

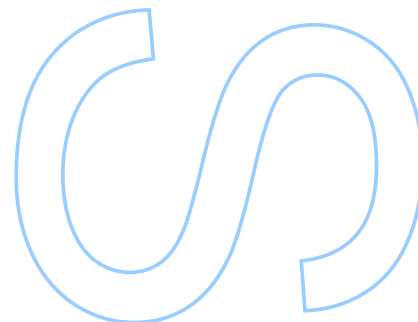
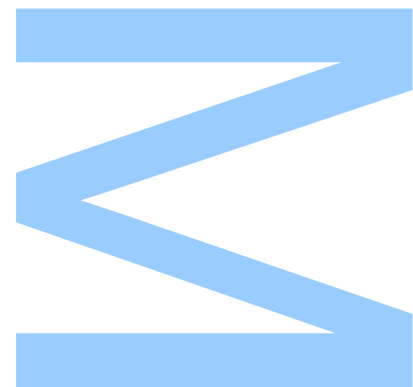
Machine Learning Methods for Time Series Forecasting- E-commerce Sales Study Case

Ana Rita Mendes Teixeira

Master in Computational Statistics and Data Analysis
2023

Supervisor

Joaquim Costa, Assistant Professor
Faculty of Sciences of Porto University



“ The goal of forecasting is not to predict the future but to tell you what you need to know to take meaningful action in the present. ”

Paul Saffo

Acknowledgements

16 Setembro de 2018. Neste dia pisei, pela primeira vez, a Faculdade de Ciências da Universidade do Porto. Era um livro em branco disposta a preencher cada página com conhecimento, memórias, experiências e desafios.

2205 dias separam esta data do dia em que entrego este projeto. As folhas, outrora brancas, estão preenchidas e nem 2205 delas seriam suficientes para agradecer a todos os que me fizeram chegar aqui. Assim, reconhecendo a insuficiência das palavras para expressar a minha gratidão, aqui deixo o meu agradecimento a todos os que comigo se cruzaram.

À minha família que, também eles absorvidos nesta viagem alucinante, sempre me incentivaram a fazer mais e melhor.

Aos meus amigos que sempre acreditaram em mim e me ajudaram em todos os momentos deste percurso. Pedro Quaresma, Beatriz Maurício, Maria Pedroso, este projeto também a vós pertence. Obrigada por serem os melhores parceiros nesta viagem.

À D.Mercedes que no seu jeito especial e ternurento me deu colo e me fez sentir sempre em casa.

Por último, a MateC e a esta família que mesmo "em dias cinzentos, mesmo estando num grande aperto" foram o meu refúgio e porto seguro.

A todos os que me ajudarem neste último capítulo do livro, neste projeto, a minha mais profunda gratidão.

UNIVERSITY OF PORTO

Resumo

Faculty of Sciences of University of Porto

Department of Mathematics

Mestrado em Estatística Computacional e Análise de Dados

Métodos de Aprendizagem de Máquina para previsão para séries temporais- caso de estudo de vendas em comércio eletrónico

por [Ana Rita TEIXEIRA](#)

Diariamente, somos confrontados com o fenómeno crescente da disponibilidade de grandes quantidades de dados históricos. Somos inundados por vastos repositórios de informações, coletadas e armazenadas de diversas fontes ao longo do tempo.

Mas o que torna este fenómeno verdadeiramente fascinante é o extraordinário potencial que possui – o poder de desvendar os segredos do futuro. Simultaneamente, há uma necessidade crescente de previsões precisas em vários campos científicos. É aqui que entra em jogo a convergência de dois elementos críticos.

A combinação destes dois componentes levou-nos a explorar técnicas inovadoras que podem preencher a lacuna entre as observações passadas e os resultados futuros.

Anteriormente, os métodos estatísticos, como os modelos ARIMA, tinham uma influência significativa no domínio da previsão. Esses métodos foram um potente auxílio, oferecendo perceções valiosas sobre dados de séries temporais há décadas. No entanto, o cenário está a mudar.

Nos últimos tempos, os modelos de aprendizagem de máquina ganharam atenção e estabeleceram-se como fortes concorrentes dos métodos estatísticos tradicionais na comunidade de previsão. A sua adaptabilidade, escalabilidade e capacidade de lidar com relações não lineares abriram novas possibilidades e horizontes no mundo da previsão.

Na previsão de dados temporais, um fator determinante para o sucesso do estudo é a escolha do algoritmo mais promissor para modelar e prever um determinado episódio.

Este projeto pretende apresentar alguns destes modelos de aprendizagem, avaliar as suas capacidades, examinar os seus pontos fortes e fracos e inferir qual é o mais adequado para um caso de estudo específico e muito relevante no nosso mundo empresarial.

Iremos mergulhar nas profundezas da previsão de vendas no comércio eletrónico, um domínio onde a previsão de movimentos de stock e flutuações de vendas pode significar a diferença entre o sucesso e o fracasso.

UNIVERSITY OF PORTO

Abstract

Faculty of Sciences of University of Porto

Department of Mathematics

MSc. Computational Statistics and Data Analysis

Machine Learning Methods for Time Series Forecasting- E-commerce Sales Study Case

by [Ana Rita TEIXEIRA](#)

On a daily basis, we are faced with the growing phenomenon of the availability of large amounts of historical data. We are inundated with vast repositories of information, collected and stored from various sources across time.

But what makes this phenomenon truly fascinating is the extraordinary potential it holds – the power to unlock the secrets of the future. Simultaneously, there is an increasing need for accurate predictions in several scientific fields. This is where the convergence of two critical elements comes into play.

These two components combined has driven us to explore innovative techniques that can bridge the gap between past observations and future outcomes.

Earlier, statistical methods, like ARIMA models, had a significant influence in the forecasting domain. These methods have served us well, offering valuable insights into time series data for decades. However, the landscape is shifting.

In recent times, machine learning models have gained attention and established themselves as strong competitors against traditional statistical methods in the forecasting community. Their adaptability, scalability, and ability to handle non-linear relationships have opened up new possibilities and horizons in the world of prediction.

In temporal data forecasting, a determining factor for the success of the study is the choice of the most promising algorithm to model and predict a given episode.

This project intends to present some of these machine learning models, evaluate their capabilities, examine their strengths and weaknesses and infer which one is most suitable for a specific study case very relevant in our business world. We will dive into the depths

of e-commerce sales forecasting, a domain where predicting stock movements and sales fluctuations can mean the difference between success and failure.

Contents

Acknowledgements	v
Abstract	vii
Resumo	ix
Contents	xi
List of Figures	xiii
1 Introduction to time series	1
1.1 Time Series Definition	1
1.2 Time Series Components	3
1.3 Time Series Stationarity: Statistical Tests	4
1.4 Time Series Normalization	7
1.5 Time Series division into train and test sets: Splitting Methods	9
1.6 Forecasting Process using Time Series	10
2 Time Series Forecasting: Statistical Methods vs ML Methods	13
2.1 From statistical methods to nowadays: historical context	13
2.2 Time Series Forecasting in Machine Learning	14
3 Modeling Time Series	15
3.1 Neural Networks	15
3.1.1 Recurrent Neural Networks	15
3.1.2 Long Short-Term Memory (LSTM)	18
3.1.3 Gated Recurrent Unit (GRU)	20
3.2 Gradient Boosting Models	22
3.2.1 XGBoost	22
3.2.2 LightGBM	24
3.3 Multi-Layer Perceptron (MLP)	25
3.4 Support Vector Machines (SVM)	27
3.5 Random Forests	29
4 Error Metrics	31

5	Real World Case Scenario	33
5.1	E-commerce Sales Forecasting: Context	33
5.2	Data preparation and processing for daily granularity	34
5.3	Stock Forecast using Machine Learning for daily granularity	44
5.4	Data preparation and processing for weekly granularity	63
5.5	Stock Forecast using Machine Learning for weekly granularity	68
5.6	Data preparation and processing for bi-weekly granularity	80
5.7	Stock Forecast using Machine Learning for bi-weekly granularity	81
5.8	Evaluating Models Accuracy	89
6	Conclusion and Final Thoughts	95
	Bibliography	99

List of Figures & Acronyms

1.1	Time Series Components. [8].	4
1.2	Cost of forecasting versus cost of inaccuracy. [24].	11
3.1	Typical RNN Block. [32].	16
3.2	Typical LSTM Block. [40].	18
3.3	Typical GRU Block. [44].	20
3.4	XGBoost Architecture. [49].	23
3.5	LGBM Architecture. [49].	24
3.6	Multilayer Perceptron Architecture. [52].	26
3.7	SVM Algorithm. [55].	28
5.1	Time Series Plot (Daily Granularity).	34
5.2	Year-wise boxplot & Month-wise boxplot	36
5.3	Year-wise boxplot & Month-wise boxplot without outliers	37
5.4	Time Series decomposition	38
5.5	Rolling statistics Plot	39
5.6	Series after First Differentiation	40
5.7	Series Normalized after First Differentiation	42
5.8	RNN applied to Time Series 1 st Difference with lookback=7	45
5.9	RNN applied to Time Series 1 st Difference with lookback=20	46
5.10	RNN applied to Time Series 1 st Difference with lookback=30	47
5.11	LSTM applied to Time Series 1 st Difference with lookback=7	48
5.12	LSTM applied to Time Series 1 st Difference with lookback=20	49
5.13	LSTM applied to Time Series 1 st Difference with lookback=30	50
5.14	GRU applied to Time Series 1 st Difference with lookback=7	51
5.15	GRU applied to Time Series 1 st Difference with lookback=20	52
5.16	GRU applied to Time Series 1 st Difference with lookback=30	52
5.17	XGBoost applied to Time Series 1 st Difference with lookback=7	54
5.18	XGBoost applied to Time Series 1 st Difference with lookback=20	54
5.19	XGBoost applied to Time Series 1 st Difference with lookback=30	55
5.20	LightGBM applied to Time Series 1 st Difference with lookback=7	56
5.21	LightGBM applied to Time Series 1 st Difference with lookback=20	57
5.22	LightGBM applied to Time Series 1 st Difference with lookback=30	57
5.23	MLP applied to Time Series 1 st Difference with lookback=7	58
5.24	MLP applied to Time Series 1 st Difference with lookback=20	59
5.25	MLP applied to Time Series 1 st Difference with lookback=30	59
5.26	SVM applied to Time Series 1 st Difference with lookback=7	60

5.27 SVM applied to Time Series 1 st Difference with lookback=20	61
5.28 SVM applied to Time Series 1 st Difference with lookback=20	61
5.29 Random Forest applied to Time Series 1 st Difference with lookback=7	62
5.30 Random Forest applied to Time Series 1 st Difference with lookback=10	62
5.31 Random Forest applied to Time Series 1 st Difference with lookback=30	63
5.32 Time Series Data- Aggregated by week	64
5.33 Time Series Decomposition- Aggregated by week	65
5.34 Logarithmic Time Series- Aggregated by week	67
5.35 Logarithmic Time Series - Aggregated by week	69
5.36 Time Series Decomposition- Aggregated by week	70
5.37 Logarithmic Time Series Decomposition- Aggregated by week	71
5.38 Time Series Decomposition- Aggregated by week	72
5.39 Logarithmic Time Series Decomposition- Aggregated by week	73
5.40 Time Series Decomposition- Aggregated by week	73
5.41 Logarithmic Time Series Decomposition- Aggregated by week	74
5.42 Time Series Decomposition- Aggregated by week	74
5.43 Time Series Decomposition- Aggregated by week	75
5.44 Time Series Decomposition- Aggregated by week	76
5.45 Time Series Decomposition- Aggregated by week	77
5.46 Time Series Decomposition- Aggregated by week	77
5.47 Time Series Decomposition- Aggregated by week	78
5.48 Time Series Decomposition- Aggregated by week	78
5.49 Time Series Decomposition- Aggregated by week	79
5.50 Time Series Decomposition- Aggregated by week	79
5.51 Rolling statistics Plot- Bi-weekly granularity	80
5.52 Recurrent Neural Network- Model Application	82
5.53 Long Short Term Memory- Model Application	83
5.54 Gated Recurrent Unit- Model Application	84
5.55 XGBoost- Model Application	85
5.56 LightGBM- Model Application	85
5.57 Multi Layer Perceptron- Model Application	86
5.58 Support Vector Machine- Model Application	87
5.59 Random Forests- Model Application	88

ARIMA: Autoregressive integrated moving average

ARMA: Autoregressive moving average

MLP: Multilayer Perceptron

SVM: Support Vector Machine

MAE: Mean Absolute Error

MSE: Mean Squared Error

MPE: Mean Percentage Error

RMSE: Root Mean Squared Error

MAPE: Mean Absolute Percentage Error

RNN: Recurrent Neural Network

LSTM: Long Short-Term Memory

ML: Machine Learning

GRU: Gated Recurrent Unit

XGBoost: Extreme Gradient Boosting

LGBM: Light Gradient Boosting Machine

CNN: Convolutional Neural Network

IQR: Interquartile Range

ADF: Augmented Dickey-Fuller

KPSS: Kwiatkowski-Phillips-Schmidt-Shin

TS: Time Series

ResNets: Residual Networks

RMSPROP: Root Mean Square Propagation

SGD: Stochastic Gradient Descent

Chapter 1

Introduction to time series

Time series forecasting is the area of data analysis and predictive modeling that based on patterns, trends, and dependencies from historical data collected over time can make informed predictions about future behaviors. But first it's important to understand what is time series data.

1.1 Time Series Definition

A time series is a sequence S of historical measurements y_t of an observable variable y at time intervals $t=0, \pm 1, \pm 2, \dots$ [1], that is, data where each observation is recorded at specific time intervals, forming a chronological sequence.

What distinguishes time series data from other types of data is its ability to reveal how variables change over time. In simpler terms, time plays a crucial role because it illustrates how the data evolves throughout various time points, including the final outcomes. This temporal aspect offers an extra layer of information and a structured sequence of dependencies within the data.

Time series analysis typically demands a substantial volume of data points to ensure its robustness and reliability. A large dataset ensures that your sample size is representative and that the analysis can effectively filter out noisy data. Furthermore, it guarantees that any identified trends or patterns are not mere outliers and can account for seasonal variations.

In general, a collection of random variables, y_t , indexed by t , is referred to as a stochastic process. In a stochastic process, the mean, autocovariance and autocorrelation can be defined respectively as:

$$\mu_t = E(y_t) \quad (1.1)$$

$$\gamma(s, t) = cov(y_s, y_t) = E[(y_s - \mu_s)(y_t - \mu_t)] \quad (1.2)$$

$$\rho(s, t) = \frac{\gamma(s, t)}{\sqrt{\gamma(s, s)\gamma(t, t)}} \quad (1.3)$$

Time Series data can even be distinguished in two types: time series with a one time-dependent variable and a single independent variable (univariate time series) and time series with more than one time series variable. Each variable depends not only on its past values but also has some dependency on other variables (multivariate time series) [2]. Moreover, time series are said in continuous time, if observations are measured at every instance of time, or discrete time if observations are measured at discrete points of time.

In general, time series analysis involves the following interests:

- understand the underlying process that generates the series, obtaining information about the series behaviour and looking for relevant patterns present in the data;
- forecast future observations of short, medium or long term to support business decisions.

In the realm of data analysis, it's essential to distinguish between three fundamental types of data: time series data, cross-sectional data, and pooled data. As mentioned before, time series data are observations recorded at regular intervals over time but there are different types of data that describe how and when that time data was recorded.

Cross-sectional data capture multiple variables simultaneously but at a single point in time [3]. They offer a snapshot view, enabling the analysis of different variables concurrently, without the temporal dimension. Pooled data, on the other hand, is a combination of both time series data and cross-sectional data. They encompass variables recorded at various time points while also considering different characteristics simultaneously. [4]

These distinctions are crucial as they dictate the choice of analytical methods and the depth of insights that can be derived from diverse datasets.

1.2 Time Series Components

Time series data can presents the following components:

- Trend, to describe increasing or decreasing behavior of the time series frequently presented in linear modes;
- Seasonality, to highlight the repeating pattern of cycles of behavior over time;
- Cyclicity, to identify the repetitive changes in the time series and define their placement in the cycle;
- Irregularity/Noise, to regard the non-systematic aspect of time series deviating from the common model values.

Delving deeper into the topic, a **trend** within time series data denotes a prolonged and consistent shift either upwards or downwards in the data points. The trend essentially encapsulates the fundamental framework of the data, capturing both the direction and magnitude of change over a longer period [5]. A prevalent practice while studying time series is to model and eliminate the trend component from the data to gain clearer insights into underlying patterns and enhance the accuracy of future predictions.

Seasonality within time series data refers to recurring patterns or fluctuations that follow a regular, predictable cycle. These patterns typically repeat at consistent intervals, and often arise due to regular events, such as holidays, weekends, or the changing of seasons. In the context of time series analysis, it is customary to identify and account for seasonality to better comprehend the underlying dynamics and enhance the accuracy of forecasting [6].

Cyclicity in time series data characterizes the recurring patterns or periodic oscillations exhibited within a specific time frame. These patterns may arise from diverse influences, including seasonality (daily, weekly, monthly, yearly), underlying trends, and other inherent structures [7]. Unlike fixed intervals, cyclicity manifests at various frequencies, presenting a challenge in both identification and modeling. The complex nature of cyclicity underscores the dynamic and multi-faceted aspects of temporal data analysis.

Last but not least, **irregularity**, often referred to as 'noise,' within time series data concerns to the random and unpredictable fluctuations or deviations present in the data.

Unlike trends and seasonality, irregularity lacks a discernible pattern or regularity. Instead, it represents the unexplained variance and unexpected disturbances that can affect data points sporadically. Irregularity introduces unpredictability and uncertainty into the data, making it challenging to model accurately.

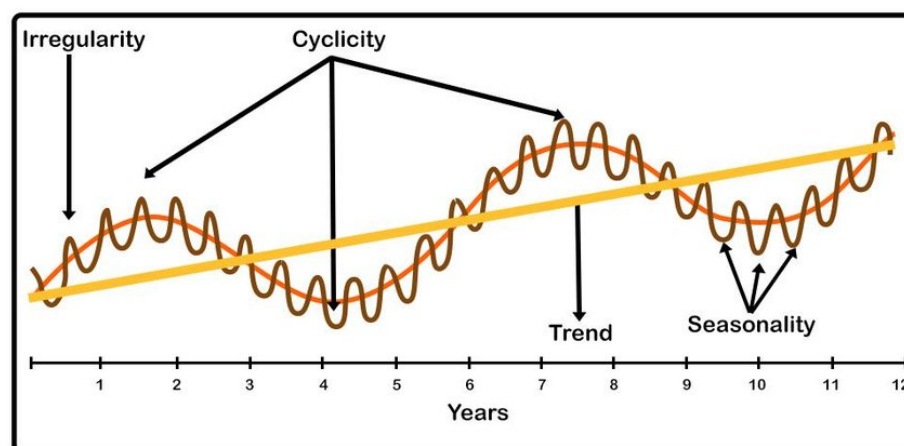


FIGURE 1.1: Time Series Components. [8].

1.3 Time Series Stationarity: Statistical Tests

A stationary series is one whose statistical properties such as mean, variance, covariance, and standard deviation do not vary with time, or these stats properties are not a function of time.[9] In other words, stationarity in time series also means series without a trend or seasonal components. This property allows models to predict effectively and precisely.

One way to visually assess stationarity is by calculating rolling statistics over the data. It can be computed moving averages and moving standard deviations with a rolling window and they can be plotted against the time index. If these rolling statistics remain relatively constant over time, it suggests stationarity. If they exhibit trends or cycles, it may indicate non-stationarity [10].

If the moving average remains relatively flat across time, it suggests constant mean (a stationary property). Conversely, if the moving average exhibits a trend, the series may not be stationary. The most basic methods for stationarity detection, like this one with rolling statistics, rely on plotting the data and visually checking for trend and seasonal components but this can be a dubious task. Combine visual inspection with formal tests is a good practice in this examination allowing the analyst to make a more informed decision about the stationarity of the series.

When it comes to statistical tests, Augmented Dickey-Fuller (ADF) and Kwiatkowski-Phillips-Schmidt-Shin (KPSS) are commonly used.

To understand how the ADF test works, it's essential to comprehend the Dickey-Fuller test first. The Dickey-Fuller test employs an autoregressive model, optimizing an information criterion across various lag values.

$$\Delta y_t = (\rho - 1)y_{t-1} + \mu + \varepsilon_t \quad (1.4)$$

$$H_0 : \rho = 1; \quad H_1 : \rho \neq 1$$

In this equation, Δy_t represents the difference between consecutive time series values, y_{t-1} is the value of the time series in the previous period, ρ is the parameter being tested for a unit root (whether it equals 1 or not), μ is a constant, and ε_t is the error or residual in period t [11]. If $\rho = 1$ (unit root), the series is non-stationary. A unit root in probability theory and statistics can complicate statistical inference involving time series models.

The Augmented Dickey-Fuller (ADF) test is a statistical tool designed to assess the presence of a unit root in a time series, a crucial factor in determining the stationarity of the data. In simple terms, a unit root suggests non-stationarity, indicating that the time series possesses a stochastic trend.

The ADF test extends the Dickey-Fuller test equation, incorporating a high-order autoregressive process to capture more complex dynamics within the time series:

$$\Delta y_t = \mu + (\rho - 1)y_{t-1} + \gamma t + \delta_1 \Delta y_{t-1} + \delta_2 \Delta y_{t-2} + \dots + \delta_p \Delta y_{t-p} + \varepsilon_t \quad (1.5)$$

$$H_0 : \rho = 1; \text{ the time series has a unit root and is non-stationary}$$

$$H_1 : \rho < 1; \text{ the time series does not have a unit root and is stationary [12]}$$

Here, Δy_t represents the difference between consecutive time series values, μ is a parameter, ρ is the coefficient multiplying y_{t-1} , γ is the coefficient multiplying t (time trend), and $\delta_1, \delta_2, \dots, \delta_p$ are coefficients of lagged differences up to order p . ε_t is the error or residual in period t . If $\rho = 1$, the null hypothesis (H_0) asserts the presence of a unit root, indicating a non-stationary time series. Conversely, the alternative hypothesis (H_1) states $\rho < 1$, signifying that the time series lacks a unit root and is therefore stationary.

However, the interpretation becomes particularly intriguing when $\rho > 1$. In such cases, the time series becomes explosive and highly non-stationary. This means that the

values in the series grow without bound over time, leading to extreme and unpredictable fluctuations. In essence, $\rho > 1$ signifies a divergence from stability, making the time series unsuitable for traditional statistical modeling due to its explosive nature.

Compared to the Dickey-Fuller test, the ADF test introduces additional differencing terms while maintaining a consistent overall equation structure. This increased complexity enhances the test's ability to capture various underlying patterns in the time series.

The Kwiatkowski-Phillips-Schmidt-Shin (KPSS) test is a statistical test used to assess the stationarity of a time series data set. The KPSS test is particularly useful because it complements other stationarity tests, such as ADF test, by focusing on a different aspect of stationarity.

H_0 : The data is stationary around a deterministic trend

H_1 : The data is not stationary around a deterministic trend

The null hypothesis assumes that the data has a trend component, but this trend is stationary and not growing indefinitely over time. If the test rejects the null hypothesis, it suggests that the data does not exhibit stationarity around a trend, indicating the presence of a stochastic trend or a unit root. A key difference from the ADF test is the null hypothesis of the KPSS test since is that the series is stationary. So practically, the interpretation of p-value is just the opposite of each other. That is, if the p-value is less than significance level, then the series is non-stationary [14]. Whereas in the ADF test, it would mean the tested series is stationary.

When applying both tests there are possible outcomes:

- Case 1: Both tests conclude that the given series is stationary: the series is stationary.
- Case 2: Both tests conclude that the given series is non-stationary: the series is non-stationary.
- Case 3: ADF concludes non-stationary, and KPSS concludes stationary: the series is trend stationary.
- Case 4: ADF concludes stationary, and KPSS concludes non-stationary: the series is difference stationary.

In case 3 to make the series strictly stationary, the trend needs to be removed and then the detrended series is checked for stationarity. This can be done by subtracting or dividing the series by the estimated trend component.

In a similar way, in case 4 differencing is to be used to make series stationary. Differentiation in time series involves calculating the differences between consecutive values. The first difference is simply the subtraction of each value in the series by the previous value. The second difference is the difference between successive differences. The differenced series then should be checked for stationarity [15].

1.4 Time Series Normalization

Normalization techniques are essential when working with time series data, especially when using forecasting models. Normalization helps transform time series data into a consistent and standardized format, which is important for several reasons.

Many machine learning and statistical models perform better when input features have similar magnitudes. Normalization ensures that the model's weights and gradients are balanced during training and thereby, the normalized data can be easier to interpret, as the values are transformed into a standard range.

There are some common normalization techniques for time series data.

Min-Max Scaling: Min-max normalization is a simple technique that rescales the data values to a range between 0 and 1, using the minimum and maximum values of the original data. This technique preserves the relative order and distance of the data points, but it also reduces the variance and magnifies the effect of outliers, that is, it can distort the data if there are extreme values or different scales [16]. The formula for Min-Max scaling is as follows:

$$X_{\text{normalized}} = \frac{X - X_{\min}}{X_{\max} - X_{\min}} \quad (1.6)$$

Where X is the original data point, $X_{\min}$ is the minimum value in the dataset and $X_{\max}$ is the maximum value in the dataset.

Z-Score Standardization: Z-score standardization is a technique that normalizes the data values by subtracting the mean and dividing by the standard deviation of the original data. This technique transforms the data into a normal distribution with a mean (μ) of 0 and a standard deviation (σ) of 1, which makes it easier to compare and analyze different variables. However, z-score normalization also changes the original scale and range of the data, and it can be affected by outliers. Z-score normalization is useful when the data

has a Gaussian distribution or when the scale and range are not important, but it can be misleading if the data has a different distribution or if the outliers have meaning [17].

The formula for Z-score standardization is as follows:

$$X_{\text{standardized}} = \frac{X - \mu}{\sigma} \quad (1.7)$$

Where X is the original data point, μ is the mean of the dataset and σ is the standard deviation of the dataset.

Log Transformation: Log transformation normalization is a technique that applies a logarithmic function to the data values, which reduces the skewness and compresses the range of the data. This technique makes the data more symmetric which can improve the performance of some statistical and machine learning models. However, log transformation normalization also changes the original scale and distribution of the data, and it can create negative values or lose information for zero or small values. Log transformation normalization is useful when the data has a skewed or exponential distribution or when the outliers are not important, but it can be inappropriate if the data has a linear or uniform distribution or if the outliers have meaning [18].

$$X_{\log} = \log(X) \quad (1.8)$$

Decimal scaling normalization Decimal scaling normalization is a technique that shifts the decimal point of the data values to reduce their magnitude, using a factor of 10. This technique preserves the relative order and proportion of the data points, but it also changes the scale and range of the data. Decimal scaling normalization is useful when the data has a large range of values or when the magnitude is not important, but it can be problematic if the data has a small range of values or if the scale and range have meaning.

$$X_{\text{normalized}} = \frac{X}{S} \quad (1.9)$$

Where $X_{\text{normalized}}$ represents the scaled (normalized) value of the original data point X , and S is the scaling factor. A common choice is to set S as a power of 10, typically based on the maximum absolute value in the dataset. For example, if the maximum absolute value is 562, it can be setted $S = 1000$ to ensure that all normalized values will have three decimal places.

1.5 Time Series division into train and test sets: Splitting Methods

Dividing time series data into training and testing sets is an essential step in the application of forecasting methods. The goal is to create a reliable evaluation framework to assess the performance of our forecasting model. There are several methods to split time series data.

Fixed Split: This is the simplest approach, where we choose a fixed point in time to separate the data into a training set and a testing set. Usually it's used data from the first 80% of the time period as the training set and the remaining 20% as the testing set [19].

However, this approach may not work well if there are seasonality or trend changes in the data. It assumes a consistent data distribution over time, which can be problematic when dealing with time series data that exhibit fluctuations or shifts due to factors like seasonality, external events, or market dynamics. In such cases, the fixed split approach might lead to suboptimal model performance, as it may inadvertently expose the forecasting model to patterns it has not been trained to capture.

Time-Based Split: In this method, the split of the data is based on a specific time period. This approach is particularly useful when we are working with a long time series with clear chronological order.

Rolling Window Split: In a rolling window split, the idea is to create multiple training and testing sets by sliding a fixed-size window through the time series data. This approach helps capture changing patterns over time.

Expanding Window Split: Similar to the rolling window split, the expanding window split starts with a small training set and gradually increases its size [20].

Walk-Forward Split: The walk-forward split is similar to the rolling window split, but it advances one step at a time. In each step, the model is trained on the past data and tested it on the next single data point. This approach closely mimics a real-world forecasting scenario.

Cross-Validation: Time series cross-validation techniques combine multiple splits to create a more robust evaluation framework. These methods often involve creating multiple training and testing sets with overlapping windows.

Regardless of the chosen method, it's essential to ensure that the training and testing sets do not overlap in time to prevent data leakage and to accurately evaluate the model's out-of-sample performance.

1.6 Forecasting Process using Time Series

The forecasting process includes some steps that the researchers should follow in order to find the most suitable model to their specific study case.

The first step is to collect historical time series data from reliable sources, **gather data** ensuring that it is relevant and covers an appropriate time span for the forecasting task. The collected data is then explored to understand its characteristics. This **data exploration** includes identify missing values or outliers, and process the data to handle any anomalies that could affect the forecasting process.

As we saw earlier [1.1](#), time series data can often be decomposed into its components, such as trend, seasonality, and residual (random fluctuations). **Time Series Decomposition** by identifying and separating these components can aid in understanding the underlying patterns better [\[21\]](#). The fourth step consists in the **selection of the best model**. There is a wide range of models to consider, including traditional statistical methods like ARIMA, exponential smoothing models, and more recently, machine learning techniques such as neural networks and support vector machines, that will be covered in this project later.

The selected model is trained using historical data and validated using a portion of the data that the model has not seen before. Proper **model training and validation** helps to assess the model's accuracy and generalization performance [\[22\]](#).

Once the model is trained and validated, it's used to predict future values. The forecasting performance is evaluated using various metrics such as mean absolute error (MAE), mean squared error (MSE), or root mean squared error (RMSE) to **measure the accuracy of the predictions**.

Besides following all of these steps, it's important to note that time series forecasting is an iterative process once it involves continuous exploration, experimentation, validation,

refinement and adaptation to ensure accurate predictions in the ever-changing context of time-dependent data. Model performance should be continually monitored to achieve more accurate forecasts.

It should be pointed out that, although in time series forecasting, the primary objective is achieving accurate predictions, it's imperative to recognize that the pursuit of absolute accuracy can sometimes come at a cost. While sophisticated and complex forecasting models might yield impressive accuracy, they often require substantial computational resources and time to implement [23]. In contrast, simpler models might sacrifice a marginal degree of accuracy but offer cost-effective and practical solutions. Thus, in real-life, it's critical to strike the right balance between the potential gains in forecast accuracy and the practical constraints and costs of model implementation.

