UNIVERSITÀ DEGLI STUDI DI MILANO

FACOLTÀ DI SCIENZE POLITICHE,
ECONOMICHE E SOCIALI

M.Sc. IN DATA SCIENCE AND ECONOMICS

DATA SCIENCE FOR FINANCE: TECHNICAL INDICATOR AND TRADING STRATEGY THROUGH MACHINE LEARNING AND ALTERNATIVE DATA

Advisor: Prof. Lorenzo Mercuri

Co-advisor: Prof.ssa Silvia Salini

By: Andrea Frattini

Student no.: 959213

Academic Year 2021-2022

Acknowledgements

Contents

List of Figures

Introduction

In the last years, with the advent of Big Data, new powerful computers and technologies, the usage and development of Artificial Intelligence (AI) and Data Science (DS) have exponentially increased in multiple fields, such as for example in economics, marketing, cybersecurity, finance and many others. In particular, thanks to the huge amount of data that now we are able to extract from almost every situation, we are experiencing the discovering of new kind of data, called **alternative data**.
These types of data are widely used in the financial sector and differ from traditional data in the information content they provide: some examples of traditional data in finance are the open/close price, the volatility or the market capitalization of a particular stock, while the alternative data could be, for instance, how a particular security is perceived by the market or how much a title is ”popular” among investors.

This thesis has been written in collaboration with FinScience, an italian firm specialized in Artificial Intelligence algorithms and alternative data analysis for the investment industry. FinScience, every day, searches over 1,5 million web pages, extrapolating, interpreting and analyzing their contents for identifying valuable informations. FinScience summarizes what it learns from these pages through two proprietary alternative metrics:

- **Sentiment**: this indicator takes value between -1 and 1 boundaries included and assesses the perception of a theme/company.
- **Popularity**: takes potential positive infinite values and measures investor interest in a topic.

The database that will be used in this final dissertation is composed by 847 titles each of which is characterized by:

- *company name*: the name of the relative company;
- *sentiment*: the FinScience metric indicating how this security is perceived by the market (timeframe: daily);
- *popularity*: the FinScience metric indicating how much this security is popular among investors (timeframe: daily);
- *adj_close*: the daily adjusted closing price;
- *adj_high*: the daily adjusted highest price;
- *adj_low*: the daily adjusted lowest price;
- *pct_change*: the percentage change in the closing price (e.i. the daily return)

In the first chapter it will be extracted a sample of six titles over which will be implemented different GARCH models with and without different exogenous variables to evaluate if the FinScience metrics help such models in improving their performances.

In the second chapter, will be added new informations (e.i. new columns)

to the aforementioned database with the aim of preparing it to be used in a supervised classification problem. Afterwards, through the usage of two particular machine learning models embedded within rolling algorithms, will be created a new, reliable, FinScience-metrics-based technical financial indicator that will have the aim of predicting the price trend one day forward in the future.

In the third chapter, the new technical indicator will be validated through its use within an algorithmic trading strategy that will rely on the FinScience metrics. Moreover, it will be provided also a second, similar algorithmic trading strategy for comparing the results and understand the actual utility of the new indicator.

Finally, as usual, some further considerations will be provided in the last chapter, in which final conclusions of this project will be drawn.

Chapter 1

Popularity and Sentiment as exogenous variables into GARCH models

1.1 Theoretical concepts in pills

Throughout this first chapter it will be carried out an analysis of the historical series of prices of some of the most famous companies listed on the most important markets and, to do so, it will be used one of the most popular model for the analysis of historical series when the volatility is a significant factor: the GARCH(p,q) model. However, before showing the actual analyses with the corresponding obtained results, it is necessary to dwell on some key concepts.

1.1.1 GARCH(p,q) model

The *Generalized Auto Regressive Conditional Heteroskedasticity* model is, as the name says, the generalization of the *Auto Regressive Conditional Het-*

eroskedasticity model and, therefore, for better understanding the GARCH model is necessary to first talk about the ARCH model. This model was developed by the economist Robert Engle in 1982 and his model assumed that the variation of financial returns are not constant over time but auto-correlated on each other.

Hence, let's start with the analysis of the name of this model: *Heteroskedasticity* can be interpreted as volatility, standard deviation or, more in general, any kind of *deviation*; *Conditional* means that the volatility of the time series is not fixed over time but it depends on which time spot you are at; *Auto Regressive* means that the volatility at a certain time point depends on the time point right before and/or the one the time spot before and so on and so forth. Taking as example the simplest model, that is the ARCH(1), we have the following equation

$$\sigma_t^2 = \alpha_0 + \alpha_1 \epsilon_{t-1}^2 + ... + \alpha_q \epsilon_{t-q}^2 \qquad (1.1)$$

$$= \alpha_0 + \sum_{i=1}^{q} \alpha_i \epsilon_{t-i}^2 \qquad (1.2)$$

which means that the volatility of a time series today is given by a constant plus the past values of the time series itself, each of which is multiplied by an *i.i.d.* white noise (ϵ_t^2), which are the corresponding errors. More specifically, a process called "white noise" is one that satisfies the following conditions:

$$\begin{aligned} \mathrm{E}(\epsilon_t) &= 0 \\ \mathrm{E}(\epsilon_t \epsilon_\tau) &= \sigma^2 && \text{if t} = \tau \\ \mathrm{E}(\epsilon_t \epsilon_\tau) &= 0 && \text{otherwise} \end{aligned}$$

However, by construction, the ARCH model has a problem: imagine a situation in which the volatility stays almost always constant but once in a while it jumps, like in the following figure:

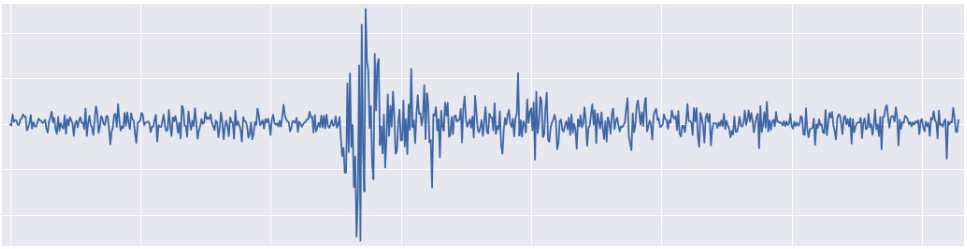


Figure 1.1: Volatility of a time series with a jump

In a situation like the one shown in Figure 1.1 the ARCH model would probably fail: the peculiarity of this model is that it predicts today on what happens yesterday and some day before it (how much days it looks back is chosen by the user). Therefore, if the days before a jump (let's assume that the day with the jump is today) there was low volatility, the ARCH model will predict low values for the today's volatility and viceversa if in the previous days there were high volatilities and today there is not.
Here is when GARCH model comes in: it was developed in 1986 by Tim Bollerslev as a way to improve the capability of forecasting volatility in asset prices and clearly he started from the previous work of the Robert Engle. Let's start again with the simplest model, e.i. GARCH(1,1), which has the following main equation

$$\sigma_t^2 = \omega + \alpha_1 \epsilon_{t-1}^2 + \beta_1 \sigma_{t-1}^2 \qquad (1.3)$$

where:

- σ_t^2 is the volatility of the process at time t (e.i. today)
- ω is the intercept
- α_1 and β_1 are constant
- ϵ_{t-1}^2 is the error associated to the past value of the time series

- σ^2_{t-1} is the volatility of the process at time t-1 (e.i. yesterday)
- α_0, α_1, $\beta_1 > 0$
- $\alpha_1 + \beta_1 < 1$

Therefore, the value today of the time series is affected the value of the time series yesterday times its error and the volatility of yesterday. Including the volatility of yesterday (or other days before) guarantees that the result is much less "bursty" than the one obtained through the ARCH model.

The general version of a GARCH model, that is the GARCH(p,q) model, which equation is:

$$\sigma^2_t = \omega + \alpha_1\epsilon^2_{t-1} + ... + \alpha_q\epsilon^2_{t-q} + \beta_1\sigma^2_{t-1} + ... + \beta_p\sigma^2_{t-p} \quad (1.4)$$

$$= \omega + \sum_{i=1}^{q} \alpha_i\epsilon^2_{t-i} + \sum_{j=1} p\beta_j\sigma^2_{t-j} \quad (1.5)$$

simply takes into account the value of the time series p days before and the value of the volatility q days before.

1.1.2 Information Criteria: AIC and BIC

One of the main problems that scientists have to face when deciding to implement a regressive model (or other models in the same family) is the choice of which and how many variables should be included. The theory (and the common sense) says that should be included all those explanatory variables whose variations actually affect the response variable and, in general, this choice is clearly conveyed by the experience of the scientist who is performing the analysis. However, since we are talking about scientists, appealing to the only (subjective) experience of them is not enough: it is necessary to have a more objective criterion that helps

the scientist in identifying the most performing model since, in general, if there are k possible regressors, there could be up to 2^k models with different combinations. Here is where AIC and BIC come in: they provide a measure of the distance between the model and the theoretical data distribution with respect to the number of estimated parameters.
The *Akaike Information Criterion* is constructed as follows: let L be the maximum likelihood value of the model with k parameters, then

$$\text{AIC} = \text{-2log}(L)\text{+2}k \qquad (1.6)$$

When selecting a model through this criteria, the preferred model is the one with the minimum AIC's value. The pros and cons of the *Akaike Information Criterion* are:

- Advantages: is thought to select the nearest unknown model to the real one and its formulation is very simple
- Disadvantages: for large samples tends to select complex patterns

The *Bayesian Information Criterion*, instead, is constructed as follows: let L be the maximum likelihood value of the model with k parameters and let n be the number of observations, then

$$\text{BIC} = \text{-2log}(L)\text{+log}(n)k \qquad (1.7)$$

Again, when selecting a model through this criteria, the one that is preferred is the one with the minimum BIC's value. The pros and cons of the *Bayesian Information Criterion* are:

- Advantages: is designed to select the true model that represents the data, so if that model is among those tested, the criterion will select

exactly that. In addition, it selects models with fewer and more easily interpretable parameters, and the number of parameters considered does not increase as the sample size increases

- Disadvantages: its behavior is unpredictable in the case in which the true model is not between those tested

1.1.3 ACF and PACF

The *AutoCorrelation Function* and *Partial AutoCorrelation Function* are generally used to tune the p and q parameters of ARIMA(p,q) models. In particular, the ACF, as the name says, gives the values of the auto-correlation of a time series with its lagged values that are plotted along with the confidence bands. Therefore, it describes how well the present values (a_t) are related (correlated) with its past values, namely a_{t-1}, a_{t-2}, a_{t-3} and so on and so forth. The ACF plot is said to be **complete** because if the data present trends, seasonality, cyclic or residuals it considers all of them while finding the correlations. In a practical way, an ACF plot as a shape like the following one

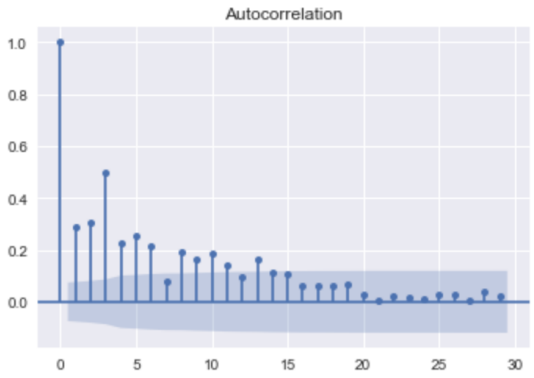


Figure 1.2: Example of ACF plot

and the attention of the user should be put over those spikes that break

the confidence bands (either above or below). In the example of Figure 1.2 all the first six spikes are significant (e.i. break the confidence band) but the theory says that it should be picked the highest: in this case, the correct value for the q parameter should be 3 even if there is no certainty that the previous lags will be actually significant.

Subsequently, the PACF plot is used for the selection of the optimal p parameter and has as a shape very similar the ACF one, as it is shown below.

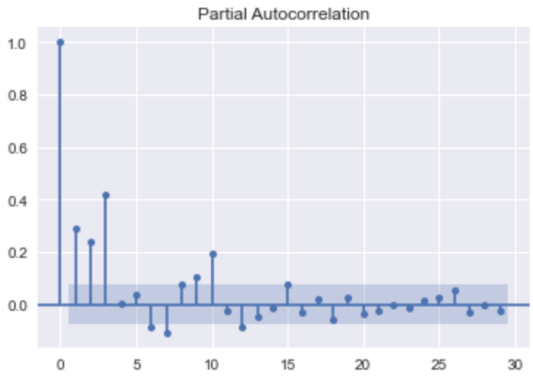


Figure 1.3: Example of PACF plot

The *Partial AutoCorrelation Function* finds the correlation of the residuals (which remain after removing the effects that are already explained by the earlier lags) with the next lag value (from which the adjective **partial** instead of "complete", since it has been removed already found variations before finding the next correlation). Therefore, if there is any hidden information in the residual which could be modeled by the next lag, it could be possible to get a good correlation keeping that next lag as a feature.
In the above example (Figure 1.3), the correct value for the the p parameter should then be again 3.

1.1.4 Dickey-Fuller and Augmented Dickey-Fuller tests

The *Dickey-Fuller* and *Augmented Dickey-Fuller tests* are worldwide spread statistical tests used to understand if a given time series is stationary or not.

The DF test assumes that the time series taken into consideration is an AR(1) of the form:

$$y_t = \mu + \phi_1 y_{t-1} + \epsilon_t \tag{1.8}$$

For the DF test the null hypothesis H_0 is that ϕ_1=1 (e.i. the time series has a unit root and therefore is not stationary) while alternative hypothesis H_1 is that $\phi_1 < 1$ (e.i. the time series has **not** a unit root). Without going too much into the mathematical detail of this statistical test (which is not the aim of this paper), the overall reasoning is the following one:

- First, make the difference between y_t and y_{t-1} getting the following equations

$$y_t - y_{t-1} = \mu + (\phi_1 - 1) y_{t-1} + \epsilon_t \tag{1.9}$$

$$\Delta y_t = \mu + \delta y_{t-1} + \epsilon_t \tag{1.10}$$

- Rewrite the hypothesis in function of δ getting

$$H_0\text{: } \delta = 0 \tag{1.11}$$

$$H_1\text{: } \delta < 0 \tag{1.12}$$

  In fact, if we assume that the null hypothesis is true, then the equation becomes $\Delta y_t = \mu + \epsilon_t$, which is some normally distributed random noise and therefore the process is stationary

- However, it's not possible to performe a simple *t-test* since y_{t-1} is assumed to be not stationary (since y_t too is assumed to be not stationary under the H_0). Therefore, it is performed a *t-test* against the *Dickey-Fuller Distribution*:

$$t_{\hat{\delta}} = \frac{\hat{\delta}}{se(\hat{\delta})} \tag{1.13}$$

and if $t_{\hat{\delta}} <$ DFcritical then H_0 is rejected (therefore is rejected the assumption that the process has a unit root, and then it actually is stationary) while if $t_{\hat{\delta}} >$ DFcritical then H_0 is not rejected (e.i. the process is not stationary).

The problem is, what to do if the process is more complex than an AR(1)? Here is when the *Augmented Dickey-Fuller test* comes in. The ADF test assumes that the process has a shape like the following one

$$y_t = \mu + \sum_{i=1}^{p} \phi_1 y_{t-1} + \epsilon_t \tag{1.14}$$

then it is performed the same transformation (e.i. $y_{t-1}$is subtracted) getting

$$\Delta y_t = \mu + \delta y_{t-1} + \sum_{i=1}^{p} \beta_i \Delta y_{t-i} + \epsilon_t \tag{1.15}$$

where the two hypothesis are $H_0 : \delta = 0$ and $H_1 : \delta < 0$ and they are tested again with a *t-test* against the *Dickey-Fuller Distribution* making the same reasoning for rejecting or not rejecting the null hypothesis. Thus, this test allows to use process much more complex than a simple AR(1).

From a computational point of view, the ADF test can be easily performed in Python by using a package called *statsmodels.tsa.stattools* from which is imported the *adfuller* function. This function mainly returns the value of the test statistic and the relative P-value: throughout this paper will be used the P-value for assessing whether to reject or not the null hypothesis.

1.2 Importing the dataset

Throughout this paper it will be used just one dataset which, in turn, is created joining two different initial datasets. In this first, introductory sub chapter, there is a description of how this dataset is actually constructed and the techniques that are necessary to do so.
The first, largest dataset contains all financial information on the securities stored in the FinScience database and therefore, as a first step, it is necessary to check which price information might be irrelevant for the analysis to perform later. In fact, the columns concerning, for example, the simple opening price, the simple maximum, the simple minimum, the volume and so on are not relevant and therefore they will be eliminated when the dataset will be loaded: only the adjusted high, adjusted low and adjusted close price will be kept in. Moreover, it is important to underline that in this dataset each security is univocally recognized not through the commonly used ticker but through an alphanumeric code obtained through an internal mapping of FinScience, called FIGI. Furthermore, each title is composed of a time series of different duration and therefore, to best represent all this information, it should be necessary to take this aspect into consideration when creating a big, unique dataset. The result of all these operation is the DataFrame shown in Figure 1.1.

	BBG000CC7LQ7			BBG000BB65D0			BBG00564Y443			BBG000FFDM15	...	BBG000KC17R4	BBG000PPNW55		
	adj_high	adj_low	adj_close	adj_high	adj_low	adj_close	adj_high	adj_low	adj_close	adj_high	...	adj_close	adj_high	adj_low	adj_close
2018-01-02	105.14	102.750	103.71	125.0000	123.08	124.85	47.74	44.82	45.12	47.633285	...	92.19	3.9	3.75	3.80
2018-01-03	106.25	103.480	105.77	126.0800	124.73	125.19	46.20	44.71	44.97	48.005831	...	92.10	3.9	3.80	3.85
2018-01-04	108.11	105.840	107.86	128.8300	125.28	127.30	45.86	45.07	45.36	48.663265	...	92.68	3.9	3.80	3.85
2018-01-05	109.95	108.010	109.54	129.3600	127.42	129.34	45.67	44.95	45.42	48.650116	...	92.79	3.9	3.75	3.85
2018-01-08	111.40	109.570	110.63	130.3900	128.98	130.31	46.51	44.99	46.41	48.676413	...	93.99	3.9	3.80	3.85
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2022-02-18	427.36	418.415	420.31	291.4000	284.10	288.23	266.00	250.16	259.45	57.750000	...	229.64	NaN	NaN	NaN
2022-02-21	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN
2022-02-22	422.71	406.650	409.76	291.5200	280.42	281.80	264.66	250.97	251.73	58.230000	...	228.13	NaN	NaN	NaN
2022-02-23	418.16	398.460	399.29	287.2725	273.11	273.57	257.00	243.80	244.42	58.310000	...	221.09	NaN	NaN	NaN
2022-02-24	415.81	385.340	415.33	281.4000	265.99	281.39	267.04	230.01	264.80	55.480000	...	228.57	NaN	NaN	NaN

Figure 1.1: DataFrame containing the time series of prices

The second dataset, instead, is much simpler than the first one since it contains the *company name*, the *popularity*, the *sentiment* and other (useless) informations such as, for example, the *isin* and the *ticker* that have been deleted during the dataset's loading phase. The result is the DataFrame shown in the following figure:

		company_name	sentiment	popularity
figi	date			
BBG007WX14X0	2021-10-26	10X Genomics Inc	0.274523	3.240878e-08
	2021-10-27	10X Genomics Inc	0.418839	2.191002e-07
	2021-10-28	10X Genomics Inc	0.346820	2.390495e-08
	2021-10-29	10X Genomics Inc	-0.152014	3.346342e-10
	2021-10-30	10X Genomics Inc	0.099108	2.079387e-09
...	...	...	...	...
BBG000DM6PM0	2022-02-20	Zurich Insurance Group AG	-0.144989	6.028620e-07
	2022-02-21	Zurich Insurance Group AG	-0.148319	2.507145e-06
	2022-02-22	Zurich Insurance Group AG	-0.087387	1.850184e-06
	2022-02-23	Zurich Insurance Group AG	0.216375	5.752008e-07
	2022-02-24	Zurich Insurance Group AG	-0.003782	1.537162e-07

Figure 1.2: DataFrame containing popularity and sentiment

Therefore, the next and crucial step is to merge these two DataFrames

into one. To join the two tables in question it is used the *merge* function of Pandas and, at the same time, it is created a column called *pct change* containing the percentage changes of prices for each security.

