# Enterprise datascience

**Coding like a pro**

*Michelangelo Puliga @linkalab*

# Summary

- Objectives
- Getting the data
- Monitoring & restarting the process
- Saving data in a reliable way
- Using redis to store mention networks
- Real time mention networks with d3js

# Objectives

- Are you a Data Scientist *and* good at programming ?
- Are you R or Python ? no matter what, you need to code properly
- How to *go pro* ?
  - **Getting *all* data**
  - **Being *reliable***: is your code getting the similar results every download ?
  - **Being *efficient*:** is your code fitting in RAM and not wasting too much disk space ?
  - ***Keep track* of errors**: can you explain what gone wrong ?
  - **Resume when the script crashes with errors**: can you restart from the last known data ?
  - ***Clean* the code**: are you able to understand your own code after a while ?

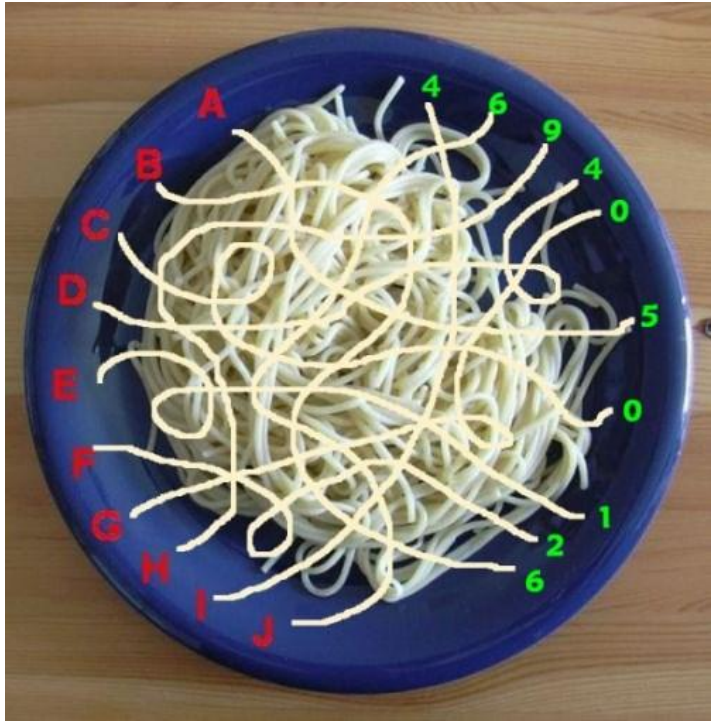
# A modern programmer...

- Do you write **spaghetti code**, or functions and classes ?
- Are you planning to document your code ? ever heard about ***docstrings*** ?
- Do you know what are the ***unit tests*** ? can you foresee a test driven programming in your job ?
- Do you speak **Git** ? are you good at saving/cleaning your stuff ?
- **Dog Food**. Are you really using your code ?
- Do you know what is ***code profiling*** ? Are you wasting CPU / RAM ?
- Can you make a ***full stack*** application ?

# Spaghetti code ...

A code that has a lot of complicated dependencies

You repeat yourself many times, you do not use functions nor classes



# Spaghetti code ...

```python
import warnings
warnings.filterwarnings("ignore")

l = 0

start = 0
end = 22+12

lenght_pred = 6

prediction_values = []
prediction_Nplus2 = []

for r in dataforP.iterrows():
    ytrain = [r[1][c]+1 for c in colsxTrain[start:end]]
    ytot = [r[1][c]+1 for c in colsxTrain]

    # create db for forecast
    df = pd.DataFrame(index=colsxTrain[start:end], data={'y' : log(ytrain), 'ds' : colsxTrain[start:end]})

    # train
    m = Prophet(yearly_seasonality=False,
                weekly_seasonality=False,
                daily_seasonality=False,
                n_changepoints = 8)
    m.fit(df)
    # predict
    future = m.make_future_dataframe(periods=lenght_pred, freq='M5')
    forecast = m.predict(future)

    #save 6 months prediction
    prediction_values.append(forecast.yhat)

    pred_len = len(forecast)

    prediction_Nplus2.append(
        {'sku': r[0],
         'prediction': round(exp(forecast.yhat[pred_len-4]))-1,
         'pred_lower': round(exp(forecast.yhat_lower[pred_len-4]))-1,
         'pred_upper': round(exp(forecast.yhat_upper[pred_len-4]))-1,
         'actual':  ytot[pred_len-4]-1,
         'date': forecast.ds[pred_len-4]
            })
    l=l+1
    if(l%20 == 0):
        print(l)

##########
# Jan df #
##########

prediction_Nplus2_Jan = pd.DataFrame(prediction_Nplus2)

prediction_Nplus2_Jan['prediction'] = abs(prediction_Nplus2_Jan['prediction'])
prediction_Nplus2_Jan['pred_lower'] = abs(prediction_Nplus2_Jan['pred_lower'])
prediction_Nplus2_Jan['pred_upper'] = abs(prediction_Nplus2_Jan['pred_upper'])

prediction_Nplus2_Jan['error']= prediction_Nplus2_Jan.prediction - prediction_Nplus2_Jan.actual
prediction_Nplus2_Jan['mape'] = round((abs(prediction_Nplus2_Jan.error)/ prediction_Nplus2_Jan.prediction)*100)
prediction_Nplus2_Jan['precision'] = round((prediction_Nplus2_Jan.error/ prediction_Nplus2_Jan.prediction)*100)

prediction_Nplus2_Jan.to_csv('prediction_Nplus2_19Jan2-ALLSKU-A.csv')
```

