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 feasibility 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, such as the Norwegian aquaculture industry, since Norway represents the forefront of innovation and development in this area.
In particular, this study is focused on data about Norwegian salmon farming, cause of the high business interest and the significant amount of generated 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.
Furthermore, 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 first like to thank my thesis supervisor Ståle Walderhaug, who allowed me to realize this work which provided me extremely useful knowledge and experience.

I would also like to express my special appreciation and thanks

To Bård Johan Hanssen

To Sara Björk for useful inspirations and tips.

To the staff of SINTEF nord for the given support and good times.

My special thanks are extended to all the people that somehow permitted me to realize my thesis participating at an amazing Erasmus project in Norway, in particular:

To my family, that always support me.

To the staff of my home university: Università di Milano Bicocca.

To the staff of my guest university: 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 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 interesting to test and evaluate different ways to apply daily technologies to its procedures and systems.

Python is a well-known interpreted, object-oriented 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 aquaculture, 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 contribute 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.
- Evaluate the feasibility of Python in the Data Science field, describing implementation procedures and reporting observation.
- Report the initial analysis, displaying and forecasting results about Norwegian salmon farming, in order to provide structured, described and accessible 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 can the data be accessed? What are the required access and usage limitations?
- Which kind of data is possible to obtain about aquaculture of single locations in Norway? Where can the data be accessed? What are the required access and usage limitations?

2) Increase accessibility and availability of the data.

- How can a unique dataset that contains all aquaculture data be created?
- Which kind of structure allows to increase the total dataset accessibility and responsiveness?

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 it possible to identify it?
- How is it possible to check out the data trend line?
- Which kind of informations have been reported for future reuse? How is it 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 parameter's trend line?
- Which kind of correlations is it possible to find between different parameters? How is it possible to show them? What is it possible to extract from that?

5) Prediction of values about the data.

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

1.3 Thesis Structure

Background Theory
This section reports the relevant background theory that is necessary 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 enough data is collected, it will be documented and structured in a proper dataset.

II Part: System implementation
During this part of the work the Python system is implemented and the procedure is reported, in order to try to find as many answers as possible to the initial goals. The implemented system will be divided in several subsystems to allow an easier implementation procedure and a higher reusability. An overview about the implemented subsystems is provided in the next section. [4.2]

III Part: Result, Discussion and Conclusions
In the current part of the study the results and the relative interpretations are reported. Furthermore, a general summarize about the study and also any kind of evaluations, challenges, limitations encountered and some recommendations for future works is reported.

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

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" [1].

1.5 Work repository

Before starting 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: 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].

Cluster Analysis: is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense or another) to each other than to those in other groups (clusters) [4].

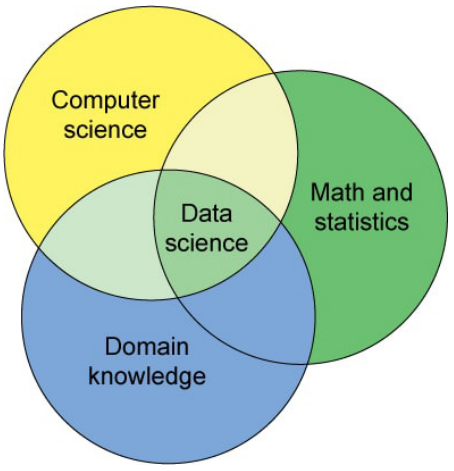


FIGURE 2.1: Data science concept

Classification: Is the problem of identifying to which of a set of categories (sub-populations) a new observation belongs [5].

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.

Databases: A database is an organized collection of data. It is the collection of schemas, tables, queries, reports, views, and other objects. The data are typically organized to model aspects of reality in a way that supports processes requiring information. [6]

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.

This study procedure is strongly based on the Data Science process designed by Joe Blitzstein and Hanspeter Pfister [7], which is reported in the following image.

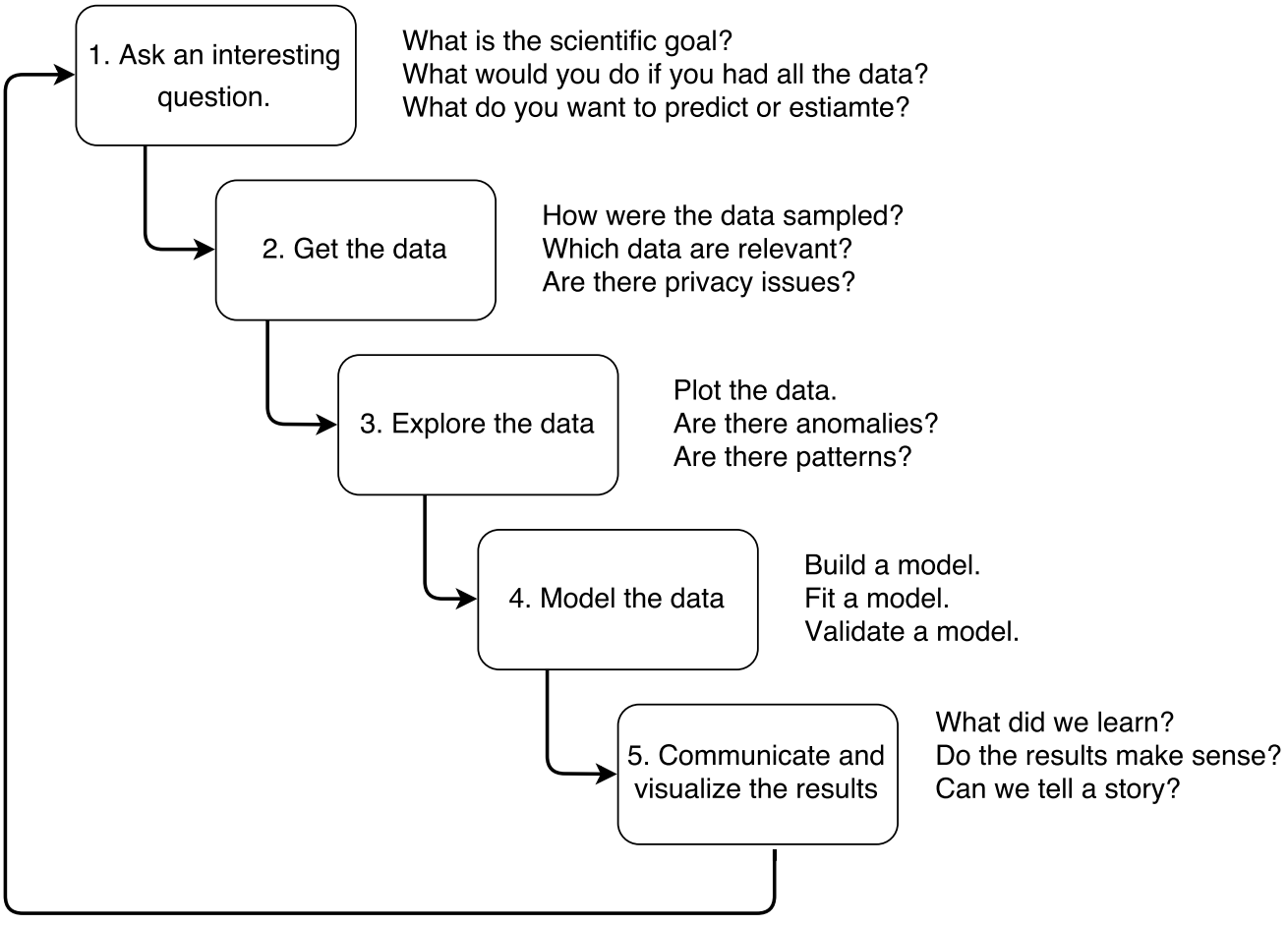


FIGURE 2.2: Data science process

2.2 Machine learning

As 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 algorithms, each one of them is used for a different purpose and a different domain. For examples:

- Artificial Neural Network: Computations are structured in terms of an interconnected group of artificial neurons. They are usually used to model complex relationships between inputs and outputs, to find patterns in data or to capture statistical structures.

- Deep Learning: This concept consists of multiple hidden layers in an artificial neural network, and it can be successfully applied for computer vision and speech recognition.

- Clustering: Is the assignment of a set of observations into subsets, called clusters, so that observations within the same cluster are similar according to some predesignated criteria, while observations drawn from different clusters are dissimilar.

- **Regression**: Is a statistical process for estimating the relationships among variables. It includes many techniques for modeling and analyzing several variables, when the focus is on the relationship between a dependent variable and one or more independent variables. Regression analysis is widely used for prediction and forecasting, where its use has substantial overlap with the field of machine learning. This specific domain contains the model used in this study [8].

- 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 [1].

" A time series is a sequence of observations taken sequentially in time [9]. "

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 technical 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)

A widely used statistical method for time series forecasting is the ARIMA model. Since this a very complicated and deep topic, this study provided just an initial implementation and description 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 particularly 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.[10]

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.[11]

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.[12]

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.[13]

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.[14]

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 fish sector Nordland is the dominant producer county, with Hordaland coming second, Møre og Romsdal third, and Troms fourth.

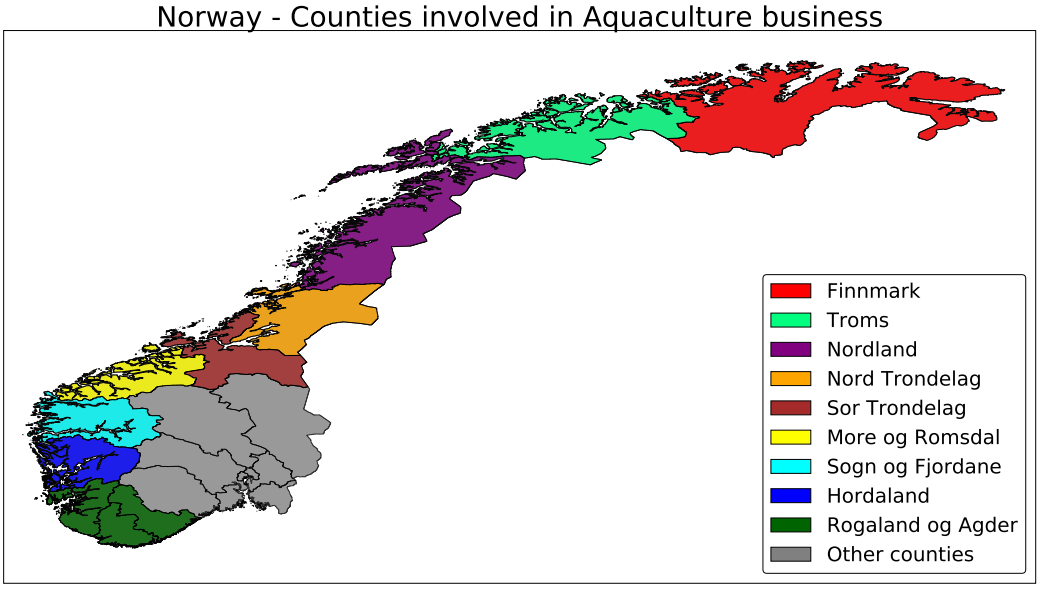


FIGURE 2.3: Norwegian counties involved in aquaculture business

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], which allows to anyone to download published just providing a valid email address [2].

The data 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
[2]"6.1 Download a public dataset" :
https://archive.norstore.no/user-guide.pdf

3.1.2 Data from Fiskeridirektoratet

Fiskeridirektoratet[3] has been the main data source for this work. They provide several statistics about Aquaculture in Norway which are subject to the Norwegian Public Data License (NLOD), that allows a free reusage of the data on the terms developed by Difi [4].

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 Fiskeridirektoratet.

About the current data source it 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.
- It is not possible to implement an automatic download script.

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

[4]Terms of use of the Directorate of Fisheries's data (Difi):
`http://data.norge.no/nlod/no`

3.1.3 Data from Indexmundi

An another source of data for this study was Indexmundi[5], which provides data about fish (salmon) monthly price, Norwegian krone per kg, with a public access policy.

