**UNIVERSITY AMERICAN COLLEGE SKOPJE**

**SCHOOL OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY**

**A blue and pink logo

Description automatically generated**

**Predicting Stock Market Trends Using Deep Learning**

**Graduation Thesis**

****

**Author: Jovan Karajanov**

**Supervisor:Prof.Igor Lazov,PhD**

**Member: Marija Stankova Medarovska**

**UNIVERSITY AMERICAN COLLEGE SKOPJE**

**SCHOOL OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY**

**Academic Year 2023/24**

**GRADUATE of Computer Science and Information Technology**

**Graduation Thesis**

**Predicting Stock Market Trends Using Deep Learning**

**Author: Jovan Karajanov**

**Supervisor:Marija Stankova Medarovska**

**Graduation Thesis Committee:**

1. **Supervisor: Igor Lazov, PhD**
2. **Collaborator: Marija Stankova Medarovska**

**Date of Graduation Exam: July, 2024**

**Grade suggestion**

* Pass
* Pass with credit
* Pass with merit
* Pass with distinction

**Awarded \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Acknowledgments**

I would like to express my sincere gratitude to those who supported me throughout the completion of this project. First and foremost, I extend my deepest thanks to Prof.Marija Stankova Medarovska, whose guidance, expertise, and encouragement were a key factor in bringing this work to reality. Their valuable insights and constructive feedback significantly enhanced the quality of this research. Additionally, I am grateful to the Dean Dr.Vladimir Radevski of School Of Computer Science And Information Technology at University American College Skopje and also my supervisor prof.Igor Lazov, whose unwavering support and leadership have fostered an environment conducive to learning and innovation. This project would not have been possible without the collective encouragement and resources provided by these individuals. My appreciation goes to everyone who contributed directly or indirectly to this endeavor.

Table of Contents

Table of Contents

Abstract. 8

Introduction. 9

Problem statement 10

Outline of the thesis 11

Overview of Technologies and Tools 12

Problem solving 14

Conclusion 34

Bibliography 36

**List of Figures**

Figure 1 Figure 1 Python’s ecosystem in machine learning/deep learning 11

Figure 2 Development Environment 12

Figure 3 Showcase of the EDA cleaned data 13

Figure 4 Sectors and their count 15

Figure 5 Pie chart for top 10 Sectors 15

Figure 6 Max values in each numerical column 16

Figure 7 Histogram for Price distribution 17

Figure 8 Histogram for Market Cap distribution 17

Figure 9 Histogram for Earning/Share distribution 18

Figure 10 Histogram for Dividend Yield distribution 19

Figure 11 Distribution of Market Cap 21

Figure 12 Distribution of Price 21

Figure 13 Scatter around mean equation 23

Figure 14 LSTM model architecture 25

Figure 15 LSTM model Actual vs Predicted Prices 26

Figure 16 LSTM multivariate model 27

Figure 17 CNN model architecture 29

Figure 18 Hybrid CNN Actual price vs Predicted Price graph 31

# Abstract

This thesis was inspired by the extreme volatility of the stock market and understanding that not everything can be predicted. The stock market can go up or down in a moment leaving investors scrambling to figure out why. By the big curiosity about how accurately deep learning models, specifically LSTM and CNN, predict stock market trends using S&P 500 historical data, despite the inherent unpredictability in market movements, the goal for this project is to look at how deep learning tools Long Short-Term Memory (LSTM) networks and Convolutional Neural Networks (CNNs), might give us a better grip on stock trends. By crunching numbers from S&P 500 companies past data, I want to create models that learn from history and make useful guesses about future price shifts and then compare them to find out which one is the better approach. But it's key to remember that perfect prediction isn't possible: no computer program can account for every economic curveball, world event, or market mood swing. Through careful data gathering, model training, and checking how well they work, this project aims to show what deep learning can and can't do in a world where even the best guesses can miss the mark.

# Introduction

Investors, economists, and researchers have always been pulled in by the task of forecasting the stock market. Even pros get cautious about guessing because the market can take wild, sharp turns out of nowhere. Old-school methods, like looking at moving averages or keeping tabs on trading volume, often miss the mark when it comes to catching the sudden changes that drive market behavior. But here's the thing: they sometimes miss the big picture. All the messy stuff that makes the market fluctuate when weird and unpredictable world-wide happenings occur, quick tech moves, or investor moods flips for some reason.

This project aims to explore how modern deep learning techniques, such as neural networks can uncover deeper patterns within the data, and give a more profound prediction for active investors or guide and inspire new people to get into this world of stock market trading. By neural networks, we specifically point to Long Short-Term Memory (LSTM) networks and Convolutional Neural Networks (CNNs). By taking advantage of these models, we can easily analyze a big chunk of stock performance data, and use it in our case, predicting future trends. The LSTM network is perfectly suited for this kind of computing and analyzing so called time-series data, as they can remember information over multiple sequences. On the other hand CNNs are typically used for finding patterns in images. As the use case for this project is the CNN is adapting to a one-dimensional time-series data by going through sequential points (the features we use) and treating them as a structured “image”. By this implementation, CNN can identify short-term changes, such as sudden spikes or dips in price and volume. By combining and comparing these two models it helps us see a more improved forecast of the stock market.

However, it is very important to accept the unpredicted movements that can occur in the financial world. This means that no matter how advanced a model or a formula is, we can’t account for every unforeseen event or market anomaly.

Through this research, we will demonstrate that implementing deep learning models can in fact enhance the analysis for the investors.

## *Problem Statement*

The stock market is a subject to some very unpredictable and drastic swings under the influence of a vast number of reasons like global economic events, political developments, corporate announcements, the unpredictable nature of investor psychology and investor sentiment with little warning. Traditional forecasting tools, examining moving averages, support and resistance levels or trading volumes, usually provide only a narrow view of these sudden changes. They tend to mean well but overly rely on linear assumptions and simplified market indicators which gives way to inaccuracies when unexpected events come about or variables interact in non-linear ways. Furthermore, the vast amount of financial data available is also a challenge. As a result, investors are flooded with information like never before, from second-by-second almost instant price movements to historical market trends dating back years and decades. Traditional analytical methods may have trouble sifting through vast datasets effectively and identifying the subtle, nonlinear relationships that might indicate in advance that stock market behavior is about to change.

This thesis takes into account the need for something more advanced in this field, such as diving into the use of deep learning, specifically Long Short-Term Memory (LSTM) and Convolutional Neural Network (CNN) models.

However, to achieve higher accuracy of predictions, advanced machine learning approaches, specifically deep learning, have been piloted. These methods try to “learn” hidden patterns and correlations that simple approaches might miss by training deep neural networks on historical data. Deep learning, however, is not a magic bullet. Events like an unexpected recession or a shock to technology and investor sentiment are what do not allow for perfect predictions, which no kind of technology can perfectly predict.

## *Outline of the Thesis*

The outline of this thesis is intended to guide the reader from a general comprehension of the stock market market prediction challenges the particular deep learning methods used and their results. The Problem Statement goes into further detail on the precise problems this research aims to solve after the Introduction, which establishes the scene by stressing the significance of precise market forecasts and the particular challenges they offer. It also explains why modern machine learning techniques like deep learning, show a slight promise and why traditional methods frequently fail. It explains why deep learning can provide a more valuable alternative and it is setting the stage for comparing the two leading models of my research: LSTM and CNN.

