TÉCNICO LISBOA

# Feature Engineering Automation for Time Series Analysis

**Hélio Jorge Benedito da Silva Domingos**

Thesis to obtain the Master of Science Degree in

## Information Systems and Computer Engineering

Supervisor: Prof. Cláudia Martins Antunes

### Examination Committee

Chairperson: Prof. Alberto Manuel Rodrigues da Silva
Supervisor: Prof. Cláudia Martins Antunes
Member of the Committee: Prof. Sara Alexandra Cordeiro Madeira

**January 2021**

Dedicated to my family.

For their endless love and support.

# Acknowledgments

I would like to thank my advisor, Prof. Claúdia Antunes, for her valuable support and help. Without her, this journey would not be the same.

I am also thankful to my friends, Daniel, Luís, Pedro, Mariana, for being kind and supporting.

Lastly, a word to my family for give me the opportunity to study at Instituto Superior Técnico and being always there when I needed.

This work was supported by national funds by Fundação para a Ciência e Tecnologia (FCT) through project VizBig PTDC/CCI-CIF/28939/2017.

# Resumo

Recentemente, dados de séries temporais têm sido os tipos de dados que mais crescem, uma vez que tanto pessoas como negócios medem normalmente os seus desempenhos ao longo de um período de tempo. Séries temporais são normalmente associadas à tarefa de previsão de dados. Para fazer previsões, um cientista de dados necessita de extrair características que ajudem a descrever o comportamento dos dados. Para ajudar os cientistas de dados, têm sido propostas ferramentas para automatizar o processo de ciência de dados, noutros campos de pesquisa, como por exemplo na classificação de dados tabulares, mas não aplicado a séries temporais. Neste trabalho, descrevemos as principais ferramentas de Automatização de Aprendizagem Máquina e propomos uma nova ferramenta que automatiza o processo de criação de um processo de análise de séries temporais. O foco está na extração de características descriptivas que se adaptem aos dados. Essas características são agregadas em conjuntos, avaliadas e selecionadas para o modelo. Com este trabalho, pretendemos desenvolver uma ferramenta que reduza o trabalho do cientista de dados durante o desenvolvimento de um processo de análise de séries. Desta forma o cientista pode concentrar-se na análise de resultados.

**Palavras-chave:** Automatização de Aprendizagem Máquina, Aprendizagem Máquina, Séries Temporais, Extração de Características, XGBoost.

## Abstract

In recent years, time series data have been one of the most growing types of data since people and business usually measure their performance over a period of time. Temporal data are usually associated with the task of forecast. To forecast, a data scientist needs to create features that help describe the behavior of data. To help data scientists, there have been proposed frameworks that automate the data science pipeline in other fields of research, for example for classification of tabular data, but not for time series. In this work, we describe the main Automate Machine Learning frameworks and propose a new framework that automates the process of creating a pipeline of analysis of time series. The focus is on creating descriptive features, that adapt to the data. Those features are aggregated in sets and they are evaluated and selected for the model. With this work, we aim to deliver a tool that will reduce data scientists' work developing pipelines of analysis, and help them concentrate on the analysis of results.



**Keywords:** AutoML, Machine Learning, Time Series, Feature Engineering, XGBoost.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

## 1.1 Motivation

*Data science* is a combination of fields from databases to data visualization passing through statistics, data mining, data analysis, and machine learning. In recent years, industry from technology to agriculture (and also organizations like European Union [17]) understood the importance of data, the importance of analyzing and extracting value from data in order to improve their business model, decision making and even products. Although most of the data science algorithms were invented some decades ago, they only started being used on a large scale in recent years due to big data.

The data science pipeline includes data collecting, cleaning, exploring and visualizing to understand their structure, creating models and interpreting results. The data scientist needs to know his target problem and be aware of the data limitations. Also, he needs to choose the best algorithm(s) to fit the data and choose the hyperparameters of the algorithms that maximizes accuracy. All these choices need to be well performed by the data scientist and each case is different from the previous one which needs study before applying any technique. This process can consume a lot of time and human resources until achieving good results.

As an example, the machine learning professionals could follow a pipeline like shown in Figure 1.1. In each step, he has different options to do the work: how to preprocess the data (e.g., check missing values – eliminate rows, calculate average, median, etc and replace – check for balanced data – oversample the minority class, undersample majority class, generate synthetic samples – feature selection – univariate selection, feature importance, correlation matrix); what model to use (e.g., k-nearest neighbors(kNN), neural networks(NN), decision tree(DT)); what hyperparameters to use (e.g. a

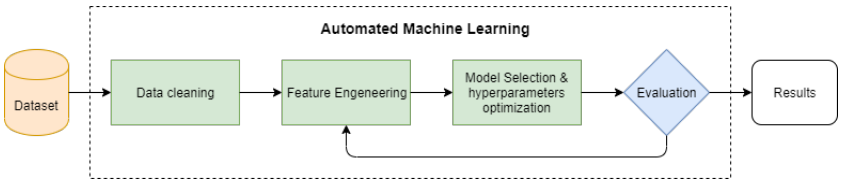



Figure 1.1: The data science pipeline with the tasks that AutoML aims to automate

big k or a small one (kNN)?, how many layers (NN)?, what should be the max-depth(DT)?).

Automated machine learning (AutoML) aims to develop tools that build machine learning models without human intervention. This includes automating each step of the machine learning pipeline that was previously presented. The goal is that users provide data and the AutoML system automatically determines the approach that performs best for their dataset[28].

Steps of the pipeline such as model selection or hyperparameter optimization were being studied by decades and they are well-defined and optimized. On the other hand, feature engineering needs more attention. There are frameworks that automate this step but with huge needs of computational and time resources.

## 1.2 Objectives

Since everyday it is produced more and more data, we need to process and extract knowledge from that data. To accomplish that we need machine learning models to help us extracting insights and hidden features that describes the behaviour of data. However, modeling data it is not easy as described before, so to help people that not have a data science degree or a deep knowledge about it, we aim to build a framework that automates the all data science pipeline as described before.

The main purpose of this work is to provide a framework that can automate feature engineering applied to time series data. Our proposal is based on the composition of a set of operators which pick the original data series and adds features, enriching the series.

Our framework will explore different models and different techniques of feature engineering. Those features will be used to forecast the series. To evaluate the forecast, we will compare the results of a neural network model without features against the results of a decision tree model based with feature engineering.

With this work, we expect to generate relevant features for each specific dataset instead of generate multiple possible feature combinations and test them all. Thus, the computational cost might get lower and we might get higher accuracy.

## 1.3 Thesis Outline

The remaining document is organized in the following way:

- **Chapter 2** introduces the time series topic gathering the basic concepts and definitions, presents and discusses related work focusing on the data science pipeline and how to automate each step;
- **Chapter 3** describes our solution;
- **Chapter 4** discusses the results of our Solution and the metrics used;
- **Chapter 5** provides major conclusions.

# Chapter 2

# Related Work

Due to the importance of this topic, between 2018 and 2019 were published five surveys explaining the state of the art in AutoML, one in 2018 [48] and four in 2019 [28] [49] [23] [45] .

Additionally, industry have been focusing on developing frameworks to substitute data scientists. Some frameworks are AUTO-Weka [44], AUTO-Sklearn [20], TPOT [38], Hyperopt-Sklearn [5]. Also, companies such as Google, H2O.ai, Tazi.ai, DataRobot, (Portuguese) Feedzai, are developing their own frameworks as their business.

In the following sections we will present the basic concepts of time series and then present a survey of the most relevant AutoML techniques proposed to date, categorized in accordance with each data science pipeline phase: data cleaning, feature engineering, model generation and hyperparameter optimization. Feature engineering is sub-divided was split into three sub-topics: feature selection, extraction, and construction, which will be discussed individually.

## 2.1 Data Cleaning

Data cleaning is an important step that depending on how is performed can create a big difference in model performance leading to different results. A clean dataset is the one that not contains missing values and their data are balanced by the different classifications. Data cleaning deals with missing values, detecting and removing errors and inconsistencies (outliers) [41]. Data quality problems can be from different scopes:

- illegal values inside the same attribute;
- values that violates dependencies between attributes;
- uniqueness violation in primary keys;
- broken links between entries;
- misspellings and abbreviations.

Additionally to this small list, there are a lot of other problems that need to be taken into account. The necessary knowledge to understand the relations between attributes and the different value possibilities are only provided by a human with strong domain knowledge.

One way to explore domain knowledge is look into the metadata from a database. Metadata, in relational databases, expresses the types of attributes, relations between entities, uniqueness of values and dependencies on the data. Metadata extracted from schemas is not enough to cover all the specifications of the data, and if the schema does not cover all the relations and constraints, metadata is useless. Besides metadata, first order logic representations can be very useful to identify noisy data in order to be either deleted or corrected.

Data cleaning is recognized as an important key to retrieve knowledge from data. Existing AutoML frameworks understood this importance and included some steps to deal with this problem. These steps usually deal with imputation of missing values, removing of outliers and scaling features to a normalized range. For example, the Data Science Machine [31] only cleans the data by removing the null values, convert categorical variables using one-hot encoding, and normalize features. AUTO-SKLEARN [20] includes categorical encoding, imputation, removing variables with low variance and scaling. In spite of these steps are not enough, the goal of most frameworks is to be domain-agnostic and give the best results whatever is the dataset.

## 2.2 Automated Model Selection

Model selection is the task of selecting a proper model from a set of candidate models and setting its hyperparameters in order to achieve a good learning performance [48]. There are many classification algorithms and for each one different hyperparameters are associated. 2.1 shows some examples of classifiers available in scikit-learn [39] and the huge search space that the different tuning of each parameter can generate.

| | Number of hyperparameters | | |
|---|---|---|---|
| | Total | Discrete | Continuous |
| Bernoulli naïve Bayes | 2 | 1 | 1 |
| kNN | 3 | 2 | 1 |
| Decision Tree | 4 | 1 | 3 |
| Linear SVM | 4 | 2 | 2 |
| Random Forest | 5 | 2 | 3 |
| Logistic regression | 10 | 4 | 6 |

Table 2.1: Examples of classifiers available in Scikit-Learn and their hyper-parameters.

In section 2.3 we discuss the state of the art in hyperparameters selection techniques. [44] first introduced the notation, adopted later by many others, of CASH problem: combined algorithm selection and hyperparameter optimization. The idea is to search for the best combination – algorithm and hyperparameters – at the same time. Given a set of learning algorithms $A$ with associated hyperparameter space $\Lambda$ and a limited amount of training data $D = (x_1, y_1), \ldots, (x_n, y_n)$, the goal is to determine the algorithm $A^* \in A$ with optimal generalization performance. Generalization performance is estimated by

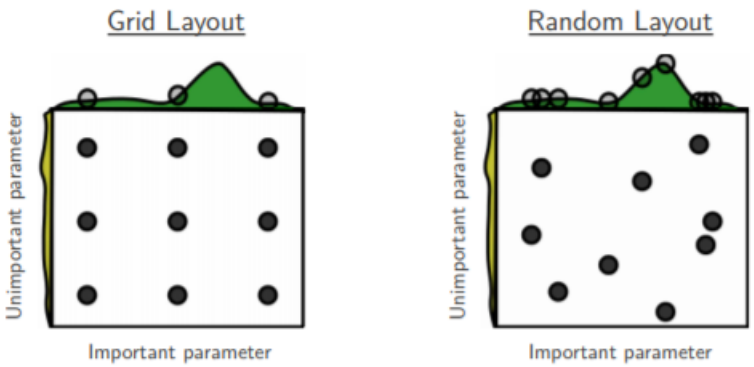



Figure 2.1: With grid search, nine trials only test three distinct places. With random search, all nine trails explore distinct values(from [3]).

splitting $D$ into disjoint training and validation sets $D_{train}^{(i)}$ and $D_{valid}^{(i)}$, learning functions $f_i$ by applying $A^*$ to $D_{train}^{(i)}$, and evaluating the predictive performance of these functions on $D_{valid}^{(i)}$. The CASH problem can be defined as:

$$A_{\lambda^*}^* \in argmin_{A^{(j)} \in A, \lambda \in \Lambda^{(j)}} \frac{1}{k} \sum_{i=1}^{k} \mathcal{L}(A_\lambda^{(j)}, D_{train}^{(i)}, D_{valid}^{(i)}) \tag{2.1}$$

where $L(A_\lambda^{(j)}, D_{train}^{(i)}, D_{valid}^{(i)})$ is the loss achieved by A with the set of hyperparameters $\lambda$ when trained on $D_{train}^{(i)}$ and evaluated on $D_{valid}^{(i)}$ . Notice that 2.1 is not easily solvable due to the complex and large search space.

The first work that allowed to select models automatically was AUTO-WEKA [44]. This framework was implemented in the standard WEKA package [22], which is a machine learning framework and combine Bayesian optimization [27] [5] methods to define a good instantiation of WEKA for a given dataset. A new version of AUTO-WEKA [35] supports regression algorithms, optimization of all performance WEKA's metrics, parallel runs to boost computation and completed integration with WEKA. Other popular framework is AUTO-SKLEARN [20] [19], which is built-in Scikit-Learn package and follows the CASH problem formulated in AUTO-WEKA.

## 2.3 Automated Hyperparameter optimization

In the whole structure of the pipeline, hyperparameters search is the most solid automated step. The first techniques used to solve hyperparameter optimization were grid and random search. Later works showed that is possible to tune hyperparameter by using a bayesian optimization or genetic algorithms.

### 2.3.1 Grid Search

The first approach proposed to explore combinations of values in the search space was grid search and is the most commonly used method. After setting the search space, this method creates an exhaustive searching through every possible configuration in the search space and evaluate the model for each

combination. It is a good method when there are a few possible values because can be very time-consuming and computationally expensive. When the hyperparameter search space is continuous it is necessary to discretizer into k equidistant values when it is categorical each value is used [26].

In the widely used version, grid search only evaluates the values pre-calculated, it does not exploit the values with the best performing. This means that the search could never found the optimal values. To solve this, contradicting grid search [24] picks the values with the best performance and creates another grid search centered around the best value. This procedure is repeated until converging to a local minimum.

### 2.3.2 Random Search

Random Search is an alternative to grid search [1]. It tries to find the optimal hyperparameters randomly, which do not guarantee that finds the optimal values. This method is more efficient when there are a smaller number of dimensions [4]. In this approach, the convergence speed is faster than the previous one, however knowledge of well-performing values are not exploited. As shown in Figure 2.1, grid search can only test three distinct configurations for nine trials, however, for the same problem, random search test different combinations. Not all hyperparameters are equally important to tune [4], however, Grid search allocates too many trials to the exploration of unimportant hyperparameters.

### 2.3.3 Sequential Model-Based Optimization

Sequential Model-Based Optimization(SMBO) is the state of the art in hyperparameter optimization. In order to run SMBO, it is needed a search space, a metric to optimize (usually accuracy), the surrogate model of the objective function, criteria to select the next promising configuration and history of information about the previous configurations explored.

A Bayesian optimization [7] is an iterative optimization tool being well suited for expensive objective functions. To implement the surrogate model of the objective function is used a Bayesian optimization. To handle the trade-off between exploration and exploitation, the acquisition function explores new regions with high uncertainty, avoiding the procedure being stuck in a local minimum. On the other hand, the acquisition function exploits well-performing regions with a low uncertainty converging to a local minimum.

In the CASH problem, the loss function can be approximated using regression methods based on the tested hyperparameter configurations [27]. As an example, a SMBO framework, a sequential model-based optimization configuration (SMAC), a tree-based Bayesian optimization tool was used in AUTO-WEKA [44] as the optimizer in order to solve the CASH problem. About the probability model, Bayesian optimization algorithms can be divided into three categories: Gaussian processes, Tree Parzen Estimators and random forests.

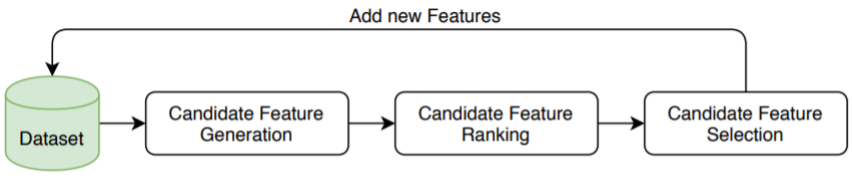



Figure 2.2: Iterative feature generation procedure(from [49])

### 2.3.4 Particle Swarm Optimization

Particle Swarm Optimization [15] is an evolutionary strategy inspired by the behaviour of biological communities such as animal species living in large colonies like birds, ants or fish.

In evolutionary algorithms, an initial population is randomly created. Each individual's performance is evaluated using an objective function and the best performing individuals are selected. A new generation of individuals is created with similar behavior from the selected ones. And the process repeats.

In Particle Swarm Optimization, each individual share a common goal that is realized by exploring the environment. A set of particles with random position and velocity is created. Each particle is evaluated against the objective function and the velocity is updated. Finally, each particle moves to a new position based on its velocity. Instead of dealing with individuals, the algorithm moves the entire population around the search space nearby the best samples. There is no guarantee of reaching the optimal solution.

## 2.4 Feature Generation

The classification algorithms receive as input a set of attributes of the classified instances. However, picking only the target's attributes might not be enough to learn an accurate model that can predict a classification. The success of a machine learning model requires the generation of features that provide useful information about the data. A feature can be an attribute, the result of some operation applied to an attribute or a set of attributes. A feature is a characteristic of the data that might help understand some behavior of it. Hence, most machine learning applications take feature engineering as a vital preposition step, where useful features are generated or selected. Traditionally, domain experts took these steps and did it manually using their intuition and statistics measures to create features.

Feature generation is the process of creation features from a given dataset. The purpose is to explore hidden relationships between features creating new features in order to maximize the model performance. It is hard to generalize because is a task that requires domain knowledge. The number of possible generated features can be limitless since it is possible to perform operations on existed features. This process is the part that most heavily involves humans since it is driven by intuition and knowledge about the domain.

The iterative feature generation procedure follows the structure displayed in Figure 2.2 where features are generated from the data. Those features are ranked and the ones with high ranks are evaluated and added to the dataset. This procedure is repeated until achieving a desired threshold.

### 2.4.1 Feature Generation without domain knowledge

To generate features it is used three types of operators [34]:

- **Unary** operators transform a single feature by applying some mathematical or semantic operator, i.e. extract the year from a date
- **Binary** operators combine two features. The most important type is feature correlation which by using regression models can create informative features.
- **High-Order** operators explore more complex relations in data using clustering and group-by to join similar values.

As part of the Data Science Machine(DSM), it was developed the Deep Feature Synthesis(DFS) algorithm [31] that demonstrated its efficacy in online data science competitions beating human teams. DFS explores the schema of entity-relation datasets. The feature space cardinally grows very quickly due to applying all operators in all features. Most of the generated features are irrelevant for the use case. In order to reduce the feature set, DFS reduces the size of the feature space by employ Truncated SVD transformation. Then, they calculate its $f - value$ according to the target and choose high ranked features.

A disadvantage of the DSM framework is that it does not support feature learning for unstructured data such as sets, sequences, series, text and so on. Features are simple basic statistics that were aggregated for every training example independently of the target variable and from other examples. It could not find structures and patterns in data and express them in features.

### 2.4.2 Feature Generation with domain knowledge

Without domain knowledge, it is possible that the model has a lack of important new features. A set of features resulted from attributes and combinations of them might be not enough to cover important information. For instance, to understand the dissemination of a virus, the data scientist has data from the countries of each patient. Using the country as a feature may be too restrictive. The data scientist can create a feature to cluster countries, so instead of use Uganda and Rwanda as origin label, it is used Central Africa.

In the work of [21], they developed an algorithm that generates features by using relational expansions. Additionally to the labeled set, the algorithm receives a body of external knowledge represented in relational form. It uses the input features as objects to construct new learning problems with the information provided by the knowledge base. A knowledge base is a library usually made from contributors specialized in some area that contains structure or unstructured data about a specific topic. Examples of knowledge bases are WordNet [18], Wikipedia, TextRunner [2], Probase [47], and many others.

Another interesting work is Feature Hub [43]. The idea is that independent data scientists collaborate on a feature engineering task, viewing and discussing each other's features in real-time. Then, the code is integrated into an automated machine learning backend that validates the predictions. Before write

scripts for feature extraction, data scientists can read background information about the problem, load the dataset, and analyze and explore data. After the data scientist submits features, FeatureHub builds a simple machine learning model on training data using submitted features and informs the user metrics that show the behaviour of those features. Then, if the results were satisfactory, the feature can be submitted to a feature database. The main disadvantage of Feature Hub is completely dependent on humans to create features and humans are responsible if a feature should be in a feature database.

## 2.5 Feature Selection

Feature Selection is the process that finds a feature subset based on the original feature set. It can have a huge impact on model performance: memory, computational cost and time. Load the model with too many features can lead to overfit if they are not the proper ones and could take too much time (the curse of dimensionality) to learn the model. These might be enough reasons to spend some time choosing what features better describe data.

Feature selection includes dimension reduction and feature ranking. When the feature set has high dimensionality or great redundancy might be necessary to apply dimension reduction techniques. The truth is some features encode irrelevant information in a specific context (e.g. the first name of people with cancer). These techniques can be divided into feature selection or feature projection. In the first type, the most common methods are lasso and greedy search. Feature projection transforms the data in high-dimensional space to a lower one, where the most used techniques are principal component analysis (PCA), non-negative matrix factorization (NMF) and linear discriminant analysis (LDA).

Considering the information retrieval that each individual feature adds to the model from a statistical perspective(univariate selection), it is possible to decide if a feature should be included or not in the result set without domain knowledge [40] [12].

To evaluate features there are three different types of methods: filter, wrapper and embedded methods [42].