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

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

3.2.1 Dataset about Norway

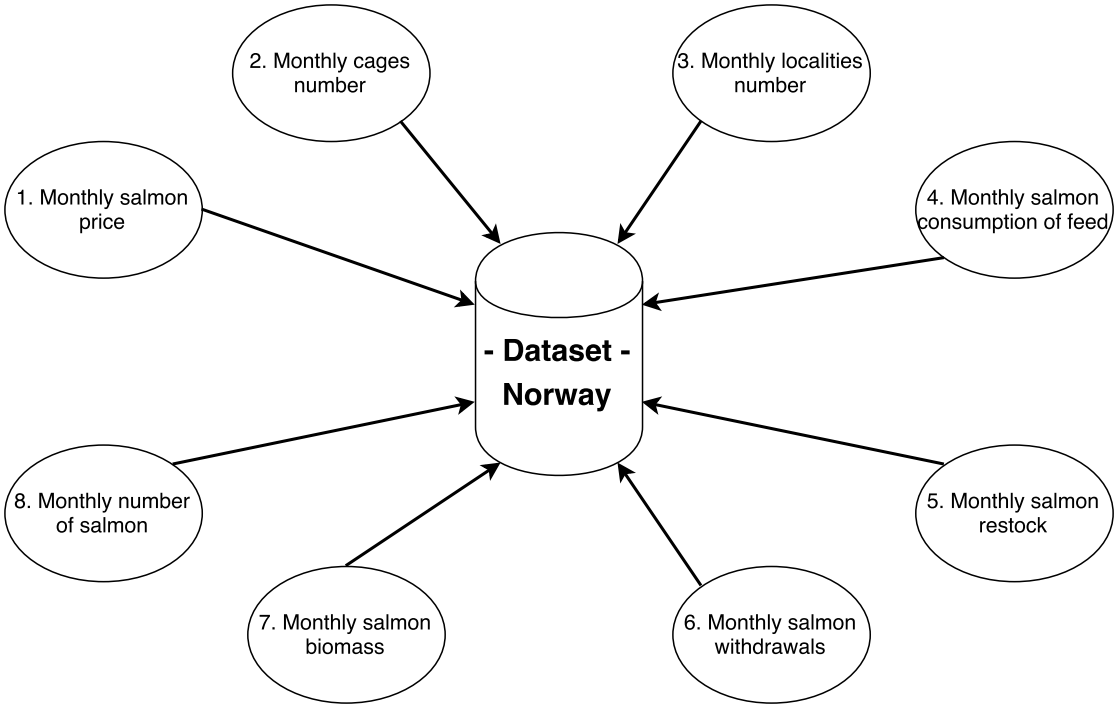


FIGURE 3.1: Dataset structure.

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

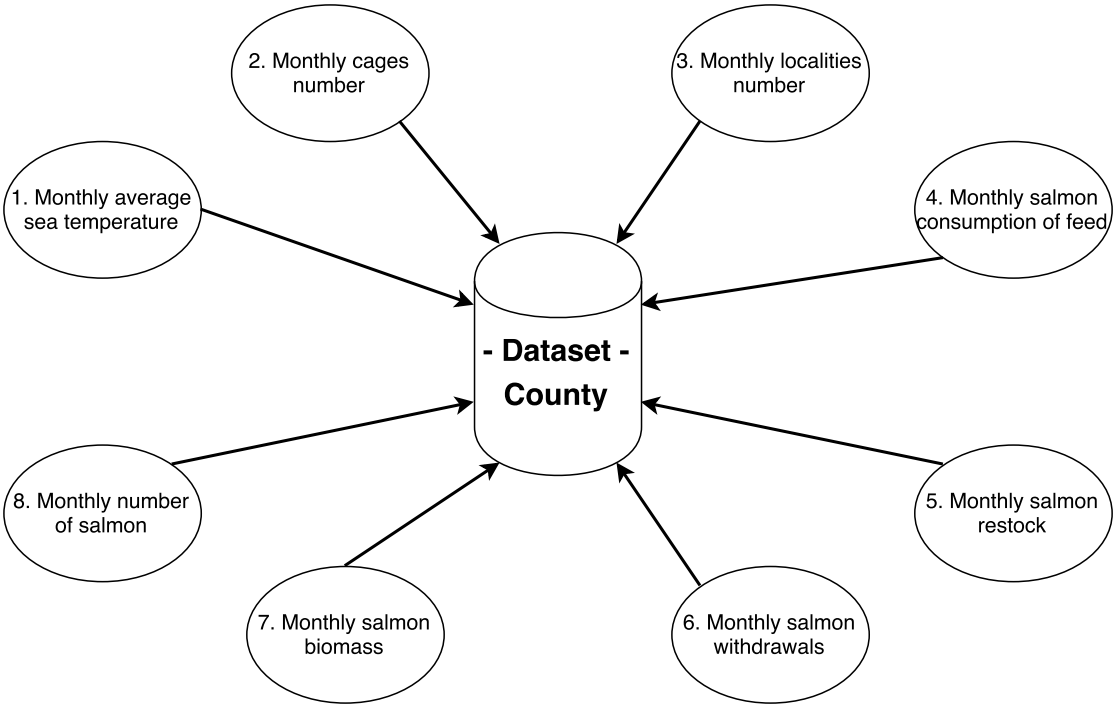


FIGURE 3.2: Dataset structure.

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 the implementation procedure for each of the system used for this study will be documented. In order to be able to redo the procedure or just simply test the final resulting systems, it's necessary to comply the following requirements.

4.1.1 Requirements for reusability

In order to let the implemented Python systems works in a good way, there is just one important requirement about the possible input datasets that has to be satisfied :

- Monthly frequency of data values.

4.1.2 System requirements

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

```
1 Python version: 2.7.12
```

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

- SciPy [1]
- Cartopy [2]
- Statsmodels [3]

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

4.2 System Design

As described 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.

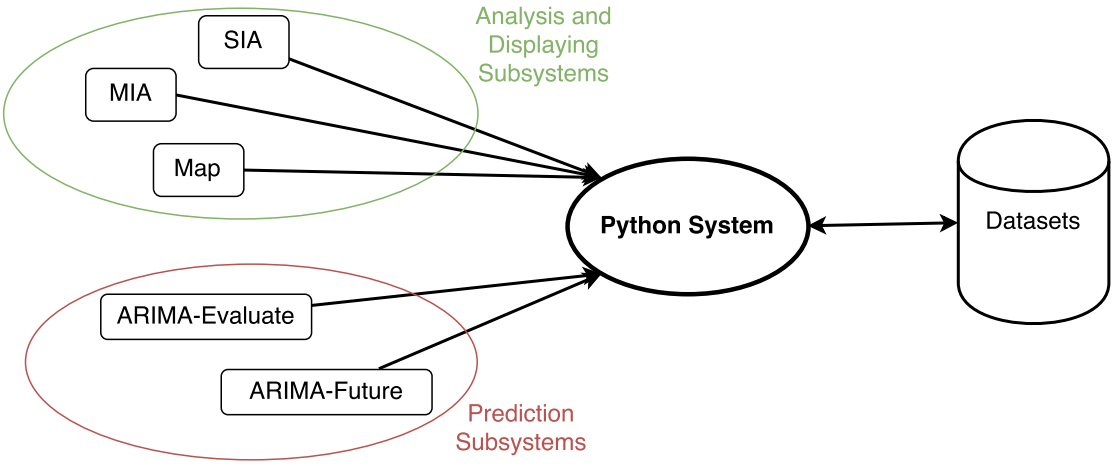


FIGURE 4.1: Subsystems overview.

- **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 results about all the parameters of the input dataset, such as the correlation coefficients between different parameters and comparison of their normalized angular coefficients.
- **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 parameter 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 information that can be useful in later steps.

The system that is 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.

```
1 series = pandas.read_csv()
2 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 the first goal of displaying and saving the basic graphic about the current input has been reached, also including the relative trend line and saving its angular coefficient in a document. An example result is shown in figure[5.1].

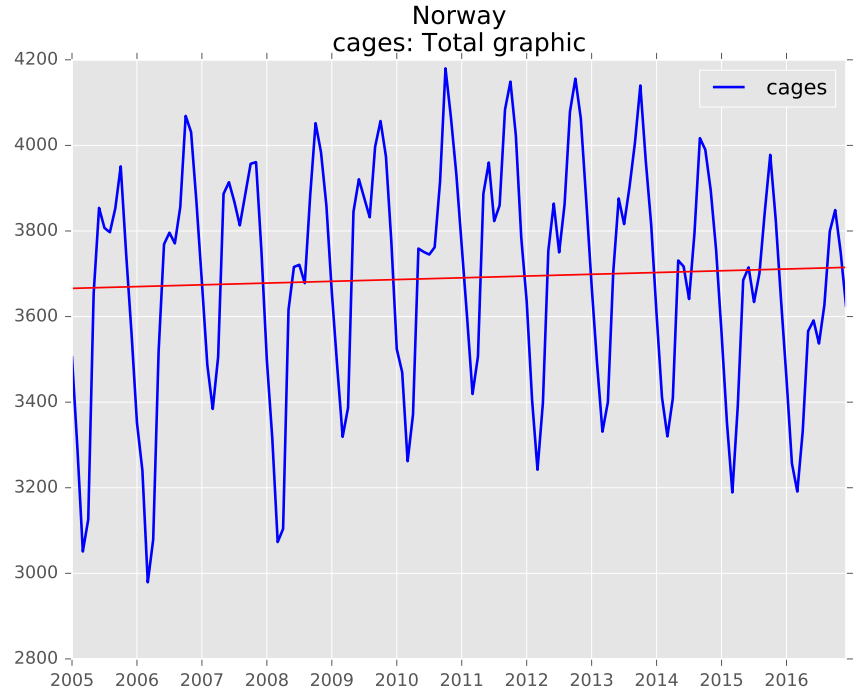


FIGURE 5.1: Total graphic about current input over the whole period.

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.

```
1 series = pandas.read_csv()
2 series.values()
3 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 the goal of displaying and saving the graphic of the plots for each single year about the current input has been reached. An example result is shown in figure [5.2].

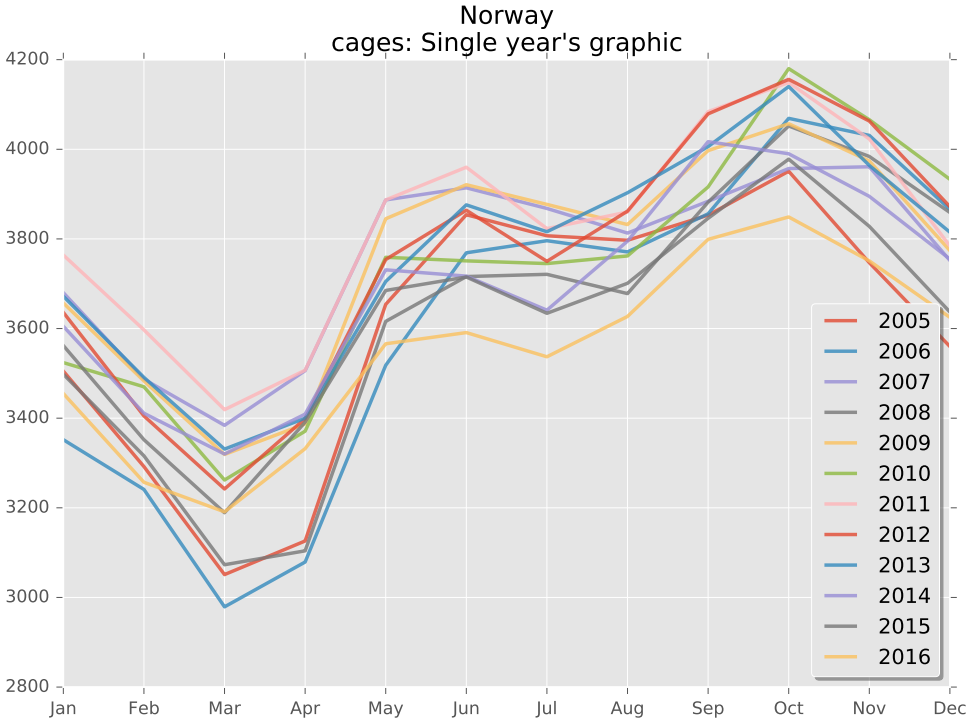


FIGURE 5.2: Graphics for each single year of the current input data.

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 the scientific computing library "numpy" has been used , this allows to calculate the correlation coefficients between data. Then the library "pyplot" has been used to display the results on a matrix.

```
1 numpy.corrcoef()
2 figure = pyplot.figure()
3 ax = figure.add_subplot()
4 ax.matshow()
```

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

Results:

With this part of the code the correlation coefficient values between each single year of the current input have been calculated, displayed and reported in a document. An example result is shown in figure [5.3].