In this phase, it's fundamental to remember one of the most important characteristic of the time series contained in the first, bigger, dataset: they have different length. Therefore, the merger phase is carried out by creating a sort of DataFrame that allows to separate each single stock while maintaining the corresponding time duration. The result of all these operations is the following Multi Index DataFrame which will be the reference dataset for all operations carried out from here on:

		company_name	sentiment	popularity	adj_close	adj_high	adj_low	pct_change
figi	**date**							
BBG000B9X8C0	2019-02-04	Ameren Corp	0.203000	4.573090e-07	63.829332	63.838584	62.820867	NaN
	2019-02-05	Ameren Corp	0.231896	2.180578e-07	63.690552	63.820080	63.255710	-0.002174
	2019-02-06	Ameren Corp	0.189137	8.789388e-07	63.736812	63.889469	63.357481	0.000726
	2019-02-07	Ameren Corp	0.207339	4.919589e-07	64.504725	64.513977	63.579528	0.012048
	2019-02-08	Ameren Corp	0.214255	4.483351e-07	65.198623	65.207875	64.282678	0.010757
...	...	...	...	...	...	...	...	...
BBG012MBFQD7	2022-02-08	Enfusion Inc	-0.029685	7.624231e-10	15.440000	15.630000	14.115000	0.098932
	2022-02-10	Enfusion Inc	0.305457	3.767604e-08	16.370000	16.940000	15.104700	0.060233
	2022-02-14	Enfusion Inc	-0.294499	4.864928e-08	17.030000	18.290000	16.050000	0.040318
	2022-02-17	Enfusion Inc	-0.035578	2.714194e-10	16.550000	17.939900	16.430000	-0.028186
	2022-02-24	Enfusion Inc	0.510079	6.312574e-09	15.910000	15.930000	13.940100	-0.038671

Figure 1.3: Principal DataFrame for the analysis

In this dataset, the total number of title is 847, but for reasons of simplicity, only a sample of this dataset will be used for the introductory analysis of this chapter. In particular, since the sample is more or less the 5% of the entire dataset, for explanatory reasons, only the following six titles will be shown as example of the whole extracted sample:

1. *Valero Energy Corp*: it's a U.S. company active in the field of oil and

gas refining;

2. *Shoe Carnival Inc*: it's an American family footwear retailer;

3. *Southern Co*: it sells electric utilities in three american states and natural gas distribution utilities in four, and provides wholesale energy, customized energy solutions and fiber optics and wireless communications;

4. *Lattice Semiconductor Corp*: it manufactures programmable electronic components. The company is the world's fourth largest manufacturer of FPGA devices and the second largest manufacturer of CPLDs and SPLDs;

5. *Wingstop Inc*: it's an American multinational chain of nostalgic aviation-themed restaurants specializing in chicken wings;

6. *Peabody Energy Corp*: Its main activity is the extraction, sale and distribution of coal, which is purchased for use in electricity generation and steel production;

Therefore, these six companies are very different from each other even if some of them work in similar sectors (like *Valero Energy Corp* and *Peabody Energy Corp*) but other work in different sectors such as catering, IT and footwear.

1.3 Volatility Analysis

The first aim of this analysis is to try to understand *if* and eventually *how much* the metrics of **popularity** and **sentiment** could help GARCH models in improving their performances. It has been chosen to use the GARCH model due to their peculiarity of taking into consideration also

the volatility of the historical series to model every instant in time, a characteristic that makes these models extremely functional for the analysis of financial historical series. Therefore, the logical steps of this chapter are to first check the stationarity of each input process, to look at the AutoCorrelation Function and Partial AutoCorrelation Function plot to understand possible starting GARCH models, then to find the best possible model for each security with and without the FinScience metric (taken alone or jointly) as exogenous variables and finally to compare the AIC and BIC values to assess the actual impact of the aforementioned metrics.

1.3.1 Stationarity, ACF and PACF plots

GARCH models are often performed, in the financial field, on the returns of a specific security which, usually, are stationary processes and therefore it should not be necessary to verify the stationarity of the inputs. However, in rare cases, it may happen that some particular returns are non-stationary and therefore it has been decided to perform the *Augmented Dickey-Fuller test* on the returns of each security randomly chosen.
To perform the ADF test on the sample, it is imported a Python package called *statsmodels.tsa.stattools* from which is used the *adfuller* function: this function mainly return the value of the test (*ADF statistic*) and the corresponding P-value. Afterwards, it is created a simple function that evaluates the value of the P-value of each ADF test and then it tells if the input is actually stationary or not.

```
The time series of the Valero Energy Corp's returns is stationary
The time series of the Shoe Carnival Inc's returns is stationary
The time series of the Southern Co's returns is stationary
The time series of the Lattice Semiconductor Corp's returns is stationary
The time series of the Wingstop Inc's returns is stationary
The time series of the Peabody Energy Corp's returns is stationary
```

Figure 1.1: Check of the stationarity of each sample's title

As shown in Figure 1.1, all the security's returns are stationary and therefore is possible to start with the analysis, plotting the AutoCorrelation and Partial AutoCorrelation Function plot for each title in the *sample* dataset.

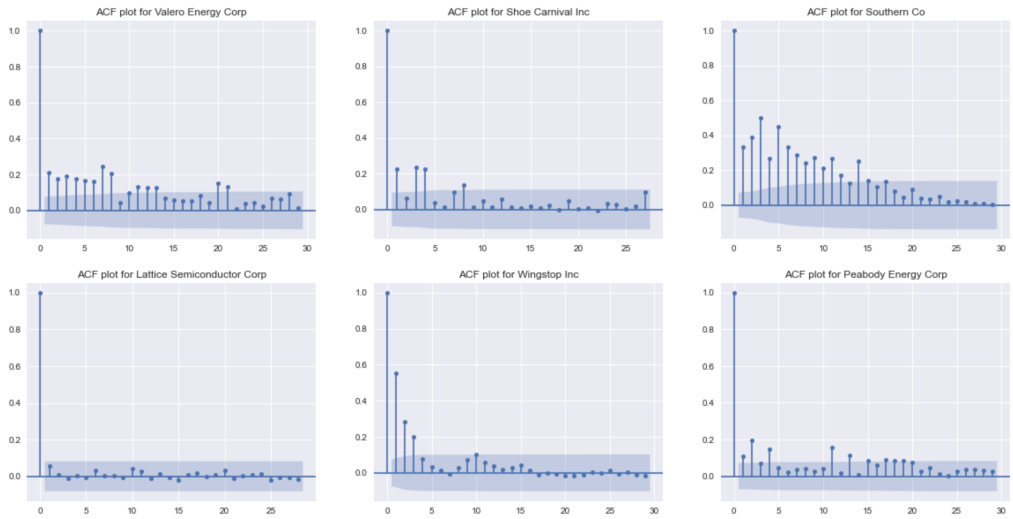


Figure 1.2: AutoCorrelation Function's plot for each title of the sample

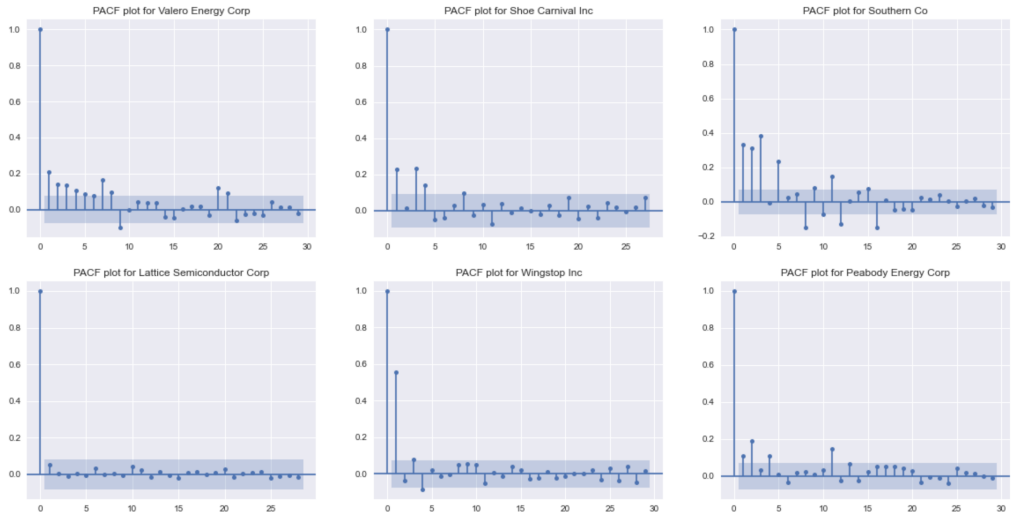


Figure 1.3: Partial AutoCorrelation Function's plot for each title of the sample

The ACF plot shows the correlation between a time series and itself lagged and therefore it is used to decide the optimal value of the p parameter (the lag order of the symmetric innovation) while the PACF plot is used for tune the q parameter (the lag order of lagged volatility).

At first sight it's possible to immediately see that the ACF and the PACF plot are quite similar to each other for some titles, in terms of significance. In fact, the two plots for *Valero Energy Corp* showed that the GARCH model to start with could be a GARCH(7,7) since both ACF and PACF have the seventh lag more significant than others. However, such a model will probably be too complex to handle and will also probably have too many not significant parameters.
For what concerns, instead, the plots for *Shoe Carnival Inc* it's possible to see that both ACF and PACF plots have the first and third spikes significant while the second is not: this suggest that it would be possible

to start with a GARCH(3,3) with probable certainty that the terms of two time spots before (e.i. a_{t-2}^2 and σ_{t-2}^2) will be not significant and therefore will be necessary to re-tuned the parameters. However, the third lag seems to be strongly significant and ignoring this information could prove fatal because, maybe, the third time spot could be fundamental for the overall performance of the model.

For *Southern Co* there are no doubts since in both ACF and PACF there are the first three spikes that are increasing in significance and, therefore, it's rather easy to asses that the best GARCH model to start with is the one with p and q equal to 3.

Going forward with this visual analysis, the plots for *Lattice Semiconductor Corp* are very particular: no spike is significant neither in the ACF nor in the PACF. In this cases there is no certainty about which model should be chosen, and therefore it will be implemented a GARCH(1,1) and it will be seen the significance of each terms for then eventually increase gradually the value of each parameter and check again the significance of each one, going on until reaching the maximum value for each term where each of them is significant.

Looking, instead, at the plots of *Wingstop Inc* it's possible to see that in both the ACF and PACF the first lag is strongly significant much over the others and therefore, the best model is rather simple to choose: it is a GARCH(1,1).

The last but not the least stock to check is the *Peabody Energy Corp*: both the AutoCorrelation Function and Partial AutoCorrelation Function plots showed clearly that the correct model to start with should be a GARCH(2,2) since the second spikes are both the highest.

Now that all the starting models have been defined, it's possible to go over with the analysis seeking for models with all significant parameters.

1.3.2 Finding the best GARCH model without exogenous variables

The next step in the analysis is to try to find the best model for each security, that means to look for models in which all the parameters are significant. Therefore, the first model to be performed over each stock is the one selected through the ACF and PACF plots and then, if there are some not significant parameters (e.i. redundant parameters) they will be phased out. So, taking the *Southern Co* as example, the first model to be applied should be a GARCH(3,3), with the following results:

Constant Mean - GARCH Model Results

Dep. Variable:	pct_change	R-squared:	0.000
Mean Model:	Constant Mean	Adj. R-squared:	0.000
Vol Model:	GARCH	Log-Likelihood:	2301.23
Distribution:	Normal	AIC:	-4586.46
Method:	Maximum Likelihood	BIC:	-4549.36
		No. Observations:	764
Date:	Mon, Apr 04 2022	Df Residuals:	763
Time:	10:52:05	Df Model:	1

Mean Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
mu	1.0691e-03	3.917e-04	2.729	6.350e-03	[3.013e-04,1.837e-03]

Volatility Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
omega	6.7857e-06	4.632e-10	1.465e+04	0.000	[6.785e-06,6.787e-06]
alpha[1]	0.0667	4.705e-02	1.417	0.156	[-2.554e-02, 0.159]
alpha[2]	0.0667	4.511e-02	1.478	0.139	[-2.174e-02, 0.155]
alpha[3]	0.0667	6.460e-02	1.032	0.302	[-5.994e-02, 0.193]
beta[1]	0.2600	0.436	0.596	0.551	[-0.595, 1.115]
beta[2]	0.2600	0.404	0.643	0.520	[-0.532, 1.052]
beta[3]	0.2600	0.263	0.987	0.324	[-0.256, 0.776]

Figure 1.4: GARCH(3,3) performed over *Southern Co*

Figure 1.4 shows that, except for the intercept (called *omega* here), all the parameters are not significant since each one has a P-value much higher than the significance threshold of 0.05 (since the test has been performed with a 95% of significance level). So, as said, the redundant parameters

(and therefore not significant) will be gradually eliminated up to obtain a model whose constituent elements resulted all with a P-value inferior to 0.05, as shown below.

Constant Mean - GARCH Model Results

Dep. Variable:	pct_change	R-squared:	0.000
Mean Model:	Constant Mean	Adj. R-squared:	0.000
Vol Model:	GARCH	Log-Likelihood:	2303.48
Distribution:	Normal	AIC:	-4596.95
Method:	Maximum Likelihood	BIC:	-4573.76
		No. Observations:	764
Date:	Mon, Apr 04 2022	Df Residuals:	763
Time:	10:52:05	Df Model:	1

Mean Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
mu	9.3014e-04	1.791e-05	51.947	0.000	[8.950e-04,9.652e-04]

Volatility Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
omega	6.7852e-06	1.575e-10	4.309e+04	0.000	[6.785e-06,6.785e-06]
alpha[1]	0.2000	5.237e-02	3.819	1.342e-04	[9.735e-02, 0.303]
beta[1]	0.3900	9.946e-02	3.921	8.806e-05	[0.195, 0.585]
beta[2]	0.3900	0.102	3.839	1.234e-04	[0.191, 0.589]

Figure 1.5: GARCH(1,2) performed over *Southern Co*

The same reasoning and procedure is then applied over the whole dataset called *sample*, obtaining the following results:

Constant Mean - GARCH Model Results

Dep. Variable:	pct_change	R-squared:	0.000
Mean Model:	Constant Mean	Adj. R-squared:	0.000
Vol Model:	GARCH	Log-Likelihood:	1415.84
Distribution:	Normal	AIC:	-2823.68
Method:	Maximum Likelihood	BIC:	-2805.74
		No. Observations:	655
Date:	Mon, Apr 04 2022	Df Residuals:	654
Time:	10:52:04	Df Model:	1

Mean Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
mu	8.3673e-04	8.613e-04	0.972	0.331	[-8.513e-04,2.525e-03]

Volatility Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
omega	2.3989e-05	4.020e-06	5.967	2.411e-09	[1.611e-05,3.187e-05]
alpha[1]	0.1000	3.222e-02	3.103	1.913e-03	[3.684e-02, 0.163]
beta[1]	0.8800	2.021e-02	43.534	0.000	[0.840, 0.920]

Constant Mean - GARCH Model Results

Dep. Variable:	pct_change	R-squared:	0.000
Mean Model:	Constant Mean	Adj. R-squared:	0.000
Vol Model:	GARCH	Log-Likelihood:	736.165
Distribution:	Normal	AIC:	-1464.33
Method:	Maximum Likelihood	BIC:	-1447.98
		No. Observations:	440
Date:	Mon, Apr 04 2022	Df Residuals:	439
Time:	10:52:05	Df Model:	1

Mean Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
mu	5.8658e-03	2.967e-03	1.977	4.804e-02	[5.065e-05,1.168e-02]

Volatility Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
omega	2.0028e-04	5.875e-04	0.341	0.733	[-9.512e-04,1.352e-03]
alpha[1]	0.3371	0.180	1.876	6.062e-02	[-1.503e-02, 0.689]
beta[1]	0.6629	0.378	1.752	7.978e-02	[-7.871e-02, 1.405]

Constant Mean - GARCH Model Results

Dep. Variable:	pct_change	R-squared:	0.000
Mean Model:	Constant Mean	Adj. R-squared:	0.000
Vol Model:	GARCH	Log-Likelihood:	2303.48
Distribution:	Normal	AIC:	-4596.95
Method:	Maximum Likelihood	BIC:	-4573.76
		No. Observations:	764
Date:	Mon, Apr 04 2022	Df Residuals:	763
Time:	10:52:05	Df Model:	1

Mean Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
mu	9.3014e-04	1.791e-05	51.947	0.000	[8.950e-04,9.652e-04]

Volatility Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
omega	6.7852e-06	1.575e-10	4.309e+04	0.000	[6.785e-06,6.785e-06]
alpha[1]	0.2000	5.237e-02	3.819	1.342e-04	[9.735e-02, 0.303]
beta[1]	0.3900	9.946e-02	3.921	8.806e-05	[0.195, 0.585]
beta[2]	0.3900	0.102	3.839	1.234e-04	[0.191, 0.589]

Constant Mean - GARCH Model Results

Dep. Variable:	pct_change	R-squared:	0.000
Mean Model:	Constant Mean	Adj. R-squared:	0.000
Vol Model:	GARCH	Log-Likelihood:	1116.55
Distribution:	Normal	AIC:	-2223.10
Method:	Maximum Likelihood	BIC:	-2201.23
		No. Observations:	587
Date:	Mon, Apr 04 2022	Df Residuals:	586
Time:	10:52:05	Df Model:	1

Mean Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
mu	3.8206e-03	1.390e-03	2.748	5.998e-03	[1.095e-03,6.546e-03]

Volatility Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
omega	1.3369e-04	1.017e-04	1.314	0.189	[-6.568e-05,3.331e-04]
alpha[1]	0.0851	4.496e-02	1.893	5.840e-02	[-3.024e-03, 0.173]
beta[1]	0.3700	0.128	2.889	3.864e-03	[0.119, 0.621]
beta[2]	0.4444	0.139	3.190	1.423e-03	[0.171, 0.717]

Constant Mean - GARCH Model Results

Dep. Variable:	pct_change	R-squared:	0.000
Mean Model:	Constant Mean	Adj. R-squared:	0.000
Vol Model:	GARCH	Log-Likelihood:	1595.33
Distribution:	Normal	AIC:	-3182.65
Method:	Maximum Likelihood	BIC:	-3164.40
		No. Observations:	709
Date:	Mon, Apr 04 2022	Df Residuals:	708
Time:	10:52:05	Df Model:	1

Mean Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
mu	3.4472e-03	1.062e-03	3.245	1.174e-03	[1.365e-03,5.529e-03]

Volatility Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
omega	2.5835e-04	5.259e-05	4.913	8.976e-07	[1.553e-04,3.614e-04]
alpha[1]	0.4343	0.123	3.520	4.315e-04	[0.192, 0.676]
beta[1]	0.2847	9.632e-02	2.956	3.116e-03	[9.594e-02, 0.474]

Constant Mean - GARCH Model Results

Dep. Variable:	pct_change	R-squared:	0.000
Mean Model:	Constant Mean	Adj. R-squared:	0.000
Vol Model:	GARCH	Log-Likelihood:	1023.44
Distribution:	Normal	AIC:	-2036.89
Method:	Maximum Likelihood	BIC:	-2013.73
		No. Observations:	758
Date:	Mon, Apr 04 2022	Df Residuals:	757
Time:	10:52:05	Df Model:	1

Mean Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
mu	-1.6229e-03	1.805e-03	-0.899	0.369	[-5.161e-03,1.915e-03]

Volatility Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
omega	7.2081e-05	7.664e-05	0.940	0.347	[-7.814e-05,2.223e-04]
alpha[1]	0.0979	4.269e-02	2.292	2.189e-02	[1.419e-02, 0.182]
beta[1]	0.5674	0.189	2.996	2.738e-03	[0.196, 0.939]
beta[2]	0.3270	0.172	1.901	5.730e-02	[-1.014e-02, 0.664]

Figure 1.6: Models with all significance parameters for each security of the sample

The order in the above figure is, from left to right and from above to below: *Valero Energy Corp*, *Shoe Carnival Inc*, *Southern Co*, *Lattice Semiconductor Corp*, *Wingstop Inc* and *Peabody Energy Corp.* As it is possible to see in Figure 1.6, in all the models all the parameters were significant and, therefore, summarizing in a more compact way:

- *Valero Energy Corp*: the model selected was a GARCH(1,1)
- *Shoe Carnival Inc*: the model selected was a GARCH(1,1)
- *Southern Co*: the model selected was a GARCH(1,2)
- *Lattice Semiconductor Corp*: the model selected was a GARCH(1,2)
- *Wingstop Inc*: the model selected was a GARCH(1,1)
- *Peabody Energy Corp*: the model selected was a GARCH(1,2)

So, all the redundant parameters have been eliminated and these are the best possible models when the input are only the returns of the corresponding stock.

