UNIVERSITY OF TROMSØ

Initial approach to Data Science using Python: Norwegian Salmon farming analysis.

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

Declaration of Authorship

I, Andrea Spreafico, declare that this thesis titled, 'Initial approach to Data Science using Python: Norwegian Salmon farming analysis.' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: ______________________________

Date: ______________________________

"We did it, we bashed them, wee Potter's the one,
and Voldy's gone moldy, so now let's have fun!"

- Peeves

UNIVERSITY OF TROMSØ

Abstract

Faculty of Computer Science
Department of Computer Science

Computer Science

by Andrea Spreafico

Moonstone (also known as the wishing stone[1]) is found in a variety of colors. Its supposed magical effects include helping a person gain emotional balance. Since Harry spent much of book five emotionally unbalanced, it is perhaps fitting that he was forced to write an essay on the stone's use in Potions-making. It is a gemstone of medium value. Moonstones are a milky colour and shine very brightly, almost as though they are a source of their own light. They are a useful potion ingredient; powdered moonstones are used as an ingredient for the Draught of Peace and in several Love Potions. Powdered Moonstone is also an ingredient in in Potion No. 86 which is likely an experimental potion. Moonstones were also known to be among the gems set into Muriel's tiara.

Acknowledgements

I would like to express my very great appreciation to Susan Bones for the ...

I would also like to offer my special thanks to Cedric Diggory for...

My special thanks are extended to the staff of the Matron for...

My special thanks goes to Pomona Sprout for taking on this thesis work.

I am particularly grateful for the support and good times given by my friends, for...

To my family, for..., I am particularly grateful.

Advice given by Helga Hufflepuff has been a great help in...

To my beloved Ernie Macmillan for all the...

Contents

List of Figures

List of Tables

Abbreviations

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

For/Dedicated to/To my…

Chapter 1

Introduction

1.1 Aim of the study

Every single day in the world is produced a huge amount of data: some of this data, if they are analyzed and interpreted in the right way, could provide useful informations.
If we watch for example at the Aquaculture business in Norway is produced a big amount of data about every single locality or about national statistcs, but most of the time this data are not analyzed and difficult to understand.
The main purposes of this thesis are basically:

- Initial approach with data science.
- Python potential in data science field, in order to show how it works and what you could do using it.
- Test and show the data potential about Aquaculture business in Norway, in order to help people related with Aquaculture to get informations about it in a easy way.

For achieve the goals reported above, this thesis will provide:

- Implementation and description of a procedure that can be used for make a Python system able to do an automatic initial analysis of big datasets and also to display the obtained results.
- Implementation and description of a procedure that can be used for make a system implemented in Python able to predict future's values using a regression model.

1.2 Initial Goals

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

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

2) Increase accessibility and availability of the data.

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

3) Analyze and display the data.

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

4) Extract information from the data.

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

5) Prediction of values about the data.

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

6) Recommendations to future work and extra ideas.

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

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.
It includes theories drawn from many field within the broad areas of mathematics, statistics, information science and computer science.

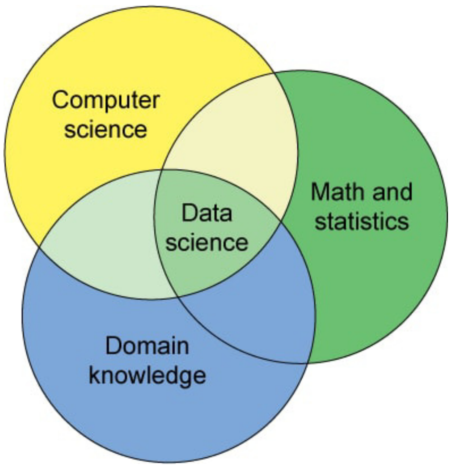


FIGURE 2.1: Data science concept

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

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

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

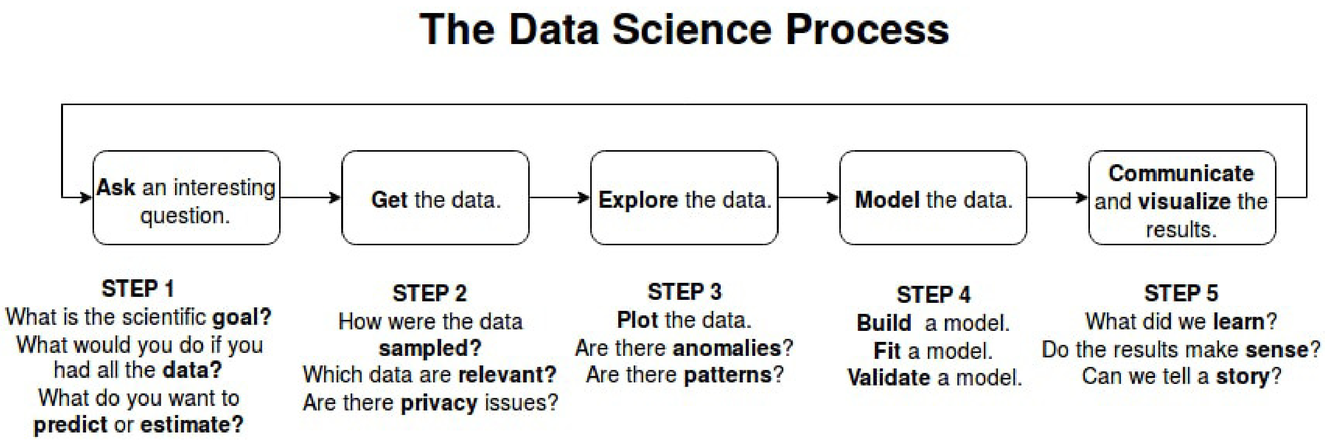


FIGURE 2.2: Data science process

2.2 Machine learning

This subfield of computer science gives ”computers the ability to learn without being explicitly programmed”.
Evolved from the study of pattern recognition and computational learning theory in artificial intelligence, machine learning explores the study and construction of algorithms that can learn from and make predictions on data.

There are several machine learning algorithm, each one of them is used for a different purpose.The following picture gives a general idea about which categories of algorithms are used and some specific types.

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.

” A time series is a sequence of observations taken sequentially in time. ” Quoted — Page 1, Time Series Analysis: Forecasting and Control.

Classic example of a time series dataset:
Time #1, observation
Time #2, observation
Time #3, observation

There are different goals depending on wheter we are interested in understanding a dataset or making predictions.

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

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

2.2.2 Autoregressive integrated moving average (ARIMA)

In statistics and econometrics, and in particular in time series analysis, an autoregressive integrated moving average (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).

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

Is the aquaculture business in Norway growing?

Aquaculture, also known as aquafarming, is the farming of fish, crustaceans, molluscs, aquatic plants, algae, and other aquatic organisms.

Aquaculture would be the future of fish: In 2030, according to the World Bank, aquaculture will supply:

- 93.6 Million tonnes of fish per year
- 25 percent less wild fish will be available
- 62 percent of the fish we eat will come from farms

2.4 Github Usage

GitHub is a code hosting platform for version control and collaboration. It lets you and others work together on projects from anywhere.

In this thesis it will be used like ”version control” since I'm the only person working on it, and I created on myself three repositories that will be very useful for better understand the work done with this thesis.

Chapter 3

Approach and Design

3.1 Development Flow

1st Phase: Data collection and validation

During this phase the most important thing is to gather as much as possible data, but they must be as much as possible reliable and useful since they are going to be indispensable for the next phases and in particular for the final results and conclusions. The data's reliability mainly depend by the kind of sources where you're able to mine.
Then you should customize the unstructured data that you collected.
This data's customizing has the main purposes of:

- Let the data structure be a summarize of all the data inputs previous collected.
- Let the new data structure be easier to access and read.
- Follow some kind of setting and standard needed in the system that will be implemented.

2nd Phase: Data Analysis and Displaying

During this phase the first thing that you're going to do is to decide some kind of analysis results that you would like to have.
Once you decided which kind of results you might reach, you will start with the analysis system implementation and meanwhile saving eviences of it.
Once the general analysis of the data is finished, and evidences have been collected, it's time to analyze it and try to extract information about it.

3rd Phase: Data Prediction
During this phase the main purpose is to predict some kind of useful data about the current dataset. To reach this goal, is first of all indispensable to choose a prediction system to implement.
Once the prediction system has been implemented, it's time to apply it on the current data and try to get as much evidences as possible.

4th Phase: Results, Discussion and Conclusions
During this phase of the work all the obtained results will be reported and discussed, trying to figure some useful conclusions.

5th Phase: Future Works
The last but not least phase is to watch at the future: try to figure out some other extra implementations about this thesis.