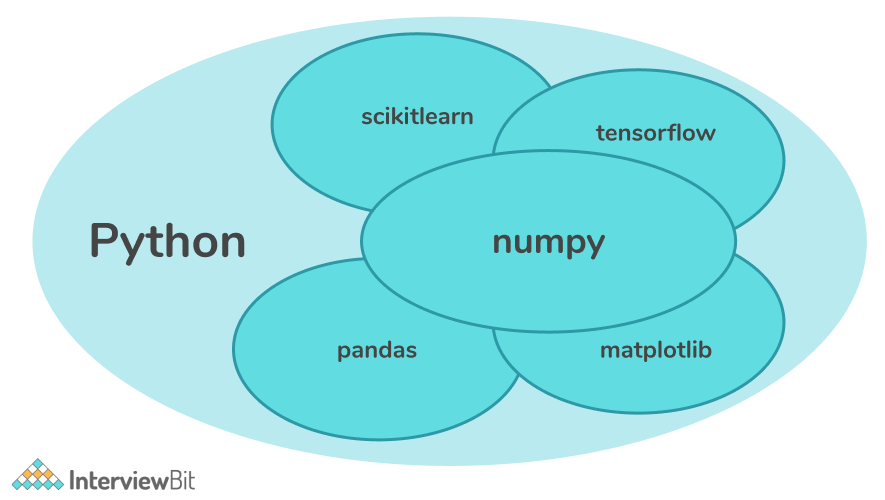
This research begins by cleaning all of the data thoroughly and carefully, so we don’t use “broken” and inconsistent data to feed into our models, this technique is called Exploratory Data Analysis (EDA). After this important starting point I decided to do a Univariate Analysis, which actually focuses on one point of the dataset. By doing this I focused on getting the Top 10 sectors by count and distributed them by industries for example, Financials, Industrials, Information Technology and so on. By implementing the Univariate Analysis, it helped me visualize the dataset and its numbers.

After establishing the data consistency the thesis delves into the architectures of the LSTM and CNN models.

Finally, the thesis ends up with the comparison of LSTM and CNN accuracy, which will highlight the model’s advantages, disadvantages and usefulness for actual stock market forecasting. The conclusion of the thesis summarizes the most important conclusion, acknowledging the unpredictability of the stock market.

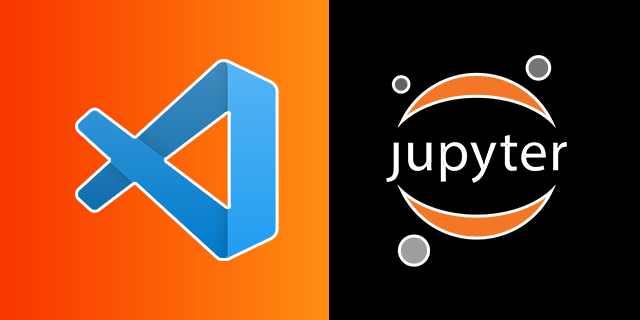
# Overview of Technologies and Tools

For my capstone project, the programming language Python was the bread and butter of making this all possible. Created by Guido van Rossum which he released it in 1991, which has evolved significantly up to now. Thanks to its readability and simplicity, making it easier to test and implement machine learning algorithms. Python is very well adopted in data science and machine learning, due to a huge ecosystem of libraries and frameworks. Beyond the programming language itself, a huge role in my project played the Pandas library, allowing me to load, clean and manipulate to my liking the dataset quickly. Also I relied on the NumPy library as well, for handling multi-dimensional arrays. Once the data was in good shape, I used TensorFlow which is an open-source library developed by Google for deep learning applications and Keras which is also an open-source library for neural networks and are known as the two of Python’s most popular deep learning frameworks. TensorFlow offers a powerful backend engine capable of distributing computations across CPUs, GPUs, or specialized hardware like TPUs, which are more efficient than CPUs and GPUs for AI task. Keras integrated into TensorFlow, provides a user-friendly, high-level interface that reduces boilerplate code, enabling faster prototyping of intricate network architectures like LSTM and CNN. These two libraries helped me construct and train my deep learning models. I used probably the most useful library for this project called scikit-learn or sklearn, used for statistical modeling and so much more. For the visualization side, I used Matplotlib for building plots and Seaborn took these plots to the next level.

Figure 1 Python’s ecosystem in machine learning/deep learning

My dataset for this project is in CSV (comma-separated values) file format, which are the go-to data storing file system that will be used for training machine learning models.

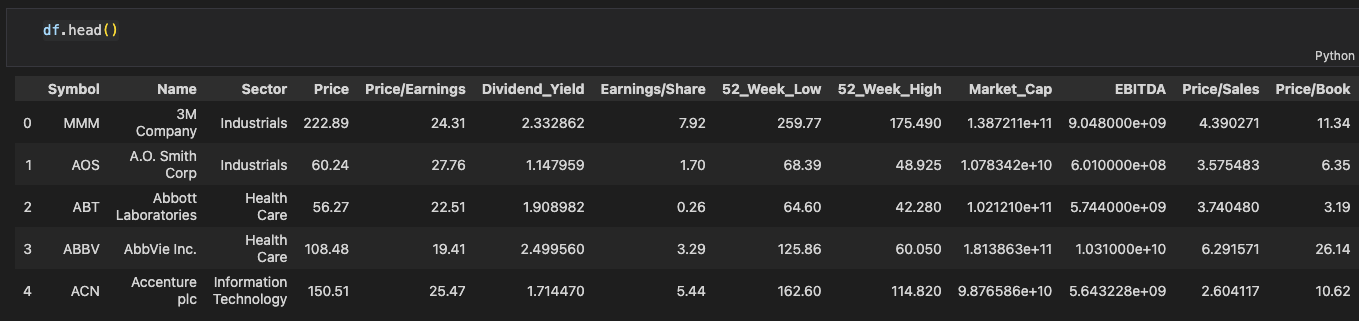
As for the development environment I used a Visual Studio Code as my code text editor with Jupiter Notebook as my combination for my project's implementation and experimental stages. I was able to rapidly test code, perform data exploration stages, and view outputs in real time thanks to Jupyter Notebook's interactive workspace. With this configuration, I could easily experiment with other deep learning architectures, including CNNs and LSTMs, because I could modify the code, run it right away, and observe the model's response without ever leaving my notebook. Visual Studio Code, on the other hand, was quite helpful for better organized programming. Integrated version control, debugging, and a vast marketplace of extensions tailored to Python and data science workflows are just a few of its strong points.

Figure 2 Development Environment

For efficiently to manage my code changes and keep track of all the developments stages I had in this project I used Git as my version control system. Git is a distributed version control system created by Linus Torvalds in 2005, which is the same man that created Linux. It enabled me to store my different point of views of my project in separate branches to avoid the risk of losing progress or getting lost into countless of changes in the code. It made my development process more flexible and organized.

# Problem solving

When it comes down to forecasting stock market movements using two specialized deep learning architectures: Convolutional Neural Networks (CNN) and Long Short-Term Memory (LSTM). Our motivation arises from the limitations of traditional approaches, which frequently fail to detect complex, nonlinear patterns in intricate financial data. We start with a thorough data cleaning phase to eliminate invalid or missing values, then conduct an in depth Exploratory Data Analysis (EDA) that guides our feature engineering efforts, building on earlier research in time-series analysis and deep learning. Exploratory Data Analysis is a crucial first step in this project’s workflow, it involved visualizing the data to understand its main features, to find patterns and to find out how different parts of the data are connected, which will help us spot any inconsistencies and outliers. My starting point in the project, my dataset contained 505 rows and 14 columns, which included a variety of financial features of publicly traded businesses. By analyzing the dataset, with checking for NaN values throughout columns, I found some NaN values which could’ve affected me badly in the long run, after I got that out of the way I checked for duplicate rows occurrence. I noticed I didn’t need some of the columns, so I dropped the “Sec Fillings” column. After I had all the needed columns for my analysis, I decided to rename all of the columns that had spacing between the words and replaced them with “\_”, this step helped me avoid confusion and keep a certain level of consistency throughout my analysis. The EDA helped me shape my dataset that was ready for the next step of the project. We can see by doing a simple command of “df.head()”, how our dataset is shaped up.

