UNIVERSITY OF TROMSØ

# Data Science using Python: Analysis of Salmon farming in Norway

by

Andrea Spreafico

A thesis submitted in partial fulfillment for the degree of Computer Science
Computer Science

in the
Faculty of Computer Science
Department of Computer Science

May 2017

# Abstract

The term Data Science refers to the collection of knowledges and skills, mainly about statistics and computer science, that allow to collect, analyze and display data in order to understand actual phenomena. Since there isn't a default technique to extract informations from the data this study investigates about the possibility of using Python in order to realize a system able to analyze, display and forecast data.
This kind of systems are commonly used to elaborate data coming from high-interest area; for instance, during this work was used a dataset about the Norwegian aquaculture industry, since Norway represents the forefront of innovation and development in this area, in particular about the Norwegian salmon farming, and also because this business produces a lot of unstructured and not interpreted data.
The implementation procedure reported in this study shows that Python provides several modules and packages that could be useful for a data analysis and displaying on dataset coming from any kind of area of interest.
Further more, this thesis provides several analysis results about each single norwegian county involved in the Norwegian aquaculture business, such as graphics, correlation coefficients, trend line indicators and some initial prediction of the future.

# Acknowledgements

I would like to offer my special thanks to my supervisor Ståle Walderhaug.
I would like to express my very great appreciation to Bård Johan Hanssen.
I am particularly grateful to Sara Björk.
I am grateful for the support and good times given by the staff of SINTEF Nord.

My special thanks are extended to:
- My family.
- Università di Milano Bicocca.
- UiT, Universitetet i Tromsø.

# Contents

# List of Figures

# List of Tables

# Abbreviations

| | |
|---|---|
| **SIA** | **S**ingle **I**nput **A**nalyzer |
| **MIA** | **M**ultiple **I**nput **A**nalyzer |
| **AR** | **A**uto **R**egressive |
| **MA** | **M**oving **A**verage |
| **ARMA** | **A**uto **R**egressive **M**oving **A**verage |
| **ARIMA** | **A**uto**R**egressive **I**ntegrated **M**oving **A**verage |
| **MAPE** | **M**ean **A**verage **P**ercentage **E**rror |

# Chapter 1

# Introduction

During the last few years we have witnessed an ever-increasing production of data in any sector all around the world. For this reason instruments and techniques for analyzing and understanding these data are becoming more and more indispensable, in order to extract useful information that might be used to improve business strategies or people's life condition.

Data Science is a recent launch field which contains processes and systems that could be used to extract knowledge from data, either structured or unstructured. Since the newness of this field, would be very interesting to test and evaluate different ways to apply daily technologies to its procedures and systems.

Python is a simple interpreted, object-oriente and high-level programming language that has a easy to learn syntax. Since it provides severals modules and package, the use of Python during a Data Science process could be very productive.

The processes and systems which belong to the Data Science field might be applied to high-interest economic areas, such as the Aquaculture industry in Norway. This business, in particular the Norwegian salmon farming, has a big economic repercussions on the country, and at the same time is producing a huge amount of data, so it would be very helpful to restructure and analyze it.

This thesis will contributes providing a documented implementation of an analysis, displaying and prediction system using Python applied to the Norwegian salmon farming.

## 1.1 Aim of the study

The focus of this study will be on:

- Initial approach with Data Science field, in order to investigate and document possible techniques, methods and approaches.

- Testing Python potential in Data Science field, describing implementation procedures and reporting pro and cons.

- Report the initial analysis and displaying results about Norwegian salmon farming, in order to provide structured, described and readable data that might be used for future works.

## 1.2 Research Objectives

The above aim will be accomplished by fulfilling the following research objectives:

**1) Collect as much data about aquaculture in Norway as possible.**

- Which kind of data is possible to obtain about aquaculture general statistics in Norway? Where is possible to find it? Are that available for everyone?
- Which kind of data is possible to obtain about aquaculture of single locations in Norway? Where is possible to find it? Are that available for everyone?

**2) Increase accessibility and availability of the data.**

- How you can create a unique dataset that contains and summarize all the data previous collected?
- Which kind of structure allows to the total dataset to be more accessable and readadble than the original single sources?

**3) Analyze and display the data.**

- Which kind of Python functions is possible to use for analyze and displaying data?
- Which kind of requirements does it need and how is possible to implement it?
- Why Python could be a good solution for data analysis and displaying?
- Which kind of relationships and patterns about the data is possible to identify using the result graphics? How is possible to identify it?
- How is possible to check out the data trend line?
- Which kind of informations have been reported for future reuse? How it's possible to access it? (Informations such as correlation coefficients, trend line equations,..)

**4) Extract information from the data.**

- Which parameters about aquaculture in Norway are increasing? How fast are they increasing/decreasing?
- How you can compare different parameters trend line?
- Which kind of correlations is possible to find out between different parameters? How is possible to show it? What is possible to extract from that?

**5) Prediction of values about the data.**

- Which kind of Python utilities is possible to use for time series predictions?
  - How Python works for time series prediction systems implementation?
  - Which kind of accuracy it provides about the predicted values?
  - Would it be a good way for let the people get some experience with the machine learning field?
- Would be useful to have the possibility of forecasting some future data?
- Which kind of data might be the most useful to know for people into the Aquaculture field?

**6) Recommendations to future work and extra ideas.**

- How it could be possible to improve the Anaysis and Displaying system?
- How it could bee possible to improve the Forecasting system?
- Which kind of services is possible to provide using the collected informations and the implemented systems?
  - How you can provide the analysis system like a service?
  - How you can provide the prediction system like a service?

## 1.3 Thesis Structure

**Background Theory**

Here is report the relevant background theory that should be needed for better understand the work done during this thesis.

**I Part: Data collection and validation**

During this initial part of the work the main purpose is to collect as much data as possible. The data have to be related with the current are of interested and at the same time considered reliable. Once collected enough data, is time to describe and structure them in a proper dataset.

**II Part: System implementation**

During this part of the work is implemented the Python system and reported the procedure, in order to try to find as many answers as possible to the initial goals. The implemented system will be divided in several subsystems, that's because it allows an easier implementation procedure and a higher reusability. It's possible to checkout an overview about the implemented subsystems in the next section. [4.2]

**III Part: Result, Discussion and Conclusions**

In the current part of the study are reported the results and the relative interpretations. Further more, is reported a general sumamrize about the study and also any kind of evaluations, challenges, limitations encountered and some recommendations for future works.

**IV Part: Full code Implementation**

In this last part of the study is reported the full commented code implementation for each single system implemented. It would be useful to checkout more details about the code, since during the II Part is reported just the most important rows and functions of the code.

## 1.4 Previous Works

The implementation of the prediction system in this thesis was based on a previous work, which provides a basic implementation of a forecasting system with Python.
That particular work was showing how to create a general ARIMA Model for Time Series Forecasting with Python. During this study that implementation has been improved, customized and applied to the current context.

The previous work source website is named "machinelearningmastery.com", and here's the reference: [1]

## 1.5 Work repository

Before start to read the implementation procedure about this work, it's important to know that is possible to check out the system's full implementation on Github.
I suggest to check it out and download the following repository. It allows to test the system and better understand how it is structured and how it works.
Further more, it's possible to find inside the same repository all the needed datasets and a "Manual" wich contains the instructions about how to use it.

The Github repository is:
`https://github.com/Sprea22/Python_Systems`

The direct Zip file download is:
`https://codeload.github.com/Sprea22/Python_Systems/zip/master`

# Chapter 2

# Background Theory

## 2.1 Data science

It's really important to have a general idea about what "Data Science" means since this thesis procedure is strongly based on the classic Data Science Process.

We can define Data Science like a "concept to unify statistics, data analysis and their related methods" in order to understand and analyze actual phenomena with data.[2] It includes theories drawn from many field within the broad areas of mathematics, statistics, information science and computer science.

In the computer science area are particular important the subdomains of:

- Machine learning
- Classification
- Cluster Analysis
- Data mining
- Databases
- Visualization

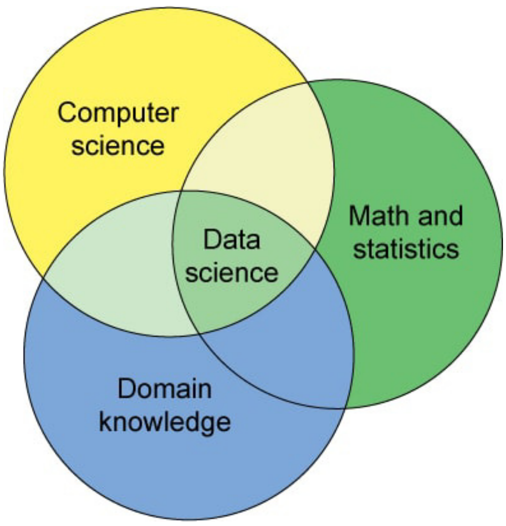



FIGURE 2.1: Data science concept

Here is reported a short definitions about the main subdomains considered by this study:

- **Data mining**: Is the computing process of discovering patterns in large data sets. The overall goal of the data mining process is to extract information from a data set and transform it into an understandable structure for further use.
- **Data Visualization**: It involves the creation and study of the visual representation of data. The primary goal of data visualization is to communication information clearly and efficiently via graphics and plots.
- **Machine learning**: Is a subfield of computer science that gives computers the ability to learn without being explicitly programmed.[3] More useful specific informations about this field are provided in the following section [2.2].

The follow image represents the "Blitzstein and Pfister's framework" and provides a clear overview of the Data Science process.

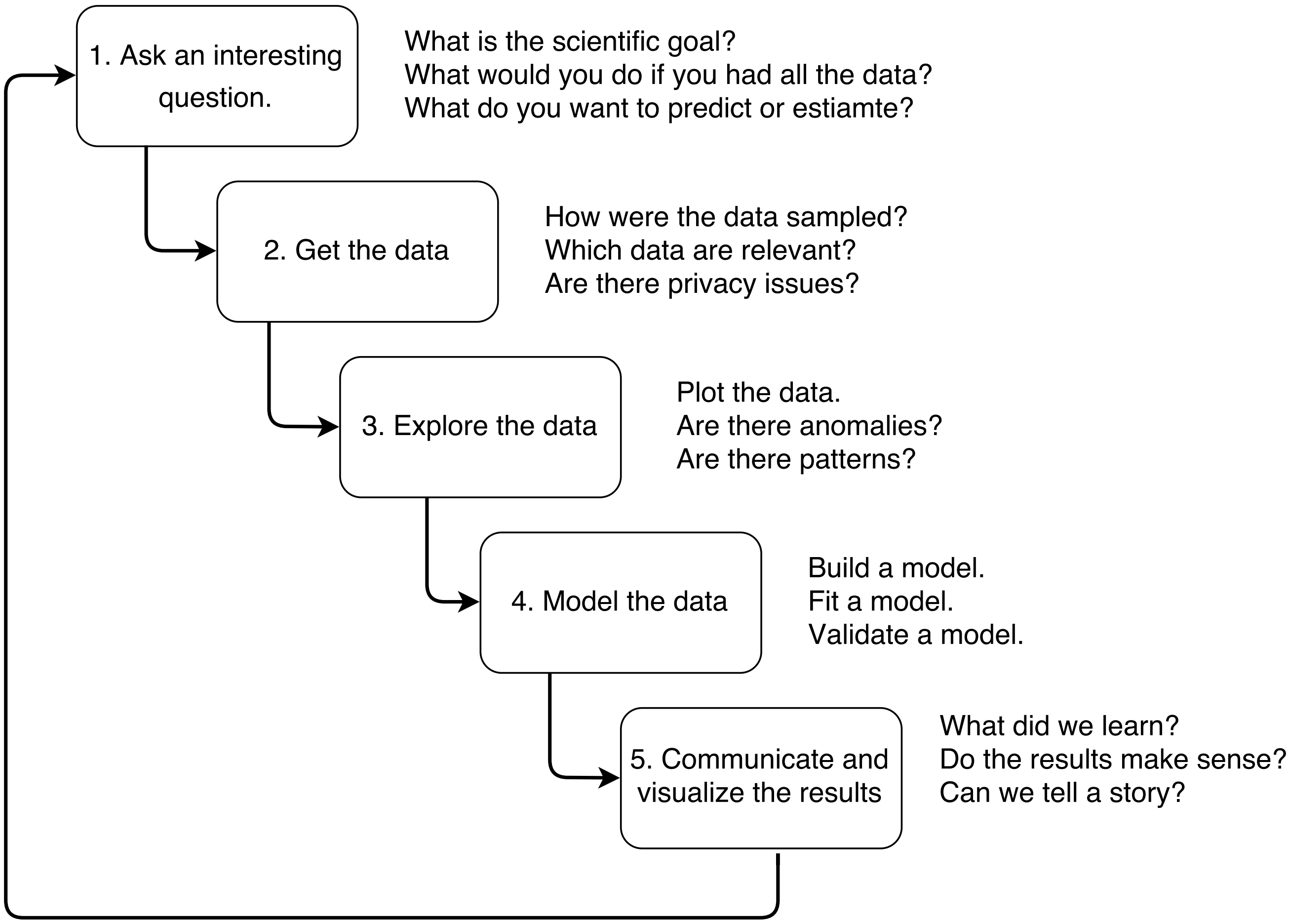



FIGURE 2.2: Data science process

## 2.2 Machine learning

How is reported in the previous section, this subfield of computer science gives ”computers the ability to learn without being explicitly programmed”.
Machine learning explores the study and construction of algorithms that can learn from and make predictions on data.

There are several machine learning algorithm, each one of them is used for a different purpose and a different domain. For examples:

- Deep Learning
- Neural Network
- Regularization
- Clustering
- **Regression**: This specific domain contains the model used in this study.
- Bayesian

### 2.2.1 Time Series analysis and predictions

Time Series forecasting is an important area of machine learning, but that is often neglected. Is that important mainly beause there are so many prediction problems that involve a time component, and these problems are neglected because it is this time component that makes time series problems more difficult to handle.[4]

” A time series is a sequence of observations taken sequentially in time. ”[5]

Classic example of a time series dataset:

| **Date** | **Paramater** |
|---|---|
| Time #1 | observation |
| Time #2 | observation |
| Time #3 | observation |

Understanding a dataset is called time series analysis and it can helps to make better prediction, but sometimes it’s not required and can result in a large of technocal investment in time and expertise.

Making predictions could be called time series forecasting and it involves taking models fit on historical data and using them to predict future observations.

