LISBON SCHOOL OF ECONOMICS & MANAGEMENT
UNIVERSIDADE DE LISBOA

MASTER

MANAGEMENT INFORMATION SYSTEMS

MASTER´S FINAL WORK

PROJECT

INTELLIGENT SYSTEM FOR TIME SERIES PATTERN IDENTIFICATION AND PREDICTION

JOANA FILIPA CAETANO CLAUDINO

NOVEMBER - 2020

LISBON SCHOOL OF ECONOMICS & MANAGEMENT
UNIVERSIDADE DE LISBOA

MASTER
MANAGEMENT INFORMATION SYSTEMS

MASTER´S FINAL WORK
PROJECT

INTELLIGENT SYSTEM FOR TIME SERIES PATTERN IDENTIFICATION AND PREDICTION

JOANA FILIPA CAETANO CLAUDINO

SUPERVISION:
PROF. DR. ANTÓNIO MARIA PALMA DOS REIS

ACKNOWLEDGEMENTS

I would like to start by expressing my deepest gratitude to my parents and brother for their invaluable encouragement and comprehension all the way through the completion of this project. Their support and the trust they place in me are my greatest sources of motivation for never giving up and always trying to do my best.

I would like to extend my gratitude to my boyfriend not only for being so understanding of my absences during this process, but specially for his unconditional support and willingness to always help me in everything he could. Without his helpfulness, I would have not been able to finish this project. A special thanks to his parents for their kindness and support, too.

In addition, I would like to emphasise my gratitude to two master's degree colleagues with whom I became good friends. All their companionship and precious help in several aspects made this whole experience much more enjoyable and memorable than I could have ever asked for.

Lastly but not least, I would like to express my sincere appreciation to Prof. Dr. António Maria Palma dos Reis for the offered guidance, agreeableness, and patience throughout this entire process, as well as for his determinant help in defining the appropriate scope for the developed project.

As a final note, I would like to thank all my family members and friends, not previously mentioned, who directly or indirectly helped me in this long journey.

GLOSSARY

ACF – Autocorrelation Function

AIC – Akaike Information Criterion

AICc – Corrected Akaike Information Criterion

ANN – Artificial Neural Network

ARIMA – Autoregressive Integrated Moving Average

CH – Canova-Hansen

DSR – Design Science Research

DSRM – Design Science Research Methodology

GD – Gradient Descent

GPU – Graphics Processing Unit

GRU – Gated Recurrent Unit

KDD – Knowledge Discovery in Databases

KDDM – Knowledge Discovery and Data Mining

KPSS – Kwiatkowski-Phillips-Schmidt-Shin

LSTM – Long Short-Term Memory

MAE – Mean Absolute Error

MdAE – Median Absolute Error

MLE – Maximum Likelihood Estimation

MLP – Multi-Layer Perceptron

MSE – Mean Square Error

PACF – Partial Autocorrelation Function

ReLU – Rectified Linear Unit

RMSE – Root Mean Square Error

RNN – Recurrent Neural Network

SARIMA	Seasonal Autoregressive Integrated Moving Average
SGD	Stochastic Gradient Descent
STL	Seasonal-Trend Decomposition Procedure Based on Loess

ABSTRACT

The current growing volumes of data present a source of potentially valuable information for companies, but they also pose new challenges never faced before. Despite their intrinsic complexity, time series are a notably relevant kind of data in the entrepreneurial context, especially regarding prediction tasks. The Autoregressive Integrated Moving Average (ARIMA) models have been the most popular approach for such tasks, but they do not scale well to bigger and more granular time series which are becoming increasingly common. Hence, newer research trends involve the application of data-driven models, such as Recurrent Neural Networks (RNNs), to forecasting. Therefore, given the difficulty of time series prediction and the need for improved tools, the purpose of this project was to implement the classical ARIMA models and the most prominent RNN architectures in an automated fashion and posteriorly to use such models as foundation for the development of a modular system capable of supporting the common user along the entire forecasting process. Design science research was the adopted methodology to achieve the proposed goals and it comprised the activities of goal definition, followed by a thorough literature review aimed at providing the theoretical background necessary to the subsequent step that involved the actual project execution and, finally, the careful evaluation of the produced artifact. In general, each the established goals were accomplished, and the main contributions of the project were the developed system itself due to its practical usefulness along with some empirical evidence supporting the suitability of RNNs to time series forecasting.

Keywords: Time Series Forecasting; Autoregressive Integrated Moving Average Models; Recurrent Neural Networks; Intelligent System; Data Mining

RESUMO

Os crescentes volumes de dados representam uma fonte de informação potencialmente valiosa para as empresas, mas também implicam desafios nunca antes enfrentados. Apesar da sua complexidade intrínseca, as séries temporais são um tipo de dados notavelmente relevantes para o contexto empresarial, especialmente para tarefas preditivas. Os modelos Autorregressivos Integrados de Médias Móveis (ARIMA), têm sido a abordagem mais popular para tais tarefas, porém, não estão preparados para lidar com as cada vez mais comuns séries temporais de maior dimensão ou granularidade. Assim, novas tendências de investigação envolvem a aplicação de modelos orientados a dados, como Redes Neuronais Recorrentes (RNNs), à previsão. Dada a dificuldade da previsão de séries temporais e a necessidade de ferramentas aprimoradas, o objetivo deste projeto foi a implementação dos modelos clássicos ARIMA e as arquiteturas RNN mais proeminentes, de forma automática, e o posterior uso desses modelos como base para o desenvolvimento de um sistema modular capaz de apoiar o utilizador em todo o processo de previsão. *Design science research* foi a abordagem metodológica adotada para alcançar os objetivos propostos e envolveu, para além da identificação dos objetivos, uma revisão aprofundada da literatura que viria a servir de suporte teórico à etapa seguinte, designadamente a execução do projeto e findou com a avaliação meticulosa do artefacto produzido. No geral todos os objetivos propostos foram alcançados, sendo os principais contributos do projeto o próprio sistema desenvolvido devido à sua utilidade prática e ainda algumas evidências empíricas que apoiam a aplicabilidade das RNNs à previsão de séries temporais.

Palavras-chave: Previsão de Séries Temporais; Modelos Autorregressivos Integrados de Médias Móveis; Redes Neuronais Recorrentes; Sistema Inteligente; *Data Mining*

TABLE OF CONTENTS

TABLE OF CONTENTS – FIGURES

TABLE OF CONTENTS – TABLES

1. INTRODUCTION

1.1. Problem description and motivation

There has been an exponential increase in the amount of data collected by organizations over the past few years, largely due to great advances in technological infrastructures concerning data storage and processing (Siddiqa et al., 2017). These huge collections of data come with the promise of being a hidden source of opportunities that companies can exploit in order to better serve customers, as well as to improve operational efficiency and, hence, to achieve competitive advantage (Rey & Wells, 2012; Strohbach et al., 2016). The challenge lies, however, on how to extract actual value from such vast amounts of data when obviously it is no longer feasible to analyse them through most traditional methods that rely on a great deal of human intervention. There is a great need for new intelligent tools and techniques that can automatically transform these data into valuable information (Han et al., 2012; Bramer, 2016).

As an effort to address the mentioned challenges and to unify novel methods and best practices, new fields and concepts have emerged, namely those of knowledge discovery in databases (KDD) and data mining (Coenen, 2011). Although the distinction between both concepts is not very sharp, being many times used interchangeably in the literature, one of the most popular definitions that prevails to this day is that of Fayyad et al. (1996) that describes KDD as being the global process of unveiling potentially useful and understandable patterns in data while data mining as being only a step of that process, that comprises the implementation of data analysis algorithms by which such patterns are extracted.

Data comes in various formats that may require different handling techniques as each type of data presents its own challenges. For instance, time series data, albeit being an actively studied subject for many years, remains an important open topic especially in the more recent context of high volumes of data. In fact, Yang & Wu (2006) ranked time series data mining as one of the ten most challenging topics in data mining research. One of the major difficulties is that, although on one hand its temporal structure offers an additional source of information, on the other hand its resultant singular characteristics demand the use of specialised data analysis methods (Fu, 2011; Esling & Agon, 2012).

While time series data mining may serve plenty of different purposes, forecasting future values is the most commonly applied time series task in real-life settings and it is of crucial importance for organizations as it enables more informed decisions towards resource allocation, demand requirements, capacity planning and many other critical activities (Chatfield, 2004; Faloutsos et al., 2019). Nevertheless, as a result from being a topic whose foundations rely on a solid statistical background, producing reliable forecasts requires a significant level of expertise that often is not readily available inside organizations (Taylor & Letham, 2017). Even if such expertise is present, as the volume of collected time series data increases, it becomes impracticable to manually and individually analyse and produce forecasts for each of them, meaning that partially automating the forecasting process could be a beneficial solution (Hyndman & Khandakar, 2008). There are various well established statistical methods to model time series that allow to attain high forecasting accuracies in numerous problems, but, besides requiring the abovementioned expertise, they usually do not scale well to bigger volumes of data (De Gooijer & Hyndman, 2006; Faloutsos et al., 2019). Therefore, as new technologies like Internet of Things' sensors give rise to more complex time series structures, such as longer time series with higher frequency, it becomes apparent the need of exploring the suitability of new modern data mining methods to address these new challenges and achieve better forecasting results (Makridakis et al., 2018; Vaughan, 2020). A recent research trend on the topic is the development and application of new neural network architectures for time series forecasting and some remarkable cases of success have been reported as, for instance, the work of Zhu & Laptev (2017) at Uber and the work of Salinas et al. (2019) at Amazon.

In the new data-oriented paradigm, python programming language has been one of the top tools of choice to develop and implement new algorithmic approaches to data analysis in business applications, as well as in scientific and academic research (Robinson, 2017). Some of the main reasons for python's popularity are its simple syntax which makes the code easier to read and maintain, its vast number of specialized libraries that makes it a very versatile language which can be used in almost any development task from simple scripting to the development of large enterprise applications, and also its great community support that keeps improving the language's capabilities and helping new users to get acquainted with its functionalities (Wu, 2019; Cass, 2019). With no exception, it also offers notably useful capabilities to analyse and manipulate time series data (VanderPlas, 2018).

1.2. Objectives definition

The main goal of the project is to devise two automatic forecasting algorithms: a statistical one that comprises the class of seasonal and non-seasonal Autoregressive Integrated Moving Average Models (ARIMA), and a data mining one that comprises a specific type of Artificial Neural Networks (ANNs), more concretely Recurrent Neural Networks (RNNs) and their most popular variants, that are especially suited to sequential data.

In addition, given the fairly high level of difficulty inherent to the time series data analysis and the forecasting process, it is intended to use the implemented algorithms as foundation for the development of a small modular system capable of supporting the user along the entire process, from the data ingestion and pre-processing steps to the production of actual forecasts into the future, while providing informative and easily interpretable reports and graphs.

1.3. Document organization

The next sections of this document are organized as follows: the literature review where an in-depth description of time series data, their positioning in the context of data mining and the steps of the time series forecasting process, as well as the most relevant forecasting models, are presented; the methodology, where the chosen approach is described and justified; the project development, where the steps taken to achieve the proposed goal are carefully detailed; results, where the output of the project is evaluated; and conclusion where the key takeaways from the project and possible future work directions are presented.

2. LITERATURE REVIEW

2.1. Time Series Description

Time series are a very common type of data that occurs naturally in a variety of fields spanning from Medicine, Geophysics and Engineering to Industry, Finance and Economics (Keogh & Kasetty, 2003; Fu, 2011). Furthermore, the continuous technological advances, such as improved time series database systems, and the widespread adoption of the Internet of Things paradigm, potentiate even further the collection and generation of such data (Tahmassebpour, 2017; Vaughan, 2020). Therefore, it should come as no surprise that an increasingly large fraction of the world's supply of data is in the form of time series and, consequently, the interest in exploiting their value is bigger than ever, as argued by many researchers (Ratanamahatana et al., 2009; Rey & Wells, 2012; Faloutsos et al., 2019).

A time series is a sequential collection of values associated with a target variable, usually, sampled at evenly spaced intervals of time. More formally, it can be expressed as $\{y_{\mathrm{t}},\ \mathrm{t} \in \mathbb{Z}\}$, with each $y_t \in \mathbb{R}$ and $t = 1, \dots N$, where N denotes the length of the time series and y_N the last or most recent observation. Some diversified examples of time series could be the number of visitors of a website per minute, the hourly air temperature, the daily closing price of a stock, the monthly demand for a product, the quarterly earnings of a company or the yearly birth rate of a country (Fu, 2011; Mahalakshmi et al., 2016).

The implicit order of time series' observations inherent to their temporal dimension leads to some peculiar characteristics of this kind of data, which can be translated into three interrelated key properties: the common patterns that they exhibit, the serial dependence that they possess and the way their statistical properties vary over time (Hyndman & Athanasopoulos, 2018; Nau, 2019).

First, time series tend to exhibit certain common patterns that reflect how their values change over time and that are broadly classified as:

- Trends, which reflect long-term changes, either upward or downward, in the values of a time series (Chatfield, 2004);
- Seasonality, which refers to predictable, regular variations in time series' values that happen in specific time intervals over a one-year period. Variables whose observations are recorded in very short time intervals such as hourly or smaller, typically exhibit multiple seasonal patterns that are considerably more difficult to handle (Hyndman, 2014; Nau, 2019);
- Cycles, which are characterised by fluctuations in time series' values that last more than one year but whose period is not previously known. A straightforward example of such type of variation, is the business cycle that reflects the rise and fall of the global economic growth over time (Amadeo, 2008).

Usually, due to the unpredictable nature of cycles, trends and seasonality are the patterns of most interest when analysing time series data. The identification of these patterns, however, is not a trivial process as, many times, they are contaminated with noise (Yang & Wu, 2006). For that reason, it may be useful to use methods that aim to explicitly separate the trend (T_t) and seasonal (S_t) patterns from the aleatoric remainder patterns (R_t). One of the most notable methods in this matter is the one introduced by Cleveland et al. (1990), designated "*Seasonal-Trend Decomposition Procedure Based on Loess*" (STL) and which consists on a sequence of operations that employ locally weighted regression smoothing to extract the trend and seasonal components. The patterns extracted through the STL or similar techniques can be subsequently used to de-trend or de-seasonalize the time series under analysis, or as inputs to compute other useful measures such as the trend (F_T) and seasonal (F_S) strength indices, introduced by Wang et al. (2006) and defined by Eq. (1) and (2), that help to quantitatively determine the relevance or impact of such patterns in the time series' behaviour.

$$F_T = \max\left(0, 1 - \frac{\mathrm{Var}(R_t)}{\mathrm{Var}(T_t + R_t)}\right) \tag{1}$$

$$F_S = \max\left(0, 1 - \frac{\mathrm{Var}(R_t)}{\mathrm{Var}(S_t + R_t)}\right) \tag{2}$$

Furthermore, as previously mentioned, time series tend to possess serial dependence, meaning that they usually are correlated with their own prior values (Kritzman, 1994). This property can be expressed through the autocorrelation coefficient (r_k), whose values lie within the range $[-1, 1]$ and is given by

$$r_k = \frac{\sum_{\mathrm{t=k+1}}^{\mathrm{N}} (y_t - \bar{\mathrm{y}})(y_{t-k} - \bar{\mathrm{y}})}{\sum_{\mathrm{t=1}}^{\mathrm{n}} (y_t - \bar{\mathrm{y}})^2} \tag{3}$$

where y_t is the observation in time period t, $\bar{y}$ is the mean of all observations in the series, and y_{t-k} is the value of the observation k periods earlier. It measures both the direct correlation of y_t and y_{t-k}, and the indirect correlation resulting from the observations in between them (Levich & Rosario, 1999; Hyndman & Athanasopoulos, 2018). The plot of the autocorrelation coefficient as a function of the lag k is denominated the Autocorrelation Function (ACF) (Box et al., 2016). The ACF is a very useful device to analyse the behaviour of time series because the aforementioned patterns, trend and seasonality, are characterized by specific autocorrelation structures and thus also reflected in the ACF. More concretely, if there is a trend, successive observations will likely be highly correlated, while if a seasonal pattern exists, there will be significant autocorrelation at multiples of the seasonal lag (Hanke & Wichern, 2014). According to Box et al. (2016), the significance of the autocorrelations in the ACF can be assessed through the lower and upper bounds that also frequently accompany the ACF plot and which, assuming a normal distribution with a mean of zero, are computed as in Eq. (4).

$$\pm z_{1-\alpha/2}\, SE(r_k) \qquad (4)$$

Here, z is the z-score, α is the confidence level, r_k is the sample autocorrelation at lag k and the standard error (SE), is computed as $1/\sqrt{n}$, for lag = 1 and as $\sqrt{\frac{1}{n}\left(1+2\sum_{k=1}^{h-1} r_k^2\right)}$, for lag > 1, being n the number of data points in the time series (Bartlett 1946 ; Quenouille 1949). If one or more autocorrelations exceed the mentioned bounds, then it means that they are significantly different from zero and there are predictable patterns in the time series. If, however, all autocorrelations stay within these bounds, the time series can be regarded as random or non-serially correlated.

A closely related concept is that of partial autocorrelation coefficient that only measures the direct correlations between y_t and y_{t-k} (Hyndman & Athanasopoulos, 2018). The plot of the partial autocorrelation coefficient as a function of the lag k is the called the Partial Autocorrelation Function (PACF) and its bounds are calculated in a similar way as in Eq. (4) (Levich & Rosario, 1999).

As asserted by Hyndman & Athanasopoulos (2018), instead of checking the significance of each autocorrelation coefficient separately through the ACF and PACF plots, it is possible to perform a more formal assessment by testing a set of autocorrelation coefficients through the Box-Ljung test (Ljung & Box, 1978), whose null hypothesis is that of serial independence and its test statistic is computed as in Eq (5).

$$Q^* = n(n+2)\sum_{k=1}^{h}\frac{r_k^2}{n-k} \qquad (5)$$

The test statistic, Q^*, has a χ^2 distribution with $h-K$ degrees of freedom, where K is the number of parameters in the model when testing the residuals of a fitted model, and $K=0$ when testing raw data.

Finally, the mentioned third key property of time series can be described through the concept of stationarity. A stationary time series is one whose statistical properties – mean, variance and autocorrelation structure – remain constant over time (Nau, 2019). This does not mean, however, that the time series does not change, just that it fluctuates around a fixed level with a constant variance and it does not depend on time (Hanke & Wichern, 2014). Nonetheless, in practice most time series exhibit a non-stationary behaviour as strong trends or seasonal patterns affect the referred statistical properties at different points in time (Chatfield, 2004; Hyndman & Athanasopoulos, 2018).

Thus, once again, formal methods to test non-stationary behaviour have been developed, more concretely, a class of statistical tests called unit root tests (Haldrup et al., 2013). A unit root in a time series indicates the presence of a systematic non-predictable pattern (Glen, 2016). For time series with a trend pattern, one of the most widely used unit root tests is the Kwiatkowski-Phillips-Schmidt-Shin (KPSS) test, whose null hypothesis is that of the time series being stationary around a constant trend against the alternative of having a unit root (Kwiatkowski et al., 1992). For time series with a seasonal pattern, in turn, one of the most popular tests is the Canova-Hansen (CH) test whose null hypothesis is that of the times series having stationary seasonal cycles against the alternative of having seasonal unit roots (Canova & Hansen, 1995).

