Contents

1 Explorer

Proper program.

1.1 Imports

1.1.1 ipython

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
```

1.1.2 pandas

```
import pandas as pd
```

1.2 Data load

1.2.1 load everything

1. file list with path

```
import os
csv_files = []
date = "2018-09-30"
for dirpath, dirs, files in os.walk("../data/raw/" + date):
  for filename in files:
    fname = os.path.join(dirpath,filename)
```

```
    if fname.endswith('.csv'):
      csv_files.append(fname)
```

2. dataframe creation

```
jobs = pd.DataFrame()

for fl in csv_files:
    print(fl+(30-len(fl)//2)*" *")
    try:
jobs_set = pd.read_csv(fl)
jobs_set.dropna(axis=0, how='any', subset=["desc"], inplace=True)
jobs_set.drop_duplicates(subset="desc", inplace=True)
try:
    jobs.iloc[0,0]
    jobs = jobs.append(jobs_set)
except IndexError:
    jobs = jobs_set
    except pd.errors.EmptyDataError:
pass
```

3. **TODO** time range selection

1.2.2 rename

use to quickly reset original df

```
df = jobs
```

1.2.3 python example TEST

```
x = 12
return x

return int(x)+1
```

1.2.4 org doc elisp example TEST

1
2
3
4
3

1.2.5 python PYTHON

```
"~/data/projects/jobseeker/data/raw/18-09-07/dsp.csv"

"~/data/projects/jobseeker/data/raw/18-09-07/dsp.csv"

"~/data/projects/jobseeker/data/raw/18-09-07/python.csv"

"~/data/projects/jobseeker/data/raw/18-09-07/data scientist.csv"

"~/data/projects/jobseeker/data/raw/18-09-07/software engineer.csv"
```

1.3 Cleansing / Formating CLEAN

1.3.1 duplicates

1. $\text{drop}_{\text{duplicates}}$

   ```
   df.drop_duplicates(subset="desc", inplace=True)
   ```

2. count

   ```
   df.title.count()
   ```

1.3.2 olders

1. map lambda TEST

   ```
   df = df[df.days_ago.str.contains("30+").map(lambda x: not x)]
   ```

2. ==False

   ```
   df = df[df.days_ago.str.contains("30+")==False]
   ```

3. count

   ```
   len(df)
   ```

1.3.3 string numbers to integers

```
df["days_ago"] = df.days_ago.apply(lambda x: int(x))
```

1.3.4 drop erratic values

1. run

   ```
   df = df[df.days_ago.lt(30)]
   ```

2. tests

   ```
   df.days_ago.lt(30)
   ```

1.3.5 rename

```
df_clean = df
```

1.4 Filtering

1.4.1 Look for 1 keywords

1. keyword definiton
   (a) org variable

   ```
   "kunst und medien"
   ```

2. look in title
   (a) boolean serie construction TEST

   ```
   df.title.str.contains(k, case=False)
   ```

   (b) reduction of our dataset

   ```
   df = df[df.title.str.contains(k, case=False, na=False)]
   ```

3. look in description

   ```
   df = df[df.desc.str.contains(k, case=False, na=False)]
   ```

4. **TODO** test goto Johnny Kitchin

   ```
   k
   ```

1.4.2 Queries

1. get queries metadata

   (a) dataframe using os results

```
import os
queries_name = []
queries_size = []
queries_path = []
queries_time = []
for dirpath, dirs, files in os.walk("../data/raw"):
  for filename in files:
    if filename.endswith('.csv'):

      path = os.path.join(dirpath, filename)
      queries_path.append(path)

      size = os.path.getsize(path)
      queries_size.append(size)

      fname = filename.replace(".csv", "")
      queries_name.append(fname)

      time = os.path.getmtime(path)
      queries_time.append(time)

queries = pd.DataFrame({"name" : queries_name, "path" : queries_path, "size"
```

   (b) remove oldests results

      i. datetime time format

```
from datetime import datetime
queries["time"] = queries.time.apply(datetime.fromtimestamp)
```

      ii. y-m-d format time

```
def format_time(x):
    y = x.strftime("%Y-%m-%d")
    return y

queries["time_formated"] = queries.time.apply(format_time)
```

   (c) remove null size results

