POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

A hardware and software system for the evaluation of time series prediction models

Tesi di Laurea Magistrale in
Computer Science and Engineering - Ingegneria Informatica

Author: **Giorgio Piazza**

Student ID: 969354
Advisor: Prof. Piero Fraternali
Co-advisors: Nicolò Oreste Pinciroli Vago
Academic Year: 2022-23

Abstract

The way modern businesses operate and innovate has been revolutionized by the proliferation of the Industrial Internet of Things (IIoT). Huge masses of data are being produced by connected devices and their analysis has caused a shift to occur in the way decisions are made in business.

The intersection between the spread of connected devices and this paradigm shift is what this thesis project in collaboration with Servitly was based on. This work proposes a prototype connected machine that simulates a controlled mechanical ventilation system composed of sensors that provide data on the machine's state of operation. The collected data are used to make predictions about the state of the machine using different types of predictive models.

Predictions of the collected time series were made using an ARIMA model, an XGBoost regressor and a neural network consisting of LSTM nodes. The mean absolute error of the ARIMA model is between 3.43% and 3.46%, that of XGBoost between 2.00% and 2.67%, and that of the LSTM network between 1.89% and 2.56%.

Keywords: iiot, time series, prediction models

Abstract in lingua italiana

Il modo in cui le aziende moderne operano e si innovano è stato rivoluzionato grazie alla proliferazione dell'Industrial Internet of Things (IIoT). Moli enormi di dati vengono prodotte dai dispositivi connessi e la loro analisi ha fatto sì che avvenisse un cambiamento nel modo in cui vengono prese le decisioni in ambito aziendale.

L'intersezione tra la diffusione dei dispositivi connessi e questo cambio di paradigma è ciò su cui si è basato questo progetto di tesi in collaborazione con Servitly. Questo lavoro propone un prototipo di macchina connessa che simula un sistema di ventilazione meccanica controllata composta da sensori che forniscono dati sullo stato di operazione della macchina. I dati raccolti vengono utilizzati per effettuare previsioni sullo stato della macchina utilizzando diversi tipi di modelli predittivi.

Le predizioni della serie temporale raccolta sono state effettuate utilizzando un modello ARIMA, un regressore XGBoost e una rete neurale composta da nodi LSTM. L'errore medio assoluto del modello ARIMA è tra 3.43% e 3.46%, quello di XGBoost tra 2.00% e 2.67% e quello della rete LSTM tra 1.89% e 2.56%.

Parole chiave: iiot, serie temporali, modelli di predizione

Contents

1 | Introduction

Over the past few decades, the introduction of IoT devices in industry has led to a revolution. The Industrial Internet of Things (IIoT) has transformed traditional manufacturing processes into an ecosystem of interconnected smart devices, sensors and machines capable of capturing real-time information about production activities. This has led to the generation of unprecedented amounts of data, changing the way industries operate, optimize and innovate and triggering a paradigm shift in decision-making. In fact, more and more businesses have adopted predictive models based on data, enabling industries to predict future trends, identify potential problems and optimize operational efficiency.

Servitly [6], a software company that provides a cloud-based platform for the digital servitization of industrial assets, is based on this new approach. Their product offers a way to process and analyze the great amounts of data received from connected products in order to detect events such as failures, identify patterns of good or bad product usage and generate valuable insights such as KPIs, trends, and benchmarks. In addition, Servitly helps manufacturers monetize their IoT data by providing tools for selling data insights, subscriptions, and pay-per-use services.

SERVITLY

This thesis project, in collaboration with Servitly, aims to design and implement a machine that simulates a simple controlled mechanical ventilation system, using sensors to generate measurements from which information is extracted. Among all the data from the machine, the data from the air velocity meter, a sensor mounted on the machine, is used to predict its behavior using different types of predictive models.

The time series is predicted using three models: a traditional model (ARIMA), a machine learning model (XGBoost), and a deep learning model (LSTM network). The prediction is also made after subtracting from the data a periodic signal obtained by FFT analysis of the signal. The forecasts with ARIMA have a mean absolute percentage error between 3.43% and 3.46%, those with XGBoost between 2.00% and 2.67% and those of the LSTM network between 1.89% and 2.56%.

This document is structured as follows:

- **Chapter 2** explains in detail the software and hardware architecture of the machine.
- **Chapter 3** dives deep into the data analysis, presenting the data set, the experiments done and the results.
- **Chapter 4** draws the conclusions and lists possible future works.

2 | Architecture

This thesis project aims to design and implement a machine that simulates a simple controlled mechanical ventilation system and to predict machine measurements.

In this chapter the architecture of the built machine will be discussed in detail.

The project encompasses two essential components:

- **Hardware part**: it consists of a main board with various electronic components attached to it.
- **Software part**: it consists of all the code that controls the behavior of the machine, the reading of measurements from the sensors and the communication with Servitly's server.

By integrating the hardware and software components, this project aims to simulate a real industrial machinery that communicates with the entire Servitly ecosystem, sending data and receiving commands and parameters.

2.1. Hardware Architecture

The machine is composed by several electronic components connected to a board that does the computation. The Raspberry Pi Zero W [4] is the central board of the project. It is a single-board computer capable of running a Linux operating system, and because it is compact and affordable, it is widely used in IoT applications. The main specifications of the Raspberry Pi Zero W include:

- Processor: Broadcom BCM2835 1GHz ARM single-core CPU
- Memory: 512MB RAM
- Connectivity: 802.11 b/g/n wireless, Bluetooth 4.1 and Bluetooth Low Energy
- GPIO Pins: 40 software-controllable GPIO for interfacing with external devices
- Ports: Micro USB (power), Mini HDMI (video output), Micro SD (storage)

Raspberry Pi Zero W provides all the computing power and connectivity options to support the functionality of the project: the wireless module provides the internet connection, which is essential to communicate with the Servitly's server, while the GPIO pins provide a convenient interface for any connected components.

For our machine, the Raspberry Pi OS is installed on the Raspberry Pi Zero W. It is an operating system based on the Debian Linux distribution, adapted for use on Raspberry Pi. The version installed is the Lite version, the one without the desktop environment.

Figure 2.1: Raspberry Pi Zero W. [4]

The ventilation machine has to be composed of several sensors and actuators. For this reason, it was decided to use a prototyping system that would provide everything needed. The solution chosen is the Seeed Studio's Grove Ecosystem [5], which consists of a board called Grove Base Hat, mounted on the Raspberry Pi Zero W, and a set of modules. The Grove Base Hat is an expansion board that provides 15 multi-function Grove ports that can be used to install Grove Ecosystem modules. It also provides an 8-channel 12-bit ADC using an integrated STM32 MCU and the SWD debug interface. Since all the pins of Raspberry Pi are digital, the presence of an ADC is essential to sample analog signals.

