UNIVERSITY OF THE PELOPONNESE & NCSR "DEMOCRITOS"

MSC PROGRAMME IN DATA SCIENCE

A framework for the collection, aggregation, collation and prediction of meteorological data

Dimitrios Xenakis
dsc19022

A thesis submitted in partial fulfillment
of the requirements for the MSc
in Data Science

Supervisor: Christos Tryfonopoulos, Professor

Athens, March 2024

UNIVERSITY OF THE PELOPONNESE & NCSR "DEMOCRITOS"

MSC PROGRAMME IN DATA SCIENCE

A framework for the collection, aggregation, collation and prediction of meteorological data

Dimitrios Xenakis
dsc19022

A thesis submitted in partial fulfillment
of the requirements for the MSc
in Data Science

Supervisor: Christos Tryfonopoulos, Professor

Approved by the examination committee on March 2024

(Signature)	(Signature)	(Signature)
……. .	……	……. .
Christos Tryfonopoulos	**Theodoros Giannakopoulos**	**Spiros Skiadopoulos**
Professor	Principal Researcher	Professor

ATHENS, MARCH 2024

Dimitrios Xenakis
MSc. Thesis, MSc. Programme in Data Science
University of the Peloponnese & NCSR “Democritos”, March 2024

UNIVERSITY OF THE PELOPONNESE & NCSR "DEMOCRITOS"

MSC PROGRAMME IN DATA SCIENCE

Declaration of Authorship

1. I declare that this thesis has been composed solely by myself and that it has not been submitted, in whole or in part, in any previous application for a degree. Except where states otherwise by reference or acknowledgment, the work presented is entirely my own.

2. I confirm that this thesis presented for the degree of Bachelor of Science in Informatics and Telecommunications, has
    a. been composed entirely by myself
    b. been solely the result of my own work
    c. not been submitted for any other degree or professional qualification

3. I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or processional qualification except as specified.

(Signature)

. . . . . .

Dimitrios Xenakis

Acknowledgments

The present postgraduate thesis "**Analysis, visualization, forecasting of meteorological data by machine learning methods**" was prepared within the inter-departmental postgraduate program in Data Science, co-organised by the Department of Informatics and Telecommunications of the University of Peloponnese and the Institute of Informatics and Telecommunications of National Centre for Scientific Research “Demokritos”.

I would like to express my warm thanks to my supervisor Professor Christos Tryfonopoulos, for the opportunity he gave me to prepare this postgraduate thesis, for the trust he showed me as well as for the excellent cooperation we had throughout its preparation.

In particular, I warmly thank the teaching staff of the University of the Peloponnese as well as the Institute of Informatics for the knowledge and experiences they offered me.

I would also like to thank Vicky for her undivided and practical support throughout the elaboration of the work.

Finally, I thank my family for the support and trust they have shown me during my studies, as well as my close friends for their invaluable support and understanding

ΠΕΡΙΛΗΨΗ

Στην παρούσα διπλωματική εργασία μελετήσαμε το κατά πόσο οι μετρήσεις δεδομένων καιρού που προέρχονται απο διαφορετικούς μετεορολογικούς σταθμούς και πηγές συγκλίνουν μετάξύ τους. Δεδομένου οτι υπάρχει πληθώρα μετεορολογικών σταθμών,για κάποια συγκεκριμένη γεωγραφική περιοχή, παραμένει ασαφές το γεγονός, κατά πόσο οι διαφορετικοί αυτοί σταθμοί παρέχουν παρόμοια ή οχι δεδομένα.

Σκοπός της εργασίας αυτής ειναι να συγκεντρώσει μετεορολογικά δεδομένα από διαφορετικές πηγές για την περιοχή της Τρίπολης , έδρας του Πανεπιστημίου Πελοποννήσου και να μπορέσει μέσα απο διαγράμματα και γραφήματα να προσφέρει ποσοτικά και ποιοτικά αποτελέσματα, όσο αφορά αποκλίσεις που ενδοχομένως υπάρχουν μεταξύ των σταθμών.

Επίσης στην παρούσα εργασία γίνεται μια προσπάθεια για την δημιουργία ενός βραχυπρόθεσμου μοντέλου πρόβλεψης θερμοκρασίας χρησιμοποιώντας ενα LSTM νευρωνικό δίκτυο και βασίζεται σε ιστορικά δεδομένα που έχουμε ήδη συγκεντρώσει.

Η μεθοδολογία που ακολουθήσαμε ήταν αρχικά η συγκέντρωση των δεδομένων απο τις διαφορετικές πηγές, χρησιμοποιώντας web scraping, ενώ έπειτα τα δεδομένα αυτά αφου μετασχηματιστούν σε κοινές μονάδες μέτρησης αποθηκεύονται μόνιμα. Έχοντας τα δεδομένα αποθηκευμένα μπορούμε να δημιουργήσουμε γραφήματα οπτικοποιώντας τυχόν διαφορές που υπάρχουν σε μια δεδομένη χρονική στιγμή. Αναφορικά με το μοντέλο πρόβλεψης, χρησιμοποιήθηκέ η τεχνική του Grid Search για την εύρεση των βέλτιστων τιμών των παραμέτρων του. Το μοντέλο μας χρησιμοποιεί τις τελεύταιες 64 μετρήσεις για να προβλέψει τις επόμενες 12 οι οποίες αντιστοιχούν σε ένα χρονικό ορίζοντα 6 ωρών.

Μέσα απο την παραπάνω διαδικασία καταλήγουμε στα συμπεράσματα οτι υπάρχουν σαφής διαφορές μεταξύ των μετεορολογικών σταθμών στην περιοχή της Τρίπολης, παρόλο που οι σταθμοί αυτοί βρίσκονται σε κοντινές αποστάσεις μεταξύ τους, όπως επίσης ότι είναι εφικτή η δυνατότητα βραχυπρόθεσμης πρόβλεψης θερμοκρασίας μέσω νευρωνικών δικτύων.

Λέξεις Κλειδιά
Συλλογή μετεορολογικών δεδομένων, Σύγκριση μετεορολογικών δεδομένωνν, Πρόβλεψη θερμοκρασίας, Machine learning, Deep learning, Long short-term memory, LSTM, Elasticsearch, Kibana, Scrapy.

ABSTRACT

Nowadays, there are several different sources (weather stations, sites) of meteorological data, regarding a specific geographical area, however it remains uncertain and unclear how much these data converge. One approach to solve this particular problem is to collect meteorological data from different stations for the same geographical area, visualize and aggregate them in a time series analysis. The above approach was used for the city of "Tripoli" – Peloponesse – Greece, and after collecting data from seven different local meteorological stations, we concluded that there are clear differences on climate parameters (temperature, humidity, wind, precipitation, etc.) between the different stations, even though these stations are very close to each other. Also, using the historical data that have been collected, we train and deploy a short-term temperature prediction model, using a biredirectional LSTM models. The deployed model proves to be accurate in the prediction of temperature for the specific area, demonstrating its efficiency for short-term predictions of temperature values.

Keywords:
Meteorological data collection, Meteorological data aggregation, Temperature forecasting, Temperature prediction, Machine learning, Deep learning, Long short-term memory, LSTM, Elasticsearch, Kibana, Scrapy.

Table of Contents

List of Tables

List of Figures

Chapter 1 – Introduction

1.1 Thesis Purpose

The initial purpose of this thesis is to examine the discrepancy between meteorological data, obtained from different stations of the same geographical area. Although meteorology vendors provide visualization and statistics tools for their data, there is no any known tool that compares data of different providers regarding their discrepancies. The goal of the curent work is to develop a tool that will visualize data from different meteorological sources in a time series scale and detect differences of weather variables (such as temperature, rain, atmospheric pressure, etc.) with quantitative and qualitative characteristics. The above tool uses various existing software technologies regarding data collection and storage while proves that there are clear differences on climate parameters between the different weather stations of the region of Tripoli – Peloponnese – Greece.

The next aim of this thesis is to create a short term weather forecast model using the data of the previous section applying machine learning methods. Related studies and literature suggest deep learning models as a novel way of weather forecasting while a state of the art artificial neural network which has been used is the Long short-term memory – LSTM. The present work, by applying related studies confirms that deep learning models can provide short-term weather forecasts with high accuracy and reliability.

It is worth noticing that it was given great emphasis on the architecture and application development tools, both in terms of operation and data storage, providing the ability of continuous and uninterrupted operation.

1.2 Existing Approach and Differentiation

Every weather data provider (websites, meteo stations) provides information about the weather progress based on their measurements and data. Although this approach offers to the public, plenty of weather data from different sources, it is difficult to examine if these different measurements converge to each other. Therefore, comparison and aggregation is a novel process for examining the convergence between data of different weather sources. The differentiation of the current work is to collect, store, visualize and aggregate data from different sources and stations for comparison and aggregation purposes.

Another important process that utilizes weather data is forecasting. The weather prediction process has always been a major factor in the everyday human life. The first reference related to the concept that the natural phenomena can be described by mathematical equations, was formulated by Isaac Newton (Newton, 1680). By the year 1922, the English mathematician Lewis Richardson describes at his work (Richardson, 1922) a weather forecast model using differential equations. The problem at this time rather than the accuracy of the model was the lack of computing power while the calculations were made by hand. At the decade of 1960 Edward Lorenz (Lorenz, 1963) introduce a model using chaotic differential equations in order to describe weather data parameters with better accuracy. At this time, the first computer systems were used also to solve the proposed weather model.

From that, time on the main problem was the computer power. Beginning from 1980 the first international centers for weather forecasting such as E.C.M.W.F. - European Center for Mesoscale Weather Forecasts [1] have been created using computers like CRAY [2] and Cyber 205. Still now, the scientific community utilizes classical weather forecasting models using differential equations, combined with modern supercomputers and techniques in order to improve the prediction process[3].

However, a part of the scientific community tries to deploy novel techniques and methods for weather forecasting, using machine learning models and algorithms overcoming the existing approach using differential equations. Research centers and organizations such as ESA, E.C.M.W.F., (Schneider, Massimo, Geer, & Arcucci, 2022) and IEEE (Mishra & Joshi, 2021) differentiate and take advantage of these new machine learning methods.

Our approach differs from the classical (through differential equations) weather forecasting process since it uses LSTM (Long short-term memory), a state of the art deep machine learning model and confirms alongside related studies, that deep learning models can provide reliable short-term weather forecasts.

[1] https://www.ecmwf.int/en/forecasts/documentation-and-support
[2] https://www.craysupercomputers.com/
[3] Weather and Forecasting - American Meteorological Society (ametsoc.org)

1.3 Organization and Presentation

In the following chapters are developed the sub-sectors of the current work. Chapter 2 presents the related work based on the mathematical approach to meteorology, while at Chapter 3 we present the tools and technologies, which have been used at the present work. The data collection, which is a major factor of our development, is presented in Chapter 4. In Chapter 5 is analyzed the weather forecasting process based on deep learning model while at Chapter 6, the reader can see the results of data analysis (aggregation) and forecasting using the corresponding interactive graphs. Finally, Chapter 7 presents the results of aggregation and forecasting and also, we suggest possible methods to improve the present work.

Chapter 2 – Related Work

In our days, the goal of high accuracy within existing models such as ECMWF, ICON, GFS, NAM, and HRRR has been achieved[4]. However the challenge of scientific community is to rely entirely on algorithms and machine learning tools in order to produce high accuracy, reliable weather forecast models, which can been used in many sectors of everyday life such as agricultural, sea transport and many others. Additionally, when using machine learning, the selection of the algorithm/ technique is not at all a trivial task. The well-established weather research and forecasting (WRF) that constitute the current state-of-the-art use algorithms such as Standard Regression (SR) (Sharaff A, 2018), Support Vector Regression (SVR) (Voyant, 2017), Random Forest (RF) (Sanchez-Fernandez, de-Prado-Cumplido, Arenas-Garcia, & Perez-Cruz, 2004), Autoregressive Integrated Moving Average (ARIMA) (M, Islam, Nadvi , & Rahman , 2013), Vector Auto Regression (VAR) (Sun, Wang, & X-R, 2008), Arbitrage of Forecasting Expert (AFE) (Cerqueira, Torgo, Pinto, & Soares, 2019).

There are also studies with novel lightweight data-driven weather forecasting models such as long short-term memory (LSTM) (Behera, Keidel, & Debnath, 2018) , (Hewage, Behera, Trovati, & Pereira, 2019) and Temporal convolutional networks (TCN) (Hewage P. , 2020)

The challenge is to determine (through experiments) the most suitable machine learning technique to use for weather forecasting. Figure 1 shows the overview of the process followed to compare the performance of novel proposed machine learning algorithms with existing classical forecasting models.

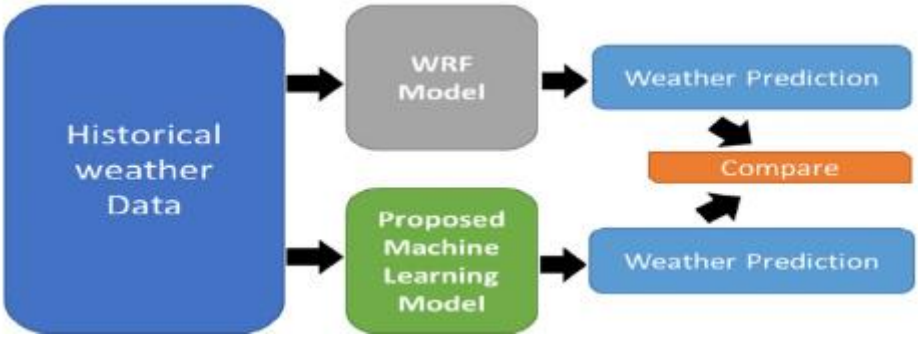

FIG. 1 RESEARCH OVERVIEW

A recent related work (Pradeep, Marcello, Ella, & Ardhendu,, 2021) demonstrates that the proposed lightweight deep model can be utilized for weather forecasting up to 12 hours for 10 surface weather parameters (TSK, PSFC, U10, V10, Q2, Rain, Snow, TSLB, and SMOIS). The model outperformed the state-of-the-art WRF model. Furthermore, the proposed model was able to overcome some challenges within the WRF model, such as the understanding of the model and its installation, as well as its execution and portability. This process was highly efficient compared to the WRF model.

[4] https://weather.us/model-charts

At figure 2, the related work (Schult, et al., 2021) suggest a workflow for end-to-end replacement of current classical methods with deep neural networks. This work suggest end-to-end replacement with deep neural networks.

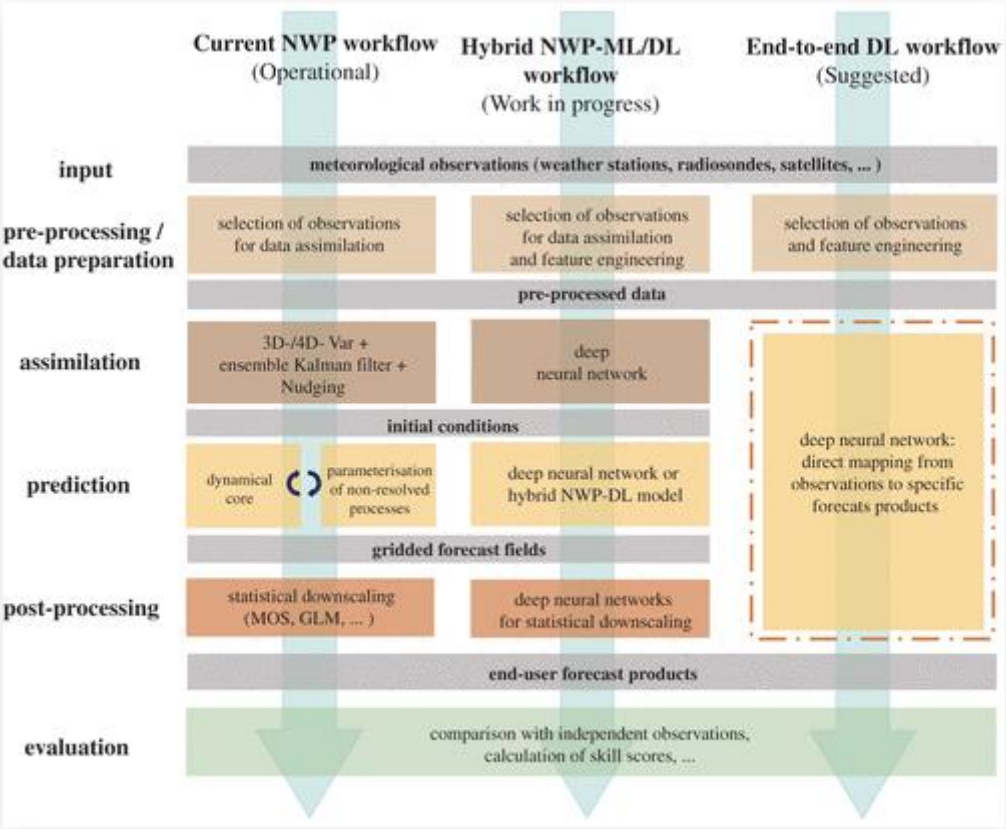

FIG. 2 PURELY DATA-DRIVEN DL WEATHER FORECASTING SYSTEM

Finally, the work, (Kreuzer & Munz, 2020) concludes that neural networks can be efficiently trained for shorter forecasting horizons deep using GPGPUs and the applicability of convLSTM networks is beneficial for temperature forecasting.

Chapter 3 – System Architecture

At this chapter, we thoroughly present the tools and technologies used at the present work as well as the proposed architecture.

3.1 End to End Analysis development

Our attempt to manage, store and visualize meteorological data needs to be deconstructed into corresponding programming tools where each one will fulfill its respective role. In particular, the need to manage and compare meteorological data from various sources initially requires a tool that will be able to communicate with these sources to extract the data we need. Next, there should be a second programming tool, which can store these data permanently. Finally, the stored data should be available and accessible from visualization applications so that end users can manage charts and visualizations.

The main pillars of the system architecture are i) the communication with the various sources for the extraction of meteorological data, ii) the data storage somewhere centralized and iii) the visualization tools about this data for end users use.

The challenges in such an undertaking are enough. To end up having aggregations visualizations we need to ensure that each variable of the weather data corresponds to common units of measurement. As typical example is that, some sources may offer the temperature in Celsius degrees, while others in Fahrenheit. This means, respectively, that apart from the basic tools for extracting, storing and visualizing data, we need additional tools that transform the variables of our data (temperature, wind direction and velocity, barometric pressure, humidity, precipitation) into a common system.

Therefore, it is somewhat clear now, that we need a **Core stack**, which will be responsible for communicating with the different meteorological sources and will transform the data into an object-oriented format – where the weather variables (temperature, humidity, etc.) consist the attributes of the object - as well as into a common system of measurement units.

Now our data are ready to be stored permanently in a structure -**Persistence stack**-, which can support a large amount of data. It is also necessary for the storage subsystem to be able to support the object-oriented nature of our data.

The third stack of our architecture is the V**iew stack.** This part is dedicated to end users. This layer should in turn utilize the attributes of the data and provide diagrams and visualizations so that the user can form conclusions about data aggregations.

