UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Tècnica Superior d'Enginyeria
de Telecomunicació de Barcelona

Time series forecasting using SARIMA and SANN models

A Degree Thesis submitted to the Faculty of the
Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona
Universitat Politècnica de Catalunya
by

Arnau Gispert Becerra

In partial fulfillment of the requirements for the degree in

BACHELOR'S DEGREE IN TELECOMMUNICATIONS TECHNOLOGIES AND SERVICES ENGINEERING

Advisor: Robert Bešťák
Co-Advisor: Alberto Aguasca Sole
Co-Advisor: Zagroz Aziz

Prague, Czech Republic. Spring 2021

FAKULTA
ELEKTROTECHNICKÁ
ČVUT V PRAZE

Abstract

Information and communications technology has evolved to the point of being present in most things in our daily lives. Even the simplest object that everyone has in their home is getting smarter, like toothbrushes, cars, phones and so on. All that devices are connected to the Internet to make our life easier.

The question is, how is all that amount of data processed?
Here is when Artificial Intelligence appears. AI is the part of ICT dedicated to the development of algorithms that allows a machine to make intelligent decisions or, at least, behave as if it has a human-like intelligence.
The use of AI is present in many sectors such finance, health, transport, or even agriculture. Machine Learning is a branch of AI based on the idea that computer systems can learn on their own from data.
Data science has implemented Machine Learning algorithm such as the Artificial Neural Network to work with Statistics and Linear Regression for data processing.
An ANN is the piece of a computing system designed to simulate the way the human brain analyses and processes information. It is the foundation of AI and solves problems that would prove impossible or difficult by human or statistical standards.

But, is this resource always the best solution?
This paper is about a comparison between Seasonal Artificial Neural Network with classic models as Seasonal Autoregressive Integrated Moving Average for rainfall forecasting.

The project started by doing an introduction to Deep Learning and Machine Learning. Afterwards, the process of obtaining an adequate amount of data to create a proper dataset began.
To do that, we used data from of some pluviometers distributed over the Hauts-de-Seine territory in France. With data from 2009 to 2020 of 19 sensors, the dataset was used to experiment with different algorithms and different configurations to obtain different predictions.

The forecasting performance of SARIMA model and that of SANN were compared with four forecast performance measures:
- Mean Forecast Error, Mean Absolute Error, Mean Squared Error and Root Mean Squared Error.
Not only will the accuracy of the model be taken into account, but also the runtime and implementation requirements will be used as a benchmark.

Finally, all models were tested in the same work environment and a conclusion was reached thanks to the results obtained from different reference points.

Resum

Les tecnologies de la informació i de la comunicació han evolucionat fins al punt d'estar presents en la majoria de coses de la nostra vida quotidiana. Fins i tot l'objecte més senzill que tothom té a casa és cada vegada més intel·ligent, com ara raspalls de dents, cotxes, telèfons, etc. Tots aquests dispositius estan connectats a Internet per facilitar-nos la vida.

La pregunta és: com es processa tota aquesta quantitat de dades?
Aquí és quan apareix la Intel·ligència Artificial. La IA és la part de les TIC dedicada al desenvolupament d'algoritmes que permet a una màquina prendre decisions intel·ligents o, si més no, comportar-se com si tingués una intel·ligència semblant a la humana.
L'ús de la IA està present en molts sectors, com el financer, la salut, el transport o fins i tot l'agricultura. L'aprenentatge automàtic és una branca de la IA basada en la idea que els sistemes informàtics poden aprendre sols a partir de dades.
La ciència de les dades ha implementat un algoritme d'aprenentatge automàtic, com és la Xarxa Neuronal Artificial, per treballar amb estadístiques i regressió lineal per al processament de dades. Una ANN és la part d'un sistema informàtic dissenyat per simular la manera com el cervell humà analitza i processa la informació. És el fonament de la IA i resol problemes que resultarien impossibles o difícils per als estàndards humans o estadístics.

Però, aquest recurs és sempre la millor solució?
Aquest article tracta sobre una comparació entre la Xarxa Neuronal Artificial Estacionals amb models clàssics com la Model Auto Regressiu Integrat de Mitjans Mòbils Estacional per a la predicció de pluges.

El projecte va començar fent una introducció a l'Aprenentatge Profund i l'Aprenentatge Automàtic. Després, es va iniciar el procés d'obtenció d'una quantitat adequada de dades per crear un conjunt de dades necessari.
Per fer-ho, hem utilitzat les dades d'alguns pluviòmetres distribuïts pel territori dels Hauts-de-Seine a França. Amb dades des del 2009 fins al 2020 de 19 sensors, el conjunt de dades es va utilitzar per experimentar amb diferents algoritmes i diferents configuracions per obtenir prediccions diferents.

El rendiment de la predicció del model SARIMA i el de SANN es van comparar mitjançant quatre mesures de rendiment:
- L'Error de Previsió Mitjà, l'Error de Previsió Absolut, l'Error Quadràtic Mig i l'Arrel de l'Error Quadràtic Mig.
No només es tindrà en compte la precisió del model, sinó que també s'utilitzaran els requisits d'execució i d'implementació com a benckmark.

Finalment, tots els models es van provar en el mateix entorn de treball i es va arribar a una conclusió gràcies als resultats obtinguts de diferents punts de referència.

Resumen

Las Tecnologías de la información y la comunicación han evolucionado hasta el punto de estar presentes en la mayoría de las cosas de nuestra vida diaria. Incluso el objeto más simple que todos tienen en su hogar se está volviendo más inteligente, como cepillos de dientes, automóviles, teléfonos, etc. Todos esos dispositivos están conectados a Internet para hacernos la vida más fácil.

La pregunta es, ¿cómo se procesa toda esa cantidad de datos?
Aquí es cuando aparece la Inteligencia Artificial. La IA es la parte de las TIC dedicada al desarrollo de algoritmos que permite que una máquina tome decisiones inteligentes o, al menos, se comporte como si tuviera una inteligencia similar a la humana.
El uso de la IA está presente en muchos sectores como las finanzas, la salud, el transporte o incluso la agricultura. El aprendizaje automático es una rama de la inteligencia artificial basada en la idea de que los sistemas informáticos pueden aprender por sí mismos a partir de datos.
La ciencia de datos ha implementado un algoritmo de aprendizaje automático como la Red Neuronal Artificial para trabajar con estadísticas y regresión lineal para el procesamiento de datos.
Una ANN es la parte de un sistema informático diseñado para simular la forma en que el cerebro humano analiza y procesa la información. Es la base de la IA y resuelve problemas que resultarían imposibles o difíciles según los estándares humanos o estadísticos.

Pero, ¿Es este recurso siempre la mejor solución?
Este artículo trata de una comparación entre la Red Neural Artificial Estacional con modelos clásicos como Modelo Autorregresivo Integrado de Media Móvil Estacional para el pronóstico de lluvia.

El proyecto comenzó con una introducción al Aprendizaje Profundo y al Aprendizaje Automático. Posteriormente, comenzó el proceso de obtener una cantidad adecuada de datos para crear un conjunto de datos necesario.
Para ello, utilizamos datos de algunos pluviómetros distribuidos en el territorio de Hauts-de-Seine en Francia. Con datos desde el 2009 hasta 2020 de 19 sensores, el conjunto de datos se utilizó para experimentar con diferentes algoritmos y diferentes configuraciones para obtener diferentes predicciones.

El rendimiento de pronóstico del modelo SARIMA y el de SANN se compararon con cuatro medidas de rendimiento:
- Error de Pronóstico Medio, Error de Pronóstico Absoluto, Error Cuadrático Medio y Raíz del Error Cuadrático Medio.
No solo se tendrá en cuenta la precisión del modelo, sino que también se utilizarán como punto de referencia el tiempo de ejecución y los requisitos de implementación.

Finalmente, todos los modelos fueron probados en el mismo entorno de trabajo y se llegó a una conclusión gracias a los resultados obtenidos de diferentes puntos de referencia.

Dedication: *"Dedico aquest projecte a la meva família, que sempre ha cregut en mi. Als meus amics que m'han fet costat des de la distància i als meus companys d'Erasmus amb els quals he viscut tantes vivències inoblidables."*

Acknowledgment

First of all, I would like to thank Robert Bešťák for giving me the opportunity to be part of this project and Zagroz Aziz for guiding and helping me throughout.
Thanks to this I have been able to work on this amazing project that has not only helped me to improve my skills in Machine Learning and Python, but also made me grow as an Engineer.

I would like to thank Alberto Aguasca from the Signal Theory and Communications Department at the UPC, Barcelona to be my final project tutor at my home university.

I would also like to thank all my friends, whether from Catalonia or Prague, for being by my side whenever I needed them. Not without forgetting my flatmate Marc Gracia, with whom I have lived all this period with many good and not so good experiences.

Finally, I have to express my eternal gratitude to my family for teaching me how to grow, for listening to my career progress even if they did not understand anything I said, and for supporting me every step of the way.

Revision history and approval record

Revision	Date	Purpose
0	04/05/2021	Document creation
1	16/05/2021	Document revision
2	21/05/2021	Document submission

DOCUMENT DISTRIBUTION LIST

Name	e-mail
Arnau Gispert Becerra	
Robert Bešťák	
Alberto Aguasca Sole	
Zagroz Aziz	

Written by:		Reviewed by:	
Date	21/05/2021	Date	dd/mm/yyyy
Name	Arnau Gispert	Name	Robert Bešťák Alberto Aguasca
Position	Project Author	Position	Project Supervisor

Contents

List of Figures

Listings

List of Tables

1 Project Plan

1.1 Introduction

In this fast-paced world where we live, it is impossible to be an expert in everything; that is why we use technology to help us with many day to day activities. With the arrival of IoT, cities are becoming smarter, what means that we are surrounded by a multitude of sensors. From an autonomous driving car to the vacuum you have at home, they produce a certain amount of data.

This is where the problem arises; how can we manage this amount of data?
We cannot improve people's lives just by giving them technological improvements if we do not know how to manage them. Artificial Intelligence has entered the game to the field. Machine Learning is a branch of AI based on the idea that systems can learn from data, identify patterns and make decisions with minimal human intervention.

In the data science world, the use of AI in conjunction with classic models is becoming more and more common. The self-learning capability helps to handle a much larger amount of data than was possible in the past.
Is it possible that in the future that AI will eventually replace the use of classical models such as linear regression and statistics?
The aim of this study is to introduce Artificial Intelligence to the debate.

1.2 Statement of purpose

The aim of this project is to implement and compare the performance between a classical model and an AI-based model for time series forecasting.
In order to make that comparison, we use the same data in all models.
The project is divided in the following sections:

- **Classical model.** Study and implementation of a linear regression model for seasonal forecasting.
- **AI-based model** Study and implementation of different univariate ML models.

Once all models are implemented, we use a dataset to compare the forecast performance. From that dataset, we forecast next 12 month.

1.3 Requirements and specifications

- **Requeriments:**
  In order to choose the best model for time series forecasting, we first have to create a dataset for do all testing. Once the dataset is created, we do a study of each model to choose the best parameters.

- **Specifications:**
  In order to choose the best parameters of each model:

  1. **SARIMA model.** Seasonal Autoregressive Integrated Moving Average.
     A near regression model for seasonal forecasting composed by two parts: non-seasonal parameters (p,d,q) from ARIMA and seasonal parameters (P,D,Q,s).
  2. **ANN-based model.** Neural Network is the part of ML used to simulate the way the human brain analyzes and processes information to forecast. There are different types, the one used in that project are:
     - **LSTM:** Long short-term memory is an RNN architecture used in the field of deep learning.
     - **CNN:** Convolutional Neural Networks is a type of neural network used for time series.
  3. **Prophet:** is an additive regression model developed by Facebook and designed for automatic forecasting of univariate time series data

  All those models have been tested under same dataset. This dataset is based on data from 19 pluviometers distributed in the Hauts-de-Seine territory in France.
  For Train/Test set is used data from 2009 to 2020 provided from French public services.