- **Filter Method** scores each feature by looking into the data and chooses by its divergence or correlation according to a certain threshold. It uses statistical information like variance, correlation coefficient, Chi-square test, and mutual information to score each feature. It is very fast and simple to apply, however, it does not take into consideration hidden relations between features: low scoring features can perform well when grouped with others.

- **Embedded method** performs variable selection as part of the learning procedure. Regularization, decision tree, and deep learning are all embedded methods. One example of an embedded method is L1 Regularization (lasso). L1 adds a penalty against complexity to reduce overfitting by adding more bias. Lasso shrinks the less important feature's coefficient to zero thus, removing some feature altogether.

- **Wrapper method** generates various subsets of features and evaluates each one. To evaluate the method, it trains and tests a specific classification model, being the accuracy of the model used

as a criterion measure for feature quality. The most common techniques are Forward selection, backward elimination and bi-directional elimination. This method can be very computationally expensive and has high chances of overfit because the evaluation measure requires the training of models.

ExploreKit [33] is claimed as the most representative in automated feature engineering. In order to generate features, ExploreKit applies the three types of operators presented in section 2.4. It is a greedy search strategy since tries to find all possible features. ExploreKit uses meta-learning techniques in the ranking step to reduce the feature set. To do that, ExploreKit uses a pre-trained classifier with historical knowledge on feature engineering. Despite the results are very interesting the computational resources needed are not available for everyone. The experiments were conducted on a 20-core machine with 64GB of RAM and the allowed time for each dataset was 3 days (the experiments used 25 different datasets).

## 2.6 Time Series

According to Esling [16], time series are sequences of measurements for a single entity over time, which are collected at equally spaced points in time, describing the behavior of a process, system or event. It is the representation of real world processes like sunlight per day or blood pressure along the day, two examples of *univariate* time series. By other hand, measurements are commonly taken from multiple parts of a system simultaneously, yielding *multivariate* time series. Some examples include the number of vehicles on roads and pollution level along time or number of sold items and marketing campaigns.

As people need and like to measure things and as companies need to register business metrics, time series data is the most growing type of datasets. So a growing need to explore and extract value from this data is emerging.

We define a time series, $ts$, as a vector of timestamps, $t_i$, with length $N$, and associated measurement $x_t$. Each measurement can be a combination of $D$ variables(if $D = 1$ we have a univariate time series). Therefore, a time series is defined as:

$$ts : X = (x_1, x_2, ..., x_N) \in \mathbb{R}^{N \times D} \tag{2.2}$$

where, for each $t \in \{1, 2, ...N\}$, $x_t \in \mathbb{R}^D$ represents the t-th measurement of all variables $D$. Also, the $N$ measurements have been collected at equally spaced time intervals.

Time series modeling allows replicating all the individual processes into a combination of signals and noise, without necessarily knowing the causes for each. The signal is the underlying trend and the noise is the difference between the observed value and the trend. According to Bisgaard [6], a time series can have several components, such as :

- The **Horizontal** component shows the variation of the data around a constant mean.
- The **Trend** shows the likelihood of the data to increase or decrease along with the measurements.

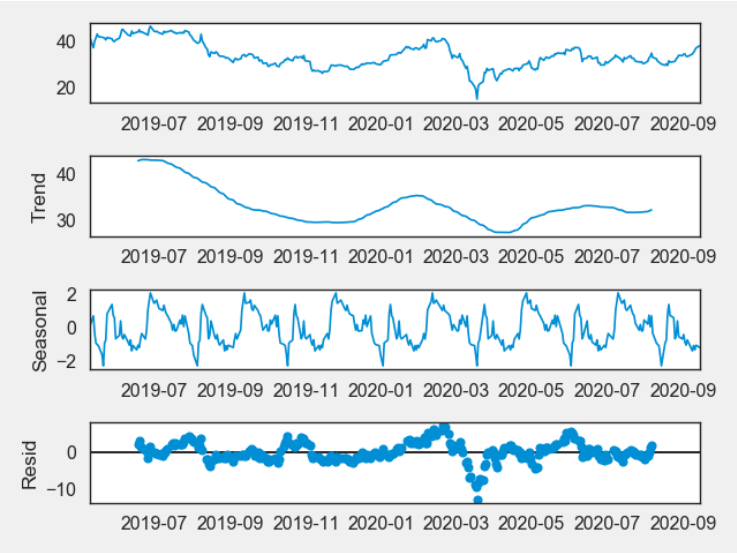



Figure 2.3: Four components of a time series: time series observed, time-trend component, seasonal component, error/residuals/random component.

The data can have different trends in different periods of time. It shows the up/downward pattern of movement but is not necessarily linear, it could be polynomial, exponential, or logarithmic.

- A **Seasonal variation** is an event that occurs within a well-defined period of time. Usually, seasonal variations have almost the same pattern along with the measurements. These occurrences can have multiple causes, e.g. biological factors or weather conditions.

- A **Cyclical variation** is the transition within a set of states. It is irregular both in height of peak and duration. Some examples can be the weather seasons or economy cycle (prosperity, recession, depression, and recovery).

- A **Random/irregular variation** is an event that never happened before or has a small likelihood to happen. Most of the times, it is considered noisy data due to some measurement error or some anomaly.

Figure 2.3 illustrates the time series decomposition. The sum (additive model) or the multiplication (multiplicative model) of the four components result in the original time series.

A stationary time series is one whose properties do not depend on the time at which the series is observed. Thus, time series with trends, or with seasonality, are not stationary — the trend and seasonality will affect the value of the time series at different times. On the other hand, a white noise series is stationary — it does not matter when you observe it, it should look much the same at any point in time.

In order to automatically know if a time series is stationary and what to do if so, we can apply the Augmented Dickey-Fuller test.

#### Augmented Dickey-Fuller test

The Augmented Dickey-Fuller test is a type of statistical test called a unit root test. There are a number of unit root tests and the Augmented Dickey-Fuller may be one of the more widely used. It uses an autoregressive model and optimizes an information criterion across multiple different lag values

- Null Hypothesis (H0): It suggests the time series has a unit root, meaning it is non-stationary. It has some time dependent structure.
- Alternate Hypothesis (H1): It suggests the time series does not have a unit root, meaning it is stationary. It does not have time-dependent structure.

If the series is non-stationary, we need to manipulate it in order to have a stationary time series. One way to make a non-stationary time series stationary is to compute the differences between consecutive observations. This is known as *differencing*. Other transformations such as logarithms can help to stabilise the variance of a time series. Differencing can help stabilise the mean of a time series by removing changes in the level of a time series, and therefore eliminating (or reducing) trend and seasonality.

### 2.6.1 Time series Forecast

Hyndman [29] defines the time series forecast task as ”predicting the future as accurately as possible, given all of the information available, including historical data and knowledge of any future events that might impact the forecasts”.

#### Statistical Approach

*Exponential smoothing* was proposed in the late 1950s [8] [25] [46], and has motivated some of the most successful forecasting methods. Forecasts produced using exponential smoothing methods are weighted averages of past observations, with the weights decaying exponentially as the observations get older. In other words, the more recent the observation the higher the associated weight.

Additionally, we have *ARIMA*. ARIMA is an acronym that stands for AutoRegressive Integrated Moving Average. It is a generalization of the simpler AutoRegressive Moving Average and adds the notion of integration.

This acronym is descriptive, capturing the key aspects of the model itself. Briefly, they are:

- AR: Autoregression. A model that uses the dependent relationship between an observation and some number of lagged observations.
- I: Integrated. The use of differencing of raw observations (e.g. subtracting an observation from an observation at the previous time step) in order to make the time series stationary.
- MA: Moving Average. A model that uses the dependency between an observation and a residual error from a moving average model applied to lagged observations.

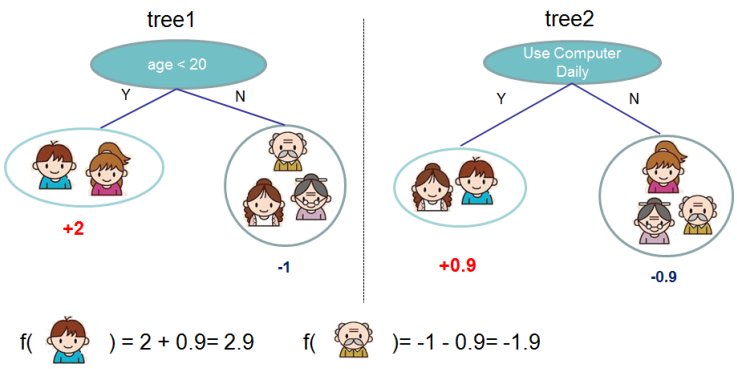



Figure 2.4: Tree Ensemble Model. The final prediction for a given example is the sum of predictions from each tree. From [9].

Thus, the model is defined as $ARIMA(p, d, q)$, where $p$ = order of the autoregressive part; $d$ = degree of first differencing involved; $q$ = order of the moving average part.

However, ARIMA models are also capable of modelling a wide range of seasonal data. A seasonal ARIMA model (SARIMA) is formed by including additional seasonal terms in the ARIMA models:

$$SARIMA \underbrace{(p, d, q)}_{\text{non seasonal part of the model}} \underbrace{(P, D, Q)m}_{\text{seasonal part of the model}}$$

where m = number of observations per unit of time. We use uppercase notation for the seasonal parts of the model, and lowercase notation for the non-seasonal parts of the model.

### Ensemble-based Approach

Ensemble learning is a machine learning paradigm where multiple learners are trained to solve the same problem. In contrast to ordinary machine learning approaches which try to learn one model with training data, ensemble methods try to construct a set of weak models and combine them to obtain a stronger one than a single model. As represented in Figure 2.4 is represented two trees and the output is the sum of predictions of both of them.

Dietterich [14] gave three reasons by viewing the nature of machine learning as searching a hypothesis space for the most accurate hypothesis. The first reason is that, the training data might not provide sufficient information for choosing a single best learner. For example, there may be many learners perform equally well on the training data set. Thus, combining these learners may be a better choice. The second reason is that, the search processes of the learning algorithms might be imperfect. For example, even if there exists a unique best hypothesis, it might be difficult to achieve since running the algorithms result in sub-optimal hypotheses. Thus, ensembles can compensate for such imperfect search processes. The third reason is that, the hypothesis space being searched might not contain the true target function, while ensembles can give some good approximation. For example, it is well-known that the classification boundaries of decision trees are linear segments parallel to coordinate axes. If the target classification boundary is a smooth diagonal line, using a single decision tree cannot lead to a good result yet a good approximation can be achieved by combining a set of decision trees.

XGBoost[10] is short for Extreme Gradient Boosting and is an efficient implementation of the stochastic gradient boosting machine learning algorithm. The stochastic gradient boosting algorithm, also called gradient boosting machines or tree boosting, is a powerful machine learning technique that performs well or even best on a wide range of challenging machine learning problems. Tree boosting has been shown to give state-of-the-art results on many standard classification benchmarks. It is an ensemble of decision trees algorithm where new trees fix errors of those trees that are already part of the model. Trees are added until no further improvements can be made to the model.

XGBoost provides a highly efficient implementation of the stochastic gradient boosting algorithm and access to a suite of model hyperparameters designed to provide control over the model training process.

**Neural Networks Approach**

Artificial neural networks draw inspiration from computational biology. The advantage of neural networks is the ability to learn highly nonlinear patterns in the data. As in the scope of machine learning, neural networks are function approximators that not only learn a mapping from $X$ to $Y$ , or $Y$ given $X$, but are able to learn novel representations of the data.

A neural network can be thought of as a network of "neurons" which are organised in layers. The predictors (or inputs) form the bottom layer, and the forecasts (or outputs) form the top layer. There may also be intermediate layers containing "hidden neurons".

In a traditional neural network, we assume that all inputs are independent of each other, and the same for the outputs. Artificial neural networks do not have the memory to understand sequential data. The idea behind Recurrent Neural Networks(RNNs) is to make use of sequential information. RNNs are called recurrent because they perform the same task for every element of a sequence, with the output being depended on the previous computations. They are networks with loops in them, allowing information to persist, [32]. A loop allows information to be passed from one step of the network to the next. A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor.

RNN's have troubles about the short-term memory. If a sequence is long enough, they have a hard time carrying information from earlier time steps to later ones. Therefore, these causes the need of Long Short Term Memory (LSTM) which is a special kind of RNN's, capable of learning long-term dependencies. LSTM's have skills to remember the information for a long periods of time.

As shown in Figure 2.5, the network takes three inputs. $X_t$ is the input of the current time step. $h_{t-1}$ is the output from the previous LSTM unit and $C_{t-1}$ is the "memory" of the previous unit. As for outputs, $h_t$ is the output of the current network. $C_t$ is the memory of the current unit. Therefore, this single unit makes decision by considering the current input, previous output and previous memory. And it generates a new output and alters its memory.

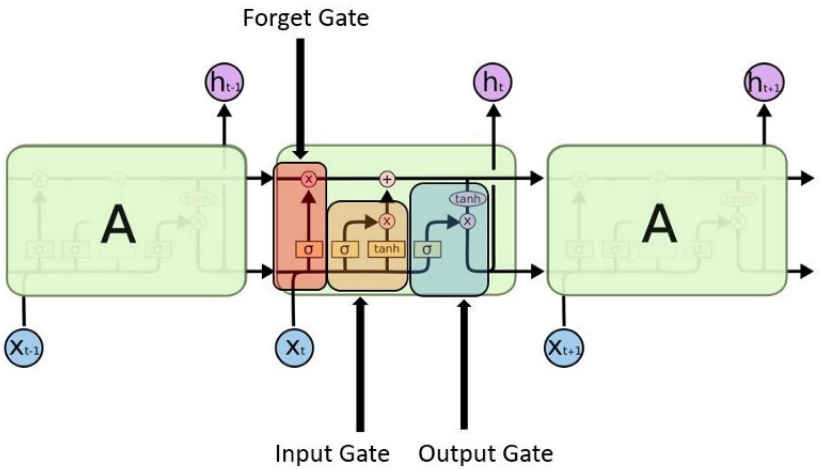



Figure 2.5: LSTM module has 3 gates named as Forget gate, Input gate, Output gate.

## 2.7 Python Libraries

The most common approach to classify time series focuses on manually calculate its properties by applying basic statics like mean, variance and others, and explore other measures like signal processing. Then these properties are used as features.

For feature engineering, libraries such as `pandas` [37] or `sickit-learn` [39] provide a lot of useful methods. Recently, were developed two `python` libraries to automatically extract features from time series which enable automated calculation of important features.

`tsfresh` [11] implements interfaces like the ones mentioned above which allow integrating with traditional pipelines. It provides 63 time series characterization methods, which compute a total of 794 time series features. Each time series is represented as a feature vector with size $[n_{samples}, n_{features}]$ rather than the raw dataset with size $[n_{samples}, n_{timestamps}]$. This output can be used as input to any machine learning algorithm. Features are ranked and selected by the `FRESH` algorithm implemented by this library. The algorithm performs hypothesis tests to measure the dependency between the target labels and each feature's values, and selects a subset of the features based on the p-values computed by these tests. Its widespread adoption shows that the market and recent needs due high production of temporal data require a way to automatize feature engineering [30] [36].

`tsfuse` [13] aims to extend the work done before by `tsfresh` to cover multivariate time series. Note that `tsfresh` can handle multivariate time series by considering each series separately which does not consider relations between data.

## 2.8 Open Issues

The frameworks present at the beginning of this section focus on supervised learning. Those frameworks enable domain experts building reasonable well-performing machine learning pipelines without the need to understand how to do it [49]. Most of them are specialized in model selection and hyperparameter optimization, while feature engineering is not so much advanced. Besides, feature engineering with domain knowledge is not very developed, and what is done requires external knowledge sources.

# Chapter 3

# Framework

In this chapter we present our approach to automatically analyze univariate time series. As explained before, it is increasing the necessity of tools that could help data scientists automatize some task of the data science process, or even automatize the all process. Our goal is to build a framework that could treat the time series analysis as a black-box problem. The data scientist, or the user, send to the framework his piece of data and a time range for forecast. After that, the all process is unseen to the user, and when the process finish, the user receives the forecast for the given time range. Figure 3.1 illustrates this flow and the interaction with the user. The framework is available online[1].

In the following sections, we will describe the architecture and each module of the framework. To illustrate how the framework works and to justify implementation decisions, we will use a sample dataset as example. This dataset was downloaded from Yahoo finance and represents the Google Stock. In section 3.1 we present the articulation of each module. In section 3.2, we present the files type supported. In section 3.3, we present the exploratory analysis. Section 3.4 is responsible for the cleaning steps. Section 3.5 is responsible to describe the operators to generate features, and how the framework chooses the best ones. In section 3.6 we present the machine learning algorithm models to forecast. Finally, in section 3.7 we present the methods to visualize the results.

## 3.1 Architecture

We designed a system with a set of modules that represent the data science pipeline. The framework follows the architecture present in Figure 3.2. This way each module can be changed by other implementation, it is especially important when we want to compare our results with other alternatives for feature engineering.

The *Time Series Structure* is a data structure responsible to store the data, and it has all the necessary methods to analyze, explore and model the series. We define our structure as a two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axis (columns). Figure 3.3 present methods available with parameters' type and returning type.

[1] https://github.com/heliodomingos/autoSeries

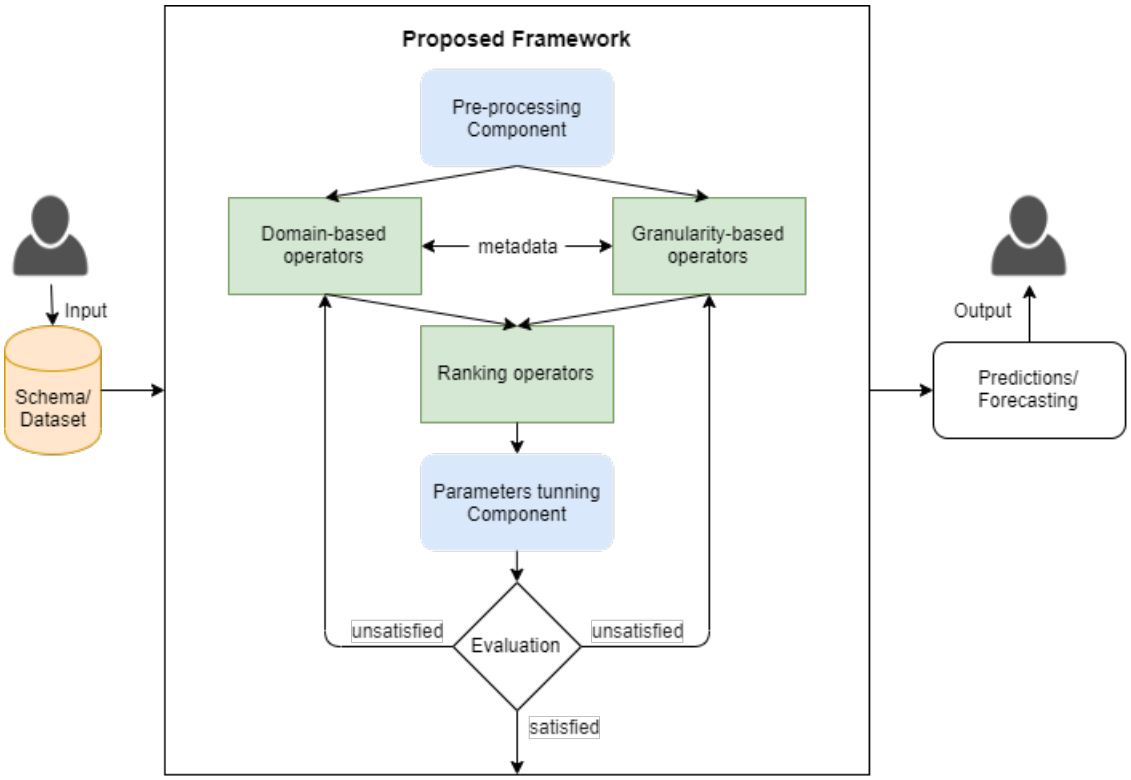



Figure 3.1: The user provides his data to our framework, and the framework will automatic explore the data. In the end, the user receives his output which can be either the classification or forecasting about the data.

## 3.2 Load Data Module

The Load Data Module is responsible for loading data from a given file, in one of several formats, such as `sql`, `csv` and `xls` files. It receives as input a string path to the data source and returns time series structure populated with given data as output. This module is also responsible for analysing the index and calculate which granularities can be explored (for example: if data corresponds to a window of one year it is not possible to forecast if the framework aggregate all the data by year).