2.2. Time series in the context of data mining

The increasingly massive volumes of time series data and the consequent demand for more sophisticated data-driven analysis tools and techniques to exploit their value has increased the attention and interest of the data mining community on this subject (Fu, 2011; Faloutsos et al., 2019). Data mining is a relatively recent and somewhat ambiguous term that has distinct meanings for different authors and is, many times, used in the literature as synonymous of Knowledge Discovery in Databases (KDD). At an early stage of the emergence of this new field, some authors, namely Fayyad et al. (1996), made an effort to concisely distinguish both terms by defining KDD as the nontrivial process, composed by many steps, of identifying new, useful and comprehensible patterns in data, and data mining as being merely one of its many steps, concerned with the implementation of intelligent algorithms that allow the extraction of such patterns. Nevertheless, throughout the years, the boundaries between both concepts became blurry and they are now mostly used interchangeably referring to the overall process of automatically identifying and extracting patterns from data, with data mining being the preferred and most widely adopted terminology (Chung & Gray, 1999; Han et al., 2012). There have also been some suggestions to rename the field with the aim to clarify this terminological issue and, simultaneously, adapt it to a more modern technological reality, given that we no longer are uniquely concerned with data stored in traditional databases. For instance, Kurgan & Musilek (2006) and Coenen (2011) advocate that Knowledge Discovery and Data Mining (KDDM), would be a more appropriate nomenclature for data coming from any source and would also emphasise the fact that both terms are related, yet distinct. This lack of full consensus regarding terminologies and boundaries of the field is, in part, due to its multidisciplinary nature as it incorporates concepts and techniques from many different areas including statistics, machine learning, database systems, information theory and data visualization (Hand, 1998; Han et al., 2012).

This being said, in order to avoid any ambiguities, in the present text, the perspective advocated by Fayyad et al. (1996) and also supported by Kurgan & Musilek (2006), of data mining being a step of the KDD process, is adopted. As stated by Bramer (2016), this standpoint has the advantage of highlighting that, although the data mining step is crucial for knowledge discovery, the pre-processing steps and the proper interpretation of the results are very important as well.

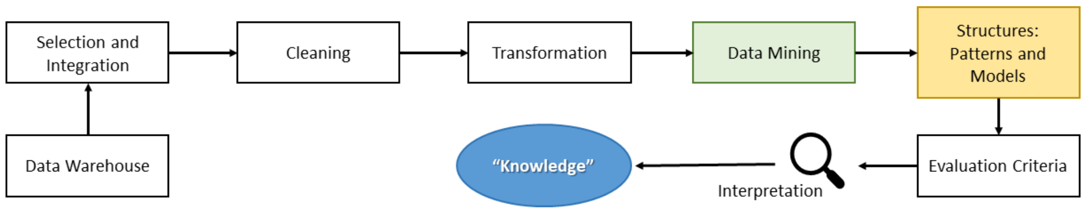


Figure 1. Summary of the KDD Process, adapted from Fayyad et al. (1996)

The full KDD process is summarized in *Figure 1*. In short, it starts with the selection and integration of data that can come from multiple sources, followed by the cleaning step that aims to deal with missing values or removing noise from the data. Then, transformation comprises the conversion of the data, if necessary, to a more convenient structure to be ingested by the selected algorithms in the data mining phase. These data mining algorithms produce an output in the form of patterns that greatly varies depending on the goal of the discovery process and on the type of data being analysed. Lastly, the relevance of the extracted pattern is evaluated based on some predefined criterion and the results are interpreted, with the aid of visualization techniques, in order to obtain new and useful knowledge (Fayyad et al., 1996; Han et al., 2012; Bramer, 2016).

As abovementioned, the type of data mining task carried out depends on the goal of the KDD process and it can be broadly categorized as descriptive or predictive (Fayyad et al., 1996). In a descriptive setting data mining tasks have an exploratory nature and their purpose is to discover patterns that summarize hidden relationships in the data, whereas in a predictive setting, data mining tasks aim at predicting the value of a particular attribute based on identified patterns (Tan et al., 2006).

Despite their peculiar features, such as the ordering and implicit dependency between successive observations, time series data are also typically analysed with either a descriptive or predictive purpose (Shmueli & Lichtendahl, 2018). Therefore, some more general data mining tasks, such as query by content, clustering, anomaly detection, association or classification, performed with other types of data are also applied in the context of time series data. There are, however, some more specific tasks related to their sequential nature and temporal component, such as segmentation, motif discovery and forecasting (Esling & Agon, 2012; Mahalakshmi et al., 2016). All these tasks are summarized in Table I.

TABLE I. MAIN DATA MINING TASKS FOR TIME SERIES DATA

Goal	Task	Description
Descriptive	Query by content	Given a time series and a similarity measure, retrieves the set of solutions that better matches the query provided by the user (Keogh & Kasetty, 2003).
	Clustering	Finds natural groups, called clusters, in an unlabelled time series dataset based on hidden similar characteristics (Liao, 2005)
	Segmentation	Reduces the dimensionality of a time series while retaining its essential features, in order to create an accurate approximation of the original series (Gullo et al., 2009).
	Motif Discovery	Enumerates the most recurring patterns, called motifs, that appear on a time series (Lin et al., 2002; Liu et al., 2005).
	Anomaly Detection	Finds observations of the time series whose values differ significantly from the rest of the data (Teng, 2010).
	Association	Derives rules for discovered associations and correlations among items within a dataset. In time series, requires the discretization of the data and subsequent

		conversion to a symbolic representation (Qin & Shi, 2006)
Predictive	Classification	Assigns the time series data to one of two or more predefined classes (Ratanamahatana & Keogh, 2004).
	Forecasting (Prediction)	Given a time series, predicts its future values by exploiting the correlations between successive observations and, possibly, with other variables. It implicitly assumes that some of the past patterns will continue into the future. (Montgomery et al., 2015; Hyndman & Athanasopoulos, 2018).

Although all the above data mining tasks have their own applications and merits, it is rather consensual in the literature that forecasting is not only the most applied and relevant time series data mining task in practical applications but, simultaneously, one of the most difficult (Esling & Agon, 2012; Tahmassebpour, 2017; Faloutsos et al., 2019).

As a matter of fact, time series forecasting is applied in areas as diverse as agriculture to improve the production of crops (Santos et al., 2019), the energy sector to predict the electric power load (Almeshaiei & Soltan, 2011), epidemiology to predict influenza outbreaks (Smolen, 2014) or in industry to predict production levels (Chen & Wang, 2007), just to mention a few. In the more specific context of business and management, time series forecasting is essential for any type of organization as well as for any of its functional lines, since it plays a key role in the optimization and monitoring of several business processes and, despite its intrinsic inaccuracies, it helps to reduce the uncertainty for management decision-making and strategic planning, being an especially critical tool in the current highly dynamic business environment (Rey & Wells, 2012; Hanke & Wichern, 2014; Faloutsos et al., 2019).

Conversely, the main difficulties of this task arise mostly from the fact that it is inherently a statistical subject that requires a considerable level of knowledge and experience, now with the added complications of the ever growing volumes of data and the consequent increasingly demand for higher quality forecasts and more efficient algorithms (Esling & Agon, 2012; Taylor and Letham, 2017; Faloutsos et al., 2019).

The main steps involved in the time series forecasting process will be further discussed in the following section.

2.3. Time Series Forecasting Process

Time series forecasting can be defined as a quantitative forecasting technique that predicts the future values of a target variable based on an information set of current and historical values, along with a model capable of summarizing the patterns contained in that observed variable and projecting them into the future (Montgomery et al., 2015; Yau, 2018). More formally, it can be expressed as

$$\hat{y}_{N+H} = f(\Omega) + \varepsilon \tag{6}$$

where $\hat{y}$ is the predicted value for each timestep, H is the forecast horizon, f is the model, Ω is the set of available information and ε is the error term that represents the random variation not explained by the fitted model. According to Hyndman & Athanasopoulos (2018), the choice of a particular model, f, may be influenced, to some extent, by the set of available information. More concretely, when Ω comprises predictor variables that are known to impact the target time series, it may be useful to build an explanatory model that takes into account their effects. Such modelling approach has, however, some added complications, namely the need to fully understand the relationships between predictor variables and target time series and, even more difficult, the need to know the future values of the predictors beforehand so that the target time series can actually be predicted. When using explanatory models it is thus very common to forecast the target variable based on forecasts of the predictor variables which ends up being an additional source of error and uncertainty and should be avoided when possible (Hanke & Wichern, 2014; Montgomery et al., 2015). Therefore, when enough historical data of the target time series is available and the main goal is to forecast its future values and not necessarily understand the forces that cause them, the use of pure time series models might be the best solution (Hyndman & Athanasopoulos, 2018). These models assume that the impact of external factors is already embodied in the patterns of the current and past values of the target time series and exploit their statistical properties in order to predict the future values, i.e., the information used by such models is given by $\Omega = \{y_N, y_{N-1}, y_{N-2}, \dots, y_1\}$ (Montgomery et al., 2015; Yau, 2018).

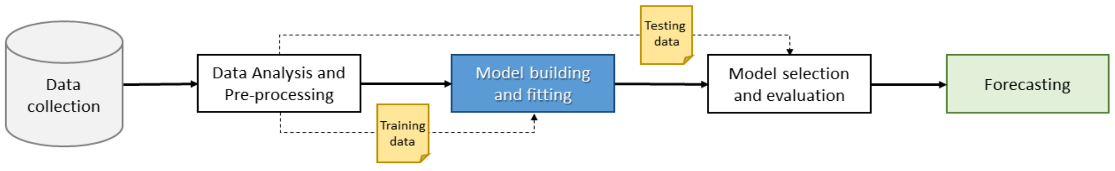

Data collection
Data Analysis and Pre-processing
Training data
Model building and fitting
Testing data
Model selection and evaluation
Forecasting

Figure 2. The time series forecasting process; adapted from Montgomery et al. (2015)

Due to the fact of time series forecasting is regarded as a predictive data mining task, the steps of its entire process are closely linked to that of KDD (in *section 2.2.*). Besides the obvious first step of data collection, it comprises four more steps, depicted in *Figure 2*: a data analysis and pre-processing step, a model building and fitting step, a model selection and evaluation step and, finally, the step of producing actual forecasts (Armstrong, 2002; Hanke & Wichern, 2014). Each of these will be further detailed in the following subsections of the document.

2.3.1. Data analysis and pre-processing

This step comprises the visualization of the time series to aid in the identification of potential patterns, as well as the computation of relevant metrics that describe the dataset. It also deals with missing value imputation, with possible necessary transformations in order to get the data into the correct form to be ingested by specific forecasting models, and with data splitting for the subsequent model fitting and evaluation (Hanke & Wichern, 2014; Montgomery et al., 2015).

Time series missing value imputation is a research topic on itself, however, some of the most applied methods involve using aggregate values such as the mean of the time series, using the most recent observation prior or following the missing value, or more advanced techniques, such as interpolation (Moritz & Bartz-Beielstein, 2017).

The transformations applied to time series are, in turn, mostly dependent on the characteristics of the particular dataset at hand and, to some extent, on the model that will be fitted to the data. The most frequently used are:

- The logarithmic transformation whose purpose is to stabilize the variance of the time series or to make its distribution normal (Lütkepohl & Xu, 2010).
- Data normalization, which involves squashing the values of time series so that all of them lie within a smaller range, usually between 0 and 1. It is especially useful to speed up the learning process and convergence in data mining models. The main normalization techniques are min-max normalization and z-score normalization (Shalabi et al., 2006; Patro & Sahu, 2015).
- Differencing, which is the most common transformation to make a time series stationary. It stabilizes the mean of a time series as it removes changes in its level or, in other words, it eliminates trend and seasonal effects (Hyndman &

Athanasopoulos, 2018). For trends, differencing is given by Eq. (7) and it simply computes the differences between consecutive observations. A first difference, i.e. $d = 1$, can eliminate a linear trend, while a second difference, $d = 2$, can eliminate a quadratic trend, and so on (Shumway & Stoffer, 2017). For seasonal variation, Hyndman & Athanasopoulos (2018) state that differencing means computing the difference between an observation and the previous observation from the same season and it is given by Eq. (8).

$$\nabla^d y_t = y_t - y_{t-d} = (1-B)^d y_t \tag{7}$$

$$\nabla_s^D y_t = y_t - y_{t-D} = (1-B^s)^D y_t \tag{8}$$

Here, d is the differencing order, D is the seasonal differencing order, s is the seasonal period and B is the backward shift operator, defined by $B^s y_t = y_{t-s}$, where $s = 1$ for non-seasonal differences.

Finally, the splitting of the dataset involves partitioning the data into a training set, used in the model fitting step to estimate the parameters of each model, and a testing set, used in the model evaluation step to assess the predictive performance. Some models, especially from the data mining field, may require the training set to be further split into a validation set as depicted in *Figure 3* (Brownlee, 2017b; Nau, 2019).

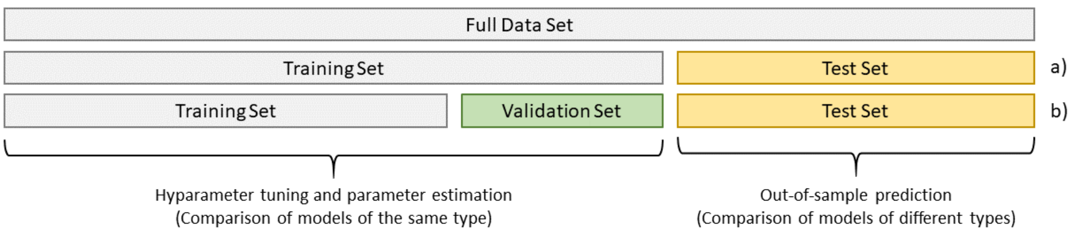

Full Data Set
Training Set
Test Set
a)
Training Set
Validation Set
Test Set
b)
Hyparameter tuning and parameter estimation
(Comparison of models of the same type)
Out-of-sample prediction
(Comparison of models of different types)

Figure 3. a) Common splitting approach for statistical methods; b) Common splitting approach for data mining methods

2.3.2. Model building and fitting

In the current forecasting landscape, time series forecasting methods are broadly categorized as statistical or data mining methods (Makridakis et al., 2018). Given its wide scope of application, there are many algorithms developed to deal with this task which would make impossible to cover all of them. Therefore, this project focuses, under the statistical category, on seasonal and non-seasonal Autoregressive Integrated Moving

Average (ARIMA) models and, under the data mining category, on Artificial Neural Networks (ANNs), more concretely on Recurrent Neural Networks (RNNs).

2.3.2.1. ARIMA models

Under the statistical methods for time series forecasting, the most notable work is that of Box & Jenkins (1970). These authors devised a practical approach for modelling linear time series that exhibit whether stationary or non-stationary behaviour and that became widely known as the Box-Jenkins method. Up to that point, simple linear stationary models were the prevailing approach in the field (Fildes & Makridakis, 1995).

The great capability of the Box-Jenkins family of models to deal with several types of patterns, as well as their fairly high accuracy in short and medium-term forecasts allied to the fact that, in practice, most time series are non-stationary makes them very popular and widely used across many areas up to this day (Petropoulos et al., 2014; Makridakis et al., 2018). The Box-Jenkins class of models encompasses three key components:

- The Autoregressive (AR) component, that aims to model the autocorrelation structure of the time series and assumes that the current value of the series can be explained by a linear combination of p previous values. It is more formally referred to as an autoregressive process of order p, denoted by $AR(p)$ (Shumway & Stoffer, 2017);
- The Integration (I) component that accounts for the number, d, of differences required to obtain a stationary time series. When any difference is needed, the time series is said to be an integrated process of order d, denoted by $I(d)$ (Box et al., 2016);
- The Moving Average (MA) component, that attempts to capture the unknown factors that affect the time series but are not explained by its past values and, hence, uses a linear combination of q past prediction errors. Referred to as a moving average process of order q, denoted by $MA(q)$ (Box et al., 2016; Nau, 2019).

From the combination of these components we can obtain the non-seasonal Autoregressive Integrated Moving Average (ARIMA), which is parametrized by p, d and q and defined by Eq. (9). While this model allows to explicitly capture the correlations between adjacent time series observations, it does not account for dependencies between observations that are several periods apart and which are common in seasonal patterns (Box et al., 2016; Shumway & Stoffer, 2017). In order to overcome such issue, a generalization of the ARIMA model, known as Seasonal ARIMA (SARIMA) and also formulated by Box & Jenkins (1970), that is able to capture multiplicative seasonality, can be used. This model has three additional parameters P, D and Q, which are the orders of the corresponding seasonal AR, I and MA components and is defined by Eq. (10).

$$\text{ARIMA(p, d, q)}: \quad \phi(B)\nabla^d y_t = c + \theta(B)\varepsilon_t \tag{9}$$

$$\text{SARIMA(p, d, q)x(P, D, Q)}_s: \quad \phi(B)\Phi(B^s)\nabla^d\nabla^D y_t = c + \theta(B)\Theta(B^s)\varepsilon_t \tag{10}$$

In these equations, $\phi(B)$ is the non-seasonal autoregressive polynomial of order p, $\theta(B)$ is the non-seasonal moving average polynomial of order q while $\Phi(B^s)$ and $\Theta(B^s)$ are their seasonal counterparts of order P and Q respectively, $\nabla^d\nabla^D y_t$ is the time series, differenced d times and seasonally differenced D times, s is the seasonal period, c is a constant, and $\varepsilon_t \sim N(0, \sigma^2)$ is a random error term with mean of zero and constant variance, more precisely, white noise.

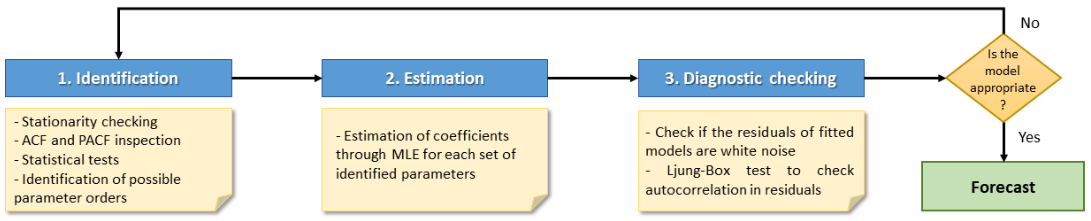

No
1. Identification
- Stationarity checking
- ACF and PACF inspection
- Statistical tests
- Identification of possible parameter orders
2. Estimation
- Estimation of coefficients through MLE for each set of identified parameters
3. Diagnostic checking
- Check if the residuals of fitted models are white noise
- Ljung-Box test to check autocorrelation in residuals
Is the model appropriate ?
Yes
Forecast

Figure 4. Summary of the Box-Jenkins method; adapted from Box et al. (2016)

The process of building and fitting the described models, depicted in *Figure 4*, is done through the iterative Box-Jenkins method. This method is comprised by three steps that precede the actual forecasting step:

1. Identification. Involves checking if the time series is stationary and, if not, to determine how many differences are required in order to achieve stationarity. This stationarity assessment can be done by inspecting the ACF, PACF and time series plot in order to identify any trends or seasonal patterns. After transforming the time series to a stationary form, a careful inspection of the ACF and PACF plots must be carried out once again, in order to determine the potentially correct orders of the autoregressive and moving average components (Box et al., 2016).
2. Estimation. Once the set of possible orders for the autoregressive and moving average components has been identified along with the number of differences, the preferred method to estimate the coefficients of the model is, typically, the maximum likelihood

estimation (MLE) (Box et al., 2016). This estimation method usually requires non-linear optimization techniques (Pelgrin, 2011).

3. Diagnostic checking. According to Box et al. (2016), the residuals from an adequate model fitted to the data should be white noise – i.e., random, independent and have zero mean – meaning that they should not possess any statistically significant autocorrelation nor exhibit any type of pattern. The tools that can be used to assess the adequacy of the model are the ACF plot of the residuals, which displays their individual autocorrelation coefficients or the already mentioned Ljung-Box test.

Once the most adequate model is found, it can be further evaluated by comparing its performance to other types of model or be directly used to produce forecasts.

2.3.2.2. ANN models

The application of ANNs to time series forecasting has been a topic of interest for the data mining community since some time ago and it has recently gained a new wave of attention, as more recent architectures are becoming the state-of-the-art for many tasks, especially for those related to sequential data (Sutskever et al., 2014; Faloutsos et al., 2019). An ANN is a data mining model inspired by the human brain used for information processing and pattern recognition. Besides having been mathematically proven to be a universal function approximator, it does not make strong assumptions about the data generation mechanism, in contrast to the previously introduced statistical models, thus being a flexible method capable of identifying and modelling complex patterns as well as learning linear and non-linear relationships (Remus & O'Connor, 2002; Zhang, 2012).

