Advanced Softwareengineering

Football World Cup Predictor

Github link:

Denis Baskan – 878 571

01 September 2018

Inhalt

[Introduction 3](#_Toc524449073)

[Crawl websites 3](#_Toc524449074)

[Predictor 3](#_Toc524449075)

[Football tournament 4](#_Toc524449076)

[UML 5](#_Toc524449077)

[Use Case Diagram 5](#_Toc524449078)

[Package Diagram 6](#_Toc524449079)

[Activity Diagram – Web Scraper 7](#_Toc524449080)

[Activity Diagram – Predictor 8](#_Toc524449081)

[Sequence Diagram 9](#_Toc524449082)

[Metrics 10](#_Toc524449083)

[Sonar Cloud 10](#_Toc524449084)

[Pylint 11](#_Toc524449085)

[Clean Code Development 12](#_Toc524449086)

[Version Control 12](#_Toc524449087)

[Source Code Conventions 13](#_Toc524449088)

[Precise Naming 13](#_Toc524449089)

[Removing unnecessary comments 14](#_Toc524449090)

[Vertical Separation 15](#_Toc524449091)

[Continuous Delivery 16](#_Toc524449092)

[AOP 17](#_Toc524449093)

[DSL 18](#_Toc524449094)

[Functional Programming 19](#_Toc524449095)

[Final Data Structures 19](#_Toc524449096)

[Side effect free functions 19](#_Toc524449097)

[High order functions – functions as parameters and return values 19](#_Toc524449098)

[Closures / anonymous functions 19](#_Toc524449099)

[Logical Solver 20](#_Toc524449100)

[Scala / Clojure 21](#_Toc524449101)

# Introduction

In football world cup 2018 we could watch many exciting matches. Although Germany has performed very bad, I was motivated to develop code in order to predict scores. Therefore, we need data.



Source: <https://www.kuredu.com/wp-content/uploads/2018/06/fifa.jpg>

## Crawl websites

What data could be relevant for our predictor? Football players usually play not more than three world cups. So my suggestion was to take many matches such as past world and european cups, U20 and U21 matches, qualifying and other matches.

The domain <http://www.weltfussball.de/alle_spiele/> and their sub-domains provide many matches. As I wrote the code snippets, the data of qualifying and other matches were available on <http://www.sportdaten.t-online.de/fussball/> . Unfortunately, this website is no longer accessible. But I’d still had the data stored on my computer.

## Predictor

I decided to use an artificial neural network (ANN) to get some predictions. Inputs are two nations, year and kind e.g. word cup, european cup and so one. Outputs are two score vectors that contain probabilities. The two highest probabilities are our predictions for a given match.

## Football tournament

You might have played a little tournament with your friends and family where points are given in respect to your predictions. If you want to compare your predictor against some of your revivals there is a function to evaluate your results.

# UML

## Use Case Diagram

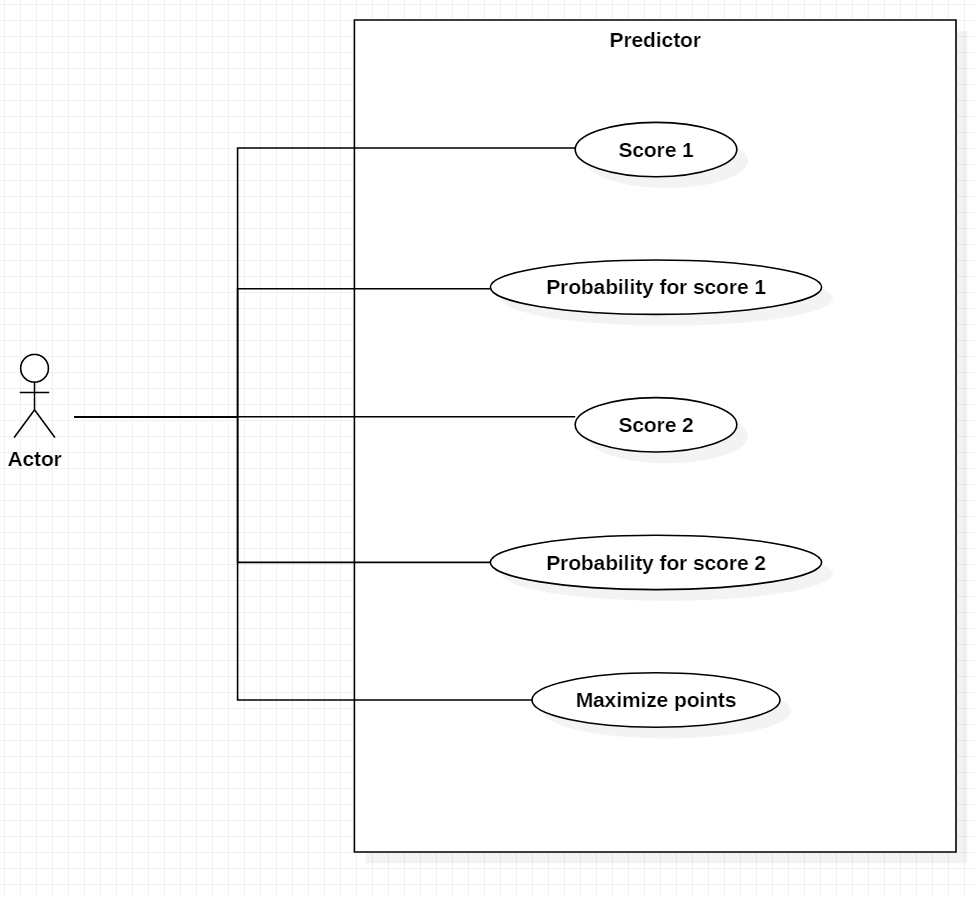


Figure : Use Case Diagram

The actor wants to get predictions for the scores for the next match. He also wants to know how confident the predictions are.