In the Load Data Module, the methods available are:

- `load_data(filename)`: this method is responsible to receive the filename of the dataset, read the extension file and call one of the following methods:
- `load_data_from_csv(filename)`: this method is responsible to create a time series structure with the content of an csv file;
- `load_data_from_sql(filename)`: this method is responsible to create a time series structure with the content of an sql file;
- `load_data_from_xls(filename)`: this method is responsible to create a time series structure with the content of an xls file;

Using our example dataset, we start by put the file on the folder ”data” located in the root of the framework. The framework will load the file. The return is an structure of a Time Series. The data structure

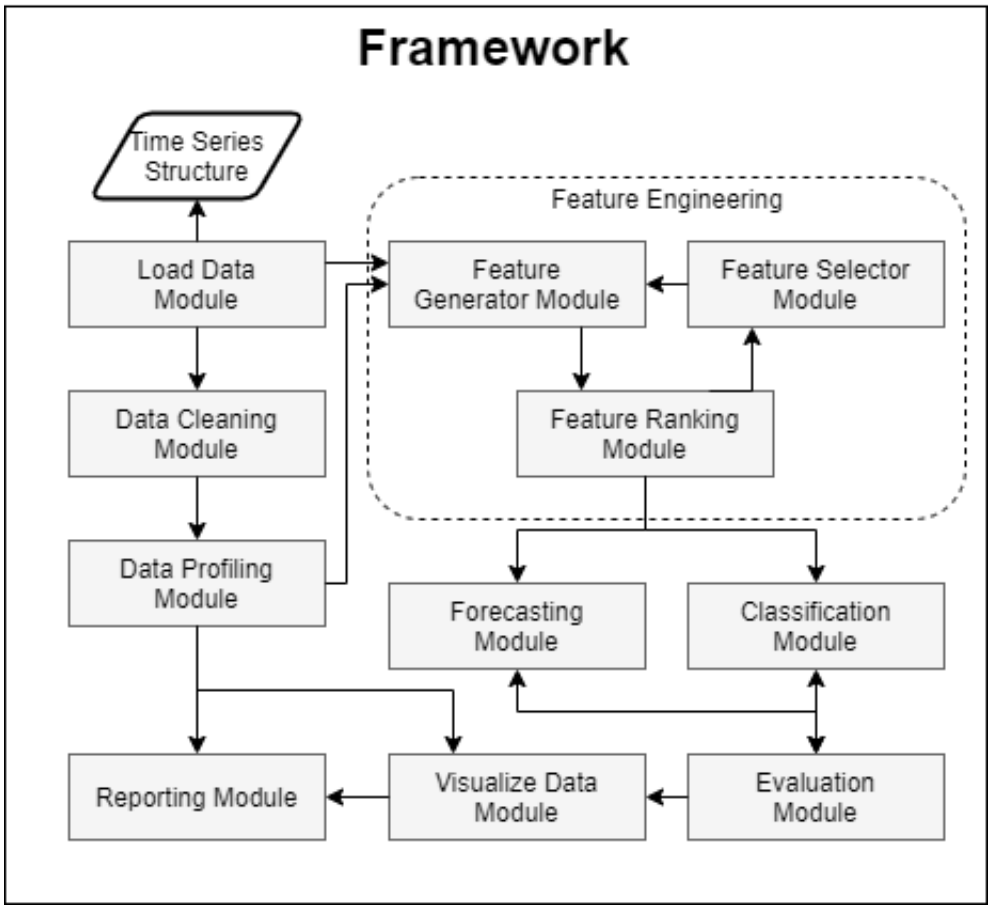



Figure 3.2: Framework's architecture.

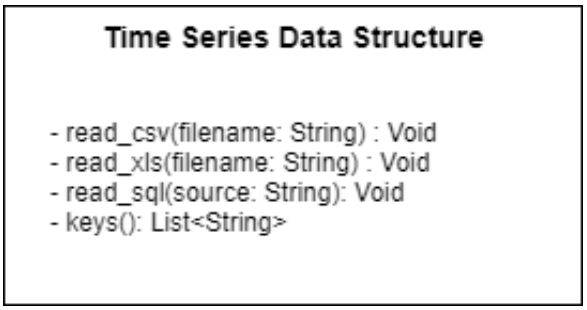



Figure 3.3: Time Series data structure and its methods.

| | close |
|---|---|
| date | |
| 2004-08-20 | 53.952770 |
| 2004-08-23 | 54.495735 |
| 2004-08-24 | 52.239193 |
| 2004-08-25 | 52.802086 |
| 2004-08-26 | 53.753517 |

Figure 3.4: Data structure of the sample datase.

looks like what is shown in Figure 3.4.

## 3.3 Data Profiling Module

We included a step of exploration of the data to better interpret and model the series. We explore stationarity, level, trend, seasonality and noise. This step can be used to help fill missing values and later help build a model. Also, these insights are shown in the visualization tool.

In the Data Profiling Module, the methods available are:

- `explore_data(time_series)`: this method receives a time series structure, and returns a dictionary with a structure with the response of each method that follows:
- `trend(time_series)`: this method is responsible calculate a line with the underlying trend of the data;
- `seasonality(time_series)`: this method is responsible calculate seasonality of the data;
- `stationarity(time_series)`: this method is responsible calculate stationarity of the data;
- `variation(time_series)`: this method is responsible calculate the difference between $y_t - y_0$;
- `noise(time_series)`: this method is responsible calculate the noise from data;

### 3.3.1 Test of Stationarity

The upward or downward trend in the line graph of a time series indicates nonstationarity. Remaining in a constant level that provides a constant mean is an indication of stationarity. But this is not a perfect way to test stationarity of a data series. Sometimes a series can be non stationary in the mean without showing a persistent upward or downward.

In statistics, the Dickey–Fuller test tests the null hypothesis that a unit root is present in an autoregressive model. The alternative hypothesis is different depending on which version of the test is used, but is usually stationarity or trend-stationarity. Stationary series has constant mean and variance over time. Rolling average and the rolling standard deviation of time series do not change over time.

#### Augmented Dickey-Fuller test

As present in section 2.6, this test is the most widely used when we want to test stationary. Here we define the threshold of the p-value and its meaning.

- $p - value > 0.05$: Accept the null hypothesis (H0), the data has a unit root and is non-stationary.
- $p - value <= 0.05$: Reject the null hypothesis (H0), the data does not have a unit root and is stationary.

| Count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|
| 4044 | 529.73 | 381.59 | 49.81 | 239.98 | 341.04 | 769.25 | 1,728.28 |

Table 3.1: Statistics of the sample Dataset.

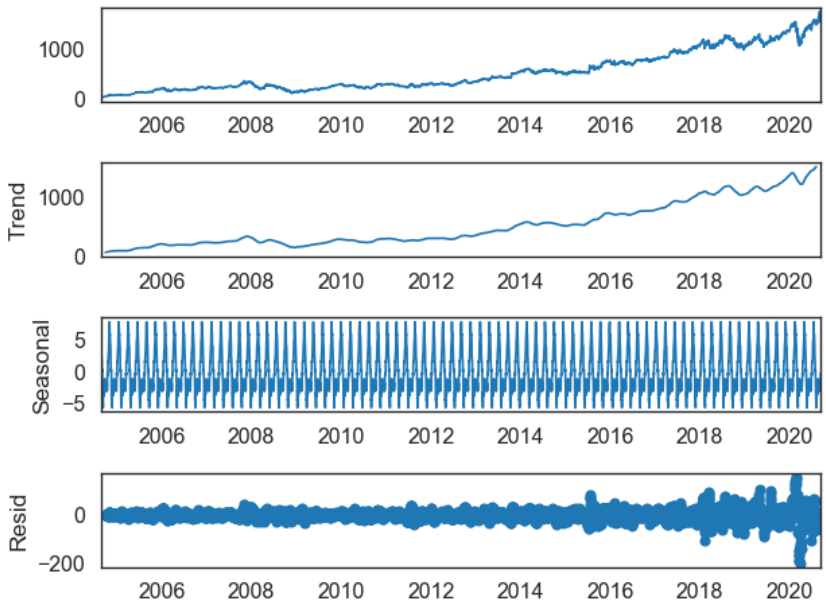



Figure 3.5: Components of a time series applied to the sample Dataset.

If the test suggest the time series is non-stationary, we will compute the differences of first order. Then, we apply the test again. If the result is negative, we repeat the process with the second order difference, until the test suggests that the time series is stationary.

Using our example data, the framework process the data and shows a table with statistics, as the one in Table 3.1, and a Figure with trend, noise, seasonal pattern, as shown in Figure 3.5.

Then, computes the stationarity property. After the computations, our time series has a new shape. Figure 3.6, illustrates how the time series changed.

## 3.4 Data Cleaning Module

The process continues and a cleaning step is performed. There are several alternatives to cleaning or curating a dataset. Some methods are linked to the domain of the problem, or some insight that the user knows about the data and he knows which method the framework should use. We did not perform all of them neither tried to automatize this step.

Usually, in time series problems data scientists start by creating a new range of dates between the first timestamp and the last one. This may originate a lot of missing values. In our case, we understand that by doing that we might creating data points that do not exist in the real world application.

Duplicate values are not analyzed or removed because for univariate time series is very likely to have different timestamps with the same value. However, we defined the methods to be used by the user as desired. Only if the timestamp is the same, the entries are removed.

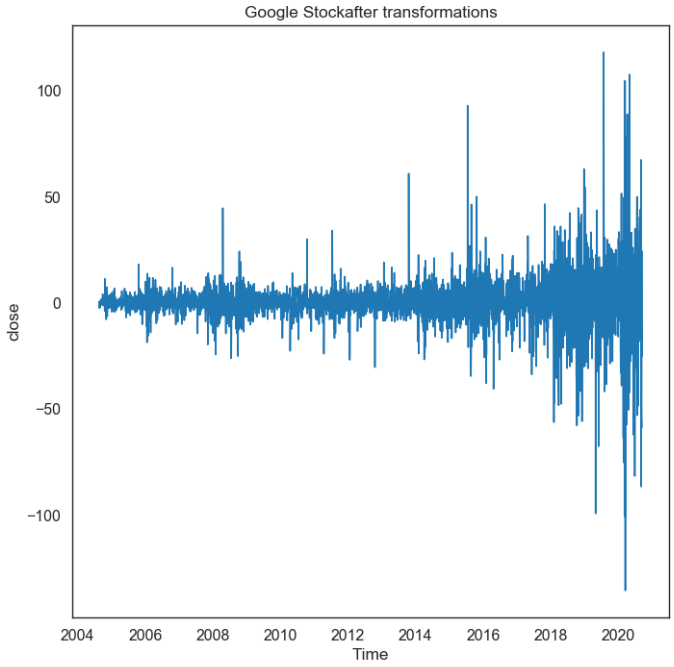



Figure 3.6: Sample dataset after data profiling.

Detection of outliers is also an important task to identify data points that do not correspond to the seasonal pattern or its value is far from the trend. Smoothing a time series remove the influence of outliers, reduce ups and downs, and the time series get close to its trend. This can improve the model performance and it is better for larger datasets.

We can split the methods in two categories: methods for missing values and methods to smooth time series. In the Data Cleaning Module, the methods for dealing with missing values are:

- `interpolate_missing_values(time_series)`: this method is responsible for receiving the time series structure and returning the same time series after removing or imputation missing values, for that calls one of the following methods:
- `delete_missing_values(time_series)`: removes data points with missing values and returns the time series;
- `insert_by_mean(time_series)`: fills missing values with the mean;
- `insert_by_seasonal_value(time_series)`: finds a seasonal pattern and fills missing values with the last value in the seasonal pattern;

In the Data Cleaning Module, the methods for smoothing the time series are:

- `moving_average(time_series, window, smoothing_factor)`: this method is responsible to smooth the time series using a sliding window and if smoothing factor has value, it uses the exponential moving average with the value given in the parameter. The smoothing factor can has values be-

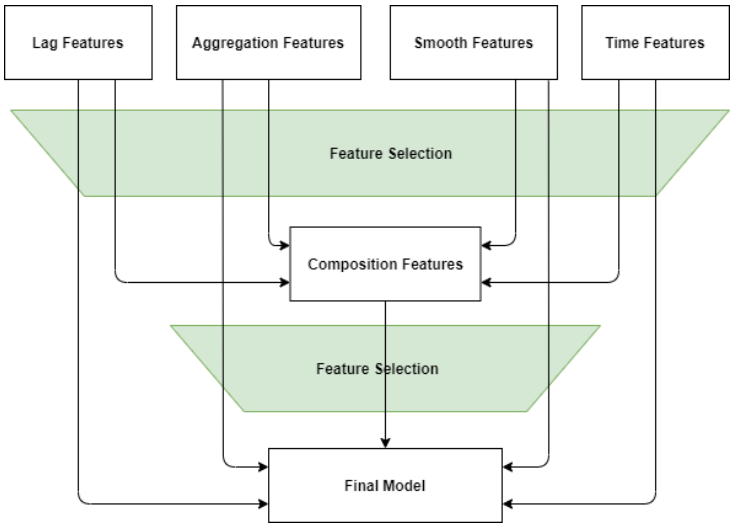



Figure 3.7: A representation of the process of Feature Engineering.

tween 0 and 1, and larger values will actually reduce the level of smoothing, and in the limiting case with smoothing_factor = 1 the output series is just the current observation.

Continuing with the example dataset, the framework analyses the missing values. It founded zero missing values, so it does not apply any technique.

## 3.5 Feature Engineering Module

In the Feature Engineering Module, we create, evaluate and select features. These features will enrich the model to have a better performance. In Figure 3.7 is represented the flow in the Feature Engineering Module. A set of 4 types of features are generated. Those sets are evaluated with an ensemble model, features are ranked and the top features are selected. Those features will be composed with each other and will be added to the ”Final Model”. The composed features are evaluated, with the same methods, ranked and selected, and the top-scoring features are inserted in the Final Model.

This chapter is divided in two sections: Feature Generator and Feature Selection. In the first section we explain the methods used to generate features, and in the second we explain how we select them.

### 3.5.1 Feature Generator

We define two types of operators: extraction-based and granularity-based. The extraction-based are operators that exploit hidden features on the target instances. The granularity-based operators will exploit different time granularities on data. They are responsible to create a new time series with a smaller time interval by aggregating data points.

To help the analysis and the testing of each operator we will separate the operators in groups. The groups are:

- **Lag Features** This group of features represent the past observations. The idea behind this operator is to use past observations has features to describe the value of a prediction. The operator

is:

  - **Lag(n)** Add the data point at time $t - n$ to the tuple of observations at time $t$.

    $Lag_{Op} : ts.lag(ts.lag(n)) -> Numeric.$

- **Time Features** This group of features represent the timestamp unities. The operators can be defined as:

  $Time_{Op} : ts.get_time(ts, unityTime) -> Numeric$

  where $unityTime = \{second, minute, hour, day, week, month, quarter, year\}$

  The operators are:

  - **second_of_minute** Add the second of minute at time $t$ to the tuple of observations at time $t$.
  - **minute_of_hour** Add the minute of hour at time $t$ to the tuple of observations at time $t$.
  - **hour_of_day** Add the hour of day at time $t$ to the tuple of observations at time $t$.
  - **day_of_year** Add the day of year at time $t$ to the tuple of observations at time $t$.
  - **day_of_month** Add the day of month at time $t$ to the tuple of observations at time $t$.
  - **day_of_week** Add the day of week at time $t$ to the tuple of observations at time $t$.
  - **week_of_year** Add the week of year at time $t$ to the tuple of observations at time $t$.
  - **month** Add the month at time $t$ to the tuple of observations at time $t$.
  - **quarter** Add the quarter of year at time $t$ to the tuple of observations at time $t$.
  - **year** Add the year at time $t$ to the tuple of observations at time $t$.

- **Aggregation Features** This group of features represent transformations of the behaviour of the trend or aggregate data point with the same time granularity by one of five aggregate functions, namely: sum, maximum, minimum, average or median.

  - **Up or down?** Looks at the result of the Difference operator and evaluate if the difference is positive or negative. If is positive, the operator returns `True`, if is negative, return `False`.

    $Up\_down_{Op} : ts.up\_down(ts.difference()) -> Boolean.$

  - **Big Up** Looks at the result of the difference operator and evaluate if the difference is above or below the standard deviation(calculated on difference operator). If is positive, the operator returns `True`, if is negative, return `False`.

    $Big\_up_{Op} : ts.big\_up(ts.difference()) -> Boolean.$

  - **Big Down** Looks at the result of the Difference operator and evaluate if the difference is below or above the negative standard deviation(calculated on difference operator). If is positive, the operator returns `True`, if is negative, return `False`.

    $Big\_down_{Op} : ts.big\_down(ts.difference()) -> Boolean.$

- **Difference** Compute a new data point given by the difference between one point and its successor.

  $Difference_{Op} : ts.difference(Void) -> Numeric.$

- **Derivative**: Compute a new data point by the $n$ derivative:

  $Derivative_{Op} : ts.derivative(n : Int) -> Void$, where $n \in \mathbf{N}$.

- **Discrete Wavelet Transform**: Compute a new data point given by the Fourier Transform.

  $DWT_{Op} : ts.dwt(Void) -> Void.$

- **Second-granularity**: Aggregate all data points by the same second:

  $Second_{Op} : ts.resample(second : String) -> Void.$

- **Minute-granularity**: Aggregate all data points by the same minute:

  $Minute_{Op} : ts.resample(minute : String) -> Void.$

- **Hour-granularity**: Aggregate all data points by the same hour:

  $Hour_{Op} : ts.resample(hour : String) -> Void.$

- **Day-granularity**: Aggregate all data points by the same day:

  $Day_{Op} : ts.resample(day : String) -> Void.$

- **Week-granularity**: Aggregate all data points by the same week:

  $Week_{Op} : ts.resample(week : String) -> Void.$

- **Month-granularity**: Aggregate all data points by the same month:

  $Month_{Op} : ts.resample(month : String) -> Void.$

- **Year-granularity**: Aggregate all data points by the same year:

  $Year_{Op} : ts.resample(year : String) -> Void.$

- **Smooth Features** This group of features represent different types of moving averages that transform the series in a series closer to the trend:

  - **Moving Average** Add a new value to the tuple of observations at time $t$ that is the $n$ moving average calculating by the average of different subsets of size $n$ of the full data set.

    $moving\_average_{Op} : ts.moving_average(n : Numeric) -> Void$, where $n$ is a natural number.

  - **Exponential Moving Average** Add a new value to the tuple of observations at time $t$ that is the $n$ exponential moving average calculated by the average of different subsets of size $n$ of the full data set and places a greater weight and significance on the most recent data points, depending on the value of $alpha$.

    $exponential\_moving\_average_{Op} : ts.exp_moving_average(n : Numeric, alpha : numeric) -> Void$, where $n$ is a natural number and $alpha$ is between 0 and 1.

- **Composition Features** After test each feature, we select the best features from each category above, with the methods described in the next section, and the framework calculates a new feature that is the composition of two features selected. For example, if a feature selected from the

group Smooth Features is ”moving average of window 5”, and a feature selected from the group Aggregation features is ”Up or down?”, we calculate the feature ”Up or down?” with the data from feature ”moving average window=5”. This way we create another feature that is the composition of two well performed features.

$Composition_{Op} : ts.composition(f1 : feature, f2 : feature) -> Void$

### 3.5.2 Feature Selection

As explained before, Feature Selection is an important task to reduce the model complexity and overfit. When a feature is generated we observe its statistics. If it has more than 30%of missing values or a standard deviation below 1, the feature is deleted.

We use a Embedded method to help feature selection. It is an ensemble of decision trees, an XGBoost model. After the computation, we calculate a score. This score represents the cover of the features, that is the number of times a feature is used to split the data across all trees weighted by the number of training data points that go through those splits. Besides that, we also allow to select features by *weight* or *gain*. Weight is the number of times a feature is used to split data, and gain is the average training loss reduction gained when using a feature to split. Features are ranked by that score and the top five will be used to compose with other features in the group of Composition Features, and in the Final model. The rest are deleted.

The method available to call feature selection is:

$select\_features(ts, method) : ts.remove\_features(ts.rank_features(method)) -> Void$

where $method$ is a function from {cover, gain, weight}.

Using the sample dataset to exemplify the results of the Feature Engineering module, the framework create a sub-group of features, the first ones are the Lag Features, then evaluates them, and iterates until generates the all groups. In Figure 3.8 is shown the rank of Lag Features, with the scores calculated by the cover metric. In this example, the feature ”lag_21” is the most important feature, and this feature means the past 21st observation. In Figure 3.9, it is plotted the points correspondent to the pair $(ActualData, Predictions)$. In this visualization, we can not see the temporal dependencies, however, the goal is to observe if the predictions are correlated to the observed values/actual data. If they are, we must see the points over a line with equation $y = x + b$. In the given example, the results are lacking correlation, so we conclude that this features did not help to predict data.

## 3.6 Forecasting Module

This module is divided into two parts: baseline approach, and an regression ensemble. The baseline approach will be used as benchmark to compare the results computed with our feature engineering techniques.

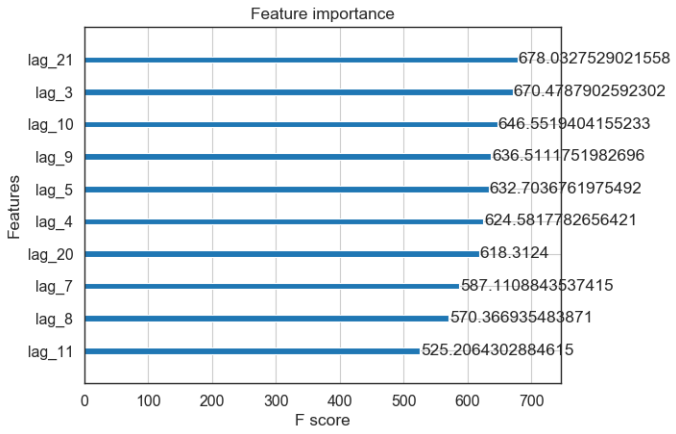



Figure 3.8: Sample dataset with Lag Features.

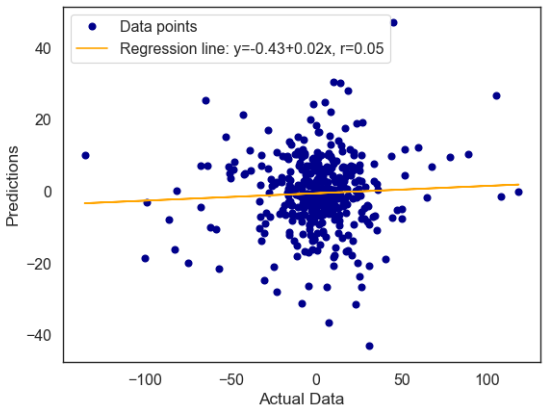



Figure 3.9: Sample dataset: correlation between observed and predicted values with Lag Features.