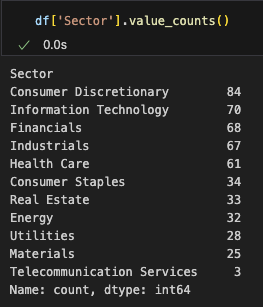
Figure 3 Showcase of the EDA cleaned data

As we can see the dataset is fitting into our need and liking, and with this crucial first step out of the way we can explore specific values which we want more deeply and thoroughly.

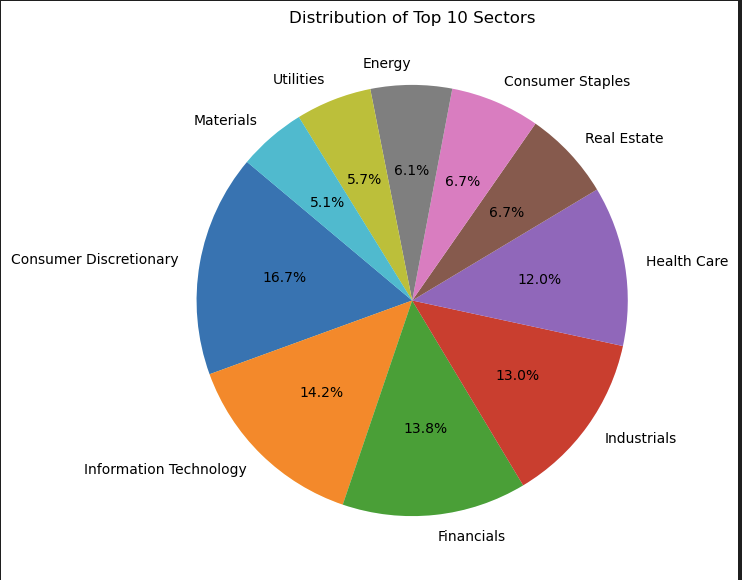
Before delving into more intricate correlations between features, the univariate analysis takes place.

Univariate Analysis involves looking at each variable in a dataset individually to comprehend its fundamental characteristics, such as distribution, central tendency, and variability. Hence the name - only one (“uni”) variable (“variate”) to summarize or describe the variable. Understanding the nature of the data we have and getting it ready for modeling requires this kind of analysis. Univariate analysis is frequently used in deep learning to examine target variables (dependent variables) and input features (independent variables) independently. This knowledge can then be used to guide preprocessing procedures like transformation, scaling, and normalization. For a number of reasons, univariate analysis is crucial in deep learning. By offering insights into the distribution, central tendency, and variability of specific attributes, it aids in developing a deeper comprehension of the data. Finding anomalies in the dataset, such as errors, missing values, or outliers, is another crucial function of this study. Univariate analysis also aids in feature engineering by pointing out which characteristics can benefit from changes like binning or logarithmic scaling in order to improve model performance. By examining the target variable to comprehend its distribution and possible difficulties, like class imbalance in classification tasks, it also helps to improve the interpretability of the model. Last but not least, univariate analysis helps guide preprocessing choices by assisting in determining whether scaling, normalization, or standards is required in light of the data's observed distributions. The goal from this step is to summarize and describe the data for a specific variable without actually considering relationships with other variables. In order to gain a deeper grasp of the dataset’s sector distribution and important financial indicators, the univariate analysis in this project focuses on both categorical and numerical aspects.

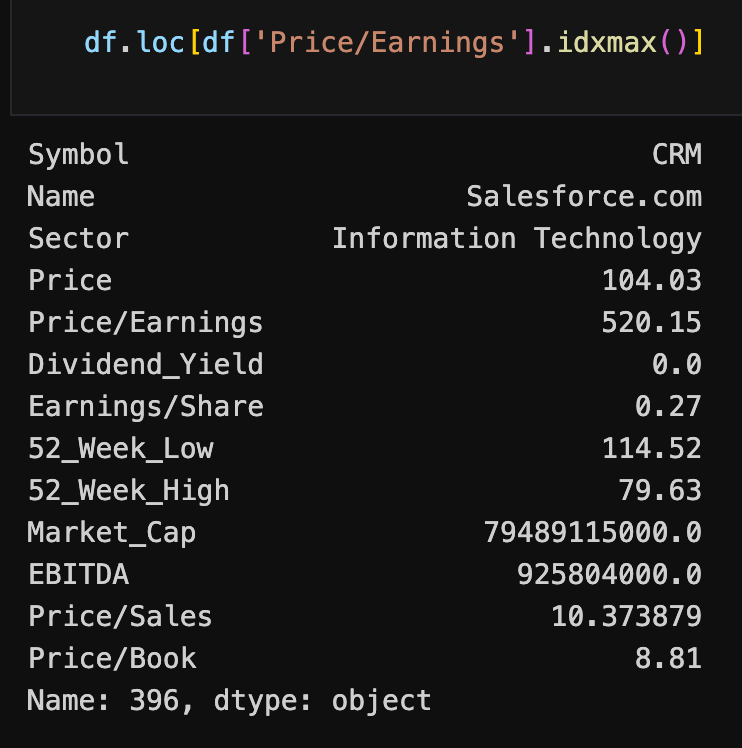
To better grasp the current state of the dataset, I analyzed the “Sector” column, which is the indicator of the distinct industries for each companies. I concentrated my research on the top 10 sectors with the highest representation because of the dataset’s enormous number of sectors. A bar chart showing the number of companies operating in these top 10 sectors is shown in Figure 4, which shows a notable difference in sector sizes. By entering “df[‘Sector’].value\_counts()” I had listed all of the sectors with its count of companies operating in that sector.

Figure 4 Sectors and their count

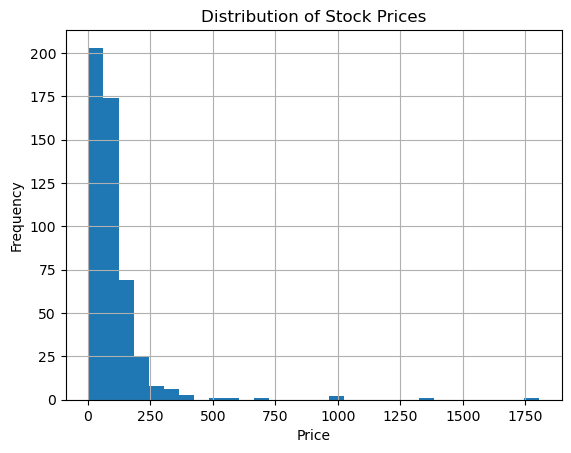
For example, Sector: Consumer Discretionary has the most companies, while Sector: Information Technology is not fat behind. A pie chart displaying the proportionate distribution of companies across different industries is also displayed in Figure 5.