At this point, we analyzed the basic parts, which consists our system architecture, namely the interaction between different data sources, the storage of the data, the data visualization while also other intermediate layers which has to normalize and transforms the data.

Below are presented, in more detail, the programming tools that were used for each part as well, as how they work together.

3.2 Core stack – Data management

As mentioned previously, the core system is responsible for communicate with the different sources of meteorological data, but also convert the data into a common reference. The specific stack consists of two subsystems, the part that gathers the data while the corresponding part that transforms and handles our data.

3.2.1 Gathering Data

The specific part should practically perform web calls periodically to the various meteorological sources. This task - technique is widely used in data processing and is called Web scraping. In our architecture, we choose "Scrapy[5]" (Scrapy Framework, n.d.) framework, which performs exactly, that our project needs. Scrapy is a key axis in the present work, as it is used to collect data from meteorological data sources from both sites and meteorological stations.

Scrapy (see its logo in Figure 3) is a free and open-source web-crawling framework written in Python. Originally designed for web scraping, it can also be used to extract data using APIs or as a general-purpose web crawler.

Scrapy project architecture is built around "spiders", which are self-contained crawlers that are given a set of instructions. Scrapy provides a web-crawling shell, which can be used by developers to test their assumptions on a site's behavior.

FIG. 3 SCRAPY FRAMEWORK LOGO

The following diagram at Figure 4 shows an overview of the Scrapy architecture with its components and an outline of the data flow that takes place inside the system (shown by the red arrows).

[5] https://scrapy.org/

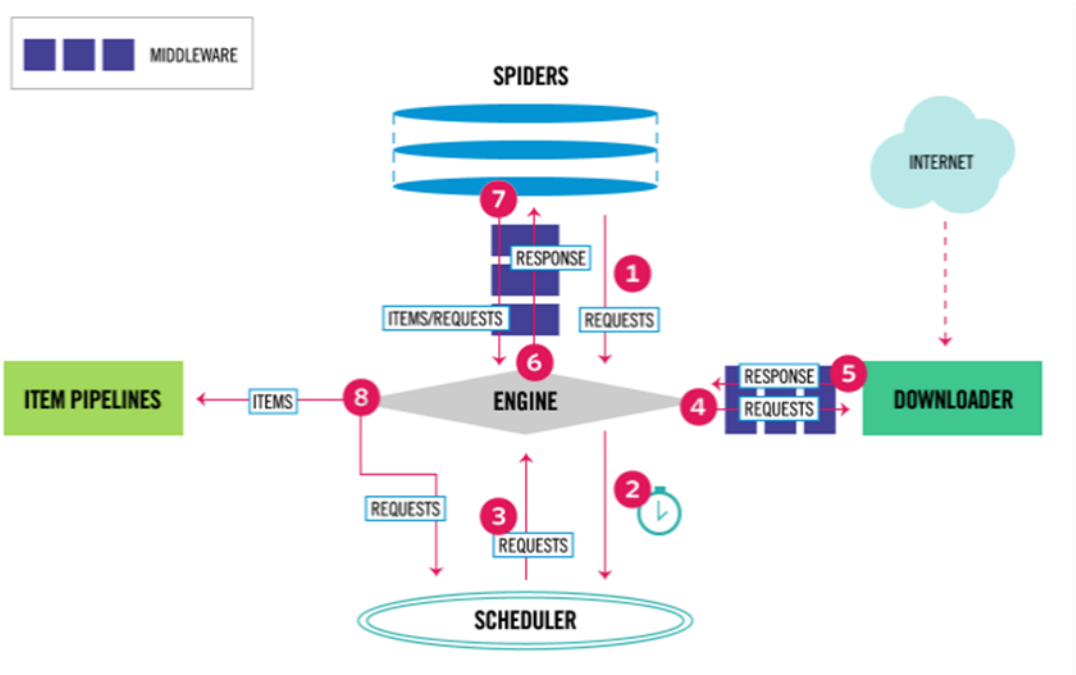

FIG. 4 – SCRAPY ARCHITECTURE

The data flow in Scrapy is controlled by the execution engine, and follows the next steps:

- The Engine gets the initial Requests to crawl from the Spider.
- The Engine schedules the Requests in the Scheduler and asks for the next Requests to crawl.
- The Scheduler returns the next Requests to the Engine.
- The Engine sends the Requests to the Downloader, passing through the Downloader Middlewares.
- Once the page finishes downloading, the Downloader generates a Response (with that page) and sends it to the Engine, passing through the Downloader Middlewares.
- The Engine receives the Response from the Downloader and sends it to the Spider for processing, passing through the Spider Middleware
- The Spider processes the Response and returns scraped items and new Requests (to follow) to the Engine, passing through the Spider Middleware.
- The Engine sends processed items to Item Pipelines, then send processed Requests to the Scheduler and asks for possible next Requests to crawl.

3.2.2 Weather Data Sources

We mentioned that the Scrapy spiders are responsible for gather our data from various metrological sources, periodically. We will examine here these sources and the reasons why they were chosen.

A basic criterion for choosing a source, is that this source should provide weather data for the area of Tripoli – Peloponnese – Greece, seat of the University of Peloponnese where the present work try to clarify if the data from different sources converge. Another key reason is that each selected source should own a weather station and all these weather stations are different from each other. Some sources also may provide data from third-party weather services. The reason is that we need to examine if the data from different stations of the same area provide relative same values on their data.

A key characteristic also among the selected sources is that they should provide common weather characteristics such as wind direction and velocity, temperature, barometric pressure, relative humidity and precipitation. We remind, that our purpose is to create aggregations visualization. For this reason, the data from different sources should have common characteristics.

Another challenge we have to face is that some sources may not provide data to services such as web scraping due to restrictions they place in their web site. Unfortunately, we cannot receive data from these sources.

Finally, among the sources where we can collect data, there are differences regarding the way of communication. Some sources have APIs where they provide data with a specific structure, while others do not, rendering data management more difficult.

At table 1 are shown summary information about the chosen weather sources.

	Weather Source	Wind direction and velocity	Temperature	barometric pressure	relative humidity	precipitation	Api	Own Weather Station
1	meteo.gr	✓	✓	✓	✓	✓		✓
2	hnms.gr	✓	✓	✓	✓	✓		✓
3	accuweather.com	✓	✓	✓	✓	✓	✓	✓
4	okairos.gr	✓	✓	✓	✓	✓		
5	freemeteo.gr	✓	✓	✓	✓	✓		✓
6	xalazi.gr	✓	✓	✓	✓	✓		
7	openweathermap.org	✓	✓		✓	✓	✓	✓

TABLE 1 – TABLE OF (STATION) DATA SOURCES

As you can see from table 1, the chosen data sources provide common weather variables such as wind direction and velocity, temperature, barometric pressure, relative humidity and precipitation, so that the data comparison process to be meaningful. From the other hand, some of the chosen sources have their own weather stations and others rely on third weather services. The stations of the sources have some distance between them, but they concern the area of Tripoli. At the Table 2 there are the weather stations coordinates of each source.

	Weather Station	longitude	latitude
1	meteo.gr	22.3723	37.5135
2	hnms.gr	22.4	37.52
3	accuweather.com	22.373	37.511
4	freemeteo.gr	22.38	37.51
5	openweathermap.org	22.3794	37.5089

TABLE 2 - WEATHER STATIONS COORDINATES

Below we describe the reasons for choosing each of our selected sources.

«meteo.gr» belongs to the National Observatory of Athens [6] (National Observatory of Athens, n.d.) and provides weather data and forecasts for about 500 areas cities and areas of Greece. For our interested area (Tripoli), meteo maintains a weather station of the center of the city. The data of the station renewed every 15 minutes. Meteo does not have an API for our integration, instead we use web crawlers to retrieve data from the page of the station [7] (Meteo Weather Station Page, n.d.). Meteo.gr provides the weather characteristics we need while also constitutes a stable weather source.

«hnms.gr» is the Hellenic national meteorological service[8]. The mission of the National Meteorological Service is to provide meteorological support for the benefit of the National Defense, the National Economy and the Social whole of our country (Hellenic National Meteorological Service, n.d.). The service maintain a weather station at the center of the city of Tripoli. The station provides the characteristics we need in our study and refresh the weather data every 30 minutes on their station page[9] .

«accuweather.com» [10] is a known worldwide weather service. The service collect data from third party stations around the world. There is no official information about the origin – position- of the stations, but when we collect data through the API of the service, there are information about the location of the station, as we have listed in Table 2. The service provide the characteristics we need in our work while allows data derivation every 1 hour.

[6] https://www.noa.gr/
[7] https://penteli.meteo.gr/stations/tripoli/
[8] http://www.emy.gr/
[9]http://www.emy.gr/emy/el/observation/sa_teleytaies_paratiriseis_stathmou?perifereia=Peloponnese&poli=Tripoli
[10] https://www.accuweather.com/

«okairos.gr»[11] is a weather service, which provide the data with the relative characteristics we need.The website presents the weather for more than 3,000 cities and airports in Greece. Forecasts are based on data from the most global sources (Okairos.gr - About Us, n.d.). It is more of a weather forecast site and does not provide information regarding the existence of a weather station. We derive data from this source because it provides an detailed hourly forecast (Okairos.gr - Hourly ForeCast, n.d.).

«freemeteo.gr» [12] is a Greek weather internet service. The service maintain its own weather station for the city of Tripoli (freemeteo.gr station information, n.d.) and provides the needed characteristics of our data while the data are refresh every 3 hours.

«xalazi.gr»[13] it is also another Greek weather internet service. The service does not have its own station and based on third party sources for historical data and forecasting. The service provide data for the city of Tripoli with the characteristic we need every 3 hours (Xalazi.gr - Weather Data, n.d.).

«openweathermap.org»[14] provides hyperlocal minutely forecast, historical data, current state, and from short-term to annual forecasted weather data. All data is available via industry standard APIs (OpenWeathermap.org - About Us, n.d.). The service maintain its own weather stations while allows to anyone to connect their weather station to the service. This source provide us the weather characteristics we need though API calls hourly.

3.2.3 Forming and Transforming Data

In the previous sections, we saw the weather sources we have chosen with the characteristics of each one, as well as the data collection process through Scrapy framework. We concluded that the object-oriented structure is more suitable to represent the data. We could think that each weather data is an object that will have specific characteristics. These characteristics are essentially the characteristics of the weather.

The item shown below constitutes a typical example of an (python) object representation that will be provided the Scrapy engine.

```
class MeteoItem(scrapy.Item):
    id           = scrapy.Field()
    source       = scrapy.Field()
    time         = scrapy.Field()
    timecrawl    = scrapy.Field()
    temperature  = scrapy.Field()
    humidity     = scrapy.Field()
    wind         = scrapy.Field()
    barometer    = scrapy.Field()
    yetos        = scrapy.Field()
    direction    = scrapy.Field()
    city         = scrapy.Field()
```

[11] https://www.okairos.gr/
[12] https://freemeteo.gr/
[13] http://www.xalazi.gr/
[14] https://openweathermap.org/

Essentially, we ended up with a class where every weather object belonging to this class is required to use these attributes. Our goal is to end up with data with common structure and measurement units, to be permanently stored later into persistence stack.

The ***id*** field is a unique identifier of each element. The Id field consist of the concatenation of the date stamp and the region (source site) of the weather station. The ***source*** field represents, the weather station's name, the ***time*** field is the time which the weather station refers to each sample, while the ***timecrawl*** fields refers to the actual time stamp when the data was crawled.

The **temperature** field is the degree of heat measured at the weather station at the time field. The temperature field uses the Celsius scale. Respectively the **humidity** field is the humidity percentage in the atmosphere measured at the time it is reported by the weather station and it is expressed as a percentage. The **wind** field is the actual wind intensity measured at the reported time (time field) by the weather station. The unit of measurement for the specific field is kilometers per hour (km/h).

The **barometer** field refers the atmospheric pressure recorded by the weather station and uses bars as unit of measure. The **yetos** field includes rain, snow, sleet, hail, and any other weather phenomenon that causes water (in any form) to fall from the atmosphere to the earth and it is measured in Millimeters (mm). Finally, the **direction** field refers to the direction of the wind calculated in degrees. The scale of degrees ranges between 0 and 360. Zero degrees points to North direction on the wind while the measurement is done clockwise. For example 90 degrees points to East direction, 180 degrees points to South direction and 270 degrees points to West direction.

The table 2 shows the fields of the actual weather data, along with details concerning each of the fields.

	Field	Measurement Unit	Min – Max values	Remarks
1	temperature	°C	Absolute Zero – N/A	
2	humidity	%	0-100	
3	Wind	km/h	0- N/A	
4	barometer	bars	0- N/A	
5	Yetos	mm	0- N/A	Precipitation in any form
6	direction	degrees	0-360	

TABLE 3 WEATHER OBJECT FIELDS

So far, we have seen a more abstract description of the object that describes the data from the weather stations, as well as the properties of each attribute. The Spider will use the above generic object, which is responsible for the respective weather station. A typical example of the code, which crawl the data from a site (weather station), is given below.

```
class MeteoSpiderSpider(scrapy.Spider):

    name = 'meteo_spider'
    allowed_domains = ['http://penteli.meteo.gr/stations/tripoli/']
    start_urls = ['http://penteli.meteo.gr/stations/tripoli/']

    def parse(self, response):
        i=0
        table = response.xpath('//*[@id="table1"]')
        rows = table.xpath('//tr')
        source       = 'meteo.gr'
        city         = 'Tripoli'
        crawldate    =  datetime.datetime.now()
        timestr         = rows[2].xpath('td//text()')[3].extract()
        datepart     = timestr[-9:].strip()
        timepart     = timestr[2:-9].strip()
        datetimep    = datepart+' '+timepart
        temperature  = float(rows[3].xpath('td//text()')[4].extract()[0:-2])
        humidity     = float(rows[4].xpath('td//text()')[4].extract()[:-1])
        windends     = (rows[6].xpath('td//text()')[4].extract()).find(" ")
        winddire     = (rows[6].xpath('td//text()')[4].extract()).find("at")
        wind         = float(rows[6].xpath('td//text()')[4].extract()[0:windends])
        barends      = rows[7].xpath('td//text()')[4].extract().find(" ")
        barometer    = float(rows[7].xpath('td//text()')[4].extract()[:barends])
        yetos        = float(rows[8].xpath('td//text()')[3].extract()[:-3])
        direction    = rows[6].xpath('td//text()')[4].extract()[winddire+3:]

        id            = source+' '+datetimep
        item = MeteoItem()
        item["id"]          = id
        item["source"]      = source
        item["time"]        = time
        item["timecrawl"]   = crawldate
        item["temperature"] = temperature
        item["humidity"]    = humidity
        item["wind"]        = wind
        item["barometer"]   = barometer
        item["yetos"]       = yetos
        item["direction"]   = direction
        item["city"]        = city
        yield item
```

The data is usually stored into HTML data tables and this is the reason why we see markings in the code like @id="table1", which refer to the actual html table, and markings like *rows,* which refer to the respective row of the table. The example above concerns the integration with «meteo.gr» weather source.

The example given below concern the Scrapy spider for hnms.gr weather source. You can notice that in the wind field there is a multiplication with the constant term 1.852. This particular weather source provide the wind velocity into knots. This is the reason why we transform the specific value into kilometers per hour, such as we describe in the previous section.

```
class HnmsSpiderSpider(scrapy.Spider):

  name = 'hnms_spider'
  def parse(self, response):
    i=0
    crawldate    =  datetime.datetime.now()
    table = response.xpath('//*[@class="table table-condensed table-
    striped table-hover small"]')
    rows = table.xpath('//tr')
    try:
       source      = 'Hnms.gr'
       city        = 'Tripoli'
       timestr     = rows[1].xpath('td//text()')[1].extract()
       time        = datetime.datetime(int(timestr[6:10]), int(timestr[3:5])
       temperature = float(rows[1].xpath('td//text()')[6].extract())
       humidity    = float(rows[1].xpath('td//text()')[13].extract()[:-1])
       wind        = float(((rows[1].xpath('td//text()')[27].extract())*1.852
       barometer   = 0
       yetos       = 0
       direction   = (rows[1].xpath('td//text()')[20].extract()
       id          = source +' '+timestr
       item = HnmsItem()
       item["id"]          = id
       item["source"]      = source
       item["time"]        = time
       item["timecrawl"]   = crawldate
       item["temperature"] = temperature
       item["humidity"]    = humidity
       item["wind"]        = wind
       item["barometer"]   = barometer
       item["yetos"]       = yetos
       item["direction"]   = direction
       item["city"]        = city
       yield item
```

We mentioned previously that the Scrapy framework is based on Python, as we also saw that the spiders for each weather station are implemented in Python. Python is an integral part of our core stack.

Python (Python Official Page, n.d.) (see in Figure 5 its logo) is an interpreted high-level general-purpose programming language. Its design philosophy emphasizes code readability with its use of significant indentation. Its language constructs as well as its object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

Python is dynamically-typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly, procedural), object-oriented and functional programming. It is often described as a "batteries included" language due to its comprehensive standard library.

Python consistently ranks as one of the most popular programming languages.

FIG. 5 PYTHON PROGRAMMING LANGUAGE LOGO

Rather than having all of its functionality built into its core, Python was designed to be highly extensible (with modules). This compact modularity has made it particularly popular as a means of adding programmable interfaces to existing applications

3.2 Persistence stack – Data Storage

In the Data Management section, we described the characteristics of the data we have to manage as well as the sources, which will provide the data. In this section, we will see the permanent storage of our data. The choice of the tool that will be responsible for storing our data is very important. We should take into account factors such as the transfer of our data from Scrapy and the scale up, i.e. the increase rate of our data volume.

The data storage framework should also take into consideration the object-oriented nature of our data while it should be relative easy for the view stack (or third services and users) connect to the data storage.

We initially rejected relational databases mostly for the reason of scale up. While relational databases are very stable tools, we need tools that work in distributed environments and are designed to respond to huge data volume.

For the above reasons, we end up that a suitable choice for our work is Elasticsearch. Elasticsearch (Figure 6 Shown its logo) is a search engine based on the Lucene library. It provides a distributed, multitenant-capable full-text search engine with an HTTP web interface and schema-free JSON documents. Elasticsearch is developed in Java and is dual-licensed under the source-available Server Side Public License and the Elastic license, while other parts fall under the proprietary (source-available) Elastic License. Official clients are available in Java, .NET (C#), PHP, Python, Apache Groovy, Ruby and many other languages. According to the DB-Engines ranking, Elasticsearch is the most popular enterprise search engine (DB-Engines Ranking of Search Engines, n.d.).

Elastic NV was founded in 2012 to provide commercial services and products around Elasticsearch and related software. In March 2015, the company Elasticsearch changed their name to Elastic.

