LUISS

Dipartimento

di Impresa e Management

Cattedra Data Analysis for Business

# EDA of stocks dataset and stocks prediction with ARIMA and non-linear models

Prof. Francesco Iafrate

RELATORE

Matr. 240471

CANDIDATO

Anno Accademico 2021/2022

**INDEX**

# I. INTRODUCTION

In this thesis, an attempt is made to try and establish how to predict stock prices through machine learning techniques that aren't based on qualitative variables but only on the price of the previous days. We all know that stock prices depend upon not only economic factors, but they relate to various physical, psychological, rational, and other important parameters, so I wanted to highlight how artificial intelligence can predict future stock prices without external pieces of information.

This thesis is split into three parts. The first part is an explorative data analysis (EDA) of the dataset we have in hand, which is an OHLC (High, Low, Close, Open) dataset of the worldwide famous firm Pfizer.

Exploratory data analysis (EDA) is a method for better understanding data that employs descriptive statistics and graphical tools. It is primarily used to gain as much insight into a dataset as possible, detect outliers and anomalies, and test underlying assumptions. It is a solid first step before employing other statistical methods that we are going to see in the second and third parts of this thesis.
In this part, I'm going to compare the close and open prices of the stock, find some statistical variables, and start to try and fit some linear model.

The second part will be the building of a famous model used a lot in this field to predict future prices based on past values. This AutoRegressive Integrated Moving Average statistical analysis model is known as "ARIMA". Autoregression is a time series model that predicts the value at the next time step by using observations from previous time steps as input to the regression equation. In other words, it predicts t by performing regression in the previous time step t-1. While for the moving average part, also known as the rolling mean, essentially we are taking the simple average in a given time frame and dividing it by the total number of time frames.

Finally, as we are approaching the finish of this thesis, the last part will be focused on trying to fit non-linear models, such as Random Forest and Knn, into an existing Time Series. To do this is important to essentially have the knowledge of the difference between time series and supervised learning as well as the knowledge of how to transform the first into the second manually as well as through functions inside of Rstudio.

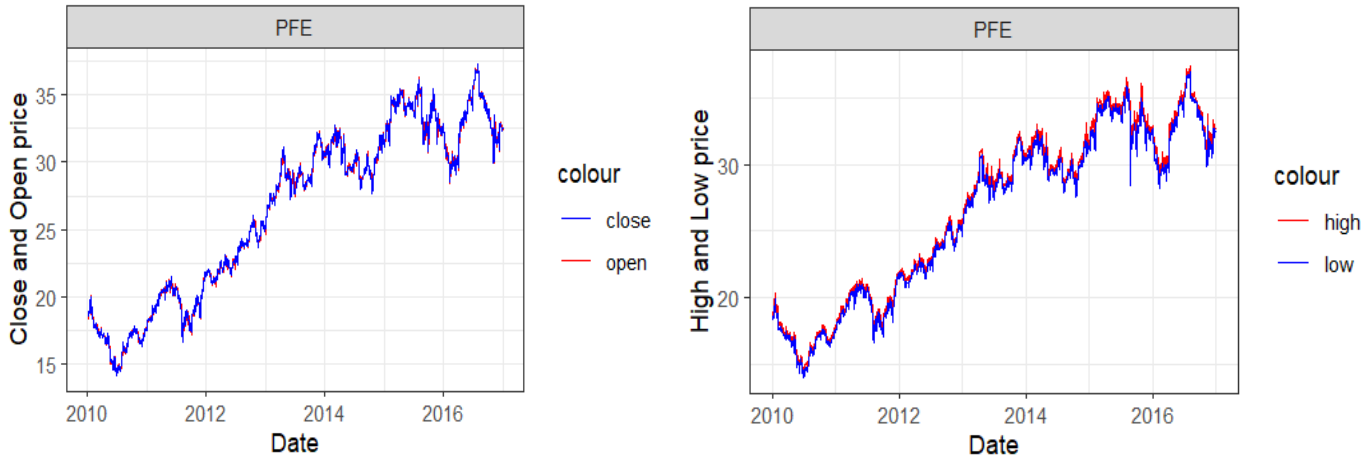
## II. EXPLORATORY DATA ANALYSIS

To start my project first of all we must understand the structure of the dataset I chose. As I mentioned in the introduction the original dataset is a classic OHLC with 851264 observations and 6 variables: Date, Symbol, High, Low, Close, and Open.

Date is in the format of day-month-year and we are going to use it as an index further in this thesis for the time series part, "Symbol" contains the ticker symbol of the firms that are in the dataset, while the other 4 variables are simply the prices of the stock during that day. The high and low prices in stock trading refer to the highest and lowest prices in a given time period. The open and close prices of a stock are the prices at which it began and ended trading in the same period. Knowing and understanding these variables we can get useful insight such as when the open and close are far apart, it indicates strong momentum; when they are close together, it indicates indecision or weak momentum. The high and low prices show the entire price range of the period, which is useful for assessing volatility.

From this impressively big dataset, I decided to utilize only one firm for my analysis, to focus more on the EDA and the models we are going to see in the following chapters I choose to do my study on Pfizer since it is a well-known corporation and I could also understand better the stock trend since it's not a niche company.

To start the analysis, first of all, I uploaded the dataset in Rstudio software and created a subset with only Pfizer using the Symbol variable as a filter, then I checked if there were NA's or duplicates and I formatted the Date such to use it after as an index. As I said before from the 4 variables of the OHLC we can get useful insight so I plotted the two couples on the y-axes to see the relative distance between them in a graph.

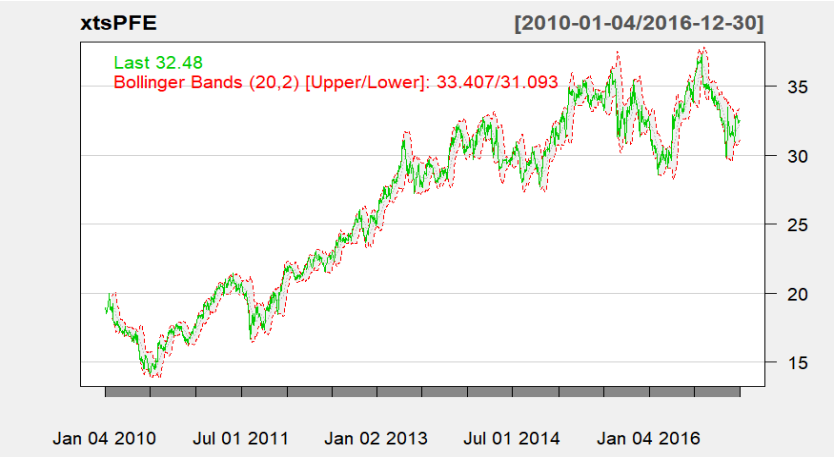



As we can see, on the first graph the close and open prices are closely spaced for most of the years but in some periods we see a disclosure between the two that indicates strong momentum. Looking instead at the High and Low graph we can see that the two prices are clearly apart from one another on the graph but still really close, this means that Pfizer stocks are not very volatile but neither stable. To have a more precise insight into this we can use the "coefficient of variation".