  The forecast performance is based on the following parameters:

  - **MFE:** Mean forecast error (or Forecast Bias)
  - **MAE:** Mean absolute error
  - **MSE:** Mean squared error
  - **RMSE:** Root-mean-square error
  - **Runtime:** (or executing time) is the period needed to do the forecast for each model.

1.4 Methods and procedures

This project is a sub-part of a larger project. It is not a continuation of any project, it is made from scratch.
This project was proposed by PhD Eng. Robert Bešťák to the student Arnau Gispert Becerra of the Telecommunication Engineering degree at ETSETB-UPC and accepted by Alberto Aguasca Sole from UPC. It was supervised by Ing. Zagroz Aziz.

From CTU, through PhD Eng. Robert Bešťák, a Python script and some documents were provided to Arnau Gispert Becerra. This Python script contained a similar unfinished student project. The documents related information about the project that had been done before.

The project is done in Python language in different scripts and Jupyter Notebook by Arnau Gispert Becerra.

1.5 Milestones

This project is composed with different parts:

- **Python scripts:** A script of each model prepared to be executed in a environment such Spyder IDE
- **Jupyter Notebook:** A Jupyther Notebook divided in different steps of the execution for each model.
- **Data:** A CSV file with the rainfall values from all pluviometers and the date when they were taken.
- **Requirement file:** A text file with a list of all of a project's dependencies.

All executables have notes so you know what each part does.

1.6 Gantt Diagram

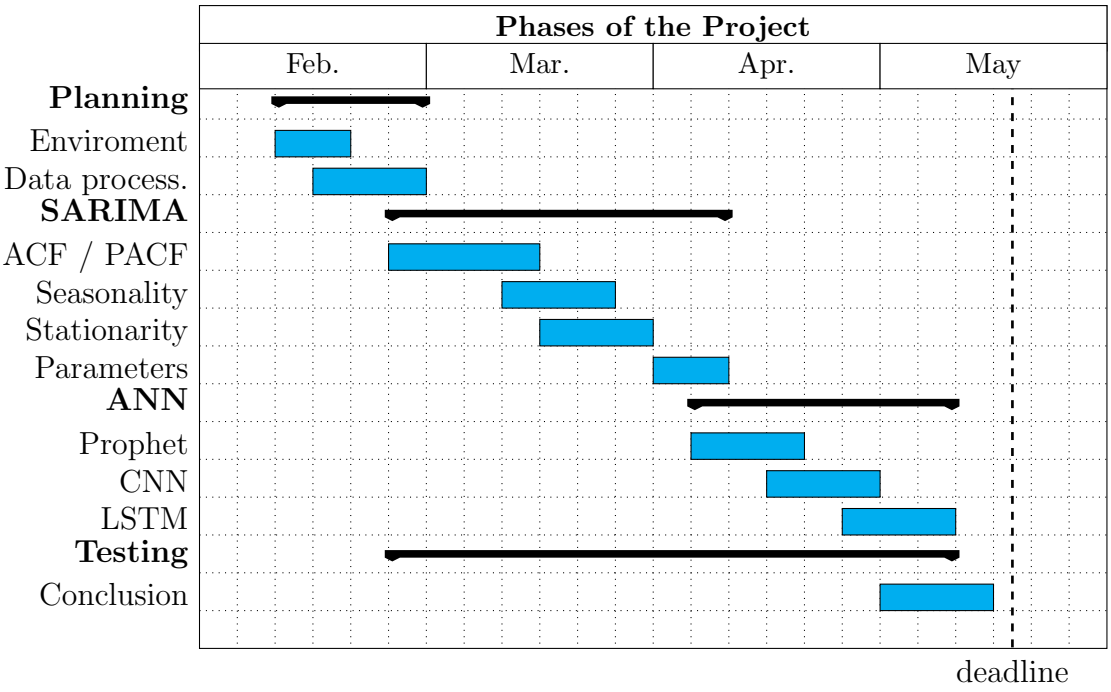

Figure 1: Project's Gantt diagram

1.7 Deviations of the original plan and and incidences

The project started in mid-February 2021. This year has been complicated due to SARS Covid-19 pandemic. Mobility limitations, prevention measures, government limitations and other pandemic-related issues made us to adapt how to manage all project development. On the other hand, we were aware about what the situation would be like as this is not the first semester with these circumstances, so adapting the work plan was not a big deal.

The start of the project is a bit uncertain. Due to the above mentioned situation, starting a project in another university and another language made the first weeks not very productive. Despite this, the first part which dealt with how to manage all the project data, progressed fruitfully as it was very similar to previous work.
The problems arose basically with the SARIMA model, all its parameters have to be studied exhaustively in order to be able to apply the model well. The fact of having to apply several mathematical functions and having to analyse them properly made it take longer than expected.
Fortunately, having learned a lot with SARIMA model, when it was time to apply the AI-based models, it was much easier, because the scripts were already prepared. Thanks to this, I was able to implement a larger number of models more efficiently. This meant that the final results had even more models to compare and draw better conclusions from.
In conclusion, the project was completed on time, with more resources than expected.

2 State of the art of the technology used or applied in this thesis

In this section, there will be an in-depth description of the implementation techniques used in the project. The study will contain information about the different prediction models as well as the different algorithms used during the whole process.

2.1 Data acquisition

All data used in this study is provided from 19 pluviometers distributed over the Hauts-de-Seineterritory. That data is provided from French public services.
There you obtain a CSV or JSON file with data of each sensor and the date it was taken from 2009.
In this project we use CSV file. Using *Python* library, we can convert that CSV file into a dataset[1], where *pluviometrie.csv* is the CSV file downloaded from French public service database.

2.2 Dataset structure

The dataset used for this project is structured with the following indicators:

- **Date**
- **Pluviometer values**

2.2.1 Date

The index of the dataset is the date when the sample was taken.
That value is expressed in **DateTime**[2] format. This is used in several languages in order to have an index with date and time. The dataset is sorted from oldest to newest.

[1] A dataset is basically a collection of data consisting of roughly two components: rows and columns. In addition, a key feature of a data set is that it is organized so that each row contains one observation.

[2] The **DateTime** value type represents dates and times with values ranging from 00:00:00 (midnight), January 1, 0001 Anno Domini (Common Era) through 11:59:59 P.M., December 31, 9999 A.D. (C.E.) in the Gregorian calendar.

2.2.2 Pluviometer values

The dataset consists of the rainfall measures from 19 pluviometers.
These pluviometers are rain gauges with tilting speeds. Each one consists of a small bucket with a receiving cone. When the maximum capacity (0.2 mm) of the bucket is reached, the pivot switches. The lever counting system then has a tipping point. The total number of switches is then continuously transmitted by the acquisition and teletransmission equipment connected to the pluviometer.
The system therefore has an accuracy of 0.2 mm.

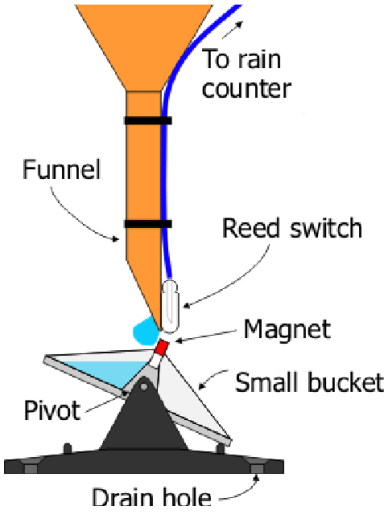

Figure 2: Pluviometrie scheme

2.3 Model classification

As mentioned in the specification section in **Requirements and specifications** the following models have been used during the thesis:

- **SARIMA:** Seasonal Autoregressive Integrated Moving Average
- **ANN:** Artificial Neural Network
  - **CNN:** Convolutional Neural Network
  - **RNN:** Recurrent Neural Network
    - Vanilla LSTM
    - Stacked LSTM
    - Bidirectional LSTM
    - CNN LSTM
    - ConvLSTM
- **Prophet**

2.3.1 SARIMA model

The *Autoregressive Integrated Moving Average* is one of the most used univariate time series methods for forecasting. Despite the fact that this method can handle time series data with trend component, it does not support data with seasonal component. That is a time series with a repeating cycle. Here is when SARIMA appears. SARIMA is an extension of ARIMA that support time series modeling with seasonal component.

SARIMA is the evolution of the grouping of several time series models.

- The **S** stands for **Seasonality**
- The **AR** stands for **Autoregressive**
- The **I** stands for **Integrated**
- The **MA** stands for **Moving Average**

For each time series model we have its corresponding parameter with its seasonality component

$$SARIMA(p,d,q)(P,D,Q)s \tag{1}$$

- **p** and seasonal **P:** indicate number of autoregressive terms (lags of the stationarized series)
- **d** and seasonal **D:** indicate differencing that must be done to stationarize series)
- **q** and seasonal **Q:** indicate number of moving average terms (lags of the forecast errors)
- **s:** indicates seasonal length in the data

Auto-Regressive (AR) model uses the dependent relationship between an observation and some number of lagged observations. *p is a parameter of how many lagged observations to be taken in.*

$$Y_t = \alpha + \beta_1 Y_{t-1} + \beta_2 Y_{t-2} + \cdots + \beta_p Y_{t-p} + \epsilon_1 \tag{2}$$

Moving Average (MA) model uses the dependency between an observation and a residual error from a moving average model applied to lagged observations. *q is a parameter of how many lagged observations to be taken in.* Contrary to the AR model, the finite MA model is always stationary.

$$Y_t = \alpha + \epsilon_t + \phi_1 \epsilon_{t-1} + \phi_2 \epsilon_{t-2} + \cdots + \phi_q \epsilon_{t-q} \tag{3}$$

Auto-Regressive Moving Average (ARMA) model is a combination from the *Auto-Regressive (AR)* and the *Moving Average (MA)* models.
In this model, the impact of previous lags along with the residuals is considered for forecasting the future values of the time series.

$$Y_t = c + \epsilon_t + \sum_{i=1}^{p} \phi_i Y_{t-i} + \sum_{i=1}^{q} \theta_i \epsilon_{t-i} \tag{4}$$

Auto-Regressive Integrated Moving Average (ARIMA) model is the evolution of *Auto-Regressive Moving Average (ARMA) model* with *Integrated (I)* part. So ARIMA model is a combination of a number of differences already applied on the model in order to make it stationary, the number of previous lags along with residuals errors in order to forecast future values.

$$Y_t = \alpha + \beta_1 Y_{t-1} + \beta_2 Y_{t-2} + \cdots + \beta_p Y_{t-p} \epsilon_t + \phi_1 \epsilon_{t-1} + \phi_2 \epsilon_{t-2} + \cdots + \phi_q \epsilon_{t-q} \tag{5}$$

2.3.2 Artificial Neural Network

ANN is an automatic learning and processing paradigm inspired by the way the nervous system of animals works. It is a system of interconnecting neurons in a network that collaborate to produce an output stimulus. This set of interconnected artificial neurons uses a mathematical or computational model of data processing based on a connectionist approach to computing.

