.iloc[:, 2:5], inplace = True, axis=1)
```

In [ ]:
```
# Dropping uneccessary columns

Textblob_share_price.drop(Textblob_share_price.iloc[:, 5:10], inplace = True, axis=1)
```

```
  Dropping uneccessary columns

Textblob_share_price.drop(['Index','Current Ratio','Cash & Cash Equivalents - Total'], inplace = True, axis=1)
```

In [ ]:
```
Textblob_share_price
```

Applying ML on Share price dataset with TEXTBLOB Polarity and Subjectivity scores

In [ ]:
```
# Choosing a subset of the data to split in between train and test:

X = Textblob_share_price
y = Textblob_share_price['Target/Non-Target']

X_train, X_test, y_train, y_test = train_test_split(X, y, shuffle=True, test_size=0.3, random_state=11)
```

In [ ]:
```
# BASELINE MODEL logistic regression

# Baseline performance 2: Logistic regression classifier

lr = LogisticRegression()

# Applying the model to the training data:

lr.fit(X_train,y_train)


# Predict the test model:
labels_lr = lr.predict(X_test)
```

```
  Let's evaluate the results with accuracy:

print('Logistic Regression Test Accuracy:', accuracy_score(y_test, labels_lr))
print('Logistic Regression Train Accuracy:', accuracy_score(y_train, lr.predict(X_train)))

# Recall - but also precision, f1-score and support:

print(classification_report(y_test, labels_lr))
print(classification_report(y_train, lr.predict(X_train)))

# Confusion matrix:

mat_lr = confusion_matrix(y_test,labels_lr)
sns.heatmap(mat_lr, square=True, annot=True, fmt="d", cbar=False,
           xticklabels=['Non-Target', 'Target'], yticklabels=['Non-Target', 'Target'])

plt.xlabel('Predicted Label')
plt.ylabel('Label')
```

In [ ]:
```
# Precision

precision = precision_score(y_test, labels_lr, average=None)
print(precision)

# Recall

recall = recall_score(y_test, labels_lr, average=None)
print(recall)

# F-score
f_score = f1_score(y_test, labels_lr, average=None)
print(f_score)
```

```
  Random Forest on feature selected dataset with Target

# Create regressor object

regressor = RandomForestRegressor(n_estimators = 100, random_state = 0)

# Applying the model to the training data:

regressor.fit(X_train,y_train)

# Predict the test model:

labels_regressor = regressor.predict(X_test)
```

In [ ]:

```
# Let's evalueate the results with accuracy:

print('Random Forest Test Accuracy:', accuracy_score(y_test, labels_regressor))
print('Random Forest Train Accuracy:', accuracy_score(y_train, regressor.predict(X_train)))

# Recall - but also precision, f1-score and support:

print(classification_report(y_test, labels_regressor))
print(classification_report(y_train, regressor.predict(X_train)))

# Confusion matrix:

mat_regressor = confusion_matrix(y_test,labels_regressor)
sns.heatmap(mat_regressor, square=True, annot=True, fmt="d", cbar=False,
           xticklabels=['Non-Target', 'Target'], yticklabels=['Non-Target', 'Target'])

plt.xlabel('Predicted Label')
plt.ylabel('Label')
```

```
  Precision

precision = precision_score(y_test, labels_regressor, average=None)
print(precision)

# Recall

recall = recall_score(y_test, labels_regressor, average=None)
print(recall)

# F-score
f_score = f1_score(y_test, labels_regressor, average=None)
print(f_score)
```

In [ ]:

```
# Neural Network NN
# Building the classifier

mlp = MLPClassifier(hidden_layer_sizes=(13,13,13),max_iter=500)

# Applying the model to the training data:

mlp.fit(X_train,y_train)

# Predict the test model:

labels_mlp = mlp.predict(X_test)
```

```
  Let's evalueate the results with accuracy:

print('NN Test Accuracy:', accuracy_score(y_test, labels_mlp))
print('NN Train Accuracy:', accuracy_score(y_train, mlp.predict(X_train)))