### 2.2.2 Autoregressive integrated moving average (ARIMA)

Since this a very complicated and deep topic, this study provided just an initial implementation and descripton of it. During this section are provided some basic definitions and overviews enough to understand the general logic behind a forecasting system. If you are particular interested in this topic my suggestion is to read more about it, in the specific the mathematic side.

**AR model**: an autoregressive model is a representation of a type of random process; as such, it is used to describe certain time-varying processes in nature, economics, etc. The autoregressive model specifies that the output variable depends linearly on its own previous values and on a stochastic term (an imperfectly predictable term); thus the model is in the form of a stochastic difference equation.[6]

**MA model**: a moving-average model is a common approach for modeling univariate time series. The moving-average model specifies that the output variable depends linearly on the current and various past values of a stochastic (imperfectly predictable) term.[7]

**ARMA model**: an autoregressive-moving-average model provides a parsimonious description of a stationary stochastic process in terms of two polynomials, one for the autoregression and the second for the moving average. Basically it combines both AR and MA models into a unique representation.[8]

**ARIMA model**: is a generalization of an autoregressive moving average (ARMA) model. Both of these models are fitted to time series data either to better understand the data or to predict future points in the series (forecasting).
This model is applied in some cases where data show evidence of non-stationarity, where an initial differencing step (corresponding to the "integrated" part of the model) can be applied one or more times to eliminate the non-stationarity.[9]

**ARIMA(p, d, q)**

- **p** is the number of autoregressive terms (How many preceding values are examinated for the current value's forecast).
- **d** is the number of nonseasonal differences needed for stationarity.
- **q** is the number of lagged forecast errors in the prediction equation.

## 2.3 Aquaculture in Norway

During the last years there has been a very rapid development of Norway's aquaculture industry, and the production of Atlantic salmon has grown to become a major sector of its economy. The industry is now an economic pillar for several Norwegian coastal communities.[10]

The Aquaculture industry in Norway is dominated by its finfish sector, with Atlantic salmon and Rainbow trout accounting for 93.9% and 5.8% respectively of total volume produced.

This business takes place in the counties along most of the country's coastline. In the finfish sector Nordland is the dominant producer county, with Hordaland coming second, Møre og Romsdal third, and Troms fourth.

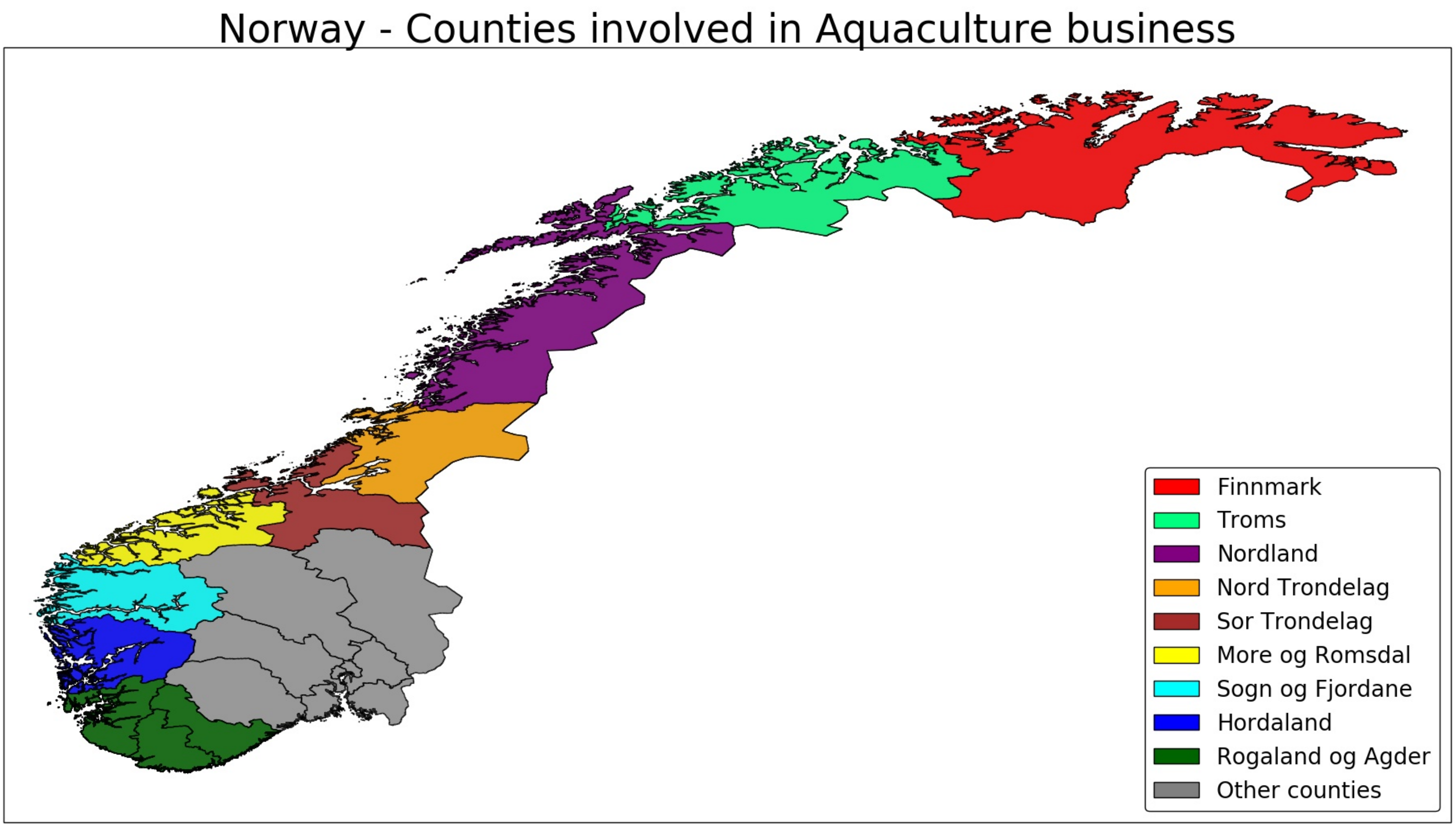



Figure 2.3: Norwegian counties involved in aquaculture business

# Part I

# Data Collection and Validation

# Chapter 3

# Data Sources and Elaboration

## 3.1 Data Sources

The collection of the data has been an important phase during this work.
Several sources have been checked and consulted in order to find reliable and useful data for the final purpose of this thesis.
In this particular case the main data collection way was internet, but some important data have been provided also from SINTEF Nord.

### 3.1.1 Data from SINTEF Nord

Some of the data used during this thesis were provided from the members of the eSushi project at SINTEF Nord. The origin source of the data is "Archieve Norstore"[1] and they are about each single norwegian county with the following details:

| **Input** | **Content** | **Unit** | **Frequency** | **Available Period** |
|---|---|---|---|---|
| 1. Average Sea Temperature | Reported number of cages with salmon and rainbow trout. | Celsius | Monthly | January 2007 - April 2014 |

Table 3.1: Data provided from SINTEF Nord.

[1]Source link for data downloaded by Archieve Norstore:
https://archive.norstore.no/pages/public/datasetDetail.jsf?id=10.11582/2015.00014

### 3.1.2 Data from Fiskeridir

Fiskeridir[2] has been the main data source for this work. It provides several statistics about Aquaculture in Norway.
The data inputs from the current website used for this thesis are reported below, and they are available for each single county in Norway involved in Aquaculture business:

| Input | Content | Unit | Frequency | Available Period |
|---|---|---|---|---|
| 1. Cages | Reported number of cages with salmon and rainbow trout. | Number | Monthly | January 2005 - April 2017 |
| 2. Localities | Reported number of localities with salmon and rainbow trout. | Number | Monthly | January 2007 - April 2017 |
| 3. Feed consumption | Reported feed consumption for Salmon. | Tonnes | Monthly | January 2007 - April 2017 |
| 4. Restock | Fish restock reported for Salmon. | 1000 pcs | Monthly | January 2007 - April 2017 |
| 5. Withdrawals | Withdrawals of Salmon for slaughter. | Tonnes | Monthly | January 2007 - April 2017 |
| 6. Biomass | Reported biomass of Salmon. | Tonnes | Monthly | January 2007 - April 2017 |
| 7. Salmon Number | Reported number of Salmon. | Number | Monthly | January 2007 - April 2017 |

Table 3.2: Data provided from Fiskeridir.

About the current data source is also important to know that:

- The data are available from 2005 to 2017.
- The data are uploaded once per month.
- The data are reported and available just in XLSX format.
- The data are available just in Norwegian.
- Is not possible to implement an automatic download script.

[2]Source link for data downloaded by Fiskeridir:
`http://www.fiskeridir.no/Akvakultur/Statistikk-akvakultur/Biomassestatistikk`

### 3.1.3 Data from Indexmundi

An another source of data for this study was Indexmundi[3]. From this particular source is possible to get data about fish (salmon) monthly price, Norwegian Krone per KG.

| Input | Content | Unit | Frequency | Available Period |
|---|---|---|---|---|
| 1. Export Salmon Price | Reported farm bred Norwegian Salmon export price. | NOK/KG | Monthly | January 2005 - April 2017 |

TABLE 3.3: Data provided from Indexmundi.

## 3.2 Increase accessibility and availability of data

In order to increase the accessibility and availability of the downloaded data, during this phase the main goals were:

- Provide an accurate description in English language, since most of the data were available just in Norwegian.
- Report the data in a standard and reusable format (CSV).
- Design and build a easily readable dataset structure.

The final decision about the datasets set up during this thesis provided the followng list of datasets, where the structure can be checked in the following two pages.

- Overview Dataset: Norway.csv
- County 1 Dataset: Finnmark
- County 2 Dataset: Troms
- County 3 Dataset: Nordland
- County 4 Dataset: Nord Trondelag
- County 5 Dataset: Sor Trondelag
- County 6 Dataset: More og Romsdal
- County 7 Dataset: Sogn og Fjordane
- County 8 Dataset: Hordaland
- County 9 Dataset: Rogaland og Agder

[3]Source link for data downloaded by Indexmundi:
`http://www.indexmundi.com/commodities/?commodity=fish`

### 3.2.1 Dataset about Norway

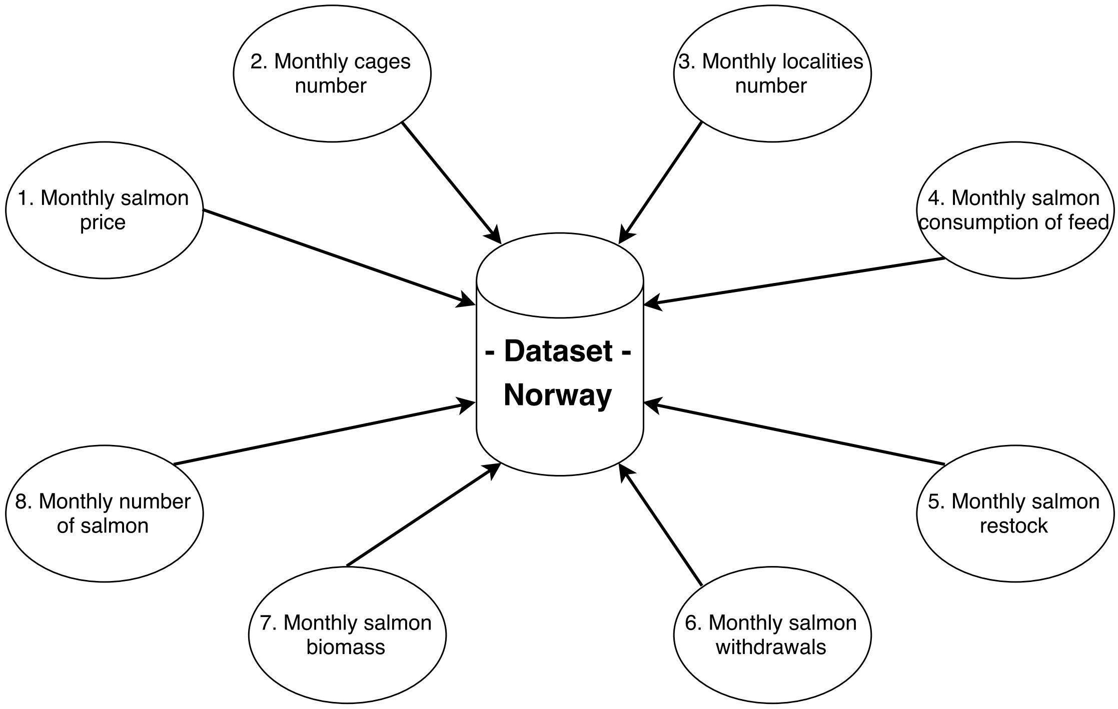



FIGURE 3.1: Dataset structure.

| **Input** | **Frequency** | **Period** | **Location** |
|---|---|---|---|
| 1. Export Salmon Price | Monthly | January 2005 - December 2016 | Norway |
| 2. Cages | Monthly | January 2005 - December 2016 | Norway |
| 3. Localities | Monthly | January 2005 - December 2016 | Norway |
| 4. Feed consumption | Monthly | January 2005 - December 2016 | Norway |
| 5. Restock | Monthly | January 2005 - December 2016 | Norway |
| 6. Withdrawals | Monthly | January 2005 - December 2016 | Norway |
| 7. Biomass | Monthly | January 2005 - December 2016 | Norway |
| 8. Salmon Number | Monthly | January 2005 - December 2016 | Norway |

TABLE 3.4: Structure of the dataset about Norway.