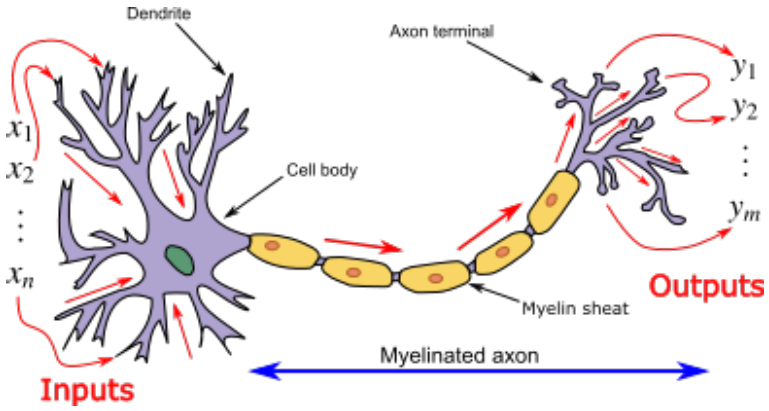

Figure 3: Input-Output of neuron and myelinated axon

CNN model is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other.
Although traditionally CNN was developed for two-dimensional image data, CNNs can be used to model univariate time series forecasting problems.

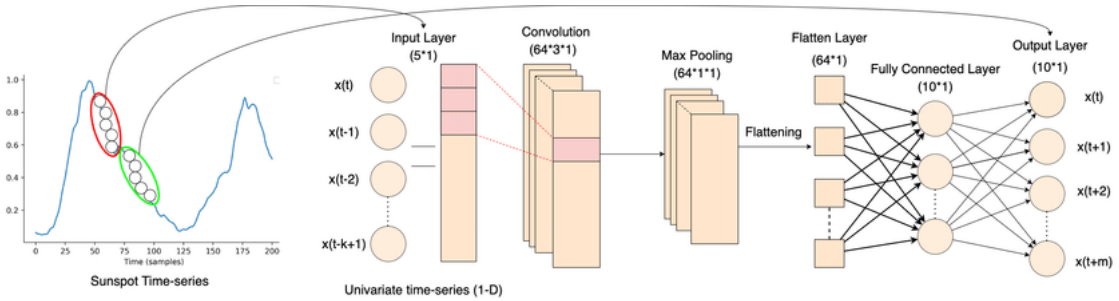

Figure 4: Convolutional Neural Network

RNN model is a class of ANNs where connections between nodes form a directed graph along a temporal sequence. This allows it to exhibit temporal dynamic behavior.

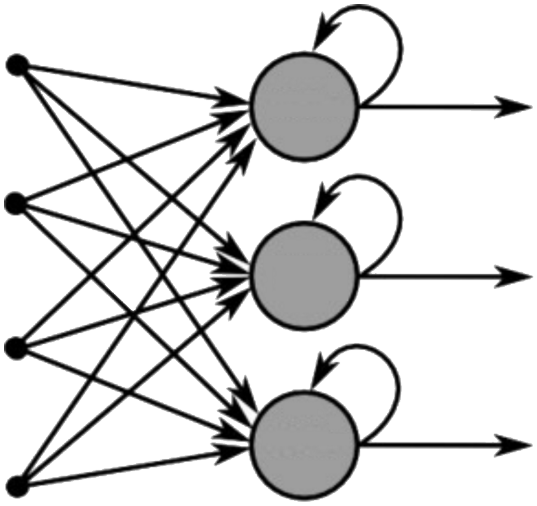

Figure 5: Recurrent Neural Network

Long short-term memory is an artificial *recurrent neural network* architecture used in the field of deep learning.
LSTMs can be used to model univariate time series forecasting problems. These are problems comprised of a single series of observations and a model is required to learn from the series of past observations to predict the next value in the sequence.

- LSTM Vanilla is an LSTM model that has not been customized from its original form. It has a single hidden LSTM units layer, and an output predictions layer.
Key in the definition is the shape of the input; as the model is working with univariate series, the number of features is one for each variable.

- LSTM Stacked is a stack of LSTM units one on top of another, necessary to realise a functional and robust solution. It has a multiple hidden LSTM units layer.
An LSTM layer above provides a sequence output rather than a single value output.

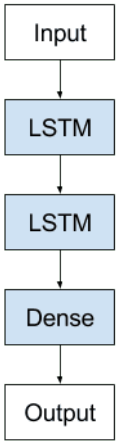

Figure 6: LSTM Stacked scheme

- LSTM Bidirectional connect two hidden layers of opposite directions to the same output. It allows the output layer to get information from past (backwards) and future (forward) states simultaneously.

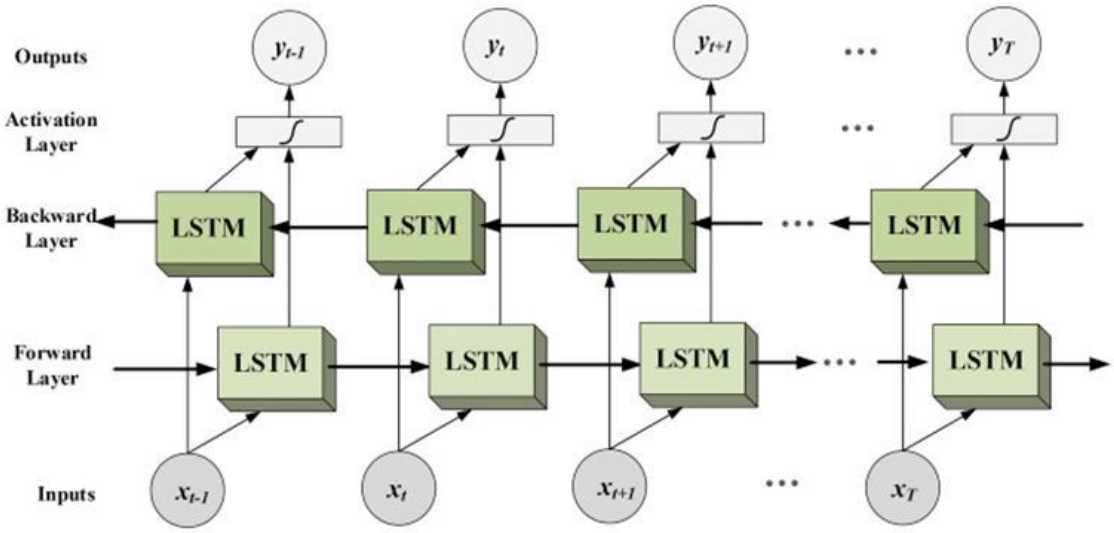

Figure 7: LSTM Bidirectional scheme

- LSTM CNN architecture involves using Convolutional Neural Network layers for feature extraction on input data combined with Long Short-Term Memory's to support sequence prediction.
A CNN model can be used in a hybrid model with an LSTM backend where the CNN is used to interpret subsequences of input that together are provided as a sequence to an LSTM model to interpret.

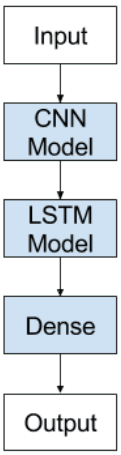

Figure 8: LSTM CNN scheme

- ConvLSTM is a further extension of the CNN-LSTM approach that performs the convolutions of the CNN as part of the LSTM for each time step.
It was developed for reading two-dimensional spatial-temporal data, but can be adapted for use with univariate time series forecasting.

2.3.3 Prophet model

Prophet, or "Facebook Prophet," is an open-source library for univariate (one variable) time series forecasting developed by Facebook.
Prophet implements what they refer to as an additive time series forecasting model, and the implementation supports trends, seasonality, and holidays. It is robust to missing data and shifts in the trend, and typically handles outliers well.

2.4 Model performance parameters

In order to test the performance of each model, we use a set of different accuracy measurement parameters as mentioned in section **Requirements and specifications** to classify the prediction error and executing time needed.

2.4.1 Forecast Error

The **forecast error** is calculated as the difference between the actual or real and the predicted or forecast value of a time series or any other phenomenon of interest.

The **forecast error** e is the result of the following equation:

$$e = y - \hat{y} \tag{6}$$

Where y is the observation and $\hat{y}$ denote the forecast of y based on all previous observations.

2.4.2 Mean Forecast Error

The **mean forecast error** is the arithmetic mean of the **forecast error**:

$$\mathrm{MFE} = \frac{\sum_{i=1}^{n} y_i - \hat{y}_i}{n} = \frac{\sum_{i=1}^{n} e_i}{n} \tag{7}$$

Where e_i is the **forecast error** of each sample and n is the length of the dataset.

2.4.3 Mean Absolute Error

The **mean absolute error**, is calculated as the average of the absolute forecast error values, where all of the forecast error values are forced to be positive.

$$\mathrm{MAE} = \frac{\sum_{i=1}^{n} |y_i - \hat{y}_i|}{n} = \frac{\sum_{i=1}^{n} |e_i|}{n} \tag{8}$$

2.4.4 Mean Square Error

The **mean square error**, is calculated as the average of the squared forecast error values. Squaring the forecast error values forces them to be positive; it also has the effect of putting more weight on large errors.

$$\mathrm{MSE} = \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{n} = \frac{\sum_{i=1}^{n} e_i^2}{n} \tag{9}$$

2.4.5 Root-mean-square error

The **root-mean-square error** represents the square root of the second sample moment of the differences between predicted values and observed values or the quadratic mean of these differences.

$$\text{RMSE} = \sqrt{MSE} = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}} \tag{10}$$

2.4.6 Runtime

In a computer, the **runtime** (or execution time) is the time interval in which a program runs on an operating system. This time starts when the program is put into main memory and the operating system starts executing its instructions.

3 Methodology

The process followed in this project has been: researching the important topics (Machine Learning, forecasting models and rainfall data), the process of obtaining and adjusting the data to create the dataset, and the implementation of all models on the dataset. A separate script has been created for each model to facilitate its execution.
Last part of the project consist that all prediction models have been tested under the same work environment and the same dataset. All results are shown in graphs comparing the model with the data obtained and the forecast of one year. To measure the accuracy of each model, forecast performance parameters have been used to reach a final conclusion of the project.

3.1 Machine Learning and Rainfall forecast research

Researching about Machine Learning and prediction models, I discovered that the best way to apply them is through languages such R and Python. As Python is one of the most popular programming languages and based on the fact that I already had a previous acknowledgement, I decided to implement the whole project with this language.
To learn about Machine Learning and classification algorithms, I enrolled the course *Machine Learning & Deep Learning in Python & R*[3] which focuses on how to use the different algorithms such ANN, CNN and SARIMA, and focuses mainly on a more practical use of Machine Learning.

3.2 Dataset creation

For the creation of the dataset used in the project tests, it was decided to use data from the rain gauges of the Hauts-de-Seine territory. To do so, we go with the following link to the web portal of the public services of the French government where we can obtain the required data.
There we find different ways of obtaining the data, either in JSON format or in CSV format. Despite the fact that JSON format has many advantages over CSV, in our case the CSV format was more useful because the data was organised by columns, which made it easier to create the desired dataset.
As each model has its own script, we have tried to make them as similar to each other as possible. Therefore, all models use the same dataset creation, and only the way of implementing the model changes in each case.

```
pluviometrie_csv = pd.read_csv("pluviometrie.csv",';')
pluviometrie_csv['date'] = pd.to_datetime(pluviometrie_csv['date'])
pluviometrie_csv.sort_values(by=['date'], inplace=True)
pluviometrie_csv = pluviometrie_csv.set_index('date')
pluviometrie_csv.insert(0, "rainfall", pluviometrie_csv.mean(axis=1))
```

Listing 1: Dataset index and sort of values

[3] an Udemy course based on the field of Data Science, Machine Learning, Python, R and Deep Learning.

With the creation of *pluviometrie_csv* we have a matrix with columns for date, rainfall mean and every pluviometer. The rainfall mean value is the mean from all sensors for each day.
That matrix is sorted by the index columns, which is the date column in order to have the dataset organized from the oldest to the newest parameter.
To implement all models, we create an auxiliary vector named **df** with just date and rainfall mean columns.

```
df = pluviometrie_csv.loc[:, :'rainfall']
df = pluviometrie_csv.loc[:, :'rainfall'].resample('M').mean()
```

Listing 2: DataFrame vector

For this project, we use a monthly mean of all sensors. If you want to have more data and therefore more precision, you can omit the last line so that the **df** vector only has the daily and not the monthly average value of all the sensors.