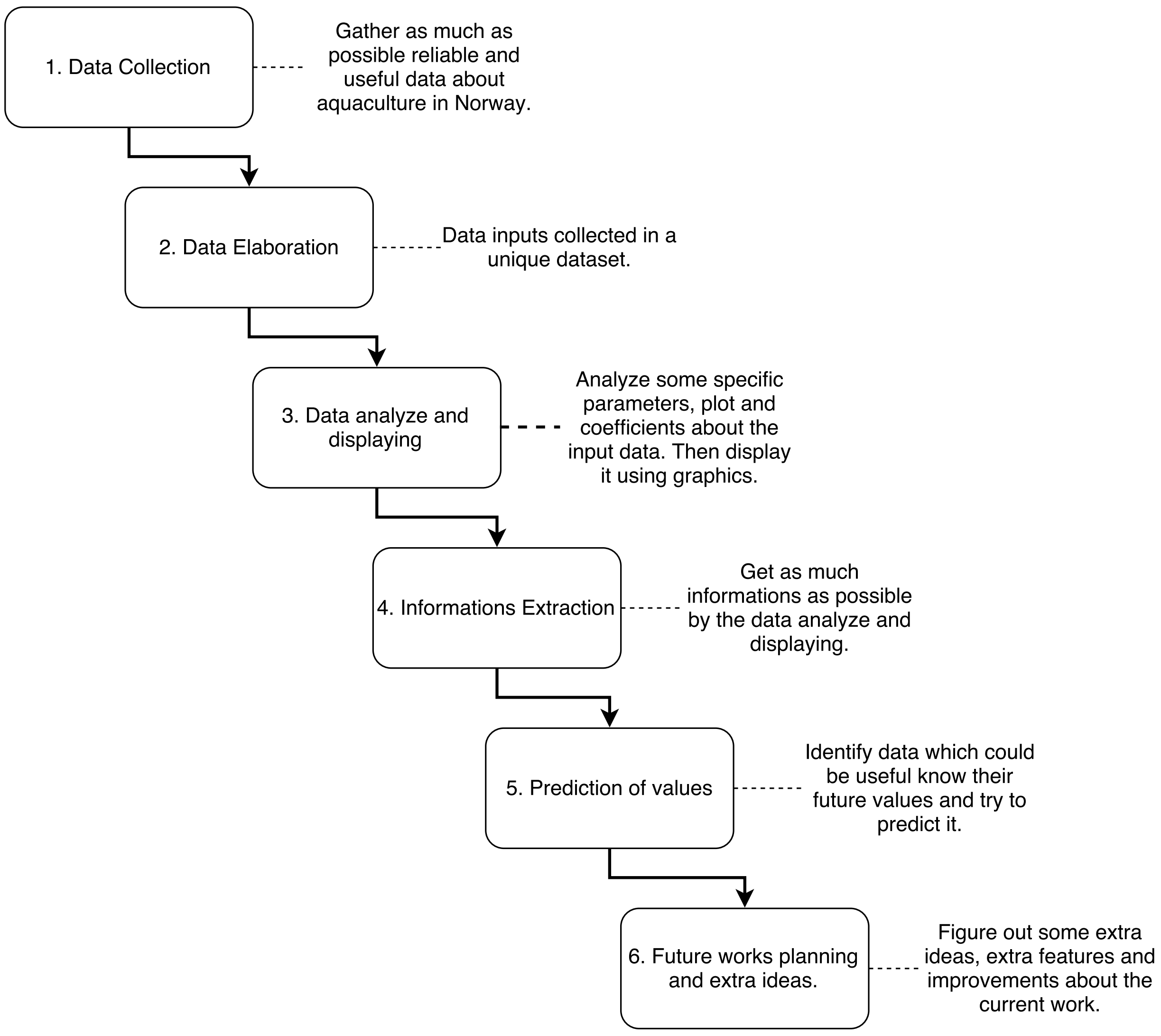

1. Data Collection
Gather as much as possible reliable and useful data about aquaculture in Norway.
2. Data Elaboration
Data inputs collected in a unique dataset.
3. Data analyze and displaying
Analyze some specific parameters, plot and coefficients about the input data. Then display it using graphics.
4. Informations Extraction
Get as much informations as possible by the data analyze and displaying.
5. Prediction of values
Identify data which could be useful know their future values and try to predict it.
6. Future works planning and extra ideas.
Figure out some extra ideas, extra features and improvements about the current work.

FIGURE 3.1: Plan flow chart

Chapter 4

Implementation

4.0.1 Implemented Systems repositories

The system that is going to be implemented during this thesis can be easily downloaded in order to test it and better understandhow it works.
The github repository of the implementation is the following:
https://github.com/Sprea22/Norway_County_Analyzer
Direct link for the already implemented Data Analyzer in Python.
https://codeload.github.com/Sprea22/Data_Analyzer_Python/zip/master

https://github.com/Sprea22/Forecasting_System_Python
Direct link for the already implemented Forecasting System in Python.
https://codeload.github.com/Sprea22/Forecasting_System_Python/zip/master

Chapter 5

Data collection

5.1 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.

5.1.1 Data from SINTEF Nord

Some of the data used during this thesis were provided from the team members of the eSushi project team at SINTEF Nord.
The data are about each single norwegian county and with the following details:

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

5.1.2 Data from Fiskeridir

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

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

It's also important to know that the following informations about this current data source:

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

5.1.3 Data from Indexmundi

Is possible to find data about fish (salmon) monthly price, Norwegian Krone per KG.

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

5.2 Increase accessibility and availability of data

In order to increase the accessibility and availability of the row data have been downloaded from the above reported sources, during this phase the main goals were:

- provide an accurate description (in English, since it was available just in Norwegian)
- Report the data in a standard and reusable standard (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

5.2.0.1 Dataset about Norway

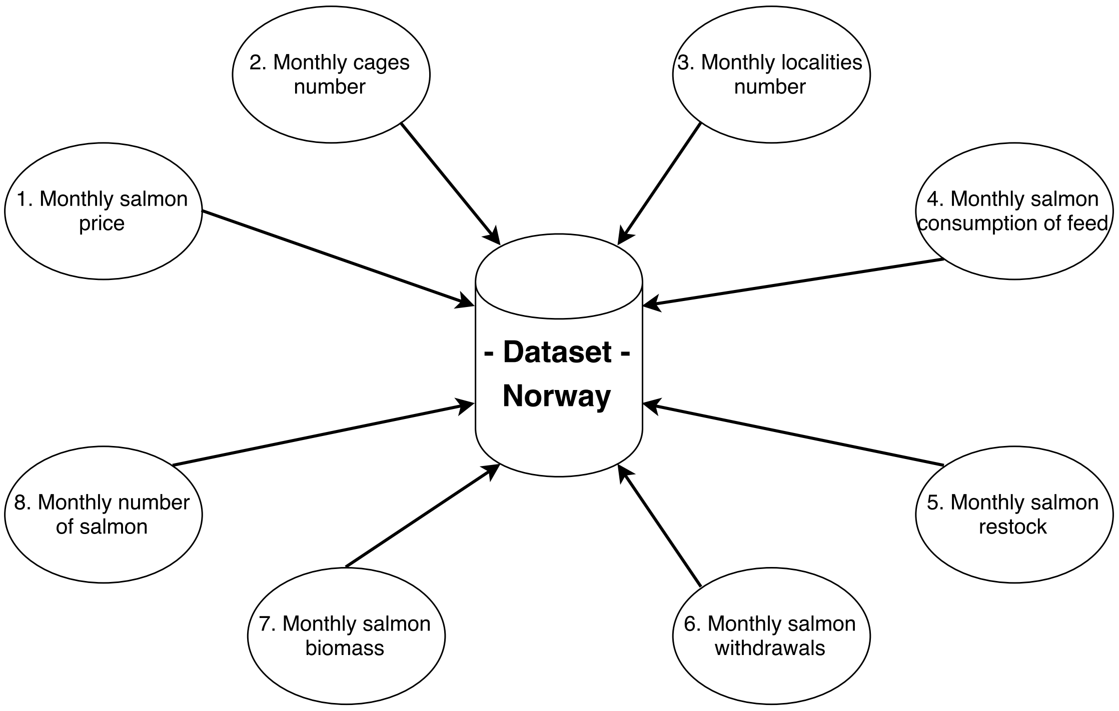

2. Monthly cages number
3. Monthly localities number
1. Monthly salmon price
4. Monthly salmon consumption of feed
- Dataset - Norway
8. Monthly number of salmon
5. Monthly salmon restock
7. Monthly salmon biomass
6. Monthly salmon withdrawals

FIGURE 5.1: 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

5.2.0.2 Dataset about each single county

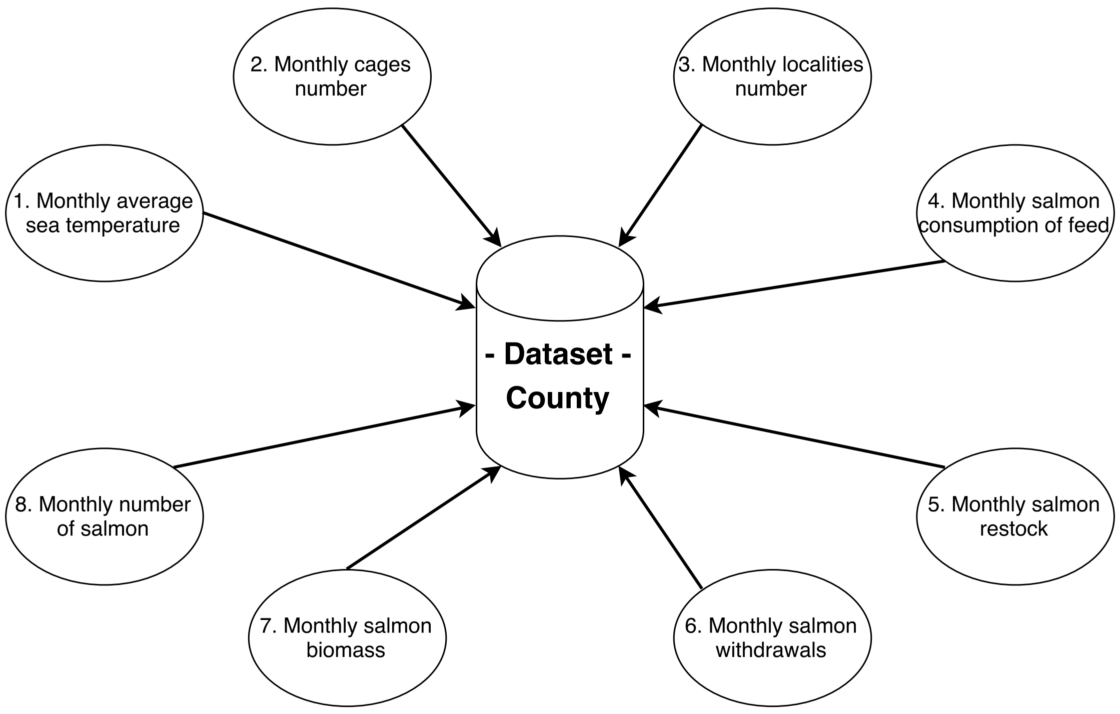

2. Monthly cages number
3. Monthly localities number
1. Monthly average sea temperature
4. Monthly salmon consumption of feed
- Dataset - County
8. Monthly number of salmon
5. Monthly salmon restock
7. Monthly salmon biomass
6. Monthly salmon withdrawals

Figure 5.2: Dataset structure.