In practice, the application of ANNs to time series forecasting has been mostly through the multi-layer perceptron (MLP) also known as feed forward neural network (Zhang, 2004). This latter designation derives from the fact that, in this model, the information flows directly from the input layer, through the intermediate layers to the output layer without any feedback to previous layers (Goodfellow et al., 2016). The MLP is a generalization of the Perceptron model originally proposed by Rosenblatt (1958). This model was composed by a single neuron – the basic information processing unit in an ANN – that would compute the sum of the inputs weighted by the connections' weights plus a bias term and then pass it to a binary function. As demonstrated by Minsky & Papert (1969), it could only solve problems of data that belonged to linearly separable classes, making it of little value to more complex problems. The MLP overcame this weakness by introducing the concept of hidden layers, which are intermediate layers between the input and output layers that apply non-linear transformations to the data (Touretzky & Pomerleau, 1989). Such transformations are carried out through activation functions, being the most common ones the sigmoid (σ), the hyperbolic tangent ($tanh$) and the rectified linear unit (ReLU) function (Nair & Hinton, 2010; Karlik & Olgac, 2011). As shown in Eq. (11), each neuron (h_j) included in the hidden layers has the role of computing the sum of the input features (x_i), once again weighted by the connections' weights (w_{ji}) plus a bias term (b), followed by the application of one of the referred activation functions (f).

$$h_j = f\left(\sum_{i=1}^{n} w_{ji}x_i + b_j\right) \tag{11}$$

All the intermediate layers perform this sort of calculation until the output layer is reached. This last layer, in turn, also applies a function to the data which strongly depends on the problem being solved. For time series forecasting, it generally is the identity function (Kolarik & Rudorfer, 1994).

Being a supervised learning algorithm, in order to learn the correlations and patterns directly from the data, ANNs must be trained by comparing their outputs to the true values, also known as labels in this context, through a loss function (Lee et al., 2004). The loss function can take many forms, however, in the particular case of time series forecasting the most common one is the mean squared error (MSE), given by

$$MSE = \frac{1}{N}\sum_{t=1}^{N}(y_t - \hat{y}_t)^2 \tag{12}$$

where N is the number of data points, y_t is the target value and $\hat{y}_t$ is the prediction made by the neural network (Géron, 2019).

The main goal of the training process is to minimize the loss function. This can be achieved by using the backpropagation algorithm introduced by Rumelhart et al. (1986), which is of utmost importance for efficiently training any ANN. This algorithm iteratively computes the gradient of the loss with respect to each parameter of the neural network and then updates each parameter in the opposite direction of the gradient, through a technique named gradient descent (GD). There are many alternative procedures to perform GD, however, the standard applied method is Stochastic Gradient Descent (SGD) (Amari, 1993; Rakhlin et al., 2012). The SGD updates the weights of the ANNs based on the gradient computed from each training example, or batch of training examples, fed to the network by using the update rule defined by:

$$\theta := \theta - \eta * \frac{\partial J(\theta)}{\partial \theta} \tag{13}$$

Here, θ denotes the parameters of the neural network, more concretely the weights of the connections, η is the learning rate which defines the step size towards the minimum of the loss, $J(\theta)$ is the loss function and $\frac{\partial J(\theta)}{\partial \theta}$ is the gradient of the loss with respect to the weights.

The great flexibility of ANNs stemming from their high number of parameters has, however, the drawback of making them very prone to overfitting, i.e., they tend to also model the noise in the data as being a relevant pattern, leading to poor generalization capacity (Lawrence et al., 1997; Lever et al., 2016). Hence, it is usual to use regularization methods that penalize large weights of the network with the aim of preventing such problem. A fairly recent method intended to address this issue, and that has been empirically shown to achieve remarkably good results, is the dropout technique introduced by Srivastava et al. (2014). Dropout works by randomly omitting neurons and their connections during training so that they do not adapt too much to the data.

Besides the feedforward connections, ANNs can also have feedback connections, meaning that the output of any layer may be fed back to itself and to earlier layers (Li et al., 2004). These are named Recurrent Neural Networks (RNNs) and, according to Karpathy (2015), their great strength lies on the fact that they allow to operate in sequences of vectors, which makes them a very flexible algorithm capable of handling a wide range of sequential data tasks. They can be arranged in such a way that allows them to take a single value as input and yield a sequence of output values (one-to-many), take a sequence of input values and return a single output value (many-to-one), or take a sequence of input values and produce a sequence of output values (sequence-to-sequence) (Olah, 2015). Natural language processing has been their main field of application (Graves, 2013; Sutskever et al., 2014). However, more recently, given their ability to explicitly account for the order and dependency of sequential data – which are desirable characteristics to model time series data –, RNNs have also attracted the interest of time series forecasting researchers, namely Zhu & Laptev (2017) and Salinas et al. (2019), who adapted and further extended their application to the field.

The presently known simplest RNN derives from the contributions of many researchers throughout time, such as those of Werbos (1990), Elman (1990) and Jordan (1997), and it is loosely designated in the literature as traditional RNN.

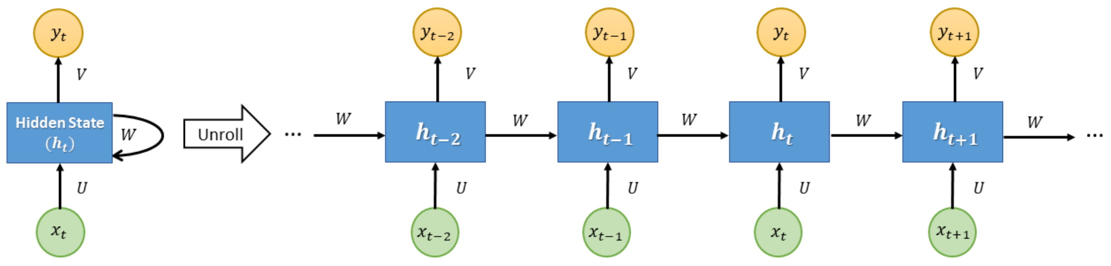


Figure 5. Traditional RNN and its unrolled version; adapted from Olah (2015)

The key component of the traditional RNN is the hidden state (h_t) defined by Eq. (14), that acts as a memory unit, and is computed at each time step by adding the previous hidden state (h_{t-1}) to the current input (x_t), each of them parametrized by their weight matrices W and U, respectively, plus a bias (b) followed by the application of an activation function (f), that usually is the *tanh*. Both weight matrices are shared across all time steps (Goodfellow et al., 2016).

$$h_t = f(Wh_{t-1} + Ux_t + b) \tag{14}$$

A RNN is trained in the same way as a feedforward network but, in this circumstance, the training algorithm has the name of backpropagation through time (BPTT) because the network, as depicted in *Figure 5*, must be first unrolled so that the gradient can, then, be propagated back through time (Werbos, 1990). It is, however, a known issue that, in practice, traditional RNNs are quite difficult to train for long sequences due to the recurrent formulation which results in the sharing of parameters through a very deep computational graph. This, in turn, leads to unstable gradients that mostly tend to vanish and, sometimes, to explode ultimately causing the network to be unable to learn long-term dependencies (Bengio et al., 1994; Pascanu et al., 2013).

The most effective solution to overcome the shortcomings of the traditional RNN, has been the use of gated memory units, specifically the Long Short-Term Memory (LSTM) and the Gated Recurrent Unit (GRU) (Hochreiter & Schmidhuber, 1997; Cho et al., 2014). Both LSTM and GRU, depicted in *Figure 6*, aim to explicitly avoid the long-term dependency problem related to vanishing gradients, by using layers that act as gates to control the flow of information. In opposition to the RNN, that fully replaces its hidden state at each time step, the LSTM and GRU are able to retain most of their hidden state's content while adding new information to it. As depicted in *Figure 6*, in contrast with the traditional RNN whose memory unit has a single $tanh$ layer, both LSTM and GRU memory units have three and two additional sigmoid layers, respectively.

The LSTM layers are named the forget gate (f_t), the input gate (i_t), the candidate

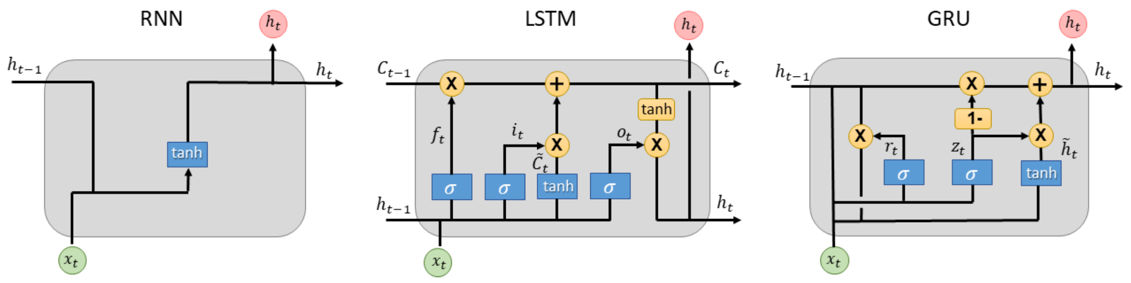


Figure 6. Diagram of a recurrent unit from a traditional RNN, a LSTM and a GRU; adapted from Olah (2015).

values gate ($\tilde{C}_t$) and the output gate (o_t). Each of them has the function of selectively discarding or adding information to the cell state (C_t), also known as the long-term memory because it is the key component in preventing the vanishing gradient problem, and to the hidden state (h_t) also known as the short-term memory because it is the key component responsible for enabling the LSTM to make decisions over short periods of time (Hochreiter & Schmidhuber, 1997; Olah, 2015; Jozefowicz et al., 2015). All these components of the LSTM are defined by the following equations:

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right) \tag{15}$$

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{16}$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{17}$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{18}$$

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \tag{19}$$

$$h_t = o_t \odot \tanh(C_t) \tag{20}$$

The W_* and b_* denote the weight matrixes and bias vectors for each gate and the $\odot$ denotes element-wise multiplications. Furthermore, C_{t-1}, h_{t-1} and x_t denote the previous cell state, the previous hidden state and the current input, respectively.

The GRU was proposed by Cho et al. (2014) with the aim of simplifying the LSTM while still preserving its benefit of circumventing the vanishing gradients problem. Its layers are named the update gate (z_t), the reset gate (r_t) and the candidate values gate ($\tilde{h}_t$). There are two crucial differences between this recurrent unit and the LSTM; the first is the fact that the update gate combines the functionality of both the forget and input gates, meaning that it is fully accountable for the decision of retaining past memory and of adding new information, and, second, it does not possess a cell state, only a hidden state (h_t) which stores both long and short-term patterns. All the components of the GRU are defined by the following equations:

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t] + b_z) \tag{21}$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t] + b_r) \tag{22}$$

$$\tilde{h}_t = \tanh(W_h \cdot [r_t \odot h_{t-1}, x_t] + b_h) \tag{23}$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \tag{24}$$

Once again, the W_* and b_* denote the weight matrixes and bias vectors for each gate, the $\odot$ represents element-wise multiplications, and h_{t-1} and x_t denote the previous hidden state and current input, respectively.

The performance of the three memory units depicted in *Figure 6* – RNN, LSTM and GRU – has been compared by many researchers in distinct sequential data tasks, especially in those related to natural language processing. While there is clear evidence that both gated units consistently outperform the traditional RNN, there is no definite conclusion on which of them is better (Chung et al., 2014; Jozefowicz et al., 2015). Moreover, in the specific case of time series forecasting, although also no concrete conclusion can be drawn yet due to the early stage of the research in the field, some studies seem to suggest a slight superiority of the GRU over LSTM (Fu et al., 2016; Kumar et al., 2018; Gallicchio et al., 2019).

Lastly, it should be noted that although the presented gated RNNs overcome the vanishing gradients problem, they do not address the exploding gradients problem. Hence, a somewhat simple but very effective solution for this issue is to combine them with the gradient clipping technique introduced by Pascanu et al. (2013). By using this technique, a threshold value is established and whenever the gradients exceed it, they are forced or "clipped" to that threshold.

2.3.3. Model selection and evaluation

Essentially, any algorithm has two types of parameters, those that are estimated from the available data and the hyperparameters which require some specialised knowledge about the model and dataset being used and must be set before any training or model fitting begins (Brownlee, 2017a). For the previously discussed models, both types of parameters are summarized in Table II.

TABLE II. HYPERPARAMETERS AND PARAMETERS FOR EACH TYPE OF MODEL

	Hyperparameters	**Parameters**
ARIMA based models	p, d, q, P, D, Q	$\phi_p, \Phi_P,\ \theta_q, \Theta_Q, \sigma^2,$ and c
ANNs (MLPs, RNNs)	Number of hidden neurons, learning rate, batch size.[1]	Weights and biases

As described in the previous section, ARIMA models are typically estimated by maximum likelihood. The model selection procedure can thus be carried out by using an information criterion, that measures the relative quality of each model belonging to a set of candidate models by balancing their goodness-of-fit, based on the MLE, and their complexity in terms of number of parameters (Ding et al., 2018). One popular IC for time series model selection is the Akaike Information Criterion (AIC) introduced by Akaike (1973) and defined by Eq. (25). For small sample sizes, Hurvich & Tsai (1989) recommend using a corrected version of the AIC, defined by Eq. (26).

$$AIC = -2ln(\hat{L}) + 2K \tag{25}$$

$$AIC_c = AIC + (2K^2 + 2K)/(n - K - 1) \tag{26}$$

In these equations, $\hat{L}$ denotes the logarithm of the maximum likelihood estimate, K the number of parameters in the model, and n the number of time series data points. According to Ding et al. (2018), the use of the AIC or AICc not only avoids the need to further split the training data into a validation set for model selection, as depicted in *Figure 3* a), but also prevents the choice of a model that overfits the data.

[1] Other hyperparameters may exist depending on the particular architecture of the ANN and on its training process

For ANNs, as for most data mining methods, it is usually recommended to save a portion of the training data for validation purposes, as depicted in *Figure 3* b). This happens because, different sets of hyperparameters are evaluated against the validation set so that the best architecture, i.e. the set of "optimal" hyperparameters, can be chosen (Brownlee, 2017a; Barry-Straume et al., 2018). In this context, the best model is the one that minimizes the loss function both in the training and validation sets which, as previously stated, typically is the MSE for time series forecasting (Géron, 2019).

Finally, after the best model has been found, its performance must be evaluated on the test set through the computation of predictive accuracy measures. The usefulness of this procedure is two-folded, i.e. besides the computed measures being a representative indicator of the accuracy of the model in real forecasts, they also allow the comparison of different algorithms, for example, an ARIMA model to a LSTM (Brownlee, 2017a; Nau, 2019). According to Hyndman & Koehler (2006), the most commonly used accuracy metrics to compare different methods on the same dataset are based on squared errors or on the absolute error, more concretely, the mean squared error (MSE) in Eq. (12), the root mean square error (RMSE) in Eq. (27), the mean absolute error (MAE) in Eq. (28) and the median absolute error (MdAE) in Eq. (29).

$$\text{RMSE} = \sqrt{MSE} \tag{27}$$

$$\text{MAE} = \text{mean}(|e|) \tag{28}$$

$$\text{MdAE} = \text{median}(|e|) \tag{29}$$

In all these equations, $e = y - \hat{y}$, and y is the actual value while $\hat{y}$ is the predicted value.

2.3.4. Forecasting

When producing actual forecasts there are two additional aspects that must be considered, the forecast horizon and the underlying uncertainty of the forecasts.

Regarding the forecast horizon, there are two possible scenarios: one-step-ahead forecasting, where $H = 1$, and multi-step forecasting, where $H > 1$, in Eq. (6). Single-step forecasting is the default behaviour of most forecasting methods. Nonetheless, most practical applications, require forecasts for longer term horizons. Thus, several methods that can be used to generate multi-step forecasts have been proposed, from which the recursive method is the most simple and intuitive one (Kline, 2004; Sorjamaa et al., 2007). As described by Sorjamaa et al. (2007), this method simply adds one-step-ahead forecasts as input to predict the immediate next unknown future value, in a recursive fashion, until the end of the forecast horizon, H, is reached. It is the standard approach used to produce multi-step forecasts for ARIMA models and it can be extended to ANNs (Graves, 2013; Nau, 2019).

Forecasting is an inherently uncertain activity. As such, in addition to the point forecasts, many authors emphasise the need and usefulness of also producing prediction intervals that reflect the degree of uncertainty underlying the point forecasts (Chatfield, 2002; Hanke & Wichern, 2014). Due to its importance, it is a well-studied subject in the context of ARIMA models whose prediction intervals are computed based on the standard deviation of the residuals, that are assumed to be uncorrelated and normally distributed (Hyndman & Athanasopoulos, 2018). Data mining models in general, however, do not account for the uncertainty associated with their predictions, although some alternative approaches have been proposed to overcome this problem (Shrestha & Solomatine, 2006). For ANNs in particular, Gal & Ghahramani (2016) have proposed a new method of computing prediction intervals that builds on the already mentioned concept of dropout. These authors suggest extending the use of the dropout technique from the ANN training period to the prediction period. Given that dropout randomly omits neurons, using it at the prediction stage leads to some variation in the predicted values if several predictions for the same time steps are made. It is, hence, possible to take advantage of this phenomenon by making N simulations and calculating their mean value along with the respective standard deviation that is, then, used to compute the prediction intervals.

3. METHODOLOGY

This project closely follows the design science research methodology (DSRM) proposed by Peffers et al. (2007), which unifies and builds on previous contributions from the Design Science Research (DSR) literature, such as the guidelines presented by Hevner et al. (2004) on the application of design science to information systems research. According to Hevner et al. (2004), DSR is, at its core, a problem-solving oriented process whose primary goal is to produce novel and useful artifacts meant to solve an identified relevant practical problem. As defined by March & Smith (1995), such artifacts can have different levels of abstraction and take the form of constructs, methods, models or instantiations. While constructs provide a language to describe a problem within its domain, models are representations of real-life situations expressed through relationships among constructs. Methods, in turn, define the steps of a problem-solving process towards a solution based on constructs and models. Finally, instantiations instrumentalize constructs, models and methods into working systems (Gregor & Hevner, 2013; March & Smith 1995). In addition, a crucial factor inherent to DSR is the demonstration and thorough evaluation of the produced artifact in terms of its utility in serving its intended purpose (Hevner et al. 2004; Sonnenberg & Brocke, 2012). The choice of a particular evaluation method is influenced by the type of artifact produced. Nevertheless, as asserted by Peffers et al. (2012), technical experiments are the dominant evaluation method for any given artifact. This form of evaluation involves simulating the execution of the artifact with real or artificial data in a controlled environment to assess if it works as expected and fulfils its main goal (Hevner et al. 2004; Peffers et al. 2012).

That being said, the DSRM established by Peffers et al. (2007) encompasses six core activities that will be described and framed in the context of the present project. The first activity, which comprises the problem identification and motivation, has been carried out in *section 1.1*. In summary, the identified problem was the inherent difficulty of time series forecasting and the need of new automatic methods to address it in the recent paradigm of big volumes of data, whereas the motivation behind it was the ascertainment of the great importance of time series forecasting across several domains, including business and industry. The second activity involves the definition of objectives for a solution regarding the identified problem. According to Peffers et al. (2007), such objectives should be inferred from the problem specification and the knowledge of the state of the problem. The overall objectives were introduced in *section 1.2* and are further extended in *section 4* into more practical and concrete goals. The third activity entails the design and development of the artifact which is described in detail in the following *section 4*. Moving from the pre-defined objectives to the actual development of the artifact requires knowledge of the useful theory that can lead to a solution (Hevner et al., 2004; Peffers et al., 2007). The theory supporting the present project has been thoroughly analysed in the literature review (*section 2*). The artifact produced in the project can be regarded as an instantiation, as it results from the confluence of constructs, models and methods. The fourth and fifth activities are the artifact demonstration and evaluation, respectively. Demonstration involves using the artifact on one example of the problem whilst evaluation is considered a more formal activity that involves the assessment of the effectiveness of the artifact in solving the problem it was intended for, by comparing the results of the demonstration with the established objectives (Hevner et al., 2004; Peffers et al., 2007). Due to their interdependency, the results of both these activities are presented in the results section (*section 5*). Finally, the sixth activity of the DSRM is the communication of the problem and its importance, as well as the utility and effectiveness of the created artifact and it is accomplished through the present document.

