

MINISTRY OF EDUCATION AND SCIENTIFIC RESEARCH



TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA

**FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT**

LICENSE THESIS TITLE

LICENSE THESIS

**Graduate: Firstname LASTNAME
Supervisor: scientific title Firstname LASTNAME**

2015



**FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT**

DEAN,
Prof. dr. eng. Liviu MICLEA

HEAD OF DEPARTMENT,
Prof. dr. eng. Rodica POTOLEA

Graduate: **Firstname LASTNAME**

LICENSE THESIS TITLE

1. **Project proposal:** *Short description of the license thesis and initial data*
2. **Project contents:** *(enumerate the main component parts) Presentation page, advisor's evaluation, title of chapter 1, title of chapter 2, ..., title of chapter n, bibliography, appendices.*
3. **Place of documentation:** *Example:* Technical University of Cluj-Napoca, Computer Science Department
4. **Consultants:**
5. **Date of issue of the proposal:** November 1, 2014
6. **Date of delivery:** June 18, 2015 *(the date when the document is submitted)*

Graduate: _____

Supervisor: _____



**FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT**

**Declarație pe proprie răspundere privind
autenticitatea lucrării de licență**

Subsemnatul(a)

_____, legiti-
mat(ă) cu _____ seria _____ nr. _____
CNP _____, autorul lucrării _____

elaborată în vederea susținerii examenului de finalizare a studiilor de licență la Facul-
tatea de Automatică și Calculatoare, Specializarea _____
din cadrul Universității Tehnice din Cluj-Napoca, sesiunea _____ a an-
ului universitar _____, declar pe proprie răspundere, că această lucrare este
rezultatul propriei activități intelectuale, pe baza cercetărilor mele și pe baza informațiilor
obținute din surse care au fost citate, în textul lucrării și în bibliografie.

Declar, că această lucrare nu conține porțiuni plagiate, iar sursele bibliografice au
fost folosite cu respectarea legislației române și a convențiilor internaționale privind drep-
turile de autor.

Declar, de asemenea, că această lucrare nu a mai fost prezentată în fața unei alte
comisii de examen de licență.

În cazul constatării ulterioare a unor declarații false, voi suporta sancțiunile admin-
istrative, respectiv, *anularea examenului de licență*.

Data

Nume, Prenume

Semnătura

De citit înainte (această pagină se va elimina din versiunea finală):

1. Cele trei pagini anterioare (foaie de capăt, foaie sumar, declarație) se vor lista pe foi separate (nu față-verso), fiind incluse în lucrarea listată. Foaia de sumar (a doua) necesită semnătura absolventului, respectiv a coordonatorului. Pe declarație se trece data când se predă lucrarea la secretarii de comisie.
2. Pe foaia de capăt, se va trece corect titulatura cadrului didactic îndrumător, în engleză (consultați pagina de unde ați descărcat acest document pentru lista cadrelor didactice cu titulaturile lor).
3. Documentul curent **nu** a fost creat în MS Office. E posibil să fie mici diferențe de formatare.
4. Cuprinsul începe pe pagina nouă, impară (dacă se face listare față-verso), prima pagină din capitolul *Introducere* tot așa, fiind numerotată cu 1.
5. E recomandat să vizualizați acest document și în timpul editării lucrării.
6. Fiecare capitol începe pe pagină nouă.
7. Folosiți stilurile predefinite (Headings, Figure, Table, Normal, etc.)
8. Marginile la pagini nu se modifică.
9. Respectați restul instrucțiunilor din fiecare capitol.