3.3 SARIMA model

In order to apply the model appropriately, the parameters mentioned in section *Model classification* have to be found. In order to find them, a step-by-step analysis will have to be done to choose them manually.

3.3.1 Stationarity

In time series data, stationarity means that the statistical properties of a process do not change over time. In general, a stationary time series will have no predictable patterns in the long-term. This is important because stationary processes are easier to analyze, detect and model.

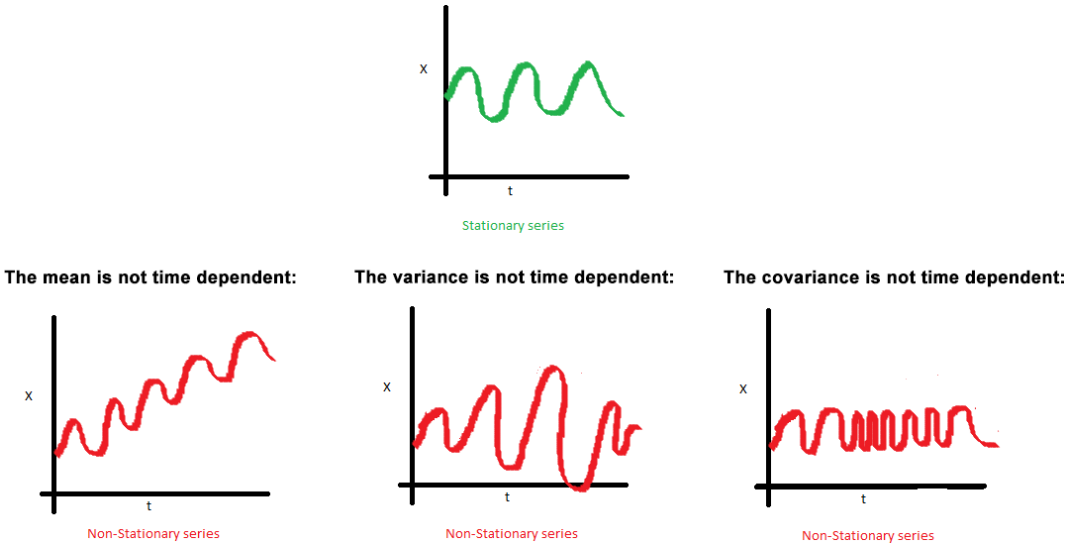

Figure 9: Data stationarity meaning

Dickey-Fuller test seeks to determine the existence or non-existence of unit roots in a time series. The null hypothesis of this test is that a unit root exists in the series. We can check the stationarity of the data thanks to this test.

```
Dickey-Fuller test results:

ADF = -5.16878383817796
p-value = 1.0178254081600643e-05
  1%: -3.479
  5%: -2.883
  10%: -2.578

p-value is less than or equal to 0.05
reject the null hypothesis because the data does not have a unit root
   and is stationary.
```

Listing 3: Dickey-Fuller test

The null hypothesis is rejected (p-value < 0.05) for this test because the data is stationary.

3.3.2 ACF and PACF Plots

In order to choose the right SARIMA parameters, it is necessary to analyse the data to find the best fit.
Applying the *Autocorrelation function* we can find the correct **q parameter** from the Moving Average part of the model.
With the *Partial Autocorrelation function*, we can find the correct **p parameter** from the Auto Regressive part.

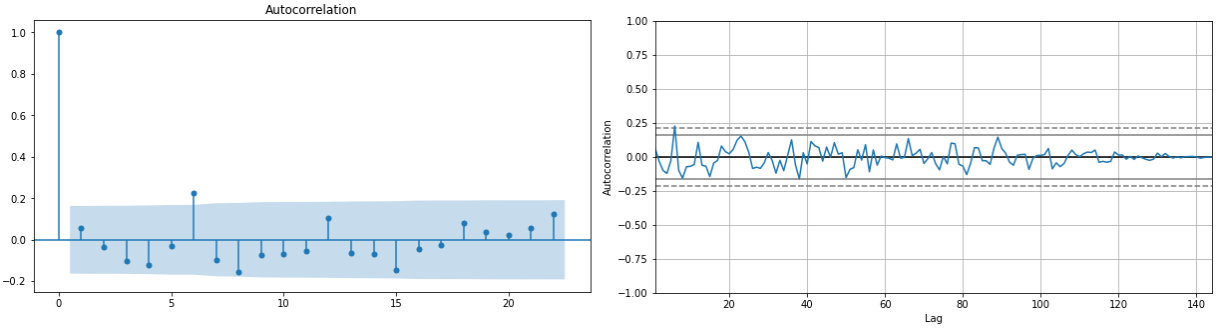

Figure 10: Autocorrelation function

Analysing the above graphs, we can select the Moving Average parameter $\mathbf{q = 3}$.

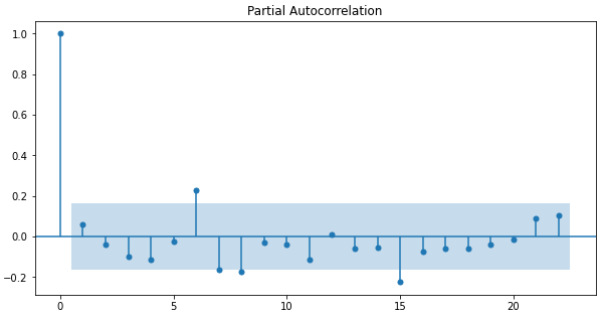

Figure 11: Partial Autocorrelation function

With the graph of the Partial Autocorrelation function we can see that the parameter corresponding to the Auto-regressive part is $\mathbf{p = 5}$.

3.3.3 SARIMA function

Once we have chosen the AR and MA parameters, we try different values for the Integrated(I) part. The values are usually $\mathbf{d = 0}$ or $\mathbf{d = 1}$.
After testing all possible values, the one that fits better is $\mathbf{d = 1}$.

The values related with the seasonal part have to be found after some iterations. On the other hand, the value of seasonal length in the data is known to be $\mathbf{s = 12}$ because it is annual (12 months).

After some iterations, the parameters with the best **AIC**[4] are the following:

$$SARIMA(5,1,3)(0,1,1)_{12} \tag{11}$$

Once the appropriate parameters have been found, we declare the function in the script through the *SARIMAX* library:

```
model = sm.tsa.statespace.SARIMAX(df, order=(5,1,3), seasonal_order
    =(0,1,1,12))
model.fit(max_iter = 1000,method = 'powell')
```

Listing 4: DataFrame vector

3.3.4 Model evaluation

Once we have the model defined, we evaluate the model on the dataset **df** we have.

```
forecast = model.predict(start=df.shape[0], end=df.shape[0] + 12)
```

Listing 5: SARIMA forecast

To this forecast, we add 12 more forecast periods (1 year) for future predictions.

[4]The **Akaike information criterion** is an estimator of prediction error and thereby relative quality of statistical models for a given set of data.

3.4 ANN models

Neural Network models can be classified as univariate and multivariate. In this project we will deal with a univariate dataset.
On the other hand, in order to test as many models as possible, we will adapt the multivariate models to be able to apply our dataset and draw more conclusions.

3.4.1 Data preparation

Consider a given univariate sequence:

[10, 20, 30, 40, 50, 60, 70, 80, 90]

We can divide the sequence into multiple input/output patterns called samples, where **n_steps** time steps are used as input and one time step is used as output for the one-step prediction.

X,	**y**
10, 20, 30	**40**
20, 30, 40	**50**
30, 40, 50	**60**
...	

The *split_sequence()* function below implements this behavior and will split a given univariate sequence into multiple samples where each sample has a specified number of time steps **n_steps** and the output is a single time step.

```
def split_sequence(sequence, n_steps):
  X, y = list(), list()
  for i in range(len(sequence)):
    # find the end of this pattern
    end_ix = i + n_steps
    # check if we are beyond the sequence
    if end_ix > len(sequence)-1:
      break
    # gather input and output parts of the pattern
    seq_x, seq_y = sequence[i:end_ix], sequence[end_ix]
    X.append(seq_x)
    y.append(seq_y)
  return array(X), array(y)
```

Listing 6: Univariate split sequence function

Using the vector **df** created earlier with the values from the dataset, we create an auxiliary vector to use as an input sequence in the function.

```
raw_seq = df.reset_index()
raw_seq = np.squeeze(np.asarray(raw_seq['rainfall']))
```

Listing 7: Univariate input sequence

As a result of this vector we obtain the following sequence:

[1.7075 1.4633 1.0659 1.5160 2.1695 2.092 1.3670 ... 3.7978]

In this project, using the *split_sequence()* function with our auxiliary sequence, we split the univariate series into samples that have **n_steps=12** input time steps and one output time step.

[1.7075 1.4633 1.0659 1.5160 2.1695 ... 3.0571] 1.595
[1.4633 1.0659 1.5160 2.1695 2.092 ... 1.595] 2.0175
...
[3.1485 0.9423 3.7135 1.6402 0.7623 0.5111] 3.7978

3.4.2 Keras library

Keras is a deep learning API written in Python, running on top of the machine learning platform *TensorFlow*[5].
TensorFlow 2 is an end-to-end, open-source machine learning platform. You can think of it as an infrastructure layer for differentiable programming. It combines four key abilities:

- Efficiently executing low-level tensor operations on CPU, GPU, or TPU.
- Computing the gradient of arbitrary differentiable expressions.
- Scaling computation to many devices
- Exporting programs ("graphs") to external runtimes such as servers, browsers, mobile and embedded devices.

This library is used to implement deep learning neural network models. All predictions and graphs are computed using this library.
As the specifications state, it can be optimised by using CPUs and GPUs.

[5]**TensorFlow** is a free and open-source software library for machine learning developed by *Google*

3.4.3 CNN model

A one-dimensional CNN is a CNN model that has a convolutional hidden layer that operates over a 1D sequence.
The convolutional and pooling layers are followed by a dense fully connected layer that interprets the features extracted by the convolutional part of the model.

We almost always have multiple samples, therefore, we have to reshape the input component of training data to fit with the requested from the model:

[samples, timesteps, features]

Now that we have the correct input shape, we can define the CNN model as follows:

```
model = Sequential()
model.add(Conv1D(filters=64, kernel_size=2, activation='relu',
    input_shape=(n_steps, n_features)))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(50, activation='relu'))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
```

Listing 8: CNN model

Key in the definition is the shape of the input; it is specified in the *input_shape* argument on the definition of the first hidden layer.
The model expects as input for each sample in terms of the number of time steps **n_steps=12** and the number of features **n_features=1**.

3.4.4 LSTM Vanilla model

A - LSTM Vanilla is an LSTM that has a single hidden layer of LSTM units, and an output layer used to make a prediction.

As with the previous model, we have to reshape the input data to fit the requirements of the model:

[samples, timesteps, features]

Now that we have the correct input shape, we can define the Vanilla LSTM model as follows:

```
model = Sequential()
model.add(LSTM(50, activation='relu', input_shape=(n_steps, n_features))
    )
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
```

Listing 9: LSTM Vanilla model

Key in the definition is the shape of the input; that is what the model expects as input for each sample in terms of the number of time steps and the number of features.
The model expects as input for each sample in terms of the number of time steps **n_steps=12** and the number of features **n_features=1**.

3.4.5 LSTM Stacked model

A - LSTM Stacked is an LSTM that has multiple hidden LSTM layers stacked one on top of another.