4. PROJECT DEVELOPMENT

The thorough literature review provided the foundations required to transform the previously established global objectives, into clearer and somewhat more practical objectives that could, more concisely, guide the project execution.

Therefore, the first devised objective towards the achievement of the main goal of forecasting algorithm automation, was to fully understand which elements form the basic building blocks of an algorithm and how they are related to the previously introduced models. In this context, it was useful to adopt the view, presented by Fayyad et al. (1996) and Chung & Gray (1999), of any algorithm consisting of three common components: a model representation, an evaluation criterion and a search method. According to these authors, the model representation is an artificial construct used to describe the patterns in the data, with each representation having its own assumptions. Regarding the models intended to be implemented, it was inferred from the literature review that ARIMA models make somewhat strong assumptions about the data, whereas ANNs being data-driven models are less restrictive and should, in theory, be able to learn any pattern. The second component, an evaluation criterion, is the means by which the goodness-of-fit of each model is evaluated. As pointed in *section 2.3.3*, the preferred criterion for ARIMA model selection, when estimated by MLE, is the AIC which measures the goodness-of-fit of the model and, simultaneously, penalizes the model complexity with the aim of preventing the overfitting problem. On the other hand, the preferred criterion for ANNs in time series forecasting is the MSE and any possible overfitting is handled by regularization techniques such as dropout. Lastly, the third component, the search method, is linked to a model's hyperparameters whose correct identification, as aforementioned, requires knowledge about the data at hand and the model being used.

They are, however, the key component for algorithm automation. As the name of the component suggests, the search method involves finding the best hyperparameters by testing several combinations and choosing the one that leads to the best results. According to Liashchynskyi & Liashchynskyi (2019), two types of search tecnhiques possible to implement in order to automate the search process are the grid search and the random search. Given a global set of hyperparameters, the grid search performs a cartesian product over the set, i.e., it tries all the possible combinations of hyperparameter subsets, whilst the random search randomly picks combinations of hyperparameter subsets up to a pre-specified maximum number of iterations. Hence, although the grid search method is guaranteed to lead to the optimal solution – from the provided hyperparameter space – it may be intractable in high dimensional spaces due to its high computational cost and, in such cases, random search comes as an efficient alternative that usually leads to good enough results (Liashchynskyi & Liashchynskyi, 2019; Bergstra & Bengio, 2012). In short, the rule of thumb would be to use grid search for low dimensional hyperparameter spaces and the random search otherwise.

The next practical goal was to search for any already existing algorithms related to the proposed system and analyse their inner workings. Not surprisingly, given its huge popularity and the many years that have passed since its introduction, there are some algorithms addressing the the Box-Jenkins method automation. In fact, there has always been a significant interest in automating such method, as suggested by the existence of proposals to do so dated back as far as the early 80's (Hopwood, 1980; Hill & Woodworth, 1980). Nonetheless, the state-of-art algorithm for automatic ARIMA forecasting is the one proposed by Hyndman & Khandakar (2008), implemented in the R programming language and adapted by some other languages and even by commercial packages. From the analysis of this algorithm in combination with the knowledge acquired through the literature review, it was possible to understand that the Box-Jenkins models could not be automated by blindly applying a search method to its hyperparameters without taking into account some theoretic aspects as, not doing so, would lead to results inconsistent with their theoretical underpinnings. More concretely, it is required to identify the correct number of necessary differences to make the time series stationary before proceeding to any hyperparameter search. This occurs because, as explained by Hyndman (2013), differencing involves the loss of a number of data points equivalent to the order of the differencing and, as such, the AIC for models with a dissimilar number of differences is computed based on different sample sizes thus making its comparison invalid. In the original paper, Hyndman & Khandakar (2008) propose the use of successive unit root tests for determining the number of differences, more specifically, the CH test for seasonal differences and KPSS for trend related differences. Nevertheless, it was possible to find out, through careful inspection of the algorithm, that in its more recent versions, instead of the CH test, the index of seasonal strength, F_S in Eq. (2), is the preferred measure to determine the required number of seasonal differences. Hence, every time the F_S exceeds a threshold value, defined as 0.64 by the authors, it means that a seasonal difference is required. Finally, after the number of differences has been inferred it is, then, possible to employ a search method to find the orders of the AR and MA polynomials. Moreover, it is important to add that the choice between a seasonal or non-seasonal ARIMA is left up to the user.

Conversely, there are not any specific authoritative algorithms for ANNs automation, which is explained by the already stated fact that these models learn patterns directly from the data without making many assumptions, thus allowing their hyperparameters to be automatically found through one of the abovementioned search methods, without compromising the validity of their results. There is, however, one additional issue, not exactly related to their automation but simply to their supervised learning nature and their general usage with time series data, which is the requirement for transforming the time series into a specific format before they are passed into the neural network models. More specifically, time series must be converted into windows of sequences and their corresponding labels. This can be achieved by means of a sliding window technique (*Appendices 1 and 2*).

At this point, it was possible to already have a clearer idea of the practical requirements needed to achieve the proposed main goal of the project, especially in what concerned the ARIMA models. Regarding the ANNs there were still some unknowns but that were thought to be more easily cleared up during the actual development of the system. Hence, with the general aspects of algorithm automation outlined, the next step was to find the Python libraries that could be used to leverage the system development. First of all, it is worth to mention that the main reason that motivated the choice of Python as the programming language for this project was its huge diversity of libraries, that greatly extend its base functionalities, in combination with its simple syntax and multi-paradigm support (Wu, 2019; Cass, 2019). After a thorough search, the libraries deemed as relevant for the project and from which some functionalities ended being used, as will be further explained, were:

- Pandas, a very useful library for data analysis and manipulation that presents important functionalities regarding datetime objects (McKinney, 2010);
- NumPy, which offers a very efficient data structure in the form of n-dimensional arrays, great for faster computations (Harris et al., 2020);
- Matplotlib, a very flexible library for graphics generation capable of creating any relevant graph for the project (Hunter, 2007);
- Scikit-learn, the most prominent python library for machine learning related functionalities (Pedregosa et al., 2011);
- Statsmodels, that takes care of statistical computations, such as model estimation, and offers important statistical hypothesis tests (Seabold & Perktold, 2010);
- Pytorch, which is a very powerful library for neural network architectures development. Its basic data structure in form of tensors, also n-dimensional, can leverage the power of Graphics Processing Unit (GPU) acceleration to perform computations even faster than those attained through Numpy's arrays. Moreover, it has the capability of keeping track of all the performed tensor operations and automatically computing the correct partial derivatives for each of them, thus greatly simplifying the backpropagation process during training (Paszke et al., 2017).

From here, the development stage could finally take place. The first concern was to implement the core functionality of the system, i.e., the ARIMA and RNN models. Since the mechanics of the ARIMA models were better understood, their implementation was the first step in the development process. Luckily, an already existing data structure in the Statsmodels library could be used as base for the model fitting and parameter estimation part which would require a heavy mathematical background otherwise. Nonetheless, all the logic underlying the application of the statistical tests and measures to determine the need of differencing as well as the search loop for an optimal model had to be independently developed. Some snippets of the code can be found in *Appendices 3, 4,* and

5 [2]. In short, the workflow of the algorithm is based on the one previously mentioned, and it also requires that the user decides if the model should be seasonal or non-seasonal, however, after all the components of the system were built, the choice between the two becomes a very easy process as it will be seen further ahead. If a non-seasonal model is chosen, the algorithm works by first determining if the inputted time series is stationary and, if it is not, it finds the necessary number of d differences in order for it to become stationary. Then, it only needs to find the p and q orders of the AR and MA polynomials, respectively. It does so by maintaining d fixed at the found number of differences and by trying all the possible combinations of the p and q hyperparameters up to a pre-defined limit, whose default value is set to 3, but that can be changed by the user. Finally, it chooses the combination of hyperparameters for which the AIC or the AICc is smaller. For a seasonal model, the procedure is very similar, except that the algorithm has also to determine the number of seasonal differences, D, and the P and Q orders of the corresponding seasonal AR and MA polynomials (*Appendix 6*). It is worth to highlight that some theoretical aspects as, for instance, first finding the seasonal differences and only after finding the non-seasonal differences, in the case of seasonal models, were taken into account in order to ensure the correctness and validity of the found models.

Next, the implementation of the ANN models was fairly more challenging, and it was done in an iterative fashion. First, the models decided to be implemented were the traditional RNN, the LSTM and the GRU. Once again, in this case the existing python libraries were very helpful, more specifically Pytorch, which had already available an implementation of each of the memory units, thus, avoiding the cumbersome process of coding them from scratch. It was, however, necessary to define several architectural aspects of the ANNs, as well as to define the forward pass of the data through each of them. Given the fact that the data that flows through neural networks is arranged in the form of tensors, time series data must be converted to 3D tensors before being passed to any of them. This could be achieved through the combination of the referred sliding window technique plus some additional functions of Numpy and Pytorch. These 3D tensors are subject to several transformations throughout the forward pass of each ANN until the output layer is reached. Such transformations had to be carefully defined, in

[2] Due to its extension, it is not possible to reproduce the entire code in the Appendices section. The full code can be found at: **https://github.com/joana94/intelligent-ts-predictor**

order to guarantee that the ANNs would produce consistent results. Furthermore, it was needed to understand the correct way of initializing the hidden states for all the ANNs, and the cell state in the case of LSTMs, and to decide which parts of their information should be passed to the output layer. The final decision was to use the information from all hidden states in the output layer as the default option since it demonstrated better results, but also to offer the alternative of allowing the user to choose to only use the information from the most recent hidden state.

At the end, all the ANNs ended up being implemented with a similar architecture, i.e., with an input layer, two stacked recurrent layers, a linear output layer and a dropout layer between the recurrent layers (*Appendices 7* and *8*). The reason behind the decision of using two fixed recurrent layers and a dropout layer in between instead of giving the option of these factors being part of the searchable hyperparameter grid, was to guarantee that after the models were trained, they could generate prediction intervals along with the point forecasts at inference time. The implementation of the prediction intervals was inspired by the method presented in *section 2.3.4*, being the variability associated with the dropout technique used to generate several simulations of the future values and their corresponding mean and standard deviation used to compute the intervals' upper and lower bounds, whose coverage probabilities were assumed to follow a normal distribution (*Appendix 9*). Thus, the only searchable hyperparameter regarding the neural networks' architecture ended being the size of the hidden dimension of the recurrent layers, i.e., the number of hidden neurons that they possess and which greatly influence the representational power of the neural networks and their capability of learning patterns.

The main hyperparameters to be searched in the neural networks are related to their training. The training process is what enables a neural network to progressively capture the patterns contained in the data and, as such, is a key part in their use for time series forecast generation. It is during training that many decisions have to be made, namely, how many samples of sequences and labels, known as the batch size, should be simultaneously passed to the network at each iteration for computing the gradients and updating the networks' weights. There is also the decision of whether to shuffle these samples or not. For independent and identically distributed random variables, shuffling is basically a pre-requisite as it prevents the neural networks from incorrectly learning patterns that simply resulted from the way that the samples were ordered when they were passed to them. In the case of time series, although the order of the observations is extremely important and, in principle, should not be changed, doing so sometimes improves pattern learning, especially in the case of long time series as has been empirically asserted by some researchers (Peralta et al., 2009). In addition, other very important hyperparameter to be searched, that highly impacts the learning process, is the learning rate of the optimizer when performing gradient descent. The learning rate controls how fast the weights are updated by gradient descent towards the minimum of the loss function and it really has a huge impact in the ability of the ANNs to learn patterns. So, in sum, the resulting hyperparameters to be automatically optimized in the implemented RNNs, ended being the hidden dimension, the batch size, the shuffling and the learning rate (*Appendix 10*). The selected decision criterion for finding the best set of hyperparameters was the validation loss with the minimum MSE. The creation of the validation set was decided to be computed as a fraction of the training set, with the default value of 10% but with the possibility of being controlled by the user. This splitting logic was incorporated as the first step of the training loop. Finally, both grid and random search methods were implemented and it is up to the user to choose which one to use to find the best set of hyperparameters depending on the dataset at hand (*Appendix 11*).

Two additional remarks about the ANNs were the fact that when testing their functioning, it was concluded that the time series data had to mandatorily go through an extra pre-processing step of normalization, before being passed to the ANNs or otherwise the models would not work properly, and the fact that in some cases the gradients would become very unstable during training which led to the inclusion of the gradient clipping technique in the training loop. The chosen normalization technique was the min-max scaling, which rescales the values of the time series to values between 0 and 1, whereas the defined threshold value for gradient clipping was 1 and it significantly aided to overcome the exploding gradients problem.

Finally, after the automation of the forecasting models had been achieved, they could be now used as foundation to build a support system able to guide the user through the entire forecasting process while abstracting its inherent complexity. As depicted in *Figure 2*, this would include the process of ingesting time series data, splitting them into training and testing sets, comparing the performance of competing models in the testing set and, lastly, choosing the best performing model to produce real forecasts into the future. To accomplish this, a modular system, whose overall functionality and outputs are summarized in *Figure 7*, was developed.

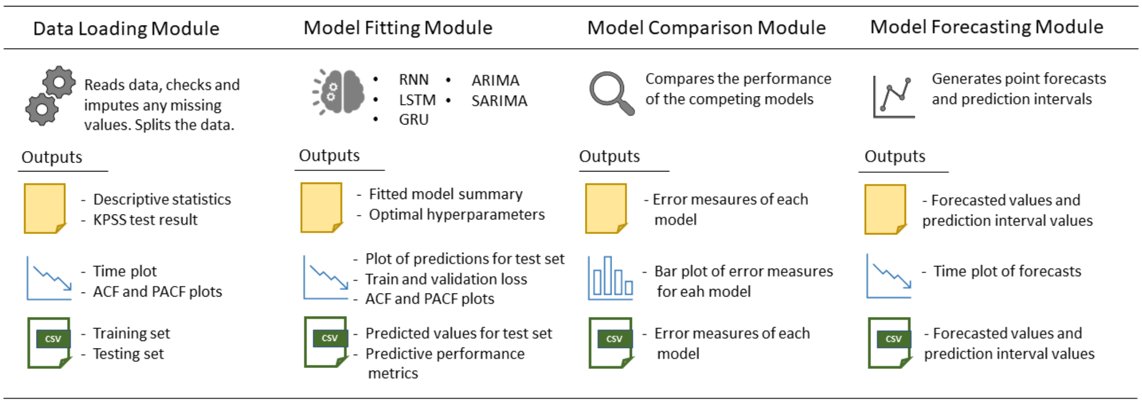

Data Loading Module
Reads data, checks and imputes any missing values. Splits the data.
Outputs
- Descriptive statistics
- KPSS test result
- Time plot
- ACF and PACF plots
CSV
- Training set
- Testing set
Model Fitting Module
RNN
LSTM
GRU
ARIMA
SARIMA
Outputs
- Fitted model summary
- Optimal hyperparameters
- Plot of predictions for test set
- Train and validation loss
- ACF and PACF plots
CSV
- Predicted values for test set
- Predictive performance metrics
Model Comparison Module
Compares the performance of the competing models
Outputs
- Error mesaures of each model
- Bar plot of error measures for eah model
CSV
- Error measures of each model
Model Forecasting Module
Generates point forecasts and prediction intervals
Outputs
- Forecasted values and prediction interval values
- Time plot of forecasts
CSV
- Forecasted values and prediction interval values

Figure 7. Overview of the system modules and their outputs

In more detail, this system is comprised by the following components:

- The data loading module, which takes care of the time series data reading and the checking of their values. The reading function expects a csv file containing a date column in a valid format, which is automatically detected, and, evidently, a column containing the respective values for each timestamp. Given the fact that any missing dates negatively impact the performance of the models, especially that of ARIMA models as ANNs are more robust to such scenario, it automatically checks for any missing dates and, if any are detected, it adds them to the original time series and also the corresponding missing values, through linear interpolation. The great flexibility of the python Pandas library to handle datetime objects was a valuable aspect that aided in the development of this module. Additionally, it produces two extra csv files, one containing the training set and the other containing the test set, which are used by the subsequent modules, and two reports. The first report provides some descriptive statistics of the time series including the trend and seasonal strength measures which give the user an hint about the possible presence of trend and seasonal patterns, and the other report contains the results of the KPSS test thus providing a prior indication of the stationarity of the time series. These are accompanied by the time plot, which also allows to visually detect any patterns, along with the ACF and PACF plots that help to assess the existence of any significant autocorrelations (*Appendix 12*). All the plots are generated through the Matplotlib library.

- The model fitting module, which is the one that makes most use of the previously developed automatic algorithms. It fits a model, chosen by the user through a configuration file, to the time series training set and then the automatic algorithms do all the work of finding the best hyperparameters and estimating the parameters for the inputted time series data. If a statistical model is chosen, it is fitted to the data after computing the required differences. Its optimal hyperparameters are found by minimizing the AIC or AICc, whilst the associated parameters are then estimated by Maximum Likelihood. On the other hand, if an ANN model is chosen, the dataset is first normalized through the min-max scaling functionality offered by the Scikit-learn library, and transformed into windows of sequences and their corresponding labels, prior to the model training. Its deemed optimal hyperparameters are those which produce the lowest MSE in the validation set, whereas the corresponding parameters are estimated by gradient descent also towards the minimum MSE but on the training set. The outputs of this module are very important and consist of reports and csv files containing the summary of the fitted/trained models, the set of the found optimal hyperparameters, the predictions of the models for the test set and the predictive performance metrics – the MSE, RMSE, MAE and MdAE – computed for those predictions, which are the key element to compare different models. It also produces graphics of the predicted values compared against the real values. Finally, for ARIMA models an additional set of plots, intended to address the diagnostics step of the Box-Jenkins method, is generated for the residuals of the model and comprise their ACF and PACF plots, that allow to check the existence of any remaining autocorrelations, and their density plot, which enables to verify if they follow a normal distribution. For the ANNs, a plot showing the training and validation set losses is provided and its usefulness lies on the fact that it provides an indication of how many iterations should be used when training the model on the entire time series (*Appendix 13*).
- The model comparison module, which is just a utility and is not an essential part for the correct functioning of the system. It simply grabs the predictive performance results of each of the fitted competing models, and produces a report and a csv file indicating which one made the lowest error accompanied by a bar plot that displays the same information but in a more intuitive way. By using this module, the user avoids the need to manually check the outputted reports of each model and can

quickly and easily compare all the different types of models, in order to decide which one is really the best for the time series at hand (*Appendix 14*).

- The forecasting module, which, as expected, produces point forecasts accompanied by their prediction intervals. It refits the chosen model to the entire dataset with the found optimal set of hyperparameters and produces forecasts for the number of periods along with the predictions' confidence level, pre-specified by the user in the configuration file. Forecasts of both ARIMA and ANN models are produced using the recursive method, where each of the predicted values are reused as inputs for the subsequent predictions. The forecasts are outputted in the form of report and a csv file, which can be used for further analysis. These are accompanied by a time plot that allows to visualize how the values are expected to change in the future, and how confident the models are on their predictions (*Appendix 15*).

In addition, before any of the referred modules can be used, an initial script responsible for creating the directory structure for each forecasting project, must be executed so that every module output is placed in the correct folder (*Appendix 16*).

5. RESULTS

The designed artifact was evaluated through experimentation, a process that was carried out in several steps. First, the automated algorithms were iteratively assessed during their development in order to assure their correct implementation. The goal was to verify that both ARIMA and ANNs could select the best hyperparameter set from the available hyperparameter grid and, simultaneously, to confirm if the ANNs were able to actually learn any patterns from the data. Therefore, to perform this evaluation an artificial dataset was used, more concretely, a sine wave which due to its simplicity would allow to test the algorithms in several occasions in a short amount of time and, due to its periodic pattern, would also allow to check the ability of the ANNs to learn its pattern[3]. This initial experiment lead to very positive results as shown by the low errors that any of the models made on the artificial dataset in *Table III*. Note that only the seasonal ARIMA error is reported, as fitting a non-seasonal ARIMA model to a dataset of a periodic nature like

[3] Furthermore, the used artificial dataset has also the advantage of being easily reproducible, thus facilitating the replication of the experiment by other researchers.

this one would be nonsensical[4]. Moreover, the extremely low error reported for the SARIMA model is explained by the fact that, in some circumstances, its forecasting function can be obtained as a mixture of sines and cosines, hence, originating this almost perfect prediction for the test data (Box et al., 2016).