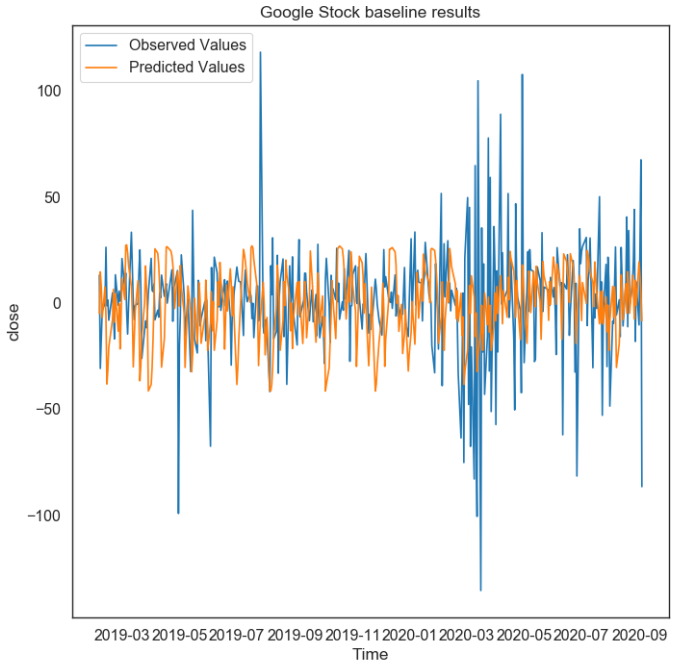



Figure 3.10: Sample dataset: baseline predictions.

### 3.6.1 Baseline Approach

A Baseline approach is important to compare the results from our forecast model with feature engineering. We wanted to use a simple model but at the same time, a model that performs well with time series data. We had two alternative to create the baseline, we could use a statistical method, such as ARIMA or SARIMA, or use a Neural Network with time dependencies, such as Long Short Term Memory neural networks. We chose a Neural Network because it is one of the models most used by data scientists, it is easier to implement and easier to interpret the results. About performance of each possibility learning time series data, they are both very good models.

In our implementation we use four LSTM layers with 50 neurons, each one with a dropout of 20%, to avoid overfitting. The last layer is a simple one with a single neuron that will indicate the value of the forecast. This means that our baselines needs to learn 6050 edges.

The training takes at most 10 epochs and we have a early stop condition that is if the error became steady in at least 3 epochs the train stops. Here, we do not use generated features to enrich data. The idea is to replicate a situation when a data scientist creates a model without explore feature engineering.

To build the baseline model, it is available the method:

$LSTM(ts) : predictions- > structure$

where $predictions$ is a structure with the predictions for the test set.

With our sample dataset, the framework creates a LSTM baseline. This is important to compare the results from the ensemble approach enriched with features. Figure 3.10, illustrates the performance of the baseline predictions.

### 3.6.2 Regression Ensemble Approach

An Ensemble model is a model with a set of weak models that together outperform a single very good model.

XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It is an ensemble of decision trees where new trees fix errors of the trees that are already part of the model. Trees are added until no further improvements can be made to the model.

The main requirement to use XGBoost for time series is to evaluate the model via walk-forward validation, instead of k-fold cross validation, as k-fold would have biased results, because it does not respect the time dependencies.

We divide the forecasting task with XGBoost in five stages. They represent the process of feature engineering, until achieve the final model, as represent in Figure 3.7. The stages are:

- Stage 1: With features from the Lag Features group;
- Stage 2: With features from the Time Features group;
- Stage 3: With features from the Aggregation Features group;
- Stage 4: With features from the Smooth Features group;
- Stage 5: With features from the Composition Features group;
- Stage 5: A Final Model with a selection of features of each group of features.

In each stage we provide a visualization of generated features ranked by:

- **Weight**, meaning the number of times a feature is used to split the data across all trees.
- **Cover**, meaning the number of times a feature is used to split the data across all trees weighted by the number of training data points that go through those splits.
- **Gain**, meaning the average training loss reduction gained when using a feature for splitting.

The method available to call the ensemble approach is:

$XGBoost(ts) : predictions, feature\_data- > structure$

where $predictions$ is a structure with the predictions for the test set and $feature\_data$ is a structure with the number of times each feature was used to split data and how many data points that feature splits.

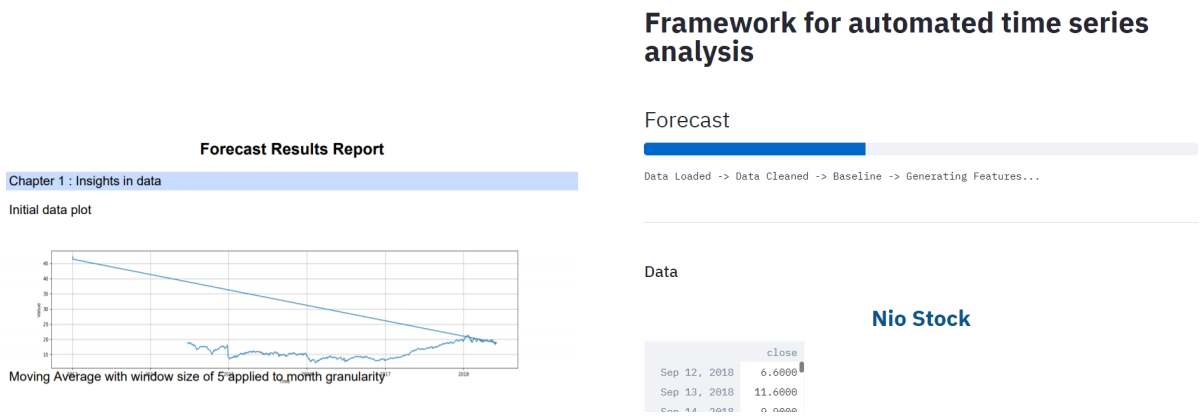
## 3.7 Visualization Module

Visualization is a key factor when we talk about process automatization. The user needs to understand the steps performed, important decisions that were taken, and more important, the results.

Our initial idea was to deliver a PDF file with data plots, following the process of the framework, displaying the computation time, errors, the overall results, and a comparison between the baseline and

other approaches computed. The first problem using this approach was when something gone wrong, the file could not be created and might be harder to the user to understand if something was wrong. Also, format the file to display everything in the correct position, would be an additional challenge that were taking to much time from the important task.

So we change the approach to a html page. Hopefully, we found a open-source framework - `streamlit` [2] - that allows us to integrate with our code and create a simple and interactive app. The user can explore the data, features and line charts. Figure 3.11 shows one of the problems on handling the PDF and a screenshot of the `streamlit` app, much better visually and better organized.




(a) PDF with problems of alignment

(b) Streamlit app

Figure 3.11: Visual comparison between the two visualization alternatives.

The framework can be launched by running the following command line in the root of the project:

```
streamlit run autots.py
```

Besides the framework tool we developed several methods to draw graphics that visualize the time series, predictions and features. The methods are:

$draw\_series(ts) : ts.line\_chart() -> file$, returns an `png` file with the plot of the entire series.

$draw\_hist\_series(ts) : ts.hist() -> file$, returns an `png` file with the histogram of values of the series.

$draw\_observed\_predicted(observedValues, predictedValues) : regression\_line(observedValues, predictedValues) -> file$, returns an `png` file with the plot of the tuples $(y_{observed}, y_{predicted})$ and draw a regression line.

$draw\_feature\_ranking(features, method) : file -> file$, returns an `png` file with the plot of the tuples $(y_{observed}, y_{predicted})$ and draw a regression line.

[2] https://www.streamlit.io/

# Chapter 4

# Case Studies

In this chapter, we present the evaluation methodology to evaluate the forecast and go through three different case studies. In each case study, we present the results for each approach and a comparative analysis. Each case study has different characteristics. The data correspond to financial series. They represent the **Tesla Stock**, the **Nio Stock**, and the **index S&P 500**. The goal is to test our framework with different case studies to observe how the framework will handle each case. In section 4.1, we discuss the methodology and structure of the tests. In the following sections, we go through each case.

## 4.1 Evaluation Methodology

In this section we explain how we propose to evaluate our framework: the test structure, with steps performed; the data used and reasons that support the chosen data; and the metrics used to evaluate each step.

### 4.1.1 Test structure

Each test, or case study analysis, follows the same structure. The data from the case study is used as input to the framework. As illustrated in Figure 4.1, the process starts by the data profiling step. In this step, the framework analyse the stationary property and if the time series is non-stationary, it computes the first-order differencing. In our analysis, we will follow each computation, to verify if the results make sense.

The second step is creating a baseline to compare the results. The baseline chosen was a LSTM neural network model, trained with five epochs and using five past observations as input.

In the following steps are performed Feature Generation, Feature Evaluation and Feature Selection. The four groups of features generated are the Lag Features, Time Features, Aggregation Features and Smooth Features. Each set of features is evaluated with a XGBoost Model, and features are selected by cover score. After these steps, the chosen features are composed between itself.

In the last step, the top features of each set of features is used to create the Final Model.

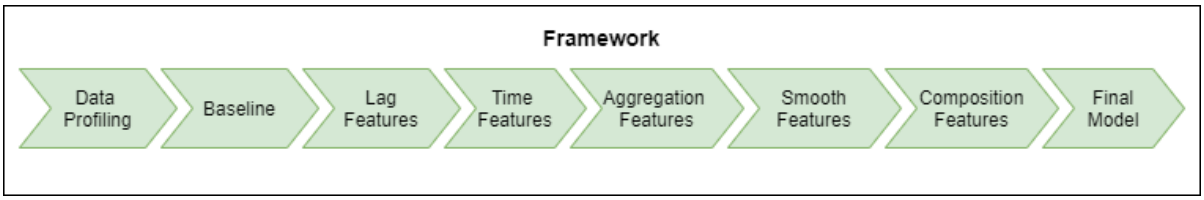



Figure 4.1: Steps performed by the framework and its analysis order of univariate time series.

Table 4.1: The datasets used and their characteristics. (*) Time ranges are up to Sep 14, 2020

| | Tesla Stock | Nio Stock | SP500 index |
|---|---|---|---|
| Time Range * | Jun 29, 2010 | Sep 12, 2018 | Jan 29, 1993 |
| Data Points | 2571 | 505 | 6957 |
| Mean | 47.93 | 5.58 | 141.38 |
| Std | 53.82 | 3.74 | 68.23 |
| min | 3.16 | 1.32 | 43.41 |
| 25% | 7.07 | 3.02 | 99.24 |
| 50% | 43.91 | 4.34 | 126.70 |
| 75% | 58.82 | 7.03 | 173.05 |
| max | 498.32 | 20.44 | 357.70 |
| is stationary | False | False | False |

### 4.1.2 Data

The data under analysis consist in a set of financial series with different characteristics. We examined the daily change of closing prices of Tesla[1], Nio[2], and the SP500 index[3]. The stock prices are downloaded from Yahoo! Finance.

As shown in Table 4.1, each dataset differs in size, mean, std, etc, but, it is important to understand the variation along time. The main reason for choosing each dataset it is their variation along time. The Tesla stock has an exponential increase in its value in the last data points, suggesting the model should learn some exponential curve. The Nio Stock looks like a valley. It starts with a high value, decreases, and then increases. Also, it was chosen by the number of data points, which are much less than Tesla or SP500 index case studies. The SP500 index is the most linear trend. This was chosen mostly by its uptrend and as a good representation of all financial time series.

### 4.1.3 Metrics

Measuring the accuracy of the overall predictions with a single metric is not simple as there is no metric that could describe the error behavior. In our framework, we display a set of different measures to help the user making decisions about the predictions. We define the error as the difference between the prediction, $\hat{y}$ at timestamp $t$ and the observed value, $y$ at timestamp $t$, as follows:

$$e_t = \hat{y}_t - y_t \tag{4.1}$$

The **mean absolute error**, or MAE, is calculated as the average of the forecast error values, where

[1] https://finance.yahoo.com/quote/TSLA/history?p=TSLA
[2] https://finance.yahoo.com/quote/NIO/history?p=NIO
[3] https://finance.yahoo.com/quote/%5EGSPC/history?p=%5EGSPC

all we use the absolute value of each error.

$$MAE = \frac{1}{n}\sum_{t=1}^{n}|e_t| \tag{4.2}$$

The **mean absolute percentage error**, or MAPE, is the average of the percentage errors. Percentage errors have the advantage of being unit-free, and so are frequently used to compare forecast performances between data sets. The most commonly used measure is:

$$MAPE = \frac{100}{n}\sum_{t=1}^{n}\left|\frac{e_t}{y_t}\right| \tag{4.3}$$

Measures based on percentage errors have the disadvantage of being infinite or undefined if $y_t = 0$ for any $t$ in the period of interest, and having extreme values if any $y_t$ is close to zero.

The **root mean squared error**, or RMSE, is a quadratic scoring rule that also measures the average magnitude of the error. It's the square root of the average of squared differences between prediction and the observed value.

$$RMSE = \sqrt{\frac{1}{n}\sum_{t=1}^{n}(e_t)^2} \tag{4.4}$$

The **Pearson correlation coefficient**, for short correlation, is a measure of the strength of a linear association between two variables and is denoted by $r$. A Pearson product-moment correlation attempts to draw a line of best fit through the data of two variables, and the Pearson correlation coefficient, $r$, indicates how far away all these data points are to this line of best fit (i.e., how well the data points fit this new model/line of best fit). $r$ can take a range of values from +1 to -1. A value of 0 indicates that there is no association between the two variables. A value greater than 0 indicates a positive association; that is, as the value of one variable increases, so does the value of the other variable. A value less than 0 indicates a negative association; that is, as the value of one variable increases, the value of the other variable decreases.

Finally, the **accuracy**. We interpret the forecast results a classification problem. In this case, we are looking to predict if the value of a series will increase or decrease relative to the day before. So we define the True Positive(TP), True Negative(TN), False Positive(FP) and False Negative(FN), as:

- TP: the data points where the observed value was higher than the data point before and the predicted value was higher than the data point before;
- TN: the data points where the observed value was lower than the data point before and the predicted value was lower than the data point before;
- FP, as the data points where the observed value was lower than the data point before and the predicted value was higher than the data point before;
- FN: the data points where the observed value was higher than the data point before and the predicted value was lower than the data point before;

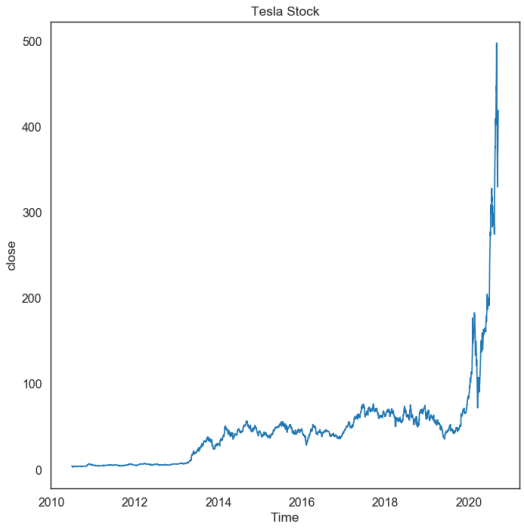



Figure 4.2: Daily close value for Tesla stock between Jun 29, 2010 and Sep 14, 2020.

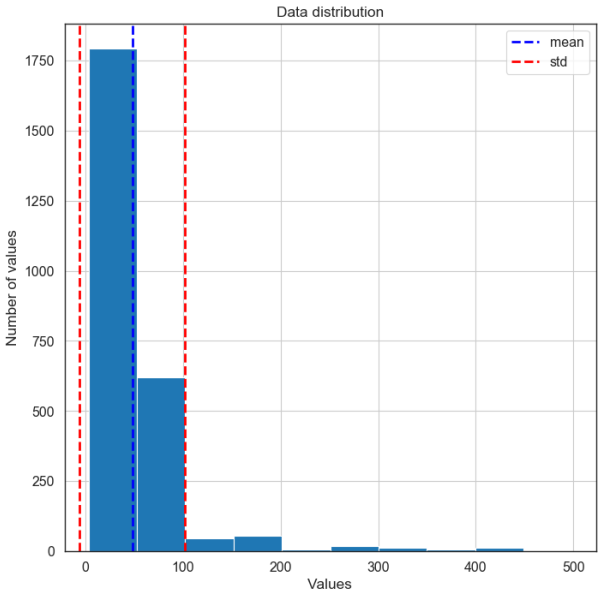



Figure 4.3: Distribution of Tesla stock values in bins of size 50.

So accuracy is given by:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.5}$$

In conclusion, the metrics are applied to different error behaviors. MAE, RMSE and MAPE show the overall distance from the observed values and predicted values. MAE gives the same weight to each error, while RMSE gives more weight to larger errors, which can be an inaccurate metric in presence of outliers. MAPE is a percentage metric, being easily to compare against other models. The correlation metric is important to observe if the predictions are correlated to observed values or can be consider noise. Finally, accuracy brings a new perspective, if the model predicted well variations through time, without consider that variation size.

## 4.2 Tesla Stock

The Tesla stock is made up of 2571 data points, with a high standard deviation of about 53 and is not stationary. From Figure 4.2, we can split the analysis into four periods of time. The first period goes from Jun 29, 2010 to mid-April 2013 the stock price almost remained constant. The second period goes from mid-April 2013 to July 2019, when the stock price had a slight uptrend, beginning with a value of 16$ and ending with a value of 50$. This period corresponds to approximately three years and the variation is not very significant. The third period extends up to mid-March of 2019. During this period, the stock saw a big increase of its value followed by a big fall. The stock started on 50$, rose to 183$ until mid-February and went back down to 72$ on March 18. Despite a small variation between the beginning and the end of the period, we have a high standard deviation. The last period corresponds to a high positive trend, rising the stock value from 72$ to 419$ on September 14, reaching 498$ on August 31.

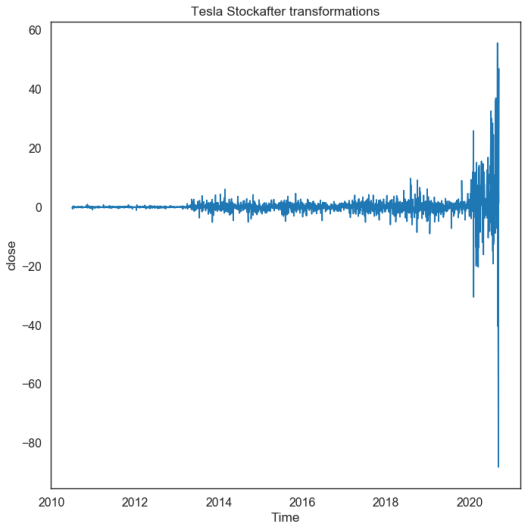



Figure 4.4: Tesla stock data after data profiling.

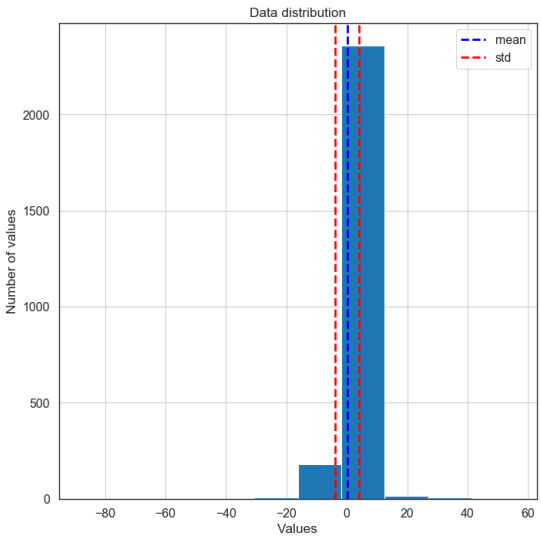



Figure 4.5: Distribution of values after data profiling in bins of size 20.

This stock is a curious case when we observe its trend. As we have two periods in which the stock price variation is not significant, the process is stationary. Meaning that we won't need to transform into a stationary process. Also, the last period can be interpreted as an outlier, if analyzed from a macro perspective. However, it is a new stage of the stock, and trades from now on should consider this new price.

Figure 4.3 illustrates the value distribution. There are more values between 0 and 50 than values between 50 and 500. And values between 0 and 100 compose 96% of the all data. It is expected that this conditions will have an high impact when making predictions.

In the following sections, we will present the results and a discussion of those results, for each step of analysis of the framework, which follows the steps shown in Figure 4.1.

### 4.2.1 Data profiling

The framework starts by analyzing the stationary property. The Augmented Dickey-Fuller test informed that the data is non stationary, so it performs a first-order differencing. After that, we scale data to have zero mean and unit standard deviation. The time series after data profiling is shown in Figure 4.4, where we can see a different time series. The new values are more bonded between -20 and 20, with some exceptions in the end of the period. From Figure 4.5 we see that mean is close to zero, and the values outside the range -20 to 20, correspond to the last period when the stock grew very fast.

### 4.2.2 Baseline

The baseline results, presented in Figure 4.6, or the LSTM results, show poor predictions when we compare the observed values and the predicted values. The overall predictions are bounded by the observed values. We can see, from Figure 4.6, that the model made predictions always in a range

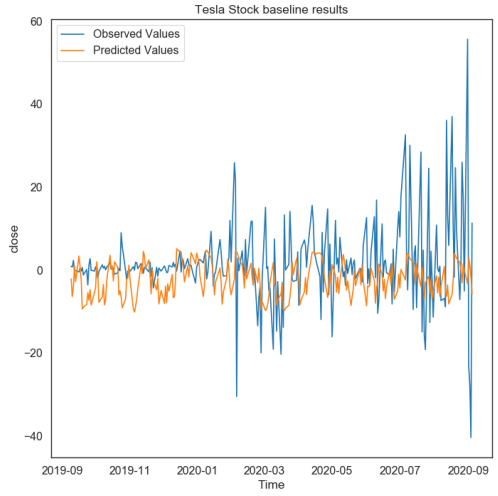



Figure 4.6: Baseline results for Tesla stock: on blue the observed values and on orange the predicted values.