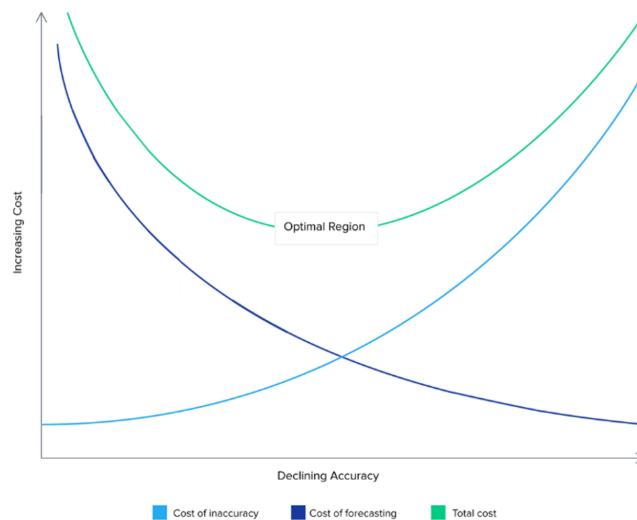


FIGURE 1.2: Cost of forecasting versus cost of inaccuracy. [24].

Chapter 2

Time Series Forecasting: Statistical Methods vs ML Methods

2.1 From statistical methods to nowadays: historical context

Time series forecasting, the art of predicting future values based on historical data collected over time, has started to be explored in the mid-20th century. In the 1950s George Box and Gwilym Jenkins introduced ARMA models which provided a framework for modeling autocorrelation and moving averages of the data [25]. These techniques have been extensively used to capture the underlying patterns, trends and seasonal components in time series data. Although they have proven to be effective for stationary time series data with predictable patterns, they present some limitations for complex and non-linear real world data. Moreover, they may struggle to handle large and high-dimensional datasets and they might not capture intricate relationships between variables in highly dynamic systems.

In the late 20th and early 21st centuries, as computational capabilities advanced and machine learning techniques emerged, researchers began exploring more powerful and flexible methods for forecasting leading to significant advancements in this domain.

Recurrent Neural Networks (RNNs), first introduced in the 1980s, became popular for handling sequential data by maintaining hidden states that preserved temporal information [26]. However, traditional RNNs suffered from the vanishing gradient problem, limiting their ability to capture long-term dependencies. This led to the development of the Long Short-Term Memory (LSTM) networks, introduced in the 1990s, by Sepp Hochreiter

and Jürgen Schmidhuber [27] and this method addressed the vanishing gradient problem in RNNs. These developments marked a major shift in the ability to model temporal dependencies in time series data.

These new machine learning methods, powered by sophisticated algorithms and the ability to automatically learn from data, offered a new paradigm for tackling forecasting challenges. These methods enabled the extraction of complex patterns and dependencies, making them well-suited for handling non-linear and high-dimensional data.

As the field of machine learning continues to advance, we can expect even more powerful and sophisticated methods to enhance our ability to predict and understand future behaviors in time series data.

2.2 Time Series Forecasting in Machine Learning

Machine learning is a necessary aspect of modern business and research for many organizations today. Machine learning algorithms automatically build a mathematical model using sample data – also known as “training data” – to make decisions.

Despite the differences between them and the traditional methods, the objective is the same: to improve the accuracy of forecasts while minimizing a loss function. The loss function is usually taken as the sum of squares due to errors in prediction/forecasting [28].

As seen before, traditional methods are limited to processing only previously collected, readily available demand history. In turn, ML autonomously defines points of interest in the unlimited flow of data to then align them with customer data insights at hand and conduct what-if analysis.

Furthermore, Machine Learning “... is about building programs with adaptable parameters that automatically adjust based on the data the programs receive. By adapting to previously seen data, the programs are able to improve their behavior. They also generalize data, meaning that the programs can perform functions on previously unseen datasets.” (Alejandro Correa Bahnsen) [29].

As in any case of working with machine learning, ML-forecasting can be supervised (requiring specific data input to work) or unsupervised (ongoing, self-learning data processing mechanisms).

Chapter 3

Modeling Time Series

Machine Learning is one of the most active research topics. Although this continuous evolution is an advantage enabling potentially better models to emerge, it also makes the search for the most suitable method more difficult since a relatively recent legacy method may bring comparatively insufficient results as opposed to more recently introduced alternatives

3.1 Neural Networks

Neural networks are mathematical models inspired by how biological nervous systems process information. They are designed to learn and extract patterns from data by simulating the interconnected network of neurons in the brain [30].

Traditional neural networks were not able to retain past or historical information since they look at each observation as independent from the others. This led to the rise of Recurrent Neural Networks (RNNs), which introduce the concept of memory to neural networks by including the dependency between data points.

3.1.1 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are designed to handle sequential data, making them suitable for tasks where the order of data matters. They are distinguished by their “memory” as they take information from prior inputs to influence the current input and output. While traditional deep neural networks assume that inputs and outputs are independent of each other, the output of recurrent neural networks depend on the prior elements within the sequence [31].

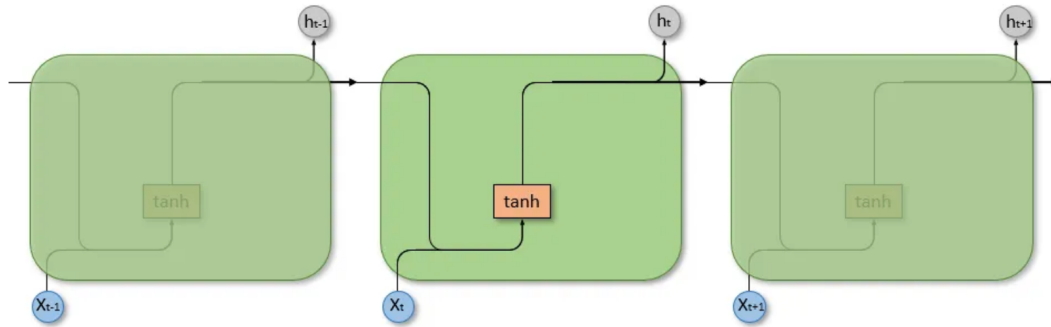


FIGURE 3.1: Typical RNN Block. [32].

$$h(t) = \tanh(W_x \cdot x(t) + W_h \cdot h(t-1) + b) \quad (3.1)$$

$$y(t) = W_y \cdot h(t) + b \quad (3.2)$$

As shown above in the figure we have a single block of RNN (the one in the middle). This block takes two inputs: $x(t)$, the input for the current time step and $h(t-1)$, the hidden state from the previous RNN block, which captures information from the previous step and is used to capture sequential dependencies in the data [32]. Each of these inputs is associated with its own set of weights: $x(t)$ weights W_x , and $h(t-1)$ weights W_h . There's also a bias term, b .

The RNN block will calculate 2 values. First of all, we will have $h(t)$ which is the result of applying the hyperbolic tangent ($\tanh$) activation function to a linear combination of the inputs and their corresponding weights, plus the bias term. This squashes the values between -1 and 1. Since it will be the input of the next block, this allows the model to build a hierarchy of features and capture more complex patterns in the data.

The second value is $y(t)$, as it is on the equation (3.2), which is the final output and represents the model's prediction at time step t [33].

In some cases, especially for simple regression tasks, we may choose to use the last hidden state $h(t)$ as the prediction, $y(t) = h(t)$, because it has already captured all the relevant information from the input sequence. It simplifies the model by directly using the hidden state as the output, and it can work well for some forecasting problems, especially when the patterns in the data are relatively straightforward. However, in more complex forecasting tasks or when the relationships between past and future values are non-linear or intricate, it might be beneficial to introduce an additional output layer with its own

weight matrix and bias term. This allows the model to learn more flexible and non-linear mappings from the hidden state to the predicted values, potentially improving prediction accuracy.

So, during training, the RNN adjusts the weights and biases to minimize the difference between its predictions and the actual target values in the training data [34]. That's why the weights are learnable parameters, there is, they are adjusted during the training process to optimize the model's ability to make accurate forecasts and minimize a loss function. Bias terms are also learnable parameters that help the model fit the data better.

When we stack these RNN blocks together, we create an RNN model.

Advantages vs Disadvantages:

Time series data often vary in length, making them a challenging input for many machine learning models. RNNs can handle sequences of different lengths effectively, which is a critical advantage when dealing with irregular or unevenly spaced time intervals in time series data.

However, one of the problems with RNN is the issue with vanishing gradients during backpropagation step of training [35].

When training an RNN, we update its internal parameters (weights and biases) to minimize a loss function. This process involves calculating gradients, which are derivatives that represent the rate of change of the loss with respect to these parameters.

Backpropagation is used to calculate gradients across the entire sequence of time steps. However, a problem arises when these gradients become extremely small as they are propagated backward in time. This is when we face the phenomenon of the "vanishing gradients."

Vanishing gradients occur because of the repeated multiplication of small values and it can be very affected by the activation functions like the hyperbolic tangent (tanh) since these function compress their input into a range that is either $[-1, 1]$, causing gradients to diminish significantly [36]. As a result, the network struggles to effectively update certain weights, particularly those associated with long-term dependencies, leading to slow or ineffective training.

The RNN, thus, tends to lose information over long sequences as it only stores the latest information. Hence, the RNN has only a short-term but not a long-term memory.

For solving the problem of vanishing gradient we may explore different versions: LSTM and GRU [37].

As a matter of interest, besides RNN, there are other types of neural network architectures designed to address other challenges and tasks. Artificial Neural Networks (ANNs) includes a first layer of nodes (input data), a second ('hidden') layer with nodes performing data aggregation followed by a nonlinear transformation, and a final 'output' layer where the predicted values are computed. The training of ANNs proceeds by iteratively adjusting the weights of links between the layers, however they might require more parameters and training time compared to RNNs [38]. Convolutional Neural Networks (CNNs) and Residual Networks (ResNets) are specialized neural networks designed for image recognition, classification and computer vision tasks. CNNs are optimized for grid-like data (e.g., images), and ResNets are designed for deep architecture optimization, which might not directly align with the sequential nature of time series data.

3.1.2 Long Short-Term Memory (LSTM)

While maintaining RNN's ability to learn the temporal dynamics of sequential data, LSTM can furthermore handle the vanishing gradient problem. As a result, complex multivariate data sequences can be accurately modeled, eliminating the necessity for predetermined time windows that are often necessary for tasks beyond the capabilities of feed-forward networks [39]. The downside of overly time-consuming supervised learning, however, remains.

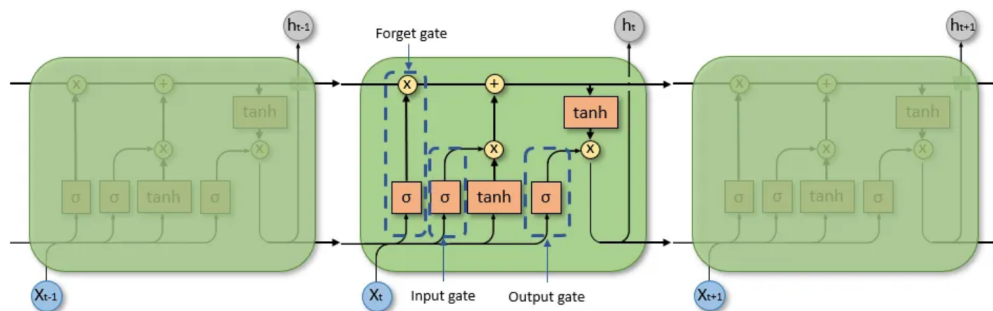


FIGURE 3.2: Typical LSTM Block. [40].

LSTMs use gate mechanisms to control the flow of information through the cell, allowing them to capture and remember important patterns over long sequences. The key gates in an LSTM are the forget gate, input gate, and output gate (marked in the central block of 3.2).

The cell state is passed from the input to the output of a cell and allows information to flow along the entire chain with only minor linear actions through the three gate that work as filters and control the flow of information that should be kept or disregarded [41].

The forget gate, (f_t) , decides how much of the previous cell state (c_{t-1}) to retain and how much to discard based on the current input (x_t) and the previous hidden state (h_{t-1}) . For this, a sigmoid function ^{*} is used and the output states how much information is kept (it ranges between 0 and 1 where 0, keep no information and 1, keep all information of the cell state).

$$f_t = \sigma(W_{f,x}x_t + W_{f,h}h_{t-1} + b_f) \quad (3.3)$$

The input gate, (i_t) , decides which information shall be added to the cell state and thus the long-term memory. Here, a sigmoid layer decides which values are updated. This gate allows the LSTM to update its memory with relevant new information [40].

$$i_t = \sigma(W_{i,x}x_t + W_{i,h}h_{t-1} + b_i) \quad (3.4)$$

The output gate, (o_t) , decides which parts of the cell state should contribute to the output (h_t)

$$o_t = \sigma(W_{o,x}x_t + W_{o,h}h_{t-1} + b_o) \quad (3.5)$$

The cell state, c_t , is updated through the forget gate and the input gate. This update mechanism allows the LSTM to selectively retain and store relevant information over time.

$$o_t = f_t \circ c_{t-1} + i_t \circ \tanh(h_{t-1}) \quad (3.6)$$

The hidden state of the current time step is then determined by the output gate and a $\tanh$ function [40]. This produces the current hidden state, which is used for making predictions and is also passed to the next time step as h_{t-1} .

$$h_t = o_t \circ \tanh(c_t) \quad (3.7)$$

^{*}SigmoidFunction : $\frac{1}{1+\exp^{-x}}$

In a forecast task, the output may be directly derived from h_t . The updated hidden state (h_t) and cell state (c_t) are passed to the next LSTM block or time step as the new h_{t-1} and c_{t-1} . This allows information to flow through the entire sequence.

Advantages vs Disadvantages:

LSTM's key advantage lies in its ability to effectively address the vanishing gradient problem. This is achieved through the introduction of specialized memory cells and gating mechanisms that enable LSTMs to capture and maintain patterns in both the long-term and short-term aspects of a sequence [42]. Crucially, since the gates within LSTMs are trainable, they facilitate the flow of gradients through the network without suffering significant attenuation.

The network, therefore, will be able to learn to remember or forget information based on the context and importance.

However, while LSTMs provide enhanced memory capabilities, due to their more complex structure, LSTMs are computationally more expensive, leading to longer training times and potentially over-fitting on smaller datasets.

3.1.3 Gated Recurrent Unit (GRU)

Similar to LSTMs, the GRU solves the vanishing gradient problem of RNNs. Besides this, GRUs use fewer gates and don't present a separate internal memory, i.e., a cell state. Hence, the GRU solely relies on the hidden state as a memory, leading to a simpler architecture [43].

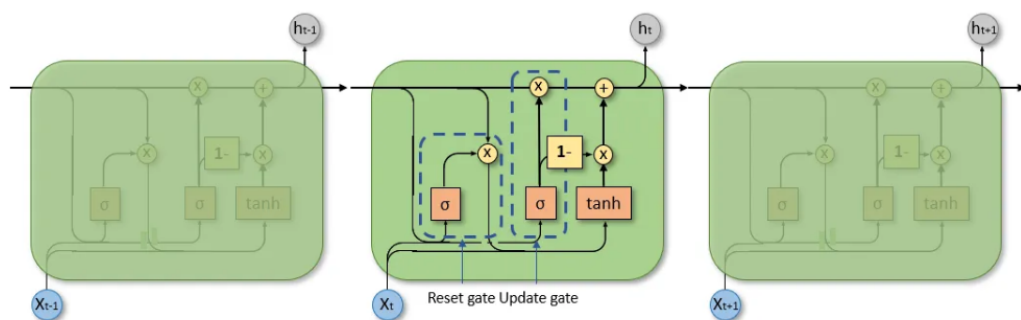


FIGURE 3.3: Typical GRU Block. [44].

The reset gate, r_t , is responsible for the short-term memory as it decides how much past information is kept and disregarded. This decision is based on two main inputs: the

current input x_t and the hidden state from the previous time step (h_{t-1}). These inputs are combined using weight matrices, $W_{r,x}$ and $W_{r,h}$, and a bias term b_r is added. The result is passed through a sigmoid activation function σ . When the reset gate outputs a value close to 0, it means that most of the past information should be ignored, otherwise, if the value is close to 1 than most of the past information should be preserved.

$$r_t = \sigma(W_{r,x}x_t + W_{r,h}h_{t-1} + b_r) \quad (3.8)$$

The update gate, u_t , in contrast, is responsible for the long-term memory (similar to the forget gate from LSTM). It decides how much of the current candidate hidden state should be used to update the current hidden state. The update gate also depends on the current input x_t and the previous hidden state h_{t-1} , which are weighted using matrices $W_{u,x}$ and $W_{u,h}$ and a bias term b_u [44]. The result is passed through a sigmoid activation and if the output is close to 0, it implies that the previous hidden state should be completely disregarded, and the new hidden state should be mostly based on the candidate hidden state.

$$u_t = \sigma(W_{u,x}x_t + W_{u,h}h_{t-1} + b_u) \quad (3.9)$$

The hidden state of the current time step is determined based on a two step process.

First, a candidate hidden state, $\hat{h}_t$, is determined. This one represents a temporary hidden state and combines the current input x_t , the hidden state of the previous time step h_{t-1} and an activation function. The influence of the previous hidden state on the candidate hidden state is controlled by the reset gate [45].

The reset gate (r_t) determines how much the previous hidden state (h_{t-1}) should contribute to the candidate hidden state. If r_t is close to 0 for a particular dimension, it means that information from that dimension of h_{t-1} will have little impact on $\hat{h}_t$.

$$\hat{h}_t = \tanh(W_{g,x}x_t + W_{g,h}(r_t \circ h_{t-1}) + b_g) \quad (3.10)$$

In the second step, the candidate hidden state is combined with the hidden state of the previous time step to generate the current hidden state.

$$h_t = u_t \circ h_{t-1} + (1 - u_t) \circ \hat{h}_t \quad (3.11)$$

How the previous hidden state and the candidate hidden state are combined is determined by the update gate. If the update gate gives a value of 0 then the previous hidden state is completely disregarded and the current hidden state is equal to the candidate hidden state. If the update gate gives a value of one, it is vice versa.

Advantages vs Disadvantages:

Compared to LSTM, GRU only has 2 gates instead of 3 and one state against the 2 of LSTM. This leads to a more efficient model and faster to train as it needs less memory. However as GRUs don't have a separate hidden and cell state they might not be able to consider observations as far into the past as the LSTM.[\[46\]](#)

3.2 Gradient Boosting Models

Gradient boosting is a machine learning technique where many weak learners, typically decision trees, are iteratively trained and combined to create a highly effective model. It aims to create a strong learner by iteratively minimizing the errors made by the previous models. The core idea is to fit subsequent models to the residuals of the previous models, gradually improving predictions with each iteration [\[47\]](#).

Although these models have traditionally been less common in forecasting, recent research has shown that they can be highly effective in this domain. Some of the key benefits of using gradient boosting models are the ease with which categorical variables can be included in the model without the need for one-hot coding, the ability to capture non-linear relationships between variables and even the high scalability, allowing models to handle large volumes of data.

3.2.1 XGBoost

XGBoost (Extreme Gradient Boosting) is a type of gradient boosting model that uses tree-building techniques to predict its final value. It was introduced by Tianqi Chen [\[48\]](#), it can be used for both regression and classification tasks and has been designed to work with

large and complicated datasets. As the XGBoost documentation states, this algorithm is designed to be highly efficient, flexible, and portable.

"The most important factor behind the success of XGBoost is its scalability in all scenarios. The system runs more than ten times faster than existing popular solutions on a single machine and scales to billions of examples in distributed or memory-limited settings." [48]

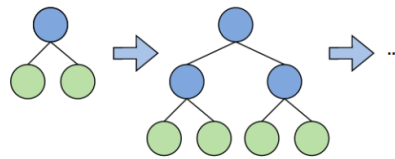


FIGURE 3.4: XGBoost Architecture. [49].

XGBoost operates by sequentially building a series of decision trees, where each new tree corrects the errors made by the previous ones. This sequential process continues until the model reaches a desired level of accuracy or a specified number of trees. Each decision tree in XGBoost is a relatively simple model that makes predictions by dividing the input data into subsets based on feature values. These trees are often shallow, which means they have a limited depth, preventing them from becoming overly complex and prone to over-fitting. It uses a level-wise (breadth-first) tree growth strategy since it grows trees to a specific depth and then prunes them. This uses an exact greedy algorithm to find the best split points by enumerating all possible split values for each feature. This can be computationally expensive, especially for large datasets.[50]

XGBoost optimizes a specified loss function during the training process and its primary innovation is its ability to optimize both the loss function and a regularization term, which prevents the model from fitting noise in the data.

Advantages and Disadvantages:

Time series data often contains nonlinear patterns or even missing values. XGBoost's ensemble of decision trees can effectively capture these nonlinear relationships and even handle missing data by making informed splits during tree construction. XGBoost includes also regularization techniques that prevent over-fitting [47]. This is particularly valuable in time series forecasting, where over-fitting can lead to poor generalization to new data points.

Despite all these advantages, there are some challenges of this algorithm because while XGBoost is effective at capturing short-term temporal dependencies, it may struggle with capturing longer-term patterns inherent in time series data. XGBoost treats each observation as independent, which might not fully exploit the sequential nature of time series data. To solve this, time series dataset must be transformed into a supervised learning problem by structuring the data in a way that captures temporal dependencies (this involves creating lag features, where each observation is associated with past observations' values as features. These features provide the algorithm with the necessary historical context to capture temporal patterns).

3.2.2 LightGBM

LightGBM (Light Gradient Boosting Machine) is an open-source technique created by Microsoft [51] and, as well as XGBoost, it uses decision trees to classify or predict data. Each tree aims to correct errors made by its predecessors, ultimately leading to a highly accurate prediction model.

What sets LightGBM apart is its unique approach to constructing decision trees. Instead of growing trees level by level (level-wise), LightGBM employs a leaf-wise tree growth strategy. It selects the leaf that reduces the loss function the most when expanding a tree. This results in shallower and more efficient trees, speeding up both training and prediction.

LightGBM optimizes the loss function using gradient-based techniques. It computes gradients for data points and applies them during tree construction. This gradient-based approach enhances both the speed of training and the overall predictive performance.

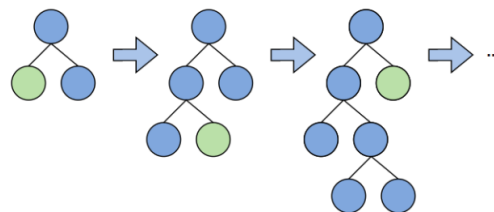


FIGURE 3.5: LGBM Architecture. [49].

One of the main differences between these two algorithms, however, is that the LightGBM tree grows leaf-wise (also known as depth-first). It selects the leaf with the maximum delta loss (reduction in loss) for the next split, which can lead to faster convergence.

However, it can be more prone to over-fitting, so regularization techniques are important.

During training, LGBM uses a gradient-based approach to decide which data points to use for building each new tree. It selects a subset of data points that have larger gradients, which can lead to faster convergence. LGBM is lightweight and requires fewer resources making it fast and efficient.

3.3 Multi-Layer Perceptron (MLP)

Multilayer Perceptron (MLP) consists of multiple layers of interconnected neurons, each layer having nodes (neurons) that process and transmit information using weighted connections.

The architecture includes an input layer of source nodes, that receives the input data for forecasting. Each neuron in the input layer represents a feature or variable that might influence the forecast. For time series forecasting, the input may include historical values, external factors, or lagged observations.

Between the input and output layers, there can be one or more hidden layers. These layers are responsible for learning complex relationships and patterns within the data. Each neuron in a hidden layer is connected to all neurons in the previous and subsequent layers.

The last layer is the output one which produces forecasts for future time steps; at this point the overall mapping of the network input is available. The number of neurons in this layer depends on the type of forecasting task. For example, if we're predicting a single continuous value, there might be just one neuron. If we're forecasting multiple variables, there could be multiple output neurons.

The number of nodes in the input and the output layers depends on the number of input and output variables, respectively. The input signal propagates through the network layer-by-layer.

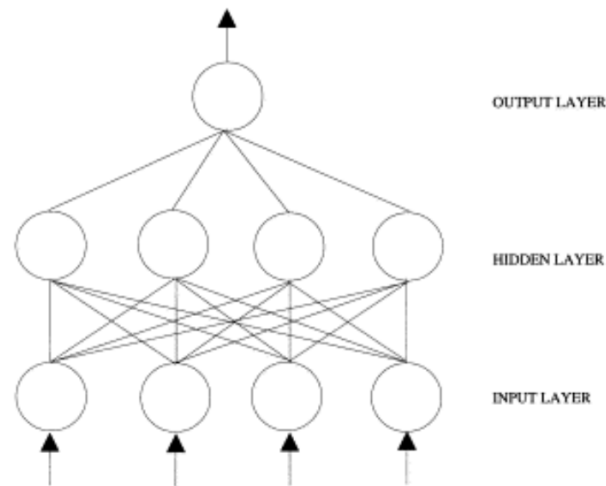


FIGURE 3.6: Multilayer Perceptron Architecture. [52].

For a time series forecasting problem, the inputs are typically the past observations of the data series and the output is the future value. The MLP performs the following function mapping:

$$y_{t+1} = f(y_t, y_{t-1}, \dots, y_{t-p}) \quad (3.12)$$

where y_t is the observation at time t . Before the model can be used, it must be trained. Training is the process of determining the weights of the connections between the nodes in adjacent layers. These connections allow the MLP to carry out complex nonlinear mappings from its input nodes to its output nodes.

The training input data is in the form of vectors of input variables and to each element in this input vector corresponds an input node in the network input layer. The total available data is usually divided into a training set (in-sample data) and a test set (out-of-sample or hold-out sample). The training set is used for estimating the connections weights while the test set is used for measuring the generalization ability of the network.

First, examples of the training set are entered into the input nodes. The activation values of the input nodes are weighted and accumulated at each node in the first hidden layer. The total is then transformed by an activation function into the node's activation value. It in turn becomes an input into the nodes in the next layer, until eventually the output activation values are found. The training algorithm is used to find the weights that minimize some overall error measure.

MLP's strengths lie in its ability to capture nonlinear relationships and adapt to diverse data characteristics. However, to effectively utilize MLP for time series forecasting, several considerations must be addressed.

To ensure MLP's convergence and stability during training, time series data often requires preprocessing, such as normalization or scaling. The ability to capture time-dependent patterns will be powered by the right selecting relevant features or creating lagged variables.

MLP's relationship with time series forecasting is symbiotic. Time series data introduces sequential dependencies, making MLP well-suited to capture and predict future patterns. Conversely, the complexity and capacity of MLP allow it to learn intricate temporal relationships, which can lead to more accurate forecasts.[53]

3.4 Support Vector Machines (SVM)

In conventional machine learning, the Support Vector Machine (SVM) algorithm is primarily employed for classification tasks, where the objective is to find a hyperplane that maximizes the margin between different classes. However, in the context of forecasting tasks, the application of SVM takes a distinctive turn.

While SVMs are traditionally known for their prowess in classifying data into distinct categories, they can also be effectively harnessed for forecasting endeavors. In forecasting tasks, the emphasis shifts from merely separating classes to identifying a hyperplane that aptly characterizes the behavior of data points, extracting meaningful patterns, relationships, and trends.

The goal in forecasting with SVMs is not just to maximize the difference between different points but to discern a hyperplane that captures the underlying characteristics of the data. This adapted approach allows us to extract valuable insights from historical data and use them to make informed predictions about future trends, values, or events. In essence, the SVM in forecasting seeks not only separation but a representation of the intrinsic patterns that enable extrapolation to future scenarios.

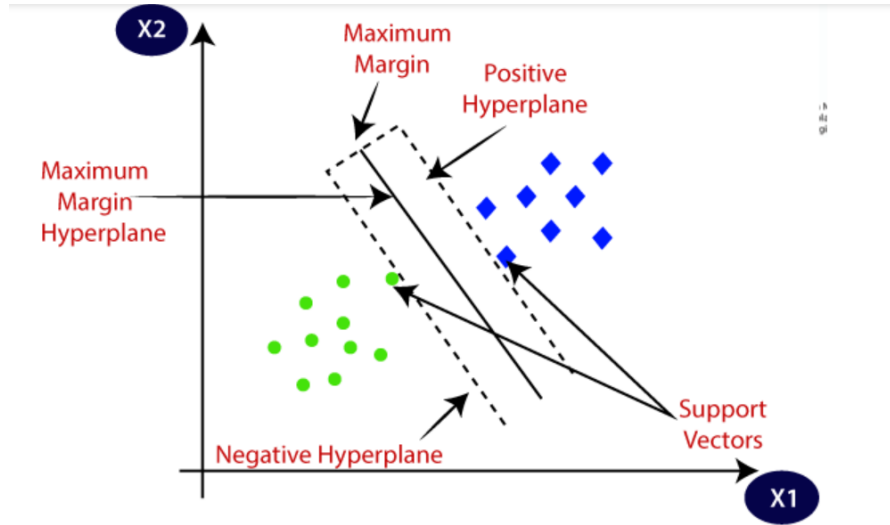


FIGURE 3.7: SVM Algorithm. [55].

For the linearly separable case (the classes can be perfectly separated and no data points from different classes lie on the wrong side of this separating boundary), a hyperplane separating the binary decision classes in the three-attribute case can be represented as the following equation:

$$y = w_0 + w_1x_1 + w_2x_2 + w_3x_3 \quad (3.13)$$

where y is the outcome, x_i are the attribute values, and there are four weights w_i to be learned by the learning algorithm. These parameters determine the hyperplane. The maximum margin hyperplane, in terms of the support vectors, can be represented by:

$$y = b + \sum_{i=1}^n \alpha_i y_i \cdot x(i) \cdot x \quad (3.14)$$

where y_i is the class value of training example $x(i)$, $\cdot$ represents the dot product and n the number of support vectors. The vector x represents a test example and the vectors $x(i)$ are the support vectors. In this equation, b and α_i are parameters that determine the hyperplane.

For the nonlinearly separable case (the classes cannot be perfectly separated thus the data points might be mixed together in a more complex manner), a high-dimensional version of Eq. (3.14) is simply represented as follows:

$$y = b + \sum_{i=1}^n \alpha_i y_i K(x(i), x) \quad (3.15)$$

where n is the number of support vectors. The function $K(x(i), x)$ is defined as the kernel function and it allows SVM to implicitly transform the original feature space into a higher-dimensional space where the classes might become linearly separable.

The transformed data points are then classified using a hyperplane in this new space. There are some different kernels for generating the inner products to construct machines with different types of nonlinear decision surfaces in the input space. Choosing among different kernels the model that minimizes the estimate, one chooses the best model.

Over-fitting is unlikely to occur with SVM since this one may be caused by too much flexibility in the decision boundary but the maximum hyperplane is relatively stable and gives little flexibility.

In conclusion, the preceding discussion delves into the application of Support Vector Machines (SVMs) specifically tailored for classification tasks. The highlighted equations and concepts illustrate the formulation of hyperplanes in both linearly and nonlinearly separable cases, emphasizing the SVM's capability to distinguish between different classes in the feature space. Now, when considering SVM for forecasting tasks, a natural extension arises. While the text primarily addresses classification scenarios, the fundamental principles of SVM—such as identifying optimal hyperplanes, leveraging support vectors, and utilizing kernel functions for nonlinearity—remain relevant. In forecasting tasks, these principles can be adapted to discern patterns and relationships within historical data, enabling accurate predictions for future data points. Thus, the foundational concepts explored in SVM classification serve as a valuable groundwork for understanding and implementing SVMs in the realm of forecasting.

3.5 Random Forests

Random Forests, introduced in 2001 by Breiman [56], were initially designed for cross-sectional data analysis but can be adapted for time series forecasting by treating it as a regression problem.

In essence, a Random Forest model is an ensemble of many decision trees, often referred to as "weak learners." This ensemble approach is an extension of a technique called bootstrap aggregation or bagging. In bagging, multiple decision trees are created, with each tree generated from a different bootstrap sample of the training dataset. A bootstrap sample is essentially a random subset of the training data, and what's unique is that an

example can appear more than once in this sample, which is known as "sampling with replacement."