## Package Diagram

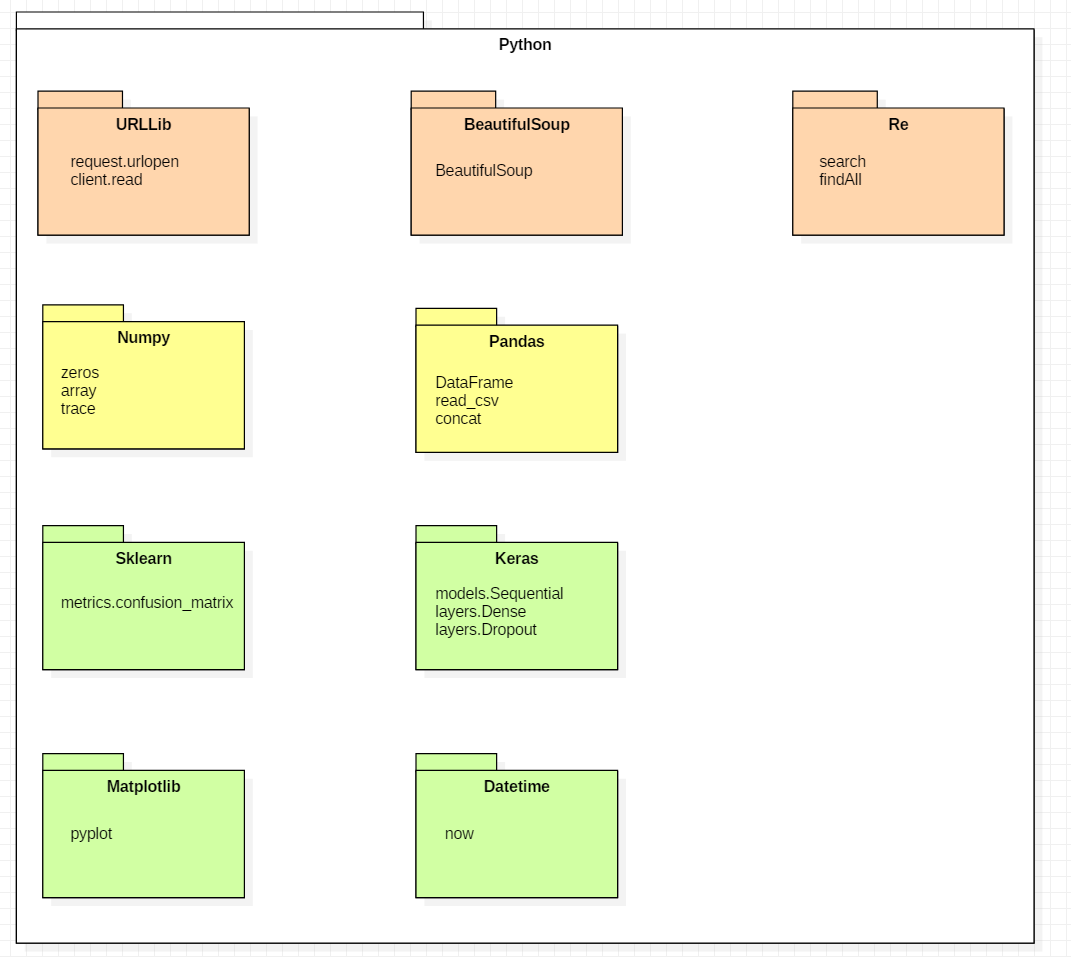


Figure : Package Diagram

These are the necessary packages in python that you need to run the code.