```
queries_null = queries[queries["size"] < 1]
queries = queries[queries["size"] > 1]
```

(d) number of entries in csv file

i. read as pandas dataframe

```
def entries_count(csv):
    return len(pd.read_csv(csv))

queries["entries"] = queries.path.apply(entries_count)
```

(e) inspection

```
import humanize
queries["size_for_humans"] = queries["size"].apply(humanize.naturalsize)
queries.sort_values("size", ascending=False)[["name", "size_for_humans", "ent
```

(f) time evolution

(g) return list for next scraper launch

i. remove null size results before (or not)

```
queries_list = list(set(queries.name))
```

ii. save in a file for editing

```
with open("/queries/queries.txt", "w") as f:
    for query in queries_list:
f.write(query + "\n")
```

2. launch scraper with the list

(a) get list from file

```
with open("queries/best.txt", "r") as f:
    queries_selected = f.read()
    queries_selected = queries_selected.splitlines()
```

(b) run shell script as subprocess

i. variables and imports

```
import subprocess
from subprocess import Popen, PIPE
import shlex

cwd = '/home/teddd/data/projects/jobseeker/data/external/indeed/'
bash_script = [cwd + 'local_crawler_launch.sh']
arguments = queries_selected
command = bash_script + arguments
```

ii. execution

A. stdout to buffer

```
session = subprocess.Popen(command, stdout=PIPE, stderr=PIPE)
stdout, stderr = session.communicate()

if stderr:
    raise Exception("Error "+str(stderr))

stdout
```

B. stdout to file

```
from datetime import datetime
date = str(datetime.now())
with open("../data/external/crawl-log-" + date + ".txt",'w') as temp_f
    crawl = subprocess.Popen(command, stdout=temp_file, cwd=cwd)
```

1.4.3 Look for multiple keywords

1. tool: keywords list use results from Queries
2. reduce dataframe

(a) boolean serie

```
df_bool = pd.DataFrame()
for query in queries_selected:
    df_bool[query] = df.desc.str.contains(query)
```

(b) binary serie

```
def bool_to_bin(x):
    if x is True:
return 1
    else:
return 0

df_bin = pd.DataFrame()

for query in queries_selected:
    df_bin[query] = df_bool[query].apply(bool_to_bin)
```

(c) score attribution

i. overview

```
pd.concat({"title":df.title, "score":df_bin.sum(axis=1)}, axis=1).sort_val
```

ii. reduce dataframe for visual exploration

```
df_print = df
df_print["score"] = df_bin.sum(axis=1)
df_print = df_print.sort_values("score", ascending=False)
```

3. guide: used words

   (a) amongst keywords

   ```
   df_bin.sum().sort_values(ascending=False)
   ```

4. which contains most of the querie keywords ?

5. add weight to keywords ?

6. keywords distance map with all keywords, you are at the center

1.4.4 companies

```
df = df[df.company.str.contains("berlin", case=False, na=False)]
```

1.5 Stats

1.5.1 overview

1. head

   ```
   df.head()
   ```

2. count

   ```
   len(df)
   ```

1.5.2 days ago

1. histogram

   (a) pd plot

   ```
   df.days_ago.plot.hist()
   ```

2. value count

   ```
   df.days_ago.value_counts()
   ```

3. groupby

   (a) basic output

   ```
   df.groupby(["days_ago"]).groups
   ```

   (b) loop print

   ```
   grouped = df.groupby("days_ago")

   for name,group in grouped:
       print(name)
       print(group)
   ```

1.6 Printing

1.6.1 quick overview

1. head

   ```
   df.head()
   ```

2. count

   ```
   df.title.count()
   ```

3. titles

   ```
   df.title
   ```

1.6.2 html pages

1. hacked around solution TEST

   (a) function to save results to html

   ```
   from datetime import datetime
   from os import mkdir

   def htmlexport(df, begin, end):
       date = str(datetime.now())
       path = "../reports/html/" + date + "/"
       mkdir(path)
       for i in range(begin, end):
   ```

```
html = ""
html = html + "\n"
html = html + "Job number " + str(i)
html = html + "\n"
html = html + "-"*100
html = html + "\n" + df.title.iloc[i]
html = html + "\n"
html = html + df.company.iloc[i]
html = html + "\n"
html = html + "-"*100
html = html + "\n"
html = html + df.desc.iloc[i]
html = html + "\n"*3
html = html + "-"*100
html = html + "\n"*3
filename = path + "job-" + str(i) + ".html"
with open(filename, "a") as file:
    file.write(html)
```

(b) call function