### 3.2.2 Dataset about single county

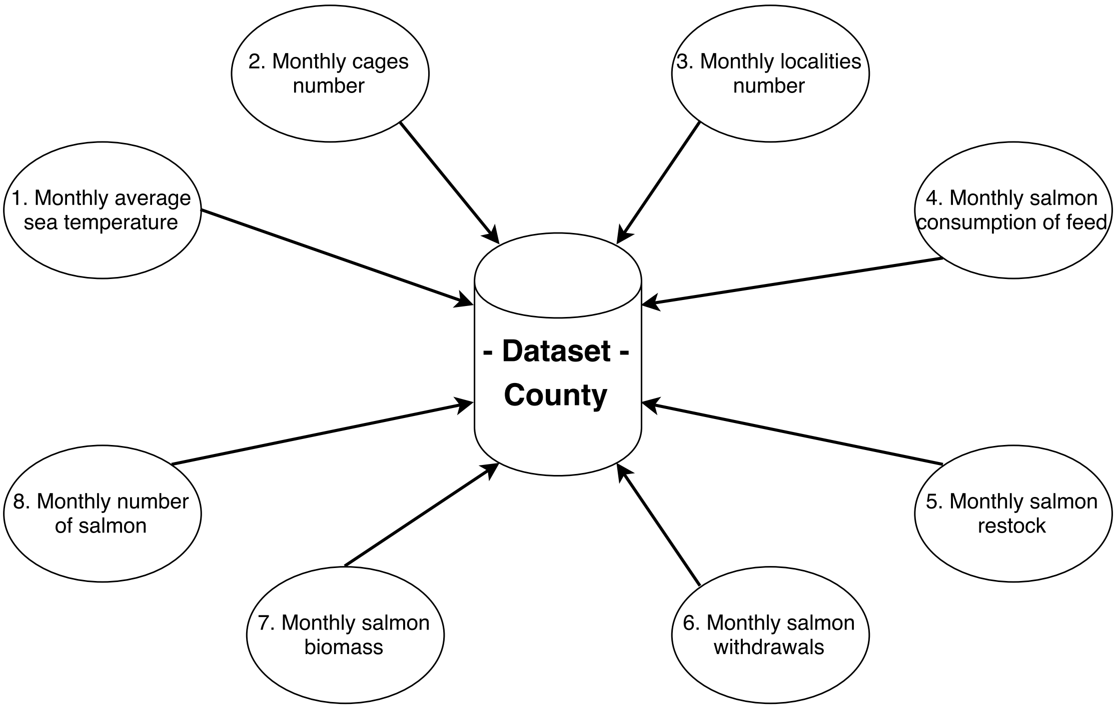



FIGURE 3.2: Dataset structure.

How is possible to check in the following table, the dataset that contains data about a single county has a shorter availability period of the values than the dataset about Norway, that is because of the sea average temperature values that have shorter range compared with the other inputs (2007-2014 instead of 2005-2016).

| **Input** | **Frequency** | **Period** | **Location** |
|---|---|---|---|
| 1. Average Sea Temperature | Monthly | January 2007 - December 2014 | Single county |
| 2. Cages | Monthly | January 2007 - December 2014 | Single county |
| 3. Localities | Monthly | January 2007 - December 2014 | Single county |
| 4. Feed consumption | Monthly | January 2007 - December 2014 | Single county |
| 5. Restock | Monthly | January 2007 - December 2014 | Single county |
| 6. Withdrawals | Monthly | January 2007 - December 2014 | Single county |
| 7. Biomass | Monthly | January 2007 - December 2014 | Single county |
| 8. Salmon Number | Monthly | January 2007 - December 2014 | Single county |

TABLE 3.5: Structure of the dataset about each norwegian county.

# Part II

# Implementation

# Chapter 4

# Implementation Design and Requirements

## 4.1 System Requirements

In the next chapters will be documented the implementation procedure for each of the system used for this study. If you want to redo the procedure or just test the final resulting system, it's important to follow the right requirements.

### 4.1.1 Requirements for reusability

Both the analysis systems that are going to be implemented during this phase of the work will need for just one requirement about the input dataset:

- Monthly frequency of data values.

### 4.1.2 System requirements

It's important to remind that this proceure will describes the system implentation using Python, so be sure to have installed all the necessary for compile and execute Python code on your platform. Current development environment:

```
Python version: 2.7.12
```

It's also necessary to have installed the following Python libraries:

- SciPy [1]
- Cartopy [2]

[1] SciPy : `https://www.scipy.org/install.html`
[2] Cartopy : `http://scitools.org.uk/cartopy/docs/latest/installing.html#installing`

## 4.2 System Design

How written above in the Thesis Structure section 1.3, the implemented Python system has been divided in different subsystems. This decision was taken because the system has to implement functions and utilities that are quite different between them, so split it in subsytems allows to maintain the reusability and increase the understanding of the implemented code. The following figure and text provide a general idea about the systems that are implemented during this study.

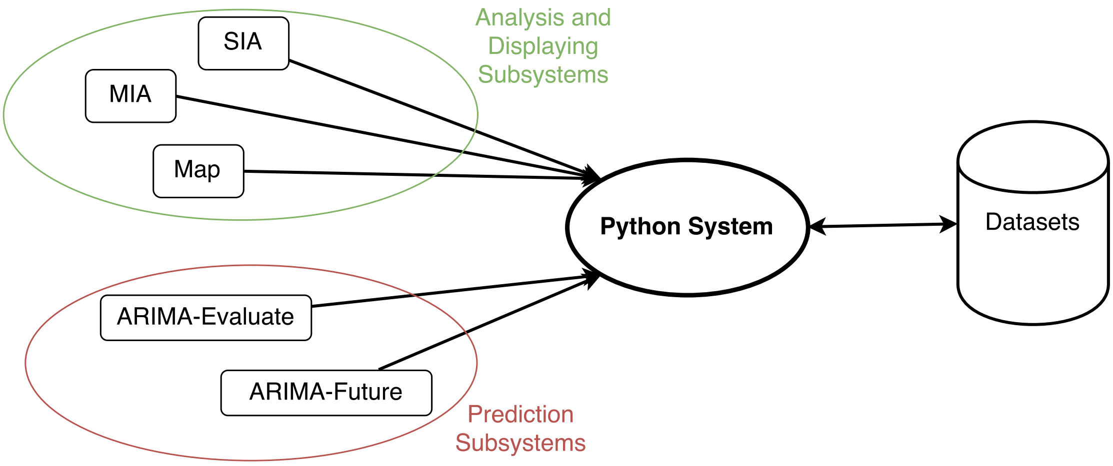



FIGURE 4.1: Subsystems overview.

- **Single Input Analyzer (SIA)**: Will provide the analysis results about a specific parameter of the input dataset.
- **Multiple Input Analyzer (MIA)**: Will provide the analysis result about all the parameters of the input dataset, such as the correlation between, comparisons,..
- **Map**: Will provide a data visualization able to display some of the data on a Norway's territory map.
- **ARIMA-Evaluate**: Will provide a system able to evaluation different configurations of an ARIMA machine about a specific paramater of the current input dataset.
- **ARIMA-Future**: Will provide a system able to get some future predictions about a parameter of the current input dataset using a specific configuration of the ARIMA machine, that should be the best one obtained during the Evaluation process.

# Chapter 5

# Analyzer System

Total implementation link for data analyzer :
`https://github.com/Sprea22/Python_Systems`

During this part the main purpose is to analyze the whole dataset in order to find some kind of useful informations later on.

The system that it's going to be implemented during this part of the work could be divided in two subsystems, with the relative outcomes:

- Single Input Analyzer (SIA): Used for analyze a single data input.
  - Total graphic of the input data for the whole period.
  - Graphic of the input data for each single year.
  - Correlation matrix between different months of the same input.
  - Correlation matrix between different years of the same input.
- Multiple Inputs Analyzer (MIA): Used for analyze multiple data inputs.
  - General correlation matrix between all the different inputs.
  - Graphic of the normalized angular coefficients of all the inputs.

## 5.1 Single Input Analyzer

It's possible to check out the total implementation code of the SIA in the appendice [A]. The implementation of this Analyzer can be divided in the following parts:

- SIA imported libraries.
- SIA part I: Generate and display a graphic about current input with total data.
- SIA part II: Generate and display a graphic about current input for each year.
- SIA part III: Generate and display a graphic that contains the correlation matrix between each single year of the current input.
- SIA part IV: Generate and display a graphic that contains the correlation matrix between each single months of the year of the current input.
- SIA part V: Generate and display a single overview image for the current input.

### 5.1.1 SIA: Imported libraries

Specific Python libraries have been imported for the implementation of this system. It's possible to find out a list of this libraries with a specific description for each of them in the appendice [A.1].

### 5.1.2 SIA section I: Total graphic for all the years

**Goal:**

Generate and display the total graphic about current input, and then calculate and display the trend line as well. Trend line angular coefficient has to be save in a document.

**Requirements:**

The current data input has to be with a monthly frequency.

**Implementation:**

To reach the current goal have been used two main functions of the "pandas" library. They allow to read the data values from the dataset and display it on a graphic.

```
series = pandas.read_csv()
seris.plot()
```

It's possible to check out the full commented implementation in the appendice: [A.3]

**Results:**

With this first part of the code has been reached the first goal of displaying and saving the basic graphic about the current input, with also the relative trend line and saving it angular coefficient in a document, that looks like:

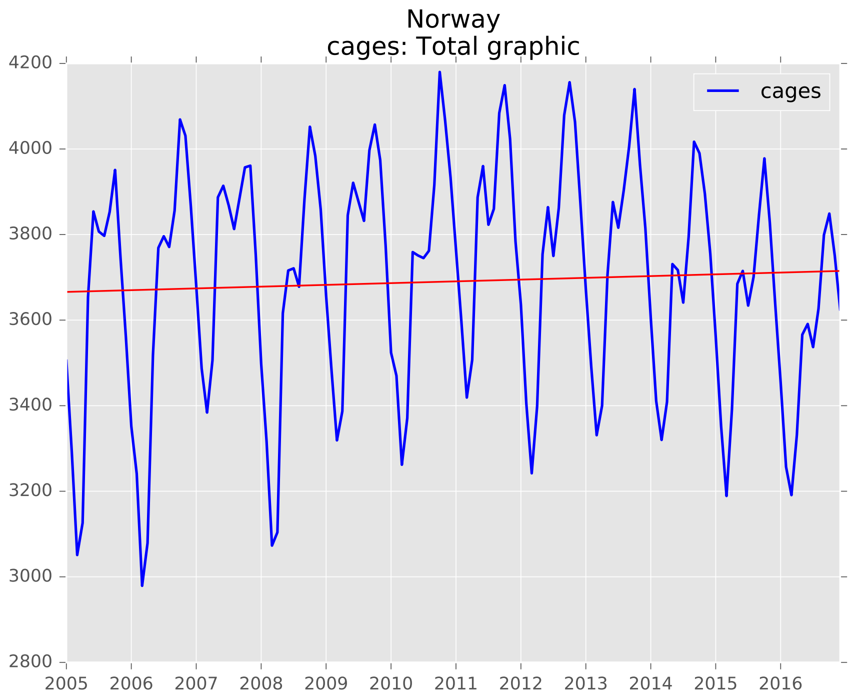



Figure 5.1: Total graphic about current input over the whole period.

### 5.1.3 SIA section II: Single graphics for each year

**Goal:**

Generate and display a graphic that contains the plots of each single year over the whole period of the current input.

**Requirements:**

The current data input has to be with a monthly frequency.

**Implementation:**

To reach the current goal have been used two main libraries.

The ”pandas” library allows to read the data values from the dataset and return it like ”ndarray” type, then the library ”pyplot” allows to display it on a graphic.

```
series = pandas.read_csv()
series.values()
pyplot.plot()
```

It’s possible to check out the full ccommented code in the appendice: [A.4]

**Results:**

With this second part of the code has been reached the goal of displaying and saving the graphic of the plots for each single year of the current input, that looks like:

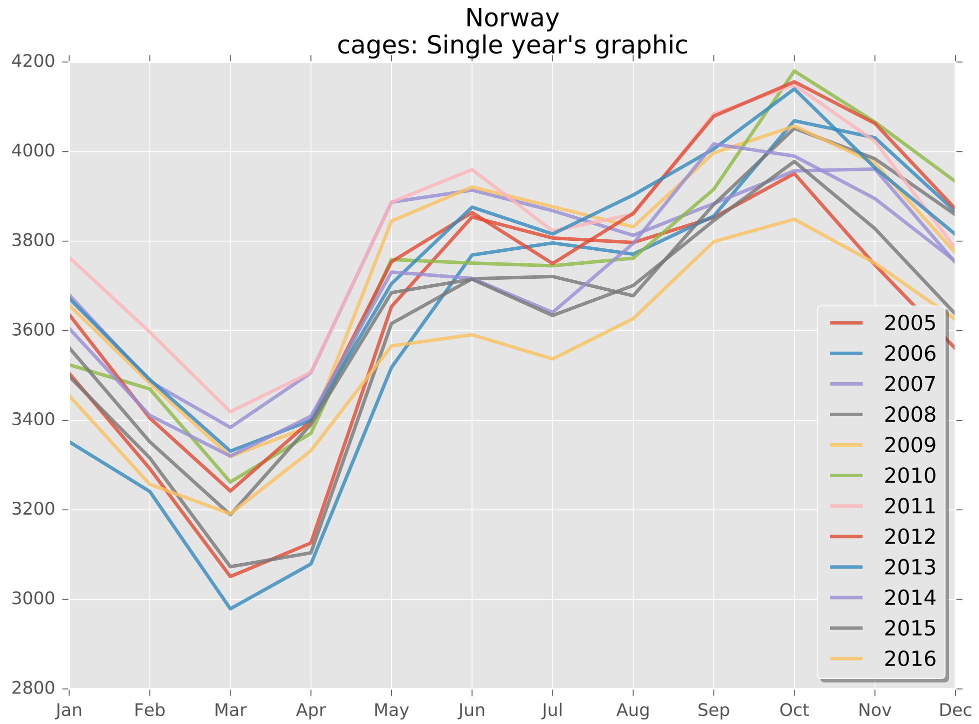



FIGURE 5.2: Graphics for each single year of the current input data.

### 5.1.4 SIA section III: Correlation matrix between years

**Goal:**

Calculate and save the correlation coefficients between each single year over the whole period of the current input and then display it with a correlation matrix.

**Requirements:**

The current data input has to be with a monthly frequency.