# Recall - but also precision, f1-score and support:

print(classification_report(y_test, labels_mlp))
print(classification_report(y_train, mlp.predict(X_train)))

# Confusion matrix:

mat_mlp = confusion_matrix(y_test,labels_mlp)
sns.heatmap(mat_mlp, square=True, annot=True, fmt="d", cbar=False,
           xticklabels=['Non-Target', 'Target'], yticklabels=['Non-Target', 'Target'])

plt.xlabel('Predicted Label')
plt.ylabel('Label')
```

In [ ]:
```
# Applying nested cross-validation check:
scores_mlp = cross_val_score(mlp, X, y, cv=10, scoring='accuracy')
print(scores_mlp)
print("%0.4f accuracy with a standard deviation of %0.4f" % (scores_mlp.mean(), scores_mlp.std()))
```

In [ ]:
```
# Precision

precision = precision_score(y_test, labels_mlp, average=None)
print(precision)

# Recall

recall = recall_score(y_test, labels_mlp, average=None)
print(recall)

# F-score
f_score = f1_score(y_test, labels_mlp, average=None)
print(f_score)
```

```
  Decision Tree

dt = DecisionTreeClassifier(criterion='entropy')

# Applying the model to the training data:

dt.fit(X_train,y_train)

# Predict the test model:

labels_dt = dt.predict(X_test)
```

In [ ]:
```
# Let's evalueate the results with accuracy:

print('Decision Tree Test Accuracy:', accuracy_score(y_test, labels_dt))
print('Decision Tree Train Accuracy:', accuracy_score(y_train, dt.predict(X_train)))

# Recall - but also precision, f1-score and support:

print(classification_report(y_test, labels_dt))
print(classification_report(y_train, dt.predict(X_train)))

# Confusion matrix:

mat_dt = confusion_matrix(y_test,labels_dt)
sns.heatmap(mat_dt, square=True, annot=True, fmt="d", cbar=False,
           xticklabels=['Non-Target', 'Target'], yticklabels=['Non-Target', 'Target'])

plt.xlabel('Predicted Label')
plt.ylabel('Label')
```

```
  Precision

precision = precision_score(y_test, labels_dt, average=None)
print(precision)

# Recall

recall = recall_score(y_test, labels_dt, average=None)
print(recall)

# F-score
f_score = f1_score(y_test, labels_dt, average=None)
print(f_score)
```

In [ ]:

```
# Support Vector Machine (SVM)

# Building the linear Support Vector Machine Classifier

Svm = LinearSVC(dual = False, random_state = 0, penalty = 'l1',tol = 1e-5)

Svm.fit(X_train,y_train)

# Predict the test model:

labels_svm = Svm.predict(X_test)
```

```
  Let's evalueate the results with accuracy:

print('SVM Test Accuracy:', accuracy_score(y_test, labels_svm))
print('SVM Train Accuracy:', accuracy_score(y_train, Svm.predict(X_train)))

# Recall - but also precision, f1-score and support:

print(classification_report(y_test, labels_svm))
print(classification_report(y_train, Svm.predict(X_train)))

# Confusion matrix:

mat_svm = confusion_matrix(y_test,labels_svm)
sns.heatmap(mat_svm, square=True, annot=True, fmt="d", cbar=False,
           xticklabels=['Non-Terget', 'Terget'], yticklabels=['Non-Terget', 'Terget'])

plt.xlabel('Predicted Label')
plt.ylabel('Label')
```

In [ ]:

```
# Precision

precision = precision_score(y_test, labels_svm, average=None)
print(precision)

# Recall

recall = recall_score(y_test, labels_svm, average=None)
print(recall)

# F-score
f_score = f1_score(y_test, labels_svm, average=None)
print(f_score)
```

Approach B: Financial dataset with sentiment scores

```
  Importing the Variable selected financial dataset created in Appendix B

# Read the pickle file of the cleaned headlines dataset created in Appendix A, into a dataframe

Variable_financial = pd.read_pickle("Variable selected financial dataset")

