Algerian People's Democratic Republic

Ministry of Higher Education and Scientific Research

University of Saida - Dr. Moulay Tahar

Faculty of MIT

Department of Mathematics

Dissertation submitted for the degree of

Master of Science

Field: MATHEMATICS

Specialization: Stochastic Statistical Analysis of Processes and Applications

by

EL Keurti Hibat Allah Wisal[1]

Under the supervision of

Dr. F. Mokhtari

Dissertation:

Artificial Intelligence for Time Series Forecasting

Defended on 14/06/2024 before the jury composed of

Dr. F. Benziadi	**University of Saida Dr. Moulay Tahar**	**President**
Dr. F. Mokhtari	**University of Saida Dr. Moulay Tahar**	**Supervisor**
Dr. R. Rouane	**University of Saida Dr. Moulay Tahar**	**Examiner**

Academic Year: 2024/2025

[1]email: elkeurtihiba@gmail.com

Acknowledgments

In the name of Allah, the Most Gracious, the Most Merciful. All praise is due to Allah alone. May peace and blessings be upon His noble Prophet Muhammad (peace be upon him).

I wish to express my profound gratitude to Allah Almighty for bestowing upon me the strength, perseverance, and guidance during my academic journey. The completion of this thesis would not have been achievable without His mercy and support.

I would like to convey my heartfelt appreciation to my supervisor, Dr. Fatiha Mokhtari, for her unwavering guidance, insightful feedback, and genuine encouragement. Her expertise and patience have been invaluable sources of motivation and support throughout this endeavour.

My sincere gratitude is also extended to all my esteemed professors, including Dr. F. Benziadi, Dr. R. Rouane, Mr. K. Chouaf, a supervisor at Sonalgaz, and Mr. Saadli, who provided me with the opportunity for an internship at Sonalgaz, along with others, whose knowledge and mentorship have made a significant impact on both my academic and personal development.

May Allah reward abundantly all those who have contributed to my journey, whether directly or indirectly.

Dedication

I would like to express my profound gratitude to **God**, Almighty, for granting me the courage, determination, and health necessary to complete this endeavour. Throughout my life, I have consistently appreciated the individuals who hold great significance in my journey. Today, I am pleased to present this work as a gesture of appreciation for their unwavering support and efforts in helping me achieve success.

I dedicate this work to:

My esteemed parents **Abd ELkrim** and **Fatiha**, especially my mother, who has been my mentor, sister, friend, and guiding teacher, I express my profound gratitude for your sacrifices, steadfast love, kindness, support, and prayers throughout my academic journey. I wish you a life filled with happiness and good health.

My dear brothers, **Mohamed Bahaa Eddin** and **Salah Eddin**, for their support and encouragement.

My late grandfather **Youssef** and grandmother **Fatma**, may God have mercy on their souls, and to my living grandparents **Meriem** and **Khelifa**, may God grant them long lives.

To all my **family** without exception.

I would like to express my sincere gratitude to all my **friends**, particularly **Karima**, for their invaluable support and companionship throughout my academic journey.

Contents

List of Figures

Introduction

In recent years, Artificial Intelligence (AI) has experienced a remarkable surge in interest, largely due to advancements in Machine Learning (ML) and Deep Learning. These technologies have significantly enhanced computers ability to process, analyse, and learn from sequential data capabilities that are fundamental to time series forecasting, a crucial field in many areas such as finance, meteorology, energy, healthcare, and economics.

The history of time series analysis dates back over a century. In the early 20th century. It began as a way to analyse sequential data collected over time and it is a fundamental statistical method used for investigating data collected at regular time intervals. Time series are distinguished by their long-term trends, seasonal fluctuations, cyclical patterns, and unpredictable variations.

Time series models have been used in forecasting for several decades and are widely used in logistics for sales or demand forecasting, see, e.g., [15], [22] and the references cited therein. A grasp of temporal relationships is vital for making precise forecasts. Foundational statistical techniques were developed, such as moving averages and exponential smoothing, which provided simple yet effective tools for trend and seasonality analysis. A major leap occurred in the 1970s with the introduction of the ARIMA (AutoRegressive Integrated Moving Average) model by Box and Jenkins. ARIMA and its variants became standard methods due to their effectiveness in modelling linear patterns in time series data. During the 1980s and 1990s, more advanced statistical models like ARCH and GARCH revolutionized financial econometrics by effectively modelling volatility in financial data.

However, as data grew more complex and nonlinear, these classical approaches showed limitations. This led to the emergence of neural network based models in the late 1990s and early 2000s, such as Recurrent Neural Networks (RNNs). These models offered more flexibility but were often difficult to train and limited by issues like vanishing gradients.

A breakthrough in deep learning came with the development of the Long Short-Term Memory (LSTM) network in 1997 by Hochreiter and Schmidhuber [11]. LSTMs addressed the limitations of RNNs and became widely adopted for time series prediction due to their ability to capture long-term dependencies.

In parallel, ensemble learning methods like Random Forest, introduced by Breiman in 2001,

and XGBoost (Extreme Gradient Boosting), developed by Chen and Guestrin in 2016 [8], gained popularity. Although not originally designed for time series forecasting, these models have been effectively adapted for it using feature engineering techniques such as lag variables, rolling statistics, and time based cross validation. They are particularly valued for their robustness, ability to model nonlinear relationships, and high predictive accuracy in structured data settings. XGBoost, in particular, has become a go to model in many data science competitions and real world forecasting applications.

Several studies have shown that ML methods such as neural networks, support vector regression, and Random Forests can outperform traditional time series models for specific demand forecasting problems. For example, a study by [10] compared the prediction power of more than ten different forecasting models, including classical methods such as ARIMA and ML techniques such as long short-term memory (LSTM) and convolution neural networks, using a single data set containing the sales history of furniture in a retail store. The results showed that the LSTM outperformed the other models in terms of prediction performance. Another study by [14] also compared the forecasting power of ARIMA and neural networks using a single commodity prices data set.

Today, AI driven time series forecasting combines large datasets, sophisticated models, and high computational power to deliver predictions with unprecedented precision. These models not only identify complex temporal patterns but also adapt to changing dynamics in real time, making them indispensable tools in both research and industry. This evolution from statistical modelling to ensemble learning and deep learning reflects the broader transformation of time series forecasting into a more intelligent, adaptive, and data driven discipline.

This dissertation delves into time series analysis through the lens of artificial intelligence, spotlighting machine learning methods like Random Forest and XGBoost and deep learning like RNNs and LSTM networks.

The initial chapter lays out the groundwork for understanding time series forecasting and gives an introduction to artificial intelligence. It explores essential methods, modelling techniques, and practical uses. Key aspects covered include defining time series, showcasing their applications in various sectors, and elaborating on their defining features. Furthermore, the chapter provides also an overview of artificial intelligence (AI), beginning with an explanation of what AI is and its fundamental concepts. It explores the various subfields within AI, highlighting their roles and functionalities.

The subsequent chapter moves into the area of machine learning, with a particular emphasis on a few algorithms that are useful for time series analysis. This chapter starts by defining machine learning and moves into explaining different learning paradigms. Central to this chapter is the use of machine learning in the context of time series, dedicating sections to Random Forest and XGBoost. These sections delve into the mechanisms, advantages, and real-world applications of these techniques by employing them on the Saida temperature data and the AirPassengers dataset through Python.

The third chapter concludes with an investigation of deep learning and neural networks. The exploration then advances into deep learning, detailing various neural network architectures such as Recurrent Neural Networks and Long Short-Term Memory (LSTM) models and their powerful capabilities for sequence prediction. To demonstrate their application on real data, we have applied these methods to Saida temperature data and AirPassengers data. The chapter wraps up with an analysis of data provided during my internship at Sonalgaz of Saida, where we compared different forecasting methods using Root Mean Squared Errors (RMSE) as the evaluation metric.

Chapter 1

Introduction to Time Series and Artificial Intelligence

A time series refers to a series of data points collected in time sequence, typically at regular intervals such as daily, monthly, or yearly. Highlighted areas of application include astronomy, weather forecasting, economics, medicine, engineering, and environmental science. Understanding temporal relationships in the data is essential for accurate predictions and informed decisions. This chapter presents an overview of time series and the concept of artificial intelligence, detailing its scope, main subfields, and real-world applications. This provides a strong basis for incorporating AI methodologies into time series forecasting, which will be expanded upon in later chapters.

1.1 What is a Time Series?

1.1.1 Definition

Time series are datasets in which observations are recorded sequentially over time, following chronological order. These observations can represent measurements of a single feature or multiple features, depending on the context. Another way to define a time series is as the outcome of a stochastic process, which is a mathematical model that describes the evolution of random variables over time.

In mathematical terms, a time series can be expressed as a collection of random variables

indexed by time. This is often represented using the notation:

$$\{X_t\}_{t \in T} \tag{1.1}$$

Here, the symbol $X(t)$ or X_t refers to the value of the random variable X at a specific time point t. The index set T typically represents the time points at which the observations are made, which could be discrete or continuous depending on the nature of the data.

Time series analysis involves studying these sequences of data points to identify patterns, trends, or dependencies over time.

1.1.2 Examples

Many disciplines, including finance, public administration, macroeconomics, energy, retail, and healthcare, are dominated by time series data. Here are a few examples of such data:

- Daily closing values of a stock index
- Number of weekly infections of a disease
- Weekly series of train accidents
- Rainfall per day
- Sensor data such as temperature measurements per hour
- Population growth per year
- Quarterly earnings of a company over a number of years
- Air Passengers dataset, time series that contains the monthly total number of airline passengers (in thousands) from 1949 to 1960.

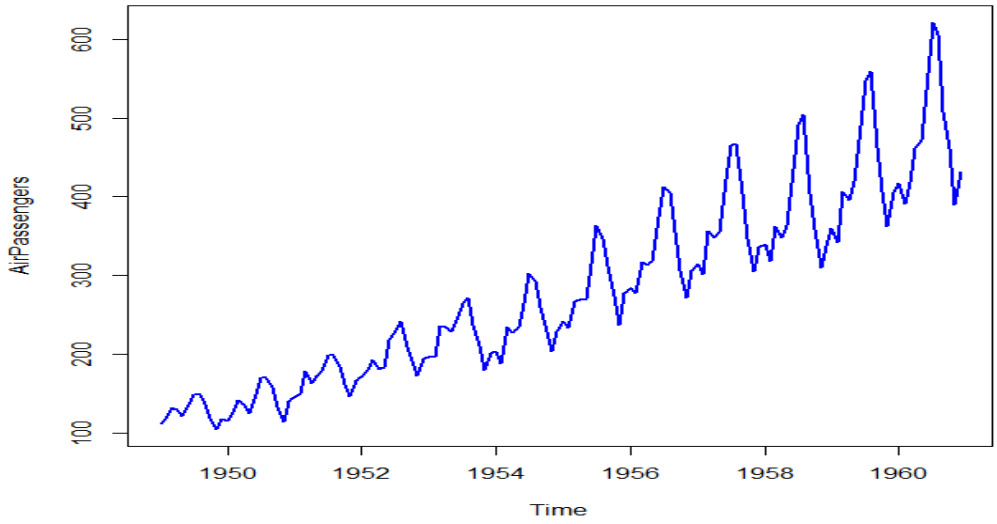

Figure 1.1: The AirPassengers time series data

- Global surface temperature changes over the last 100 years from the GISS (Goddard Institute for Space Studies).

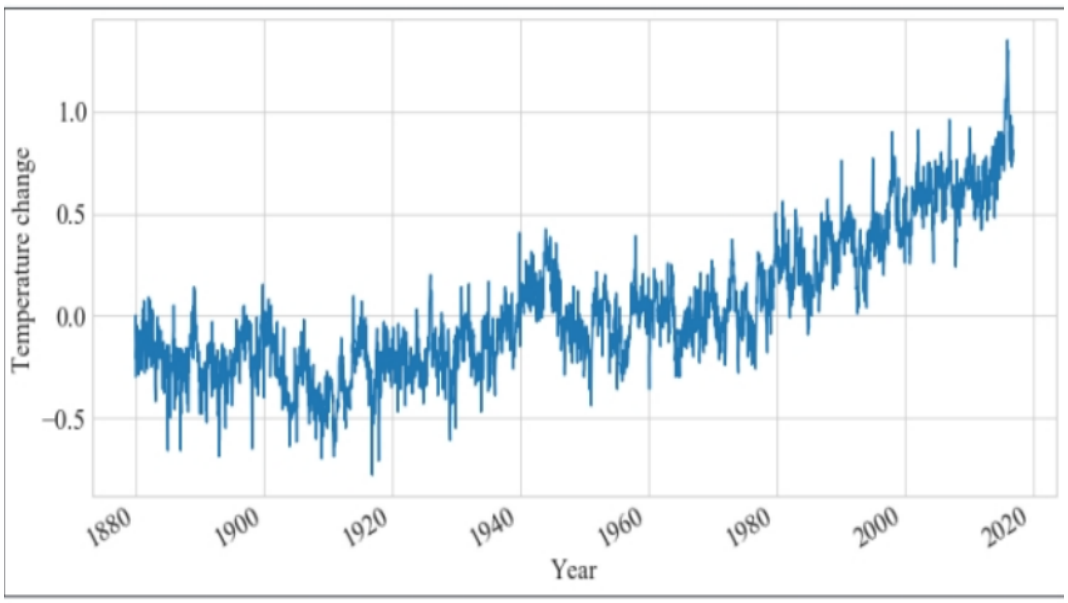

Figure 1.2: GISS surface temperature analysis from 1880 to 2019

These are just a few examples. Any data tracking changes over time is a time series.

1.2 Time series applications

Time series analysis and forecasting have a wide range of applications in various fields, including astronomy, economics, medicine, weather forecasting, and politics. Below are some

examples.

1.2.1 Astronomy

The prediction of meteors and space movements, along with their impact on Earth, is based on astrophysics.

1.2.2 Weather forecasting

Weather forecasting relies on time series analysis to predict atmospheric conditions based on historical and real-time data. Meteorologists use models to analyse temperature, humidity, wind speed, and pressure patterns to improve accuracy and provide reliable forecasts.

1.2.3 Economics

Stock market prediction involves analysing historical data to forecast future stock prices, helping investors make informed decisions. Similarly, exchange rate forecasting examines currency trends to predict fluctuations in foreign exchange markets. Additionally, economic indicators such as inflation rates and interest rates are modelled using time series analysis to assess economic health.

1.2.4 Medicine

Some laboratories use time series datasets to make new discoveries and to develop an understanding of disease progression and clinical trajectories in a wide variety of conditions, including cancer, cystic fibrosis, Alzheimer's, cardiovascular disease, and COVID-19.

1.3 Characteristics of time series

The data of a time series are under the influence of various factors, and they record some fluctuations depending on the effects of the factors in different directions and intensities. In time series models, it is assumed that these fluctuations are due to the simultaneous and combined effects of four different types of movements. These movements, called components (elements) of the time series:

- Long-term movements of the values (trend)
- Seasonal variations (seasonality)
- Cyclical fluctuations
- Random movements

1.3.1 Long-term movements of the values (trend)

A trend T_t represents the long-term evolution of the time series. It reflects the 'average' behaviour of the series. It is most often modelled by a linear or polynomial function of time, for example:

- Linear trend: $T_t = a + bt$
- Quadratic trend: $T_t = a + bt + ct^2$
- Logarithmic trend: $T_t = log(t)$

1.3.2 Seasonal variations (seasonality)

S_t corresponds to a phenomenon that repeats at regular time intervals (periodic). Generally, this is a seasonal phenomenon, hence the term seasonal variations. They linked to the rhythm imposed by weather seasons (agricultural production, gas consumption, sales of sun creams, etc.) or by economic and social activities (festivals, holidays, sales, etc.). S_t is a phenomenon that reproduces in an analogous manner across each successive time interval.

1.3.3 Cyclical fluctuations

These are the constantly recurring fluctuations in the value of the variable, observed at equal periodic intervals, occurring with the economic conjuncture. The value reaches a maximum with an improvement, then regresses and reaches a minimum, then increases again and reaches a maximum. These fluctuations are repeated periodically

1.3.4 Random movements

These are irregular fluctuations that do not occur continuously. They can be caused by natural events such as earthquakes or social events such as wars. They are difficult to predict. We typically consider white noise, that is, a sequence of random variables (ε_t) such that the expectation $\mathbb{E}[\varepsilon_t] = 0$ and the variance is constant, meaning $\mathrm{Var}(\varepsilon_t)$ does not depend on t, and such that $\mathrm{Cov}(\varepsilon_t, \varepsilon_{t+h}) = 0$ for $h \neq 0$. If the variables are Gaussian, the white noise is said to be Gaussian, and in this case, the ε_t are also independent.

Figure 1.3 represents the different components of time series.

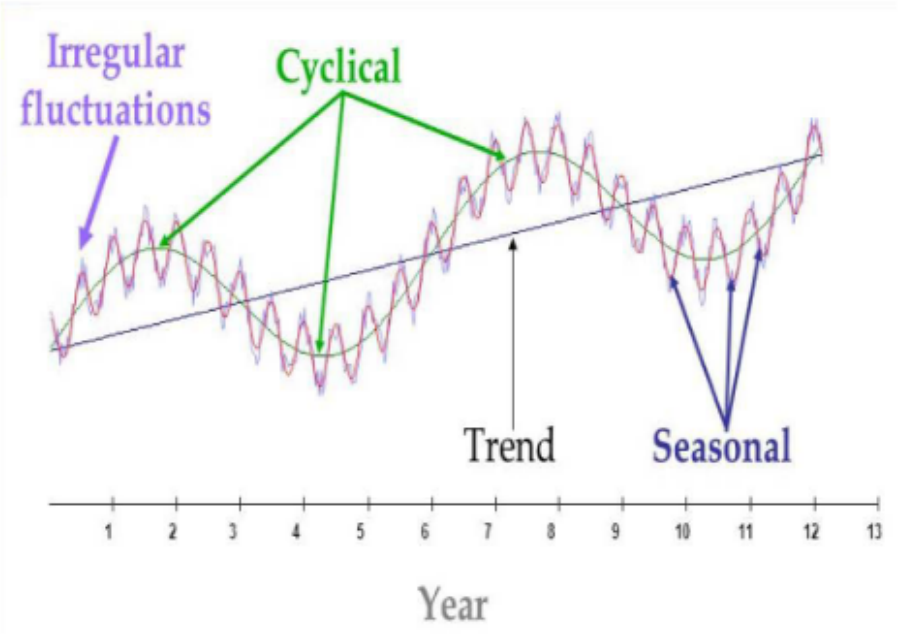

Figure 1.3: Components of time series

1.4 Describing vs. Predicting

We have different goals depending on whether we are interested in understanding a dataset or making predictions. Understanding a dataset, called time series analysis, can help to make better predictions but is not required and can result in a large technical investment in time and expertise not directly aligned with the desired outcome, which is forecasting the future.

1.4.1 Time series analysis

Time series analysis studies data patterns over time to uncover trends, seasonal variations, and causes of fluctuations using statistical methods. It aims to model and explain observed behaviour.

1.4.2 Time series forecasting

Time series forecasting utilizes historical data to predict future trends, based on the assumption that past patterns will continue.

The accuracy of a time series forecasting model is determined by its ability to predict future outcomes. However, this often comes at the expense of explaining why a specific prediction was made, determining confidence intervals, and gaining a deeper understanding of the underlying causes of the problem.

A popular approach for predicting time series data is Holt-Winters exponential smoothing technique; it is characterized by both trends and seasonal fluctuations. This method enhances simple exponential smoothing by including components that account for level, trend, and seasonality. Holt-Winters methods can be categorized into two primary types: additive and multiplicative, based on the approach used to model seasonality.

Additive Holt-Winters model

The additive model is suitable when the seasonal variations are roughly constant over time. The components are updated as follows:

$$
\begin{aligned}
L_t &= \alpha(X_t - S_{t-m}) + (1-\alpha)(L_{t-1} + T_{t-1}) \\
T_t &= \beta(L_t - L_{t-1}) + (1-\beta)T_{t-1} \\
S_t &= \gamma(X_t - L_t) + (1-\gamma)S_{t-m} \\
\hat{X}_{t+h} &= L_t + hT_t + S_{(t-m+h) \bmod m}
\end{aligned}
$$

where:

- X_t is the observed value at time t,
- L_t is the level component,
- T_t is the trend component,

- S_t is the seasonal component,
- m is the seasonal period (e.g., 12 for monthly data with yearly seasonality),
- α, β, γ ($0 < \alpha, \beta, \gamma < 1$) are smoothing parameters.

Multiplicative Holt-Winters Model

The multiplicative model is appropriate when seasonal variations change proportionally with the level of the series:

$$L_t = \alpha \frac{X_t}{S_{t-m}} + (1-\alpha)(L_{t-1} + T_{t-1}) \tag{1.2}$$

$$T_t = \beta(L_t - L_{t-1}) + (1-\beta)T_{t-1} \tag{1.3}$$

$$S_t = \gamma \frac{X_t}{L_t} + (1-\gamma)S_{t-m} \tag{1.4}$$

$$\hat{X}_{t+h} = (L_t + hT_t) \times S_{(t-m+h) \bmod m} \tag{1.5}$$

Proper initialization of L_0, T_0, and S_j ($j = 0, 1, \ldots, m-1$) is critical for model performance and can be done by:

- **Level** L_0: the average of the first season's observations or a weighted average.
- **Trend** T_0: estimated as the average difference between points in the first two seasons divided by m:

$$T_0 = \frac{1}{m}\left(\frac{\sum_{j=1}^{m}(X_{j+m} - X_j)}{m}\right)$$

- **Seasonal components** S_j: computed as deviations from the average level for each season:

$$S_j = X_j - L_0 \quad \text{(additive)} \quad \text{or} \quad S_j = \frac{X_j}{L_0} \quad \text{(multiplicative)}$$

for $j = 1, \ldots, m$.

1.5 Introduction to Artificial Intelligence

1.5.1 What is artificial intelligence?

Artificial intelligence (AI) is a branch of technology focused on the study and development of theories, methodologies, technologies, and application systems designed to emulate, enhance,

and elevate human intelligence. Fundamentally, AI aims to enable machines to reason, think, and mimic intelligent behaviour akin to human capabilities. Originally defined by John McCarthy during the 1956 Dartmouth Conference, AI was framed as the project of creating machines with the capability to replicate human intelligence as accurately as possible. Drawing from the theory of multiple intelligences, which divides human intelligence into categories such as linguistic, logical-mathematical, spatial, bodily-kinaesthetic, musical, interpersonal, and intrapersonal intelligences, AI seeks to mirror or augment these varied dimensions of human intellect. Presently, AI has grown into an interdisciplinary domain that merges computer science, cognitive science, and various other fields, continually advancing to develop more sophisticated and proficient machines.