## Activity Diagram – Web Scraper

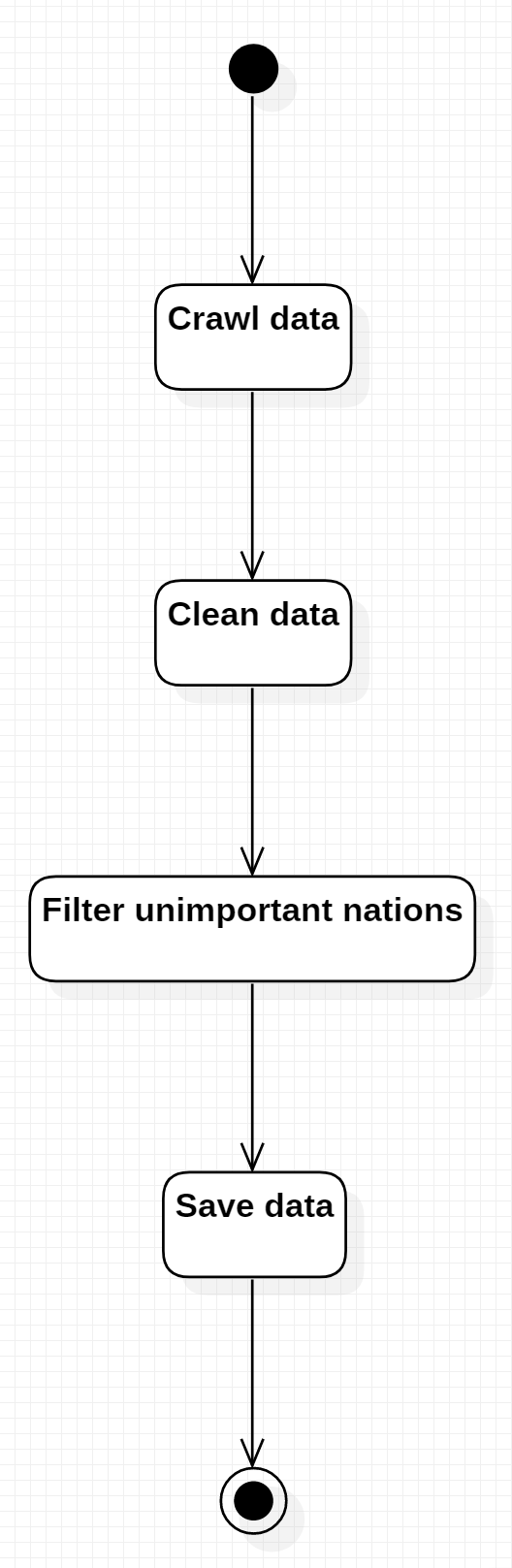


Figure : Activity Diagram – Web Scraper

This diagrams shows the process of getting clean data from the web.

## Activity Diagram – Predictor

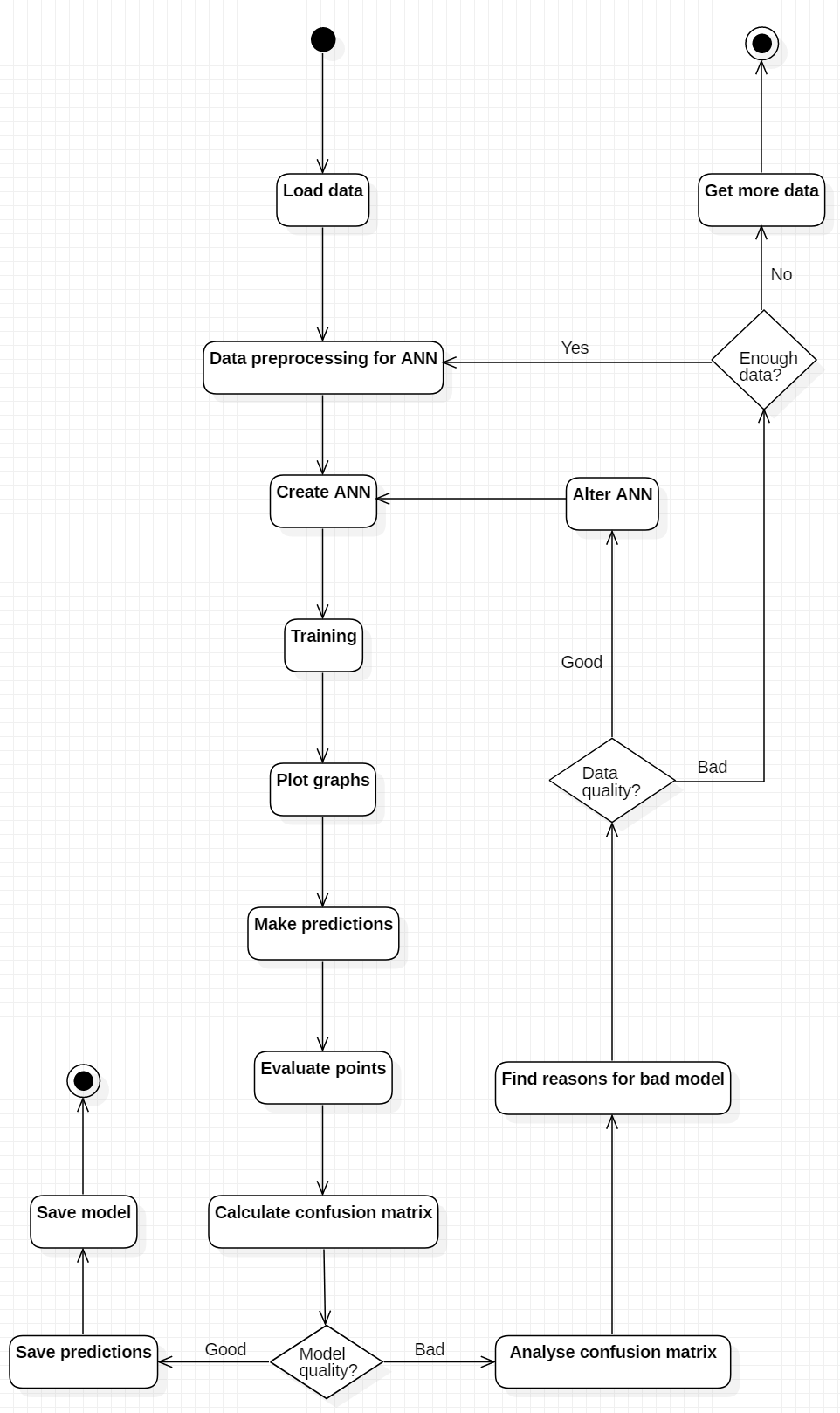


Figure : Activity Diagram – Predictor

This activity diagram shows the complex steps of the predictor.