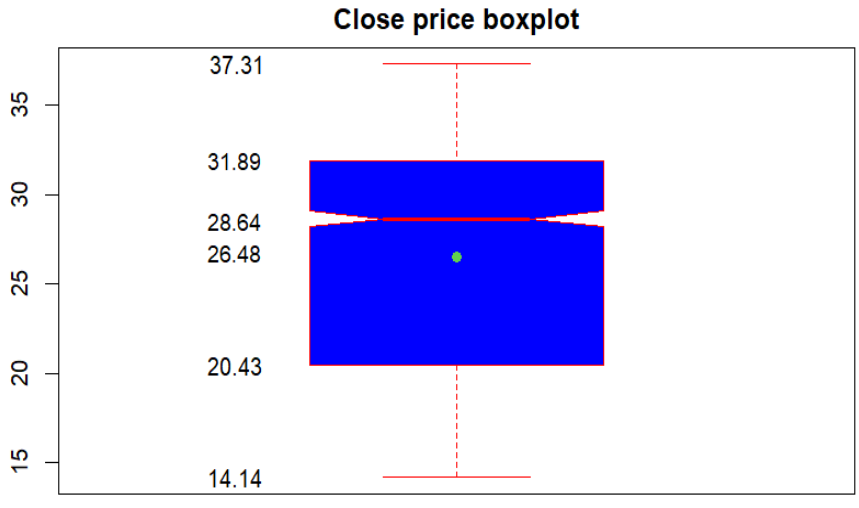
The coefficient of variation (CV) is a statistical measure of the dispersion of data points in a data series around the mean from which we can extrapolate the volatility of the stock. It is computed as the standard deviation of the vector over its mean, based on the distribution of the data points in your observation, the coefficient of variation varies. In general, a CV of 20 to 30 is considered appropriate, whereas a CV of more than 30 is considered unsatisfactory. Our value is 23.93% which is in line with our previous observation on the graphs I presented before.

Continuing with the volatility topic, I also did another graph to understand the theme of volatility visually thanks to the help of Bollinger Bands created by John Bollinger. Here, upper and lower price range levels are indicated by price envelopes. Bollinger Bands are envelopes that are drawn above and below a price's simple moving average at a certain standard deviation level. Since it's based on StdDev the bands' width adjusts to changes in the underlying price's volatility. Standard Deviations, or StdDev, and Period are the two parameters used by Bollinger Bands and although you can alter the pairings, the default settings for the period and standard deviation are 20 and 2, respectively.

On the right there is the graph of Bollinger bands with standard deviation equal to 2 and period equal to 20 adapted on the Pfizer stock's close price and we can clearly see that, in line with our previous constatations, the bottom and top lines are not so distant from the daily close price but there we can see that there is volatility.

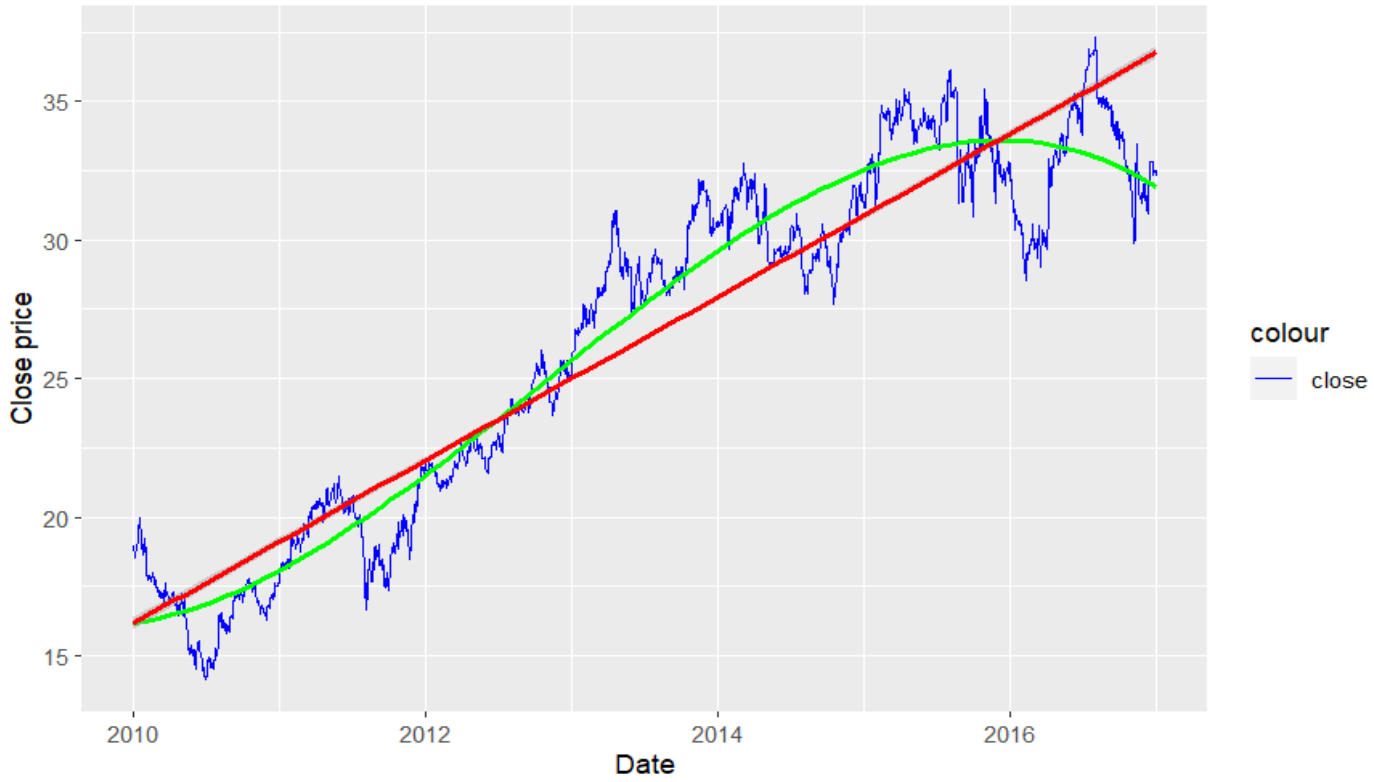



Going ahead, from the following box-plot, we can see some important descriptions of the variable df$Close that will be our response variable for the future models I'm going to train in this paper.




Here we can see, from the bottom to the top, the minimum value, the first quartile, the mean (which is the green point at the center of the plot), the median, the third quartile and finally the maximum value of the variable df$Close. We can see also see that there are no outlayers or anormalities but the median is closer to the top of the box and far from the mean, which means that the distribution is negatively skewed or skewed left.

Finally, before going to the true models, I tried to plot two simple linear regressions on the Close price to see how would it fit the response we are going to use from here on. We can see that the red one doesn't fit at all but just gives a direction, in fact taking the interval from 2010 to 2016 we can clearly see also from the graph that generally the stock price was rising, with different ups and downs. If we take into consideration the polynomial linear model in green we can see that with degree 3 it starts to resemble a little bit better the trend of the close price but it is too simplified and there will be the need for a polynomial of a greater degree to capture better the movement of price in time.

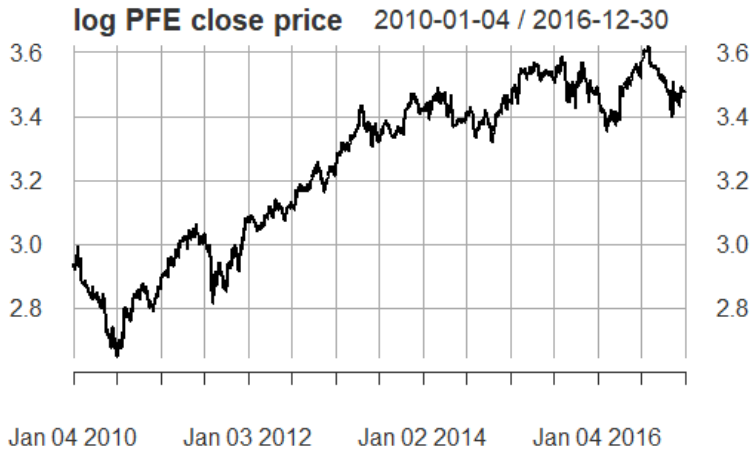


# III. ARIMA

In this chapter, we enter finally the field of predicting future values of a time series through the ARIMA model. First of all, until now we didn't give a proper definition for time series; A time series is a collection of well-defined data points gathered through repeated measurement in intervals of time and can be decomposed into three components: the trend (long-term direction), the seasonal (systematic, calendar related movements) and the irregular (unsystematic, short term fluctuations). These components are really important in the prediction that I'm going to do.