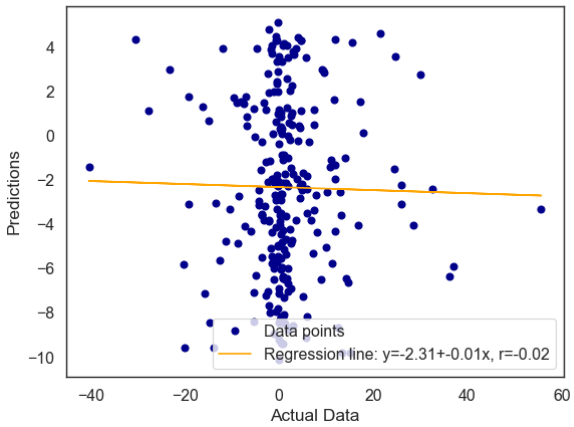



Figure 4.7: Regression Line of Tesla baseline results.

between -8 and 4 whereas the observed values correspond to a range between -40 and 60. Additionally, in the period between 2019-09 and 2020-01, the observed values have a low variation, and in the period between 2020-07 to 2020-09, the variations are higher. However, the LSTM model made predictions with the same variations along time.

The MAE is about 7.92 and RMSE of 11.41. Accuracy is near 0.46, which means that the model only could predict correctly half of the positive variations. As shown in Figure 4.7, the results are lacking in correlation with observed values, and the correlation coefficient is near zero.

These results mean that, although the MAE and RMSE are relatively low, and visually the predictions are near the observed values, our model had a bad performance. First, the model can not predict the variations, resulting in a low accuracy; its predictions are close to the observed values meaning a relative low MAE; for the same reason, RMSE is low, but higher because in the last points the observed values are further away from the average, which squared distance increases the RMSE value; finally, a low correlation means that the model is predicting noise.

### 4.2.3 Lag Features

Now the goal is to surpass the baseline. These lags features results show poor predictions. The MAE is about 6.71, RMSE is about 11.97, accuracy 0.54, and correlation 0.05. Although very poor results, they are slightly better than the baseline. The performance of the model is very similar to what is described about the baseline model in section 4.2.2.

As shown in Figure 4.9, the results of this approach show that the features used are not very descriptive of the data. The features used, corresponding to past data points, have similar F-scores. This score means that the XGBoost model built different trees without finding a feature that could be transverse to all, or a majority, data points. So it created several trees, with different features and those trees predicted

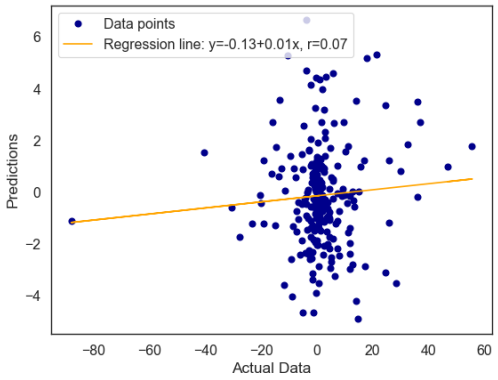



Figure 4.8: Regression Line of Tesla lag features results.

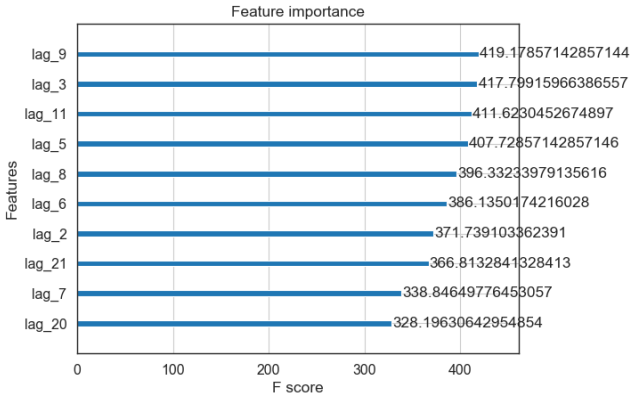



Figure 4.9: Tesla Lag Feature importance.

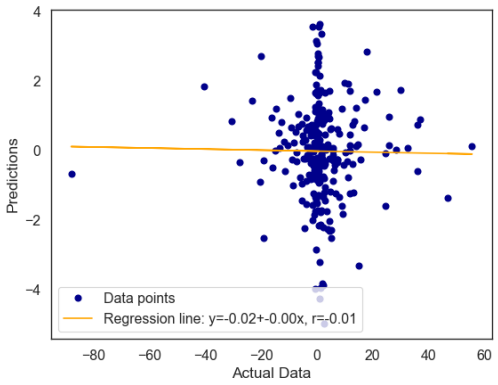



Figure 4.10: Regression Line of Tesla Time Features results.

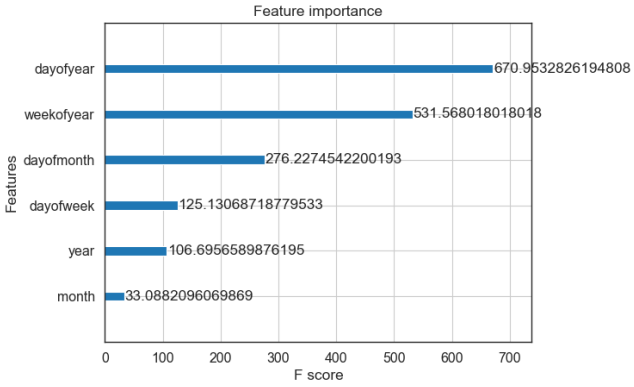



Figure 4.11: Tesla Time Features importance.

different values, resulting in a poor prediction.

The best lag feature is the 9th previous day, or lag_9, with a cover of 419. The second most used feature is the 3th previous day, or lag_3. Looking into the data, we see that the data is almost changing between a positive value and a negative value and the last quarter has a different behavior with a higher standard deviation. So the model trained with different data and while predicting on test data, the correlation pattern between those lag features and data could be changed, resulting in poor results.

### 4.2.4 Time Features

Time related features show poor predictions. The MAE is about 6.88, RMSE is about 12.03, accuracy 0.53 and correlation -0.01.

As shown in Figure 4.10, the test data has a lot of observed values close to zero. However, our model predicts values in a range of -4 and 4, leading to a high error and low correlation. In fact, correlation is negative, but almost zero, meaning the observed values and predicted values do not have any correlation. Also, for the last data points where we observe values more distant from the mean than in the first data points, the model predicts values close to zero, increasing the error and lowering the correlation.

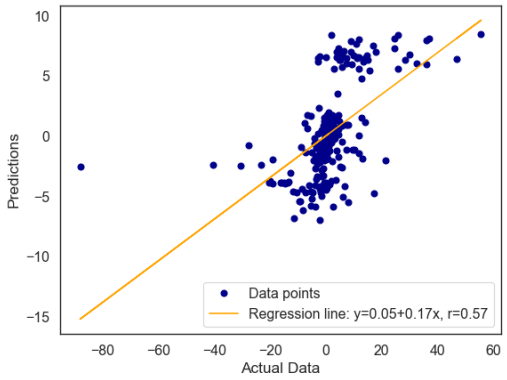



Figure 4.12: Regression Line of Tesla Aggregation Features results.

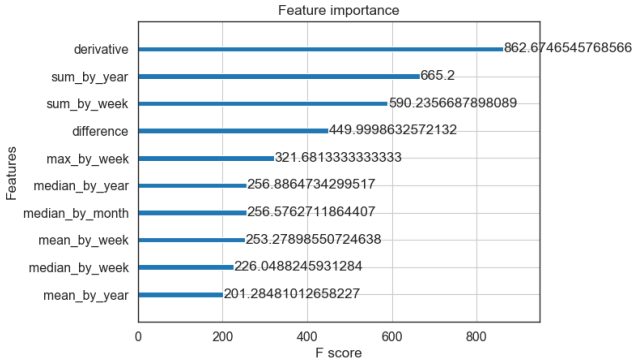



Figure 4.13: Tesla Aggregation Features importance.

These features are related to the day of the year, day of the week, as explained in section 3.5. The bad results could be explained by a change of pattern between the train data and test data. While in train data the model chooses the day of the year, year and the day of the week as features that most split the data, in test data these features could be not so important. ”Day_of_year” is the feature most used to split the training data, followed by ”week_of_year”.

### 4.2.5 Aggregation Features

Similar to Lag Features, aggregation features are correlated to past values, however they are transformations of those past values. MAE is about 5.44, RMSE is about 10.24, accuracy is 0.74 and correlation is 0.59. These results show a better performance than previous approaches and a significant increase in both accuracy and correlation when compared against our baseline.

From Figure 4.12, it is not clear a correlation between observed and predicted values. However, with these features the predicted values range widened to -15 and 10, still far away from the -80 to 60 range observed. This is mostly explained by two features: ”derivative” and ”difference”. ”Derivative” was used 7903 times in all trees to split training data, and ”difference” was used 4702 times. These two combined were used more than all the other together. Unlike ”derivative” and ”difference”, the other features are always the same inside their time range, so they are very poor, resulting in not so good results.

### 4.2.6 Smooth Features

The results of smooth features are the best ones. MAE is about 5.44, RMSE is about 10.24, accuracy 0.94, and correlation 0.78.

Looking at Figure 4.14, it is observable three different zones: the middle one, which corresponds to data values similar to the observed in the testing data; the upper zone, corresponding to the last data points in test data with positive values and they were predicted as ”10”; the lower zone, corresponding to the last data points in the test data with negative values and they were predicted as ”-10”. Although MAE and RMSE are not close to zero, actually it’s the lowest errors between all approaches and accuracy is

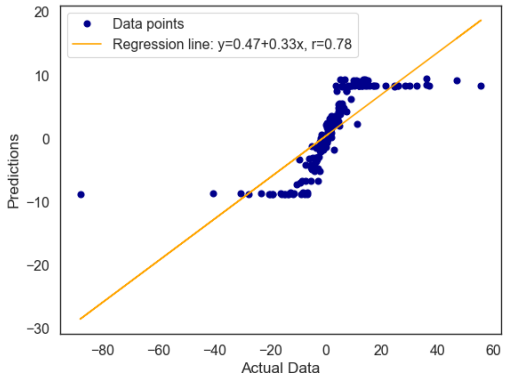



Figure 4.14: Regression Line of Tesla Smooth Features results.

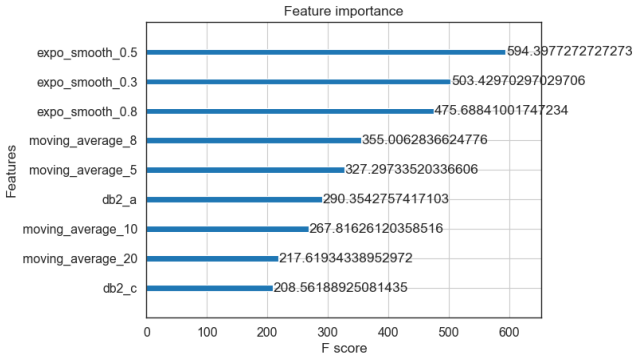



Figure 4.15: Tesla Smooth Features importance.

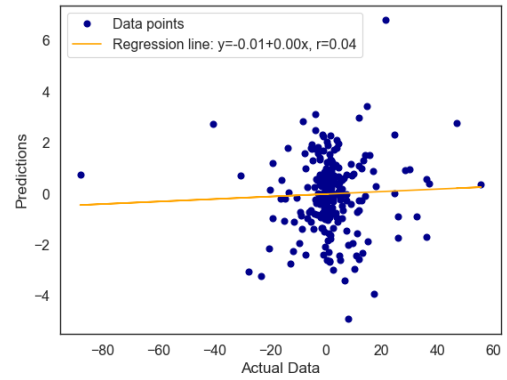



Figure 4.16: Regression Line of Tesla Composition Features results.

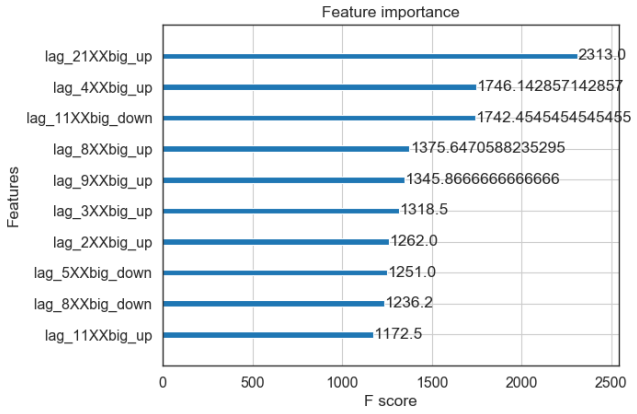



Figure 4.17: Tesla Composition Features importance.

close to 1, or 100%, meaning the model with those features work very well.

These features are moving averages with different time windows or different weights to past values in the case of the exponential moving average. This way, they are very descriptive of the most recent behaviour of the time series, similar to lags features. The best features are ”moving_average_8”, meaning moving average of 8 data points, and ”expo_smooth_0.5” meaning an exponential smoothing moving average with alpha of 0.5.

### 4.2.7 Composition Features

The results of composition features are far from the results expected. MAE is about 6.79, RMSE is about 11.99, MAPE is 98.65, accuracy is 0.52, and correlation 0.05. The observed values and predicted values do not have correlation. In Figure 4.16 we see a random dispersion of the points. The range of the predicted values are between -4 and 6, although the observed values are in a range between -80 and 60. The top features are all related to lag features.This can explain the bad results, since Lag features obtained similar errors. Also, composition features have more features than the other sets of analysis. This could lead to overfit and increase the complexity of the model.

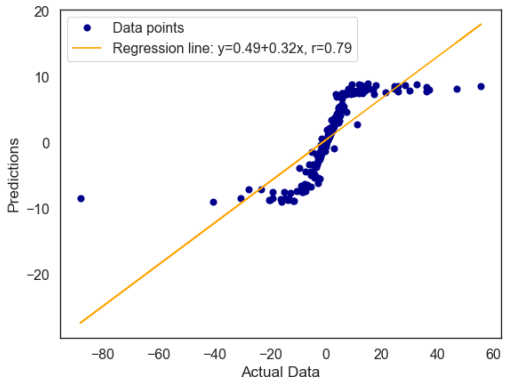



Figure 4.18: Regression Line of Tesla Final model results.

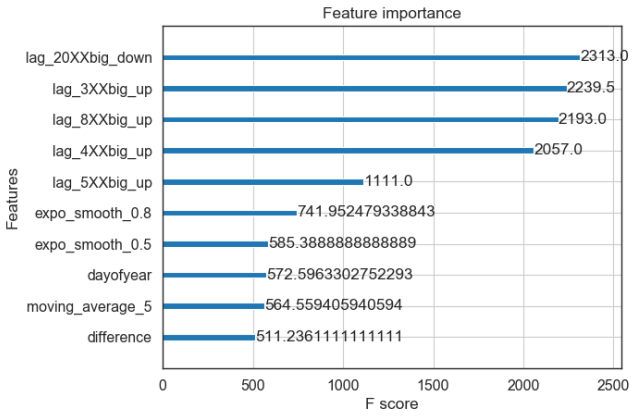



Figure 4.19: Tesla Final Model Features importance.

### 4.2.8 Final Model

The final model joins all features selected from each approach. The final result is not better than the smooth features approach. MAE is about 3.96, RMSE is about 9.30, accuracy is 0.89, and correlation 0.72.

Looking at Figure 4.18, it's observable the same pattern as in Figure 4.14, but this time more scattered.

The best features chosen, shown on Figure 4.19, are composition features ans smooth features, that are taken from the two best set of features.

In Figures 4.20, 4.22, 4.21, 4.23, 4.24, are visualized the results for each metric used in each step of analysis. The final model has a superior performance compared to the baseline, the observed values are correlated to predicted values, which do not happen on the baseline, and the final model is able to predict more correctly if the next data point will be higher or lower than the actual one. This can be very important in the context of stock forecast, where the user could know if the stock value will increase or decrease and that way do a more informative action.

## 4.3 Nio Stock

The Nio stock is made up of 505 data points, with a low standard deviation of about 3.7 and is not stationary. On Figure 4.25 is shown the daily values at the closing market time, from Sep 12, 2018 to Sep 14, 2020. Visually, the stock value behaviour can be split into three periods of time.

The first period goes from Sep 12, 2018 to Mar 7, 2019. The stock opens negotiating at 6.6$, and quickly goes up to 11.6$, a 75% appreciation, in the second day on the market. This value will be the maximum until Jul 7, 2020 when it reaches the price of 13.22$. This period is marked by a high volatility with significantly changes day per day.

The second period ends on May 22, 2020. During this period, the price decreases, devaluing 81%, when it reaches 1.32$ on Oct 1, 2019. This is the historical minimum. After that, the trend is positive, and the stock appreciate 166%.

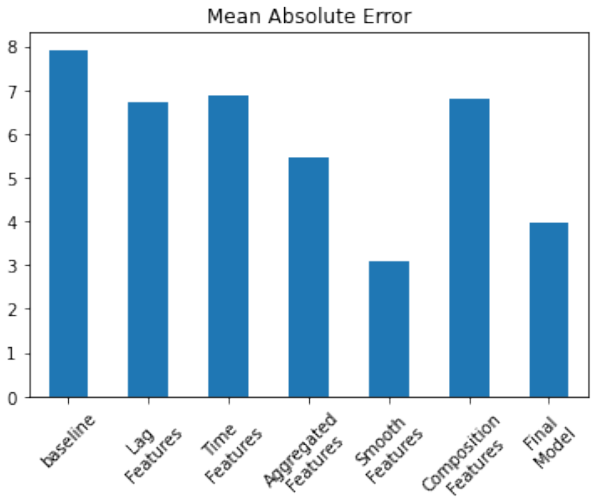



Figure 4.20: Mean Absolute Error of each step of the Tesla case study analysis.

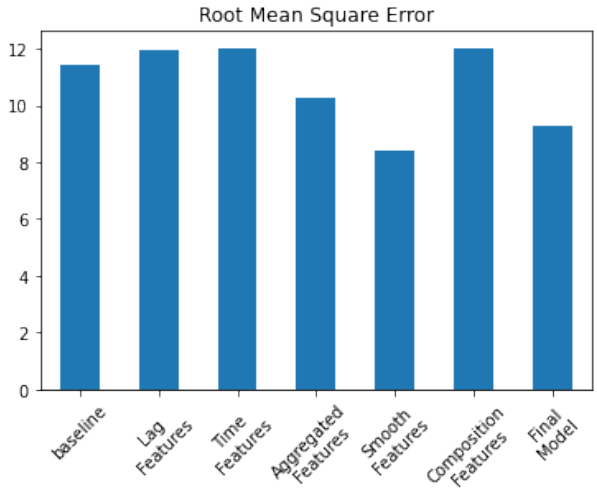



Figure 4.21: Root Mean Square Error of each step of the Tesla case study analysis.

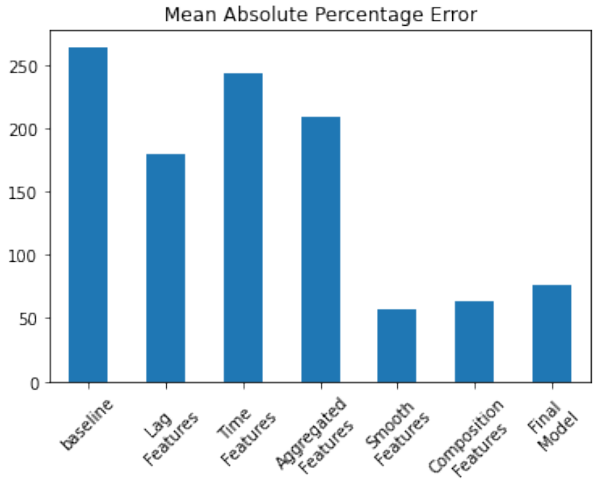



Figure 4.22: Mean Absolute Percentage Error of each step of the Tesla case study analysis.

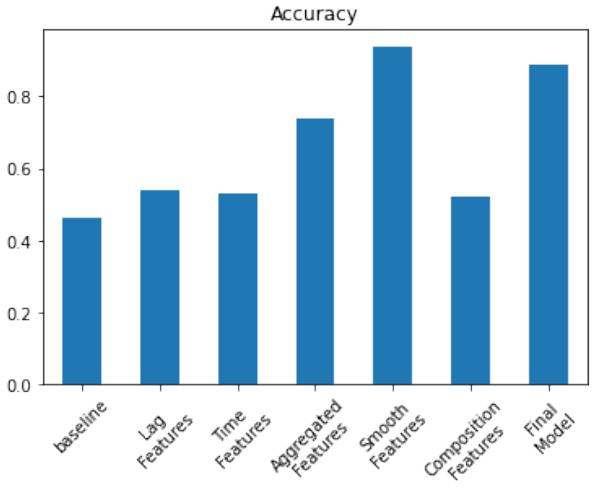



Figure 4.23: Accuracy of each step of the Tesla case study analysis.