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

Chapter 6

Analysis of the data

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

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

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

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

6.1 Requirements for reusability

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

- Monthly frequency of data values contained.

6.2 System requirements

It's important to remind that this phase can be implemented in different ways and with different programming language;
This proceure will describes the system implentation using Python, so be sure to have installed all the necessary for compile and execute Python code on your platform.

```
Current development environment:
Python version: 2.7.12
Linux kernel version number: Linux Asus 4.4.0-71-generic SMP
```

6.3 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.

6.3.1 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].

6.3.2 SIA section I: Total graphic for all the years

Goal:

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

Requirements:

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

Implementation:

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

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

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

Results:

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

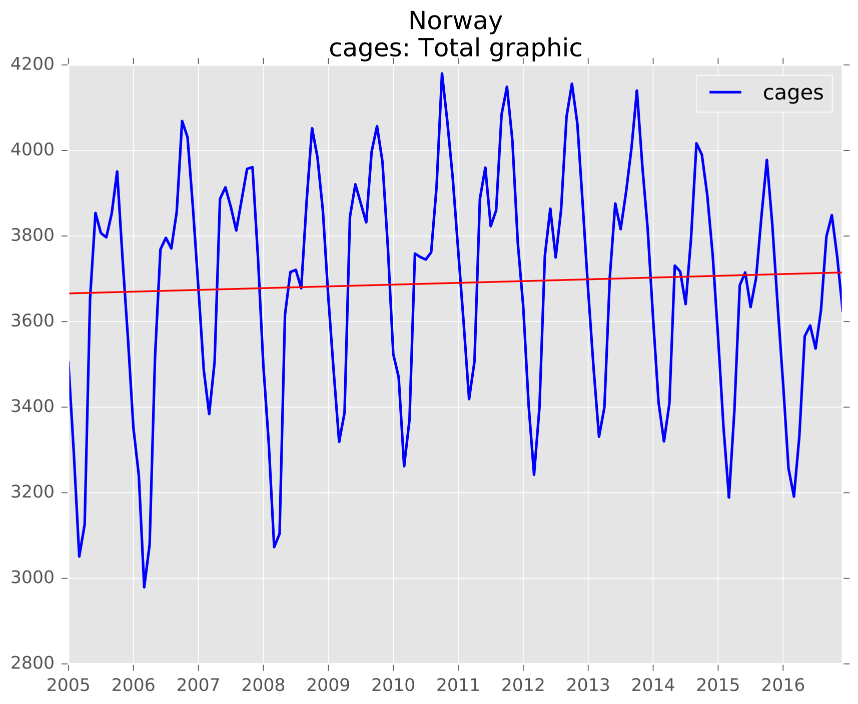


FIGURE 6.1: Total graphic about current input over the whole period.

6.3.3 SIA section II: Single graphics for each year

Goal:

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

Requirements:

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

Implementation:

To reach the current goal have been used two main libraries.
The "pandas" library allows to read the data values from the dataset and return it like "ndarray" type, then the library "pyplot" allows to display it on a graphic.

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

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

Results:

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

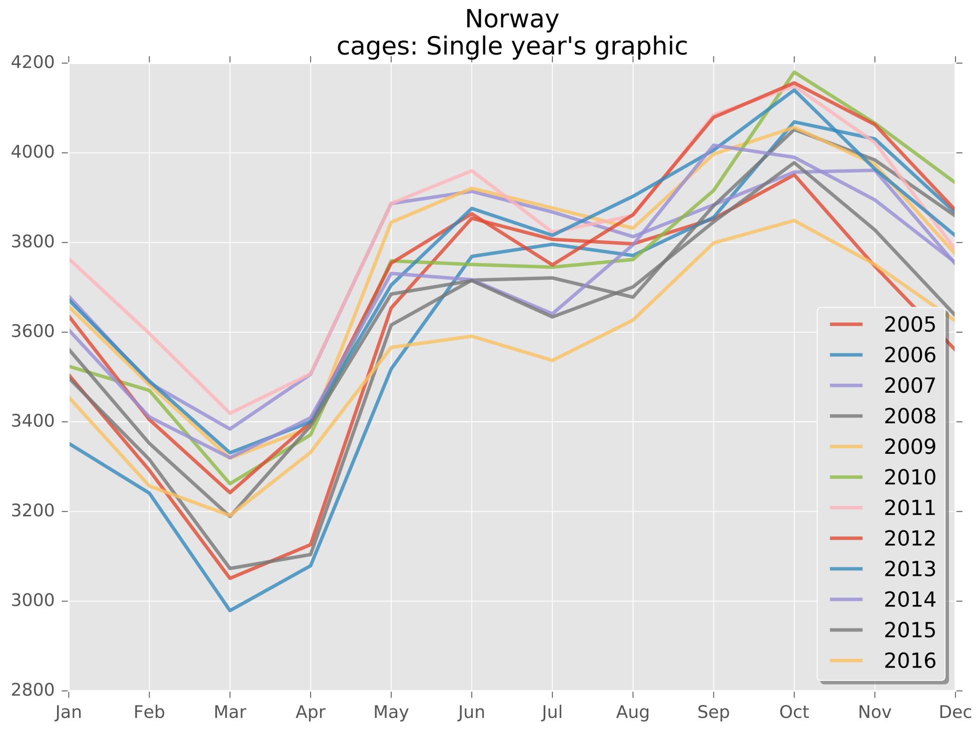


FIGURE 6.2: Graphics for each single year of the current input data.

6.3.4 SIA section III: Correlation matrix between years

Goal:

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

Requirements:

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

Implementation:

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

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

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

Results:

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

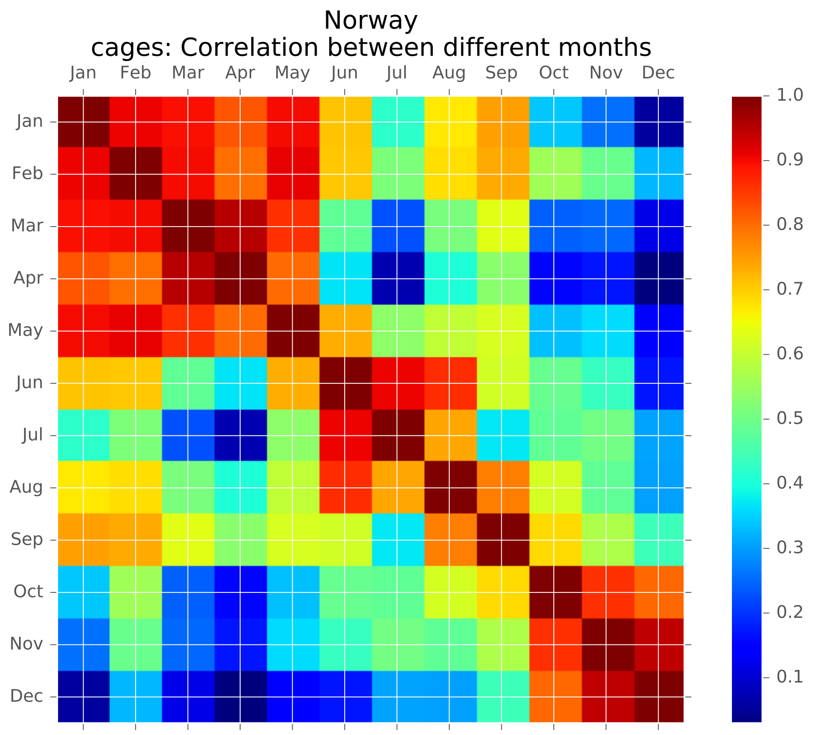


FIGURE 6.3: Correlation matrix between different months of the same input

6.3.5 SIA section IV: Correlation matrix between months

Goal:

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

Requirements:

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

Implementation:

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

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

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

Results:

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

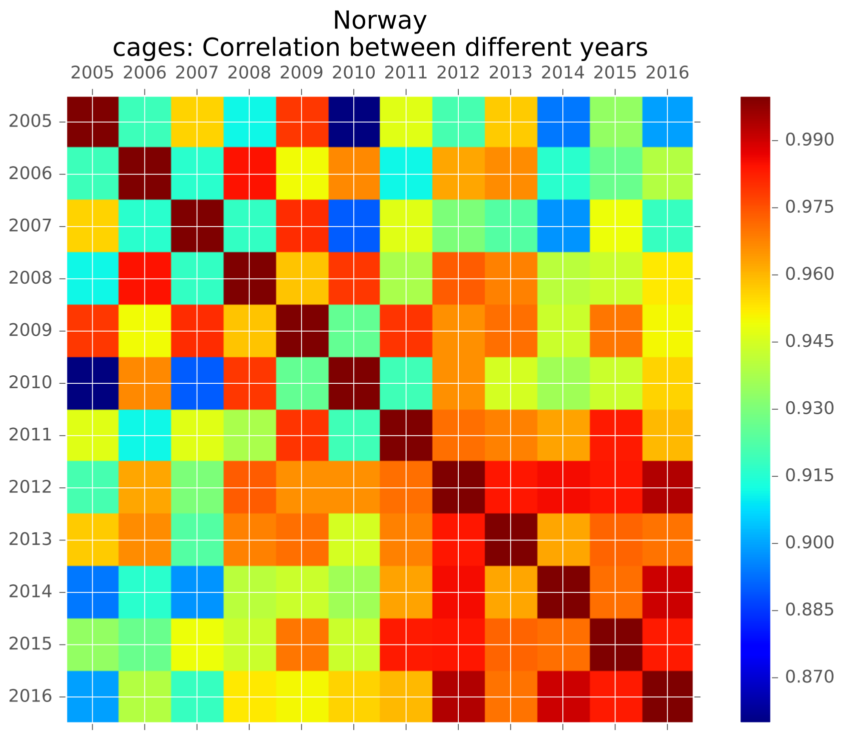


FIGURE 6.4: Correlation matrix between different years of the same input

6.3.6 SIA section V: Single overview

Goal:

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

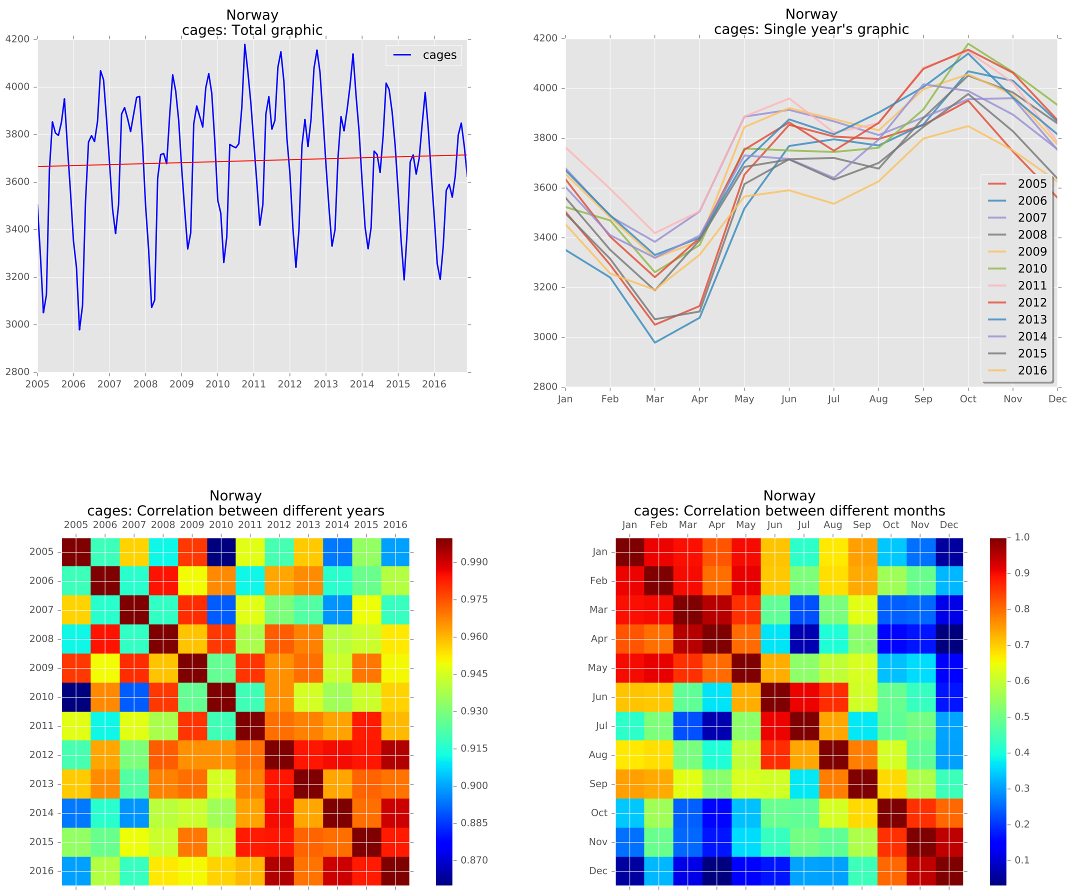
Implementation:

It's possible to check out the full ccommented code in the appendice: [A.7]

Requirements:

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

Results: With this part of the code it's possible to have a single overview image for the current input, that is basically showing and comparing all the graphics that have already been calculated about this input. It looks like this example:



6.4 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].

6.4.1 MIA: Imported libraries

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

6.4.2 MIA section I: Total Correlation Coefficients

Goal:

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

Requirements:

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

Implementation:

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

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

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

Results:

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

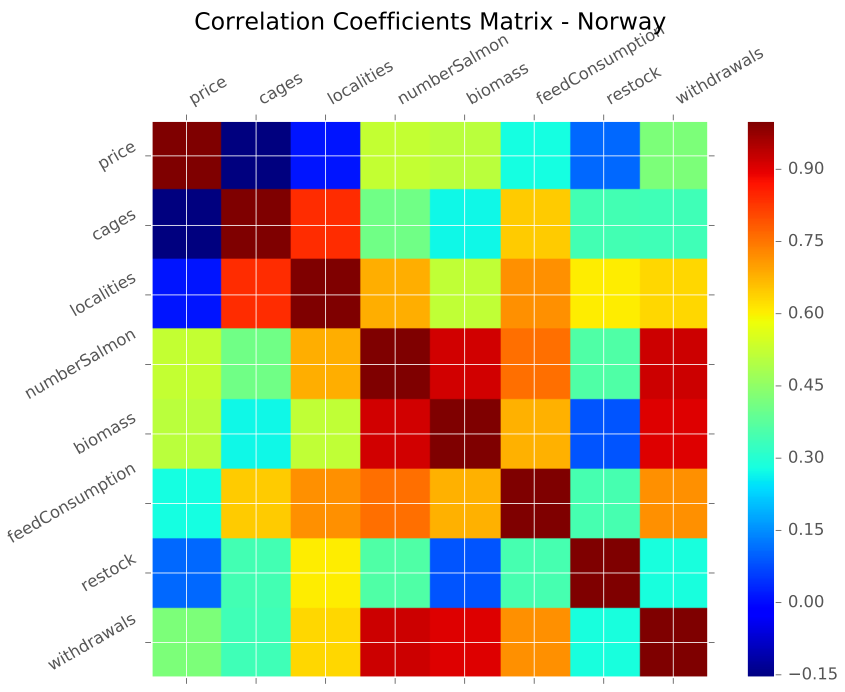


FIGURE 6.7: Correlation matrix between different inputs with data.

6.4.3 MIA section II: Normalized Angular Coefficients

Goal:

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

Requirements:

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

Implementation:

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

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

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

Results:

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

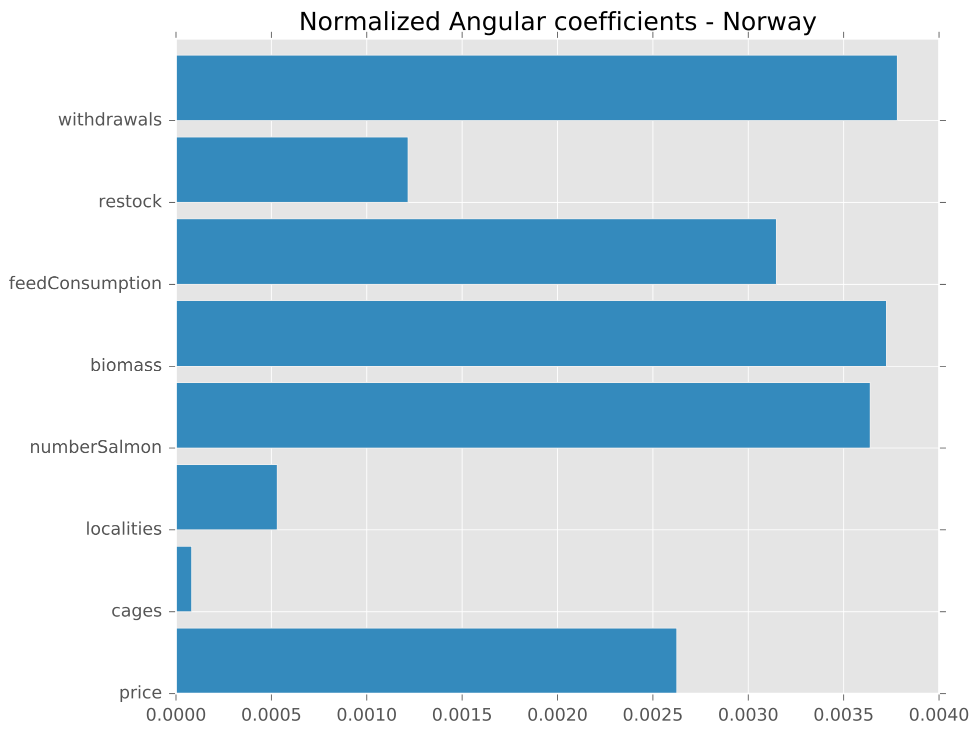


FIGURE 6.8: Normalized angular coefficients of each input’s trendline.