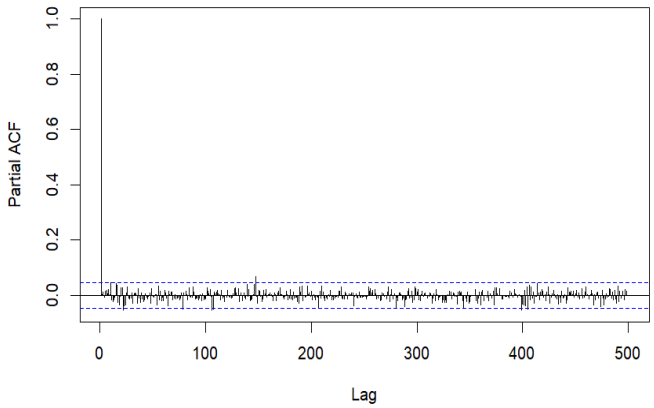
Time series analysis is a crucial component of statistics that studies a data set's features and aids in projecting the series' future values based on those qualities. In this thesis, my focus will be on the ARIMA which finds its basis in the Autoregressive and Moving Average (ARMA) model which is a crucial tool for time series analysis. The contributions of Yule, Slutsky, Walker, and Yaglom helped develop the ideas of autoregressive (AR) and moving average (MA) models. The difference between the two is that ARIMA Model converts non-stationary data to stationary data before working on it. The ARIMA model is frequently used to forecast data from linear time series. As the ARIMA technique was first made prominent by Box and Jenkins, the models are frequently referred to as Box-Jenkins models.

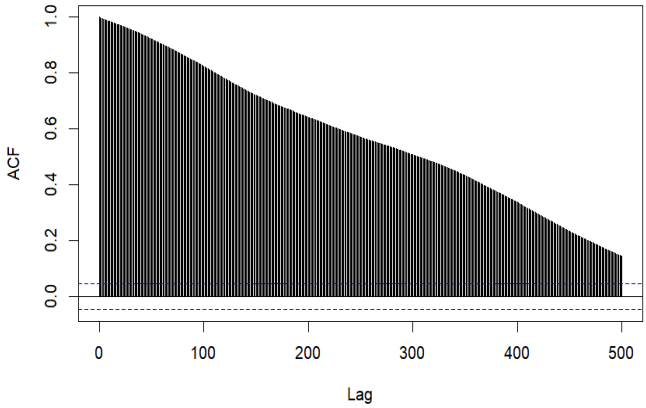
To start the data preparation for the ARIMA model, it's useful to know that in financial time series analysis, a rule of thumb is to do log transformation on the data. This will show the stock's growth rate, and log transformation will scale each data's unit value (price in USD) so that it is equally scaled and the analysis is simpler. As the data has been log-transformed we can clearly see that the series shows some upward and downward trends in given time intervals for example a spike in 2011 following a huge drop and from 2012 it starts rising always with ups and downs. The stock also consists of some volatility and swings. These are clear signs that the stock close price is non-stationary. Actually, most of financial data is non-stationary since stocks typically have a trend and inconsistent variance and mean over the course of a given period. Most of them, in fact, follow a random-walk model with or without drift (RWD or RWM). Given that we can guess that it's a random walk, which means the current value (price) is equal to its price at the time (t - 1) plus a random shock (White Noise), hence we should **differentiate** the data with a certain lag in order to fit ARIMA model as we'll see later.




Another important step is to analyze if there's any correlation between today's price and yesterday's. This is called autocorrelation, where the current value is correlated or affected by yesterday or n-days

backward. The tools we need to analyze the autocorrelation among the time series are using Autocorrelation Function (ACF) and Partial Autocorrelation (PACF) depicted by the Correlogram.



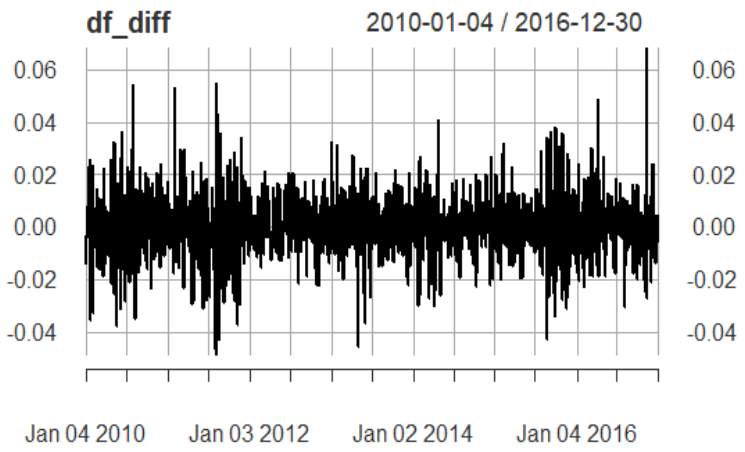




The average correlation between data points in a time series and earlier values of the series calculated for various lag lengths is measured and plotted using an ACF. The difference between an ACF and a PACF is that a PACF accounts for correlations between shorter lag length observations in each partial correlation. Furthermore, the blue region that can be seen in both graphs is the 95% confidence interval which serves as a sign of the threshold for significance. This indicates that anything inside the blue area is statistically close to zero while outside the blue area is statistically non-zero.

Knowing these pieces of information, if we look at the ACF correlogram we can see that the data shows strong and significant autocorrelation for most of the graph. For the PACF, we can see just a few significant correlations around the 150ish, for the rest of the graph, the autocorrelation oscillates around 0 and doesn't seem to have any seasonal pattern or trend.

From this evidence, we can say almost surely that the data is non-stationary as we supposed also before through the log data. To have our final verdict we can use the Augmented Dickey-Fuller Test which is a stationary testing using unit root testing. In simple terms, we compare the null hypothesis—that the data are non-stationary—to the alternative hypothesis, which is that the data are stationary. As per usual, if the resulting p-value is less than 0.05, we deduce that the result is significant and that the null hypothesis can be rejected; if not, the result has a unit root and is non-stationary. The result of the ADF test on the log data shows an insignificant p-value of 0.3848, we can say that it has a unit root. Therefore we are going to differentiate the data by lag 1 to stabilize the mean of the time series by removing (or reducing) trend and

seasonality. After differencing our time series we will have this graph which shows how the data has now mean that oscillates around 0 and it's free of any seasonality or trend. Also, the ACF and PACF give us a good result, and to have the final confirmation that we are now working with a stationary time series I ran again the ADF test. This time the p_value is a significant 0.01 (<0.05) so our hypothesis is correct, and we can now start to fit the ARIMA model.