1.5.2 AI subfields

Artificial intelligence is an attempt to imitate human intelligence; humans can see and analyse scenes, inspiring the development of computer vision. They can learn and interact with their environment, leading to the creation of robotics. They also make decisions in complex situations, which gave rise to planning, scheduling, and optimization. The ability to speak inspired the development of speech technologies, while reading and writing led to Natural Language Processing (NLP). Faced with the challenge of extracting patterns from large datasets in a short time, we turned to machines, giving birth to machine learning. The complexity of the human brain, composed of interconnected neurons responsible for understanding, memory, and learning, inspired Neural Networks. When these networks are built with multiple layers, they enable powerful analysis, marking the rise of deep learning.

In the 1970s and 1980s, researchers aimed to store expert knowledge in computers to make it accessible to non specialists. This led to the emergence of expert systems.

The following figure illustrates the various subfields of AI.

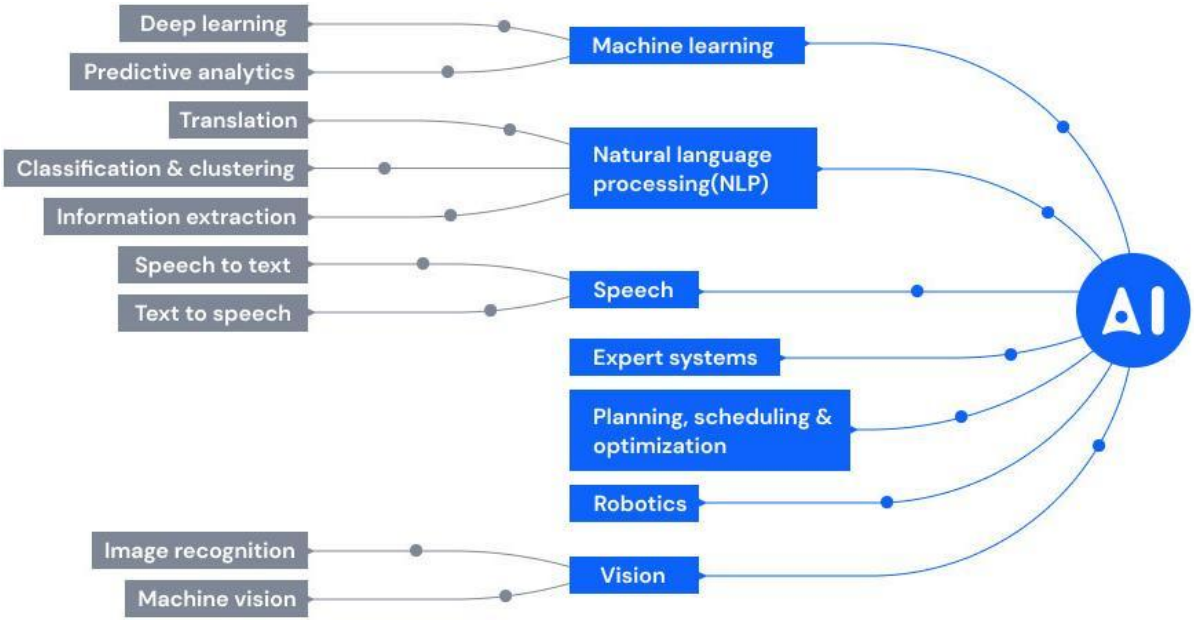

Figure 1.4: Artificial intelligence subfields

1.5.3 The applications of AI

Smart healthcare

We can enable AI to learn professional medical knowledge, memorize a lot of health records, and analyse medical images with computer vision to provide doctors with reliable and efficient assistance. For example, for the medical imaging widely used today, AI can build models based on historical data to analyse medical images and quickly detect lesions, thus improving the efficiency of consultation.

Smart Retail

AI will also revolutionize the retail industry. A typical case is the unmanned supermarket. At Amazon Go, the shopping experience begins by opening the Amazon Go app on your phone and scanning the QR code at the store entrance to enter. Once inside, you can freely browse and pick up the products you want, such as a bag of chips or a bottle of juice. During this process, cameras and sensors record every movement and automatically update your "virtual shopping cart". If you return an item to the shelf, it is removed from the cart. When you are done shopping, you simply leave the store without waiting in checkout lines. Your purchases are automatically calculated, and the amount is deducted from the card linked to your Amazon Prime account. You also receive an electronic receipt through the app. This seamless process relies on artificial intelligence and deep learning algorithms to ensure an accurate and efficient shopping experience.

Unmanned supermarkets face several significant challenges. One major challenge is ensuring accuracy in billing, as the system must accurately record every instance of picking up or returning products. Any errors in this process could lead to incorrect billing for customers or financial losses for the store. In addition, handling multiple customers simultaneously presents another difficulty. When a large number of customers are moving throughout the store, it becomes challenging for the system to differentiate between each customer's behaviour. Furthermore, preventing product theft is crucial, especially in the absence of human intervention. Stores must implement advanced systems to detect and record unauthorized activities accurately to avoid potential losses.

Smart security

The application of AI in the field of security is relatively mature due to the availability of massive amounts of security-related image and video data, which provide a solid foundation for training AI algorithms and models. AI security applications can be classified into civilian and police use. For civilian use, AI is employed in facial recognition, early warning of potential dangers, and home defensive systems, enhancing the safety and security of individuals and properties. For police use, AI is used to identify suspicious targets, analyse vehicles, track suspects, search for and compare criminal suspects in databases, and manage access to key supervised areas. These advancements allow for real time analysis, more efficient threat detection, and improved crime prevention, making AI a critical tool in modern security operations.

Smart home

Smart home refers to a IoT (the Internet of Things) technology-based home ecosystem of hardware, software, and cloud platforms, which provides users with customized life services and a more convenient, comfortable, and safe living environment in a home. The smart housewares are designed to be controlled by voice processing technology, such as adjusting the temperature of the air conditioner, opening the curtains, and controlling the lighting system. Home security relies on computer vision technology, such as unlocking through facial or fingerprint recognition, real time smart camera monitoring, and detection of illegal intrusion into the residence. With the help of machine learning and deep learning, the smart home can build user portraits and make recommendations based on the historical records stored in smart speakers and smart TVs.

Smart driving

The Society of Automotive Engineers (SAE) defines six levels of driving automation, ranging

from level 0 (no automation) to level 5 (full automation). At level 0, vehicles require complete driver control. Levels 1 and 2 introduce driver assistance features, such as adaptive cruise control and lane-keeping assistance. Level 3 allows for conditional automation, where the vehicle can handle certain driving tasks but requires the driver to be ready to intervene. Levels 4 and 5 represent high and full automation, respectively, with Level 5 vehicles capable of operating without any human intervention under all conditions.

As of 2024, only a limited number of commercial passenger vehicles, such as the Audi A8, Tesla models, and Cadillac vehicles, are equipped with level 2 and level 3 Advanced Driver-Assistance Systems (ADAS). The year 2020 marked the emergence of more level 3 models, thanks to advancements in sensors and onboard processors. Vehicles with level 4 and level 5 autonomous driving systems are expected to first be deployed in commercial vehicle platforms within enclosed industrial parks. However, for high-level autonomous driving on passenger vehicle platforms, further optimization in technology, relevant policies, and infrastructure development is required. It is estimated that such passenger vehicles will not be in use on common roads until 2025.

Chapter 2

Machine Learning

Machine learning has become a cornerstone of modern time series forecasting, enabling systems to analyse complex data patterns and generate accurate predictions with minimal explicit programming.

The chapter further delves into the application of machine learning techniques in time series forecasting, with a particular focus on two powerful ensemble methods: Random Forest and XGBoost. For each method, it discusses core concepts such as decision trees, bagging, and boosting and their specific advantages and limitations in forecasting contexts. Supported by empirical examples, the chapter offers both foundational theories and practical strategies for implementing these advanced models. Overall, it aims to provide a comprehensive overview of how machine learning can enhance time series analysis and predictive accuracy. In summary, this chapter offers both theoretical perspectives and practical recommendations for utilizing machine learning in sophisticated time series modelling and forecasting endeavours.

2.1 What is Machine Learning?

Arthur Samuel, an early American leader in the field of computer gaming and artificial intelligence, coined the term "Machine Learning" in 1959 while at IBM. He defined machine learning as the field of study that gives computers the ability to learn without being explicitly programmed. However, there is no universally accepted definition for machine learning, as different authors define the term in various ways.

Machine learning (including its branch, deep learning) is the study of "learning algorithms." In another sense, machine learning refers to programming computers to optimize a performance

criterion using example data or past experience. We have a model defined up to some parameters, and learning is the execution of a computer program to optimize the parameters of the model using training data or past experience. The model may be predictive (making future predictions) or descriptive (gaining insights from data), or it can serve both purposes.

The term "learning" here refers to the process of learning from experience (E) in relation to a specific class of tasks (T) and a performance measure (P), such that performance on tasks (T), as measured by (P), improves with experience (E).

Example: A Robot Driving Learning Problem:

- **Task (T)**: Driving on highways using vision sensors.
- **Performance Measure (P)**: Average distance travelled before an error.
- **Training Experience (E)**: A sequence of images and steering commands recorded while observing a human driver.

2.2 Types of Machine Learning

What is training data in machine learning? It all depends on the type of machine learning model used. Broadly speaking, there are three types of models used in machine learning.

2.2.1 Supervised learning

Supervised learning is a machine learning model that uses labelled training data (structured data) to map a specific feature to a label. In supervised learning, the output is known (e.g., recognizing an image of an apple), and the model is trained on the data from these results. In other words, to train the algorithm to recognize images of apples, it must be provided with images labelled as apples.

The most commonly used supervised learning algorithms today are:

- Linear regression
- Polynomial regression
- K nearest neighbours
- Naive Bayes

- Decision trees

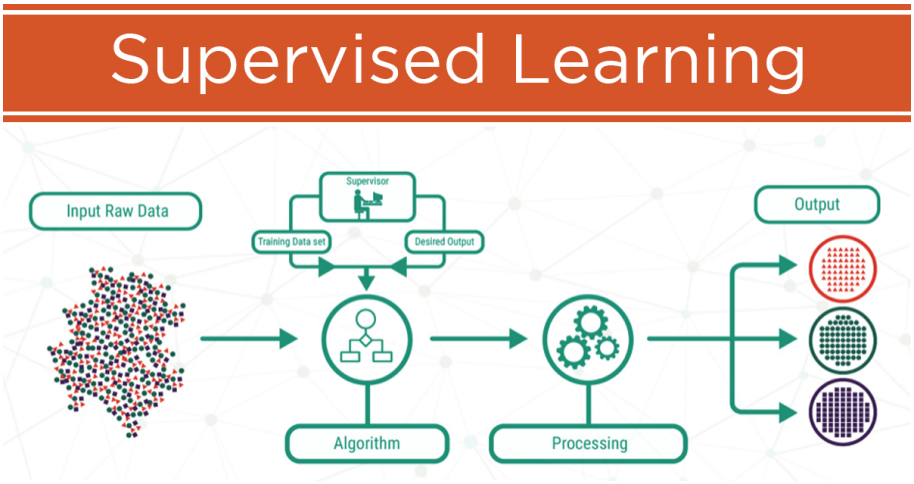

Figure 2.1: Supervised learning

2.2.2 Unsupervised learning

Unsupervised learning is a machine learning model that uses unlabelled data (unstructured data) to learn patterns. Unlike supervised learning, the accuracy of the output is not known in advance. The algorithm learns from the data without human intervention (and therefore unsupervised) and classifies it into groups based on attributes. For example, if the algorithm is given images of apples and bananas, it will work on its own to classify the images that correspond to apples and bananas. Unsupervised learning is effective for descriptive modelling and establishing pattern matching. The most commonly used unsupervised learning algorithms today are:

- Partial averages
- K-means partitioning
- Hierarchical Clustering
- Partial least squares

There is also a mixed approach to machine learning, called "partially supervised learning," in which only some of the data is labelled. In partially supervised learning, the algorithm must figure out how to organize and structure the data to get a known result. For example, the machine learning model is told that the result is a pear, but only some of the training data is labelled as a

pear.

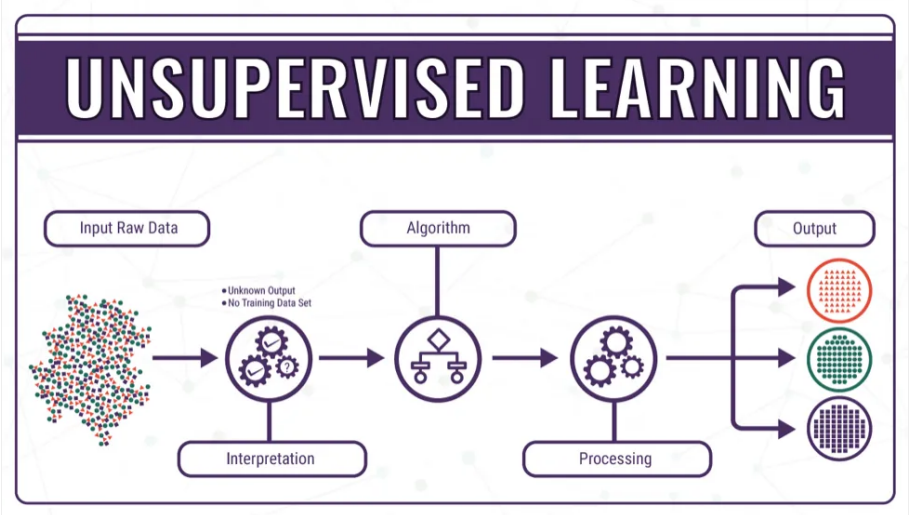

Figure 2.2: Unsupervised learning

2.2.3 Reinforcement learning

Reinforcement learning is a machine learning model that can be described as "learning by doing" through a series of trials and errors. An "agent" learns to perform an empirically defined task (feedback loop) until its performance falls within a desired range. The agent receives positive reinforcement when it performs the task well and negative reinforcement when it gets it wrong. For example, Google researchers taught a reinforcement learning algorithm to play the game of Go. The model had no prior knowledge of the rules of Go. It simply moved pieces randomly and "learned" the best moves. The algorithm was trained.

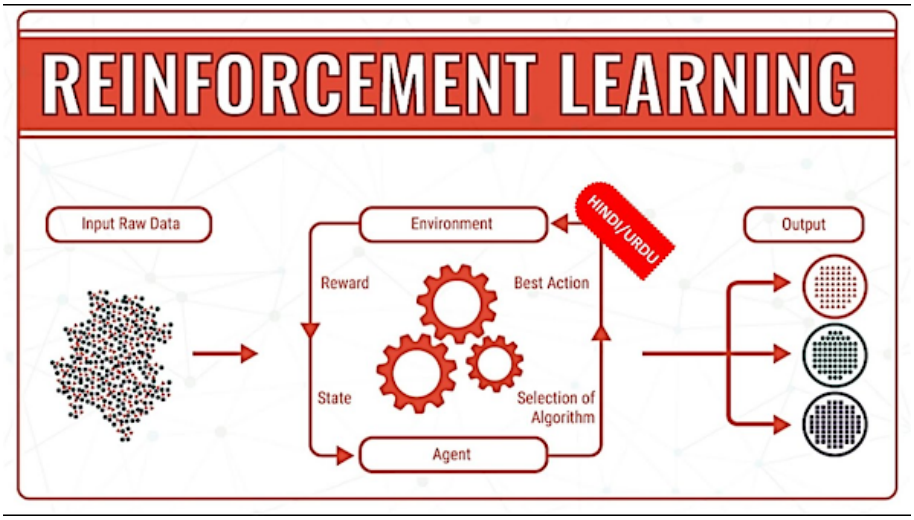

Figure 2.3: Reinforcement learning

2.3 Machine Learning for Time Series

Machine learning offers robust methodologies for modelling and forecasting time series data. In contrast to traditional statistical techniques, machine learning models adeptly capture complex, non-linear relationships and interactions among variables. Within this framework, two prominent models are Random Forest and XGBoost (Extreme Gradient Boosting). However, prior to delving into these specific models, it is essential to comprehend several foundational concepts.

2.4 Random Forest

2.4.1 Decision tree

Decision trees are a technique used in statistics, data mining, and machine learning, and they fall under the category of supervised learning. A decision tree models decision making processes that often resemble human reasoning, making it intuitive and easy to understand.

A decision tree is composed of decision nodes, leaf nodes, and a root node.

- **Root node**: This is the starting point of the tree and represents the entire dataset. From here, the data is split into two or more subsets based on specific features or criteria.
- **Decision nodes**: These are intermediate nodes where the data is further divided based on certain conditions.

- **Leaf nodes**: These represent the final output or prediction. Once the tree reaches a leaf node, no further splitting occurs.

Key processes involved in decision trees include:

- **Splitting**: The process of dividing a node (either the root or a decision node) into two or more sub nodes based on specific criteria or attributes.
- **Pruning**: The process of removing unnecessary branches from the tree. This helps reduce complexity, prevents overfitting, and speeds up decision making by focusing only on the most relevant paths.

Decision trees are widely appreciated for their simplicity, interpretability, and effectiveness in solving both classification and regression problems.

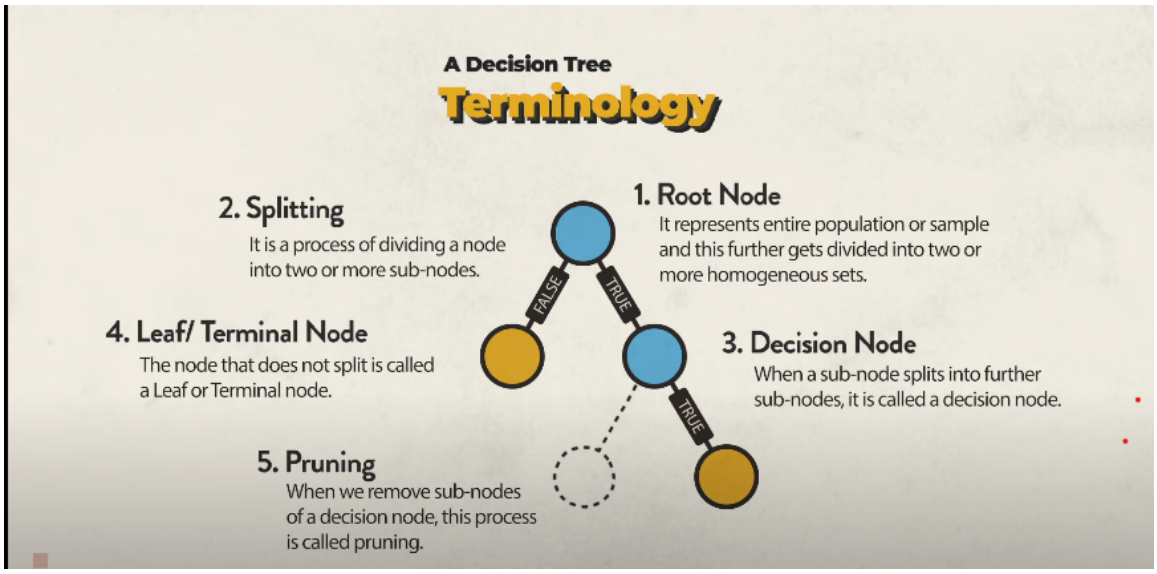

Figure 2.4: Decision tree terminology

Classification trees

Classification trees are a specific kind of decision tree that can handle discrete values for the target variable. The decision tree is based on a categorical variable, with the possible value being "yes" at the first position. It eliminates the outcome following the evaluation of the provided data. A classification or judgment tree is an example of a tree structure that has two or more branches, each dependent on a distinct set of values. An illustration of a rating tree may be seen in a dataset that assesses the decision of playing golf based on the prevailing weather conditions. The forecast serves as a decision node, which is then categorized into three branches: sunny, overcast, and rainy. In this context, a leaf node is activated when specific circumstances are met, and it determines whether the answer is "Yes" or "No" by traversing the tree.

Regression trees

Regression trees are a specific kind of decision tree that can handle target variables with continuous values. The decision tree is designed to handle continuous variables, where the values can be represented as real numbers. The outcome of the tree can consist of numerical values, such as 246. Typically, the creation of a regression tree includes utilizing inputs that consist of a mixture of continuous and discrete variables. Every decision node examines the input to evaluate the value of the variable. The regression tree utilizes a binary recursive division method. At each iteration, the data is divided into parts, which are then further divided into smaller groups when ascending the branch.

A regression tree is capable of predicting the sale values of houses. The outcome of this example is a continuous dependent variable, where multiple constraining factors can include the size of the house in square feet, which remains constant. Additionally, there may exist categorical variables, such as the architectural style of the house, geographical region or location, and others.

2.4.2 Bagging

The bagging method's goal is to reduce the model's large variance. The decision trees are poor in variation and have a low bias. Subsamples are taken from the large dataset. The multiple decision trees are constructed using the training data from each subsample. By slamming the subsampled data into the various decision trees, the risk of each decision tree becoming over-fit

with training data is minimized. To increase the model's productivity, each of the individual decision trees is grown deep using subsampled training data. To comprehend the final forecast, the outcomes of each decision tree are aggregated. The volatility in aggregated results decreases. The precision of the model's estimation in the bagging process is proportional to the number of decision trees used. With substitution, the different sub samples of a sample data set are randomly selected. Each tree's production is highly correlated.

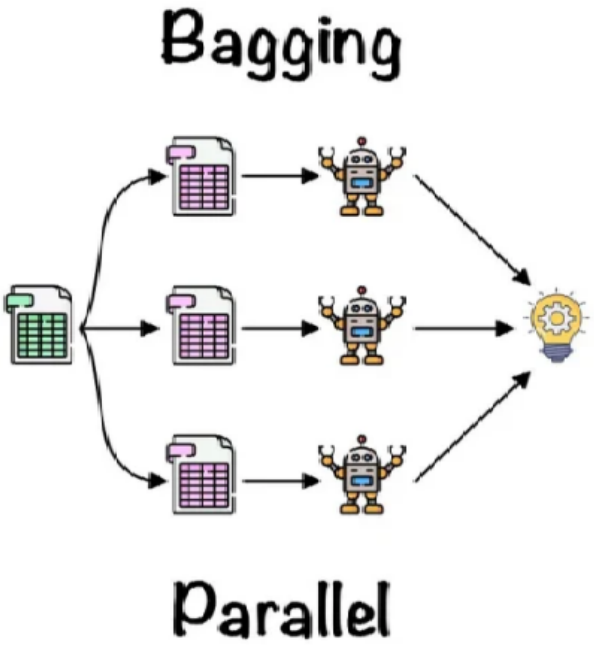

Figure 2.5: The bagging method

2.4.3 Definition of Random Forest

Definition: What is a Random Forest?

Random Forest is a powerful and versatile supervised machine learning algorithm that builds and combines multiple decision trees to form a "forest." It can be applied to both classification and regression problems using programming languages like R and Python. Random Forest is widely used among data scientists, and for good reason, as it offers several advantages over many other algorithms.