The 15 ports of the shield are divided as follows:

- 6 x Digital ports: each port is connected to two GPIO pins
- 4 x Analog ports: each port is connected to the STM32 that samples the analog signals. The MCU converts the analog data to digital data and inputs the digital data to the Raspberry Pi through the I2C interface
- 1 x PWM port: it connects to pin 12 (PWM0) and pin 13 (PWM1), which are the hardware PWM pins of Raspberry Pi
- 3 x I2C ports: they all connect to pin 3 (SDA) and pin 5 (SCL)
- 1 x UART port: it connects to the pin 8 (UART0 TX) and pin 10 (UART0 RX)

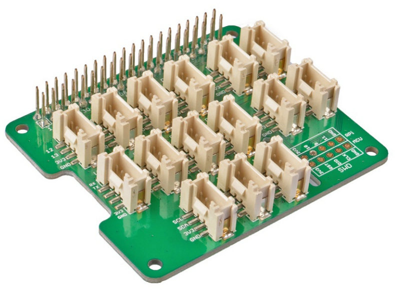
PWM (Pulse Width Modulation) is a technique used to control the power delivered to a load. It involves rapidly switching a signal between ON and OFF states with varying duty cycles. The duty cycle is the ratio of the ON time to the total time of the signal. By adjusting the duty cycle, the average power delivered to the load can be controlled.

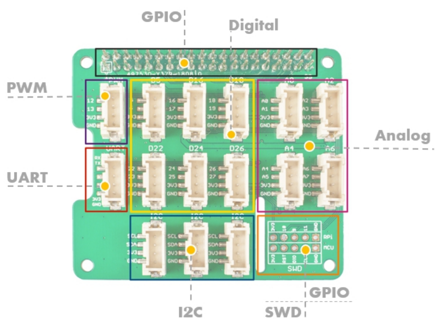
I2C (Inter-Integrated Circuit) is a synchronous serial communication protocol used to interconnect integrated circuits in electronic devices. It consists of a master-slave configuration in which a master device controls one or more slave devices. The protocol uses two lines to communicate:

- SDA (Serial Data Line): line used for data transfer
- SCL (Serial Clock Line): line used for communication synchronization

Each slave device has a unique address that allows the master to send communication to the desired device. The protocol provides a simple and efficient means of communication with minimal wiring and hardware requirements.

UART (Universal Asynchronous Receiver-Transmitter) is another communication protocol for transferring serial data between devices.
Unlike I2C, UART operates in full-duplex mode, allowing simultaneous transmission and reception of data (TX and RX lines). It is asynchronous and therefore does not rely on a shared clock signal: each device has its own clock.


(a) Grove Base Hat.



(b) Grove Base Hat Pinout.

Figure 2.2: Grove Base Hat overview.[5]

Several Grove modules were attached to the shield to make up the ventilation machine:

- Dual Button & Red LED Button modules: provides buttons that are used in the ventilation machine to turn on/off the fan and to simulate a cleaning cycle or an error.
  They are attached to some digital ports of the hat.
- Encoder module: provides an encoder that is used to adjust the rotation speed of the fan.
  It is attached to a digital port of the hat.
- RGB LED module: provides a RGB LED used to signal the measured air speed.
  It is attached to a digital port of the hat.
- Ultrasonic Ranger module: provides an ultrasonic proximity sensor used to detect the presence of an operator in front of the ventilation machine.
  It is attached to a digital port of the hat.
- OLED Display module: provides an OLED display used to show information about the status of the machine.
  It is attached to a I2C port of the hat.
- 3-Axis Digital Accelerometer module: provides a 3-Axis accelerometer used to detect a bump or a tilt of the machine.
  It is attached to a I2C port of the hat.
- Motor Driver module: provides a motor driver used to control the speed of the fan.
  It is attached to a I2C port of the hat.

In addition to all these modules, a DC motor is connected to one of the analog ports. This motor serves as an air speed meter: wind blades are attached to the rotor. Thus the rotor rotates by the air generated by the fan and generates a DC voltage between the ends of the motor winding. The generated voltage is directly proportional to the rotational speed and thus provides an air speed measurement. In the data analysis phase, a time series prediction will be performed based on samples of the measurements from this sensor.