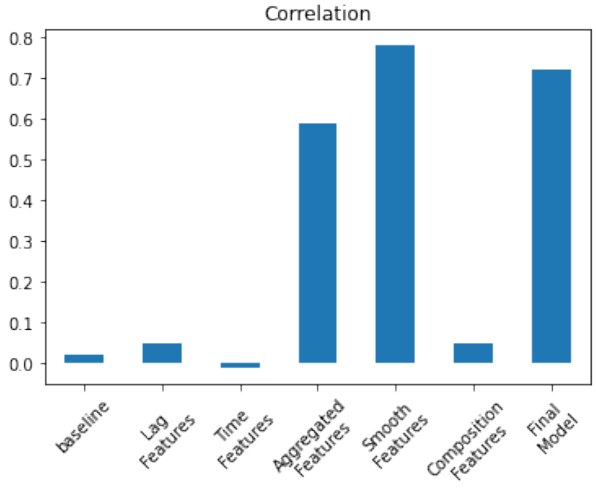



Figure 4.24: Correlation results of each step of the Tesla case study analysis.

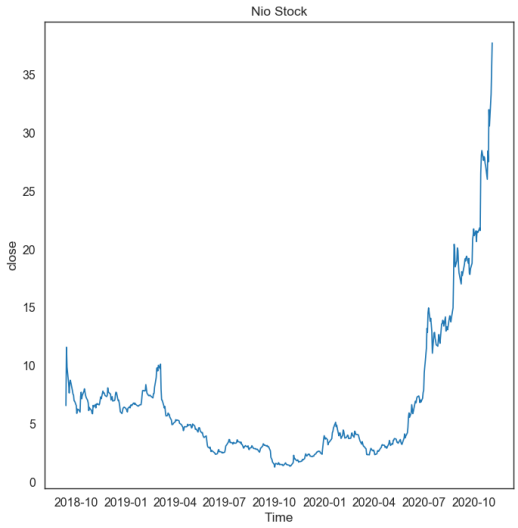



Figure 4.25: Daily close value for Nio stock between Sep 12, 2018 and Sep 14, 2020.

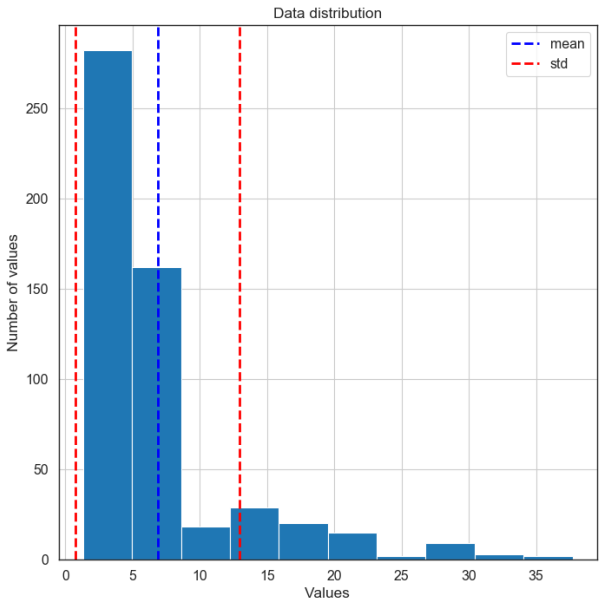



Figure 4.26: Distribution of Nio stock values in bins of size 5.

The third period has a notable upward trend. The stock appreciate 1050%, reaching 37.7$. From Figure 4.25, it is visible a pattern that can explain future values. That pattern consists in a appreciation of 7$ in one or two days, followed by a small period of devaluation. This stock data belongs to an Electric Vehicle Manufacture based in China, and has a similar performance to Tesla, another Electric Vehicle manufacture.

Finally, Figure 4.26 illustrates the value distribution. There are almost the same number of values between 0 and 5 than values between 5 and 40. And values between 0 and 10 compose 90% of the all data. It is expected that this conditions will have an high impact when making predictions.

This dataset is relative small, with only 505 data points. This is an important aspect to take in consideration in the analysis that follows. Also, the test period corresponds to the one with higher changes that aren't observed before.

### 4.3.1 Data profiling

The framework starts by analyzing the stationary property. The Augmented Dickey-Fuller test informed that the data is non stationary, so it performs a first-order differencing. The time series after data profiling is shown in Figure 4.27, where we can see a different time series. The new values are more bonded between -10 and 10. From Figure 4.28 we see that mean is close to zero, and the values outside the range -10 to 10, correspond to the first and last period, specially the last one when the stock grew very fast.

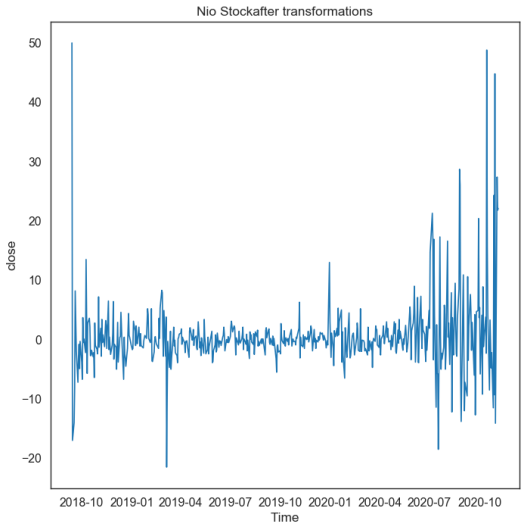



Figure 4.27: Nio stock data after data profiling.

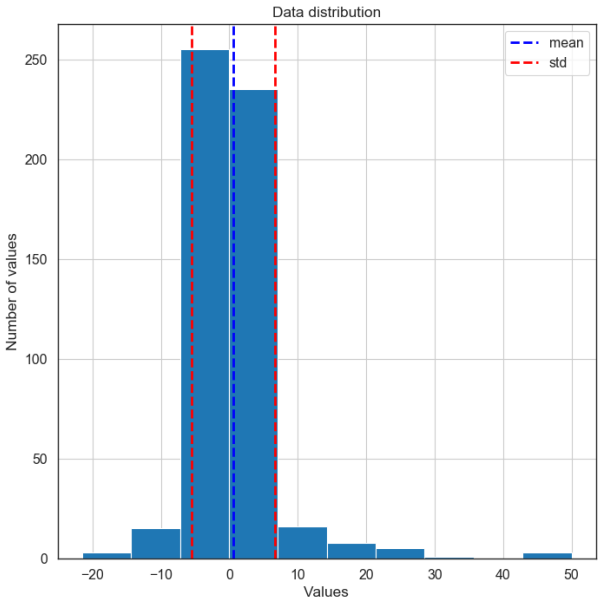



Figure 4.28: Distribution of values after data profiling in bins of size 10.

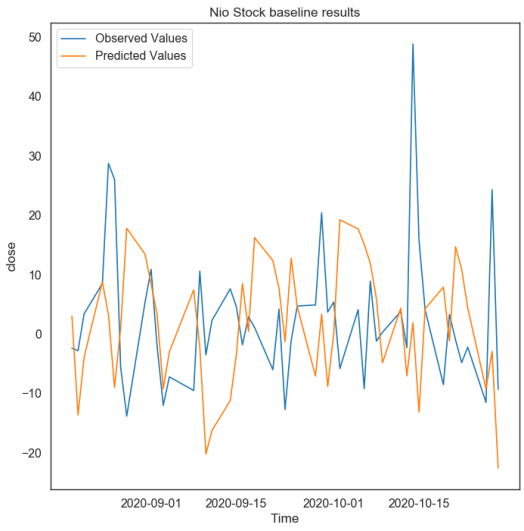



Figure 4.29: Baseline results for Nio stock: on blue the observed values and on orange the predicted values.

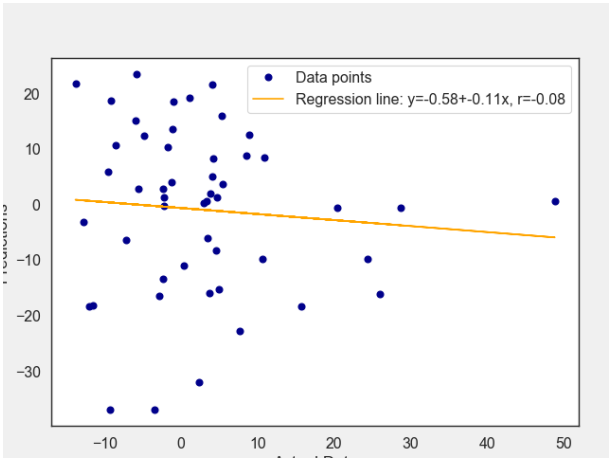



Figure 4.30: Regression Line of baseline results.

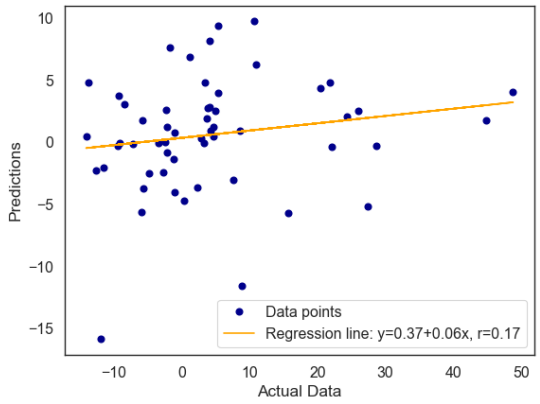



Figure 4.31: Regression Line of lags results.

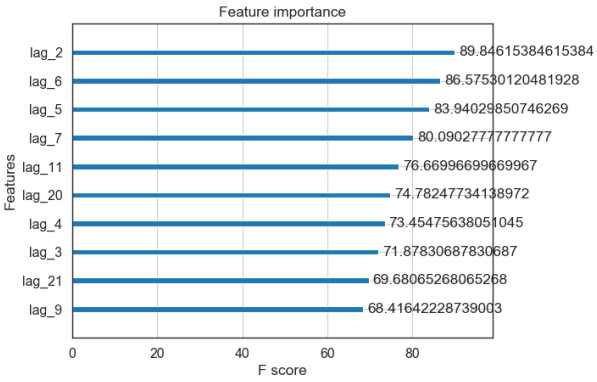



Figure 4.32: Lag Feature Importance.

### 4.3.2 Baseline

The baseline results, or the LSTM results, show poor predictions. The MAE is about 12.86, RMSE of 16.48 and MAPE is 435.07. Accuracy is near 0.46, which means that the model only could predict correctly less than half of the positive variations. From Figure 4.29, we observe that the predicted values are relative well bounded in the range of observed values. A closer look will notice that in the observed time series we have a high value followed by a low value, resulting in consecutive ups and downs. However, in the predicted time series we have two ups or two downs consecutive before changing the trend. This result in a low accuracy and high errors. In Figure 4.30, the points correspond to the tuple $(y_{observed}, y_{predicted})$ and the orange line corresponds to an regression line. The results are lacking correlation with the observed values, and the correlation coefficient is near zero. If the model would had good results, the data points would be sat on the bisector resulting on a equation near $y = x$.

### 4.3.3 Lag Features

These lag features show poor predictions. The MAE is about 10.09, RMSE is about 14.09, MAPE is 163.78, accuracy 0.47, and correlation 0.08. Although very poor results, they are slightly better than the baseline.

As shown in Figure 4.31, the predictions are very disperse without a clear correlation between observed and predicted. Analysing the features used that corresponds to previous observations, we can't name a feature or a set of features that can describe the data. The score of each feature shown in Figure 4.32 indicate that lag_2 was the most used feature to split the data. Since XGBoost built different trees each time we run the algorithm, these feature scores can have different values. We ran the procedure several time and the top features, namely lag_2, lag_6 and lag_5, always scored high.

In the baseline analysis, we saw that the observed time series has consecutive ups and downs. So, an even number of previous observations will help to notice that pattern. And since in the last period we have a very high upwards trend, recently previous observations will help to notice the upward trend, than the older observation, when the value was lower. Finally, lag_11 and lag_20 play and important role

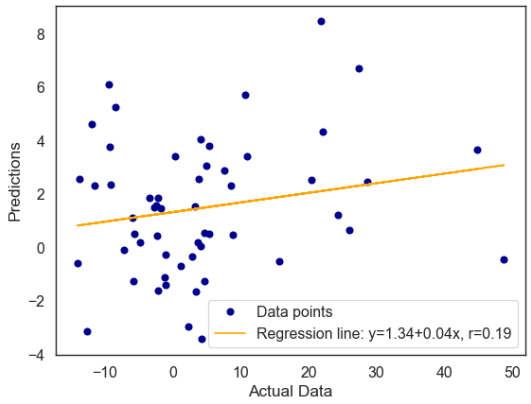



Figure 4.33: Regression Line of time features results.

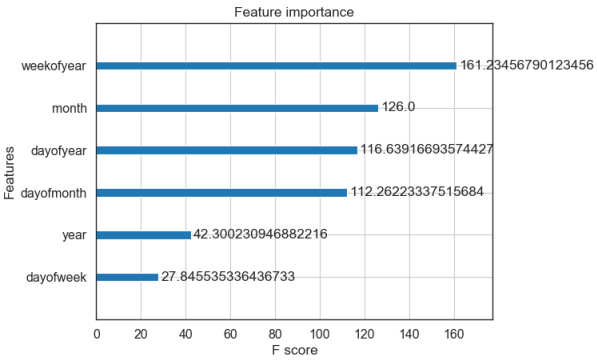



Figure 4.34: Time Features Importance.

to help discover the patterns in the last period.

### 4.3.4 Time Features

Time features results show poor predictions. The MAE is about 9.43, RMSE is about 13.44, MAPE is 120.10, accuracy 0.56 and correlation 0.19. The results are better than baseline and Lag Features. This means that the time of buying the Nio Stock is more important that previous observations.

Again, we can apply the analysis made about Lag Feature and apply here. As shown in Figure 4.33, the data points are too disperse to say that there is a correlation between observed values and predicted values.

These features are related to the day of the year, day of the week, as explained in section 3.5. The feature with the highest score is ”week of the year”. Since we have a small dataset, with only 3 years of data, it wouldn’t be clear that this feature can split the most data points.

### 4.3.5 Aggregation Features

Similar to Lag Features, aggregation features are correlated to past values, however they are transformations of those past values. MAE achieved about 8.11, RMSE is about 11.22, MAPE is 113.62, accuracy is 0.67 and correlation is 0.60. These results show a better performance than previous approaches and a significant increase in both accuracy and correlation when compared against our baseline. In Tesla analysis similar results were obtained.

Now we have a clear correlation between actual data and predicted values. In Figure 4.35, the distribution of the points follow the regression line. Also, we can see some outliers with the predicted value of 30. This can be result of a limitation of the model. Since XGBoost builds trees, the model can’t learn values outside the range of the training set.

”Derivative” is the feature with highest score. It is a very representative feature, it always scores higher if we ran the procedure several times. Features aggregated by weeks follow the top. ”Mean

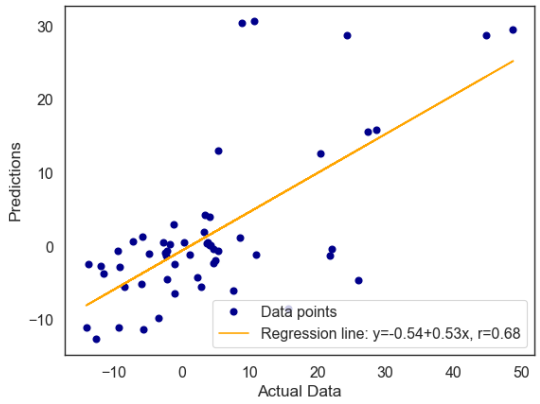



Figure 4.35: Regression Line of aggregation features results.

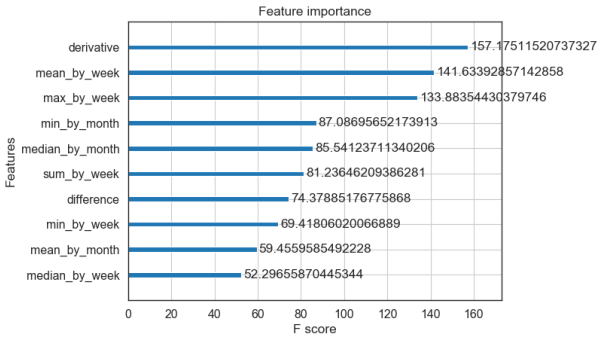



Figure 4.36: Aggregation Features Importance.

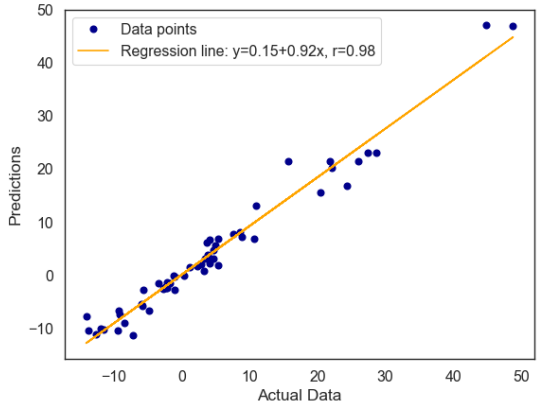



Figure 4.37: Regression Line of smooth features results.

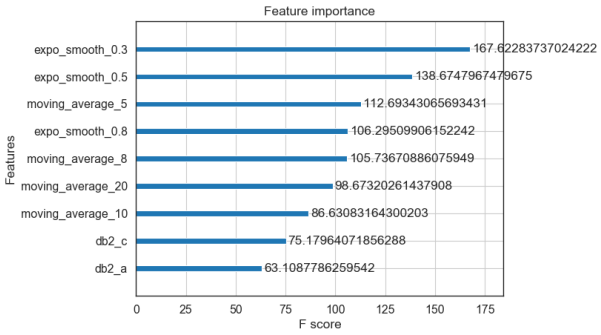



Figure 4.38: Smooth Features Importance.

by week" and "max by week" can help describe the patterns mentioned before. The rest of features are aggregations to the values of weeks and months. Days are not present since we have only one observation for each day.

### 4.3.6 Smooth Features

The results of smooth features are the best ones. MAE is about 2.84, RMSE is about 5.65, MAPE is 31.25, accuracy is 0.93, and correlation 0.93.

From Figure 4.37, the data points are over the regression line. Resulting in a high correlation and small errors. In Figure 4.38, we have the top scoring features. The three Exponential smoothing features used rank in the top, meaning that the most recent values describe better than previous ones. This is in line with the results from Aggregation features, where the top ones are related to small unities of time. Also, it is according to Lag Features, where the most recent observations scored higher in feature importance.

Followed by Exponential moving average features, we have normal moving averages. The first one

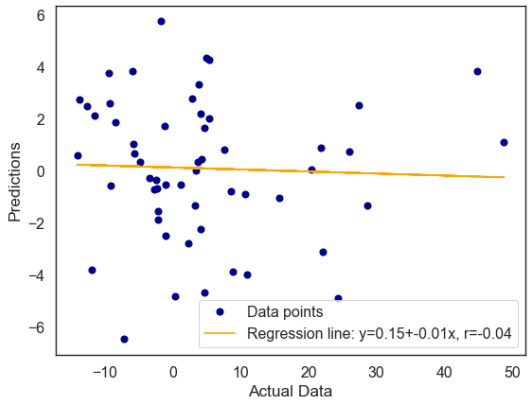



Figure 4.39: Regression Line of composition features results.

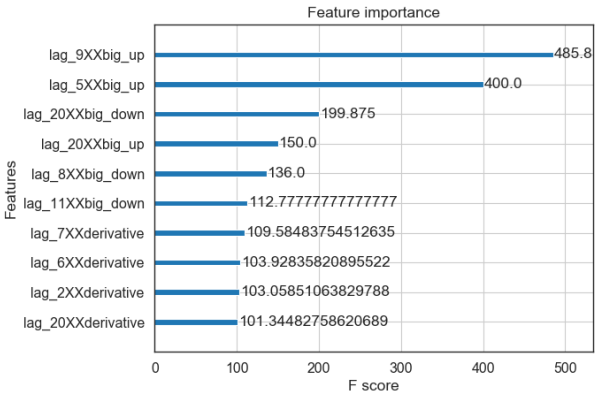



Figure 4.40: Composition Features Importance.

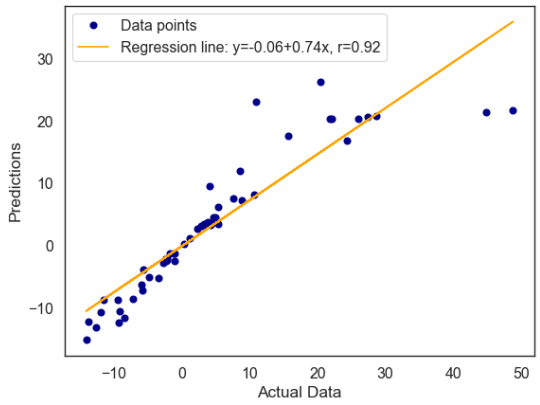



Figure 4.41: Regression Line of the final model.

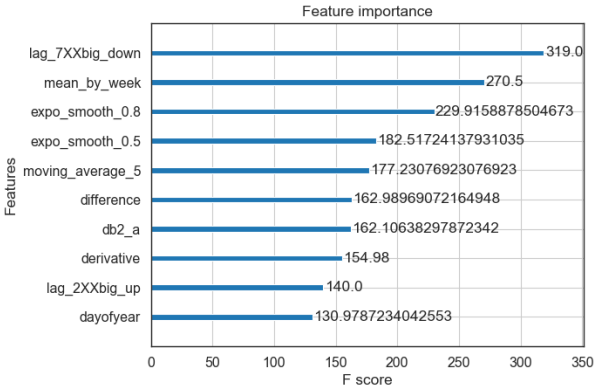



Figure 4.42: Final Model Features Importance.

to appear in the rank is the one with lowest range, of 8. Finally, features extracted from wavelet transformations have a lower score.

### 4.3.7 Composition Features

The results of composition features are far from the results expected. MAE is about 10.32, RMSE is about 14.22, MAPE is 129.40, accuracy is 0.40, and correlation -0.04.