As with the previous model, we have to reshape the input data to fit the requirements of the model:

[samples, timesteps, features]

Now that we have the correct input shape, we can define the Stacked LSTM model as follows:

```
model = Sequential()
model.add(LSTM(50, activation='relu', return_sequences=True, input_shape
    =(n_steps, n_features)))
model.add(LSTM(50, activation='relu'))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
```

Listing 10: LSTM Stacked model

Key in the definition is the shape of the input; an LSTM layer requires a three-dimensional input and LSTMs by default will produce a two-dimensional output as an interpretation from the end of the sequence.
We can address this by having the LSTM output a value for each time step in the input data by setting the *return_sequences=True* argument on the layer. This allows us to have 3D output from hidden LSTM layer as input to the next.
The model expects, as in previous LSTM models, as input for each sample in terms of the number of time steps **n_steps=12** and the number of features **n_features=1**.

3.4.6 LSTM Bidirectional model

A - LSTM Bidirectional is an LSTM that allow to learn the input sequence both forward and backwards and concatenate both interpretations.

As with the previous model, we have to reshape the input data to fit the requirements of the model:

[samples, timesteps, features]

Now that we have the correct input shape, we can define the Bidirectional LSTM model as follows:

```
model = Sequential()
model.add(Bidirectional(LSTM(50, activation='relu'), input_shape=(
    n_steps, n_features)))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
```

Listing 11: LSTM Bidirectional model

Key in the definition is that we can implement a Bidirectional LSTM for univariate time series forecasting by wrapping the first hidden layer in a wrapper layer called Bidirectional. The model expects, as in previous LSTM models, as input for each sample in terms of the number of time steps **n_steps=12** and the number of features **n_features=1**.

3.4.7 LSTM CNN model

A - LSTM CNN is an CNN that can be used in a hybrid model with an LSTM backend where the CNN is used to interpret subsequences of input that together are provided as a sequence to an LSTM model to interpret.

Compared to the previous models, as this is a hybrid of CNN and LSTM, the input data must meet more requirements for the implementation of the model:

[samples, subsequences, timesteps, features]

Although we use the *split_sequence()* function as explained in Data preparation, when applying the model, we have to change some parameters to reshape the input to match the requirements of the model.
Now the number of time steps, unlike the previous models which was **n_steps=12**, will be **n_steps=2**.
On the other hand, we change the number of features to **n_features=1** and the number of sequences to **n_seq=6**.

Now that we have the correct input shape, we can define the CNN LSTM model as follows:

```
model = Sequential()
model.add(TimeDistributed(Conv1D(filters=64, kernel_size=1, activation='
    relu'), input_shape=(None, n_steps, n_features)))
model.add(TimeDistributed(MaxPooling1D(pool_size=2)))
model.add(TimeDistributed(Flatten()))
model.add(LSTM(50, activation='relu'))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
```

Listing 12: LSTM CNN model

Key in the definition is that we can implement a CNN model, so we need to split the input sequences into subsequences to be processed.
We first split our univariate time series data into input/output samples with twelve (**n_steps=12**) steps as input and one as output.
Each sample then is splited into six (**n_seq=6**) sub-samples, each with two time steps. The CNN can interpret each subsequence of two (**n_steps=2**) time steps and provide a time series of interpretations of the subsequences to the LSTM model to process as input.

3.4.8 ConvLSTM

A - ConvLSTM is a type of LSTM related to the - LSTM CNN, where the convolutional reading of input is built directly into each LSTM unit.
The ConvLSTM was developed for reading two-dimensional spatial-temporal data, but can be adapted for use with univariate time series forecasting.

Compared to the previous models, as the layer expects input as a sequence of two-dimensional images, the input data must meet more requirements for the implementation of the model:

[samples, timesteps, rows, columns, features]

As with the previous LSTM CNN model, we split each sample into subsequences where timesteps will become the number of subsequences (**n_seq=6**), and columns will be the number of time steps for each subsequence (**n_steps=2**). The number of rows is fixed at 1 as we are working with one-dimensional data.

Now that we have the correct input shape, we can define the ConvLSTM model as follows:

```
model = Sequential()
model.add(ConvLSTM2D(filters=64, kernel_size=(1,2), activation='relu',
    input_shape=(n_seq, 1, n_steps, n_features)))
model.add(Flatten())
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
```

Listing 13: ConvLSTM model

Key in the definition is that we can define the ConvLSTM as a single layer in terms of the number of filters and a two-dimensional kernel size in terms of (rows, columns). As we are working with a one-dimensional series, the number of rows is always fixed to 1 in the kernel.

3.4.9 Evaluation

Once we have the dataset and model defined, it is time to test the set.
The Python library includes a function that trains the model for a fixed number of epochs (iterations on a dataset).

```
model.fit(X, y, epochs=1000, verbose=0)
```

Listing 14: Model evaluation

For all ANN models, we use the same function with the same dataset and the same number of epochs.

```
yhat = []
for i in range(len(X[0])):
    yhat.append(X[0][i][0])

for i in range(len(X)):
    x_input = X[i]
    x_input = x_input.reshape((1, n_steps, n_features))

    yhat.append(model.predict(x_input, verbose=0)[0][0])

for i in range(12):
    x_input = array(yhat[len(yhat)-n_steps:])
    x_input = x_input.reshape((1, n_steps, n_features))

    yhat.append(model.predict(x_input, verbose=0)[0][0])
```

Listing 15: Forecasting

In order to create our **yhat** model vector, we divide the process into three parts; the first **n_step=12** month of data of **yhat** is the same as the dataset, the following data is created from the original data through the model and finally a 1 year (12 months) prediction is made respect to the last data from the model.

3.5 Prophet model

Prophet model is an open source time series forecasting algorithm designed by Facebook for ease of use without any expert knowledge in statistics or time series forecasting.
As this model also works in univariate forecasting, we only have to adapt the dataset in order to implement the it.

3.5.1 Data preparation

As we have done with ANN models, we take the vector **df** from the dataset and adapt it to our model.

```
df = df.reset_index()
df.columns = ['ds', 'y']
```

Listing 16: Prophet sequence

The Prophet model asks for a dataset with columns **ds** and **y**, where first column is the datetime and second column is the value of the sample.

3.5.2 Model

Once we have the input ready, we can define the parameters of the model to be applied.

```
my_model = Prophet(interval_width=0.3, yearly_seasonality=True,
    changepoint_prior_scale=2)
my_model.add_seasonality(name='monthly', period=30.5, fourier_order=5,
    prior_scale=0.02)
```

Listing 17: Prophet definition

The width of the uncertainty intervals can be set using the parameter *interval_width*
The model also allows us to define the seasonality, in our case we can specify that there is annual seasonality. In addition, with another function we add the monthly definition of the interval.
Finally, we can use *changepoint_prior_scale* to increase the uncertainty of the forecast.

3.5.3 Evaluation

Once the model has been defined and the dataset has been prepared for correct operation, we can evaluate its performance.

```
my_model.fit(df)

future_dates = my_model.make_future_dataframe(periods=12, freq='m')
forecast = my_model.predict(future_dates)
```

Listing 18: Prophet evaluation

First of all we evaluate the model defined with the dataset **df**.
Next, also with the model defined above, we make a forecast adding 1 year (12 months) of prediction.

3.6 Plotting

Once we have prepared the dataset and each model has applied a one year forecast, we show on the screen a graph of the comparison of the data collected by the pluviometer and the model applied.

```
pyplot.plot(fc['forecast'], color='r', label='model')
pyplot.plot(df['rainfall'], label='actual')
pyplot.legend()
pyplot.show()
```

Listing 19: Graphs plotting

In blue the dataset data is shown and in red the data of the applied model.
On the axes the value and the month in which the sample was taken is shown together with a legend of the graphs.

3.6.1 SARIMA plots

In addition to plotting the model, the *SARIMAX* library also allows to display of graphs that can help to analyse better the parameters that have been chosen appropriately.

```
model.plot_diagnostics(figsize=(15,12))
```

Listing 20: SARIMA residual plot

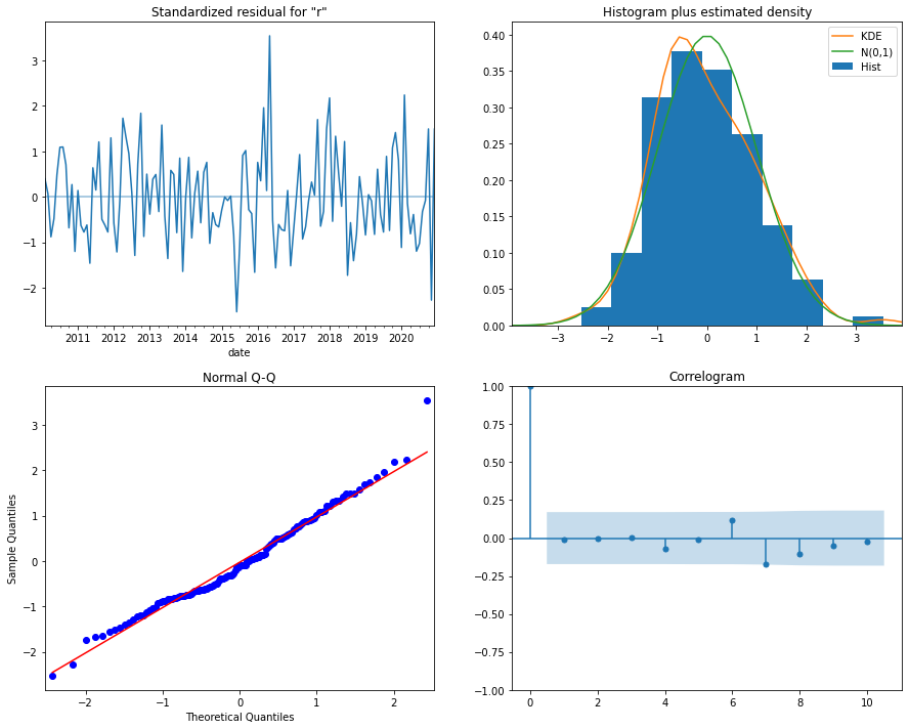

Figure 12: Residual diagnostics

In order to be able to interpret the residual value plots of the model, we can divide it into 4 parts:

- **Standardized residual:** The residual errors fluctuate around zero mean and with an uniform variance.
- **Histogram:** The density plot suggest normal distribution with zero mean.
- **Normal Q-Q:** All the dots fall perfectly in line with the red line.
- **Correlogram (ACF):** The plot shows the residual errors are not autocorrelated. Any autocorrelation would imply that there is some pattern in the residual errors which are not explained in the model.

3.6.2 Prophet plots

The *Facebook* library allows us to display on screen the decomposition of the dataset according to trend and seasonality.