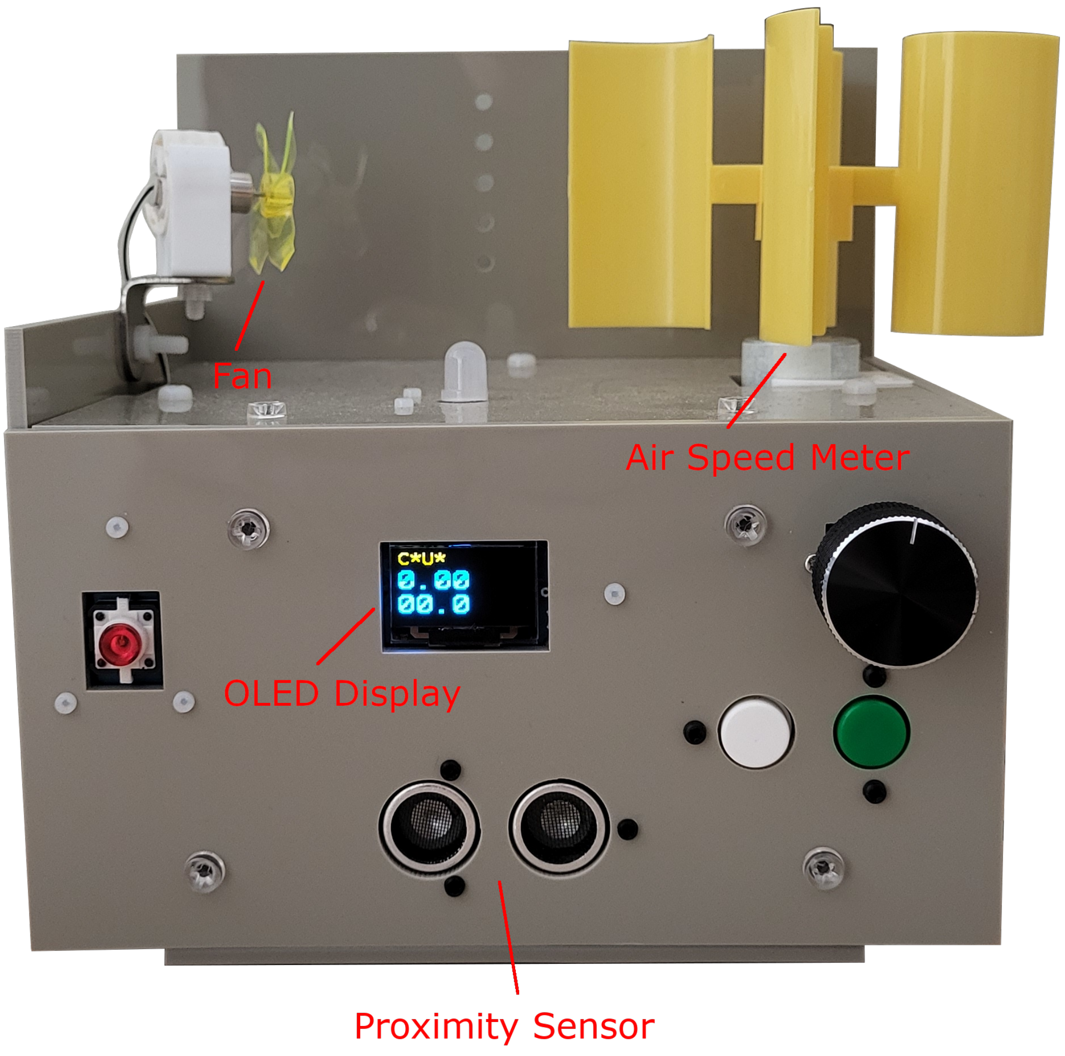


Figure 2.3: Ventilation Machine.

2.2. Software Architecture

The machine's software consists of a program developed in Python 3.10 that runs as a systemd service on the Raspberry Pi OS. The service starts after the system has finished booting and after the network has been set up by the network management software. The software is divided into:

- Main routine: sets up the connection, the board and manages the communication with the electronic components.
- sy_client package: contains all the code to instantiate a Servitly client that handles the communication with the Servitly server via MQTT.

- grove_components package: contains all the code to instantiate and use Grove modules.

When the service starts, the main routine loads the MQTT credentials configuration file, connects to the MQTT broker server instantiating a Servitly Client object and subscribes to the machine topic.

MQTT (Message Queuing Telemetry Transport) is a publish-subscribe network protocol for message queuing that is widely used in IoT applications due to its low bandwidth requirements.
There are three main components in MQTT:

- Client: it is any device or application that sends or receives messages.
- Broker: it is a server that receives messages from clients and forwards them to their destination clients.
- Topic: it is a string that defines the subject of the message.

In MQTT, clients can publish messages on a specific topic to the broker. The broker then receives the message and forwards it to all clients that have expressed an interest in that particular topic via a subscription.

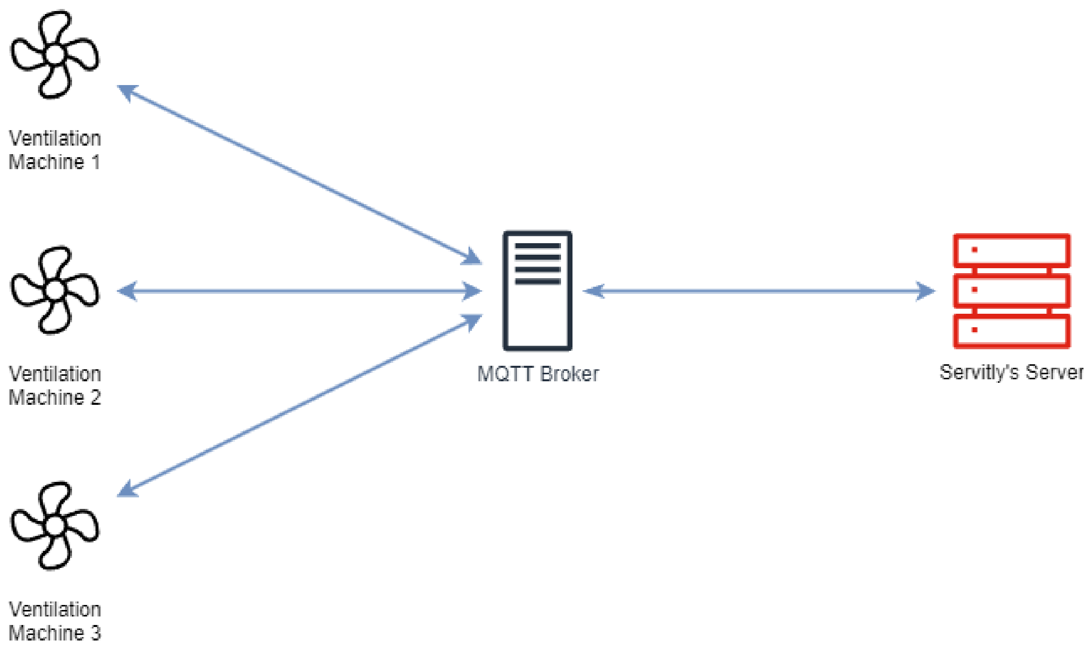


Figure 2.4: MQTT Communication diagram

The library used for MQTT communication is paho-mqtt [3], a MQTT implementation developed by Eclipse.

```json
{
    "username": "example",
    "password": "password",
    "host": "127.0.0.1",
    "force_publish_interval": 60
}
```

Listing 2.1: Example of MQTT configuration file

After connecting to the main routine, the software loads the board configuration, a file with all the pin connections for each component and functional parameters, and starts setting up all the different components.

```json
{
    "connections": {
        "anemometer": 0,
        "sonar": 5,
        "encoder": 24,
        "red_led": 16,
        "rgb_led": 18,
        "red_button": 17,
        "green_button": 27,
        "white_button": 26
    },
    "motor": {
        "direction": -1,
        "min_speed": 60,
        "channel": "A",
        "max_rpm": 3400
    },
    ...
}
```

Listing 2.2: Example of Board configuration file

As already mentioned in the *grove_ components* package there are all the Python classes to use each component. To implement them grove.py [1] library was used. It is the official Python library for Seeed Studio Grove Devices.

Each component has a own thread and everything works in a event-driven fashion. When a piece of data is sampled, a callback is called by the thread in order to update the board status.