Figure 5 Pie chart for top 10 Sectors

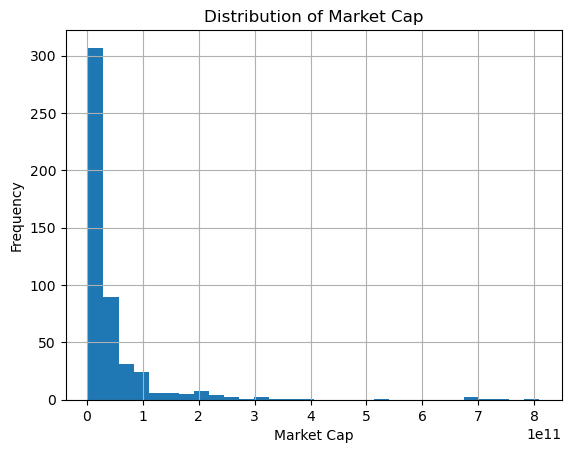
In order to achieve equitable learning across all sectors, these visualizations draw attention to an imbalance in sector representation, which may call for techniques like oversampling or weighting during model training. The dataset's price, market cap, earnings per share, dividend yield, and price/earnings were among the numerical features I looked at. I computed descriptive statistics, such maximum values, for every attribute and used histograms to show their distributions. The maximum values recorded for every numerical column are compiled in Figure 5. Large firms were included in the dataset, as seen by the Market Cap feature's remarkably high values. Similar to this, there was a great deal of variation in the Price feature, with some businesses reporting very high stock prices.

Figure 6 Max values in each numerical column

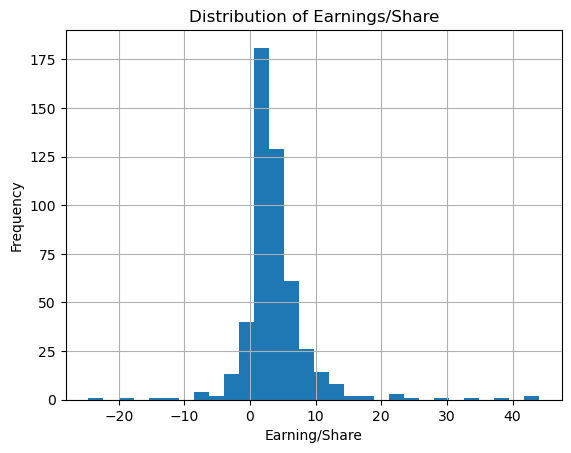
I used histograms to display the distributions of the numerical features in order to obtain a thorough grasp of the underlying patterns and variability in the data. For deep learning preprocessing and model creation, these visualizations offer vital insights about the data's central tendencies, spread, and forms. The histogram of the Price feature in Figure 7 showed a right-skewed distribution, suggesting that most businesses have comparatively low stock prices, while a select few had unusually high prices. This skewness indicates the existence of outliers that can have an excessive impact on the model when it is being trained. Transformations like normalization or logarithmic scaling could be required to improve model convergence and stabilize the variance in order to lessen this effect.

Figure 7 Histogram for Price distribution

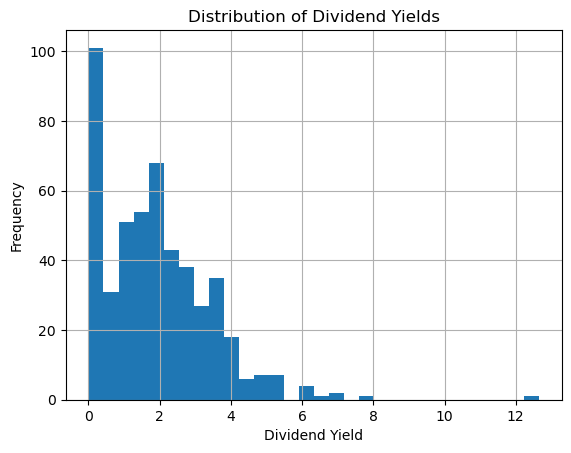
A similar right-skewed pattern could be seen in the Market Cap histogram (Figure 8), which showed that most companies had smaller market capitalizations than a select few huge corporations. This discrepancy shows how diverse the dataset's firm sizes are and emphasizes how crucial scaling strategies are to preventing higher values from controlling the learning process. Furthermore, the high degree of variation in Market Cap numbers can call for additional research into data clusters or subgroups.

Figure 8 Histogram for Market Cap distribution

A bimodal distribution was seen in the Earnings/Share histogram (Figure 9), with one peak centered on positive earnings and another on negative earnings. Companies' financial performance is reflected in this pattern, with some reporting profits and others reporting losses. As was previously mentioned, the existence of negative Earnings/Share values is especially significant and needs to be handled carefully during preprocessing. For example, classifying profitable and unprofitable businesses into different groups may make the model easier to understand and more predictive.

Figure 9 Histogram for Earning/Share distribution

With many companies offering low dividend yields and fewer offering better returns, the Dividend Yield histogram (Figure 10) displayed a left-skewed distribution. Since not all businesses place a higher priority on dividend payments than reinvesting in expansion prospects, this distribution is in line with expectations from the actual world. To make sure the model adequately captures the subtleties of this characteristic, the skewness can suggest that transformation or normalization is required.

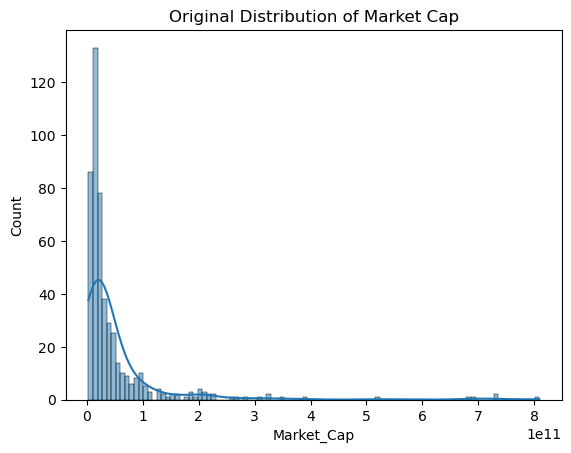
Figure 10 Histogram for Dividend Yield Distribution

The results of the univariate analysis reveal a number of important dataset features that call for careful preprocessing in order to guarantee the best possible performance from the deep learning model. We discuss the ramifications of these discoveries and offer solutions to the problems that have been discovered below.

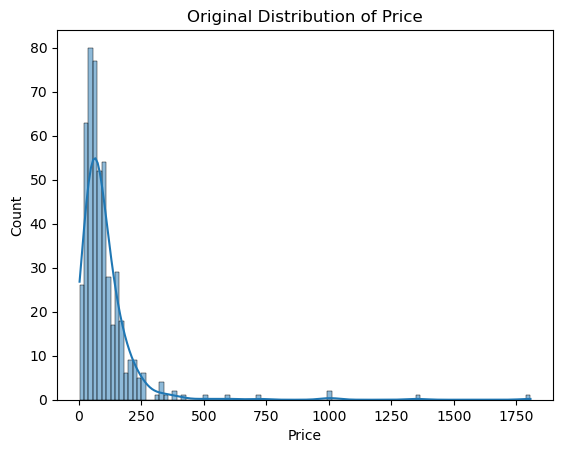
The large diversity of scales across numerical parameters like Price, Market Cap, and Dividend Yield is among the most important findings from the distribution visualizations. While certain features, such Market Cap, have comparatively narrower ranges, others, like Dividend Yield, show incredibly high numbers. Since deep learning algorithms frequently assume that input features are on a same scale, this scale variability may result in less-than-ideal model performance. The data will be rescaled using normalization or standards procedures to remedy this. While standardization (z-score scaling) will change the data to have a zero mean and unit variance, normalization (min-max scaling) will guarantee that all characteristics fall inside a predetermined range. The particular needs will determine which of these approaches is best.

A number of numerical features were found to have outliers, most notably Price and Market Cap. Although they may be instructive, these extreme values have the potential to distort learning and control model parameter updates. We suggest using strong outlier handling strategies to lessen their effects. By establishing upper and lower bounds, capping or winsorization, for example, can be utilized to reduce the impact of extreme numbers.