Bagging is a potent ensemble algorithm because it introduces variability into each decision tree's training process. Unlike traditional decision tree models like Classification and Regression Trees (CART), the trees used in a Random Forest ensemble are left unpruned. This lack of pruning results in these individual trees being slightly overfit to the training data. This over-fitting is actually desirable in this context because it makes each tree more distinct and less correlated in terms of predictions or prediction errors.

Each tree in the Random Forest makes its own decisions based on the available data. The final prediction is then obtained by averaging the predictions from all the trees.

This averaging process, where each tree contributes its own decision, contributes to the strength of random forest regression compared to many other algorithms. Another crucial aspect of Random Forest is the feature selection process. At each split point in the construction of the decision trees, only a subset of input features (variables) is considered. In typical decision tree construction, all input variables are evaluated to select the best split. However, in Random Forest, this is restricted to a random subset of variables at each split point. This limitation forces each decision tree in the ensemble to be more distinct.

The result of this feature selection is that predictions and prediction errors made by individual trees in the ensemble are less correlated. When these less correlated predictions are combined by averaging, it often results in improved performance compared to simple bagged decision trees.

For time series forecasting, each tree can be trained on a subset of historical time series data. In this context, the input features are typically lagged values of the time series itself. This approach allows the model to capture temporal dependencies and patterns inherent in the data. The ensemble nature of Random Forests is advantageous for capturing complex patterns and relationships in time series data, making it a valuable tool for forecasting tasks.

Chapter 4

Error Metrics

To evaluate the previous models it's crucial to evaluate their performance. These measures allow us to assess how well is the model predicting future values and provide insights into the accuracy and reliability of the forecast done.

- **MAE (Mean Absolute Error)**

Starting with the MAE (Mean of Absolute Errors), this measure is presented as a simple but effective measure. It measures the average absolute difference between the predicted values and the actual values. [57]

This one is a linear score once all the individual errors are equally weighted in the average. Therefore, the large errors are not penalized, which may be very helpful. A notable drawback of utilizing MAE stems from its reliance on absolute values, a characteristic that is undesirable in numerous mathematical computations. [58]

$$MAE = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i| \quad (4.1)$$

- **MSE (Mean Squared Error)**

MSE measures the average of the squared differences between predicted and actual values.

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2 \quad (4.2)$$

This one amplifies the impact of larger errors due to squaring. Also, it is not easily interpretable since it's expressed in squared units of the data. An alternative to this

disadvantage is Root Mean Squared Error (RMSE). It does not differentiate between overestimation and underestimation and $RMSE = \sqrt{MSE}$.

- **MPE (Mean Percentage Error)**

MPE measures the average percentage difference between predicted and actual values but it can suffer from sign issues. Its ability to interpret errors in terms of percentage, makes it easier to understand and compare across different models.

$$MPE = \frac{1}{n} \sum_{i=1}^n \frac{(y_i - \hat{y}_i)}{y_i} \times 100 \quad (4.3)$$

- **MAPE (Mean Absolute Percentage Error)** Mean Absolute Percentage Error (MAPE) calculates the average percentage difference between the predicted values and the actual values. MAPE is calculated by taking the absolute difference between the predicted and actual values, dividing it by the actual value, and then averaging the results. One disadvantage (in common with MPE) is the fact that cannot handle zero values since division by zero is not possible. MAPE treats overestimations and underestimations differently, which means that it is not a symmetric metric. [59]

$$MAPE = \frac{1}{n} \sum_{i=1}^n \left| \frac{(\hat{y}_i - y_i)}{y_i} \right| \times 100 \quad (4.4)$$

Chapter 5

Real World Case Scenario

5.1 E-commerce Sales Forecasting: Context

In the fast-paced world of e-commerce, where trends can change overnight, sales forecasting has emerged as a vital tool for businesses aiming to maintain a competitive edge.

One of the key advantages of accurate sales forecasting is its capacity to enable companies to allocate resources with exceptional efficiency and therefore improved customer satisfaction. When customers can find the products they want when they want them, they are more likely to have a positive shopping experience and become repeat buyers. This can lead to increased customer loyalty and higher revenues for e-commerce businesses. Besides this, e-commerce companies can now easily meet demand, reduce lead times and to set competitive and profitable pricing strategies, adjust prices based on expected market conditions.

E-commerce sales forecasting involves the use of historical sales data and various algorithms to predict future demand for products.

For our study case we will use a dataset which contains information about daily sales during 3 years (from 2017/01/01 to 2019/12/03) from an e-commerce company.

To unlock the data potential, it needs to be prepared and formatted appropriately before putting it through the analytics pipeline. Such pre-processing can involve estimating missing values, removing outliers and accounting for seasonal variations. After data preparation, we will proceed with building and applying models.

5.2 Data preparation and processing for daily granularity

The following analysis will be performed in Python. In our time series Y_t , $t \in N$ represents the sales amount on a specific day.

The first analysis to be carried out was checking for missing values. These represent 8% of our series. To avoid removing them and leaving gaps in the data, it has been applied the technique of replacing them with the average sales value. After this we plot our series. In this daily time series graph, we examine a dataset representing daily sales for a specific store over a given period. This first visualization is crucial to start our analysis since many key insights can be derived from this graph.

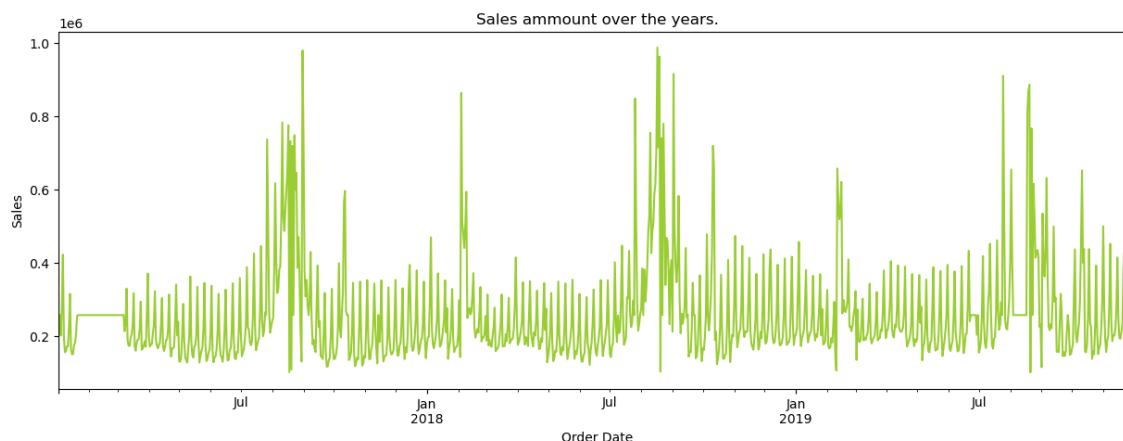


FIGURE 5.1: Time Series Plot (Daily Granularity).

We can observe daily fluctuations which can be attributed to operational factors such as the day of the week, holidays, promotional or seasonal events, all of which have a direct impact on daily sales. For example, one thing we can learn from this visualization is the existence of a sales peak every year between August and September.

Due to the granularity of daily data, it can be challenging to identify long-term trends or substantial changes in sales over time. Daily fluctuations may obscure these trends.

To gain clearer insights into long-term trends, it may be more beneficial to aggregate data at a lower granularity, such as weekly or bi-weekly, for simplified analysis and clearer trend identification. In short, while daily data provides detailed information, this detail level can increase the complexity of our analysis.

For now, the initial approach is to analyse the data in its raw form, even though it presents a high level of complexity, and study its components and characteristics. The reason for this approach is to gain a deep understanding of the underlying patterns, trends, and anomalies within the dataset. If the models struggle to provide accurate forecasts or fail to capture the nuances of this raw time series, it will be important to consider alternative strategies.

To spot trends and variations over different years we will create a year-wise boxplot. The idea here is to create a boxplot for each year to visualize how the data's trend varies from one year to the next. A boxplot is a graphical representation that displays the distribution of a dataset. It consists of a box, which represents the interquartile range (IQR) of the data, along with "whiskers" that extend to show the data's range. Any data points that fall outside the whiskers are considered outliers.

In this context, we will group our e-commerce data by years and create a separate boxplot for each year. This allows us to see how the central tendency and spread of data change from one year to another. Similarly, we will construct a month-wise boxplot, that generates a boxplot for each month. This visualization helps us to understand the seasonal variations in data such as higher sales during certain months (e.g., holiday seasons) or lower sales during others.

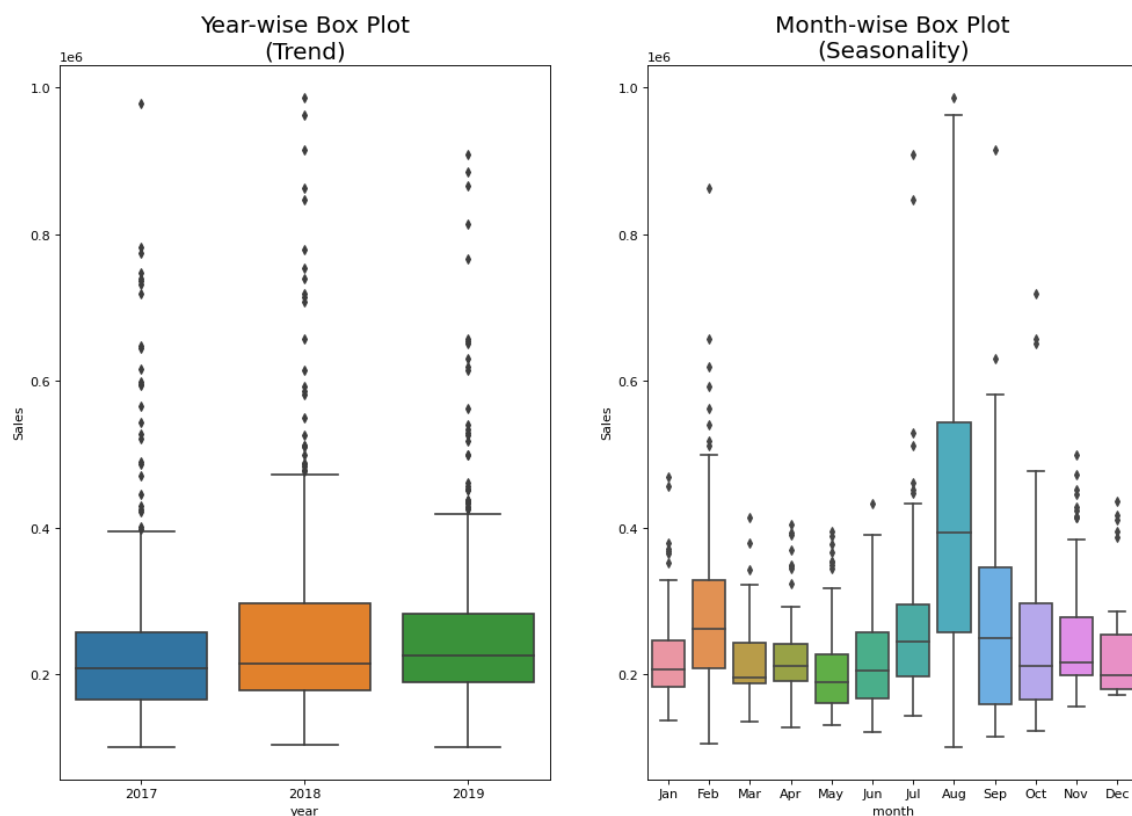


FIGURE 5.2: Year-wise boxplot & Month-wise boxplot

Analyzing the first graph, the median in the center of the box indicates a central trend in sales over time. The height of the box also does not vary significantly which indicates that there is no variability in the data.

However, between 2017 and 2018 there was an increase in the length of the “whiskers” (lines outside the box) which indicates an increase in data dispersion and in turn an indication of changes in sales over time.

The summer months are the ones with the highest number of sales for the company (as we saw earlier in the daily data plot). This increase may suggest seasonality in the data.

The presence of outliers makes it challenging to discern clear trends; thus, by removing them, we will obtain a more focused and accurate analysis of the underlying patterns. To confirm this thesis, we will plot the boxplots without the outliers.

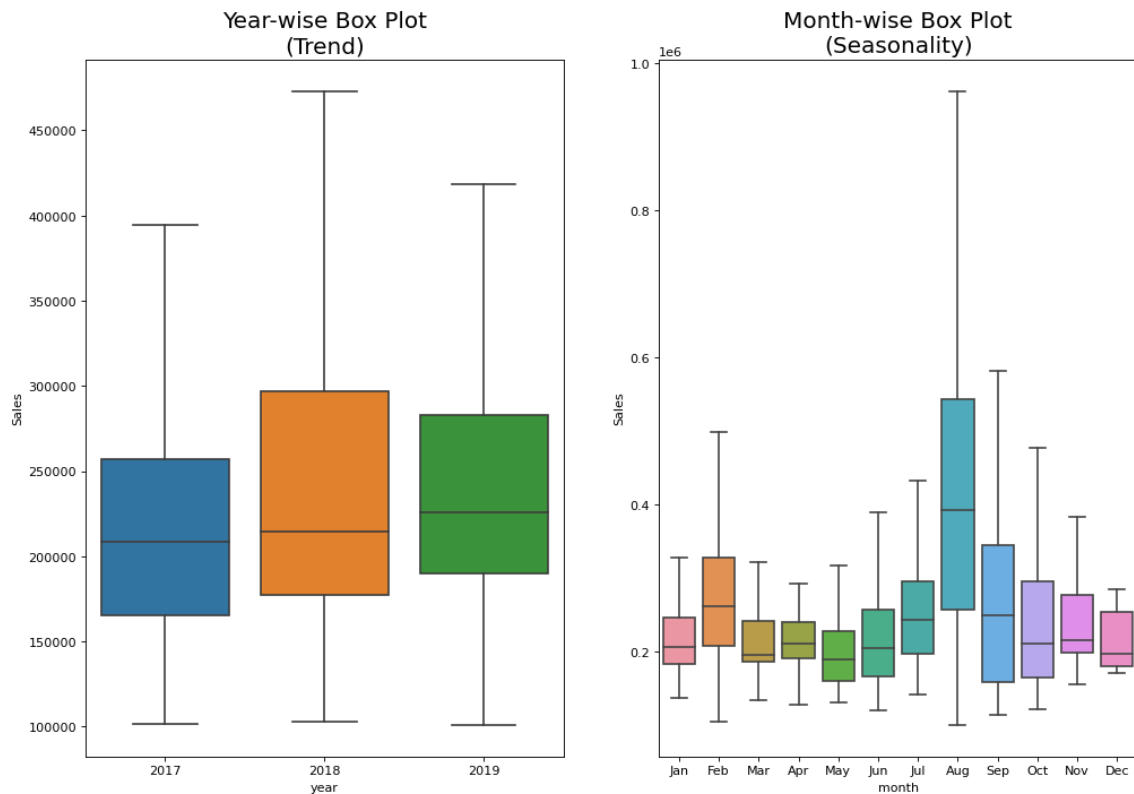


FIGURE 5.3: Year-wise boxplot & Month-wise boxplot without outliers

Our initial conclusions, from the first boxplots, seem to be correct. To confirm the conclusions drawn from these graphs, we will decompose the series.

The isolation of components allows a more detailed analysis of each one, being especially useful when patterns in data are not obvious in simpler visualizations, such as those plotted previously (line graph or boxplots).

There are 2 types of decomposition. The additive decomposition assumes that the series is a linear combination of the three components: $\text{Time Series} = \text{Trend} + \text{Seasonality} + \text{Residuals}$. It's used when the amplitude of seasonality appears constant over time, that is, the variation in seasonality is approximately constant. The multiplicative decomposition assumes that the series is a multiplicative combination of the three components: $\text{Time Series} = \text{Trend} \times \text{Seasonality} \times \text{Residuals}$. It is used when the amplitude of seasonality varies with the level of the series, that is, the variation in seasonality is proportional to the level of the series.

In this case, we will use an additive decomposition.



FIGURE 5.4: Time Series decomposition

The trend component represents the general direction of the time series over time, that is, whether there is an upward slope (growing trend) or downward slope (falling trend) in the observations.

If we look at the graph 5.4 above, this upward or downward behavior is not visible. This reinforces the idea taken from the boxplots that this series does not present a trend.

In relation to seasonality, there is a recurring pattern, that is, there are regular and predictable variations in the data at fixed time intervals. Therefore this indicates that there may be seasonality and hence it can be modeled. The residuals hover around zero. This is a good sign, because models often assume that residuals are stationary.

Recovering previous chapters of this report, we saw that many models require the time series to be stationary. In case it's not, we must apply methods that provide these characteristics. For practical purposes we can assume the series to be stationary if it has constant statistical properties over time: constant mean and constant variance.

For that we can plot the moving average or moving variance and see if it varies with time. If the moving average remains relatively constant over time, it suggests stationarity, as there are no significant trends or shifts in the data. If the moving variance or standard deviation remains relatively constant over time, it suggests stationarity, as there is consistent dispersion of data points.

Important to note that rolling statistics are meant to capture trends and patterns over time. To do so, they rely on historical data within the rolling window. Only after the rolling window has moved beyond the initial data points it will start producing meaningful rolling statistics.

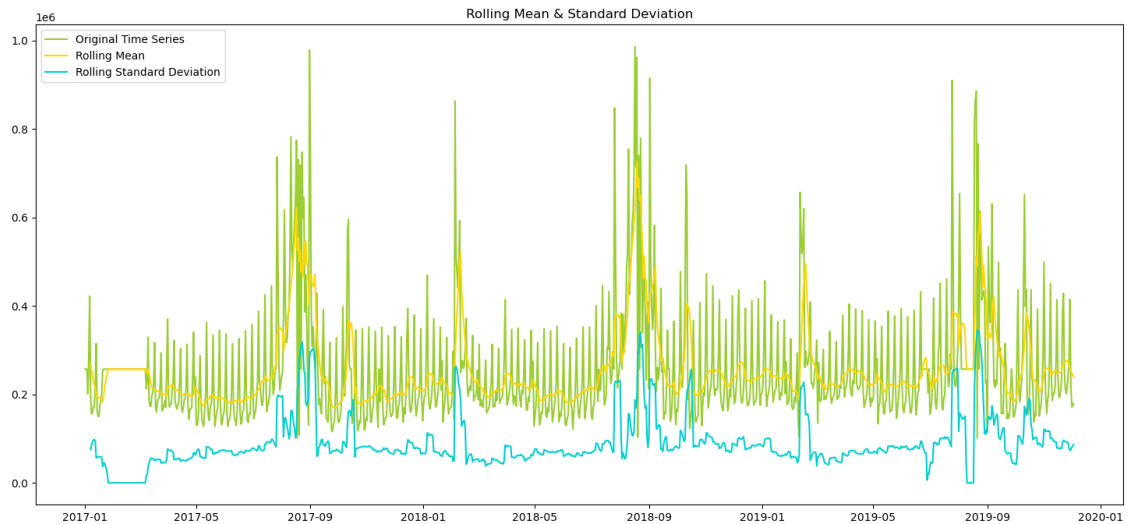


FIGURE 5.5: Rolling statistics Plot

Due to the huge amount of data, it's difficult to analyse the rolling mean and rolling variance.

Let's proceed to the second approach to test stationary: ADF test e KPSS test.

In accordance with the conventional acceptance of statistical significance at a P-value of 0.05, CI are frequently calculated at a confidence level of 95%. In general, if an observed result is statistically significant at a P-value of 0.05, then the null hypothesis should not fall within the 95% CI.

Test Statistic	p-value
-4.699	8.400×10^{-5}

TABLE 5.1: ADF Test Results

The p-value is lower than the significance level, so we reject the null hypothesis and this means that the time series is stationary. In other words, it lacks time-dependent patterns or changes, leading to consistent data variability (spread of values) over time.

Now, let's focus on the KPSS test. This one focuses on detecting stationarity around a stochastic trend, which is different from the ADF test approach.

Test Statistic	p-value
2.094	0.01

TABLE 5.2: KPSS Test Results

The p-value obtained from the KPSS test is equal to 0.01, so the null hypothesis should be rejected. This means that the data does not exhibit stationarity around a deterministic trend and indicates non-stationarity.

Now we are faced with an intricate situation: the ADF test says that the series is stationary and the KPSS test in turn says that it is non-stationary. But the reality is that they do not cancel each other out.

The ADF test is focused on detecting the presence of a deterministic trend in the time series. The KPSS test focuses on detecting stationarity around a stochastic trend. The contradiction between the results of these two tests can occur when the series has characteristics that do not fit well with either hypothesis.

What the tests suggest is that the series exhibits a clear, consistent trend over time, but this trend can be removed through differencing or detrending techniques, leaving behind a stationary series. Let's then apply a differentiation (first difference) to remove the trend. Then, the ADF and KPSS tests will be applied again to the differentiated data to check stationarity.

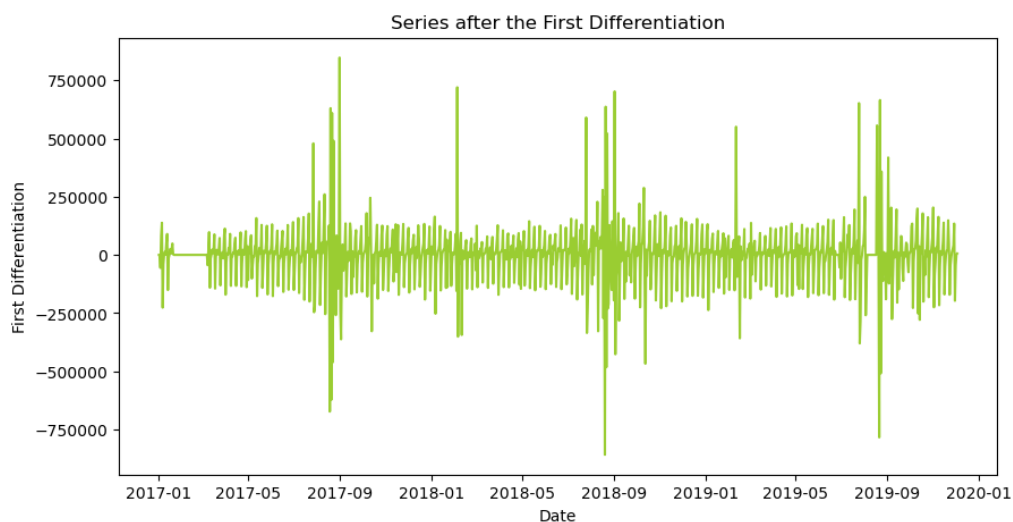


FIGURE 5.6: Series after First Differentiation

The key advantage of the previous graph is that if the original series had significant trends or temporal patterns, the first difference often removes these features, making

the series more stationary. We can see that the plot shows a series of fluctuations around a mean value. However, some peaks continue to stand out alone. This shows that on consecutive days there was an abrupt oscillation in sales (hence the difference in values being high in absolute terms).

This first differentiation curve also assumes an approximately constant mean over time. This is an indicator that this differentiation eliminated the trend and seasonality of the original series. The peaks and valleys in the differences represent changes between consecutive values, which can be attributed to random noise or variability. Now to confirm our suspicions coming from the graph, we carried out both ADF and KPSS tests.

Test Statistic	p-value
-8.531	1.038×10^{-13}

TABLE 5.3: Augmented Dickey-Fuller Test Results

Test Statistic	p-value
0.107	0.100

TABLE 5.4: KPSS Test Results

The ADF test result shows that the p-value is extremely low (close to zero). With such a low p-value, we can reject the null hypothesis and conclude that the time series is stationary.

In the KPSS test the null hypothesis is that the series is stationary around a deterministic trend. As the p-value is greater than the significance level of 0.05 (5%) then we cannot reject the null hypothesis, which suggests that the series may be stationary around a deterministic trend.

Therefore, the series after the first differentiation is stationary and this will be the target of the following first investigations.

It's important to note that the series after the first differentiation contains negative values and, although we are working with sales that only accepts positive values, this is not necessarily a problem. Negative values after differentiation can occur when the original series has fluctuations that result in positive and negative values in the differences. The objective of differentiation is not to create a series with all positive values, but rather to make the series stationary so that it is possible to apply forecasting models appropriately.

When making predictions using a model adjusted to the differentiated series, at the end of the process it is important to add the calculated differences to the original values to reconstruct the predictions at the original scale.

1st Investigation Resume

Scope: Time Series after first defferenciation

Normalization Technique: MinMaxScaler

Split Technique: Fixed Split (80% for the train set and the remaining 20% to the test)

Lookbacks considered: 7, 20, 30

Models Applied: RNN, LSTM, GRU, XGBoost, LightGBM, MLP, SVM, Random Forests

After conducting appropriate statistical tests and cleaning our data we are now able to apply Machine Learning models to our data to forecast the sales from the company in study. One good practice while using ML models is normalize the data. Despite our input features having all the same scale, normalization can help the algorithm to converge more quickly so we scaled the values between [0, 1] using *MinMaxScaler()* Python function.

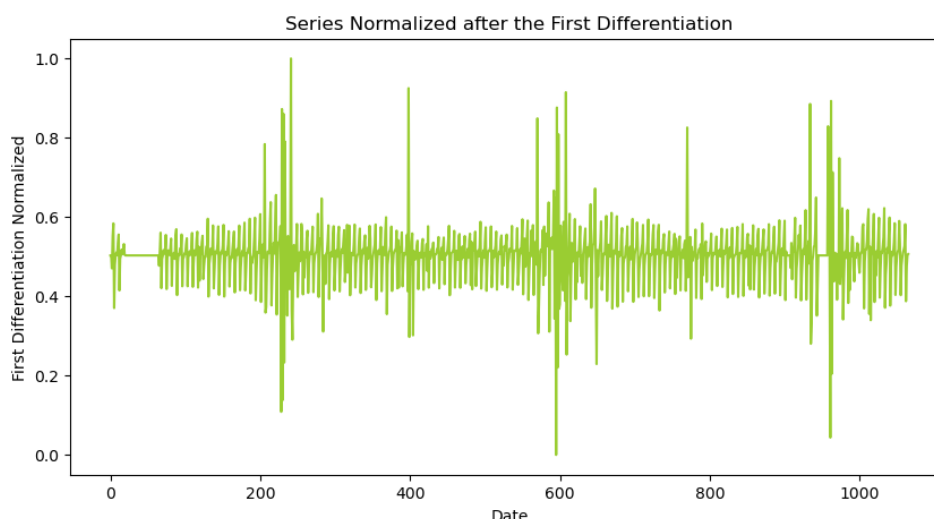


FIGURE 5.7: Series Normalized after First Differentiation

One first crucial step is dividing the dataset into training and testing subsets. With the train set, the model will learn the underlying patterns, relationships, and associations within this data. As the model iteratively processes the training data, it

becomes better at making predictions based on the input features. After the training phase is complete, the model's performance will be evaluated using the test set, which the model has never seen during training. This evaluation assesses how well the model can make predictions on data it hasn't encountered before. If the model performs well on the test dataset, it suggests that it has successfully learned the underlying patterns in the data and can make reliable predictions on real-world data.

For the following models, it was used 80% of the data for training and the remaining 20% for testing, ie, for the time series with 1066 elements (one less than the original series due to consecutive subtractions) we define our training set with 853 elements and a test set with 213. Thereby, we will have 4 sets: $x_{train}, y_{train}, x_{test}$ and y_{test} .

The training set contains historical data up to a certain point in time. x_{train} represents the inputs (historical data), and y_{train} represents the corresponding outputs (target values) at those historical points (in other words, x_{train} is the previous points in time that we use to predict the following values). At this stage, we are not predicting the future; we are using past data to teach the model how to make predictions.

The test set contains a subsequent portion of the time series. The x_{test} are also earlier points in time, but are subsequent to the points in x_{train} . The y_{test} are the actual or target values corresponding to x_{test} . The aim is to use the trained model to predict values based on x_{test} and see how well these predictions match the real values in y_{test} to judge the model's performance.

Furthermore, the same sets were always considered when applying all the models so that we could later compare them.

The lookback or lag is a parameter used in time series forecasting models to specify how many past data points or time steps the model should consider when making predictions for future time steps. The choice of the lookback value is important because it directly influences the model's ability to capture patterns, seasonality, and dependencies in the data.

If the lookback is too short, the model may miss important patterns in the data. If it's too long, the model may become overly complex and overfit the data. So, to start, it has been choosed a lookback of 7. This is a reasonable starting point because it aligns with the natural weekly seasonality in the data. Besides this lookback value we should proceed to hyperparameter tuning, that is, try different lookback values

in order to find the one that minimizes the forecasting error. For that it will be used $\text{lookback} = 20$ and $\text{lookback} = 30$.

There's often a trade-off between a longer look back (which can capture more historical information) and model complexity. Longer lookbacks may require more parameters and computational resources. Therefore, it's important to strike a balance that provides accurate forecasts without overcomplicating the model.

Now that we have processed our data, we have transformed our series into stationary, we have divided the series into training and testing sets and we have defined the look backs we are ready to apply the models.

5.3 Stock Forecast using Machine Learning for daily granularity

Recurrent Neural Networks (RNN) with daily granularity

When building the RNN model, there are several parameters necessary to define to configure the architecture and behavior of the network, namely the number of neural units that will make up the RNN layer. It's important to note that a larger number of units can handle more complex relationships but it can also increase the risk of over-fitting. Therefore, different parameter values were tested in order to obtain the lowest possible error without over-fitting: 32, 64, 128 and 256.

Although *a priori* we have some indications of the range in which these parameters should be found, the best choice was made through a process of trials.

In addition to this parameter, it was needed to define the activation function, responsible for introducing non-linearity in the outputs of the layers of a neural network. Several functions were tested (ReLU, Hyperbolic Tangent Function, Sigmoid Function) and the popular ReLU was chosen as it helped to overcome the problem of "vanishing gradients".

The model was applied to three different historical windows: sizes 7, 20, and 30, which means that we will use historical windows with 7, 20, and 30, respectively, sales values to predict the next value in the time series. This sliding window allows the model to learn the relationship between the last x data points and the next value in the series.

RNN with Lookback = 7

In our model, X_{train} is a list of training data subsequences, each with 7 data points (up to the point where it's possible to form a historical window of size 7). Each of these sequences is a matrix with 7 rows and 1 column (as we are dealing with a single feature or variable - Sales) and y_{train} is a list of corresponding target values (the values we want to predict) for each training subsequence.

These lists X_{train} and y_{train} will then be transformed into NumPy arrays to be used as input in training the RNN.

The importance of this data preparation lies in creating input-output pairs that are suitable for training a RNN. The RNN, being a sequential model, learns based on temporal relationships in the data. By providing the RNN with input sequences of size 7 and their respective target values, we are training the model to capture temporal dependencies and make predictions based on the information from the last 7 data points.