# Renaming columns

Variable_financial.rename(columns = {"Instrument": "RIC"}, inplace = True)
```

In [ ]:
```
# Droping unneccessary index column

Variable_financial.drop(['Index'],inplace = True, axis=1)
```

In [ ]:
```
Variable_financial
```

Applying ML on full dataset with VADER compound scores

VADER

In [ ]:
```
# Merging the VADER scores dataset together with the financial datasite on column RIC

Vader_num_full_dataset = Variable_financial.merge(Headlines_Vader_ok[['Compound', 'RIC']], on = 'RIC')
```

In [ ]:
```
# Removing duplicated in column RIC

Vader_num_full_dataset.drop_duplicates(subset=['RIC'], inplace = True)
```

In [ ]:
```
# Droping unneccessary RIC column

Vader_num_full_dataset.drop(['RIC'],inplace = True, axis=1)
```

In [ ]:
```
Vader_num_full_dataset
```

```
  Choosing a subset of the data to split in between train and test:

X = Vader_num_full_dataset
y = Vader_num_full_dataset['Target/Non-Target']

X_train, X_test, y_train, y_test = train_test_split(X, y, shuffle=True, test_size=0.3, random_state=11)
```

In [ ]:

```
# BASELINE MODEL logistic regression

# Baseline performance 2: Logistic regression classifier

lr = LogisticRegression()

# Applying the model to the training data:

lr.fit(X_train,y_train)


# Predict the test model:
labels_lr = lr.predict(X_test)
```

In [ ]:

```
# Let's evaluate the results with accuracy:

print('Logistic Regression Test Accuracy:', accuracy_score(y_test, labels_lr))
print('Logistic Regression Train Accuracy:', accuracy_score(y_train, lr.predict(X_train)))

# Recall - but also precision, f1-score and support:

print(classification_report(y_test, labels_lr))
print(classification_report(y_train, lr.predict(X_train)))

# Confusion matrix:

mat_lr = confusion_matrix(y_test,labels_lr)
sns.heatmap(mat_lr, square=True, annot=True, fmt="d", cbar=False,
           xticklabels=['Non-Target', 'Target'], yticklabels=['Non-Target', 'Target'])

plt.xlabel('Predicted Label')
plt.ylabel('Label')
```

```
  Precision

precision = precision_score(y_test, labels_lr, average=None)
print(precision)

# Recall

recall = recall_score(y_test, labels_lr, average=None)
print(recall)

# F-score
f_score = f1_score(y_test, labels_lr, average=None)
print(f_score)
```

In [ ]:
```
# Applying nested cross-validation check:
scores_lr = cross_val_score(lr, X, y, cv=10, scoring='accuracy')
print(scores_lr)
print("%0.4f accuracy with a standard deviation of %0.4f" % (scores_lr.mean(), scores_lr.std()))
```

In [ ]:
```
# Random Forest on feature selected dataset with Target

# Create regressor object

regressor = RandomForestRegressor(n_estimators = 100, random_state = 0)

# Applying the model to the training data:

regressor.fit(X_train,y_train)

# Predict the test model:

labels_regressor = regressor.predict(X_test)
```

```
  Let's evalueate the results with accuracy:

print('Random Forest Test Accuracy:', accuracy_score(y_test, labels_regressor))
print('Random Forest Train Accuracy:', accuracy_score(y_train, regressor.predict(X_train)))

# Recall - but also precision, f1-score and support:

print(classification_report(y_test, labels_regressor))
print(classification_report(y_train, regressor.predict(X_train)))

# Confusion matrix:

mat_regressor = confusion_matrix(y_test,labels_regressor)
sns.heatmap(mat_regressor, square=True, annot=True, fmt="d", cbar=False,
           xticklabels=['Non-Target', 'Target'], yticklabels=['Non-Target', 'Target'])

plt.xlabel('Predicted Label')
plt.ylabel('Label')
```

In [ ]:
```
# Precision

precision = precision_score(y_test, labels_regressor, average=None)
print(precision)