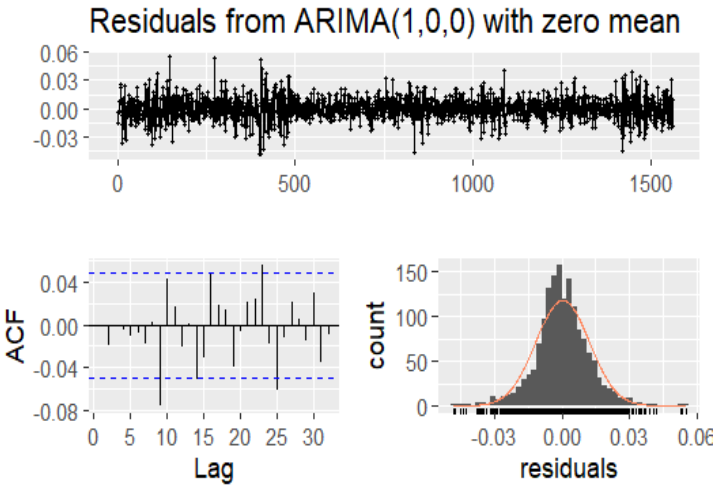



R provides a simple and automatic way to generate an ARIMA model with appropriate variables p, d, and q using the "auto.arima()" function in "forecast" package. Passing our differenced time series and the penalized method, for which I chose "aicc", the function will compute the best model that can fit our time series. The auto.arima function will first of all fit models using approximation and after finding the best models with the help of the penalized approach, it will run again only the best models without approximation to see which is the best model. In my case with approximation, the best model was ARIMA(2,0,2) with non-zero mean, but after re-fitting without approximation the best model was now ARIMA(1,0,0) with zero mean. ARIMA(1,0,0) means first-order autoregressive model: if the series is stationary and autocorrelated, perhaps it can be predicted as a multiple of its own previous value, plus a constant. The forecasting equation, in this case, will be

$$\hat{Y}t = \mu + \phi 1 Yt\text{-}1$$

which is Y regressed on itself lagged by one period. This is an "ARIMA(1,0,0)+constant" model. In our case the mean of Y is zero, then the constant term would not be included.

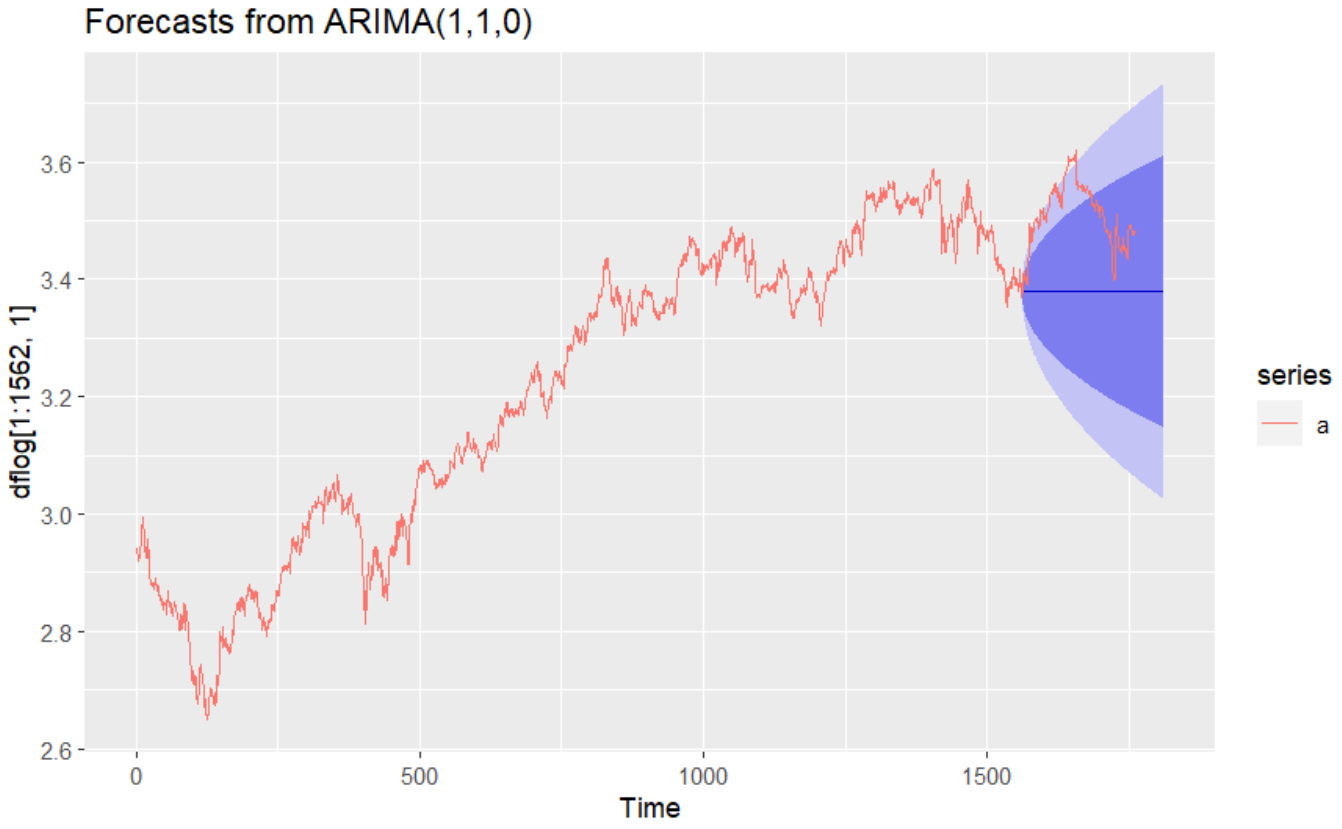
We then check for the presence of any residuals in our ARIMA model and based on the results of the Ljung-Box test, we conclude that the p-value > 0.05 (insignificant). As a result, the residuals of the model are independent and not autocorrelated. It follows that we are not required to perform volatility modeling using Garch models, which are frequently used on financial data with heteroscedasticity issues. We can see also graphically that the first image gives us a



mean that oscillates around the 0 and the graph on the bottom right let us see that our ARIMA model is following a normal distribution and it is stationary.

Now that we have the best model for our time series, we are now ready to do the forecasting with the "forecast" package. First of all, we create the arima model with the variables given by the auto.arima function but changing the "d" variable from 0 to 1 to visualize the result of the model on the log time series. We must do this because a model with no orders of differencing assumes that the original series is stationary while with one order of differencing it assumes that the original series has a constant average trend ( like a random walk). My thinking was that it was more interesting to visualize our model in the original log data instead of seeing it on the mean-reverting time series we created after. Furthermore to compare the results of ARIMA and the true test values I ran the model over a window of  1562 observations in a way to use the remaining 200 as a test. In the following graph we can see the work of the model:

We can see from the chart that most of the test observation values lie within the 95% confidence band, but the predicted forecasts are consistently different from the actual values. I tried also many other models, changing ARIMA variables, but also if they visually seemed better, their AIC was way higher, so I decided to stick with the auto.arima suggested a model.



## IV. KNN ON TIME SERIES

As we saw in the previous chapter, traditional time series forecasting techniques include exponential smoothing and ARIMA models, among other statistical techniques. But in recent decades, time series forecasting has been made possible by computational intelligence methods. Despite the fact that artificial neural networks are the most widely used machine learning method for time series forecasting, other techniques, like KNN, have also been used.

When compared to traditional statistical models, computational intelligence techniques have some interesting characteristics, such as nonlinearity or the absence of an underlying model, or non-parametricity.