**Implementation:**

To reach the current goal have been used the scientific computing library ”numpy”, that allows to calculate the correlation coefficients between data. Then the library ”pyplot” has been used to display the results on a matrix.

```
numpy.corrcoef()
figure = pyplot.figure()
ax = figure.add_subplot()
ax.matshow()
```

It’s possible to check out the full ccommented code in the appendice: [A.5]

**Results:**

With this part of the code have been calculated and displayed the correlation coefficients between each single year of the current input, that looks like:

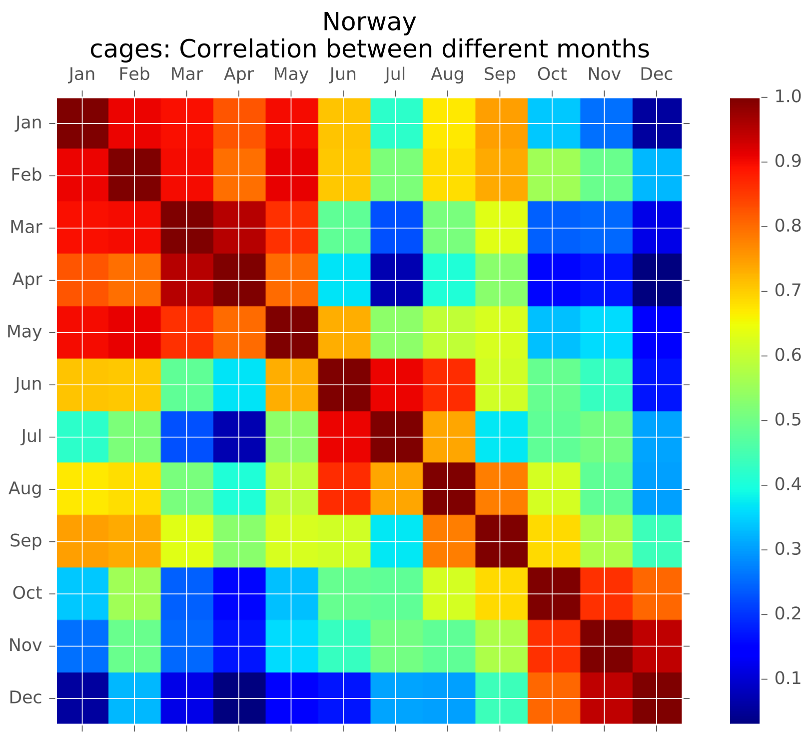



FIGURE 5.3: Correlation matrix between different months of the same input

### 5.1.5 SIA section IV: Correlation matrix between months

**Goal:**

Calculate and save the correlation coefficients between each single month of the current input and then display it with a correlation matrix.

**Requirements:**

The current data input has to be with a monthly frequency.

**Implementation:**

To reach the current goal have been used the scientific computing library ”numpy”, that allows to calculate the correlation coefficients between data. Then the library ”pyplot” has been used to display the results on a matrix.

```
numpy.corrcoef()
figure = pyplot.figure()
ax = figure.add_subplot()
ax.matshow()
```

It’s possible to check out the full ccommented code in the appendice: [A.6]

**Results:**

With this part of the code have been calculated and displayed the correlation coefficients between each single month of the current input, that looks like:

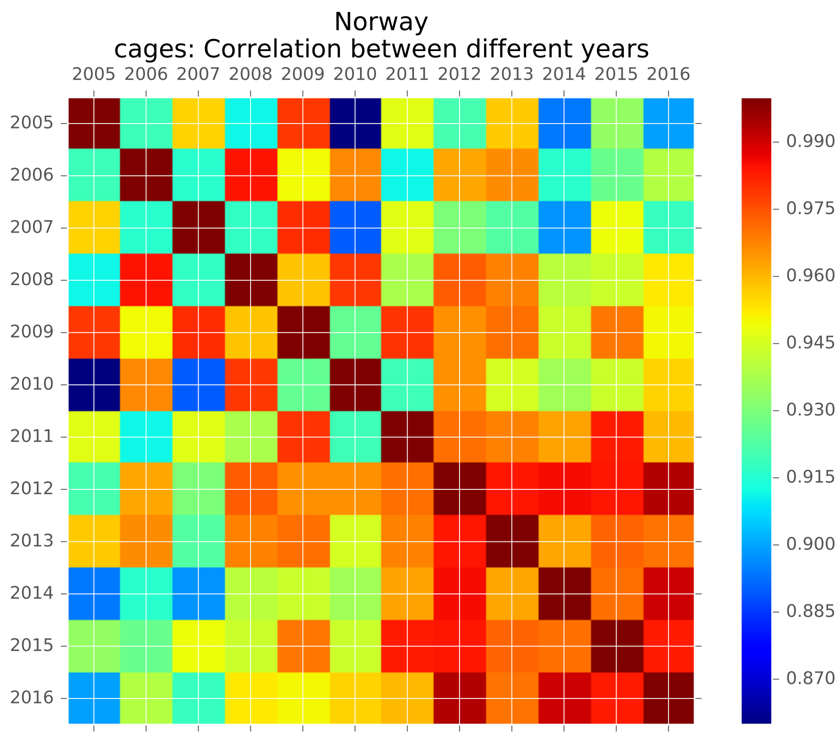



Figure 5.4: Correlation matrix between different years of the same input

### 5.1.6 SIA section V: Single overview

**Goal:**

Generate and display a single overview image that contains all the graphics previous calculated about the current input.

**Requirements:**

All the graphics about the current input have to be already calculated and saved.

**Implementation:**

During this part of the implemented system has been indispensable the Python Imaging Library, called also PIL.

```
from PIL import Image
```

It basically allowed to create a new "empty" image and then create a sort of collage pasting the already calculated graphic's images on it.

```
new_im = Image.new()
new_im.paste()
```

The following method contains the full code that allows to create the overview image.

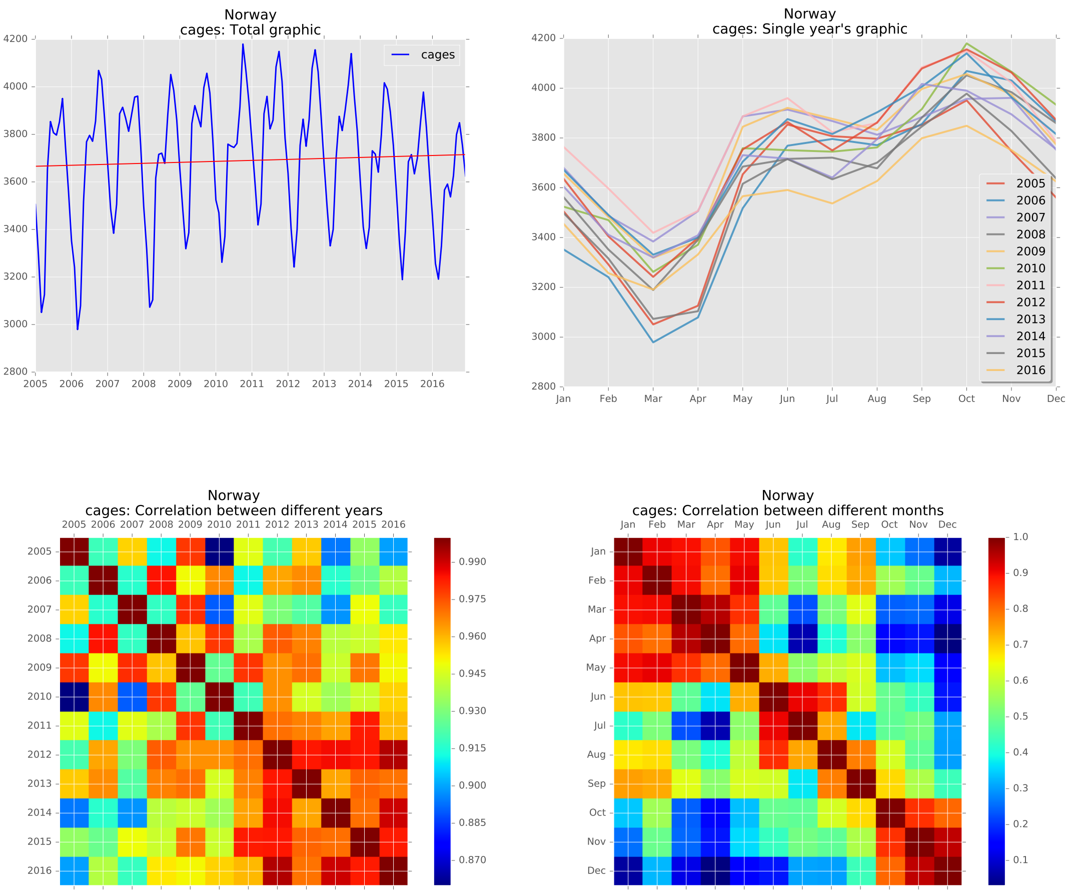
```
def create_single_overview(cols, rows, dest, width, height, listofimages):
```

The output of this phase depends by the input to this method, that are basically the list of image and the preferences about the collage's structure.
Is possible to view the final result of this phase in the next page and is possible to check out the full ccommented code in the appendice: [A.7]

**Results:**

With this part of the code it's possible to have a single overview image about the current input, that basically allows to compare all the graphics already calculated about this input. The general overview graphic contains:

- Total graphic of the input data for the whole period.
- Graphic of the input data for each single year.
- Correlation matrix between different months of the same input.
- Correlation matrix between different years of the same input.



## 5.2 Multiple Inputs Analyzer

The implementation of this Analyzer can be divided in the following parts:

- MIA imported libraries.
- MIA part I: Calculate the correlation coefficients between the different input of a dataset, save the result and display it in a matrix.
- MIA part II: Display the comparison graphic between the different input's trend line normalized angular coefficients.

It's possible to check out the total implementation of the MIA in the appendice [B].

### 5.2.1 MIA: Imported libraries

Specific Python libraries have been imported for the implementation of this system. It's possible to find out a list of this libraries with a specific description for each of them in the appendice [B.1].

### 5.2.2 MIA section I: Total Correlation Coefficients

**Goal:**

Calculate and save the correlation coefficients between different inputs of the current dataset and then show it with a matrix.

**Requirements:**

To let the MIA system works in a proper way, is necessary that the current dataset has been already analyzed from the SIA system.

**Implementation:**

To reach the current goal have been used the scientific computing library "numpy", that allows to calculate the correlation coefficients between data. Then the library "pyplot" has been used to display the results on a matrix.

```
numpy.corrcoef()
figure = pyplot.figure()
ax = figure.add_subplot()
ax.matshow()
```

It's possible to check out the full ccommented code in the appendice: [B.2]

**Results:**

This part of the MIA implementation allows to calculate the correlation coefficients value between each single inputs and then also to display and save it. It looks like:

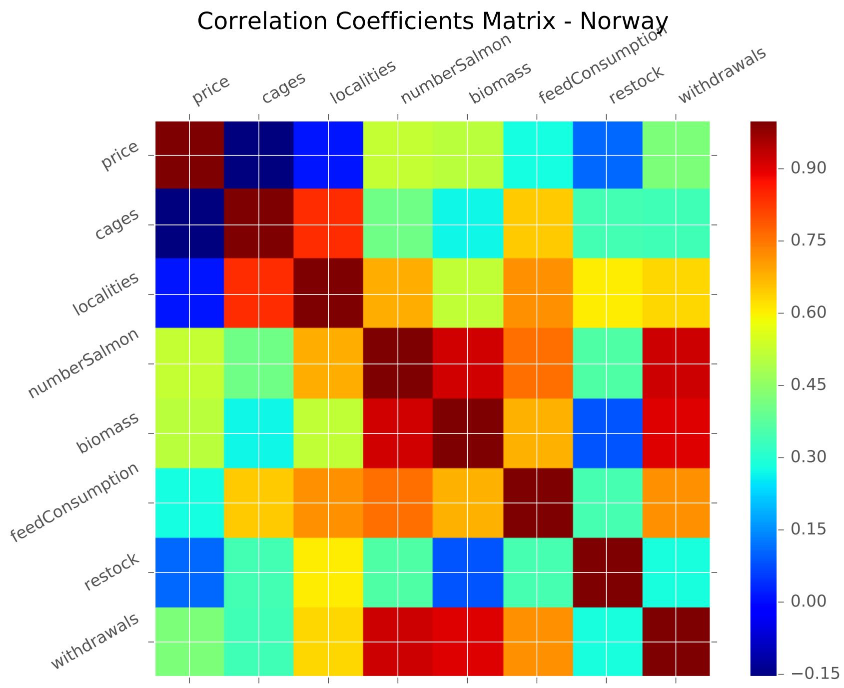



FIGURE 5.7: Correlation matrix between different inputs with data.

### 5.2.3 MIA section II: Normalized Angular Coefficients

**Goal:**

Display the comparison graphic between the normalized angular coefficient of each input trend line.

**Requirements:**

To let the MIA system works in a proper way, is necessary that the current dataset has been already analyzed from the SIA system.

**Implementation:**

Also to reach this goal have been used the two libraries "pandas" and "pyplot". The first one allows us to read the values that the library "pyplot" will display, in this case in a histogram.

```
pandas.read_csv()
pyplot.barh()
```

It's possible to check out the full ccommented code in the appendice: [B.3]

**Results:**

This part of the MIA implementation allows to display a graphic that compare the normalized angular coefficients for each single input that have been already calculated and reported in a document. The result graphic look like:

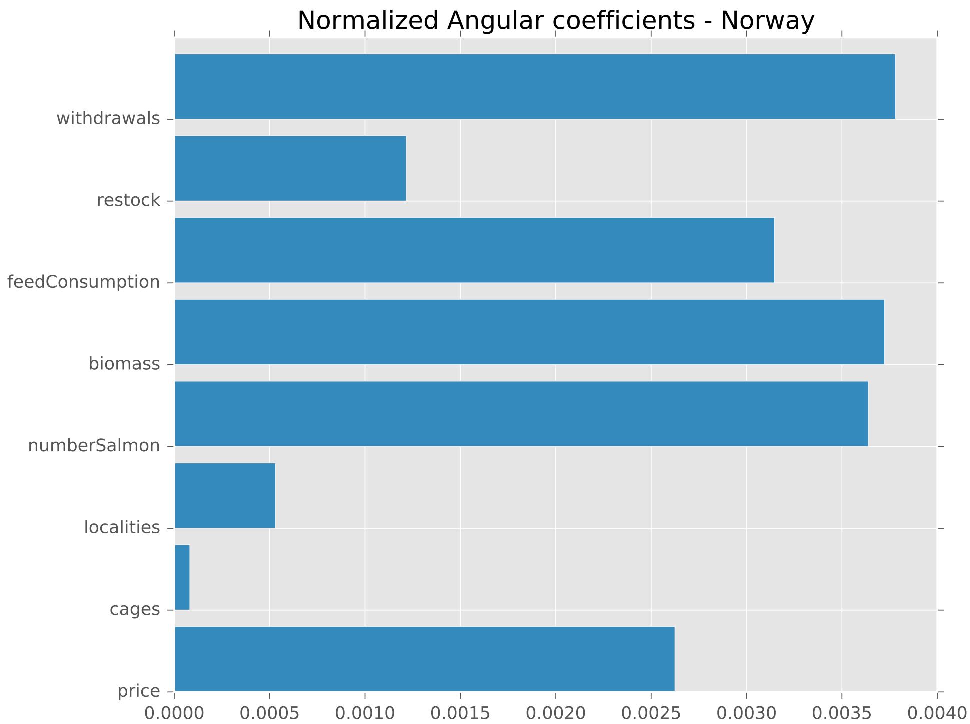



FIGURE 5.8: Normalized angular coefficients of each input's trendline.