Contents

Chapter 1	Introduction - Project Context	12
1.1	Project context	12
1.1.1	Social Media and why people care about what it says	13
1.1.2	Data Analysis tools today	13
1.1.3	Crossroads: Social Media in Business Intelligence	13
Chapter 2	Project Objectives and Specifications	15
2.1	Web architecture	15
2.1.1	Data storage	16
2.2	ATHENA breakdown	16
2.2.1	Twitter as data source	16
2.2.2	Harvesting	17
2.2.3	Enhancement	18
2.2.4	Normalisation and analysis	18
2.3	Non-functional requirements	18
Chapter 3	Bibliographic research	20
3.1	Title	20
3.2	Other title	20
Chapter 4	Analysis and Theoretical Foundation	21
4.1	Title	21
4.2	Other title	21
Chapter 5	Detailed Design and Implementation	22
5.1	Python and Django	22
5.2	Storage with Cassandra	23
5.3	Harvesting tools	25
5.3.1	Celery and Redis brokers for asynchronous jobs	25
5.3.2	Twitter API and the Tweepy library	25
5.4	Enhancement tools	25
5.4.1	Simple numerical data	25

5.4.2	Sci-kit learn	25
5.4.3	Related hashtag calculation	25
5.4.4	Related hashtag clustering using K-Means	25
5.4.5	Page and bot vs. users modeling	25
5.5	Normalisation and Analysis tools	25
Chapter 6	Testing and Validation	26
6.1	Title	26
6.2	Other title	26
Chapter 7	User's manual	27
7.1	Title	27
7.2	Other title	27
Chapter 8	Conclusions	28
8.1	Title	28
8.2	Other title	28
	Bibliography	29
	Appendix A Relevant code	30
	Appendix B Other relevant information (demonstrations, etc.)	31
	Appendix C Published papers	32

Chapter 1

Introduction - Project Context

Not long time ago, big data and the need for content analysis were idealistic, intangible concepts. However, recent developments in computer performance and the growing popularity of social media has enhanced the need for novel academic approaches in this field.

The biggest challenge with processing these huge amounts of data lies in its sheer volume. This is highly problematic in many different ways:

- storing and managing data is costly and sometimes needs special structuring attention
- irregularities across data streams often indicate the need for extra transformations
- filtering out noise and the extraction of meaningful data from thousands of entries requires specialised algorithms, architectures and platforms which are either inaccessible or difficult to use by regular users

1.1 Project context

Social media platforms these days have a strong presence in "the Cloud", which allows more and more developers to solve the problem of storage space and scalability in a relatively easy way. The high adoption of platforms like Facebook, Twitter, Instagram and many others have resulted in the generation of hundreds of thousands of online documents, text and media alike. Such a deluge of information is highly applicable for business purposes, but in is often stuck as a "diamond in the rough". An array of new and exciting job titles has also appeared, such as "Social Media Marketer", "Social Media Manager", "Social Media Expert" and so on. Not only do they represent career alternatives for a new generation, but they are also still outliers for decision support software.

1.1.1 Social Media and why people care about what it says

While the concept of Social Media is not a new one to be precise, it is clearly just starting to engulf us. The spread of Social Media alongside the spread of worldwide Internet access, the emergence of specialised applications and the recent shift to mobile online presence are objective realities. Subjectively as well, people have become more engaged to consuming, creating and sharing content online, which is evident in newspaper sales drops, app sales increase, events "going viral" within groups and even the online presence of a political critical mass. Companies offer barter possibilities for a variety of bloggers and vloggers to push products to their followers (which is not only more effective¹, but also cheaper).

No wonder social media has a stake in public decisions! But for the first time in decades, people's choices are also quite open and public. Which means that some parties can get clues about society's preferences by using public data available on platforms such as Facebook, Youtube and Twitter. Consider two basic use cases, which I will emphasise throughout the course of this thesis:

- What does social media say about X?
- What does social media say about X vs. Y?

where X and Y can be public people, events, brands, companies etc.

1.1.2 Data Analysis tools today

There are a number of current platforms for data analysis, in the large sense including chart generators, format converters and importers and data visualisation plug-ins of sorts. Crossing into the realm of powerful analysis applications, most of them are complex and confusing to non-engineers, examples including Tableau², RapidMiner³ and WolframAlpha⁴. Figure 1.1 shows Tableau's interface, which may seem quite intimidating.