FIG. 6 ELASTICSEARCH LOGO

Elasticsearch can be used to search all kinds of documents. It provides scalable search, has near real-time search, and supports multitenancy. "Elasticsearch is distributed, which means that indices can be divided into shards and each shard can have zero or more replicas. Each node hosts one or more shards, and acts as a coordinator to delegate operations to the correct shard(s). Rebalancing and routing are done automatically" (ElasticSearch official Page, n.d.). Related data is often stored in the same index, which consists of one or more primary shards, and zero or more replica shards. Once an index has been created, the number of primary shards cannot be changed.

Elasticsearch is developed alongside a data collection and log-parsing engine called Logstash, an analytics and visualization platform called Kibana, and Beats, a collection of lightweight data shippers. The four products are designed for use as an integrated solution, referred to as the "Elastic Stack" (formerly the "ELK stack").

Elasticsearch uses Lucene and tries to make all its features available through the JSON and Java API. It supports faceting and percolating, which can be useful for notifying if new documents match for registered queries. Another feature is called "gateway" and handles the long-term persistence of the index; for example, an index can be recovered from the gateway in the event of a server crash. Elasticsearch supports real-time GET requests, which makes it suitable as a NoSQL data store, but it lacks distributed transactions.

On 20 May 2019, Elastic made the core security features of the Elastic Stack available free of charge, including TLS for encrypted communications, file and native realm for creating and managing users, and role-based access control for controlling user access to cluster APIs and indexes. The corresponding source code is available under the “Elastic License”, a source-available license. In addition, Elasticsearch now offers SIEM and Machine Learning as part of its offered services.

We should also mention that the entire infrastructure of Elasticsearch is based on the java programming language.

Java (Figure 7 shown its logo) is a high-level, class-based, object-oriented programming language that is designed to have as few implementation dependencies as possible. It is a general-purpose programming language intended to let application developers write once, run anywhere meaning that compiled Java code can run on all platforms that support Java without the need for recompilation.

FIG. 7 JAVA PROGRAMMING LANGUAGE LOGO

Since we mentioned the tools for the persistence stack, it is interesting to see how our data transfer from Scrapy to Elasticsearch. The data transfer is achieved through Scrapy engine pipelines. A typical configuration of settings.py file concerning the pipelines is the following.

At the above part of the code, we declare the parameters of the Elasticsearch engine, such as the index name and the unique identifier of our data. The source code for the custom process, which is responsible for the data input into Elasticsearch, is open source, under the Apache License (Scrapy - ElasticSearch, n.d.).

After the configuration, the data successfully stored into Elasticsearch. Figure 8 shows data representation into Elasticsearch. At the upper part, we see the count (hits) of our data in the scale of time while at the lower part the see the details (weather attributes) of each record.

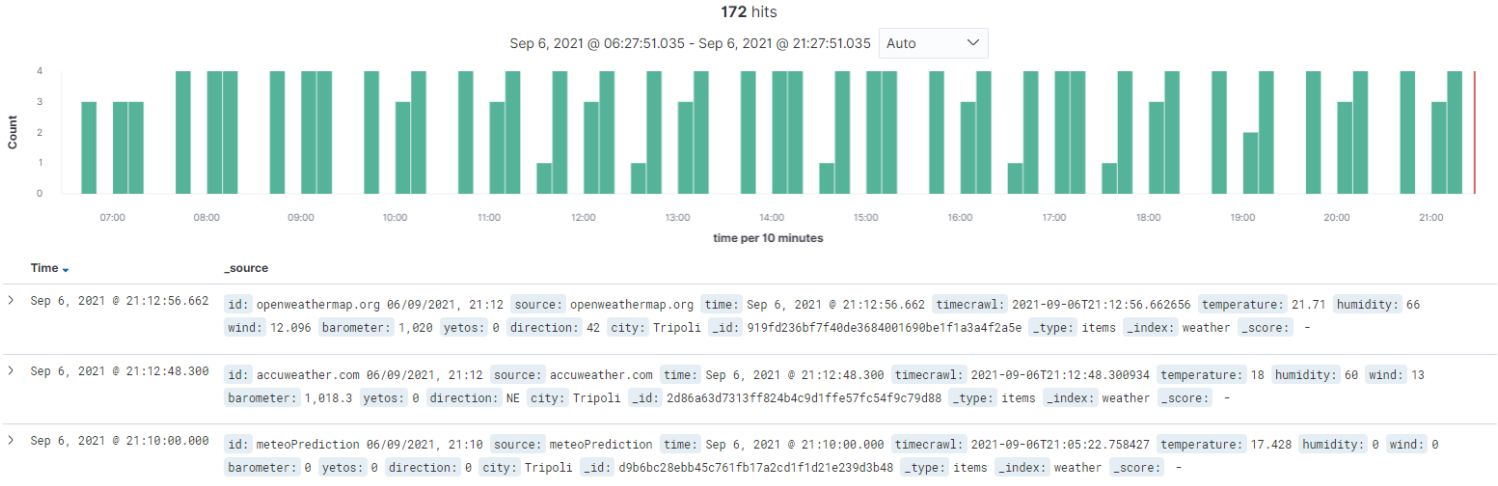

FIG. 8- DATA REPRESENTATION INTO ELASTICSEARCH

3.3 View stack – Data View

The final part of our architecture is the View stack. It is responsible for the visualizations and diagrams. Theses visualization through aggregations will examine if the data from the various weather data sources coverage or not. Elasticsearch has a built in analytics and visualization platform called Kibana (Kibana Official Site, n.d.).

Kibana (in Figure 10 you may find its logo) is a proprietary data visualization dashboard software for Elasticsearch, whose open source successor in OpenSearch is OpenSearch Dashboards.

It provides visualization capabilities on top of the content indexed on an Elasticsearch cluster. Users can create bar, line and scatter plots, or pie charts and maps on top of large volumes of data. Please find a Kibana visualization example in Figure 9. Kibana also provides a presentation tool, referred to as Canvas that allows users to create slide decks that pull live data directly from Elasticsearch.

The combination of Elasticsearch, Logstash, and Kibana, referred to as the "Elastic Stack" (formerly the "ELK stack"), is available as a product or service. Logstash provides an input stream to Elasticsearch for storage and search, and Kibana accesses the data for visualizations such as dashboards. Elastic also provides "Beats" packages which can be configured to provide pre-made Kibana visualizations and dashboards about various database and application technologies.

In May 2021, OpenSearch released the first beta of OpenSearch Dashboards, the Apache-licensed fork of Kibana sponsored by Amazon Web Services after Elastic discontinued the open source project and switched to proprietary software development.

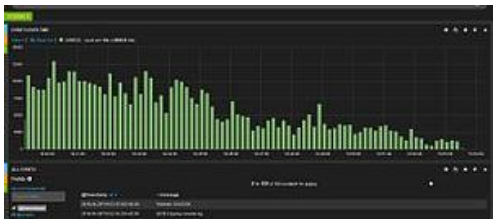

FIG. 9 KIBANA VISUALIZATION EXAMPLE

FIG. 10 KIBANA LOGO

3.4 Other Tools – Jupyter Notebook

During the implementation process of core stack, we used a very helpful tool called Jupyter notebook.

Project Jupyter is a project, whole community goal is to develop open-source software, open-standards, and services. Project Jupyter's name is a reference to the three core programming languages supported by Jupyter, which are Julia, Python and R, and also a homage to Galileo's notebooks recording the discovery of the moons of Jupiter. Project Jupyter has developed and supported the interactive computing products Jupyter Notebook (find its logo in Figure 11), JupyterHub, and JupyterLab.

Project Jupyter's operating philosophy is to support interactive data science and scientific computing across all programming languages via the development of open-source software. According to the Project Jupyter website, "Jupyter will always be 100% open-source software, free for all to use and released under the liberal terms of the modified BSD license

Jupyter Notebook (formerly IPython Notebooks) is a web-based interactive computational environment for creating notebook documents.

FIG. 11- JUPYTER NOTEBOOK LOGO

Closing Chapter 3, we described and analyzed the architecture of our system, the nature and characteristics of our data, the sources we choose to receive our data while the tools we used for each layer of our architecture. At Figure 12, you may find a visual representation of our system architecture.

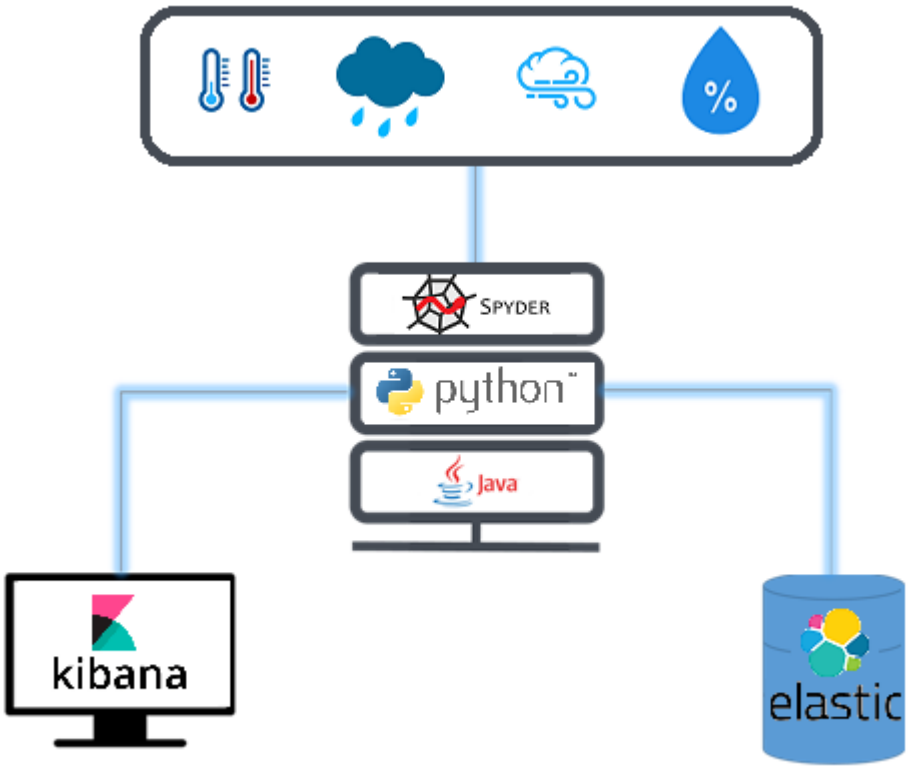

Fig. 12 – System architecture

Chapter 4 – Data Analysis and Visualization

4.1 Visualizations

At this section, we visualize the dataset in order to understand data distribution and to highlight the differences among the values given by different weather station. Figure 13 is capture the general data distribution for temperature with respect to time (define by the user), while at Figure 14 we can see in percentage of the temperature data obtained per weather station. Figure 15 and 16 visualize the maximum and minimum values respectively of the temperature per station and per year.

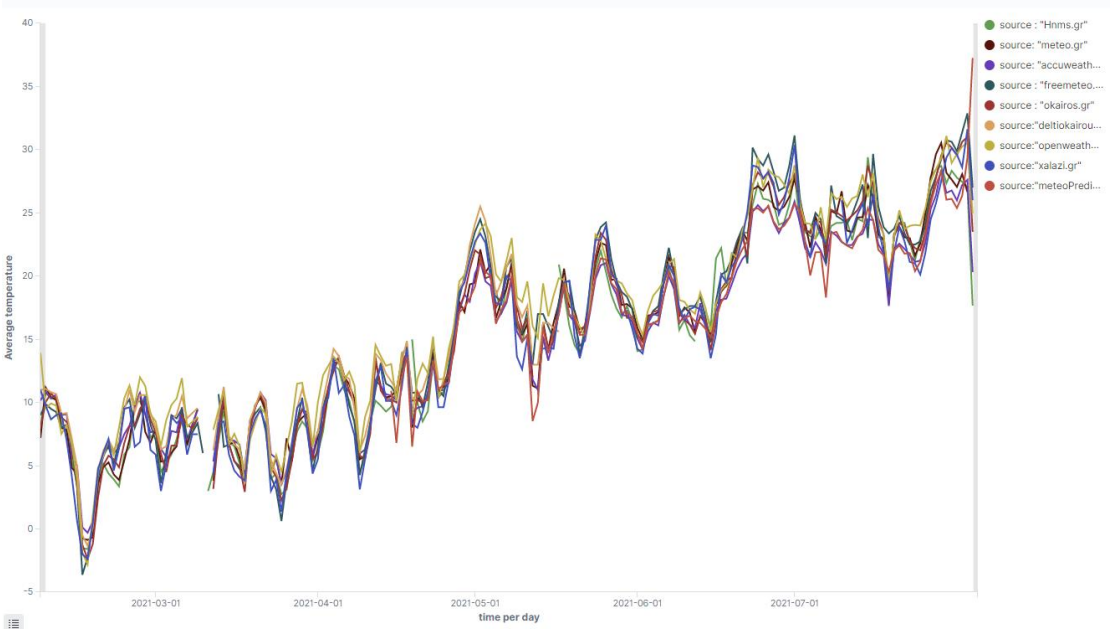

FIG. 13 - DATA AGGREGATIONS

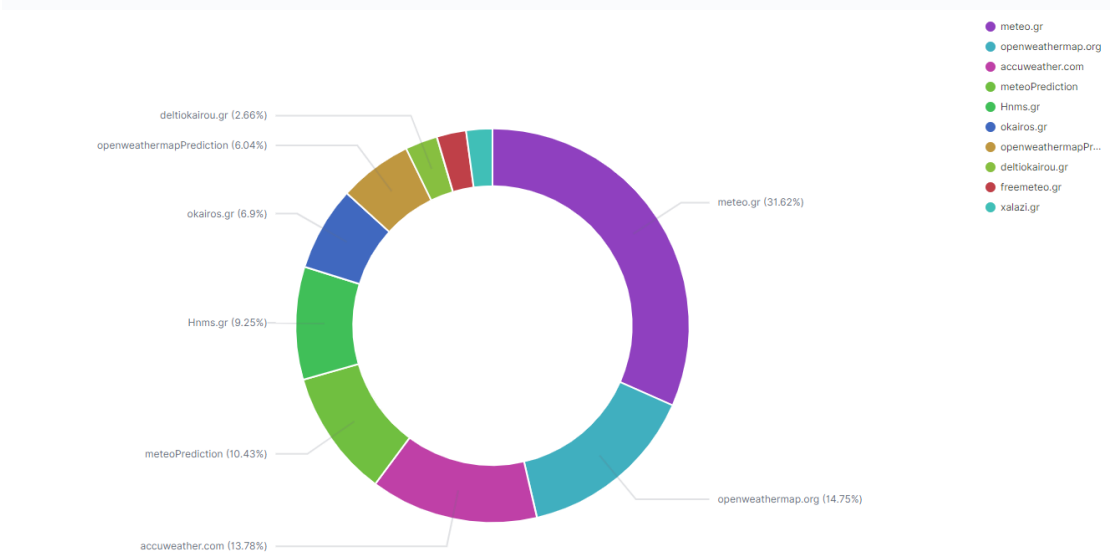

FIG. 14 - WEATHER DATA PER STATION DISTRIBUTION

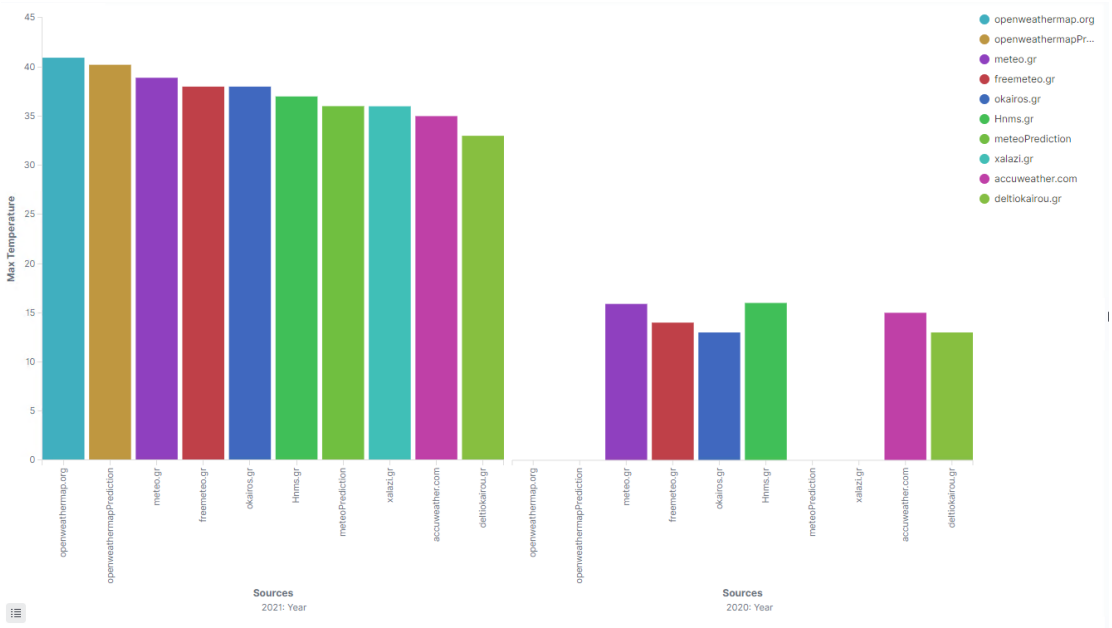

FIG. 15 MAXIMUM TEMPERATURE PER STATION AND YEAR

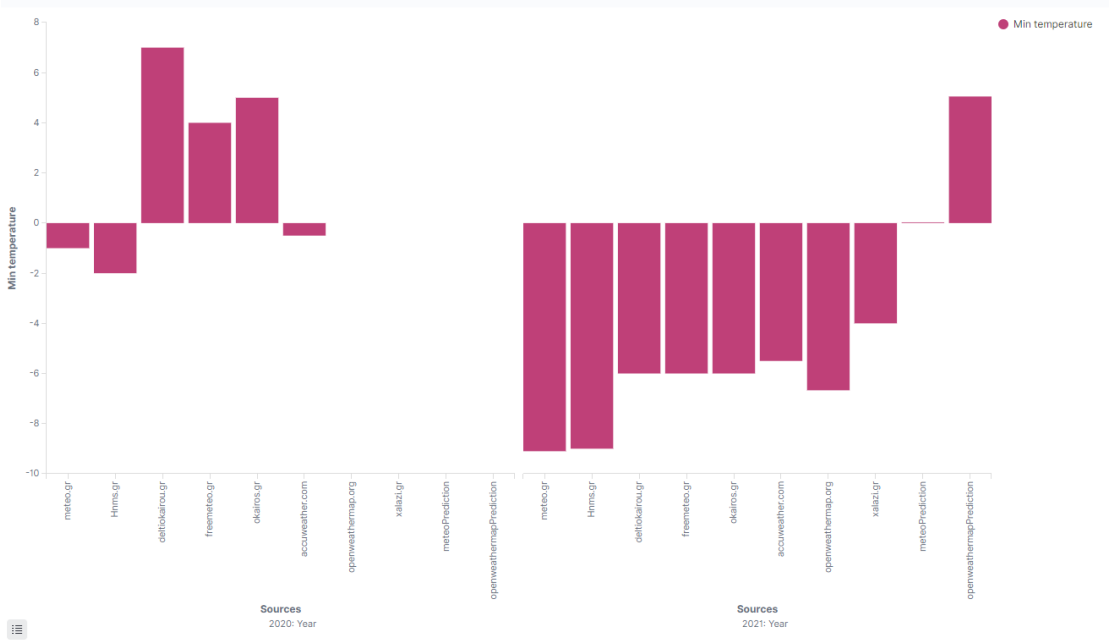

FIG. 16 – MINIMUM TEMPERATURE PER STATION AND YEAR

We could say that Figureσ 15,16,17,18 capture the actual data so that the user can form some initial conclusions, for the distribution of temperature data, the percentage of participation of each meteorological station, as well as for the maximum and minimum temperature values for the region we are studying, in the corresponding date range selected by the user.