## 5.3 Data Displaying on a map

**Goal:**

The main goal of this phase is to find a way to visualize some data values on a map graphic using Python. In this particular case the map graphic has to represents the Norway territory and its every single county.

**Requirements:**

This displaying system was implemented just for displaying data about Norway, that means it's not reusable for other input datasets.

During this work has been created a specific dataset for test the system works. It contains the average value of a specific input about a single county on the whole available period. The following table shows some examples about the dataset structure: for each county has been calculated the average value from 2007 to 2014 of different parameters.

| **county** | **averageSeaTemp** | **cages** | **localities** | **...** | **feedConsumption/biomass** |
|---|---|---|---|---|---|
| Finnmark | 5.2128134819 | 257.2395833333 | 33.8333333333 | ... | 0.1611964666 |
| Troms | 6.2185416667 | 393.3958333333 | 52.1666666667 | ... | 0.1831404686 |
| Nordland | 6.8333444959 | 804.5104166667 | 109.0208333333 | ... | 0.1849358645 |
| Nord-Trondelag | 7.322600258 | 231.6875 | 30.3645833333 | ... | 0.1852350478 |
| Sor-Trondelag | 7.5381376237 | 306.9479166667 | 51.3645833333 | ... | 0.1862036956 |
| More_og_Romsdal | 8.0087820154 | 347.3229166667 | 59.5729166667 | ... | 0.1831662176 |
| Sogn_og_Fjordane | 8.1081250683 | 318.9583333333 | 52.5 | ... | 0.1863151035 |
| Hordaland | 7.8033025443 | 738.8854166667 | 131.1770833333 | ... | 0.1925203347 |
| Rogaland_og_Agder | 7.1951075619 | 338.53125 | 53.0416666667 | ... | 0.1840209916 |

**Implementation:**

The library "cartopy", that basically provdes cartographic tools for Python. More specifically, the most useful classes used during this part of the work have been "cartopy.io.shapereader", that allows to read the file extension ".shp"[1], and "cartopy.crs", that allows to use several projections with the same interface.

It's possible to check out more details about the needed libraries in the appendice: [C.1]

```python
import cartopy.crs as ccrs
import cartopy.io.shapereader as shpreader
shpreader.Reader(filename).geometries()
```

[1]See the definition of Shapefile:
`https://en.wikipedia.org/wiki/Shapefile`

Then the input shapely geometries were displayed to the axes using the "matplotlib".

```
plt.figure()
ax = plt.axes()
axes.add_geometries
```

Once displayed the geometries on the map, is possible to set their colors based on some input values with the library "matplotlib".

```
plt.get_cmap
matplotlib.colors.Normalize
```

It's possible to check out the full ccommented code in the appendice: [C]

**Results:**

During this implementation was implemented a cartographic representation of some parameters about each single county involved in the Norwegian aquaculture business, but is possible to use the reported library to implement a system about an another territory or an another country.

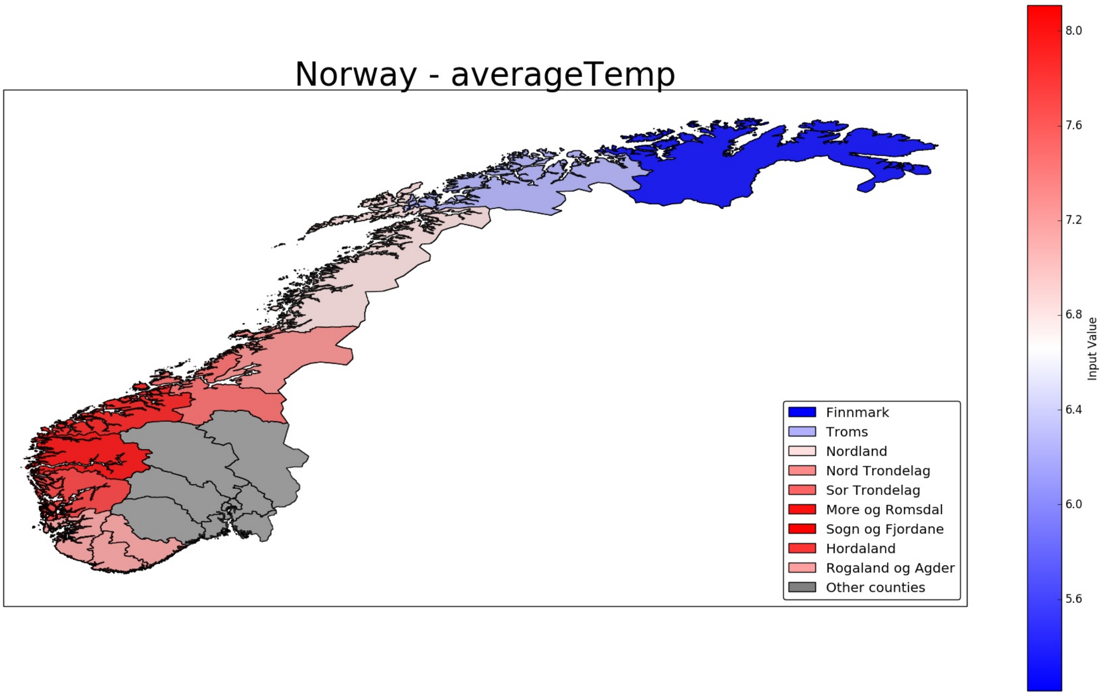



Figure 5.9: Average Sea Temperature from 2007 to 2014 in Norway.

# Chapter 6

# Prediction System

The main goal during this phase was to find a way to implement a forecast system in Python. Since the datasets used during this thesis could be considered as a time series, was decide to implement and test an Autoregressive Integrated Moving Average (ARIMA) model.
Since there are several possible configurations for fit an ARIMA model, is important to find the right one to use with each input dataset because it would allows to have much better prediction results. In order to find the best ARIMA configuration there are different methods and procedure, like one of the most known that is "Box–Jenkins method"[1]. In this study was decied to use an easier method, in order to have a first approach with this system and a general idea about the problem. It basically consists in testing different ARIMA model configurations for the same input dataset and then check the results.
For this reason during this phase of the work have been implemented 2 different subsytems for two different purposes:

1. Evaluating System
2. Prediction System

[1]Check out the Box–Jenkins method at the current link:
`https://en.wikipedia.org/wiki/Box%E2%80%93Jenkins_method`

## 6.1 Evaluating System

**Goal:**

Used for evaluate different configurations of ARIMA machine.
It tests 112 different configurations for the current input that we would like to forecast and report the results with each MAPE (Mean Average Percentage Error) values.

**Requirements:**

There are not any kind of needed requirements. It's possible to use this system on dataset of arbitrary length.

**Code implementation:**

This time the full commented code has not been reported in the appendice since is longer and more complicated than the previous. If you are interested to check out more details about the code, is possible to find the Github repository here : [1.5]

The most important part of the code about the Evaluating System is the following. Basically the method ARIMA() allows to train a model based on historic values (history) and a specific order (p,d,q). After that it's possible to call the method forecast() through the trained model and having some predictions like result.

```
model = ARIMA(history, order=arima_order)
model_fit = model.fit(disp=0)
yhat = model_fit.forecast()[0]
```

More in the specific, the 112 different ARIMA configurations that were tested are all the possible combinations between the following three parameters values:

```
p_values = [0, 1, 2, 4, 6, 8, 10]
d_values = [0, 1, 2, 3]
q_values = [0, 1, 2, 3]
```

**Results:**

The system will display the MAPE between real value and predicted values for each of the 112 tested ARIMA machine. In particular, once tested all the configurations, the system will provide the configuration that gave the best MAPE result.
All these results have been reported in a document that is possible to check for check out the different configurations result.

The following graphic display the different ARIMA configurations tested, providing also:

- General overview about MAPE values for each single tested configuratons.
- Best configuration with relative MAPE value.
- Color legend, where the red means lower MAPE, so more accurate predictions, and blue means higher MAPE, so less accurate predictions.

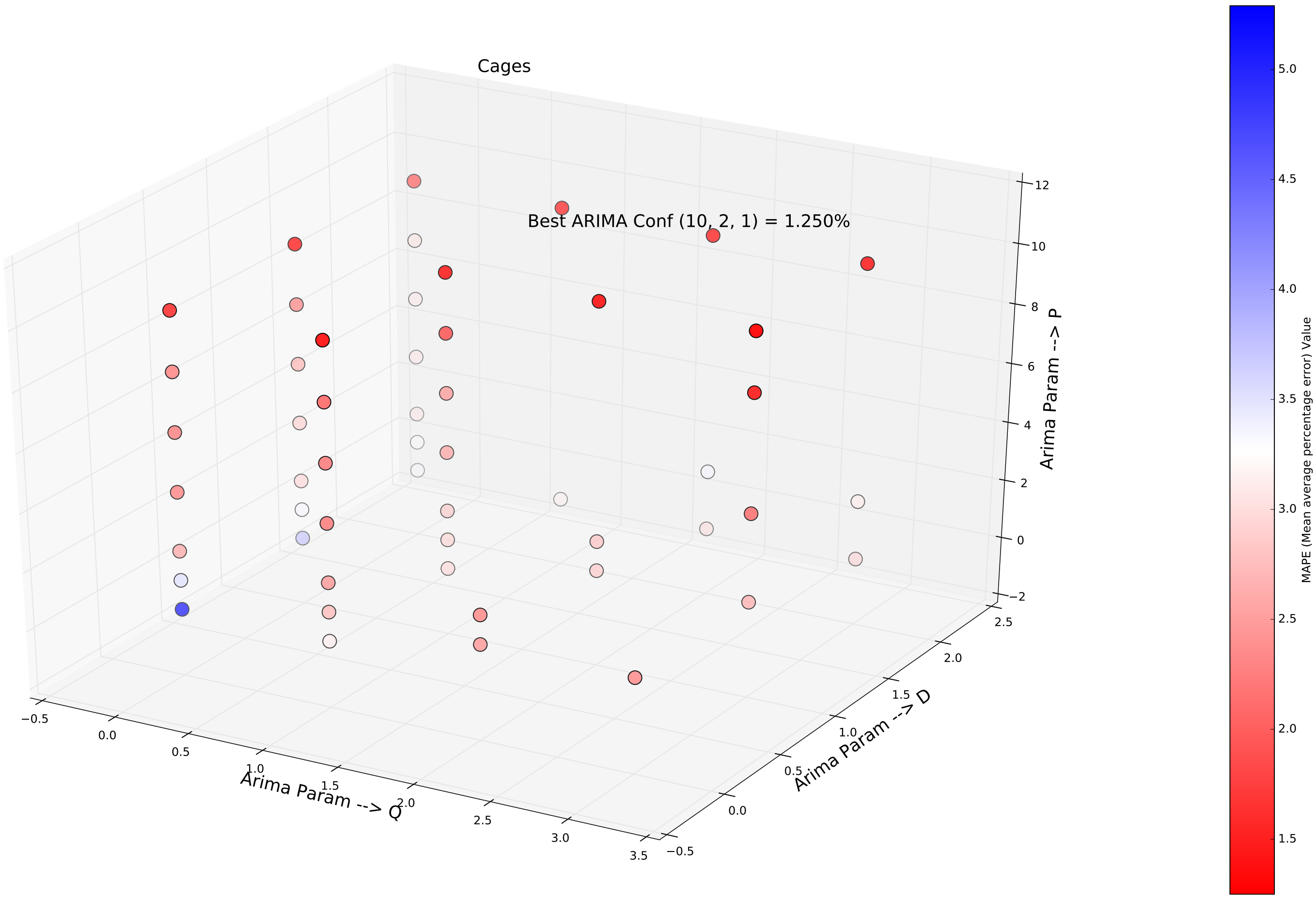



Figure 6.1: Graphic that displays different MAPE values for each ARIMA order.

## 6.2 Prediction System

**Goal:**

This system has three main goals:

- Testing a specific ARIMA configuration on a particular data input, and display how much accurate it is (MAPE).
- Predict some future value with the same ARIMA configuration.
- Display the historic data together with the testing and future predictions.

**Requirements:**

There are not any kind of needed requirements. It's possible to use this system on input dataset of arbitrary length.

**Code implementation:**

To reach the first of the goals reported above the system will divide the input dataset in two parts, train and test. It allows to train the ARIMA model with just the "train" part of the dataset, that usually is 66% of the whole dataset, and then try to predict the rest of the dataset values, comparing in the end with the values contain in the "test" part to have a general idea about the accuracy.