The board status is an object managed by the main routine composed by several variables that store the current status of the board.
Every second, if something has changed, the status is published via MQTT to Servitly's message broker at topic `"{username}/{machine_id}/measures"`. The payload is a JSON string of the format:

```
{
    "ts": 1672531200000,
    "data": {
        "motor_speed": 190,
        "motor_running": true,
        "user_detected": true,
        "cleaning_cycle": false,
        "error_code": "OK",
        ...
    }
}
```

Listing 2.3: Example of board status MQTT paylod

In addition to publishing the status of the board, the machine can receive messages, such as commands and parameters, from Servitly's server. Commands can be used to perform a task, operation, or routine on the remote product. Some examples of commands are:

- Cleaning cycle: triggers the cleaning procedure on the machine.
- Motor ON/OFF: turns the machine's fan ON/OFF.

Commands messages are received on the topic `"{username}/{machine_id}/commands"`.

Through the parameters, on the other hand, it is possible to update the operating settings of the machine. For instance:

- Cleaning cycle duration: sets the duration of the machine's cleaning procedure.

- Motor speed: sets the speed of the machine's fan.

Parameters messages are received on the topic `"{username}/{machine_id}/parameters"`.

3 | Analysis

3.1. Objectives of the data analysis

As explained in the previous chapter, the purpose of the constructed machine is to simulate a controlled ventilation system. The air speed meter installed on the machine measures the speed of the air generated by the fan: it provides information about the operating status of the machine. Since analyzing and predicting these measurements could help detect an anomaly in a timely manner, it was decided to perform data analysis to predict air speed meter measurements. This information can be very useful in a real-world application. This chapter shows how it has been attempted to predict machine sampled data using three different prediction models.

3.2. Preliminary experiments

The data set was built sampling the measurements of the air speed meter with the Raspberry Pi configuration already discussed. Data were sampled at 20Hz for a period of 30 minutes at the normal operating speed of the machine.

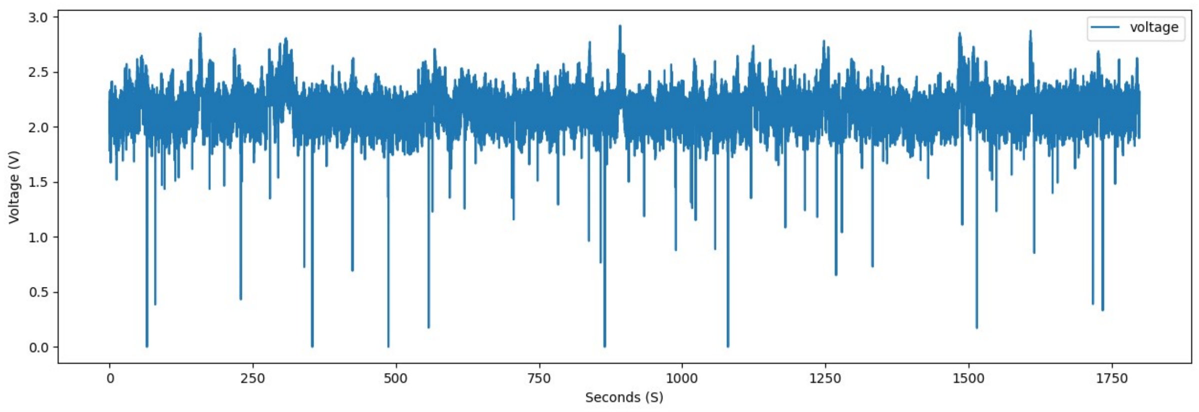


Figure 3.1: Plot of the sampled data.

After sampling the data, some preliminary analyses were performed. First, some tests were performed to examine the stationary nature of the time series. A time series is defined as stationary if the statistical properties (such as mean and variance) do not depend on when the series is observed. For this reason, time series with trends or seasonality are not stationary [14].

Stationary time series are easier to predict, which is why this first analysis was performed. Augmented Dickey-Fuller [17] and Kwiatkowski–Phillips–Schmidt–Shin [15] tests were used to study stationarity.

The Augmented Dickey-Fuller (ADF) test is a test used to identify the presence of unit root in the set. The null hypothesis is the presence of unit root, while the alternative hypothesis is that there is no unit root.

A unit root is a nonstationary process characterized by the fact that the first difference is stationary. It has the form:

$$y_t = y_{t-1} + \varepsilon_t$$

where ε_t is a stationary process.

The Kwiatkowski–Phillips–Schmidt–Shin (KPSS) test, on the other hand, is a test whose null hypothesis is that the process is trend stationary, while the alternative hypothesis is the presence of unit root.

When we run ADF on our data set, we get these results:

```
Results of ADF Test:
Test Statistic                -1.832752e+01
p-value                        2.259300e-30
Critical Value (1%)           -3.430532e+00
Critical Value (5%)           -2.861620e+00
Critical Value (10%)          -2.566813e+00
```

Given the obtained p-value, we can reject the null hypothesis of the presence of unit root.

When we run KPSS on our data set, we get these results:

```
Results of KPSS Test:
Test Statistic              0.126969
p-value                     0.100000
Critical Value (10%)        0.347000
Critical Value (5%)         0.463000
Critical Value (2.5%)       0.574000
Critical Value (1%)         0.739000
```

Given the obtained p-value, we cannot reject the null hypothesis: this means that the process is trend stationary. The test results suggest the absence of unit root and the fact that the time series is trend stationary.

To explain the concept of trend stationary we need to introduce 3 concepts used when describing time series [14]:

- Trend: The trend represents the long-term increase or decrease in a time series data over an extended period of time.
- Seasonal: The seasonal component in a time series refers to the regular variations that occur at specific known periods.
- Cyclic: The cyclic component represents fluctuations in the time series data that do not have a fixed frequency, such as seasonality, but occur over irregular, non-repeating cycles.

A trend-stationary process is a stochastic process from which a trend, a function solely of time, can be subtracted to obtain a stationary process. It has the form:

$$y_t = f(t) + \varepsilon_t$$

where ε_t is a stationary process.

Trend-stationarity does not guarantee stationarity, so the absence of seasonality is not ruled out. Therefore, the analysis was continued in an attempt to decompose the time series.

A time series can be seen as the sum of three components [14]:

$$y_t = S_t + T_t + R_t$$

where y_t is the time series, S_t is the seasonal component, T_t is the trend-cycle component, and R_t is the remainder component, all at period t.