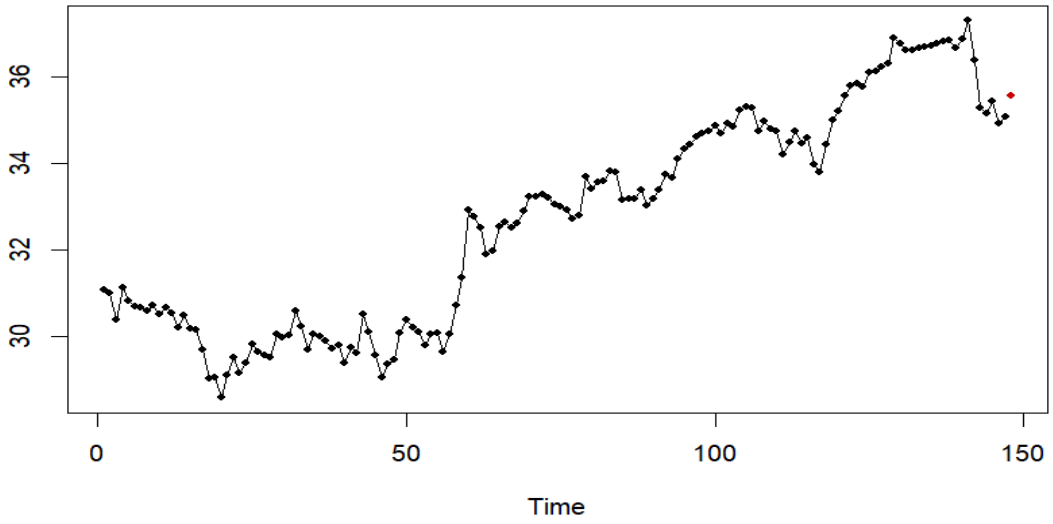
Excellent packages for time series forecasting statistics are available in R. For instance, the forecast package incorporates the Theta method, ARIMA, exponential smoothing, and other basic methods that can serve as benchmarks, like the random walk approach. However, despite the fact that R offers a wide range of computational intelligence regression approaches, including the caret package, these approaches cannot be directly applied to time series forecasting. Fortunately, some new packages are filling this gap.

KNN is a very popular algorithm for classification and regression and it has its own method to predict future values. Simply put, this algorithm keeps a list of examples. Every regression example has a target value that is a numeric vector of features that describes the example. According to a distance metric, like the Euclidean distance, KNN finds its k most similar pairs, or nearest neighbors, for a new example, and predicts its value as an accumulation of the target values connected to its closest neighbors. As we said before, though, in this paper we are going to see KNN linked to time series, and after we are going to see how the package tsfknn R package for univariate time series forecasting can help us in this topic.

KNN regression holds a collection of training instances, as was previously mentioned. The i-th training instance is made up of a target vector and a vector of n features that describe it. The firsts are used to find a new instance's k most similar training instances using the feature vectors and a similarity or distance metric for which we know the features but not the target. The k training instances that are closest to the new instance are known as the k nearest neighbors or the k most similar instances. The foundation of KNN is analogous learning. If a new instance is given, we believe that its closest neighbors' targets are probably comparable to its unidentified target. This way, to predict the target of the new instance, the targets of the closest neighbors are combined. Now to use KNN on time series the core remains the same; the target associated with a training instance is a collection of values of the time series while the feature describing it are lagged values of the target. We have so an autoregressive model. In the right image, we can see the head (first 5 instances) of our lagged dataset H1 is the target and the 2 lags are the two features. In the next instance, H1 will become Lag

| LAG 1 | LAG 2 | H1 |
|---|---|---|
| 18.93 | 18.66 | 18.60 |
| 18.66 | 18.60 | 18.53 |
| 18.60 | 18.53 | 18.68 |
| 18.53 | 18.68 | 18.83 |
| 18.68 | 18.83 | 18.77 |

1 and previous Lag 1 will become Lag 2 and so on. The use of KNN in time series forecasting is justified by the possibility of repetitive patterns in a time series. Given the last pattern of a time series, we search for earlier patterns that are similar in the hope that their later patterns will be comparable to the time series' future values. Putting this into code and plotting the prediction we have a predicted value of 35.57 while the original is 35.13 which is really close. In the graph below we can see the train set in black (which was obviously lagged 1:2) and in red the single prediction.

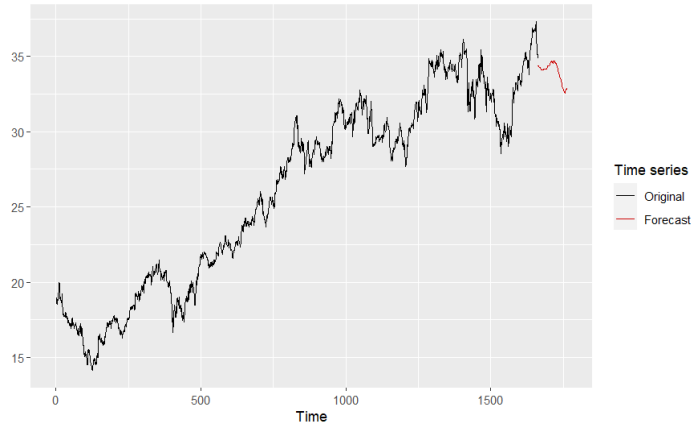



Above we did a one-step ahead prediction with KNN, but it is very common and often more useful to forecast more than one value into the future. For this objective, we must use one multi-step ahead strategy from the same package as before. The two methods we are going to use are the multiple input multiple outputs (MIMO) and the recursive approach.

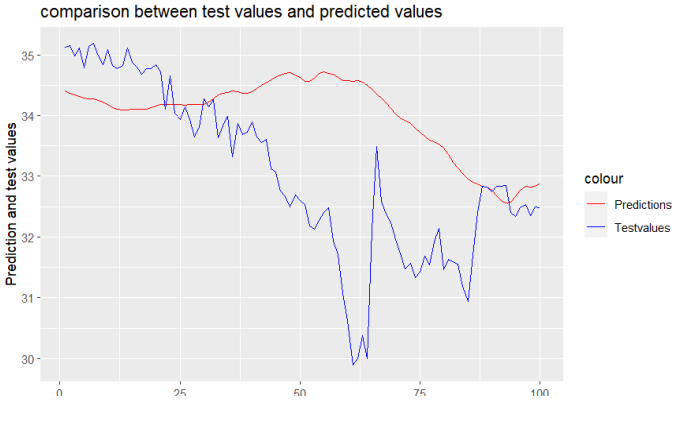
The MIMO strategy is characterized by the use of a vector of target values. The length of this vector is equal to the number of periods to be forecast, in this case, we are going to forecast 100 values in the future and we could use a lag of 1:100 to see if the model captures some trend, though we expect the opposite since in the PACF of the time series we can't see almost any lag, so we could use fewer lags to have higher accuracy. To choose the k parameter, furthermore, we must know that if k is too small the prediction can more easily be affected by white noise, while if we are using too large k's then we are using parameters too far apart from the new instance that could ruin the prediction. The recommended k usually is the square root of the length of the training set, so here that we have 1662 observations we are going to use k = 40.