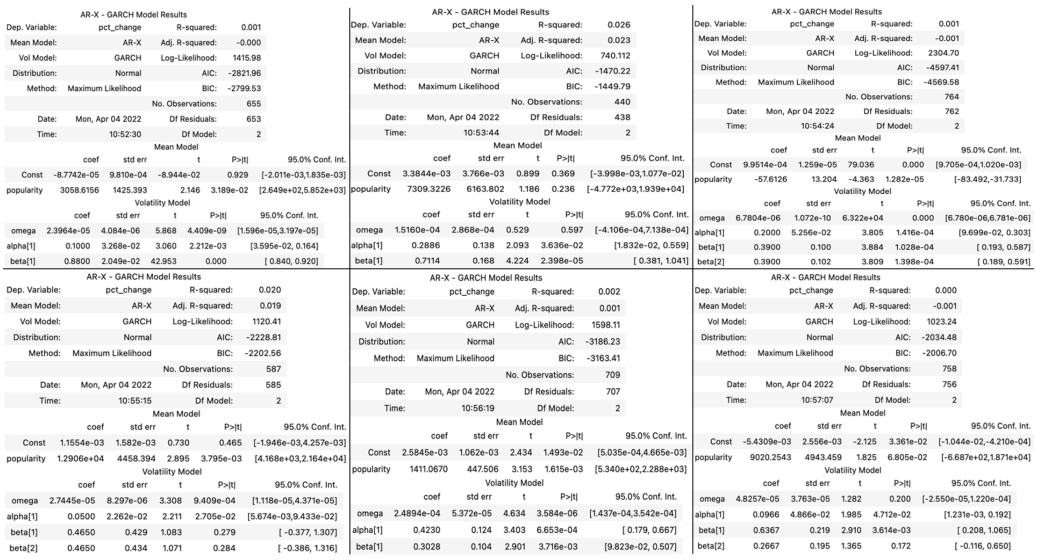
1.3.3 Finding the best GARCH model with exogenous variables

Now that all the best possible models have been selected for each security, it is time to understand *if* and eventually *how much* the FinScience metrics of **popularity** and **sentiment** actually affect the capability of GARCH models to capture the movement of a stock's time series.

To do that, are performed different analyzes:

- Implementation of the best GARCH model for each security with only the corresponding popularity as exogenous variable
- Implementation of the best GARCH model for each security with only the corresponding sentiment as exogenous variable
- Implementation of the best GARCH model for each security with corresponding popularity and sentiment as exogenous variables

Therefore, the models presented in the Figure 1.6 are taken as standard ones and then are added exogenous variables to see if, first of all, they are significant within the model and above all if they actually improve the performance of the corresponding GARCH model.
As said, the first analysis to be performed is the one with the **popularity** as exogenous variable and the results, for each security, are shown below:



AR-X - GARCH Model Results

Dep. Variable:	pct_change	R-squared:	0.001
Mean Model:	AR-X	Adj. R-squared:	-0.000
Vol Model:	GARCH	Log-Likelihood:	1415.98
Distribution:	Normal	AIC:	-2821.96
Method:	Maximum Likelihood	BIC:	-2799.53
		No. Observations:	655
Date:	Mon, Apr 04 2022	Df Residuals:	653
Time:	10:52:30	Df Model:	2

Mean Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
Const	-8.7742e-05	9.810e-04	-8.944e-02	0.929	[-2.011e-03,1.835e-03]
popularity	3058.6156	1425.393	2.146	3.189e-02	[2.649e+02,5.852e+03]

Volatility Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
omega	2.3964e-05	4.084e-06	5.868	4.409e-09	[1.596e-05,3.197e-05]
alpha[1]	0.1000	3.268e-02	3.060	2.212e-03	[3.595e-02, 0.164]
beta[1]	0.8800	2.049e-02	42.953	0.000	[0.840, 0.920]

AR-X - GARCH Model Results

Dep. Variable:	pct_change	R-squared:	0.026
Mean Model:	AR-X	Adj. R-squared:	0.023
Vol Model:	GARCH	Log-Likelihood:	740.112
Distribution:	Normal	AIC:	-1470.22
Method:	Maximum Likelihood	BIC:	-1449.79
		No. Observations:	440
Date:	Mon, Apr 04 2022	Df Residuals:	438
Time:	10:53:44	Df Model:	2

Mean Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
Const	3.3844e-03	3.766e-03	0.899	0.369	[-3.998e-03,1.077e-02]
popularity	7309.3226	6163.802	1.186	0.236	[-4.772e+03,1.939e+04]

Volatility Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
omega	1.5160e-04	2.868e-04	0.529	0.597	[-4.106e-04,7.138e-04]
alpha[1]	0.2886	0.138	2.093	3.636e-02	[1.832e-02, 0.559]
beta[1]	0.7114	0.168	4.224	2.398e-05	[0.381, 1.041]

AR-X - GARCH Model Results

Dep. Variable:	pct_change	R-squared:	0.001
Mean Model:	AR-X	Adj. R-squared:	-0.001
Vol Model:	GARCH	Log-Likelihood:	2304.70
Distribution:	Normal	AIC:	-4597.41
Method:	Maximum Likelihood	BIC:	-4569.58
		No. Observations:	764
Date:	Mon, Apr 04 2022	Df Residuals:	762
Time:	10:54:24	Df Model:	2

Mean Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
Const	9.9514e-04	1.259e-05	79.036	0.000	[9.705e-04,1.020e-03]
popularity	-57.6126	13.204	-4.363	1.282e-05	[-83.492,-31.733]

Volatility Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
omega	6.7804e-06	1.072e-10	6.322e+04	0.000	[6.780e-06,6.781e-06]
alpha[1]	0.2000	5.256e-02	3.805	1.416e-04	[9.699e-02, 0.303]
beta[1]	0.3900	0.100	3.884	1.028e-04	[0.193, 0.587]
beta[2]	0.3900	0.102	3.809	1.398e-04	[0.189, 0.591]

AR-X - GARCH Model Results

Dep. Variable:	pct_change	R-squared:	0.020
Mean Model:	AR-X	Adj. R-squared:	0.019
Vol Model:	GARCH	Log-Likelihood:	1120.41
Distribution:	Normal	AIC:	-2228.81
Method:	Maximum Likelihood	BIC:	-2202.56
		No. Observations:	587
Date:	Mon, Apr 04 2022	Df Residuals:	585
Time:	10:55:15	Df Model:	2

Mean Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
Const	1.1554e-03	1.582e-03	0.730	0.465	[-1.946e-03,4.257e-03]
popularity	1.2906e+04	4458.394	2.895	3.795e-03	[4.168e+03,2.164e+04]

Volatility Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
omega	2.7445e-05	8.297e-06	3.308	9.409e-04	[1.118e-05,4.371e-05]
alpha[1]	0.0500	2.262e-02	2.211	2.705e-02	[5.674e-03,9.433e-02]
beta[1]	0.4650	0.429	1.083	0.279	[-0.377, 1.307]
beta[2]	0.4650	0.434	1.071	0.284	[-0.386, 1.316]

AR-X - GARCH Model Results

Dep. Variable:	pct_change	R-squared:	0.002
Mean Model:	AR-X	Adj. R-squared:	0.001
Vol Model:	GARCH	Log-Likelihood:	1598.11
Distribution:	Normal	AIC:	-3186.23
Method:	Maximum Likelihood	BIC:	-3163.41
		No. Observations:	709
Date:	Mon, Apr 04 2022	Df Residuals:	707
Time:	10:56:19	Df Model:	2

Mean Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
Const	2.5845e-03	1.062e-03	2.434	1.493e-02	[5.035e-04,4.665e-03]
popularity	1411.0670	447.506	3.153	1.615e-03	[5.340e+02,2.288e+03]

Volatility Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
omega	2.4894e-04	5.372e-05	4.634	3.584e-06	[1.437e-04,3.542e-04]
alpha[1]	0.4230	0.124	3.403	6.653e-04	[0.179, 0.667]
beta[1]	0.3028	0.104	2.901	3.716e-03	[9.823e-02, 0.507]

AR-X - GARCH Model Results

Dep. Variable:	pct_change	R-squared:	0.000
Mean Model:	AR-X	Adj. R-squared:	-0.001
Vol Model:	GARCH	Log-Likelihood:	1023.24
Distribution:	Normal	AIC:	-2034.48
Method:	Maximum Likelihood	BIC:	-2006.70
		No. Observations:	758
Date:	Mon, Apr 04 2022	Df Residuals:	756
Time:	10:57:07	Df Model:	2

Mean Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
Const	-5.4309e-03	2.556e-03	-2.125	3.361e-02	[-1.044e-02,-4.210e-04]
popularity	9020.2543	4943.459	1.825	6.805e-02	[-6.687e+02,1.871e+04]

Volatility Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
omega	4.8257e-05	3.763e-05	1.282	0.200	[-2.550e-05,1.220e-04]
alpha[1]	0.0966	4.866e-02	1.985	4.712e-02	[1.231e-03, 0.192]
beta[1]	0.6367	0.219	2.910	3.614e-03	[0.208, 1.065]
beta[2]	0.2667	0.195	1.365	0.172	[-0.116, 0.650]

Figure 1.7: Best GARCH models with popularity as exogenous variable

Again, the order in Figure 1.7 is, from left to right: in the above section there are *Valero Energy Corp*, *Shoe Carnival Inc*, *Southern Co* and in the

below part there are *Lattice Semiconductor Corp*, *Wingstop Inc* and *Peabody Energy Corp.*

Analyzing the impact that the FinScience metric has on each model, it's possible to see that in the 66% (four over six) of the cases reported here it is actually significant: *Valero Energy Corp* with a P-value of 0.03, *Southern Co* with a P-value of 0.0000128, *Lattice Semiconductor Corp* with a P-value of 0.003795 and *Wingstop Inc* with a P-value of 0.0016; while in the other cases the popularity has P-values much higher than the threshold of 0.05, in particular: 0.236 for *Shoe Carnival Inc* and 0.068 for *Peabody Energy Corp.* As a result, it appears that the **popularity** actually helps GARCH models in making their predictions more accurate in 66% of the cases, which suggests that this metric probably carries valuable informations.

Subsequently, the same kind of procedure is implemented but, this time, with the FinScience metric of **sentiment** as exogenous variable. The result are again in the same, usual, order:

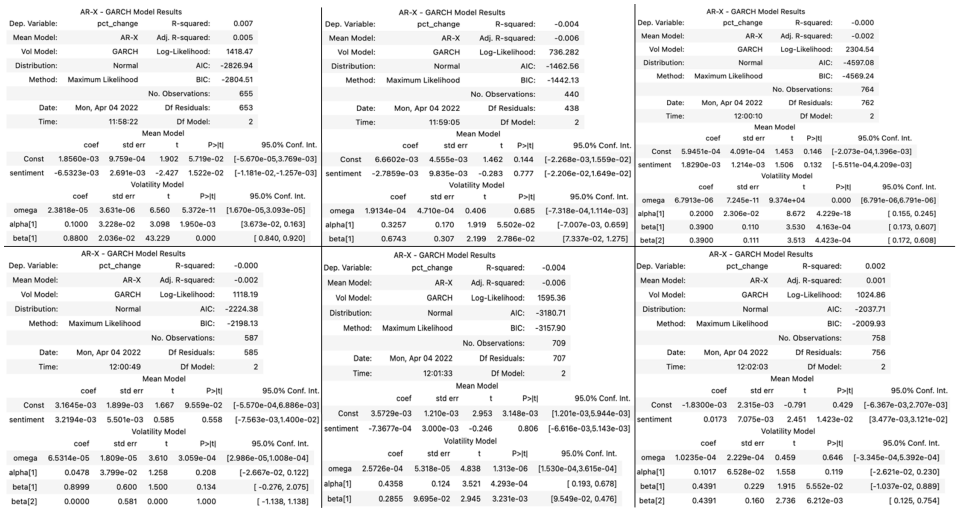


Figure 1.8: Best GARCH models with sentiment as exogenous variable

At a first sight seems that the number of models in which this metrics is actually significant is lower than result obtained through the popularity. In fact, looking more in depth, only for the first and the last title, that are *Valero Energy Corp* and *Peabody Energy Corp* the **sentiment** has an impact in the GARCH model, while in the remaining four models there are P-values too far from the crucial threshold, namely: 0.777 for *Shoe Carnival Inc*, 0.132 for *Southern Co*, 0.558 for *Lattice Semiconductor Corp* and 0.806 for *Wingstop Inc.*
Therefore, mainly because of the last four P-values mentioned above, the sentiment metric does not seem to have any impact on GARCH models when is alone taken.

However, it may happen that **sentiment** and **popularity** taken alone are less impactful than when they are put together as exogenous variables: it may be that together they compensate for each other's missing information. To explore this opportunity, the GARCH models in the Figure 1.6 are implemented using the two FinScience metrics as exogenous variables, and the results are as follows:

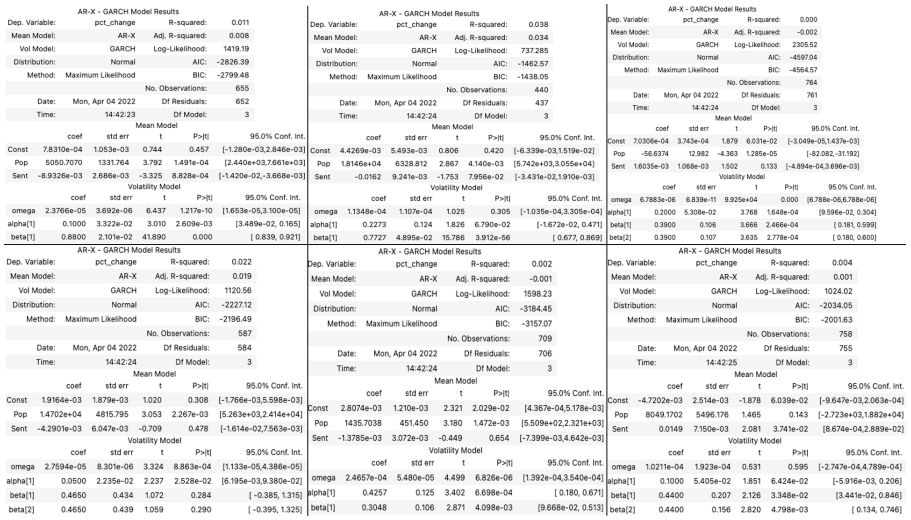


Figure 1.9: Best GARCH models with sentiment and popularity as exogenous variables

The results showed in Figure 1.9 are rather surprising: **in all models there is at least one between popularity and sentiment that is significant**.

In particular, for what concerns the first title, *Valero Energy Corp*, both the FinScience metrics are significant with a P-value of 0.00015 for the popularity and 0.00088 for the sentiment.

However, the *Valero Energy Corp* is the only case in which both metrics are significant: in all the other 5 titles only one between popularity and sentiment results actually significant. Going deeper in details, *Shoe Carnival Inc* has a significant popularity but a not-significant sentiment with a P-value of more or less 0.08: here, the interesting thing is that when the popularity was taken alone in Figure 1.7, it was not-significant while, as said, when it is paired up with the sentiment it becomes strongly significant. *Southern Co*, instead, is not surprising since when the two metrics were taken alone only the popularity resulted to be significant and the same

situation occurs when they are taken together.
Looking at the result of the GARCH(1,2) over *Lattice Semiconductor Corp* it's possible to see that only the popularity is significant with a P-value of 0.002 while the sentiment has 0.478 as P-value.
The last two titles are one the opposite of the other: *Wingstop Inc* has only a significant popularity while *Peabody Energy Corp* has only the sentiment and in both cases the result when the metrics are taken alone or together does not change.