To use a decomposition algorithm, it is necessary to know the period of the seasonality component, if any. To attempt to identify the period, the FFT of the time series was performed using SciPy [20]. Plotting the result gives the following graph:

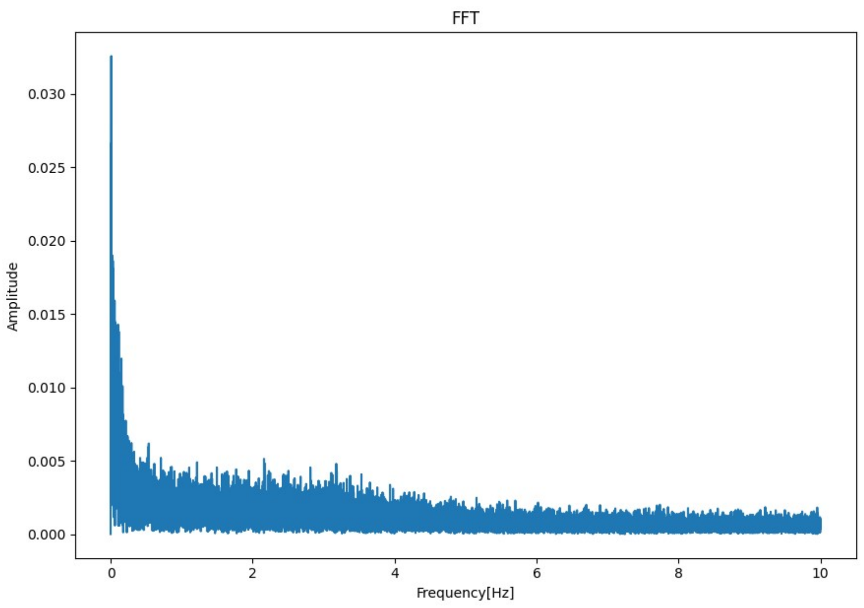


Figure 3.2: Plot of the FFT.

By zooming in on the first peaks of the FFT, it can be seen that there is a slight dominance of one peak over the remaining peaks at frequency 0.008 Hz.

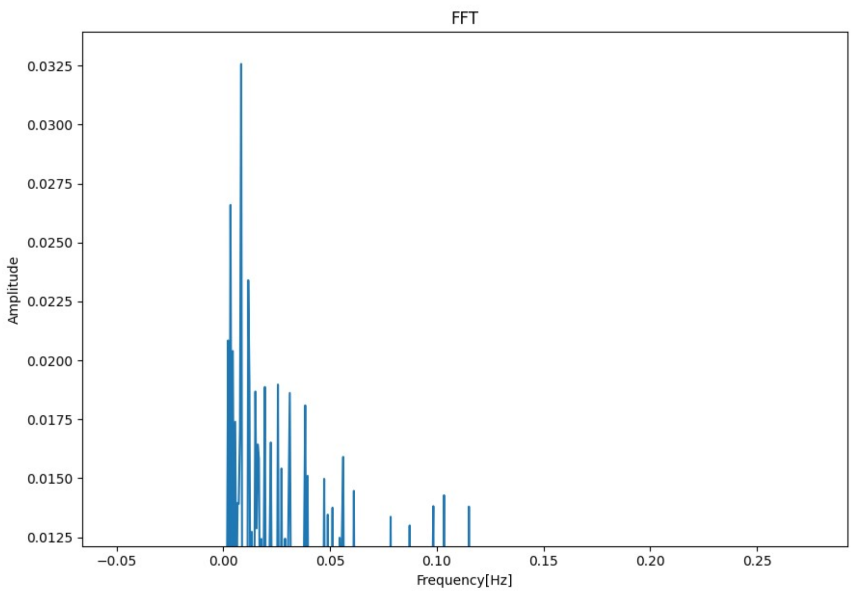


Figure 3.3: Close-up on major peaks.

For this reason, a decomposition was attempted using a period equal to $1/0.008 = 125$s.

The first algorithm employed was Seasonal-Trend decomposition using LOESS (STL) [10]. It is an algorithm that attempts to decompose the starting time series into the three components already introduced: trend, seasonal and residual. The implementation of statsmodels [18] was used in our analysis.

Several attempts were made to adjust the parameters of the algorithm, but without obtaining a satisfactory decomposition result. One of the decomposition attempts made can be seen in the following figure:

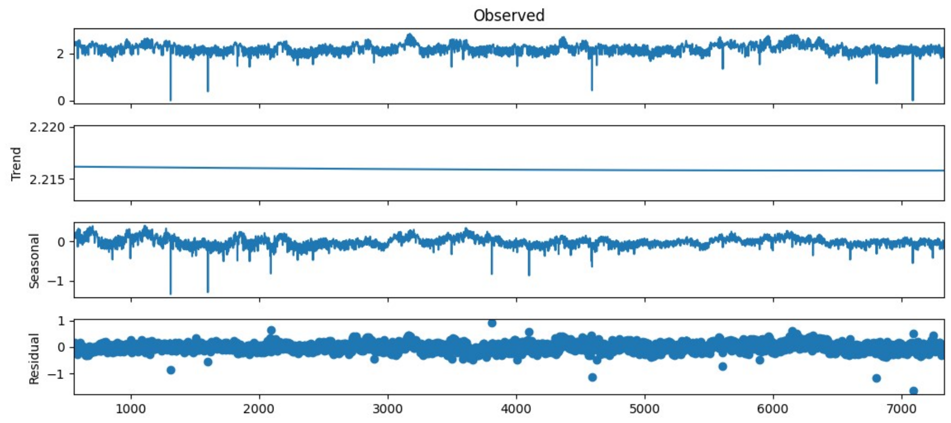


Figure 3.4: One of the experiments with STL decomposition.

The seasonal component in the graph is clearly non-periodic and therefore does not represent useful information for our purposes.

Further decomposition tests were then performed. In the first one, the RobustSTL algorithm was used. RobustSTL [21] is an algorithm that attempts to make up for some of the shortcomings of STL by proving to be more effective in decomposition in the presence of outliers and a long seasonality period. No official implementation of the algorithm has been released; however, there is an unofficial open source implementation [16] that was used for our tests. Then a decomposition attempt was made using MSTL [7], an extension of the traditional Seasonal-Trend Decomposition using Loess (STL) procedure. It was used to decompose time series with multiple seasonal patterns. The implementation of statsmodels [18] was used in our analysis.

The periods used in the decomposition with the two algorithms were calculated from the three highest peaks frequencies of the FFT i.e. 0.008 Hz, 0.011 Hz and 0.003Hz. Again, despite several adjusting of the algorithm parameters, a satisfactory result was not obtained.