The code for Jan

```python
import warnings
warnings.filterwarnings("ignore")

l = 0

start = 0
end = 30+12
lenght_pred = 6

prediction_values = []
prediction_Nplus2 = []

for r in dataforP.iterrows():
    ytrain = [r[1][c]+1 for c in colsxTrain[start:end]]
    ytot = [r[1][c]+1 for c in colsxTrain]

    # create db for forecast
    df = pd.DataFrame(index=colsxTrain[start:end], data={'y' : log(ytrain), 'ds' : colsxTrain[start:end]})

    # train
    m = Prophet(yearly_seasonality=False,
                weekly_seasonality=False,
                daily_seasonality=False,
                n_changepoints = 8)
    m.fit(df)
    # predict
    future = m.make_future_dataframe(periods=lenght_pred, freq='M5')
    forecast = m.predict(future)

    #save 6 months prediction
    prediction_values.append(forecast.yhat)

    pred_len = len(forecast)

    prediction_Nplus2.append(
        {'sku': r[0],
         'prediction': round(exp(forecast.yhat[pred_len-4]))-1,
         'pred_lower': round(exp(forecast.yhat_lower[pred_len-4]))-1,
         'pred_upper': round(exp(forecast.yhat_upper[pred_len-4]))-1,
         'actual':  ytot[pred_len-4]-1,
         'date': forecast.ds[pred_len-4]
            })
    l=l+1
    if(l%20 == 0):
        print(l)

##########
# Sep df #
##########

prediction_Nplus2_Sep = pd.DataFrame(prediction_Nplus2)

prediction_Nplus2_Sep['prediction'] = abs(prediction_Nplus2_Sep['prediction'])
prediction_Nplus2_Sep['pred_lower'] = abs(prediction_Nplus2_Sep['pred_lower'])
prediction_Nplus2_Sep['pred_upper'] = abs(prediction_Nplus2_Sep['pred_upper'])

prediction_Nplus2_Sep['error']= prediction_Nplus2_Sep.prediction - prediction_Nplus2_Sep.actual
prediction_Nplus2_Sep['mape'] = round((abs(prediction_Nplus2_Sep.error)/ prediction_Nplus2_Sep.prediction)*100)
prediction_Nplus2_Sep['precision'] = round((prediction_Nplus2_Sep.error/ prediction_Nplus2_Sep.prediction)*100)

prediction_Nplus2_Sep.to_csv('prediction_Nplus2_19Sep2-A.csv')
```

And the same code for Sep !

# Docstrings

docstrings.py

```
8
9   def multiply(a, b):
10      """This method multiplies the given two numbers.
11
12      Input Arguments: a, b must be numbers.
13      Returns: Multiplication of a and b.
14      """
15      return a * b
16
17
18  class Employee:...
37
38
39  print(multiply.__doc__)
40
```

Accessing Python Function Docstring

Run: docstrings

```
/Users/pankaj/Documents/PycharmProjects/PythonTutorialPr
This method multiplies the given two numbers.

    Input Arguments: a, b must be numbers.
    Returns: Multiplication of a and b.


Process finished with exit code 0
```

Output

Adding documentation is easy and the *help* will help !

# Unit tests

```python
# -*- coding: utf-8 -*-
"""
Created on Sat Apr 25 20:16:58 2020

@author: Aditya
"""
from volume_cuboid import *
import unittest

class TestCuboid(unittest.TestCase):
    def test_volume(self):
        self.assertAlmostEqual(cuboid_volume(2),8)
        self.assertAlmostEqual(cuboid_volume(1),1)
        self.assertAlmostEqual(cuboid_volume(0),0)

    def test_input_value(self):
        self.assertRaises(TypeError, cuboid_volume, True)
```

If possible write the tests along the main code. The testing suites will make the debug extremely easy.