```
htmlexport(dfk, 0, dfk.title.count())
```

(c) PB : imossible to add links because of some encoding pb

2. use xml.dom TEST

(a) use

```
from xml.dom import minidom
minidom.parseString(dfk.desc.iloc[10])
```

(b) PB : some descs are separated by comas

i. change spider
ii. use regexp to parse again
iii. test with proper html files : maybe it is just not working with html ?

```
from xml.dom import minidom
minidom.parseString("~/code/web/plasma-city/application/static/front.html'
```

3. use yattag

(a) imports

```
from datetime import datetime
from os import mkdir
from yattag import Doc
```

(b) html page generation

i. functions definition

```
def linksgen(filename_base, pagenum, url):
    doc, tag, text = Doc().tagtext()

    with tag("div"):
with tag('a', href = "."):
    text('Home page')
with tag("div"):
    with tag("a", href = filename_base + str(pagenum - 1) + ".html"):
text("Previous page")
    text(" ")
    with tag("a", href = filename_base + str(pagenum + 1) + ".html"):
text("Next page")
with tag("a", href = url, target="_blank"):
    text("Original page")

    return doc.getvalue()


def pagegen(filename_base, pagenum, title, desc, company, days, url):
    doc, tag, text = Doc().tagtext()

    doc.asis('<meta charset="UTF-8">')
    with tag("title"):
text(title)
    with tag("body"):
doc.asis(linksgen(filename_base, pagenum, url))
with tag("h1"):
    text(title)
with tag("h2"):
    text(company)
with tag("p"):
    text(str(days) + " days ago")
with tag("div"):
    doc.asis(desc)
```

```
    doc.asis(linksgen(filename_base, pagenum, url))

        return doc.getvalue()
```

ii. test pagegen TEST

```
pagegen("nom", 0, "titre", "desc", "firm", "days", "www")
```

iii. test linksgen TEST

```
linksgen("file", 10, "wwwww")
```

(c) htmlexport function

i. definition

```
def htmlexport(df, begin, end):
    date = str(datetime.now())
    path = "../reports/html/" + date + "/"
    mkdir(path)
    for i in range(begin, end):
filename_base = "job-"
html = pagegen(filename_base,
       i,
       df.title.iloc[i],
       df.desc.iloc[i],
       df.company.iloc[i],
       df.days_ago.iloc[i],
       df.url.iloc[i]
)
filename = path + filename_base +  str(i) + ".html"
with open(filename, "a") as file:
    file.write(html)
```

ii. call

```
htmlexport(df_print, 0, 40)
```

iii. link `home/teddd/data/projects/jobseeker/reports/html/`

1.6.3 server

1. flask ? :D !!!

1.6.4 org table (python) PYTHON

1. john kitchin example TEST

```
import pandas as pd
test = pd.DataFrame({'A': [1000, 1000], 'B' : [60, 100]})
test2 = [list(test)] + [None] + test.values.tolist()
test3 = test.values.tolist()
return (test, test2, test3)
```

2. my program SLOW

```
import pandas as pd
df = pd.read_csv(data)
return  [list(df)] + [None] + df.values.tolist()
```

1.6.5 org results: html TEST

```
dfk.desc.iloc[0]
```

1.6.6 soupprint

1. session functions
   (a) souper (using get text)

```
from bs4 import BeautifulSoup

def souper(html):
    "returns only the text from a html string"
    soup = BeautifulSoup(html, 'html.parser')
    return soup.get_text()
```

   (b) soupprint
      i. definition

```
from bs4 import BeautifulSoup

def souper(html):
    soup = BeautifulSoup(html, 'html.parser')
    print(soup.get_text())

def soupprint(df, begin, end):
    for i in range(begin, end):
print(i, df.title.iloc[i])
print("\n")
print(df.company.iloc[i])
```

```
print("\n")
souper(df.desc.iloc[i])
print("\n"*3)
print("-"*100)
print("\n"*3)
```

ii. call