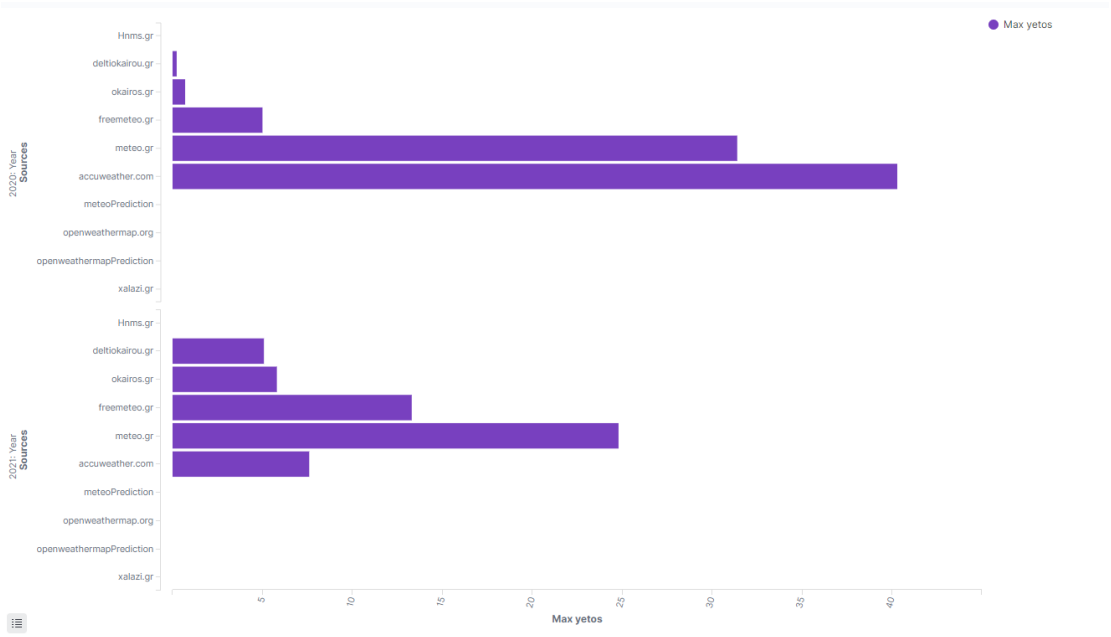

FIG. 17 MAXIMUM PRECIPITATION PER STATION AND YEAR

In Figure 17 the maximum value of the precipitation (any form) per station per year is given, while Figure 18 capture the minimum values observed for humidity per station and year.

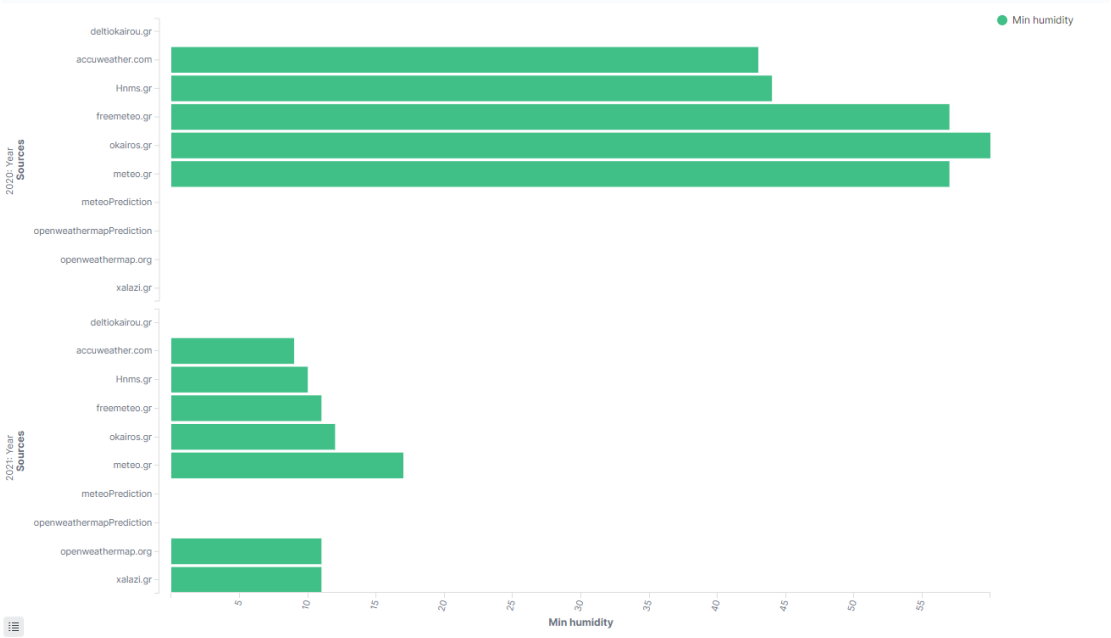

FIG. 18 – MINIMUM HUMIDITY PER STATION AND YEAR

Figure 17 shows the maximum value for precipitation since the minimum value is zero, while correspondingly, Figure 18 shows the minimum value for humidity since the maximum value is 100%.

Figures 19, 20, 21 and 22 visualize the actual (mouse interactive) data distributions for temperature, humidity, wind (velocity) and precipitation respectively.

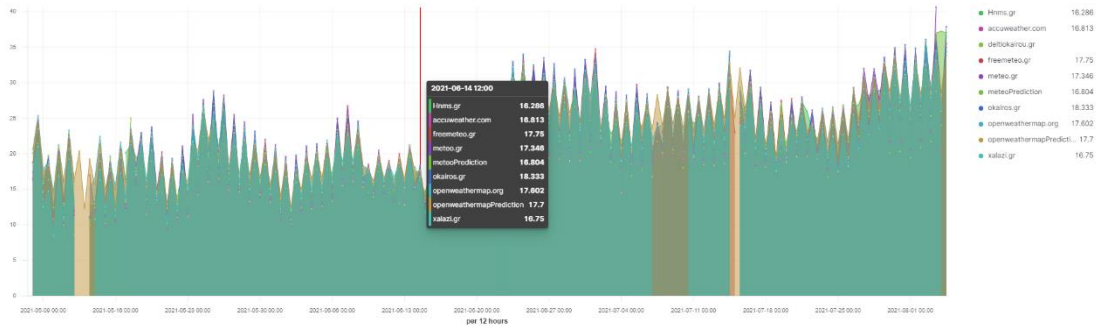

FIG. 19 – TEMPERATURE DISTRIBUTION

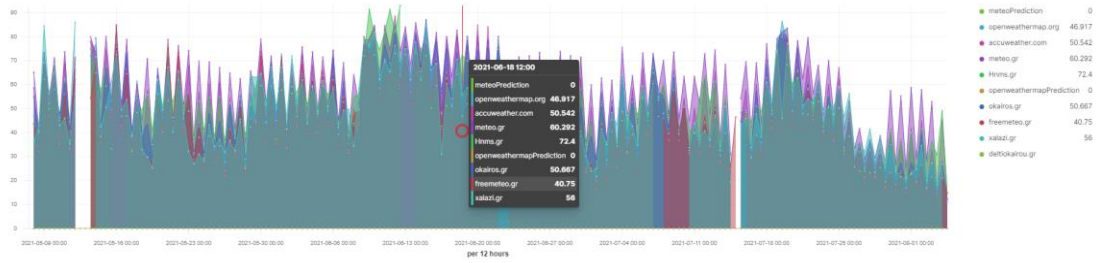

FIG. 20 – HUMIDITY DISTRIBUTION

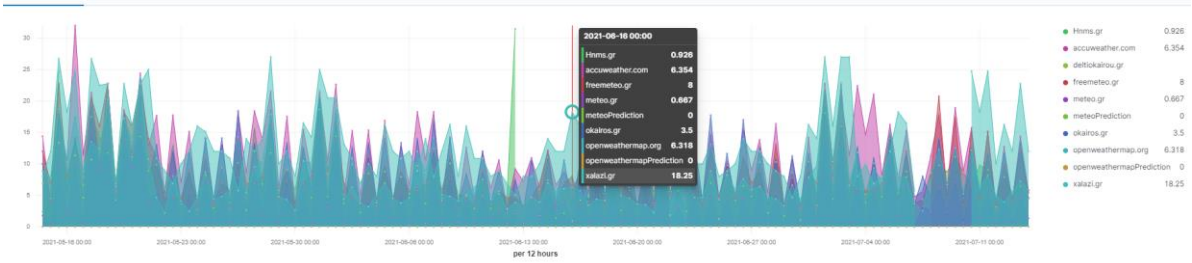

FIG. 21 – WIND VELOCITY DISTRIBUTION

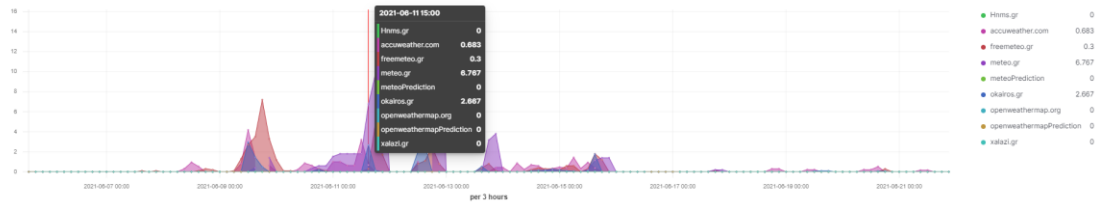

FIG. 22 – PRECIPITATION DISTRIBUTION

The specific Figures are quite important as they achieve the purpose of aggregation. Here the user can compare the data reported by the meteorological sources, for each weather variable (temperature, humidity, wind velocity and precipitation) for a selected date range.

Figures 23 and 24 show the next hour and the next six hours temperature prediction respectively.

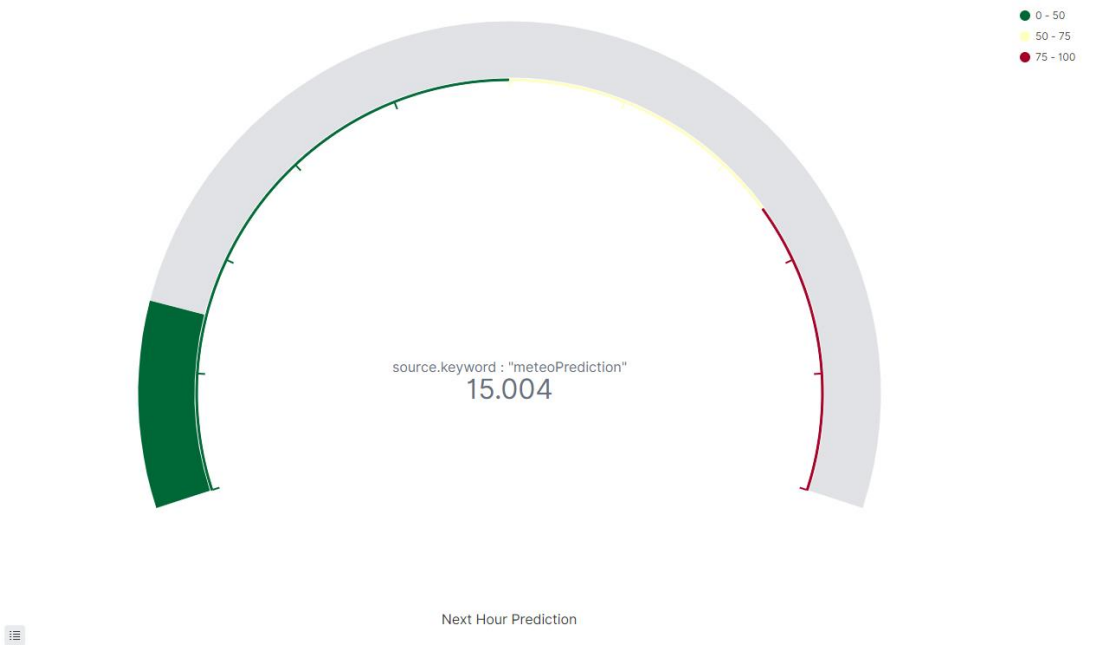

FIG. 23 – NEXT HOUR TEMPERATURE PREDICTION

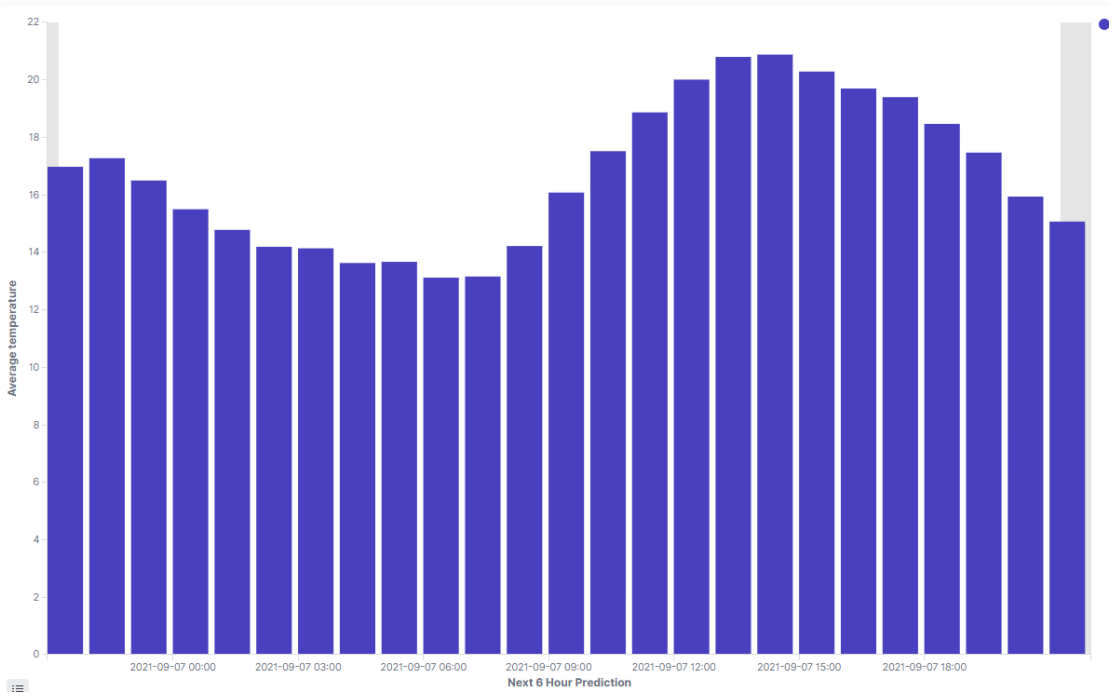

FIG. 24 – NEXT SIX HOURS TEMPERATURE PREDICTION

A qualitative way of visualizing the similarity coefficient is the heatmap visualization. We took advantage of Kibana *vega* visualization, an open source example that was used in our work to extract the following visualizations.[15]

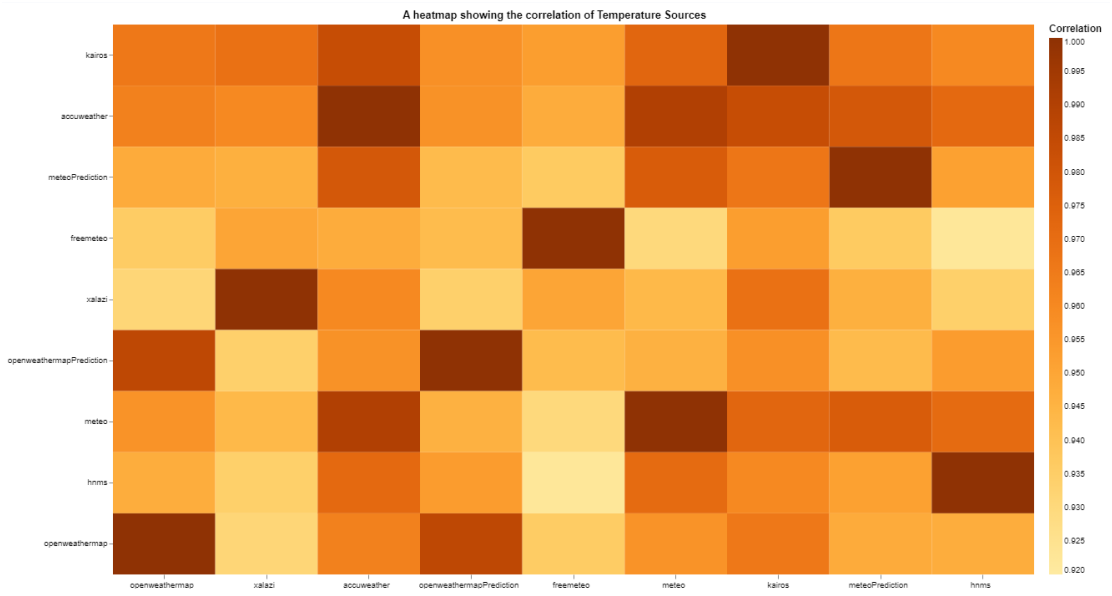

FIG. 25 – TEMPERATURE HEATMAP CORRELATION

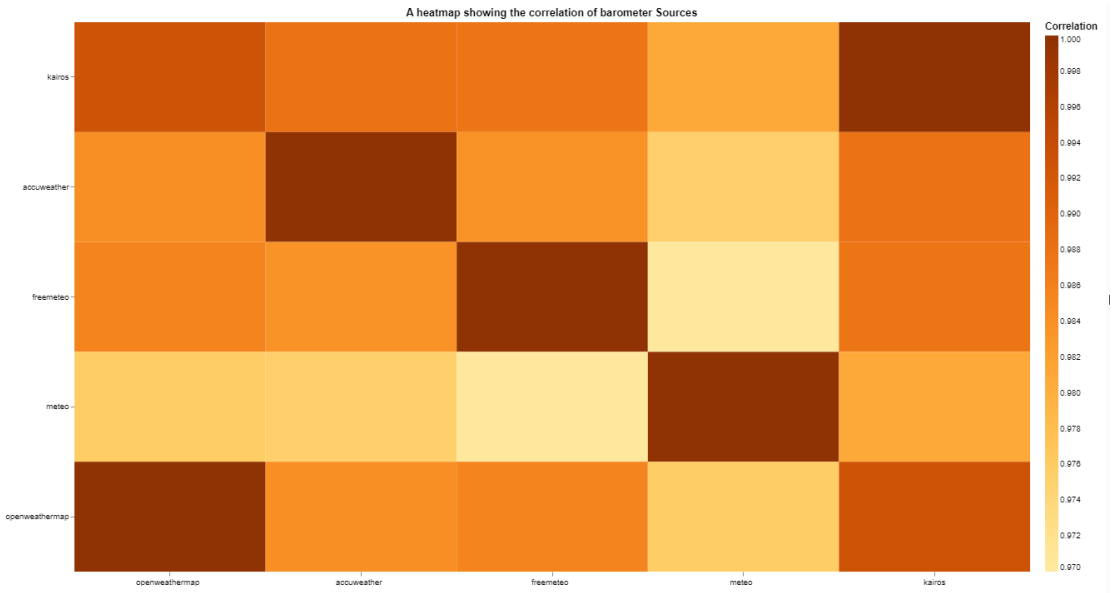

FIG. 26 – BAROMETER HEATMAP CORRELATION

[15] https://discuss.elastic.co/t/vega-heatmap-chart-for-matrix-stats-correlation/226099

Figures 25, 26, 27 and 28 show the correlation matrix through heatmaps for the temperature, humidity, precipitation and wind velocity respectively between the different weather stations.