1.3.4 Model comparison

To understand if the FinScience metrics of sentiment and popularity actually has an impact on the performance of GARCH models, it's possible to apply the usual model selection performed through the AIC and BIC values, that tell which model is the best taking into account different criteria. In particular, the following figure represent a table containing the AIC values of each model

Stock name	GARCH model	Simple GARCH model's AIC	GARCH with popularity's AIC	GARCH with sentiment's AIC	GARCH with sentiment and popularity's AIC
Valero Energy Corp	(1,1)	-2823.679583	-2821.957809	-2826.935732	-2826.389757
Shoe Carnival Inc	(1,1)	-1464.329110	-1470.225033	-1485.050670	-1462.569732
Southern Co/The	(1,2)	-4596.952032	-4597.407216	-4597.072599	-4597.037146
Lattice Semiconductor Corp	(1,2)	-2223.104100	-2228.810404	-2224.376167	-2227.115125
Wingstop Inc	(1,1)	-3182.651380	-3186.227943	-3180.714651	-3184.451281
Peabody Energy Corp	(1,2)	-2032.595298	-2034.481533	-2044.913526	-2034.045763

Figure 1.10: AIC values for each model

and it tells that

- for *Valero Energy Corp* the best selected model is the one with **only the sentiment as exogenous variable**, with an AIC value of -2826.94
- for *Shoe Carnival Inc* the best selected model is the one with **only the sentiment as exogenous variable**, with an AIC value of -1485

- for *Southern Co* the best selected model is the one with **only the popularity as exogenous variable**, with an AIC value of -4597.4

- for *Lattice Semiconductor Corp* the best selected model is the one with **only the popularity as exogenous variable**, with an AIC value of -2228

- for *Wingstop Inc* the best selected model is the one with **only the popularity as exogenous variable**, with an AIC value of -3186

- for *Peabody Energy Corp* the best selected model is the one with **only the sentiment as exogenous variable**, with an AIC value of -2044.9

Moreover, the results obtained using the Bayesian information criterion are the following

Stock name	GARCH model	Simple GARCH model's BIC	GARCH with popularity's BIC	GARCH with sentiment's BIC	GARCH with sentiment and popularity's BIC
Valero Energy Corp	(1,1)	-2805.741042	-2799.534633	-2804.512556	-2799.481946
Shoe Carnival Inc	(1,1)	-1447.982011	-1449.791159	-1460.530021	-1438.049083
Southern Co/The	(1,2)	-4573.759193	-4569.575809	-4569.241192	-4564.567171
Lattice Semiconductor Corp	(1,2)	-2201.228976	-2202.560255	-2198.126018	-2196.489952
Wingstop Inc	(1,1)	-3164.395958	-3163.408665	-3157.895373	-3157.068148
Peabody Energy Corp	(1,2)	-2009.441881	-2006.697432	-2017.129426	-2001.630979

Figure 1.11: BIC values for each model

In fact, in this case:

- for *Valero Energy Corp* the best selected model is the one **without exogenous variables**, with an BIC value of -2805, even if the model with the sentiment is basically equivalent with a BIC value of -2804.5

- for *Shoe Carnival Inc* the best selected model is the one with **only the sentiment as exogenous variable**, with an BIC value of -1460

- for *Southern Co* the best selected model is the one **without exogenous variables**, with an BIC value of -4573
- for *Lattice Semiconductor Corp* the best selected model is the one with **only the popularity as exogenous variable**, with an BIC value of -2202
- for *Wingstop Inc* the best selected model is the one **without exogenous variables**, with an BIC value of -3164
- for *Peabody Energy Corp* the best selected model is the one with **only the sentiment as exogenous variable**, with an BIC value of -2017

Therefore, making an average between the two criteria, it seems that in the majority of the cases, the GARCH model exploits the information carried by the FinScience metrics.
Thus, even if visually is not possible to grasp the difference as shown here

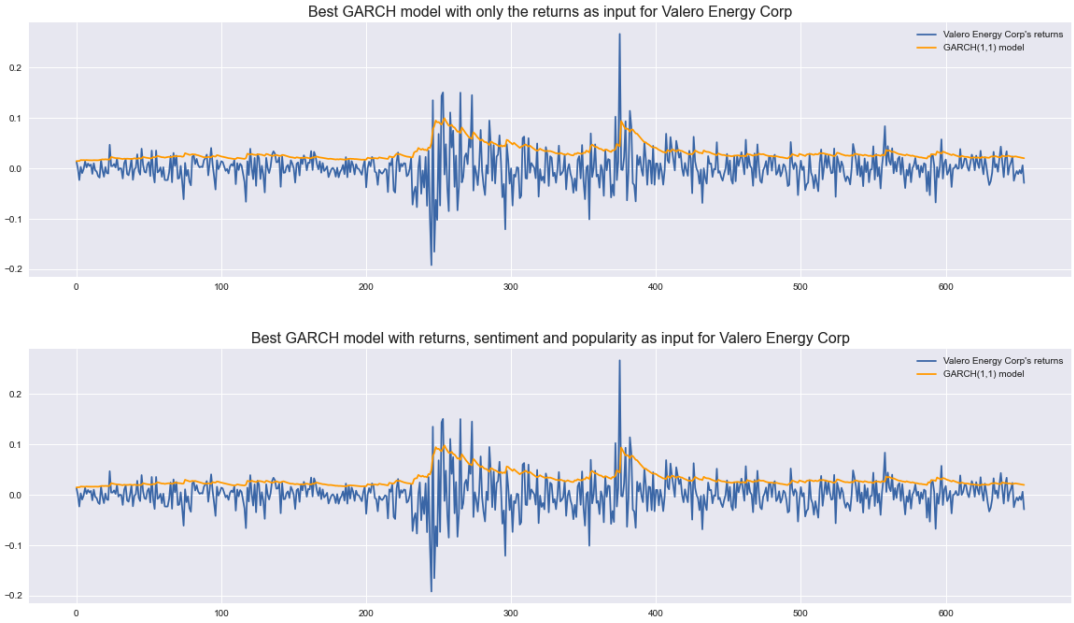


Figure 1.12: Plot of returns and best GARCH models

the results obtained through this first chapter allow one to think that it might be meaningful to probe the utility of such metrics within more complex models (e.g. machine learning models), and, moreover, that it would be reasonable to shift the focus from volatility to returns, given the good results achieved so far.

Chapter 2

Machine Learning for Trading

The aim of this chapter is to try to create a new financial technical indicator through machine learning models applied over both traditional and alternative data with, of course, a particular attention over the FinScience metrics of *popularity* and *sentiment*.

In this chapter will be shown two different algorithms belonging to the same family although, in phase of analysis, there have been tried algorithms coming from different families (*Support Vector Machines*, *Simple Neural Network* and *Long-Short-Term-Memory Neural Networks*) obtaining however scarce results. This is probably due to the type of data analyzed here, which seems not to be suitable for the above models: we will see later possible explanations in details.

As usuale, before starting with the presentation of the experiments and corresponding results, will be provided a brief introduction to the theory of the models that will be used for the analysis.

2.1 Theoretical concepts in pills

Although several models were implemented during the analysis phase, only two actually provided consistent results: XGBoost and LightGBM. Both these models come from the same macro-family of DecisionTree-based Algorithms, but they are the most complicated and performant models since they represent the most advanced level of boosting.

Figure 2.1: Evolution of DecisionTree-based algorithms

In particular, boosting is a technique used for building models from a single *weak learner* (e.i. a less efficient and much simpler algorithms like, as said, *Decision Trees*), in an iterative way and the power of this technique is that, with it, it's possible to implement either regression, classification or ranking algorithms. By the way, all the simplest boosting algorithms are described by the following equation:

$$F(x) = \sum_{i=1}^{M} f_i(x) \qquad (2.1)$$

where $F(x)$ is a predictive boosting function and $f_i(x)$ are the M so called *weak learners*.

To make an example, when there are few features to evaluate, then it's possible to use a simple model like *Decision Trees* but when there are tens or hundreds of features, then it should be much more efficient to use more complex algorithms like *Random Forest* (that uses multiple decision trees

with a subset of features for each one). So, what is the difference between *Random Forest* algorithm and *Boosting*? The main difference between these algorithms is that unlike *Random Forest*, *Boosting* builds models that are not constructed upon random subsets of features but it sequentially puts more weights on instances which have led to wrong predictions: *Boosting* learns from past errors.

The next-level algorithm of this family is the *Gradient Boosting* that uses gradient descent to minimize the loss function and it's crucial to first understand the mechanisms behind it to then completely understand the more complex algorithms construct upon this technique. The models based on gradient boosting are very useful and widely used for solving real world situations. To understand the idea of *Gradient Boosting* let's fix two conditions: the first one is that it's necessary the definition of a differentiable (hopefully in an efficient and quick way) loss function (e.g. $L(y, \hat{y})$), while the second one is to start with a very simple model that it's easy to improve. The key step is to compute the derivative (or gradient) shown in equation 2.2, that is the derivative of the loss function $L(y_i, \hat{y}_i)$ with respect to the prediction $\hat{y}_i$ in order to look at the sign of the gradient to understand in which direction should the prediction be moved to decrease (hopefully minimize) the loss function (e.i. if the gradient is negative the prediction will be moved to the right, while if the gradient is positive then the prediction will be moved to the left, according with the following graph).

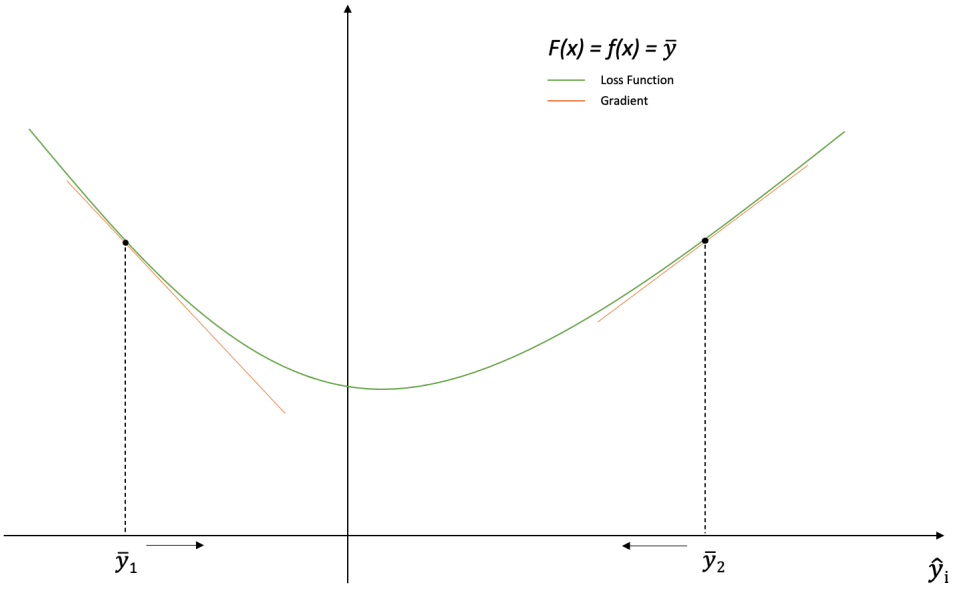


Figure 2.2: Example of gradient analysis with a simple predictive function F(x)

$$\hat{r_{ji}} = - \left. \frac{\partial L(y_i, \hat{y}_i)}{\partial \hat{y}_i} \right|_{\hat{y}_i = F_j(x_i)} \tag{2.2}$$

In the above equation, j is the current weak learner and i is the current data point which we are interested in. After having defined the gradients $\hat{r_{ji}}$, the next step is to fit the new weak learner over the sample set of feature (e.i. the original dataset) with the $\hat{r_{ji}}$ as target variable. The key question now is: how much of the new model add to the previous one? The answer of this question is provided by the result of the following equation:

$$\hat{\gamma_2} = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{N} L(y_i, f_1(x_i) + \gamma f_2(x_i)) \tag{2.3}$$

where there is the sum over N (every single data point) of the loss between the true value y_i and the new prediction proposed, generated by the concatenation of the previous weak learner and the new one, times γ. Therefore,

the model $F_2(x)$ now is $F_2(x) = f_1(x) + \hat{\gamma} f_2(x)$ which should be better than F_1 because the predictions ($\overline{y}$ in Figure 2.2) have been moved closer to the minimum of the loss function. Then, the process continue to differentiate the loss function and to find the optimal amount of each new weak learner to add to the model, till the end: therefore, the general formula is the following one.

$$F_{m+1}(x) = f_m(x) + \hat{\gamma}_{m+1} f_{m+1}(x) \tag{2.4}$$

$$\approx F_m(x) + \hat{\gamma}_{m+1} \hat{r_m} \tag{2.5}$$

$$= F_m(x) + \hat{\gamma}_{m+1} \frac{\partial L(y_i, F_m(x))}{\partial F_m(x)} \tag{2.6}$$

and therefore, in words, this means that the current model is the result of the previous model $F_m(x)$ plus a little step (e.i. $\hat{\gamma}_{m+1}$) in the direction of the decreasing loss function (e.i. $\frac{\partial L(y_i, F_m(x))}{\partial F_m(x)}$).
The cons of *Gradient Boosting* are that, at the final iteration when $F_m(x)$ is defined, it is hard to be interpreted and therefore we sacrifice some interpretability in exchange of high performances, and the second cons is that these algorithms could be computationally hard to manage.

2.1.1 Theory of XGBoost model

The *Extreme Gradient Boosting* (*XGBoost* or XGB from here on), has been developed only 7 years ago, in 2015 by the University of Washington and has been credited with a lot of winnings in Kaggle competitions. The main characteristics of this algorithms are the following:

- It computes 2^{nd} order gradients to understand the direction of gradients themselves
- It uses L1, L2 regularizations and tree pruning to prevent overfitting

- It parallelizes on your own machine for improving the velocity of the computations

Suppose there are K trees, then the model is

$$\sum_{k=1}^{K} f_k \qquad (2.7)$$

where each f_k is a prediction from a decision tree and, therefore, the model is composed by many decision trees. In particular, having all the decision trees, it's possible to make predictions by

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i) \qquad (2.8)$$

where x_i is the feature for the i-th data point. With this model construction, the train phase is carried by optimizing a loss function, which in case of multi-classification (like in our case) is the ***mlogloss***:

$$L = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{i,j} log(p_{i,j}) \qquad (2.9)$$

Moreover, another important aspect is the regularization phase, in which the model controls its complexity preventing the overfitting problem. In fact, for *XGBoost*, the authors define the following regularizing equation:

$$\Omega = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} {\omega_j}^2 \qquad (2.10)$$

where T is the number of leaves on a tree, ω_j^2 is the score on the j-th leaf of that tree, γ and λ are instead the parameters used for controlling the degree of regularization. Therefore, putting together the loss function and the regularization term, it's possible to get the objective of the model:

$$Obj = L + \Omega \tag{2.11}$$

where the loss function L is used for controlling the predictive power and the regularization term Ω is used for controlling the complexity of the model itself. Afterwards, XGBoost exploits the ***Gradient Descent*** to optimize the objective: it is an iterative technique that computes the following equation at each iteration (given an objective function).

$$\frac{\partial Obj(y,\hat{y})}{\partial \hat{y}} \tag{2.12}$$

Then, the prediction $\hat{y}$ is improved along with the direction of the gradient, to minimize the objective (actually, in order to make XGBoost to convert faster, it also takes into consideration the second order gradient using the Taylor approximation since not all the objective functions have derivatives). Therefore, in the end, removing all the constant terms the resulting objective function is

$$Obj^{(t)} = \sum_{i=1}^{n} [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) \tag{2.13}$$

where $g_i = \dfrac{\partial l(y_i, \hat{y}^{(t-1)})}{\partial \hat{y}^{(t-1)}}$ and $h_i = \dfrac{\partial^2 l(y_i, \hat{y}^{(t-1)})}{\partial \hat{y}^{(t-1)}}$. Therefore, the equation 2.13 is the objective at the t-th step and the goal of the model is to find a f_t that optimizes it.

The main problem is that the aforementioned f_t **is a tree and not a**

number: so how is possible to build a tree that improves predictions along with the gradient? To find such a tree it's necessary to answer to two further questions:

1. How to find a good tree structure?

2. How to assign prediction scores?

First, let's assume to already have the answer to question 1 and let's try to answer at question number two. It's possible to define a tree as

$$f_t(x) = \omega_{q(x)} \tag{2.14}$$

where $q(x)$ is a "directing" function which assigns every data point to the $q(x)$-th leaf. Therefore, it's possible to describe the prediction process as:

- Assign the data point x to a leaf by q
- Assign the corresponding score $\omega_{q(x)}$ on the $q_(x)$-th leaf to the data point

Then, it's necessary to define the set that contains the indices of the data points that are assigned to the j-th leaf as following:

$$I_j = \{i|q(x_i) = j\} \tag{2.15}$$

So, now it's possible to rewrite the objective function as

$$Obj^{(t)} = \sum_{i=1}^{n} [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{t} \omega_j^2 \tag{2.16}$$

$$= \sum_{j=1}^{T} [(\sum_{i \in I_j} g_i)\omega_j + \frac{1}{2}(\sum_{i \in I_j} h_i + \lambda)\omega_j^2] + \gamma T \tag{2.17}$$

In equation 2.16, the summation is over the data points but in the equation 2.17 the summation is performed leaf by leaf for all the T leaves. Since it is a quadratic problem of ω_j, the optimal value is

$$\omega_j^* = \frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda} \tag{2.18}$$

and therefore, simply substituting, the corresponding value of the objective function is

$$Obj^{(t)} = -\frac{1}{2} \sum_{j=1}^{T} \frac{(\sum_{i \in I_j} g_i)^2}{\sum_{i \in I_j} h_i + \lambda} + \gamma T \tag{2.19}$$

where the leaf score (e.i. ω_j) is always related to the first and second order of the loss function g and h and the regularization parameter λ. This is how it's possible to find the score associated to a leaf assuming to know the structure of a tree.

By the way, now it's time to come back to the first question: how to find a good tree structure? Since this is a tough question, more difficult than the previous one, it would be better to split it into two sub-questions:

1. How to choose the feature split?
2. When stop the split?

Starting, this time, from the question number one, the first thing to say is that in any split the goal is of course to find the best split-point that will optimize the objective function and, therefore, for each feature is crucial to first sort the numbers, then scan the best split-point and finally choose the

best feature.

Every time that a split is performed, a leaf is transformed into an internal node having leaves with different scores than the initial one.

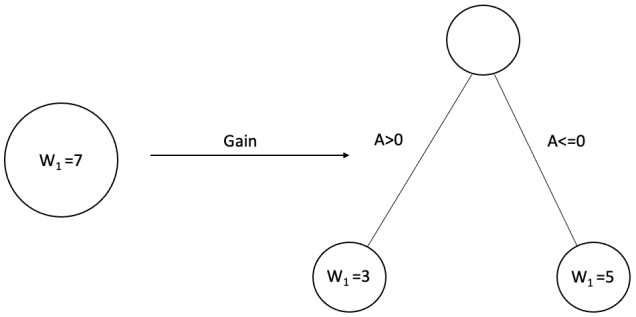


Figure 2.3: Example of leaf split with corresponding scores

In the example shown in Figure 2.3, the leaf with score W=7 is replaced with an internal node based on feature A with threshold 0 having two leaves with different scores (e.i. 3 and 5). Clearly, the principal aim is to calculate the gain (or the eventual loss) obtained from a such a split. Usually, in other tree-based algorithm the computation of this gain is generally made through the Gini index or entropy metric, but here in XGBoost this calculation is based on the objective function. In particular, XGBoost exploits the set of indices I of data points assigned to the node, where I_L and I_R are the subsets of indices of data points assigned to the two new leaves. Now, recalling that the best value of the objective function on the j-th leaf is the equation 2.19 without the first summation and the T value in the last term, the gain of the split is:

$$gain = \frac{1}{2}\left[\frac{(\sum_{i\in I_L} g_i)^2}{\sum_{i\in I_L} h_i + \lambda} + \frac{(\sum_{i\in I_R} g_i)^2}{\sum_{i\in I_R} h_i + \lambda} - \frac{(\sum_{i\in I} g_i)^2}{\sum_{i\in I} h_i + \lambda}\right] - \gamma \qquad (2.20)$$

where

- $\frac{(\sum_{i \in I_L} g_i)^2}{\sum_{i \in I_L} h_i + \lambda}$ is the value of the left leaf

- $\frac{(\sum_{i \in I_R} g_i)^2}{\sum_{i \in I_R} h_i + \lambda}$ is the value of the right leaf

- $\frac{(\sum_{i \in I} g_i)^2}{\sum_{i \in I} h_i + \lambda}$ is the objective of the previous leaf

- γ is the parameter which controls the number of leaves (e.i. the complexity of the algorithm)

to understand whether transforming one leaf into two new leaves there is an improvement of the objective or not it's enough to look at the value (positive or negative) of this gain.
Therefore, in conclusion, the XGBoost algorithm, to build a tree, first finds the best-split point recursively until the maximum depth (specifiable by the user) is reached and then it prunes out the nodes with negative gains with a bottom-up order.

2.1.2 Theory of LightGBM model

In this section we will talk about the LightGBM algorithm without going too much into the details since it works more or less like XGBoost with only some differences that we will discuss. LightGBM is a fast, distributed, high-performance tree-based gradient boosting framework developed by Microsoft and released in January 2016. The most important features of this algorithm that differentiates it from XGBoost are the faster training speed, the better accuracy, the fact that it supports parallel, distributed and GPU learning and that it is capable to handle large-scale data. Another main difference between LightGBM and XGBoost is the way in which they

grow a tree: the first one uses **leaf-wise tree growth** while the second uses level-wise tree growth.