TABLE III. SINE WAVE PREDICTION RESULTS

	SARIMA	**Traditional RNN**	**LSTM**	**GRU**
MSE	1.373644e-07	0.030167	0.007415	0.020094
RMSE	3.706270e-04	0.173688	0.086109	0.141754
MAE	2.768829e-04	0.150737	0.077472	0.123896
MdAE	2.425354e-04	0.135817	0.085139	0.120943

An additional experiment, with another artificial dataset, was carried out to further assess the capacity of the neural networks to learn patterns. This dataset comprised, once again, a sine wave but this time with some gaussian noise added to it. As expected, the error increased slightly but the results were still pretty remarkable as shown in *Table IV*, meaning that the relevant patterns were still captured.

TABLE IV. SINE WAVE WITH GAUSSIAN NOISE PREDICTION RESULTS

	SARIMA	**Traditional RNN**	**LSTM**	**GRU**
MSE	0.046699	0.131497	0.064780	0.071001
RMSE	0.216100	0.362624	0.254519	0.266461
MAE	0.177564	0.284824	0.208243	0.214373
MdAE	0.136900	0.258987	0.202693	0.185101

The use of these two artificial datasets was very important as it not only made clear that both algorithms were choosing the best set of hyperparameters, but it also showed that, even in the presence of noise distorting the target signal, the RNNs were able to approximate the data generating function, thus capturing the relevant patterns and achieving very competitive results when compared to the SARIMA model, with the

[4] In addition, the correct functioning of the SARIMA model implies the also correct functioning of ARIMA, as both rely on the same automatic algorithm.

LSTM being the best performing ANN for both datasets. Furthermore, as previously explained, this experiment also allowed to conclude the necessity of further tweaking the ANN algorithms by incorporating the min-max normalization as a pre-processing step and by clipping the gradients in the training loop to avoid exploding gradients.

The next experiment intended to check if the entire system built on top of the automated algorithms also worked as expected. In order to do so, two publicly available real datasets[5], with diverging characteristics regarding length and frequency, were used in an end-to-end time series forecasting task. This allowed to demonstrate and assess the correct functioning of each of the modules and the performance of the system as a whole.

The dataset 1 comprised a short time series with less than two hundred observations and monthly frequency, encompassing a clear seasonal pattern, and did not have any missing dates or values. Each module of the system was able to produce the expected outputs for this dataset, and the automatic algorithms were able to effortlessly find the best hyperparameters for each model. The best performing model on this dataset was the GRU, followed by the LSTM, SARIMA and traditional RNN. The MSE computed in the test set, as well as the found best hyperparameter set for each model is reported in *Table V*. Furthermore, the time plots outputted by the model fitting and the forecasting module for the two best performing models, GRU and LSTM, are shown in *Figure 8*.

The dataset 2 consisted of a long time series, with more than three thousand observations and daily frequency, also encompassing a seasonal pattern, and with some missing dates and values. Once again, each module of the system worked as expected, namely by detecting the missing data and filling it appropriately, and by producing the correct outputs. For this dataset the automatic algorithms, however, required a much longer fitting/training time and were slower to converge to a solution, with SARIMA being the model that struggled the most, which was an expected outcome since this type of model does not scale well for big time series datasets with a high frequency. In fact, SARIMA simply cannot model data with higher than daily frequency, which has prevented the assessment of the system in a more complex hourly dataset. Anyway, at the end all models were able to fit dataset 2 and actually produced similar results as can be seen in *Table V*, with the LSTM being the best performing one.

[5] Dataset 1**: https://www.kaggle.com/chirag19/air-passengers**
Dataset 2: **https://github.com/jbrownlee/Datasets/blob/master/daily-max-temperatures.csv**

TABLE V. SUMMARY OF THE OPTIMAL HYPERPARAMETERS FOUND FOR EACH MODEL AND THE CORRESPONDING PREDICTIVE PERFORMANCE FOR EACH DATASET

	Models	Optimal Hyperparameters	MSE
Dataset 1	SARIMA	$p = 2, d = 0, q = 0, P = 1, D = 1$ and $Q = 2$	2693.78
	Traditional RNN	Batch size = 10, learning rate = 0.001, Hidden dimension = 512, Shuffle = True	3315.31
	LSTM	Batch size = 10, learning rate = 0.001, Hidden dimension = 512, Shuffle = True	2540.68
	GRU	Batch size = 10, learning rate = 0.01, Hidden dimension = 256, Shuffle = True	2231.89
Dataset 2	SARIMA	$p = 3, d = 0, q = 2, P = 0, D = 0$ and $Q = 2$	43.28
	Traditional RNN	Batch size = 60, learning rate = 0.001, Hidden dimension = 256, Shuffle = True	53.10
	LSTM	Batch size = 60, learning rate = 0.001, Hidden dimension = 512, Shuffle = True	42.83
	GRU	Batch size = 60, learning rate = 0.001, Hidden dimension = 512, Shuffle = True	43.63

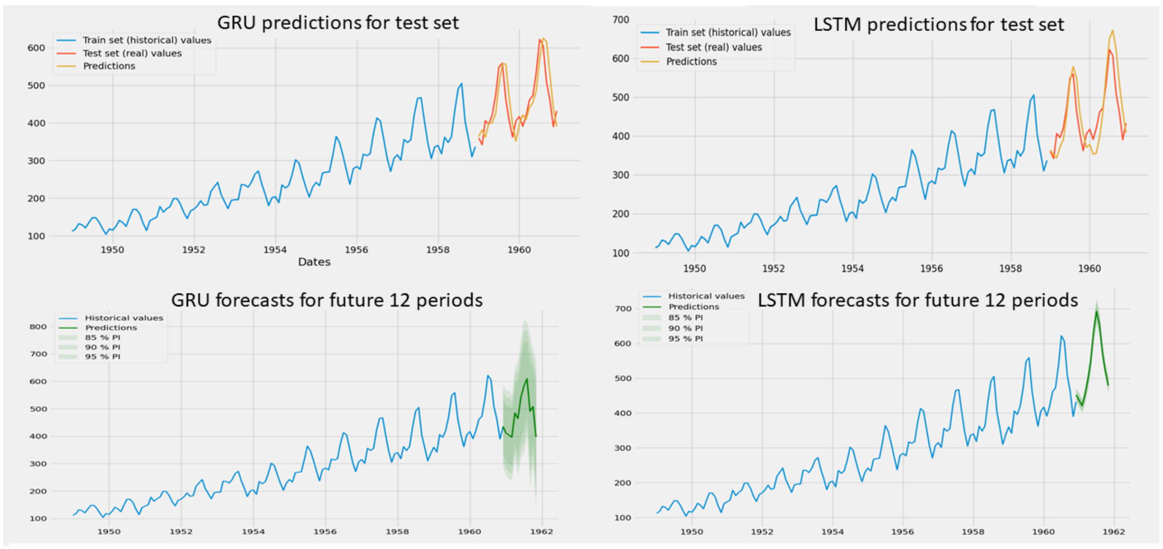


Figure 8. Forecasting module output for dataset 1. In the upper row of the image, we find the time plots outputted by the model fitting module for both GRU and LSTM models for dataset 1, where the predictions for the test set are plotted against the real values. In the bottom row, we find the time plots outputted by the forecasting model which show the forecasted values along with their prediction intervals.

As shown in *Figure 8*, the implemented RNNs are able to quantify the uncertainty of their predictions. The wide prediction intervals produced by the GRU imply a high uncertainty of the model about its predicted future values, in contrast with the very narrow prediction intervals produced by the LSTM which indicate a high confidence of the model regarding its predicted future values. Furthermore, from *Table V* it is possible to conclude that for both datasets, the automatic algorithm found that shuffling the time series data before feeding it to any of the ANNs would lead to the best results.

In sum, from the obtained results the RNN models revealed to be a competitive alternative to classical ARIMA models. Moreover, it should be noted that the optimal set of hyperparameters for the RNNs is chosen conditioned on the hyperparameter grid provided by the user. In the carried experiments the provided grid was considerably small, due to computational resources limitations, meaning that there is some probability that a potentially better performing model could be found if a larger grid was provided.

6. CONCLUSIONS, CONTRIBUTIONS, LIMITATIONS AND FUTURE WORK

For this project, a design science research approach was used to develop a purposeful artifact that addresses the identified problem of the need for more advanced automated data-driven forecasting algorithms, and their usage to reduce the underlying complexity of the time series forecasting process. In global terms, all the established objectives towards the development of the proposed artifact were successfully achieved. The final designed artifact can be considered an instantiation as it combines constructs, models and methods into a fully working system. From the models included in the system, the statistical ARIMA models are a well-studied subject in the context of time series forecasting, whereas the application of RNNs to time series forecasting is a fairly recent subject and remains an open topic requiring further research.

Regarding the contributions of the project, the experiments carried out to test the system suggested that RNNs are indeed capable of learning patterns from time series data and to extrapolate them into the future, actually leading to competitive results when compared to classical models, thus supporting the recent interest in using these models for time series forecasting. Moreover, in accordance with recent research, they also suggested a better performance of the gated RNNs, the LSTM and GRU, over the traditional RNN. The system also hinted at the possible suitability of shuffling time series data during the ANN training phase. Nevertheless, the main contribution of the project is the artifact itself and it is two-folded. First, it offers a unique fully automated tool capable of performing time series forecasting while including the most well-known statistical models, as well as some of the most recent and promising models for such task. This is useful either for the expert and non-expert user, as it provides a way of quick and easily fitting competing models and compare their performance while obtaining insightful reports and graphical representations. It also mitigates the burden of dealing with the most complex technical issues linked with timestamped data and of having to implement the models or to manually tune their hyperparameters, which are far from trivial tasks. On the other hand, given the fact that the system's code was made publicly available along with the fact that it was developed in an object oriented fashion, with its main functionalities being abstracted and encapsulated into classes, more experienced programmers can take advantage of this factor and break down the system into its core components, in order to use them as in a regular python package. This could lead to a great gain of flexibility and control over the model building process, especially for the neural networks, and would allow to further extend the offered capabilities.

The limitations faced in the project, were mainly related to computational performance constraints which precluded the use of bigger time series datasets with the system. The use of such datasets would possibly grant the system's evaluation in more representative conditions of the real-world time series forecasting scenario. Nevertheless, the performed experiments still indicate an overall good performance of the system, as long as the required computational resources are available.

Finally, some possible future work directions regarding the system's usability and performance improvements are provided. Concerning the usability aspect of the system, a clear enhancement would be to provide it with a more user-friendly interface in opposition to its current interface which relies on the operating system's command line. Concerning the performance aspect, the main factors to consider would be the assessment of more advanced methods for hyperparameter search, such as Genetic Algorithms, and the implementation of more complex recurrent neural network architectures capable of taking into the account the influence of other predictors in the target time series other than its own current and past values.

REFERENCES

Agrawal, R., Imieliński, T. and Swami, A. (1993). Mining association rules between sets of items in large databases. *ACM SIGMOD Record*, 22(2), pp.207–216.

Akaike, H. (1973). Information theory and an extension of the maximum likelihood principle. In B. N. Petrov and F. Csaki (Eds.), Second international symposium on information theory (pp. 267-281). Budapest: Academiai Kiado

Almeshaiei, E. and Soltan, H. (2011). A methodology for Electric Power Load Forecasting. *Alexandria Engineering Journal*, 50(2), pp.137–144.

Amadeo, K. (2008). *What Is the Business Cycle*? [online] The Balance. Available at: **https://www.thebalance.com/what-is-the-business-cycle-3305912** [Accessed 26 Mar. 2020].

Amari, S. (1993). Backpropagation and stochastic gradient descent method. *Neurocomputing*, 5(4–5), pp.185–196.

Armstrong, J.S. (2002). Introduction. In: *Principles of Forecasting: A Handbook for Researchers and Practitioners*. Springer US, pp.1–12.

Barry-Straume, J., Tschannen, A., Engels, D.W. and Fine, E. (2018). An Evaluation of Training Size Impact on Validation Accuracy for Optimized Convolutional Neural Networks. *SMU Data Science Review*, [online] 1(4). Available at: **https://scholar.smu.edu/datasciencereview/vol1/iss4/12/** [Accessed 22 Jul. 2020].

Bartlett, M.S. (1946). On the Theoretical Specification and Sampling Properties of Autocorrelated Time-Series. *Supplement to the Journal of the Royal Statistical Society*, 8(1), pp.27–41.

Bengio, Y., Simard, P. and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2), pp.157–166.

Bergstra, J. and Bengio, Y. (2012). Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research*, [online] 13(10), pp.281–305.

Available at: **https://www.jmlr.org/papers/v13/bergstra12a.html** [Accessed 7 Jun. 2020].

Box, G.E.P. and Cox, D.R. (1964). An Analysis of Transformations. *Journal of the Royal Statistical Society. Series B (Methodological)*, 26(2), pp.211–252.

Box, G.E.P. and Jenkins, G.M. (1970). *Time series analysis: Forecasting and control*. San Francisco: Holden-Day.

Box, G.E.P., Jenkins, G.M., Reinsel, G.C. and Ljung, G.M. (2016). *Time Series Analysis: Forecasting and Control*. 5th ed. Hoboken, New Jersey: John Wiley & Sons.

Bramer, M. (2016). *Principles of Data Mining*. *Undergraduate Topics in Computer Science*. London: Springer London.

Brownlee, J. (2017a). *What is the Difference Between a Parameter and a Hyperparameter?* [online] Machine Learning Mastery. Available at: **https://machinelearningmastery.com/difference-between-a-parameter-and-a-hyperparameter/** [Accessed 11 Jul. 2020].

Brownlee, J. (2017b). *What is the Difference Between Test and Validation Datasets*? [online] Machine Learning Mastery. Available at: **https://machinelearningmastery.com/difference-test-validation-datasets/** [Accessed 6 Jul. 2020].

Canova, F. and Hansen, B.E. (1995). Are Seasonal Patterns Constant Over Time? A Test for Seasonal Stability. *Journal of Business & Economic Statistics*, 13(3), pp.237–252.

Cass, S. (2019). *The Top Programming Languages 2019*. [online] IEEE Spectrum: Technology, Engineering, and Science News. Available at: **https://spectrum.ieee.org/computing/software/the-top-programming-languages-2019** [Accessed 18 Feb. 2020].

Chatfield, C. (2002). Prediction Intervals for Time-Series Forecasting. In: *Principles of Forecasting: A Handbook for Researchers and Practitioners*. Springer US, pp.475–494.

Chatfield, C. (2004). *The Analysis of Time Series: An Introduction*. 6th ed. Boca Raton, FL: Chapman & Hall/Crc.

Chen, K.-Y. and Wang, C.-H. (2007). A hybrid SARIMA and support vector machines in forecasting the production values of the machinery industry in Taiwan. *Expert Systems with Applications*, 32(1), pp.254–264.

Cho, K., van Merrienboer, B., Bahdanau, D. and Bengio, Y. (2014). On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. *arXiv:1409.1259 [cs, stat]*. [online] Available at: **https://arxiv.org/abs/1409.1259** [Accessed 3 Aug. 2020].

Chung, H.M. and Gray, P. (1999). Special Section: Data Mining. *Journal of Management Information Systems*, 16(1), pp.11–16.

Chung, J., Gulcehre, C., Cho, K. and Bengio, Y. (2014). *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*. [online] arXiv.org. Available at: **https://arxiv.org/abs/1412.3555** [Accessed 21 Aug. 2020].

Cleveland, R.B., Cleveland, W.S. and Terpenning, I. (1990). STL: A Seasonal-Trend Decomposition Procedure Based on Loess. *Journal of Official Statistics*, 6(1), pp.3–73.

Coenen, F. (2011). Data Mining: Past, Present and Future. *The Knowledge Engineering Review*, 26(1), pp.25–29.

De Gooijer, J.G. and Hyndman, R.J. (2006). 25 Years of Time Series Forecasting. *International Journal of Forecasting*, 22(3), pp.443–473.

Ding, J., Tarokh, V. and Yang, Y. (2018). Model Selection Techniques: An Overview. *IEEE Signal Processing Magazine*, 35(6), pp.16–34.

Elman, J.L. (1990). Finding Structure in Time. *Cognitive Science*, [online] 14(2), pp.179–211. Available at: **https://onlinelibrary.wiley.com/doi/abs/10.1207/s15516709cog1402_1** [Accessed 7 Apr. 2020].

Esling, P. and Agon, C. (2012). Time-series data mining. *ACM Computing Surveys*, 45(1), pp.1–34.

Faloutsos, C., Flunkert, V., Gasthaus, J., Januschowski, T. and Wang, Y. (2019). Forecasting Big Time Series: Theory and Practice. In: *Proceedings of the 25th ACM*

SIGKDD International Conference on Knowledge Discovery & Data Mining. KDD ’19. pp.3209–3210.

Fayyad, U., Piatetsky-Shapiro, G. and Smyth, P. (1996). *Knowledge Discovery and Data Mining: Towards a Unifying Framework*. pp.82–88.

Fildes, R. and Makridakis, S. (1995). The Impact of Empirical Accuracy Studies on Time Series Analysis and Forecasting. *International Statistical Review / Revue Internationale de Statistique*, 63(3), pp.289–308.

Fu, R., Zhang, Z. and Li, L. (2016). Using LSTM and GRU neural network methods for traffic flow prediction. In: *31st Youth Academic Annual Conference of Chinese Association of Automation (YAC)*. pp.324–328.

Fu, T. (2011). A review on time series data mining. *Engineering Applications of Artificial Intelligence*, 24(1), pp.164–181.

Gallicchio, C., Micheli, A. and Pedrelli, L. (2019). Comparison between DeepESNs and gated RNNs on multivariate time-series prediction. *arXiv:1812.11527 [cs, stat]*. [online] Available at: **https://arxiv.org/abs/1812.11527** [Accessed 16 Aug. 2020].

Gal, Y. and Ghahramani, Z. (2016). Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. *arXiv:1506.02142 [cs, stat]*. [online] Available at: **https://arxiv.org/abs/1506.02142** [Accessed 12 Aug. 2020].

Géron, A. (2019). *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems*. 2nd ed. 1005 Gravenstein Highway North, Sebastopol, CA 95472: O’Reilly Media, Inc.

Glen, S. (2016). *Unit Root: Simple Definition, Unit Root Tests*. [online] Statistics How To. Available at: **https://www.statisticshowto.com/unit-root/** [Accessed 20 Apr. 2020].

Goodfellow, I., Bengio, Y. and Courville, A. (2016). Sequence Modeling: Recurrent and Recursive Nets. In: Deep Learning. [online] MIT Press, pp.367–415. Available at: **http://www.deeplearningbook.org** [Accessed 16 May 2020].

Graves, A. (2013). *Generating Sequences With Recurrent Neural Networks*. [online] arXiv.org. Available at: https://arxiv.org/abs/1308.0850 [Accessed 24 Jun. 2020].

Gregor, S. and Hevner, A.R. (2013). Positioning and Presenting Design Science Research for Maximum Impact. *MIS Quarterly*, 37(2), pp.337–355.

Gullo, F., Ponti, G., Tagarelli, A. and Greco, S. (2009). A time series representation model for accurate and fast similarity detection. *Pattern Recognition*, 42(11), pp.2998–3014.

Haldrup, N., Kruse, R., Teräsvirta, T. and Varneskov, R.T. (2013). Unit roots, non-linearities and structural breaks. In: N. Hashimzade and M.A. Thornton, eds., *Handbook of Research Methods and Applications in Empirical Macroeconomics*. Cheltenham: Edward Elgar Publishing, pp.61–94.

Hand, D.J. (1998). Data Mining: Statistics and More? *The American Statistician*, 52(2), pp.112–118.

Han, J., Kamber, M. and Pei, J. (2012). *Data Mining : Concepts and Techniques*. 3rd ed. 225 Wyman Street, Waltham, MA 02451, USA: Elsevier.

Hanke, J.E. and Wichern, D.W. (2014). *Business forecasting*. 9th ed. Harlow, Essex: Pearson.