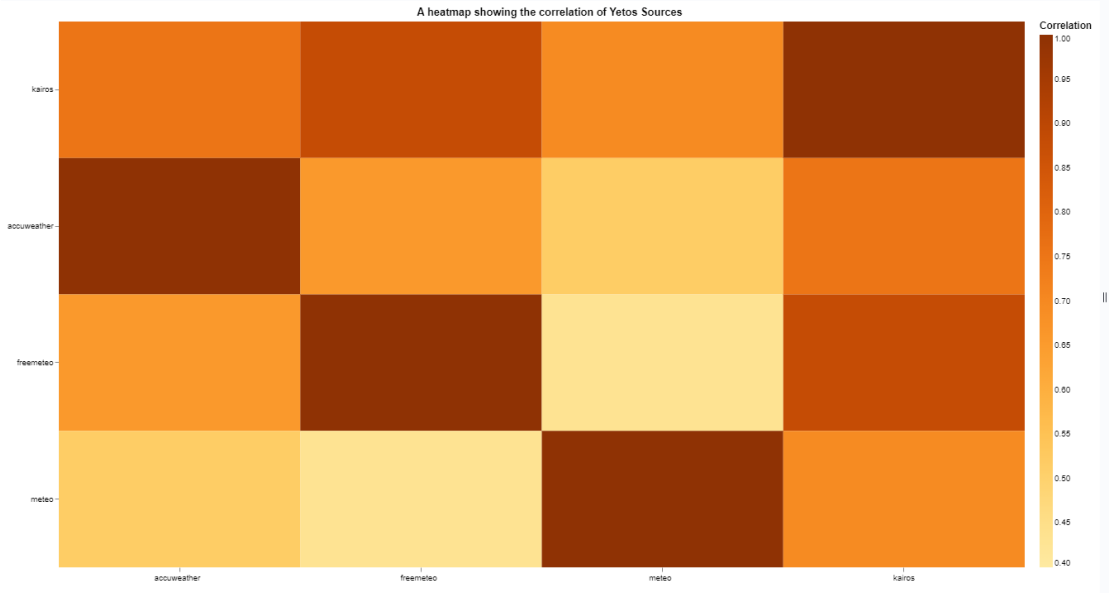

FIG. 27 – PRECIPITATION HEATMAP CORRELATION

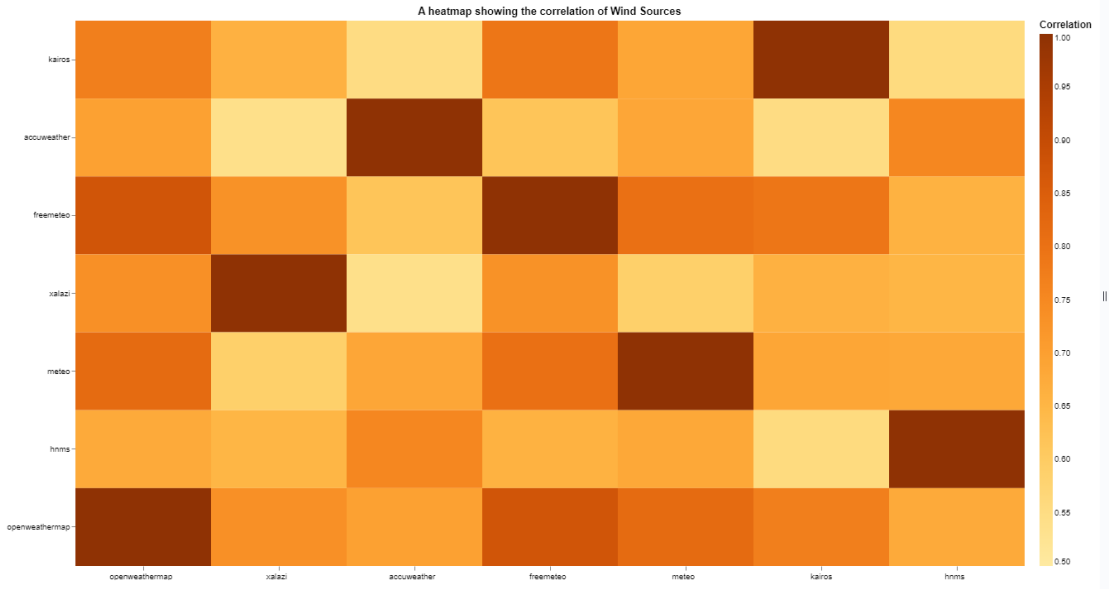

FIG. 28 – WIND (VELOCITY) HEATMAP CORRELATION

In a correlation heatmap, values can range from -1 to 1, and they represent the strength of the linear relationship between two distributions (Moore, McCabe, & Craig, 2016). Correlation closer to 1 represent a strong positive correlation while values close to 0 represent no correlation and values close to -1 represent a strong negative correlation. Heatmap visualizations are Figures where the user can qualitatively compare the way, weather variables converge or not, across our weather stations. User can be informed which meteorological sources provide similar or not weather data distributions.

Temperature Correlation Values	kairos	accuweather	freemeteo	xalazi	meteo	hnms	openweathermap
kairos	1	0.98	0.98	0.99	0.98	0.98	0.98
accuweather	0.98	1	0.97	0.97	0.98	0.97	0.97
freemeteo	0.98	0.97	1	0.98	0.97	0.98	0.98
xalazi	0.99	0.97	0.98	1	0.97	0.97	0.98
meteo	0.98	0.98	0.97	0.97	1	0.98	0.97
hnms	0.98	0.98	0.97	0.97	0.98	1	0.96
openweathermap	0.984	0.977	0.98	0.98	0.971	0.968	1

TABLE 4 - CORRELATION VALUES - TEMPERATURE

Wind Velocity Correlation Values	kairos	accuweather	freemeteo	xalazi	meteo	hmns	openweathermap
kairos	1	0.72	0.87	0.83	0.81	0.78	0.86
accuweather	0.72	1	0.69	0.7	0.68	0.77	0.71
freemeteo	0.87	0.69	1	0.79	0.81	0.73	0.91
xalazi	0.89	0.7	0.79	1	0.68	0.78	0.81
meteo	0.81	0.68	0.81	0.68	1	0.78	0.8
hnms	0.78	0.77	0.73	0.78	0.78	1	0.77
openweathermap	0.86	0.71	0.91	0.81	0.80	0.77	1

TABLE 5 - CORRELATION VALUES - WIND VELOCITY

Precipitation Correlation Values	accuweather	freemeteo	meteo	kairos
kairos	0.7	0.75	61	1
accuweather	1	0.96	0.58	0.7
freemeteo	0.63	1	0.74	0.75
meteo	0.58	0.74	1	0.61

TABLE 6 - PRECIPITATION CORRELATION VALUES

Humidity Correlation Values	kairos	accuweather	freemeteo	xalazi	meteo	hmns	openweathermap
kairos	1	0.91	0.91	0.9	0.9	0.84	0.93
accuweather	0.91	1	0.87	0.84	0.94	0.91	0.89
freemeteo	0.91	0.87	1	0.9	0.87	0.83	0.96
xalazi	0.9	0.84	0.9	1	0.86	0.81	0.9
meteo	0.9	0.94	0.87	0.86	1	0.94	0.89
hnms	0.84	0.91	0.83	0.81	0.94	1	0.84
openweathermap	0.89	0.89	0.96	0.9	0.89	0.84	1

TABLE 7 - HUMIDITY CORRELATION VALUES

Barometer Correlation Values	kairos	accuweather	freemeteo	meteo	openweathermap
kairos	1	0.99	0.99	0.8	0.99
accuweather	0.99	1	0.98	0.8	0.99
freemeteo	0.99	0.98	1	80	0.99
meteo	0.8	0.8	0.8	1	0.8
openweathermap	0.99	0.99	0.99	0.8	1

TABLE 8 - BAROMETER CORRELATION VALUES

Tables 4, 5, 6, 7 and 8 show the actual correlation values between different sources for each weather variable for all historical data. We can see that the temperature and barometer variables present a strong correlation coefficient while wind velocity, precipitation and humidity variables present less strong correlation coefficient.

4.2 Dashboards

Below, we present dashboards as a way to provide a bigger picture of the used dataset. Figure 29 shows the data distribution in terms of temperature, wind, humidity and precipitation.

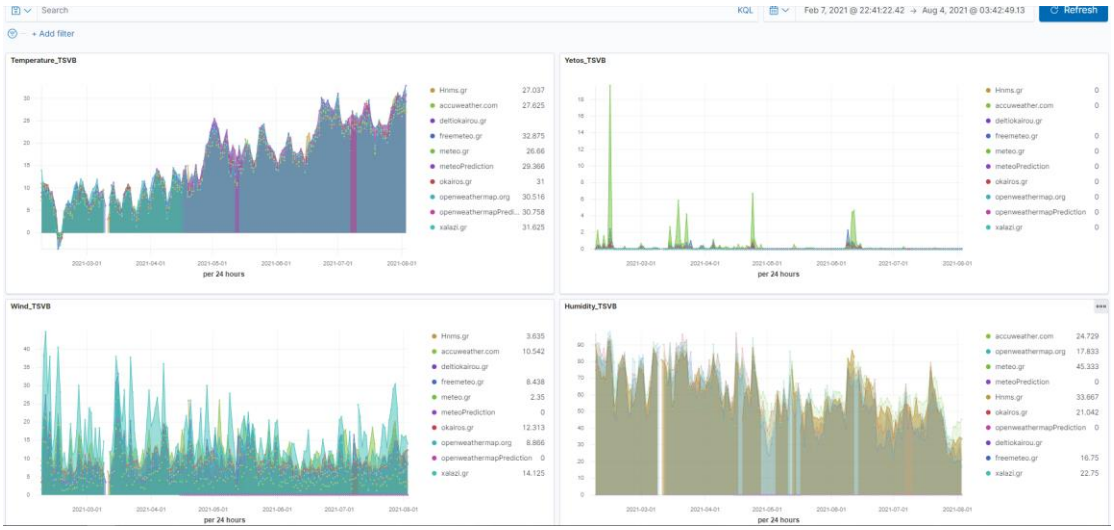

FIG. 29 - DATA DISTRIBUTION DASHBOARD

Figure 30 presents the data correlation among the different weather stations in terms of temperature, wind, humidity and precipitation. Our Dashboards practically unify and group multiple visualizations on a single page, making visualizations more user-friendly.

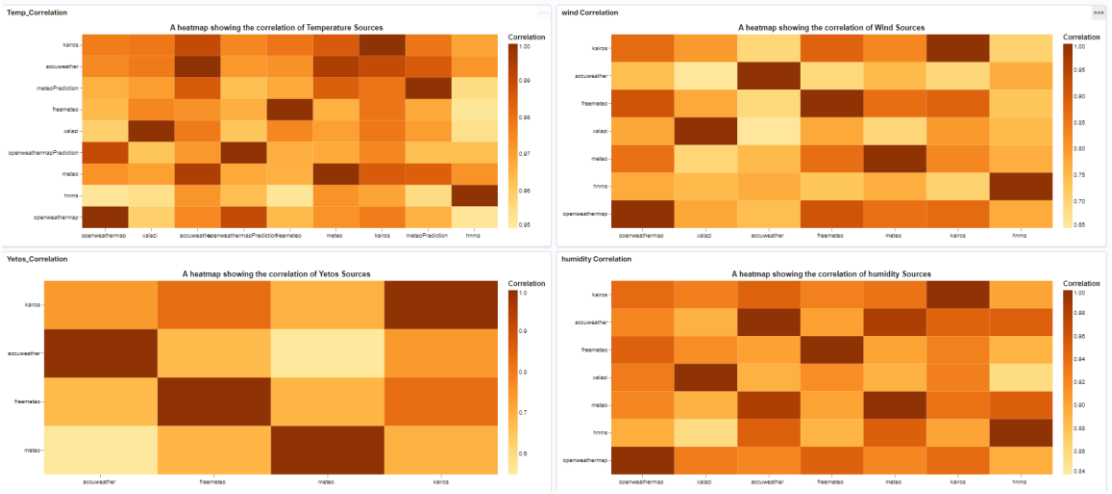

FIG. 30 – CORRELATION DASHBOARD

Chapter 5 – Weather Data Forecasting

In this chapter, we discuss the analysis performed on model training process and the parameters optimization. Our aim is to create a short term **temperature** forecast model using the data we have already collect. The model we choose for this task is the Long short-term memory – LSTM artificial neural network. The reasons why we choose the LSTM for this task are plenty. LSTM is consider one of the state of the art weather forecast model. It is a specialized form of Recurrent Neural Network (RNN) and it is widely applied to handle temporal data. The key concepts of the LSTM include layers of nodes that allows the passing of data through a multistep process to enable the recognition of the right pattern (Hinton G. E., 2006). RNNs and LSTMs in particular vary from other neural networks in that they have a temporal dimension and take time and sequence into account. In fact, they are considered to be the most effective and well-known subgroup of RNNs. they belong to a class of artificial neural networks made to identify patterns in data sequences, including numerical times series data. Long Short Term Memory (LSTM) networks are frequently used in sequence prediction problems. These recurrent neural networks have the capacity to learn sequence dependency. The output from the prior step is utilized as the upcoming step's input in the RNN. It was Hochreiter and Schmidhuber (Hochreiter & Schmidhu, 1997) who originally created the Long-Short Term Memory architecture. It addressed the problem of "long-term reliance" in RNNs. This is where RNNs can predict variables based on information in the current data, but cannot predict variables held in long-term memory. By design, LSTMs are known to store data for a long time. This method is employed when analyzing time-series data, making predictions, and categorizing data.

The dataset consist of historical data coming from region of Peloponnese (Tripoli) between the years 2014 and 2019. This dataset is made available by Dr. Lagouvardos Kostas, research director of National Observatory of Athens. [16] The data frame, shown at Figure 31, has 87604 rows of data. Each row declares the actual temperature at the respective time column. Each measurement differs from the next one by 30 minutes.

	temperature	time
0	1.8	2014-12-31 22:10:00
1	1.8	2014-12-31 22:40:00
2	1.8	2014-12-31 23:10:00
3	1.9	2014-12-31 23:40:00
4	2.0	2015-01-01 00:10:00
...	...	...
87599	3.6	2019-12-30 21:40:00
87600	3.4	2019-12-30 22:10:00
87601	3.8	2019-12-30 22:40:00
87602	3.1	2019-12-30 23:10:00
87603	3.0	2019-12-30 23:40:00

Fig. 31 – Dataset Data Frame

[16] https://www.iersd.noa.gr/en/staff/researchers/dr-lagouvardos-kostas/

Before starting the training process of our model, we should first investigate its hyper-parameters. Hyperparameters are parameters whose values control the learning process and determine the values of model parameters that a learning algorithm ends up learning. The prefix 'hyper_' suggests that they are 'top-level' parameters that control the learning process and the model parameters that result from it.

Train-test split ratio
We have choose to split our dataset at 80% for train data and 20% for test data. This train-test split ratio is a typical choice. Bigger percentage for the train dataset may cause overfitting where the learning model gives accurate predictions for training data but not for new data. From the other hand, small percentage of train dataset may cause underfitting, where the model has not learned the patterns in the training data well and is unable to generalize well on the new data.

Hidden Layers of the Model
The layers between the input and output layers are called hidden layers. There is no final number on how many nodes (hidden neurons) or hidden layers one should use. Hidden layers are the ones that are actually responsible for the excellent performance and complexity of neural networks. They perform multiple functions at the same time such as data transformation, automatic feature creation, etc.

Optimizer
An optimizer refers to an algorithm or method used to minimize or optimize the loss function during the training of the network. Optimizers play a crucial role in the training process by adjusting the model parameters in a way that reduces the value of the loss function. They use various optimization techniques to find the minimum of the loss function, which corresponds to a well-performing set of model parameters. The choice of optimizer can affect the training speed, convergence, and final performance of a neural network.

Common optimizers used in neural networks include:

1. **Stochastic Gradient Descent (SGD)**: A variation of gradient descent that uses a randomly selected subset of the training data (mini-batch) to compute the gradient, making it computationally more efficient (Robbins, 1951).

2. **Adam (Adaptive Moment Estimation)**: An adaptive learning rate optimization algorithm that combines ideas from RMSprop and Momentum. It adjusts the learning rates for each parameter based on their historical gradients (Diederik P. Kingma, 2015).

3. **RMSprop (Root Mean Square Propagation)**: An optimizer that adapts the learning rates of each parameter by dividing the learning rate by the moving average of the squared gradients (Hinton G. , 2012).

4. **Adagrad (Adaptive Gradient Algorithm)**: An optimizer that adapts the learning rates for each parameter based on the historical gradients, giving larger updates to infrequently occurring parameters (Duchi, Hazan, & Singer, 2010).

Units

Units are also commonly known as neurons or nodes. Each unit receives one or more inputs, performs a computation based on those inputs (often involving weights and biases), and produces an output. The number of units in a layer is a design choice and is determined by the architecture of the neural network. The number of units in the input layer is usually equal to the number of features in the input data. Units in a neural network are the individual computational entities within each layer, and the configuration of these units across layers defines the architecture of the neural network. Many units may cause overfitting to our model as well as the need for computational power is increasing. Small number of units may cause overfitting while the model is unable to learn and generalize the core function, which governs the data.

Batch Size

This hyperparameter defines the number of samples to work on before the internal parameters of the model are updated. Large sizes make large gradient steps compared to smaller ones for the same number of samples "seen". Batch size refers to the number of training examples utilized in one iteration. During each iteration, the neural network processes a batch of training samples and updates the model's weights. The training dataset is divided into batches, and the optimization algorithm updates the model based on the loss calculated for each batch.