1.1.3 Crossroads: Social Media in Business Intelligence

The Computer Science field is very much branched nowadays in clearly-delimited subjects. But it is such a case where the large-scale processing capacities of data analytics should merge with approaches for computer-assisted business decisions. Consider that data analytics software often requires extensive training and a specialised knowledge, while most users desire only simple, straightforward functionalities for basic data analysis. One

¹<http://www.techrepublic.com/article/election-tech-why-social-media-is-more-powerful-than-advertising/>

²<https://public.tableau.com/s/>

³<http://rapidminer.com>

⁴<http://www.wolframalpha.com>

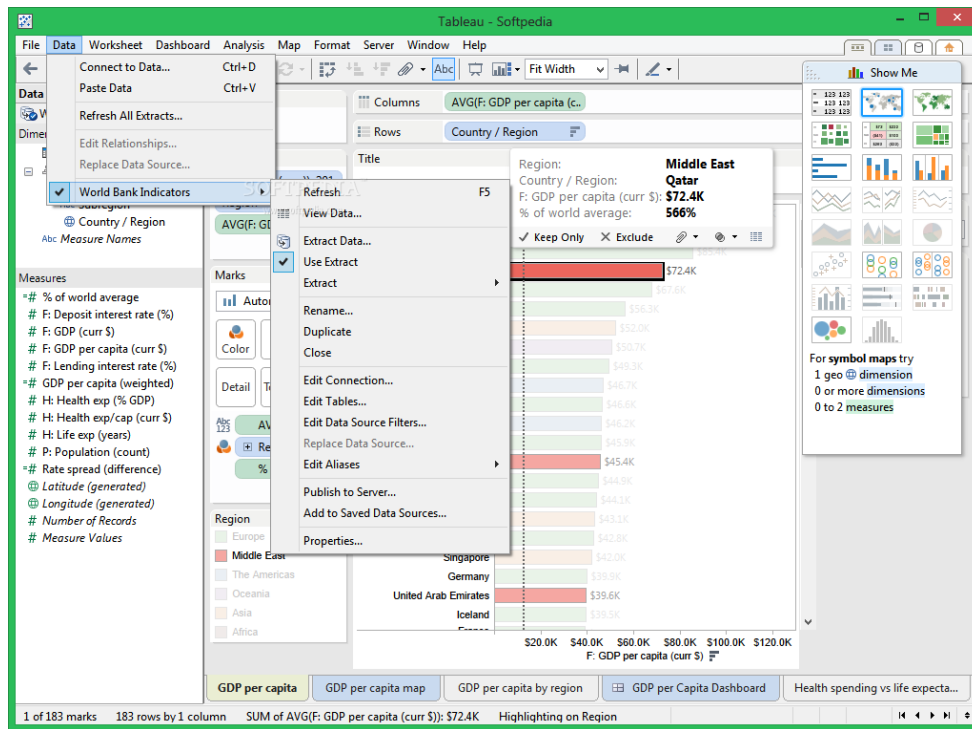


Figure 1.1: Tableau's interface

of Object Oriented Programming's fundamental principles is that of abstraction, in which end users need not (and should not) be aware of what is under the hood. Yet in data analytics, this is still the case on a

The following Master's Thesis is the result of such an endeavour, to combine solid concepts from big data analysis, web application architecture and design, as well as enterprise software and economics for a better understanding of social media trends and opinions.

Thesis structure

TODO

Chapter 2

Project Objectives and Specifications

The project's purpose is to help non-expert users interested in social media analysis to get relevant and reliable information regarding specific concepts.

2.1 Web architecture

A web architecture is preferable as a modern-day alternative to installable applications. The fast evolution of mobile devices has pushed web developers to segregate User Interfaces from Backend implementations, permitting devices to seamlessly connect to the same under the hood implementation without much development cost and overhead. Communication via REST services is crucial in this concern. Beyond user preference towards such applications, a web architecture presents even more advantages:

- virtually infinite space extensible with storage backends such as AWS3 or Google Cloud
- scalability and outsourcing of time-costly services such as getting stream histories
- database distribution over nodes
- easily available documentation and integration throughout the development process

Using a web framework eases development even more so and presents security advantages such as stable features, packages and quick patches in the odd case of a security breach. Plugging in services is also considered in the project's context, with handling of specific use cases done in separate managers, loosely coupled with the Model-View-Controller architecture.