The method ARIMA() allows to train a model based on historic values (history) and a specific order (p,d,q). After that it's possible to call the method forecast() through the trained model and having some predictions like result.

```
model = ARIMA(history, order=arima_order)
model_fit = model.fit(disp=0)
yhat = model_fit.forecast()[0]
```

Then the system will also predict a number of future values choosen by the system user.

```
model = ARIMA(dataset, order=order)
model_fit = model.fit(disp=0)
forecast = model_fit.forecast(int(sys.argv[3]))[0]
```

The final step is to display the historic data together with the test prediction and the future prediction on the same graphic.

```
# Plot current input's historic values
series.plot(color="blue", linewidth=1.5, label="Series: "+sys.argv[1])

# Plot current input's test prediction
predHistoric.plot(color="red", linewidth=1.5, label="Prediction test:")

# Plot current input's future prediction
predFuture.plot(color="green", linewidth=1.5, label="Future Prediction:")
```

This time the full commented code has not been reported in the appendice since is longer and more complicated than the previous. If you are interested to check out more details about the code, is possible to find the Github repository here : [1.5]

**Results:**
This system will automatically generate two documents that contain:

- Test predictions values
- Future predictions values

And then it provides also the possibility to visualize the historic, test and future predictions values on the same graphic, that looks like the following example:

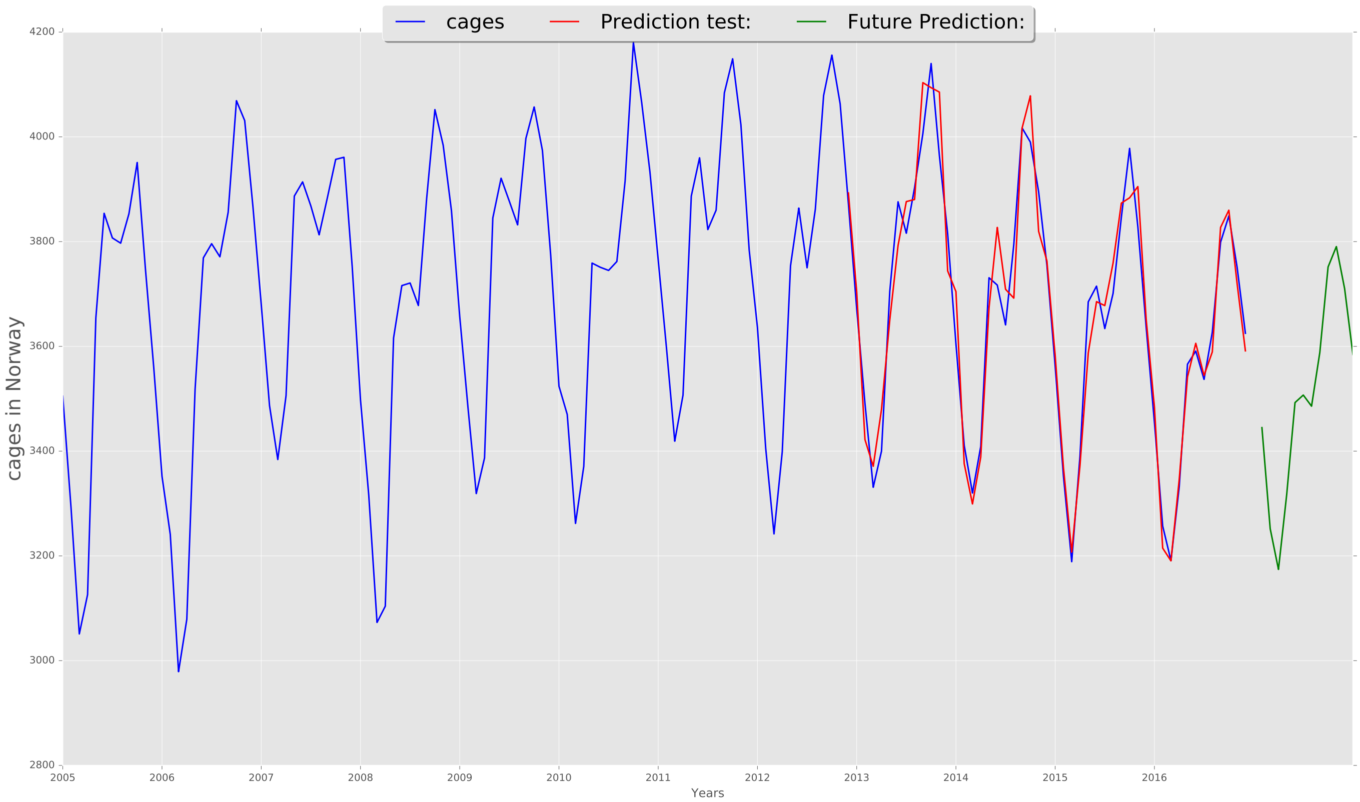



FIGURE 6.2: Graphic that display historic, future and predicted values of a input.

# Part III

# Results, Discussion, and Conclusions

# Chapter 7

# Results Overview

This study provides several results, that are basically the implemented systems themselves and the generated evidences.
Since the big amount of evidences made during this work, below here is reported an overview of the final results. If you're interested to check out the whole results and even use it, it's possible to find it on the already citated Github repository. [1]

Further more you can find the Datasets collection and all the implemented python systems file.

**Results** - Contains several analysis results, basically all the outputs of SIA and MIA systems. In particular
Dataset X folder: A general folder that contains: - Correlation coeff between all the inputs - Document that contains the corr coeff values - Comparison graphic of the trend line normalized angular coefficients of all the inputs - Folder with all the inputs overview
For each paramater of the current dataset is reported a folder that contains:
- Document that contains trend line angular coefficeints and trend line normalized angular coefficient.
- Document that contains months correlation coefficients values
- Document that contains years correlation coefficients values
- Total graphic
- Years graphic
- Corr matrix between years
- Corr matrix between month
- Overview

[1] System repository: https://github.com/Sprea22/Python_Systems

**Results_Forecast** - Contains several results about the forecasting system. Dataset X folder: For each paramater of the current dataset is reported a document that contains the evaluation results, so MAPE values for each ARIMA configuration tested.

**Results_Maps** - Contains several results about the map system. In particular different cartographic map about Norway with different average inputs on a range of time between 2007 and 2014.

# Chapter 8

# Discussion and Evaluations

The present study investigates about the possibility of testing the Data Science process using Python, trying to apply it to the Norwegian salmon farming industry, in order to gather and let be available as many useful results as possible about.

---

Justify your approach

---

Is possible to say that the results obtained are providing an answer to most of the initial objectives of this thesis.
In particular, this study shows that is actually possible to have a first approach to the Data Science field using Python and all the considerations reported during each step of the implementation are showing an overview of the Data Science process.

Further more, all the output results, such as graphic or coefficients value, are showing which kind of modules and packages Python provides in order to solve data analysis problems that allow to understand better Python potential in this field.

Since the implemented system is has a high reusability and high automatization level, is possible to use it in order to make the same initial data analysis, displayig and forecast in a really short time about different datasets composed by data coming from every kind of area of interest.

The results about the Norwegian salmon farming are actually not reporting any kind of new interesting informations. That because this thesis has been based mainly on the Data Science process and Python using more than a concrete information extraction.

Some useful results are anyway provided, such as a structured dataset, more accessable and readable, together with descriptions of the data that could be helpful in the future for more specific analysis in this area. Further more, is possible to check out clear and readable graphics about several parametrs of each single county of Norway that could be used for an initial view analysis. Also coefficients values are already calculated, and an initial forecast system allows to have a general idea about the future expectations in this area.

## 8.1 Forecast of feed consumption values

The forecasted values of feed consumption in Norway are not that accurated cause of the big difference between north and south about that parameter.

- Strong relation between average sea temperature and feedConsumption/biomass.

- Strong relation between normalized values of average sea temperature and feedConsumption

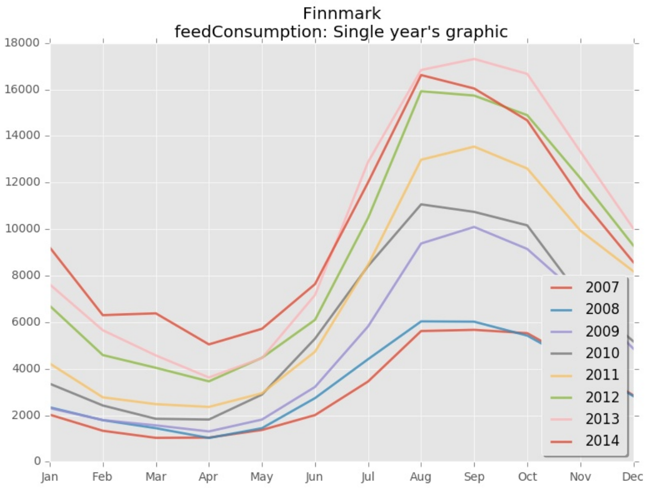



FIGURE 8.1: Annual consumption of feed trend in Finnmark.

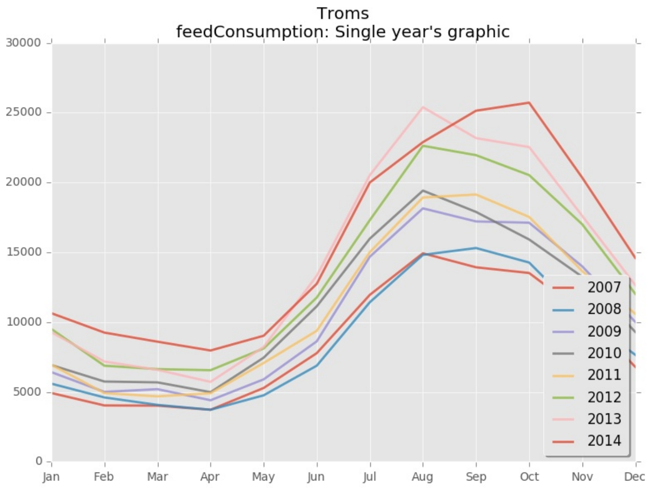



FIGURE 8.2: Annual consumption of feed trend in Troms.

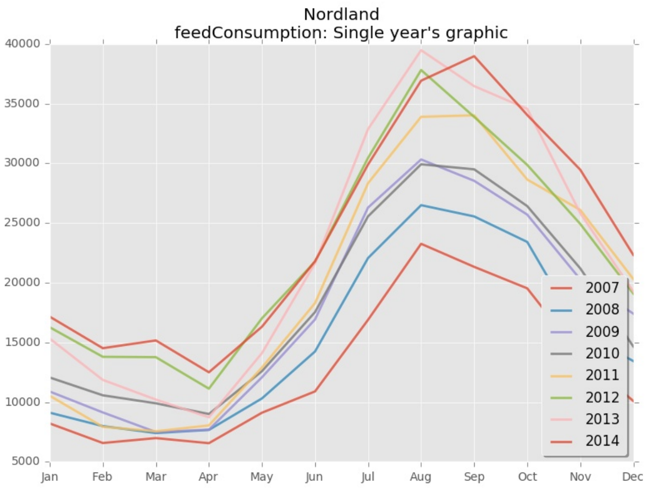



FIGURE 8.3: Annual consumption of feed trend in Nordland.

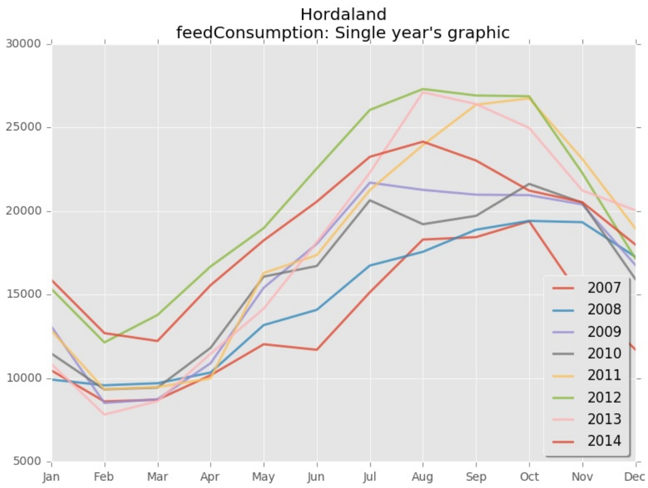



FIGURE 8.4: Annual consumption of feed trend in Hordaland.

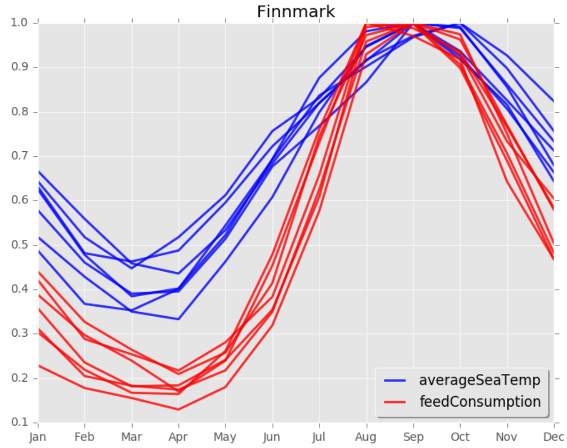



FIGURE 8.5: Comparison between average sea temperature and feed consumption in Finnmark

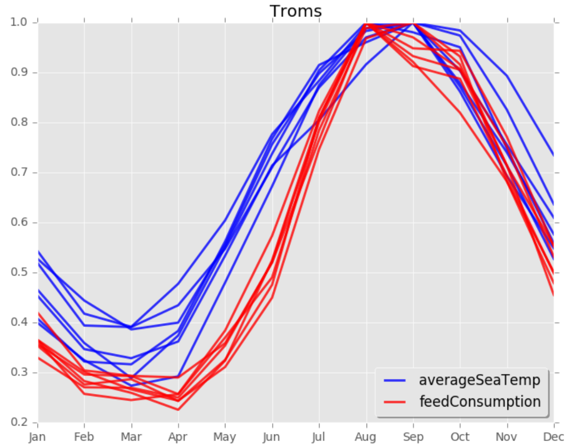



FIGURE 8.6: Comparison between average sea temperature and feed consumption in Troms

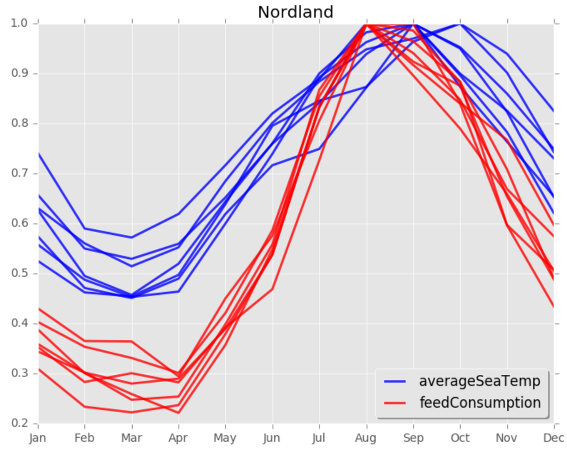



FIGURE 8.7: Comparison between average sea temperature and feed consumption in Nordland

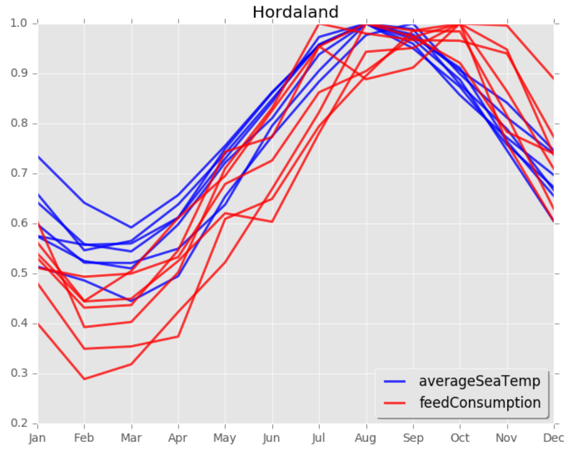



FIGURE 8.8: Comparison between average sea temperature and feed consumption in Hordaland