Activation function(s)

An activation function is a mathematical operation applied to the output of a neuron (or node) in a neural network. It introduces non-linearity to the network, allowing it to learn complex patterns in the data. Without activation functions, the neural network would essentially be a linear model, incapable of learning from non-linear relationships in the data. The choice of activation function can affect the model's capacity to learn, training speed, and the ability to generalize to new data. Some commonly used activation functions in neural networks:

1. **Sigmoid Function (Logistic Function):** The sigmoid function, also known as the logistic function, is a mathematical function that maps any real-valued number to a value between 0 and 1. It is commonly used in machine learning and statistics, particularly in binary classification problems, where the goal is to assign an input to one of two possible categories. The sigmoid function has an S-shaped curve, which makes it suitable for tasks where the output needs to be in the range of 0 to 1. This function is particularly used in logistic regression and artificial neural networks (Nair & Hinton, 2010).

2. **Rectified Linear Unit (ReLU):** ReLU replaces all negative values with zero and is widely used in hidden layers. It helps with computational efficiency and has been found to work well in practice (Hastie, Tibshirani, & J, 2009).

3. **Hard sigmoid:** The hard sigmoid function is a modified version of the sigmoid function that provides a faster and computationally less expensive approximation while maintaining a piecewise-linear shape (Aggarwal).

4. **Swish:** Swish over other activation functions like ReLU tends to preserve more information in the network, potentially leading to improved performance. Swish has been observed to perform well in deep neural networks, and its smoothness allows for better optimization during training (Ramachandran, Zoph, & V., 2018).

Learning rate.
The learning rate is a hyperparameter that determines the size of the steps taken during the optimization process of training a neural network. It influences how much the model's parameters are adjusted with respect to the gradient of the loss function. In other words, the learning rate controls the step size in the optimization algorithm as it searches for the minimum of the loss function. If the learning rate is too high, the optimization algorithm might overshoot the minimum of the loss function, causing it to oscillate or diverge. Large steps may prevent the algorithm from converging to the optimal solution. If the learning rate is too low, the optimization process may take a very long time to converge or may be stuck in a local minimum.

Small steps may lead to slow convergence, especially in deep or complex neural networks. The learning rate is a critical hyperparameter, and choosing an appropriate value is crucial for successful training. It is common practice to start with a moderate learning rate and adjust it based on the observed training behavior.

DROPOUT
Every LSTM layer should be accompanied by a dropout layer. Such a layer helps avoid overfitting in training by bypassing randomly selected neurons, thereby reducing the sensitivity to specific weights of the individual neurons. While dropout layers can be used with input layers, they should not be used with output layers as that may mess up the output from the model and the calculation of error. While adding more complexity may risk overfitting (by increasing nodes in dense layers or adding more number of dense layers and have poor validation accuracy), this can be addressed by adding dropout. The 20% value is widely accepted as the best compromise between preventing model overfitting and retaining model accuracy.

The method for the appropriate hyperparameters selection of our model has been carried out with Grid Search method. Grid search is a hyperparameter tuning technique used in machine learning to find the optimal hyperparameter values for a model. In grid search, we can define a set of hyperparameter values for try, and the algorithm trains and evaluates the model for all possible combinations of these hyperparameter values. The combination that yields the best performance metric (such as accuracy, precision, or recall) is then selected as the optimal set of hyperparameters.

The parameter that guides us to choose the suitable hyper-parameter is the metric mean squared error. Mean Squared Error (MSE) is a measure of the average squared difference between predicted and actual values. It is commonly used in statistics and machine learning to assess the performance of a predictive model. MSE provides a way to quantify the accuracy of predictions and is particularly useful in regression analysis. The smaller the MSE, the better the model's predictions align with the actual data. In other words, a lower MSE indicates a better fit of the model to the observed data. MSE is widely used as a loss function during the training of regression models, and minimizing it is a common objective in model optimization.

To the next subsections, there is available the fine-tuning process for each hyperparameter of our model.

5.1 - Hidden Layer

Firstly, we had to decide about the interval structure of the model concerning the hidden layers. We therefore tested three different models, using one, two and three layers respectively, in order to choose the structure resulting in the best performance. Figure 32 shows the code we used to come up with the best choice. At the end of Figure 32, you may find the result of the training process. You may observed that the lower value of MSE has the model with three hidden layers. This is why we choose the three hidden layer option for our model.

```
x_train = []
y_train = []
train_split= 0.8
split_idx = int(len(temperature_df) * 0.8)
training_set = temperature_df[:split_idx].values
test_set = temperature_df[split_idx:].values
n_future = 12 #Next  observations temperature forecast
n_past = 64 #Past  observations

sc = MinMaxScaler(feature_range = (0, 1))
#training_set = sc.fit_transform(training_set)
#test_set = sc.fit_transform(test_set)

for i in range(0, len(training_set) - n_past - n_future + 1):
    x_train.append(training_set[i : i + n_past, 0])
    y_train.append(training_set[i + n_past : i + n_past + n_future, 0])

x_train , y_train = np.array(x_train), np.array(y_train)
x_train = np.reshape(x_train, (x_train.shape[0] , x_train.shape[1], 1))
```

```
loss = []
historytbl = []
UNITS = n_past
EPOCHS =10
BATCH_SIZE= 32
optimizer ='Adam'

regressor1 = Sequential()
regressor1.add(Bidirectional(LSTM(units=UNITS,return_sequences=True,  input_shape = (x_train.shape[1],1) ) ))
regressor1.add(Dropout(0.2))
regressor1.add((LSTM(units= UNITS )))
regressor1.add(Dropout(0.2))
regressor1.add(Dense(units = n_future,activation='linear'))
regressor1.compile(optimizer=optimizer, loss='mean_squared_error',metrics=['mse'])
regressor1.fit(x_train, y_train, epochs=EPOCHS,batch_size=BATCH_SIZE )
loss_metrics = regressor1.evaluate(x_train, y_train, verbose=0)
loss.append(['1 hidden Layer',loss_metrics[0],loss_metrics[1]])

regressor2 = Sequential()
regressor2.add(Bidirectional(LSTM(units=UNITS, return_sequences=True, input_shape = (x_train.shape[1],1) ) ))
regressor2.add(Dropout(0.2))
regressor2.add((LSTM(units= UNITS , return_sequences=True)))
regressor2.add(Dropout(0.2))
regressor2.add((LSTM(units= UNITS )))
regressor2.add(Dropout(0.2))
regressor2.add(Dense(units = n_future,activation='linear'))
regressor2.compile(optimizer=optimizer, loss='mean_squared_error',metrics=['mse'])
regressor2.fit(x_train, y_train, epochs=EPOCHS,batch_size=BATCH_SIZE )
loss_metrics = regressor2.evaluate(x_train, y_train, verbose=0)
loss.append(['2 hidden Layers',loss_metrics[0],loss_metrics[1]])



regressor3 = Sequential()
regressor3.add(Bidirectional(LSTM(units=UNITS, return_sequences=True, input_shape = (x_train.shape[1],1) ) ))
regressor3.add(Dropout(0.2))
regressor3.add((LSTM(units= UNITS , return_sequences=True)))
regressor3.add(Dropout(0.2))
regressor3.add((LSTM(units= UNITS , return_sequences=True)))
regressor3.add(Dropout(0.2))
regressor3.add((LSTM(units= UNITS )))
regressor3.add(Dropout(0.2))
regressor3.add(Dense(units = n_future,activation='linear'))
regressor3.compile(optimizer=optimizer, loss='mean_squared_error',metrics=['mse'])
regressor3.fit(x_train, y_train, epochs=EPOCHS,batch_size=BATCH_SIZE )
loss_metrics = regressor3.evaluate(x_train, y_train, verbose=0)
loss.append(['3 hidden Layers',loss_metrics[0],loss_metrics[1]])
```

```
loss
```

```
[['1 hidden Layer',loss:  2.706162929534912  ,mse:  2.706162929534912],
 ['2 hidden Layers',loss:  2.6740121841430664 ,mse:  2.6740121841430664],
 ['3 hidden Layers',loss:  2.6096272468566895 ,mse:  2.6096272468566895]]
```

FIG. 32 - HIDDEN LAYERS OPTIMIZATION

5.2 - Optimizer

Keeping the structure with the three hidden layers, the next stage is to fine-tune the model optimizer. We experimented with optimizers such as Adam, Stochastic Gradient descent (SGD), RMSprop and Adagrad. Running the test shown in Figure 33, Adam optimizer concluded the best in results, since it manages to give the lowest loss values when compared to the other optimizers.

```
loss = []
UNITS = n_past
EPOCHS =5
BATCH_SIZE= 32
optimizers = ['Adam', 'SGD',  'RMSprop', 'Adagrad']
for optimizer in optimizers:
    regressor = Sequential()
    regressor.add(Bidirectional(LSTM(units=UNITS, return_sequences=True,␣
 ↪input_shape = (x_train.shape[1],1) ) ))
    regressor.add(Dropout(0.2))
    regressor.add((LSTM(units= UNITS , return_sequences=True)))
    regressor.add(Dropout(0.2))
    regressor.add((LSTM(units= UNITS , return_sequences=True)))
    regressor.add(Dropout(0.2))
    regressor.add((LSTM(units= UNITS )))
    regressor.add(Dropout(0.2))

    regressor.add(Dense(units = n_future,activation='linear'))
    regressor.compile(optimizer=optimizer,␣
 ↪loss='mean_squared_error',metrics=['acc'])
    regressor.fit(x_train, y_train, epochs=EPOCHS,batch_size=BATCH_SIZE )
    loss_metrics = regressor.evaluate(x_train, y_train, verbose=0)
    loss.append([optimizer,loss_metrics[0],loss_metrics[1]])

for l in range(len(loss)):
    print('Optimizer: ',loss[l][0],'\t ,loss:',loss[l][1],'\t,mse:',loss[l][2])
```

```
Optimizer:  RMSprop     ,loss: 3.2166054248809814      ,mse: 3.2166054248809814
Optimizer:  SGD         ,loss: 3.6969220638275146      ,mse: 3.6969220638275146
Optimizer:  Adam        ,loss: 2.9892845153808594      ,mse: 2.9892845153808594
Optimizer:  Adagrad     ,loss: 42.63507080078125       ,mse: 42.63507080078125
```

FIG. 33 – OPTIMIZER OPTIMIZATION

5.3 - Units

Units consist of the internal structure (capacity) of each Layer, which means that they are the neurons of each layer. Units receives one or more inputs, performs a computation based on those inputs (often involving weights and biases), and produces an output. The idea behind the units-parameter selection has to do with the decision of "how back we should search in order to predict future values". Notice that our constant, taking into consideration the specifics of our dataset, making a prediction of 12 future samples results in covering weather data for the next 6 hours, since the weather station provide two samples of data per hour. As also shown in Figure 34, we used 64, 96 or 128 past observations to make our future predictions, the case with the 64 samples resulted the best values in term of loss.

```
overal  = []
n_future = 12 #Next # observations temperature forecast
n_past_list = [64, 96, 128] #Past # of observations
for n_past in n_past_list:
    x_train =[]
    y_train =[]

    for i in range(0, len(training_set) - n_past - n_future + 1):
        x_train.append(training_set[i : i + n_past, 0])
        y_train.append(training_set[i + n_past : i + n_past + n_future, 0])


    x_train , y_train = np.array(x_train), np.array(y_train)
    x_train = np.reshape(x_train, (x_train.shape[0] , x_train.shape[1], 1))

    EPOCHS =5
    BATCH_SIZE= 32 #len(x_train) #1024
    UNITS = n_past
    optimizer = 'Adam'
```

```
    regressor = Sequential()
    regressor.add(Bidirectional(LSTM(units=UNITS, return_sequences=True,␣
 ↪input_shape = (x_train.shape[1],1) ) ))
    regressor.add(Dropout(0.2))
    regressor.add((LSTM(units= UNITS , return_sequences=True)))
    regressor.add(Dropout(0.2))
    regressor.add((LSTM(units= UNITS , return_sequences=True)))
    regressor.add(Dropout(0.2))
    regressor.add((LSTM(units= UNITS )))
    regressor.add(Dropout(0.2))

    regressor.add(Dense(units = n_future,activation='linear'))
    regressor.compile(optimizer=optimizer,␣
 ↪loss='mean_squared_error',metrics=['acc'])
    regressor.fit(x_train, y_train, epochs=EPOCHS,batch_size=BATCH_SIZE )
    Train_loss, train_acc = regressor.evaluate(x_train, y_train, verbose=0)
    overal.append([n_past,Train_loss, train_acc])
```

```
for o in range(len(overal)):
    print('n_past: ',overal[o][0],' \t ,loss:',overal[o][1],'\t,Acc:',overal[o][2])
```

```
n_past:  64      ,loss: 3.2412593364715576      ,mse: 3.2412593364715576
n_past:  96      ,loss: 3.529353380203247       ,mse: 3.529353380203247
n_past:  128     ,loss: 3.2818715572357178      ,mse: 3.2818715572357178
```

FIG. 34 – UNITS OPTIMIZATION

5.4 – Batch Size

The size of the batch (split) of our dataset is another hyperparameter, which should be considered. We tested with 32, 64 and 128 for batch size, as also shown in Figure 35. The best results in terms of loss were achieved when the batch size was 32.

```
overal  = []
n_future = 12 #Next # observations temperature forecast
n_past = 64 #Past # of observations
BATCH_SIZE_LIST = [32, 64 ,128]
for BATCH_SIZE in BATCH_SIZE_LIST :
    x_train =[]
    y_train =[]

    for i in range(0, len(training_set) - n_past - n_future + 1):
        x_train.append(training_set[i : i + n_past, 0])
        y_train.append(training_set[i + n_past : i + n_past + n_future, 0])


    x_train , y_train = np.array(x_train), np.array(y_train)
    x_train = np.reshape(x_train, (x_train.shape[0] , x_train.shape[1], 1))

    EPOCHS =5
    #BATCH_SIZE= 32 #len(x_train) #1024
    UNITS = n_past
    optimizer = 'Adam'
    regressor = Sequential()
```

```
    regressor.add(Bidirectional(LSTM(units=UNITS, return_sequences=True,␣
 ↪input_shape = (x_train.shape[1],1) ) ))
    regressor.add(Dropout(0.2))
    regressor.add((LSTM(units= UNITS , return_sequences=True)))
    regressor.add(Dropout(0.2))
    regressor.add((LSTM(units= UNITS , return_sequences=True)))
    regressor.add(Dropout(0.2))
    regressor.add((LSTM(units= UNITS )))
    regressor.add(Dropout(0.2))

    regressor.add(Dense(units = n_future,activation='linear'))
    regressor.compile(optimizer=optimizer,␣
 ↪loss='mean_squared_error',metrics=['acc'])
    regressor.fit(x_train, y_train, epochs=EPOCHS,batch_size=BATCH_SIZE )
    Train_loss, train_acc = regressor.evaluate(x_train, y_train, verbose=0)
    overal.append([BATCH_SIZE,Train_loss, train_acc])
```

```
for o in range(len(overal)):
    print('Batch Size: ',overal[o][0],' \t ,loss:',overal[o][1],'\t,mse:',overal[o][2])
```

```
Batch Size:  32         ,loss: 3.2588484287261963      ,mse: 3.2588484287261963
Batch Size:  64         ,loss: 3.4360921382904053      ,mse: 3.4360921382904053
Batch Size:  128        ,loss: 3.7178285121917725      ,mse: 3.7178285121917725
```

FIG. 35 – BATCH SIZE OPTIMIZATION

5.5 – Activation Function

The activation function of a node defines the output of that node given an input or set of inputs and it is a major hyperparameter of the training model. We experimented with activation functions such swish, relu, sigmoid and hard sigmoid and as shown at Figure 36, relu manages highest score having the lowest loss value.

```
overal  = []
n_future = 12 #Next # observations temperature forecast
n_past = 64 #Past # of observations
activations_list =  ['swish', 'relu', 'sigmoid', 'hard_sigmoid']
for activs in activations_list:
    x_train =[]
    y_train =[]

    for i in range(0, len(training_set) - n_past - n_future + 1):
        x_train.append(training_set[i : i + n_past, 0])
        y_train.append(training_set[i + n_past : i + n_past + n_future, 0])


    x_train , y_train = np.array(x_train), np.array(y_train)
    x_train = np.reshape(x_train, (x_train.shape[0] , x_train.shape[1], 1))
    optimizer = 'Adam'
    regressor = Sequential()
    regressor.add(Bidirectional(LSTM(units=UNITS , return_sequences=True,␣
↪input_shape = (x_train.shape[1],1) ) ))
    regressor.add(Dropout(0.2))
    regressor.add((LSTM(units= UNITS , return_sequences=True)))
    regressor.add(Dropout(0.2))
    regressor.add((LSTM(units= UNITS , return_sequences=True)))
    regressor.add(Dropout(0.2))
    regressor.add((LSTM(units= UNITS )))
    regressor.add(Dropout(0.2))
    regressor.add(Dense(units = n_future,activation='linear'))
    regressor.add(layers.Activation(activs))
    regressor.compile(optimizer=optimizer,␣
↪loss='mean_squared_error',metrics=['acc'])
    regressor.fit(x_train, y_train, epochs=EPOCHS,batch_size=BATCH_SIZE )
    Train_loss, train_acc = regressor.evaluate(x_train, y_train, verbose=0)
    overal.append([activs,Train_loss, train_acc])
```

```
for o in range(len(overal)):
    print('Activation:',overal[o][0],'\t ,loss:',overal[o][1],'\t,mse:',overal[o][2])
```

```
Activation: swish               ,loss: 3.450103521347046       ,mse: 3.450103521347046
Activation: relu                ,loss: 3.429978609085083       ,mse: 3.429978609085083
Activation: sigmoid             ,loss: 226.70079040527344      ,mse: 226.70079040527344
Activation: hard_sigmoid        ,loss: 226.70079040527344      ,mse: 226.70079040527344
```

FIG. 36 – ACTIVATION FUNCTION OPTIMIZATION

5.6 - Learning Rate

Learning rate is one of the most critical hyperparameter since it affects the most the learning process. A good learning rate would mean a more complete understanding of the cost function, which the model should define. . We tested with 0.0001, 0.001, 0.01 and 0.1 for the learning rate, as also shown in Figure 37. The best results in terms of loss were achieved when the learning rate was 0.01.

```
n_future = 12 #Next # observations temperature forecast
n_past = 48 #Past # of observations
x_train = []
y_train = []
for i in range(0, len(training_set) - n_past - n_future + 1):
    x_train.append(training_set[i : i + n_past, 0])
    y_train.append(training_set[i + n_past : i + n_past + n_future, 0])
x_train , y_train = np.array(x_train), np.array(y_train)
x_train = np.reshape(x_train, (x_train.shape[0] , x_train.shape[1], 1))
```

```
acc_list = []
EPOCHS =10
BATCH_SIZE= 64 #len(x_train) #1024
UNITS = 48
lr_LIST = [0.0001, 0.001, 0.01, 0.1]
for lr in lr_LIST:
    print('learning rate:',lr)
    regressor = Sequential()
    regressor.add(Bidirectional(LSTM(units=UNITS, activation='elu', ␣
 ↪return_sequences=True, input_shape = (x_train.shape[1], 1))))
    regressor.add(LSTM(units= UNITS,activation='elu'))

    regressor.add(Dense(units = n_future, activation='elu'))
    regressor.compile(optimizer=tf.optimizers.Adam(), loss='mean_squared_error',␣
 ↪metrics=['accuracy'])
    regressor.fit(x_train, y_train, epochs=EPOCHS, batch_size=BATCH_SIZE)
    Train_loss, acc = regressor.evaluate(x_train, y_train, verbose=0)
    acc_list.append(acc)
```

```
for o in range(len(overal)):
    print('Learning Rate:',overal[o][0],'\t ,loss:',overal[o][1],'\t,mse:',overal[o][2])
```

```
Learning Rate: 0.0001    ,loss: 279.49139404296875      ,mse: 279.49139404296875
Learning Rate: 0.001     ,loss: 137.3489990234375       ,mse: 137.3489990234375
Learning Rate: 0.01      ,loss: 3.5090298652648926      ,mse: 3.5090298652648926
Learning Rate: 0.1       ,loss: 326.1465759277344       ,mse: 326.1465759277344
```