2.1.1 Data storage

Since scalability and distribution are a must for a large-scale stream analysis project, data storage should be handled using a method which supports such breakdown while still remaining query-efficient. NoSQL databases are a popular choice for such implementation. The correspondence between the database storage backend and the model part of the application should also be loosely coupled, allowing for the possibility of switching between database backends if the need arises.

Big players in the field of non-relational databases are obvious possible choices: HBase, Cassandra, MongoDB, CouchDB, etc.

2.2 ATHENA breakdown

This thesis is based on the ATHENA original project, which comprises different pipeline steps in acquiring and analysing Twitter feeds:

- Harvesting: acquisition of collections of related Twitter statuses, by hashtag and dates (start and end).
- Enhancement: basic analysis of a single harvest with classical unsupervised algorithms
- Normalisation and Analysis: comparative analysis of harvests

2.2.1 Twitter as data source

For the development of this project, I chose to only implement one social media backend as data source. Twitter was the choice, since the character limit they impose is a great asset in regards of data processing: First of all, the classical data analysis pipeline deals with special cases such as documents of different lengths providing unusual skewing to the data set. Another advantage of Twitter is the prevalence of the hashtag model, which has failed to catch on with Facebook to the same degree. This means that statuses basically come out of the box already tagged for content.

Twitter is arguably the second most popular social media platform at the time, gaining on a large number of users even before the mobile device explosion, by integrating with telephone service providers to facilitate posting through SMS. Its 140 character limit makes for a good spin, which has pushed Twitter into a more textual realm than its more successful counterpart, Facebook. The platform is largely popular in the United States and especially in the entertainment domain¹, with the most popular Twitter accounts belonging to celebrities.

¹<https://en.wikipedia.org/wiki/Twitter>

It notably also has a history of being the "go to" source for data analysis, after the United States' current president Barack Obama purportedly used it in his 2008 campaign as one of the principal media outlets, choosing to have a strong Twitter presence, akin to advertisement presence of past candidates.

In short, although this approach is social platform-independent, Twitter was chosen even from the specifications step of this project for a clear start towards the data mining part.

Fetching data through asynchronous jobs

Since most social media platforms communicate with external applications via REST APIs with rate limits, the project's structure should include asynchronous modules. In this way, the user will be able to send an API-intensive job for queueing and asynchronous processing, without hindering their overall experience of using the application. This becomes apparent in the Harvesting part.

2.2.2 Harvesting

If we interpret ATHENA as a classical pipeline, then it becomes clear that the Harvesting part is the acquisition part. In order to perform analysis on data, it must first be fetched according to predefined rules and user options. Being time-intensive, these jobs must be designed asynchronously, following a FIFO model.

Figure 2.1 represents the conceptual model of data acquisition, starting from a collection of grouped documents (in our case they will be collections of statuses with a common hashtag). The Harvest manager handles sending asynchronous jobs via a FIFO queue towards a Consumer-type Harvesting job, represented as black box and further detailed in Figure 2.2. It is important to note that this pipeline is not in any way adjusted to the domain or the particular API platform; the only constraint we impose is that of periodically checking for compliance to rate limits. Social APIs usually limit requests either by number of requests or requested data size per unit of time. It is therefore a good idea to consider if any limitations alter our pipelines before starting on implementations.