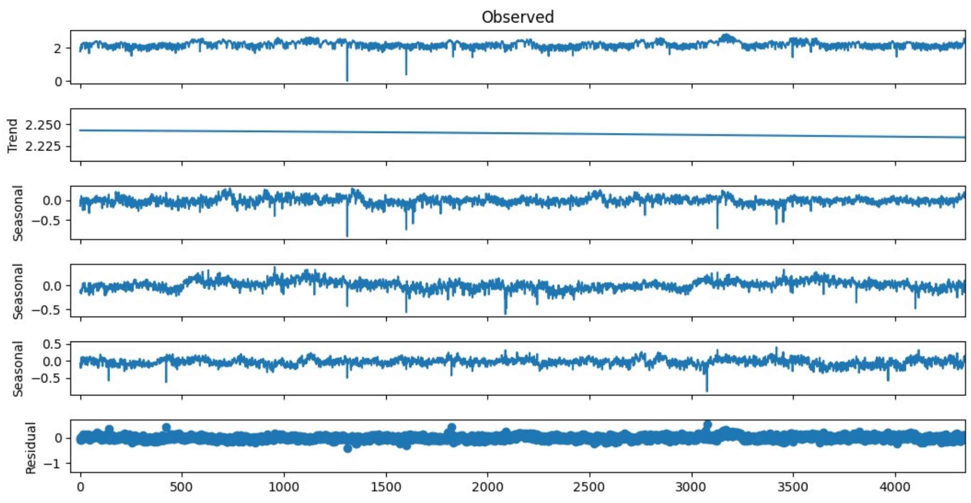


Figure 3.5: One of the experiments with MSTL decomposition.

After performing all these preliminary tests, it was concluded that the seasonal component, if present, is not trivial. Therefore, we moved on to the prediction stage.

3.3. Data prediction and result

For the prediction phase, it was decided to compare three prediction models: a traditional model, a machine learning model, and a deep learning model. For the traditional model, ARIMA was chosen. ARIMA stands for AutoRegressive Integrated Moving Average. It's a model composed of three parts [14]:

- AutoRegressive (AR): it's a regression made by a linear combination of its previous values, where each of them is weighted by a coefficient. The order "p" of the model represents the number of past time steps (lags) used to build the regression. An autoregressive model of order p, or $AR(p)$, can be written as

  $$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \varepsilon_t$$

  where ε_t is white noise and $\phi_1 \cdots \phi_p$ are the weights.

- Integrated (I): It's the differencing of the time series data. Differencing is used to make the time series stationary. The parameter "d" represents the number of times the data is differenced to achieve stationarity.

- Moving Average (MA): Instead of using past values, a moving average model uses past forecast errors to build a regression. It's made up of a linear combination of past prediction errors, each weighted by a coefficient. The order "q" of the model represents the number of past time steps (lags) used to build the regression. A moving average model of order q, or $MA(q)$, can be written as

$$y_t = c + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \cdots + \theta_q \varepsilon_{t-q}$$

where ε_t is white noise and $\theta_1 \cdots \ \theta_q$ are the weights.

The full model can be written as:

$$y'_t = c + \phi_1 y'_{t-1} + \cdots + \phi_p y'_{t-p} + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q} + \varepsilon_t$$

where y'_t is the differenced series. This is called $ARIMA(p, d, q)$ model.
The model fitting is done with pmdarima [19], a library that iteratively calculates information criteria such as Akaike information criterion (AIC) and Bayesian information criterion (BIC) to select a better order of each component (p, d and q). AIC and BIC are both statistical criteria used for model selection, particularly in the context of regression analysis, time series modeling, and other statistical modeling techniques. pmdarima tries to minimize them to find a good model: the lower they are, the better the model will be.

For the machine learning model, gradient boosting [11, 12] was chosen. Gradient boosting is a popular machine learning technique used for classification and regression tasks. It is an iterative algorithm that builds a strong predictive model by summing several simple models. The simple models are called weak learners and are typically decision trees. At each iteration, a new weak model is added to the strong model, and the new model is trained by minimizing a loss function; while training a weak model, the algorithm attempts to minimize a loss function by adjusting the parameters of the new model to move in the opposite direction of the gradient of the function. Training stops after a fixed number of iterations, when the model is good enough, or when no further improvements are made.
In this project, XGBoost [8] is used, a library that implements machine learning algorithms using the gradient boosting technique. Since the task to execute is a regression task, the squared error was used as loss function. Training was done using lagged data with a lag of 15. The lag value is the best found after several attempts to improve the model.

Finally, for the deep learning approach, it was decided to build a LSTM [13] neural network. Long Short-Term Memory (LSTM) is a type of RNN that solves the vanishing gradient problem of traditional RNNs. A recurrent neural network (RNN) is a type of neural network specifically designed to work with sequences of data. It has connections that loop back on themselves, allowing information from previous inputs to persist. RNNs process sequences one element at a time, updating their internal state as new elements are fed in. However, standard RNNs can suffer from the vanishing gradient problem, which makes it difficult for them to capture long-range dependencies in sequences because the gap between the relevant information and the point where it is needed can become very large. LSTMs are explicitly designed to avoid the long-range dependency problem; instead of a single neural network layer, there are four interacting neurons. The LSTM has the ability to selectively remember and forget information over long sequences [2]. In this project, Keras [9] was used to implement and train the LSTM network. The architecture of the network is composed one input layer, three LSTM layers and on output layer. The network was trained with a window of 15 and for 50 epochs.

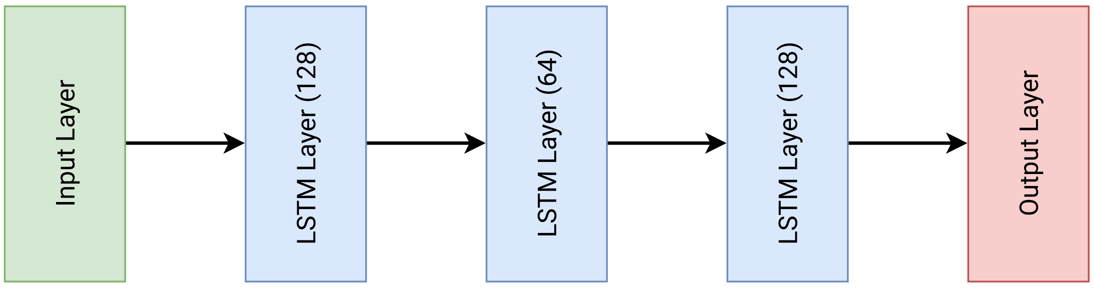


Figure 3.6: Architecture of LSTM network.

The forecasting algorithm developed is iterative. The basic idea is to try to predict the time series from which a periodic signal composed of the main periodic components of the time series has been subtracted. At each iteration, a periodic component is added to the periodic signal.