In data science along the code testing we need the *data for testing*

# Git git git git and still git

```
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   public/index.html
#       deleted:    public/sitemap.xml
#       new file:   public/stylesheets/mobile.css
#
# Changes not staged for commit:
#   (use "git add/rm <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       deleted:    app.rb
#       deleted:    test/add_test_crash_report.sh
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       public/javascripts/
```

Git is your friend for saving code, versioning, sharing and cleaning...

# Dog food: are you using your code ?

## Evolution of Dogfooding

centercode


**Alpo's Dog Food Advertisements**

Alpo, with the help of Lorne Greene, convinced consumers to buy their products because they themselves used them.

**Microsoft Email to Employees**

Paul Maritz sends an email titled "Eating our own Dogfood" to challenge his test manager to increase internal usage of the company's product.

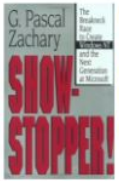
**Project "Alpo" Goes Live**

Hewlett-Packard and Mozilla create "Project Alpo" to push employees to use their own products.

1970s — 1980 — 1988 — 1991 — 1999 — 2000-Present

**Memo from the CEO of Apple**

"We believe the typewriter is obsolete. Let's prove it inside before we try and convince our customers." - Michael Scott




**Microsoft Windows NT**

Dave Cutler lead his team to develop the largest software program Microsoft had seen to date with dogfooding to shake out the bugs.

**It's Pretty Much Adopted**

Now most companies run some kind of internal testing or "dogfooding" program.

# Code profiling

Measure CPU time, RAM memory and execution time of your code

Track the execution

In jupyter just put

***%%time***

On a cell

```
In [1]: import pympler.tracker as tracker

In [2]: import trie_serializer

In [3]: mem = tracker.SummaryTracker()

In [4]: t = trie_serializer.load('10000000')
Loading 10000000 Trie...
10000000 Trie loaded!

In [5]: mem.print_diff()
                                          types |   # objects |   total size
================================================ | =========== | ============
                <class 'compact_trie.CompactTrie |     4629599 |    388.53 MB
                                      <class 'list |      854338 |    168.24 MB
                                     <class 'float |     3845716 |     88.02 MB
                                     <class 'bytes |      290863 |      9.71 MB
                                      <class 'dict |       12268 |      1.67 MB
                                       <class 'str |        9927 |    685.70 KB
          <class 'parso.pgen2.generator.DFAPlan |        5203 |    284.54 KB
                                       <class 'set |         774 |    237.81 KB
      <class 'parso.pgen2.grammar_parser.NFAArc |        2472 |    135.19 KB
    <class 'parso.pgen2.grammar_parser.NFAState |        2176 |    119.00 KB
             <class 'parso.python.tree.Operator |         923 |     86.53 KB
                 <class 'parso.python.tree.Name |         688 |     53.75 KB
            <class 'parso.python.tree.PythonNode |         822 |     51.38 KB
                                       <class 'int |        1651 |     45.16 KB
         <class 'parso.pgen2.generator.DFAState |         770 |     42.11 KB

In [6]:
```

# Full stack

From Exploratory Data Analysis

To a deployed application

This is the full stack journey of software development

# Part 2

A full stack application

# A Twitter app for live exploration of mention networks

- Get ***stream*** data from Twitter
- Save them *safely*
- Compute mention network
- Upload code and data to github
- Visualize live networks

# Register on Twitter Developer

Get the API Keys from Twitter dev

Install Twython

*Read the fucking manual* (RTFM)

```
from twython import TwythonStreamer
from twython import Twython
import sys


APP_KEY = 'YH0TYc2Uui2fzdYE0vEAQXepV' ## application key
APP_SECRET = 'IzO3XN***' ## application secret
OAUTH_TOKEN =  '245734683***' ## consumer key
OAUTH_TOKEN_SECRET =  'BuGtydrhvDGjce6qN***' ## consumer secret
```

# Problem 1

Keys are along the main code

# Problem 2

Passing parameters , how to =

# Problem 3

Add a configuration file ?

# Problem 4

Tweets are truncated !

# Problem 5

We are losing data at every crash !

# Problem 6

How to extract networks ? first try

# Problem 7

How to extract networks ? second try

# Using REDIS

How about a professional key-value database ?

Expire the keys !

# Using REDIS to build networks

Get the live network from Redis

# Go live with github

Yes you have your free space on github to publish your research :)