The existence of businesses reporting negative earnings per share and EBITDA was a noteworthy finding from the univariate study. Negative numbers usually indicate losses or operational difficulties, making these measures crucial markers of a business's financial health. Building strong deep learning models that can precisely capture the subtleties of both lucrative and unprofitable entities requires an understanding of the traits and distribution of such businesses within the dataset. A company's financial health might be inferred from negative earnings or EBITDA. While some businesses may be experiencing difficult times, others may be making significant investments in expansion, which may result in temporary losses. By incorporating these businesses into our study, we are able to provide a more comprehensive view of the business world, encompassing both failing and successful organizations. After carefully examining the relations between certain properties by doing histograms, I further analyzed the skewness and kurtosis of the Market Cap and Price properties. Because it offers profound insights into the shape and behavior of the dataset’s distribution, skewness and kurtosis analysis is essential. Knowing these characteristics makes it easier to see possible problems like non-normality or the existence of outliers, which can have a big effect on how well machine learning models work. We can make confident choices regarding preprocessing procedures, including transformations or scaling, to make sure the data is appropriate for modeling by identifying how skewed or peaked the data is. In the end, my study guarantees that the model gains knowledge from precise data representations, resulting in predictions that are more dependable and resilient. Without this knowledge, important data trends could be missed, which would lower the quality of the findings. I discovered that the distributions of the Market Cap and Price data are not at all symmetrical or “normal.”. As we can see from “Figure 11”, the data is heavily skewed to the right for Market Cap. This indicates that while the majority of businesses have lesser market caps, a tiny number of really large businesses push the distribution in the direction of larger values. Additionally, the distribution shows a strong peak with heavy tails, indicating that certain very large organizations stand out, even if most companies cluster around smaller sizes.

Figure 11 Distribution of Market Cap

Compared to Market Cap, the Price data shown in “Figure 12”, exhibits an even more pronounced skew to the right. A few companies have very high stock prices, but the majority have very low prices. There are even more severe values on the higher end of the distribution, which is even more peaked and has thicker tails than Market Cap.

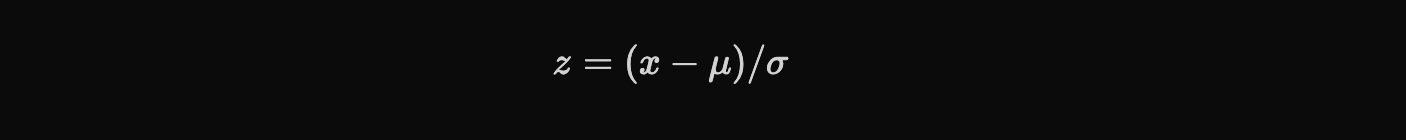
Figure 12 Distribution of Price

Simply put, there are many little values and a few very large ones for both price and market cap. As a result, the data becomes asymmetrical and unequal with long tails. If this isn't done, the extreme numbers could make it more difficult for the model to recognize patterns in the data.

The next thing I did is Feature Preparation. Any deep learning project must include feature preparation since it ensures that the data is in the proper format and shape for the model to learn from. Real-world data is frequently unstructured, irregular, and inappropriate for direct application in deep learning or machine learning models. Poor preparation could make it difficult for the model to recognize patterns in the dataset, which could result in poor results or inaccurate predictions. First of all, the size and format of the input data have an impact on deep learning models. For instance, the model may concentrate too much on the larger-scaled variable and overlook the smaller ones if one feature, like market cap, has values in the billions and another, like price/earnings, has values below 100. The learning process of the model may be distorted by this imbalance. We make sure that every input has an equal impact on the model's decision-making by scaling numerical features. Second, since deep learning models can only handle numerical data, categorical data must be transformed into a numerical format. Approaches like one-hot encoding must be used to change features like “Sector” (such as Technology, Healthcare and more). The model can now understand these categories as unique and significant inputs thanks to this change. Finally, feature preparation helps in resolving data inconsistencies, outliers, and missing values. These issues may cause the model to become confused or produce unreliable findings. We establish a strong basis for training by cleaning and preprocessing the data, guaranteeing that the model learns from accurate and significant information.

In conclusion, feature preparation is required to prepare your data for deep learning. It guarantees that the model can learn from the data efficiently, enhances model performance, and expedites training. Ignoring this step could lead to a model that performs badly or to training effort being wasted. The foundation for a successful deep learning project is preparedness.

So with that being said, I followed a structured approach to prepare my dataset for deep leaning models. The first step was dealing with my dataset’s numerical elements, which included Market Cap and Price/Earning. The scales of these features varied greatly. Some had values in the billions, while others were considerably less. I used the StandardScaler from the “sklearn.preprocessing” library to make sure that no feature overshadowed the others during training. For each feature, the StandardScaler subtracts the mean and divides the result by the standard deviation to standardize the data. Because of this transformation, all numerical features are comparable and simpler for the model to comprehend because they have a mean of 0 and a standard deviation of 1. I helped the model train more efficiently and prevented problems brought on by significant size inconsistencies by scaling the numerical features. I implemented this equation (Figure 13), so I can measure how much data scatter around the mean.

 Figure 13 Scatter around mean equation

Next I delved into categorical features, which I mentioned above. I used One-hot encoding technique to transform the “Sector” property into binary columns for each Sector. Despite the benefits i got from this approach, one-hot encoding can present certain side effects.

The encoded columns may become linearly dependent if there are numerous categories (for example, if a corporation isn't in Technology or Finance, it's probably in Healthcare). The model may become confused by this redundancy. Also this approach can produce a huge number of columns if the Sector column contains a lot of unique values, which will increase training time and memory usage. And some of the categories that are low in occurrences throughout the dataset, might just add noise to the model training. So to address these potential issues, I validated the encodings after applying the one-hot encoding. This step made sure the dataset was efficient and clean by assisting me in locating and eliminating any problematic rows or categories.

Next thing I did is, I utilized “sklearn.compose” ColumnTransformer to speed up the preprocessing step. I was able to apply various transforms to various feature types in a single step thanks to this tool. In particular:

* + I used StandardScaler to scale the numerical features.
  + I used OneHotEncoder to encode the categorized Sector feature.

By using these transformations , I made sure the dataset was prepared effectively and consistently, which will save time and effort in future research.

I carried out further validation after implementing these transformations to make sure the encodings were accurate and free of issues. In the process of this step:

* + I eliminated unnecessary columns and looked for multicollinearity. In order to address dimensionality issues, I eliminated uncommon categories that made no significant contributions to the dataset.
  + One row that had deviations introduced by the encoding was found and eliminated. I saved the cleaned and more trustworthy dataset as “data\_preprocessed\_updated\_with\_encoding\_validation.csv” as a result of these improvements.

To that end, the final dataset included both the scaled numerical features and the one-hot encoded categorical characteristics. With all features properly organized and processed, this dataset was now prepared for input into the deep learning model.

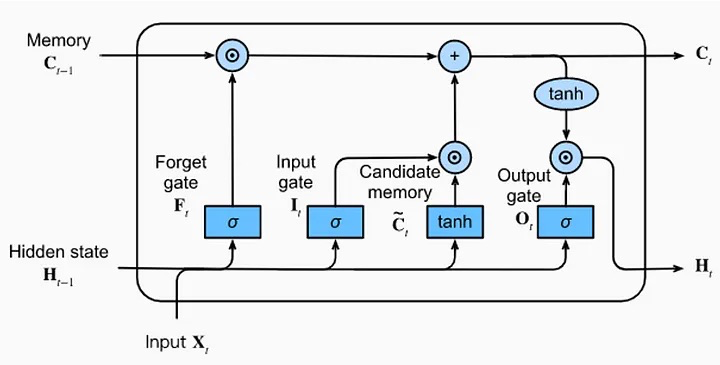
Having said all that, I can start by implementing and training the LSTM model. LSTM or Long Short-Term Memory, is an updated version of the RNN or Recurrent Neural Network, which is designed by Hochreiter & Schmidhuber. LSTMs are perfect for tasks like language translation, speech recognition, and time series forecasting because they can identify long-term dependencies in sequential data. In order to overcome the difficulty of learning long-term dependencies, LSTMs introduce a memory cell that stores information over longer periods of time, in contrast to typical RNNs that employ a single hidden state transferred across time. An LSTM unit is essentially made up of a cell state and a number of key components that cooperate to control the information flow throughout the network. The LSTM is very successful at processing sequences because of these components, which allow it to selectively recall or forget information over time.