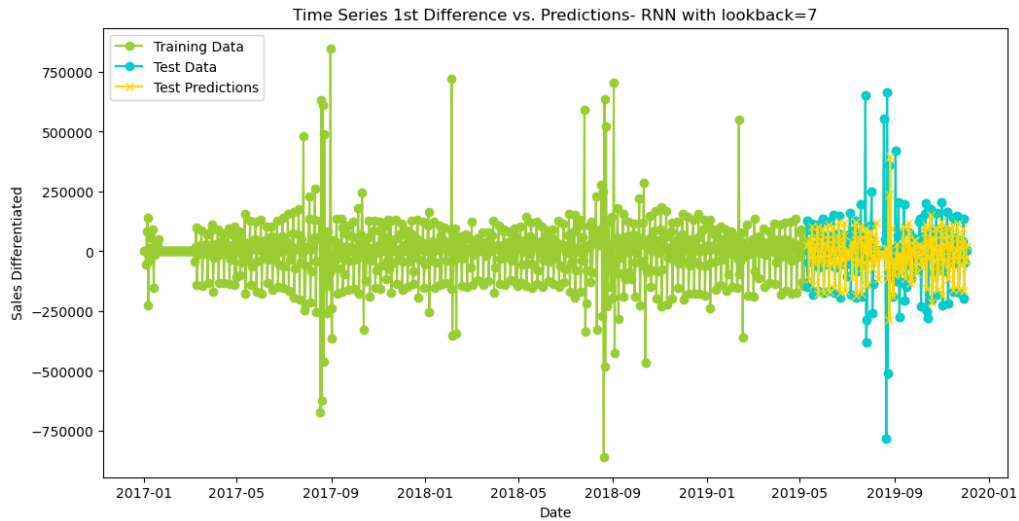


FIGURE 5.8: RNN applied to Time Series 1st Difference with lookahead=7

At first glance we can see that the model is detecting the peaks that stand out. This adaptability is a positive point of this model

Let's check what happens when we increase the historical size to consider.

RNN with Lookback = 20

It has been obtained almost the same results when using a lookahead of 20 in this forecasting model, except for the peaks.

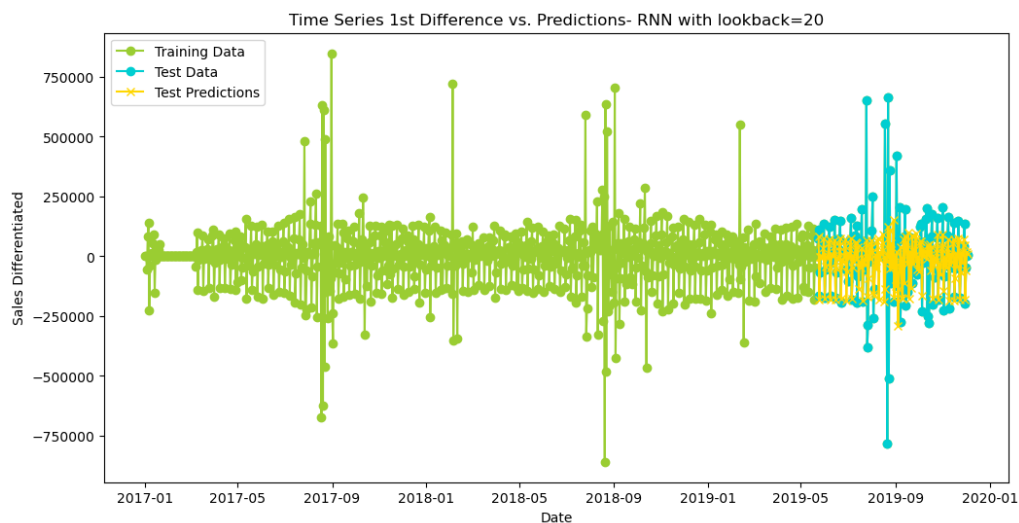


FIGURE 5.9: RNN applied to Time Series 1st Difference with lookahead=20

This suggests that the model with this lookahead is struggling to capture meaningful patterns or relationships in data. The similarity in results between lookahead=7 and lookahead=20 indicates that the time series may not exhibit strong short-term patterns or dependencies that can be effectively captured within a 20-time step window.

But if we think about the nature of sales data, trying to abstract ourselves from a more analytical view and adopting more of a business view, some consumers prepare monthly budgets, leading to regular spending patterns tied to the beginning of each month. Furthermore, companies usually schedule promotions, discounts or marketing campaigns monthly to attract customers, leading to periodic sales peaks. Stock replenishment can also contribute to a monthly pattern and logically, when there is no stock there is a decrease in sales. By knowing this, we should test a lookahead of 30 days.

RNN with Lookback = 30

This model is being trained to capture temporal dependencies and make predictions based on the information from the last 30 data points, that is, the last month.

The predicted values assume a straight behaviour around 0. This poor prediction may be caused by several factors.

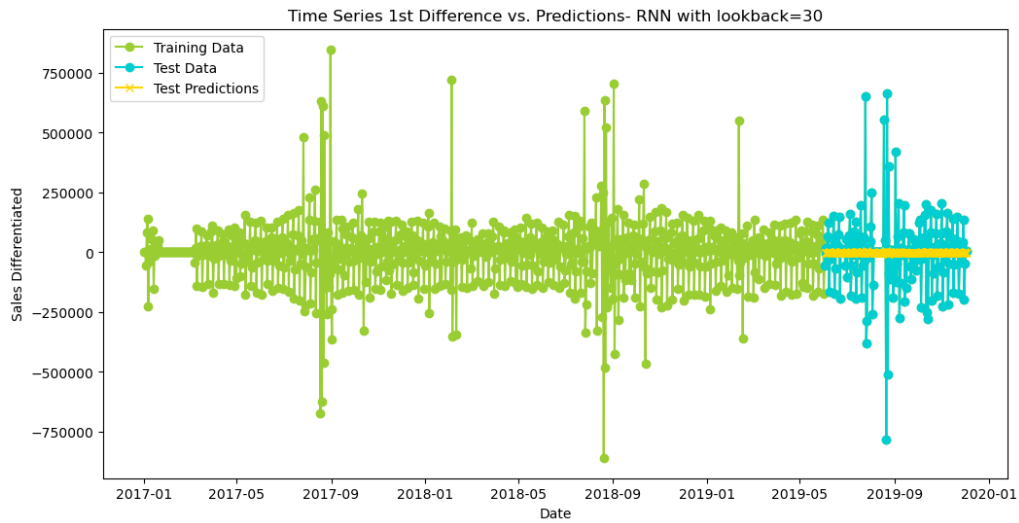


FIGURE 5.10: RNN applied to Time Series 1st Difference with lookback=30

When using a short historical window, the model relies on a very limited set of past data points to make predictions. This limited historical context might not capture the complex patterns or seasonality present in our sales data. As a result, the model might struggle to make accurate forecasts. Since the dataset contains 1066 elements with this lookback value we are discarding a significant portion of data. With such a short lookback, the model may oversimplify the problem and assume that the best prediction it can make is to estimate a constant value (in this case, approximately 0). This is because it doesn't have enough historical context to make more nuanced predictions.

The model architecture or complexity may still be insufficient to capture the nuances in this sales data, even with a longer lookback. Traditional RNNs may struggle with capturing complex thus it should be considered the use of more advanced architectures like Long Short-Term Memory (LSTM).

Long Short-Term Memory (LSTM) with daily granularity

To achieve the most accurate results in this sales forecasting it's relevant to focus on the parameters selection and model optimization tailored to the unique characteristics of daily sales patterns. Once again one of the pivotal parameters needed to be defined was the lookback value. For LSTM and all the following models it will be used the values 7,20 and 30 so it will be possible to compare the results from the different models.

In addition other critical components were selected, including the loss function and optimizer. For the loss function, we opted for the Mean Squared Error (MSE), a robust metric that quantifies the squared difference between our model's daily sales predictions and the actual daily sales values. MSE serves as an effective measure of prediction accuracy, enabling the model optimization for minimizing forecasting errors.

Furthermore, it has been conducted some tests with various optimizers, including *adam*, *sgd* and *rmsprop*. Through this systematic exploration, *adam* optimizer consistently delivered the best results for our daily sales forecasting task. The *adam* optimizer's adaptive learning rate and momentum optimization capabilities proved crucial in guiding this model toward convergence, resulting in more precise and stable daily sales predictions, comparing to RNN.

LSTM with Lookback = 7

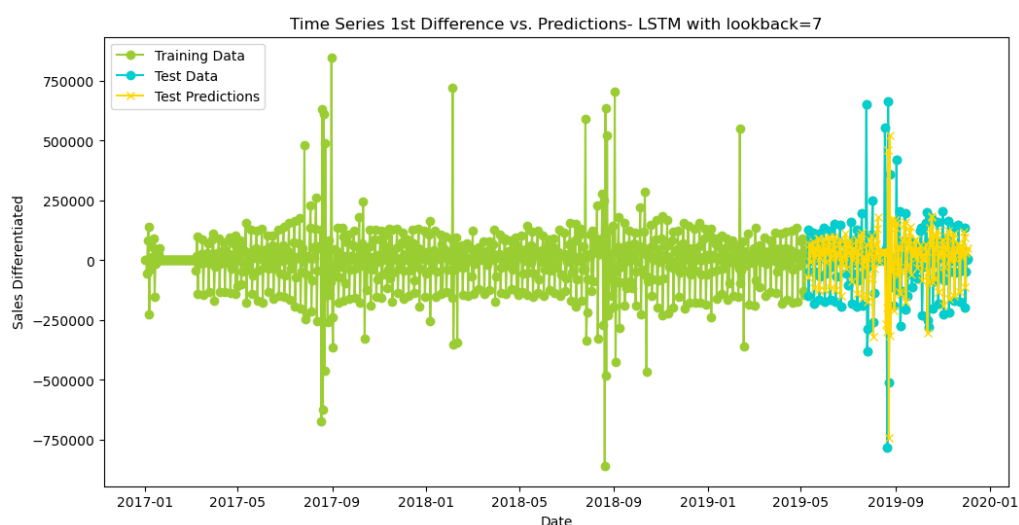


FIGURE 5.11: LSTM applied to Time Series 1st Difference with lookback=7

Compared to traditional RNNs, LSTM networks have enhanced capabilities for capturing longer-term dependencies and mitigating the vanishing gradient problem.

However, despite these advantages, LSTM may still face challenges in achieving low forecasting errors. One possible reason for this lack of accuracy is the lower lookback- maybe this value is not enough for LSTM to capture long-terms tendencies.

LSTM with Lookback = 20

Analyzing the graph with a lookback of 20, it's evident that the LSTM model shows a significant number of predicted values both above and below zero. Notably, the model performs equally well in both the positive and negative regions of the graph which highlights the model's ability to adapt to diverse data patterns.

This suggests that the LSTM model is responsive to data's fluctuations, showcasing its versatility in capturing varied historical trends, whether positive or negative.

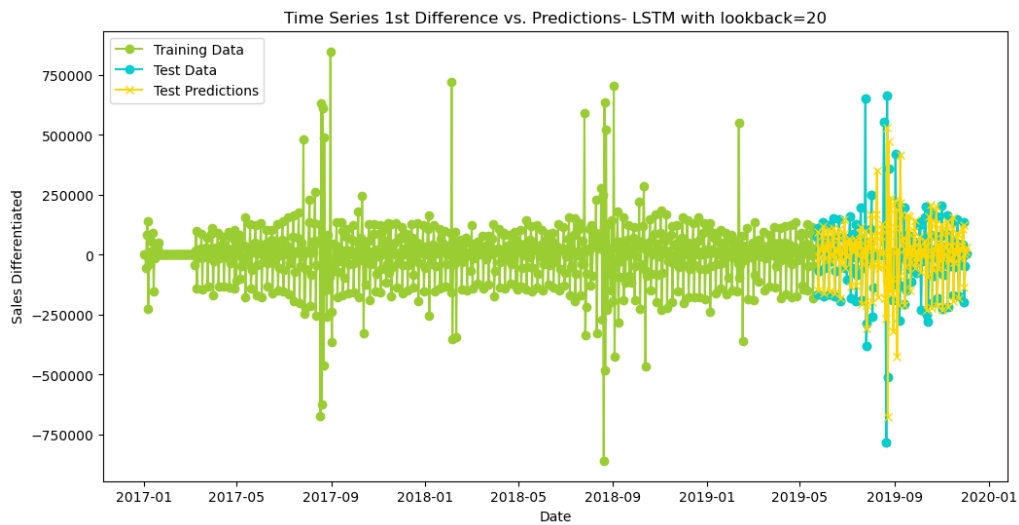


FIGURE 5.12: LSTM applied to Time Series 1st Difference with lookback=20

From our experience with applying the RNN model, having the same behavior with lookback 7 and 20, probably increasing the lookback will not change much the distribution of our predicted data since LSTM may not be strong enough to capture monthly trends, if these exist. So we need to test another approach like changing the number of epochs and the batch size.

LSTM with Lookback = 30

The previous two LSTM's application used 300 epochs with a batch size of 1. The parameter epoch determines the number of iterations through the entire training dataset thus, so far, the models updated its internal parameters (weights and biases) 300 times based on the training data. The batch size refers to the number of data points used in each update of the model's parameters during one epoch. A batch size of 1 means that the model updates its parameters after processing each individual data point. So the model made small but frequent updates which can lead to

noisy training and slower convergence. For this new value of lookback it has been tested batch sizes of 32,64 and 128. The best results were achieved with batch size = 64.

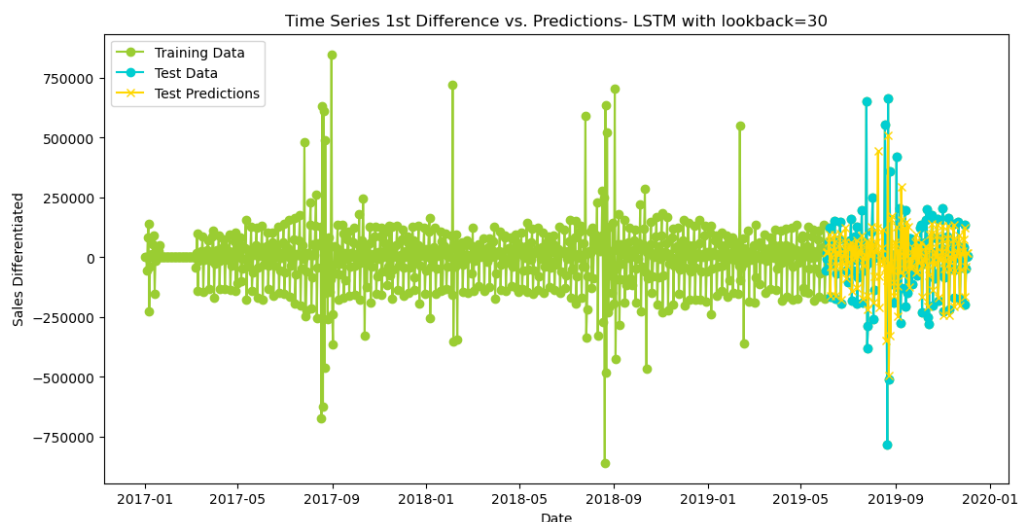


FIGURE 5.13: LSTM applied to Time Series 1st Difference with lookback=30

The results are very similar which suggests that the model is learning the underlying patterns. GRU is a simpler variant of the LSTM with fewer gates, which can make it easier to train and less prone to vanishing gradient problems.

Gated Recurrent Unit (GRU) with daily granularity

GRU model has some parameters that should be choose in order to achieve better results. Starting with the parameter *units*. This value sets the number of GRU units or neurons in the GRU layer responsible for capturing and learning patterns in the input data. After the GRU layer, a Dense layer with one neuron is added. This is the output layer responsible for producing the final prediction. The single neuron is used to produce a single continuous value as the forecast.

Similar to LSTM the model was trained using the *adam* optimizer and the *mean squared error* loss function for 300 epochs, with a batch size of 1. GRU may be preferred when we have limited training data, as its simplicity can help avoid over-fitting.

GRUs are often more effective at capturing short-term dependencies in data. In time series forecasting, this can result in predictions that closely follow rapid fluctuations and sudden changes in the data. RNN and LSTM networks tend to capture

longer-term dependencies but as we saw before, this might not be as relevant in this scenario.

GRU with Lookback = 7

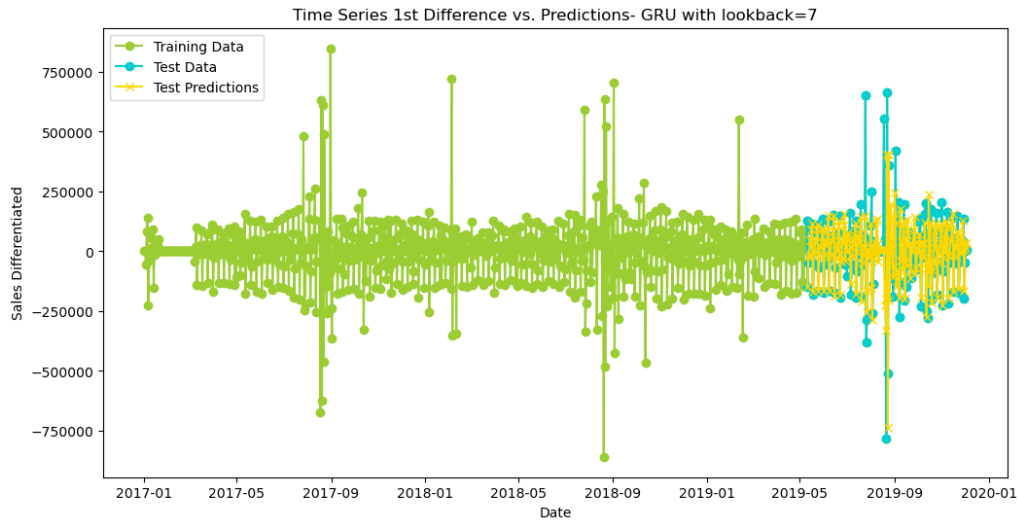


FIGURE 5.14: GRU applied to Time Series 1st Difference with lookback=7

GRUs often generate predictions that are less prone to overshooting the target values. In the context of forecasting, this means that GRU-based models may produce forecasts that are closer to the actual data points, without excessive oscillations or extreme values. This may be a factor that justifies the errors in applying this model since in our data there are some extremist values that stand out from the general score. GRUs have a simpler architecture compared to LSTMs and in prediction plots, this simplicity can manifest as smoother and more stable predictions. Let's check if the curve behaviour change with other lookback values.

GRU with Lookback = 20

Switching this GRU lookback value, the predictions are more spread out. This indicates that the GRU model is learning and generalizing differently from the LSTM model. Since there was an improvement in our forecast we can suspect that if there is any pattern in this data, it can be found in longer periods of time. So let's test our thesis with a superior look back, of 30 times.

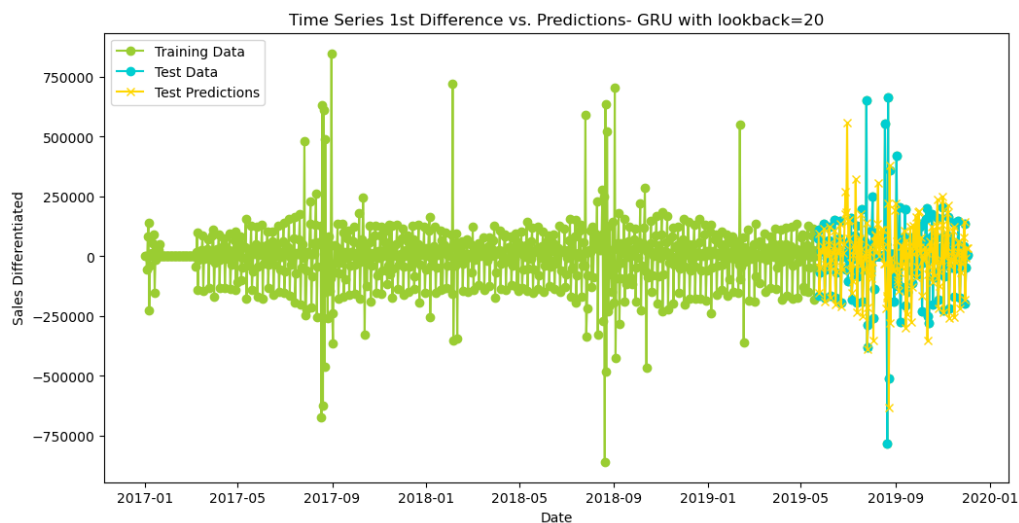


FIGURE 5.15: GRU applied to Time Series 1st Difference with lookahead=20

GRU with Lookback = 30

Comparing the graph obtained with the RNN and LSTM graphs for the same look back, the general behavior remains the same thus the error should be similar.

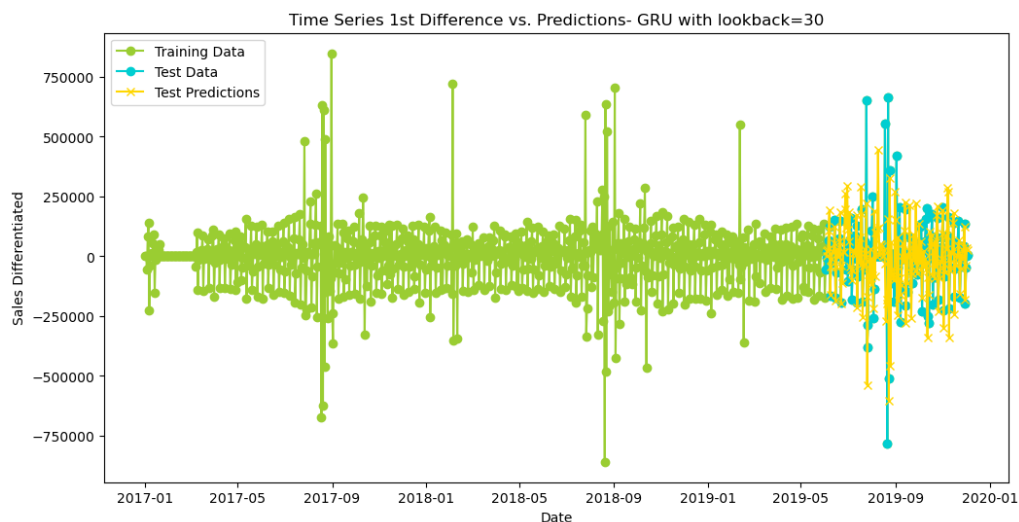


FIGURE 5.16: GRU applied to Time Series 1st Difference with lookahead=30

In light of the limited success achieved with the GRU model, we need to explore alternative approaches. XGBoost it's really good at handling tricky data patterns and finding complex relationships and it's exactly this detail that is causing the greatest obstacle to the application of previous models. Plus, with XGBoost, we can tweak parameters like learning rate, max depth, and number of estimators to fine-tune our analysis and get better predictions.

XGBoost with daily granularity

The tree-based ensemble methods like XGBoost contemplate the parameter *number of estimators* . Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks, which are neural network architectures for sequence data, do not have an equivalent parameter.

The *number of estimators* parameter in the XGBoost determines how many decision trees the XGBoost model will create. Each tree contributes to the final prediction, and the model combines the predictions of all these trees to make the final forecast. The choice of this value involves a trade-off. On one hand, having more trees can lead to a more accurate model because it can capture complex patterns and relationships in the data. However, increasing the number of trees also increases computational cost and training time. Increasing the number of trees, at the same time, can act as a form of regularization since it helps prevent over-fitting by averaging out the individual tree predictions

To find the optimal value for this parameter it has been tested the values 100,150, 1000, 1500 and the chosen one was 1000. Another key advantage is that, in practice, we can often employ a technique called "early stopping". This allows us to monitor the model's performance on a validation set during training and stop training when performance starts to degrade. This can help us find an optimal value for the number of estimators without training for too many iterations.

XGBoost with Lookback = 7

Since we are working with a large amount of data, visual analysis is not the easiest as the points seem to overlap because they are so close. However, for the higher differences (absolute value) XGBoost model tends to produce forecasts that are systematically smaller. Optimal hyperparameters have resulted in smaller forecasts so this will be revised for the next look back analysis.

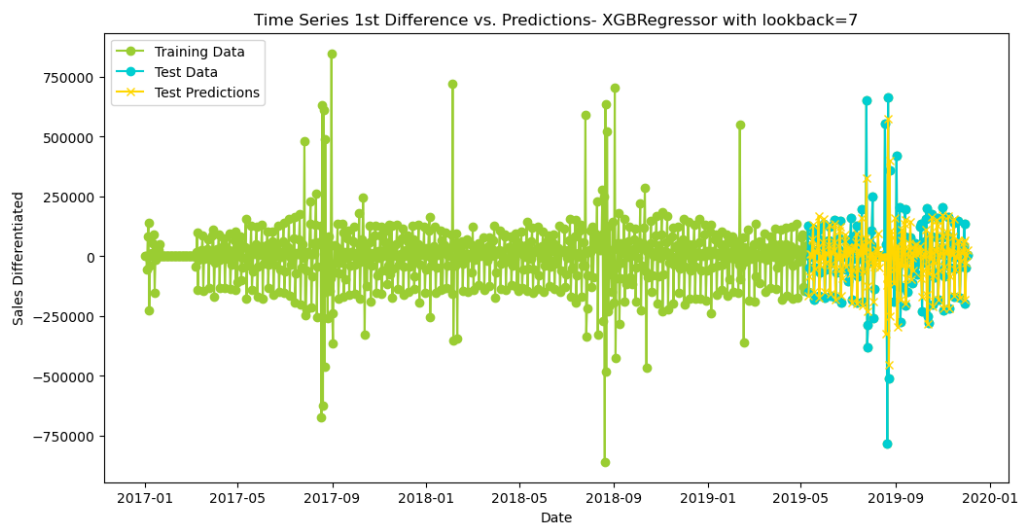


FIGURE 5.17: XGBoost applied to Time Series 1st Difference with lookahead=7

XGBoost with Lookback = 20

The prediction curve for a look back of 7 is very similar to the one with a look back of 20. It may indicate that the model is not fully benefiting from the increased number of trees therefore reducing the number of estimators might be beneficial.

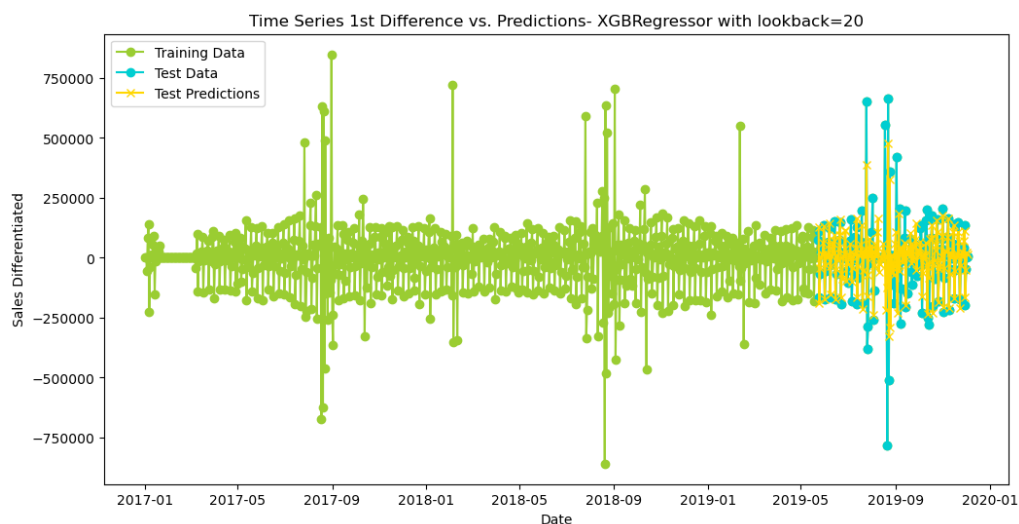


FIGURE 5.18: XGBoost applied to Time Series 1st Difference with lookahead=20

Having 1000 estimators for a dataset of 1000 points can lead to over-fitting. By decreasing the number of estimators, we can encourage the model to generalize better to unseen data and reduce the risk of fitting noise in the training data. Besides this it will be tested regularization techniques like lambda (L2 regularization) and alpha

(L1 regularization) to control model complexity and prevent over-fitting. L2 regularization penalizes large weights but doesn't force them to be exactly zero. It helps in reducing over-fitting while keeping all features. L1 regularization encourages sparsity by driving many feature weights to exactly zero, effectively performing feature selection. For this scenario, it will be used the L2 regularization. Let's analyse the results for the $\text{lookback}=30$.

XGBoost with Lookback = 30

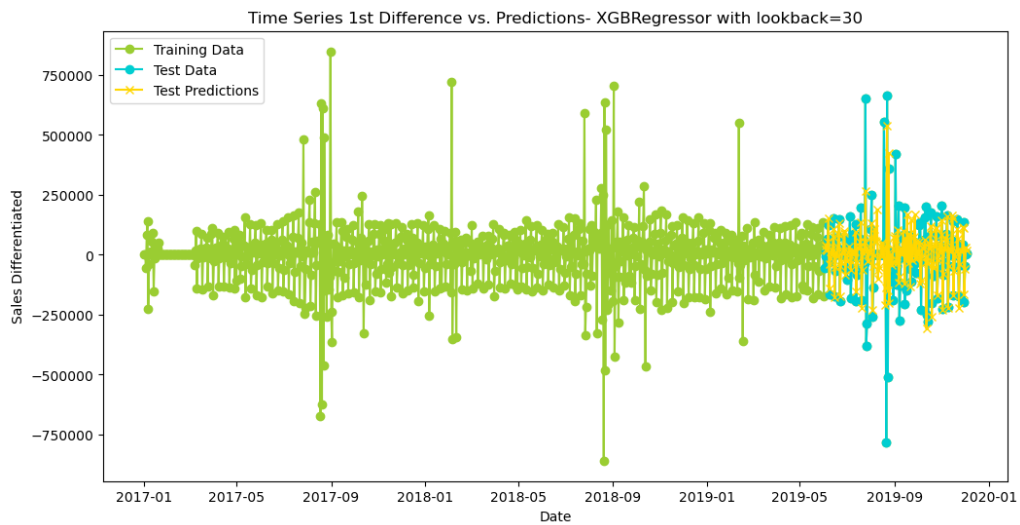


FIGURE 5.19: XGBoost applied to Time Series 1st Difference with $\text{lookback}=30$

There are no significant differences. Although, visually, the fit is still not perfect, so we will apply the next model.

LightGBM with daily granularity

LightGBM is a powerful gradient boosting framework that can be a valuable choice for forecasting problems when traditional methods like XGBoost, LSTM, RNN, and GRU have not yielded satisfactory results. LightGBM can be effective in handling forecasting tasks with specific configurations and hyperparameters tailored to this problem. The first key parameter has been already addressed in this report: number of estimators. From previous experience we realized that we were considering a value that was too high and so now as a baseline it has been defined 100 estimators, indicating that LightGBM will create 100 decision trees sequentially. Then *max depth* sets the maximum depth for the individual decision trees in the ensemble where a

higher max depth allows trees to capture more complex relationships in the data. This has been set to 15 which will provide us moderately deep trees.

There's another parameter, the *min data in leaf* which has been defined as 6. Setting a small value like 2 means that LightGBM can create leaves with very few data points, potentially capturing fine-grained patterns.