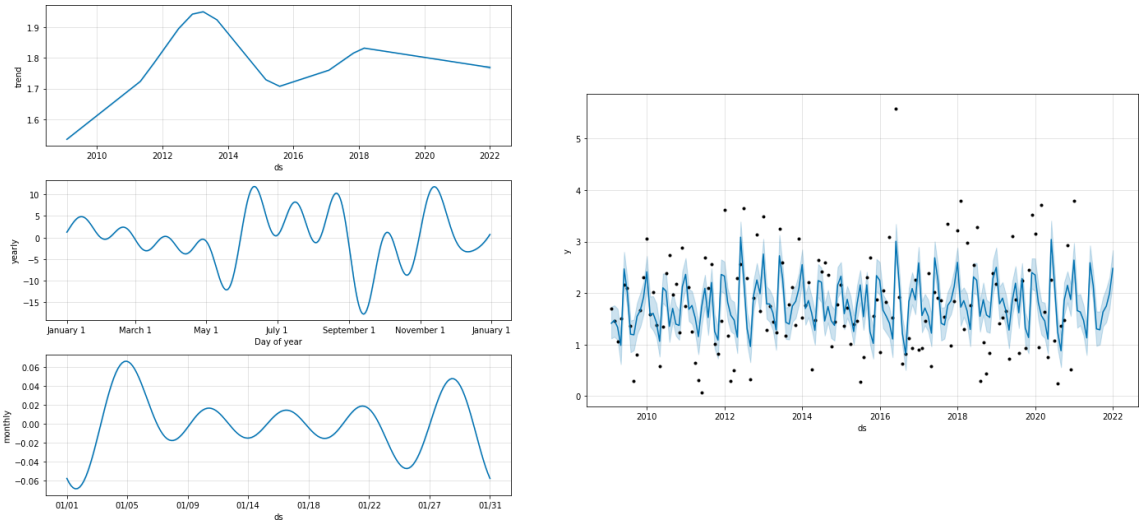

Figure 13: Prophet plots

With the first line of code we call the vector **forecast** created previously to plot it with its uncertainty threshold. On the other hand, the second line of code plots its components.

```
1 my_model.plot(forecast, uncertainty=True)
2 my_model.plot_components(forecast)
```

Listing 21: Prophet components plot

4 Experiments and Results

The aim of this section is to show the results obtained by applying the different set of models selected for this project to our dataset.
The results are shown in different ways, either in the form of graphs to visually show how the model behaves, or in the form of different accuracy measurement values mentioned in the section *Model performance parameters*

4.1 Models forecasting

This section shows the comparison of the original data with the chosen prediction model. For the ANN models, two predictions have been made with different number of epochs[6]; the first prediction with 100 and the second with 1000.
All tests include a 12-month forecast from the end of the dataset. As the dataset data ends in 2020, the forecast is for the whole of 2021.

4.1.1 SARIMA model

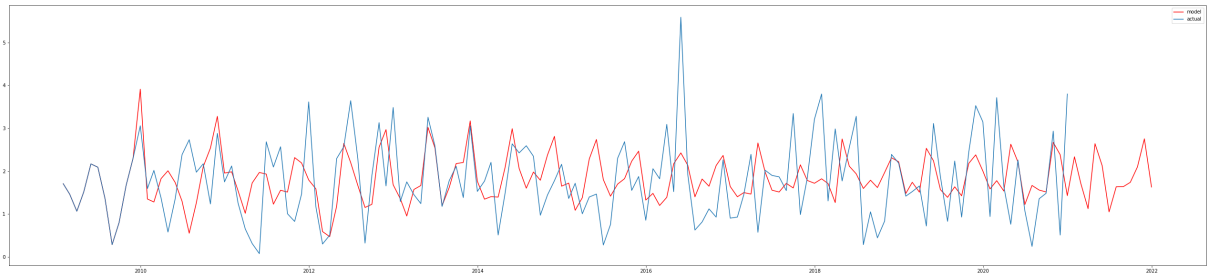

Figure 14: SARIMA test

Performance parameters

- **MFE:** -0,034
- **MAE:** 0,721
- **MSE:** 0,930
- **RMSE:** 0,965
- **Consumed time:** 7,962s

[6]The number of **epochs** is a hyperparameter that defines the number times that the learning algorithm will work through the entire training dataset.

4.1.2 CNN model

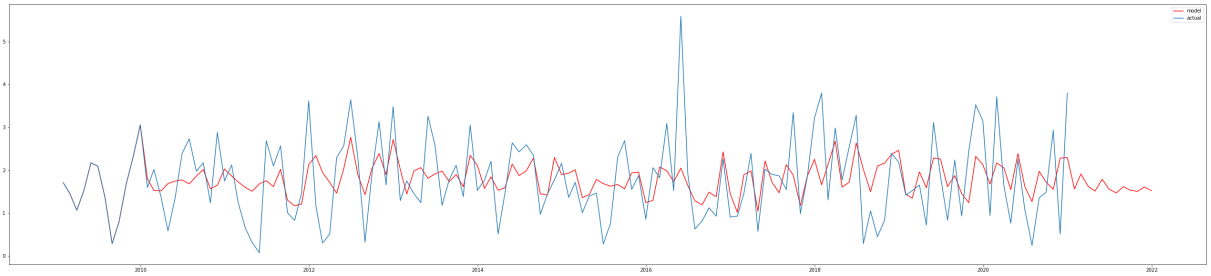

Figure 15: CNN test (100 epochs)

Performance parameters 100 epochs

- **MFE:** -0,083
- **MAE:** 0,646
- **MSE:** 0,710
- **RMSE:** 0,843
- **Consumed time:** 5,759s

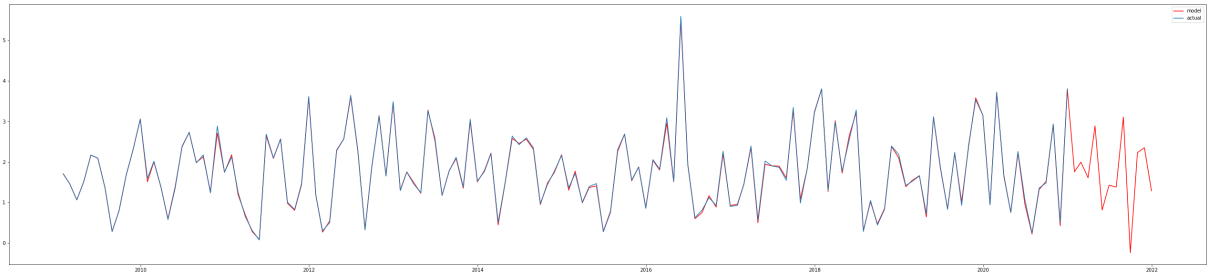

Figure 16: CNN test (1000 epochs)

Performance parameters 1000 epochs

- **MFE:** 0,008
- **MAE:** 0,246
- **MSE:** 0,128
- **RMSE:** 0,357
- **Consumed time:** 12,246s

4.1.3 LSTM Vanilla model

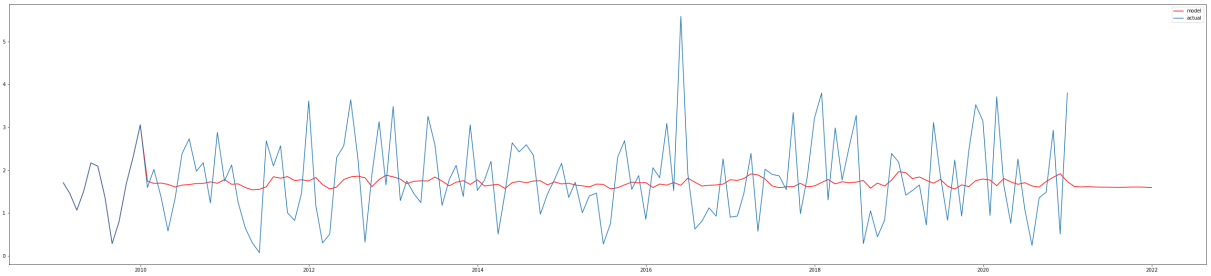

Figure 17: LSTM Vanilla test (100 epochs)

Performance parameters 100 epochs

- **MFE:** 0,111
- **MAE:** 0,696
- **MSE:** 0,845
- **RMSE:** 0,920
- **Consumed time:** 9,497s

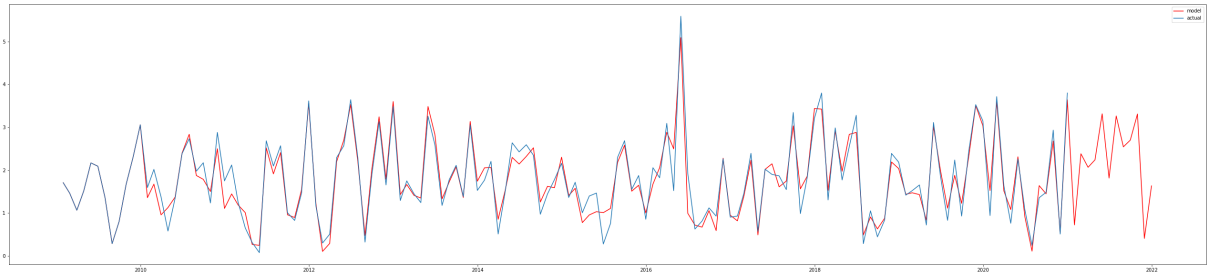

Figure 18: LSTM Vanilla test (1000 epochs)

Performance parameters 1000 epochs

- **MFE:** 0,017
- **MAE:** 0,187
- **MSE:** 0,065
- **RMSE:** 0,256
- **Consumed time:** 31,729s

4.1.4 LSTM Stacked model

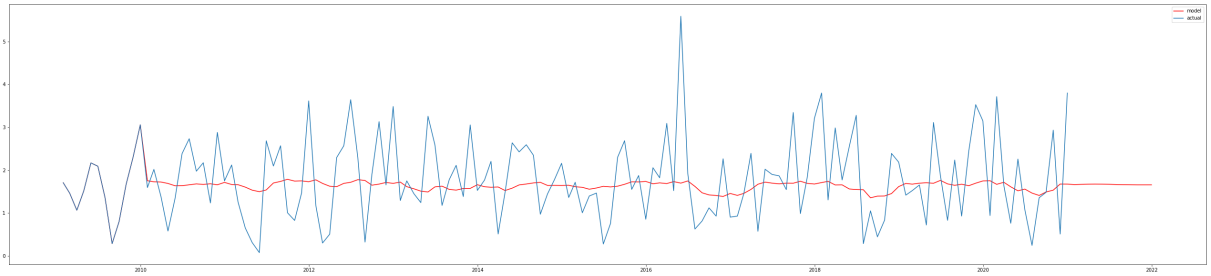

Figure 19: LSTM Stacked test (100 epochs)

Performance parameters 100 epochs

- **MFE:** 0,155
- **MAE:** 0,681
- **MSE:** 0,835
- **RMSE:** 0,914
- **Consumed time:** 13,644s

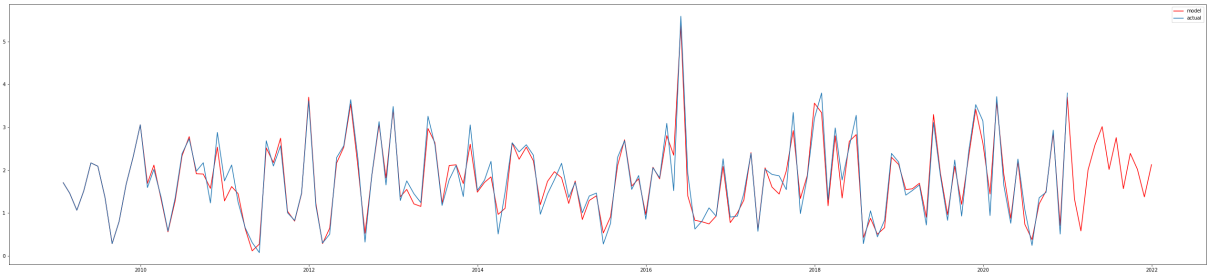

Figure 20: LSTM Stacked test (1000 epochs)

Performance parameters 1000 epochs

- **MFE:** 0,026
- **MAE:** 0,158
- **MSE:** 0,047
- **RMSE:** 0,218
- **Consumed time:** 52,857s

4.1.5 LSTM Bidirectional model

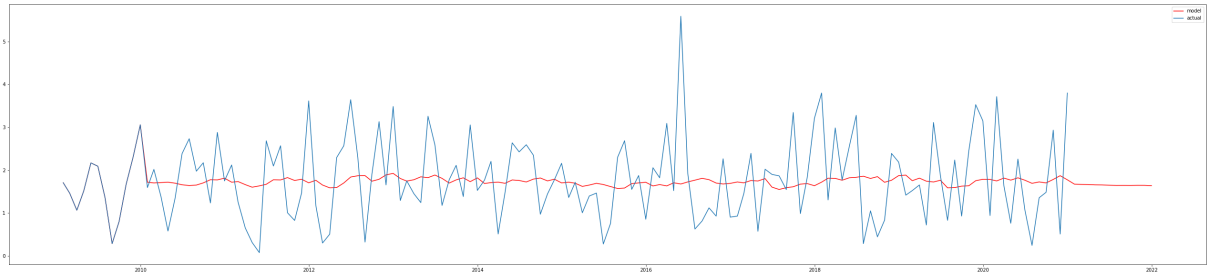

Figure 21: LSTM Bidirectional test (100 epochs)