Harris, C.R., Millman, K.J., van der Walt, S.J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N.J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M.H., Brett, M., Haldane, A., del Río, J.F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C. and Oliphant, T.E. (2020). Array programming with NumPy. *Nature*, 585(7825), pp.357–362.

Hevner, A.R., March, S.T., Park, J. and Ram, S. (2004). Design Science in Information Systems Research. *MIS Quarterly*, 28(1), pp.75–105.

Hill, G.W. and Woodworth, D. (1980). Automatic Box-Jenkins Forecasting. *The Journal of the Operational Research Society*, 31(5), pp.413–422.

Hochreiter, S. and Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8), pp.1735–1780.

Hopwood, W.S. (1980). On the Automation of the Box-Jenkins Modeling Procedures: An Algorithm with an Empirical Test. *Journal of Accounting Research*, 18(1), pp.289–296.

Hunter, J.D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, 9(3), pp.90–95.

Hurvich, C.M. and Tsai, C.-L. (1989). Regression and time series model selection in small samples. *Biometrika*, 76(2), pp.297–307.

Hyndman, R.J. (2011). Business Forecasting Methods. In: *International Encyclopedia of Statistical Science*. pp.185–187.

Hyndman, R.J. (2013). *Facts and fallacies of the AIC*. [online] Hyndsight. Available at: **https://robjhyndman.com/hyndsight/aic/** [Accessed 7 Sep. 2020].

Hyndman, R.J. (2014). *Seasonal periods*. [online] Hyndsight. Available at: **https://robjhyndman.com/hyndsight/seasonal-periods/** [Accessed 9 Feb. 2020].

Hyndman, R.J. and Athanasopoulos, G. (2018). *Forecasting: Principles and Practice*. 2nd ed. [online] Heathmont, Vic.: Otexts. Available at: **https://otexts.com/fpp2/index.html** [Accessed 13 Jan. 2020].

Hyndman, R.J. and Khandakar, Y. (2008). Automatic Time Series Forecasting: The Forecast Package for R. *Journal of Statistical Software*, 27(3), pp.1–22.

Hyndman, R.J. and Koehler, A.B. (2006). Another Look at Measures of Forecast Accuracy. *International Journal of Forecasting*, [online] 22(4), pp.679–688. Available at: https://www.sciencedirect.com/science/article/pii/S0169207006000239 [Accessed 28 Jun. 2020].

Jordan, M.I. (1997). Serial Order: A Parallel Distributed Processing Approach. *Neural-Network Models of Cognition - Biobehavioral Foundations*, 121, pp.471–495.

Jozefowicz, R., Zaremba, W. and Sutskever, I. (2015). An Empirical Exploration of Recurrent Network Architectures. In: *Proceedings of the 32nd International Conference on Machine Learning, in PMLR*. ICML'15. pp.2342–2350.

Karlik, B. and Olgac, A. (2011). Performance Analysis of Various Activation Functions in Generalized MLP Architectures of Neural Networks. *International Journal of Artificial Intelligence And Expert Systems (IJAE)*, 1(4), pp.111–122.

Karpathy, A. (2015). *The Unreasonable Effectiveness of Recurrent Neural Networks*. [online] Andrej Karpathy blog. Available at: **http://karpathy.github.io/2015/05/21/rnn-effectiveness/** [Accessed 16 Jul. 2020].

Keogh, E. and Kasetty, S. (2003). On the Need for Time Series Data Mining Benchmarks: A Survey and Empirical Demonstration. *Data Mining and Knowledge Discovery*, 7(4), pp.349–371.

Kolarik, T. and Rudorfer, G. (1994). Time series forecasting using neural networks. *ACM SIGAPL APL Quote Quad*, 25(1), pp.86–94.

Kritzman, M. (1994). What Practitioners Need to Know about Serial Dependence. *Financial Analysts Journal*, 50(2), pp.19–22.

Kumar, S., Hussain, L., Banarjee, S. and Reza, M. (2018). Energy Load Forecasting using Deep Learning Approach-LSTM and GRU in Spark Cluster. In: *2018 Fifth International Conference on Emerging Applications of Information Technology (EAIT)*. pp.1–4.

Kurgan, L.A. and Musilek, P. (2006). A Survey of Knowledge Discovery and Data Mining Process Models. *The Knowledge Engineering Review*, 21(1), pp.1–24.

Kwiatkowski, D., Phillips, P.C.B., Schmidt, P. and Shin, Y. (1992). Testing the Null Hypothesis of Stationarity Against the Alternative of a Unit Root. *Journal of Econometrics*, 54(1–3), pp.159–178.

Lawrence, S., Giles, C.L. and Tsoi, A.C. (1997). Lessons in Neural Network Training: Overfitting May be Harder than Expected. In: *Proceedings of the Fourteenth National Conference on Artificial Intelligence*. AAAI-97. AAAI Press, pp.540–545.

Lee, K., Booth, D. and Alam, P. (2004). Backpropagation and Kohonen Self-Organizing Feature Map in Bankruptcy Prediction. Neural Networks in Business Forecasting, Idea Group Publishing, pp.158–171.

Lever, J., Krzywinski, M. and Altman, N. (2016). Model selection and overfitting. *Nature Methods*, 13(9), pp.703–704.

Levich, R. and Rosario, R. (1999). *Alternative Tests for Time Series Dependence Based on Autocorrelation Coefficients.* [online] *NYU Stern*. Available at: **http://pages.stern.nyu.edu/~rlevich/wp/LR1.pdf** [Accessed 2 Apr. 2020].

Liao, T.W. (2005). Clustering of Time Series Data — A Survey. *Pattern Recognition*, 38(11), pp.1857–1874.

Liashchynskyi, P. and Liashchynskyi, P. (2019). Grid Search, Random Search, Genetic Algorithm: A Big Comparison for NAS. *arXiv:1912.06059 [cs, stat]*. [online] Available at: **https://arxiv.org/abs/1912.06059** [Accessed 7 Sep. 2020].

Li, L.-K., Pang, W.-K. and Yu, W.-T. (2004). Forecasting Short-Term Exchange Rates: A Recurrent Neural Network Approach. In: *Neural Networks for Business Forecasting*. Idea Group Inc, pp.195–212.

Lin, J., Keogh, E., Lonardi, S. and Patel, P. (2002). Finding Motifs in Time Series. In: *Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2nd Workshop on Temporal Data Mining. pp.53–68.

Liu, Z., Yu, J.X., Lin, X., Lu, H. and Wang, W. (2005). Locating Motifs in Time-Series Data. In: *PAKDD 2005: Advances in Knowledge Discovery and Data Mining*. Pacific-Asia Conference on Knowledge Discovery and Data Mining. pp.343–353.

Ljung, G.M. and Box, G.E.P. (1978). On a measure of lack of fit in time series models. *Biometrika*, 65(2), pp.297–303.

Lütkepohl, H. and Xu, F. (2010). The role of the log transformation in forecasting economic variables. *Empirical Economics,* 42(3), pp.619–638.

Mahalakshmi, G., Sridevi, S. and Rajaram, S. (2016). A Survey on Forecasting of Time Series Data. In: *2016 International Conference on Computing Technologies and Intelligent Data Engineering*. ICCTIDE'16. pp.1–8.

Makridakis, S., Spiliotis, E. and Assimakopoulos, V. (2018). Statistical and Machine Learning forecasting methods: Concerns and ways forward. PLoS ONE, 13(3).

March, S.T. and Smith, G.F. (1995). Design and natural science research on information technology. *Decision Support Systems*, 15(4), pp.251–266.

McKinney, W. (2010). Data Structures for Statistical Computing in Python. In: *Proceedings of the 9th Python in Science Conference*. [online] pp.51–56. Available at: **http://conference.scipy.org/proceedings/scipy2010/mckinney.html** [Accessed 19 Jul. 2020].

Minsky, M. and Papert, S. (1969). *Perceptrons: An introduction to computational geometry*. Cambridge, Mass.-London.

Montgomery, D.C., Jennings, C.L. and Kulahci, M. (2015). *Introduction to Time Series Analysis and Forecasting*. 2nd ed. Wiley-Interscience.

Moritz, S. and Bartz-Beielstein, T. (2017). imputeTS: Time Series Missing Value Imputation in R. *The R Journal*, 9(1), pp.207–218.

Nair, V. and Hinton, G. (2010). Rectified linear units improve restricted boltzmann machines. In: *ICML'10: Proceedings of the 27th International Conference on International Conference on Machine Learning*. ICML'10. pp.807–814.

Nau, R. (2019). *Statistical forecasting: notes on regression and time series analysis*. [online] Duke.edu. Available at: **https://people.duke.edu/~rnau/411home.htm** [Accessed 9 Feb. 2020].

Olah, C. (2015). *Understanding LSTM Networks*. [online] colah's blog. Available at: **http://colah.github.io/posts/2015-08-Understanding-LSTMs/** [Accessed 2 Jul. 2020].

Pascanu, R., Mikolov, T. and Bengio, Y. (2013). On the difficulty of training Recurrent Neural Networks. *arXiv:1211.5063 [cs]*. [online] Available at: **https://arxiv.org/abs/1211.5063** [Accessed 2 Jul. 2020].

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L. and Lerer, A. (2017). Automatic differentiation in PyTorch. In: *NIPS 2017 Autodiff Workshop*.

Patro, S.G.K. and Sahu, K.K. (2015). Normalization: A Preprocessing Stage. *arXiv:1503.06462 [cs]*. [online] Available at: **https://arxiv.org/abs/1503.06462** [Accessed 16 Jul. 2020].

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, É. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, [online] 12, pp.2825–2830. Available at: **http://jmlr.org/papers/v12/pedregosa11a.html** [Accessed 19 Jul. 2020].

Peffers, K., Rothenberger, M., Tuunanen, T. and Vaezi, R. (2012). Design Science Research Evaluation. In: *Design Science Research in Information Systems. Advances in Theory and Practice*. International Conference on Design Science Research in Information Systems. pp.398–410.

Peffers, K., Tuunanen, T., Rothenberger, M.A. and Chatterjee, S. (2007). A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems*, 24(3), pp.45–77.

Pelgrin, F. (2011). *Lecture 4: Estimation of ARIMA models*. [online] Available at: **https://math.unice.fr/~frapetti/CorsoP/Chapitre_4_IMEA_1.pdf** [Accessed 8 Apr. 2020].

Peralta, J., Gutierrez, G. and Sanchis, A. (2009). Shuffle design to improve time series forecasting accuracy. In: *2009 IEEE Congress on Evolutionary Computation*. pp.741–748.

Petropoulos, F., Makridakis, S., Assimakopoulos, V. and Nikolopoulos, K. (2014). 'Horses for Courses' in Demand Forecasting. *European Journal of Operational Research*, 237(1), pp.152–163.

Qin, L. and Shi, Z. (2006). Efficiently Mining Association Rules from Time Series. *International Journal of Information Technology*, 12(4), pp.30–38.

Quenouille, M.H. (1949). Approximate Tests of Correlation in Time-Series. *Journal of the Royal Statistical Society. Series B (Methodological)*, 11(1), pp.68–84.

Rakhlin, A., Shamir, O. and Sridharan, K. (2012). Making Gradient Descent Optimal for Strongly Convex Stochastic Optimization. *arXiv:1109.5647 [cs, math]*. [online] Available at: **https://arxiv.org/abs/1109.5647** [Accessed 12 Jul. 2020].

Ratanamahatana, C.A. and Keogh, E. (2004). Making Time-series Classification More Accurate Using Learned Constraints. In: *Proceedings of the 2004 SIAM International Conference on Data Mining*. pp.11–22.

Ratanamahatana, C.A., Lin, J., Gunopulos, D., Keogh, E., Vlachos, M. and Das, G. (2009). Mining Time Series Data. In: O. Maimon and L. Rokach, eds., *Data Mining and Knowledge Discovery Handbook*. Boston, MA: Springer, pp.1049–1077.

Remus, W. and O'Connor, M. (2002). Neural Networks for Time-Series Forecasting. In: J.S. Armstrong, ed., *Principles of Forecasting: A Handbook for Researchers and Practitioners*. Springer, pp.245–256.

Rey, T. and Wells, C. (2012). Integrating data mining and forecasting. *ORMS Today*, [online] 39(6). Available at: **https://www.informs.org/ORMS-Today/Public-Articles/December-Volume-39-Number-6/Integrating-data-mining-and-forecasting** [Accessed 24 Feb. 2020].

Robinson, D. (2017). *Why is Python Growing So Quickly?* [online] Stack Overflow Blog. Available at: **https://stackoverflow.blog/2017/09/14/python-growing-quickly/** [Accessed 21 Feb. 2020].

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6), pp.386–408.

Rumelhart, D.E., Hinton, G.E. and Williams, R.J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088), pp.533–536.

Salinas, D., Flunkert, V., Gasthaus, J. and Januschowski, T. (2019). DeepAR: Probabilistic Forecasting with Autoregressive Recurrent Networks. *International Journal of Forecasting*. [online] Available at: **https://arxiv.org/pdf/1704.04110** [Accessed 22 Jul. 2020].

Santos, U., Pessin, G., Costa, C. and Righi, R. (2019). AgriPrediction: A proactive internet of things model to anticipate problems and improve production in agricultural crops. *Computers and Electronics in Agriculture*, 161, pp.202–213.

Seabold, S. and Perktold, J. (2010). Statsmodels: Econometric and Statistical Modeling with Python. In: *Proceedings of the 9th Python in Science Conference*. [online] Available at: **https://conference.scipy.org/proceedings/scipy2010/pdfs/seabold.pdf** [Accessed 28 Jul. 2020].

Shalabi, L.A., Shaaban, Z. and Kasasbeh, B. (2006). Data Mining: A Preprocessing Engine. *Journal of Computer Science*, 2(9), pp.735–739.

Shmueli, G. and Lichtendahl, K.C. (2018). *Practical time series forecasting with R : a hands-on guide*. Axelrod Schnall Publishers.

Shrestha, D.L. and Solomatine, D.P. (2006). Machine learning approaches for estimation of prediction interval for the model output. *Neural Networks*, 19(2), pp.225–235.

Shumway, R.H. and Stoffer, D.S. (2017). *Time Series Analysis and Its Applications*. *Springer Texts in Statistics*. Cham: Springer International Publishing.

Siddiqa, A., Karim, A. and Gani, A. (2017). Big data storage technologies: a survey. *Frontiers of Information Technology & Electronic Engineering*, 18(8), pp.1040–1070.

Smolen, H.J. (2014). Development Of An Influenza Outbreak Forecasting Model Using Time Series Analysis Methods. *Value in Health*, 17(7).

Sonnenberg, C. and Brocke, J. (2012). Evaluation Patterns for Design Science Research Artefacts. In: *EDSS 2011: Practical Aspects of Design Science*. European Design Science Symposium. pp.71–83.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, [online] 15(56), pp.1929–1958. Available at: **https://jmlr.org/papers/v15/srivastava14a.html** [Accessed 3 Aug. 2020].

Strohbach, M., Daubert, J., Ravkin, H. and Lischka, M. (2016). Big Data Storage. In: J.M. Cavanillas, E. Curry and W. Wahlster, eds., *New Horizons for a Data-Driven Economy*. Springer, pp.119–141.

Sutskever, I., Vnyals, O. and Le, Q.V. (2014). *Sequence to Sequence Learning with Neural Networks*. [online] arXiv:1409.3215v3 [cs.CL]. Available at: **https://arxiv.org/pdf/1409.3215.pdf** [Accessed 12 Jul. 2020].

Tahmassebpour, M. (2017). A New Method for Time-Series Big Data Effective Storage. *IEEE Access*, 5, pp.10694–10699.

Tan, P., Steinbach, M. and Kumar, V. (2006). *Introduction to Data Mining*. San Francisco: Pearson Education.

Taylor, S.J. and Letham, B. (2017). Forecasting at Scale. *The American Statistician*, 72(1), pp.37–45.

Teng, M. (2010). Anomaly detection on time series. In: *2010 IEEE International Conference on Progress in Informatics and Computing*. PIC. IEEE Xplore, pp.603–608.

Touretzky, D.S. and Pomerleau, D.A. (1989). What's hidden in the hidden layers? *BYTE*, 14(8), pp.227–233.

VanderPlas, J. (2018). *Working with Time Series*. [online] Pythonic Preambulations. Available at: **https://jakevdp.github.io/PythonDataScienceHandbook/03.11-working-with-time-series.html** [Accessed 2 Mar. 2020].

Vaughan, J. (2020). *IoT Developers May Need Time Series Data Analysis Skills*. [online] IoT World Today. Available at: **https://www.iotworldtoday.com/2020/01/03/iot-developers-may-need-time-series-data-analysis-skills/** [Accessed 9 Mar. 2020].

Wang, X., Smith, K. and Hyndman, R. (2006). Characteristic-Based Clustering for Time Series Data. *Data Mining and Knowledge Discovery,* 13(3), pp.335–364.

Werbos, P.J. (1990). Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10), pp.1550–1560.

Wu, J. (2019). *Python's Advantages and Disadvantages Summarized*. [online] Medium. Available at: **https://medium.com/better-programming/pythons-advantages-and-disadvantages-summarized-212b5fdf8883** [Accessed 18 Feb. 2020].

Yang, Q. and Wu, X. (2006). 10 Challenging Problems in Data Mining Research. *International Journal of Information Technology & Decision Making*, 05(04), pp.597–604.

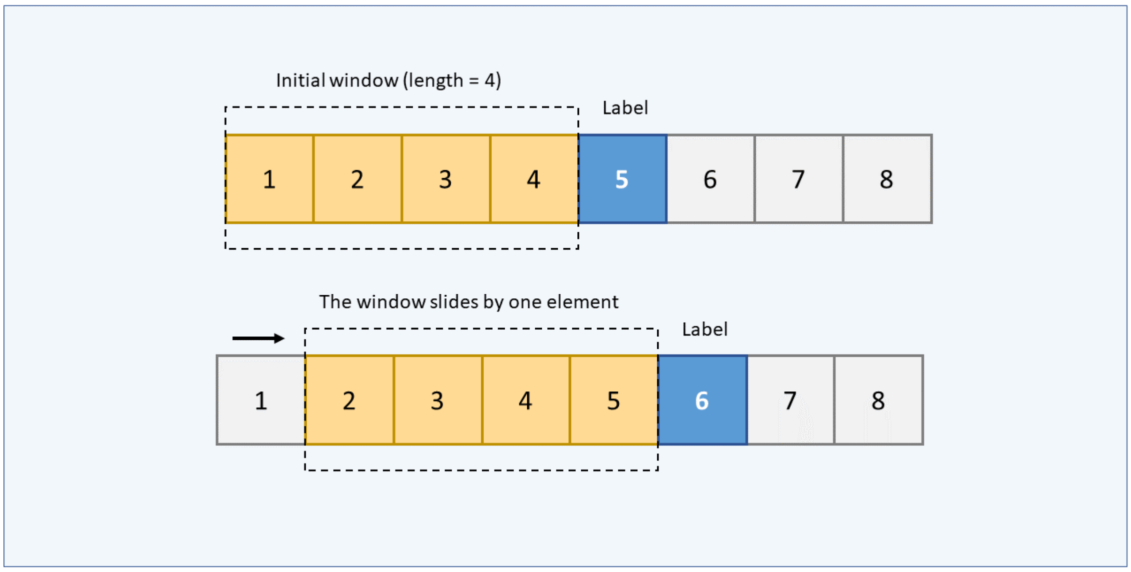
Yau, J. (2018). *Time Series Forecasting Using Recurrent Neural Network and Vector Autoregressive Model: When and How*. [Video] Available at: **https://youtu.be/i40Road82No** [Accessed 13 May 2020].

Zhang, G.P. (2012). Neural Networks for Time-Series Forecasting. In: G. Rozenberg, T. Bäck and J.N. Kok, eds., *Handbook of Natural Computing*. Springer, pp.461–477.