Information is carried over time steps through the cell state, which serves as the LSTM's memory. It enables the network to eliminate unnecessary details while keeping significant patterns. The LSTM uses a mechanism called gates to control the flow of information into, out of, and within the cell state in order to manage this process.

Based on the current input and the previously hidden state, one of these gates—the forget gate—decides which portions of the cell state should be deleted. By doing this, the network is able to "forget" irrelevant data and concentrate on relevant patterns. Which new data from the current input should be added to the cell state is determined by another gate called the input gate. This guarantees that the network can add relevant data to its memory. Lastly, at the current time step, the output gate determines which portion of the modified cell state will be sent as the output. As a result, the network can provide forecasts or intermediate outcomes while keeping the remaining data for use in subsequent processes.

By moderating the information flow, the interaction of these elements allows LSTMs to analyze sequential data effectively. By maintaining crucial information in the cell state, eliminating noise or extraneous data, and generating precise outputs at every time step, the gating mechanism enables the network to maintain long-term dependencies.

To visualize what was said, “Figure 14” illustrates the internal structure of a LSTM model. It shows how each component is working together to manage the flow of information.

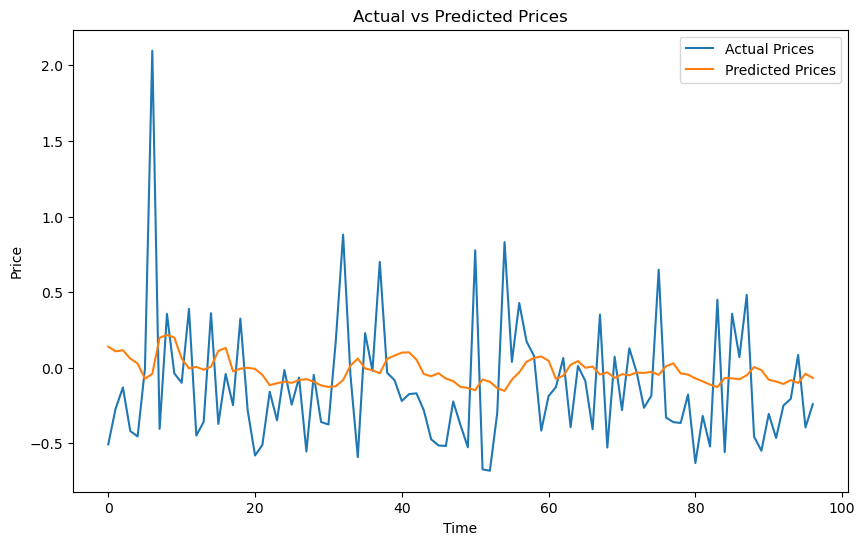
Figure 14 LSTM model architecture

Because the LSTM model is specifically made to handle sequential data and capture long-term dependencies, two critical components of financial time-series research, I decided to use it for my project on stock market trend analysis. Because stock market data is sequential by nature—that is, each data point depends on earlier ones, traditional models sometimes find it difficult to learn from such patterns over long time periods. LSTM is ideally suited for forecasting future trends based on historical data since it can "remember" significant information while removing noise. I can also deal with multiple time frames, such daily or weekly stock prices, thanks to its versatility in handling variable-length sequences, which eliminates the need to modify the input structure. Because of these characteristics, LSTM is the best option for creating a reliable and accurate stock market trend prediction model.

As my first step of predicting the stock market trends, I implemented an univariate LSTM model. I started with preparing the data by creating sequences of historical prices. A function divided the data into training (80%) and testing (20%) sets and produced sequences of a given duration (for example, 10 days). This made it possible for the model to learn from historical price trends.

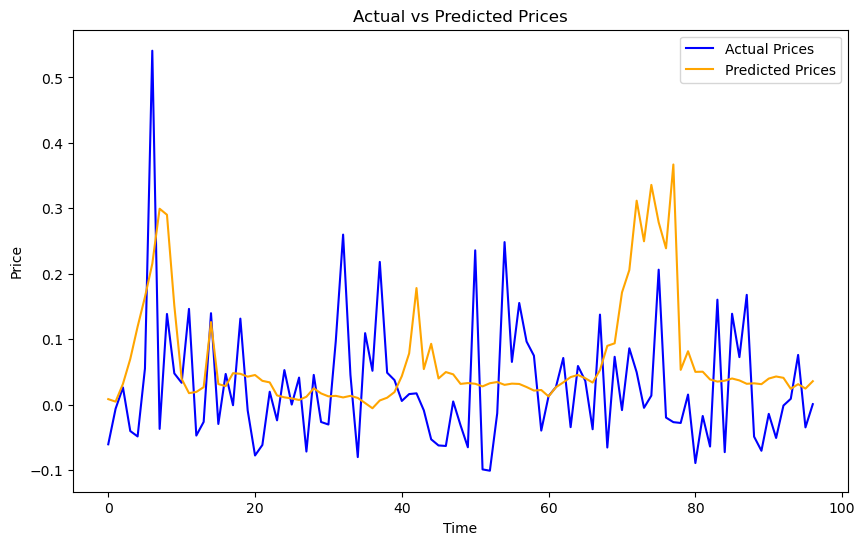
I then added dropout layers to avoid overfitting and constructed the LSTM model with two layers, each of which has 50 memory units. The last layer was a Dense layer with a single neuron for stock price prediction for the following day. The'mean\_squared\_error' loss function and the 'adam' optimizer were used to compile the model. The model was trained over 50 epochs with a batch size of 32, using 20% of the training data for validation to monitor performance and detect overfitting early. After training, I plotted the actual vs predicted prices and calculated the Mean Squared Error (MSE) on the test data to assess the model, which came up to 0.18917252805703294, which is acceptable for my kind of research. This graphic made it easier to evaluate how well the model predicted future pricing.

Given the complexity of financial markets, this method offers a strong basis for forecasting stock values based on past data, but there is still opportunity for improvement as we can see in Figure 15.

Figure 15 LSTM model Actual vs Predicted Prices.

Based on these results I got, I decided to move on with a multivariate LSTM model which is basically a model that have 2 or more dependent variables. I created a multivariate LSTM model to increase prediction accuracy by combining several characteristics rather than depending only on the Price column, building on the univariate LSTM model. When compared to the univariate model, this method usually produces superior results since it increases the data's richness and gives the model more information to learn from.