6.5 Data Displaying: Map graphic

Goal:

The main goal of this phase is to display the county data values in a easy way to understand and analyze.

In this particular case would be useful to display it on a real map of Norway, in order to be able to have an overview about all the counties about different parameters.

Requirements:

One parameter in input which has a single value for each county.

Implementation:

The library "shpreader" allows to read the extension file ".shp", which in this particular case contains the Norway's shape.

Then we can display the current shape using the "pyplot" library and set the colors based on the input values with the library "matplotlib".

```
shpreader.Reader().geometries()

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

plt.get_cmap
matplotlib.colors.Normalize
```

Results:

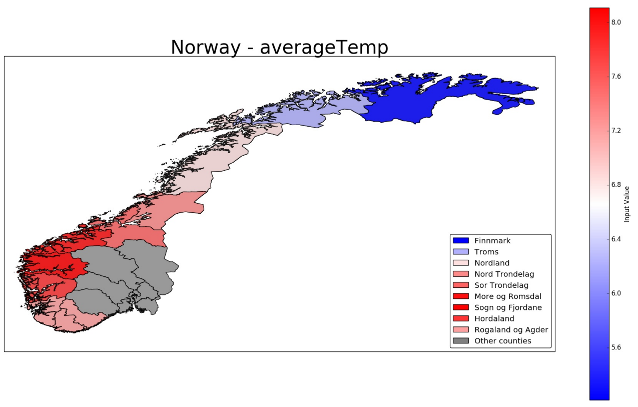


FIGURE 6.9: Average Sea Temperature from 2007 to 2014 in Norway.

6.6 Extract information from data

Chapter 7

Prediction of values

Some basic and general goals were defined before starting this phase, with the idea of "doing more if it's possible". Basically the main purpose was the one of, after the previous analysis, predict some values and evaluate the quality of the results. This prediction system was not defined with some specific requirements, so the first main problem was to find a reliable, accurated and user-friendly way to predict and display prediction of values.

Since the current dataset can be considered like a time series, in this phase we will develop the data prediction system using an ARIMA machine implemented in python.

The ARIMA machine can be configured with several configurations, it allows you to have more accurated results; so the first thing was to find the right configuration of the ARIMA machine of each single input which we are interested to forecast.

During this phase of the work have been implemented 3 different subsystems for different purposes:

1. Evaluating System
2. Training System
3. Future Prediction System

7.1 Requirements for reusability

The subsystems implemented during this phase of the work are almost completely reusable.
The reusability this systems allows to get some prediction of values about different kind of dataset, in particular:

- The "Evaluating System" is actually 100% reusable, and you can use it for evaluate any kind of dataset.
- The "Future Prediction System" is completely reusable as well, you should only modify the historic values and the real values inside the dataset for it works in a proper way.
- The "Training System" has been implemented for testing the current dataset, so it's not completely reusable but could be changed very easily and let it works also for other data input.

7.2 Evaluating System

Goal:

Used for evaluate different configurations of ARIMA machine.

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

Requirements:

There are not strict requirements needed. There are no type of restriction neither about the length or about the type of data.

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.

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

This system will provide 112 different ARIMA configurations results for each single input, and in particular it will display the best ARIMA configuration, that is the one with the lower MAPE.

Results:

The system will display the MAPE between real value and predicted values for each single tested ARIMA machine, in particular the configuration that gives the best result. All these results have been reported in a document and then also displayed with a 3D graphic that allows to see the MAPE value for each different order in input.

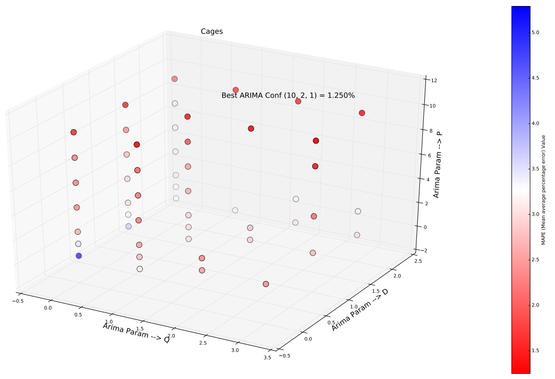


FIGURE 7.1: Graphic that displays different MAPE values for each ARIMA order.

7.3 Training System

Goal:

This system has the goal of training/testing a specific ARIMA configuration on a particular data input, and see how much accurate it is.

Requirements:

Since this Traning System has been used mainly for train and test the current dataset, it need to have like input a dataset that follows the same format:
- Data content: 144 values, 1 value for each month from 2005 to 2016

Code implementation:

First of all this system it's going to split the input data in two part:
- Train data: values which the ARIMA model is going to use for training.
- Test data: values which are hided by the forecasting model.
Once the ARIMA model has been created, the system will try to predict the future values, that are actually the "Test data".

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

Once the predictions have been calculated it's possible to display the "Test data" (that are the real values) and the predicted values, just to see how much the ARIMA configuration is accurate.

```
series = pd.read_csv("Dataset.csv", usecols=[sys.argv[1]])
series.plot(color="blue", linewidth=1.5,
        label="Series: "+sys.argv[1])


output = Series.from_csv('Output_Files/predictions.csv')
output.plot(color="red", linewidth=1.5,
       label="Prediction test:")
```

Results:

The following picture is an output example of this training system.

It actually allows to have an idea about how accurate is that ARIMA configuration for predictions.

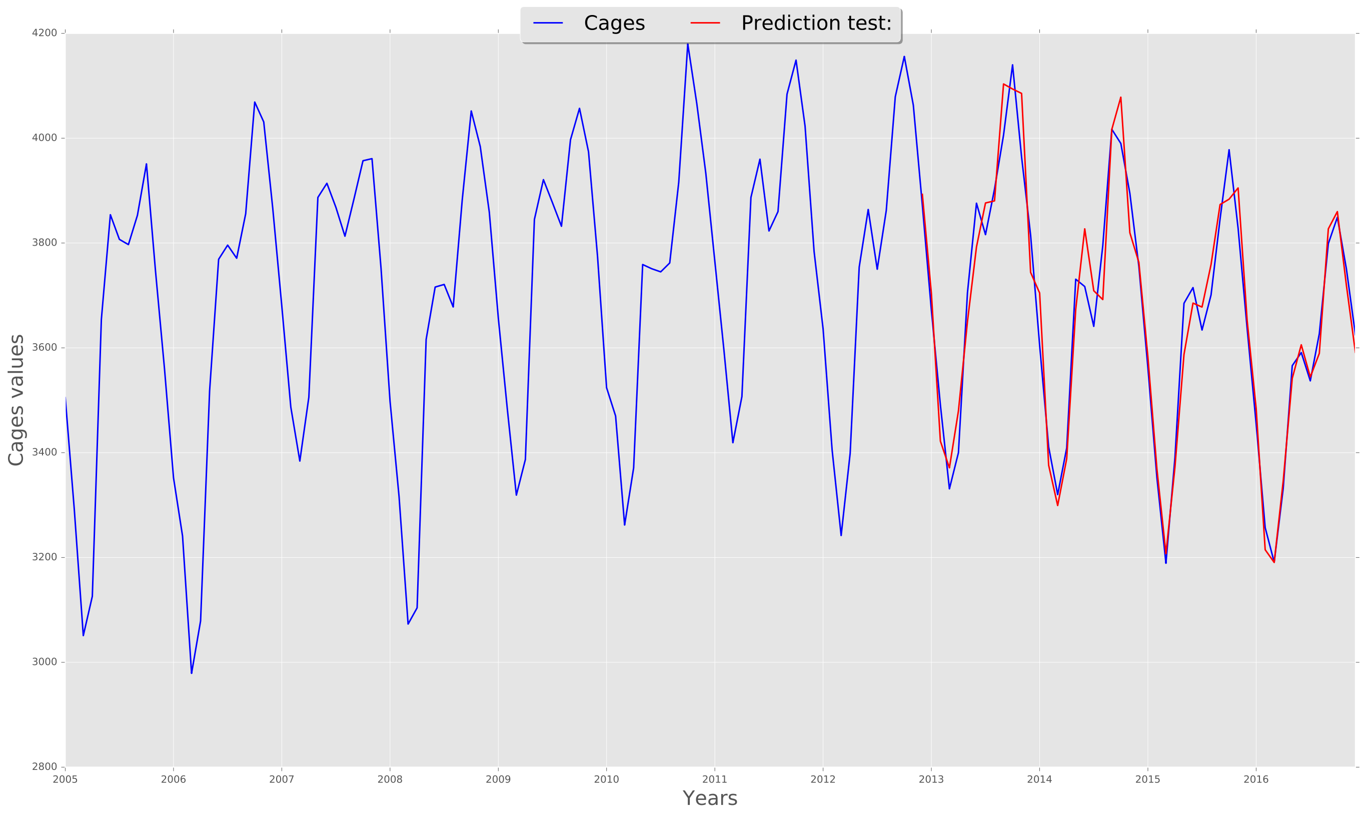


Figure 7.2: Graphic that displays the predicted values from a particular ARIMA machine configuration and the historic real values.