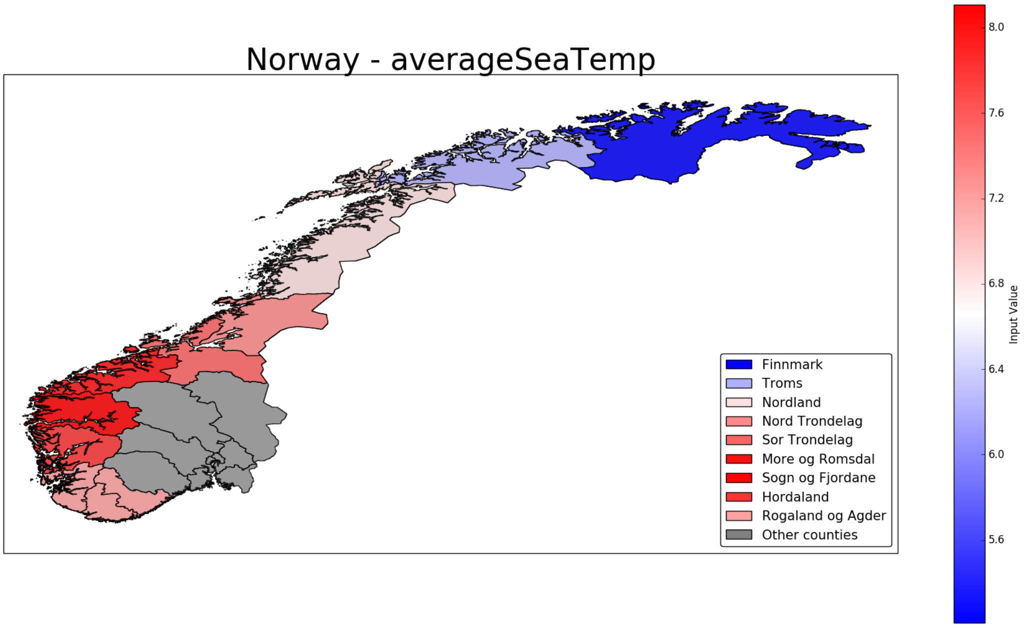



FIGURE 8.9: Monthly average sea temperature from 200

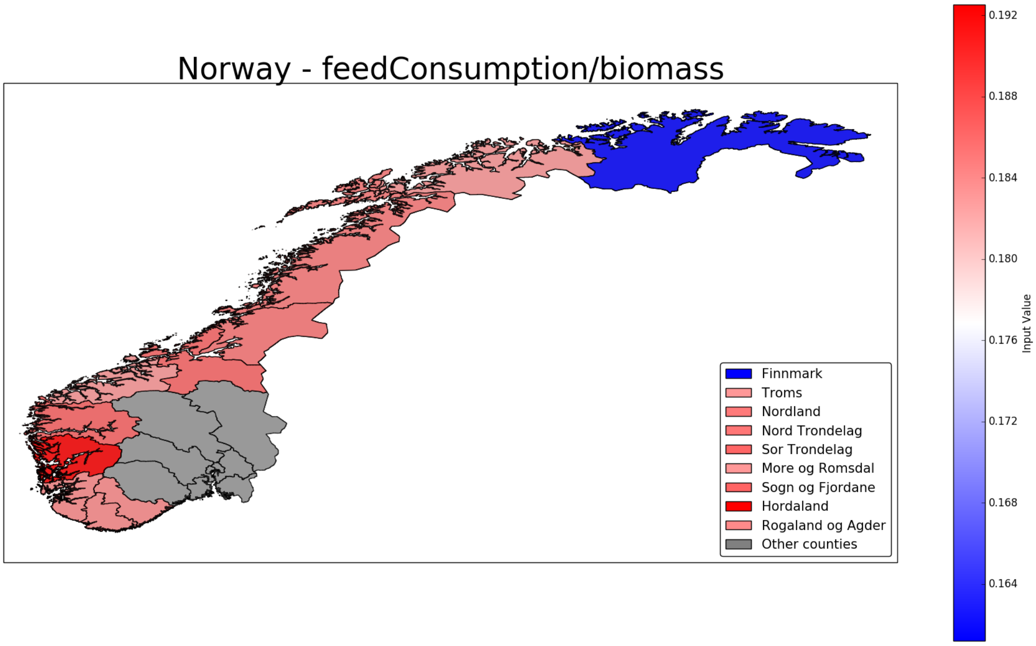



FIGURE 8.10: Data science concept

## 8.2 Evaluation and limitations of the study

This study has a number of possible limitations, mainly due to a lack of background knowledge and experience about the current area, and also because of the relatively short time available.

ABOUT PYTHON RESULTS:

- PRO: The implemented Python system has a quite high reusability. It means that is possible to use it also with dataset filled with data parameter coming from other area of interest.
- CONS: The implemented system in this thesis is probably efficient and productive for a personal use. That's because there is no GUI implemented and to change the customization of the output graphics you have to know Python language.
- CONS: Not enough informations about machine learning field, so during this thesis was implemented just a basic implementation of the forecasting system, just to give an idea about how it works and give the possibility to improve it in future works.

ABOUT DATA ANALYSIS RESULTS:

- PRO: Well structured datasets provided
- PRO: Usability of the results
- CONS: Since the data sources are public, it's not possible to know the real reliability of it.
- CONS: The data collected are allowing just limitated analysis.
- CONS: Not enough informations have been extracted from the data cause of a lack in the background theory about Norwegian salmon farming, and not enough time to document myself in a proper way.

# Chapter 9

# Conclusion

## 9.1 Summary

During this study different Python utilities have been tested to show their potential in the Data Science field.

## 9.2 Recommendations to future work

In this section are reported some ideas for future work and some extra implementation that have not been implemented during this thesis due to the time limitation. Some of them could be considered interesting for future reserach or analysis.

- **Improve the dataset content**
  The data collection that has done during this thesis provides just public data about territorial statistics. Would be interesting to test the same system with data coming from single reality, like for example in this case gather data from a single locality of salmon farming and then run the system on it.
- **Visualization of the data**
  Also if the library used during this study allow a quite good visualization of the data, it would be useful to check out other ways to realize it, which could imply the use of Python or not. For instance, if you still want to use Python could be possible to check out other libraries, such as "Plotly Python Library".
- **Improve the prediction system**
  This is the part which has much more possible future works. Forecasting system development is today a fundamental issue that involves different area of studies. To achieve an accurate model and significant results it needs much more specific research than the one reported in this study, that was just for report a general idea about it. I strongly recommend to future work to research about Artificial intelligence methods, such as "Artificial Neural Networks".
- **System as a service**
  Would be really interesting to investigate about a possible way toprovide this kind of analysis, displaying and forecasting systems like a service.

# Part IV

# Full Code Implementation

# Appendix A

# SIA Implementation code

## A.1 SIA: Imported libraries

The library "os" is really important since provides a waay of using operating system dependent functinality.

```
import os
```

Also the library "sys" would be very useful for test and execute the program, mainly because it allows to input directly from terminal.

```
import sys
```

The "pandas" library will be very useful for read the data from CSV dataset and setup the plot abut it.

```
import pandas as pd
```

The "numpy" library it's used for mathematic purpose, such as calculating the correlation coefficent between two series.

```
import numpy as np
```

The "pyplot" library it's used for basic graphic displaying and customization, easy to use but very efficent.

```
import matplotlib.pyplot as pyplot
```

The library "PIL" supports many file formats, and provides powerful image processing and graphics capabilities.

```
from PIL import Image
```

## A.2 SIA: Implemented methods

This pyplot style configuration allows to customize the graphic desig. In this case was used ggplot, a popular plotting package.

```
pyplot.style.use('ggplot')
```

The two methods reported below here were used for calculating the trend line angular coefficients and the normalized one. In order to reach this goal is used the python library "numpy".

```
def trendline(x, y, col):
    z = np.polyfit(x, y, 1)
    p = np.poly1d(z)
    pyplot.plot(x,p(x), c=col)
    z2 = trendlineNorm(x, normalization(y))
    return z[0], z2

def trendlineNorm(x, y):
    z = np.polyfit(x, y, 1)
    return z[0]
```

The following method was used for normalize the input values, that means adjusting values measured on different scales to a notionally common scale , in this case (0,1).

```
def normalization(values):
    column = list(float(a) for a in range(0, 0))
    val = np.array(values)
    val.astype(float)
    column = val / val.max()
    return column
```

The following code shows the code of the two methods that are allowing to save images and matrix values to the library "os".

```
def saveFigure(descr):
     script_dir = os.path.dirname(__file__)
     results_dir = os.path.join(script_dir, "Results/" + sys.argv[1] + "/" +
      sys.argv[2]+"/")
     if not os.path.isdir(results_dir):
         os.makedirs(results_dir)

def saveMatrix(corrRes, dest):
    mat = np.matrix(corrRes)
    dataframe = pd.DataFrame(data=mat.astype(float))
    dataframe.to_csv(dest, sep=',', header=False, float_format='%.2f', index
     =False)
```

The following code represents the method that was used for creating the overview image generated by the SIA system, that is basically a collage of all the generated graphics about a particular parameter of the current input dataset. In order to reach this goal has been strongly used the library "PIL", that allows image elaboration using Python, and the libary "OS", for saving the results.

```
def create_single_overview(cols, rows, dest, width, height, listofimages):
    thumbnail_width = width//cols
    thumbnail_height = height//rows
    size = thumbnail_width, thumbnail_height
    new_im = Image.new('RGB', (width, height))
    ims = []
    for p in listofimages:
        im = Image.open(p)
        im.thumbnail(size)
        ims.append(im)
    i = 0
    x = 0
    y = 0
    for col in range(cols):
        for row in range(rows):
            new_im.paste(ims[i], (x, y))
            i += 1
            y += thumbnail_height
        x += thumbnail_width
        y = 0
    if dest==0:
      script_dir = os.path.dirname(__file__)
      results_dir = os.path.join(script_dir, "Results/" + sys.argv[1]+"/"+
sys.argv[2]+"/")
      if not os.path.isdir(results_dir):
         os.makedirs(results_dir)
        new_im.save(results_dir+"/"+ sys.argv[1] +"_"+sys.argv[2]+"
_Graphics_Overview.jpg")
        new_im.show()
    if dest==1:
      script_dir2 = os.path.dirname(__file__)
      results_dir2 = os.path.join(script_dir2, "Results/" + sys.argv[1]+"/
Total_Evidences/Single_Inputs")
      if not os.path.isdir(results_dir2):
         os.makedirs(results_dir2)
        new_im.save(results_dir2+"/"+ sys.argv[1] +"_"+sys.argv[2]+"
_Overview.jpg")
```

## A.3 SIA section I: Total graphic for all the years

**Code implementation:**

During this section of the code was used ”pandas” library for read the dataset.

```
series1 = pd.read_csv("Datasets/" + sys.argv[1]+".csv", usecols=[1,sys.argv[2]])
```

Then using the ”pyplot” library has been possible to setup the plot of the input data.

```
series1.plot(color="blue", linewidth=1.5)
```

Thera are some settings about the axis x just to display the data in the right format, are easy to change and to costume.

```
years = []
j = 0
for i in range(len(yearInput)):
    if j==11:
        years.append(yearInput.values[i][0])
        j=0
    else:
        j=j+1
x = range(0, len(yearInput.values))
pyplot.xticks(np.arange(min(x), max(x)+1, 12.0), years)
pyplot.title(sys.argv[1] + "\n" + sys.argv[2]+ ": Total graphic")
```

Once setted up the plot of the current data, the next step was to display the trendline of the current graphic.

At this point the current data values have been read again and passed to the method just impleneted above for calculating the trendline.

```
series1 = pd.read_csv("Datasets/" + sys.argv[1]+".csv", usecols=[sys.argv[2]], squeeze=True)
z1, z2 = trendline(x, series1.values.astype(float), "red")
saveFigure("_Total.jpg")
results_dir = "Results/" + sys.argv[1]+"/"+sys.argv[2]+"/"+sys.argv[1]+"_"+sys.argv[2]+"_AngCoeff.csv"
with open(results_dir, "w") as text_file:
    text_file.write("," + sys.argv[1] + "-" + sys.argv[2]+"\n")
    text_file.write("," + "Ang_Coeff " + "," + str(z1)+"\n")
    text_file.write("," + "Norma_Ang_Coeff " + "," + str(z2)+"\n")
```

## A.4 SIA section II: Single graphics for each year

**Code implementation:**

During this section of the code was used "pandas" library for read the dataset.

```
series2 = pd.read_csv("Datasets/" + sys.argv[1]+".csv", index_col=['month'], usecols=[0,1,sys.argv[2]])
```

Some initialization of variables that are going to be useful.

```
fig2 = pyplot.figure()
ax = fig2.add_subplot(111)
months = ["Jan","Feb","Mar","Apr","May","Jun","Jul","Aug","Sep","Oct","Nov","Dec"]
x_pos = np.arange(len(months))
```

The following code allows the system to split the values and display them in the right way: that means that are going to be splitted for each single year and then plotted on the same graphic.

```
tempValues = []
j = 0
for i in range(len(series2.values)):
    if j in range(12):
        tempValues.append(series2.values[i][1])
        j = j + 1
        if(i == len(series2.values)-1):
            pyplot.plot(x_pos, tempValues, linewidth=2, alpha=0.8, label = int(series2.values[i-1][0]))
    else:
        pyplot.plot(x_pos, tempValues, linewidth=2, alpha=0.8, label = int(series2.values[i-1][0]))
        tempValues = []
        tempValues.append(series2.values[i][1])
        j = 1
```

These are some personalization settings that could be easily changed as you want.

```
ax.legend(loc=4, ncol=1, fancybox=True, shadow=True)
pyplot.xticks(x_pos,months)
pyplot.xlim(0,11)
pyplot.title(sys.argv[1] + "\n" + sys.argv[2]+ ": Single year's graphic")
pyplot.tight_layout()
```

There is the possibility to save the graphic like an image and/or display it.

```
saveFigure("_Years.jpg")
```

## A.5 SIA section III: Correlation matrix between years

**Code implementation:**

During this section of the code was used "pandas" library for read the dataset.

```
series3 = pd.read_csv("Datasets/" + sys.argv[1]+".csv", index_col=['month'
    ], usecols=[0,1,sys.argv[2]])
```

```
corr = []
tempValues = []
j = 0
# Collecting the correct values to elaborate.
for i in range(len(series3.values)+1):
    if j in range(12):
        tempValues.append(series3.values[i][1])
        j = j + 1
    else:
        corr.append(tempValues)
        tempValues = []
        if i in range(len(yearInput)):
            tempValues.append(series3.values[i][1])
            j = 1
```

With the library "numpy" is possible to calculate the correlation coefficents between all the variables in the series just read.

```
corrRes = np.corrcoef(corr)
```