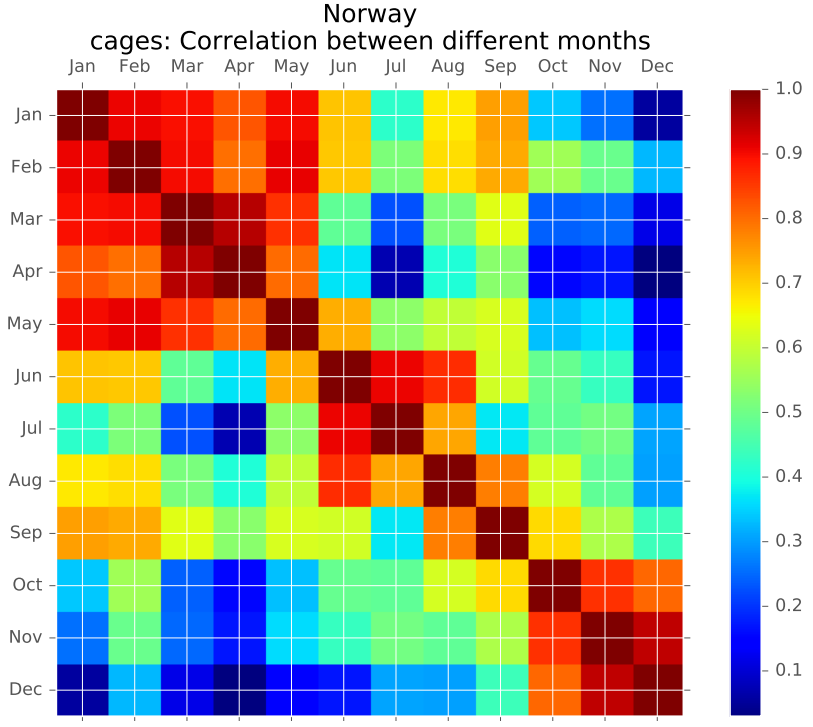


FIGURE 5.3: Correlation matrix between different months of the same input

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 the scientific computing library "numpy" has been used, this allows to calculate the correlation coefficients between data. Then the library "pyplot" has been used to display the results on a matrix.

```
1 numpy.corrcoef()
2 figure = pyplot.figure()
3 ax = figure.add_subplot()
4 ax.matshow()
```

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

Results:

With this part of the code the correlation coefficient values between each single month of the current input have been calculated, displayed and reported in a document. An example result is shown in figure [5.4].

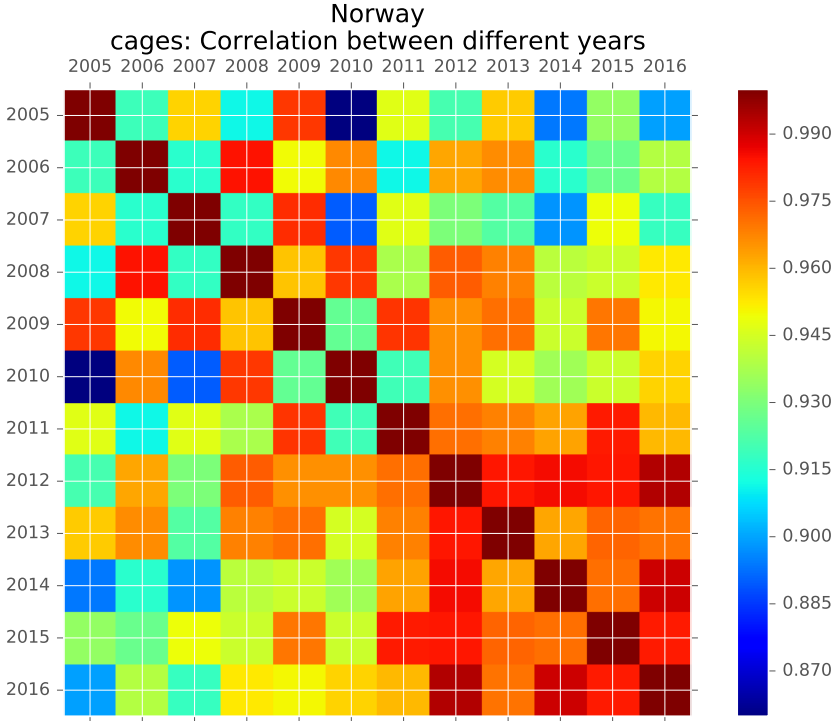


FIGURE 5.4: Correlation matrix between different years of the same input

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 the Python Imaging Library has been indispensable, called also PIL[1].

```
1 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.

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

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

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

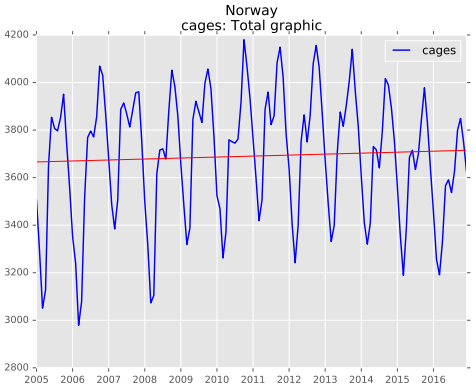
The output of this phase depends by the input to this method, that are basically the list of image and the preferences about the collage's structure.
It's possible to view the final result of this phase in the next page and it's possible to check out the full commented code in the appendice: [A.7]

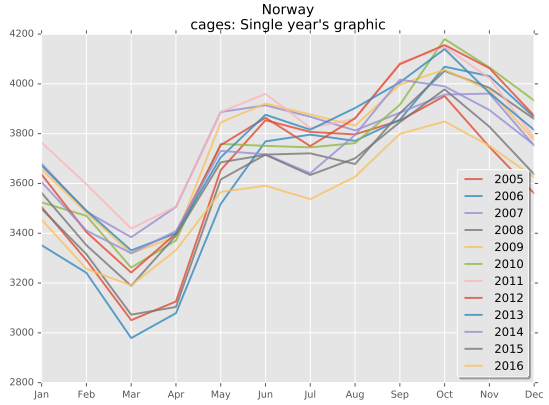
[1]Python Imaging Library : `http://www.pythonware.com/products/pil/`

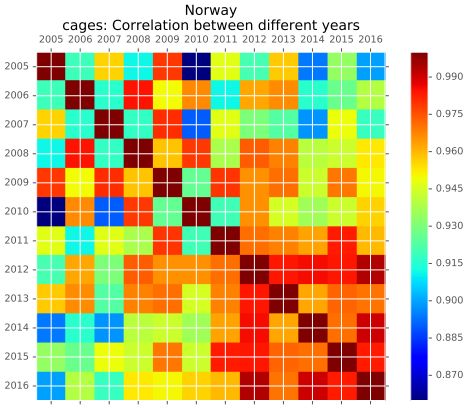
Results:

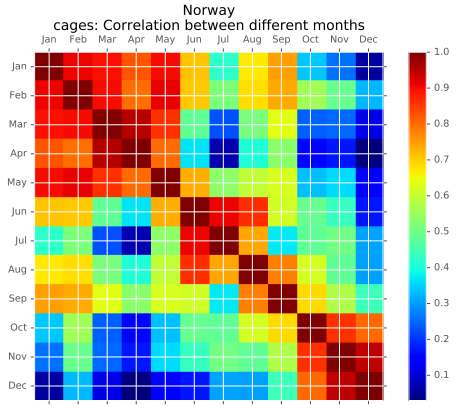
With this part of the code it's possible to have a single overview image about the current input. This basically allows to compare all the graphics already calculated about this input. The general overview graphic contains:

- 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 a list of the 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, it's necessary that the current dataset has been already analyzed from the SIA system.

Implementation:

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

```
1 numpy.corrcoef()
2 figure = pyplot.figure()
3 ax = figure.add_subplot()
4 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:

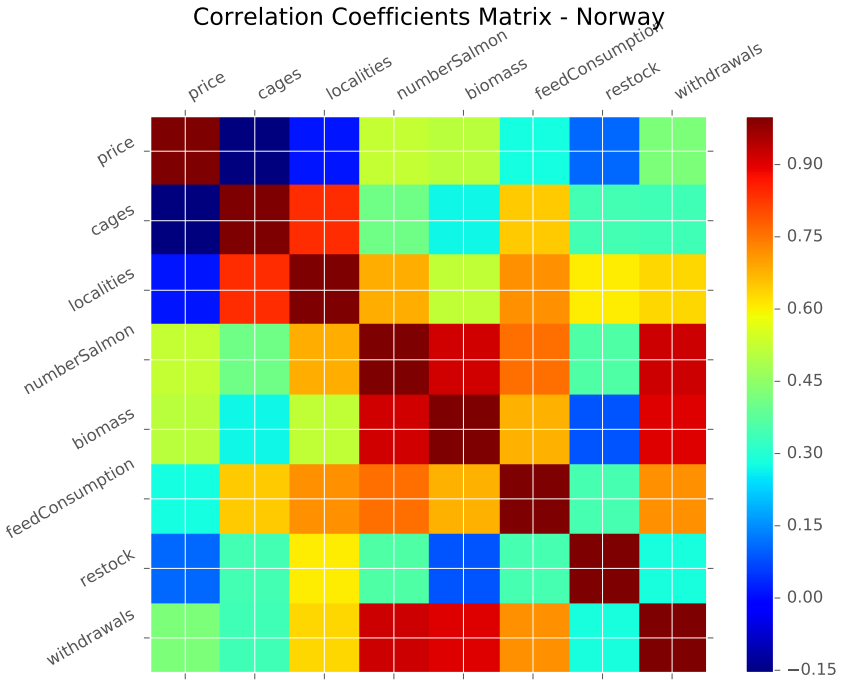


FIGURE 5.7: Correlation matrix between different inputs with data.

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

```
1 pandas.read_csv()
2 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:

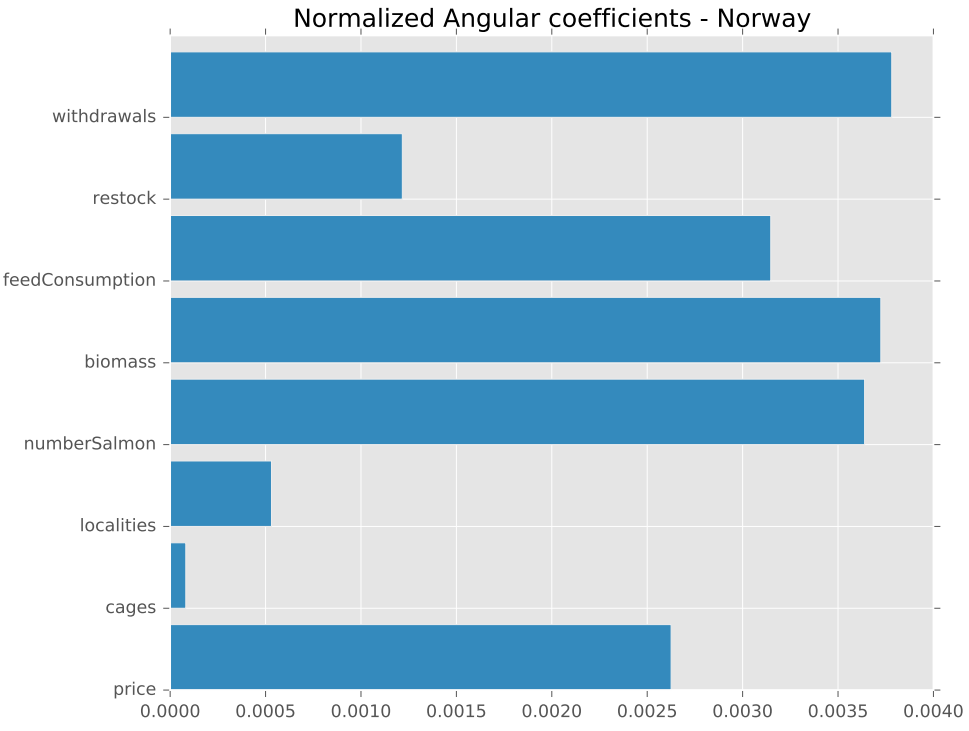


FIGURE 5.8: Normalized angular coefficients of each input's trendline.

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 a specific dataset for test the system works has been created. 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 count the average value from 2007 to 2014 of different parameters have been calculated.

county	**averageSeaTemp**	**cages**	**...**	**feedConsumption/biomass**
Finnmark	5.2128134819	257.2395833333	...	0.1611964666
Troms	6.2185416667	393.3958333333	...	0.1831404686
Nordland	6.8333444959	804.5104166667	...	0.1849358645
Nord-Trondelag	7.322600258	231.6875	...	0.1852350478
Sor-Trondelag	7.5381376237	306.9479166667	...	0.1862036956
More_og_Romsdal	8.0087820154	347.3229166667	...	0.1831662176
Sogn_og_Fjordane	8.1081250683	318.9583333333	...	0.1863151035
Hordaland	7.8033025443	738.8854166667	...	0.1925203347
Rogaland_og_Agder	7.1951075619	338.53125	...	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"[2], 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]