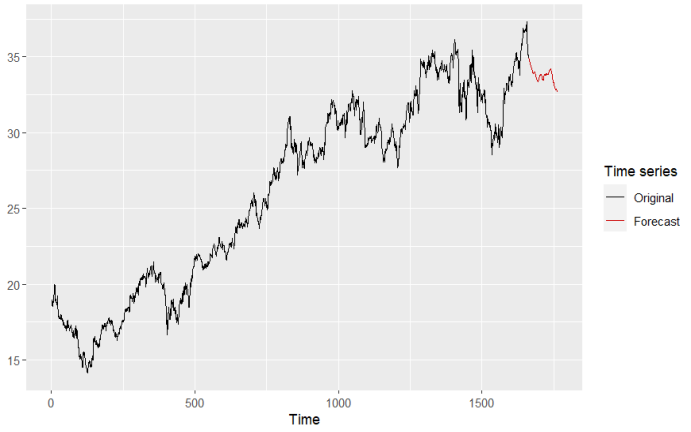
We are going to try both lags and see which one has the best accuracy and after this, we are going to do also the recursive method to see how it works and if the model can improve or be worse. To evaluate the models we are going to train, two methods are used. The first being the RMSE and the second the R2. Briefly, the Root Mean Square Error (RMSE) is the residuals' standard deviation (prediction errors). The distance between the data points and the regression line is measured by residuals, and the spread of these residuals is measured by RMSE. In other words, it provides information on how tightly the data is clustered around the line of best fit. As a threshold, we know that a value between 0.2 and 0.5 shows that the model can predict the data

accurately. The R-squared value, denoted by R2, is the square of the correlation. It measures the proportion of variation in the dependent variable that can be attributed to the independent variable (in our case between the predicted values and the test values. It can be between 0 and 1 and the more is near 1 the better the model fit. In the following images, we are going to see on the left the KNN with MIMO method with k = 40 and lags 1:100, under which there will be a first evaluation of the model and a graph to see the difference between test and prediction on the 100 values to be predicted. On the right, there will be the same structure but this time with lags 1:2 in order to not catch any misleading trend.

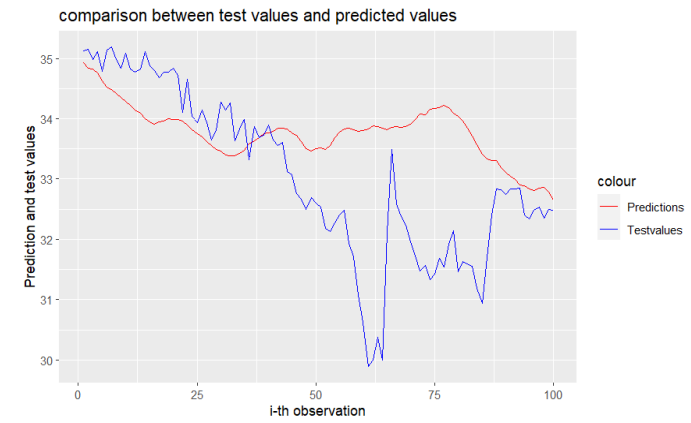



| RMSE | Rsquare |
|---|---|
| 1.680 | 0.045 |



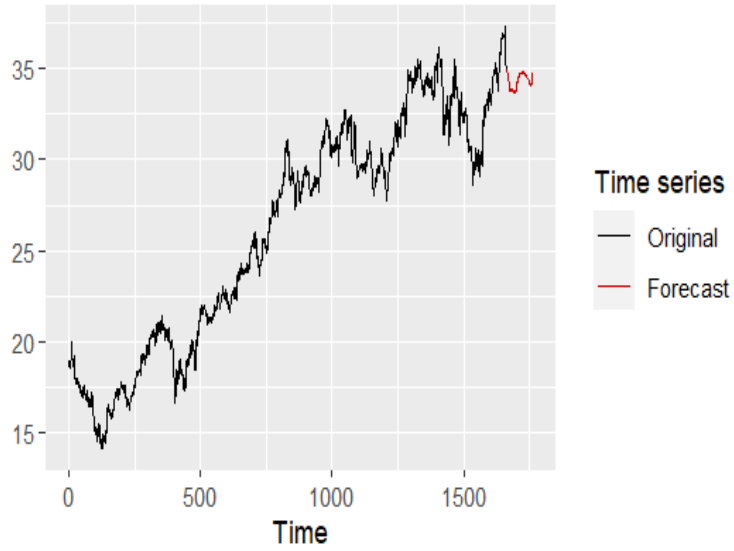

| RMSE | Rsquare |
|---|---|
| 1.484 | 0.105 |

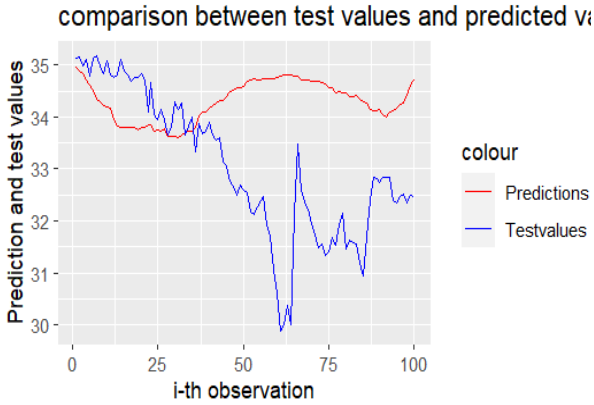



We can see that both models struggle to accurately predict the data mainly because of the huge drop that the test set has around the 40-ish observation. We can say, though, that the model on the right that observe less trend and use more noise did better than the one with 100 lags with an R2 of 0,10 and an RMSE of 1.48, two values that are not good for the most of prediction models but as we know stocks are very volatile and depend on hundreds of factors that are now being ignored. We can say also that if the test set hadn't that vertiginous drop, the model's prediction values seem very near to the test set values. To continue our tries with the KNN model we are also going to see, as we mentioned before, the recursive method and we will see if it performs better or worse than the MIMO option.

The recursive strategy is the same approach used by models such as ARIMA or exponential smoothing to forecast several targets. In this strategy, a model that only forecasts one-step is used but this model is applied

iteratively to forecast all future targets, when the training set is finished and cannot be used anymore as features for the new instances, previous predictions are used instead. In this case, as parameters, we are going to use the same as before k = 40 and lags 1:100. Since it is an iterative model, if, in our case, we use low lags and a large k, such as ours, the output will be a straight line, which is very inaccurate as a prediction. Fitting the model with the parameters we said before we have this prediction:




| RMSE | Rsquare |
|---|---|
| 2.051 | 0.312 |

In the model above we can see an R2 of 0.31, the higher until now, but we also see visually that this value is given by the fact that the correlation is negative and we can see that by the red line that goes upward while the blue line goes downward. As a matter of fact, another aspect that can underline this hypothesis is the fact that the RMSE is way higher than the first two MIMO models. We can so conclude that the best model between the three is the one with k = 40 and lags 1:2.

There is also another modality to evaluate the forecast accuracy of KNN, which is built through a function in the same tsfknn package. This function is called rolling_origin. It uses the same classical approach of dividing our time series in train and test; furthermore, it is capable of dividing it directly through the number of targets chosen inside the function (h parameter). The first parameter for the rolling origin function is a "knnForecast" object. Information about the time series and the KNN model's meta parameters, such as the autoregressive lags, the number of nearest neighbors, or the multi-step ahead strategy, are obtained from this. The size of the test set is rolling_origin's second parameter. This function then creates test_set, predictions, a vector consisting of errors and also several forecasting accuracy measures such as root mean square error, mean absolute error and mean absolute percentage error (respectively RMSE, MAE, MAPE). I will put an example in the appendix in image n°1. Furthermore, plotting the rolling_origins will give us the entire plot consisting of train, test,  and prediction values.

# V. RANDOM FOREST

Random forest is a machine learning algorithm used commonly for handling both regression and classification problems. It was created by Leo Breiman and Adele Cutler and it combines the output of multiple decision trees to reach a single result and became very common thanks to its ease of use, flexibility, and accuracy. Decision trees are flowcharts-like tree structures, where each internal node denotes a test on an attribute, each branch represents an outcome of the test and each leaf node holds a class label. Its objective is to learn straightforward decision rules inferred from the data features in order to build a model that predicts the value of a target variable. A tree can be seen as a piecewise constant approximation. Decision trees have many advantages compared to other algorithms, for example, they are simple to visualize, interpret and understand, require almost no data preparation like normalization or dummy variables, are able to handle both numerical and categorical data, and in general have good accuracy. But at the same time, decision trees can overfit through the creation of over-complex trees that don't generalize enough the data, and in this case, the model will be very good in accuracy for your dataset but very inaccurate if the data change behavior. Furthermore, they can be unstable because of the fact that small variations could change completely the decision of the tree itself.