Along with the price column, other scaled numerical features like market cap and earnings per share, as well as one-hot encoded categorical features like sector\_industrials and sector\_energy, are included in the model's input in this multivariate setup. When the model makes predictions, these extra features enable it to take into account different facets of the data. The model's input shape is (samples, sequence\_length, num\_features), where num\_features is the sum of all the features that were chosen for training. The data was normalized using the StandardScaler method to guarantee consistent scaling of numerical features. The model's primary focus remains on forecasting the Price column's next value, even with the inclusion of numerous features. To make this prediction, it now uses every feature that is available over the given sequence length, which allows it to recognize increasingly intricate patterns. Similar to the univariate version, the multivariate LSTM model maintains its architecture. It is made up of LSTM layers that are intended to process temporal dependencies between various features over the specified length of the sequence. In order to avoid overfitting and guarantee that the model performs well when applied to new data, dropout layers are used. Ultimately, a single value that matches the anticipated price is output by a dense layer. The create\_sequences() function was adapted to handle several features for data preparation. This entails constructing sequences that incorporate every feature dimension so that the model can gradually discover correlations and dependencies between various variables. By upgrading the model to multivariate one, we can see (Figure 16) that the model is actually improving and is following the trends of the stock prices.

Figure 16 LSTM multivariate model

After implementing and training the LSTM, the next model I will implement is CNN. CNN, also called Convolutional Neural Network is a type of a deep learning algorithm that is essentially used for analyzing visual data. However, it is extremely adaptable so that it can be used with different types of structured data, such as time-series data like stock prices, because of its capacity to identify patterns. CNNs handle time-series data in this way as one-dimensional "images," using filters to find significant patterns at various scales and locations. CNNs can be applied to stock market analysis to find trends, cycles, or anomalies in financial data sequences by treating the time-series data as a one-dimensional "image." Because of this, CNNs are an effective tool for examining stock market patterns, particularly when paired with methods like multi-scale feature extraction or sliding windows.

Based on this project the CNN is built according to a particular architecture that is intended to efficiently handle time-series data and identify significant patterns. Fundamentally, CNN is made up of a number of essential parts that cooperate to monitor patterns in the stock market. The convolutional layer, the CNN's initial layer, applies filters, sometimes known as kernels, to the input data. By moving over the data, these filters are able to identify local patterns like transient price changes or spikes in trade activity. The model learns to identify different features at different scales and positions within the data by applying numerous filters.

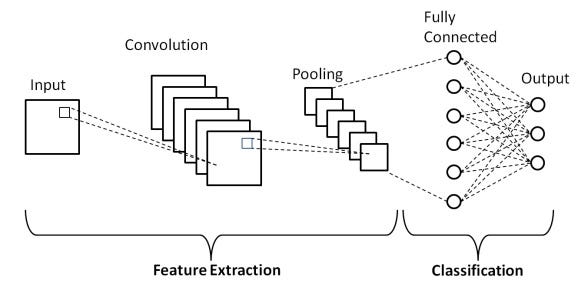
A pooling layer is applied after the convolutional layer to lower the data's dimensionality while keeping the most crucial information. Pooling increases the model's resilience to little changes in the input data and helps it concentrate on important patterns. Max pooling, for instance, highlights the most noticeable features by choosing the greatest value within every area.

To capture complex patterns in the data, more convolutional and pooling layers might be stacked after the pooling layer. The CNN can learn both basic and complex correlations between financial data, thanks to this hierarchical structure.

At last, one or more dense layers are applied after the output from the convolutional and pooling layers has been flattened. To estimate future stock prices, for example, these dense layers integrate the learnt information.

With this being said, “Figure 17” will provide a clear visual of this model.

Figure 17 CNN model architecture



With that being said, I choose CNN for this project because it has number of benefits over different approaches. CNNs can automatically find relevant patterns in the data, such as price trends, spikes in trade volume, or correlations between various financial indicators, rather than depending on manually created features. CNNs effectively identify patterns on a variety of time scales by employing convolutional layers, which allows them to record both short-term swings and long-term trends in stock prices.

I used a hybrid deep learning design that combines the advantages of Long Short-Term Memory networks (LSTMs) and Convolutional Neural Networks (CNNs) to forecast stock values using the CNN-LSTM model. This method enables the model to efficiently capture long-term temporal dependencies (by LSTM) as well as local patterns in the data (via CNN). Preparing the data for the model was the initial step. I segregated the characteristics (all columns except Price) from the target (Price) because the model needs sequential input. To make sure all values lie within the range [0, 1], the StandardScaler was used to scale both the features and the target. By lessening the effect of significant variations in feature scales, scaling makes it possible for the model to learn more efficiently. I used a method called “create\_windows” to generate openings of sequential data after scaling. From the dataset, this method creates sequences of a given length (for example, 10 days). For instance, the model learns how the data from the previous 10 days affects the price the following day if the sequence length is 10. A 3D array (num\_samples, window\_size, and num\_features) that can be fed into the CNN-LSTM model is the end product. After that, the data was divided into testing (15%) and training (85%) sets. This guarantees that, in order to appropriately evaluate the model's performance, it is trained on a subset of the data and tested on unknown data.

The CNN-LSTM model combines the advantages of LSTM layers for capturing temporal dependencies and convolutional layers for pattern extraction. Convolutional layers at the top of the architecture process the incoming data to find local patterns like short-term trends or feature correlations. To lower dimensionality and avoid overfitting, pooling and dropout methods come after these layers.

The collected features are sent to a bidirectional LSTM layer following the convolutional block. In order to enable the model to capture more complex temporal correlations throughout time, this layer processes the data both forward and backward. By decreasing reliance on particular neurons during training, a second dropout layer is introduced to improve generalization even more.

Dense layers are then added to the model to improve its ability to learn intricate relationships. These layers prepare the data for the last prediction step by combining the learned features into higher-level representations. The anticipated stock price is then produced by a single neuron output layer, guaranteeing that the model delivers a continuous value appropriate for regression tasks.

Mean Squared Error (MSE) is used as the loss function and the Adam optimizer is used to compile the model. L2 regularization strength, dropout rate, and learning rate are examples of well selected hyperparameters that strike a compromise between the model's complexity and generalizability.

I implemented two callbacks in order to effectively train the model:

* EarlyStopping: Prevents needless computation by stopping training if the validation loss does not improve for 20 consecutive epochs.
* If the validation loss reaches a particular level, ReduceLROnPlateau dynamically lowers the learning rate to make sure the model keeps learning efficiently.

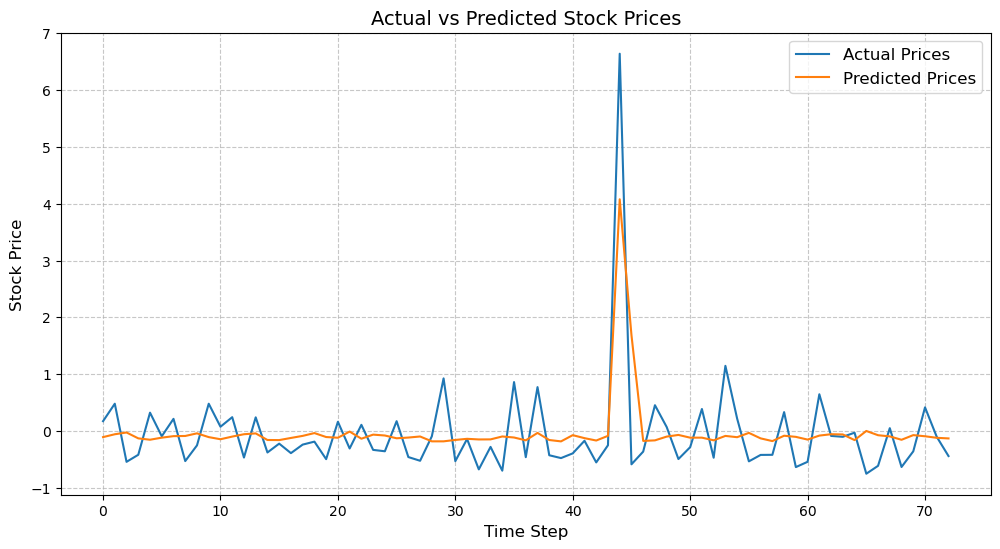
With a batch size of 32, the model was trained for a maximum of 200 epochs. However, when no more progress was shown, the EarlyStopping callback caused training to end early.