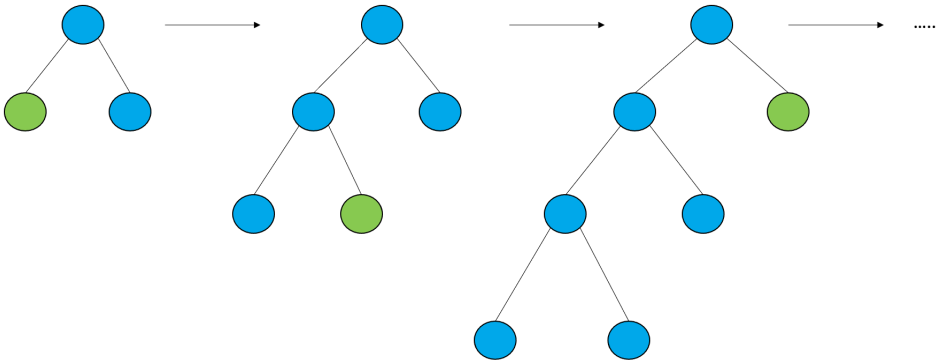


Figure 2.4: Example of leaf-wise tree growth used by LightGBM

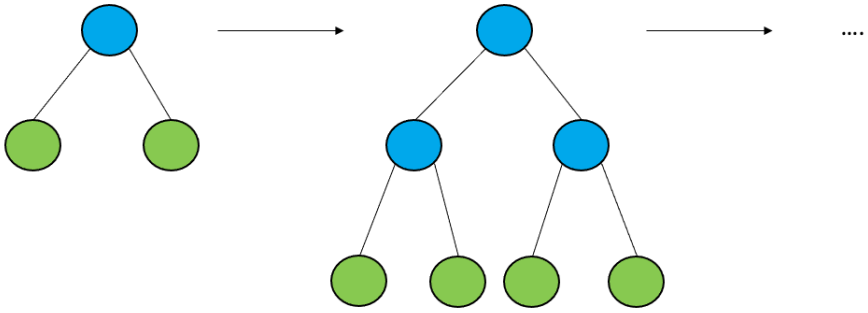


Figure 2.5: Example of level-wise tree growth used by XGBoost

The first thing that makes LightGBM faster is the way it sorts the numbers: this algorithm takes the inputs and divides them into bins (e.i. boxes): for example, if a feature has values 50, 55, 60, 70, 75 and 80, LightGBM divides them into two categories like 50-60 and 70-80, then it sorts them, tests them and finds the optimal split. So, previously there were six candidates for the split, while after the division into categories there are only two candidates, reducing a lot the computation effort needed to test all the possible combinations of splits .This process, called *Histogram* or *Bin way of*

splitting, clearly makes computations much faster than the ones of XGBoost.

The second improving characteristic of this algorithm is called *Exclusive Feature Bundling* (EFB), which reduces the dimension of the features that are mutually exclusive. For example, if there are two features called *Green* and *Red* which correspond to the color of a financial candle, taking either value 1 or 0 based on the corresponding candlestick's color, then these features are mutually exclusive since a candle can't be green and red at same time. Therefore, EFB could work on these features in the following way:

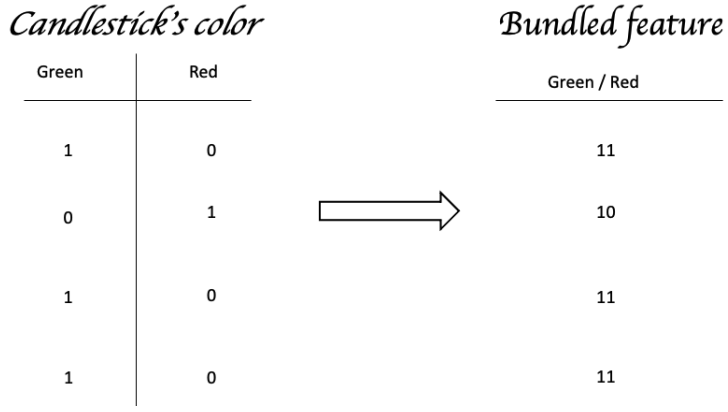


Figure 2.6: Example of feature bundling

So, this process creates a new bundled feature with a lower dimension, using new values for identifying the two cases, which in this situation are number 11 for the green candle and number 10 for the red one. Therefore, by reducing the dimensionality of some of the input features, the algorithm is able to run faster, since it has less features to evaluate.

The third characteristic the differs LGBM from XGB is the so called *Gradi-*

ent Based One Side Sampling (GOSS), which help the LGBM algorithm to iteratively choose the sample to use for the computations. Suppose that the dataset used has 100 features, then the algorithm computes 100 gradient $G_1, G_2, ..., G_{100}$ and sorts them in descending order, for example $G_{73}, G_{24}, ..., G_8$. Then, the first 20% of these records are taken out and an additional 10% is also randomly taken from the remaining 80% of gradient records. Therefore, since the gradients are descending ordered, the algorithm takes only the 10% of the features on which it performs well and the 20% of those features on which it performs bad (high gradients and high errors) and, so, on which it still has a lot to learn. Afterwards, these two percentages of features are put together creating the sample on which the LGBM trains, calculates again the gradients and applies again the GOSS in an iterative way.

All these specifications, hence, make the LGBM algorithm faster and more performant especially when run on large datasets with many features.

2.2 Pre-Processing Phase

First, it's important to recall that the starting dataset is the one shown in Figure 1.3, containing simply the names of the companies, the historical adjusted close, high and low prices, the daily popularity, sentiment and the simple returns. Therefore, the first additional informations to insert are the variations of popularity and sentiment for then evaluating whether point values or the changes themselves for these two metrics are more influential. In this phase is also crucial to look for missing or infinite values that would lead to an error during the model training step. So, it should be pretty easy to assess that in the variations of popularity and sentiment there may not be missing values, but there could be infinite values due to sudden spikes in the

news about some company and, so, all these infinite values are substitute with a very large (but finite) value. Moreover, the daily point value of the two FinScience metrics may contain some missing values, corresponding to those days in which there were not news about that particular stock. Hence, these NaNs are filled in the following way:

- *Popularity's NaNs*: since, as said, they represent days in which there were not news about the stock takes into account, it's correct to fill them with a 0 value

- *Sentiment's NaNs*: in this case it wouldn't be correct to fill them with a simple 0, because the value zero has a specific meaning in this metric (the news is neutral, that is neither positive nor negative for the company). Therefore, these NaNs are filled with the average of the last 7 days in a rolling way.

Now that the dataset has been enriched, is time to understand which will be the kind of problem to face. The purpose of this document is to build a financial technical indicator that can indicate the possible presence of a price trend and its direction and, therefore, the problem to deal with will be a classification one since the directions that the prices can take are only three:

- **Upward trend** to which, for example, can be associated the numerical label "2"

- **Lateral phase** to which, for example, can be associated the numerical label "1"

- **Downward trend** to which, for example, can be associated the numerical label "0"

Hence, it's necessary to construct such labels in order to carry on this classification problem using supervised machine learning models like the aforementioned XGBoost and LightGBM.

2.2.1 Labels Creation

Generally, a common and widely used technique to detect the trend of prices is to calculate simple returns and then investigate the sign of each daily return to asses the price movement. However, this procedure would be too granular, making the labels too sensitive and therefore losing sight of the longer-term trend (weekly), which is quite useful information in trading, as well as the research objective of this study.
So, in order to better catch the weekly trend of prices, it has been decided to use a traditional financial indicator in a new, unusual way. The selected indicator is the so called ***Donchian Channel***, in honor of his creator Richard Donchian who developed this indicator in the mid 20-th century to try to identify price trends. This indicator consists of three lines, called *Upper Channel (UC)*, *Center Channel (CC)* and *Lower Channel (LC)* each of which wants a number of periods N as input in order to plot a line in the selected period in correspondence of the highest high for the UC, the lowest low for the LC and the lowest low plus the highest high divided by two for the CC.

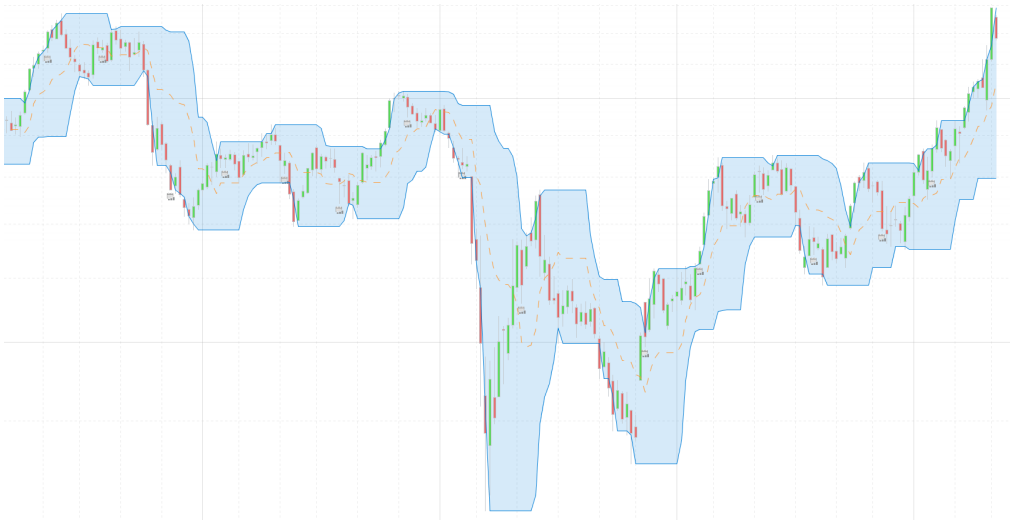


Figure 2.1: Example of Donchian Channal with 10 periods over *Valero Energy Corp*

However, not all the three lines are necessary for detecting a trend and therefore only the UC will be used even if it presents a problem: it is shifted forward in time by the set number of periods (10 in the example of Figure 2.1).

So, thanks to the *talib* python package (in which there are almost all the financial technical indicators) it is created a new column in the dataset in which there are upper-Donchian-channel's values shifted back in time by the optimal number of periods that, after many attempts, have been found to be 5 (the financial week).

Afterwards, from the new column is created another new column, containing the daily variations of the *Upper Donchian Channel* following the idea that, in case of lateral phase the higher high contains all the candles in the selected period, while in a upward trend the higher high is above the previous UC's value making the line upward sloping (and viceversa for the downward price movement), like shown in the following plot.

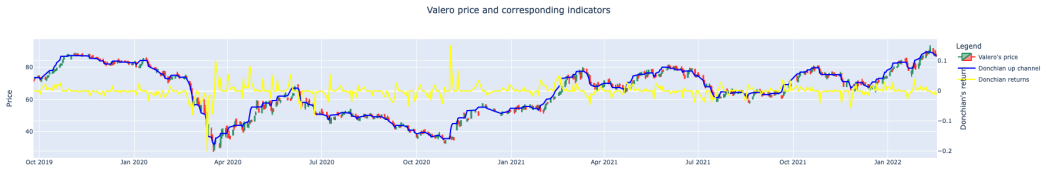


Figure 2.2: Upper Channel and corresponding changes over *Valero Energy Corp*

As it's possible to see from Figure 2.2 the UC shifted back in time perfectly matches the price movement while remaining sufficiently rigid to best capture short-to-medium term trends. Finally, exploiting the signs of the returns of the UC (the yellow line), it's possible to assign to each day a label describing the kind of movement made by the price.
Therefore, the resulting labels from the entire time series of *Valero Energy Corp* (taken as explanatory example) is the following

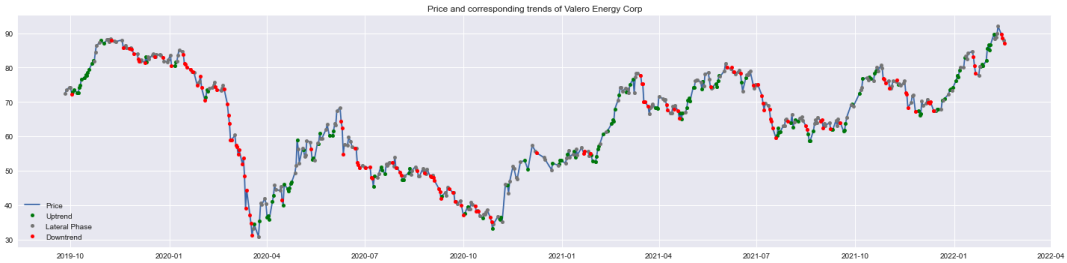


Figure 2.3: Trends detected over *Valero Energy Corp*

Looking at the result obtained in Figure 2.3 it's possible to conclude that the trends are captured and classified in a correct way and therefore the dataset is now ready for being used in a supervised learning problem, exactly like the one we are about to perform. However, our goal is to create predictive models and, therefore, the data we are really interested in is not the label at time *t*, but rather the label at time *t+1*. For this reason, a new column is created, called *Target*: this column is nothing more than

the result of moving back one period of the column containing the labels. In this way, at each time instant we will have all the data corresponding to that day (which will make up the set of explanatory variables) and the label data of the next day (which will be our response variable, e.i. the output). Thus, the model will take as input the set of explanatory variables and with them try to predict tomorrow's label, check its result and, in case of discrepancy, adjust the parameters.

2.2.2 Addition of Financial Technical Indicators

Now that the dataset contains both alternative data labels at time t and labels at time $t+1$, it's time to add some traditional informations for helping the models get the best understanding possible of the market and its corresponding movements. For this purpose, the following financial technical indicators are chosen:

- The *Simple Moving Average* with 7 days, called from here on **SMA(7)**
- A second *Simple Moving Average* with 80 days, called from here on **SMA(80)**
- A third *Simple Moving Average* with 160 days, called from here on **SMA(160)**
- The **slope of each SMA** previously described
- The *Average True Range* for a short-term period, that is 7 days, called from here on **ATR(7)**, and **its slope**
- The *Relative Strength Index* for a short-term period, that is 7 days, called from here on **RSI(7)**, and **its slope**

The *Simple Moving Average*, as the name says, calculate the simple mean of the close prices in the selected period. It is very useful for understanding the direction in which the prices are moving in different timeframe and, in fact, here are used the SMA(7) for assessing the short-term price direction, the SMA(80) for the medium-term price direction and the SMA(160) for the long-run. Clearly, to detect these directions, is much more useful the slope of each SMA and that is why they have been calculated and inserted into the dataset.

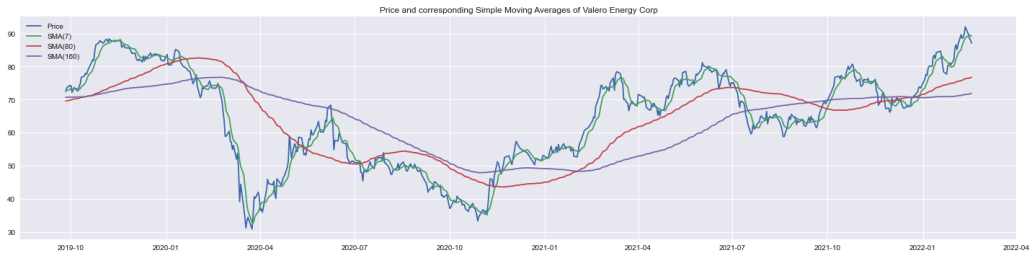


Figure 2.4: Simple Moving Averages over *Valero Energy Corp*

The *Average True Range* is a market volatility indicator whose formula is the following:

$$TR = max[(H - L), abs(H - C_p), abs(L - C_p)] \qquad (2.21)$$

$$ART = \frac{1}{n}\sum_{i=1}^{n} TR_i \qquad (2.22)$$

where C_p is the previous close price, TR_i is a particular true range, n is the selected period of time. This indicator gives to the model the information about the volatility of the stock period by period, which could be useful in a trend detection perspective.

The *Relative Strength Index*, instead, is a momentum indicator used to measure the magnitude of price changes. The RSI is an oscillator with

limits at 0 and 100 and also with overbought and oversold threshold, placed at 70 and 30 respectively. It is calculated in two steps:

$$RSI_{step1} = 100 - \left[\frac{100}{1 + \frac{AverageGain}{AverageLoss}}\right] \tag{2.23}$$

$$RSI_{step2} = 100 - \left[\frac{100}{1 + \frac{(PreviousAverageGainx13) + CurrentGain}{(PreviousAverageLossx13) + CurrentLoss}}\right]$$

where the *average gain* and *average loss* are the percentage of gain/loss during the look-back period. This indicator is widely used for detecting the trends and when they could expire (overbought region) or start (oversold region).

In the end, the dataset looks like in the following Figure

figi	date	adj_close	adj_high	adj_low	adj_open	company_name	sentiment	popularity	pct_change	popularity_variation	sentiment_variation	...	SMA(7)	slope_SMA(7)	SMA(80)	slope_SMA(80)	SMA(160)	slope_SMA(160)	ATR(7)
BBG000BBGGQ1	2019-09-26	72.423116	72.771640	71.151007	72.536386	Valero Energy Corp	0.158813	1.488379	-0.006455	-0.129674	-1.656716	...	72.694467	-0.061987	69.523016	0.132943	70.661390	0.005147	1.892250
	2019-09-27	73.573243	74.113454	72.754213	73.015606	Valero Energy Corp	0.757611	423.131591	0.015881	283.290222	3.770461	...	72.799024	-0.036595	69.654824	0.129320	70.677899	0.007250	1.863405
	2019-09-30	74.270289	74.932483	73.494825	73.773644	Valero Energy Corp	0.231537	1.764838	0.009474	-0.995829	-0.694385	...	73.009382	0.060618	69.781018	0.124835	70.702978	0.013704	1.802584
	2019-10-01	74.252863	74.915057	73.581956	74.618812	Valero Energy Corp	0.710236	108.126022	-0.000235	60.266835	2.067477	...	73.304382	0.123850	69.904194	0.123992	70.728535	0.019694	1.735515
	2019-10-02	72.144298	73.538391	71.682505	73.538391	Valero Energy Corp	0.757255	204.492103	-0.028397	0.891239	0.066203	...	73.137589	0.139160	69.988841	0.118102	70.734894	0.019764	1.854778
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
BBG00GBV88T6	2022-02-11	17.020000	17.100000	14.900000	14.980000	Peabody Energy Corp	0.180586	681.310086	0.139224	-0.561813	0.109168	...	13.802857	0.438429	11.431750	-0.030087	12.442563	0.041163	1.365414
	2022-02-14	18.020000	18.200000	16.160000	16.900000	Peabody Energy Corp	-0.114329	908.062238	0.058754	0.332818	-1.633103	...	14.631429	0.577571	11.475875	-0.010600	12.511500	0.050287	1.461784
	2022-02-15	17.210000	17.745000	16.690000	17.620000	Peabody Energy Corp	-0.050760	283.608560	-0.044950	-0.687677	-0.556017	...	15.227143	0.657286	11.515875	0.013638	12.575813	0.059000	1.442958
	2022-02-16	16.350000	17.930000	16.220000	17.500000	Peabody Energy Corp	-0.008858	176.459520	-0.049971	-0.377806	-0.825500	...	15.754286	0.669000	11.543375	0.033537	12.628438	0.062063	1.481106
	2022-02-17	17.430000	17.810000	16.315000	16.420000	Peabody Energy Corp	-0.057978	102.863268	0.066055	-0.417072	5.545492	...	16.365714	0.624857	11.586375	0.037675	12.686187	0.060419	1.483091

Figure 2.5: Complete dataset

and it is now ready to be analyzed through the selected machine learning models: *XGBoost* and *LightGBM*.

2.3 Application of different Machine Learning models

In this section, are shown in details only the two models that obtained consistent results even if other models have been implemented, unfortunately with unsatisfactory results. In particular, it has been trained and tested *Support Vector Machines*, *Simple Neural Networks and Long-Short-Term-Memory Neural Network* (LSTMNN) but all these models yielded to constant predictions (e.i. flat prediction curve) and, in the case of the neural networks, also to a flat learning curve, probably due to the kind and length of input data.