LightGBM with Lookback = 7

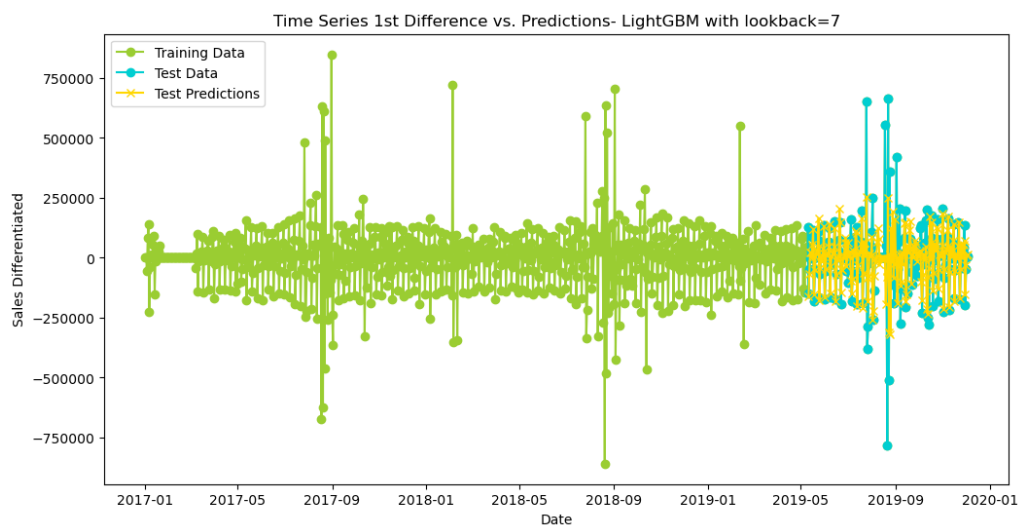


FIGURE 5.20: LightGBM applied to Time Series 1st Difference with lookback=7

Enhancing forecasting accuracy with LightGBM involves systematic tuning of hyperparameters, including *learning rate*, *max depth*, *n estimators* and regularization terms (*lambda L1* and *lambda L2*). By knowing this, we will increase the max depth for the look back=20.

The max depth parameter controls how far back in time a tree can consider information when making predictions. The look back parameter, on the other hand, determines how many past time steps (lags) are provided as input features to the model. These parameters are related since there's no point in having a long look back if the maximum depth is small. For the next 2 LightGBM applications it will be considered 35 and 50 (obtained through trials).

LightGBM with Lookback = 20

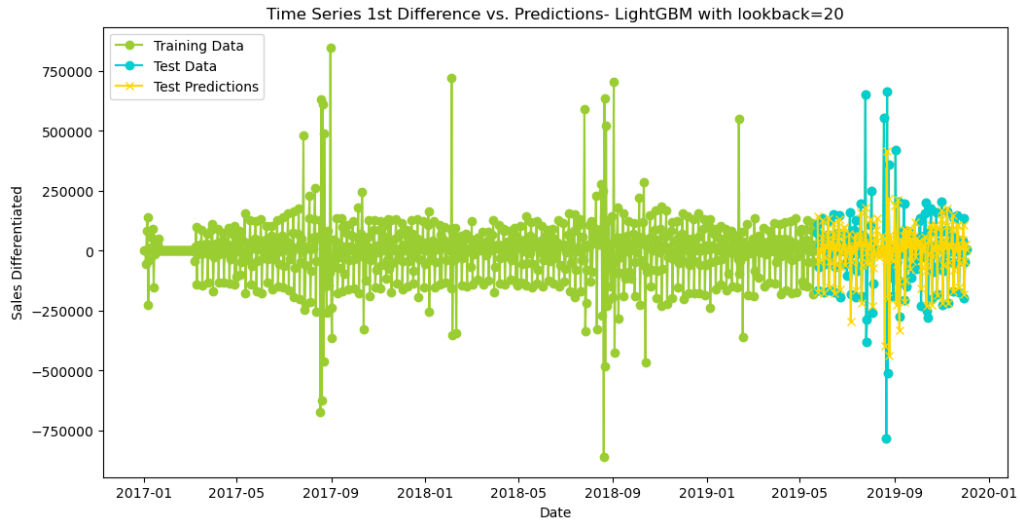


FIGURE 5.21: LightGBM applied to Time Series 1st Difference with lookback=20

LightGBM with Lookback = 30

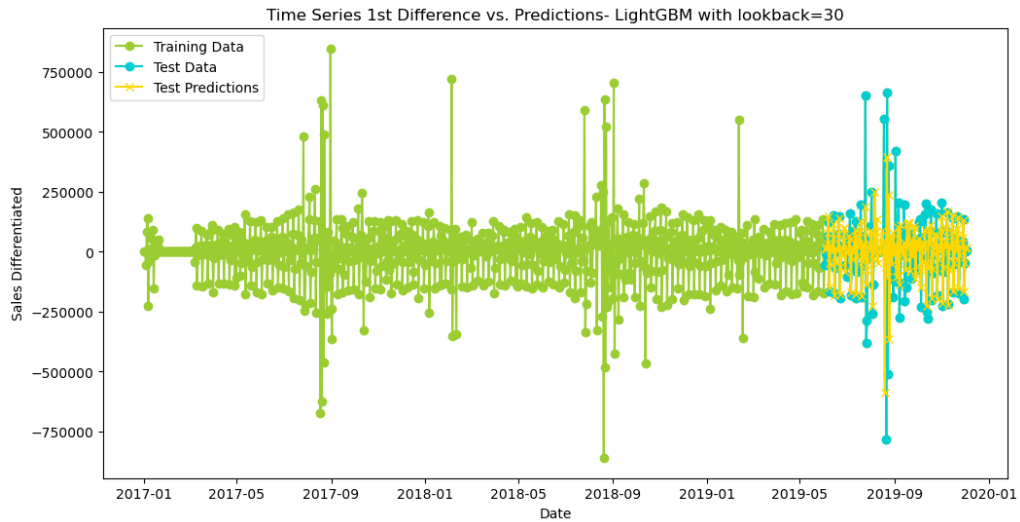


FIGURE 5.22: LightGBM applied to Time Series 1st Difference with lookback=30

The behavior of the prediction curve does not seem to change drastically. So, for a lookback of 20 we can see that a max depth of 15 or 35 produce very similar results but the max depth of 15 is computationally more efficient.

There's no major shifts comparing to the other look backs used for the LightGBM model. A fact that justifies the error in this results is the sensitivity to hyperparameters. LightGBM's performance can be highly sensitive to hyperparameter and if they are not tuned correctly for the dataset, it can result in suboptimal performance.

Since the way these optimal parameters are being defined is through trial and error analysis, it's very easy for them to not be the most correct.

Multi-Layer Perceptron (MLP) with daily granularity

Previously we talked about the definition of epoch. In this model, this is also one of the parameters that we set at 300. Choosing an appropriate number of epochs is crucial. Too few epochs may result in underfitting, where the model hasn't learned enough from the data. Too many epochs may lead to over-fitting, where the model starts memorizing the training data instead of generalizing from it. Proper tuning is essential to strike the right balance. Other repeated parameter is the batch size which we set as 256.

For this model is also important to define the learning rate: this controls the step size during gradient descent optimization. It determines how much the model's weights are updated in each iteration and will work with $lr=0.0003$. Besides this, another time, it has been choosed the optimizer *adam*.

MLP with Lookback = 7

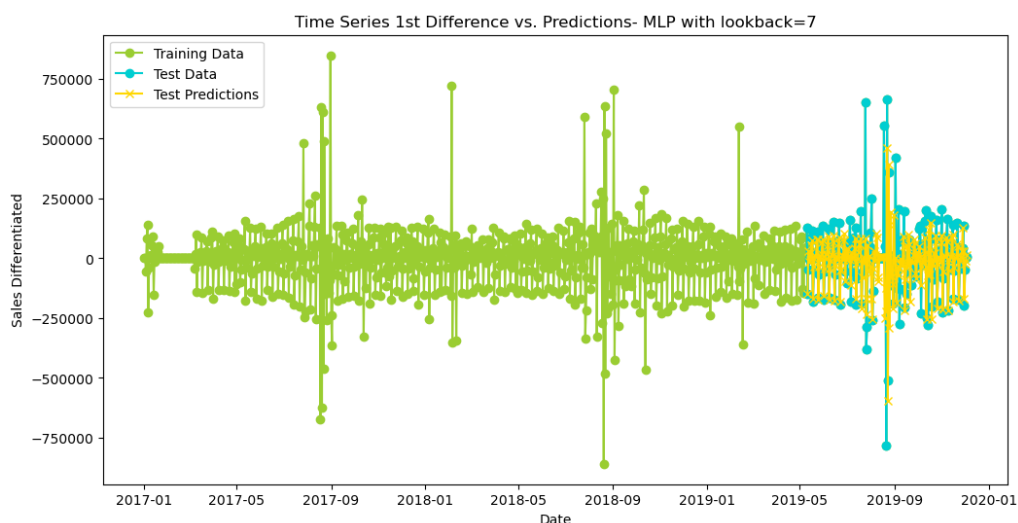


FIGURE 5.23: MLP applied to Time Series 1st Difference with lookback=7

Visually, the results obtained are very similar to the GRU results (for the same look back). The predictions continue to present an error greater than the desired one. It's possible that both models are underfitting the data. Underfitting occurs when a model is too simple to capture the underlying patterns and variations in the data.

MLP with Lookback = 20

With a longer lookback period, the model can capture longer-term trends and patterns in the data that might not be evident with a shorter lookback. It has a better chance of identifying seasonality, cyclical patterns or other long-term dependencies in the time series data.

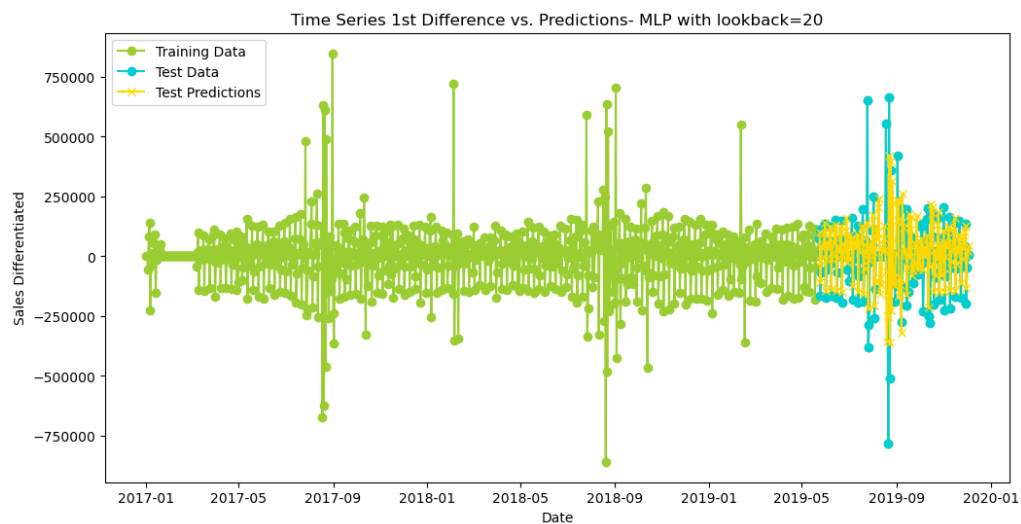


FIGURE 5.24: MLP applied to Time Series 1st Difference with lookback=20

MLP with Lookback = 30

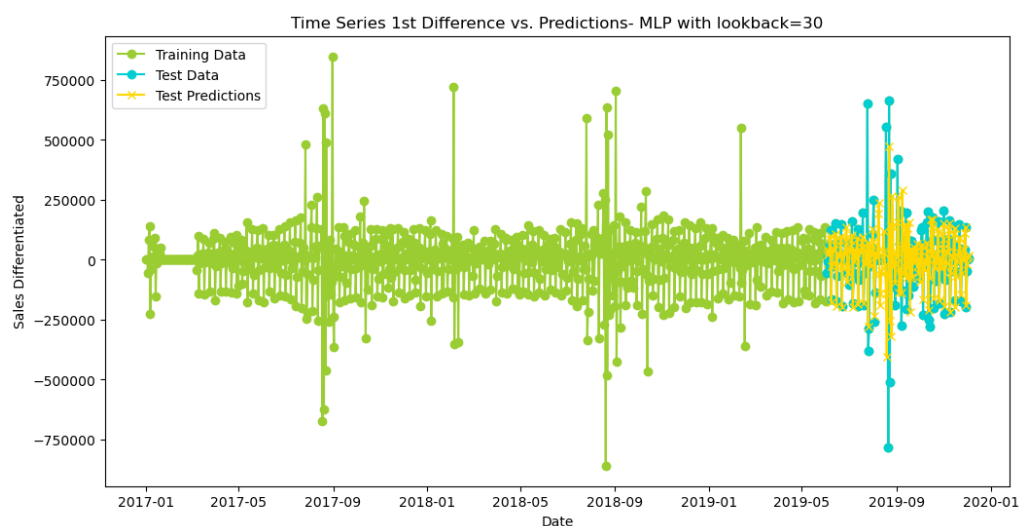


FIGURE 5.25: MLP applied to Time Series 1st Difference with lookback=30

Support Vector Machine (SVM) with daily granularity

In the construction of this Support Vector Machine (SVM) model, the selection of appropriate hyperparameters played a pivotal role in achieving accurate and robust predictions. In this context, we have opted for the following SVM parameters, specifically kernel=rbf, C=1, and epsilon=0.1.

The choice of the radial basis function (RBF) kernel is motivated by its ability to capture complex nonlinear relationships within the data. Given the intricate nature of our forecasting problem, an RBF kernel is well-suited to uncover underlying patterns that may not be linear or easily separable.

The regularization parameter C controls the trade-off between maximizing the margin and minimizing the classification error. By setting C=1, we strike a balance between encouraging a wide margin and allowing some margin violations. This choice reflects our intention to avoid over-fitting while ensuring that the SVM captures important trends in the data.

SVM with Lookback = 7

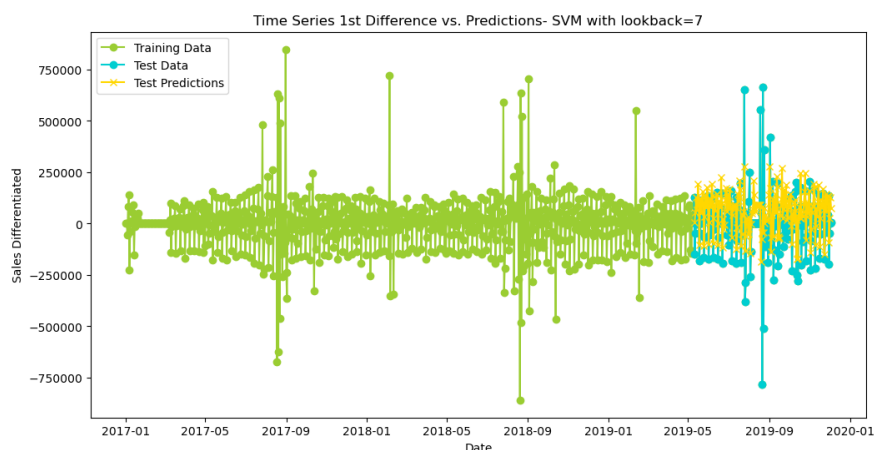


FIGURE 5.26: SVM applied to Time Series 1st Difference with lookback=7

Analyzing the graph, this model does not seem to be able to predict the peaks of the series and for the remaining points it always seems to be giving a higher prediction than the expected one. This is, so far, the model that presents the poorest performance.

SVM with Lookback = 20

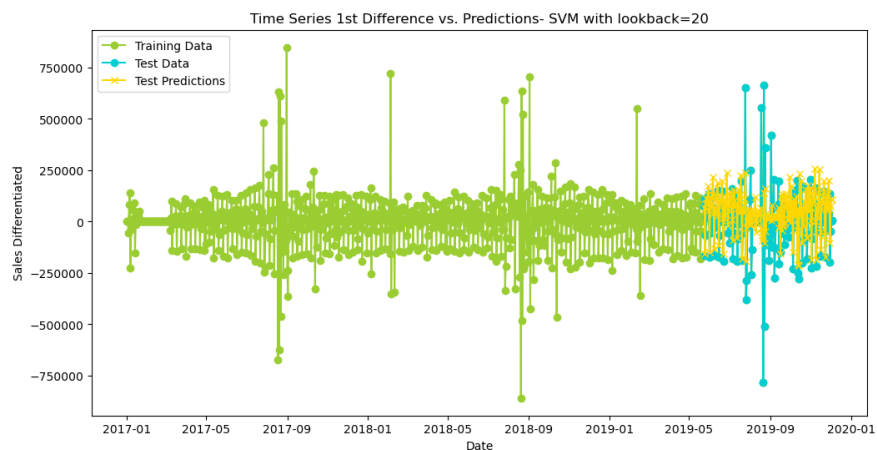


FIGURE 5.27: SVM applied to Time Series 1st Difference with lookback=20

Once again, this model application does not seem to be able to capture the most outlier values.

SVM with Lookback = 30

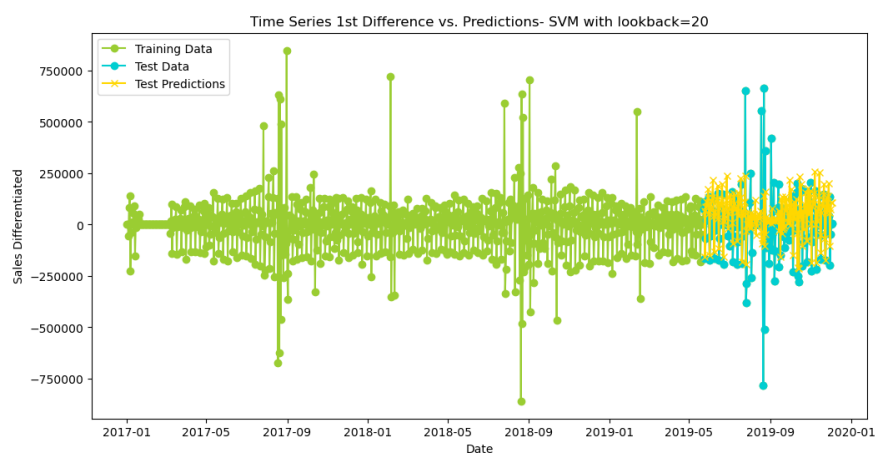


FIGURE 5.28: SVM applied to Time Series 1st Difference with lookback=20

Once again the model does not predict correctly for larger differences. What we can conclude is that for this model the increase in lookback did not lead to an improvement in results.

Random Forest with daily granularity

Random Forest is an ensemble learning technique that combines multiple decision trees to create a robust and accurate model. Several parameters like *n estimators* or

max depth govern how a Random Forest model behaves but all of them have been already talked here.

Random Forest with Lookback = 7

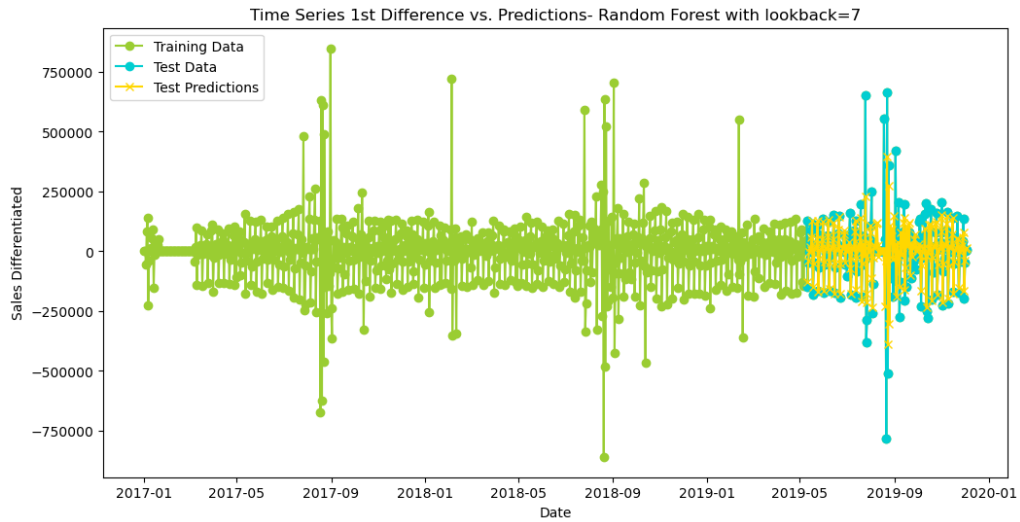


FIGURE 5.29: Random Forest applied to Time Series 1st Difference with lookback=7

This model revealed significant improvements compared to the previous model, SVM. Comparing with others studied so far, although the forecast for the peaks continues to be not perfect, the forecast reasonably follows the expected values.

Random Forest with Lookback = 20

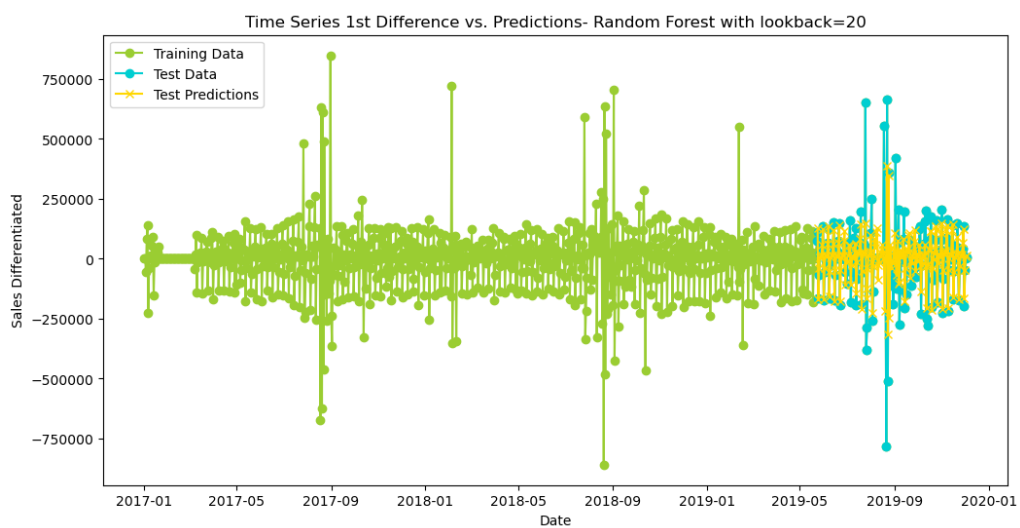


FIGURE 5.30: Random Forest applied to Time Series 1st Difference with lookback=10

Random Forest with Lookback = 30

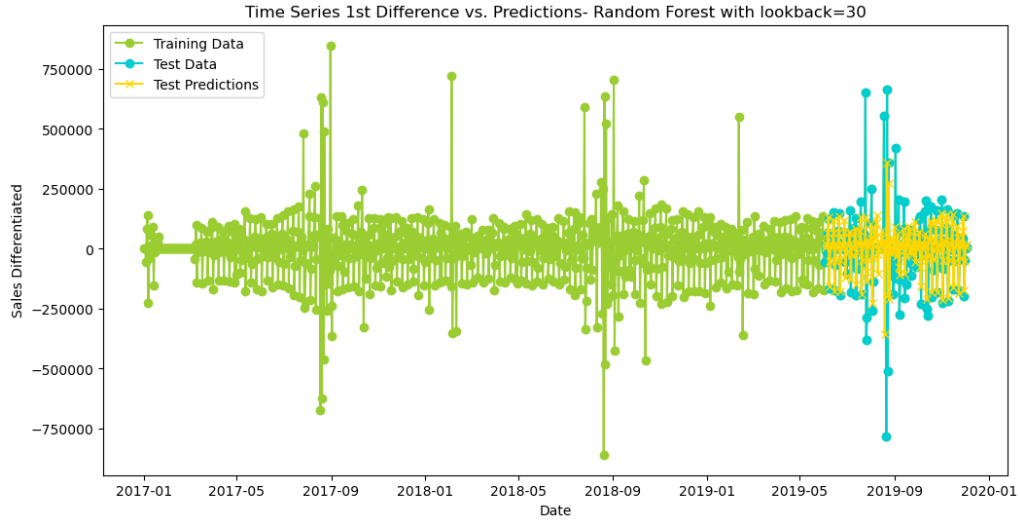


FIGURE 5.31: Random Forest applied to Time Series 1st Difference with lookback=30

This model presents more positive values than SVM, especially for the highest peaks in the series, however it is still not a perfect approximation.

5.4 Data preparation and processing for weekly granularity

In the realm of sales forecasting, the granularity of data can significantly impact the quality of insights and predictions. Daily granularity provides a detailed view of sales patterns but as seen above it often presents challenges due to the sheer volume of data points.

The daily sales data, when plotted over time, was overwhelming. The multitude of data points creates intricate plots that may obscure underlying trends and patterns. Aggregating sales data into weekly intervals simplifies the visual representation, making it easier to identify overarching patterns and variations. The series corresponding to the weekly average of sales is useful for obtaining a consolidated and smoothed view of sales performance over time since it eliminates noise and daily fluctuations.

Since the errors for daily data are not comfortable and in order to explore other data groupings, these models will be applied to weekly sales values. With this change, now our dataset has 154 data points, from which 123 are training set and the remaining points belong to x_{test} .

Now, the goal of the following analysis is to assess the impact of granularity on forecasting accuracy. But first, since we are working with a “new” time series, all the data processing should be repeated.

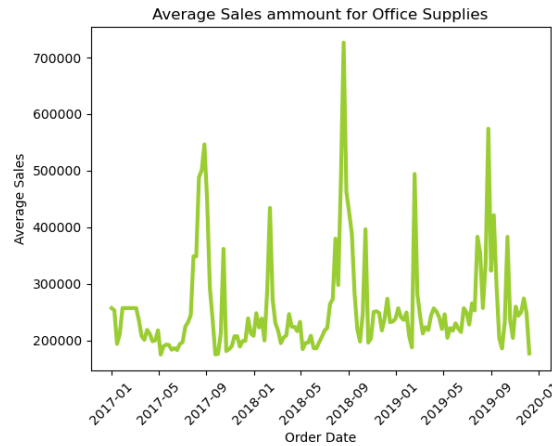


FIGURE 5.32: Time Series Data- Aggregated by week

This reduction in data complexity enhances the interpretability of the analysis. Besides this, daily sales often exhibit short-term fluctuations, driven by various factors like weekday-specific consumer behaviors or ad-hoc promotions. Weekly aggregation helps smooth out these fluctuations.

Analysing the previous graph 5.32, it’s possible to see that September 2018 proved to be a good season since there were weeks when sales on average exceeded 700000\$. About trend and seasonality, with a naked eye it doesn’t seem to have but the idea for now it to prove it with analytical methods.

Let’s start by decomposing our time series and exploring the seasonality and trend.

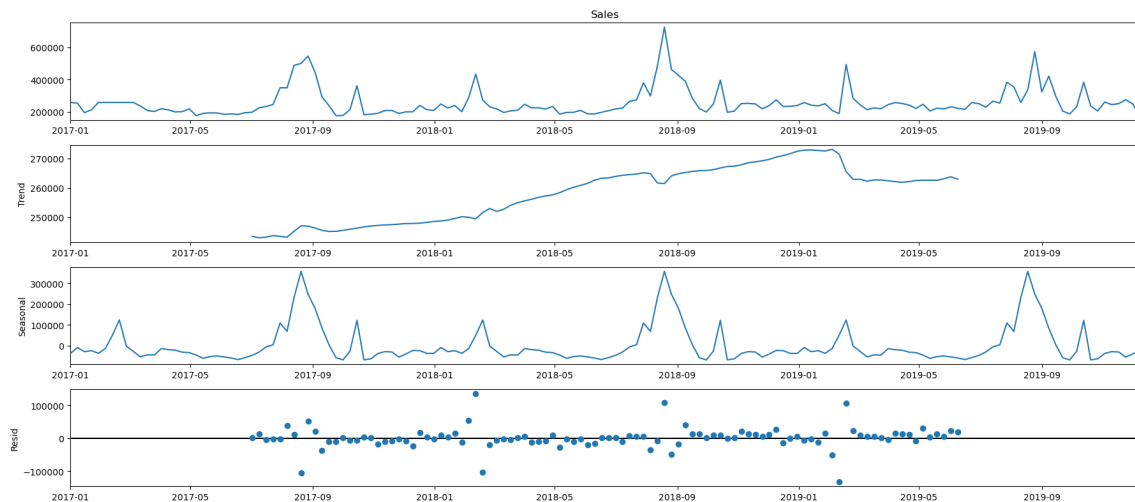


FIGURE 5.33: Time Series Decomposition- Aggregated by week

By analysing the decomposition graph, unlike the original series, there's a growing trend which indicates that over the weeks, there has been a systematic increase in the sales averages which suggests a steady growth in sales performance over the considered period. However from the beginning of 2019 there seems to be a behavioral change. To find out if we can consider a trend in this series we have to carry out some tests.

About seasonality, the observation of a fixed seasonal pattern implies that there are regular fluctuations in sales that occur at consistent intervals, regardless of the growing trend. These seasonal patterns may be related to factors such as days of the week, seasons of the year or recurring events. Identifying and modeling these seasonalities is crucial for accurate forecasts.

The visualization of residuals close to zero suggests that after removing the trend and seasonality, there are no clear and systematic patterns in the remaining data fluctuations. This is a positive indication as residuals close to zero typically reflect an appropriate decomposition model, where the unexplained variation not accounted for the trend and seasonality is random.

The fact that the original series did not exhibit a growing trend but such a trend emerged in the series after aggregation on a weekly basis can be explained by how the data was grouped. Weekly aggregation may reveal trends that were not evident at the daily granularity. This underscores the importance of selecting the appropriate data granularity for analysis.

Let's jump to stationarity check by applying ADF test and KPSS test.

Test Statistic	p-value
1.335	0.080

TABLE 5.5: Augmented Dickey-Fuller Test Results

Test Statistic	p-value
0.109	0.083

TABLE 5.6: KPSS Test Results

The ADF P-value is 0.080 , which is greater than 0.05. Therefore the null hypothesis that the series is non-stationary should be accepted. In other words, the data is not stationary.

The KPSS p-value is 0.083. Since 0.083 is more than 0.05 than we should refuse the null hypothesis.

The ADF test says that the series is non-stationary and the KPSS test in turn says that it is stationary. In this situations the trend needs to be removed and then the detrended series is checked for stationarity. This can be done in multiple ways but one particularly interesting is the log transformation: since the trend is exponential in nature, taking the natural logarithm (log) of the data can linearize the trend.

Important to note that this log transformation is only applicable to this time series because it only has positive values. We cannot take the log of zero or negative numbers. If, hypothetically, this data included zero values (weeks with 0\$ sales) or can potentially be negative (sales ≥ 0), these cases needed to be handled separately, such as adding a constant to all values to ensure they are positive.

Now that our series has undergone the logarithmic transformation, let's recalculate the rolling statistics and apply the statistical tests to see if they are already in agreement.

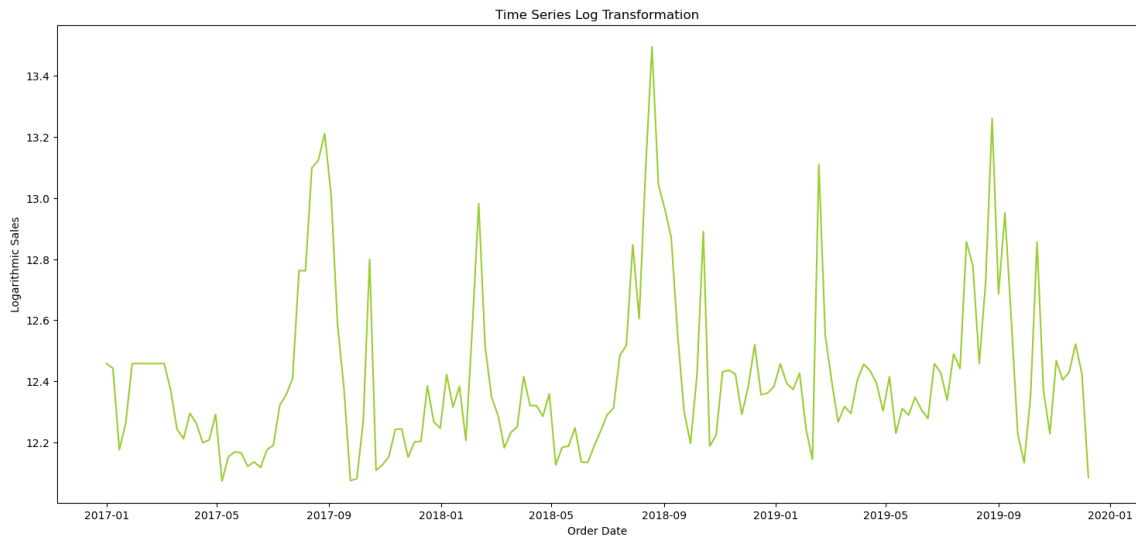


FIGURE 5.34: Logarithmic Time Series- Aggregated by week

Test Statistic	p-value
-5.335	5.000×10^{-6}

TABLE 5.7: Augmented Dickey-Fuller Test Results

Test Statistic	p-value
1.135	0.100

TABLE 5.8: KPSS Test Results

The highly negative ADF statistic of -5.335 suggests that the time series is unlikely to have a unit root and therefore provides evidence of stationarity. The p-value indicates that the null hypothesis of non-stationarity can be rejected.