```
1 import cartopy.crs as ccrs
2 import cartopy.io.shapereader as shpreader
3 shpreader.Reader(filename).geometries())
```

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

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

```
1 plt.figure()
2 ax = plt.axes()
3 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".

```
1 plt.get_cmap
2 matplotlib.colors.Normalize
```

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

Results:

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

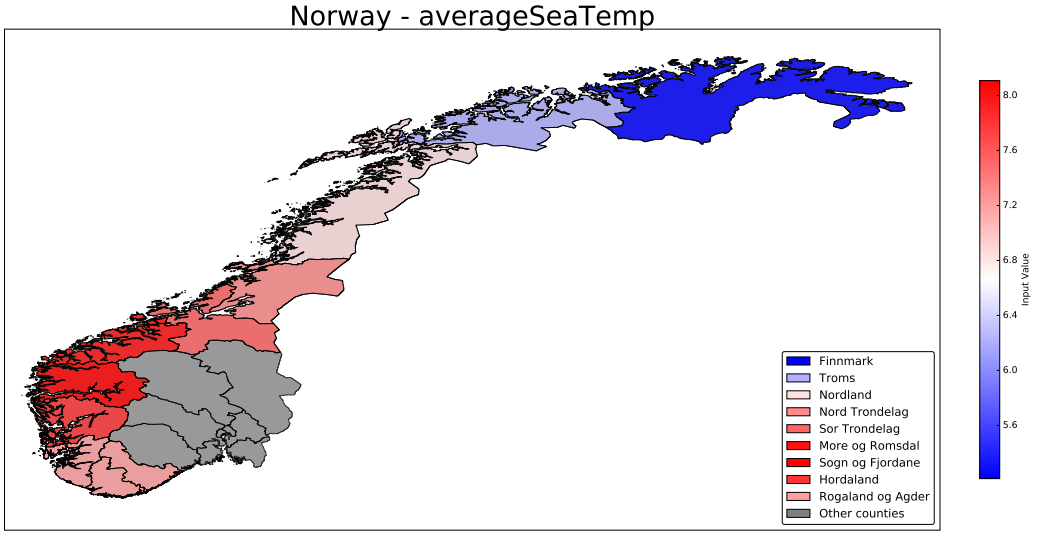


FIGURE 5.9: Monthly average sea temperature from 2007 to 2014 in Norway

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, it was decide to implement and test an Autoregressive Integrated Moving Average (ARIMA) model.
Since there are several possible configurations that fits an ARIMA model, it is important to find the right one to use with each input dataset because it would give much better prediction results. In order to find the best ARIMA configuration there are different methods and procedures. The most known is "Box–Jenkins method"[1]. In this study it was decided 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 of testing different ARIMA model configurations for the same input dataset and then comparing the results.
For this reason, during this phase of the work two different subsytems for two different purposes have been implemented:

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 evaluating different configurations of the ARIMA machine.
It tests 112 different configurations for the current input and reports the tested configurations with the corresponding MAPE (Mean Average Percentage Error)[2] value in a document. MAPE is a measure of prediction accuracy of a forecasting method in statistics, for example in trend estimation, and it usually expresses accuracy as a percentage.

Requirements:

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

Code implementation:

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.

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

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

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

It's important to remind that the following evaluation system is fitting the model with the first 66% of the dataset values, and then the model forecasting is tested and compared on the final 34% .

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

[2]MAPE formula and description:
https://en.wikipedia.org/wiki/Mean_absolute_percentage_error

Results:

The system will report the MAPE between real value and predicted values for each of the 112 tested ARIMA machine in a document. During this part of the work also a script[3] for executing an ARIMA evaluation about each single parameter of each single dataset was created, but once executed it required too much time to complete all the evaluations. For that reason, just some relevant evaluations result have been reported in a document. The table below reports some results about specific parameters that are going to be used in the next phase.

Input	**Parameter**	**ARIMA Conf**	**MAPE result**
Finnmark	feedConsumption	(6, 1, 0)	13.771%
Hordaland	feedConsumption	(8, 0, 0)	6.811%
Troms	feedConsumption	(2, 0, 0)	11.593%
Nordland	feedConsumption	(6, 0, 0)	12.741%
Norway0714	feedConsumption	(6, 1, 0)	7.296%

TABLE 6.1: MAPE Results for some particular dataset about the parameter "feedConsumption"

[3]Link to the implemented script "autoEvaluate.sh" : urlhttps://github.com/Sprea22/Personal_Utilities

6.2 Prediction System

Goal:

This system has two main goals:

- Predict some future value with the a specific ARIMA configuration.
- Display the historic data together with real future values and predicted future values.

Requirements:

The only requirement is to use in a correct way this system is that the real future values are available, in order to compare it with the predicted one. It's possible to use this system on input dataset of arbitrary length.

Code implementation:

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 with a "int" parameter that represents the desired number of predictions that have to be calculated.

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

Once calculated and saved the predictions in a document, the system will basically display on the same graphic "realVaues" that contains the real future values, "predFuture" that contains the predicted future values and "series" that contains the dataset historic values.

```
1 ax.plot(realValues, "g", label='Real 2015 Values', linewidth=2)
2 ax.plot(predFuture, "r", label='Prediction 2015 values', linewidth=2)
3 ax.plot(series, "b", label='Historic values', linewidth=2)
```

It's possible to check out the full implemented code in the appendice: [D.4]

Results:

This system will automatically generate a document that contain:

- Real future values
- Predicted future values
- MAPE between each prediction and the corresponding real value.

It provides also the possibility to visualize the historic, real and predicted future values on the same graphic, as shown in the example figure [6.1].

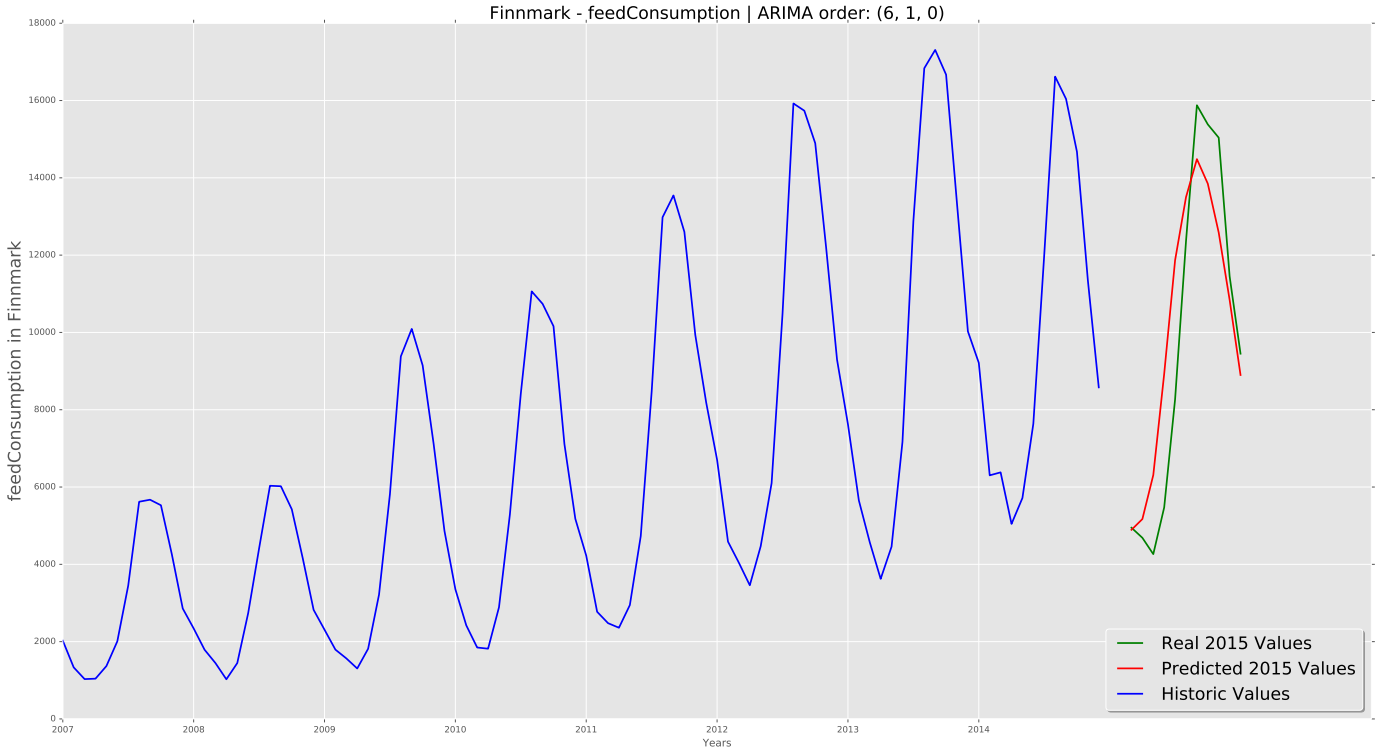


FIGURE 6.1: Graphic that display historic, real and predicted future values of a input.

Part III

Results, Discussion, and Conclusions

Chapter 7

Results Overview

This study provides several results that can be divided into two categories:

- Implemented Python Systems
- Generated evidences and values

Furthermore, since a significant amount of time and effort was spend for setting up the datasets, it's possible to consider them like results as well.
If you're interested to check out the whole results and even reuse it, it's possible to find it on the already cited Github repository[1].

7.1 Implemented Python Systems

- **SIA.py** : System able to provide an initial analysis and displaying about a specific dataset's parameter dataset.
- **MIA.py** : System able to provide an initial analysis and displaying about all the parameters of a specific dataset.
- **Map.py** : System able to provide an initial cartographic visualization about a specific parameter in the Norwegian territory.
- **ARIMA-Evaluate.py** : System able to provide an evaluation about different configurations of an ARIMA machine, appplied to a specific dataset's parameter.
- **ARIMA-Future.py** : System able to calculate and display some future values prediction and compare it with the real values (if available) using the ARIMA model.

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

7.2 Generated Evidences and values

Analysis Results - For each dataset have been provided:

- Document containing the correlation coefficients between all the dataset parameters and the corresponding correlation matrix graphic.
- Graphic displaying and comparing the normalized angular coefficients of all the dataset paramaters.
- For each single parameter of the the dataset:
  - Total graphic and trend line of the dataset values.
  - Graphic displaying the different years plot.
  - Document containing the correlation coefficients values between different years and the corresponding correlation matrix graphic.
  - Document containing the correlation coefficients values between different months and the corresponding correlation matrix graphic.
  - Document containing the angular coefficient and the normalized angular coefficients of the trend line.

Forecast Results - The Evaluation and Future prediction systems have been executed just on the most interesting parameter of some dataset, due to the short time left and to the Evaluation procedure that takes a lot of time. More in particular, it's possible to check out the Evaluation MAPE Results, Preditcted future values results and Real future values about the parameter "Feed Consumption" for the following counties: Finnmark, Troms, Nordland, Hordaland and whole Norway,

Map Results - 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.

7.3 Particularly interesting results

In the following two pages is reported a small portion of the results written above.
The reason for this choice is that they will be indispensable for the discussion during the next chapter.
The list of figures [7.1, 7.2, 7.3, 7.4, 7.5] shows the values shape for each year of some specific datasets (Norway, Finnmark, Troms, Nordland, Hordaland).
The table 7.1 and 7.2 show some specific results of the Forecast System.