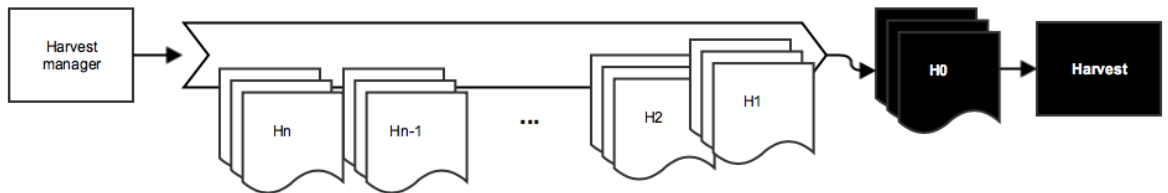


Figure 2.1: Harvesting pipeline

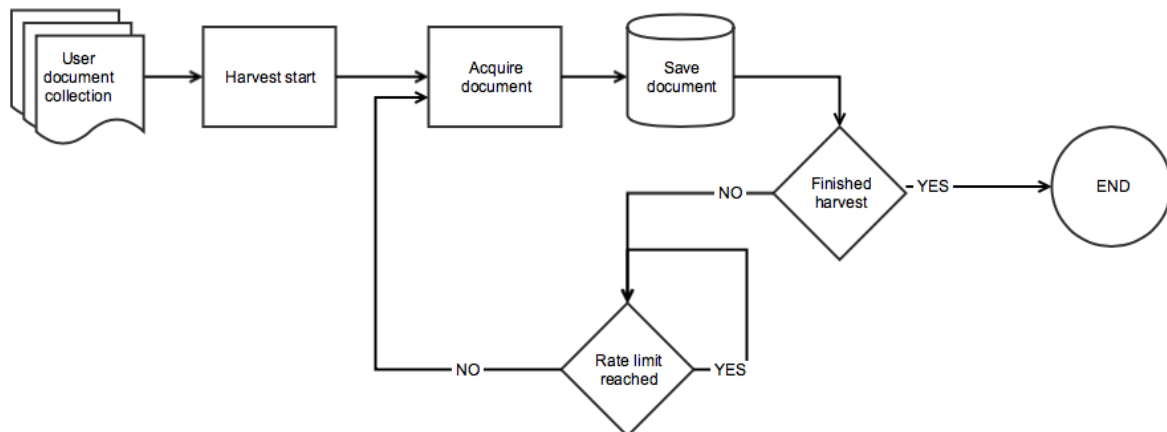


Figure 2.2: Harvesting job sub-pipeline

2.2.3 Enhancement

As previously stated, it is not of great importance to have the data, but the goal is to obtain meaningful information from it. In this thesis I will refer to "harvests" as per the following definition:

Definition A Harvest is a collection of documents related by content, containing a token T and spanning from a start date and to an end date, which have gone through the harvesting pipeline and are stored for further analysis.

After completing the Harvesting pipeline, resulting harvests are processed using various data extraction algorithms and presented to the user in the Enhancement step. These algorithms each have their own sub-pipelines of transformation, which will be detailed in the Analysis step.

User contribution modeling

TODO

2.2.4 Normalisation and analysis

TODO

2.3 Non-functional requirements

I am much concerned with the user's perspective in developing this application. As explained previously, one of the main objectives of this application is undoubtedly to be as simple to use as possible, to benefit non-expert users.

Firstly, I believe that a familiar and streamlined user experience should be implemented. The UI should have components found in most web application nowadays, in order to seem approachable. The customisation options should also be loaded without input from the user, so that things like algorithm choices and parameters should stay hidden. Non-expert users do not know what *Tfidf Vectorizer* or *KMeans clustering* means, all they care about is to have some paplable results to their queries. In short, it is in the benefit of the non-expert users, which I target through this application, to have as little inputs, panels and buttons to figure out.

Besided the look and feel, users will also be interested in having a fast and reliable application. Exceptions and possible bugs should be properly filtered out and time-consuming jobs moved as per Figure 2.2 to asynchronous jobs.

Chapter 3

Bibliographic research

Bibliographic research has as an objective the establishment of the references for the project, within the project domain/thematic. While writing this chapter (in general the whole document), the author will consider the knowledge accumulated from several dedicated disciplines in the second semester, 4th year (Project Elaboration Methodology, etc.), and other disciplines that are relevant to the project theme.

Represents about 15% of the paper.

Each reference must be cited within the document text, see example below (depending on the project theme, the presentation of a method/application can vary).

This section includes citations for conferences or workshop [?], journals [?], and books [?].