The only models that generated interpretable results were the *XGBoost* and *LightGBM* which, as said in the theoretical chapter, come from the same macro-family: tree-based algorithm. One possible explanation is that decision trees are particularly suited and close to human reasoning when it comes to financial data and trading strategies. In fact, a human being when trying to assess, for example, price direction using the previously selected traditional indicators makes reasoning such as "if the SMA(7) and RSI(7) are increasing probably an upward trend is evolving, so let's check the ATR(7) to validate this theory" and so on, which is very similar reasoning to that performed by decision trees. This is why these algorithms were chosen in the first place and also probably because they are the only ones with consistent results.

2.3.1 XGBoost and LightGBM models

To implement and run the XGBoost model, must be installed the *xgboost* python package from which is imported the *XGBClassifier* function, that

has a lot of tunable parameters. In fact, the first thing to do is to tune the parameter and in order to do that it is imported the *RandomizedSearchCV* from *sklearn* to end up with a good model to start with, tuning the following parameters:

- *n_estimators*: (50, 100, 200, 300, 500, 1000, 2000)
- *learning_rate*: (0.05, 0.10, 0.15, 0.2, 0.25, 0.3)
- *objective*: (multi:softprob, 'multi:softmax)

Since it is a randomized search, at each iteration the result changes and therefore are selected only those values that appear most of the times in the various iterations to then conclude the tuning manually, testing the various combinations. Hence, the final best XGBoost model results to be the following one:

XGBClassifier("objective"='multi:softprob', "n_estimators"=100, "learning_rate"=0.15, "random state"=0)[1]

For what concerns the LightGBM model, the reasoning is pretty much the same as the one just made for XGBoost: it's possible to implement it by installing the package *lightgbm* and then importing from it the *LGBMClassifier* function. First, it is performed an auto parameter tuning through, again, the *RandomizedSearchCV* function with 5 iterations and 10 cross validations over the following parameters and relative values:

- *n_estimators*: (50, 100, 200, 300, 500, 1000, 2000)
- *learning_rate*: (0.05, 0.10, 0.15, 0.2, 0.25, 0.3)

[1]The complete XGB algorithm code is available at Figure 3.4 in the Appendix, at the bottom of the book

- *max_depth*: (5, 10, 15, 20, 50, 100)

Finally, the ultimate model is again obtained through a manual tuning starting from the ones the appeared most in the different RandomizedSearch iterations. Hence, the final best LightGBM model results to be the following one:

LGBMClassifier("objective"='softmax', "n_estimators"=300, "learning_rate"=0.3, "max_depth"=15, "random state"=123)[2]

Now that the best models are defined, it's crucial to put the attention on the kind of data that will be the input of these models. All the most important features on which are constructed all the others (e.i. adjusted close prices, popularity and sentiment) are time series: this means that the order of the data is fundamental. Therefore, it's not possible to split them randomly into training and test set because, in this way, you would lose the original order and consequently also the very meaning of the data themselves. So, for each title is taken the first 90% of the series as training set and the remaining 10% is used as test set: in this way the intrinsic meaning of the series is maintained for both sets.
However, both XGBoost and LightGBM, when take an input, return the predictions for the whole test set in once. This is clearly the common behavior of almost all machine learning models, so it should all make sense, but let's dwell on it a bit more. Suppose to have three years of history as a baseline, this would mean having about 756 days of history (252x3) resulting in a training set of 680 data points (90% of the dataset) and a test set with 76 days (10%). So, applying the above models would result in the

[2]The complete LGBM algorithm code is available at Figure 3.5 in the Appendix, at the bottom of the book

prediction for 76 consecutive days based on the previous 680: therefore, it is not difficult to assert that the very first predictions will have extremely higher accuracy than the last ones, since the further one goes into the future the more difficult it is to make reliable predictions. For this reason, both models are implemented with the so called ***rolling*** technique. This procedure consists of starting with the entire dataset, divide it as previously specified into training and test sets (680 and 76 data points in our example), define a number of days in the future of which we want the predictions (just one day, e.i. the prediction of tomorrow) and then iteratively perform the following steps:

1. Use the entire training set for train the model
2. Use the trained model to make predictions for the entire test set
3. Extract and save the first prediction (e.i. the prediction of tomorrow)
4. Add the first data point of the test set to the training set
5. Go back to step 1 and continue until there are no more data points in the test set

Therefore, continuing with our example, in the first phase there are 680 training points through which 76 predictions are made, and from these predictions only the first one is extracted and saved in an array. Then, the first data point of the test set is extracted and added (in the last position) to the training set: now there are 681 training points and 75 test points. Then, the 681 training points are used to re-train the model from scratch to then make the 75 predictions from which, again, only the first one is extracted and saved, in second position, in the same array of predictions as before. By performing this procedure until the test set is exhausted, we

obtain an array of the same length of the original test set containing the predictions day by day with which the accuracy of the predictions can be calculated through the *accuracy score* function of the *sklearn* package, that evaluates how many of the predicted labels are correct out of the total.

2.4 Results and Model Comparison

Summarizing what has been said so far, there is a dataset (Figure 2.5) composed by 23 columns and two machine learning models embedded within algorithms capable of making day-by-day predictions. However, not all features have the same relevance and importance within a model: some may be irrelevant and may not add valuable information to how much there already is. Therefore, it's important to perform also another kind of tuning, that is also referred to as *feature selection*. In fact, for improving the capability of XGB or LGBM to make predictions and also to speed up their running time, it is crucial to give them as input only those attributes that actually give them gain, excluding the others, to prevent the models from "wasting time" evaluating unnecessary and useless features.
Therefore, three attributes are excluded a priori because they contain redundant informations: in fact, the *adjusted high*, *low* and *open* prices have been used for constructing other, more useful, indicators like the ATR and RSI and, so, these metrics already contain those informations in an implicit form. Moreover, also the columns containing the values of *Donchian Channel* and *Donchian's change* are excluded a priori, because they have been used for creating the labels and, therefore, if models had access to such informations they would incur a bias that would drive the results upward, making them inconsistent.
Hence, the remaining attributes to evaluate are the following: *adjusted close*

price, sentiment and its variations, popularity and its variations, SMA(7) and its slope, SMA(80) and its slope, SMA(160) and its slope, ART(7) and its slope, RSI(7) and its slope, Labels. Starting from these 16 features, there are multiple interesting combinations of inputs for both XGB and LGBM summarized below:

- **Model 1**: model with all the inputs; *closing price, sentiment and its variations, popularity and its variations, SMA(7) and its slope, SMA(80) and its slope, SMA(160) and its slope, ART(7) and its slope, RSI(7) and its slope,* $Labels_t$.

- **Model 2**: model with all the inputs except for the point-value of the simple moving averages (but maintaining their slopes); *closing price, sentiment and its variations, popularity and its variations, SMA(7)'s slope, SMA(80's slope, SMA(160)'s slope, ART(7) and its slope, RSI(7) and its slope,* $Labels_t$.

- **Model 3**: model with all the inputs except for the point-value of the simple moving averages (but maintaining their slopes) and the variations of sentiment and popularity; *closing price, sentiment, popularity, SMA(7)'s slope, SMA(80's slope, SMA(160)'s slope, ART(7) and its slope, RSI(7) and its slope,* $Labels_t$.

- **Model 4**: model with all the inputs except for the variations of sentiment/popularity and $Labels_t$; *closing price, sentiment, popularity, SMA(7) and its slope, SMA(80) and its slope, SMA(160) and its slope, ART(7) and its slope, RSI(7) and its slope.*

- **Model 5**: model with all the inputs except for any value related to the FinScience metrics; *closing price, SMA(7) and its slope, SMA(80)*

and its slope, SMA(160) and its slope, ART(7) and its slope, RSI(7) and its slope, Labels$_t$.

Company_names	XGBoost model 1	XGBoost model 2	XGBoost model 3	XGBoost model 4	XGBoost model 5
Valero Energy Corp	58.930000	58.930000	57.140000	60.710000	60.710000
Shoe Carnival Inc	53.490000	58.140000	51.160000	62.790000	67.440000
Southern Co	55.000000	60.000000	60.000000	60.000000	56.670000
Lattice Semiconductor Corp	53.700000	50.000000	59.260000	50.000000	55.560000
Wingstop Inc	61.020000	66.100000	59.320000	61.020000	59.320000
Peabody Energy Corp	55.000000	53.330000	55.000000	61.670000	60.000000
Average Accuracy	56.190000	57.750000	56.980000	59.370000	59.950000

Figure 2.1: Results of the XGB model with different input combinations

Company_names	LGBM model 1	LGBM model 2	LGBM model 3	LGBM model 4	LGBM model 5
Valero Energy Corp	53.570000	53.570000	55.360000	62.500000	58.930000
Shoe Carnival Inc	60.470000	53.490000	58.140000	60.470000	58.140000
Southern Co	58.330000	61.670000	60.000000	58.330000	58.330000
Lattice Semiconductor Corp	51.850000	51.850000	57.410000	59.260000	55.560000
Wingstop Inc	57.630000	61.020000	55.930000	59.320000	57.630000
Peabody Energy Corp	60.000000	58.330000	60.000000	61.670000	65.000000
Average Accuracy	56.980000	56.660000	57.810000	60.260000	58.930000

Figure 2.2: Results of the LGBM model with different input combinations

Analyzing first Figure 2.1 it's possible to observe that, as expected, the XGBoost model with all inputs is the one with the lowest average accuracy validating once again the thesis that not all attributes are necessary. In fact, removing features like SMAs' point values (but maintaining the much more useful slopes of them) and percentage changes in sentiment and popularity, the model's average accuracy improves of almost 1%. However, by keeping out only one of the two categories just mentioned, things get better, because using all the inputs expect for SMAs' point values the XGBoost model gets an average accuracy of 57.75%, while excluding only the variations of

sentiment and popularity the XGB gets 59.37%, improving a lot (almost 1.6%) the performance. However, unfortunately, the right-most column contains the results for the XGB model without the FinScience metrics (neither the value point nor the percentage change) and it shows a small increase in the average accuracy in the prediction of the tomorrow's price trend with respect to the previous case.
Instead, for what concerns the Figure 2.2, the things seems to be slightly better than before. In fact, the worst average accuracy is reached when the point values of the SMAs are not included (56.66%) and the second worst result is 56.98%, obtained with all the inputs. Reducing at minimum the the inputs, instead, the result is an average accuracy of 57.81% which is actually good, but not better that the best XGB's result of 59.95% when the FinScience metrics are not included. Moreover, when to the LightGBM model are given all the inputs except for sentiment, popularity and their variations it, too, improves his performance by getting a value really similar to that of XGBoost, which is 58.93%. However, looking at the *LGBM model 4*, which corresponds to the model's performance when it takes all the inputs except only for the variations of sentiment and popularity and the labels at time t, it gets a very good 60.26% of average accuracy, improving the previous best result and breaking through the ceiling of 60% of accuracy. That's a great result since the the LightGBM model is the most efficient, speed and scalable between the two.

Moreover, neither model suffers from overfitting, as is easy to see from the following tables.

Company_names	Training Set Score	Test Set Score
Valero Energy Corp	0.8395	0.8036
Shoe Carnival Inc	0.9206	0.9667
Southern Co	0.8180	0.8667
Lattice Semiconductor Corp	0.8384	0.9815
Wingstop Inc	0.8305	0.8814
Peabody Energy Corp	0.8096	0.7167
Average Accuracy	0.8400	0.8700

Figure 2.3: Training and test accuracy for the selected XGBoost model

Company_names	Training Set Score	Test Set Score
Valero Energy Corp	1.0	0.9821
Shoe Carnival Inc	1.0	1.0000
Southern Co	1.0	1.0000
Lattice Semiconductor Corp	1.0	1.0000
Wingstop Inc	1.0	1.0000
Peabody Energy Corp	1.0	0.9833
Average Accuracy	1.0	0.9900

Figure 2.4: Training and test accuracy for the selected LightGBM model

By delving into the details, it is possible to analyze a random performance of each algorithm to understand the accuracy visually and find out how much each feature affects the learning capabilities of the models. In particular, in the following figures, we can see the behavior od both models over *Wingstop Inc.*

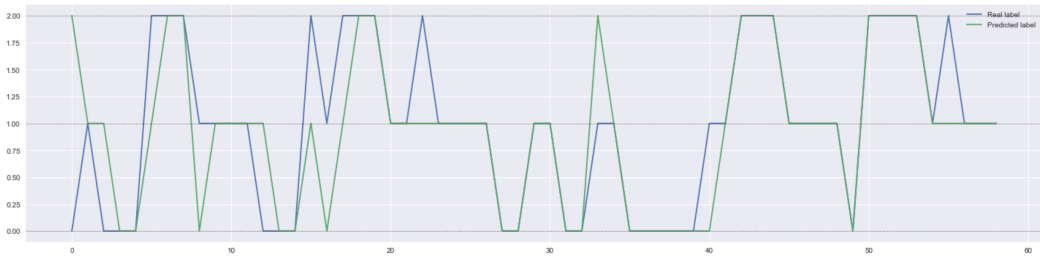


Figure 2.5: Predictions of XGB over *Wingstop Inc*

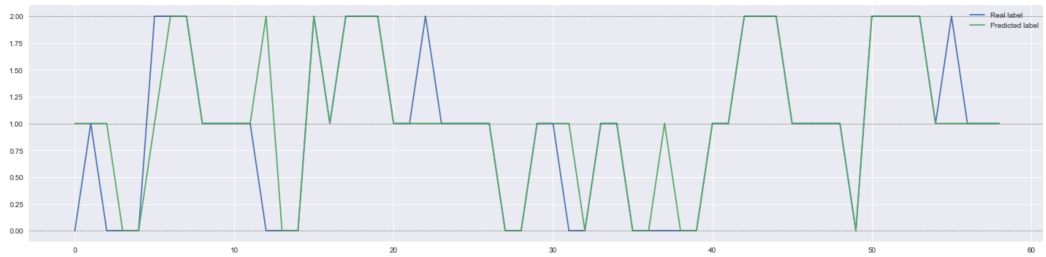


Figure 2.6: Predictions of LGBM over *Wingstop Inc*

Looking at the two figures, one detail immediately stands out: it seems that both the XGBoost and the LightGBM performed the same but, looking deeper into the details of both plots, it's possible to observe some important differences. In fact, in the period between 13 and 20, the two models behave differently: the LightGBM's predictions overlap perfectly with the actual labels, whereas the XGBoost basically underestimates the motion that actually happened. In addition, even the latest predictions of the LightGBM, i.e., those from about period 38 onward, match reality perfectly, while those of the XGB do not. On the other hand, with regard to gross errors, i.e., when a model predicts the exact opposite of what actually happens, over the entire dataset it was noted that the XGBoost tends to make more of them than the LightGBM.

Now, it would be interesting to understand whether and, eventually, how much the FinScience metrics affected the ability of the models to generate such accurate predictions. Fortunately, a function called *plot importance* of the *xgboost* package (but also applicable to the *lightgbm* package) comes to our aid: it returns the F-score and therefore the importance of each single input feature, ranking them from the most important to the less one. For reasons of space, "importance plots" related only to the sample of six stocks

considered so far are shown here to get a general idea of the behavior of popularity and sentiment:

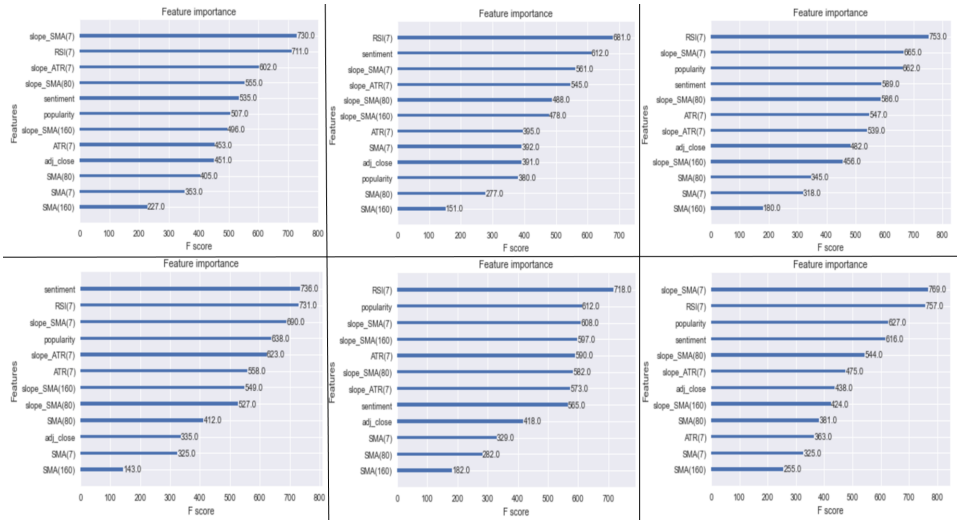


Figure 2.7: Feature importance in XGBoost model

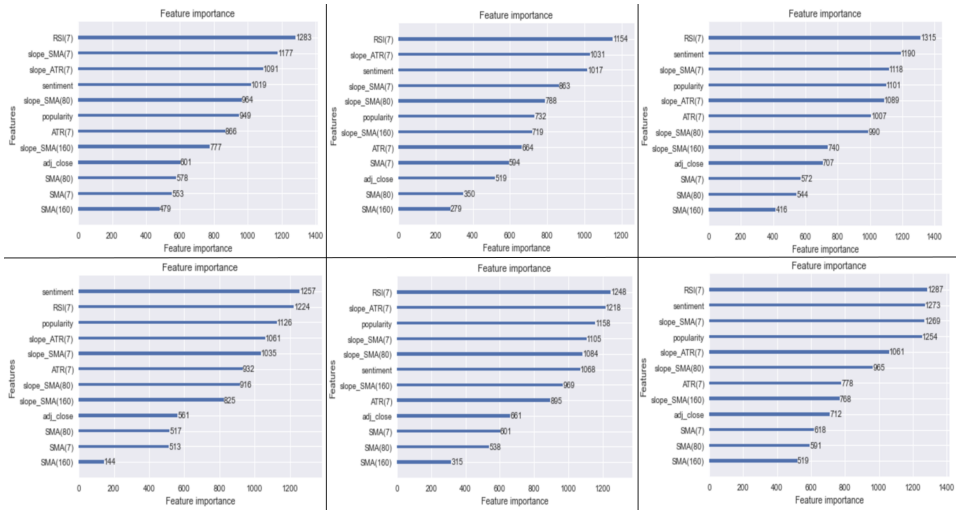


Figure 2.8: Feature importance in LightGBM model

As usual, the order of the stocks shown in both figures is the following, from up-left to down-right: *Valero Energy Corp*, *Shoe Carnival Inc*,

Southern Co, *Lattice Semiconductor Corp*, *Wingstop Inc*, *Peabody Energy Corp*. Focusing first to Figure 2.7, in the Valero's plot (most up-left one) the FinScience metrics are in 5th and 6th position, which is not a particularly exciting result, while the most importante features are the slope of moving average with 7 periods and the RSI. However, things get better in the second plot (*Shoe Carnival Inc*), where the sentiment is the second-most important feature after only the RSI. In the third and fourth plots the FinScience metrics have a huge impact on the XGB model placing third and fourth respectively for *Southern Co*, and first and fourth then. Finally, in the plot of *Wingstop Inc* the popularity is in second position for importance while the sentiment is only eighth and in the last chart there are popularity and sentiment at, respectively, the third and the fourth position.

Looking, instead, at Figure 2.8 relative to the feature importance for the LightGBM model it's possible to observe that again, FinScience metrics have a major impact in the model, always placing in the top half of the ranking. In particular, the "worst" result is for *Valero Energy Corp* (first chart) where there is the sentiment at the 4th position and the popularity at the 6th one. For all the other plots, instead, at least one of the two alternative metrics is always among the first three most important attributes, namely for *Southern Co* (sentiment 2nd), *Wingstop Inc* (popularity 3rd), *Peabody Energy Corp* (sentiment 2nd), getting the best result in the fourth plot, corresponding to *Lattice Semiconductor Corp*, where the sentiment is 1st and the popularity is 3rd.