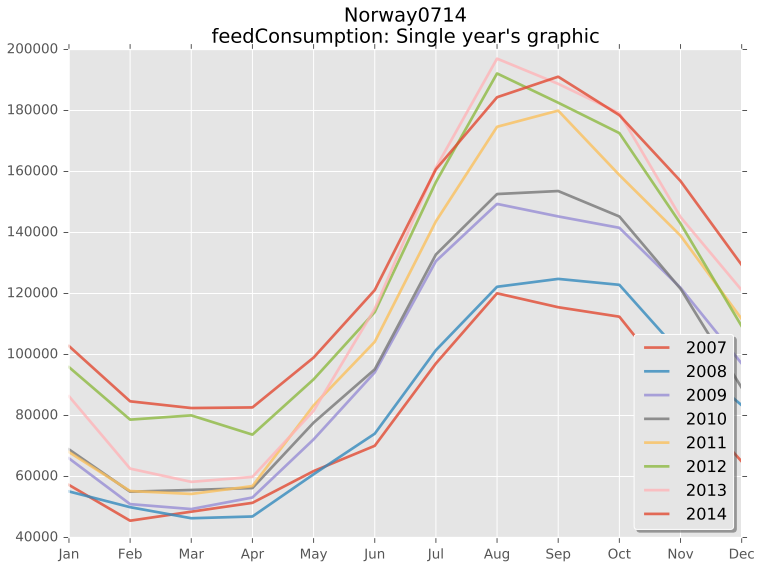


FIGURE 7.1: Annual consumption of feed trend in Norway.

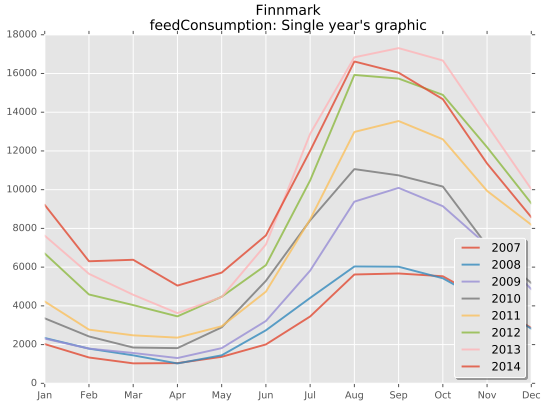


FIGURE 7.2: Annual consumption of feed trend in Finnmark.

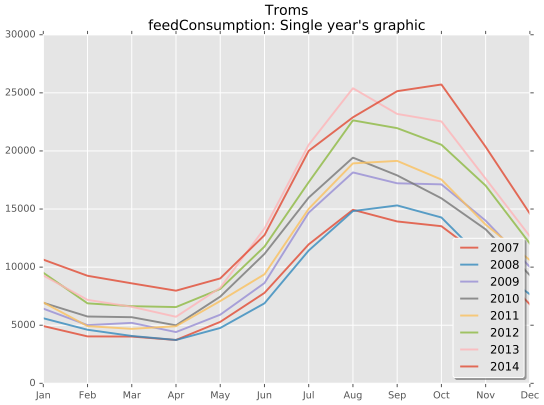


FIGURE 7.3: Annual consumption of feed trend in Troms.

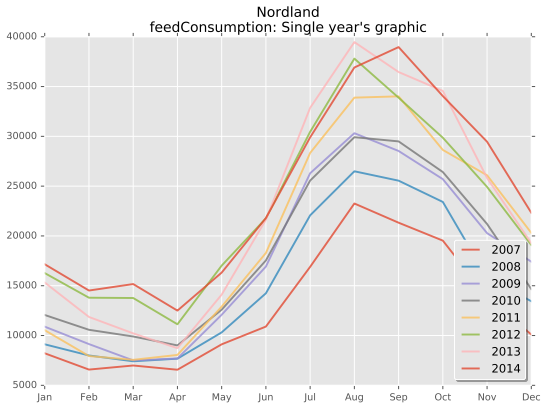


FIGURE 7.4: Annual consumption of feed trend in Nordland.

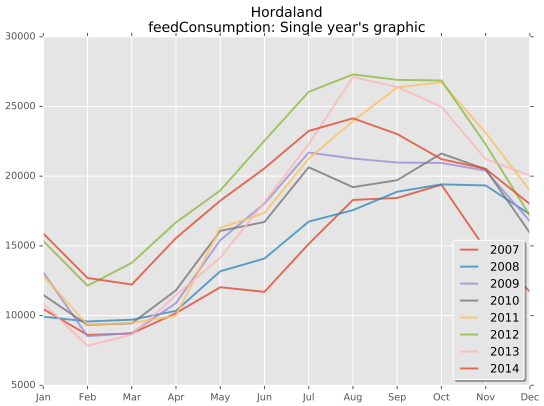


FIGURE 7.5: Annual consumption of feed trend in Hordaland.

The table [7.1] shows the Prediction results about the parameter "Feed Consumption" of the dataset "Norway". In particular are reported:

- ARIMA order used for the predictions (just on the right of the dataset name).
- Real 2015 consumption of feed values.
- Predicted 2015 consumption of feed values.
- MAPE between real and predicted values.

The Prediction results have been reported for some of the Norwegian counties, such as: Finnmark and Hordaland in the table [7.2].
Nordland and Troms in the table [7.3].

	Norway : (6, 1, 0)		
Pred Months	**Real**	**Pred**	**Error**
January 2015	109459	101149.42	7.59%
February 2015	86701.5	83164.98	4.08%
March 2015	87614.3	78983.87	9.85%
April 2015	84920.3	91419.33	7.65%
May 2015	94545.2	114388.79	20.99%
June 2015	112220	141620.77	26.20%
July 2015	152023	166470.71	9.50%
August 2015	186931	181705.78	2.80%
September 2015	181735	184349.66	1.44%
October 2015	177401	173328.08	2.30%
November 2015	153730	152941.90	0.51%
December 2015	126324	129326.56	2.38%

TABLE 7.1: Comparison between real 2015 "feed consumption" values and their prediction, with the corresponding MAPE for the counties: Norway

Pred Months	Finnmark : (6,1,0)			Hordaland : (8, 0, 0)		
	Real	**Pred**	**Error**	**Real**	**Pred**	**Error**
January 2015	7571.038	6902.50	8.83 %	15039.948	16199.46	7.71%
February 2015	4947.371	4892.92	1.10%	12432.481	14638.11	17.74%
March 2015	4686.465	5171.14	10.34%	12523.293	14596.72	16.56%
April 2015	4263.749	6309.80	47.99%	14231.265	15673.42	10.13%
May 2015	5472.568	8927.99	63.14%	14543.612	17305.97	18.99%
June 2015	8270.913	11867.40	43.48%	15314.691	18694.21	22.07%
July 2015	12356.162	13499.17	9.25%	19617.51	19996.28	1.93%
August 2015	15877.981	14484.53	8.78%	26429.183	20583.11	22.12%
September 2015	15382.371	13848.08	9.97%	28152.314	20524.85	27.09%
October 2015	15039.109	12583.28	16.33%	26869.594	19565.15	27.18%
November 2015	11453.453	10842.37	5.34%	23914.498	18086.76	24.37%
December 2015	9450.334	8898.79	5.84%	20332.347	16519.30	18.75%

TABLE 7.2: Comparison between real 2015 "feed consumption" values and their prediction, with the corresponding MAPE for the counties: Finnmark, Hordaland

Pred Months	Nordland : (6, 0, 0)			Troms : (2, 0, 0)		
	Real	**Pred**	**Error**	**Real**	**Pred**	**Error**
January 2015	20077.7	16178.44	19.42%	12534.708	8631.63	31.14%
February 2015	15478.6	11811.61	23.69%	9098.07	5043.87	44.56%
March 2015	15477.5	9251.12	40.23 %	9381.861	4951.90	47.22%
April 2015	12938.3	9571.32	26.02%	9493.93	6484.26	31.70%
May 2015	15237.5	11685.79	23.31%	11215.47	10884.56	2.95%
June 2015	20228.9	15136.22	25.18%	16370.732	16048.17	1.97%
July 2015	29049.6	18922.35	34.86%	21041.845	20529.13	2.44%
August 2015	37051.1	22044.15	40.50%	26981.845	23438.90	13.13%
September 2015	35364.5	24037.46	32.03%	25229.126	23317.43	7.58%
October 2015	33985.7	24545.52	27.78%	27327.632	21126.09	22.69%
November 2015	28439.4	23736.48	16.54%	22476.374	17330.05	22.90%
December 2015	21447.6	22001.57	2.58%	17239.549	13037.86	24.37 %

TABLE 7.3: Comparison between real 2015 "feed consumption" values and their prediction, with the corresponding MAPE for the counties: Nordland, Troms

Chapter 8

Discussion and Evaluations

The current 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.

8.1 Evaluation and limitations of the study

This study has a number of possible limitations, mainly due to:

- A lack of background knowledge about Data Science procedures.
- A lack of background knowledge about Salmon farming in Norway.
- Relatively short time available for this work.
- Limited availability of data sources.

"No one expects science to be perfect the first time and while your peers can be highly critical, no one's work is beyond limitations. Our knowledge base is built on uncovering each piece of the puzzle, one at a time, and limitations show us where new efforts need to be made. So much like peer review, don't think of limitations as being inherently bad, but more an opportunity for a new challenge. In the end, your limitation may be someone else's inspiration." [15]

During this work I got more and more knowledge and experience mainly about the fields reported above (Data Science procedures and Salmon farming in Norway), and it allowed to get anyway some positive and useful results that provide 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. All the documented steps of this works allow to have a complete overview of the general Data Science process.

Furthermore, the resulting Python systems and the corresponding evidences are showing which modules and packages are provided by Python in order to analyze, display and forecast data values. The high reusability and automation levels of the implemented system allow to easily reuse it in order to apply the same analysis, displaying and forecasting on a different dataset that contains data from a different area of interest.
On the other side the current system is probably efficient and productive for a personal use. That's because is not provided an implemented GUI, and to customize the system's outputs you have to have some basic knowledge of Python language, that could be a sort of limitation for several people, but it could also be considered like a kind of incentive for people to get to know this powerful programming language.

The approach used for this thesis didn't provide any kind of results that can be considered as new informations about the Norwegian salmon farming field, but it provided several ideas and discussion's starting points for further works. That's because this thesis was mainly focused on the Data Science process, Python utilities and to find out observations for future researches instead of the information extraction process itself.

Even that, as reported in the Results Overview chapter, several systems, evidences and values have been calculated and reported during this work.

In the next sections are reported evaluations and limitations about this study, and then the discussions that I considered most relevant and interesting about this work, which would also be useful for further works.

8.2 Considerations about implemented Evaluation System

The following resulting table reveals several informations.
The column "Evaluation MAPE" is the average MAPE of the forecasted values during the Evaluation System with the reported ARIMA order.
The column "Forecast MAPE" represents the average MAPE of the 12 values predicted in the future, already reported in the results. [7.1]

County	**Parameter**	**ARIMA Order**	**Evaluation MAPE**	**Forecast MAPE**
Finnmark	feedConsumption	(6, 1, 0)	13.771%	19.20%
Hordaland	feedConsumption	(8, 0, 0)	6.811%	17.89%
Troms	feedConsumption	(2, 0, 0)	11.593%	21.05%
Nordland	feedConsumption	(6, 0, 0)	12.741%	26.01%
Norway0714	feedConsumption	(6, 1, 0)	7.296%	7.94%

Table 8.1: Comparison between Evaluation MAPE and Prediction MAPE

From the values contained in the table reported above is possible to see that the average MAPE for the forecasted future values (**Forecast MAPE**) is much higher than the one reported from the evaluation process (**Evaluation MAPE**) .
There is just one particular case where the Evaluation MAPE and Forecast MAPE values are really close, that is the one about the Norway dataset.

At this point, would be extremely useful to discover why the predicted future values about the dataset 'Norway0714' are much more accurate and why their average MAPE is really close to the one calculated during the evaluation procedure.

In order to understand the reason, certain analysis have been made through the resulting evidences. Below here are reported considerations about it:

- Checked the annual feed consumption trend during different years for the current datasets, that is possible to check out in the reported graphics [7.1]. Were not found any kind of big differences between the Norway's trend compared with the others.

- Checked correlation coefficients between different years of the feed consumption values for each tested dataset, but there were not any kind of significant high/low correlation levels with the years before.