In paper [?] the authors present a detection system for moving obstacles based on stereovision and ego motion estimation. The method is ... *discus the algorithms, data structures, functionality, specific aspects related to the project theme, etc....* Discussion: *pros and cons.*

In chapter 4 of [?], the *similar-to-my-project-theme algorithm* is presented, with the following features ...

3.1 Title

3.2 Other title

Chapter 4

Analysis and Theoretical Foundation

Together with the next chapter takes about 60% of the whole paper

The purpose of this chapter is to explain the operating principles of the implemented application. Here you write about your solution from a theory standpoint - i.e. you explain it and you demonstrate its theoretical properties/value, e.g.:

- used or proposed algorithms
- used protocols
- abstract models
- logic explanations/arguments concerning the chosen solution
- logic and functional structure of the application, etc.

YOU DO NOT write about implementation.

YOU DO NOT copy/paste info on technologies from various sources and others alike, which do not pertain to your project.

4.1 Title

4.2 Other title

Chapter 5

Detailed Design and Implementation

The following chapter presents the architecture, components and implementation particularities of the ATHENA application.

5.1 Python and Django

The choice of Python as main programming language for this application, alongside the choice of Django as a framework, were based on the industry's long-time endorsement of these technologies. According to the TIOBE programming language index, Python ranks as the 6th most popular programming language, the index being calculated according to the number of engineers world-wide, courses and third-party vendors concerned with this programming language [1]. Its higher ranked competitors are Java and the C family (C, C++ and C#), which are not as well diversified in the field of research and string processing as Python.

Python for scientists

Besides its prevalence in the web application realm, Python is also one of the preferred languages for research, as argued in [2], with a plethora of scientific-oriented third-part libraries including NumPy, SciPy and Matplotlib. String processing is also generally considered faster and more cohesive than in other programming languages.

Python libraries

One of the main libraries used here is Django, arguably the best known Python web framework. Django was chosen in the detriment of others such as Flask and CherryPy, due to its active and dynamic open source community, its good documentation resources, high number of compatible third party libraries and generally available support with installation, development and running.

For the purpose of this application, a number of libraries were installed via `pip`, the package manager preferred by Python applications. The entire application was build using `virtualenv`, a virtual environment creator that helps Python developers manage different Python versions in different environments, to avoid unnecessary creation of virtual machines. Running the command `pip freeze` inside the activated virtual environment we get the list of installed third party libraries:

```
amqp==1.4.9
anyjson==0.3.3
beautifulsoup4==4.4.1
billiard==3.3.0.23
cassandra-driver==3.4.1
celery==3.1.18
cssselect==0.9.1
cyclr==0.10.0
DistributedLock==1.2
Django==1.8.13
django-annoying==0.9.0
django-m2m-history==0.3.5
django-oauth-tokens==0.6.3
django-picklefield==0.3.2
django-taggit==0.19.1
futures==3.0.5
kombu==3.0.35
lxml==3.6.0
nltk==3.2.1
numpy==1.11.0
oauthlib==1.1.1
pandas==0.18.1
pyparsing==2.1.4
pyquery==1.2.13
python-dateutil==2.5.3
python-memcached==1.58
pytz==2016.4
redis==2.10.3
requests==2.10.0
requests-oauthlib==0.6.1
scikit-learn==0.17.1
scipy==0.17.1
simplejson==3.8.2
six==1.10.0
sklearn==0.0
tweepy==3.5.0
```

The advantage of using the `pip` package manager is that upon installing either library, its dependencies are installed as well. Throughout this thesis I will refer to the list of installed libraries when explaining the choice and particular implementation where each library is used. For now, please note the Django 1.8 installation, which is the current long-term release, chosen for its stability and support.

5.2 Storage with Cassandra

The Cassandra NoSQL database was chosen for this project's non-relational database requirement, due to its large flexibility and scalability to more clusters when the need arises.