## Sequence Diagram

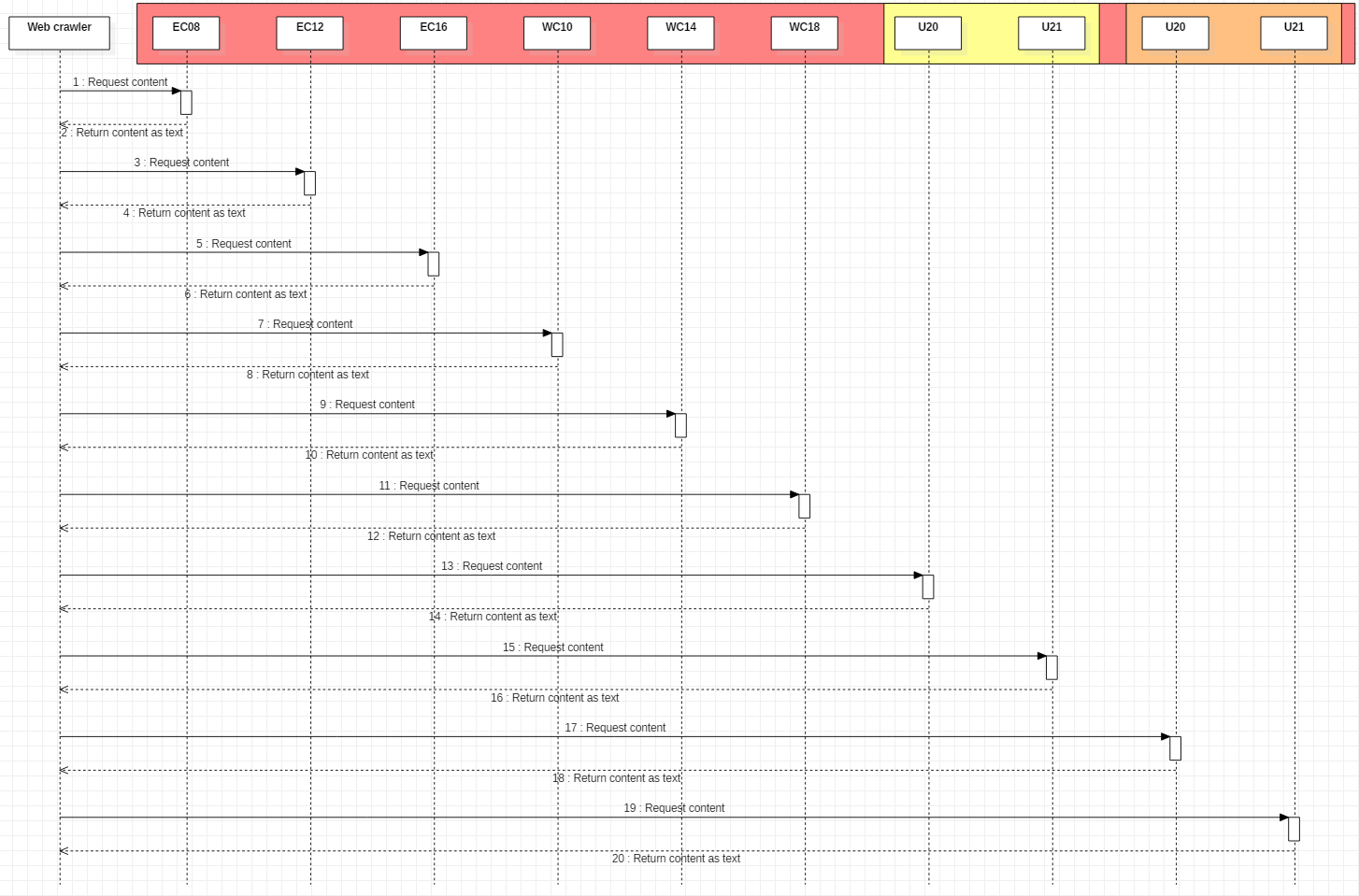


Figure : Sequence Diagram

This Sequence diagram shows the requests to the websites.