# Recall

recall = recall_score(y_test, labels_regressor, average=None)
print(recall)

# F-score
f_score = f1_score(y_test, labels_regressor, average=None)
print(f_score)
```

```
  Neural Network NN
# Building the classifier

mlp = MLPClassifier(hidden_layer_sizes=(13,13,13),max_iter=500)

# Applying the model to the training data:

mlp.fit(X_train,y_train)

# Predict the test model:

labels_mlp = mlp.predict(X_test)
```

In [ ]:
```
# Let's evalueate the results with accuracy:

print('NN Test Accuracy:', accuracy_score(y_test, labels_mlp))
print('NN Train Accuracy:', accuracy_score(y_train, mlp.predict(X_train)))

# Recall - but also precision, f1-score and support:

print(classification_report(y_test, labels_mlp))
print(classification_report(y_train, mlp.predict(X_train)))

# Confusion matrix:

mat_mlp = confusion_matrix(y_test,labels_mlp)
sns.heatmap(mat_mlp, square=True, annot=True, fmt="d", cbar=False,
           xticklabels=['Non-Target', 'Target'], yticklabels=['Non-Target', 'Target'])

plt.xlabel('Predicted Label')
plt.ylabel('Label')
```

In [ ]:
```
# Applying nested cross-validation check:
scores_mlp = cross_val_score(mlp, X, y, cv=10, scoring='accuracy')
print(scores_mlp)
print("%0.4f accuracy with a standard deviation of %0.4f" % (scores_mlp.mean(), scores_mlp.std()))
```

```
  Precision

precision = precision_score(y_test, labels_mlp, average=None)
print(precision)

# Recall

recall = recall_score(y_test, labels_mlp, average=None)
print(recall)

# F-score
f_score = f1_score(y_test, labels_mlp, average=None)
print(f_score)
```

In [ ]:
```
# Decision Tree

dt = DecisionTreeClassifier(criterion='entropy')

# Applying the model to the training data:

dt.fit(X_train,y_train)

# Predict the test model:

labels_dt = dt.predict(X_test)
```

```
  Let's evalueate the results with accuracy:

print('Decision Tree Test Accuracy:', accuracy_score(y_test, labels_dt))
print('Decision Tree Train Accuracy:', accuracy_score(y_train, dt.predict(X_train)))

# Recall - but also precision, f1-score and support:

print(classification_report(y_test, labels_dt))
print(classification_report(y_train, dt.predict(X_train)))

# Confusion matrix:

mat_dt = confusion_matrix(y_test,labels_dt)
sns.heatmap(mat_dt, square=True, annot=True, fmt="d", cbar=False,
           xticklabels=['Non-Target', 'Target'], yticklabels=['Non-Target', 'Target'])

plt.xlabel('Predicted Label')
plt.ylabel('Label')
```

In [ ]:

```
# Precision

precision = precision_score(y_test, labels_dt, average=None)
print(precision)

# Recall

recall = recall_score(y_test, labels_dt, average=None)
print(recall)

# F-score
f_score = f1_score(y_test, labels_dt, average=None)
print(f_score)
```

```
  Support Vector Machine (SVM)

# Building the linear Support Vector Machine Classifier

Svm = LinearSVC(dual = False, random_state = 0, penalty = 'l1',tol = 1e-5)

Svm.fit(X_train,y_train)

# Predict the test model:

labels_svm = Svm.predict(X_test)
```

In [ ]:

```
# Let's evalueate the results with accuracy:

print('SVM Test Accuracy:', accuracy_score(y_test, labels_svm))
print('SVM Train Accuracy:', accuracy_score(y_train, Svm.predict(X_train)))

# Recall - but also precision, f1-score and support:

print(classification_report(y_test, labels_svm))
print(classification_report(y_train, Svm.predict(X_train)))