The key concept here is the "forest," which signifies that the algorithm relies on an ensemble of decision trees. These trees are created by drawing random samples from the dataset through a

technique called bootstrap sampling and by selecting random subsets of features when building each tree. Each individual tree in the forest functions as a simple decision tree.

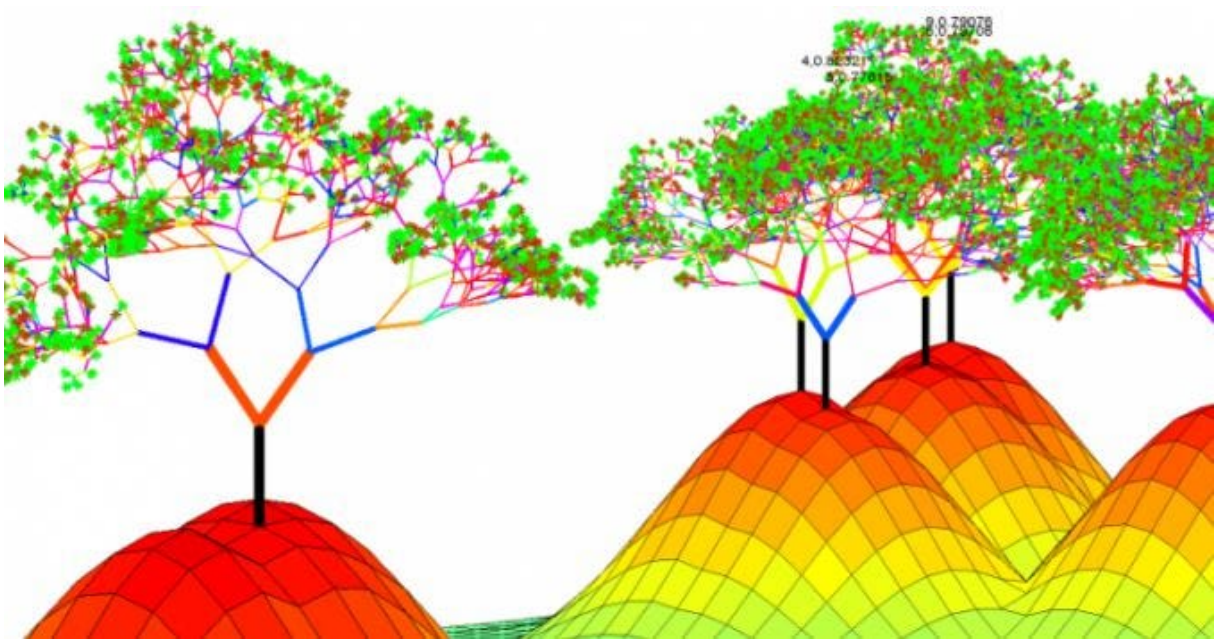

Figure 2.6: Random Forest

2.4.4 Random Forest applications

Some of the applications of the random forest may include:

- **Banking:** Random Forest is used in banking to predict the creditworthiness of a loan applicant. This helps the lending institution make a good decision on whether to give the customer the loan or not. Banks also use the Random Forest algorithm to detect fraudsters.
- **Health care:** Health professionals use Random Forest systems to diagnose patients. Patients are diagnosed by assessing their previous medical history. Past medical records are reviewed to establish the right dosage for the patients.
- **Stock market:** Financial analysts use it to identify potential markets for stocks. It also enables them to identify the behaviour of stocks.
- **E-commerce:** Through Random Forest algorithms, e-commerce vendors can predict the preferences of customers based on past consumption behaviour.

- **Time Series:** Random Forest can be applied in time series forecasting to predict future values of a variable based on historical data. For instance, it is used in predicting electricity demand, weather conditions, and sales forecasting by modelling temporal patterns and trends.

2.4.5 Random Forest for time series forecasting

To make forecasting for time series, the general process involves several key steps:

Data preparation

- Convert your time series data into a suitable format. In Python, the `pandas` library is commonly used to handle time series data efficiently.
- Create **lag features** to capture temporal patterns. Lag features represent previous values of the time series and are used as predictor variables in the model.

Data splitting

- Divide the data into **training** and **testing** sets. The training set consists of historical data, while the testing set includes future data you aim to forecast.
- Ensure that the **chronological order** of the data is preserved to prevent data leakage and maintain the integrity of the time-dependent structure.

Model building

- Fit a Random Forest model to the training data using the `RandomForestRegressor` class from the `sklearn.ensemble` module.
- Specify the **target variable** (the value to be forecasted) and the **predictor variables**, which include lag features and any other relevant information.
- Random Forest is an **ensemble method** that combines multiple decision trees to make robust predictions. Each tree is trained on a **bootstrapped sample** of the data and a **random subset of predictors**.

Prediction

- Use the trained Random Forest model to make predictions on the testing data.
- The model will generate forecasts for future time points based on historical patterns captured during training.

Model evaluation

- Evaluate the model's performance using metrics such as:
  - **Mean Absolute Error (MAE)**
  - **Mean Squared Error (MSE)**
  - **Root Mean Squared Error (RMSE)**
- These metrics are available in the `sklearn.metrics` module and help assess the **accuracy** and **reliability** of the forecasts.

Visualization

- Visualize the original time series data alongside the forecasted values.
- Use the `matplotlib` or `seaborn` libraries to plot the actual and predicted values on the same graph. This provides insight into the model's accuracy and how well it captures **trends** and **seasonality**.

2.4.6 Applications to real data

Example 01

One of the most widely used datasets for such analysis is the Air Passengers dataset, which records the monthly number of international airline passengers from January 1949 to December 1960.
In this study, we apply machine learning techniques, specifically the Random Forest algorithm, to model and forecast passenger traffic.

```
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestRegressor
```

```
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

# --- Load AirPassengers Data ---
data = {
    'Month': pd.date_range(start='1949-01', periods=144, freq='MS'),
    'Passengers': [
        112, 118, 132, 129, 121, 135, 148, 148, 136, 119, 104, 118,
        115, 126, 141, 135, 125, 149, 170, 170, 158, 133, 114, 140,
        145, 150, 178, 163, 172, 178, 199, 199, 184, 162, 146, 166,
        171, 180, 193, 181, 183, 218, 230, 242, 209, 191, 172, 194,
        196, 196, 236, 235, 229, 243, 264, 272, 237, 211, 180, 201,
        204, 188, 235, 227, 234, 264, 302, 293, 259, 229, 203, 229,
        242, 233, 267, 269, 270, 315, 364, 347, 312, 274, 237, 278,
        284, 277, 317, 313, 318, 374, 413, 405, 355, 306, 271, 306,
        315, 301, 356, 348, 355, 422, 465, 467, 404, 347, 305, 336,
        340, 318, 362, 348, 363, 435, 491, 505, 404, 359, 310, 337,
        360, 342, 406, 396, 420, 472, 548, 559, 463, 407, 362, 405,
        417, 391, 419, 461, 472, 535, 622, 606, 508, 461, 390, 432
    ]
}
df = pd.DataFrame(data)
df.set_index('Month', inplace=True)

# --- Create Lag Features ---
lags = 12
for lag in range(1, lags + 1):
    df[f'lag_{lag}'] = df['Passengers'].shift(lag)
df.dropna(inplace=True)

# --- Split Data into Train/Test ---
train_size = int(len(df) * 0.8)
train, test = df[:train_size], df[train_size:]
```

```
# Features and Target
X_train = train.drop(columns=['Passengers'])
y_train = train['Passengers']
X_test = test.drop(columns=['Passengers'])
y_test = test['Passengers']

# --- Train Random Forest ---
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

# --- Predict ---
predictions = rf.predict(X_test)

# --- Evaluation ---
rmse = np.sqrt(mean_squared_error(y_test, predictions))
print(f'RMSE: {rmse:.2f}')

# --- Plot Actual vs Forecast ---
plt.figure(figsize=(12, 6))
plt.plot(df.index, df['Passengers'], label='Original', color='blue')
plt.plot(test.index, predictions, label='Forecast', color='red')
plt.title('Random Forest Forecasting - AirPassengers')
plt.xlabel('Date')
plt.ylabel('Number of Passengers')
plt.legend()
plt.grid(True)
plt.show()
```

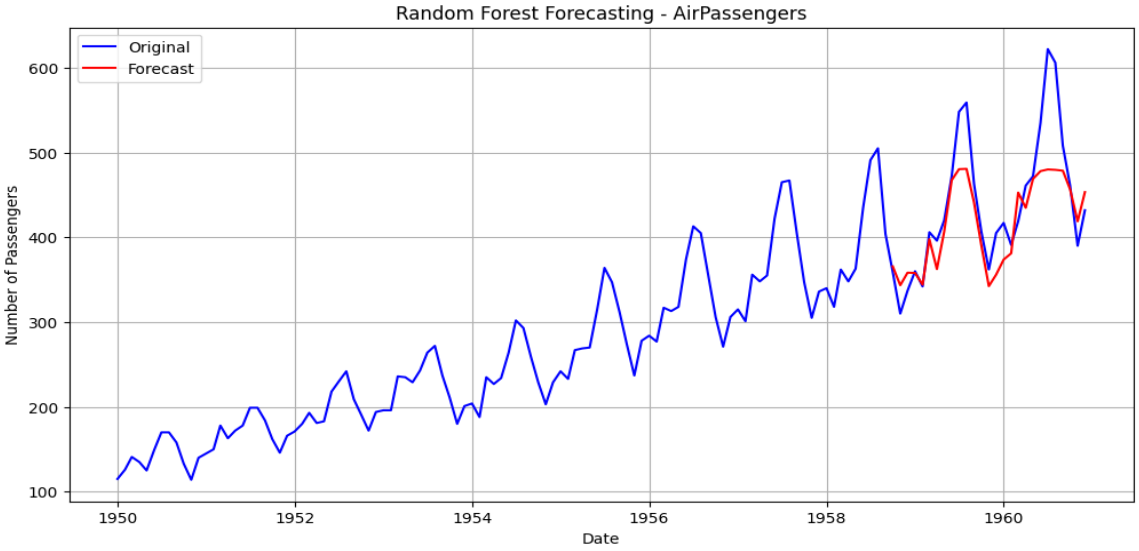

Figure 2.7: Random Forest forecasting for AirPassengers

Example 02

This study focuses on forecasting the temperature patterns in Saida, Algeria, using its historical data.

```
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error
import matplotlib.pyplot as plt

# Load the dataset
df = pd.read_csv("/kaggle/input/correc/corrected_temperature_data.csv
    ", parse_dates=['Datetime'])
df.sort_values('Datetime', inplace=True)
df.set_index('Datetime', inplace=True)
# Create lag features (e.g., Lag 1, Lag 2, Lag 3)
df['Lag_1'] = df['Temperature'].shift(1)
df['Lag_2'] = df['Temperature'].shift(2)
df['Lag_3'] = df['Temperature'].shift(3)
df.dropna(inplace=True)

# --- Split Data into Train/Test ---
```

```
train_size = int(len(df) * 0.8)
train, test = df[:train_size], df[train_size:]
X_train = train[['Lag_1', 'Lag_2', 'Lag_3']]
y_train = train['Temperature']
X_test = test[['Lag_1', 'Lag_2', 'Lag_3']]
y_test = test['Temperature']

# Model Building
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Prediction
predictions = model.predict(X_test)
test['Predicted'] = prediction

# Model Evaluation
mae = mean_absolute_error(y_test, predictions)
mse = mean_squared_error(y_test, predictions)
rmse = np.sqrt(mse)

# --- Plot Actual vs Forecast ---
plt.figure(figsize=(12, 6))
plt.plot(df.index, df['Temperature'], label='Original', color='blue')
plt.plot(test.index, test['Predicted'], label='Forecast', color='red'
    )
plt.title('Random Forest Forecasting - Temperature in Saida, Algeria
    ')
plt.xlabel('Date')
plt.ylabel('Temperature (ÂřC)')
plt.legend()
plt.grid(True)
plt.show()
```

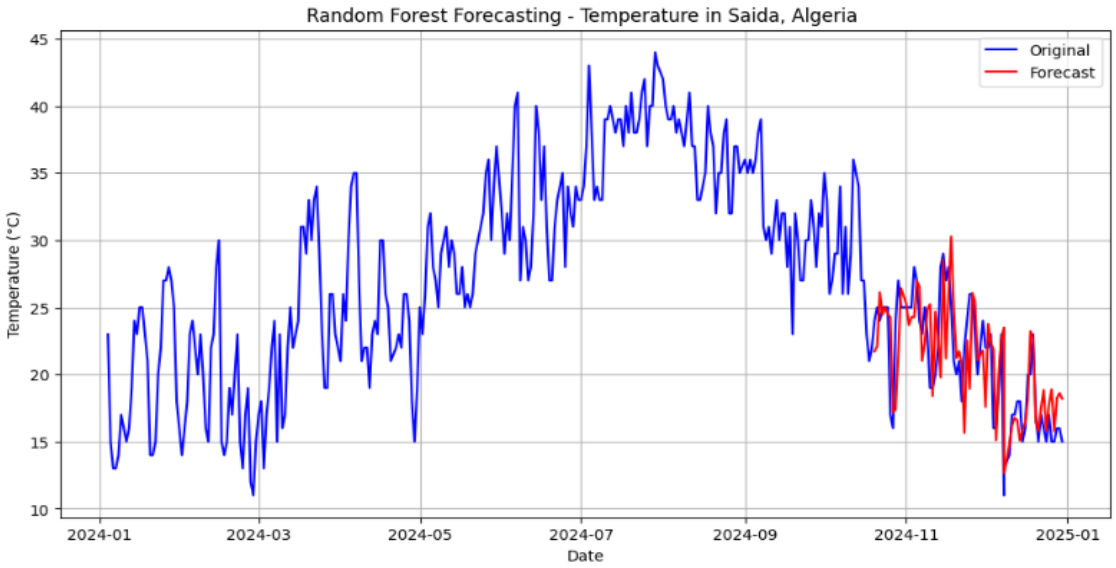

Figure 2.8: Random Forest forecasting for temperature in Saida

2.4.7 Benefits and challenges of Random Forest

Random Forests mitigate the issue of overfitting by training several decision trees on random subsets of data, hence enhancing their ability to generalize to novel data. Random Forests are highly regarded as one of the most efficient and potent techniques in the domain of machine learning. They find extensive use in various applications such as automatic categorization, data forecasting, and supervisory learning.

One advantage of the Random Forest algorithm is its versatility. This approach is applicable to both regression and classification issues. The algorithm might be deemed advantageous as it yields superior outcomes even in the absence of hyperparameter adjustment. Furthermore, they possess a high level of clarity, making them easily comprehensible. Additionally, their number is quite limited.

The primary constraint of Random Forests is the extensive number of trees, resulting in a pro-longed training time that renders it sluggish and inefficient for real-time predictions. Typically, these algorithms exhibit rapid training speed but significantly slower prediction speed after training. While the Random Forest algorithm is often efficient in real-world applications, there may be situations where runtime performance is crucial and alternative approaches may be more desirable.

2.5 XGBoost

2.5.1 Definition

XGBoost is an ensemble ML (Machine Learning) algorithm that is dependent on decision trees and employs a gradient boosting method. The XGBoost algorithm was created at the University of Washington as part of a research project. It was introduced in 2016 by Tianqi Chen and Carlos Guestrin. It is commonly used for its ability to model newer attributes and classify marks. The XGBoost algorithm's use has exploded in popularity due to its implementations in tabular and structured datasets.

2.5.2 Boosting

The trees are constructed sequentially, with each following tree attempting to minimize the errors of the preceding tree. Each tree builds on the knowledge gained by its predecessors and updates the residual errors. As a result, the tree that follows will benefit from a modified version of the residuals. In boosting, the foundation learners are weak learners with a strong bias and predictive ability that is only a tad stronger than random guessing. Each of these weak learners contributes critical knowledge for estimation, allowing the boosting technique to successfully combine these weak learners to create a strong learner. The final powerful learner reduces both bias and variation.

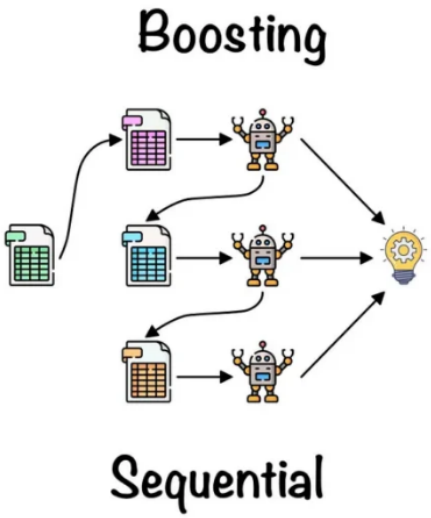

Figure 2.9: The boosting method

2.5.3 Gradient boosting

Boosting algorithms combine weak learners, i.e., learners slightly better than random, into a strong learner in an iterative way. Gradient boosting is a boosting-like algorithm for regression. Given a training dataset $D = \{x_i, y_i\}_{i=1}^{N}$

1. Initialize model with a constant value:

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma)$$

where y_i refers to the observed values and γ refers to the predicted values.
The first step is creating an initial constant value prediction F_0. L is the loss function, and it is squared loss in our regression case.

$$L(y_i, \gamma) = (y_i - \gamma)^2$$

We want to find the value of γ that minimizes the total loss:

$$\sum_{i=1}^{n} (y_i - \gamma)^2$$

Differentiate with respect to γ:

$$\frac{d}{d\gamma} \sum_{i=1}^{n} (y_i - \gamma)^2 = \sum_{i=1}^{n} -2(y_i - \gamma) = -2 \sum_{i=1}^{n} (y_i - \gamma)$$

Set the derivative to zero:

$$\sum_{i=1}^{n} (y_i - \gamma) = 0 \Rightarrow \sum y_i - n\gamma = 0 \Rightarrow \gamma = \frac{1}{n} \sum_{i=1}^{n} y_i$$

Thus, for squared loss, the minimizing value is the **mean** of the y_i's:

$$\gamma = \frac{1}{n} \sum_{i=1}^{n} y_i$$

2. for $m = 1$ to M:

The whole step 2 processes from a to d are iterated M times. M denotes the number of trees we are creating, and the small m represents the index of each tree.

(a) Compute residuals r_{im}:

$$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F=F_{m-1}(x)} \qquad for\ \ i = 1, \ldots, n$$

We are calculating residuals r_{im} by taking a derivative of the loss function with respect to the previous prediction F_{m-1} and multiplying it by 1. r_{im} is computed for each single sample i.

(b) Train regression tree with features x against r and create terminal node reasons R_{jm} for $j = 1, \ldots, J_m$.

j represents a terminal node (i.e. leave) in the tree, m denotes the tree index, and capital J means the total number of leaves.

(c) Compute

$$\gamma_{jm} = \arg\min_{\gamma} \sum_{x_i \in R_{jm}} L(y_i, F_{m-1}(x_i) + \gamma)$$

We are searching for γ_{jm} that minimizes the loss function on each terminal node j. Let's plug-in the loss function into the equation.

$$\gamma_{jm} = \arg\min_{\gamma} \sum_{x_i \in R_{jm}} L(y_i, F_{m-1}(x_i) + \gamma)$$

$$= \arg\min_{\gamma} \sum_{x_i \in R_{jm}} (y_i - F_{m-1}(x_i) - \gamma)^2$$

$$\frac{\partial}{\partial \gamma} \sum_{x_i \in R_{jm}} (y_i - F_{m-1}(x_i) - \gamma)^2 = 0$$

$$-2 \sum_{x_i \in R_{jm}} (y_i - F_{m-1}(x_i) - \gamma) = 0$$

$$n_j \gamma = \sum_{x_i \in R_{jm}} (y_i - F_{m-1}(x_i))$$

$$\gamma = \frac{1}{n_j} \sum_{x_i \in R_{jm}} r_{im}$$

(d) Update the model:

$$F_m(x) = F_{m-1}(x) + \nu \sum_{j=1}^{J_m} \gamma_{jm} \mathbf{1}_{(x \in R_{jm})}$$

The learning rate $\nu \in (0, 1]$ controls how much each new tree γ contributes to the model's prediction F_m. The effects of smaller ν is slower and more stable learning, reduces the risk of overfitting and requires more boosting iterations (more trees).

2.5.4 Mechanisms and working principles of XGBoost

- **Given data and initial predictions:**
  We are given input features (X) and a target feature (Y). Now we start with a default set of predictions (by default set to 0.5 in regression, but you can start from other values as well).

- **Calculating pseudo residuals:**
  Calculate the (pseudo) residuals by subtracting Y from default initial predictions.

- **Building XGBoost trees:**
  Each tree starts out as a single leaf

  – Start with all residuals in the same leaf.

  – Calculate the similarity score using the following formula:

$$\text{Similarity Score} = \frac{\left(\sum_{i=1}^{n} r_i\right)^2}{n + \lambda}$$

  where:

  a) r_i are the residuals,

  b) n length of residuals,

  c) λ is the regularization parameter.

  – Finding the best splitting feature and values:

  a) Iterate through each input feature.

  b) Sort feature values in ascending order.

  c) For every pair of consecutive values, compute the midpoint:

$$\text{Threshold} = \frac{v_1 + v_2}{2}$$

  d) Use the threshold to split the samples:

  · Samples with values less than the threshold go to the **left node**.

- Samples with values greater than or equal to the threshold go to the **right node**.

e) Repeat this for all features and all value pairs.

– Calculating Gain:

For each split, compute the gain as:

$$\text{Gain} = \text{Similarity}_{\text{left}} + \text{Similarity}_{\text{right}} - \text{Similarity}_{\text{parent}}$$

Choose the split with the **highest gain**, as it most effectively improves the model.

– Creating the Tree:

We continue repeating Step 3 (Splitting) and Step 4 (Calculating Gain) recursively on the child nodes of the tree. This process is repeated until one of the stopping conditions is met. These conditions include reaching the maximum tree depth, which by default is set to 6, or encountering a leaf node that contains too few residuals, with the default minimum being 1. Additionally, another stopping condition is based on a threshold for residuals determined by a concept called cover, which will be discussed in a later section. Once one of these stopping criteria is met, the construction of the decision tree is complete, resulting in the final tree structure.

– Pruning the Tree:

a) We use the hyperparameter γ (gamma) to prune the tree.

b) Pruning is performed in a bottom-up manner.

c) For each parent node:

- If the gain of the node is less than gamma (gain $- \gamma < 0$), we prune its children.
- If pruning happens, we continue checking upward.
- If pruning does not happen at a node, we stop going further up from there.

d) **Note**: Setting $\gamma = 0$ does **not** turn off pruning entirely because some nodes can still have negative gain values, leading to pruning where gain $- \gamma < 0$.