FIG. 37 – LEARNING RATE OPTIMIZATION

5.7 - Training and Evaluation Process

Now, after having complete the hyperparameter (fine) tuning process, we can analyze the train and evaluation process. At Figure 38, we can see the code used for the visualization of the dataset. The visualization shows the temperature values along with the increment number of each data sample. We saw at chapter four that our dataset consist of 87604 records of data. Figure 39 shows the loss during the training process of the model, take into consideration that we used the best value accrued in the previous stage for each one of the parameters.

```
x_axis = range(len(temperature_df.values))
plt.figure()
plt.plot(x_axis, temperature_df.values, "b", label="Data Visualization")

plt.title("Data Visualization")
plt.xlabel("TimeLine")
plt.ylabel("Temperature")
plt.legend()
plt.show()
```

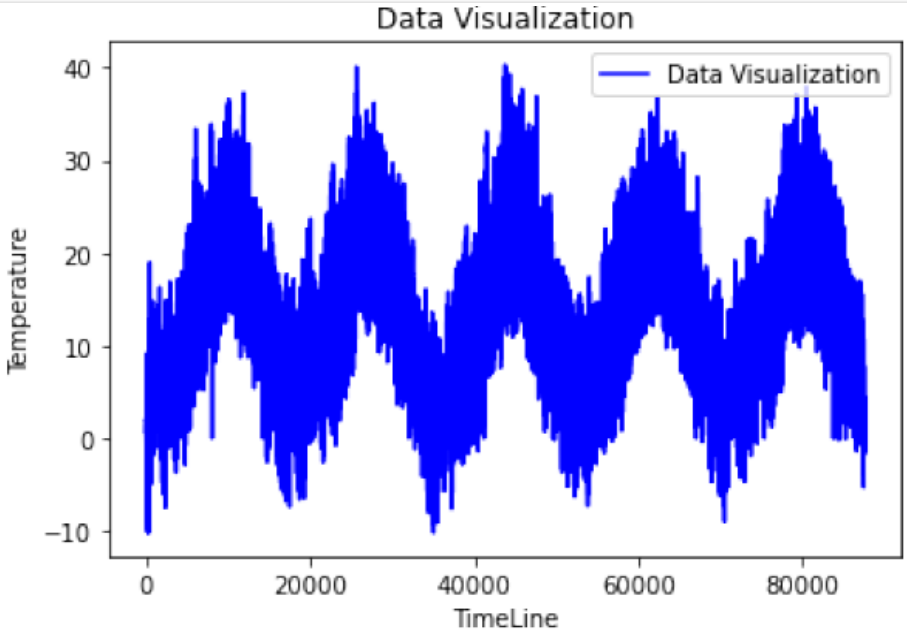

FIG. 38 – HISTORICAL DATA VISUALIZATION

```
UNITS = n_past
EPOCHS =20
BATCH_SIZE= 32
def get_lr_metric(optimizer):
    def lr(y_true, y_pred):
        return optimizer.lr
    return lr

optimizer = tf.keras.optimizers.Adam()
lr_metric = get_lr_metric(optimizer)
regressor = Sequential()
regressor.add(Bidirectional(LSTM(units=UNITS,activation ='elu',␣
 ↪return_sequences=True, input_shape = (x_train.shape[1],1) ) ))
regressor.add(Dropout(0.2))
regressor.add((LSTM(units= UNITS , return_sequences=True)))
regressor.add(Dropout(0.2))
regressor.add((LSTM(units= UNITS, return_sequences=True)))
regressor.add(Dropout(0.2))
regressor.add((LSTM(units= UNITS  )))
regressor.add(Dropout(0.2))

regressor.add(Dense(units = n_future,activation='linear'))


regressor.compile(optimizer=optimizer, loss='mean_squared_error',metrics=['acc'])
regressor.fit(x_train, y_train, epochs=EPOCHS,batch_size=BATCH_SIZE )
```

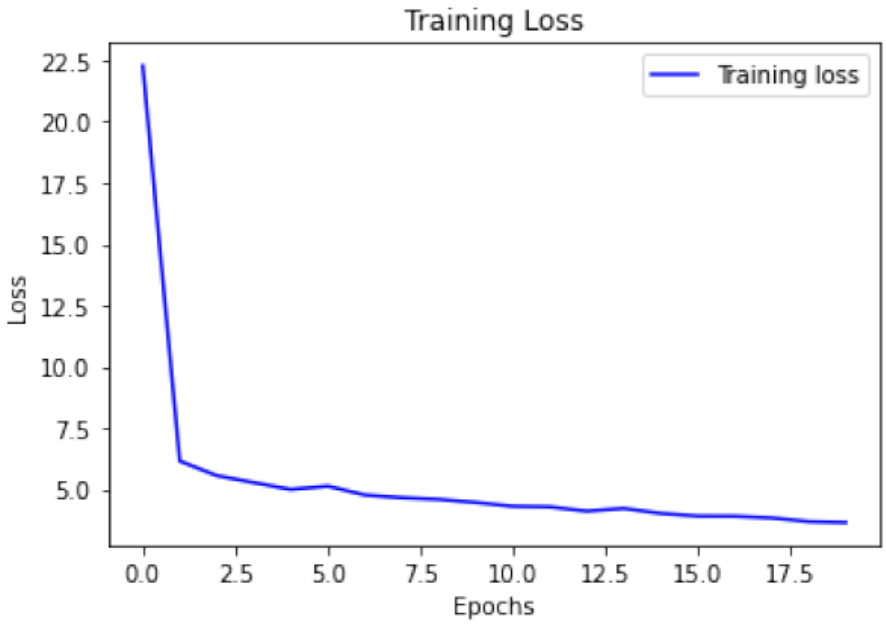

FIG. 39 – LOSS DURING TRAINING PROCESS

```
Train_loss = regressor.evaluate(x_train, y_train, verbose=0)
```

```
print("Loss:",Train_loss)
```

```
Loss: [2.508274555206299, 0.6153011322021484]
```

```
#Mean difference between train,test and real data
import statistics
from numpy import mean



predicted_train_temperature = regressor.predict(x_train)
predicted_test_temperature = regressor.predict(x_test)
min_train =  y_train.min()
min_test  =  y_test.min()
max_train =  y_train.max()
max_test  =  y_test.max()
min_pred_train = predicted_train_temperature.min()
max_pred_train = predicted_train_temperature.max()
min_pred_test  = predicted_test_temperature.min()
max_pred_test  = predicted_test_temperature.max()
mtrain = mean(predicted_train_temperature - y_train)
mtest  = mean(predicted_test_temperature - y_test)

print('The mean difference between train and real data is:',mtrain)
print('The mean difference between test  and real data is:',mtest)
print('Min value of real(train) Data:',min_train ,'Predicted Min:
 ↪',min_pred_train)
print('Min value of real(test) Data:',min_test ,'Predicted Min:',min_pred_test)

print('Max value of real(train) Data:',max_train ,'Predicted Min:
 ↪',max_pred_train)
print('Max value of real(test) Data:',max_test ,'Predicted Min:',max_pred_test)
```

```
The mean difference between train and real data is: -0.20162443163668223
The mean difference between test  and real data is: -1.1455204407374062
Min value of real(train) Data: -10.2 Predicted Min: -10.03505
Min value of real(test) Data: 3.3 Predicted Min: 3.450125
Max value of real(train) Data: 40.3 Predicted Min: 38.35743
Max value of real(test) Data: 6.4 Predicted Min: 4.0550685
```

FIG. 40 PRE-EVALUATION STATISTICS

We explain in the previous section that we have split our dataset into 80% for train process and 20% for the evaluation process. We need to compare the predicted values of our model both for the train and evaluation process. Figure 40 shows the code, which help us to check the mentioned comparison. The results show that during the **train process,** the temperature values predicted by our model have a mean difference of minus 0.201 Celsius degrees compared to the actual temperature values. The minimum temperature value of our dataset into train process was minus 10.2-Celsius degrees while the minimum temperature value, which our model predicted, was minus 10.035 Celsius degrees. Correspondingly, the maximum temperature value of our dataset into train process was 40.3-Celsius degrees while the maximum temperature value, which our model predicted, was 38.35-Celsius degrees.

Respectively, during the **evaluation process**, the temperature values predicted by our model have a mean difference of minus 1.1455-Celsius degrees compared to the actual temperature values. The minimum temperature value of our dataset into evaluation process was 3.3-Celsius degrees while the minimum temperature value, which our model predicted, was 3.45-Celsius degrees. Correspondingly, the maximum temperature value of our dataset into evaluation process was 6.4-Celsius degrees while the maximum temperature value, which our

model predicted, was 4.055-Celsius degrees. Table 4 shows the min and max values of train and evaluation data as long as the values, which our model predicted based on train and test dataset.

	Prediction | Train	Prediction | Test
Min	-10.2 |-10.35	3.3 | 3.45
Max	40.3 | 38.35	6.4 | 4.05

TABLE 9 – MIN MAX VALUES

Figure 41 and 42 visualize the prediction over 600 samples of the dataset when using then training and the evaluation data respectively, compared to the real temperature distribution of these data samples.

```
pre_values = []
real_values= []
look_back =50
for i in range(look_back-1,-1,-1):
    last_observations =np.array(training_set[len(training_set)-n_past-n_future␣
 ↪-n_future*i : len(training_set) -n_future -n_future*i])
    last_observations = np.reshape(last_observations, (1, last_observations.
 ↪shape[0], 1))
    pre_values.append( regressor.predict(last_observations)[0])
pre_values  = np.reshape(pre_values  , n_future*look_back)


visualize_temerature_trend(pre_values,training_set[-n_future*look_back:
 ↪],n_future*look_back, "Prediction Vs Real Distribution On Training Data")
```

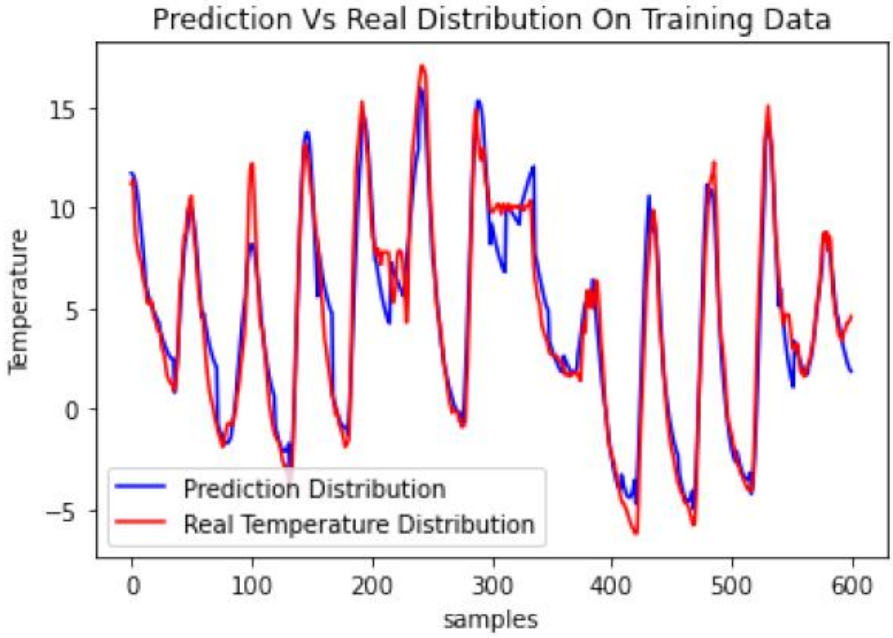

FIG. 41 – PREDICTION VISUALIZATION ON TRAINING DATA

```
pre_values = []
real_values= []
look_back =50
for i in range(look_back-1,-1,-1):
```

```
    last_observations =np.array(test_set[len(test_set)-n_past-n_future
 ↪-n_future*i : len(test_set) -n_future -n_future*i])
    last_observations = np.reshape(last_observations, (1, last_observations.
 ↪shape[0], 1))
    pre_values.append( regressor.predict(last_observations)[0])
pre_values  = np.reshape(pre_values  , n_future*look_back)


visualize_temerature_trend(pre_values,test_set[-n_future*look_back:
 ↪],n_future*look_back, "Prediction Vs Real Distribution On Test Data")
```

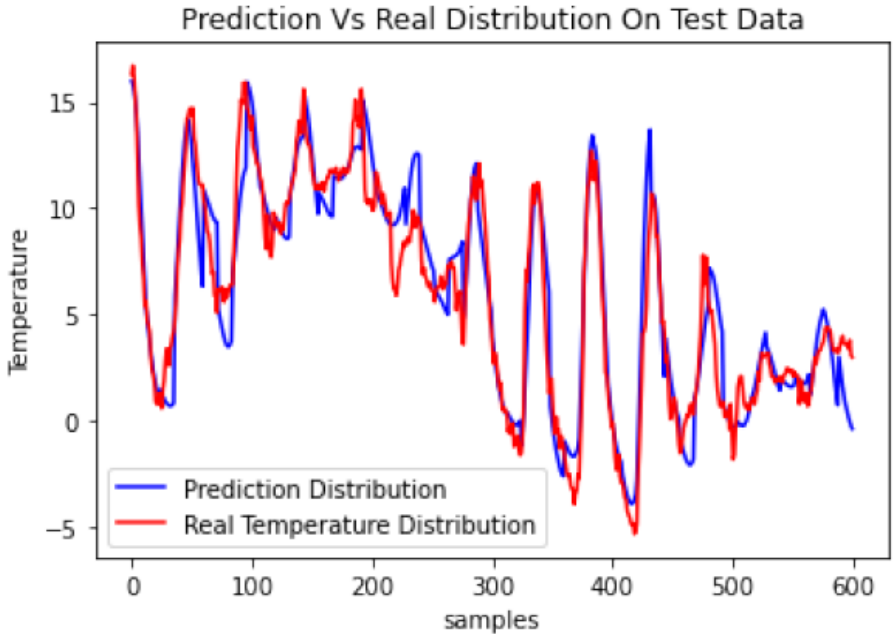

FIG. 42 – PREDICTION VISUALIZATION ON TEST DATA

Finally, we tested the proposed prediction model over future data. Specifically, our model predicted 120 temperature values, which were then compared to the current observations as arriving by the weather station. Figure 43 presents the predicted and the real temperature values; it is clear that our model managed to predict what the temperature would be in the specific resign.

```
pre_values = []
real_values= []
look_back =10
for i in range(look_back-1,-1,-1):
    last_observations =np.array(dataset_predict["temperature"].
 ↪values[len(dataset_predict["temperature"].values)-n_past-n_future -n_future*i :
 ↪ len(dataset_predict["temperature"].values) -n_future -n_future*i])
    last_observations = np.reshape(last_observations, (1, last_observations.
 ↪shape[0], 1))
    pre_values.append( regressor.predict(last_observations)[0])
pre_values  = np.reshape(pre_values  , n_future*look_back)


visualize_temerature_trend(pre_values,dataset_predict["temperature"].
 ↪values[-n_future*look_back:],n_future*look_back, "Prediction Vs Real␣
 ↪Distribution on Current Data")
```

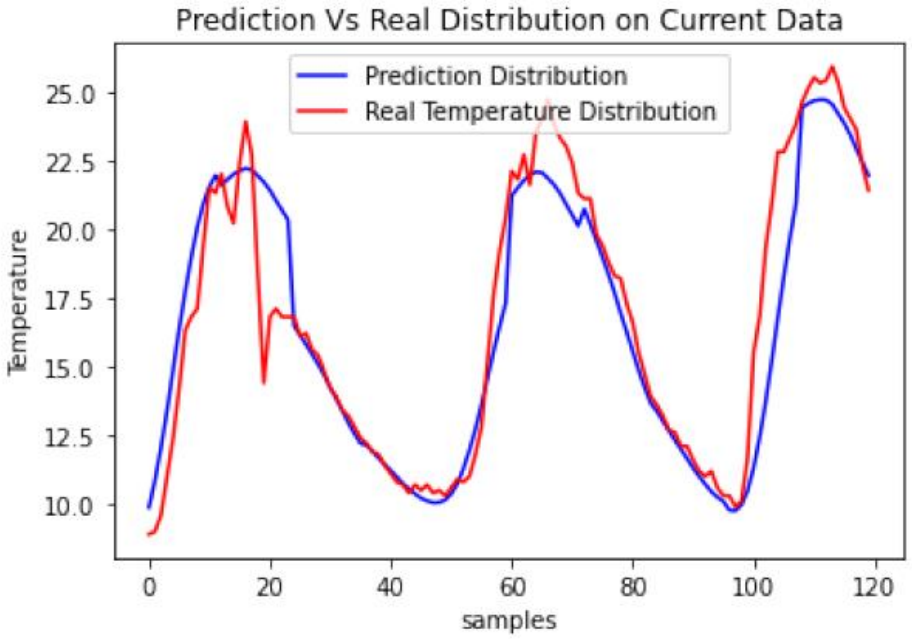

FIG. 43 – PREDICTION EVALUATION

Figure 44 shows on the prediction process our model achieve a 0.98 correlation Coefficient between the predicted values and the actual values which the weather station report on future time