Performance parameters 100 epochs

- **MFE:** 0,063
- **MAE:** 0,700
- **MSE:** 0,839
- **RMSE:** 0,916
- **Consumed time:** 9,794s

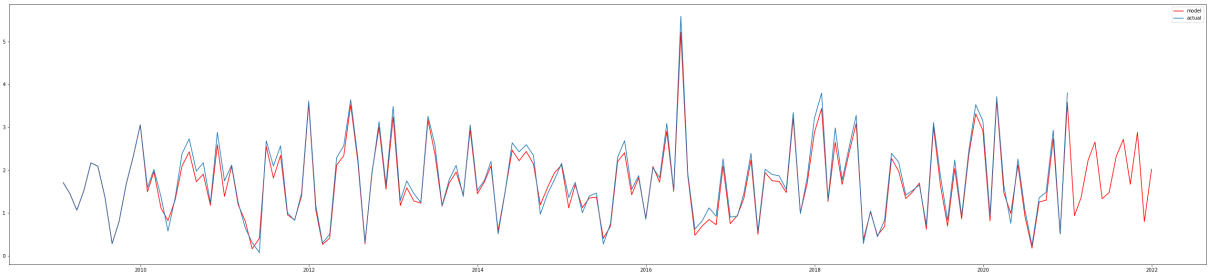

Figure 22: LSTM Bidirectional test (1000 epochs)

Performance parameters 1000 epochs

- **MFE:** 0,093
- **MAE:** 0,123
- **MSE:** 0,024
- **RMSE:** 0,153
- **Consumed time:** 36,056s

4.1.6 LSTM CNN model

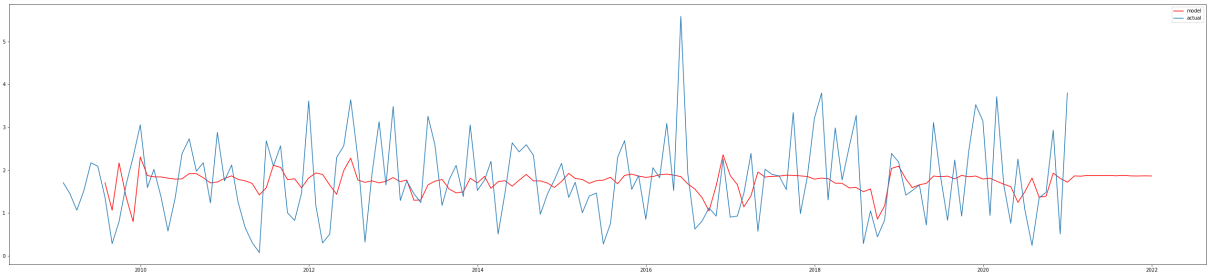

Figure 23: LSTM CNN test (100 epochs)

Performance parameters 100 epochs

- **MFE:** -0,013
- **MAE:** 0,594
- **MSE:** 0,633
- **RMSE:** 0,796
- **Consumed time:** 8,171s

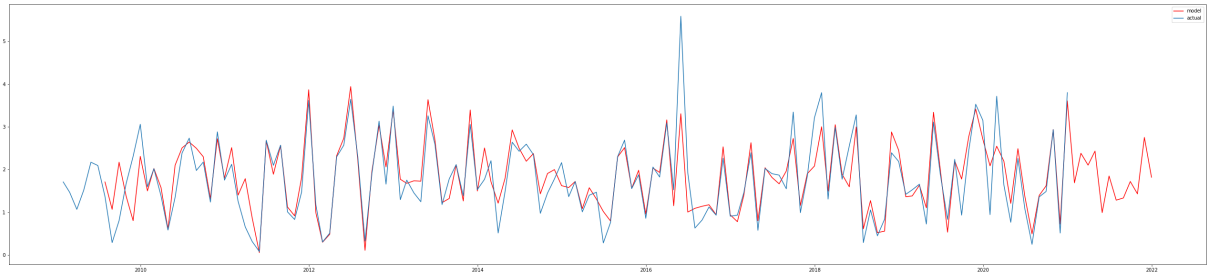

Figure 24: LSTM CNN test (1000 epochs)

Performance parameters 1000 epochs

- **MFE:** 0,010
- **MAE:** 0,030
- **MSE:** 0,002
- **RMSE:** 0,041
- **Consumed time:** 12,385s

4.1.7 ConvLTSM model

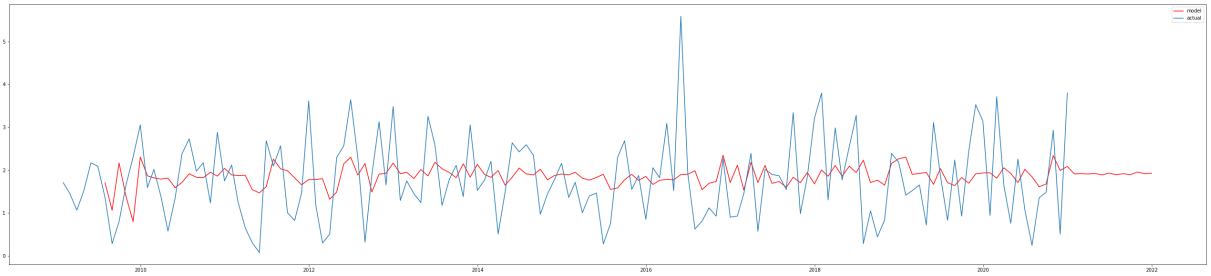

Figure 25: ConvLTSM test (100 epochs)

Performance parameters 100 epochs

- **MFE:** -0,072
- **MAE:** 0,737
- **MSE:** 0,860
- **RMSE:** 0,927
- **Consumed time:** 10,532s

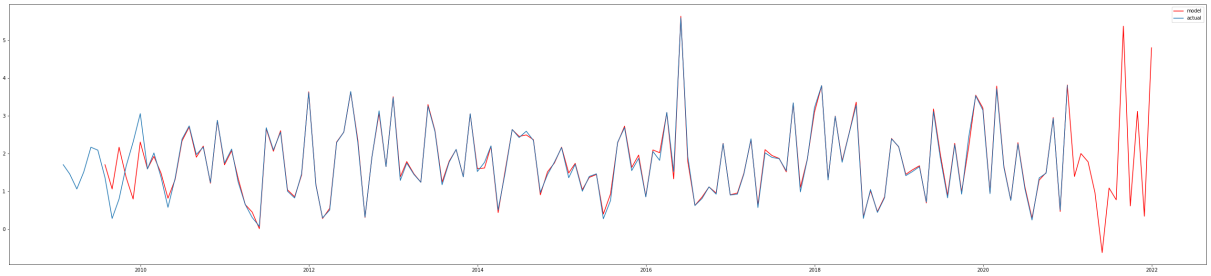

Figure 26: ConvLTSM test (1000 epochs)

Performance parameters 1000 epochs

- **MFE:** 0,010
- **MAE:** 0,906
- **MSE:** 1,374
- **RMSE:** 1,172
- **Consumed time:** 43,912s

4.1.8 Prophet model

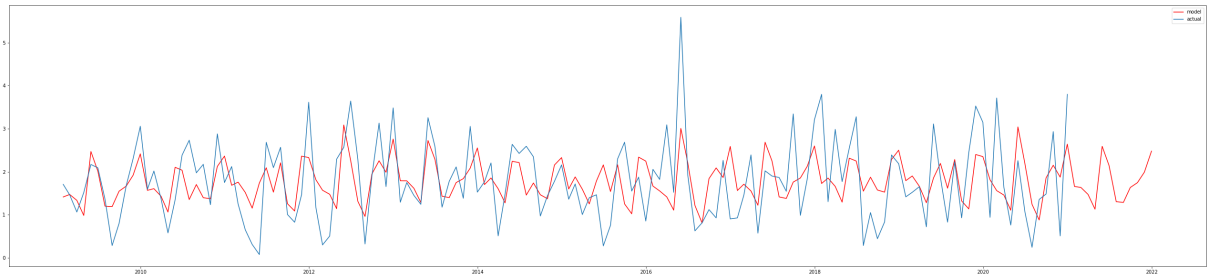

Figure 27: Prophet test

Performance parameters

- **MFE:** 0,000
- **MAE:** 0,640
- **MSE:** 0,644
- **RMSE:** 0,803
- **Consumed time:** 1,992s

4.2 Perform parameters comparison

Below is a summary table of the forecasting performance parameters mentioned previously grouping the results of all models used in the project.

Model	runtime	MFE	MAE	MSE	RMSE
SARIMA	7,962s	-0,034	0,721	0,930	0,965
CNN	5,759s	-0,083	0,646	0,710	0,843
LTSM Vanilla	9,497s	0,111	0,696	0,845	0,920
LTSM Stacked	13,644s	0,155	0,681	0,835	0,914
LSTM Bidirectional	9,794s	0,063	0,700	0,839	0,916
LSTM CNN	8,171s	-0,013	0,594	0,633	0,796
ConvLTSM	10,532s	-0,072	0,737	0,860	0,927
Prophet	1,992s	0,000	0,640	0,644	0,803

Table 1: Summary table of forecasting perform parameters with 100 epochs

Model	runtime	MFE	MAE	MSE	RMSE
SARIMA	7,962s	-0,034	0,721	0,930	0,965
CNN	12,246s	0,008	0,246	0,128	0,357
LTSM Vanilla	31,729s	0,017	0,187	0,065	0,256
LTSM Stacked	52,857s	0,026	0,158	0,047	0,218
LSTM Bidirectional	36,056s	0,093	0,123	0,024	0,153
LSTM CNN	12,385s	0,010	0,030	0,002	0,041
ConvLTSM	43,912s	0,010	0,906	1,374	1,172
Prophet	1,992s	0,000	0,640	0,644	0,803

Table 2: Summary table of forecasting perform parameters with 1000 epochs

4.3 Analysis of the results

4.3.1 Mean Forecast Error

The Mean Forecast Error can be analysed according to its bias.
A value other than zero suggests a tendency of the model to **over forecast (negative error)** or **under forecast (positive error)**.

Starting to analyse with the models that are not based on the use of ANN, we can observe that SARIMA is slightly over forecast. On the other hand, we can highlight that the Prophet model is the one that is closest to zero.
In the case of ANN models, all models except the LSTM Bidirectional model, improve their bias as epochs increase tending more to zero from positive error.

We can summarize this parameter by saying that the Prophet model and the CNN-based models are the ones with the best Mean Forecast Error.

4.3.2 Mean Absolute Error

The Mean Absolute Error is calculated as the average of the forecast error values, where all the forecast error values are forced to be positive.
These error values are in the original units of the predicted values. A **mean absolute error of zero indicates no error**.

On this parameter we can see that the SARIMA and Prophet models differ considerably from the zero value.
On the other hand, it is observed that the ANN models are closer to zero the more epochs we use, except for ConvLSTM, whose error increases.

In conclusion, we can say that the LSTM CNN model is the one with the best Mean Absolute Error.

4.3.3 Mean Square Error

The Mean Square Error is calculated as the average of the squared forecast error values. Squaring the forecast error values forces them to be positive; it also has the effect of putting more weight on large errors.
The error values are in squared units of the predicted values. A **mean squared error of zero indicates perfect skill**, or no error.

As with Mean Absolute Error, the SARIMA and Prophet models present a difference of zero which, in this parameter, is even more considerable.
The same also applies to the ANN models, where we can note that ConvLSTM has considerably increased its error and, in contrast, the value of LSTM CNN is practically zero.