# Confusion matrix:

mat_svm = confusion_matrix(y_test,labels_svm)
sns.heatmap(mat_svm, square=True, annot=True, fmt="d", cbar=False,
           xticklabels=['Non-Terget', 'Terget'], yticklabels=['Non-Terget', 'Terget'])

plt.xlabel('Predicted Label')
plt.ylabel('Label')
```

```
  Precision

precision = precision_score(y_test, labels_svm, average=None)
print(precision)

# Recall

recall = recall_score(y_test, labels_svm, average=None)
print(recall)

# F-score
f_score = f1_score(y_test, labels_svm, average=None)
print(f_score)
```

Applying ML on full dataset with TEXTBLOB polarity and subjectivity scores

TEXTBLOB

In [ ]:
```
# Merging the TEXTBLOB scores dataset together with the financial datasite on column RIC

Textblob_num_full_dataset = Variable_financial.merge(df[['Polarity','Subjectivity','RIC']], on = 'RIC')
```

In [ ]:
```
# Removing duplicated in column RIC

Textblob_num_full_dataset.drop_duplicates(subset=['RIC'], inplace = True)
```

In [ ]:
```
# Droping unneccessary RIC column

Textblob_num_full_dataset.drop(['RIC'],inplace = True, axis=1)
```

In [ ]:
```
Textblob_num_full_dataset
```

```
  Choosing a subset of the data to split in between train and test:

X = Textblob_num_full_dataset
y = Textblob_num_full_dataset['Target/Non-Target']

X_train, X_test, y_train, y_test = train_test_split(X, y, shuffle=True, test_size=0.3, random_state=11)
```

In [ ]:

```
# BASELINE MODEL logistic regression

# Baseline performance 2: Logistic regression classifier

lr = LogisticRegression()

# Applying the model to the training data:

lr.fit(X_train,y_train)


# Predict the test model:
labels_lr = lr.predict(X_test)
```

In [ ]:

```
# Let's evaluate the results with accuracy:

print('Logistic Regression Test Accuracy:', accuracy_score(y_test, labels_lr))
print('Logistic Regression Train Accuracy:', accuracy_score(y_train, lr.predict(X_train)))

# Recall - but also precision, f1-score and support:

print(classification_report(y_test, labels_lr))
print(classification_report(y_train, lr.predict(X_train)))

# Confusion matrix:

mat_lr = confusion_matrix(y_test,labels_lr)
sns.heatmap(mat_lr, square=True, annot=True, fmt="d", cbar=False,
            xticklabels=['Non-Target', 'Target'], yticklabels=['Non-Target', 'Target'])

plt.xlabel('Predicted Label')
plt.ylabel('Label')
```

```
  Let's evalueate the results with accuracy:

print('Random Forest Test Accuracy:', accuracy_score(y_test, labels_regressor))
print('Random Forest Train Accuracy:', accuracy_score(y_train, regressor.predict(X_train)))

# Recall - but also precision, f1-score and support:

print(classification_report(y_test, labels_regressor))
print(classification_report(y_train, regressor.predict(X_train)))

# Confusion matrix:

mat_regressor = confusion_matrix(y_test,labels_regressor)
sns.heatmap(mat_regressor, square=True, annot=True, fmt="d", cbar=False,
           xticklabels=['Non-Target', 'Target'], yticklabels=['Non-Target', 'Target'])

plt.xlabel('Predicted Label')
plt.ylabel('Label')
```

In [ ]:
```
# Precision

precision = precision_score(y_test, labels_regressor, average=None)
print(precision)

# Recall

recall = recall_score(y_test, labels_regressor, average=None)
print(recall)

# F-score
f_score = f1_score(y_test, labels_regressor, average=None)
print(f_score)
```

```
  Precision

precision = precision_score(y_test, labels_lr, average=None)
print(precision)

# Recall

recall = recall_score(y_test, labels_lr, average=None)
print(recall)