Setup the figure that will display the correlation matrix using the library "pypot".

```
fig3 = pyplot.figure()
ax = fig3.add_subplot(111)
```

Creating the correlation matrix using the already calculated correlation coefficents.

```
cax = ax.matshow(corrRes, interpolation='nearest')
```

Settings for display the matrix in the right way, in particular for the values to display on both the axis x and y, in this case every single year from 2005 to 2016

```
pyplot.title(sys.argv[1] + "\n" + sys.argv[2]+ ": Correlation between
    different years")
x_pos = np.arange(yearsLen)
y_pos = np.arange(yearsLen)
pyplot.yticks(y_pos,years)
pyplot.xticks(x_pos,years)
pyplot.colorbar(cax)
```

Adding a title to the graphic that we are going to display and also a bar that works like a legend for the colors of the matrix, allowing the reader to better understand the values reported inside the matrix.

```
pyplot.tight_layout()
saveFigure("_years_Matrix.jpg")
saveMatrix(corrRes, "Results/"+sys.argv[1]+"/"+sys.argv[2]+"/"+sys.argv[1]+
    "_"+sys.argv[2]+"_years_CorrCoeff.csv")
```

## A.6 SIA section IV: Correlation matrix between months

**Code implementation:**

During this section of the code was used "pandas" library for read the dataset.

```
series4 = pd.read_csv("Datasets/" + sys.argv[1]+".csv", usecols=[0,1,sys.
    argv[2]])
```

```
corr = []
for month, year in series4.groupby(["month"], sort=False):
    corr.append(year[sys.argv[2]].values)
corrRes = np.corrcoef(corr)
```

Setup the figure that will display the correlation matrix using the library "pypot".

```
fig4 = pyplot.figure()
ax = fig4.add_subplot(111)
```

Creating the correlation matrix using the already calculated correlation coefficents.

```
cax = ax.matshow(test, interpolation='nearest')
```

Settings for display the matrix in the right way, in particular for the values to display on both the axis x and y, in this case every single months of the year.

```

months = ["Jan","Feb","Mar","Apr","May","Jun","Jul","Aug","Sep","Oct","Nov"
    ,"Dec"]
x_pos = np.arange(len(months))
y_pos = np.arange(len(months))
pyplot.yticks(y_pos,months)
pyplot.xticks(x_pos,months)
```

Adding a title to the graphic that we are going to display and also a bar that works like a legend for the colors of the matrix, allowing the reader to better understand the values reported inside the matrix.

```
pyplot.title(sys.argv[1] + "\n" + sys.argv[2]+ ": Correlation between
    different months")
pyplot.colorbar(cax)
```

There is the possibility to save the correlation matrix like an image and/or display it.

```
pyplot.tight_layout()
saveFigure("_months_Matrix.jpg")
saveMatrix(corrRes, "Results/"+sys.argv[1]+"/"+sys.argv[2]+"/"+sys.argv[1]+
    "_"+sys.argv[2]+"_months_CorrCoeff.csv")
```

## A.7 SIA section V: Single overview

**Code implementation:**

create_single_overview() : this method will use the "Image" library for autogenerate a collage of the current input's graphics and save it like an overview image. The content of the params will basically decide how the "Current input overview image" will looks like.

It uses each single "current input overview image" of all the inputs and the "correlation matrix between all the inputs image" for combine them in a unique "total overview" and save it using the PDF format.

```
listofimages=["Results/" + sys.argv[1]+"/"+sys.argv[2]+"/"+sys.argv[1]+"_"+
    sys.argv[2]+"_Total.jpg",
              "Results/" + sys.argv[1]+"/"+sys.argv[2]+"/"+sys.argv[1]+"_"+
    sys.argv[2]+"_years_Matrix.jpg",
              "Results/" + sys.argv[1]+"/"+sys.argv[2]+"/"+sys.argv[1]+"_"+
    sys.argv[2]+"_years.jpg",
              "Results/" + sys.argv[1]+"/"+sys.argv[2]+"/"+sys.argv[1]+"_"+
    sys.argv[2]+"_months_Matrix.jpg"]

create_single_overview(4, 1, 1, 3200, 600, listofimages)
create_single_overview(2, 2, 0, 1600, 1200, listofimages)
```

# Appendix B

# MIA Implementation code

## B.1 MIA: Imported libraries

The ”pandas” library will be very useful for read the data from CSV dataset and setup the plot abut it.

```
import pandas as pd
```

The ”numpy” library it’s used for mathematic purpose, such as calculating the correlation coefficent between two series.

```
import numpy as np
```

Also the library ”sys” would be very useful for test and execute the program, mainly because it allows to input directly from terminal.

```
import sys
```

This pyplot style configuration allows to customize the graphic desig. In this case was used ggplot, a popular plotting package.

```
pyplot.style.use('ggplot')
```

The ”pyplot” library it’s used for basic graphic displaying and customization, easy to use but very efficent.

```
import matplotlib.pyplot as pyplot
```

## B.2 MIA section I: Total Correlation Coefficients

During the first part of the implementation of this system was used again the "pandas" library for reading all the values of each single paramaeter of the current input dataset.

```
series = pd.read_csv("Datasets/" + sys.argv[1]+".csv", usecols=range(2,10),
    header=0)
corr = []
for column in series:
    corr.append(series[column].values)
```

Then, once read and organized the values, are calculated all the correlation coefficients between each single paramater values

```
corrRes = np.corrcoef(corr)
mat = np.matrix(corrRes)
dataframe = pd.DataFrame(data=mat.astype(float))
```

The resulting correlation coefficients values are reported in a CSV output file.

```
dataframe.to_csv("Results/"+sys.argv[1]+"/Total_Evidences/"+sys.argv[1]+"
    _CorrCoeff.csv", sep=',', header=False, float_format='%.2f', index=
    False)
```

The final step of this first part of the MIA implementation is to display the calculated correlation coefficients on a correlation matrix. The following code show how to set it up, customize both the tick labels and in end how to save it.

```
fig = pyplot.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(corrRes, interpolation='nearest')
labels = []
j = 1
for i in range(len(series.columns)+1):
      if i == 0:
         labels.append("")
      else:
         labels.append(series.columns[i-1])
ax.set_xticklabels(labels)
ax.set_yticklabels(labels)
pyplot.setp(ax.get_xticklabels(), rotation=30, horizontalalignment='left')
pyplot.setp(ax.get_yticklabels(), rotation=30, horizontalalignment='right')
pyplot.colorbar(cax)
pyplot.title("Correlation Coefficients Matrix - " + sys.argv[1], y=1.15)
pyplot.tight_layout()
pyplot.savefig("Results/" + sys.argv[1]+"/Total_Evidences/"+sys.argv[1]+"
    _Total_Matrix.jpg", format="jpg")
```

## B.3 MIA section II: Normalized Angular Coefficients

During this part of the code the system will read for each single paramater of the current input dataset the trend line normalized angular coefficient. The coefficients values are organized and saved in a temporary data structure.

```
temp = []
for i in series.columns:
    index = sys.argv[1]+"-"+i
    tempSeries = pd.read_csv("Results/"+sys.argv[1]+"/"+i+"/"+sys.argv[1]+"_"+i+"_AngCoeff.csv", header=0)
    temp.append(tempSeries[index].values[1])
```

Once the values are ready to be elaborated, the system displays it on a horizontal bar plot using the library "pyploy".

```
fig2 = pyplot.figure()
ax2 = fig2.add_subplot(111)
x = range(len(series.columns))
pyplot.barh(x, temp)
```

Last part of the code was used for graphic's customization and for saving it.

```
pyplot.yticks(x,series.columns)
pyplot.title("Normalized Angular coefficients - " + sys.argv[1])
pyplot.tight_layout()
pyplot.savefig("Results/"+sys.argv[1]+"/Total_Evidences/"+sys.argv[1]+"_Norm_Ang_Coeffs.jpg", format="jpg")
```

# Appendix C

# Norway's Map System Implementation Code

## C.1 Map System: Imported libraries

The library report below here have already been used during the previous phase of the implementation work, and is possible to check out their utilities here: [A.1]

```
import os
import sys
import matplotlib
import pandas as pd
import matplotlib.pyplot as plt
```

The library "cartopy" provides cartographic tools for Python, in particular "crs" (Coordinate Reference Systems) that is the very core of cartopy since allow to use a list of several projections with the same interface. Further more, the class "shapereader" provides an interface for accessing the contents of a shapefile.

```
import cartopy.crs as ccrs
import cartopy.io.shapereader as shpreader
```

The library "matplolib" provides with the module "cm" a large set of colormaps and with the "patches" module it allows to draw some geometry figures.

```
import matplotlib.cm as cmx
import matplotlib.patches as mpatches
```

## C.2 Norwegian map implementation

The following implemented method allows to give a specific shapely geometries to the axes. The method get this particular parameter with "shapeInput". Furhtermore, "labelInput" is in this case the name of the current Norwegian county and the "colorInput" the specific color which will be used for display the geomtry.

```
def add_geom(axes, shapeInput, labelInput, colorInput):
    axes.add_geometries(shapeInput, ccrs.Robinson(), edgecolor='black',label
      = labelInput, facecolor=colorInput, alpha=0.8)
    return mpatches.Rectangle((0, 0), 1, 1, facecolor=colorInput)
```

The shapefile "NOR_adm1.shp"[1] contains the needed data for display each single Norwegian county. The variable "NOR_shapes" allows to access in a easier way to this values, since it's a list which each position corresponds to a specific county shape.

```
fname = 'Datasets/NOR/NOR_adm1.shp'
NOR_shapes = list(shpreader.Reader(fname).geometries())
```

How reported in the implementation partm during this part of the work was created a specific dataset "countiesAverages". [5.3] The system creates a list that contains the value of the paramater "dataInput" for each county.

```
dataInput = sys.argv[1]
inputSeries = pd.read_csv("Datasets/countiesAverages.csv")
inputValues = [inputSeries[dataInput][0], inputSeries[dataInput][1],
     inputSeries[dataInput][2], inputSeries[dataInput][3], inputSeries[
     dataInput][4], inputSeries[dataInput][5], inputSeries[dataInput][6],
     inputSeries[dataInput][7], inputSeries[dataInput][8]]
```

In the following code the class "Normalize" is used for normalize the data input into [vmin , vmax] interval, that are respectively max and min for input values.

```
minValues = min(inputValues)
maxValues = max(inputValues)
cNorm = matplotlib.colors.Normalize(vmin=minValues, vmax=maxValues)
```

[1]Shapefile about Norway's territory download link:
`http://biogeo.ucdavis.edu/data/gadm2.8/shp/NOR_adm_shp.zip`

The "colMap" variable contains a specific range of colors and it will be used together with thenormalized range of values by the class "ScalarMappable" that returns RGBA colors, that with the function "colorbar" are used for generate a legend about the values and the colors.

```
colMap='bwr'
cm = plt.get_cmap(colMap)
scalarMap = cmx.ScalarMappable(norm=cNorm, cmap=cm)
col = scalarMap.to_rgba(inputValues)
scalarMap.set_array(inputValues)
plt.colorbar(scalarMap,label='Input Value')
```

Once the system has the values and the relative colors, it uses the "add_geom" method in order to display the geometric shape of each country with the related color and label.

```
ax = plt.axes(projection=ccrs.Robinson())
ax.coastlines(resolution='10m')
ax.set_extent([4, 32, 57, 72], ccrs.Robinson())
norway = add_geom(ax, NOR_shapes, "Norway", "gray")
finnmark = add_geom(ax, NOR_shapes[4], "Finnmark", col[0])
troms = add_geom(ax, NOR_shapes[16], "Troms", col[1])
nordland = add_geom(ax, NOR_shapes[9], "Nordland", col[2])
nord_trondelag = add_geom(ax, NOR_shapes[8], "Nord Trondelag", col[3])
sor_trondelag = add_geom(ax, NOR_shapes[13], "Sor Trondelag", col[4])
more_og_romsdal = add_geom(ax, NOR_shapes[7], "More og Romsdal", col[5])
sogn_og_fjordane = add_geom(ax, NOR_shapes[14], "Sogn og Fjordane", col[6])
hordaland = add_geom(ax, NOR_shapes[6], "Hordaland", col[7])
rogaland_og_agder = add_geom(ax,NOR_shapes[2], "Rogaland og Agder",col[8])
rogaland_og_agder = add_geom(ax,NOR_shapes[12], "Rogaland og Agder",col[8])
rogaland_og_agder = add_geom(ax,NOR_shapes[17], "Rogaland og Agder",col[8])
```

Final settings for modify the title of the graphic and display a legend with the correct labels.

The "manager.resize(*manager.window.maxsize())" function allows to maximize to fullscreen the displayed graphic with "plt.show".

```
plt.title('Norway - '+sys.argv[1] , fontsize=35)
labels = ['Finnmark', 'Troms', 'Nordland','Nord Trondelag', 'Sor Trondelag'
    , 'More og Romsdal', 'Sogn og Fjordane', 'Hordaland',    'Rogaland og
    Agder','Other counties', ]
plt.legend([finnmark, troms, nordland, nord_trondelag, sor_trondelag,
    more_og_romsdal, sogn_og_fjordane, hordaland, rogaland_og_agder, norway
    ],
    labels, loc='lower right', fancybox=True)
manager = plt.get_current_fig_manager()
manager.resize(*manager.window.maxsize())
plt.show()
```

# Bibliography


[1] Jason Brownlee. How to create an arima model for time series forecasting with python. http://machinelearningmastery.com/arima-for-time-series-forecasting-with-python/, January 9, 2017. Online.

[2] Wikipedia. Data science — wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Data_science&oldid=778145331", 2017. [Online].

[3] Arthur Samuel, 1959.

[4] Jason Brownlee. Machine learning mastery, 2017.

[5] Time series analysis: Forecasting and control. [page 1].

[6] Wikipedia. Autoregressive model — wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Autoregressive_model&oldid=768765293", 2017. [Online].

[7] Wikipedia. Moving-average model — wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Moving-average_model&oldid=775234540", 2017. [Online].

[8] Wikipedia. Autoregressive–moving-average model — wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Autoregressive%E2%80%93moving-average_model&oldid=781957673", 2017. [Online].

[9] Wikipedia. Autoregressive integrated moving average — wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Autoregressive_integrated_moving_average&oldid=777845495", 2017. [Online].

[10] Standing Senate Committee on Fisheries and Oceans. Volume two - aquaculture industry and governance in norway and scotland, June 2016. [Online].