# Metrics

## Sonar Cloud

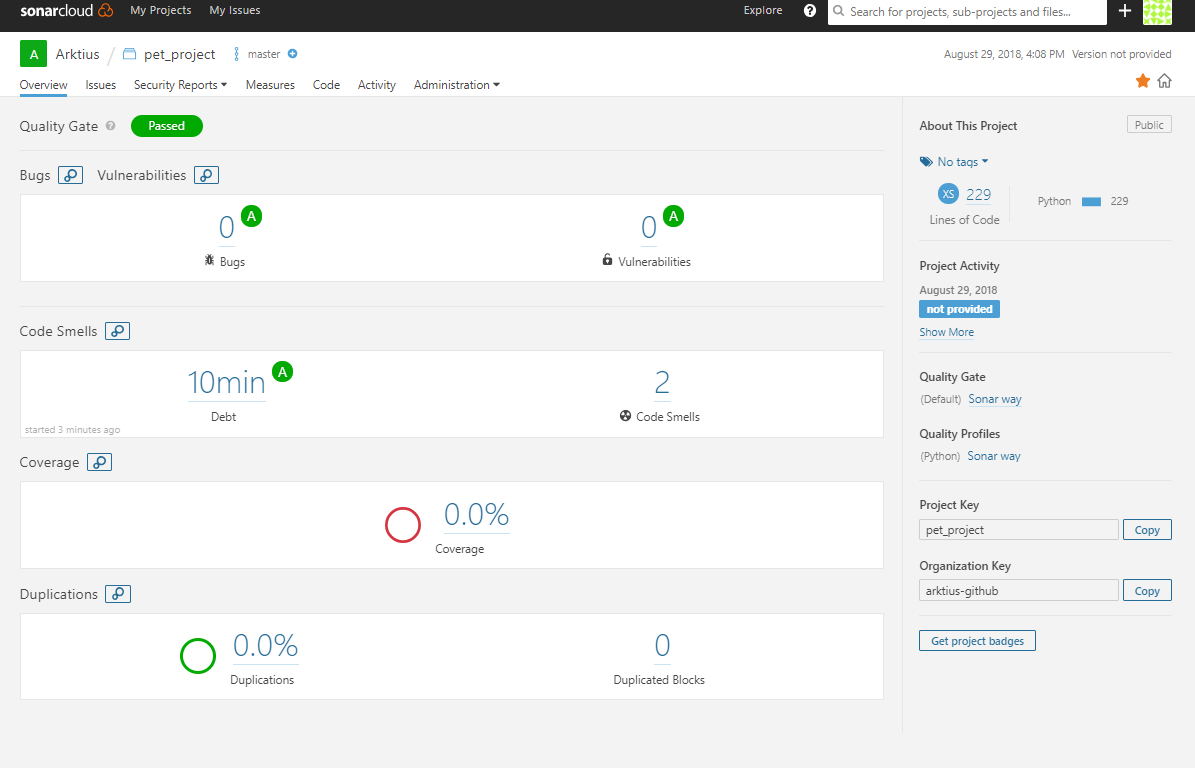


Figure : Sonar Cloud result

The result is pretty fine. Code smells were detected, because I commented two packages out which might be needed in the future.

## Pylint

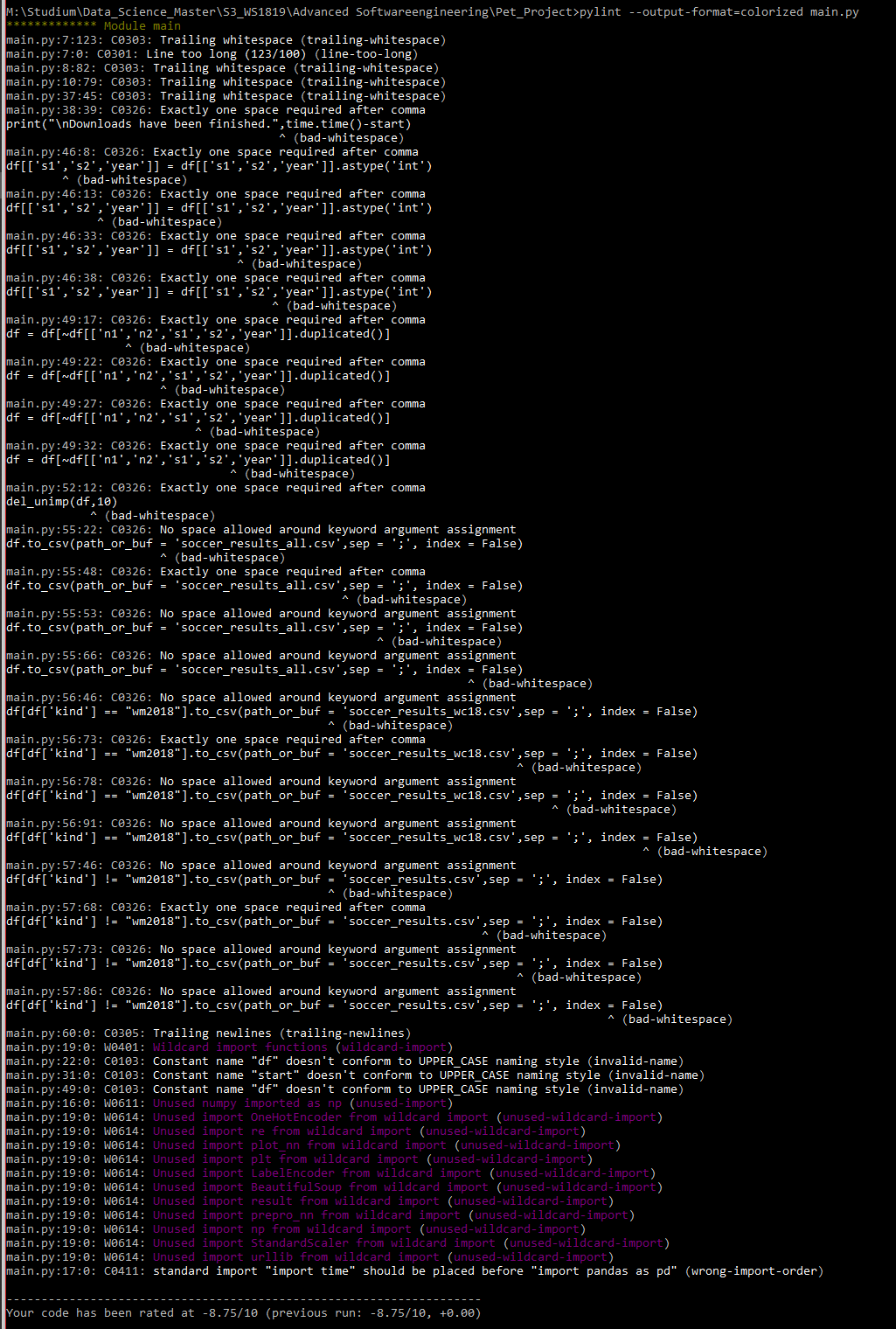


Figure : Pylint

Pylint was more informative than Sonar. Pylint has showed me way more things that need to be corrected such as all the whitespaces after a comma, name styles and unused imports.