After training, the model's performance was evaluated using the test set. In the scaled domain, the test loss (MSE) was 0.3676. The objective of the scaler was used to make predictions and inversely convert them back to the original pricing scale.

To measure the accuracy of the model, a number of evaluation metrics were computed:

* The average squared difference between expected and actual data is measured by the Mean Squared Error (MSE) (0.3385).
* An easier-to-understand indicator of error is the Root Mean Squared Error (RMSE) (0.5818).
* The average absolute difference between expected and actual values is represented by the mean absolute error, or MAE (0.4121).
* The model explains more than 57% of the variability in the target variable, according to the R2 Score, which shows the percentage of variance explained by the model (0.5766).

With that being said we can see how the model turned out below in (Figure 18). The model captures the high spike trends, but occasionally lags in predicting sudden small price spikes.

Figure 18 Hybrid CNN Actual price vs Predicted Price graph

With everything implemented and trained, all that is left is to compare the two models. First I will reflect on my LSTM multivariate model shown on (Figure 16). In terms of effectively managing abrupt price surges and capturing general trends, the LSTM model performs wonderfully. For instance, the LSTM model demonstrates its capacity to react rapidly to sudden shifts by correctly predicting a notable rise in the real prices. Furthermore, for the majority of the time steps, the model stays closely aligned with the real values, demonstrating its ability to accurately forecast both short-term volatility and long-term patterns. On the other hand, the CNN-LSTM model (Figure 18) performs worse in this specific case even though it combines LSTM layers for temporal dependency learning and convolutional layers for local pattern extraction. There are some differences between the real prices (blue line) and the forecast prices (orange dashed line) in the CNN-LSTM model's time series graph, especially during volatile times. For example, there is a noticeable trail between the expected and actual spikes around time step 45, but overall it caught the trend in the right direction.. Additionally, the model tends to smooth out small differences more than the LSTM model, which causes modest price movements to be slightly underestimated. In this particular case, the comparison shows the advantages of the LSTM multivariate model over the CNN-LSTM model. The LSTM multivariate model performs better at catching both abrupt changes and long-term trends, even though the CNN-LSTM model provides a hybrid architecture that blends spatial and temporal information. This emphasizes how crucial it is to choose the best model depending on the particulars of the dataset and the forecasting task at hand. For this stock price prediction problem, the LSTM multivariate model is a better option due to its effective handling of complex sequential data.

# Conclusion

Everything considered, my research’s goal was to explore the potential of deep learning models in managing the extreme level of the stock market’s volatility, a field where unpredictability is something we must accept. The stock market can suddenly shift direction due to a variety of factors, including global events, economic events, and investor psychology. Even experienced investors may wonder why such sudden shifts occur. Given this underlying unpredictability, the purpose of this study was to examine the predictive power of deep learning methods, notably Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks, utilizing historical data from S&P 500 businesses. The goal was to create models that could learn from the past and make accurate forecasts about future price movements by examining historical data, while also taking into account the constraints imposed by the unpredictability of financial markets.

This effort revealed a number of significant discoveries through careful data preparation, model construction, and evaluation. First, a baseline for prediction accuracy was established by the univariate LSTM model, which only used past price data. Performance was significantly enhanced by adding more features, though, like Market Cap, Earnings/Share, and one-hot encoded sector information. The multivariate LSTM model stood out as the best performer, proving that adding relevant characteristics to the dataset is essential to increasing predictive power.

In contrast, the multivariate LSTM performed better in this particular test than the CNN-LSTM model, which was created to combine the advantages of LSTM layers (for temporal dependency learning) with convolutional layers (for feature extraction). Although the hybrid design has potential for other applications, the multivariate LSTM's ease of use and effectiveness made it a better choice for stock price prediction. This emphasizes how crucial it is to choose the best model architecture depending on the data's properties and the issue at hand.

The project's results highlight deep learning's potential and limitations in financial forecasting. Even though stock market data is noisy, non-stationar, and subject to outside influences, deep learning models such as LSTMs exhibit a surprising capacity to spot patterns and trends that conventional approaches could miss. It's important to keep in mind, though, that perfect prediction is still an impossible objective. A geopolitical crisis, a sudden change in consumer behavior, or an unanticipated economic collapse are just a few examples of the unexpected events that no algorithm can predict. In order to make well-informed decisions, deep learning must be combined with human intuition and expertise in the field, even though it offers useful tools for comprehending stock movements.

Although this project has shed light on the use of deep learning models for stock price prediction, there are a number of interesting possibilities for further research. Examining sophisticated structures like transformer-based models or attention mechanisms is one possible direction. These methods might provide fresh perspectives on identifying contextual relationships and long-range dependencies in financial data, which could result in more precise forecasts. Another crucial path is to expand the models to manage real-time data streams. The ability to process and forecast using real-time market data could greatly expand the usefulness of these models in real-world trading settings, facilitating quicker and better-informed decision-making. Also, adding external data sources might provide forecasts a more comprehensive context. The models might be enhanced by adding elements like social media trends, news mood, and macroeconomic data, which would provide a more comprehensive understanding of market dynamics and possibly increase their prediction ability.

To sum up, this experiment clarifies the limits and potential of deep learning in stock market trend prediction. The multivariate LSTM model showed remarkable effectiveness in identifying historical trends and producing insightful predictions, despite the fact that no model can attain perfect foresight because financial markets are chaotic. It was able to balance accuracy and computing economy by utilizing a variety of characteristics and temporal connections. In addition, to adding to the vast filed of stock market, this research highlights how crucial it is to comprehend the capabilities and accept the limitations of these models. It is important to approach predictions with humility as we continue to push the limits of technology in finance, understanding that even the finest algorithms are vulnerable to the unpredictable movements of the market, to say it simpler, getting a 100% correct prediction is impossible.

# Bibliography

Liu, H. and Long, Z., 2020. An improved deep learning model for predicting stock market price time series. *Digital Signal Processing*, *102*, p.102741. Available at: <https://www.sciencedirect.com/science/article/abs/pii/S1051200420300865> (Accessed: 5 March 2024)

Nabipour, M., Nayyeri, P., Jabani, H., Shahab, S. and Mosavi, A., 2020. Predicting stock market trends using machine learning and deep learning algorithms via continuous and binary data; a comparative analysis. *Ieee Access*, *8*, pp.150199-150212. Available at: <https://ieeexplore.ieee.org/abstract/document/9165760> (Accessed: 4 March 2024)

Nabipour, M., Nayyeri, P., Jabani, H., Mosavi, A., Salwana, E. and S, S., 2020. Deep learning for stock market prediction. *Entropy*, *22*(8), p.840. Available at: <https://www.mdpi.com/1099-4300/22/8/840> (Accessed: 5 March 2024)

Rouf, N., Malik, M.B., Arif, T., Sharma, S., Singh, S., Aich, S. and Kim, H.C., 2021. Stock market prediction using machine learning techniques: a decade survey on methodologies, recent developments, and future directions. *Electronics*, *10*(21), p.2717. Available at: <https://www.mdpi.com/2079-9292/10/21/2717> (Accessed: 5 March 2024)

Shen, J. and Shafiq, M.O., 2020. Short-term stock market price trend prediction using a comprehensive deep learning system. *Journal of big Data*, *7*, pp.1-33. Available at: <https://link.springer.com/article/10.1186/s40537-020-00333-6> (Accessed: 4 March 2024)