In conclusion, we can still say that the LSTM CNN model is the one with best error performance, now the one with the best Mean Square Error.

4.3.4 Root-Mean-Square Error

The Root-mean-square error is the the square root of the mean square error described above.
The RMSE values are in the same units as the predictions. As with the mean squared error, an **RMSE of zero indicates no error**.

Following the same behaviour of the previous parameters, the SARIMA and Prophet models follow the same upward error trajectory. On the behaviour of the ANN models, it is observed that all models follow the same behaviour as with the previous parameters. The high value of the ConvLSTM model and the tending to zero value of the LSTM CNN model stand out.

As with the previous parameters, we come to the same conclusion that the LSTM CNN model is the one with the lowest error compared to the other models.

4.3.5 Runtime

The Runtime is the time it takes for the script to run. You can easily see that the lower the value, the faster it will be to implement the model.

We can see that the fastest model to apply is the Prophet model followed for SARIMA model.
On the other hand, we only consider ANN models with 1000 epochs test because the other tests do not have sufficient accuracy.
Therefore, the CNN and LSTM CNN models are the fastest of this set.

5 Economical analysis

The financial analysis of this project is detailed as follows:

- **Computer:** Apple MacBook Pro 13-Inch "Core i7" 3.1 Early 2015
- **Software:** Anaconda Navigator
  - Spyder
  - Jupyter Notebook
  - Python
- **Cloud services:** GitHub
- **Salary:** Telecommunications student salary
  - 9€per hour
  - Final project of 20 ECTS
  - 1 ECTS = 25 hours
- **Office:** Coworking Office Spaces
  - 200€per month
  - Project duration of 4 months

Concept	**Amount**
Computer	2.124,09 €
Software	Free
Cloud services	Free
Salary	4.500€
Office	800€
Total:	7.424.09€

Table 3: Economic budget

The final budget for the project is **7.424.09€**

6 Conclusions and Future Work

The goal of this section is to make a comparison of the different models through the results mentioned previously, as well as to mention the weaknesses of the project and also to define possible improvements for future developments.

6.1 Conclusions

The aim of this project is to compare a classical model such as SARIMA with a SANN model. In the search for models for univariate forecasting we have come across the existence of another recent type of forecasting model created by Facebook called Prophet. Therefore, the final comparison will be between the SARIMA model, the Prophet model and the best ANN based model for our dataset.

With the help of the experiments carried out in the section on Experiments and Results we can see that the best ANN-based model is the CNN LSTM model.
We can observe that CNN models need much less runtime to be implemented than the others ANN models. Also this is the one with the lowest error rate when applying the model and, therefore, its forecasting will be more accurate.

We will now come to the final conclusion of this project by comparing the SARIMA, Prophet and LSTM CNN models.
The model with the shortest execution time is the Prophet model with approximately 2s, in contrast to SARIMA which needs 8s and LSTM CNN with 12s.
Looking at MFE, we can see that Prophet and LSTM CNN are the models that come closest to being ideal because of their closeness to zero. In contrast, SARIMA presents a slightly over forecast.
Talking about MAE, MSE and RMSE we can observe that the model with more proximity to zero than the other models is the LSTM CNN model.

We can affirm that despite the fact that LSTM CNN model has longer execution time, it that can offer us the best prediction.
Another good option would be Prophet, as its runtime is by far the longest of the other models and its error ratio is still considerably favourable.

In conclusion, the SARIMA model is currently not a good option to be applied for forecasting. Its error ratio is slightly worse than the others, but if on top of that its execution time is higher and in order to apply it you need an exhaustive analysis of the dataset to find the SARIMA parameters, this makes it a better option today to implement any of the other two models.

6.2 Future Work

Although a wide variety of forecasting models have been implemented in this project, many of them are multivariate models. The fact that we have adapted our dataset to be able to implement them has turned its implementation far from optimal.
This can be solved if in our dataset we add data related to rainfall like temperature, humidity, wind speed, air pollution, etc.

Another aspect to improve could be the optimization of the model implementation. Now the models are implemented using a monthly average of the dataset, but a daily average could be used to have higher accuracy.
This is an impediment due to the hardware limitations of the test environment.
ANN-based models are implemented using the Keras library which is more optimal with the use of GPU for computing.

Keras optimization is available if our work environment has compatible CPU and GPU. The best way to improve the performance of a neural network is through CUDA Cores[7] and Windows operating system.
On the other hand, if our work environment does not have this requirements, it can also be optimised by configuring it according to the hardware we have available.

Finally, one of the improvements that could be applied to the models is the use of k-Fold Cross-Validation that will help us to measure the behaviour of the models to find a better model quickly.

k-Fold Cross-Validation use the original sample randomly partitioned into k equal sized subsamples. Of the k subsamples, a single subsample is retained as the validation data for testing the model, and the remaining *k-1* subsamples are used as training data. The cross-validation process is then repeated k times, with each of the k subsamples used exactly once as the validation data. The k results can then be averaged to produce a single estimation.

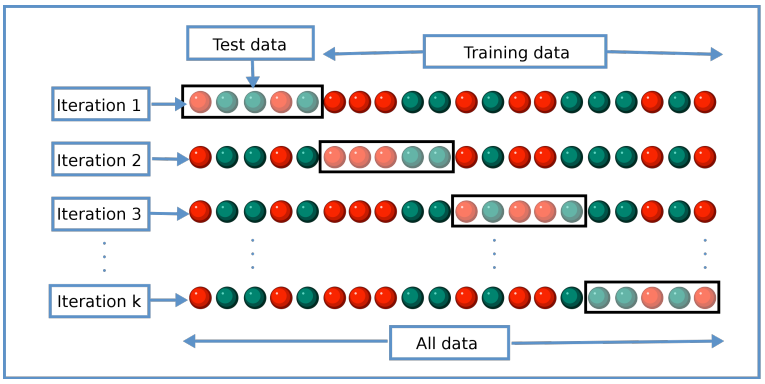

Figure 28: k-Fold Cross-Validation

[7]**CUDA Cores** is a parallel computing platform and API model created by *nVIDIA*

References

[1] How to Develop Convolutional Neural Network Models for Time Series Forecasting
https://machinelearningmastery.com/how-to-develop-convolutional-neural-network-models-for-time-series-forecasting

[2] How to Develop LSTM Models for Time Series Forecasting
https://machinelearningmastery.com/how-to-develop-lstm-models-for-time-series-forecasting

[3] Time Series Forecasting Performance Measures With Python
https://machinelearningmastery.com/time-series-forecasting-performance-measures-with-python

[4] A Gentle Introduction to k-fold Cross-Validation
https://machinelearningmastery.com/k-fold-cross-validation

[5] How to Check if Time Series Data is Stationary with Python
https://machinelearningmastery.com/time-series-data-stationary-python

[6] Facebook Prophet For Time Series Forecasting in Python
https://towardsdatascience.com/facebook-prophet-for-time-series-forecasting-in-python-part1-d9739cc79b1d

[7] Time Series Forecasting with SARIMA in Python
https://towardsdatascience.com/time-series-forecasting-with-sarima-in-python-cda5b793977b

[8] ARIMA Model Python Example — Time Series Forecasting
https://towardsdatascience.com/machine-learning-part-19-time-series-and-autoregressive-integrated-moving-average-model-arima-c1005347b0d7

[9] Time Series Models
https://towardsdatascience.com/time-series-models-d9266f8ac7b0

[10] Comparing SANN and SARIMA for forecasting frequency of monthly rainfall in Umuahia
https://www.sciencedirect.com/science/article/pii/S2468227620303586

[11] Pluviométrie
https://www.data.gouv.fr/en/datasets/pluviometrie

[12] Prophet: forecasting at scale
https://research.fb.com/blog/2017/02/prophet-forecasting-at-scale

[13] Internet enabled tipping bucket rain gauge
https://ieeexplore.ieee.org/document/6921828

[14] XLA: Optimizing Compiler for Machine Learning
https://www.tensorflow.org/xla

Appendices

A Installation

The following will detail how to install the programs and libraries necessary to test the project.

A.1 Work environment

In order to develop the whole project, we have used the program *Anaconda*.
Anaconda is a distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment.

Using the following link will direct us to its main web page where we can download the program.
This program contains a multitude of programs and libraries. We are mainly interested in it because it contains *Python, Spyder* and *Jupyter Notebook*.

Once the program is installed, using the **Anaconda Prompt** application, we can update its components with the command *conda update anaconda*

A.2 Python libraries

In order to be able to run all models correctly, we will need to install a number of external libraries. To do this we will use the **Anaconda Prompt** tool mentioned above.

The following libraries are required:

- **pmdarima**: ARIMA estimators
- **tensorflow**: TensorFlow for machine learning
- **fbprophet** : Facebook automatic forecasting procedure

To install the libraries, use the following command in the *Terminal* to install all the dependencies.

```
pip install -r requirements.txt
```

Listing 22: Requirements installation

In order to install it manually, use the command *pip install* and add the name of each library.

A.3 CUDA Cores

If our environment has a compatible NVIDIA GPU, we will be able to optimise our variable computation through the CURA Cores of this GPU. To do this we will go to the following link where we will get more information on compatibility.

In order to use this optimisation we will need to have previously installed the following software on our PC:

- **NVIDIA® GPU drivers**
- **CUPTI** ships with the CUDA® Toolkit.
- **cuDNN SDK**
- **cuDNN SDK**
- (Optional) **TensorRT** to improve latency and throughput for inference on some models.

On the other hand also have the latest version of TensorFlow installed.

A.4 Tensorflow XLA

The computer with which the project tests have been carried out does not have a graphics card compatible with the optimization for Tensorflow.
However, with the following code, we can activate the **Tensorflow XLA**[8] accelerator:

```
import os

os.environ['TF_XLA_FLAGS'] = '--tf_xla_enable_xla_devices'
```

Listing 23: Optimizing Compiler for Machine Learning

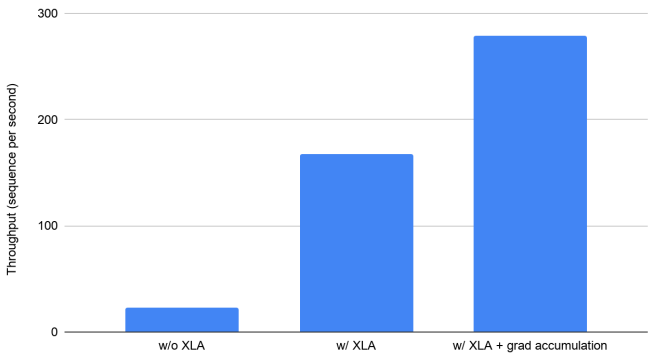

Figure 29: XLA comparative

[8] **Accelerated Linear Algebra** is a domain-specific compiler for linear algebra that can accelerate TensorFlow models with potentially no source code changes.

Abbreviations

AI Artificial Intelligence

AIC Akaike information criterion

ANN Artificial Neural Network

API Application Programming Interface

CNN Convolutional Neural Network

CPU Central Processing Unit

CSV Comma-Separated Values

CTU Czech Technical University (CVUT)

ETSETB Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona

GPU Graphics Processing Unit

IA Intel·ligència artificial

ICT Information and communications technology

IoT Internet of Things

JSON JavaScript Object Notation

LSTM Long Short-Term Memory

MAE Mean Absolute Error

MFE Mean Forecast Error

ML Machine Learning

MSE Mean Squared Error

RMSE Root-Mean-Square Error

RNN Recurrent Neural Network

SANN Seasonal Artificial Neural Network

SARIMA Seasonal Autoregressive Integrated Moving Average

TIC Tecnologies de la informació i la comunicació

UPC Universitat Politècnica de Catalunya