To construct the periodic signal, the frequencies and phases of the principal peaks of the FFT are extracted. At each iteration i, a signal is created by summing the cosine waves of the same frequency and phase as those extracted up to the i-th peak. The periodic signal is then subtracted from the original time series.

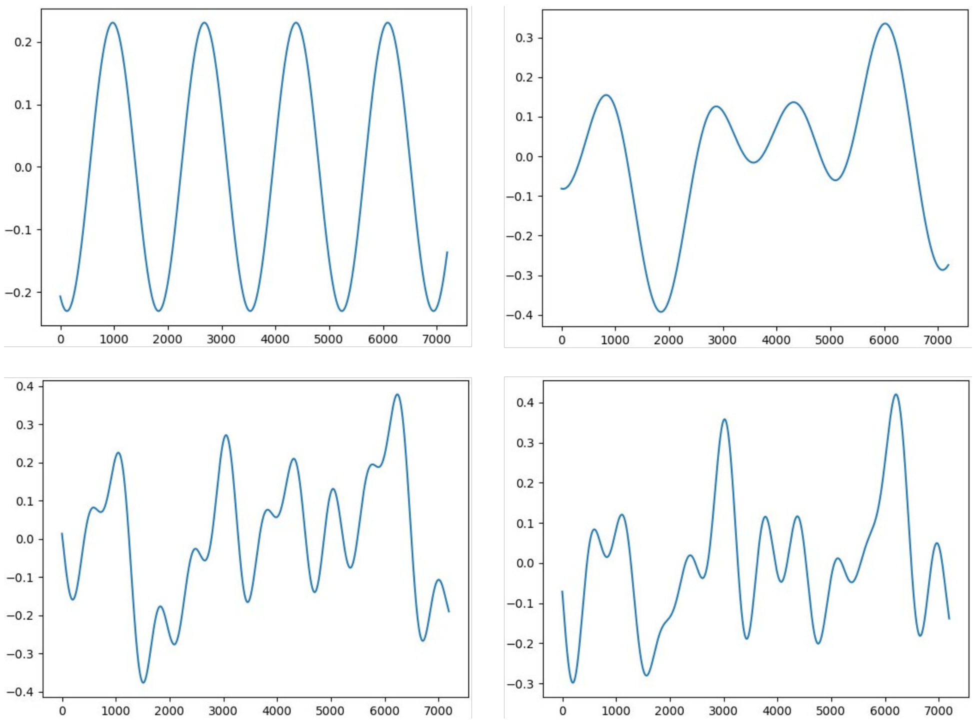


Figure 3.7: Periodic signal at 1st, 3rd, 5th and 7th iteration.

After subtraction, the data set is split into a training set and a test set, with a proportion of 0.8 and 0.2 of the total data set, respectively. As mentioned before, the training set is used for fitting:

- An ARIMA model
- A gradient boosting regressor
- A neural network made up of LSTM cells

The test set is then predicted, updating the models with the real measures at each prediction. For each predicted measure and for each model, the absolute error with the real measure is computed. At the end of the test set, the average of the errors is computed for each model.

The results of the mean absolute error at each iteration for each of the methods are as follows:

Iteration	**ARIMA**	**XGBoost**	**LSTM**
0	0.0682	0.0367	0.0353
1	0.0681	0.0471	0.0507
2	0.0681	0.0475	0.0446
3	0.0682	0.0476	0.0441
4	0.0683	0.0477	0.0409
5	0.0684	0.0492	0.0420
6	0.0683	0.0492	0.0422
7	0.0684	0.0497	0.0407
8	0.0684	0.0493	0.0435
9	0.0686	0.0501	0.0430
10	0.0685	0.0495	0.0433
11	0.0685	0.0507	0.0414
12	0.0686	0.0499	0.0431
13	0.0687	0.0519	0.0445
14	0.0688	0.0513	0.0470

Table 3.1: Mean absolute error results for each model.

Iteration 0 is the prediction of the time series without subtracting the periodic signal.

It can be clearly seen that the best results are obtained at iteration zero, when no periodic signal is subtracted. Furthermore, the best prediction model seems to be the LSTM network, followed by XGBoost regressor and finally ARIMA.
The prediction results also do not seem to improve as the iterations increase, i.e., as the components used to form the periodic signal increase.

Iteration	ARIMA	XGBoost	LSTM
0	3.43%	2.00%	1.89%
1	3.43%	2.43%	2.56%
2	3.43%	2.46%	2.34%
3	3.43%	2.45%	2.30%
4	3.44%	2.46%	2.15%
5	3.44%	2.53%	2.22%
6	3.44%	2.54%	2.21%
7	3.44%	2.56%	2.14%
8	3.44%	2.54%	2.28%
9	3.45%	2.58%	2.24%
10	3.45%	2.55%	2.27%
11	3.45%	2.60%	2.18%
12	3.45%	2.56%	2.25%
13	3.46%	2.67%	2.33%
14	3.46%	2.64%	2.43%

Table 3.2: Mean absolute percentage error results for each model.

The same conclusion can be drawn if we look at the mean absolute percentage error.

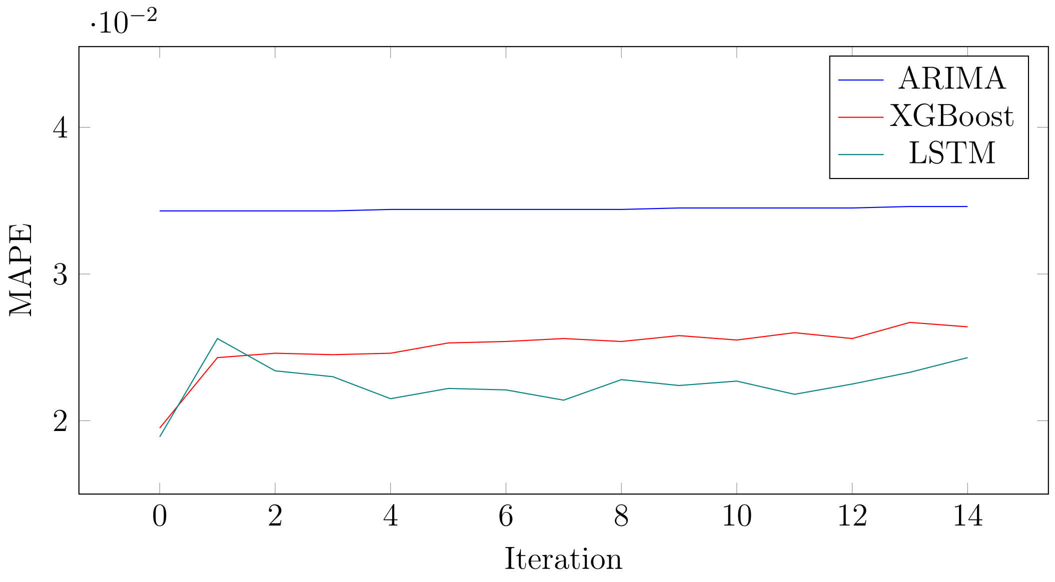


Figure 3.8: Plot of mean absolute percentage error.