A Cassandra server was installed locally and needs to be up for all queries run

from the application. The Python library `cassandra-driver` allows for easy connection to Cassandra clusters and execution of queries, similarly to SQL ones. Empyrial debugging is easy via the `cqlsh` command line utility, which acts like a Cassandra interactive console. Figure 5.1 presents a demo of these functionalities, including starting up the console, connecting to a cluster and submitting a query.

```

[Calinas-MacBook-Pro:~ calina$ cqlsh
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.5 | CQL spec 3.4.0 | Native protocol v4]
Use HELP for help.
[cqlsh> use demo;
[cqlsh:demo> select * from tweet limit 2;

  twitterid      | content                                     | date                | hashtags | history
-----+-----+-----+-----+-----+-----+-----+-----
  737546691679649794 | New bug: 825882 - #python-letsencrypt - python-letsencrypt: Dependency to pyth
on-acme broken... https://t.co/zAy4w6EVtU | 2016-05-31 07:30:08.000000+0000 | null | 316deed4-28
ab-11e6-8873-60f81dce4d6e | null | 0 | DebianBug
  737490189967327232 | RT @blarson424: #python list and dictionaries https://t.co/iA7bFv6LX6
#coding #tutorial https://t.co/2Y0r6lgady | 2016-05-31 03:45:37.000000+0000 | null | 316deed4-28
ab-11e6-8873-60f81dce4d6e | null | 2 | agroinformatik

(2 rows)
cqlsh:demo>

```

Figure 5.1: Demo of the `cqlsh` utility

The same functionality can be mirrored in Python using the following set of instructions:

```

from cassandra.cluster import Cluster

cluster = Cluster()
session = cluster.connect('demo')

harvests = session.execute(
    """
    select * from tweets limit 2
    """
)

```

with the result being a generator which can be further consumed by the application. This means that it is easy to perform queries programmatically, without much overhead, by employing the usage of this driver.

Cassandra queries are used throughout the application, both in synchronous and asynchronous application steps. Its speedy retrieval and updating of records is not a bottleneck, as API rate limits are.

5.3 Harvesting tools

5.3.1 Celery and Redis brokers for asynchronous jobs

5.3.2 Twitter API and the Tweepy library

5.4 Enhancement tools

5.4.1 Simple numerical data

5.4.2 Sci-kit learn

5.4.3 Related hashtag calculation

5.4.4 Related hashtag clustering using K-Means

5.4.5 Page and bot vs. users modeling

TODO

5.5 Normalisation and Analysis tools

TODO

Chapter 6

Testing and Validation

About 5% of the paper

6.1 Title

6.2 Other title

Chapter 7

User's manual

In the installation description section you should detail the hardware and software resources needed for installing and running the application, and a step by step description of how your application can be deployed/installed. An administrator should be able to perform the installation/deployment based on your instructions.

In the user manual section you describe how to use the application from the point of view of a user with no inside technical information; this should be done with screen shots and a stepwise explanation of the interaction. Based on user's manual, a person should be able to use your product.

7.1 Title

7.2 Other title

Chapter 8

Conclusions

About. 5% of the whole
Here your write:

- a summary of your contributions/achievements,
- a critical analysis of the achieved results,
- a description of the possibilities of improving/further development.

8.1 Title

8.2 Other title

Bibliography

- [1] “TIOBE index May 2016,” http://www.tiobe.com/tiobe_index?page=index, 2016, [Online; accessed 5-June-2008].
- [2] K. J. Millman and M. Aivazis, “Python for scientists and engineers,” *Computing in Science & Engineering*, vol. 13, no. 2, pp. 9–12, 2011.

Appendix A

Relevant code

```
/** Maps are easy to use in Scala. */
object Maps {
  val colors = Map("red" -> 0xFF0000,
                   "turquoise" -> 0x00FFFF,
                   "black" -> 0x000000,
                   "orange" -> 0xFF8040,
                   "brown" -> 0x804000)

  def main(args: Array[String]) {
    for (name <- args) println(
      colors.get(name) match {
        case Some(code) =>
          name + " has code: " + code
        case None =>
          "Unknown color: " + name
      }
    )
  }
}
```

Appendix B

Other relevant information
(demonstrations, etc.)

Appendix C

Published papers