# Clean Code Development

## Version Control

Since the beginning of this project, git was used as version control system.

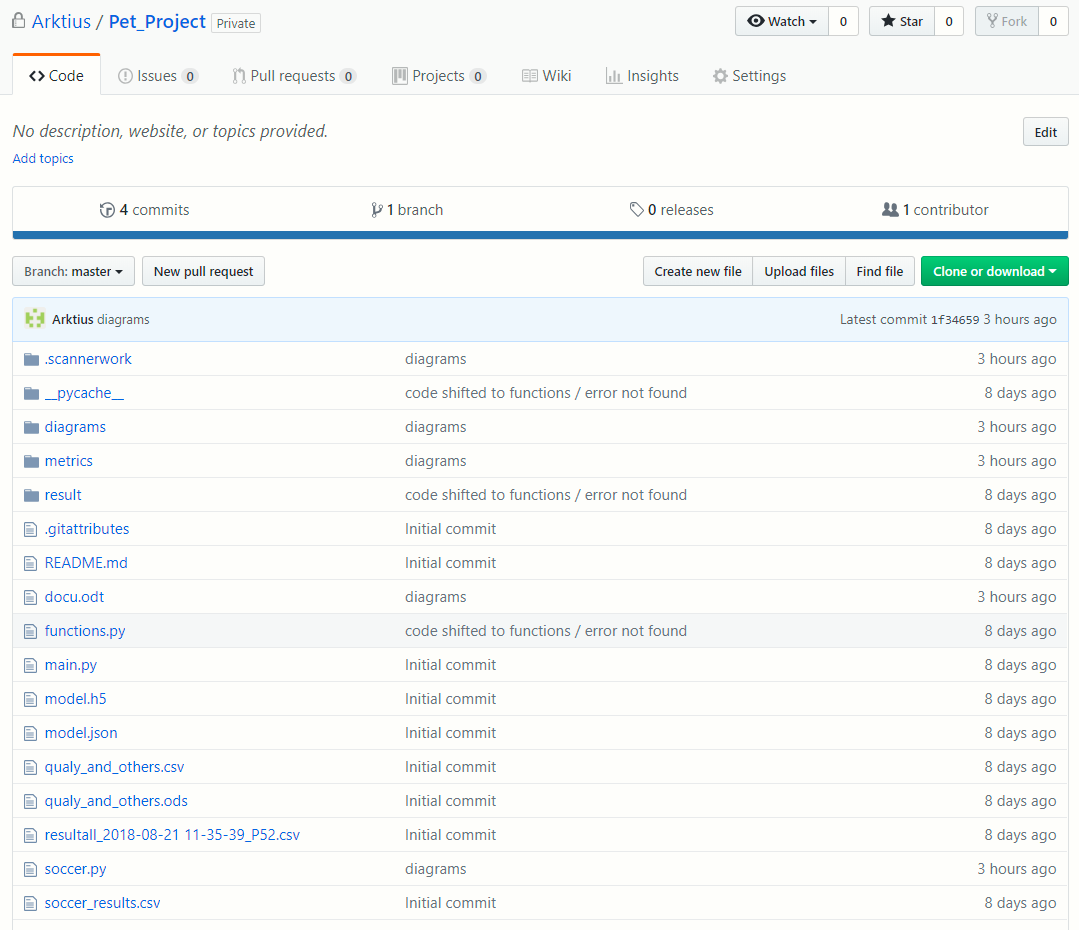


Figure : GitHub Repository

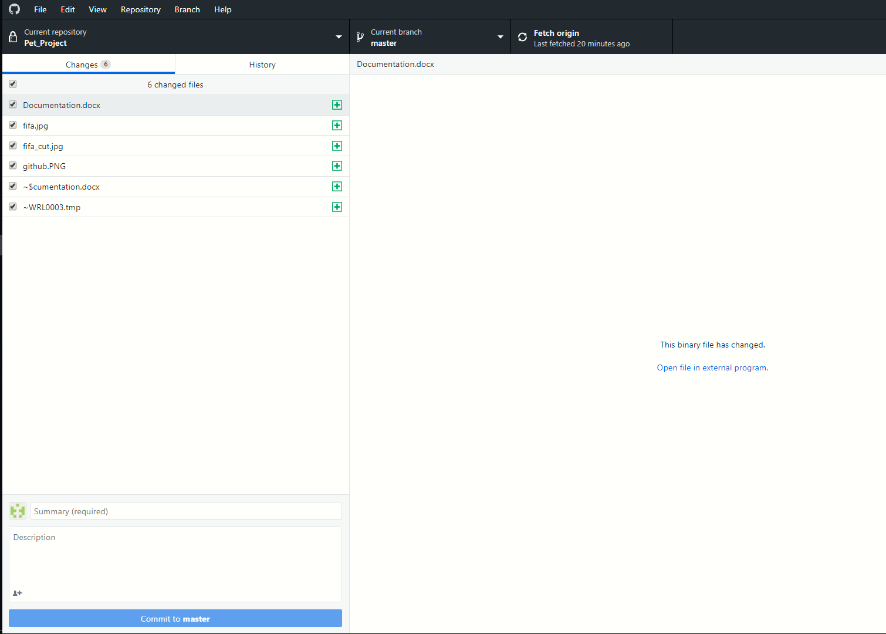


Figure : GitHub for Desktop

To handle the operations easily, I have been using GitHub for Desktop all the time.