The KPSS statistic of 1.135 is slightly above 1, suggesting that the data may deviate somewhat from strict stationarity. Although, the p-value of 0.100 indicates that there may be some evidence against strict stationarity around a deterministic trend.

To conclude this stationarity analysis, our time series transformed is already stationary so we can follow with it. After applying this log transformation, one important consideration is that the following models will predict outcomes in a logarithmic scale. To interpret and use these predictions effectively, we need to reverse the log transformation, a process known as back-transformation. This involves taking the antilogarithm (exponential) of the predictions to return them to their original scale.

To normalize our data, once again, it will be used *MinMaxScaler()* Python function.

2nd Investigation Resume

Scope: Logarithmic Time Series aggregated by week

Normalization Technique: MinMaxScaler

Split Technique: Fixed Split (80% for the train set and the remaining 20% to the test)

Lookbacks considered: 1, 3

Models Applied: RNN, LSTM, GRU, XGBoost, LightGBM, MLP, SVM, Random Forests

Since we have 154 weeks we will split them so we use 123 of them to train our model and the remaining 31 for testing. This reduction has led to a smaller training dataset. Therefore we have less historical data available to train the model. This means that when choosing a higher value for lookback, that might be including a disproportionately large amount of historical data relative to the size of the training set. This can lead to over-fitting, where the model fits too closely to the training data but doesn't generalize well to new data.

To avoid this scenario we will use only the safe value of look back=3. Relevant to notice that this length of 3 doesn't mean an historical period of 3 days but, instead, this means that we will be looking for the previous 3 weeks to predict the next week.

It will be presented both visualizations of the logarithmic data and after the back-transformation.

5.5 Stock Forecast using Machine Learning for weekly granularity

Recurrent Neural Network (RNN) with weekly granularity

To work with weekly data there are many parameters in common to those explained in the previous section for the daily data. Thus, it will be discussed only the settings which differ from those used for daily granularity series forecast.

For this model it has been chosen a lookback of 1 meaning that for each prediction, the model only considers the immediate past data point (the previous week) as its historical context. This setting implies that the model relies heavily on the most

recent observation to make predictions and doesn't take into account any longer-term patterns or dependencies in the data.

The sequence length of 3 determines how many consecutive time steps are used to create each input sequence. In this configuration, each input sequence consists of three consecutive weeks, where the target (output) is the fourth week. This setup allows the model to consider a short-term trend over a three-week window but still maintains a focus on very recent data.

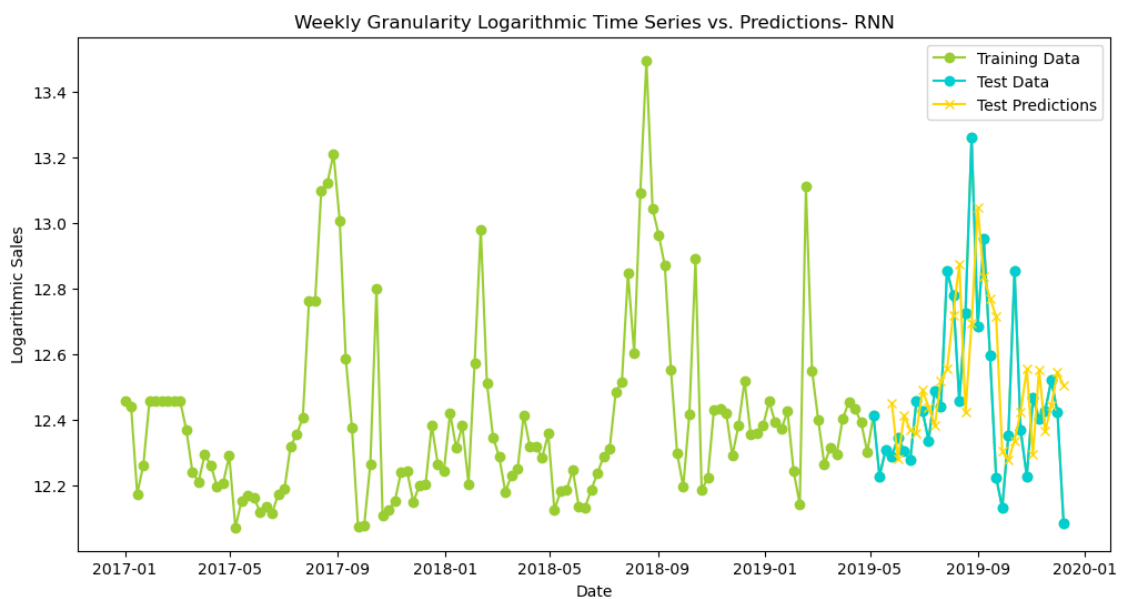


FIGURE 5.35: Logarithmic Time Series - Aggregated by week

The predicted values have lower amplitude which suggests that this setting (look-back=1, sequence-length=3) may be suitable for capturing immediate changes and fluctuations in the data but may not effectively capture more extended patterns or seasonality that could be present in weekly sales data.

Working with logarithmic sales values can be beneficial for various reasons, such as stabilizing variance or making data more suitable for modeling. However, when it comes to presenting or interpreting the results, it's often necessary to convert these logarithmic values back to the original scale for readability and practical understanding (the transformation graph is only shown for understanding)

Converting logarithmic values to their original scale is crucial for comparing the model's predictions with historical or actual sales data. For meaningful comparisons, both the predictions and the historical data should be in the same units, which means converting predictions from the logarithmic scale to the original scale.

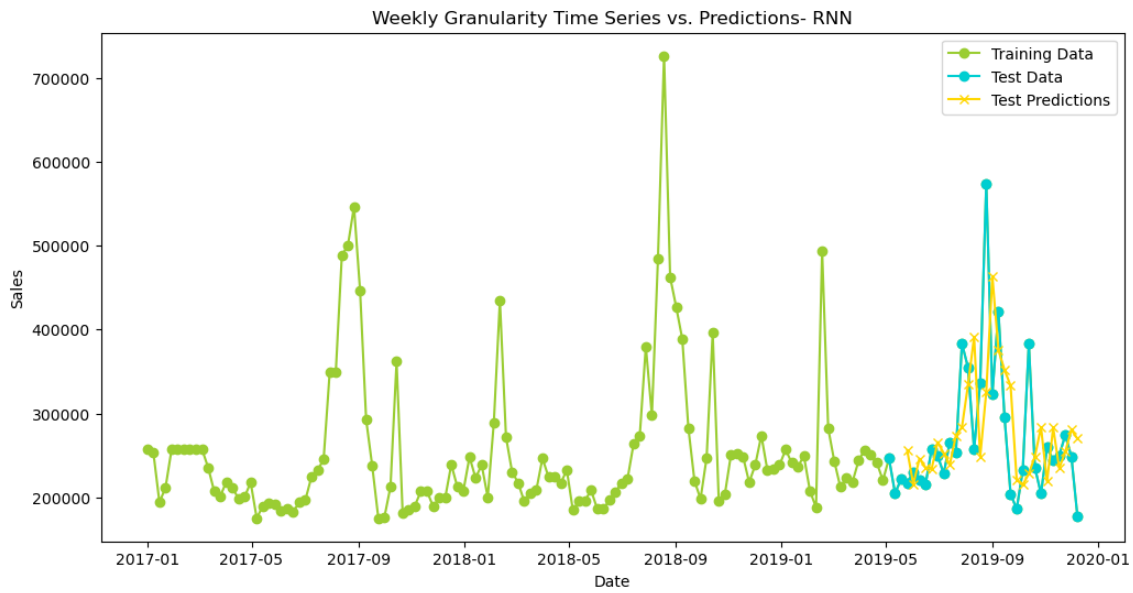


FIGURE 5.36: Time Series Decomposition- Aggregated by week

To convert logarithmic sales values back to the original scale, it was used the inverse of the logarithm function that was originally applied: the exponential function for natural logarithms.

This RNN model is a little bit different from the others since we define that parameter sequence length. Because of this, the graph presents a particularity that with a higher granularity, despite also happening, ends up going unnoticed: the first three data points from x_{test} doesn't present any prediction.

The absence of predictions for the first three data points in x_{test} can be explained by the way the sequences are created and used for forecasting.

So, the input sequences in x_{test} are constructed from the beginning of the x_{test} data, starting from index 0. For the first three data points, there aren't enough previous data points to create a complete input sequence of three weeks. In other words, the lookback of 1 restricts the model from considering any data earlier than the immediately preceding week for making predictions. Consequently, there are no input sequences available to the model for the first three data point and as a result, there are no predictions associated with them. Therefore, predictions start appearing after the first three data points, where sufficient historical context is available for forecasting.

Long Short-Term Memory (LSTM) with weekly granularity

For this LSTM model application, it was choosed a lookback of 3 which means that the LSTM model takes the last three consecutive time steps as input to predict the next time step.

During testing, the model processes each input sequence from x_{test} one by one. For each input sequence of 3 consecutive time steps, the model predicts the next time step, which is the one immediately following the last time step in the sequence. With a lookback of 3, the model needs three previous time steps to predict the next one. When the model is being applied to the test data x_{test} , it starts processing input sequences from the beginning of the test data, moving one time step at a time. For the very first time step in x_{test} (the time step immediately following the training data), there are no previous time steps within the test data itself to serve as a historical context.

As a result, the model cannot make a prediction for the first time steps in x_{test} .

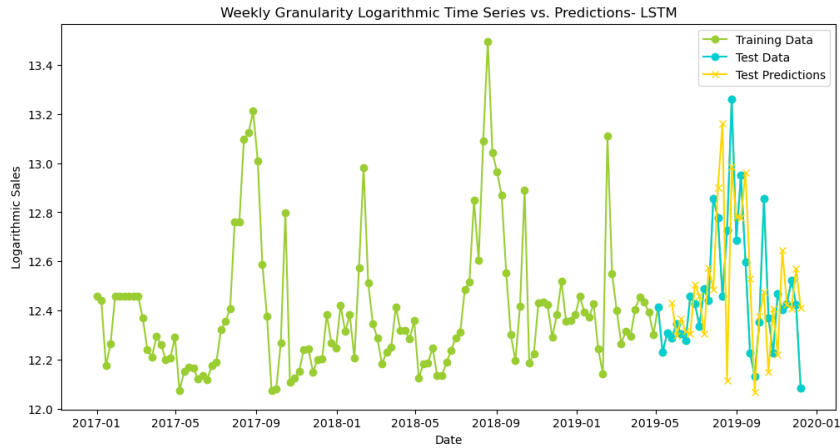


FIGURE 5.37: Logarithmic Time Series Decomposition- Aggregated by week

For the logarithmic sales predicted, this model, compared with RNN, seems to be learning less correctly since the amplitude of the predictions is larger and deviated from the supposed.

Logarithmic graphs have no relevance from a commercial perspective, however we can see that the application of exponential will increase the scale of our data and consequently, the error will increase considerably.

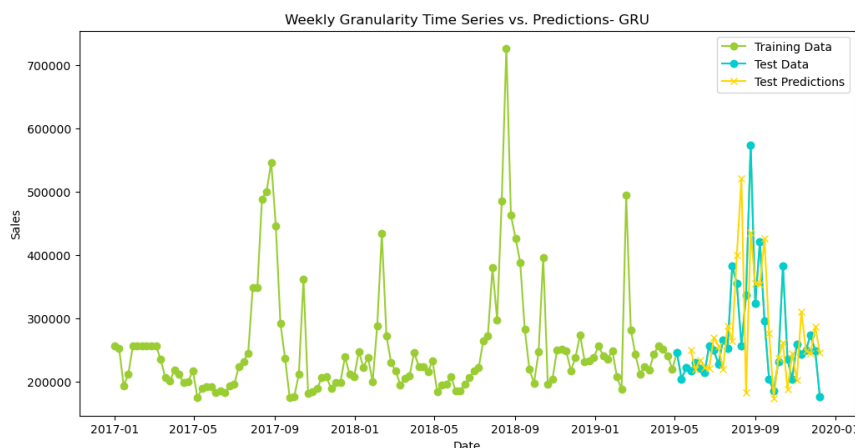


FIGURE 5.38: Time Series Decomposition- Aggregated by week

The prediction curve doesn't follow strictly the behaviour from y_{test} . LSTM models, while powerful for capturing sequential patterns, may struggle when the historical context is limited. If the model's lookback window is too short, it might not capture the longer-term patterns or seasonality in the data, leading to discrepancies between predictions and actual values.

To manage the amplitude of predictions curve we may experiment other parameters.

Gated Recurrent Unit (GRU) with weekly granularity

For GRU application it has been used the same parameters as those used for daily granularity data, with the exception of the lookback. We cannot no longer use higher lookbacks since our training set is significantly smaller.

Weekly data inherently has less granularity compared to daily data. Weekly patterns and trends might be less intricate and more predictable, reducing the potential advantage of using a more complex model like GRU over LSTM. With a weekly lookback of 3 (predicting based on the previous 3 week's data), both LSTM and GRU have the capacity to capture similar patterns and dependencies. The advantage of GRU in handling longer-term dependencies may not be fully leveraged in this scenario.

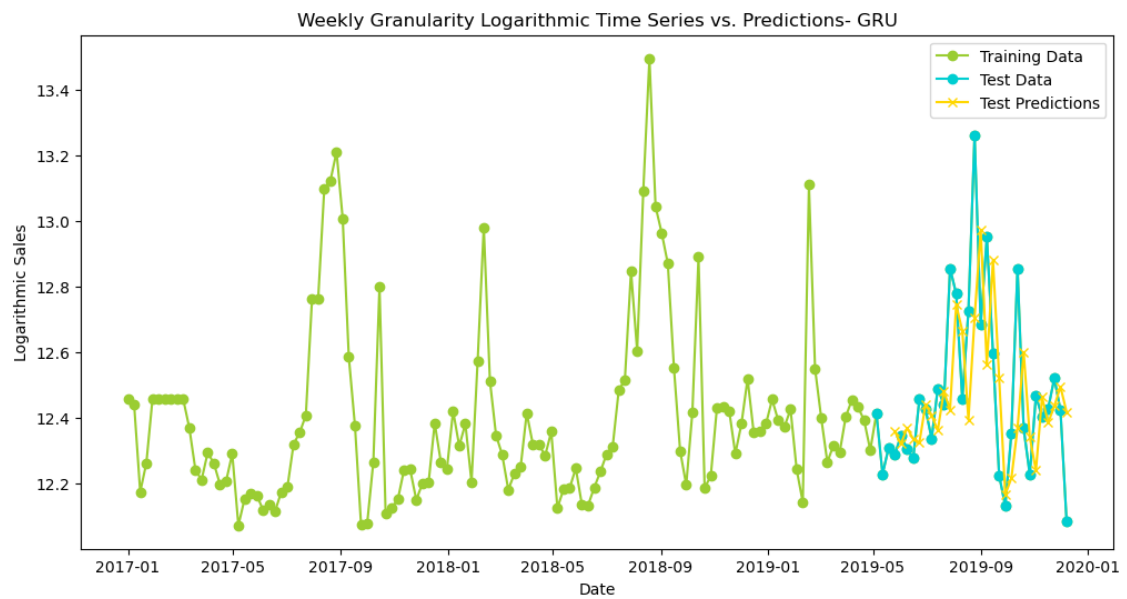


FIGURE 5.39: Logarithmic Time Series Decomposition- Aggregated by week

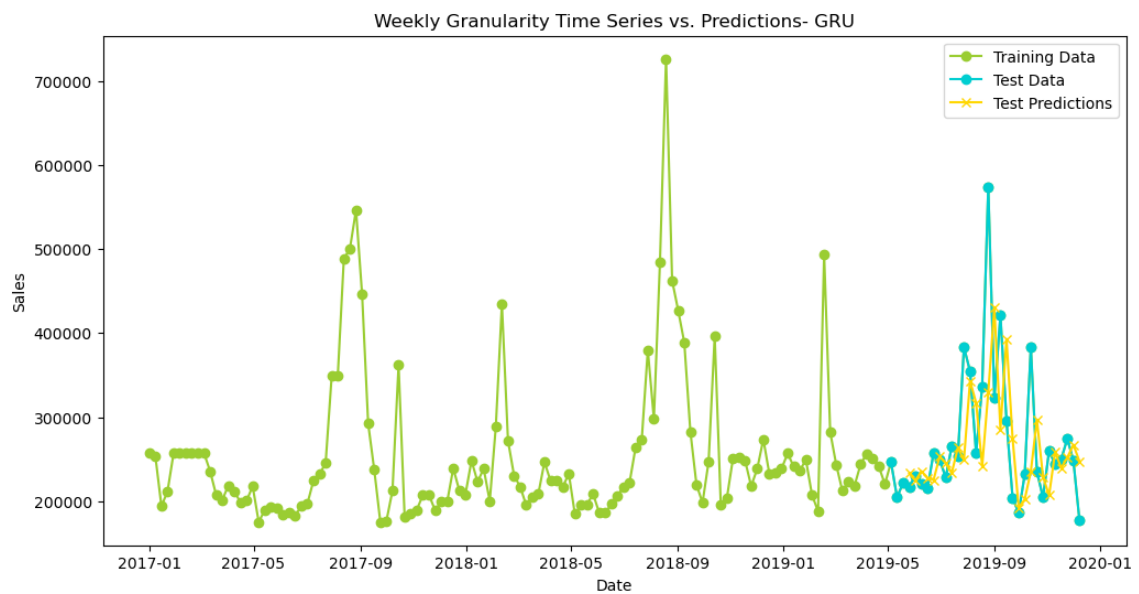


FIGURE 5.40: Time Series Decomposition- Aggregated by week

On both scales the predictions seem to be very close to the real values, which proves the effectiveness of this model for our data.

XGBoost with weekly granularity

XGBoost (Extreme Gradient Boosting) often produces predictions with a higher amplitude compared to some other models like LSTM or GRU. Let's see what are the results for weekly granularity.

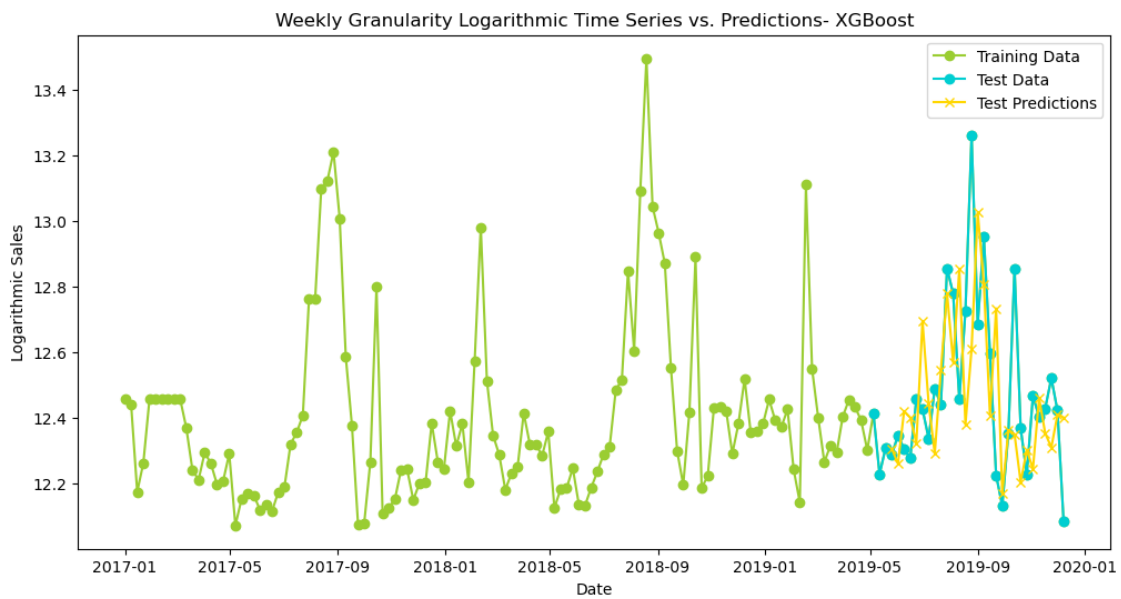


FIGURE 5.41: Logarithmic Time Series Decomposition- Aggregated by week

Comparing to the previous models the general behaviour of predicted values are very similar.

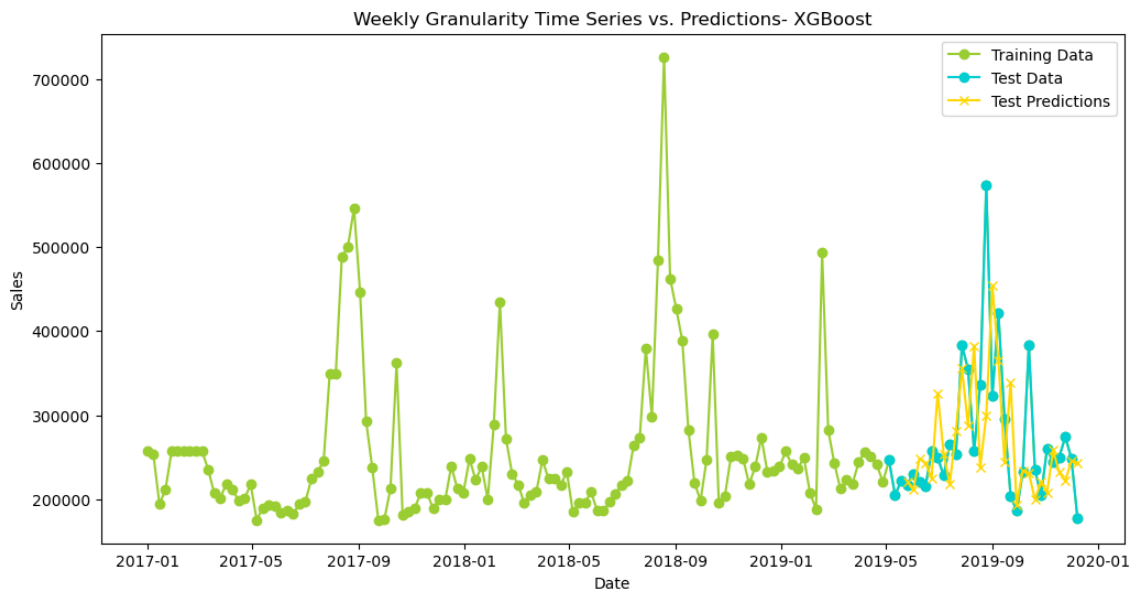


FIGURE 5.42: Time Series Decomposition- Aggregated by week

LightGBM with weekly granularity

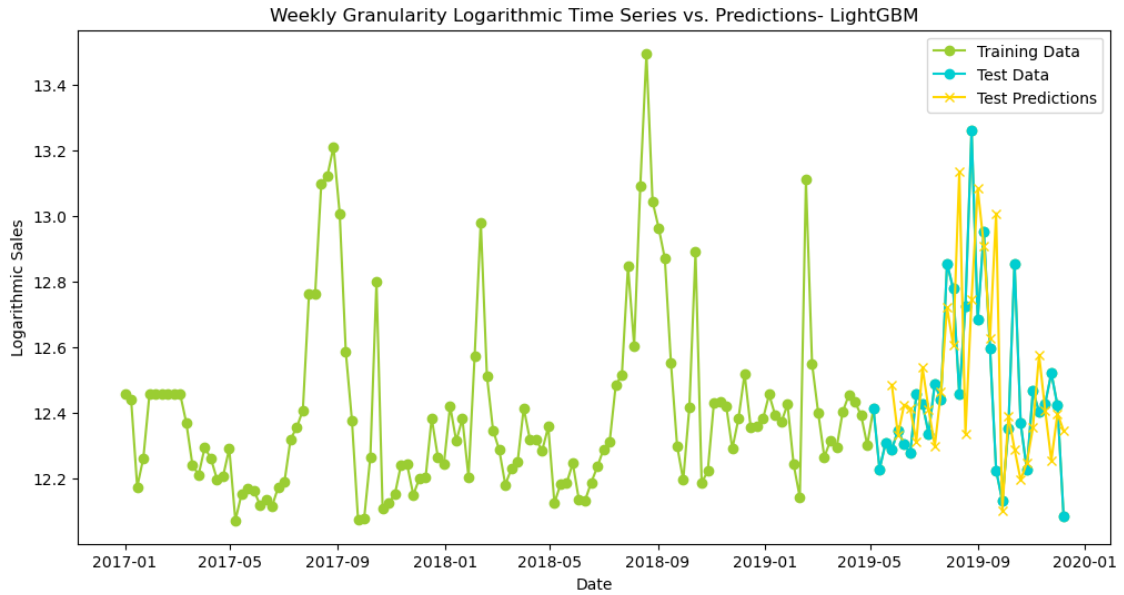


FIGURE 5.43: Time Series Decomposition- Aggregated by week

LightGBM and XGBoost both offer a range of hyperparameters for controlling model complexity, regularization, and boosting-specific parameters. These were configured in a similar way so that the results could be compared but this uniformity may lead to minor differences in the application of models.

While LightGBM and XGBoost share similarities and provide similar results, when choosing, we must consider other unique features such as training speed and memory efficiency. Additionally, the choice of hyperparameters and feature engineering can have a significant impact on model behavior, so fine-tuning and experimentation may lead to different outcomes.

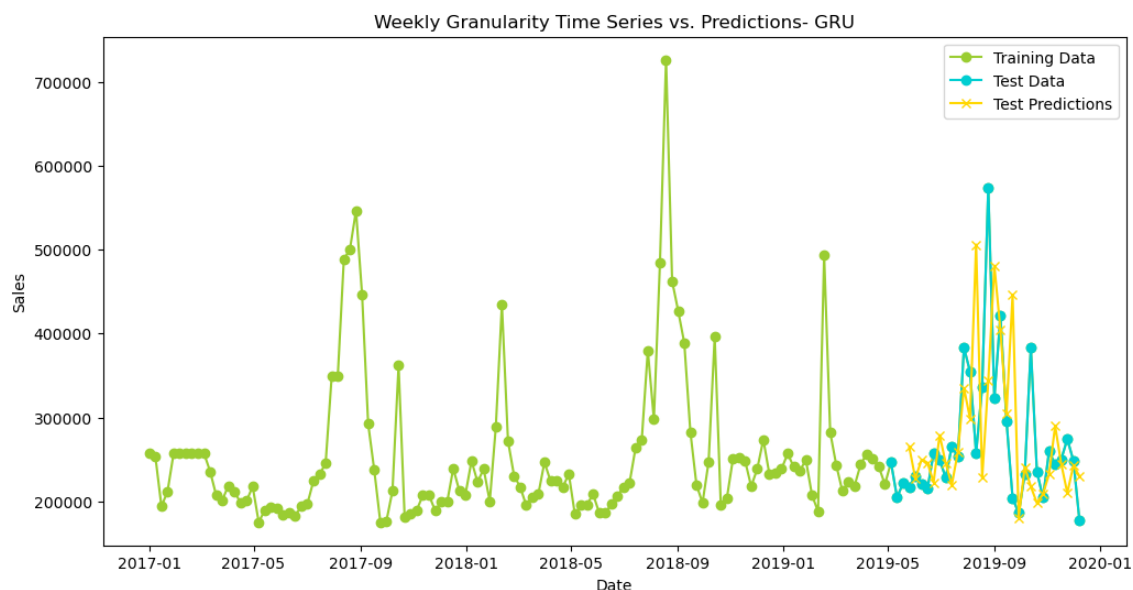


FIGURE 5.44: Time Series Decomposition- Aggregated by week

Multi-Layer Perceptron (MLP) with weekly granularity

This Multilayer Perceptron (MLP) neural network requires special attention since it produces forecasts that deviate significantly from the expected values. The predictions do not align with the observed data but several factors could be contributing to this poor performance. The first suspicion focused on the choice of hyperparameters (like the learning rate, batch size, activation functions, and the number of hidden layers/neurons). Poorly tuned hyperparameters can lead to unstable forecasts but several combinations have been tested and all provided similar results. Other reason may be the model complexity: an MLP with too many layers or neurons can overfit the training data, effectively memorizing it rather than learning meaningful patterns. This can lead to poor generalization to unseen data and erratic forecasts.

Besides this we already noticed that this dataset is noisy and lacks relevant features, therefore MLP may struggle to capture meaningful relationships in the data and it can be learning the noise instead of the underlying patterns. Another possibility are the MLP weights. These are typically initialized randomly and thus they can affect convergence.

To improve this MLP forecast maybe the right way is experiment even more different hyperparameters until we achieve minimum quality in our forecasted data.