– Calculating the Output Value:

a) After building and pruning the tree, we compute the final prediction values at each leaf node.

b) For regression, the output value for a leaf node is given by the formula:

$$\text{Output Value} = \frac{\sum \text{residuals}}{n + \lambda}$$

where:

- $\sum$ residuals: sum of residuals in the node
- n: number of samples in the node
- λ: regularization parameter

c) We compute this value for every terminal (leaf) node to produce the predictions.

- **Updating the residuals and getting the output:**
  Now that we have a tree capable of predicting residuals, we can update our initial default vector originally set to 0.5 predictions by adding these residuals to it (multiplied by a learning rate, of course). We avoid directly jumping to the final prediction to prevent overfitting.

2.5.5 Advantages of XGBoost for time series forecasting

XGBoost offers several advantages that make it an excellent choice for time series forecasting:

- **Handling non-linear relationships:**
  XGBoost can capture complex non-linear relationships between input features and the target variable, making it suitable for time-series data with intricate patterns.
- **Feature importance:**
  XGBoost provides insights into the importance of different features, allowing analysts to identify the most influential factors in the time-series data.
- **Regularization:**
  XGBoost incorporates regularization techniques to prevent overfitting, ensuring that the model generalizes well to unseen data.
- **Handling missing values and outliers:**
  XGBoost can handle missing values and outliers in the data, reducing the need for extensive data preprocessing.

2.5.6 Applying XGBoost to time series forecasting

- **Data Preparation**
  - **Formatting:** Use the `pandas` library to organize the time series data into a supervised learning format.
  - **Lag features:** Create lag features (e.g., $y_{t-1}, y_{t-2}, \dots$) to use previous values as inputs to predict future values.
  - **Date features:** Extract date components such as day of the week, month, or hour to capture seasonality or trends.
- **Data splitting**
  - **Chronological split:** Split the dataset into training and testing sets while preserving the order of time.
  - **Avoid leakage:** Ensure future data is not used to predict past data.
- **Model building**
  - Use `XGBRegressor` from the `xgboost` Python library.
  - Input features should include lagged variables and date-based features.
  - Configure hyperparameters such as `max_depth`, `learning_rate`, and `n_estimators`.
  - XGBoost automatically handles missing values and applies regularization to reduce overfitting.
- **Prediction**
  - Use the trained model to predict future values on the test set.
  - For multi-step forecasting, use a recursive strategy or extend the feature set dynamically.
- **Model evaluation**
  - Use metrics such as:
    a) Mean Absolute Error (MAE)
    b) Mean Squared Error (MSE)

c) Root Mean Squared Error (RMSE)

– These are available in the `sklearn.metrics` module.

- **Visualization**

  – Plot actual vs. predicted values using `matplotlib` or `seaborn`.

  – Visualization helps identify how well the model captures trends and anomalies.

The following figure gives a summary about the tree-based algorithms.

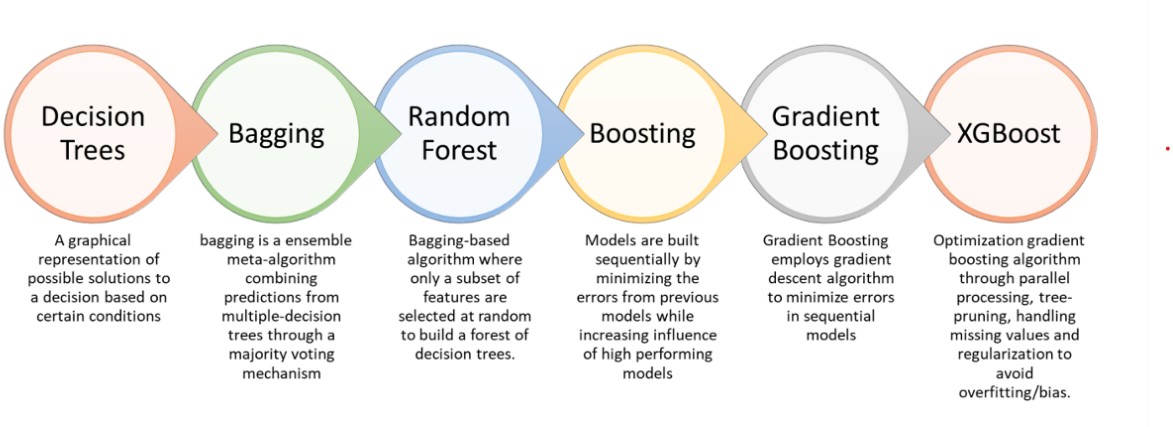

Figure 2.10: Tree-based algorithms

2.5.7 Applications

The dataset from the earlier phase is retained for this step to preserve consistency in data processing and model evaluation.

Example 1: AirPassengers data

```
import pandas as pd
import numpy as np
import xgboost as xgb
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

# Load the dataset
```

```
date_range = pd.date_range(start='1949-01', periods=len(data), freq='
    MS')
df = pd.DataFrame({'Month': date_range, 'Passengers': data})
df.set_index('Month', inplace=True)

# Create lag features
lags = 12
for lag in range(1, lags + 1):
    df[f'lag_{lag}'] = df['Passengers'].shift(lag)
df.dropna(inplace=True)

# Split into training and testing sets
train_size = int(len(df) * 0.8)
train, test = df[:train_size], df[train_size:]
X_train = train.drop(columns=['Passengers'])
y_train = train['Passengers']
X_test = test.drop(columns=['Passengers'])
y_test = test['Passengers']

# Model training
model = xgb.XGBRegressor(
    n_estimators=1000,
    learning_rate=0.01,
    max_depth=5,
    objective='reg:squarederror',
    early_stopping_rounds=50,
    eval_metric='rmse')
model.fit(
    X_train, y_train,
    eval_set=[(X_train, y_train), (X_test, y_test)],
    verbose=100)

# Prediction
predictions = model.predict(X_test)
```

```
# Evaluation
rmse = np.sqrt(mean_squared_error(y_test, predictions))
print(f'RMSE: {rmse:.2f}')

# --- Plot Actual vs Forecast ---
plt.figure(figsize=(12, 6))
plt.plot(df.index, df['Passengers'], label='Original', color='blue')
plt.plot(test.index, predictions, label='Forecast', color='red')
plt.title('XGBoost Forecast - AirPassengers')
plt.xlabel('Date')
plt.ylabel('Number of Passengers')
plt.legend()
plt.grid(True)
plt.show()
```

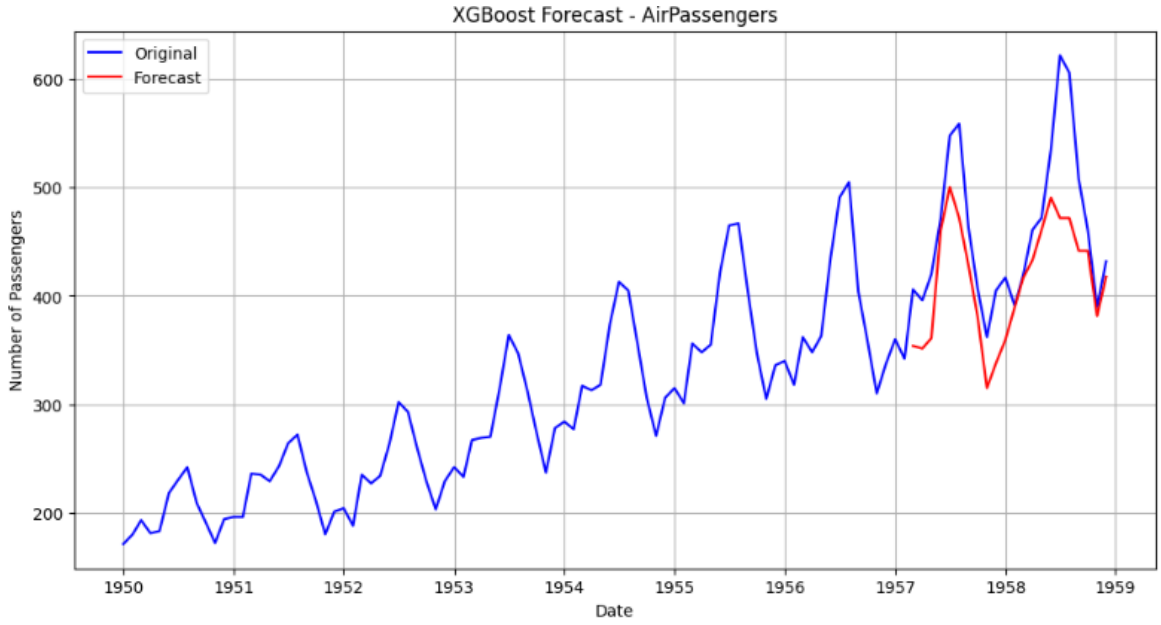

Figure 2.11: XGBoost forecasting for AirPassengers

Example 2: Saida temperature data

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
import seaborn as sns
import xgboost as xgb
from sklearn.metrics import mean_squared_error
color_pal = sns.color_palette()
plt.style.use('fivethirtyeight')
# Load the dataset
df =pd.read_csv('../input/correc/corrected_temperature_data.csv')
df = df.set_index('Datetime')
df.index = pd.to_datetime(df.index)
df.dropna(inplace=True)

# Create lag features
def create_features(df):
    df = df.copy()
    df['dayofweek'] = df.index.dayofweek
    df['month'] = df.index.month
    df['dayofyear'] = df.index.dayofyear
    return df
df = create_features(df)

def add_lags(df, target_col='Temperature'):
    target_map = df[target_col].to_dict()
    df['lag1'] = (df.index - pd.Timedelta(days=1)).map(target_map)
    df['lag2'] = (df.index - pd.Timedelta(days=2)).map(target_map)
    df['lag3'] = (df.index - pd.Timedelta(days=3)).map(target_map)
    return df

df = add_lags(df)
# Split into training and testing sets
train_size = int(len(df) * 0.8)
train, test = df[:train_size], df[train_size:]
FEATURES = ['dayofweek', 'month', 'dayofyear', 'lag1', 'lag2', 'lag3'
   ]
TARGET = 'Temperature'
```

```
X_train = train[FEATURES]
y_train = train[TARGET]
X_test = test[FEATURES]
y_test = test[TARGET]

# Model training
model = xgb.XGBRegressor(
    n_estimators=1000,
    learning_rate=0.01,
    max_depth=5,
    objective='reg:squarederror',
    early_stopping_rounds=50,
    eval_metric='rmse'
)

model.fit(
    X_train, y_train,
    eval_set=[(X_train, y_train), (X_test, y_test)],
    verbose=100
)
#Prediction
y_pred = model.predict(X_test)
#Evaluation
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print(f"Test RMSE: {rmse:.2f}")

# --- Plot Actual vs Forecast ---
plt.figure(figsize=(20, 6))
plt.plot(df.index, df['Temperature'], label='Original', color='blue')
plt.plot(test.index,y_pred, label='Forecast', color='red')
plt.title('XGBoost Forecast -Temperature in Saida, Algeria')
plt.xlabel('Date')
plt.ylabel('Temperature (C)')
plt.legend()
```