7.4 Future Prediction System

Goal:
This part of the work has the goal of display some real prediction of values in the future. It basically collects real future values, in this particular are values about 2017, of each single input and then, once calculated also the prediction values, it's going to display on the same graphic:
- Historic values
- Predicted future values
- Real values (if are available)

Requirements:
This system has to be as much reusable as possible, so there are not that strict requirements. You can reuse this Future Prediction System with any kind of dataset with no restrictions about length.
The only requirement to let it works in a proper way is that you have to set the historic and real values datasets in the right way; it means that you have to write down the historic values in the dataset in this way:

Index	Input1	Input2
1	Value1	Value1
2	Value2	Value2
3	Value3	Value3
...	...	...
120	Value120	Value120
121	Value121	Value121
122	Value122	Value122

Table 7.1: Historic dataset structure

And then, if you want to compare the predicted values with some real values that are already available, you have to set the real values dataset in this way:

Index	Input1	Input2
123	Value123	Value123
124	Value124	Value124
125	Value125	Value125
126		
127		
128		
129		

Table 7.2: Future real values dataset structure

Code implementation:

First of all a new ARIMA model it's created using the current historic values and a specific ARIMA-Order, that it should be the Best ARIMA-Order calculated during by the Evaluation System.

Then, once the model is ready, it's possible to calculate as many prediction values in the future as you want.

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

Once the predictions have been calculated, it's possible to display a graphic that contains the historic values, the future predictions value and the real future values (if already available).

```
# 1) Historic values
series = pd.read_csv("HISTORIC DATASET",
            usecols=[sys.argv[1]])
series.plot(color="blue",linewidth=1.5)

# 2) Predicted future values
series = Series.from_csv("PREDICTIONS DATASET")
series.plot(color="red", linewidth=1.5,
            label="Prediction Results")

# 3) Real future values
series = pd.read_csv("REAL VALUES DATASET")
pyplot.plot(series["Index"], series[sys.argv[1]],
    color="green", linewidth=1.5, label="Real values")
```

Results:

The system implemented during this phase allows to predict future for as many months as you want in the future and to display it, compared also with the historic values and real values once are available. The output graphic look like:

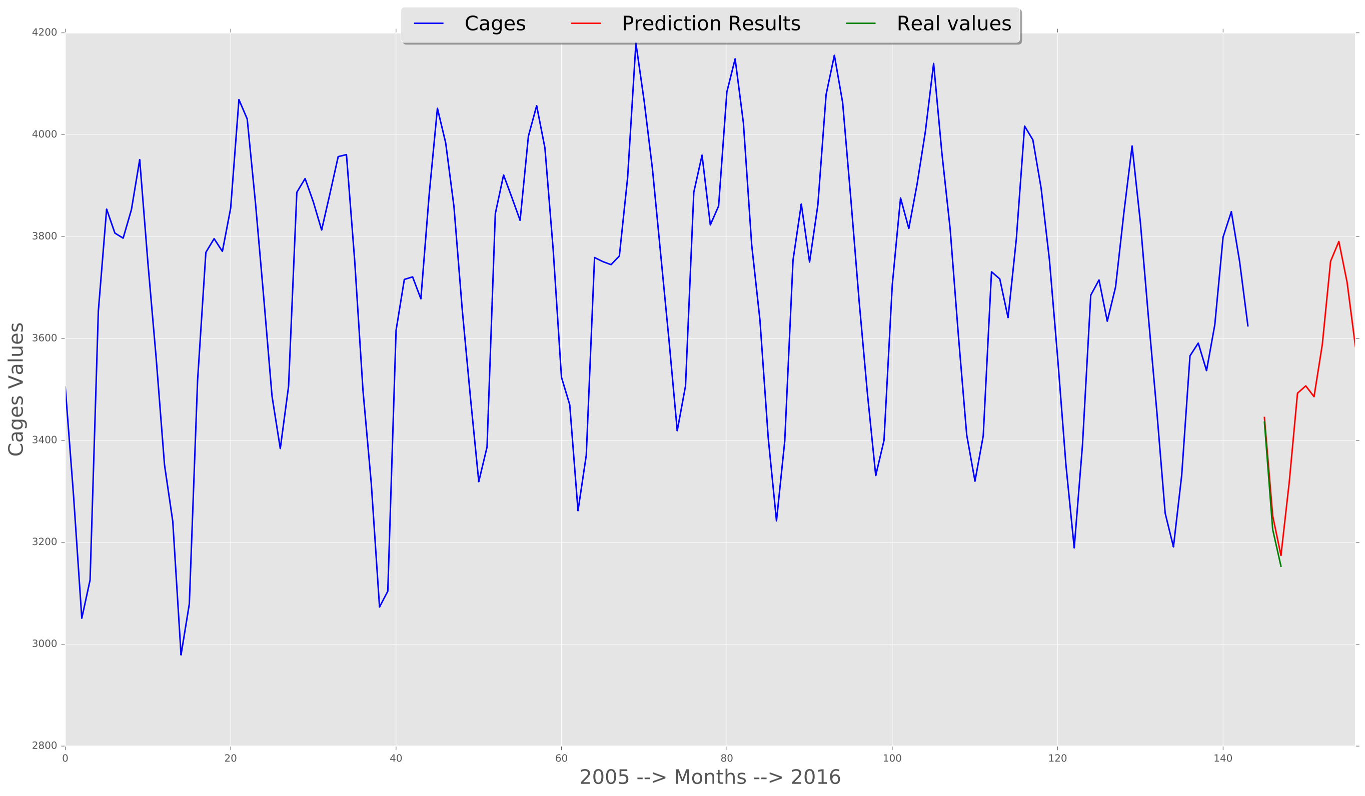


FIGURE 7.3: Graphic that display historic, future and predicted values of a input.

Chapter 8

Results

The main result of this work is the implemented "Python Analyzer System" itself. It actually provides an automatic way to make an initial analysis and display the results of any input dataset.

Further more has been provided an initial Python implementation of:

- Future Prediction System
- Country Map Displaying System

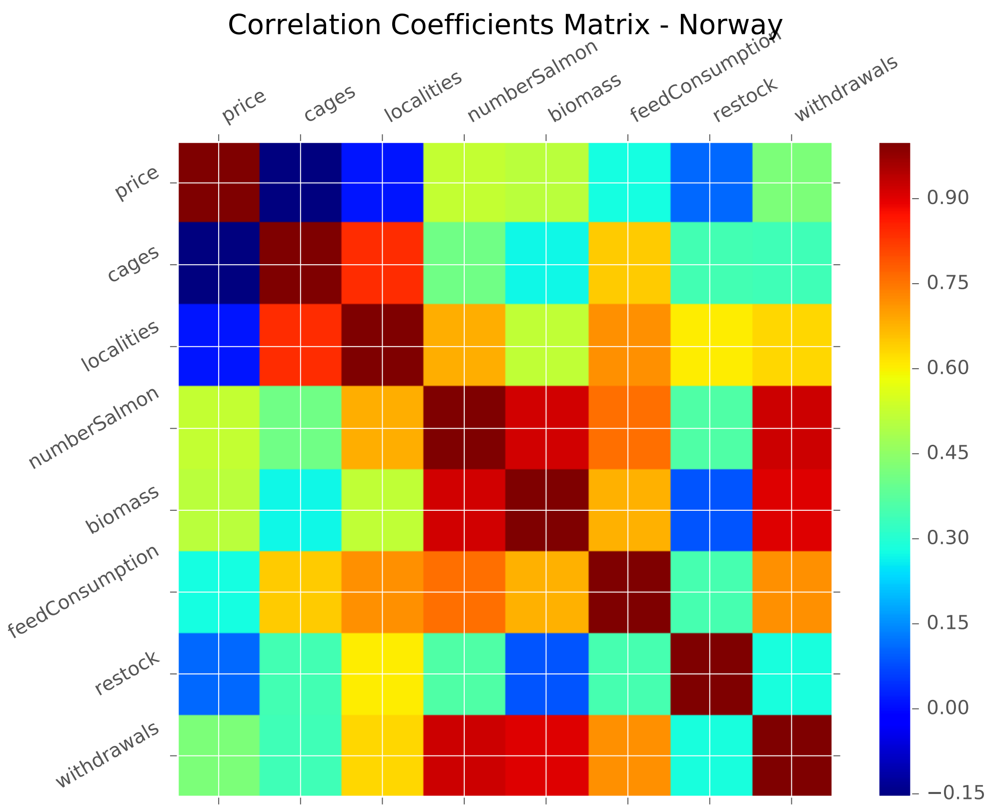


FIGURE 8.1: Correlation matrix between different inputs with data from 2005 to 2016.

INPUTS	Price	Cages	Localities	numberSalmon	biomass	feedConsumption	restock	withdrawals
Price	1	-0.16	0.02	0.52	0.51	0.28	0.11	0.43
Cages	-0.16	1	0.84	0.41	0.27	0.64	0.34	0.34
Localities	0.02	0.84	1	0.68	0.52	0.72	0.6	0.63
numberSalmon	0.52	0.41	0.68	1	0.92	0.76	0.36	0.92
biomass	0.51	0.27	0.52	0.92	1	0.68	0.09	0.91
feedConsumption	0.28	0.64	0.72	0.76	0.68	1	0.35	0.72
restock	0.11	0.34	0.6	0.36	0.09	0.35	1	0.28
withdrawals	0.43	0.34	0.63	0.92	0.91	0.72	0.28	1

TABLE 8.1: Dataset inputs correlation coefficients value.