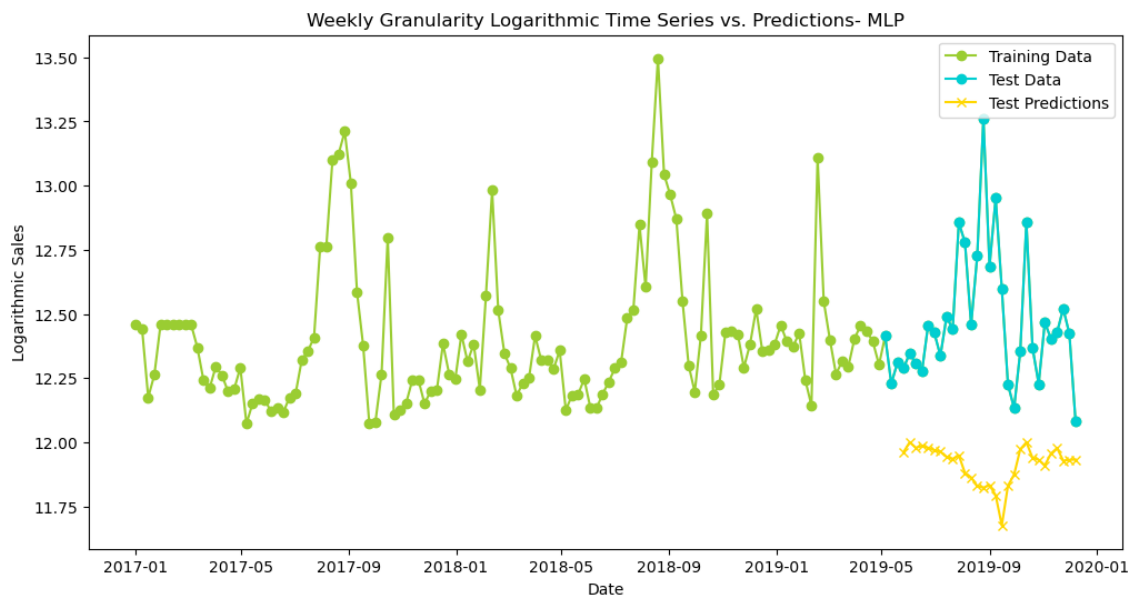


FIGURE 5.45: Time Series Decomposition- Aggregated by week

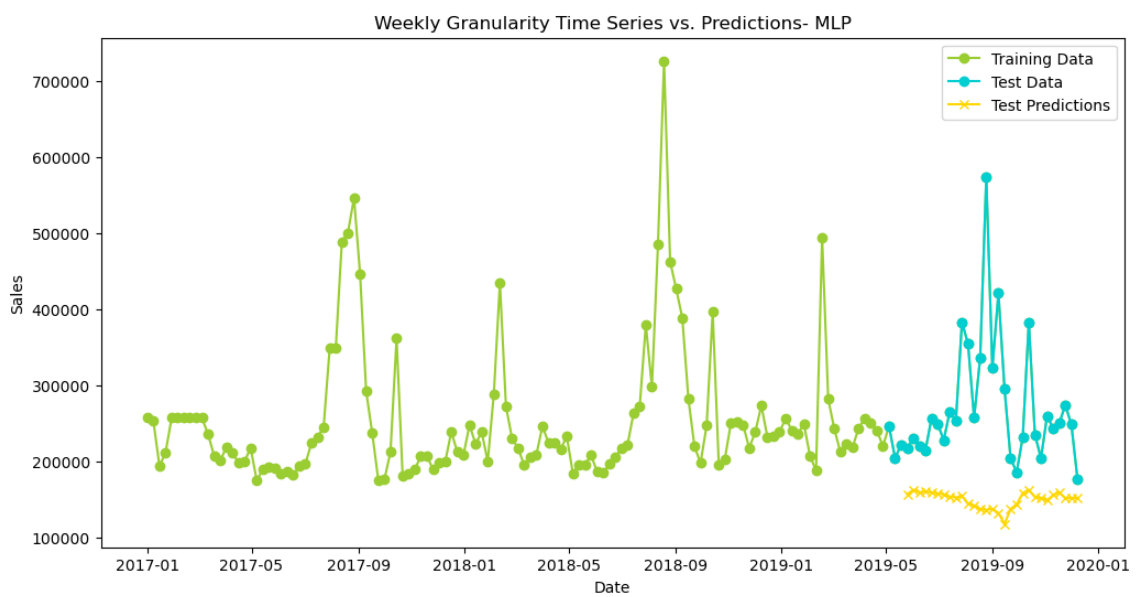


FIGURE 5.46: Time Series Decomposition- Aggregated by week

Support Vector Machine (SVM) with weekly granularity

All the parameters used were the same for the daily granularity. The curve shows an improved fit when comparing with the MLP.

One hypothesis to improve the results was tuning hyperparameters like regularization parameter (C), and kernel-specific parameters using cross-validation.

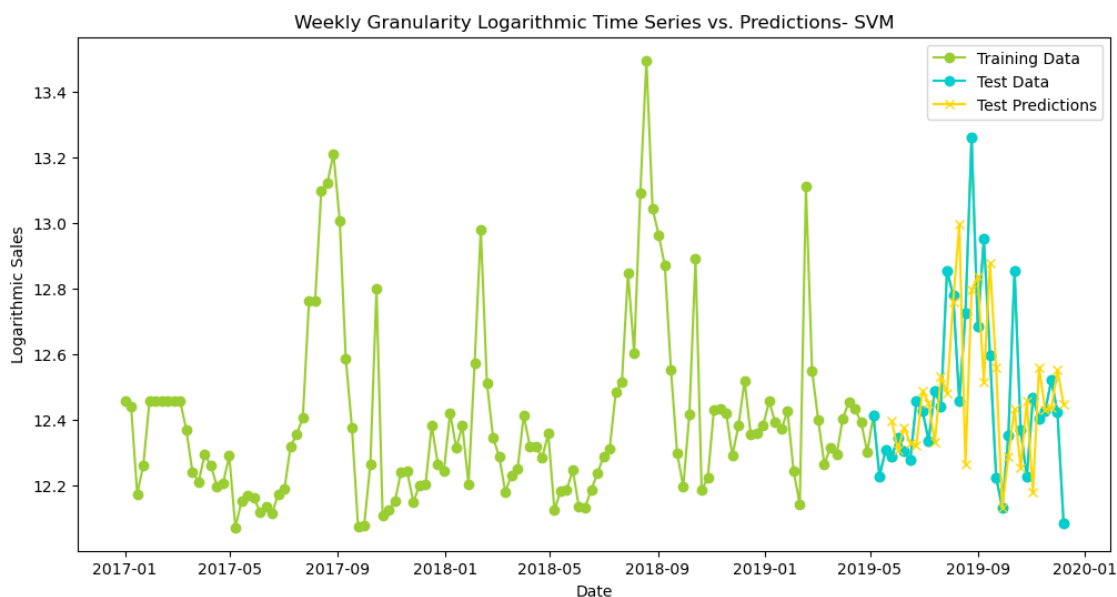


FIGURE 5.47: Time Series Decomposition- Aggregated by week

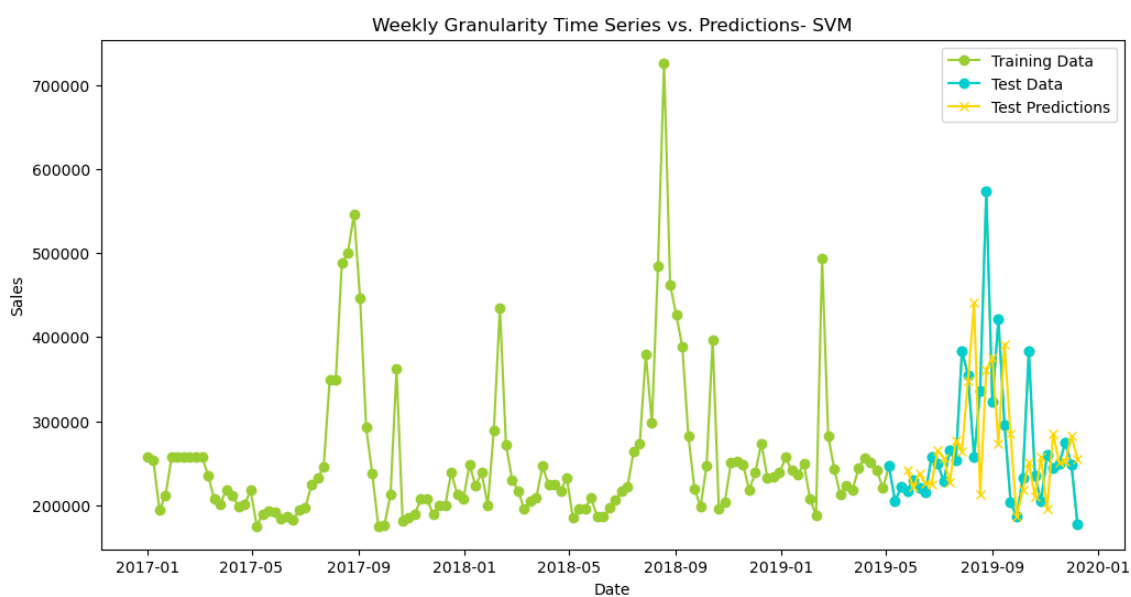


FIGURE 5.48: Time Series Decomposition- Aggregated by week

Random Forest with weekly granularity

Last but not least, the application of Random Forest model to our average sales per week showed a good approximation on the logarithmic scale. However, when we invert this transformation, the predicted data shows more differences from the actual values. Random Forests are known for their ability to capture complex, non-linear relationships in the data. When applied to the logarithmic scale of the data, they may effectively model the underlying patterns and variations present in the

transformed data, however, these patterns might not directly translate when we revert to the original scale.

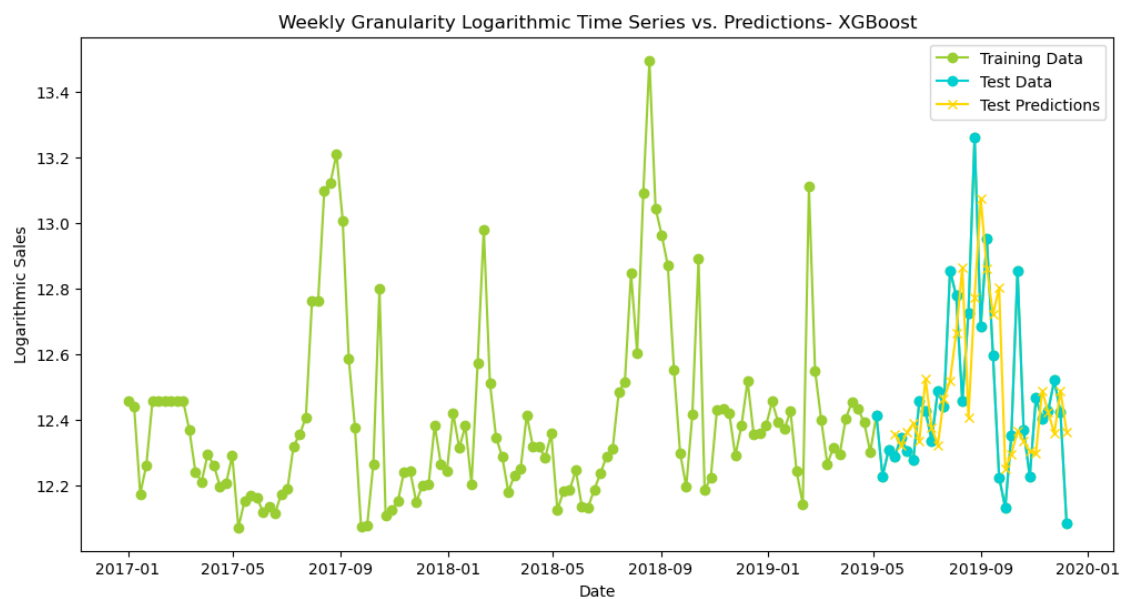


FIGURE 5.49: Time Series Decomposition- Aggregated by week

Another possibility for this behaviour is the Random Forest assumption, in some cases, that the target variable follows a specific distribution. When we transform the target variable, we may violate this assumption, leading to discrepancies between predicted and actual values in the original scale. Despite this topics, we can consider this forecast as very positive.

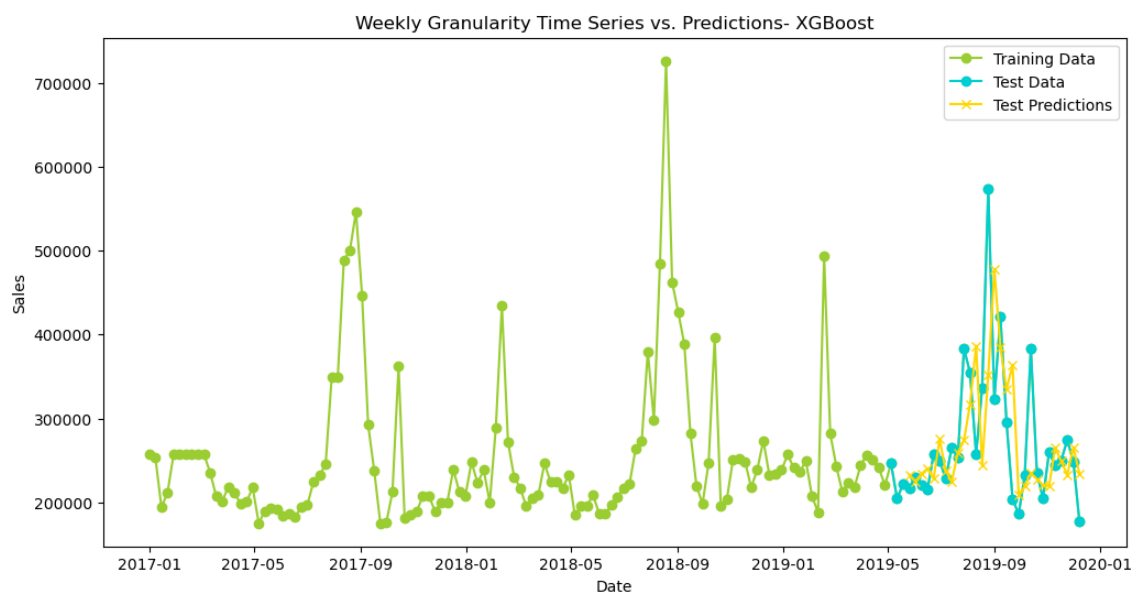


FIGURE 5.50: Time Series Decomposition- Aggregated by week

5.6 Data preparation and processing for bi-weekly granularity

Now let's start our third and last data preparation. In this section we will focus on the study of the time series aggregated by bi-week.

Using bi-weekly data offers several advantages, especially in situations where the quantity of data points is limited.

Firstly, employing bi-weekly intervals provides a balance between granularity and data availability. While daily data might be too fine-grained for the limited dataset we have, monthly aggregation could lead to oversimplification and loss of important insights, since we would have a very limited quantity of points to study.

Secondly, sales trends can be influenced by various external factors, such as holidays, events or marketing campaigns. With bi-weekly data, we can detect shifts in consumer behavior more quickly than with monthly data, allowing the companies to adjust strategies in a timely manner.

So, let's plot the average sales per bi-week and, at the same time the rolling statistics over the plot so we can already infer what is happening to the mean and standard deviation of this series.

Important to note that rolling statistics are meant to capture trends and patterns over time. To do so, they rely on historical data within the rolling window. Only after the rolling window has moved beyond the initial data points it will start producing meaningful rolling statistics.

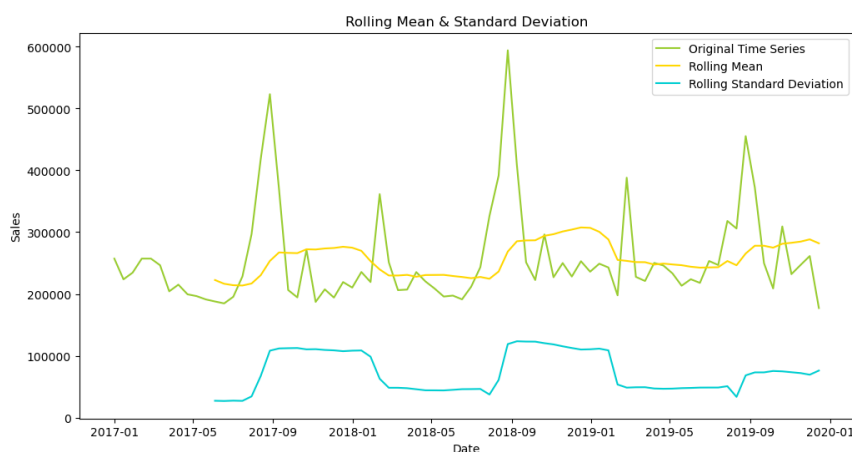


FIGURE 5.51: Rolling statistics Plot- Bi-weekly granularity

Let's proceed to the second approach to test stationary: ADF test e KPSS test.

Test Statistic	p-value
-4.661	$9,900 \times 10^{-5}$

TABLE 5.9: Augmented Dickey-Fuller Test Results

Test Statistic	p-value
0.097	0.100

TABLE 5.10: KPSS Test Results

The ADF test's null hypothesis is that the time series data is non-stationary. In this case, the extremely low p-value suggests strong evidence to reject the null hypothesis. Therefore, the time series is likely stationary.

For the KPSS test, the null hypothesis is that the time series data is stationary around a deterministic trend. In this case, the p-value of 0.100 is higher than the typical significance level of 0.05. This suggests that there is strong evidence to accept the null hypothesis of stationarity.

In summary, both tests support the stationarity of the time series, thus this will be used for the following analysis.

3rd Investigation Resume

Scope: Time Series aggregated by bi-week

Normalization Technique: MinMaxScaler

Split Technique: Fixed Split (80% for the train set and the remaining 20% to the test)

Lookbacks considered: 1,2

Models Applied: RNN, LSTM, GRU, XGBoost, LightGBM, MLP, SVM, Random Forests

5.7 Stock Forecast using Machine Learning for bi-weekly granularity

Recurrent Neural Network (LSTM)

The network scanned the training set 300 times (the greater the number of epochs the better but slower the model) with batch size=10 (approximately 4 interactions per epoch since we have 38points in x_{train}).

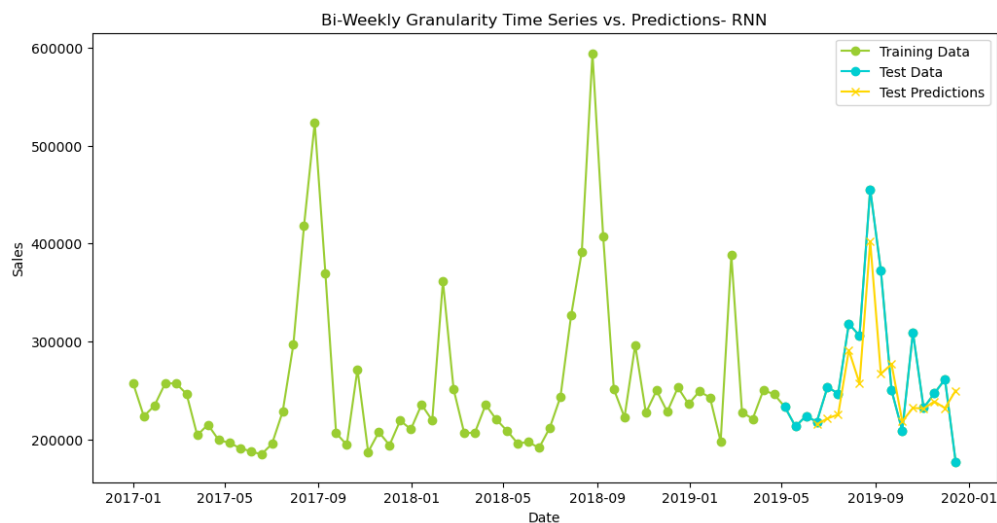


FIGURE 5.52: Recurrent Neural Network- Model Application

As we can see above the sales forecasted curve seems to fit almost perfectly the test data. As for previous applications of the same model, RNN, there were not such good results, we can conclude that the bi-weekly division seems to have a positive impact on the results of our forecast

Another important factor, which is easier to grasp from the graphical visualization, is that the first 3 points of the x_{test} don't have predictions. This occurs because we defined a sliding window of size 3. This means that, in order to make a prediction at a specific point in time, the model uses the three immediately preceding data points. When applying the model to the x_{test} , it starts from the fourth point onwards, as the first three points do not have a sufficient number of previous data points to make predictions. Therefore, the model begins making predictions starting from the third point onward.

When we set a reduced sequence length at the initial moments of testing, we prevent the existence of gaps in predicted data. An opportunity for improvement would be to increase this parameter, the *sequence_length* as more data becomes available, allowing the model to start making predictions based on longer and more accurate sequences. However, this improvement is not possible for this case study since our x_{test} is not large enough to accommodate its implementation.

Long Short-Term Memory (LSTM)

Visualizing the results of our sales forecasting using a trained Long Short-Term Memory (LSTM) model we obtained a slightly worse fit of our forecast to the test curve, comparing with RNN model. To obtain the following results it was needed to define some parameteres like lookback, the number of previous time steps (lags) to use as input features for predicting the next time step. In this case, it makes sense to be 1 since the prediction of average sales for a specific bi-week tends to be more accurate when we consider the data from the previous 2 weeks. This is because market conditions vary significantly between different weeks, and the most recent observations provide us with a better understanding of the trends in the upcoming time.

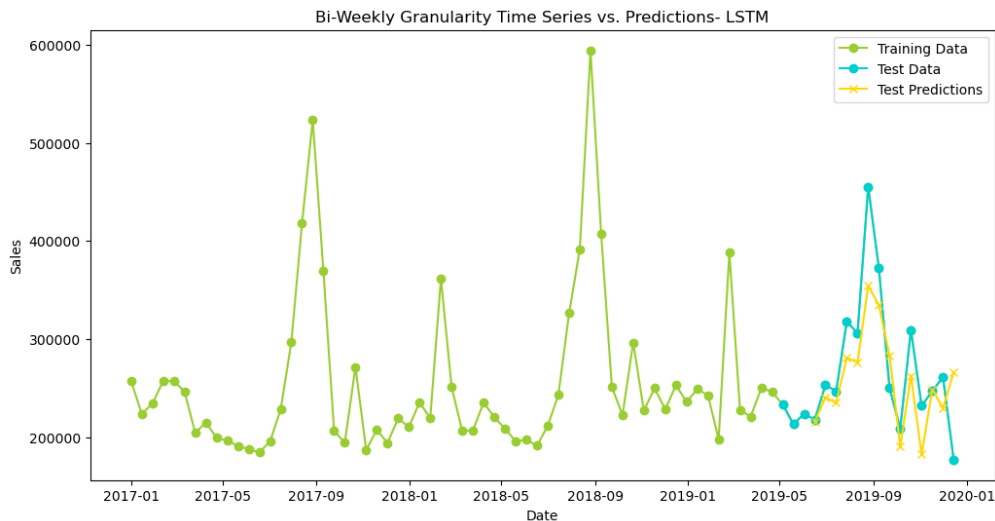


FIGURE 5.53: Long Short Term Memory- Model Application

There can be several reasons for this difference in performance comparing to RNN. By setting the lookback to 1, it means that LSTM considers the immediate previous time step to make predictions. However, it still maintains and updates its internal memory over longer sequences, allowing it to capture long-term dependencies effectively.

This sales data contains complex patterns and seasonality that span multiple time steps, making LSTMs more capable of capturing such patterns.

Gated Recurrent Unit (GRU)

Moving to GRU, to apply this model we needed to define some of the parameters of LSTM. This is probably one of the reasons why, visually, this model behaves very

similar to LSTM. Since this model only looks to the previous three weeks to predict the following three, the first 2 elements of the x_{test} doesn't have elements behind it to apply the trained model and get a predicted value. So our model will only give us a $y_{predicted}$ for the third element of x_{test} , that is, the first element of y_{test} .

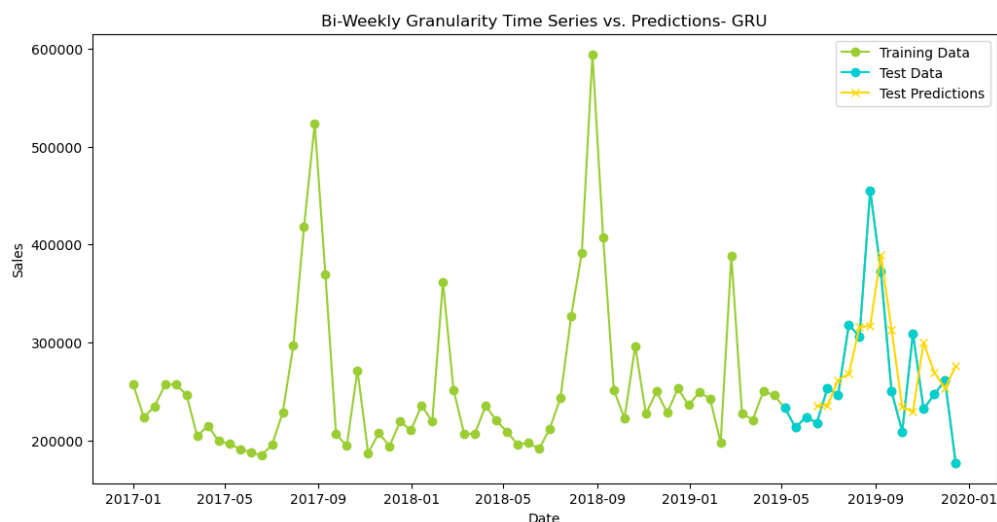


FIGURE 5.54: Gated Recurrent Unit- Model Application

XGBoost

The XGBoost regressor model employed in this analysis uses an ensemble of 100 decision tree estimators. This model is trained on the provided training dataset, with verbose mode enabled to monitor its training progress and receive updates on its performance.

Following the training phase, the trained XGBoost model is employed to generate predictions for the test dataset. This algorithm leverages the collective predictions of multiple decision trees, known for its capability to achieve high predictive accuracy and capture intricate relationships within the data.

During the model experimentation process, various configurations were tested, including the use of 10, 100, and 1000 estimators (trees) within the ensemble. Ultimately, it was determined that configuring the model with 100 estimators provided the optimal balance between predictive accuracy and avoiding over-fitting. A higher number of estimators allows the model to discern more complex patterns in the data, but finding the right balance is crucial to prevent over-fitting.

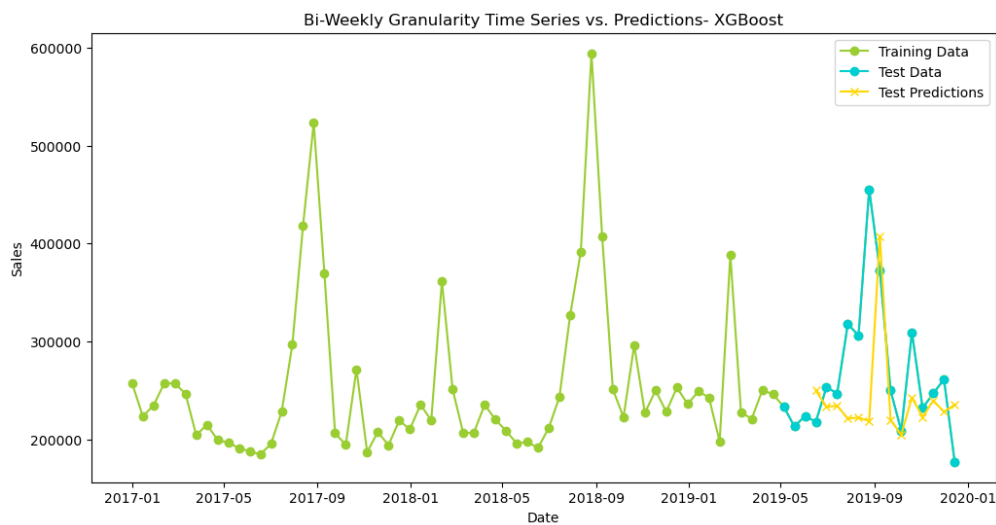


FIGURE 5.55: XGBoost- Model Application

The results of applying this model are interesting: although the model can detect the peaks, the forecasts seem to be arriving late, hence the curve's undoing. This phenomenon can be justified by the choice of non-optimal parameters.

LightGBM

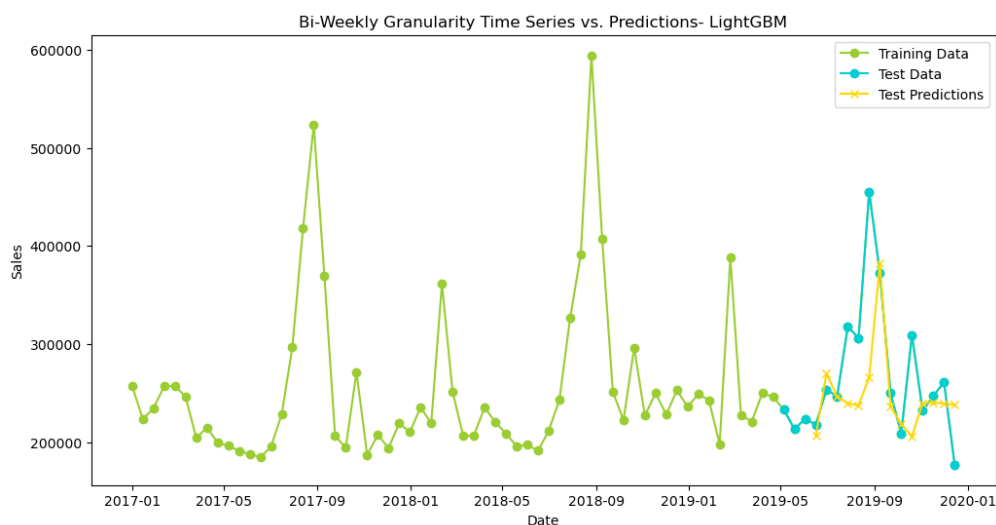


FIGURE 5.56: LightGBM- Model Application

What's interesting in this application is that despite the different growth strategies, LightGBM produces predictions very closely to those generated by XGBoost. This similarity can be attributed to the fact that both models are highly effective at capturing complex data patterns. While their tree structures differ, their overall predictive power allows them to generate similar forecasts.

Multi-Layer Perceptron (MLP)

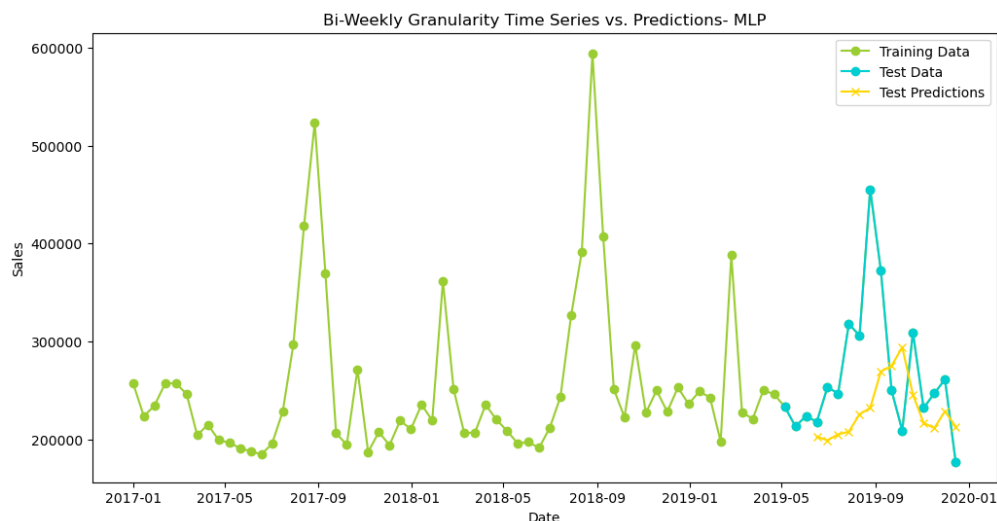


FIGURE 5.57: Multi Layer Perceptron- Model Application

Multi-Layer Perceptrons (MLPs) offer a high degree of versatility in designing neural network architectures, granting control over critical aspects such as the number of layers, the quantity of units within each layer, and the selection of activation functions.

In our study case, extensive experimentation led us to identify a specific MLP architecture that excels in forecasting sales data. This neural network architecture consists of two dense (fully connected) layers: the first layer comprises 100 units and employs the ReLU (Rectified Linear Unit) activation function. ReLU is known for its ability to introduce non-linearity into the model, which is crucial for capturing complex relationships in the data. The second layer contains a single unit, representing the output layer. As the model aims to perform regression (predicting continuous values, such as sales), a single output unit suffices for the task.

The learning rate was set to 0.0003. It governs the size of the steps taken during the optimization process and this smaller learning rate can help the model converge to the optimal

For weekly granularity, MLP demonstrated a deficit in its performance. For the bi-weekly analysis, the same behavior remains. This might indicate that the network is not complex enough to capture the intricacies of the underlying patterns in data since this is the worse approach we have so far for this granularity.

Support Vector Machines (SVM)

It was also constructed a Support Vector Machine model. It was needed to specify some parameters like the kernel. It was used the RBF kernel and for epsilon was set to 0.1 because we are working with a logarithmic scale, which means that predictions within 0.1 units of the actual values are considered accurate.

We experimented the model with different values for epsilon and kernel types but those mentioned above were the better ones.