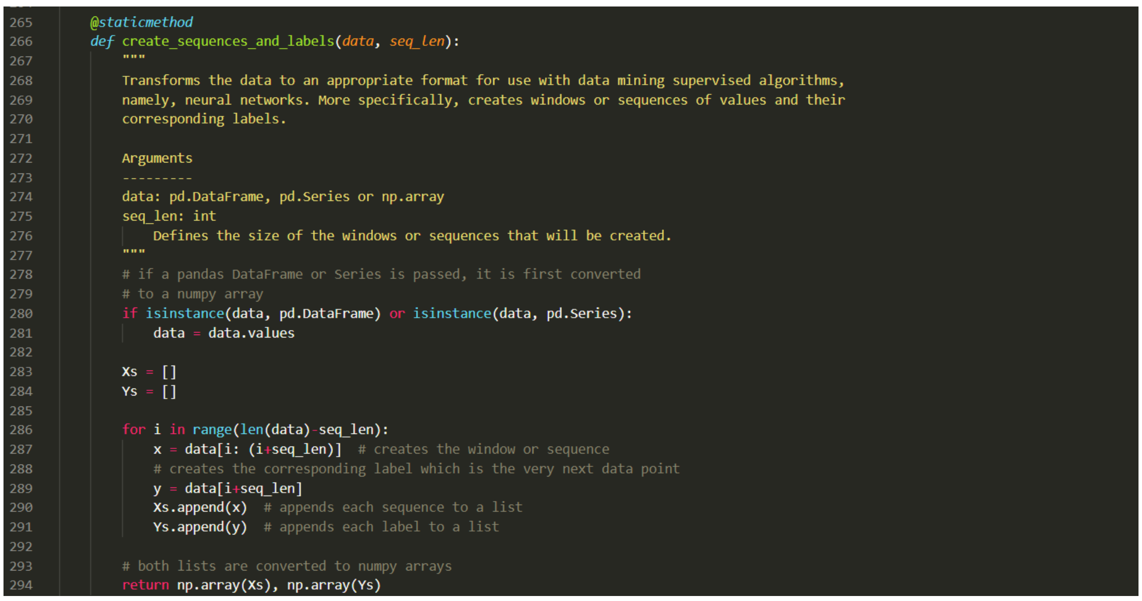
Zhang, P. (2004). Business Forecasting with Artificial Neural Networks: An Overview. In: *Business Forecasting with Artificial Neural Networks*. Idea Group Publishing, pp.1–14.

Zhu, L. and Laptev, N. (2017). Deep and Confident Prediction for Time Series at Uber. *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*. [online] Available at: **https://arxiv.org/abs/1709.01907** [Accessed 18 Jul. 2020].

APPENDICES

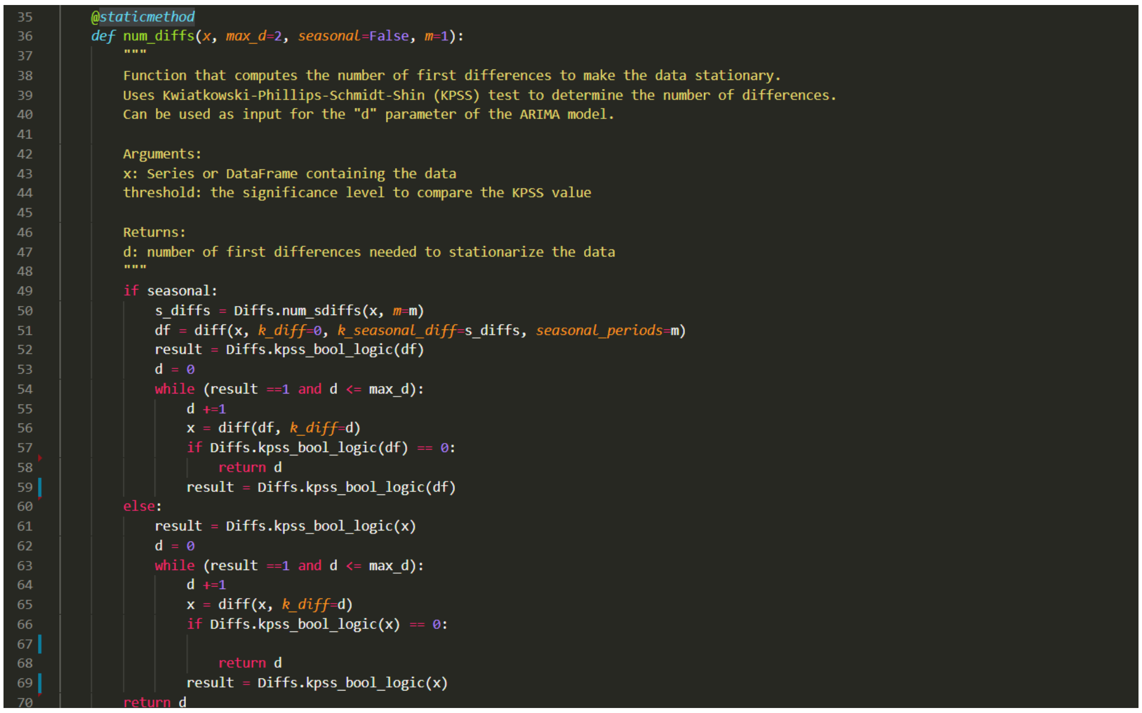


Appendix 1. Sliding window technique



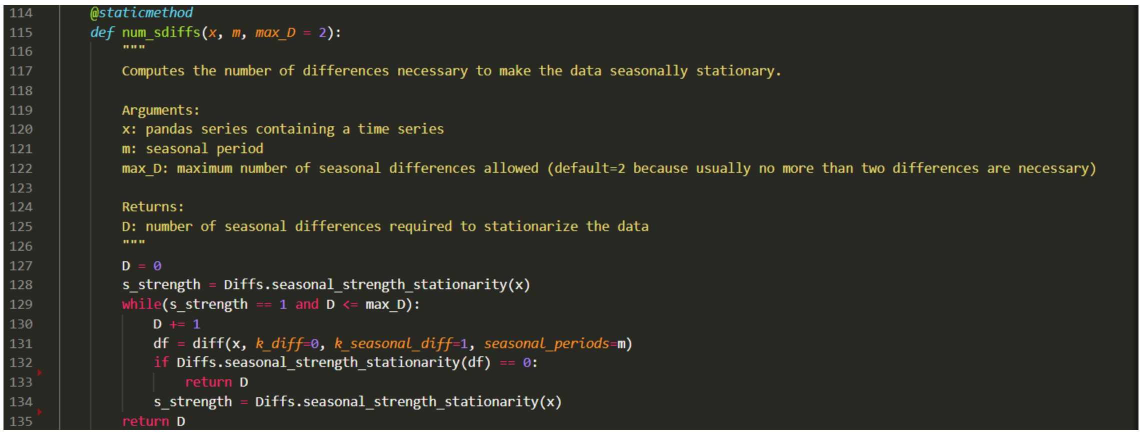
```
    @staticmethod
    def create_sequences_and_labels(data, seq_len):
        """
        Transforms the data to an appropriate format for use with data mining supervised algorithms,
        namely, neural networks. More specifically, creates windows or sequences of values and their
        corresponding labels.

        Arguments
        ---------
        data: pd.DataFrame, pd.Series or np.array
        seq_len: int
            Defines the size of the windows or sequences that will be created.
        """
        # if a pandas DataFrame or Series is passed, it is first converted
        # to a numpy array
        if isinstance(data, pd.DataFrame) or isinstance(data, pd.Series):
            data = data.values

        Xs = []
        Ys = []

        for i in range(len(data)-seq_len):
            x = data[i: (i+seq_len)]  # creates the window or sequence
            # creates the corresponding label which is the very next data point
            y = data[i+seq_len]
            Xs.append(x)  # appends each sequence to a list
            Ys.append(y)  # appends each label to a list

        # both lists are converted to numpy arrays
        return np.array(Xs), np.array(Ys)
```

Appendix 2. Sliding window function



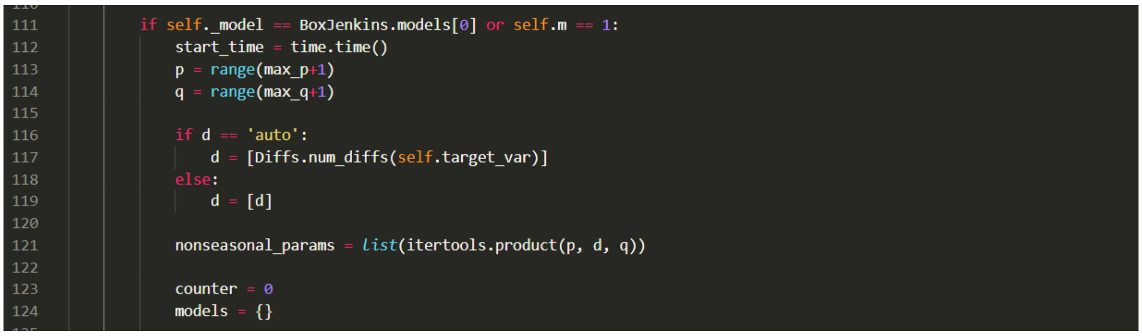
```
    @staticmethod
    def num_diffs(x, max_d=2, seasonal=False, m=1):
        """
        Function that computes the number of first differences to make the data stationary.
        Uses Kwiatkowski-Phillips-Schmidt-Shin (KPSS) test to determine the number of differences.
        Can be used as input for the "d" parameter of the ARIMA model.

        Arguments:
        x: Series or DataFrame containing the data
        threshold: the significance level to compare the KPSS value

        Returns:
        d: number of first differences needed to stationarize the data
        """
        if seasonal:
            s_diffs = Diffs.num_sdiffs(x, m=m)
            df = diff(x, k_diff=0, k_seasonal_diff=s_diffs, seasonal_periods=m)
            result = Diffs.kpss_bool_logic(df)
            d = 0
            while (result ==1 and d <= max_d):
                d +=1
                x = diff(df, k_diff=d)
                if Diffs.kpss_bool_logic(df) == 0:
                    return d
                result = Diffs.kpss_bool_logic(df)
        else:
            result = Diffs.kpss_bool_logic(x)
            d = 0
            while (result ==1 and d <= max_d):
                d +=1
                x = diff(x, k_diff=d)
                if Diffs.kpss_bool_logic(x) == 0:

                    return d
                result = Diffs.kpss_bool_logic(x)
        return d
```

Appendix 3. Computation of the required non-seasonal differences



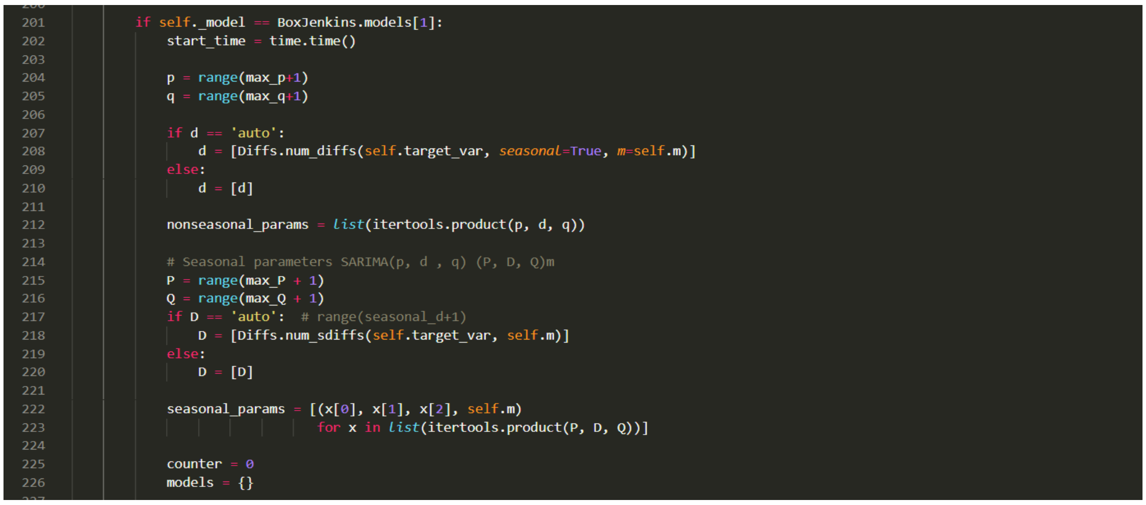
```
    @staticmethod
    def num_sdiffs(x, m, max_D = 2):
        """
        Computes the number of differences necessary to make the data seasonally stationary.

        Arguments:
        x: pandas series containing a time series
        m: seasonal period
        max_D: maximum number of seasonal differences allowed (default=2 because usually no more than two differences are necessary)

        Returns:
        D: number of seasonal differences required to stationarize the data
        """
        D = 0
        s_strength = Diffs.seasonal_strength_stationarity(x)
        while(s_strength == 1 and D <= max_D):
            D += 1
            df = diff(x, k_diff=0, k_seasonal_diff=1, seasonal_periods=m)
            if Diffs.seasonal_strength_stationarity(df) == 0:
                return D
            s_strength = Diffs.seasonal_strength_stationarity(x)
        return D
```

Appendix 4. Computation of the required seasonal differences



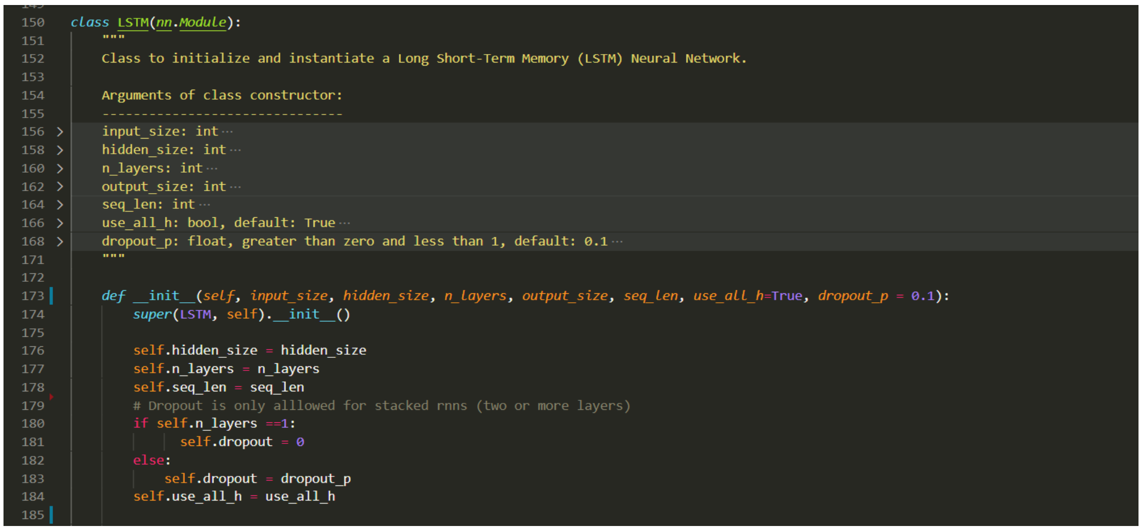
```
        if self._model == BoxJenkins.models[0] or self.m == 1:
            start_time = time.time()
            p = range(max_p+1)
            q = range(max_q+1)

            if d == 'auto':
                d = [Diffs.num_diffs(self.target_var)]
            else:
                d = [d]

            nonseasonal_params = list(itertools.product(p, d, q))

            counter = 0
            models = {}
```

Appendix 5. Code snippet from the ARIMA hyperparameter search



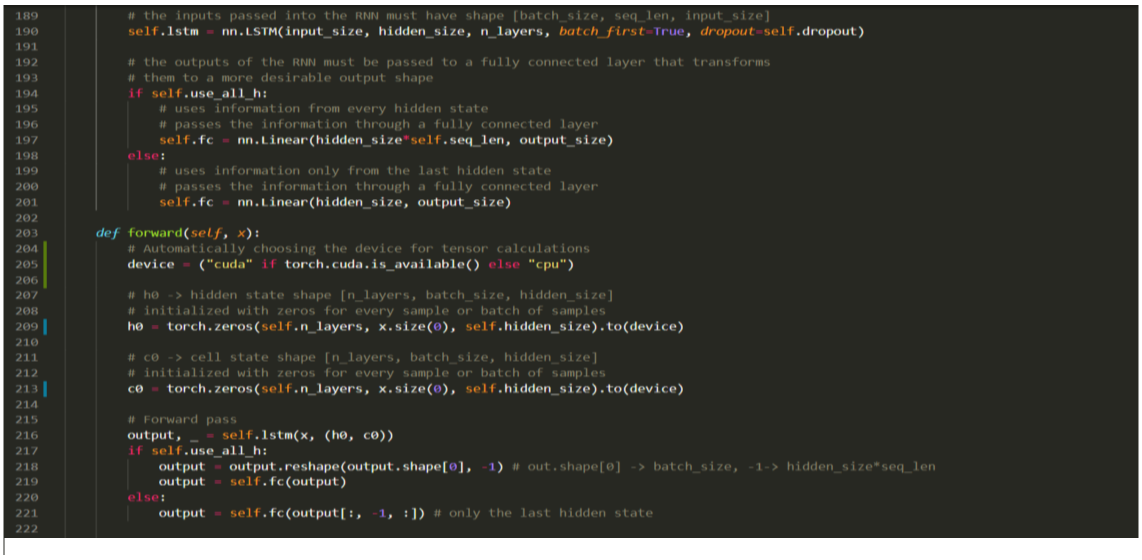
```python
        if self._model == BoxJenkins.models[1]:
            start_time = time.time()

            p = range(max_p+1)
            q = range(max_q+1)

            if d == 'auto':
                d = [Diffs.num_diffs(self.target_var, seasonal=True, m=self.m)]
            else:
                d = [d]

            nonseasonal_params = list(itertools.product(p, d, q))

            # Seasonal parameters SARIMA(p, d , q) (P, D, Q)m
            P = range(max_P + 1)
            Q = range(max_Q + 1)
            if D == 'auto':  # range(seasonal_d+1)
                D = [Diffs.num_sdiffs(self.target_var, self.m)]
            else:
                D = [D]

            seasonal_params = [(x[0], x[1], x[2], self.m)
                               for x in list(itertools.product(P, D, Q))]

            counter = 0
            models = {}
```

Appendix 6. Code snippet from the SARIMA hyperparameter search



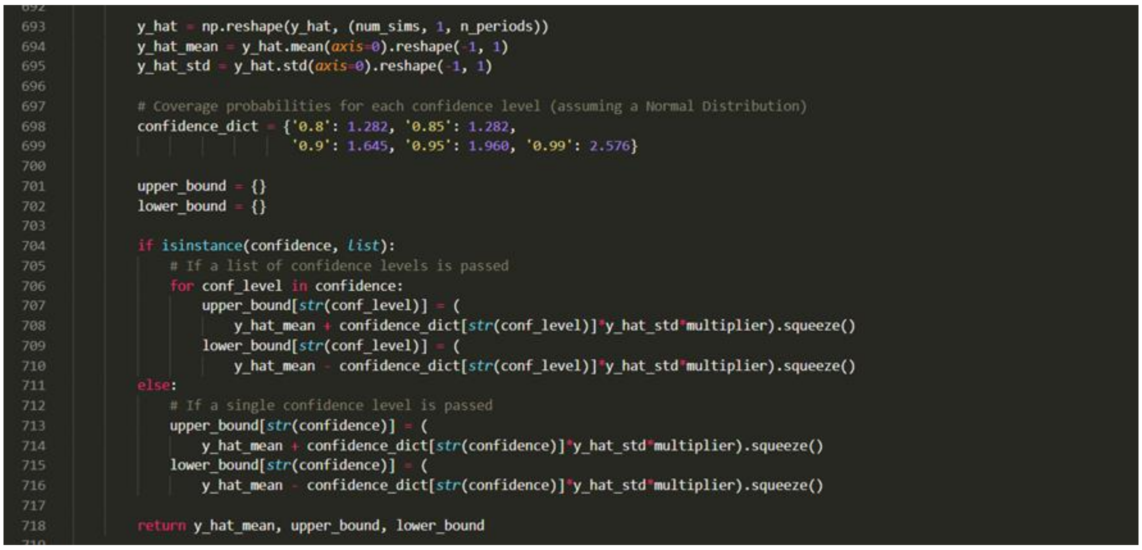
```python
class LSTM(nn.Module):
    """
    Class to initialize and instantiate a Long Short-Term Memory (LSTM) Neural Network.

    Arguments of class constructor:
    -------------------------------
    input_size: int···
    hidden_size: int···
    n_layers: int···
    output_size: int···
    seq_len: int···
    use_all_h: bool, default: True···
    dropout_p: float, greater than zero and less than 1, default: 0.1···
    """

    def __init__(self, input_size, hidden_size, n_layers, output_size, seq_len, use_all_h=True, dropout_p = 0.1):
        super(LSTM, self).__init__()

        self.hidden_size = hidden_size
        self.n_layers = n_layers
        self.seq_len = seq_len
        # Dropout is only allowed for stacked rnns (two or more layers)
        if self.n_layers ==1:
            self.dropout = 0
        else:
            self.dropout = dropout_p
        self.use_all_h = use_all_h