## Source Code Conventions

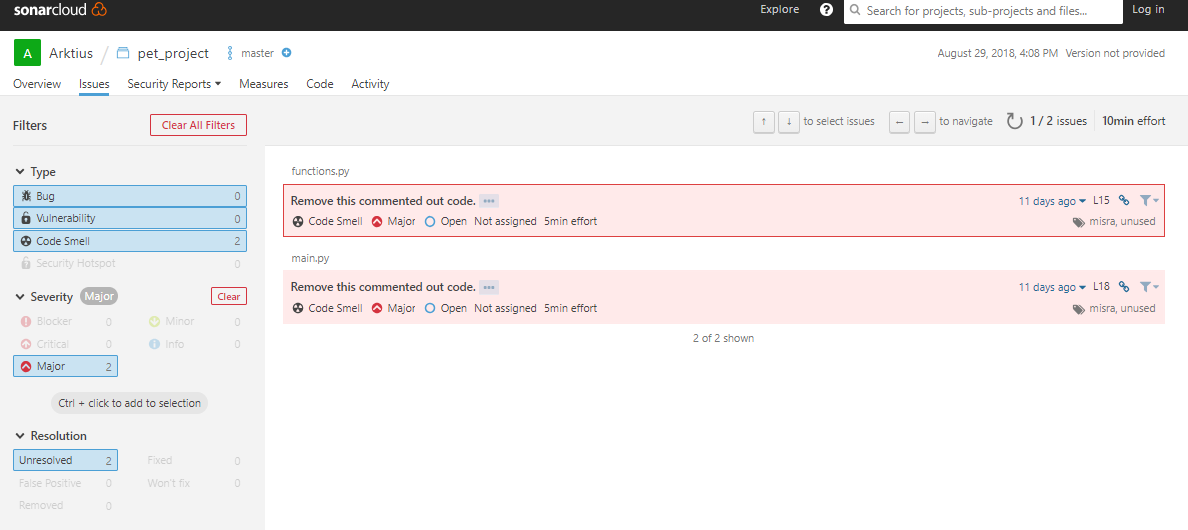


Figure : Sonar Cloud detects code smells

## Precise Naming

In this project, functions and variables were considered to be named as precise but still simple as possible. This functions iterates through all domains or URLs called. There is one mistake highlighted in red. The iterator i should have named clearer. Since cup only contains URLs, the iterator could be named ‘url’.

def crawl\_cups(df):

print() #prints empty line

domain = r'http://www.weltfussball.de/alle\_spiele/' #domain of matches

cup = [] #list to store cups

#sub-URLs European / World Cup

cup.append(domain + r'em-2016-in-frankreich/')

#go through all European / World Cups

for i in range(0,len(cup)):

year = int(re.search('\d+',cup[i]).group())

kind = cup[i][39:41]

#crawl website

client = urllib.request.urlopen(cup[i])

page = client.read()

soup = BeautifulSoup(page,"lxml")

## Removing unnecessary comments

Since I’ve been working as a software developer in a company for more than a year, I tend to write more comments than might be needed. More comments lead to less questions from colleagues later on, but clear given names for variables and a well-documented paper make comments superfluous. Nevertheless, comments before loops and for bunches of code lines are reasonable.

def crawl\_cups(df):

print() #prints empty line

domain = r'http://www.weltfussball.de/alle\_spiele/' #domain of matches

cup = [] #list to store cups

#sub-URLs European / World Cup

cup.append(domain + r'em-2016-in-frankreich/')

#go through all European / World Cups

for i in range(0,len(cup)):

year = int(re.search('\d+',cup[i]).group())

kind = cup[i][39:41]

#crawl website

client = urllib.request.urlopen(cup[i])

page = client.read()

soup = BeautifulSoup(page,"lxml")

## Vertical Separation

Variables should be used immediately after they are created. In this function the variables year and kind are good examples where this is not happened. They could have created inside the if in the inner for loop where they will be used.

#go through all European / World Cups

for i in range(0,len(cup)):

year = int(re.search('\d+',cup[i]).group())

kind = cup[i][39:41]

#crawl website

client = urllib.request.urlopen(cup[i])

page = client.read()

soup = BeautifulSoup(page,"lxml")

#go through all matches in a cup

for tr in soup.findAll('table',{"class": r"standard\_tabelle"})[0].findAll('tr'):

td = tr.findAll('td')

if len(td) == 8:

year = int(re.search('\d+',cup[i]).group())

kind = cup[i][39:41]

score = re.findall('\d+',td[5].getText())

df.loc[len(df)] = [td[2].getText(),td[4].getText(),score[0],score[1],year,kind]

#translate into english for printing

if kind == 'em':

kind = kind.replace('em','EC')

else:

kind = kind.replace('wm','WC')

print("Data of {}-{} were downloaded.".format(kind,year))