Therefore, summarizing all the informations discussed so far, both the XGBoost model and the LightGBM, have confirmed that it is possible to

make predictions one day ahead into the future with an average accuracy of 60 percent, and that these predictions are strongly influenced by two alternative data-based metrics such as *sentiment* and *popularity*. This can be called an excellent result as well as the one hoped for from the beginning of the project itself.

The last interesting thing to analyze, are the tree built by the two algorithms:

Figure 2.9: XGBoost tree for the stock with the best accuracy

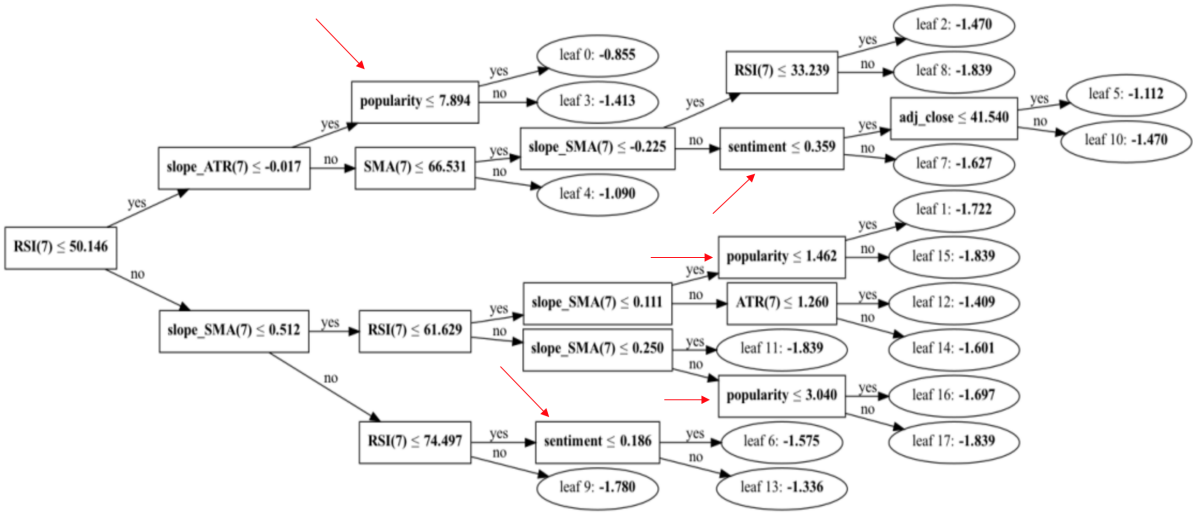


Figure 2.10: LightGBM tree for the stock with the best accuracy

As it is possible to observe in both the above figures, the trees have the roots based on the information carried by the RSI(7) which, also according to Figure 2.7 and 2.8, is probably the most important feature among all. Then, the first leaf-level of the trees seems to mainly depend on the value

of the SMA(7)'s slope since it appears in both models for a node (and also, for the other node, on SMA(80) for the XGB model and ATR(7)'s slope for the LGBM). Moreover, from the second level on, popularity and sentiment begin to appear in the trees, confirming once again their importance in these algorithms.

However, the results obtained until now may be due to the sample being particularly suitable for analysis and may not be reflected in the entire dataset. That is why, through the use of a virtual machine (VM, powered by Google Platform) much more performant than a normal PC, it was possible to run both algorithms on the real dataset, excluding only those stocks with a history of less than 500 days. The average accuracy is actually similar for both XGBoost and LightGBM with a 53.95% and 53.64% respectively, but the time needed for getting the results is significantly different: 6.25 hours for XGB (with only 100 estimators) and 3.75 hours for LGBM (with 300 estimators), which is almost the half of the time for getting the same accuracy. Moreover, it's also important to remember that the classifying categories were three, meaning that both algorithms performe much better than a random gambler, who should have an average accuracy around the 33%. Finally, also the variances are basically equal, with a value of 0.0091 for XGBoost and 0.0098 for LGBM, meaning that these two algorithms have more or less the same prediction stability.
Therefore, in the end, since almost all the statistics are very close to each other both the algorithms, the only characteristic that makes difference is the running time, which is why the LightGBM is the chosen model, as shown in the following Table.

	XGBoost model	*LightGBM model*
Tot. Titoli:	753	753
N.Titoli < 33%:	12	14
N.Titoli > 33%:	741	739
Avg. accuracy:	53.95%	53.64%
Running time:	6.25h	3.5h

Figure 2.11: Summary of the performances of XGB and LGBM models

In conclusion, in this chapter **has been created a new, reliable, alternative-data-strongly-dependent financial indicator** that is **able to predict a stock's price trend of tomorrow with a 53% accuracy** and that **becomes more and more reliable every day** as the history on which it can train increases.
Now it's time to see how this new indicator performs within a complete trading strategy.

Chapter 3

A Trading Strategy Based on Alternative Data

Now that a new financial technical indicator has been created, it's time to see if it is actually useful in a real world situation. Usually, in finance, indicators are not used alone but, rather, many of them are put together forming an *operating trading strategy*. Clearly, such strategies are not simply a group of indicators used for market analysis, but also include other equally important concepts such as money management, risk management, psychology management, asset allocation (when creating a portfolio), and so on. However, the purpose of this thesis is not to go into the details of such mechanisms or to construct and monitor a complete portfolio, but rather to see how the new *Trend Indicator* behaves when placed within a strategy, which is why in this chapter only the part related to the actual operation will be covered, neglecting the other aforementioned components.

Hence, the idea is to create a trading strategy that works well, composed by traditional indicators replaceable with the new FinScience trend indicator

in order to compare them and understand the reliability of such indicator.

3.1 Signal Screener Strategy

The *Signal Screener Strategy*[1] (**SiSS** from here on) is an algorithmic trading strategy designed to take advantage of the *Signal Screener*, which is a feature of the FinScience platform that allows you to view, monitor, and filter sentiment, popularity, and their changes for each security in the FinScience database. Moreover, it's also possible to set the so called *Alerts*, which are personalized notifications based on what the user wants to monitor: for example, a user could set an alert if the popularity of a particular stock (Datrix, for example) increases of 100% with respect to yesterday. Clearly, this tool is fully customizable, with the possibility of filtering stocks based on price changing, volume changing, market capitalization, working sector , sentiment changing, popularity changing and all these metrics have different timeframes among which the user can choose (daily, weekly, monthly, etc.).

The basic idea of this strategy is to try to exploit upward trends of prices with *long positions* to open and close at specific and objective conditions. Hence, *SiSS* is a multiday, long-term and rather risk-averse algorithmic strategy whose trades (LONG only) can last for weeks or even months: it is in fact designed for those who do not have time to check the markets every day and want to make investment trades with the odds in their favor.
The strategy, therefore, is composed by two main components: one to filter stocks to trade on and another to set the timing for opening and closing long positions.

[1]The complete strategy's code is available at Figure 3.6 in the Appendix, at the bottom of the book

The filtering component consists of three classical indicators that aim to find, among all existing stocks, the ideal candidates for the application of the strategy:

- ***Simple Moving Average at with a high number of periods***: is used to identify the underlying trend since the goal is to trade in favor of trend, so we will look only for those stocks with a bullish underlying trend to execute our long trades
- ***ADX Line***: it is the main line of the *Directional Movement Index* that, when increasing, indicates the presence of a trending phase of the market, without however telling us the direction of such trend
- ***Momentum***: is characterized by a line that oscillates above and below zero and it'sused to determine the strength or weakness of a stock's price (e.i. it measures the rate of the rise or fall of stock prices)

So, thanks to this filter, we can select only those titles that have a bullish underlying trend, which means they have been growing steadily for years (*SMA*) and also with a short-term upward trend of prices currently ongoing (due to the informations carried by the *ADX Line* and the *Momentum*). Clearly, each of these indicators takes as input a period of time chosen by the user, which can be of any length: the goal is to find for each of them the best possible period for our aim. Therefore, several periods have been tried, like 5 days, 7 days, two weeks (14 days) and a month (30 days) for the *ADX Line* and the *Momentum* obtaining the best results when the input period was equal to 7 days, while for the *SMA* it has been tried 200, 180 and 160, selecting the *SMA(160)* due to the length of the time series of the dataset. In fact, the history of each stock is, more or less, 3 years of data (e.i. 768

days) and, therefore, using a moving average with 200 or 180 periods would mean cutting off too much datapoints from the dataset and that's why 160 has been chosen as input period.

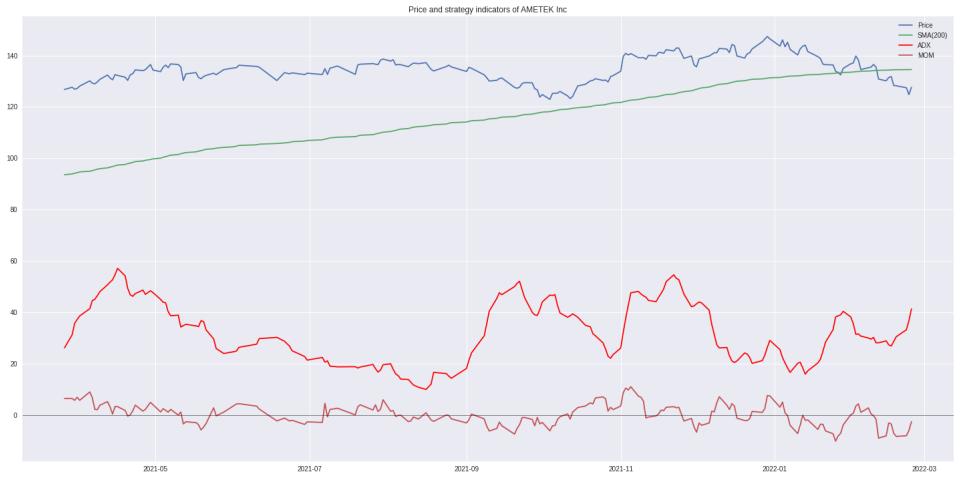


Figure 3.1: AMETEK Inc with the SiSS filtering indicators

Once selected the stocks through the filtering indicators just described, it's crucial to define a set of rules to follow to understand which is the proper time to buy and sell: here is when *sentiment* and *popularity* come in. The two FinScience metrics will answer to the following questions:

- **When to buy?** When the current value of popularity is greater than the moving average of the popularity itself of the last 7 days and the change in popularity from the previous available data is at least +100% and the current sentiment is greater than 0.05

- **When to sell?** When the current value of popularity is greater than the moving average of popularity itself over the last 7 days and the change in popularity from the previous available data is at least +100% and current sentiment is less than -0.05

Here should be rather simple to understand which is the advantage of the FinScience *Alert*: it allows a user to be always updated through timely notifications (via e-mail) when the aforementioned conditions occur on the selected stocks, in order to be always timely in opening and closing profitable positions. Therefore, the really interesting thing of this strategy is the fact that it completely relies on FinScience metrics to decide whether and when to open/close positions.

Obviously, even in this case the parameters are not randomly chosen, but rather selected after severals implementations. In fact, it has been chosen a period of 7 days for calculating the moving average of the popularity because we want to be aware when a stock is making a bigger splash than usual, and we also want to be timely in recognizing the headlines of the moment. Using two weeks or a month as input would make the selection less broad with the consequent risk missing some interesting stocks. Moreover, the condition of having an increase of the popularity of at least 100% with respect yesterday is because a less threshold would lead to false positive and a higher threshold would lower the pace of trades even more, which is already low. However, also threshold like 70% and 120% has been tried but with results lower than the 100% threshold. Finally, it was decided to use a sentiment greater than 0.05 to avoid including those borderline situations where a news item has a sentiment extremely close to zero, such as 0.0001 or -0.0001, and therefore labeled as positive or negative news, respectively. These situations could create bias because the buffer between being positively or negatively labeled is really small, and therefore the assignment made by the machine could be erroneous. Using, instead, a threshold of 0.05 and -0.05 we go for only that news that has an assignment of which we can be reasonably certain.

Before showing the results of the application of the *SiSS*, it is important to understand which is the procedure behind the algorithm applying it. First, the starting dataset is modified by removing all those stocks having less than 500 datapoints (days) for avoiding too young titles and then, since the data are stored in a Multi-Index DataFrame, the algorithm performs a double loop for accessing one single title at the time. Once having accessed a title, the *SiSS* :

1. stores the day before *Popularity's* value and sums the previous week's values of the *Momentum*.

2. checks if the *SMA(160)* and *ADX Line* slopes of the last 7 days are greater than zero and if the summed value of the *Momentum* is greater than 3.

3. if all these conditions are true, then it saves the current value and the mean of the last seven days of the *Popularity*, calculates the DPV change with respect to yesterday using the value stored in step 2 and saves the current value of *Sentiment*.

4. then, if the current DPV is greater than it's last week rolling mean **and** the DPV change is greater than 100% **and** the sentiment value is greater than 0.05, it modifies the current cell of the column *entry_long* putting a 1.

5. otherwise, if the current DPV is greater than it's last week rolling mean **and** the DPV change is greater than 100% **and** the sentiment value is lower than -0.05, it modifies the current cell of the column *close_long* putting a 1.

6. then, it goes outside the two main loops and create two new lists: one for containing the performance of the current stock and another one to store the name of the corresponding title.

7. then it goes again within the dataset and for each stock it creates a new list for saving the logarithm returns of each trade and also initializes a counter called *index_close_long* set to -1.

8. afterwards, for each stock, it skips the first 2 weeks for having only complete weekly rolling average values and then it seeks for a day in which there were the conditions to open a position (e.i. *entry_long* =1) if and only if the current counter is greater than *index_close_long*, which means that the eventual previous position has been already closed.

9. then it saves the buying price and initializes a dummy variable (*flag_close*) equal to *False*.

10. afterwards, it searches from the day in which it opened a position on looking for a day in which *close_long* is equal to 1, meaning that there are the conditions to close the position.

11. if it finds a match, it saves the closing price and update the value of *index_close_long* putting it equal to the counter corresponding to that closing day, and it also update *flag_close* turning it into *True*.

12. then, if *flag_close* = *True* (e.i. it opened and also closed a position) it sets to 1 and 0 in the *positions* column, respectively, the days in which it opened and closed the position; it computes the logarithm of the closing price over the opening price and it stores the log return into the *log_returns* array previously defined.

13. after having performed this loop, it sums up the values of the array *log_returns*, it calculates the single title performance ((np.exp(sum_all_log_ret) - 1)*100) and it stores this value along with the title's name in two distinct vectors.

14. finally, once outside all the loops, it creates a dataframe containing all the stock's names and their corresponding total returns, and therefore it returns them as output.

This is how the *SiSS* works, assuming that at each trade a user buys just one share at exactly the close price of that day and that he or she, when closing the position, sells that share at, again, exactly the closing price.

The performance of this strategy performed over the entire dataset, without removing too young stocks, is the following. Out of a total of 679 stocks on which the strategy is tested:

- **185** securities are discarded because they do not present, in their history, the necessary conditions to be able to operate
- **61** securities report a negative final return
- **433** securities achieve a positive return

which is a very good result. Moreover, performing the same strategy only on those titles having a long history (greater than 500 datapoints) the results are the following. Out of a total of 527 stocks on which the strategy is tested:

- **106** securities are discarded because they do not present, in their history, the necessary conditions to be able to operate

- **60** securities report a negative final return
- **361** securities achieve a positive return

which, again, is a very interesting result since we need to remember that the timing for buying and selling only depends on the FinScience metrics of *popularity* and *sentiment*.

However, despite the fact that this is a great result, it is not our ultimate goal. In fact, the *SiSS* will be the yardstick by which we will assess the operational reliability of the new *Trend Indicator*. Therefore, the new *Trend Indicator - Signal Screener Strategy* (***TI-SiSS*** from here on) is structured as following. There are again two main components, one for filter out the optimal stocks on which operate, and one for the operating time of each trade (long).
The filtering component consists of two (instead of three) indicators that aim to find, among all existing stocks, the ideal candidates for the application of the strategy:

- ***Simple Moving Average at 160 periods***: is used to identify the underlying trend since the goal is to trade in favor of trend, so we will look only for those stocks with a bullish underlying trend to execute our long trades
- ***Trend Indicator***: it is used to seek for upward price trends and, therefore, when it is greater or equal than 1, it summarizes all in one the informations previously provided by the *ADX Line* and *Momentum* when put together

The time component, however, remains exactly the same as in *SiSS*, making it rather easy to compare with *TI-SiSS*.

3.2 Comparison between SiSS and TI-SiSS

The real problem of simply run both strategies on the entire history of each title (e.i. backtesting) is that the *TI-SiSS* is based on the *Trend Indicator* which, in turn, needs a good training set to be reliable. Therefore, comparing the trades performed in the first half of history of each title would not be fair, since clearly the *Trend Indicator* would perform very bad (having a too small training set), as shown by Figure 3.1.

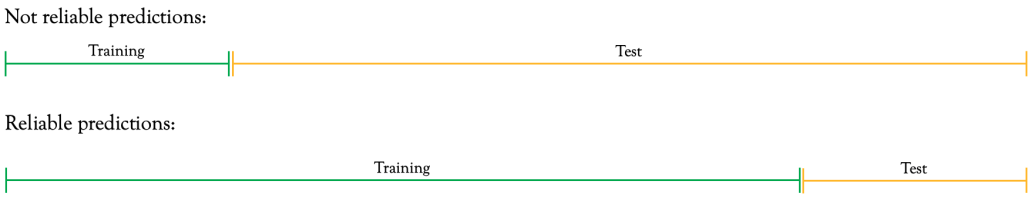


Figure 3.1: Explanation of why TI-SiSS is not reliable at the beginning of a stock's history

Therefore, it would be more logical to compare only the trades made in the last 20% of each stock's history for both *SiSS*[2] and *TI-SiSS*[3] so that the latter has sufficient history behind it to correctly run the *Trend Indicator*. In fact, as mentioned, "as time goes on," the trend indicator becomes increasingly reliable and robust as the data set on which it can base its predictions also increases. Clearly, looking at only 20 percent of a stock is equivalent to consider a part for the whole, which is not and does not claim to be a complete yardstick. However, it provides a good starting point for making some general assessments of the *TI-SiSS*. The results of

[2]The complete strategy's code is available at Figure 3.7 in the Appendix, at the bottom of the book

[3]The complete strategy's code is available at Figure 3.8 in the Appendix, at the bottom of the book

applying the strategies to the last 20% of each stock are:

- ***SiSS***: out of a total of 527 stocks, in **393** the strategy does not enter to trade, in **104** it makes a positive return, and in **30** it has a negative return. The highest return is 59.68% on *Devon Energy Corp*, while the worst return is -19.87% on *Alnylam Pharmaceuticals Inc.* The average return is **1.98**%, while the variance is **55.3**%.

- ***TI-SiSS***: out of a total of 527 stocks, in **218** the strategy does not enter to trade, in **207** it makes a positive return, and in **102** it has a negative return. The highest return is 56.17% on *Pioneer Natural Resources Co*, while the worst return is -24.97% on *Plug Power Inc.* The average return is **3.12**%, while the variance is **84.02**%.

Clearly, by using this comparison technique, we incur in a bias: it might happen that, in a stock, all trades in the first 80 percent bring profit while those made in the last 20 percent get a loss. In this case, the stock's performance would be classified as negative although, calculating all trades, the result would actually be positive. This, of course, can also happen in the opposite direction. However, comparing only the last 20 percent of each stock on such a large dataset should be sufficient to balance it toward the natural average, thus becoming a good estimator of reality.