```
soupprint(df, 0, 10)
```

2. soupprint as org function

(a) definition

```
from bs4 import BeautifulSoup

def souper(html):
    soup = BeautifulSoup(html, 'html.parser')
    print(soup.get_text())

def soupprint(df, begin, end):
    for i in range(begin, end):
print(i, df.title.iloc[i])
print("\n")
print(df.company.iloc[i])
print("\n")
souper(df.desc.iloc[i])
print("\n"*3)
print("-"*100)
print("\n"*3)
```

(b) call

```
soupprint(dfk, 0, dfk.title.count())
```

2 External Documentation

2.1 doc : look for matching patern DOC

```
help(df.title.str.contains)
```

2.2 pandas

Pandas cheat sheet

3 Tests

3.1 ob-ipython

3.1.1 hands-on tryout

1. hello world

   ```
   print 'hello world'
   ```

2. function definition

   ```
   def fn():
       print "I am in the session !"
   ```

3. function call

   ```
   fn()
   ```

3.1.2 doc tutorial

1. imports

   ```
   %matplotlib inline
   import matplotlib.pyplot as plt
   import numpy as np
   ```

2. ex2

   ```
   def foo(x):
       return x + 9

   [foo(x) + 7 for x in range(7)]
   ```

3. images

   (a) ex1

   ```
   plt.hist(np.random.randn(20000), bins=200)
   ```

   (b) ex2

   ```
   plt.hist(np.random.randn(20000), bins=200)
   ```

(c) config

```
%config InlineBackend.figure_format = 'svg'
```

4. other kernel

```
(+ 1 2)
```

5. async

```
import time
time.sleep(3)
plt.hist(np.random.randn(20000), bins=200)
```

3.1.3 other tryouts

1. functions

(a) call

```
lol()
```

2. formater

(a) init

```
import IPython
from tabulate import tabulate

class OrgFormatter(IPython.core.formatters.BaseFormatter):
    def __call__(self, obj):
try:
    return tabulate(obj, headers='keys',
    tablefmt='orgtbl', showindex='always')
except:
    return None

ip = get_ipython()
ip.display_formatter.formatters['text/org'] = OrgFormatter()
```

(b) arrays

3. kernel tests

(a) session header arg after run console

```
print("hello")
```

(b) kernel headerarg

```
print("hello")
```

3.2 nltk

3.2.1 text selection

1. sample text base

```
from nltk.book import *
```

2. access text as string

   (a) imports

```
import nltk, re, pprint
from nltk import word_tokenize
```

   (b) with one description

      i. definition

```
string = df.iloc[0].desc
```

      ii. formating

         A. html

```
string = souper(string)
```

         B. case

```
string = string.lower()
```

         C. punctiations

         D. definition

```
def multi_replace(string, *args, replace=" "):
    for target in args:
string = string.replace(target, replace)
    return string

trash_car = (",", "\'", "\"", "&", "#", "{", "}",
     "(", ")", "[", "]", "_", "\\", "~", "-",
     ",", ";", ":", ".", "?", "!", "+", "|",
     "@", "/", "-", "*", "‘‘", ",,", "%", " ",
     "€")
```

E. call

```
string = multi_replace(string, *trash_car)
```

(c) to ntlk text object

i. tokenizing

```
tokens = word_tokenize(string)
```

ii. use as nltk text

```
text = nltk.Text(tokens)
```

3.2.2 search

1. concordance

```
text.concordance("data")
```

2. similar word

```
text.similar("analyst")
```

3. dispersion

```
text.dispersion_plot(["up", "with", "in", "the", "for", "team"])
```

3.2.3 generation TEST

```
text.generate(["The", "job", "is", "for", "data", "team"])
```

3.2.4 normalizing

1. steaming
2. lemmatization

3.2.5 vocabulary

1. sorted set

```
sorted(set(text))
```

2. lexical richness

(a) tryout

```
len(text) / len(set(text))
```

(b) function

```
def lexical_diversity(text):
    return len(text) / len(set(text))
```

3. specific word

(a) tryout

```
100 * text.count('for') / len(text)
```

(b) functyion

```
def word_percentage(word):
    return 100 * text.count(word) / len(text)
```

3.2.6 TODO Build a corpus !

1. sklearn

```
docs = df['desc']

tfs = tfidf.fit_transform(docs)
```