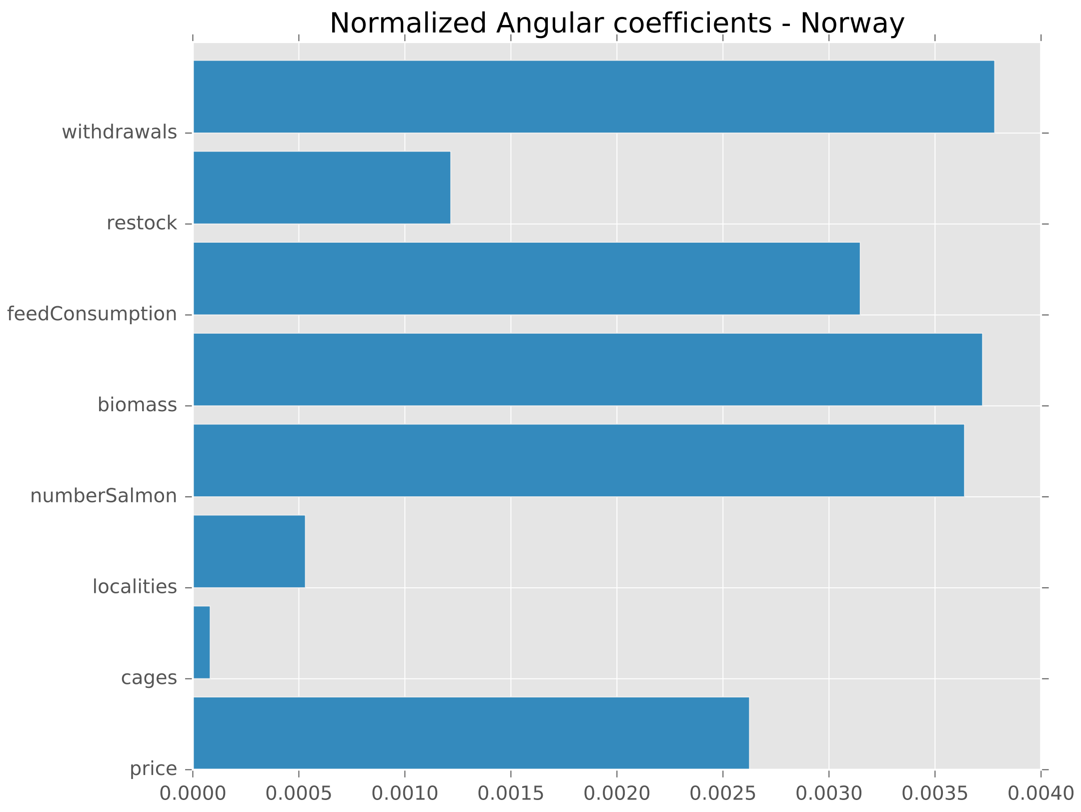


FIGURE 8.2: Normalized angular coefficients of each input's trendline.

Input	Equation	Coeff
Salmon_withdrawals	y=464.755139x+(46295.729945)	464.755139
Salmon_biomass_end_month	y=2832.712270x+(354138.727889)	2832.71227
Salmon_number_end_month	y=1543.298421x+(205325.455772)	1543.298421
Salmon_consumption_of_feed	y=620.070855x+(58330.012273)	620.070855
Salmon_price	y=0.178175x+(22.643654)	0.1781753878
Salmon_restock	y=89.230600x+(13390.363406)	89.2306
Localities	y=0.343533x+(539.979023)	0.343533
Cages	y=0.342834x+(3665.904023)	0.342834

TABLE 8.2: Dataset inputs trendline equation

Input	Normalized equation	Norm Ang Coeffs
Salmon_withdrawals	y=0.003782x+(0.376694)	0.003782
Salmon_biomass_end_month	y=0.003724x+(0.465599)	0.003724
Salmon_number_end_month	y=0.003639x+(0.484184)	0.003639
Salmon_consumption_of_feed	y=0.003147x+(0.296085)	0.003147
Salmon_price	y=0.002625x+(0.333633)	0.002625
Salmon_restock	y=0.001217x+(0.182583)	0.001217
Localities	y=0.000531x+(0.834589)	0.000531
Cages	y=0.000082x+(0.877011)	0.000082

TABLE 8.3: Dataset inputs normalized trendline equation

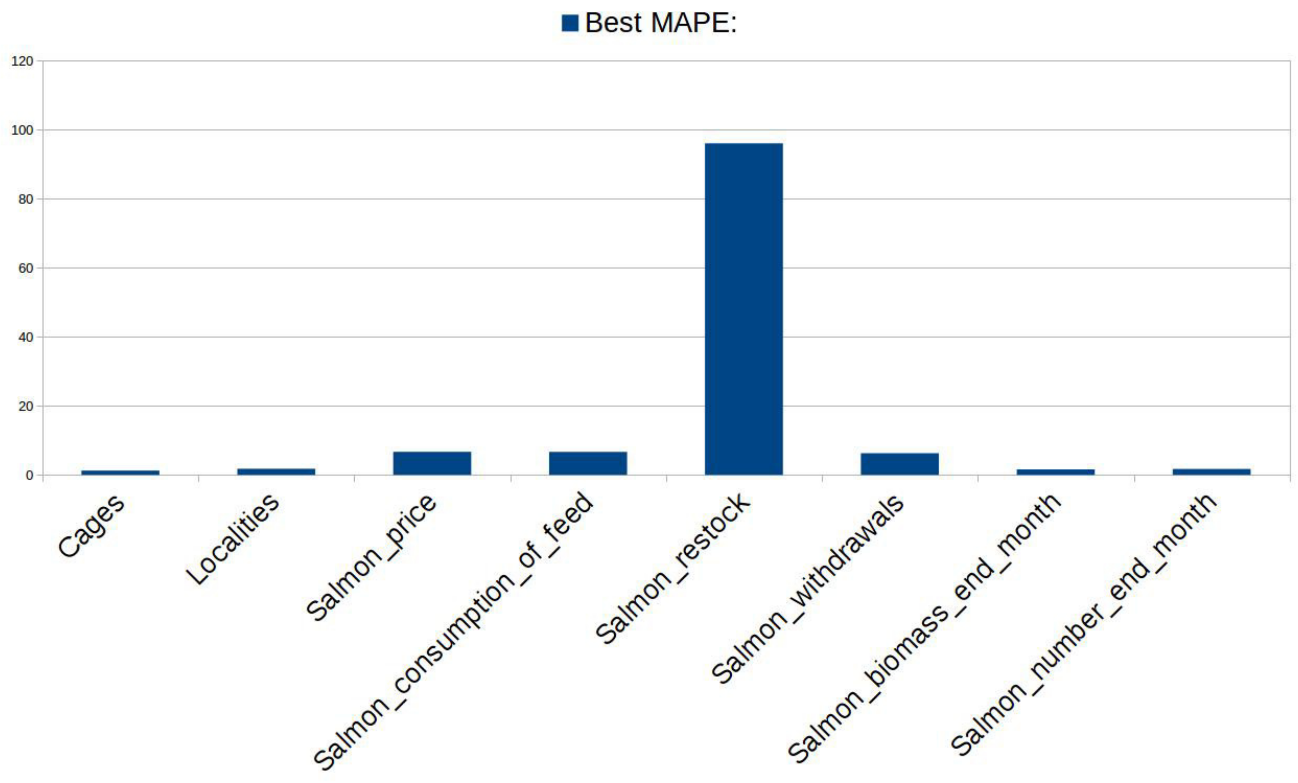


FIGURE 8.3: Lower MAPE with best ARIMA Configuration for each tested input.

Input	ARIMA Conf	MAPE
Cages	(10,2,1)	1.251%
Localities	(10,0,1)	1.779%
Salmon_price	(0,1,1)	6.686%
Salmon_consumption_of_feed	(6,1,0)	6.659%
Salmon_restock	(10,0,1)	96.006%
Salmon_withdrawals	(10,0,1)	6.277%
Salmon_biomass_end_month	(8,1,0)	1.601%
Salmon_number_end_month	(10,2,0)	1.723%

TABLE 8.4: Dataset inputs normalized trendline equation

Months	Cages			Localities			Salmon Biomass			Salmon Number		
	Real	Pred	Error	Real	Pred	Error	Real	Pred	Error	Real	Pred	Error
January 2017	3436	3444.87	0.26%	539.000	543.41	0.82%	738902	732841.36	0.82%	369274	366826.189	0.66%
February 2017	3225	3251.915	0.83%	523.000	529.05	1.16%	712981	709931.42	0.43%	347824	352905.30	1.46%
March 2017	3153	3164.190	0.67%	529.000	534.29	1.00%	667749	679405.11	1.75%	343636	349747.77	1.78%
April 2017		3317.814			549.14			657418.41			369616.83	
May 2017		3492.701			550.64			646850.14			387244.43	
June 2017		3507.062			545.66			653574.18			387630.48	
July 2017		3485.804			560.58			678469.77			384314.00	
August 2017		3588.373			584.55			707646.99			394062.43	
September 2017		3751.633			596.32			734628.28			411164.01	
October 2017		3790.521			589.11			757679.66			411094.09	
November 2017		3710.033			576.75			770838.51			396102.24	
December 2017		3584.505			563.56			771279.10			380399.93	

Months	Consumption of feed			Salmon restock			Salmon Withdrawals			Salmon price		
	Real	Pred	Error	Real	Pred	Error	Real	Pred	Error	Real	Pred	Error
January 2017	109341	98174.28	10.21%	4415	4734.43	7.23%	87609	90488.98	3.29%			
February 2017	88704	77998.20	12.07%	991	6904.51	596.72%	3.29%	101295.55	11.00%			
March 2017	87033	74726.18	14.14%	13594	15427.63	13.49%	109498	101724.09	7.10%			
April 2017		88768.11			39781.36			95032.95				
May 2017		113280.67			39438.14			92065.64				
June 2017		140413.56			22562.89			88775.54				
July 2017		164511.15			20449.85			92643.44				
August 2017		179690.48			39487.67			104102.60				
September 2017		181556.12			49991.91			110419.37				
October 2017		169502.10			31449.69			107588.69				
November 2017		147770.73			11698.23			102943.86				
December 2017		123447.55			7957.45			99561.98				

Chapter 9

Discussion

Chapter 10

Conclusion

10.1 Summary

10.2 Recommendations to future work

10.2.1 Dataset about single locality

This thesis allows to have a general overview and predictions of values about the aquaculture business in Norway.