```
plt.grid(True)
plt.show()
```

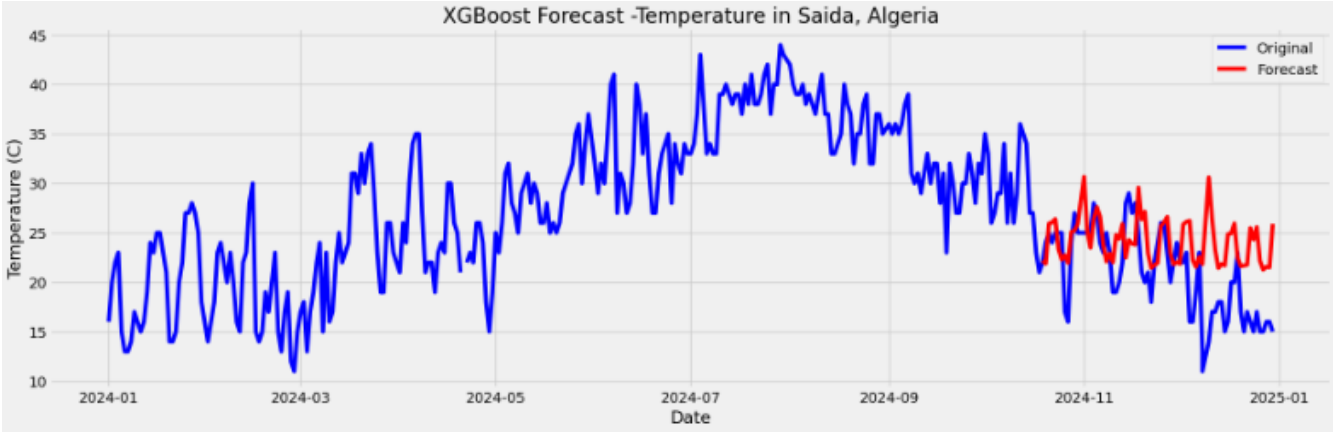

Figure 2.12: XGBoost forecasting for temperature in Saida

Chapter 3

Deep Learning

In this third chapter, the foundations of deep learning are presented with an emphasis on its structure, components, and importance in time series forecasting. The chapter begins by defining deep learning and elucidating its core mechanism through artificial neural networks (ANNs).

It subsequently elaborates on the essential building blocks of neural networks, which include neurons, synapses with associated weights, biases, and activation functions, all of which are crucial for enabling networks to learn from data.

The discourse extends to various ANN architectures and the distinct types of neural networks, such as Feed Forward Neural Networks (FFNs), Generative Adversarial Networks (GANs), Convolutional Neural Networks (CNNs), Autoencoders, Recurrent Neural Networks (RNNs), and Long Short-Term Memory (LSTM) networks.

Particular emphasis is placed on RNNs and LSTMs, detailing their architectures, types, functional gates (input and output), and their capability to manage sequential and temporal data, highlighting their efficacy in time series forecasting tasks. Additionally, their real-world applications are discussed.

The chapter concludes with a comparison of deep learning forecasting techniques against traditional statistical models, such as Holt Winters, as well as machine learning methods like Random Forest and XGBoost, utilizing real datasets including AirPassengers, Saida temperature, and Sonalgas Saida data. This analysis underscores the superior predictive capabilities, adaptability, and practical significance of deep learning models in capturing intricate patterns within time-dependent data.

3.1 Definition and Overview

Deep learning is a subset of machine learning that employs deep neural networks models composed of multiple layers of artificial neurons to learn complex patterns and representations from data. Unlike traditional machine learning techniques that rely on manually engineered features, deep learning utilizes representation learning, allowing models to extract features automatically from raw inputs such as images, audio, or text. This method is inspired by the architecture of the human brain, particularly the way neurons connect and communicate. Originally rooted in early neuroscientific studies by Santiago Ramon y Cajal, modern deep learning has driven major advances in speech recognition, computer vision, and natural language processing. A notable example of its capabilities is AlphaGo, a program developed by DeepMind, which used deep learning to defeat a world champion in the game of Go in 2016 (Silver et al., 2016). Deep learning is typically unsupervised or semi-supervised, and it excels with large scale datasets. It continues to be a key driver of innovation in fields such as autonomous driving, healthcare, and language translation.

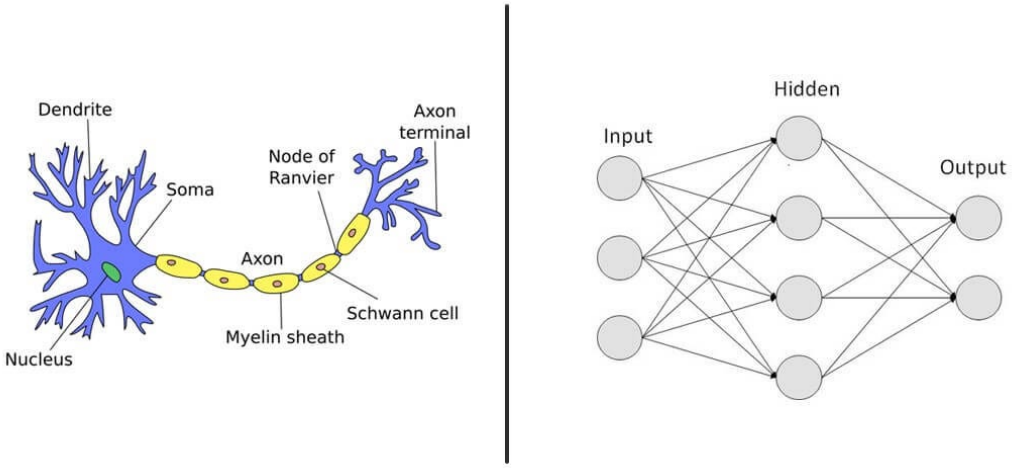

Figure 3.1: Artificial Neural Network v/s Biological Neural Network

3.2 Fundamentals of Artificial Neural Networks

Artificial Neural Networks (ANNs) are powerful computing techniques that mimic functions of the human brain to solve complex problems arising from big and messy data. it consists of neurons and synapses organized into layers.

ANN can have millions of neurons connected into one system or many hidden layers (Deep

ANN) which makes it extremely successful at analysing and even memorizing various information.

3.3 Components of Neural Networks

While neural networks exhibit diverse architectures, they consistently comprise a core set of elements: neurons, synapses, weights, biases, and activation functions.

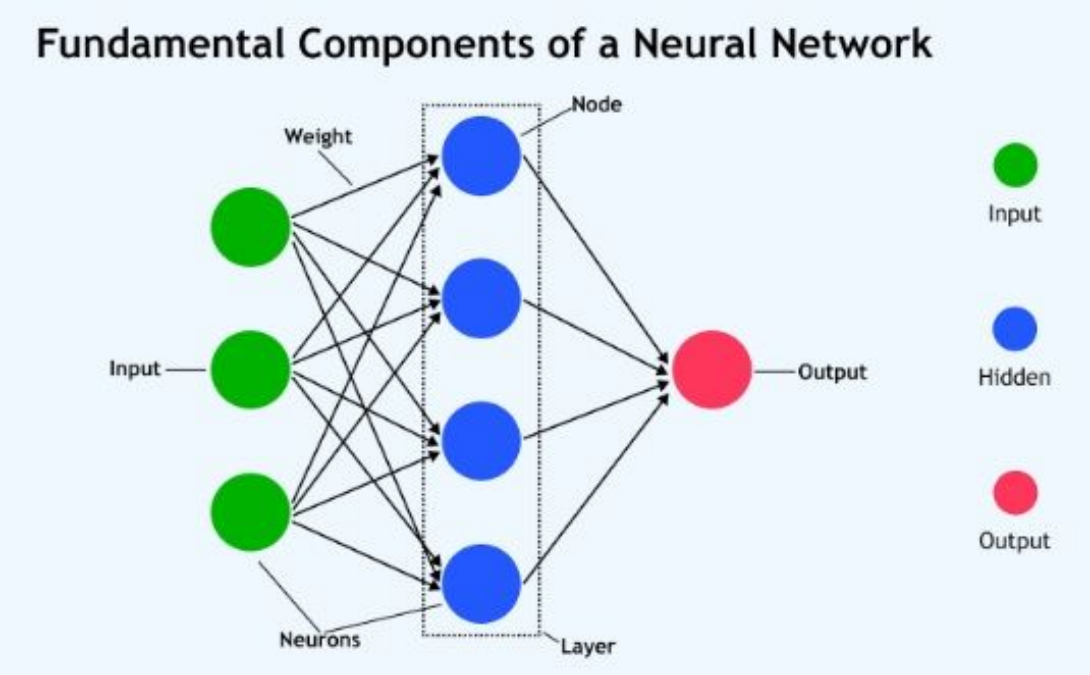

Figure 3.2: Components of a neural network

3.3.1 Neurons

A neuron, also known as a node, is the basic unit of a neural network. It receives information from other neurons or the external environment, performs a simple mathematical operation on the input, and then passes the result forward to other neurons. Neurons in a neural network are typically divided into three groups: input neurons receive data from the outside world, such as image pixels or numerical values; hidden neurons are responsible for processing this information through multiple layers, learning patterns and features in the data; finally, output neurons generate the final result or prediction based on the information processed by the hidden layers.

In large neural networks, neurons are arranged in a layered structure called the network architecture. This architecture includes an input layer that takes in the data, one or more hidden layers that perform computations and learn patterns, and an output layer that produces the result. Each neuron in a layer is usually connected to many others in the next layer, and these connections are associated with weights that influence how much one neuron's output affects another.

3.3.2 Synapses and weights

In neural networks, a synapse represents the connection between two neurons, similar to an electrical cable that transmits signals. Each synapse is associated with a weight, which determines the importance or strength of the signal it carries. These weights play a critical role in adjusting how input information is transformed as it passes from one neuron to the next. When multiple inputs are fed into a neuron, those with higher weights have a stronger influence on the neuron's output. In contrast, inputs with lower weights may have little to no impact. Therefore, the behaviour of the entire network is heavily influenced by the matrix of weights, which essentially governs how the network learns and makes decisions. During training, these weights are adjusted to minimize the difference between the predicted and actual outputs, a process known as optimization.

3.3.3 Bias

Bias is a small but important value added to the weighted sum of inputs before the result is passed through the activation function. Its purpose is to give the neuron more flexibility in learning. Without a bias, the neuron's output would always be zero when all inputs are zero, which can limit the network's ability to learn certain patterns. The bias allows the model to shift the activation function left or right, enabling it to better fit the data. In simple terms, just as weights determine how much influence each input has, the bias determines whether the neuron should activate even when inputs are weak. During training, the bias, like the weights, is adjusted to help minimize the loss function and improve the model's performance.

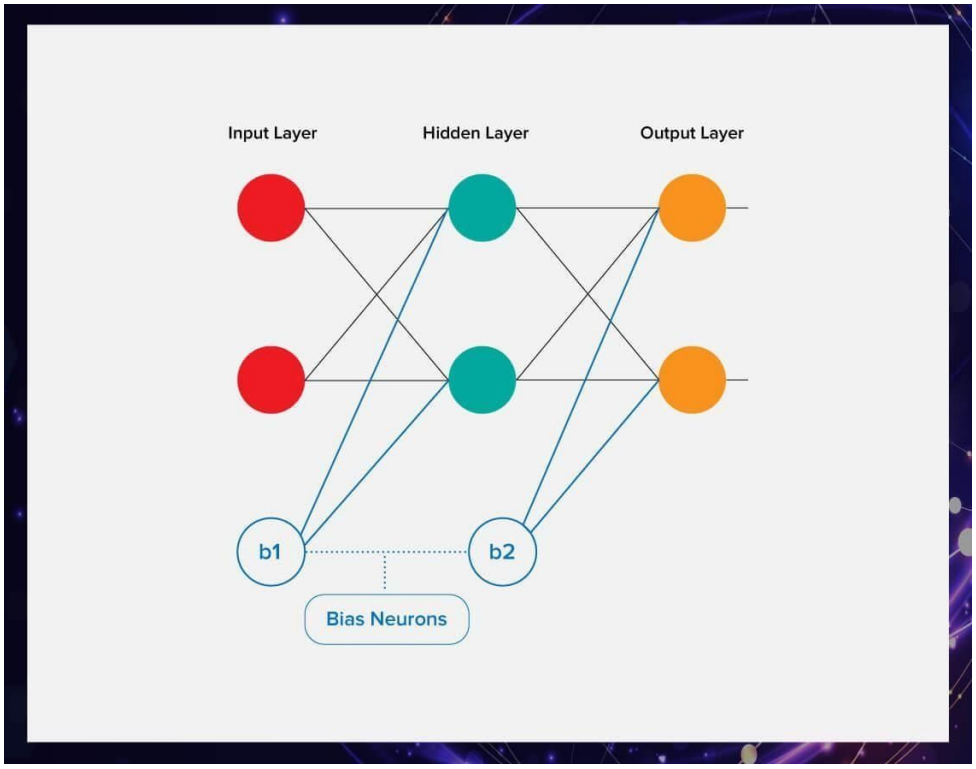

Figure 3.3: Neural network with bias neurons

3.3.4 Activation functions

In deep learning, activation functions play a crucial role in determining the output of neural network layers. They introduce non-linearity into the network, allowing it to learn and perform complex tasks. The figure represents four common activation functions: Sigmoid, Tanh, ReLU, and leaky ReLU.

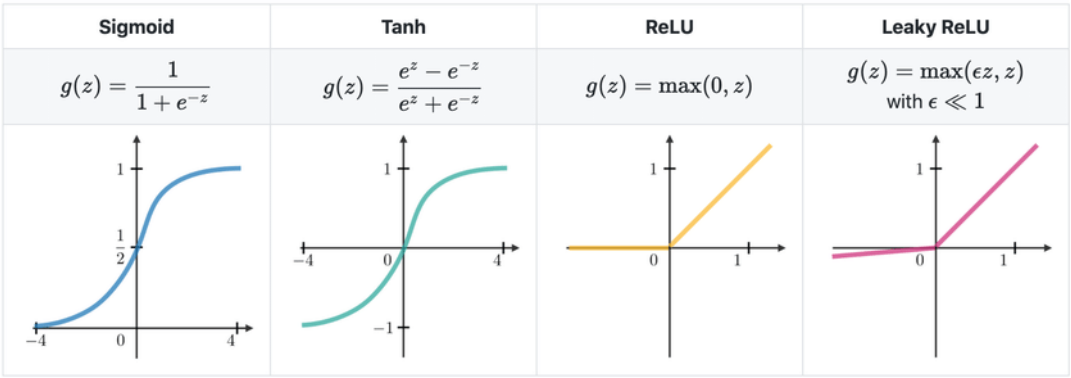

Figure 3.4: Activation functions for artificial neural networks

3.4 ANN Architectures

Feed forward multilayer perceptron MLP is the most used in ANN architectures. Figure below demonstrates the architecture of a simple feedforward MLP where P identifies the input layer, next one or more hidden layers, which is followed by the output layer containing the fitted values. The feed forward MLP networks is evaluated in two stages.

First, in the feedforward stage information comes from the left and each unit evaluates its activation function f. The results (output) are transmitted to the units connected to the right.

The second stage involves the backpropagation (BP) step and is used for training the neural network using gradient descent algorithm in which the network parameters are moved along the negative of the gradient of the performance function. The process consists of running the whole network backward and adjusting the weights (and error) in the hidden layer. The feedforward and backward steps are repeated several times, called epochs. The algorithm stops when the value of the loss (error) function has become sufficiently small.

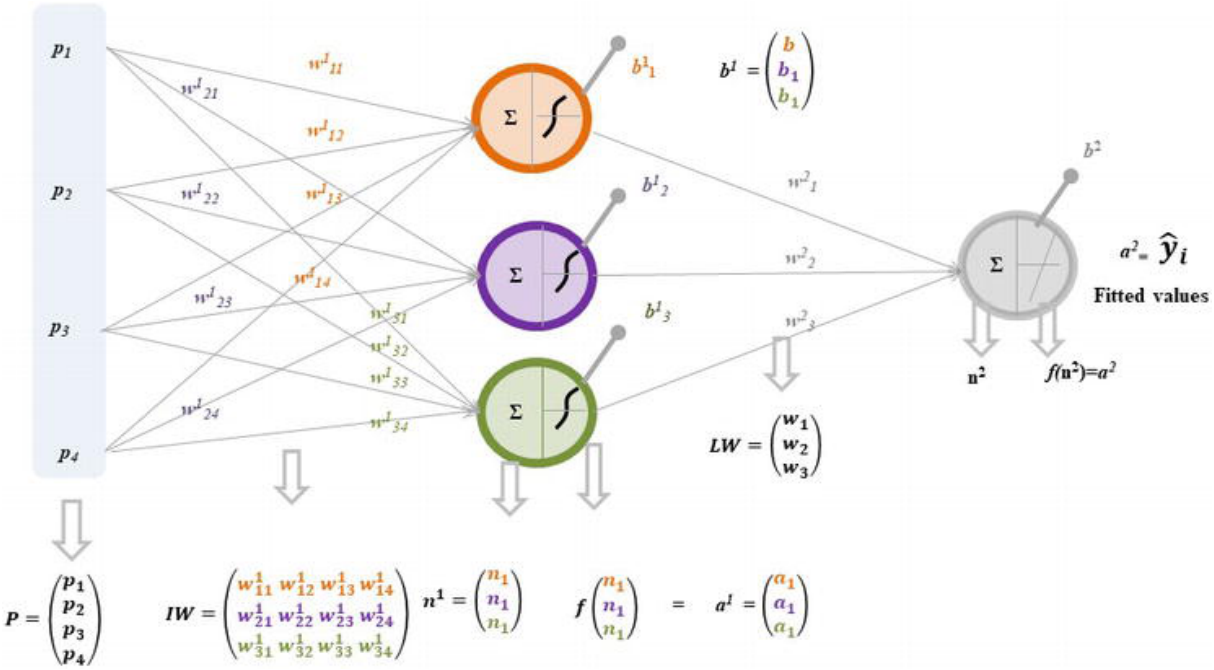

Figure 3.5: Multilayer perceptron neural network architecture

3.5 Types of Neural Networks

Generally speaking, in terms of network architectures, ANNs tend to be classified into two different classes, feedforward and recurrent ANNs, each may have several subclasses. and this

is some kinds of neural networks: **Feed Forward Neural Networks (FFNs)** This is the simplest neural network algorithm. A feed-forward network doesn't have any memory. That is, there is no going back in a feed-forward network. In many tasks, this approach is not very applicable. For example, when we work with text, the words form a certain sequence, and we want the machine to understand it.

Feedforward neural networks can be applied in supervised learning when the data that you work with is not sequential or time-dependent. You can also use it if you don't know how the output should be structured but want to build a relatively fast and easy NN. **Generative Adversarial Networks (GANs)** are from a different breed of networks, they are twins: two networks working together. GANs consist of any two networks (although often a combination of FFs and CNNs), with one tasked to generate content and the other to judge content. The discriminating network receives either training data or generated content from the generative network. How well the discriminating network was able to correctly predict the data source is then used as part of the error for the generating network. This creates a form of competition where the discriminator is getting better at distinguishing real data from generated data and the generator is learning to become less predictable to the discriminator. This works well in part because even quite complex noise-like patterns are eventually predictable, but generated content similar in features to the input data is harder to learn to distinguish. GANs can be quite difficult to train, as you don't just have to train two networks (either of which can pose its own problems), but their dynamics need to be balanced as well. If prediction or generation becomes too good compared to the other, a GAN won't converge as there is intrinsic divergence.

GANs are powerful tools used to generate realistic human faces, enhance image resolution, create images from text descriptions, and power DeepFake technologies. **Convolutional Neural Networks (CNNs)** Convolutional Neural Networks (CNNs) are different from other types of neural networks and are mainly used for image processing, though they can also handle other types of data such as audio. Instead of feeding the entire image into the network at once, CNNs use a kind of "scanner" to read small parts of the image (e.g., 20*20 pixels) step by step.

CNNs operate through convolutional layers, where each neuron is only connected to nearby data, not the entire input. As the network goes deeper, these layers typically shrink in size, often by divisible numbers like 32 or 16. Pooling layers (such as max pooling) are also used to reduce less important details and focus on the most relevant features.

CNNs can also be used with audio by dividing the sound wave into segments and analyzing

them one by one. **Autoencoders** Autoencoders are a type of neural network designed to **automatically compress and reconstruct data**, which is where the name "autoencoder" comes from (encoding as in compression, not encryption).

The structure of an autoencoder typically looks like an hourglass, where the input layer gradually narrows down through smaller hidden layers until it reaches a central bottleneck layer, the most compressed representation of the input. Then, the network expands again through symmetrical layers back to the output, aiming to reconstruct the original input as closely as possible. The network is divided into three main parts:

- **Encoder**: the layers leading up to the bottleneck that compress the input.
- **Code**: the bottleneck layer where the data is most condensed.
- **Decoder**: the layers after the bottleneck that attempt to reconstruct the original input.

Training is typically done using backpropagation, where the model is trained to minimize the difference between the original input and the reconstructed output. Some autoencoders are also designed with tied weights, meaning the weights used in the encoder are the same as those in the decoder, just reversed, ensuring a more compact and efficient model.

Autoencoders are employed for various purposes, including dimensionality reduction, image denoising, anomaly detection, data compression, and the generation or reconstruction of images. **Recurrent Neural Networks (RNN)** Recurrent Neural Networks are specifically designed to process sequential data, such as speech and time series. They are widely employed in applications like language translation and speech recognition. **Long Short-Term Memory (LSTM)** The long short term memory network is a specialized variant of RNN that is adept at managing long-term dependencies more effectively.

Given that both RNNs and LSTMs are specifically designed to process sequential data, we aim to explore their application in time series forecasting. We will examine these models in greater detail.

3.6 Recurrent Neural Network (RNN)

Recurrent Neural Network (RNN) is a type of artificial neural network designed specifically for processing sequential data (e.g., time series, text, audio). Unlike traditional feedforward

neural networks, RNNs have connections that form directed cycles, allowing information to persist across time steps. This structure gives RNNs a kind of memory, enabling them to model temporal dependencies and patterns over time. another way to think about RNNs is that a recurrent neural network has multiple copies of the same network, each passing a message to a successor. The information in recurrent neural networks cycles through a loop to the middle hidden layer.

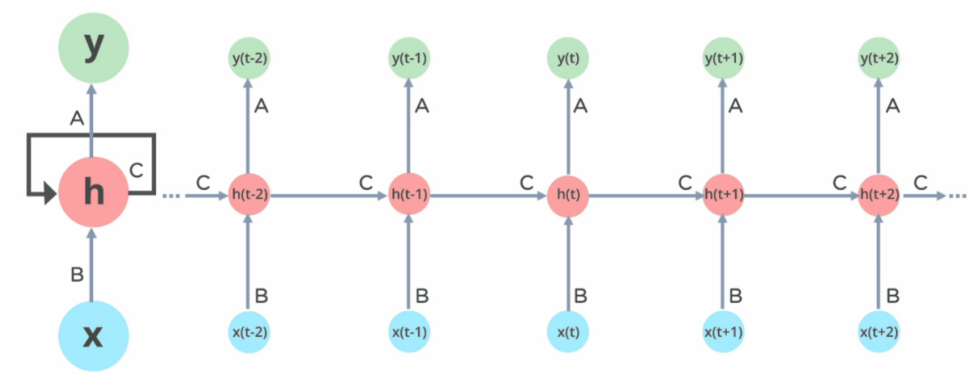

Figure 3.6: Basic recurrent neural network (RNN) flow

3.6.1 Types of RNN

RNN architecture can vary depending on the problem you're trying to solve. It can range from those with a single input and output to those with many.

In other words, the main reason that the recurrent nets are more exciting is that they allow us to operate over sequences of vectors: Sequence in the input, the output, or in the most general case, both. A few examples may this more concrete:

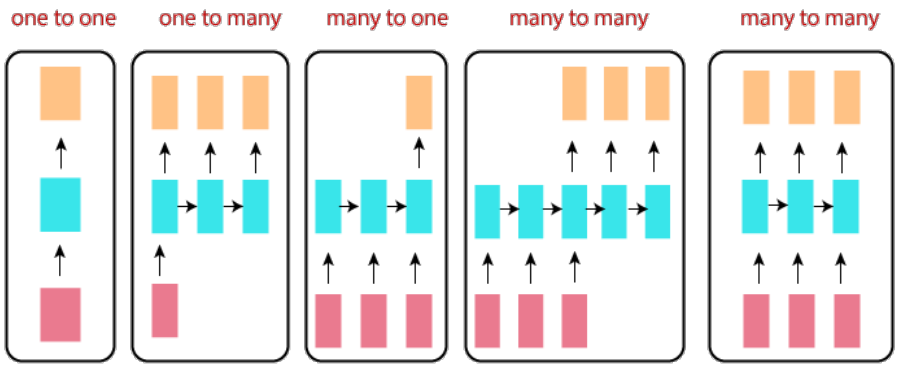

Figure 3.7: Types of recurrent neural networks (RNNs)

1. One-to-One:

This architecture is commonly referred to as Plain Neural Networks. It operates with a fixed-size input corresponding to a fixed-size output, where the two are independent of any preceding information or output.

2. One-to-Many:

This structure takes a fixed-size input and produces a sequence of output data.

3. Many-to-One: This configuration receives a sequence of input information and generates a fixed-size output.

4. Many-to-Many: This model processes a sequence of input information and recurrently outputs a sequence of data.

5. Bidirectional Many-to-Many: This approach involves synchronized sequence inputs and outputs. It is noteworthy that in all these cases, there are no pre specified constraints on the lengths of the sequences, as the recurrent transformation is consistent and can be applied iteratively as needed.

3.6.2 The Architecture of RNNs

Recurrent Neural Networks (RNNs) represent a class of neural networks characterized by the incorporation of hidden states, enabling the utilization of prior outputs as inputs. Their operation typically follows this structure: For each time step t, the activation $a^{\langle t \rangle}$ and the output $y^{\langle t \rangle}$ are

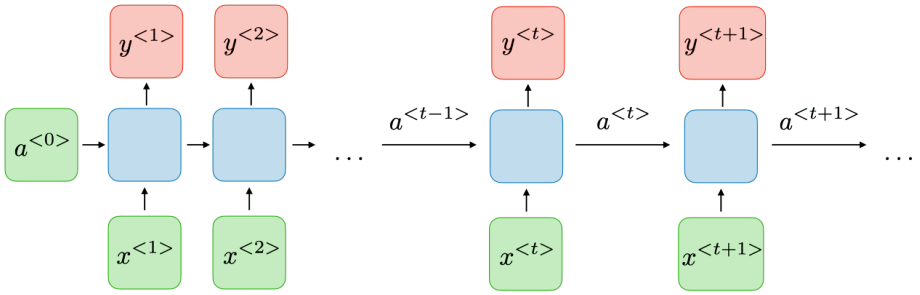

Figure 3.8: The architecture of recurrent neural networks

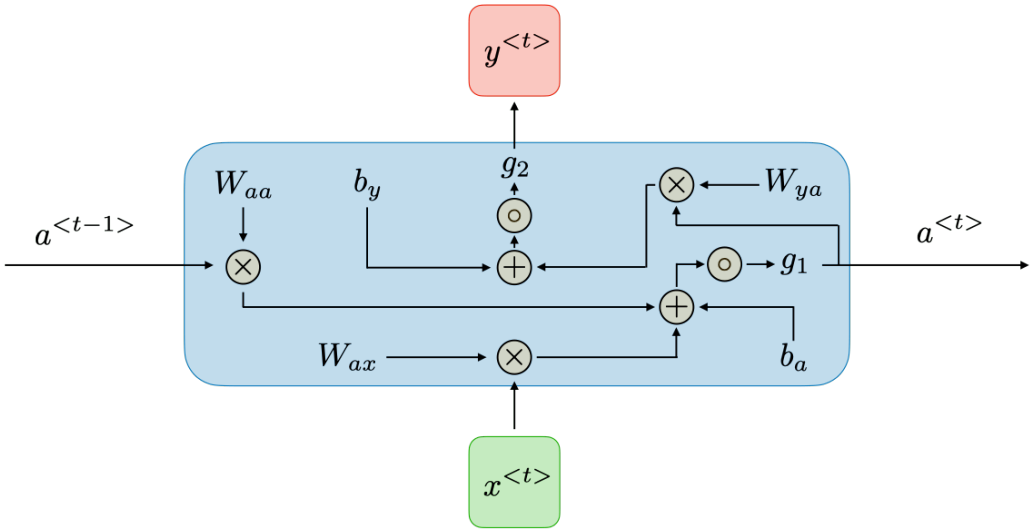

Figure 3.9: Inside architecture of a recurrent neural network (RNN)

expressed as follows:

$$a^{\langle t \rangle} = g_1(W_{aa}a^{\langle t-1 \rangle} + W_{ax}x^{\langle t \rangle} + b_a) \quad \text{and} \quad y^{\langle t \rangle} = g_2(W_{ya}a^{\langle t \rangle} + b_y)$$

where $W_{ax}, W_{aa}, W_{ya}, b_a, b_y$ are coefficients that are shared temporally and g_1, g_2 are activation functions.

3.6.3 Applications

The dataset from the earlier phase is retained for this step to preserve consistency in data processing and model evaluation.

Example 1: AirPassengers data

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense

# --- Load Data ---
 date_range = pd.date_range(start='1949-01', periods=len(data), freq=
    'MS')
df = pd.DataFrame({'Month': date_range, 'Passengers': data})
df.set_index('Month', inplace=True)

# --- Normalize data ---
scaler = MinMaxScaler()
df['Passengers_scaled'] = scaler.fit_transform(df[['Passengers']])

# --- Create sequences (lags) ---
def create_sequences(series, window):
    X, y = [], []
    for i in range(window, len(series)):
        X.append(series[i - window:i])
        y.append(series[i])
    return np.array(X), np.array(y)

WINDOW_SIZE = 12
series = df['Passengers_scaled'].values
```

```
X, y = create_sequences(series, WINDOW_SIZE)

# --- Train/test split ---
split_index = int(len(X) * 0.8)
X_train, y_train = X[:split_index], y[:split_index]
X_test, y_test = X[split_index:], y[split_index:]

# --- Reshape for RNN ---
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))

# --- Build SimpleRNN model ---
model = Sequential([
    SimpleRNN(64, activation='relu', input_shape=(WINDOW_SIZE, 1)),
    Dense(1)
])
model.compile(optimizer='adam', loss='mse')
model.summary()

# --- Train ---
history = model.fit(X_train, y_train, epochs=100, batch_size=16,
                    validation_data=(X_test, y_test), verbose=1)

# --- Predict ---
y_pred = model.predict(X_test)

# --- Inverse transform predictions ---
y_test_inv = scaler.inverse_transform(y_test.reshape(-1, 1))
y_pred_inv = scaler.inverse_transform(y_pred)

# --- Evaluation ---
rmse = np.sqrt(mean_squared_error(y_test_inv, y_pred_inv))
print(f'RMSE: {rmse:.2f}')
```

```
# --- Plot results ---
test_index = df.index[WINDOW_SIZE + split_index:]
plt.figure(figsize=(12, 6))
plt.plot(df.index, df['Passengers'], label='Original')
plt.plot(test_index, y_pred_inv, label='SimpleRNN Forecast', color='
    red')
plt.title('SimpleRNN - AirPassengers Forecast')
plt.xlabel('Date')
plt.ylabel('Number of Passengers')
plt.legend()
plt.grid(True)
plt.show()
```

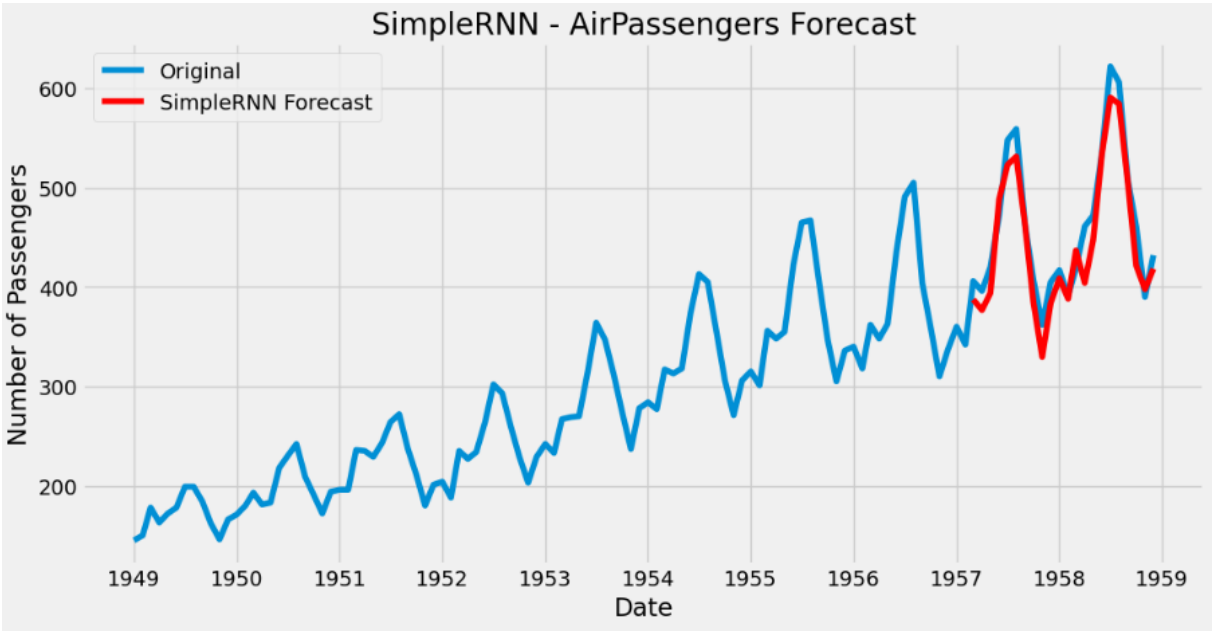

Figure 3.10: RNN forecasting for AirPassengers data

Example 2: Saida temperature data

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense

# --- Load & prepare data ---
df = pd.read_csv('../input/correc/corrected_temperature_data.csv')
df = df.set_index('Datetime')
df.index = pd.to_datetime(df.index)
df.dropna(inplace=True)

temperature_data = df[['Temperature']]

# --- Normalize data ---
scaler = MinMaxScaler()
temperature_scaled = scaler.fit_transform(temperature_data)

# --- Create sequences ---
def create_sequences(data, seq_length=10):
    X, y = [], []
    for i in range(len(data) - seq_length):
        X.append(data[i:i + seq_length])
        y.append(data[i + seq_length])
    return np.array(X), np.array(y)