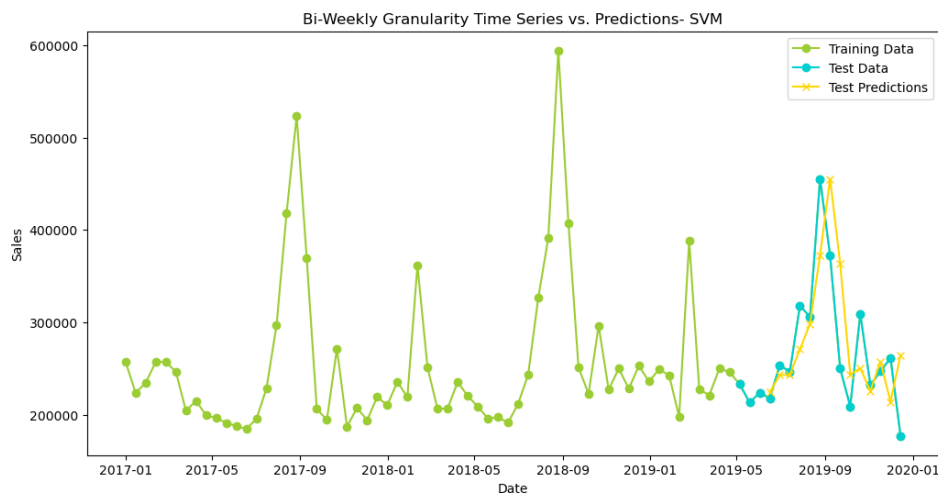


FIGURE 5.58: Support Vector Machine- Model Application

The forecast seems to track the actual behavior reasonably well.

Random Forests

One of the most critical parameters is the number of estimators, which plays a central role in defining the architecture of the Random Forest model. We've set this number of estimators into 1000, which indicates that this Random Forest ensemble comprises 1000 decision trees. This substantial number of trees can help the model capture intricate patterns in the data, resulting in accurate predictions.

Aside from the number of estimators, there are additional hyperparameters that can be tuned to optimize the Random Forest's performance like the random state. This parameter ensures reproducibility by initializing the random number generator. By setting a specific value we are ensuring that the random processes used in tree construction are consistent across different runs of the model. This was useful and necessary for obtaining consistent results during experimentation.

By looking for the visualizations we encounter predicted curves that differ from the expected or observed behavior but not significantly. This discrepancy that we're observing is due to the intrinsic complexity of the dataset. This data contains irregularities and nonlinear relationships therefore Random Forest models, although powerful, may struggle to capture complex patterns.

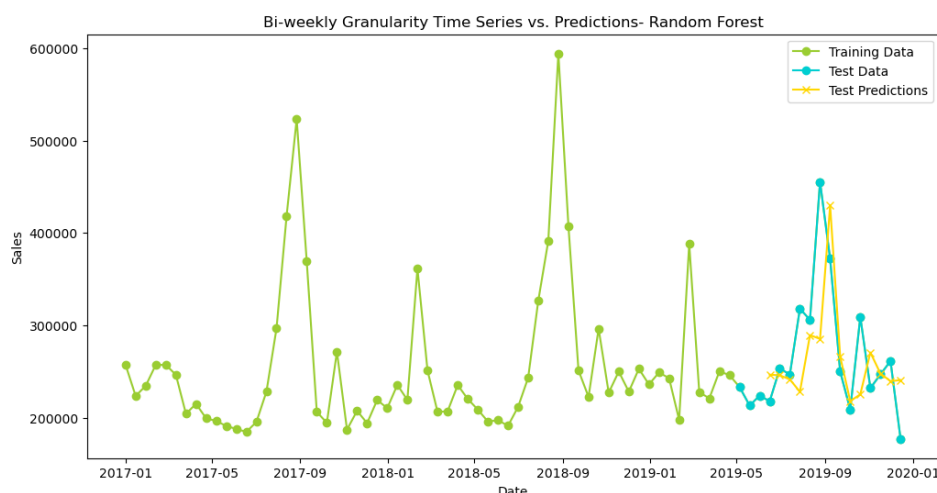


FIGURE 5.59: Random Forests- Model Application

Another important factor to analyse since it may be the cause for the discrepancies we are watching in this Random Forest application, is the selection of lookback.

In the context of bi-weekly data, it's essential to emphasize that even seemingly small lookback values can carry substantial implications. For instance, when setting lookback to a value like 2, it implies that the model is tasked with predicting the value for the current bi-week based on the information from the preceding two bi-weeks. This may appear to be a concise historical context, but it encapsulates the temporal dependencies and patterns within the data over that specific timeframe.

5.8 Evaluating Models Accuracy

Evaluating the accuracy of a model is a crucial step in the forecasting process since it allows us to assess how well the model is performing compared to actual data. Visually we can draw some conclusions and take some insights but just by checking the next metrics we know for sure how accurate our models are. This evaluation consists in comparing the model-generated forecasts to the actual values of the time series, y_{test} .

ML Model	Lookback=7			Lookback=20			Lookback 30		
	MAE	MAPE	MSE	MAE	MAPE	MSE	MAE	MAPE	MSE
RNN	7283.34	2.51	18291089.83	8380.76	2.41	21501136.81	6458.94	2.96	13824854.27
LSTM	7067.85	3.24	13111528.23	7466.35	2.74	17942830.18	6990.56	2.50	16523043.32
GRU	6155.25	3.04	12064624.94	8090.09	2.77	17865887.40	8635.66	2.49	19433094.48
XGBoost	5662.45	1.80	11642629.23	6046.60	1.66	12992230.43	6746.38	1.50	15408819.05
LightGBM	6634.09	2.09	15657723.17	6587.74	1.77	16739491.46	6870.80	1.66	18147014.83
MLP	6245.34	1.87	13195219.23	7758.15	3.20	17238865.05	7399.87	3.17	18758647.68
SVM	10005.30	5.46	18785481.70	11191.81	5.33	22155955.18	10953.03	5.06	24387400.66
R.Forest	5526.87	1.65	12560221.17	5824.51	1.33	13682373.74	6223.82	1.31	15545449.42

TABLE 5.11: Daily Granularity - Models Evaluation for Different Lookback Values

ML Model	Logarithmic TS			Original TS		
	MAE	MAPE	MSE	MAE	MAPE	MSE
RNN	0.18	0.01	0.07	416.73	0.17	304421.16
LSTM	0.21	0.01	0.07	429.64	0.22	318739.74
GRU	0.17	0.01	0.05	424.97	0.16	311288.23
XGBoost	0.19	0.01	0.06	463.77	0.19	357931.96
LightGBM	0.20	0.01	0.08	491.40	0.21	367592.14
MLP	0.42	0.03	0.25	428.03	0.32	320474.72
SVM	0.19	0.01	0.06	400.22	0.19	296245.88
R.Forest	0.17	0.01	0.05	454.19	0.16	342219.20

TABLE 5.12: Weekly Granularity- Models Evaluation

ML Model	Original TS		
	MAE	MAPE	MSE
RNN	36736.97	0.13	2246714.10
LSTM	35857.51	0.13	2080889.41
GRU	45031.03	0.16	3486526.89
XGBoost	51822.78	0.16	6080105.48
LightGBM	42750.95	0.13	4460791.06
MLP	65771.15	0.21	7121328.83
SVM	42853.19	0.15	3127922.29
R.Forest	43525.90	0.14	3918225.57

TABLE 5.13: Bi-weekly Granularity- Models Evaluation

We will analyse the accuracy of 3 different situations: time series forecasting for daily, weekly and bi-weekly granularity.

The second investigation was based on the series transformed using the logarithmic function. When working with this type of time series data that have undergone a logarithmic transformation, to assess the accuracy of forecasts in relation to the original data, it makes sense to apply an exponential transformation to the forecasts and then calculate the error on the original scale. However, it's important to note that when making this transformation, the error calculated on the original series may be larger than the error calculated on the logarithmic scale. This can happen for several reasons. The exponential transformation amplifies the influence of extreme values (outliers) in the forecasts. Errors in logarithmic forecasts may be less affected by outliers. Values transformed exponentially tend to have an asymmetric distribution with long tails, which can result in larger errors when compared to the original series. Therefore, although the error may be larger in the original series after the exponential transformation, it makes sense because this transformation considers the scale of the actual values and takes into account the nature of the time series.

If we look into the results from daily granularity the Mean Absolute Percentage Error (MAPE) may not be the most suitable choice due to the nature of daily sales data, which can sometimes include zero values. The MAPE calculation involves dividing the absolute error by the actual value, which can lead to issues when the actual value is zero. In such cases, the denominator becomes zero, and the MAPE is undefined.

To handle this issue, when using the *meanabsolutepercentageerror* function from *sklearn.metrics*, it automatically adds a small epsilon value to the denominator to avoid division by

zero errors. This epsilon value is typically set to a small positive number to ensure the calculation doesn't break down. However, the use of this epsilon value can skew the MAPE results when dealing with daily sales data because it may produce artificially high MAPE values. This is because the epsilon value effectively becomes the lower limit for error calculation in cases where actual values are close to zero. Therefore, when analyzing daily granularity data, it's essential to be aware of this limitation of MAPE and consider alternative evaluation metrics.

Looking for the **daily granularity**, the model that performs worse was SVM and the top 2 models were Random Forest and XGBoost. A relevant insight to take away from these results is that increasing Lookback does not necessarily involve an improvement in model performance. RNN and MLP are the models that break the rule and therefore this lookback quality relationship is not so linear. One possible explanation for this behaviour is that RNN and MLP, having higher capacities due to their complex architectures, are more prone to overfitting. Increasing lookback might lead to overfitting, causing a decline in performance. Hence, the optimal lookback beyond which the model's performance deteriorates may be lower than the optimal value for the others.

SVM's performance may be affected by the choice of kernel and hyperparameters. In the case of daily data, finding the right combination of hyperparameters can be challenging, especially when dealing with a large number of features (lags) that result from high-frequency data. SVM tends to work well when data is linearly separable or nearly so. However, daily sales data often exhibits non-linear patterns, which could lead to suboptimal performance.

The higher performance of Random Forest with lookback=30 suggests that this model is less prone to overfitting, especially because the number of trees in the forest was controlled (parameter defined). By averaging the predictions of multiple trees, Random Forests reduce the risk of modeling the noise in the data. Besides this, the consistent performance of this model across different lookbacks indicates its adaptability to varying data characteristics.

In summary, the choice of the best-performing model may depend on the specific characteristics of the daily sales data and the desired forecasting horizon.

Focusing now on **weekly data**. When comparing the performance of machine learning models on the logarithmic time series (Logarithmic TS) and the original time series (Original TS), it's evident that there are notable differences in evaluation metrics. Mean Absolute Error (MAE) and Mean Absolute Percentage Error (MAPE) are higher when applied to the original time series, indicating that the models tend to perform better on the logarithmic scale. This suggests that converting data to a logarithmic scale helps in modeling and reducing errors.

On the logarithmic scale, several models exhibit relatively low MAE values, indicating good predictive accuracy. These models include Random Forest, RNN and GRU.

When evaluating models on the original time series, GRU stands out with the lowest MAE, indicating strong performance in terms of forecasting sales at the weekly granularity. Models like MLP exhibit higher MAE values on the original scale, indicating potential challenges in accurately capturing the underlying patterns in the data at this granularity.

The reasons given for the good performance of Random forest at daily granularity now also apply to the weekly case. However, the GRU also stands out

The performance of GRU stems from its exceptional capability to capture intricate temporal dependencies and non-linear patterns within the data. Its adaptability to subtle and evolving patterns empowers GRU to excel in modeling the intricate dynamics inherent in the dataset, making it a potent choice for time series forecasting tasks.

In contrast, MLP produced poorer results due to its sensitivity to data complexity, leading to challenges in effectively learning intricate patterns. Additionally, inadequate hyperparameter tuning and the risk of overfitting further impacted its performance, resulting in suboptimal predictions.

In summary, the choice of the best-performing model and the scale of the data should be aligned with the goals of the forecasting task. The logarithmic scale may be preferable for certain models, while others may perform well directly on the original scale. Additionally, it's essential to consider the implications of high MAPE values when working with high-frequency data and small sales values in the original time series.

About **bi-weekly aggregation**, the model that presented the best results was LSTM. It presented an MAPE of 13%. The one that fit the least to our dataset was MLP. RNN and LightGBM also present satisfactory errors with a MAPE of 13%.

When looking at the error metrics for the original series the values can be frightening at first glance, but it's important to interpret these errors within the specific context of the problem and the characteristics of this time series. This evaluation of the original series provides a more accurate assessment of the model's performance in terms of the actual units of the variable of interest. MAE is in the interval [35857;65771], ie, the average absolute difference between the predicted values and the actual values is on that range. If we look into the time series plot we can see that there are consecutive points with an average decrease of approximately \$202700 (for example, between 2018-08-26 and 2018-08-12) and others in which there is a decrease of approximately \$3000 (between 2017-04-23 and 2017-05-07). Given these oscillations, having this MAE values does not mean that the models do not apply at all to our dataset. MSE is bigger because it amplifies the impact of larger errors due to squaring.

In an overall evaluation, particularly focusing on errors in the original scale of the time series data, the standout performer was the LSTM model in time series data grouped by bi-week. This choice is substantiated by the model's remarkable ability to make accurate predictions on the original sales data, and it can be attributed to several factors.

The LSTM model is able to capture long-term dependencies and intricate patterns because it was meticulously fine-tuned to precisely align with the inherent characteristics of the bi-weekle sales data. Through systematic experimentation with diverse configurations of memory cells, layers, and activation functions, it was determined the most proper architecture to this study case. This level of customization empowered the LSTM to discern nuanced underlying patterns in the data, resulting in significantly reduced errors.

Furthermore, the model's ability to retain and utilize historical context over extended periods allowed it to capture subtle nuances within the data, leading to remarkably accurate predictions.

To conclude the choice of granularity and lookback values significantly influences the model's predictive performance. It's essential to consider the nature of the data and the specific forecasting requirements when selecting the appropriate model.

Chapter 6

Conclusion and Final Thoughts

In this project, different machine learning models were explored for sales forecasting, including RNN, LSTM, GRU, XGBoost, LightGBM, MLP and more. Our primary objective was to develop accurate sales forecasts to aid in decision-making processes. Although the results were not as promising as initially anticipated, this study provided valuable insights into the strengths and weaknesses of each model, as well as the complexities of sales forecasting. Throughout our analysis, several insights emerged. A wide range of models were implemented which gave us a unique perspective and set of assumptions to the problem.

A very time-consuming stage of this project was data processing, but it made us realize that this step is essential and could compromise the entire analysis. Thus, it was possible to study characteristics of time series such as seasonality, trend or even stationarity (and if the series is not stationary, methods to turn it on).

Having different granularity levels in data analysis, such as daily, weekly, and bi-weekly, is crucial for gaining a comprehensive understanding of underlying trends and details within a dataset. Daily granularity allows us to capture intricate day-to-day fluctuations and spot micro-level insights. In contrast, weekly and bi-weekly granularity help identify larger-scale patterns and trends, providing a more holistic perspective on the data without sacrificing a considerable sample size .

Daily granularity is essential for tracking daily sales, monitoring short-term changes, and reacting to fluctuations. It enables businesses to make real-time decisions and address immediate concerns. On the other hand, weekly and bi-weekly granularity

offer a broader view, revealing seasonality, long-term trends, and irregular patterns that might not be apparent at a daily level.

Since we chose to study 3 different date granularities, we were forced to carry out this step 3 times as the series was different and, as such, its characteristics were also different.

In daily granularity analysis, SVM performed poorly, possibly due to the challenge of hyperparameter tuning and non-linear patterns in daily sales data. Random Forest presented the best results with a lookback of 30 proving to be less susceptible to overfitting.

For weekly data, models generally performed better on the logarithmic scale, with Random Forest and GRU standing out on both scales. MLP faced challenges in capturing data complexities, resulting in higher errors. However, higher Mean Absolute Percentage Error (MAPE) values on the original scale were influenced by near-zero sales values, emphasizing the importance of scale alignment and consideration of high-frequency data implications.

Before moving on to the bi-weekly analysis, it's interesting to carry out a cross-sectional analysis of the values obtained between the two different granularities. As we noticed, forecast results have improved significantly when grouping data into weeks.

This happens because with weekly grouping we can capture the inherent cyclic patterns in consumer behavior. Sales data often exhibit weekly trends influenced by various factors such as weekends, promotional events, and purchasing patterns. By grouping the data into weeks, our models can more accurately capture these recurring patterns, enabling them to make more precise predictions.

About the bi-weekly data, the model that presented the best results was LSTM with an error of 13%.

The observed deviations between the predicted sales curves and the actual sales data can be attributed to several factors, including suboptimal model parameters and the unique characteristics of the sales forecasting problem. To address these issues and improve model performance, there are some improvement opportunities.

It's very important to perform a comprehensive search for optimal hyperparameters tailored to this dataset. Adjusting parameters such as the learning rate, the number

of layers, units per layer, and regularization can lead to more accurate forecasts and sometimes this process wasn't the easiest. It didn't consist of finding an optimal parameter but rather a combination of them.

Sales can be influenced by external factors such as economic conditions, holidays, and promotions. By knowing these events we can treat them as outliers and thus prevent them from biasing the results.

One procedure that could possibly improve our results is Cross-Validation. By implementing it to evaluate the models on out-of-sample data, we can prevent overfitting and improve generalization to unseen sales patterns.

Another major challenge that was faced was the complexity of the time series at the beginning, before any treatment. We were facing some troubles in studying trends and seasonality in daily data due to the noise and variability associated with it. Monthly averages provided a smoother view of underlying trends and seasonality patterns.

While this project did not yield exceptional forecasting results, it provided valuable insights into the complexities of sales forecasting and the capabilities of different models. Sales forecasting is a challenging task influenced by various factors, including seasonality, external events, and market dynamics. Continued research and refinement of models, along with access to more comprehensive and accurate data, may lead to improved forecasting accuracy in the future. This project serves as a foundation for future work in sales forecasting and there are several promising directions for improvement. Enhanced hyperparameter tuning, incorporation of external factors, and robust outlier handling can boost forecasting accuracy. Implementing cross-validation techniques and exploring ensemble models offer avenues for more reliable predictions. It is important to emphasize always the importance of model selection, data preprocessing, and continuous evaluation. As the reliance of businesses on data-driven choices continues to grow, the quest for enhanced precision in sales predictions remains a worthwhile pursuit.

Bibliography

- [1] G. Bontempi, S. Ben Taieb, and Y.-A. Le Borgne, *Machine Learning Strategies for Time Series Forecasting*, 01 2013, vol. 138. [Cited on page 1.]
- [2] A. Singh, “Multivariate time series analysis with python for forecasting and modeling (updated 2023),” Apr 2023. [Online]. Available: <https://www.analyticsvidhya.com/blog/2018/09/multivariate-time-series-guide-forecasting-modeling-python-codes/> [Cited on page 2.]
- [3] L. Thomas, “Cross-sectional study: Definition, uses amp; examples,” Jun 2023. [Online]. Available: <https://www.scribbr.com/methodology/cross-sectional-study/> [Cited on page 2.]
- [4] A. Gupta, “All about time series analysis and forecasting,” Oct 2021. [Online]. Available: <https://www.linkedin.com/pulse/all-time-series-analysis-forecasting-angad-gupta> [Cited on page 2.]
- [5] *Time Series*. New York, NY: Springer New York, 2008, pp. 536–539. [Online]. Available: https://doi.org/10.1007/978-0-387-32833-1_401 [Cited on page 3.]
- [6] R. J. Hyndman and G. Athanasopoulos, “Time series,” Oct 2013. [Online]. Available: https://link.springer.com/referenceworkentry/10.1007/978-0-387-32833-1_401#citeas [Cited on page 3.]
- [7] R. S, “Time series analysis: Understanding seasonality and cyclicity,” Sep 2023. [Online]. Available: <https://aaweg-i.medium.com/in-time-series-forecasting-what-do-you-think-is-the-difference-between-seasonality-and-cyclicity-f4e8d9523d24> [Cited on page 3.]

- [8] A. Pant and R. Rajput, *Time Series Analysis of Gold Price Using R*, 10 2019. [Cited on pages [xiii](#) and [4](#).]
- [9] V. K. G, "Statistical tests to check stationarity in time series," Sep 2023. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/06/statistical-tests-to-check-stationarity-in-time-series-part-1/#What_Is_a_Stationary_Series? [Cited on page [4](#).]
- [10] F. Morales, "Time series: Stationarity check," Mar 2020. [Online]. Available: <https://medium.com/analytics-vidhya/time-series-stationarity-check-57f0f018201a> [Cited on page [4](#).]
- [11] B. Ozcan and I. Öztürk, *Time Series Analysis (Stationarity, Cointegration, and Causality)*. Academic Press, an imprint of Elsevier, 2019. [Cited on page [5](#).]
- [12] S. Prabhakaran, "Augmented dickey-fuller (adf) test - must read guide - ml+," Apr 2022. [Online]. Available: https://www.machinelearningplus.com/time-series/augmented-dickey-fuller-test/?utm_content=cmp-true [Cited on page [5](#).]
- [13] "How to use and interpret adf test for time series," Sep 2023. [Online]. Available: <https://www.linkedin.com/advice/0/how-do-you-interpret-results-augmented-dickey-fuller> [Cited on page [6](#).]
- [14] L. Sjösten, "A comparative study of the kpss and adf tests in terms of size and power," 2022. [Online]. Available: <https://www.diva-portal.org/smash/get/diva2:1668033/FULLTEXT01.pdf> [Cited on page [6](#).]
- [15] [Online]. Available: https://www.statsmodels.org/dev/examples/notebooks/generated/stationarity_detrending_adf_kpss.html [Cited on page [7](#).]
- [16] [Online]. Available: <https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html> [Cited on page [7](#).]
- [17] Manish, "Different normalization methods," Mar 2022. [Online]. Available: <https://medium.com/@mkc940/different-normalization-methods-a1be71fe9f1> [Cited on page [8](#).]

- [18] I. Lindgren, "Transformations, scaling and normalization," Jul 2019. [Online]. Available: <https://medium.com/@isalindgren313/transformations-scaling-and-normalization-420b2be12300> [Cited on page 8.]
- [19] Jan 2020. [Online]. Available: <https://towardsdatascience.com/finally-why-we-use-an-80-20-split-for-training-and-test-data-plus-an-alternative-method-oh-yes-edc77e96295d> [Cited on page 9.]
- [20] M. Filho, "How to do time series cross-validation in python," Jul 2023. [Online]. Available: <https://forecastegy.com/posts/time-series-cross-validation-python/#simple-time-split-validation> [Cited on page 9.]
- [21] May 2023. [Online]. Available: <https://www.geeksforgeeks.org/steps-of-forecasting/> [Cited on page 10.]
- [22] F. Petropoulos, D. Apiletti, V. Assimakopoulos, M. Z. Babai, D. K. Barrow, and S. B. et al., "Forecasting: theory and practice," *International Journal of Forecasting*, vol. 38, no. 3, pp. 705–871, 2022. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0169207021001758> [Cited on page 10.]
- [23] Y. Bokobza, "Time series forecasting (part 1 of 3): Understanding the fundamentals," May 2021. [Online]. Available: <https://medium.com/data-science-at-microsoft/time-series-forecasting-part-1-of-3-understanding-the-fundamentals-13b52eda3e5> [Cited on page 11.]
- [24] J. C. Chambers, S. K. Mullick, and D. D. Smith. [Online]. Available: https://www.researchgate.net/publication/344658764_Time_Series_Analysis_of_Gold_Price_Using_R [Cited on pages xiii and 11.]
- [25] G. Tunnicliffe Wilson, "Time series analysis: Forecasting and control, 5th edition, by george e. p. box, gwilym m. jenkins, gregory c. reinsel and greta m. ljung, 2015. published by john wiley and sons inc., hoboken, new jersey, pp. 712. isbn: 978-1-118-67502-1," *Journal of Time Series Analysis*, vol. 37, pp. n/a–n/a, 03 2016. [Cited on page 13.]
- [26] S. S. Wulff, "Time series analysis: Forecasting and control, 5th edition," *Journal of Quality Technology*, vol. 49, no. 4, p. 418–419, 2017. [Cited on page 13.]

- [27] S. Hochreiter and J. Schmidhuber, "Long short-term memory," Nov 1997. [Online]. Available: <https://doi.org/10.1162/neco.1997.9.8.1735> [Cited on page 14.]
- [28] S. Hochreiter and Schmidhuber, "Brief history of machine learning," Nov 1997. [Online]. Available: <https://doi.org/10.1162/neco.1997.9.8.1735> [Cited on page 14.]
- [29] A. Bahnsen, "Introduction to machine learning and decision trees," Nov 2017. [Online]. Available: <https://www.dataversity.net/introduction-machine-learning-decision-trees/> [Cited on page 14.]
- [30] M. Cruz-Cunha, *Handbook of Research on ICTs and Management Systems for Improving Efficiency in Healthcare and Social Care*, 01 2013. [Cited on page 15.]
- [31] "What are recurrent neural networks?" [Online]. Available: <https://www.ibm.com/topics/recurrent-neural-networks> [Cited on page 15.]
- [32] J. Dancker, "A brief introduction to recurrent neural networks," Dec 2022. [Online]. Available: <https://towardsdatascience.com/a-brief-introduction-to-recurrent-neural-networks-638f64a61ff4> [Cited on pages xiii and 16.]
- [33] R. DiPietro and G. D. Hager, "Chapter 21 - deep learning: Rnns and lstm," in *Handbook of Medical Image Computing and Computer Assisted Intervention*, ser. The Elsevier and MICCAI Society Book Series, S. K. Zhou, D. Rueckert, and G. Fichtinger, Eds. Academic Press, 2020, pp. 503–519. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/B9780128161760000260> [Cited on page 16.]
- [34] T. Navamani, "Chapter 7 - efficient deep learning approaches for health informatics," in *Deep Learning and Parallel Computing Environment for Bioengineering Systems*, A. K. Sangaiah, Ed. Academic Press, 2019, pp. 123–137. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/B9780128167182000142> [Cited on page 17.]
- [35] D. Parikh, "Disadvantages of rnn," Jan 2021. [Online]. Available: <https://iq.opengenus.org/disadvantages-of-rnn/> [Cited on page 17.]

- [36] B. Or, "The exploding and vanishing gradients problem in time series," Oct 2020. [Online]. Available: <https://towardsdatascience.com/the-exploding-and-vanishing-gradients-problem-in-time-series-6b87d558d22> [Cited on page 17.]
- [37] S. M, "Let's understand the problems with recurrent neural networks," Jul 2021. [Online]. Available: <https://www.analyticsvidhya.com/blog/2021/07/lets-understand-the-problems-with-recurrent-neural-networks/> [Cited on page 17.]
- [38] A. Pai, "Analyzing types of neural networks in deep learning," Jul 2023. [Online]. Available: <https://www.analyticsvidhya.com/blog/2020/02/cnn-vs-rnn-vs-mlp-analyzing-3-types-of-neural-networks-in-deep-learning/> [Cited on page 18.]
- [39] B. Lindemann, T. Müller, H. Vietz, N. Jazdi, and M. Weyrich, "A survey on long short-term memory networks for time series prediction," *Procedia CIRP*, vol. 99, pp. 650–655, 2021, 14th CIRP Conference on Intelligent Computation in Manufacturing Engineering, 15-17 July 2020. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2212827121003796> [Cited on page 18.]
- [40] J. Dancker, "A brief introduction to recurrent neural networks," Dec 2022. [Online]. Available: <https://towardsdatascience.com/a-brief-introduction-to-recurrent-neural-networks-638f64a61ff4> [Cited on pages xiii, 18, and 19.]
- [41] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, pp. 1735–80, 12 1997. [Cited on page 19.]
- [42] J. Razin, M. A. Karim, M. Ph. D., S. Rifat, and T. Alam, *A Long Short-Term Memory (LSTM) Model for Business Sentiment Analysis Based on Recurrent Neural Network*, 01 2021, pp. 1–15. [Cited on page 20.]
- [43] M. Phi, "Illustrated guide to lstm's and gru's: A step by step explanation," Jun 2020. [Online]. Available: <https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21> [Cited on page 20.]

- [44] J. Dancker, "A brief introduction to recurrent neural networks," Dec 2022. [Online]. Available: <https://towardsdatascience.com/a-brief-introduction-to-recurrent-neural-networks-638f64a61ff4> [Cited on pages xiii, 20, and 21.]
- [45] S. Kostadinov, "Understanding gru networks," Nov 2019. [Online]. Available: <https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be> [Cited on page 21.]
- [46] R. C. Staudemeyer and E. R. Morris, "Understanding LSTM - a tutorial into long short-term memory recurrent neural networks," *CoRR*, vol. abs/1909.09586, 2019. [Online]. Available: <http://arxiv.org/abs/1909.09586> [Cited on page 22.]
- [47] J. Brownlee, "A gentle introduction to the gradient boosting algorithm for machine learning," Aug 2020. [Online]. Available: <https://machinelearningmastery.com/gentle-introduction-gradient-boosting-algorithm-machine-learning/> [Cited on pages 22 and 23.]
- [48] T. Chen and C. Guestrin. [Online]. Available: <https://arxiv.org/pdf/1603.02754.pdf> [Cited on pages 22 and 23.]
- [49] K. J. Wong, "Catboost vs. lightgbm vs. xgboost," Nov 2022. [Online]. Available: <https://towardsdatascience.com/catboost-vs-lightgbm-vs-xgboost-c80f40662924> [Cited on pages xiii, 23, and 24.]
- [50] A. Lev, "Xgboost versus random forest," Dec 2022. [Online]. Available: <https://www.qwak.com/post/xgboost-versus-random-forest> [Cited on page 23.]
- [51] [Online]. Available: <https://lightgbm.readthedocs.io/en/stable/> [Cited on page 24.]
- [52] M. Shiblee, P. K. Kalra, and B. Chandra, "Time series prediction with multi-layer perceptron (mlp): A new generalized error based approach," in *Advances in Neuro-Information Processing*, M. Köppen, N. Kasabov, and G. Coghill, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 37–44. [Cited on pages xiii and 26.]
- [53] G. Zhang, B. Eddy Patuwo, and M. Y. Hu, "Forecasting with artificial neural networks:: The state of the art," *International Journal of Forecasting*, vol. 14,

- no. 1, pp. 35–62, 1998. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0169207097000447> [Cited on page 27.]
- [54] K. jae Kim, “Financial time series forecasting using support vector machines,” *Neurocomputing*, vol. 55, no. 1, pp. 307–319, 2003, support Vector Machines. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0925231203003722> [Cited on page 27.]
- [55] A. Saini, “Guide on support vector machine (svm) algorithm,” Jul 2023. [Online]. Available: <https://www.analyticsvidhya.com/blog/2021/10/support-vector-machinessvm-a-complete-guide-for-beginners/> [Cited on pages xiii and 28.]
- [56] L. Breiman, “Random forests - machine learning.” [Online]. Available: <https://link.springer.com/article/10.1023/A:1010933404324> [Cited on page 29.]
- [57] V. Kontopoulou, A. Panagopoulos, I. Kakkos, and G. Matsopoulos, “A review of arima vs. machine learning approaches for time series forecasting in data driven networks,” *Future Internet*, vol. 15, p. 255, 07 2023. [Cited on page 31.]
- [58] Vinithavn, “Error metrics in machine learning,” Jul 2021. [Online]. Available: <https://medium.com/analytics-vidhya/error-metrics-in-machine-learning-f9eed7b139f> [Cited on page 31.]
- [59] Jun 2023. [Online]. Available: <https://mlpills.dev/time-series/error-metrics-for-time-series-forecasting/> [Cited on page 32.]