```

Appendix 7. LSTM implementation (part 1)



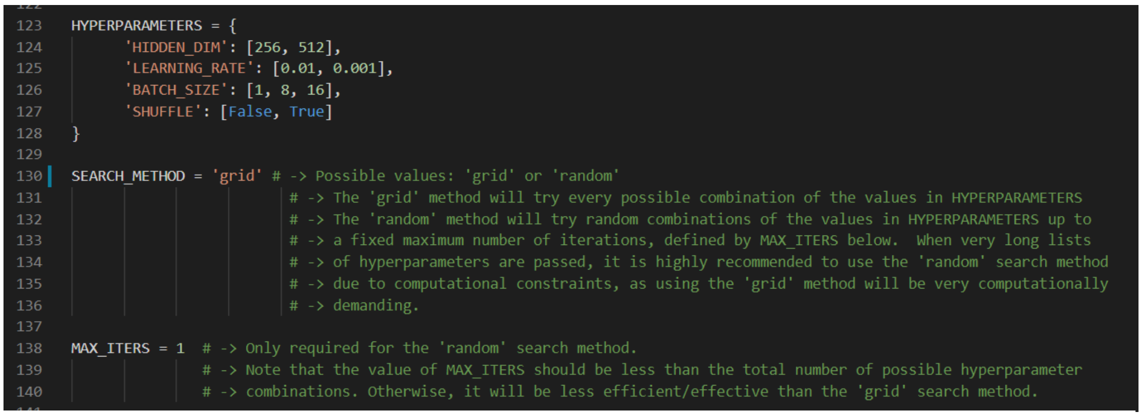
```python
        # the inputs passed into the RNN must have shape [batch_size, seq_len, input_size]
        self.lstm = nn.LSTM(input_size, hidden_size, n_layers, batch_first=True, dropout=self.dropout)

        # the outputs of the RNN must be passed to a fully connected layer that transforms
        # them to a more desirable output shape
        if self.use_all_h:
            # uses information from every hidden state
            # passes the information through a fully connected layer
            self.fc = nn.Linear(hidden_size*self.seq_len, output_size)
        else:
            # uses information only from the last hidden state
            # passes the information through a fully connected layer
            self.fc = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        # Automatically choosing the device for tensor calculations
        device = ("cuda" if torch.cuda.is_available() else "cpu")

        # h0 -> hidden state shape [n_layers, batch_size, hidden_size]
        # initialized with zeros for every sample or batch of samples
        h0 = torch.zeros(self.n_layers, x.size(0), self.hidden_size).to(device)

        # c0 -> cell state shape [n_layers, batch_size, hidden_size]
        # initialized with zeros for every sample or batch of samples
        c0 = torch.zeros(self.n_layers, x.size(0), self.hidden_size).to(device)

        # Forward pass
        output, _ = self.lstm(x, (h0, c0))
        if self.use_all_h:
            output = output.reshape(output.shape[0], -1) # out.shape[0] -> batch_size, -1-> hidden_size*seq_len
            output = self.fc(output)
        else:
            output = self.fc(output[:, -1, :]) # only the last hidden state

```

Appendix 8. LSTM implementation (part 2)



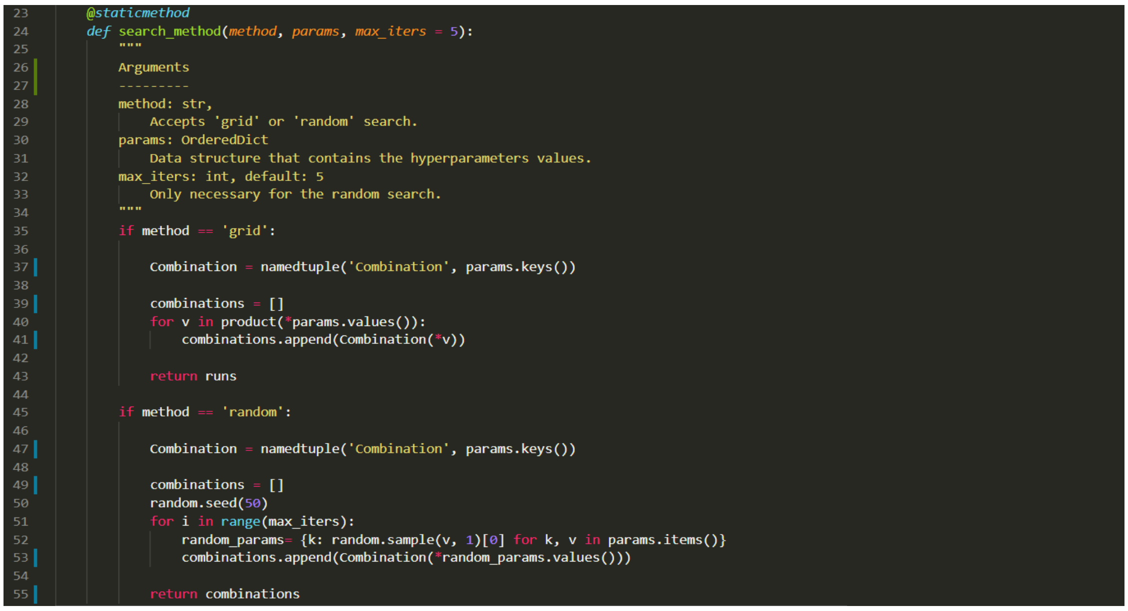
```python
        y_hat = np.reshape(y_hat, (num_sims, 1, n_periods))
        y_hat_mean = y_hat.mean(axis=0).reshape(-1, 1)
        y_hat_std = y_hat.std(axis=0).reshape(-1, 1)

        # Coverage probabilities for each confidence level (assuming a Normal Distribution)
        confidence_dict = {'0.8': 1.282, '0.85': 1.282,
                           '0.9': 1.645, '0.95': 1.960, '0.99': 2.576}

        upper_bound = {}
        lower_bound = {}

        if isinstance(confidence, list):
            # If a list of confidence levels is passed
            for conf_level in confidence:
                upper_bound[str(conf_level)] = (
                    y_hat_mean + confidence_dict[str(conf_level)]*y_hat_std*multiplier).squeeze()
                lower_bound[str(conf_level)] = (
                    y_hat_mean - confidence_dict[str(conf_level)]*y_hat_std*multiplier).squeeze()
        else:
            # If a single confidence level is passed
            upper_bound[str(confidence)] = (
                y_hat_mean + confidence_dict[str(confidence)]*y_hat_std*multiplier).squeeze()
            lower_bound[str(confidence)] = (
                y_hat_mean - confidence_dict[str(confidence)]*y_hat_std*multiplier).squeeze()

        return y_hat_mean, upper_bound, lower_bound
```

Appendix 9. Code snippet from the ANNs' forecast function



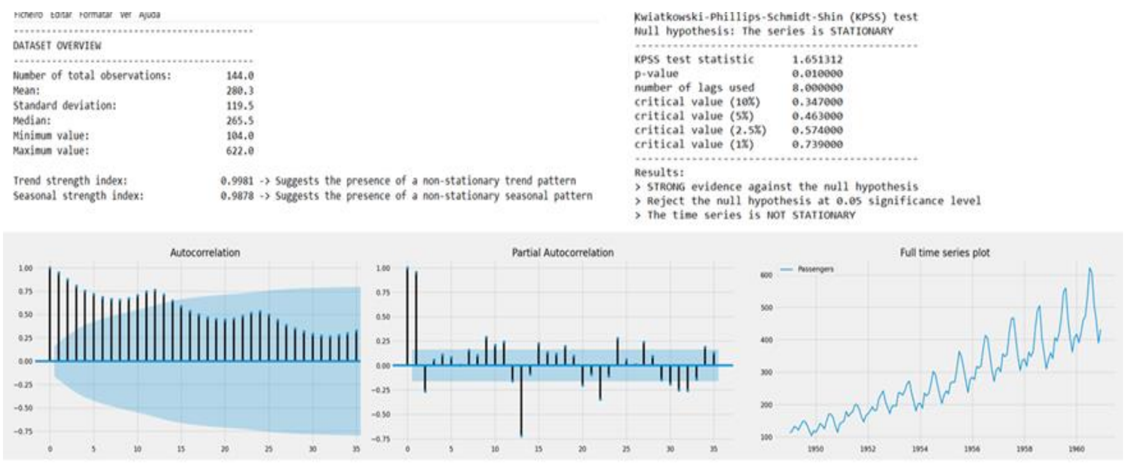
```python
HYPERPARAMETERS = {
        'HIDDEN_DIM': [256, 512],
        'LEARNING_RATE': [0.01, 0.001],
        'BATCH_SIZE': [1, 8, 16],
        'SHUFFLE': [False, True]
}

SEARCH_METHOD = 'grid' # -> Possible values: 'grid' or 'random'
                        # -> The 'grid' method will try every possible combination of the values in HYPERPARAMETERS
                        # -> The 'random' method will try random combinations of the values in HYPERPARAMETERS up to
                        # -> a fixed maximum number of iterations, defined by MAX_ITERS below.  When very long lists
                        # -> of hyperparameters are passed, it is highly recommended to use the 'random' search method
                        # -> due to computational constraints, as using the 'grid' method will be very computationally
                        # -> demanding.

MAX_ITERS = 1  # -> Only required for the 'random' search method.
               # -> Note that the value of MAX_ITERS should be less than the total number of possible hyperparameter
               # -> combinations. Otherwise, it will be less efficient/effective than the 'grid' search method.
```

Appendix 10. ANNs' searchable hyperparameter grid



```python
    @staticmethod
    def search_method(method, params, max_iters = 5):
        """
        Arguments
        ---------
        method: str,
            Accepts 'grid' or 'random' search.
        params: OrderedDict
            Data structure that contains the hyperparameters values.
        max_iters: int, default: 5
            Only necessary for the random search.
        """
        if method == 'grid':

            Combination = namedtuple('Combination', params.keys())

            combinations = []
            for v in product(*params.values()):
                combinations.append(Combination(*v))

            return runs

        if method == 'random':

            Combination = namedtuple('Combination', params.keys())

            combinations = []
            random.seed(50)
            for i in range(max_iters):
                random_params= {k: random.sample(v, 1)[0] for k, v in params.items()}
                combinations.append(Combination(*random_params.values()))

            return combinations
```

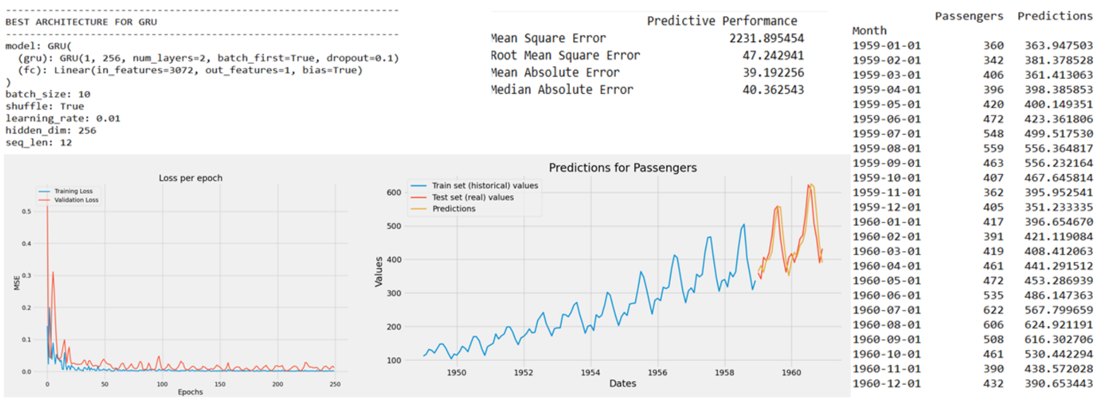
Appendix 11. Grid and Random search methods implementation



```
------------------------------------------
DATASET OVERVIEW
------------------------------------------
Number of total observations:          144.0
Mean:                                  280.3
Standard deviation:                    119.5
Median:                                265.5
Minimum value:                         104.0
Maximum value:                         622.0

Trend strength index:                 0.9981 -> Suggests the presence of a non-stationary trend pattern
Seasonal strength index:              0.9878 -> Suggests the presence of a non-stationary seasonal pattern
```

```
Kwiatkowski-Phillips-Schmidt-Shin (KPSS) test
Null hypothesis: The series is STATIONARY
------------------------------------------------
KPSS test statistic       1.651312
p-value                   0.010000
number of lags used       8.000000
critical value (10%)      0.347000
critical value (5%)       0.463000
critical value (2.5%)     0.574000
critical value (1%)       0.739000
------------------------------------------------
Results:
> STRONG evidence against the null hypothesis
> Reject the null hypothesis at 0.05 significance level
> The time series is NOT STATIONARY
```

Appendix 12. Data loading module output example

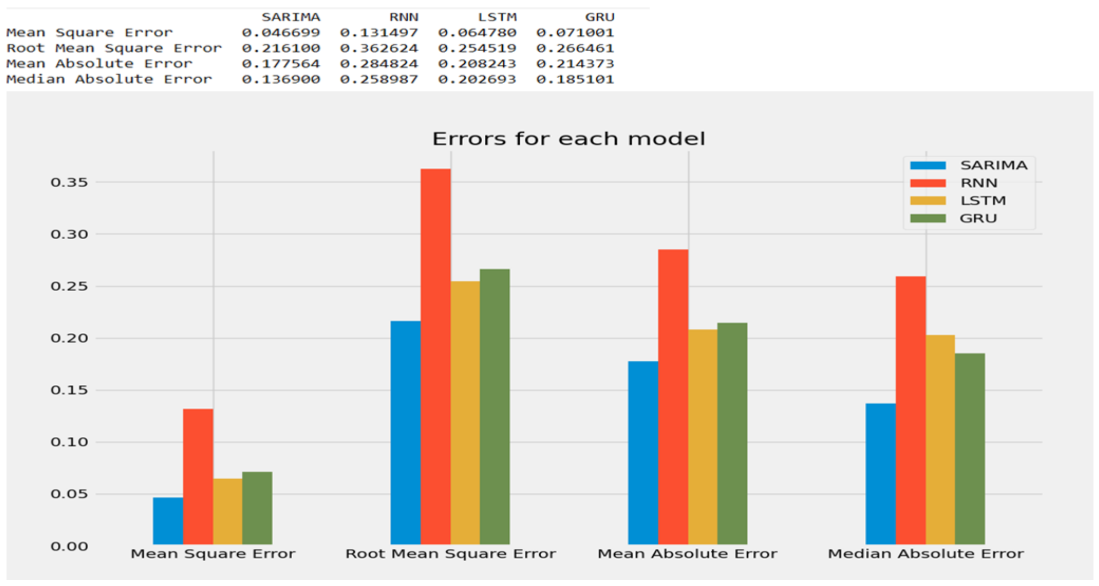


```
----------------------------------------------------------------
BEST ARCHITECTURE FOR GRU
----------------------------------------------------------------
model: GRU(
  (gru): GRU(1, 256, num_layers=2, batch_first=True, dropout=0.1)
  (fc): Linear(in_features=3072, out_features=1, bias=True)
)
batch_size: 10
shuffle: True
learning_rate: 0.01
hidden_dim: 256
seq_len: 12
```

```
                       Predictive Performance
Mean Square Error                 2231.895454
Root Mean Square Error              47.242941
Mean Absolute Error                 39.192256
Median Absolute Error               40.362543
```

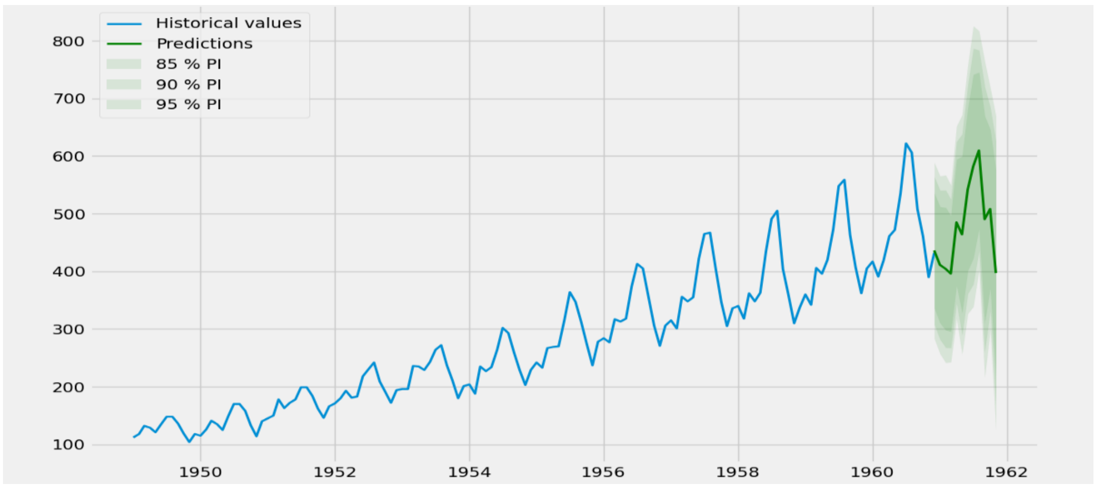
```
            Passengers  Predictions
Month
1959-01-01         360   363.947503
1959-02-01         342   381.378528
1959-03-01         406   361.413063
1959-04-01         396   398.385853
1959-05-01         420   400.149351
1959-06-01         472   423.361806
1959-07-01         548   499.517530
1959-08-01         559   556.364817
1959-09-01         463   556.232164
1959-10-01         407   467.645814
1959-11-01         362   395.952541
1959-12-01         405   351.233335
1960-01-01         417   396.654670
1960-02-01         391   421.119084
1960-03-01         419   408.412063
1960-04-01         461   441.291512
1960-05-01         472   453.286939
1960-06-01         535   486.147363
1960-07-01         622   567.799659
1960-08-01         606   624.921191
1960-09-01         508   616.302706
1960-10-01         461   530.442294
1960-11-01         390   438.572028
1960-12-01         432   390.653443
```

Appendix 13. Model fitting module output example

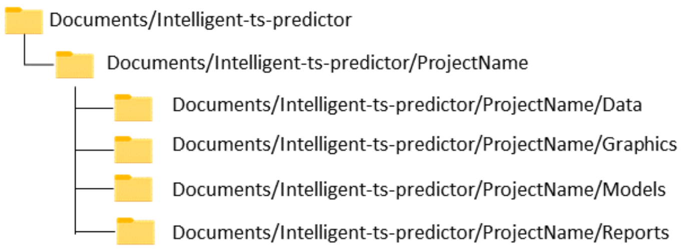


```
                          SARIMA       RNN      LSTM       GRU
Mean Square Error       0.046699  0.131497  0.064780  0.071001
Root Mean Square Error  0.216100  0.362624  0.254519  0.266461
Mean Absolute Error     0.177564  0.284824  0.208243  0.214373
Median Absolute Error   0.136900  0.258987  0.202693  0.185101
```

Appendix 14. Model comparison module output example



Appendix 15. Forecasting module output example


Documents/Intelligent-ts-predictor
Documents/Intelligent-ts-predictor/ProjectName
Documents/Intelligent-ts-predictor/ProjectName/Data
Documents/Intelligent-ts-predictor/ProjectName/Graphics
Documents/Intelligent-ts-predictor/ProjectName/Models
Documents/Intelligent-ts-predictor/ProjectName/Reports

Appendix 16. Directory structure created by the initialization script