- Tried to execute the prediction system with settings that are different by the one suggested in the Evaluation system results. Found that for some particular dataset is possible to get better predictions with a different configuration, like for example: Prediction system about "feed Consumption" in Nordland executed with the suggested configuration (6,0,0) gave a average MAPE value equal to 26.01% for the predicted values. If the Prediction system is tested on the same input but fitting the ARIMA model with the order (6,1,0) the average MAPE value decrease to 16.82% for the predicted values.
  It's possible to clearly see this different watching at the graphics reported in the following page: [8.1] and [8.2].

This shows that the initial Evaluation system is not that accurate and reliable for each kind of input dataset. For this reason, in order to improve it during further works, is strongly suggest to try the "Box-Jenkinks method" for determine the best ARIMA order's parameters, that would probably be better and more specific for each single type of dataset.

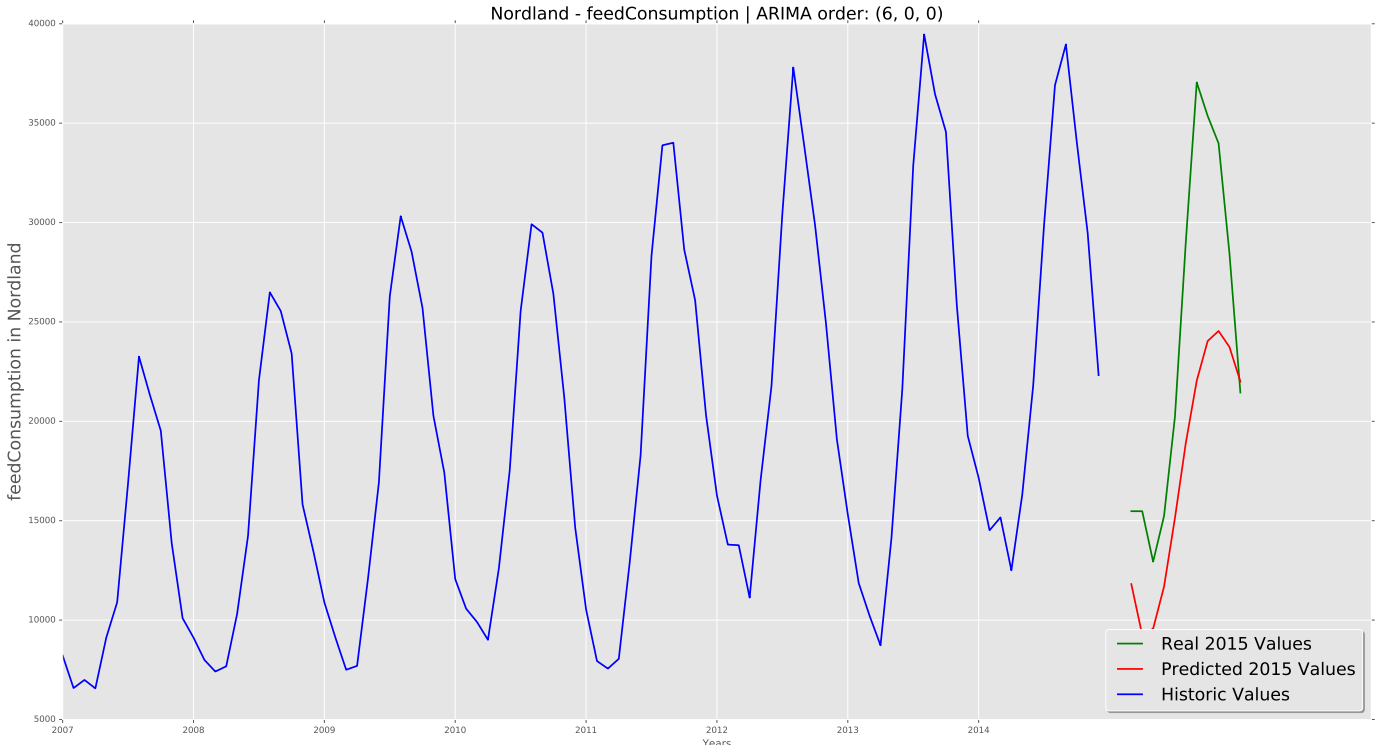


FIGURE 8.1: Predictions of 2015 feed consumption values in Nordland using ARIMA model fitted with the order suggested by the evaluation system results (6,0,0).
It provides an average MAPE of 26.01% between the real and predicted 2015 values.

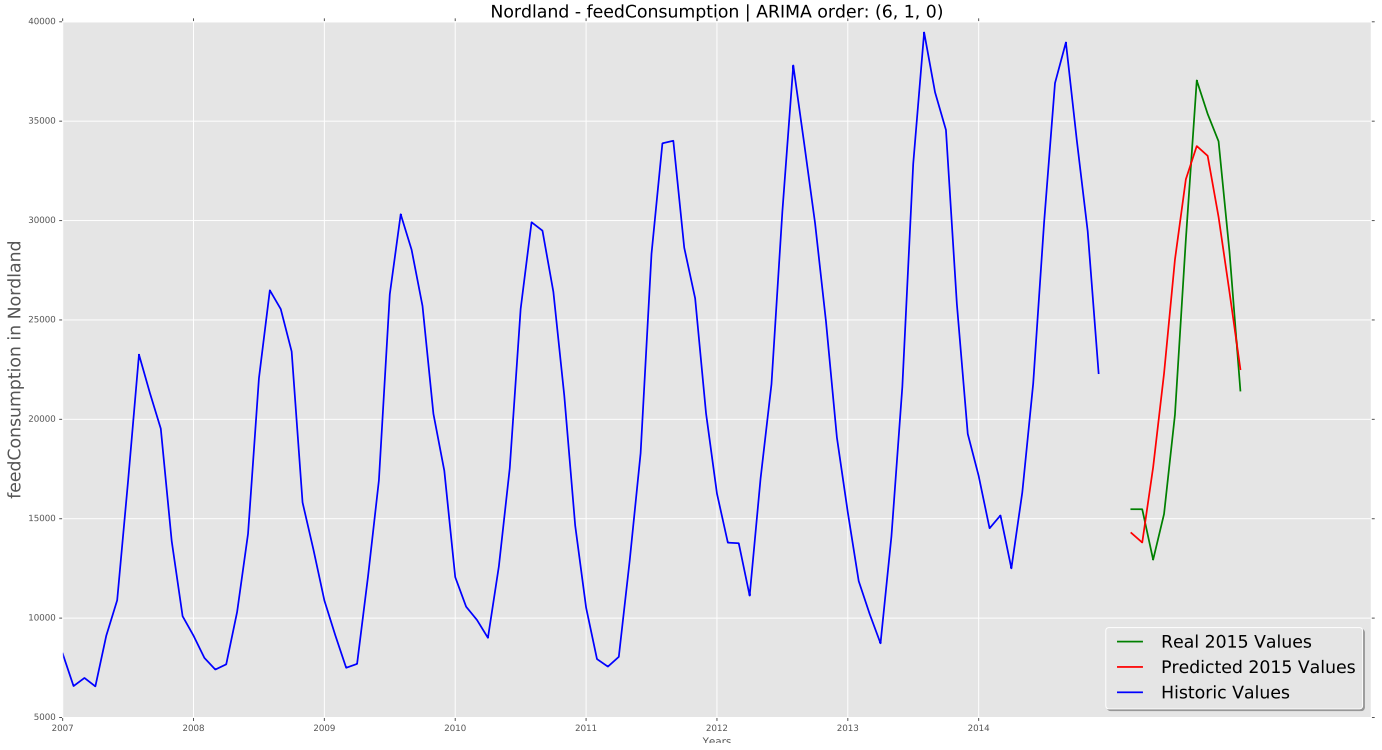


FIGURE 8.2: Predictions of 2015 feed consumption values in Nordland using ARIMA model fitted with an order manually chosen (6,1,0).
It provides an average MAPE of 16.82% between the real and predicted 2015 values.

8.3 Improvements for feed consumption forecasting

During the implementation of this work, was also noticed a strong relation between the two parameters "Sea Average Temperature" and "Feed Consumption". I decided to focus on this particular correlation because, also if it's already well known that is possible to feed more the salmon when the temperature is higher, it could be very useful for further predictions about feed consumption values, since the sea average temperature would be considered like a parameter to use for improve the final results.

This correlation is significant for all the Norwegian counties and it's clearly possible to see it with the example graphic[1] reported below here.

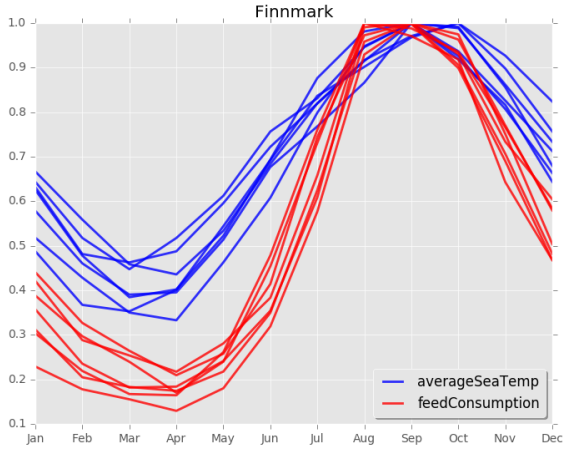


FIGURE 8.3: Comparison between average sea temperature and feed consumption in Finnmark

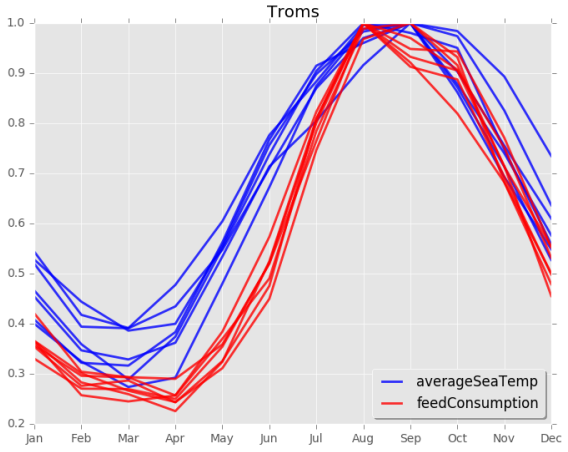


FIGURE 8.4: Comparison between average sea temperature and feed consumption in Troms

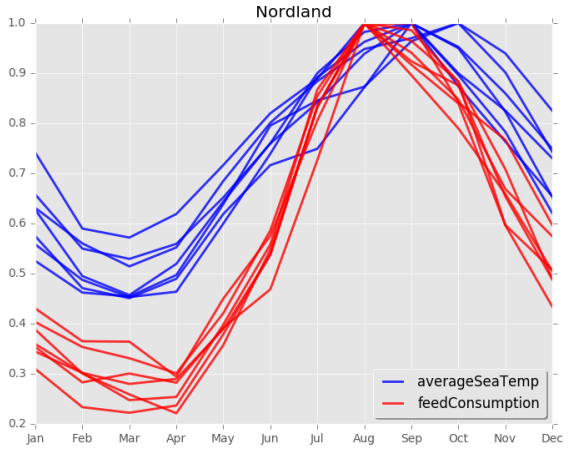


FIGURE 8.5: Comparison betwen average sea temperature and feed consumption in Nordland

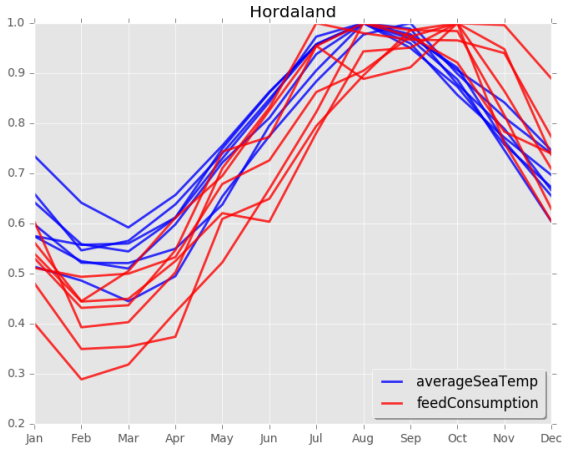


FIGURE 8.6: Comparison between average sea temperature and feed consumption in Hordaland

[1]The graphics reported above have been displayed with a Python system implemented during this study, that allows to display different parameters in a normalized range. The system has not been reported in the thesis work, but is possible to find it here named "Analysis.py":
`https://github.com/Sprea22/Personal_Utilities`

The following graphics allow to have a very clear overview of what just written above.