Most of those weaknesses can be solved thanks to Random Forest, specifically, which is an extension of the bagging method which uses bagging and feature randomness to create an uncorrelated forest of decision trees. Low correlation between decision trees is ensured by feature bagging, which generates a random subset of features. This is the main distinction between decision trees and random forests. Random forests only choose a portion of those feature splits, whereas decision trees take into account all possible feature splits. Furthermore, it fits a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve predictive accuracy. as we mentioned earlier the benefits of using Random Forest are the reduced risk of overfitting in fact decision trees have a tendency to tightly fit all the samples in the training data, which increases the risk of overfitting. The classifier won't, however, overfit the model when there are a large number of decision trees in a random forest because the averaging of uncorrelated trees reduce the overall variance and prediction error. Random Forest, moreover, provides more flexibility, data scientists frequently use it because it is highly accurate at handling both regression and classification tasks. The random forest classifier benefits from feature bagging by maintaining accuracy even when some of the data is missing. And, also if in our case this is not a useful aspect, in general, Random Forest makes it easy to determine feature importance thanks to many ways such as MDI (mean decrease in impurity), that measure how much the model accuracy decreases when a variable is excluded, and MDA (mean decrease accuracy) that identifies the average decrease in accuracy permutating randomly feature values in OOB samples.

Random Forest algorithm can be also used to forecast values in a time series dataset so, as done before with the other models we are going to try to predict the close price of Pfizer stocks. To do this, first of all, we

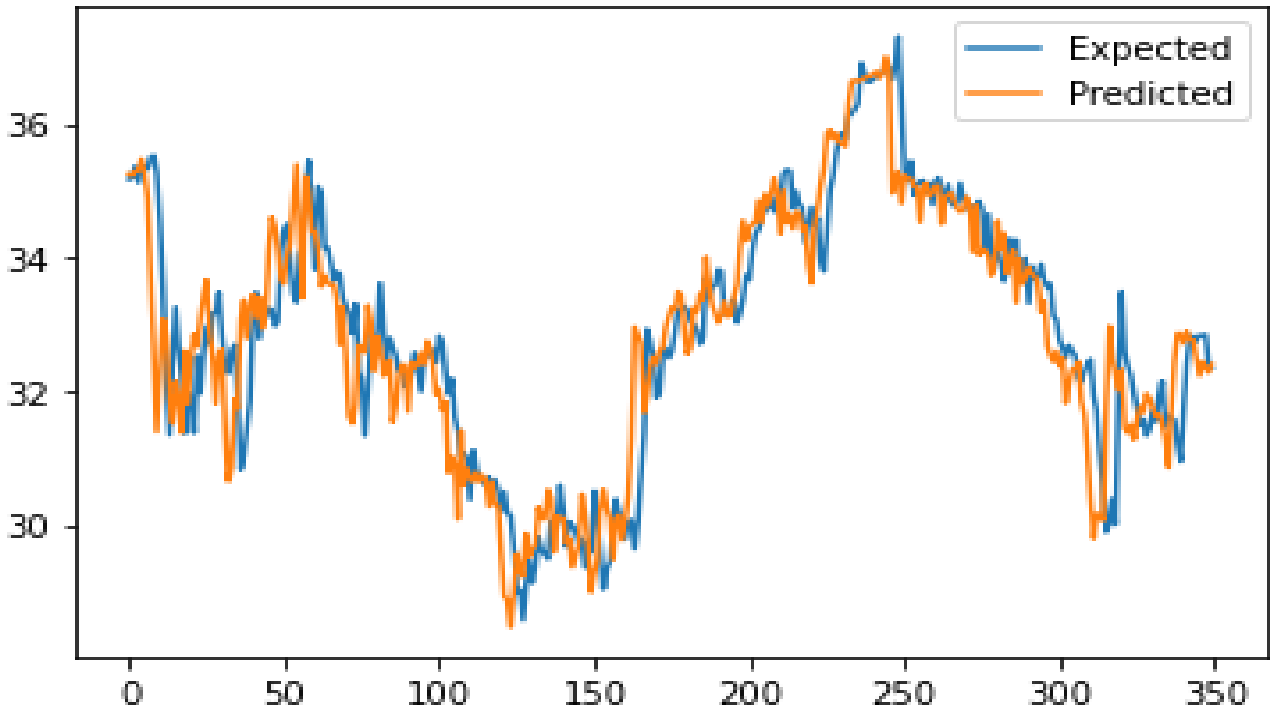
must transform the time series into a supervised dataset. To do this we are going to use the same method as in knn but this time it must be done manually. As before we create lags 1:6 to create the supervised dataset and we are now working with 6 features and one target:

| | **0** | **1** | **2** | **3** | **4** | **5** | **6** |
|---|---|---|---|---|---|---|---|
| *0* | 18.93 | 18.66 | 18.6 | 18.53 | 18.68 | 18.83 | 18.77 |
| *1* | 18.66 | 18.6 | 18.53 | 18.68 | 18.83 | 18.77 | 19.21 |
| *2* | 18.6 | 18.53 | 18.68 | 18.83 | 18.77 | 19.21 | 19.38 |
| *3* | 18.53 | 18.68 | 18.83 | 18.77 | 19.21 | 19.38 | 19.49 |
| *4* | 18.68 | 18.83 | 18.77 | 19.21 | 19.38 | 19.49 | 20 |
| *5* | 18.83 | 18.77 | 19.21 | 19.38 | 19.49 | 20 | 19.94 |

After doing so, we are going to use a method called walk_forward validation and it is a particular application of a method called cross-validation. It means using a portion of your data to validate and another portion to optimize a system. Here we are going to use it with the intent of using the model to make a one-step forecast for each window. The random forest model will do one prediction at a time and then store the predictions inside a list that will become at last the prediction array to compute the accuracy of the model. Every "i" in the observation will also print the expected value ad predicted value, then while the predicted value is stored, the expected value is used as the train set to do the next prediction and so on for the whole length of the test set.

```
In [1]: runfile('C:/Users/bosch/.spyder-py3/temp.py', wdir='C:/Users/bosch/.spyder-py3')
>expected=35.4, predicted=35.2
>expected=35.3, predicted=35.2
>expected=35.5, predicted=35.3
>expected=35.5, predicted=35.3
>expected=35.3, predicted=35.5
>expected=34.5, predicted=35.2
>expected=33.5, predicted=34.9
>expected=32.1, predicted=33.8
>expected=31.3, predicted=32.6
>expected=32.4, predicted=31.4
```

Plotting the result, splitting the train and test into 80%-20% respectively we find out, at least visually, that the model is performing really well since it fits almost perfectly the data.




Though we can see many imprecisions in the predicted values so we are going to assess the accuracy of the model with 3 accuracy methods, MAE, RMSE, and R2.

| | |
|---|---|
| **MAE** | **0.3590** |
| **RMSE** | **0.4799** |
| **R2** | **0.9347** |

As we can see the accuracy of this model is about 93% with really low errors throughout the predictions. We could've expected this result since the Random Forest algorithm is one of the best for accuracy. Furthermore, the method of windows with the walk-forward validation really helped the precision of the model.