# Continuous Delivery

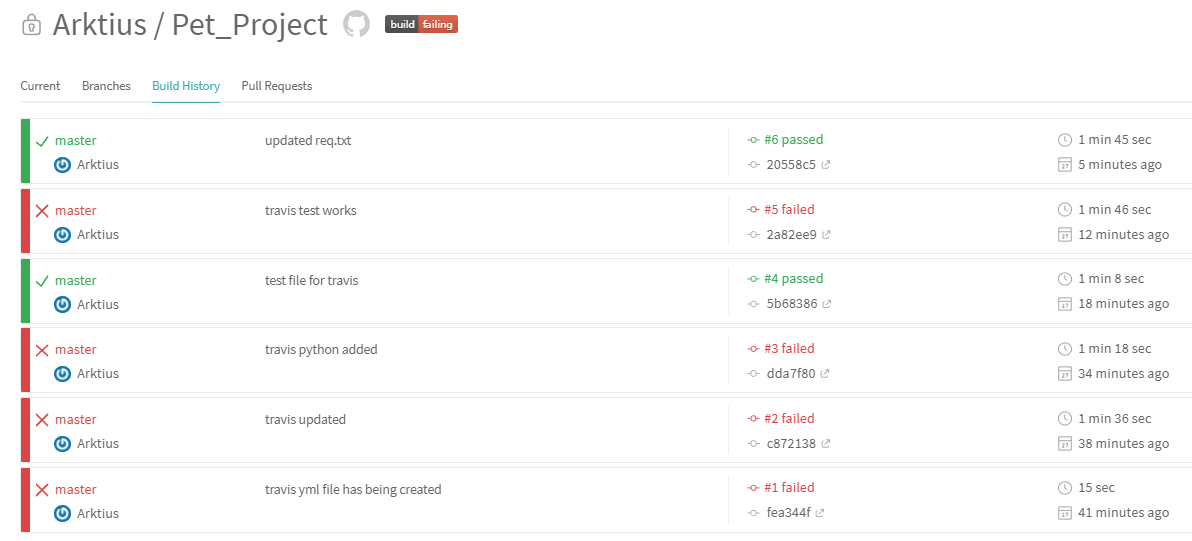


Figure : Build history in Travis-CI

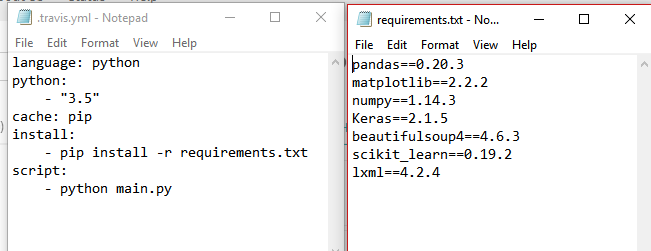


Figure : Needed files for Travis-CI

In Order to get the requirements of a project you can use the python package pipreqs. It can be installed by typing the following line in your terminal:

Pip install pipreqs

Navigate then to the folder above your project and run this command:

pipreqs --force Pet\_Project

# AOP

Aspect Oriented Programming is applied by so called decorators in Python. These decorators are high order functions that wrap their arguments with code and usually precede a function. An important information for the user are the responses while the program is downloading data. Therefore, the timestamp could be printed out while data are being downloaded.

def time\_execution(f):

def wrapped(\*args, \*\*kws):

now = datetime.datetime.now()

print('[' + str(now) + '] Call Function: ' + f.\_\_name\_\_ + '()')

return f(\*args, \*\*kws)

return wrapped

@time\_execution

def crawl\_cups(df):

#download data…

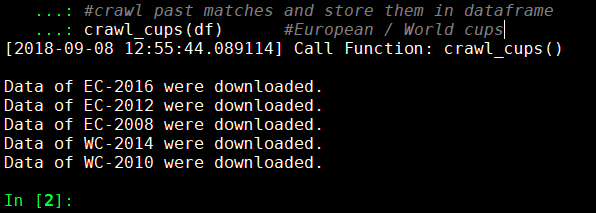


Figure : Decorator in usage

# DSL

# Functional Programming

## Final Data Structures

Final data structures as known in other programming languages as final variables (Java) or constants (C++) are not implemented in Python. So, assign a value to a variable only once and never change it again.

## Side effect free functions

Since I only use each functions once, all functions are nearly side effect free. There are few exceptions where I have been using methods of Pandas Dataframes, but these calls are irreplaceable.

## High order functions – functions as parameters and return values

High order functions accept other functions as arguments or return functions as values. Decorators, as listed above in section AOP, are high order functions and return functions as a value.

## Closures

A Closure is a function object that remembers values in enclosing scopes even if they are not present in memory. To avoid the usage of global variables, Closures can be used. [[source]](https://www.geeksforgeeks.org/python-closures/)

Example:

# Python program to illustrate

# nested functions

def outerFunction(text):

text = text

def innerFunction():

print(text)

innerFunction()

if \_\_name\_\_ == '\_\_main\_\_':

outerFunction('Hey!') #prints ‘Hey!’

## Anonymous Functions

Anonymous functions are functions without a name. You can use the keyword lambda for this in Python. I could have used a lambda function to save multiple rows.

Usage of a lambda function in project:

def umlaute(df):

#replace umlaute

df[['n1','n2']] = df[['n1','n2']].replace(to\_replace = 'Rumänien', value ='Rumaenien')

df[['n1','n2']] = df[['n1','n2']].replace(to\_replace = 'Österreich', value ='Oesterreich')

…

#code with lambda function

#replace = ['Rumänien',...]

#value = ['Rumaenien',...]

#f = lambda replace,val: df[['n1','n2']].replace(to\_replace = replace, value = val)

#df[['n1','n2']] = f(replace,value)

# Logical Solver

# Scala / Clojure