But it would be much more useful, in particular for people into the aquaculture business, to use this system to have an overview and predictions of data provided by a single locality of aquaculture.
In this case the system could be used from the owner of the locality to analyze historic values and use the prediction system to have a forecast about some particular parameters.

10.2.2 Visualization of the data

10.2.3 Test prediction system with a bigger dataset

10.3 Prediction system as a service

This system has been developed with the idea that it could become a "Service system", that is basically a configuration of technology and organizational networks designed to deliver services that satisfy the needs or wants of customers. Since the prediction system implemented during this work is almost 100% reusable, it could be used from people for prediction about any kind of data.

Basically the idea is to create a web application that allows to let you upload your own dataset, choose your own preferences and prediction settings, and then the system will calculate and display prediction of the current values in the future together with the MAPE (Mean Average Percentage Error) to have an idea bout how accurate are the.

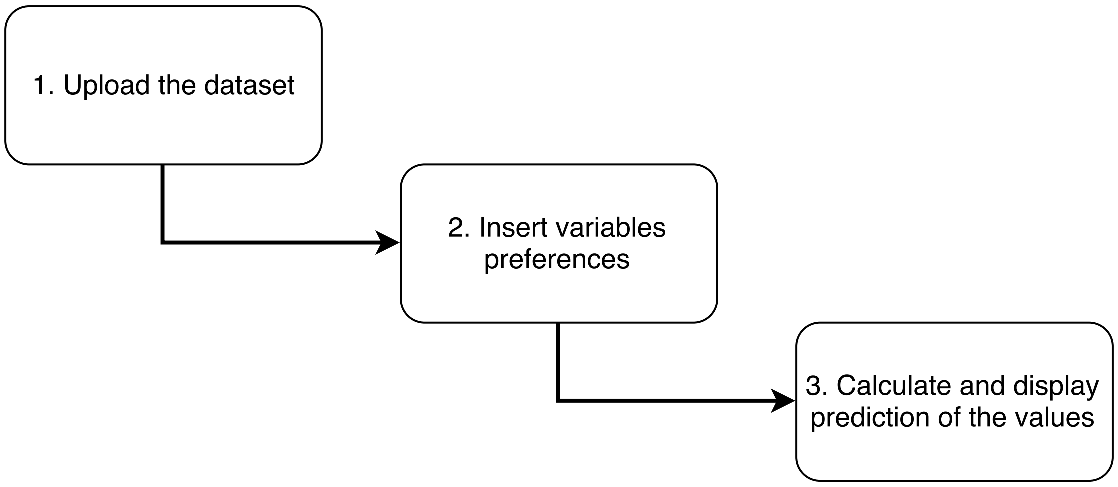

1. Upload the dataset
2. Insert variables preferences
3. Calculate and display prediction of the values

FIGURE 10.1: Idea of the Servie System for predictions.

Chapter 11

Bibliography

[1] http://www.fiskeridir.no/Akvakultur/Statistikk-akvakultur/Biomassestatistikk

[2] https://www.quandl.com/data/ODA/PSALM_USD-Fish-Salmon-Price

[3] http://machinelearningmastery.com/time-series-forecasting/

[4] https://github.com/Sprea22/Thesis_Latex_Doc

[5] https://github.com/Sprea22/Data_Analyzer_Python

[6] https://github.com/Sprea22/Forecasting_System_Python

[7] https://en.wikipedia.org/wiki/Data_science

[8] http://machinelearningmastery.com/time-series-forecasting/

[9] http://www.ulb.ac.be/di/map/gbonte/ftp/time_ser.pdf

[10] http://users.dma.unipi.it/~flandoli/AUTCap4.pdf

[11] http://fishpool.eu/wp-content/uploads/2016/04/final-dag.pdf

[12] mysalmon.no

[13] http://munin.uit.no/bitstream/handle/10037/5913/thesis.pdf?sequence=1

[14] http://www.indexmundi.com/commodities/?commodity=fish&months=180&currency=nok

[15] http://www.fao.org/3/a-i5555e.pdf%20

Appendix A

SIA Implementation code

A.1 SIA: Imported libraries

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

```
import os
```

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

```
import sys
```

The ”pylab” library will be useful for plot data.

```
import pylab
```

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

```
import pandas as pd
```

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

```
import numpy as np
```

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

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

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

```
from PIL import Image
```

A.2 SIA: Implemented methods

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

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

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

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

```python
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
```

```python
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)
```

```python
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)
```

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.

```python
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.

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

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

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

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

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

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

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

Code implementation:

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

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

Some initialization of variables that are going to be useful.

```
fig2 = pyplot.figure()
ax = fig2.add_subplot(111)
```

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

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

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

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

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

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

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

A.5 SIA section III: Correlation matrix between years

Code implementation:

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

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

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

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

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

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

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

Creating the correlation matrix using the already calculated correlation coefficents.

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

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

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

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

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

A.6 SIA section IV: Correlation matrix between months

Code implementation:

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

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

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

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

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

Creating the correlation matrix using the already calculated correlation coefficents.

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

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

```

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

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

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

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

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

A.7 SIA section V: Single overview

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

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

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

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

Appendix B

MIA Implementation code

B.1 MIA: Imported libraries

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

```
import pandas as pd
```

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

```
import numpy as np
```

```
import sys
```

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

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

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

B.2 MIA section I: Total Correlation Coefficients

```
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)
```

```
# Calculatic che correlation coefficent between each year of the input
    dataset
corrRes = np.corrcoef(corr)

mat = np.matrix(corrRes)
dataframe = pd.DataFrame(data=mat.astype(float))
dataframe.to_csv("Results/"+sys.argv[1]+"/Total_Evidences/"+sys.argv[1]+"
    _CorrCoeff.csv", sep=',', header=False, float_format='%.2f', index=
    False)

fig = pyplot.figure()
ax = fig.add_subplot(111)
# Displaying the matrix with the results about correlation coefficents
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')
#cax.set_clim(vmin=0.5, vmax=1)
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

```
fig2 = pyplot.figure()
ax2 = fig2.add_subplot(111)
temp = []
for i in series.columns:
    index = sys.argv[1]+"-"+i
    tempSeries = pd.read_csv("Results/"+sys.argv[1]+"/"+i+"/"+sys.argv[1]+"_
    "+i+"_AngCoeff.csv", header=0)
    temp.append(tempSeries[index].values[1])
x = range(len(series.columns))

pyplot.barh(x, temp)
# Displaying and saving the bar graphic
```

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

Appendix C

Prediction System Implementation code

C.1 Evaluating System

```python
import warnings
import sys
import numpy as np
import pandas as pd
from pandas import Series
from statsmodels.tsa.arima_model import ARIMA
from sklearn.metrics import mean_squared_error
```

```python
def mean_absolute_percentage_error(y_true, y_pred):
   rng = len(y_true)
   diff = []
   for i in range(0,rng):
      diff.append(y_true[i] - y_pred[i])
      diff[i] = diff[i] / y_true[i]
   abs = np.abs(diff)
   mn = np.mean(abs)
   percentageError = mn * 100
   return percentageError
```

```python
def evaluate_arima_model(X, arima_order):
   # prepare training dataset
   train_size = int(len(X) * 0.66)
   train, test = X[0:train_size], X[train_size:]
   history = [x for x in train]
   # make predictions
   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])
    # calculate out of sample error
    error = mean_absolute_percentage_error(test, predictions)
    return error
```

```
dataset = dataset.astype('float32')
best_score, best_cfg = float("inf"), None
for p in p_values:
for d in d_values:
for q in q_values:
order = (p,d,q)
try:
mape = evaluate_arima_model(dataset, order)
if mape < best_score:
best_score, best_cfg = mape, order
print('ARIMA%s MAPE=%.3f%%' % (order,mape))
except:
print('ARIMA%s MAPE=Nil' % str(order))
continue
print('Best ARIMA%s MAPE=%.3f%%' % (best_cfg, best_score))
```

```
series = pd.read_csv("Dataset.csv", header=0, usecols=[sys.argv[1]])
# evaluate parameters
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)
```

C.2 Training System

C.3 Future Prediction System