The observed values and predicted values do not have correlation. In Figure 4.39 we see a random dispersion of the points. The range of the predicted values are between -6 and 6, although the observed values are in a range between -10 and 50. The top features are all related to lag features. This can explain the bad results, since Lags features obtained similar errors. Also, composition features have more features than the other sets of analysis. This could lead to overfit and increase the complexity of the model.

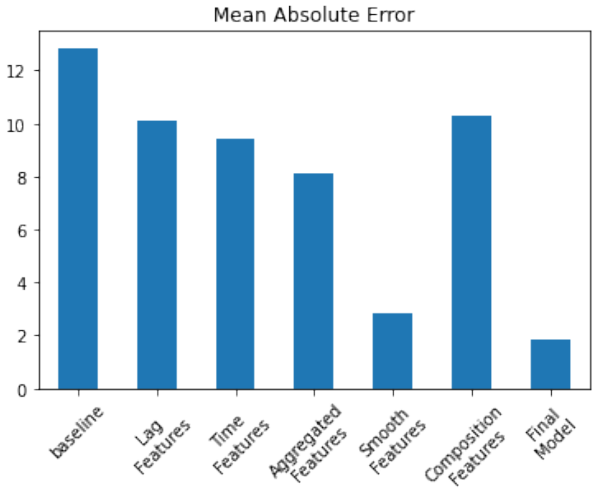



Figure 4.43: Mean Absolute Error of each step of the Nio case study analysis.

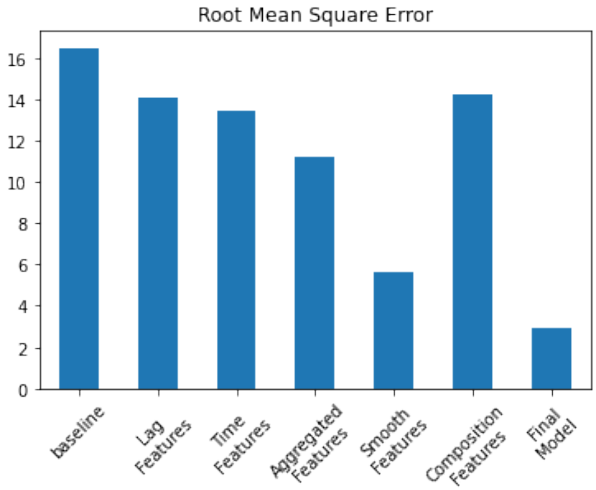



Figure 4.44: Root Mean Square Error of each step of the Nio case study analysis.

### 4.3.8 Final Model

The final model joins all features selected from each approach. The final result has the best performance above all. MAE is about 1.87, RMSE is about 2.93, MAPE is 19.02, accuracy is 0.98, and correlation 0.92.

Looking at Figure 4.37 and Figure 4.41, it's observable that they have a similar look, however in the final model the points with observed values above 20 are all predicted with a value close to 20.

The best features chosen, shown in Figure 4.42, are picked from all sets of study. Meaning that in spite of Smooth Features results have an high accuracy, high correlation and low errors, the other features could be important too. "lag_7 composed with big down" is the most descriptive feature, followed by "mean by week" and "exponential smoothing with alpha of 0.8". In the analysis of smooth features, we said that an exponential moving average with a lower alpha describes the data better than an higher alpha. Here we see the opposite, meaning that is the set of features that matter and not only one feature by itself.

In Figures 4.43, 4.45, 4.44, 4.46, 4.47, are visualized the results for each metric used in each step of analysis. The final model has a superior performance compared to the baseline, the observed values are correlated to predicted values, which do not happen on the baseline, and the final model is able to predict more correctly if the next data point will be higher or lower than the actual one. This can be very important in the context of stock forecast, where the user could know if the stock value will increase or decrease and that way do a more informative action.

## 4.4 SP500 index

The SP500 index is made up of 6957 data points, with a high standard deviation of about 68.23 and is not stationary. On Figure 4.48 is shown the daily values at the closing market time, from Jan 29, 1993 to Sep 14, 2020. Visually, the stock value behaviour can be split into two periods of time.

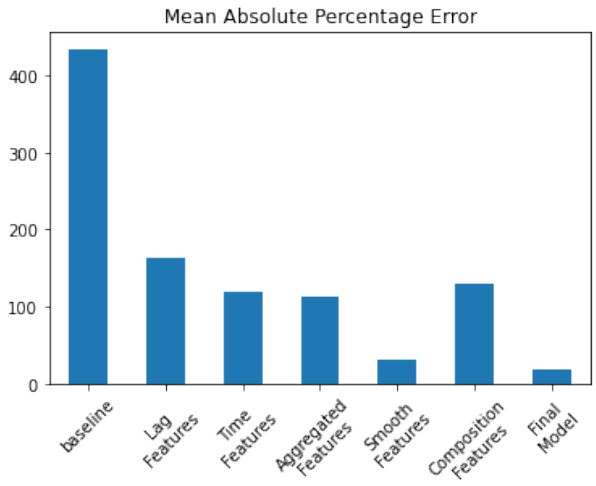



Figure 4.45: Mean Absolute Percentage Error of each step of the Nio case study analysis.

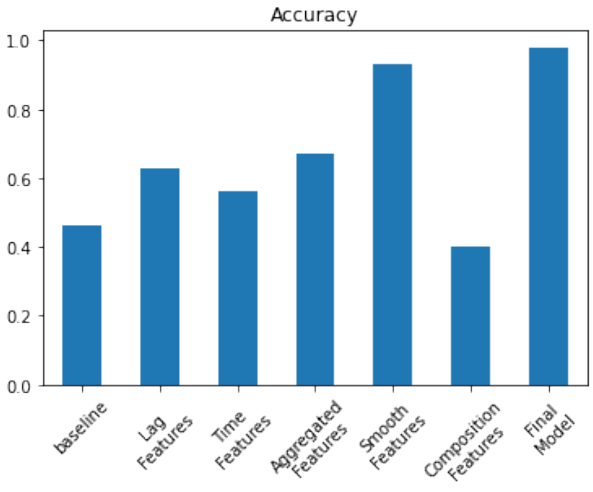



Figure 4.46: Accuracy of each step of the Nio case study analysis.

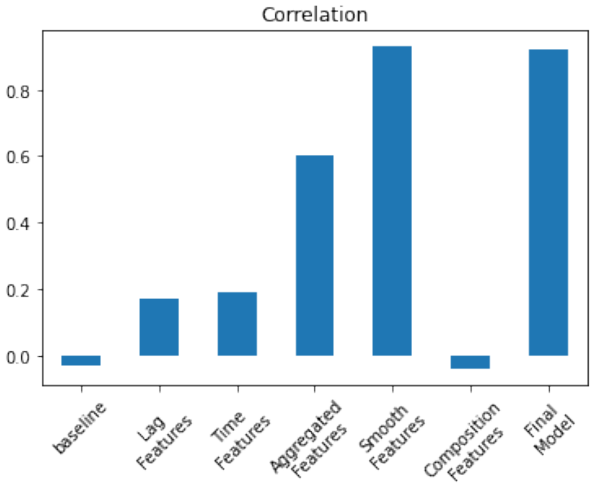



Figure 4.47: Correlation results of each step of the Nio case study analysis.

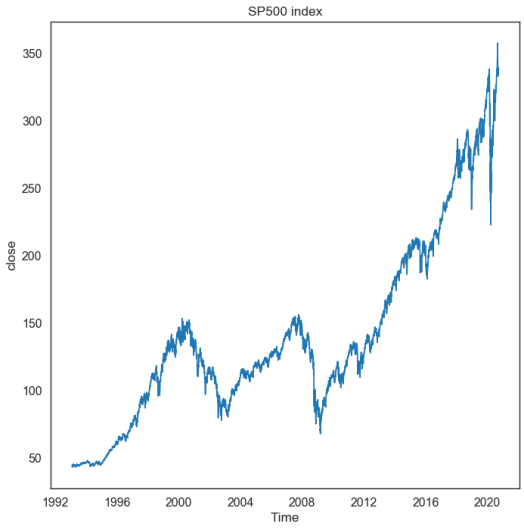



Figure 4.48: Daily close value for SP500 index between Jan 29, 1993 and Sep 14, 2020.

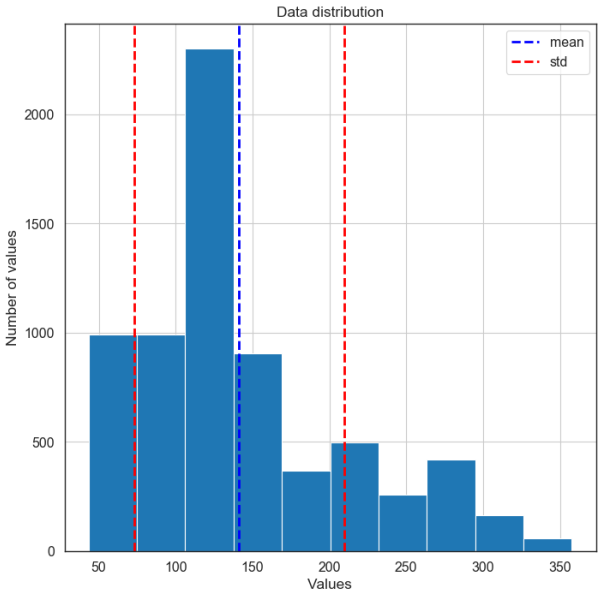



Figure 4.49: Distribution of SP500 index values in bins of size 50.

The first period goes from Jan 29, 1993 to Mar 9, 2009. The index value opens negotiating at 44, and gradually goes up to 154, a 250% appreciation, after 7 years. Then it depreciates 55% along 2 years. The pattern repeats once more.

The second period has a notable upward trend. The index value appreciate 500%, reaching 360. From Figure 4.48.

Finally, Figure 4.49 illustrates the value distribution. The majority of values are in a range between 50 and 150. Values between the range of 50 and 100 have more than double than any other range.

This dataset is big, with 6957 data points. This will be more informative than other dataset with a smaller size. This way, the model can learn better the trend and patterns. Also, the last period is the most interesting and it corresponds to one third of the all dataset. In the Nio stock analysis, we saw that the most interesting period only corresponded to 1/10 of the all dataset. Of course, this factor will impact the model performance.

### 4.4.1 Data profiling

The framework starts by analyzing the stationary property. The Augmented Dickey-Fuller test informed that the data is non stationary, so it performs a first-order differencing. The time series after data profiling is shown in Figure 4.50, where we can see a different time series. The new values are more bonded between -100 and 100, with some exceptions. From Figure 4.51 we see that mean is close to zero, and the distribution fit a normal distribution.

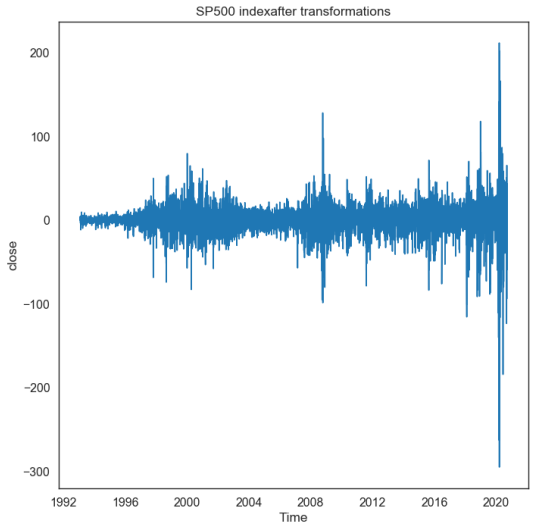



Figure 4.50: SP500 index data after data profiling.

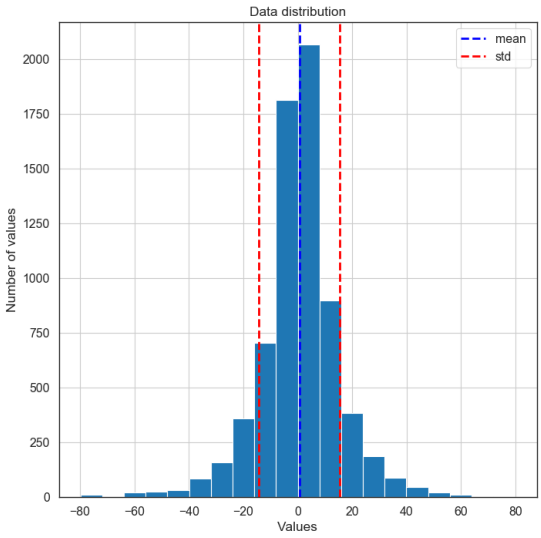



Figure 4.51: Distribution of values after data profiling in bins of size 10.

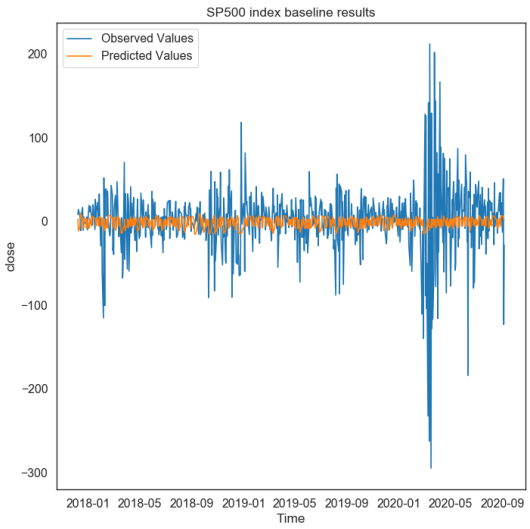



Figure 4.52: Baseline results for SP500 index: on blue the observed values and on orange the predicted values.

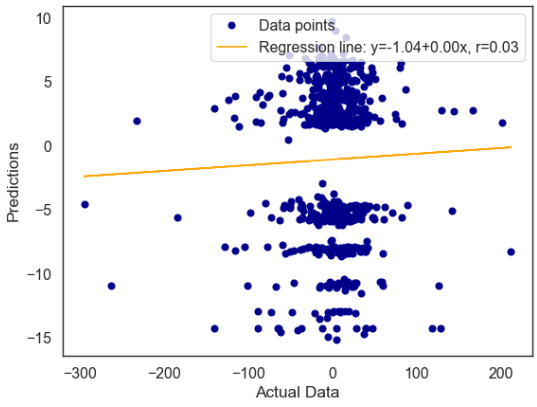



Figure 4.53: Regression Line of baseline results.

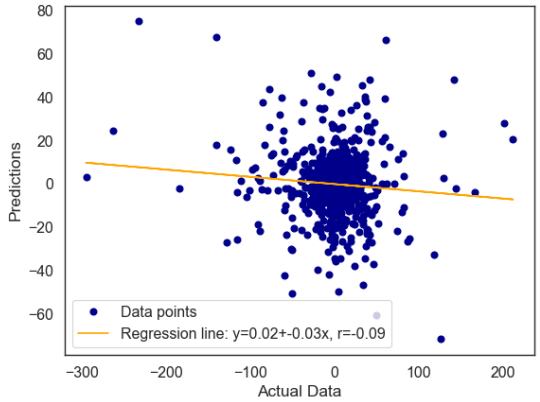



Figure 4.54: SPY500 index: Regression Line of lagstance.

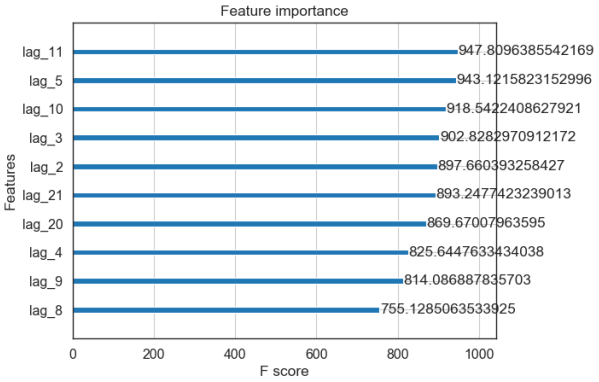



Figure 4.55: SPY500 index: Lag Feature Importance results.

### 4.4.2 Baseline

The baseline results, or the LSTM results, show poor predictions. The MAE is about 25.92, RMSE of 40.50 and MAPE is 435.07. Accuracy is near 0.48, which means that the model only could predict correctly less than half of the positive variations. From Figure 4.52, we observe that the predicted values are relative well bounded in the range -10 to 0 and between 5 and 15. We can say that the model was not able to predict value outside this range and because of that it has a bad performance. Hovewer, if we take a look to Figure 4.51, we see that the most of the value are in these ranges. From Figure 4.53, we see that results are lacking correlation with the observed values, and the correlation coefficient is near zero.

### 4.4.3 Lag Features

These lags features results show poor predictions The MAE is about 27.49, RMSE is about 44.14, MAPE is 400.10, accuracy 0.48, and correlation -0.08.

As shown in Figure 4.54, the predictions are very disperse without a clear correlation between observed and predicted values. Analysing the features used that corresponds to previous observations, we can't name a feature or a set of features that can describe the data. The score of each feature shown in Figure 4.55 indicate that lag_11 was the most used feature to split the data, tied with lag_5.

### 4.4.4 Time Features

Time features results show poor predictions. The MAE is about 26.29, RMSE is about 41.90, MAPE is 362.67, accuracy 0.52 and correlation -0.02. Again, we can apply the analysis made about Lag Feature and apply here. As shown in Figure 4.56, the data points are too disperse to say that there is a correlation between observed values and predicted values.

These features are related to the day of the year, day of the week, etc, as explained in section 3.5. The feature with the highest score is "week of the year".

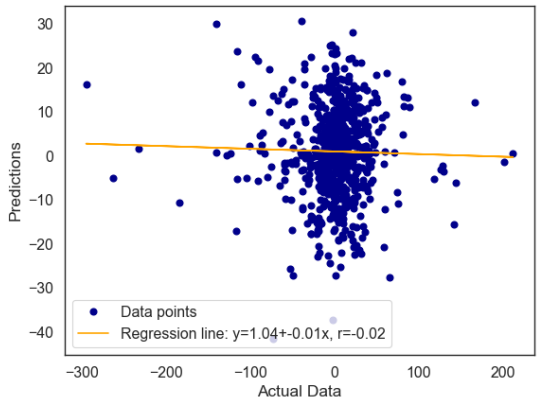



Figure 4.56: SPY500 index: Regression Line of time features results.

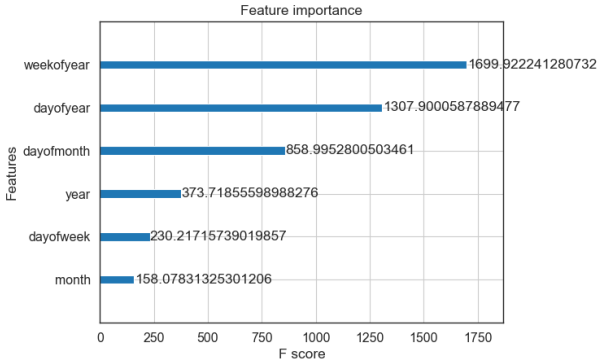



Figure 4.57: SPY500 index: Time Features Importance.

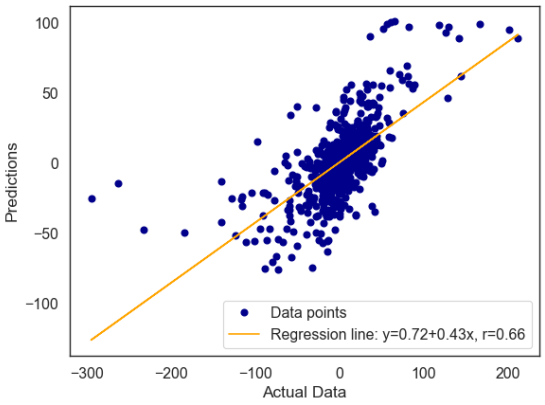



Figure 4.58: Regression Line of aggregation features results.

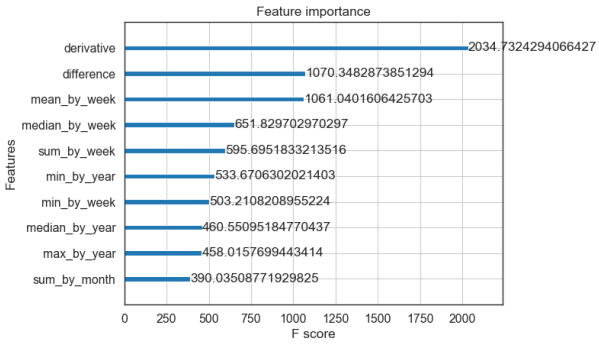



Figure 4.59: Aggregation Features Importance.

### 4.4.5 Aggregation Features

Similar to Lag Features, aggregation features are correlated to past values, however they are transformations of those past values. MAE is about 19.14, RMSE is about 30.63, MAPE is 270.64, accuracy is 0.73 and correlation is 0.65. These results show a better performance than previous approaches and a significant increase in both accuracy and correlation when compared against our baseline.

Now we have a clear correlation between actual data and predicted values. In Figure 4.58, the distribution of the points follow the regression line.

”Derivative” is again (as in the Tesla and Nio case study) the feature with highest score. It is a very representative feature, because it double the score of the second most used feature, ”difference”. Features aggregated by weeks follow the top. ”Mean by week” and ”max by week” can help describe the patterns mentioned before. The rest of features are aggregations to the values of weeks and months. Days are not present since we have only one observation for each day.