# F-score
f_score = f1_score(y_test, labels_lr, average=None)
print(f_score)
```

In [ ]:
```
# Applying nested cross-validation check:
scores_lr = cross_val_score(lr, X, y, cv=10, scoring='accuracy')
print(scores_lr)
print("%0.4f accuracy with a standard deviation of %0.4f" % (scores_lr.mean(), scores_lr.std()))
```

In [ ]:
```
# Random Forest on feature selected dataset with Target

# Create regressor object

regressor = RandomForestRegressor(n_estimators = 100, random_state = 0)

# Applying the model to the training data:

regressor.fit(X_train,y_train)

# Predict the test model:

labels_regressor = regressor.predict(X_test)
```

```
  Neural Network NN
# Building the classifier

mlp = MLPClassifier(hidden_layer_sizes=(13,13,13),max_iter=500)

# Applying the model to the training data:

mlp.fit(X_train,y_train)

# Predict the test model:

labels_mlp = mlp.predict(X_test)
```

In [ ]:
```
# Let's evalueate the results with accuracy:

print('NN Test Accuracy:', accuracy_score(y_test, labels_mlp))
print('NN Train Accuracy:', accuracy_score(y_train, mlp.predict(X_train)))

# Recall - but also precision, f1-score and support:

print(classification_report(y_test, labels_mlp))
print(classification_report(y_train, mlp.predict(X_train)))

# Confusion matrix:

mat_mlp = confusion_matrix(y_test,labels_mlp)
sns.heatmap(mat_mlp, square=True, annot=True, fmt="d", cbar=False,
           xticklabels=['Non-Target', 'Target'], yticklabels=['Non-Target', 'Target'])

plt.xlabel('Predicted Label')
plt.ylabel('Label')
```

In [ ]:
```
# Applying nested cross-validation check:
scores_mlp = cross_val_score(mlp, X, y, cv=10, scoring='accuracy')
print(scores_mlp)
print("%0.4f accuracy with a standard deviation of %0.4f" % (scores_mlp.mean(), scores_mlp.std()))
```

```
  Precision

precision = precision_score(y_test, labels_mlp, average=None)
print(precision)

# Recall

recall = recall_score(y_test, labels_mlp, average=None)
print(recall)

# F-score
f_score = f1_score(y_test, labels_mlp, average=None)
print(f_score)
```

In [ ]:
```
# Decision Tree

dt = DecisionTreeClassifier(criterion='entropy')

# Applying the model to the training data:

dt.fit(X_train,y_train)

# Predict the test model:

labels_dt = dt.predict(X_test)
```

```
  Let's evalueate the results with accuracy:

print('Decision Tree Test Accuracy:', accuracy_score(y_test, labels_dt))
print('Decision Tree Train Accuracy:', accuracy_score(y_train, dt.predict(X_train)))

# Recall - but also precision, f1-score and support:

print(classification_report(y_test, labels_dt))
print(classification_report(y_train, dt.predict(X_train)))

# Confusion matrix:

mat_dt = confusion_matrix(y_test,labels_dt)
sns.heatmap(mat_dt, square=True, annot=True, fmt="d", cbar=False,
           xticklabels=['Non-Target', 'Target'], yticklabels=['Non-Target', 'Target'])

plt.xlabel('Predicted Label')
plt.ylabel('Label')
```

In [ ]:
```
# Precision

precision = precision_score(y_test, labels_dt, average=None)
print(precision)

# Recall

recall = recall_score(y_test, labels_dt, average=None)
print(recall)

# F-score
f_score = f1_score(y_test, labels_dt, average=None)
print(f_score)
```

```
  Support Vector Machine (SVM)

# Building the linear Support Vector Machine Classifier

Svm = LinearSVC(dual = False, random_state = 0, penalty = 'l1',tol = 1e-5)

Svm.fit(X_train,y_train)

# Predict the test model:

labels_svm = Svm.predict(X_test)
```

In [ ]:
```
# Let's evalueate the results with accuracy:

print('SVM Test Accuracy:', accuracy_score(y_test, labels_svm))
print('SVM Train Accuracy:', accuracy_score(y_train, Svm.predict(X_train)))