# VI. CONCLUSION

Our research has shown that stock prices can be predicted through non-linear methods such as Knn and Random-forest and that those algorithms can also reach an accuracy way higher than many other more famous linear algorithms such as ARIMA. Furthermore, our paper underlines the fact that also if stocks depend on many variables that cannot always be comprehended by linear and non-linear models since they are too unpredictable or cannot be transformed into numbers or categories, non-linear models can predict the price with a valid accuracy through methods such as sliding windows and walk forward validation. Obviously, these models will work better with stocks that are not too volatile and if the market itself is very volatile we expect a huge drop in performance precisely due to the fact that these models take only the price of the previous days as features. In conclusion, stock prices will always remain uncertain and there will never be a perfect model for them, though this study is a continuous finding of new methods, models, and variables to use to make this uncertainty smaller. In this paper, we had good results overall so we can predict that with the integration of other types of models that take as inputs new variables the accuracy of the prediction of stock prices can become more and more precise in the time to come.

# VII. APPENDIX

## 1- Rolling_origin function

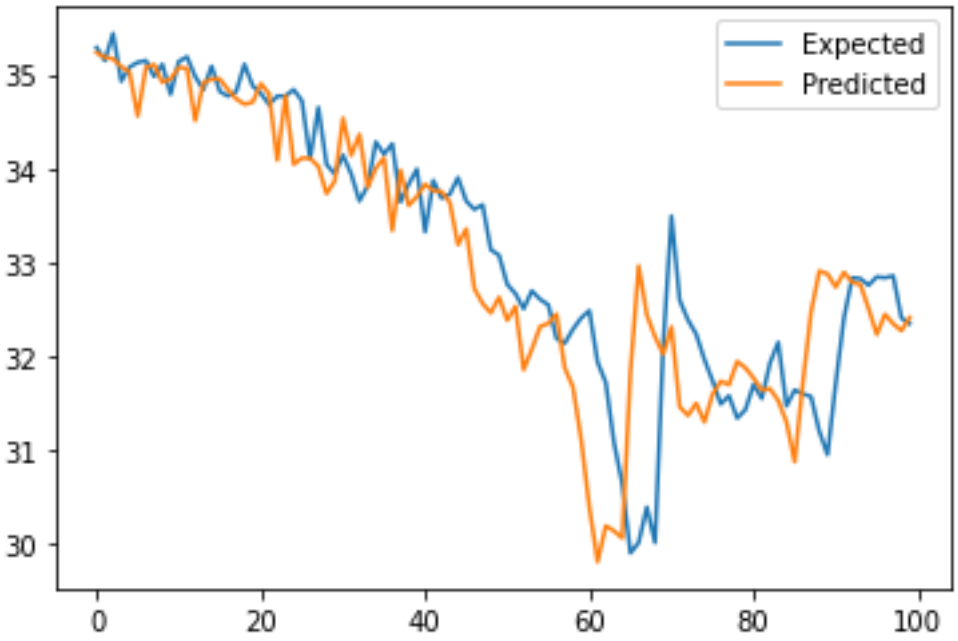
```r
pred4 <- knn_forecasting(ts(xtsPFE), h = 100, lags = 1:2, k = 40,
msas = "MIMO", transform = "none")

ro = rolling_origin(pred4, h = 100)

ro$test_sets[1:10]
ro$predictions[1:10]
ro$errors[1:10]
ro$global_accu
```

```
 [1] 35.13 35.15 34.98 35.11 34.79 35.14 35.19 34.98 34.84
[10] 35.09
 [1] 34.92375 34.89375 34.93850 34.87000 34.89350 34.78450
 [7] 34.91750 34.93850 34.88200 34.72450
 [1]  0.20625107  0.25625210  0.04150015  0.24000110
 [5] -0.10349888  0.35549903  0.27249900  0.04150015
 [9] -0.04199987  0.36549998
     RMSE       MAE      MAPE
1.616306 1.286520 4.041506
```

## 2- Random forest with 100 observations



**3-**

```
def walk_forward_validation(data, n_test):
    predictions = list()
    # split dataset
    train, test = train_test_split(data, n_test)
    # seed history with training dataset
    history = [x for x in train]
    # step over each time-step in the test set
    for i in range(len(test)):
        # split test row into input and output columns
        testX, testy = test[i, :-1], test[i, -1]
        # fit model on history and make a prediction
        yhat = random_forest_forecast(history, testX)
        # store forecast in list of predictions
        predictions.append(yhat)
        # add actual observation to history for the next loop
        history.append(test[i])
        # summarize progress
        print('>expected=%.1f, predicted=%.1f' % (testy, yhat))
    # estimate prediction error
    error = [skm.mean_absolute_error(test[:, -1], predictions),
          np.sqrt(skm.mean_squared_error(test[:,-1], predictions)),
          skm.r2_score(test[:, -1], predictions)]
    return error, test[:, 1], predictions
```

```
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
    n_vars = 1 if type(data) is list else data.shape[1]
    df = pd.DataFrame(data)
    cols = list()
    # input sequence (t-n, ... t-1)
    for i in range(n_in, 0, -1):
        cols.append(df.shift(i))
    # forecast sequence (t, t+1, ... t+n)
    for i in range(0, n_out):
        cols.append(df.shift(-i))
    # put it all together
    agg = pd.concat(cols, axis=1)
    # drop rows with NaN values
    if dropnan:
        agg.dropna(inplace=True)
    return agg.values
```

```
def random_forest_forecast(train, testX):
    # transform list into array
    train = np.asarray(train)
    # split into input and output columns
    trainX, trainy = train[:, :-1], train[:, -1]
    # fit model
    model = sk.RandomForestRegressor(n_estimators=1000)
    model.fit(trainX, trainy)
    # make a one-step prediction
    yhat = model.predict([testX])
    return yhat[0]
```

## BIBLIOGRAPHY


- **TIME SERIES FORECASTING WITH KNN IN R:** the tsfknn Package by Francisco Martínez, María P. Frías, Francisco Charte and Antonio J. Rivera https://journal.r-project.org/archive/2019/RJ-2019-004/RJ-2019-004.pdf

- **STUDY OF EFFECTIVENESS OF TIME SERIES MODELING (ARIMA) IN FORECASTING STOCK PRICES** Prapanna Mondal1 , Labani Shit1 and Saptarsi Goswami2 1 Student, Bachelor of Technology Department of Computer Science and Engineering Institute of Engineering and Management 2Asst. Professor Department of Computer Science and Engineering Institute of Engineering and Management **https://www.researchgate.net/profile/Saptarsi-Goswami-2/publication/276197260_Study_of_Effectiveness_of_Time_Series_Modeling_Arima_in_Forecasting_Stock_Prices/links/56cc7d2b08ae1106370d9496/Study-of-Effectiveness-of-Time-Series-Modeling-Arima-in-Forecasting-Stock-Prices.pdf**

- **RANDOM FOREST FOR TIME SERIES FORECASTING** by Jason Brownlee on November 2, 2020 **https://machinelearningmastery.com/random-forest-for-time-series-forecasting/**


## SITOGRAPHY


**https://www.geeksforgeeks.org/decision-tree/**

**https://rpubs.com/kevinTongam/arimaforecast**

**https://bookdown.org/kochiuyu/technical-analysis-with-r-second-edition/bollinger-band.html**

**THESIS ACKNOWLEDGMENT**

I would like to express my deepest appreciation to professor Francesco Iafrate for the help and pieces of advice he gave me in moments of difficulty during this thesis which would've been a lot more complicated without it.

I am also thankful to my colleagues that gave me feedback and helped me during some hard times when I was stuck in some lines of code and, finally, I would be remiss in not mentioning my family that always supported my work during these three years and permitted me to have a beautiful university path in LUISS.