4 | Conclusions

This thesis proposes an example of the design and implementation of a connected machine and a comparison of prediction methods applied to real measurements obtained from the machine itself.

The proposed approach consists in the comparison of three prediction methods, one traditional, one machine learning and one deep learning. The effectiveness of subtracting a signal composed of the main periodic components of the measured signal is also evaluated.

The results of the comparison show that the LSTM network achieves the best result with a mean absolute percentage error between 1.89% and 2.56%. It is followed by the XGBoost model with a MAPE between 2.00% and 2.67% and finally the ARIMA model with an error between 3.43% and 3.46%. The best results are obtained without subtracting the periodic signal.

A possible future development could be to use these models in the context of anomaly detection to detect unexpected machine operating behaviors and evaluate their performance in detecting anomalies.

Bibliography

[1] Python library for Seeedstudio Grove Devices. `https://github.com/Seeed-Studio/grove.py`.

[2] Understanding LSTM Networks. `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`. Accessed 2023-08-29.

[3] Eclipse Paho MQTT python client library. `https://pypi.org/project/paho-mqtt/`.

[4] Raspberry Pi. `https://www.raspberrypi.com/`.

[5] Seeed Studio. `https://www.seeedstudio.com/`.

[6] Servitly. `https://www.servitly.com/`.

[7] K. Bandara, R. J. Hyndman, and C. Bergmeir. Mstl: A seasonal-trend decomposition algorithm for time series with multiple seasonal patterns. *arXiv preprint arXiv:2107.13462*, 2021. doi: 10.48550/arXiv.2107.13462. URL `https://doi.org/10.48550/arXiv.2107.13462`.

[8] T. Chen and C. Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4232-2. doi: 10.1145/2939672.2939785. URL `http://doi.acm.org/10.1145/2939672.2939785`.

[9] F. Chollet et al. Keras. `https://keras.io`, 2015.

[10] R. B. Cleveland, W. S. Cleveland, J. E. McRae, and I. Terpenning. Stl: A seasonal-trend decomposition procedure based on loess. *Journal of Official Statistics*, 6:3–73, 1990.

[11] J. H. Friedman. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29(5):1189 – 1232, 2001. doi: 10.1214/aos/1013203451. URL `https://doi.org/10.1214/aos/1013203451`.

[12] J. H. Friedman. Stochastic gradient boosting. *Computational Statistics & Data Analysis*, 38(4):367–378, 2002. ISSN 0167-9473. doi: https://doi.org/10.1016/S0167-9473(01)00065-2. URL `https://www.sciencedirect.com/science/article/pii/S0167947301000652`. Nonlinear Methods and Data Mining.

[13] S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 11 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL `https://doi.org/10.1162/neco.1997.9.8.1735`.

[14] R. J. Hyndman and G. Athanasopoulos. *Forecasting: principles and practice*. OTexts, Melbourne, Australia, 3rd edition, 2021. URL `https://otexts.com/fpp3`. Accessed 25 July 2023.

[15] D. Kwiatkowski, P. C. Phillips, P. Schmidt, and Y. Shin. Testing the null hypothesis of stationarity against the alternative of a unit root. *Journal of Econometrics*, 54 (1-3):159–178, Oct. 1992. doi: 10.1016/0304-4076(92)90104-y. URL `https://doi.org/10.1016/0304-4076(92)90104-y`.

[16] D. Lee. Robuststl implementation. `https://github.com/LeeDoYup/RobustSTL`, 2019.

[17] S. E. Said and D. A. Dickey. Testing for unit roots in autoregressive-moving average models of unknown order. *Biometrika*, 71(3):599–607, 1984. doi: 10.1093/biomet/71.3.599. URL `https://doi.org/10.1093/biomet/71.3.599`.

[18] S. Seabold and J. Perktold. statsmodels: Econometric and statistical modeling with python. In *9th Python in Science Conference*, 2010.

[19] T. G. Smith et al. pmdarima: Arima estimators for Python, 2017–. URL `http://www.alkaline-ml.com/pmdarima`. Accessed 28 July 2023.

[20] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17: 261–272, 2020. doi: 10.1038/s41592-019-0686-2.

[21] Q. Wen, J. Gao, X. Song, L. Sun, H. Xu, and S. Zhu. Robuststl: A robust seasonal-trend decomposition algorithm for long time series. *Proceedings of the AAAI Con-*

ference on Artificial Intelligence, 33(01):5409–5416, 2019. doi: 10.1609/aaai.v33i01.33015409. URL `https://doi.org/10.1609/aaai.v33i01.33015409`.

List of Figures

List of Tables

Listings

Ringraziamenti

Desidero ringraziare coloro hanno contribuito al completamento di questa tesi: il mio relatore, il professor Piero Fraternali, e il mio correlatore Nicolò Oreste Pinciroli Vago. Grazie per avermi guidato in questo percorso, per i vostri preziosi consigli e per la vostra disponibilità.

Desidero anche ringraziare Servitly, in particolar modo Stefano Butti ed Andrea Ghezzi. Grazie per avermi concesso la possibilità di svolgere questo progetto in collaborazione con voi e per la calorosa accoglienza che mi avete riservato.

Ringrazio i miei amici, dagli storici della *Stanza* ai più recenti della *Gentooneria*. Grazie per aver contribuito ad alleggerire giornate difficili, per aver condiviso risate e scambiato consigli.

Grazie alla mia famiglia per aver sempre creduto in me e per essermi stata accanto in questo tortuoso percorso. In particolare voglio ringraziare i miei genitori per aver reso possibile questo traguardo e per avermi spronato anche nei momenti più difficili.

Grazie a Noemi, la mia ragazza. Grazie per avermi supportato ed incoraggiato durante tutti questi mesi e per aver creduto in me anche quando nemmeno io lo stavo facendo. Grazie anche per essere riuscita a sopportarmi nei momenti più intensi, sei stata un punto di riferimento essenziale in questo mio viaggio.

Ci tengo infine a ringraziare anche tutte le persone non nominate ma che, ogni giorno o anche solo una volta, mi sono state accanto o mi hanno offerto consigli e supporto durante il mio percorso di studi.