# Recall - but also precision, f1-score and support:

print(classification_report(y_test, labels_svm))
print(classification_report(y_train, Svm.predict(X_train)))

# Confusion matrix:

mat_svm = confusion_matrix(y_test,labels_svm)
sns.heatmap(mat_svm, square=True, annot=True, fmt="d", cbar=False,
            xticklabels=['Non-Terget', 'Terget'], yticklabels=['Non-Terget', 'Terget'])

plt.xlabel('Predicted Label')
plt.ylabel('Label')
```

```
  Precision

precision = precision_score(y_test, labels_svm, average=None)
print(precision)

# Recall

recall = recall_score(y_test, labels_svm, average=None)
print(recall)

# F-score
f_score = f1_score(y_test, labels_svm, average=None)
print(f_score)
```

Hypothesis testing: High share prices, together with no or very low positive news sentiment, is an indication of an imminent M&A announcement.

VADER

In [ ]:
```
# Filtering the dataset for ontly Target companies

Vader_share_target = Vader_share_price[Vader_share_price['Target/Non-Target'] == 1]
```

In [ ]:
```
# Scaling my dataset for better visualization

scaler = MinMaxScaler()
scaler.fit(Vader_share_target)
scaled = scaler.transform(Vader_share_target)
scaled_df = pd.DataFrame(scaled, columns=Vader_share_target.columns)

scaled_df
```

```
  Plotting VADER Compound scores together with Price To Sales Per Share (Daily Time Series Ratio)

scaled_df.plot(x="Compound", y=["Price To Sales Per Share (Daily Time Series Ratio)"], kind="kde", figsize=(9, 8))

plt.xlabel('Compound scores', fontsize=10)
plt.ylabel('Density', fontsize=10)
plt.title('VADER Compound scores and Price To Sales Per Share (Daily Time Series Ratio)' , fontsize=20)
```

In [ ]:
```
# Plotting VADER Compound scores together with Price To Book Value Per Share (Daily Time Series Ratio)

scaled_df.plot(x="Compound", y=["Price To Book Value Per Share (Daily Time Series Ratio)"], kind="kde", figsize=(9,

plt.xlabel('Compound scores', fontsize=10)
plt.ylabel('Density', fontsize=10)
plt.title('VADER Compound scores and Price To Book Value Per Share (Daily Time Series Ratio)' , fontsize=20)
plt.legend(bbox_to_anchor=(0.5, 0.9))
```

TEXTBLOB

In [ ]:
```
# Filtering the dataset for ontly Target companies

Textblob_share_target = Textblob_share_price[Textblob_share_price['Target/Non-Target'] == 1]
```

In [ ]:
```
# Scaling my dataset for better visualization

scaler = MinMaxScaler()
scaler.fit(Textblob_share_target)
scaled = scaler.transform(Textblob_share_target)
scaled_df = pd.DataFrame(scaled, columns=Textblob_share_target.columns)

scaled_df
```

```
  Plotting TEXTBLOB Polarity and Subjectivity scores together with Price To Sales Per Share (Daily Time Series Rati

scaled_df.plot(x="Polarity", y=['Subjectivity', "Price To Sales Per Share (Daily Time Series Ratio)"], kind="kde",

plt.xlabel('Polarity scores', fontsize=10)
plt.ylabel('Density', fontsize=10)
plt.title('TEXTBLOB Polarity and Subjectivity scores and Price To Sales Per Share (Daily Time Series Ratio)' , font
```

In [ ]:
```
# Plotting TEXTBLOB Polarity and Subjectivity scores together with Price To Book Value Per Share (Daily Time Series

scaled_df.plot(x="Polarity", y=['Subjectivity', "Price To Sales Per Share (Daily Time Series Ratio)"], kind="kde",

plt.xlabel('Polarity scores', fontsize=10)
plt.ylabel('Density', fontsize=10)
plt.title('TEXTBLOB Polarity and Subjectivity scores and Price To Book Value Per Share (Daily Time Series Ratio)'
```