- In the first graphic is reported the average sea temperature, where blue means lower temperature and red higher temperature.
- In the second graphic is reported the feed consumption per biomass, where red means an higher consumption and blue a lower one.

So it's clearly possible to see how the average sea temperature and the feed consumption per biomass have a significant correlation for every single Norwegian county.

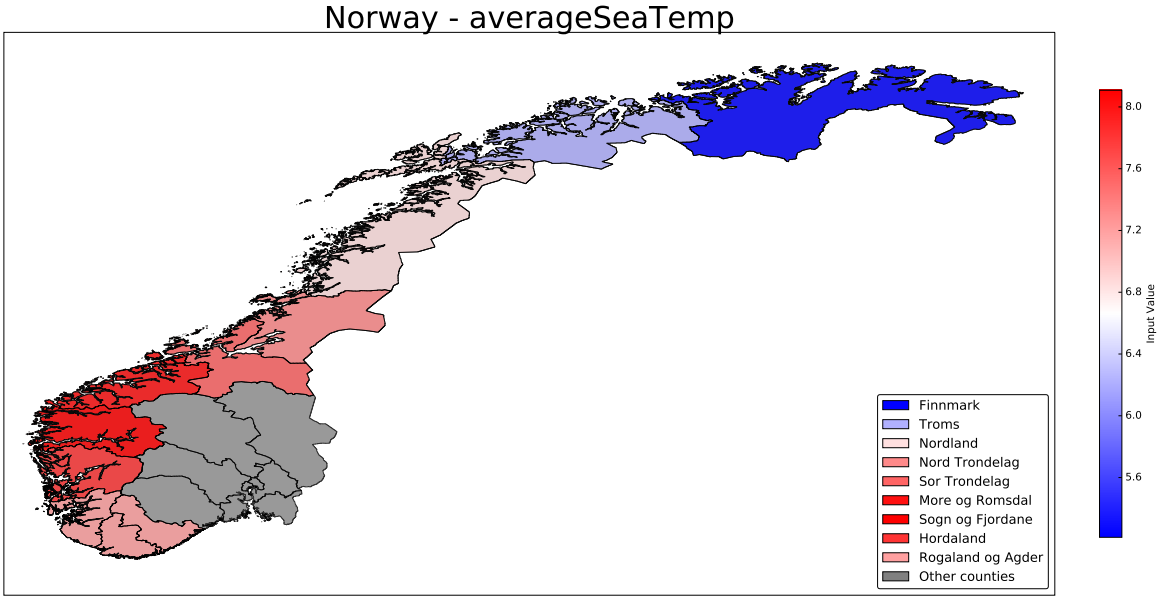


FIGURE 8.7: Monthly average sea temperature from 2007 to 2014 in Norway

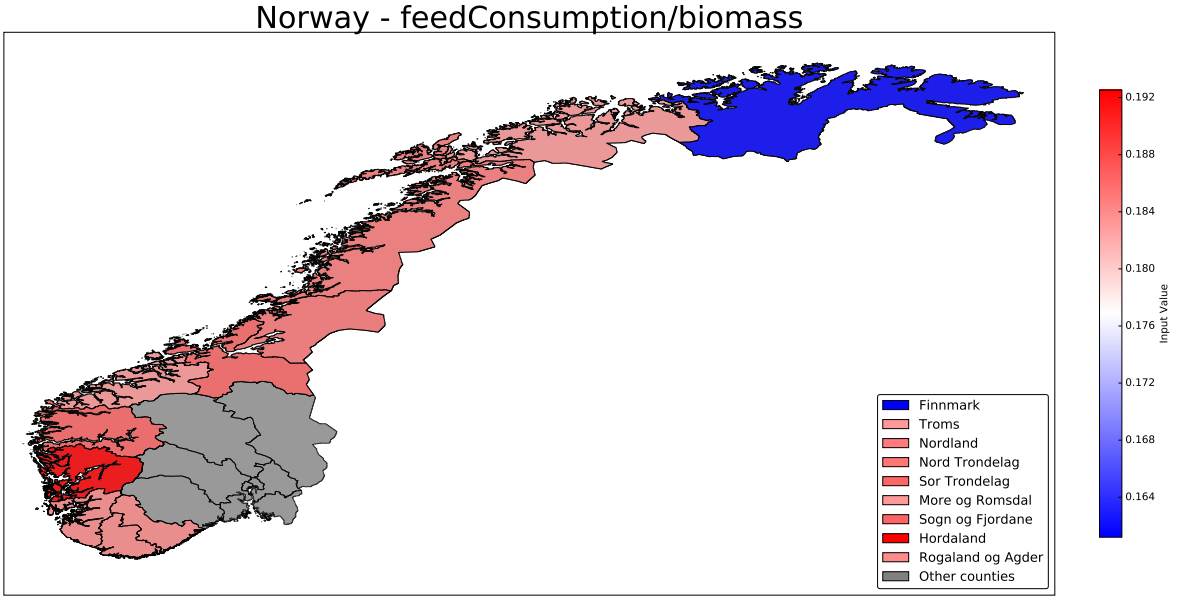


FIGURE 8.8: Monthly average feed consumption per biomass from 2007 to 2014 in Norway

Furthermore, is even possible to check the correlation coefficient values reported in the correlation matrix below here. These graphic represent the correlation coefficient between different parameter about the same dataset (county in this case).
It's possible to see how the "feedConsumption" and "averageSeaTemp" correlation value is representing an high correlation between the two parameters in example datasets reported here and, if you check through the other counties graphics, this correlation is valid for all the Norwegian counties.

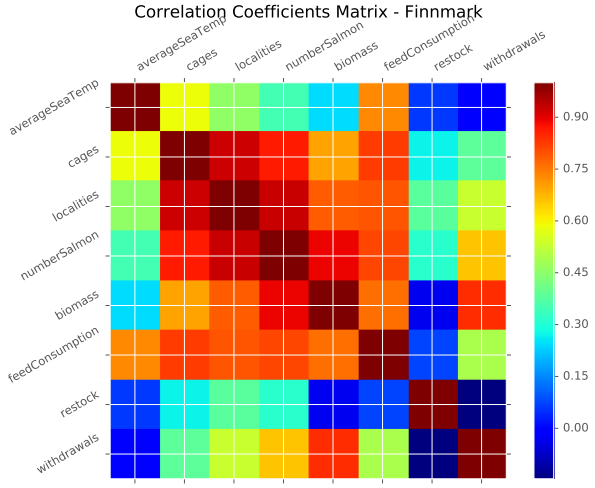


FIGURE 8.9: Correlation matrix between all the parameters of the dataset about Finnmark

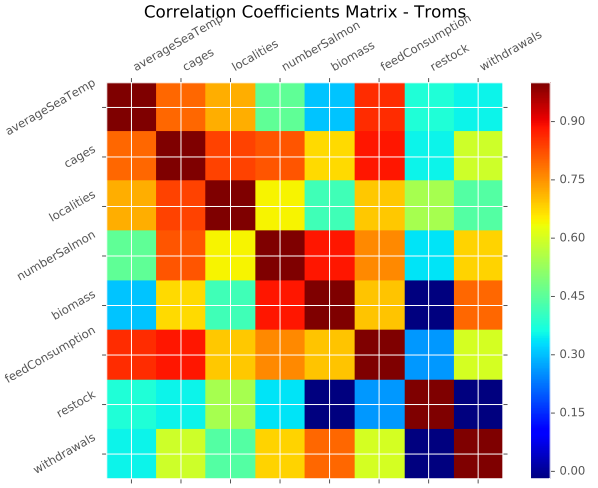


FIGURE 8.10: Correlation matrix between all the parameters of the dataset about Troms

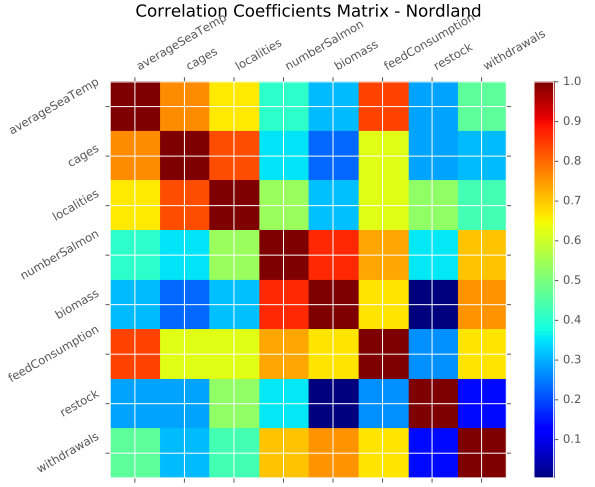


FIGURE 8.11: Correlation matrix between all the parameters of the dataset about Nordland

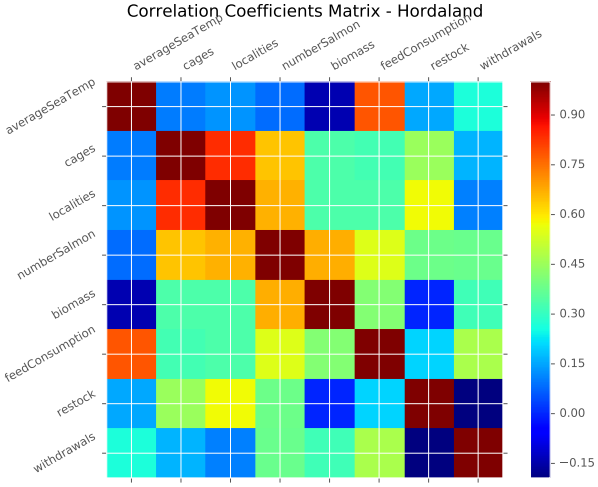


FIGURE 8.12: Correlation matrix between all the parameters of the dataset about Hordaland

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"[1].

- **System as a service**
  Would be really interesting to investigate about a possible way to provide this kind of analysis, displaying and forecasting systems like a service, such as this following simple idea:

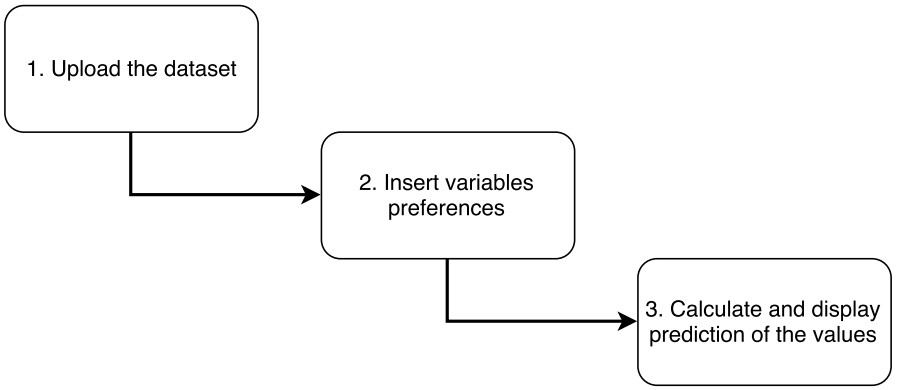


FIGURE 9.1: General idea about Prediction Systems as a service.

[1]Link to Plotly Library : `https://plot.ly/python/`

- **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 for further works:

  – Improve this forecasting model, applying and studying the "Box-Jenkins method" [2] in order to improve the Evaluation system and get better predictions of future value.

  – Research about a new forecasting model, in particular Artificial intelligence methods, such as "Artificial Neural Networks"[3].

  If you are interested in "salmon feed consumption" forecasting or similar topics I strongly suggest you to consider the discussion about the relation between "Feed Consumption" and "Average Sea Temperature", since it could be a significant help to the accuracy level of the system.

[2]Box-Jenkins method : `https://en.wikipedia.org/wiki/Box%E2%80%93Jenkins_method`
[3]ANN link: `https://en.wikipedia.org/wiki/Artificial_neural_network`

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.

```
1 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.