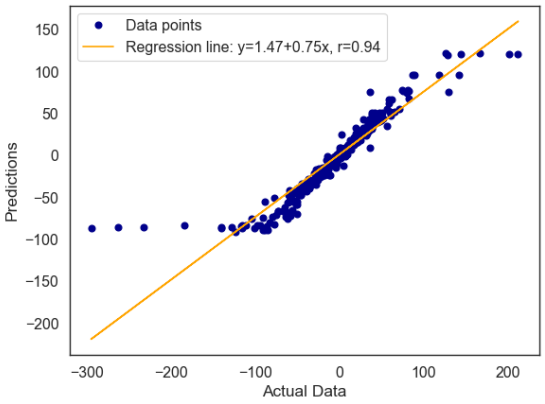



Figure 4.60: Regression Line of smooth features results.

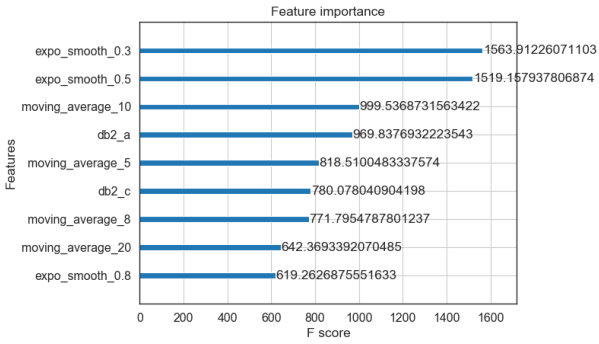



Figure 4.61: Smooth Features Importance.

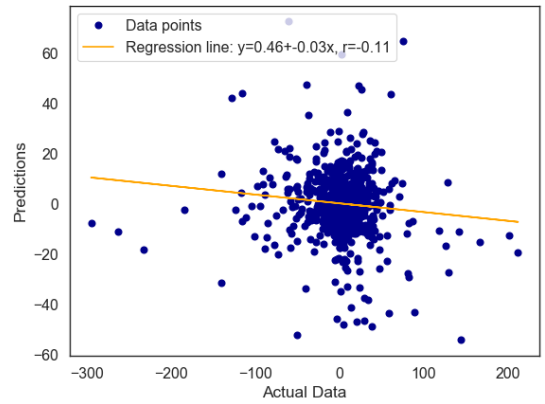



Figure 4.62: Regression Line of composition features results.

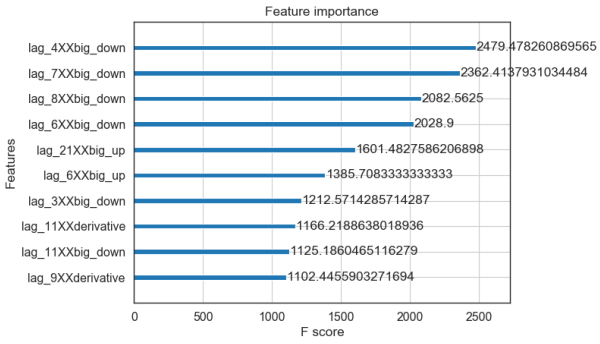



Figure 4.63: Composition Features Importance.

### 4.4.6 Smooth Features

The results of smooth features are the best ones. MAE is about 4.95, RMSE is about 15.28, MAPE is 32.48, accuracy is 0.96, and correlation 0.93.

From Figure 4.60, the data points are over the regression line. Resulting in a high correlation and small errors. In Figure 4.61, we have the top scoring features. The three Exponential smoothing features used rank in the top, meaning that the most recent values describe better than previous ones. This is in line with the results from Aggregation features, where the top ones are related to small unities of time.

Followed by Exponential moving average features, we have normal moving averages. The first one to appear in the rank is the one with lowest range, of 5. Finally, features extracted from wavelet transformations have a tied score with the rest of moving average features.

### 4.4.7 Composition Features

The results of composition features are far from the results expected. MAE is about 27.40, RMSE is about 43.22, MAPE is 98.65, accuracy is 0.49, and correlation -0.05.

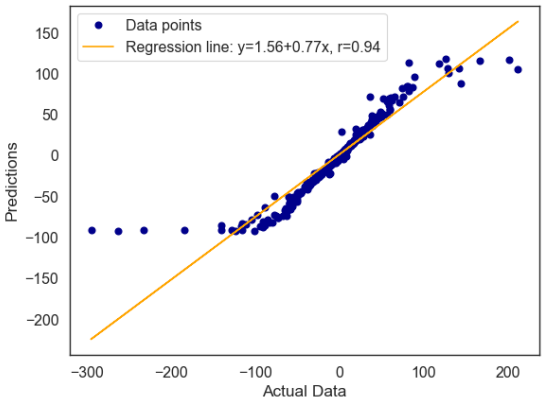



Figure 4.64: Regression Line of the final model.

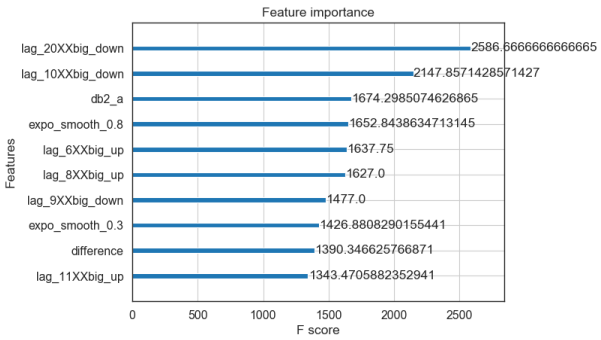



Figure 4.65: Final Model Features Importance.

The observed values and predicted values do not have correlation. In Figure 4.62 we see a random dispersion of the points. The range of the predicted values are between -60 and 60, although the observed values are in a range between -300 and 200. The top features are all related to lag features. This can explain the bad results, since Lag features obtained similar errors. Also, composition features have more features than the other sets of analysis. This could lead to overfit and increase the complexity of the model.

### 4.4.8 Final Model

The final model joins all features selected from each approach. The final result has the best performance above all. MAE is about 3.37, RMSE is about 14.46, MAPE is 21.05, accuracy is 0.98, and correlation 0.94.

Looking at Figure 4.60 and Figure 4.64, it's observable that they have a similar look, however in the final model the points with observed values below -100 are all predicted with a value close to -100.

The best features chosen, shown in Figure 4.65, are picked from all sets of study. Meaning that in spite of Smooth Features results have an high accuracy, high correlation and low errors, the other features could be important too. "lag_20 composed with big down" is the most descriptive feature, followed by another composed feature, "lag_10 composed with big down". In the analysis of smooth features, we said that an exponential moving average with a lower alpha describes the data better than an higher alpha. Here we see the opposite, meaning that is the set of features that matter and not only one feature by itself.

In Figures 4.66, 4.68, 4.67, 4.69, 4.70, are visualized the results for each metric used in each step of analysis. The final model has a superior performance compared to the baseline, the observed values are correlated to predicted values, which do not happen on the baseline, and the final model is able to predict more correctly if the next data point will be higher or lower than the actual one. This can be very important in the context of stock forecast, where the user could know if the stock value will increase or decrease and that way do a more informative action.

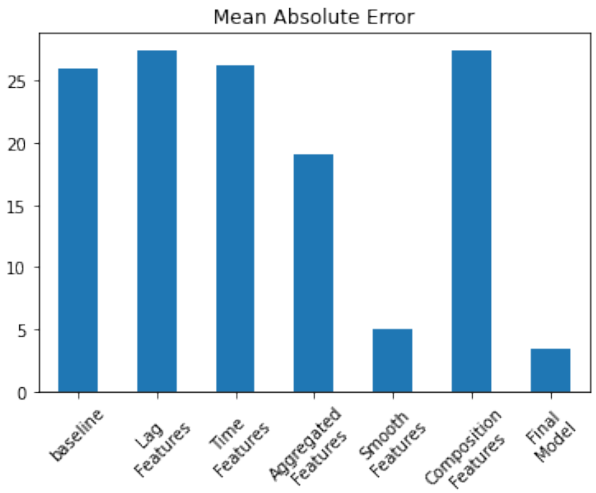



Figure 4.66: Mean Absolute Error of each step of the SP500 index case study analysis.

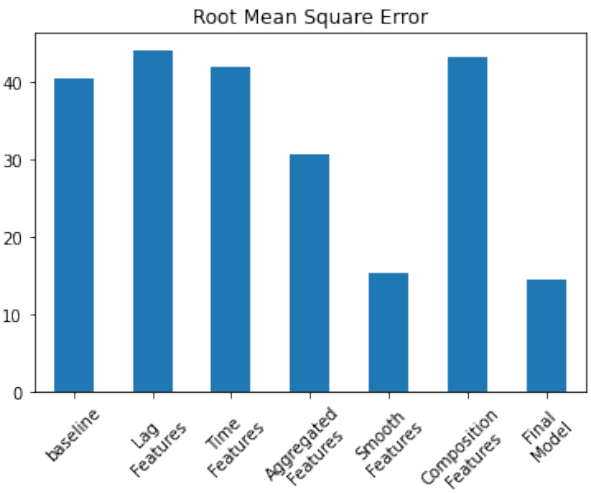



Figure 4.67: Root Mean Square Error of each step of the SP500 index case study analysis.

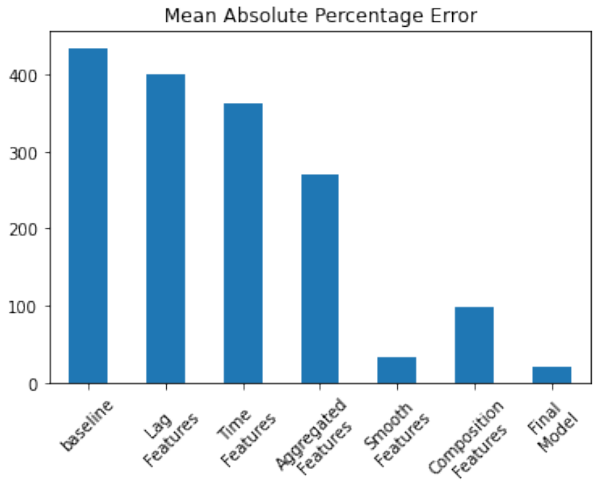



Figure 4.68: Mean Absolute Percentage Error of each step of the SP500 index case study analysis.

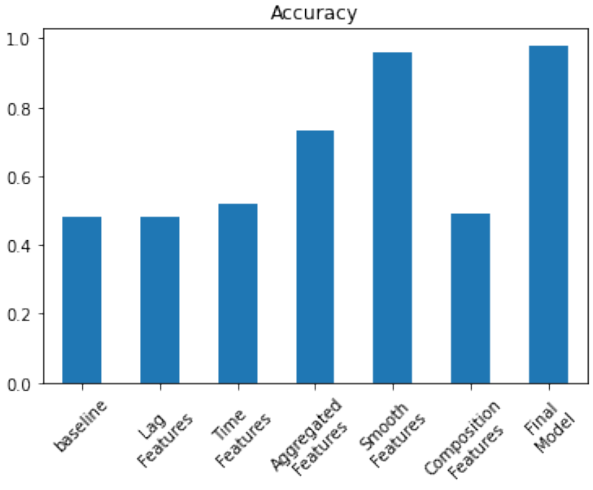



Figure 4.69: Accuracy of each step of the SP500 index case study analysis.

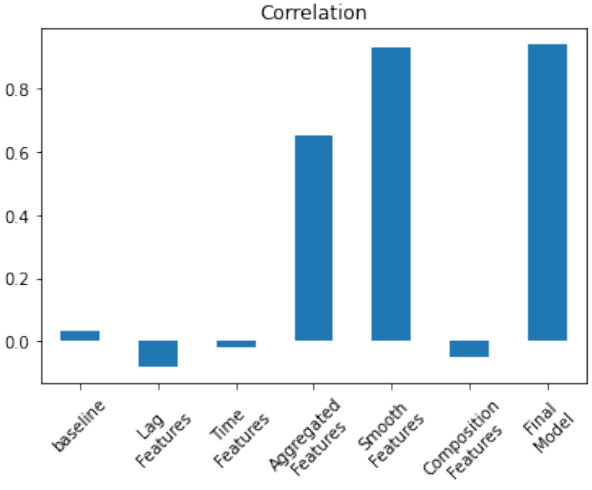



Figure 4.70: Correlation results of each step of the SP500 index case study analysis.

# Chapter 5

# Conclusions

## 5.1 Achievements

The existing tools to Automate the Data Science Process are very focused on tabular data not considering the temporal relations among observations. Some example are AUTO-Weka [44], AUTO-Sklearn [20], TPOT [38], Hyperopt-Sklearn [5]. None of these frameworks accept time series data.

Time series is a specially type of data where the value at time $t$ is predicted with the time values before $t$, and without the values after that. In this work we survey the state of the art of tools that allow to create a Data Science pipeline as black-box problem, and the state of the art algorithms to analyse time series. In time series analysis, Feature Engineering is the most complex and difficult task for data scientist. The options to generate feature are a lot and it is not possible to test all. We presented packages that generate features for a given time series. However, those packages generate a big amount of features without selecting the best ones.

We proposed a framework that automatize the process of analysis of a time series. The framework was developed by modules, with allow to change very quickly some method or technique by other without necessarily change a lot of code. Or add more model to forecast, for example.

The framework starts by cleaning the data, creates a simple baseline model, generates features, creates an ensemble model to evaluate those features and selects the best ones. With the best features it creates a XGBoost model that can predict values with low error. All of this process is shown in the visualization tool with tables, graphics and messages.

The framework was able to process a set of different datasets and deal with their differences and create different models with different sets of features. Through the visualization tool, the user can follow the process and be informed of each decision of the framework, for example what is the transformation applied if the time series is not stationary, or if it has missing values. Then it shows the sets of features computed and the results of forecast with the XGBoost model. The user can see the Mean Absolute Error, the Root Mean Squared Error, and the Mean Absolute Percentage Error, accuracy and a plot of the correlation between the observed data and predicted values.

With our cases studies we observe that the Final Model of each dataset has always a better perfor-

mance than the chosen baseline.

## 5.2 Future Work

To improve this framework, we suggest the Optimization of parameters of the XGBoost model, tuning the hyperparameters is an important task to achieve the best performance of the model. There are several ways to do that, we used the grid search but implementations using a Bayesian optimization was proven to deliver better and faster results; Test other methods to select features, specially to select features to compose: for example, using PCA to decrease dimensionality; and explore Wavelet transformations to generate features.

# Bibliography


[1] Anderson, R. L. (1953). Recent advances in finding best operating conditions. *Journal of the American Statistical Association*, 48(264):789–798.

[2] Banko, M., Cafarella, M. J., Soderland, S., Broadhead, M., and Etzioni, O. (2007). Open information extraction from the web. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, IJCAI'07, pages 2670–2676, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[3] Bergstra, J. and Bengio, Y. (2012a). Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13:281–305.

[4] Bergstra, J. and Bengio, Y. (2012b). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305.

[5] Bergstra, J. S., Bardenet, R., Bengio, Y., and Kégl, B. (2011). Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*, pages 2546–2554.

[6] Bisgaard, S. and Kulahci, M. (2011). *Time series analysis and forecasting by example*. John Wiley & Sons.

[7] Brochu, E., Cora, V. M., and De Freitas, N. (2010). A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*.

[8] Brown, Robert Goodell, . (1959). *Statistical forecasting for inventory control / Robert G. Brown*. McGraw-Hill New York.

[9] Chen, T. and Guestrin, C. (2016a). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794.

[10] Chen, T. and Guestrin, C. (2016b). Xgboost: A scalable tree boosting system. *CoRR*, abs/1603.02754.

[11] Christ, M., Braun, N., Neuffer, J., and Kempa-Liehr, A. W. (2018). Time series feature extraction on basis of scalable hypothesis tests (tsfresh–a python package). *Neurocomputing*, 307:72–77.

[12] Dash, M. and Liu, H. (1997). Feature selection for classification. *Intelligent Data Analysis*, 1(1):131 – 156.

[13] De Brabandere, A., Robberechts, P., De Beéck, T. O., and Davis, J. (2019). Automating feature construction for multi-view time series data. In *ECMLPKDD Workshop on Automating Data Science*, pages 1–16. N/A.

[14] Dietterich, T. G. (1997). Machine-learning research. *AI magazine*, 18(4):97–97.

[15] Escalante, H. J., Montes, M., and Sucar, L. E. (2009). Particle swarm model selection. *Journal of Machine Learning Research*, 10(Feb):405–440.

[16] Esling, P. and Agon, C. (2012). Time-series data mining. *ACM Computing Surveys (CSUR)*, 45(1):12.

[17] European Union (2017). Da.re.- data science pathways to re-imagine education. `http://dare-project.eu/download-2/`. Last checked on Jan 06, 2020.

[18] Fellbaum, C., editor (1998). *WordNet: An Electronic Lexical Database*. Language, Speech, and Communication. MIT Press, Cambridge, MA.

[19] Feurer, M., Eggensperger, K., Falkner, S., Lindauer, M., and Hutter, F. (2018). Practical automated machine learning for the automl challenge 2018. In *International Workshop on Automatic Machine Learning at ICML*, pages 1189–1232.

[20] Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., and Hutter, F. (2015). Efficient and robust automated machine learning. In *Advances in neural information processing systems*, pages 2962–2970.

[21] Friedman, L. and Markovitch, S. (2018). Recursive feature generation for knowledge-based learning. *CoRR*, abs/1802.00050.

[22] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. (2009). The weka data mining software: An update. *SIGKDD Explor. Newsl.*, 11(1):10–18.

[23] He, X., Zhao, K., and Chu, X. (2019). Automl: A survey of the state-of-the-art. *ArXiv*, abs/1908.00709.

[24] Hesterman, J., Caucci, L., Kupinski, M., Barrett, H., and Furenlid, L. (2010). Maximum-likelihood estimation with a contracting-grid search algorithm. *IEEE Transactions on Nuclear Science*, 57(3 PART 1):1077–1084.

[25] Holt, C. (1957). Forecasting seasonals and trends by exponentially weighted moving averages, onr memorandum (vol. 52), pittsburgh, pa: Carnegie institute of technology. *Available from the Engineering Library, University of Texas at Austin*.

[26] Hsu, C.-w., Chang, C.-c., and Lin, C.-J. (2003). A practical guide to support vector classification chih-wei hsu, chih-chung chang, and chih-jen lin.

[27] Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2011). Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization*, pages 507–523. Springer.

[28] Hutter, F., Kotthoff, L., and Vanschoren, J. (2019). *Automatic machine learning: methods, systems, challenges*. Challenges in Machine Learning. Springer, Germany.

[29] Hyndman, R. and Athanasopoulos, G. (2018). *Forecasting: Principles and Practice*. OTexts, Australia, 2nd edition.

[30] Ignatov, D., Spesivtsev, P., and Zyuzin, V. (2019). Multilabel classification for inflow profile monitoring. *MACSPro'2019, Vienna*, pages 21–23.

[31] Kanter, J. M. and Veeramachaneni, K. (2015). Deep feature synthesis: Towards automating data science endeavors. *2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 1–10.

[32] Karpathy, A., Johnson, J., and Li, F. (2015). Visualizing and understanding recurrent networks. *CoRR*, abs/1506.02078.

[33] Katz, G., Shin, E. C. R., and Song, D. X. (2016). Explorekit: Automatic feature generation and selection. *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 979–984.

[34] Khurana, U., Turaga, D., Samulowitz, H., and Parthasrathy, S. (2016). Cognito: Automated feature engineering for supervised learning. In *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*, pages 1304–1307. IEEE.

[35] Kotthoff, L., Thornton, C., Hoos, H. H., Hutter, F., and Leyton-Brown, K. (2017). Auto-weka 2.0: Automatic model selection and hyperparameter optimization in weka. *The Journal of Machine Learning Research*, 18(1):826–830.

[36] Lazarevich, I., Prokin, I., and Gutkin, B. (2018). Neural activity classification with machine learning models trained on interspike interval series data. *arXiv preprint arXiv:1810.03855*.

[37] McKinney, W. et al. (2010). Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, volume 445, pages 51–56. Austin, TX.

[38] Olson, R. S. and Moore, J. H. (2019). Tpot: A tree-based pipeline optimization tool for automating machine learning. In *Automated Machine Learning*, pages 151–160. Springer.

[39] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

[40] Pudil, P., Novovičová, J., and Kittler, J. (1994). Floating search methods in feature selection. *Pattern Recognition Letters*, 15(11):1119 – 1125.

[41] Rahm, E. and Do, H. H. (2000). Data cleaning: Problems and current approaches. *IEEE Data Engineering Bulletin*, 23:2000.

[42] Saeys, Y., Inza, I., and Larrañaga, P. (2007). A review of feature selection techniques in bioinformatics. *Bioinformatics*, 23(19):2507–2517.

[43] Smith, M. J., Wedge, R., and Veeramachaneni, K. (2017). Featurehub: Towards collaborative data science. In *2017 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 590–600. IEEE.

[44] Thornton, C., Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2012). Auto-weka: Automated selection and hyper-parameter optimization of classification algorithms. *CoRR*, abs/1208.3719.

[45] Tuggener, L., Amirian, M., Rombach, K., Lörwald, S., Varlet, A., Westermann, C., and Stadelmann, T. (2019). Automated machine learning in practice: State of the art and recent results. *CoRR*, abs/1907.08392.

[46] Winters, P. R. (1960). Forecasting sales by exponentially weighted moving averages. *Management science*, 6(3):324–342.

[47] Wu, W., Li, H., Wang, H., and Zhu, K. Q. (2012). Probase: A probabilistic taxonomy for text understanding. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, pages 481–492, New York, NY, USA. ACM.

[48] Yao, Q., Wang, M., Escalante, H. J., Guyon, I., Hu, Y., Li, Y., Tu, W., Yang, Q., and Yu, Y. (2018). Taking human out of learning applications: A survey on automated machine learning. *CoRR*, abs/1810.13306.

[49] Zöller, M. and Huber, M. F. (2019). Survey on automated machine learning. *CoRR*, abs/1904.12054.