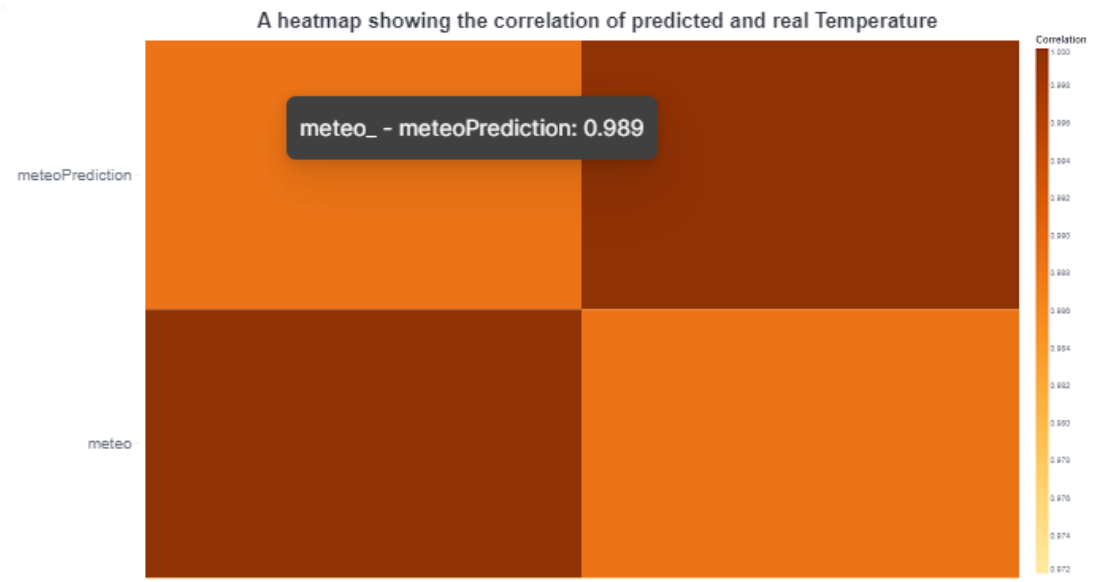

FIG. 44 – CORRELATION HEATMAP FOR PREDICTED AND REAL TEMPERATURE VALUES

At Figure 45, we can see the output of our model, predicting the next 12 temperature samples based on the last 64 observations. Figure 46 shows, in a common plot for the last 120 samples, the actual temperature values, the temperature values predicted by the model for the corresponding timestamp as well as the next 12 values which our model predict based on the last 64 samples.

```
last_observations =np.array(dataset_predict["temperature"].values[-n_past:])
last_observations = np.reshape(last_observations, (1, last_observations.
 ↪shape[0], 1))
predicted_temperature = regressor.predict(last_observations)
print('Next 12 Predicted temperature {}'.format(predicted_temperature))
```

```
Next 12 Predicted temperature [[13.275452  12.9748955 12.690847  12.413246
12.158142  11.900476
  11.635497  11.427635  11.241165  11.083024  10.989482  10.937527 ]]
```

FIG. 45 – OUTPUT FOR THE NEXT 12 TEMPERATURE SAMPLES

```
predicted = np.append(pre_values,np.zeros(n_future))
real      = np.append(dataset_predict["temperature"].values[-n_future*look_back:
 ↪],np.zeros(n_future))
zeros     = np.zeros(n_future*look_back)
next_pred = np.append(zeros,predicted_temperature)
```

```
visualize_temerature_trend2(predicted,real,next_pred,(n_future*look_back)+n_future
 ↪"Next Temperature values Prediction")
```

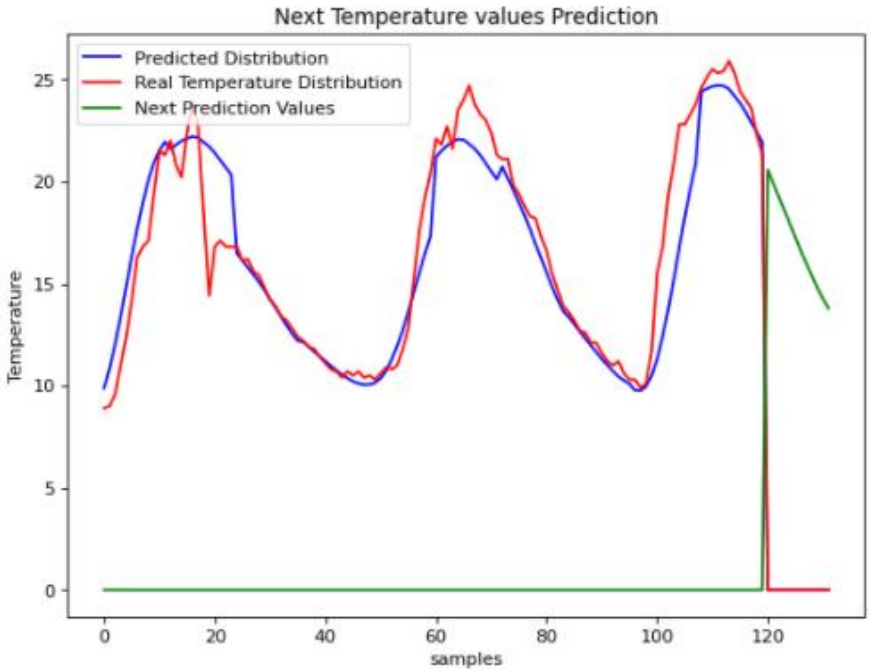

FIG. 46 – NEXT 12 SAMPLES PREDICTION

Figure 47 is visualized the correlation between the prediction and real values from weather station of meteo.gr and openweathermap.org. Figure 48 is a dashboard concerning the prediction process. The left Gauge visualization shows the temperature at according the next hour while the right histogram shows the next 12 predicted values for the temperature.

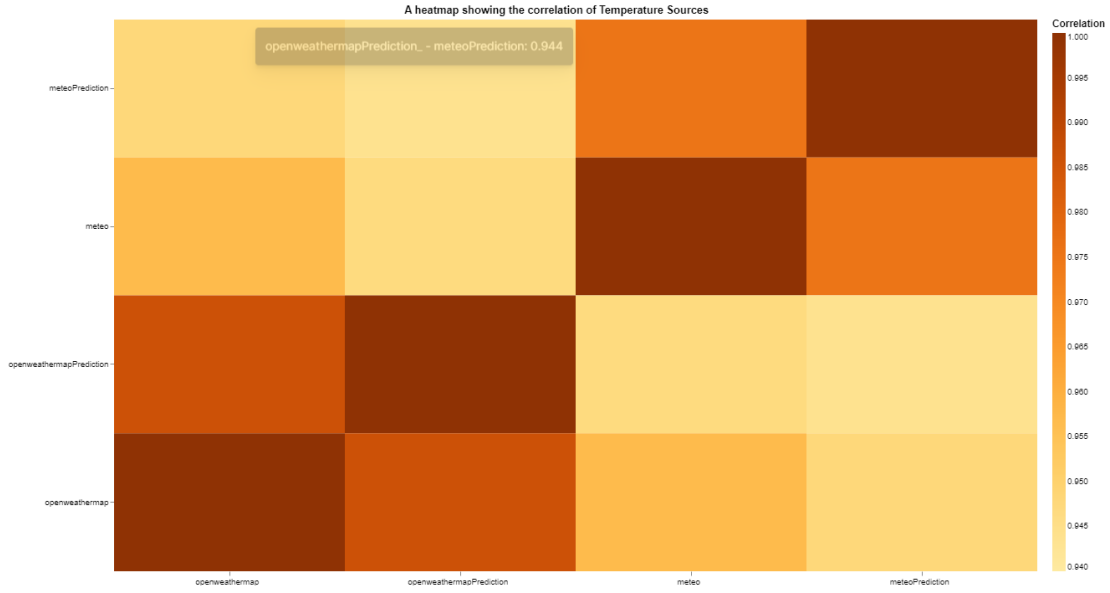

FIG. 47 – PREDICTION AND REAL VALUES HEATMAP CORRELATION

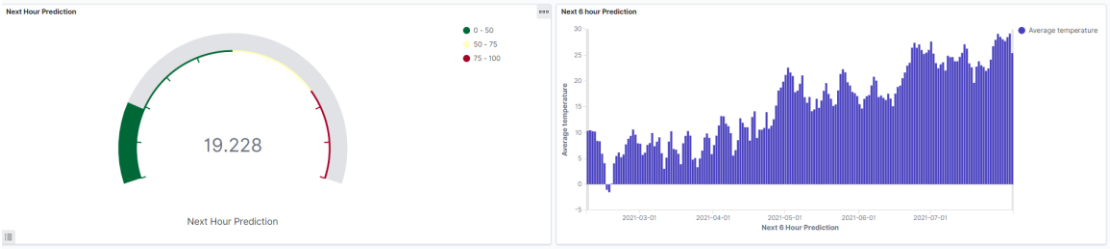

FIG. 48 – PREDICTION DASHBOARD NEXT TEMPERATURE VALUES

Figure 49 is also a dashboard, which tries to assess the correlation between the actual reported values of the weather stations with the coresponding, which the forecast models had predicted for the same time. Our model create the predicted values for meteo.gr while the prediction values for the openweathermap.org are gathered from the openweathermap api for the weather forecasting.

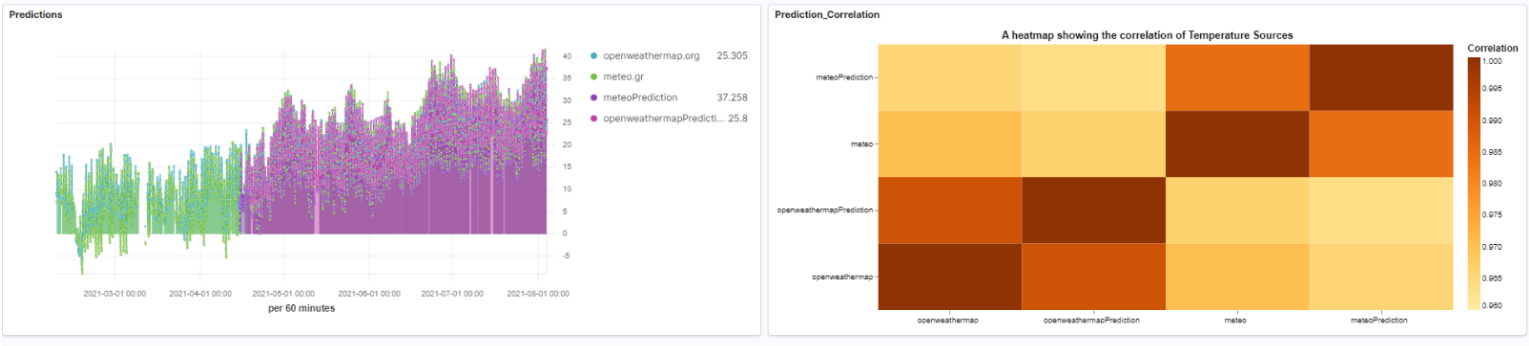

FIG. 49 – PREDICTION DASHBOARD DISTRIBUTION AND CORRELATION

Closing chapter 5 we saw how we chose the parameters of our model. We used the Grid Search technique to select the optimal options for the corresponding parameter. We concluded that our model using 3 hidden layers, adam optimizer, 64 units, 32 as batch size, relu as activation function and 0.01 value for learning rate achieves the best performance in terms of loss. We saw that during train process, the temperature values predicted by our model had a mean difference of minus 0.201 Celsius degrees compared to the actual temperature values while the respective value during evaluation process was about minus 1.1455-Celsius degree. The above results indicated that in the worst-case scenario our prediction might deviate by one degree Celsius less. The above mean difference shows the expected outcome, that our model achieved better results during train than the evaluation process.

We could also conclude that our model is sufficient in the short-term forecast, achieving a 0.98 correlation coefficient between the predicted values and the actual values which the weather station report on future time.

Finally, we end up that our model observing the last 64 pasts can satisfactorily predict the next 12 observations. We could not know how our model would respond to extreme changes and fluctuations in temperature within the last measurements. This particular issue is possibly a good example for further research.

Chapter 6 – Conclusions and Future Extensions

6.1 Conclusion

The purpose of the current work was to try to identify, if there are differences between the meteorological data reported by the different sources. The main tool to identify any differences is the visualizations through the kibana platform. The difficulties we had to overcome were that, we had to scrape our data from different sources with the particularities of each one, as well as we had to transform our data into a common format and units of measurement.

We can conclude from our results coming from our correlation coefficients that the temperature and barometer variables present a strong correlation coefficient while wind velocity, precipitation and humidity variables present less strong correlation coefficient. User can also see the actual values for each source and weather variable at a specific time range.

The visualizations, which have been developed at the current work, consist an effective and useful tool in order to locate climate data aggregation between different weather stations and sources while it can also be used efficiently for short-term temperature prediction. User can use these two modules independently, once for values and distributions of weather (historical) data and the other one for the corresponding future values of our proposed prediction model. It is worth noting that with the given technologies and processes with which the work has been developed, it is relatively easy to add even more meteorological stations and weather parameters, while kibana provides the ability to create new default and custom visualizations and dashboard.

Regarding the temperature prediction process, the problems we had to face were the huge amount of data, as well as the lack of computing resources. For this reason, our model was limited to a short-term prediction process of 6-hour prediction (12 prediction temperature values). However, on the prediction process our model achieve a 0.98 correlation Coefficient between the predicted values and the actual values, which the weather station report on future time.

As a final presumption, the work proves that there are clear differences on climate parameters (temperature, humidity, wind, precipitation, etc.) between the different stations in the Tripoli – Peloponnese – Greece- area, even though these stations are very close to each other, while it is possible to predict short term temperature values through the model we developed, using Long Short-Term Memory (LSTM) neural network.

6.2 Future Work

The present work is a remarkable introduction to the meteorological prediction process. The proposed model can be used for short term forecasting. A future extension, could be the training, for medium-term forecast which however requires advance computing power and memory capacity. Greater computing power could enable the model to use more past temperature values to create predictions.

This work is limited to meteorological data of a specific geographical area – Tripoli, Peloponnese, Greece. It would be very interesting if this work were carried out for other regions of the country, to see if we have similar results

An extension would be the use of the model in sectors such as agriculture to predict impending diseases in related crops.

Finally, an interesting extension for our model would be the prediction of all-weather parameters such as temperature, humidity, precipitation, barometer, wind velocity and direction.

References

Aggarwal, C. (n.d.). *Neural Networks and Deep Learning.*

Behera, Keidel, & Debnath. (2018). *Context-driven multi-stream LSTM (M-LSTM) for recognizing fine-grained activity of drivers.*

Cerqueira, Torgo, Pinto, & Soares. (2019). *Arbitrage of forecasting experts.*

DB-Engines Ranking of Search Engines. (n.d.). Retrieved from https://db-engines.com/en/ranking/search+engine

Diederik P. Kingma, J. B. (2015). Adam: A Method for Stochastic Optimization. *International Conference on Learning Representations (ICLR).*

Duchi, J., Hazan, E., & Singer, Y. (2010). Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Neural Information Processing Systems (NeurIPS).*

ElasticSearch official Page. (n.d.). Retrieved from https://www.elastic.co/elasticsearch/

freemeteo.gr station information. (n.d.). Retrieved from https://freemeteo.gr/kairos/tripoli/o-kairos-tora/simeio/?gid=252601&language=greek&country=greece

Hastie, T., Tibshirani, R., & J, J. F. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction (2nd ed.).*

Hellenic National Meteorological Service. (n.d.). Retrieved from http://www.emy.gr/

Hewage, Behera, Trovati, & Pereira. (2019). *Long-short term memory for an effective short-term weather forecasting model using surface weather data.*

Hewage, P. (2020). *Temporal convolutional neural (TCN) network for an effective weather forecasting using time-series data from the local weather station.*

Hewage, P., Trovati, M., Pereira, E., & Behera , A. (2021). *Deep learning-based effective fine-grained weather forecasting model.*

Hinton, G. (2012). Neural Networks for Machine Learning. In G. Hinton.

Hinton, G. E. (2006, July). Reducing the Dimensionality of Data with Neural Networks. *Science*, pp. 504-507.

Hochreiter, S., & Schmidhu, J. (1997). *Long Short-term Memory.*

Kibana Official Site. (n.d.). Retrieved from https://www.elastic.co/kibana

Kreuzer, D., & Munz, M. S. (2020). *Short-term temperature forecasts using a convolutional neural network — An application to different weather stations in Germany.*

Lorenz, E. N. (1963). *Journal of the Atmospheric Sciences*, pp. 130 - 141.

M, R., Islam, A., Nadvi , S., & Rahman , R. (2013). *Comparative study of ANFIS and ARIMA model for weather forecasting in Dhaka.*

Meteo Weather Station Page. (n.d.). Retrieved from https://penteli.meteo.gr/stations/tripoli/

Mishra, D., & Joshi, P. (2021). *A Comprehensive Study on Weather Forecasting using Machine Learning.* Retrieved from https://ieeexplore.ieee.org/document/9596117/authors#authors

Nair, V., & Hinton, G. E. (2010). Rectified Linear Units Improve Restricted Boltzmann Machines. *In Proceedings of the 27th International Conference on Machine Learning (ICML-10)* , (pp. 807-814).

National Observatory of Athens. (n.d.). Retrieved from https://www.noa.gr/

Newton, I. (1680). *Philosophiæ Naturalis Principia Mathematica .*

Okairos.gr - About Us. (n.d.). Retrieved from https://www.okairos.gr/faq.html

Okairos.gr - Hourly ForeCast. (n.d.). Retrieved from https://www.okairos.gr/%CF%84%CF%81%CE%AF%CF%80%CE%BF%CE%BB%CE%B7.html?v=%CF%89%CF%81%CE%B9%CE%B1%CE%AF%CE%B1

OpenWeathermap.org - About Us. (n.d.). Retrieved from https://openweathermap.org/faq

Pradeep, H., Marcello, T., Ella, P., & A. B. (2021). *Deep learning-based effective fine-grained weather forecasting model.*

Python Official Page. (n.d.). Retrieved from https://www.python.org/

Ramachandran, P., Zoph, B., & V., L. Q. (2018). *Searching for Activation Functions.*

Richardson, L. F. (1922). *Weather forecasting with numerical methods .*

Robbins, H. (1951). *A Stochastic Approximation Method.* Retrieved from https://www.semanticscholar.org/paper/A-Stochastic-Approximation-Method-Robbins/34ddd8865569c2c32dec9bf7ffc817ff42faaa01?p2df.

Sanchez-Fernandez, M., de-Prado-Cumplido, M., Arenas-Garcia, J., & Perez-Cruz, F. (2004). *SVM multiregression for nonlinear channel estimation in multiple-input multiple-output.*

Schneider, R., Massimo, B., Geer, A., & Arcucci, R. (2022). ESA-ECMWF Report on recent progress and research directions in machine learning for Earth System observation and prediction. *Nature*. Retrieved from https://www.nature.com/articles/s41612-022-00269-z

Schult, M., Betancourt, B. G., Kleinert, F., Langguth, M., L. H., L., Mozaffari, & Stadtler, S. (2021). *Can deep learning beat numerical weather prediction?*

Scrapy - ElasticSearch. (n.d.). Retrieved from https://github.com/jayzeng/scrapy-elasticsearch

Scrapy Framework. (n.d.). Retrieved from https://scrapy.org/: https://scrapy.org/

Sharaff A, R. (2018). SR Comparative analysis of temperature prediction using regression methods and back propagation neural network., (pp. 739–742). Retrieved from https://doi.org/10.1109/icoei.2018.8553803

Sun, C.-S., Wang, & X-R, L. (2008). *A vector autoregression model of hourly wind speed and its applications in hourly wind speed forecasting.*

Voyant, C. (2017). *Machine learning methods for solar radiation forecasting: a review.* Retrieved from https://doi.org/10.1016/j.renene.2016.12.095

Xalazi.gr - Weather Data. (n.d.). Retrieved from http://www.xalazi.gr/prognwsh-kairou/prognosi-5-imeron?type=FiveDays&city=1178#