SEQ_LENGTH = 10
X, y = create_sequences(temperature_scaled, SEQ_LENGTH)

# --- Split data ---
train_size = int(len(X) * 0.8)
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]

# --- Fix input shape ---
X_train = X_train.reshape((X_train.shape[0], SEQ_LENGTH, 1))
X_test = X_test.reshape((X_test.shape[0], SEQ_LENGTH, 1))
```

```
# --- Build model ---
model = Sequential([
    SimpleRNN(64, activation='relu', input_shape=(SEQ_LENGTH, 1)),
    Dense(1)
])
model.compile(optimizer='adam', loss='mse')
model.summary()

# --- Train model ---
history = model.fit(X_train, y_train, epochs=100, batch_size=16,
                    validation_data=(X_test, y_test), verbose=1)

# --- Predict ---
y_pred = model.predict(X_test)

# --- Inverse scaling ---
y_test_inv = scaler.inverse_transform(y_test.reshape(-1, 1))
y_pred_inv = scaler.inverse_transform(y_pred)

# --- Evaluation ---
rmse = np.sqrt(mean_squared_error(y_test_inv, y_pred_inv))
print(f"SimpleRNN Test RMSE: {rmse:.2f}")

# --- Set prediction dates ---
prediction_dates = df.index[-len(y_test_inv):]

# --- Plot ---
plt.figure(figsize=(20, 6))
df_2024 = df[(df.index >= '2024-01-01') & (df.index < '2025-01-01')]
plt.plot(df_2024.index, df_2024['Temperature'], label='Actual
   Temperature 2024', color='blue')
plt.plot(prediction_dates, y_pred_inv, label='Predicted Temperature (
   Nov-Dec 2024)', color='red')
```

```
plt.title('Temperature in SaÃŕda, Algeria (Actual 2024 + Predicted
    Nov-Dec)')
plt.xlabel('Date')
plt.ylabel('Temperature (ÂřC)')
plt.legend()
plt.grid(True)
plt.axvline(pd.to_datetime('2024-10-30'), color='black', ls='--')
plt.tight_layout()
plt.show()
```

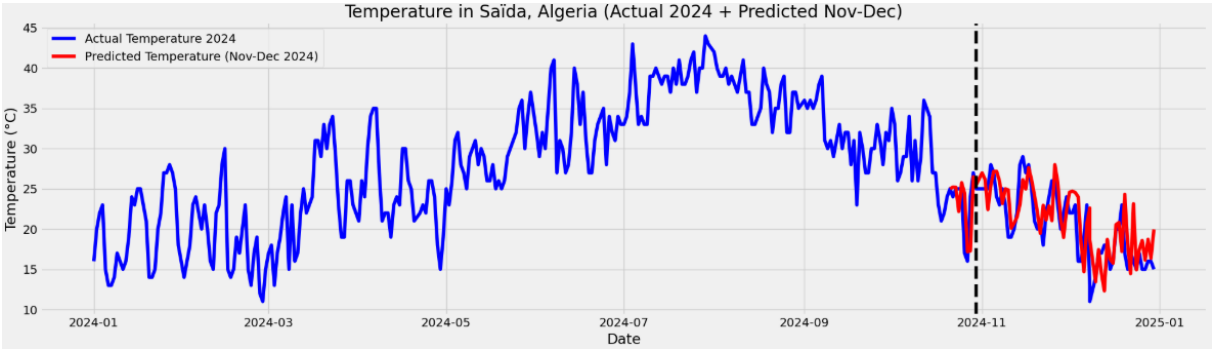

Figure 3.11: RNN forecasting for temperature in Saida

3.7 Long Short-Term Memory (LSTM) networks

Long Short-Term Memory (LSTM) networks represent a distinct class of Recurrent Neural Networks (RNNs) specifically designed to learn long-term dependencies in data. Initially introduced by Hochreiter and Schmidhuber in 1997, LSTMs have undergone extensive refinement and have been popularized by numerous researchers in the field. Their efficacy has been demonstrated across a diverse array of applications, leading to their widespread adoption in practical scenarios. LSTM networks were developed to address the inherent limitations of traditional RNNs, particularly the challenges associated with learning long-term dependencies, which often arise from the vanishing and exploding gradient problems.

3.7.1 The architecture of LSTM

The neural network architecture for an LSTM block, as shown in Figure below, demonstrates that the LSTM network extends the memory capabilities of a traditional RNN. It can selectively

remember or forget information through specialized structures known as the *cell state* and three types of *gates*.

Thus, in addition to the hidden state found in standard RNNs, an LSTM block typically includes four additional components. These are:

- the **cell state** (C_t),
- the **input gate** (i_t),
- the **output gate** (o_t), and
- the **forget gate** (f_t).

Each of these layers interacts with the others in a carefully designed manner to regulate the flow of information and effectively learn patterns from the training data.

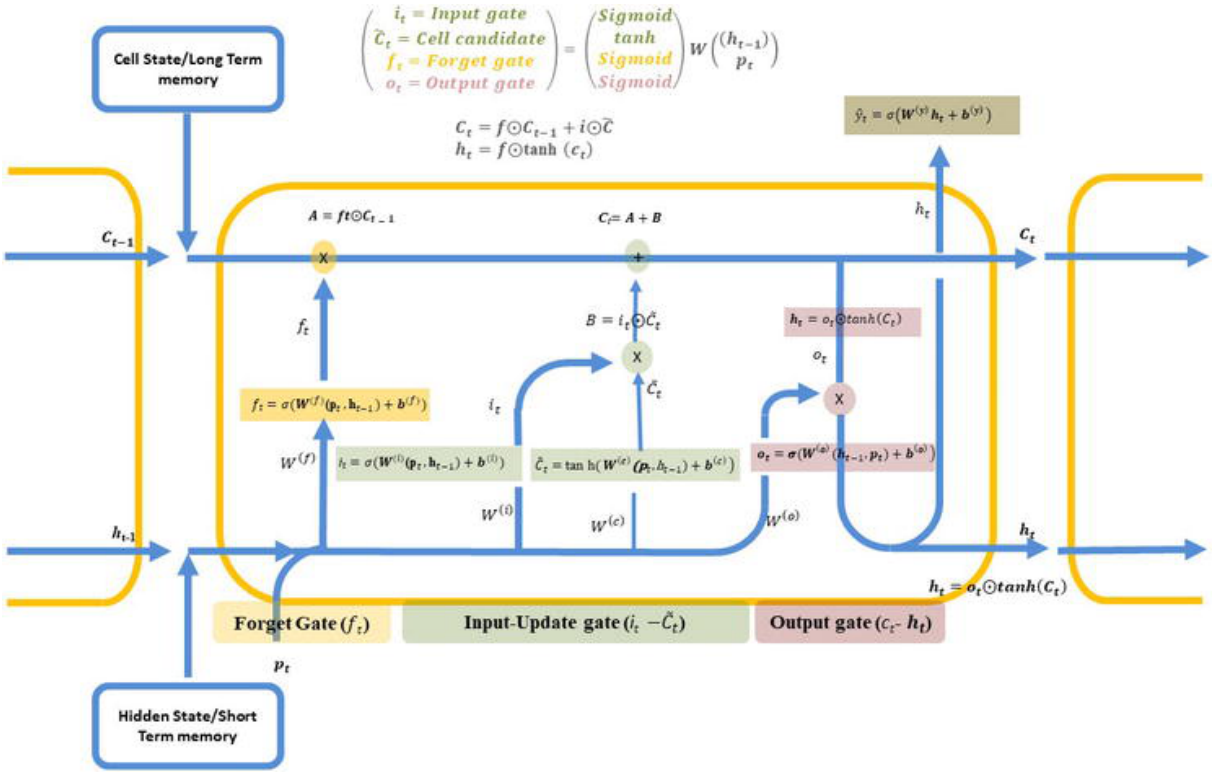

Figure 3.12: The architecture of LSTM

Cell state (C_t):

The cell state is a fundamental component of LSTM networks and serves as the memory of the system. the flow of the cell state can be likened to a conveyor belt or a production line, where information moves through the sequence mostly unchanged, unless altered by simple operations

such as multiplication and addition. in this setup, information flows smoothly along the chain, and if there are no interactions, it remains unchanged. however, the LSTM block has the ability to add or remove information from the cell state through mechanisms known as gates. These gates allow selective information to pass into or out of the cell state, depending on what the network learns during training. This selective control is what enables LSTM networks to retain or forget information as needed, unlike traditional RNNs.

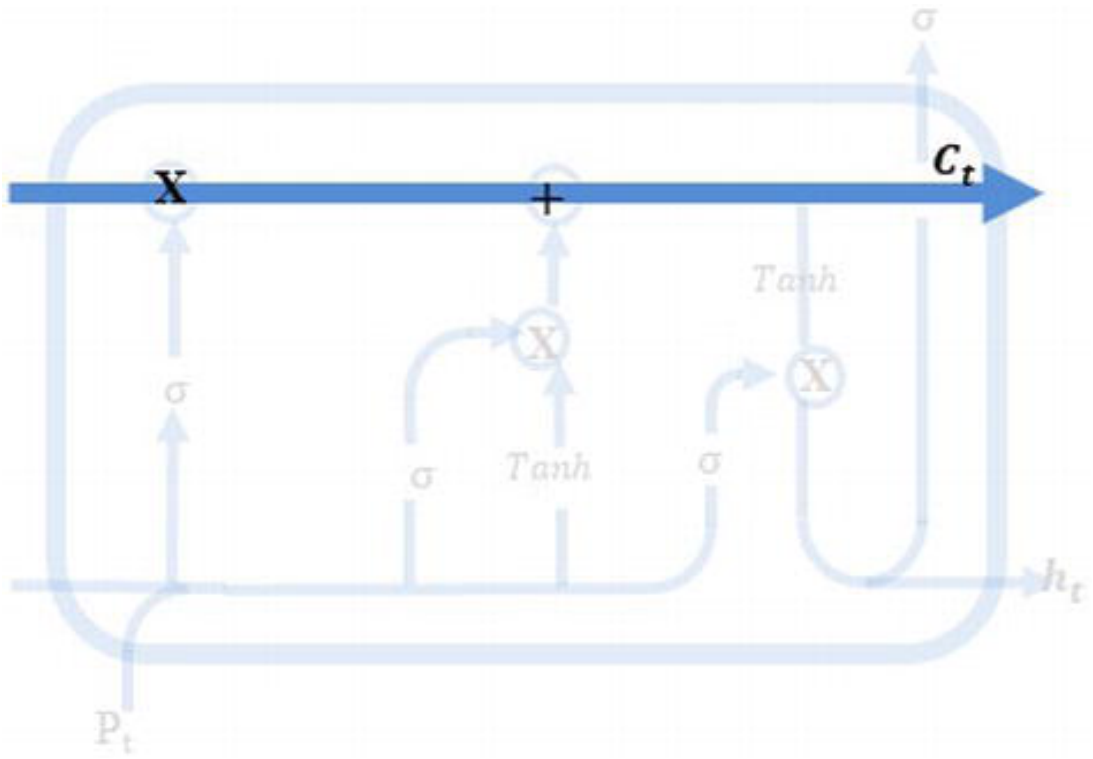

Figure 3.13: Cell state (C_t)

Forget gate (f_t)

The Forget Gate (f_t) determines what information should be discarded or retained from the cell state. This process is implemented using a sigmoid activation function, which outputs values in the range of 0 to 1. The output depends on the current input p_t, the previous hidden state h_{t-1}, a weight matrix $W^{(f)}$, and a bias vector $b^{(f)}$.

The forget gate is defined by the following equation:

$$f_t = \sigma\left(W^{(f)} \cdot [p_t, h_{t-1}] + b^{(f)}\right) \tag{3.1}$$

Here, σ denotes the sigmoid activation function, defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3.2}$$

In this context:

- $W^{(f)}$ is the weight matrix for the forget gate.
- $b^{(f)}$ is the bias vector.
- $[p_t, h_{t-1}]$ is the concatenation of the current input and the previous hidden state.

The forget gate output f_t determines the extent to which the values in the previous cell state are retained or forgotten. A value close to 0 means the information is forgotten, while a value close to 1 means the information is retained.

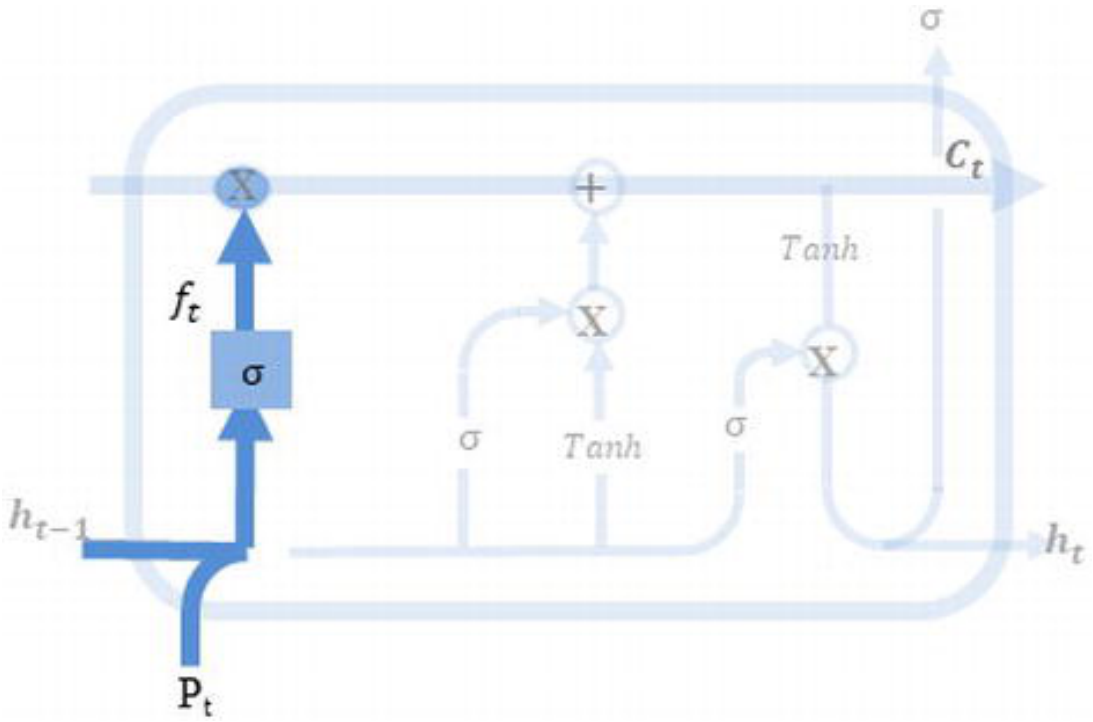

Figure 3.14: Forget gate in LSTM cell

3.7.2 Input gate

(i_t) The input-update gate decides what new information should be stored in the cell state, which has two parts: A sigmoid layer and a hyperbolic tangent (tanh) layer. The sigmoid layer is called the "input gate layer" because it decides which values should be updated. The tanh layer is a vector of new candidate values $\tilde{C}_t$ that could be added to the cell state.

Mathematically, the input gate is defined as:

$$i_t = \sigma\left(W^{(i)} \cdot [p_t, h_{t-1}] + b^{(i)}\right) = \frac{1}{1 + e^{-\left(W^{(i)} \cdot [p_t, h_{t-1}] + b^{(i)}\right)}} \tag{3.3}$$

Where:

- $W^{(i)}$ is the weight matrix associated with the input gate.
- $b^{(i)}$ is the bias vector.
- p_t is the input at the current time step t.
- h_{t-1} is the hidden state from the previous time step.
- $\sigma(x)$ is the sigmoid function: $\sigma(x) = \frac{1}{1+e^{-x}}$.

Just like in the forget gate, the parameters $W^{(i)}$ and $b^{(i)}$ are learned during training. At each time step, the model uses the input p_t and the previous hidden state h_{t-1} to determine how much new information to write to the memory cell, before calculating the candidate cell state.
Next, a vector of new candidate values $\tilde{C}_t$ is computed. This candidate represents potential new information that may be added to the cell state. The computation process is similar to that used in the forget and input gates, but instead of a sigmoid activation, it uses the hyperbolic tangent (*tanh*) activation function, which outputs values in the range $(-1, 1)$. The formula is given by:

$$\tilde{C}_t = \tanh\left(W^{(c)} \cdot [p_t, h_{t-1}] + b^{(c)}\right) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \quad \text{where } x = W^{(c)} \cdot [p_t, h_{t-1}] + b^{(c)} \tag{3.4}$$

Where:

- $W^{(c)}$ is the weight matrix for the candidate cell state.
- $b^{(c)}$ is the bias vector.
- p_t is the input at the current time step.
- h_{t-1} is the hidden state from the previous time step.
- $\tanh(x)$ is the hyperbolic tangent function:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

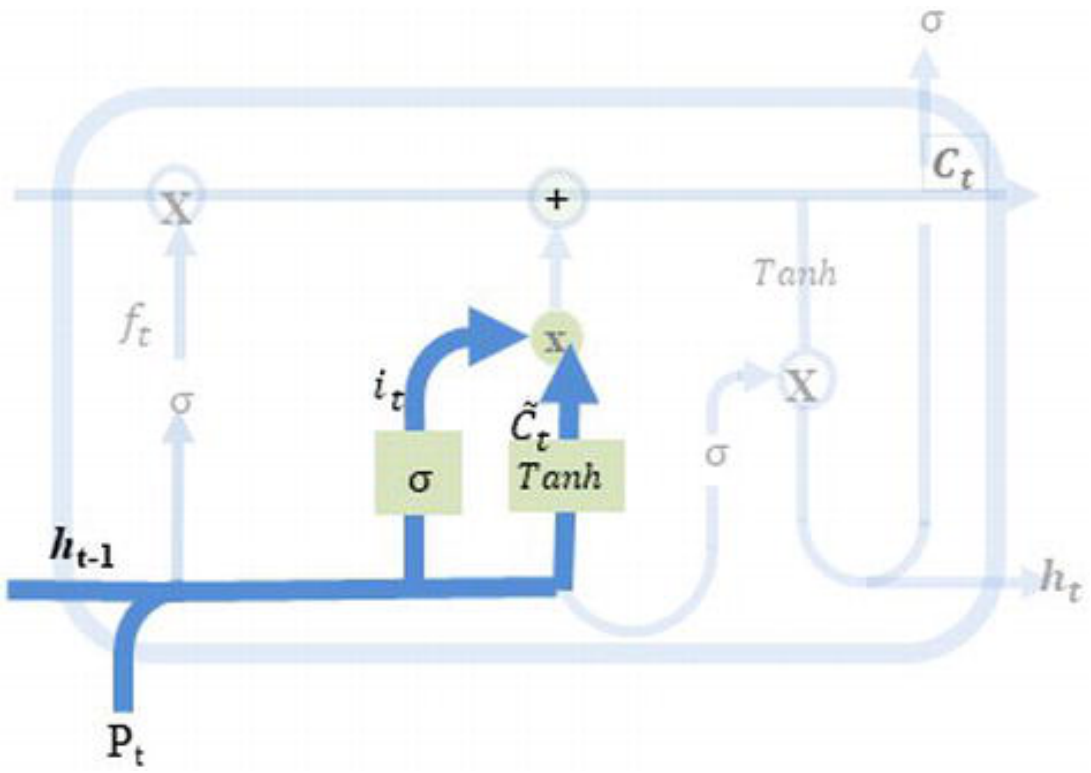

Figure 3.15: Input gate in the LSTM cell

In the next step, the values from the input gate and the candidate cell state are combined to create and update the memory cell. This process is represented by Equation(1). It involves a linear combination of two elements:

- The **forget gate** (f_t), which determines how much of the previous cell state (C_{t-1}) should be retained.
- The **input gate** (i_t), which decides how much of the newly generated candidate state ($\tilde{C}_t$) should be added.

The update is done using element-wise (Hadamard) multiplication, denoted by $\circ$. The resulting updated cell state C_t is given by:

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

This equation allows the LSTM to maintain long-term dependencies by carefully deciding what information to preserve from the past and what new data to incorporate, enabling effective memory control throughout the sequence.

3.7.3 Output gate

The output gate (o_t) controls which information to reveal from the updated cell state (C_t) to the output in a single time step. In other words, the output gate determines what the value of the next hidden state should be in each time step. After updating the cell state, the LSTM computes the output gate o_t. The output gate uses a sigmoid activation function applied to the previous hidden state and the current input:

$$o_t = \sigma\left(W^{(o)} \cdot [h_{t-1}, p_t] + b^{(o)}\right) = \frac{1}{1 + e^{-\left(W^{(o)} \cdot [h_{t-1}, p_t] + b^{(o)}\right)}} \tag{3.5}$$

Where:

- $W^{(o)}$, $b^{(o)}$ are the weight matrix and bias for the output gate.
- h_{t-1} is the previous hidden state.
- p_t is the current input.
- $\sigma(x)$ is the sigmoid function.

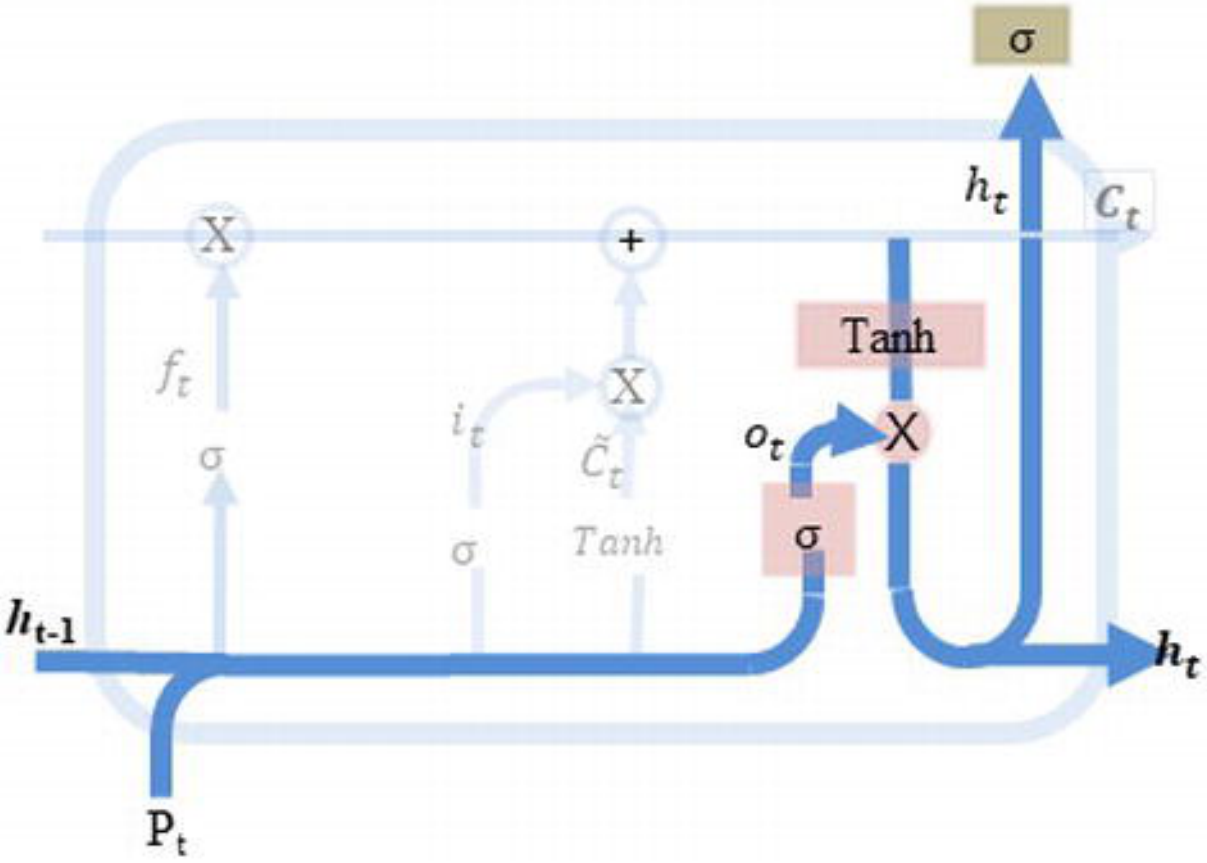

Figure 3.16: Output gate in the LSTM cell

The current hidden state h_t is then computed by applying the output gate o_t to the hyperbolic tangent of the updated cell state C_t:

$$h_t = o_t \circ \tanh(C_t) = o_t \circ \frac{e^x - e^{-x}}{e^x + e^{-x}}, \quad \text{where } x = f_t \circ C_{t-1} + i_t \circ \tilde{C}_t \tag{3.6}$$

This allows the network to filter what part of the updated cell state should influence the current hidden state.

The final predicted output $\hat{y}_t$ is computed by passing the hidden state through another sigmoid activation:

$$\hat{y}_t = \sigma\left(W^{(y)} \cdot h_t + b^{(y)}\right) = \frac{1}{1 + e^{-\left(W^{(y)} \cdot h_t + b^{(y)}\right)}} \tag{3.7}$$

Where:

- $W^{(y)}$ and $b^{(y)}$ are the weight matrix and bias for the output layer.
- $\hat{y}_t$ is the model's predicted output at time t.

3.7.4 LSTM memory cell summary

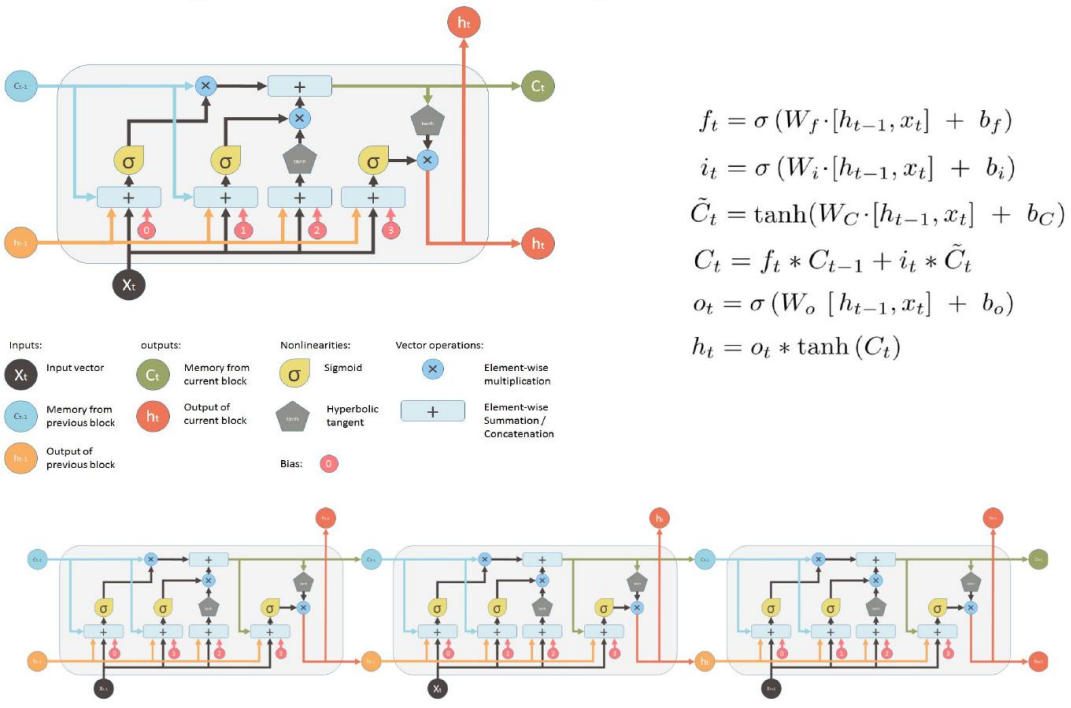

Figure 3.17: LSTM memory cell summary

3.7.5 Applications

The dataset from the earlier phase is retained for this step to preserve consistency in data processing and model evaluation.

Example 1: AirPassengers data

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
# --- Load Data ---
date_range = pd.date_range(start='1949-01', periods=len(data), freq='
    MS')
df = pd.DataFrame({'Month': date_range, 'Passengers': data})
df.set_index('Month', inplace=True)

# --- Normalize data ---
scaler = MinMaxScaler()
df['Passengers_scaled'] = scaler.fit_transform(df[['Passengers']])

# --- Create sequences (lags) for LSTM ---
def create_sequences(series, window):
    X, y = [], []
    for i in range(window, len(series)):
        X.append(series[i - window:i])
        y.append(series[i])
    return np.array(X), np.array(y)

WINDOW_SIZE = 12
series = df['Passengers_scaled'].values
X, y = create_sequences(series, WINDOW_SIZE)
```

```
# --- Split into train/test ---
split_index = int(len(X) * 0.8)
X_train, y_train = X[:split_index], y[:split_index]
X_test, y_test = X[split_index:], y[split_index:]

# Reshape for LSTM: [samples, time_steps, features]
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))

# --- Build LSTM model ---
model = Sequential([
    LSTM(64, activation='relu', input_shape=(WINDOW_SIZE, 1)),
    Dense(1)
])
model.compile(optimizer='adam', loss='mse')
model.summary()

# --- Train ---
history = model.fit(X_train, y_train, epochs=100, batch_size=16,
                    validation_data=(X_test, y_test), verbose=1)

# --- Predict ---
y_pred = model.predict(X_test)

# --- Inverse scale predictions ---
y_test_inv = scaler.inverse_transform(y_test.reshape(-1, 1))
y_pred_inv = scaler.inverse_transform(y_pred)

# --- Evaluation ---
rmse = np.sqrt(mean_squared_error(y_test_inv, y_pred_inv))
print(f'RMSE: {rmse:.2f}')

# --- Plot results ---
```

```
test_index = df.index[WINDOW_SIZE + split_index:]
plt.figure(figsize=(12, 6))
plt.plot(df.index, df['Passengers'], label='Original')
plt.plot(test_index, y_pred_inv, label='LSTM Forecast', color='red')
plt.title('LSTM - AirPassengers Forecast')
plt.xlabel('Date')
plt.ylabel('Number of Passengers')
plt.legend()
plt.grid(True)
plt.show()

# --- Show Predictions vs Actual ---
comparison = pd.DataFrame({
    'Actual': y_test_inv.flatten(),
    'Predicted': y_pred_inv.flatten()
}, index=test_index)
print(comparison.head(15))
```