Naturally, the number of stocks on which both strategies do not operate is higher, which is logical, because the probability that at least one trade is completed (opened and closed) in only the last 20 percent of a stock strongly decreases.
Therefore, looking at the results, the *TI-SiSS* turns out to be more risky since, on average, in the last 20% it operates over many more stocks (309

vs 134) making gains on 207 (versus 104 of the *SiSS*) and making losses on 102 (instead of 30 of *SiSS*). Nevertheless, the cost of being more risky is repaid by the fact that the medium gain is higher (3.12% vs 1.9%).

Moreover, from Figure 3.2 and Figure 3.3 below, it's possible to observe the different trades performed by the two strategies in a situation in which the *TI-SiSS* performs better than the *SiSS* (56% vs 47%).

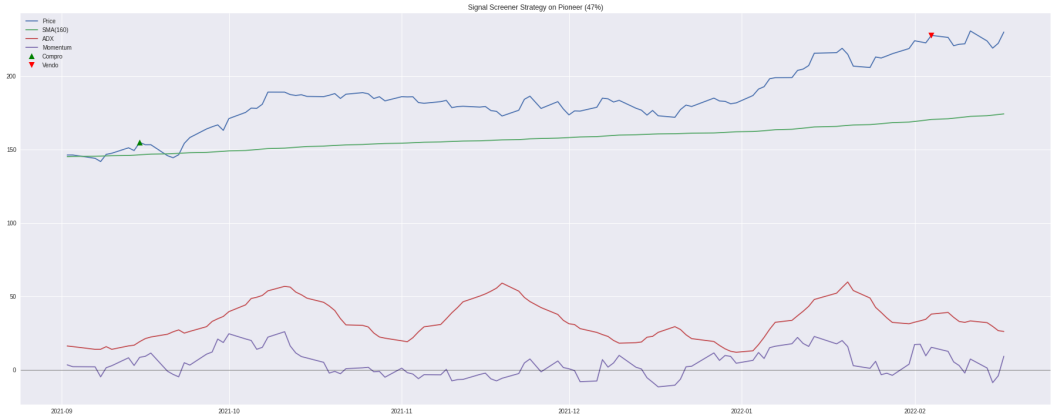


Figure 3.2: Trades performed by *SiSS* in the last 20% of *Pioneer*

Figure 3.3: Trades performed by *TI-SiSS* in the last 20% of *Pioneer*

Hence, it's possible to see that the *SiSS* makes just one trade, opening a long position around the half of September 2021 in correspondence of an increasing *ADX Line* and a positive *Momentum*, for then closing it around the first period of February 2022. The *TI-SiSS*, instead, thanks to the predictive power of the *Trend Indicator*, opens a long position a few days before the *SiSS*, anticipating the market. Moreover, the *TI-SiSS* concludes the trade during a sideways phase in prices, executes another that closes at a slight loss but then executes a third one that can fully ride the bullish movement that occurred between mid-December and early February.

Conclusions

In conclusion, throughout this project it has been created a new technical indicator for financial analysis, called ***Trend Indicator*** based on the FinScience metrics of *sentiment* and *popularity* that is able to make prediction of the tomorrow's price trend thanks to the use of a machine learning model called *LightGBM*. The average accuracy of such an indicator is more than 53%, which is far better than the 33% (the random gambler probabilities, due to the presence of three classification categories), making it reliable and gradually becoming more and more credible as time passes, due to the increase in the training set used to produce the predictions themselves.
Moreover, the reliability of the *Trend Indicator* has been also validated through an algorithmic trading strategy (e.i. ***TI-SiSS***) that is strongly FinScience-dependent (*popularity* and *sentiment* appear both in the indicator as input and in the strategy per se) for making trades with a 67% probability of profits (observing the results obtained in the last 20% of each stock). However, this strategy turned out to be risky and, therefore, it has been created another less-risky strategy, called ***SiSS*** which do not exploits the *Trend Indicator* but is still dependent on the FinScience metrics in the timing component. This last strategy has obtained very good results too, with 87,65% of probabilities of making gains with it.

Therefore, summarizing, there have been obtained not one, but three very important results:

1. ***Trend Indicator***: a new financial technical indicator able to make predictions with a 53% of accuracy based on the FinScience metrics.

2. ***Signal Screener Strategy***: an algorithmic strategy based on FinScience metrics and alerts, able to make profits in the 87,65% of cases.

3. ***Trend Indicator - Signal Screener Strategy***: an algorithmic strategy based on FinScience metrics in two components (as input in the *Trend Indicator* and as timing component for buying and selling) and FinScience alerts, able to make profits in the 67% of cases with potential higher gains than the *SiSS*.

In the end, the possible next stop would be:

- to figure out which stocks the strategies had higher/lesser returns on to investigate any commonalities (e.g., sector, capitalization, ..).

- Understand whether it is possible to go for ad hoc strategies based on step 1.

- Truly test the strategies with a virtual portfolio and monitor the history of returns.

Finally, the *Trend Indicator*, the *SiSS* and the *TI-SiSS* are available to anyone, simply installing the python package **FinScience** running the code *!pip3 install FinScinece* or downloading it directly from this link: `https://pypi.org/project/FinScience/`.

Appendix

```python
def rolling_XGB_prediction(title):
    name = title['company_name'].unique()
    title = title.dropna()
    test_size = int(len(title['adj_close'])*0.1)

    Y = title['Target']
    X = pd.DataFrame({'adj_close': title['adj_close'], 'sentiment': title['sentiment'], 'popularity': title['popularity'],
    'SMA(7)': title['SMA(7)'], 'slope_SMA(7)': title['slope_SMA(7)'], 'SMA(80)': title['SMA(80)'],
    'slope_SMA(80)': title['slope_SMA(80)'], 'SMA(160)': title['SMA(160)'], 'slope_SMA(160)': title['slope_SMA(160)'],
    'ATR(7)': title['ATR(7)'], 'slope_ATR(7)': title['slope_ATR(7)'], 'RSI(7)': title['RSI(7)'],
    'slope_RSI(7)': title['slope_RSI(7)']})
    #  'popularity_variation': title['popularity_variation'], 'sentiment_variation': title['sentiment_variation'], 'Labels': title['Labels']

    X_train = X[:(-test_size)]
    Y_train = Y[:(-test_size)]
    X_test = X[(-test_size):]
    Y_test = Y[(-test_size):]
    test_predictions = []

    for i in range(test_size):
        x_train = X[:(-test_size+i)]
        y_train = Y[:(-test_size+i)]
        x_test = X[(-test_size+i):]
        y_test = Y[(-test_size+i):]

        best_XGBoost_model.fit(x_train,y_train)
        pred_test = best_XGBoost_model.predict(x_test)
        test_predictions.append(pred_test[0])

    acc_test = accuracy_score(Y_test, test_predictions)

    print(f'**** Test Results of {name[0]} ****')
    print("Accuracy: {:.2%}".format(acc_test))

    # print('**** Checking overfitting ****')
    # print('Training set score: {:.4f}'.format(best_XGBoost_model.score(X_train, Y_train)))
    # print('Test set score: {:.4f}'.format(best_XGBoost_model.score(X_test, Y_test)))

    return best_XGBoost_model, test_predictions
```

Figure 3.4: Algorithm performing a rolling prediction through the XGBoost model

```python
def rolling_LGBM_prediction(title):
    name = title['company_name'].unique()
    title = title.dropna()
    test_size = int(len(title['adj_close'])*0.1)

    Y = title['Target']
    X = pd.DataFrame({'adj_close': title['adj_close'], 'sentiment': title['sentiment'], 'popularity': title['popularity'],
    'SMA(7)': title['SMA(7)'], 'slope_SMA(7)': title['slope_SMA(7)'], 'SMA(80)': title['SMA(80)'],
    'slope_SMA(80)': title['slope_SMA(80)'], 'SMA(160)': title['SMA(160)'], 'slope_SMA(160)': title['slope_SMA(160)'],
    'ATR(7)': title['ATR(7)'], 'slope_ATR(7)': title['slope_ATR(7)'], 'RSI(7)': title['RSI(7)'],
    'slope_RSI(7)': title['slope_RSI(7)']})
    # 'popularity_variation': title['popularity_variation'], 'sentiment_variation': title['sentiment_variation'], 'Label': title['Labels']

    X_train = X[:(-test_size)]
    Y_train = Y[:(-test_size)]
    X_test = X[(-test_size):]
    Y_test = Y[(-test_size):]
    test_predictions = []

    for i in range(test_size):
        x_train = X[:(-test_size+i)]
        y_train = Y[:(-test_size+i)]
        x_test = X[(-test_size+i):]

        best_LGBM_model.fit(x_train,y_train)
        pred_test = best_LGBM_model.predict(x_test)
        test_predictions.append(pred_test[0])

    acc_test = accuracy_score(Y_test, test_predictions)

    print(f'**** Test Results of {name[0]} ****')
    print("Accuracy: {:.2%}".format(acc_test))

    # print('**** Checking overfitting ****')
    # print('Training set score: {:.4f}'.format(best_LGBM_model.score(X_train, Y_train)))
    # print('Test set score: {:.4f}'.format(best_LGBM_model.score(X_test, Y_test)))

    return best_LGBM_model, test_predictions
```

Figure 3.5: Algorithm performing a rolling prediction through the LightGBM model

```python
def SiSS(dati):

    for figii in dati.index.get_level_values(0).unique():
        current_stock = dati.loc[(figii,slice(None))]

        for i, date in enumerate(current_stock.index):
            pop_previous_day = current_stock['popularity'][i-1]
            sum_momentum = current_stock['Momentum'][i-7:i].sum()

            if ((current_stock['slope_SMA(160)'][i]>0) and (current_stock['slope_ADX'][i]>0) and (sum_momentum > 3)):
                # Conditon 1: DPV (summed) of today is greater that the rolling average of the popularity itself
                dpv = current_stock['popularity'][i]
                sma7 = current_stock['pop_SMA(7)'][i]
                # Conditon 2: popularity variation greater than 100% with respect to yesterday
                pop_var = ((dpv - pop_previous_day) / abs(pop_previous_day))*100
                # Conditon 3: sentiment of today greater than 0.05
                sent = current_stock['sentiment'][i]

                # if all the above conditions are true, then I open a long position
                if ((dpv>sma7) and (pop_var>100) and (sent>0.05)):
                    current_stock['entry_long'][i] = 1
                if ((dpv>sma7) and (pop_var>100) and (sent<(-0.05))):
                    current_stock['close_long'][i] = 1

    single_performance = []
    name_company = []
    for figiii in dati.index.get_level_values(0).unique():
        current_stock2 = dati.loc[(figiii,slice(None))]
        log_returns = []
        close_long_index = -1
        for g, val in enumerate(current_stock2.index):
            if ((g > 14) and (current_stock2['entry_long'][g] == 1) and (g > close_long_index)):
                open_long_price = current_stock2['adj_close'][g]
                flag_close = False
                for j, elem in enumerate(current_stock2.index[(g+1):]):
                    if (current_stock2['close_long'][g+1+j] == 1):
                        close_long_index = g+1+j
                        close_long_price = current_stock2['adj_close'][close_long_index]
                        flag_close = True
                        break
                if flag_close:
                    current_stock2['positions'][g] = 1
                    current_stock2['positions'][g+1+j] = 0
                    single_trade_log_ret = np.log(close_long_price/open_long_price)
                    log_returns.append(single_trade_log_ret)

        sum_all_log_ret = sum(log_returns)
        performance = (np.exp(sum_all_log_ret) - 1)*100
        single_performance.append(performance)
        name_title = current_stock2.company_name.unique()[0]
        name_company.append(name_title)

    aggregate_data = {
        'Company Name': name_company,
        'Return (%)': single_performance
    }

    results = pd.DataFrame(aggregate_data)

    return(results)
```

Figure 3.6: Signal Screener Strategy (SiSS) applied over the entire dataset

```python
def SiSS_last20(dati):

    for figii in dati.index.get_level_values(0).unique():
        current_stock = dati.loc[(figii,slice(None))]

        for i, date in enumerate(current_stock.index):
            pop_previous_day = current_stock['popularity'][i-1]
            sum_momentum = current_stock['Momentum'][i-7:i].sum()

            if ((current_stock['slope_SMA(160)'][i]>0) and (current_stock['slope_ADX'][i]>0) and (sum_momentum > 3)):
                # Conditon 1: DPV (summed) of today is greater that the rolling average of the popularity itself
                dpv = current_stock['popularity'][i]
                sma7 = current_stock['pop_SMA(7)'][i]
                # Conditon 2: popularity variation greater than 100% with respect to yesterday
                pop_var = ((dpv - pop_previous_day) / abs(pop_previous_day))*100
                # Conditon 3: sentiment of today greater than 0.05
                sent = current_stock['sentiment'][i]

                # if all the above conditions are true, then I open a long position
                if ((dpv>sma7) and (pop_var>100) and (sent>0.05)):
                    current_stock['entry_long'][i] = 1
                if ((dpv>sma7) and (pop_var>100) and (sent<(-0.05))):
                    current_stock['close_long'][i] = 1

    single_performance = []
    name_company = []
    for figiii in dati.index.get_level_values(0).unique():
        current_stock2 = dati.loc[(figiii,slice(None))]
        log_returns = []
        close_long_index = -1
        size_to_skip = int(len(current_stock2['adj_close'])*0.8)
        for g, val in enumerate(current_stock2.index):
            if ((g >= size_to_skip) and (current_stock2['entry_long'][g] == 1) and (g > close_long_index)):
                open_long_price = current_stock2['adj_close'][g]
                flag_close = False
                for j, elem in enumerate(current_stock2.index[(g+1):]):
                    if (current_stock2['close_long'][g+1+j] == 1):
                        close_long_index = g+1+j
                        close_long_price = current_stock2['adj_close'][close_long_index]
                        flag_close = True
                        break
                if flag_close:
                    current_stock2['positions'][g] = 1
                    current_stock2['positions'][g+1+j] = 0
                    single_trade_log_ret = np.log(close_long_price/open_long_price)
                    log_returns.append(single_trade_log_ret)

        sum_all_log_ret = sum(log_returns)
        performance = (np.exp(sum_all_log_ret) - 1)*100
        single_performance.append(performance)
        name_df = current_stock2.company_name.unique()[0]
        name_company.append(name_df)

    aggregate_data = {
        'Company Name': name_company,
        'Return (%)': single_performance
    }

    results = pd.DataFrame(aggregate_data)

    return(results)
```

Figure 3.7: Signal Screener Strategy (SiSS) applied over the last 20% of each title

```python
def SiSS_with_indicator_last20(dati):

    for figii in dati.index.get_level_values(0).unique():
        current_stock = dati.loc[(figii,slice(None))]

        for i, date in enumerate(current_stock.index):
            pop_previous_day = current_stock['popularity'][i-1]

            if ((current_stock['slope_SMA(160)'][i]>0) and (current_stock['Trend_Predictions'][i] is not None) and (current_stock['Trend_Predictions'][i]>=1)):
                # Conditon 1: DPV (summed) of today is greater that the rolling average of the popularity itself
                dpv = current_stock['popularity'][i]
                sma7 = current_stock['pop_SMA(7)'][i]
                # Conditon 2: popularity variation greater than 100% with respect to yesterday
                pop_var = ((dpv - pop_previous_day) / abs(pop_previous_day))*100
                # Conditon 3: sentiment of today greater than 0.05
                sent = current_stock['sentiment'][i]

                # if all the above conditions are true, then I open a long position
                if ((dpv>sma7) and (pop_var>100) and (sent>0.05)):
                    current_stock['entry_long'][i] = 1
                if ((dpv>sma7) and (pop_var>100) and (sent<(-0.05))):
                    current_stock['close_long'][i] = 1

    single_performance = []
    name_company = []
    for figiii in dati.index.get_level_values(0).unique():
        current_stock2 = dati.loc[(figiii,slice(None))]
        log_returns = []
        close_long_index = -1
        size_to_skip = int(len(current_stock2['adj_close'])*0.8)
        for g, val in enumerate(current_stock2.index):
            if ((g >= size_to_skip) and (current_stock2['entry_long'][g] == 1) and (g > close_long_index)):
                open_long_price = current_stock2['adj_close'][g]
                flag_close = False
                for j, elem in enumerate(current_stock2.index[(g+1):]):
                    if (current_stock2['close_long'][g+1+j] == 1):
                        close_long_index = g+1+j                    (variable) close_long_index: int
                        close_long_price = current_stock2['adj_close'][close_long_index]
                        flag_close = True
                        break
                if flag_close:
                    current_stock2['positions'][g] = 1
                    current_stock2['positions'][g+1+j] = 0
                    single_trade_log_ret = np.log(close_long_price/open_long_price)
                    log_returns.append(single_trade_log_ret)

        sum_all_log_ret = sum(log_returns)
        performance = (np.exp(sum_all_log_ret) - 1)*100
        single_performance.append(performance)
        name_title = current_stock2.company_name.unique()[0]
        name_company.append(name_title)

    aggregate_data = {
        'Company Name': name_company,
        'Return (%)': single_performance
    }

    results = pd.DataFrame(aggregate_data)

    return(results)
```

Figure 3.8: Signal Screener Strategy (SiSS) with *Trend Indicator* applied over the last 20% of each title

Bibliography

[1] James D. Hamilton, *Time Series Analysis*, 1994.

[2] Philip Hans Franses & Dick Van Dijk, *Forecasting Stock Market Volatility Using (Non-Linear) Garch Models*, 1996.

[3] Anton Sorin Gabriel, *Evaluating the Forecasting Performance of GARCH Models*, 2012.

[4] Niklas Karlsson,*Forecasting accuracy for ARCH models and GARCH (1,1) family*, 2014.

[5] Kenneth P. Burnham & David R. Anderson, *Multimodel Inference: Understanding AIC and BIC in Model Selection*, 2004.

[6] Selva Prabhakaran, *Augmented Dickey Fuller Test (ADF Test)* , 2019.

[7] Peng Chen, Hongyong Yuan, Xueming Shu, *Forecasting Crime Using the ARIMA Model*, 2008.

[8] Md Yeasin, K.N. Singh, Achal Lama and Ranjit Kumar Paul *Modelling Volatility Influenced by Exogenous Factors using an Improved GARCH-X Model*, 2020.

[9] www.andreadd.it, *Classificazione dei dat*, Available at: https://www.andreadd.it/appunti/polimi/ingegneria/corsi/ing_biomedica

/Magistrale/BBB/modelli_analisi_dati/appunti/analisideidati
/Classificazione.pdf. (Accessed: 20 April 2022)

[10] xgboost.readthedocs.io, *XGBoost Documentation*, 2021. Available at: https://xgboost.readthedocs.io/en/stable/. (Accessed: 2 May 2022)

[11] www.youtube.com, *XGBoost Algorithm - Learn from its Author, Tong He*, 2015. Available at: https://www.youtube.com/watch?v=ufHo8vbk6g4&t=2420s. (Accessed: 3 May 2022)

[12] Ramraj S, Nishant Uzir, Sunil R, and Shatadeep Banerjee, *Experimenting XGBoost Algorithm for Prediction and Classification of Different Datasets*, 2016.

[13] lightgbm.readthedocs.io, *LightGBM's Documentation*, 2022. Available at: https://lightgbm.readthedocs.io/en/latest/. (Accessed: 4 May 2022)

[14] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, Tie-Yan Liu, *LightGBM: A Highly Efficient Gradient Boosting Decision Tree*, 2017. Available at: https://lightgbm.readthedocs.io/en/latest/. (Accessed: 4 May 2022)

[15] Martin J. Pring, *Analisi Tecnica dei Mercati Finanziari*, 1995.