```
1 import sys
```

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

```
1 import pandas as pd
```

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

```
1 import numpy as np
```

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

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

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

```
1 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.

```
1  def create_single_overview(cols, rows, dest, width, height, listofimages):
2      thumbnail_width = width//cols
3      thumbnail_height = height//rows
4      size = thumbnail_width, thumbnail_height
5      new_im = Image.new('RGB', (width, height))
6      ims = []
7      for p in listofimages:
8          im = Image.open(p)
9          im.thumbnail(size)
10         ims.append(im)
11     i = 0
12     x = 0
13     y = 0
14     for col in range(cols):
15         for row in range(rows):
16             new_im.paste(ims[i], (x, y))
17             i += 1
18             y += thumbnail_height
19         x += thumbnail_width
20         y = 0
21     if dest==0:
22       script_dir = os.path.dirname(__file__)
23       results_dir = os.path.join(script_dir, "Results/" + sys.argv[1]+"/"+
     sys.argv[2]+"/")
24       if not os.path.isdir(results_dir):
25          os.makedirs(results_dir)
26         new_im.save(results_dir+"/"+ sys.argv[1] +"_"+sys.argv[2]+"
     _Graphics_Overview.jpg")
27         new_im.show()
28     if dest==1:
29       script_dir2 = os.path.dirname(__file__)
30       results_dir2 = os.path.join(script_dir2, "Results/" + sys.argv[1]+"/
     Total_Evidences/Single_Inputs")
31       if not os.path.isdir(results_dir2):
32          os.makedirs(results_dir2)
33         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.

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

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

```
1 pyplot.tight_layout()
2 saveFigure("_months_Matrix.jpg")
3 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.

```
1 listofimages=["Results/" + sys.argv[1]+"/"+sys.argv[2]+"/"+sys.argv[1]+"_"+
      sys.argv[2]+"_Total.jpg",
2               "Results/" + sys.argv[1]+"/"+sys.argv[2]+"/"+sys.argv[1]+"_"+
      sys.argv[2]+"_years_Matrix.jpg",
3               "Results/" + sys.argv[1]+"/"+sys.argv[2]+"/"+sys.argv[1]+"_"+
      sys.argv[2]+"_years.jpg",
4               "Results/" + sys.argv[1]+"/"+sys.argv[2]+"/"+sys.argv[1]+"_"+
      sys.argv[2]+"_months_Matrix.jpg"]
5
6 create_single_overview(4, 1, 1, 3200, 600, listofimages)
7 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.

```
1 import pandas as pd
```

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

```
1 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.

```
1 import sys
```

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

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

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

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

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.

```
1 temp = []
2 for i in series.columns:
3     index = sys.argv[1]+"-"+i
4     tempSeries = pd.read_csv("Results/"+sys.argv[1]+"/"+i+"/"+sys.argv[1]+"_
      "+i+"_AngCoeff.csv", header=0)
5     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".

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

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

```
1 pyplot.yticks(x,series.columns)
2 pyplot.title("Normalized Angular coefficients - " + sys.argv[1])
3 pyplot.tight_layout()
4 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]

```
1 import os
2 import sys
3 import matplotlib
4 import pandas as pd
5 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. Furthermore, the class "shapereader" provides an interface for accessing the contents of a shapefile.

```
1 import cartopy.crs as ccrs
2 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.

```
1 import matplotlib.cm as cmx
2 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.

```
1 def add_geom(axes, shapeInput, labelInput, colorInput):
2     axes.add_geometries(shapeInput, ccrs.Robinson(), edgecolor='black',label
      = labelInput, facecolor=colorInput, alpha=0.8)
3     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.

```
1 fname = 'Datasets/NOR/NOR_adm1.shp'
2 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.

```
1 dataInput = sys.argv[1]
2 inputSeries = pd.read_csv("Datasets/countiesAverages.csv")
3 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.

```
1 minValues = min(inputValues)
2 maxValues = max(inputValues)
3 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()
```

Appendix D

Prediction System Implementation code

D.1 Common libraries

All the libraries reported below here are used both from the Evaluation system and the Prediction system. In particular, these initial libraries have already been used during the previous system implementations, is possible to check out their utilities here: [A.1]

```
1 import os
2 import sys
3 import numpy as np
4 import pandas as pd
5 from pandas import Series
```

This "warnings" library is used just to warn programmers about something wrong or also to suppresses repeated warning from the same source.

```
1 import warnings
```

This following library provides an ARIMA model, that is basically a class of statistical models for analyzing and forecasting time series data.[1]

```
1 from statsmodels.tsa.arima_model import ARIMA
```

[1] ARIMA_model source code:
http://www.statsmodels.org/0.6.1/_modules/statsmodels/tsa/arima_model.html#ARIMA

D.2 Common methods

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

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

The following method allows to calculate the MAPE between a list of values and an another one. In particular, MAPE [2] is a measure of prediction accuracy of a forecasting method in statistics, for example in trend estimation. It usually expresses accuracy as a percentage.

```
1  def mean_absolute_percentage_error(y_true, y_pred):
2      try:
3          rng = len(y_true)
4          diff = []
5          for i in range(0,rng):
6              diff.append(y_true[i] - y_pred[i])
7              diff[i] = diff[i] / y_true[i]
8          abs = np.abs(diff)
9          mn = np.mean(abs)
10         percentageError = mn * 100
11     except:
12         rng = 0
13         abs = np.abs((y_true-y_pred)/y_true)
14         percentageError = abs * 100
15     return percentageError
```

[2] MAPE formula and description:
https://en.wikipedia.org/wiki/Mean_absolute_percentage_error

D.3 Evaluating System

This system uses the common library reported above: [D.1]

This system uses the common methods reported above: [D.2]

This method is used for test all the ARIMA orders required, that are basically made by the combination of the input lists "p_values" , "d_values", "q_values". During this code is possible to get the resulting MAPE for each single configuration from the method "evaluate_arima_model()" and then write them down in a report document.

```
1
2  def evaluate_models(dataset, p_values, d_values, q_values):
3      dataset = dataset.astype('float32')
4      best_score, best_cfg = float("inf"), None
5      filename = "Results_Forecast/"+sys.argv[1]+"/"+sys.argv[2]+"_MAPE.csv"
6      if not os.path.exists(os.path.dirname(filename)):
7          os.makedirs(os.path.dirname(filename))
8      with open(filename, "w") as myfile:
9          myfile.write("P, D, Q, MAPE\n")
10     for p in p_values:
11         for d in d_values:
12             for q in q_values:
13                 order = (p,d,q)
14                 try:
15                     mape = evaluate_arima_model(dataset, order)
16                     with open(filename, "a") as myfile:
17                         myfile.write('%d, %d, %d, %.3f%% \n' % (p,d,q,mape))
18                     if mape < best_score:
19                         best_score, best_cfg = mape, order
20                     print('ARIMA%s MAPE=%.3f%%' % (order,mape))
21                 except:
22                     print('ARIMA%s MAPE=Nil' % str(order))
23                     continue
24     print('Best ARIMA%s MAPE=%.3f%%' % (best_cfg, best_score))
```

During this method, the input dataset is split in two:

- 66% for the initial training dataset
- 34% for the test dataset

Then the ARIMA model is fitted with the values contained in the training dataset, and a single prediction is made each iteration and stored in a list. This is so that at the end of the test set, all predictions can be compared to the list of expected values and an error score calculated. [16]
Then in this case a mean absolute percentage error is calculated and returned.

```
def evaluate_arima_model(X, arima_order):
    train_size = int(len(X) * 0.66)
    train, test = X[0:train_size], X[train_size:]
    history = [x for x in train]
    predictions = list()
    for t in range(len(test)):
        model = ARIMA(history, order=arima_order)
        model_fit = model.fit(disp=0)
        yhat = model_fit.forecast()[0]
        predictions.append(yhat)
        history.append(test[t])
    error = mean_absolute_percentage_error(test, predictions)
    return error
```

Once defined the needed methods, this system reads the input dataset and parameter decided by the user via shell. How is possible to see, the ARIMA parameters testing lists are already given in the code, but it's easily possible to change it, in order to add or remove configurations tested during this system. It's suggested that warning be ignored for this code to avoid a lot of noise form the runn procedure.

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

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

warnings.filterwarnings("ignore")
evaluate_models(series.values, p_values, d_values, q_values)
```

D.4 Prediction System

This system uses the common library reported above: [D.1]

This system uses the common library reported above: [D.2]

During the initial part of this system's code all the needed data values are read and formatted in the right way.
"series" contains the historic values of the current dataset.
"yearInput" contains the years of the current dataset's period.
"realValues" contains the real future value for the next year.
"order" contains the ARIMA order inserted by the user via shell.

```
series = pd.read_csv("Datasets/"+sys.argv[1]+".csv", usecols=[sys.argv[2]])
yearInput = pd.read_csv("Datasets/" + sys.argv[1]+".csv", usecols=[0])
realValues = pd.read_csv("Results_Forecast/"+sys.argv[1]+"/"+sys.argv[1]+"_
    "+sys.argv[2]+"_2015.csv")

realValuesData = realValues[sys.argv[2]].values
realValuesData = realValuesData.astype('float32')
dataset = series.values
dataset = dataset.astype('float32')

p_values = int(sys.argv[4])
d_values = int(sys.argv[5])
q_values = int(sys.argv[6])

order = (p_values, d_values, q_values)
warnings.filterwarnings("ignore")
```

Once read all the required data, the ARIMA model is fitted with the historic values of the current dataset, and it's used for forecast an arbitrary number of values in the future choosen by the user with the "sys.argv[3]".

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

The following rows of code were implemented just to report the results into a document. It allows to have a document with the real future value, predicted future value and the MAPE between each of them.

```
index = []
for i in range(1,int(sys.argv[3])+1):
   index.append(len(dataset) +i)
```

```
4  mape_list = []
5  for i in range(0,len(forecast)):
6      mape_list.append(mean_absolute_percentage_error(realValuesData[i],
       forecast[i]))
7  rows = zip(index, realValuesData, forecast, mape_list)
8  f = open("Results_Forecast/"+sys.argv[1]+"/"+sys.argv[1]+"_"+sys.argv[2]+"
       _futurePred.csv", 'w')
9  csv.writer(f).writerows(rows)
10 f.close()
```

Once elaborated and saved the results, this system will show a graphic of the historic data values and also of the real and predict future values.

```
1  realValues= pd.read_csv("Results_Forecast/"+sys.argv[1]+"/"+sys.argv[1]+"_"
       +sys.argv[2]+"_futurePred.csv", index_col=[0], usecols=[0,1])
2  predFuture = pd.read_csv("Results_Forecast/"+sys.argv[1]+"/"+sys.argv[1]+"_
       "+sys.argv[2]+"_futurePred.csv", index_col=[0], usecols=[0,2])
3
4  ax = pyplot.subplot(111)
5  ax.plot(realValues, "g", label='Real 2015 Values', linewidth=2)
6  ax.plot(predFuture, "r", label='Prediction 2015 values', linewidth=2)
7  ax.plot(series, "b", label='Historic values', linewidth=2)
```

This last part of the code were implemented just to customize the output graphic. It basically allows to display the current period's years on the xlabel and some other features, such as legend, title and full screen image once displayed.

```
1  ax.legend(loc='upper center', bbox_to_anchor=(0.5, 1.05), ncol=3, fancybox=
       True, shadow=True, fontsize=20)
2  years = []
3  j = 0
4  for i in range(len(yearInput)):
5      if j==11:
6          years.append(yearInput.values[i][0])
7          j=0
8      else:
9          j=j+1
10 x = range(0, len(yearInput.values))
11 pyplot.xticks(np.arange(min(x), max(x)+1, 12.0), years)
12 pyplot.xlabel("Years")
13 pyplot.ylabel(sys.argv[2]+" in "+sys.argv[1], fontsize=20)
14 manager = pyplot.get_current_fig_manager()
15 manager.resize(*manager.window.maxsize())
16 pyplot.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] Wikipedia. Cluster analysis — wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Cluster_analysis&oldid=782997586, 2017. [Online].

[5] Wikipedia. Statistical classification — wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Statistical_classification&oldid=781889759, 2017. [Online].

[6] Wikipedia. Database — wikipedia, the free encyclopedia, 2017. [Online].

[7] Quora. The data science process, 2014. [Online].

[8] Wikipedia. Regression analysis — wikipedia, the free encyclopedia, 2017. [Online].

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

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

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

[12] 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].

[13] 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].

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

[15] Senior Editor Amanda Hindle. How to write about your study limitations without limiting your impact. https://www.edanzediting.com/blogs/how-write-about-your-study-limitations-without-limiting-your-impact, January 23, 2015. Online.

[16] Jason Brownlee. How to grid search arima model hyperparameters with python. http://machinelearningmastery.com/grid-search-arima-hyperparameters-with-python/, January 18, 2017. Online.