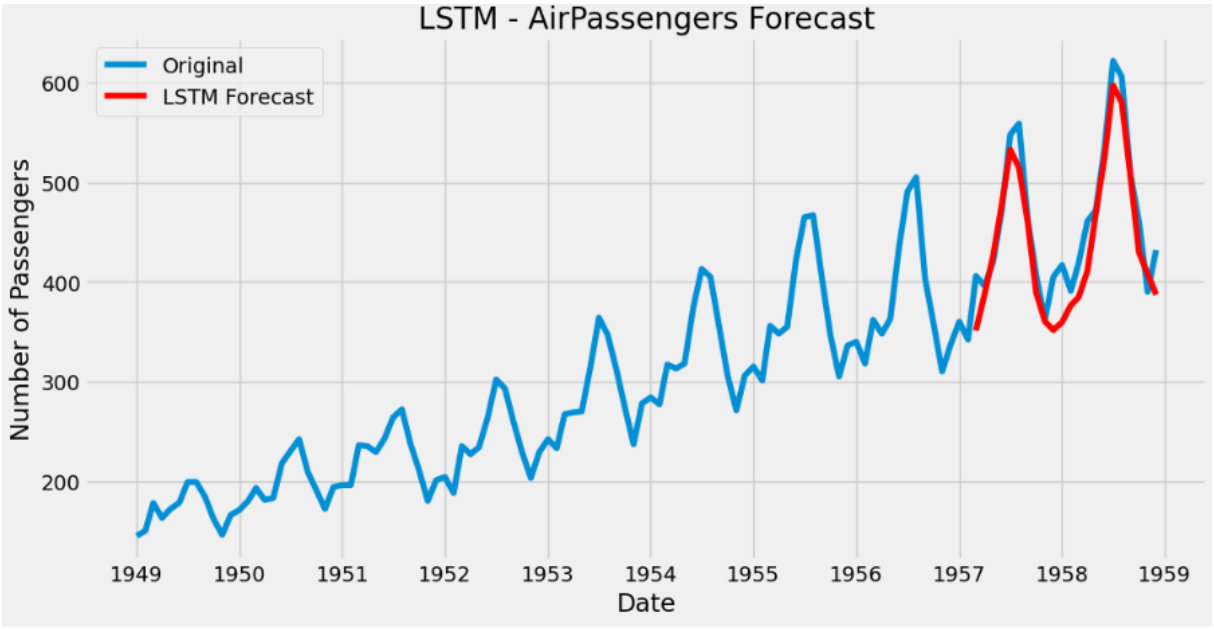

Figure 3.18: LSTM forecasting for AirPassengers

Example 2: Saida temperature data

```
import pandas as pd
```

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from sklearn.metrics import mean_squared_error
# --- Load Data ---
df = pd.read_csv('../input/correc/corrected_temperature_data.csv')
df = df.set_index('Datetime')
df.index = pd.to_datetime(df.index)
df.dropna(inplace=True)
temperature_data = df[['Temperature']]
# --- Normalize data ---
scaler = MinMaxScaler()
temperature_scaled = scaler.fit_transform(temperature_data)
# --- Create sequences (lags) for LSTM ---
def create_sequences(data, seq_length=10):
    X, y = [], []
    for i in range(len(data) - seq_length):
        X.append(data[i:i + seq_length])
        y.append(data[i + seq_length])
    return np.array(X), np.array(y)

SEQ_LENGTH = 10
X, y = create_sequences(temperature_scaled, SEQ_LENGTH)
# --- Split into train/test ---
train_size = int(len(X) * 0.8)
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]
# --- Build LSTM model ---
model = Sequential()
model.add(LSTM(64, activation='relu', input_shape=(SEQ_LENGTH, 1)))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
```

```
# --- Train ---
history = model.fit(X_train, y_train, epochs=30, batch_size=32,
    validation_split=0.1, verbose=1)
# --- Predict ---
y_pred = model.predict(X_test)

# --- Inverse scale predictions ---
y_test_inv = scaler.inverse_transform(y_test)
y_pred_inv = scaler.inverse_transform(y_pred)
# --- Evaluation ---
rmse # --- Evaluation ---= np.sqrt(mean_squared_error(y_test_inv,
    y_pred_inv))
print(f"LSTM Test RMSE: {rmse:.2f}")
# --- Plot results ---
prediction_dates = df.index[-len(y_test_inv):]
plt.figure(figsize=(20, 6))
df_2024 = df[(df.index >= '2024-01-01') & (df.index < '2025-01-01')]
plt.plot(df_2024.index, df_2024['Temperature'], label='Actual
    Temperature 2024', color='blue')
plt.plot(prediction_dates, y_pred_inv, label='Predicted Temperature (
    Nov-Dec 2024)', color='red')
plt.title('Temperature in SaÃŕda, Algeria (Actual 2024 + Predicted
    Nov-Dec)')
plt.xlabel('Date')
plt.ylabel('Temperature (ÂřC)')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.axvline(pd.to_datetime('2024-10-30'), color='black', ls='--')
plt.show()
```

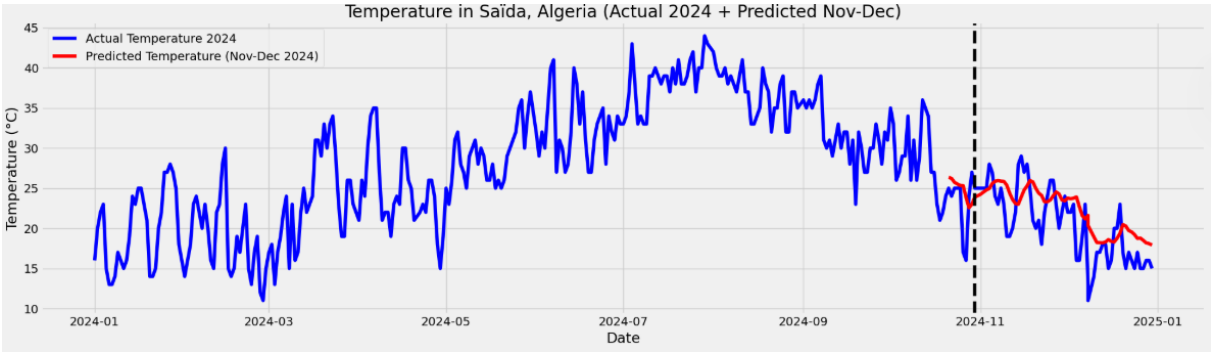

Figure 3.19: LSTM forecasting for temperature in Saida

3.7.6 The Relationship between AI, machine learning, and deep learning

Although artificial intelligence (AI) and machine learning (ML) are often used synonymously, they are not interchangeable terms.

Artificial intelligence is an area of computer science that concerns building computers and machines that can reason, learn, and act in a way resembling human intelligence, or systems that involve data whose scale exceeds what humans can analyse. The field includes many different disciplines including data analytics, statistics, hardware and software engineering, neuroscience, and even philosophy.

Whereas artificial intelligence is a broad category of computer science, machine learning is an application of AI that involves training machines to execute a task without being specifically programmed for it. Machine learning is used more explicitly as a means to extract knowledge from data using techniques such as neural networks, supervised and unsupervised learning, decision trees, and linear regression. Like machine learning is a subset of artificial intelligence, deep learning is a subset of machine learning.

Deep learning works by training neural networks on sets of data. A neural network is a model that uses a system of artificial neurons that are computational nodes used to classify and analyse data. Data are fed into the first layer of a neural network, with each node making a decision, and then passing that information onto multiple nodes in the next layer. Training models with more than three layers are referred to as deep neural networks or deep learning. Some modern neural networks have hundreds or thousands of layers.

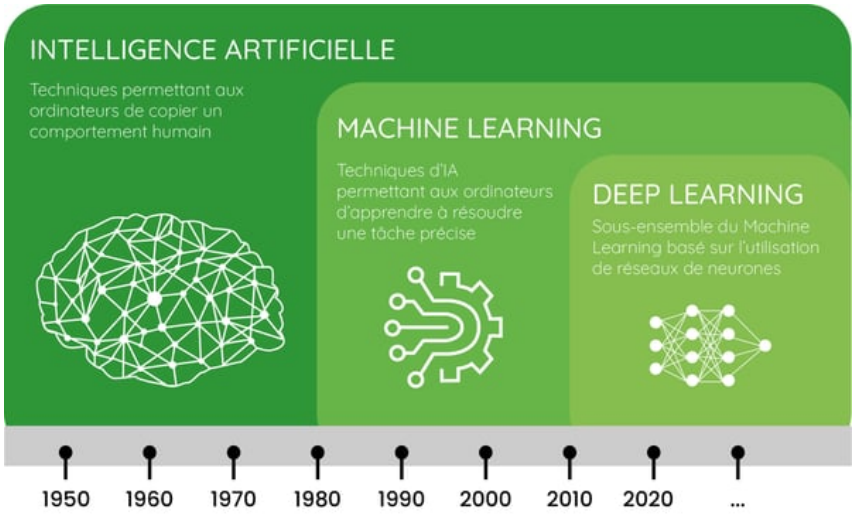

Figure 3.20: AI, machine learning, and deep learning relationship

3.8 Comparing forecasting methods

In this section, we will conduct a comparative analysis between a traditional forecasting method, specifically the Holt-Winters technique, and various artificial intelligence methods, including Random Forest, XGBoost, Long Short-Term Memory (LSTM), and Recurrent Neural Networks (RNN).

3.8.1 AirPassengers data

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.holtwinters import ExponentialSmoothing
from sklearn.metrics import mean_squared_error
from sklearn.ensemble import RandomForestRegressor
import xgboost as xgb
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, SimpleRNN, Dense
from tensorflow.keras.callbacks import EarlyStopping
# --- Load AirPassengers Data ---
data = [
    145, 150, 178, 163, 172, 178, 199, 199, 184, 162, 146, 166,
    171, 180, 193, 181, 183, 218, 230, 242, 209, 191, 172, 194,
```

```
    196, 196, 236, 235, 229, 243, 264, 272, 237, 211, 180, 201,
    204, 188, 235, 227, 234, 264, 302, 293, 259, 229, 203, 229,
    242, 233, 267, 269, 270, 315, 364, 347, 312, 274, 237, 278,
    284, 277, 317, 313, 318, 374, 413, 405, 355, 306, 271, 306,
    315, 301, 356, 348, 355, 422, 465, 467, 404, 347, 305, 336,
    340, 318, 362, 348, 363, 435, 491, 505, 404, 359, 310, 337,
    360, 342, 406, 396, 420, 472, 548, 559, 463, 407, 362, 405,
    417, 391, 419, 461, 472, 535, 622, 606, 508, 461, 390, 432
]
date_range = pd.date_range(start='1949-01', periods=len(data), freq='
   MS')
df = pd.DataFrame({'Month': date_range, 'Passengers': data})
df.set_index('Month', inplace=True)

# --- Holt-Winters Forecast ---
split_index = int(len(df) * 0.8)
train = df.iloc[:split_index]
test = df.iloc[split_index:]
hw_model = ExponentialSmoothing(train['Passengers'], trend='mul',
   seasonal='mul', seasonal_periods=12)
hw_fit = hw_model.fit()
hw_forecast = hw_fit.forecast(steps=len(test))
hw_rmse = np.sqrt(mean_squared_error(test['Passengers'], hw_forecast)
   )
print(f"Holt-Winters RMSE: {hw_rmse:.2f}")

# --- Create Lag Features ---
def create_lag_features(data, lags=12):
    df_lag = data.copy()
    for i in range(1, lags + 1):
        df_lag[f'lag_{i}'] = df_lag['Passengers'].shift(i)
    df_lag.dropna(inplace=True)
    return df_lag
```

```
lags = 12
df_lagged = create_lag_features(df[['Passengers']], lags)

# --- Split for ML and RNN/LSTM ---
split_idx = int(len(df_lagged) * 0.8)
train_ml = df_lagged.iloc[:split_idx]
test_ml = df_lagged.iloc[split_idx:]

X_train = train_ml.drop('Passengers', axis=1).values
y_train = train_ml['Passengers'].values
X_test = test_ml.drop('Passengers', axis=1).values
y_test = test_ml['Passengers'].values
# --- Random Forest ---
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
rf_pred = rf_model.predict(X_test)
rf_rmse = np.sqrt(mean_squared_error(y_test, rf_pred))
print(f"Random Forest RMSE: {rf_rmse:.2f}")

# --- XGBoost ---
xgb_model = xgb.XGBRegressor(objective='reg:squarederror',
   n_estimators=100)
xgb_model.fit(X_train, y_train)
xgb_pred = xgb_model.predict(X_test)
xgb_rmse = np.sqrt(mean_squared_error(y_test, xgb_pred))
print(f"XGBoost RMSE: {xgb_rmse:.2f}")

# --- Prepare for LSTM and RNN ---
X_train_rnn = X_train.reshape((X_train.shape[0], lags, 1))
X_test_rnn = X_test.reshape((X_test.shape[0], lags, 1))

# --- LSTM ---
lstm_model = Sequential([
    LSTM(50, activation='relu', input_shape=(lags, 1)),
```

```
    Dense(1)
])
lstm_model.compile(optimizer='adam', loss='mse')
early_stop = EarlyStopping(monitor='val_loss', patience=10,
   restore_best_weights=True)
lstm_model.fit(X_train_rnn, y_train, epochs=100, batch_size=8,
               validation_split=0.1, callbacks=[early_stop], verbose
                  =0)
lstm_pred = lstm_model.predict(X_test_rnn).flatten()
lstm_rmse = np.sqrt(mean_squared_error(y_test, lstm_pred))
print(f"LSTM RMSE: {lstm_rmse:.2f}")

# --- Simple RNN ---
rnn_model = Sequential([
    SimpleRNN(50, activation='relu', input_shape=(lags, 1)),
    Dense(1)
])
rnn_model.compile(optimizer='adam', loss='mse')
rnn_model.fit(X_train_rnn, y_train, epochs=100, batch_size=8,
              validation_split=0.1, callbacks=[early_stop], verbose
                 =0)
rnn_pred = rnn_model.predict(X_test_rnn).flatten()
rnn_rmse = np.sqrt(mean_squared_error(y_test, rnn_pred))
print(f"Simple RNN RMSE: {rnn_rmse:.2f}")

# --- Plot ---
plt.figure(figsize=(14, 6))
plt.plot(test.index, test['Passengers'], label='Actual', color='green
   ')
plt.plot(test.index, hw_forecast, label=f'Holt-Winters (RMSE={hw_rmse
   :.2f})', color='red')
plt.plot(test_ml.index, rf_pred, label=f'Random Forest (RMSE={rf_rmse
   :.2f})', color='purple', linestyle='--')
```

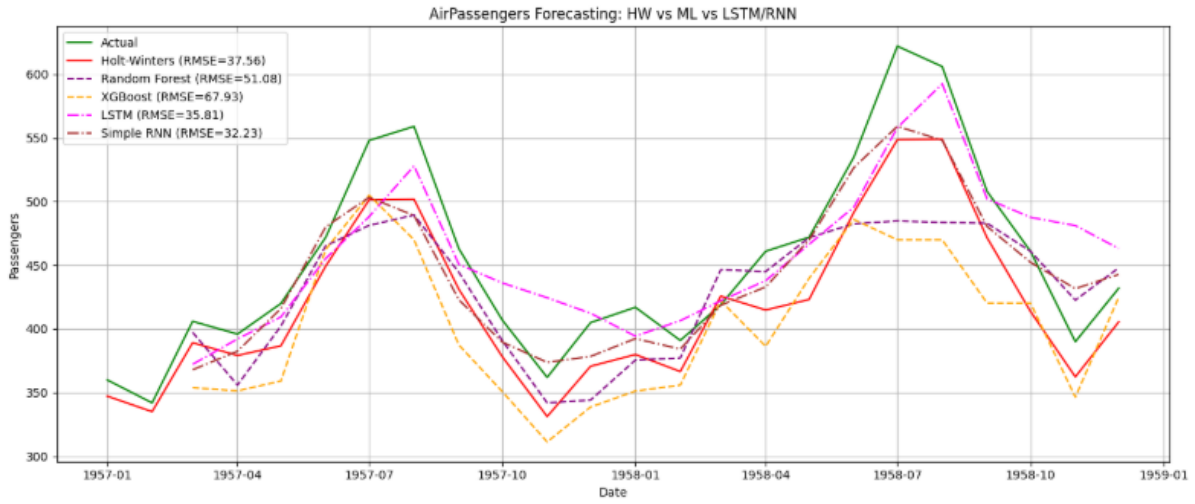

Figure 3.21: AirPassengers data

```
plt.plot(test_ml.index, xgb_pred, label=f'XGBoost (RMSE={xgb_rmse:.2f
    })', color='orange', linestyle='--')
plt.plot(test_ml.index, lstm_pred, label=f'LSTM (RMSE={lstm_rmse:.2f
    })', color='magenta', linestyle='-.')
plt.plot(test_ml.index, rnn_pred, label=f'Simple RNN (RMSE={rnn_rmse
    :.2f})', color='brown', linestyle='-.')
plt.title('AirPassengers Forecasting: HW vs ML vs LSTM/RNN')
plt.xlabel('Date')
plt.ylabel('Passengers')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

3.8.2 Saida temperature

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.holtwinters import ExponentialSmoothing
from sklearn.metrics import mean_squared_error
from sklearn.ensemble import RandomForestRegressor
import xgboost as xgb
```

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, SimpleRNN
from tensorflow.keras.callbacks import EarlyStopping

# --- Load and prepare data ---
df = pd.read_csv('../input/correc/corrected_temperature_data.csv')
df['Datetime'] = pd.to_datetime(df['Datetime'])
df.set_index('Datetime', inplace=True)
df = df[~df.index.duplicated(keep='first')]
df = df.asfreq('D')
df = df.fillna(method='ffill')
# --- Feature Engineering ---
def create_lag_features(data, lags=7):
    df_lag = data.copy()
    for i in range(1, lags + 1):
        df_lag[f'lag_{i}'] = df_lag['Temperature'].shift(i)
    df_lag.dropna(inplace=True)
    return df_lag

lags = 7
df_features = create_lag_features(df[['Temperature']], lags=lags)

# --- Train-Test Split ---
split_idx = int(len(df_features) * 0.8)
train = df_features.iloc[:split_idx]
test = df_features.iloc[split_idx:]

X_train = train.drop('Temperature', axis=1).values
y_train = train['Temperature'].values
X_test = test.drop('Temperature', axis=1).values
y_test = test['Temperature'].values

# --- Holt-Winters ---
```

```
hw_model = ExponentialSmoothing(train['Temperature'], trend='mul',
   seasonal='add', seasonal_periods=29)
hw_fit = hw_model.fit()
hw_forecast = hw_fit.forecast(steps=len(test))
hw_rmse = np.sqrt(mean_squared_error(y_test, hw_forecast))
print(f"Holt-Winters RMSE: {hw_rmse:.2f}")
# --- XGBoost ---
xgb_model = xgb.XGBRegressor(objective='reg:squarederror',
   n_estimators=100)
xgb_model.fit(X_train, y_train)
xgb_pred = xgb_model.predict(X_test)
xgb_rmse = np.sqrt(mean_squared_error(y_test, xgb_pred))
print(f"XGBoost RMSE: {xgb_rmse:.2f}")

# --- Random Forest ---
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
rf_pred = rf_model.predict(X_test)
rf_rmse = np.sqrt(mean_squared_error(y_test, rf_pred))
print(f"Random Forest RMSE: {rf_rmse:.2f}")
# --- Prepare data for LSTM and RNN ---
 X_train_rnn = X_train.reshape((X_train.shape[0], lags, 1))
X_test_rnn = X_test.reshape((X_test.shape[0], lags, 1))

# --- LSTM Model ---
lstm_model = Sequential([
    LSTM(50, activation='relu', input_shape=(lags, 1)),
    Dense(1)
])
lstm_model.compile(optimizer='adam', loss='mse')

early_stop = EarlyStopping(monitor='val_loss', patience=10,
   restore_best_weights=True)
```

```
lstm_model.fit(X_train_rnn, y_train, epochs=100, batch_size=16,
               validation_split=0.1, callbacks=[early_stop], verbose
                  =0)

lstm_pred = lstm_model.predict(X_test_rnn).flatten()
lstm_rmse = np.sqrt(mean_squared_error(y_test, lstm_pred))
print(f"LSTM RMSE: {lstm_rmse:.2f}")

# --- Simple RNN Model ---
rnn_model = Sequential([
    SimpleRNN(50, activation='relu', input_shape=(lags,1)),
    Dense(1)
])
rnn_model.compile(optimizer='adam', loss='mse')

rnn_model.fit(X_train_rnn, y_train, epochs=100, batch_size=16,
              validation_split=0.1, callbacks=[early_stop], verbose
                 =0)

rnn_pred = rnn_model.predict(X_test_rnn).flatten()
rnn_rmse = np.sqrt(mean_squared_error(y_test, rnn_pred))
print(f"Simple RNN RMSE: {rnn_rmse:.2f}")

# --- Plotting ---
plt.figure(figsize=(14, 6))
plt.plot(test.index, y_test, label='Actual Temperature', color='green
   ')
plt.plot(test.index, hw_forecast, label=f'Holt-Winters (RMSE={hw_rmse
   :.2f})', color='red')
plt.plot(test.index, xgb_pred, label=f'XGBoost (RMSE={xgb_rmse:.2f})'
   , color='orange', linestyle='--')
plt.plot(test.index, rf_pred, label=f'Random Forest (RMSE={rf_rmse:.2
   f})', color='purple', linestyle='--')
```

```
plt.plot(test.index, lstm_pred, label=f'LSTM (RMSE={lstm_rmse:.2f})',
    color='magenta', linestyle='-.')
plt.plot(test.index, rnn_pred, label=f'Simple RNN (RMSE={rnn_rmse:.2f
   })', color='brown', linestyle='-.')

plt.title('Forecasting Comparison on Test Data')
plt.xlabel('Date')
plt.ylabel('Temperature (ÂřC)')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

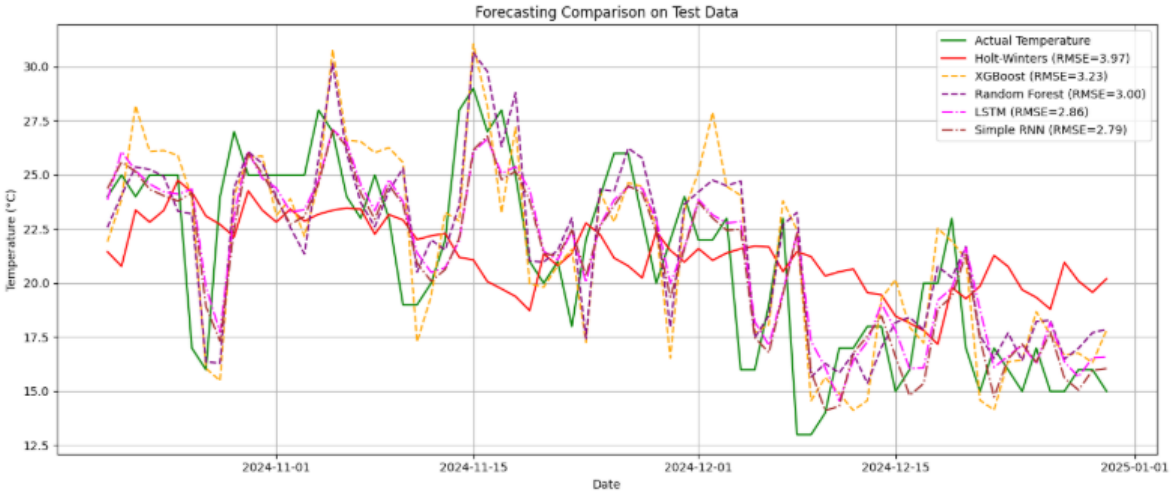

Figure 3.22: Saida temperature

3.8.3 Data on the electricity consumption of regular customers in Saida

Sonelgaz: or the Algerian Electricity and Gas Company, was established in 1969. Its main missions are the transportation and distribution of electricity, as well as the transportation and distribution of natural gas through pipelines within the national market.

Energy Types

The trading company, produces and purchases high-intensity energy, then gradually reduces it to reach consumers at medium or low intensity.

The electricity consumption data collected from Sonalgaz Saida is illustrated in the figure below.

	Ventes (GWh)			Ventes (GWh)	ACHATS DP_G	Chiffre d'Affaire (KDA)			
Eléments Elec	AO	FSM	MT	HTB		AO	FSM	MT	HTB
Janvier	30,53685	8,334383	13,0515342	4,586	38,5383816	84046,6836	50155,8655	48481,5889	11658,8389
Février	31,884858	8,050516	10,5888536	3,513	35,0050501	85308,1508	43358,0456	44863,5445	5383,35033
Mars	16,016538	5,306583	13,0453133	4,563	43,0383683	58680,138	38800,6568	44886,8584	11638,6486
Avril	33,058448	11,385456	13,1535546	4,505	45,5854869	88808,8155	63053,3489	45653,3086	11430,0084
Mai	36,450818	8,888165	13,1843554	5,454	46,5660885	100416,307	45443,0515	45534,3035	13531,5504
Juin	18,583518	8,104865	11,8316606	4,653	48,6848038	50880,1831	48531,8531	45386,5556	11461,8632
Juillet	33,545853	10,045488	13,3463804	3,816	51,3410888	133333,644	54653,0566	51855,5686	8538,45188
Août	45,188536	8,558834	13,3540888	4,456	66,0054835	183850,814	45841,0038	53330,5038	10450,8037
Septembre	38,31588	5,151838	11,6343836	4,636	41,4835553	111085,304	38351,3555	45503,5341	10815,3364
Octobre	33,645545	8,453554	10,5804564	5,343	38,3684835	143338,383	51514,1382	45545,5585	13353,5517
Novembre	33,568655	5,406655	10,8063013	5,04	36,5364006	86408,8638	38,5308305	55338,8819	13036,3566
Décembre	11,881155	0,585385	8,04545036	5,351	40,8558313	45544,3811	44030,1145	43514,3039	13554,3163
CUM	362,676654	88,271722	142,622332	55,916	532,009022	1171701,67	523771,021	569894,51	134853,076

Figure 3.23: Saida electricity consumption

Where AO (Abonnés ordinaires) value represents the electricity consumption of ordinary customers in low intensity, FSM (Facture Sur Mémoire) is the electricity consumption of the entities, MT is medium intensity, and HTB is the consumption of high intensity electricity. We have chosen to work with the AO data.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.holtwinters import ExponentialSmoothing
from sklearn.metrics import mean_squared_error
from sklearn.ensemble import RandomForestRegressor
import xgboost as xgb
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, SimpleRNN, Dense
from tensorflow.keras.callbacks import EarlyStopping
```

```
import seaborn as sns
# --- Load  Data ---
data = [
   17.364768,18.883886,13.888631,18.304709,17.344866,14.761669,
   66.68407,34.417864,66.866398,68.183489,60.708418,10.948368,
   47.88,47.85,43.58,84.78,83.49,48.45,34.99,40.36,88.05,34.84,
   84.88,44.84,30.53685,31.884858,16.016538,33.058448,36.450818,
   18.583518,33.545853,45.188536,38.31588,33.645545,33.568655,
   11.881155
]
date_range = pd.date_range(start='2022-01', periods=len(data), freq='
   MS')
df = pd.DataFrame({'Month': date_range, 'GWH': data})
df.set_index('Month', inplace=True)
color_pal = sns.color_palette()
plt.style.use('fivethirtyeight')
print(df)
df.plot(
        figsize=(15, 5),
        color=color_pal[0],
        title='GWH')
plt.show()
# --- Holt-Winters Forecast ---
split_index = int(len(df) * 0.8)
train = df.iloc[:split_index]
test = df.iloc[split_index:]
hw_model = ExponentialSmoothing(train['GWH'], trend='mul', seasonal='
   mul', seasonal_periods=12)
hw_fit = hw_model.fit()
hw_forecast = hw_fit.forecast(steps=len(test))
hw_rmse = np.sqrt(mean_squared_error(test['GWH'], hw_forecast))
print(f"Holt-Winters RMSE: {hw_rmse:.2f}")

# --- Create Lag Features ---
```

```
def create_lag_features(data, lags=12):
    df_lag = data.copy()
    for i in range(1, lags + 1):
        df_lag[f'lag_{i}'] = df_lag['GWH'].shift(i)
    df_lag.dropna(inplace=True)
    return df_lag

lags = 12
df_lagged = create_lag_features(df[['GWH']], lags)
# --- Split for ML and RNN/LSTM ---
split_idx = int(len(df_lagged) * 0.8)
train_ml = df_lagged.iloc[:split_idx]
test_ml = df_lagged.iloc[split_idx:]

X_train = train_ml.drop('GWH', axis=1).values
y_train = train_ml['GWH'].values
X_test = test_ml.drop('GWH', axis=1).values
y_test = test_ml['GWH'].values

# --- Random Forest ---
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
rf_pred = rf_model.predict(X_test)
rf_rmse = np.sqrt(mean_squared_error(y_test, rf_pred))
print(f"Random Forest RMSE: {rf_rmse:.2f}")

# --- XGBoost ---
xgb_model = xgb.XGBRegressor(objective='reg:squarederror',
   n_estimators=100)
xgb_model.fit(X_train, y_train)
xgb_pred = xgb_model.predict(X_test)
xgb_rmse = np.sqrt(mean_squared_error(y_test, xgb_pred))
print(f"XGBoost RMSE: {xgb_rmse:.2f}")
# --- Prepare for LSTM and RNN ---
```

```
X_train_rnn = X_train.reshape((X_train.shape[0], lags, 1))
X_test_rnn = X_test.reshape((X_test.shape[0], lags, 1))

# --- LSTM ---
lstm_model = Sequential([
    LSTM(50, activation='relu', input_shape=(lags, 1)),
    Dense(1)
])
lstm_model.compile(optimizer='adam', loss='mse')
early_stop = EarlyStopping(monitor='val_loss', patience=10,
   restore_best_weights=True)
lstm_model.fit(X_train_rnn, y_train, epochs=100, batch_size=8,
               validation_split=0.1, callbacks=[early_stop], verbose
                  =0)
lstm_pred = lstm_model.predict(X_test_rnn).flatten()
lstm_rmse = np.sqrt(mean_squared_error(y_test, lstm_pred))
print(f"LSTM RMSE: {lstm_rmse:.2f}")

# --- Simple RNN ---
rnn_model = Sequential([
    SimpleRNN(50, activation='relu', input_shape=(lags, 1)),
    Dense(1)
])
rnn_model.compile(optimizer='adam', loss='mse')
rnn_model.fit(X_train_rnn, y_train, epochs=100, batch_size=8,
              validation_split=0.1, callbacks=[early_stop], verbose
                 =0)
rnn_pred = rnn_model.predict(X_test_rnn).flatten()
rnn_rmse = np.sqrt(mean_squared_error(y_test, rnn_pred))
print(f"Simple RNN RMSE: {rnn_rmse:.2f}")
plt.figure(figsize=(14, 6))

# Plot the entire actual data (train + test)
plt.plot(df.index, df['GWH'], color=color_pal[0])
```

```
# Forecasts
plt.plot(test.index, hw_forecast, label=f'Holt-Winters (RMSE={hw_rmse
    :.2f})', color='red')
plt.plot(test_ml.index, rf_pred, label=f'Random Forest (RMSE={rf_rmse
    :.2f})', color='purple')
plt.plot(test_ml.index, xgb_pred, label=f'XGBoost (RMSE={xgb_rmse:.2f
    })', color='orange')
plt.plot(test_ml.index, lstm_pred, label=f'LSTM (RMSE={lstm_rmse:.2f
    })', color='magenta')
plt.plot(test_ml.index, rnn_pred, label=f'Simple RNN (RMSE={rnn_rmse
    :.2f})', color='brown')
plt.title('Forecasting: Train + Test vs HW vs ML vs LSTM/RNN')
plt.xlabel('Date')
plt.ylabel('GWH')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

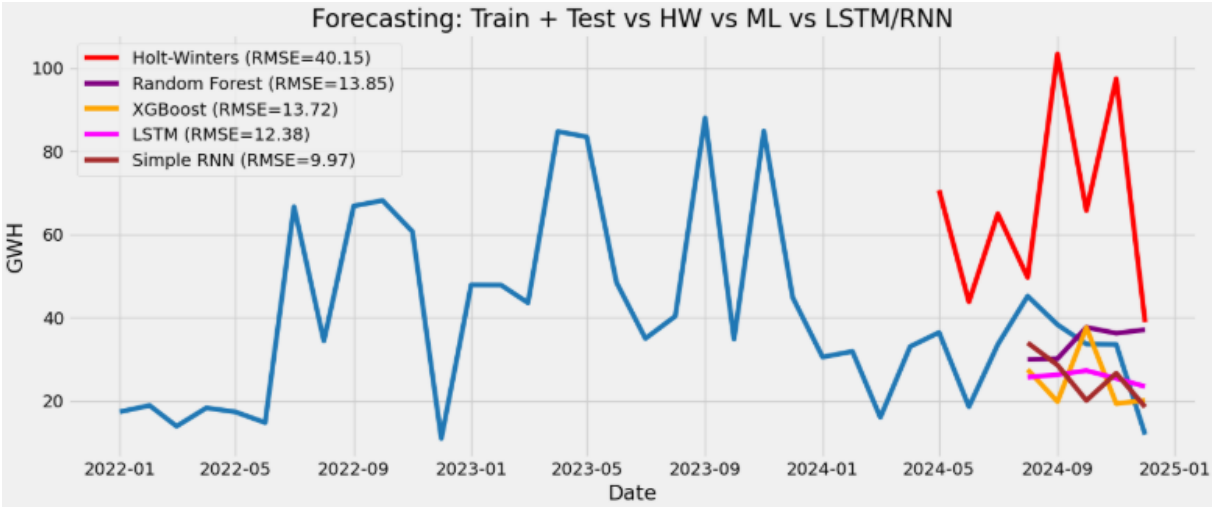

Figure 3.24: Data on electricity consumption of regular customers in Saida

From the comparative results of different forecasting methods, it is evident that AI techniques consistently demonstrate superior performance in time series forecasting. Numerous studies and empirical comparisons have shown the same thing, that machine learning models, particularly deep learning architectures such as multilayer perceptrons, outperform traditional

statistical methods like ARIMA in handling complex, non-stationary data patterns. Unlike classical approaches, AI models adapt flexibly to large, high-dimensional datasets and capture long-term dependencies more effectively, which enhances their predictive accuracy and robustness.

Bibliography

[1] Ali, Z. A., Abduljabbar, H., Tahir, A., Sallow, A. B., & Almufti, S. M. (2023). *Extreme gradient boosting algorithm with machine learning: A review*. Academic Journal of Nawroz University, 12(2), 320-334.

[2] Alpaydin, E. (2020). *Introduction to machine learning*. MIT Press.

[3] Aydin, S. (2022). *Time series analysis and some applications in medical research*. Journal of Mathematics and Statistics Studies, 3(2), 31-36.

[4] Bentéjac, C., Csörgo, A., & Martínez-Muñoz , G. (1911). *A comparative analysis of xgboost*. arXiv preprint arXiv:392.

[5] Brockwell, P. J., & Davis, R. A. (1991). *Time series: theory and methods*. Springer Science & Business Media.

[6] Chatfield, C. (2013). *The analysis of time series: theory and practice*. Springer.

[7] Chen, L. P. (2019). *Foundations of machine learning*. The MIT Press, Cambridge, MA, 2018, 504 pp.

[8] Chen, T., & Guestrin, C. (2016, August). *XGBoost: A scalable tree boosting system*. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 785-794).

[9] Choi, R. Y., Coyner, A. S., Kalpathy-Cramer, J., Chiang, M. F., & Campbell, J. P. (2020). *Introduction to machine learning, neural networks, and deep learning*. Translational Vision Science & Technology, 9(2), 14.

[10] Ensafi, Y., Amin, S. H., Zhang, G., & Shah, B. (2022). *Time-series forecasting of seasonal items sales using machine learning A comparative analysis*. International Journal of Information Management Data Insights, 2, 100058.

[11] Hochreiter, S., Schmidhuber, J. (1997). *Long short-term memory*. Neural computation, 9(8), 1735-1780.

[12] Jose, J. (2022). *Introduction to time series analysis and its applications*. Christ University, Bangalore.

[13] Kendall, M. G., & Company Ltd. (1976). *Time series*. London and High Wycombe.

[14] Kohzadi, N., Boyd, M. S., Kermanshahi, B., & Kaastra, I. (1996). *A comparison of artificial neural network and time series models for forecasting commodity prices*. Neurocomputing, 10, 169-81.

[15] Kuhlmann, L. & Pauly, M. (2023). *A dynamic systems model for an economic evaluation of sales forecasting methods*. Technische Glas, 17, 397-404.

[16] Marsland, S. (2011). *Machine learning: An algorithmic perspective*. Chapman and Hall/CRC.

[17] Mitchell, T. M. (1997). *Machine learning*. McGraw-Hill.

[18] Okut, H. (2021). *Deep learning for sub typing and prediction of diseases: Long-short term memory*. In Deep Learning Applications. IntechOpen.

[19] Pang, S., Pang, C., Zeng, D., & Chen, X. (2022, June). *Exploration on the talent training path of "Post, Course, Competition, Certificate" for Artificial Intelligence Technology Application Major based on Huawei 1+X Certificate*. In 2022 8th International Conference on Humanities and Social Science Research (ICHSSR 2022) (pp. 2212-2215). Atlantis Press.

[20] Peng, Z. (2022). *Development model of artificial intelligence education course involving high-tech enterprises*. Scientific Journal of Technology, 4(9).

[21] Salman, H. A., Kalakech, A., & Steiti, A. (2024). *Random forest algorithm overview*. Babylonian Journal of Machine Learning, 2024, 69-79.

[22] Shukla, M., & Jharkharia, S. (2011). *ARIMA models to forecast demand in fresh supply chains*. International Journal of Operations Research, 11(1), 1-18.