Get and Clean data(15-23)

```
def get_and_clean_data():

    data = pd.read_csv('resource/software_developer_united_states_1971_20191023_1.csv')

    description = data['job_description']

    cleaned_description = description.apply(lambda s: s.translate(str.maketrans('', '',
string.punctuation + u'\xa0')))

    cleaned_description = cleaned_description.apply(lambda s: s.lower())

    cleaned_description = cleaned_description.apply(lambda s:
s.translate(str.maketrans(string.whitespace, ' ' * len(string.whitespace), '')))

    cleaned_description = cleaned_description.drop_duplicates()

    return cleaned_description
```

`def get_and_clean_data():` is designed to read data from `resource/software… .csv` and clean data by removing punctuation, converting to lowercase, removing whitespace, and removing duplicate entries, respectively. Then return `cleaned_description`

Breaking down text(15-23)

```
def simple_tokenize(data):

    cleaned_description = data(lambda s: [x.strip() for x in s.split()])

    return cleaned_description
```

`def simple_tokenize(data):` is a function that splits the text into a list of tokens by using `split()` Then return `cleaned_description`

Combines functions(15-23)

```
def parse_job_description():
    cleaned_description = get_and_clean_data()
    cleaned_description = simple_tokenize(cleaned_description)
    return cleaned_description
```

`def parse_job_description():` is a function that combine the function that cleaning and tokenisation for job description. Then return `cleaned_description`

Count word(15-23)

```
def count_python_mysql():
    parsed_description = parse_job_description()
    count_python = parsed_description.apply(lambda s: 'python' in s).sum()
    count_mysql = parsed_description.apply(lambda s: 'mysql' in s).sum()
    print('python: ' + str(count_python) + ' of '+str(parsed_description.shape[0]))
    print('mysql: ' + str(count_mysql) + ' of ' + str(parsed_description.shape[0]))
```

`def count_python_mysql():` is a function that count word “python” and “mysql” in the list of tokens within `parsed_description` by using lambda function that return ‘true’ if respective word is present then counts the number of ‘true’ and print.

Parse database(15-23)

```
def parse_db():
    html_doc = requests.get("https://db-engines.com/en/ranking").content
    soup = BeautifulSoup(html_doc, 'html.parser')
    db_table = soup.find("table", {"class": "dbi"})
    all_db = [''.join(s.find('a').findAll(text=True, recursive=False)).strip() for s in
db_table.findAll("th", {"class":"pad-l"})]
    all_db = list(dict.fromkeys(all_db))
    db_list = all_db[:10]
    db_list = [s.lower() for s in db_list]
    db_list = [[x.strip() for x in s.split()] for s in db_list]
    return db_list
```

`def parse_db():` is a function that is using content from `https://db-engines.com/en/ranking"` and find data by using `soup.find()` then extract text from each `<th>` then select top 10 in database, convert to lower case and tokenise into units by using `split()` and return result.

Parse database(15-23)

```
cleaned_db = parse_db()
parsed_description = parse_job_description()
raw = [None] * len(cleaned_db)
for i,db in enumerate(cleaned_db):
    raw[i] = parsed_description.apply(lambda s: np.all([x in s for x in db])).sum()
    print(' '.join(db) + ': '+str(raw[i]) + ' of ' + str(parsed_description.shape[0]))
```

In this source code is a comparing each database name that is obtained from `parse_db` and `parse_job_description`

Then print the count of job description that contain all tokens of each databases.

Output

```
oracle: 1392 of 7583

mysql: 667 of 7583

microsoft sql server: 868 of 7583

postgresql: 261 of 7583

mongodb: 296 of 7583

redis: 106 of 7583

elasticsearch: 161 of 7583

ibm db2: 48 of 7583

sqlite: 28 of 7583

microsoft access: 256 of 7583
```

Parse database + “python”(15-23)

```
with_python = [None] * len(cleaned_db)

for i,db in enumerate(cleaned_db):

    with_python[i] = parsed_description.apply(lambda s: np.all([x in s for x in db]) and 'python' in
s).sum()

    print(' '.join(db) + ' + python: ' + str(with_python[i]) + ' of ' +
str(parsed_description.shape[0]))
```

In this source code is similar with previous one, but in this source code is checks for job description that contain all tokens in ‘db’ `np.all` and include the word ‘python’ `'python' in s`

Output

```
oracle + python: 243 of 7583

mysql + python: 207 of 7583

microsoft sql server + python: 51 of 7583

postgresql + python: 90 of 7583

mongodb + python: 111 of 7583

redis + python: 38 of 7583

elasticsearch + python: 73 of 7583

ibm db2 + python: 12 of 7583

sqlite + python: 7 of 7583

microsoft access + python: 28 of 7583
```

Parse database + “python” with in percentage(15-23)

```
for i, db in enumerate(cleaned_db):

    print(' '.join(db) + ' + python: ' + str(with_python[i]) + ' of ' + str(raw[i]) + ' (' +
str(np.around(with_python[i] / raw[i]*100,2))+ '%)')
```

In this source code is print the percentage of job descriptions that contain all tokens of given database and include ‘python’ relative to the oral number of job description containing all tokens of that database.

Output

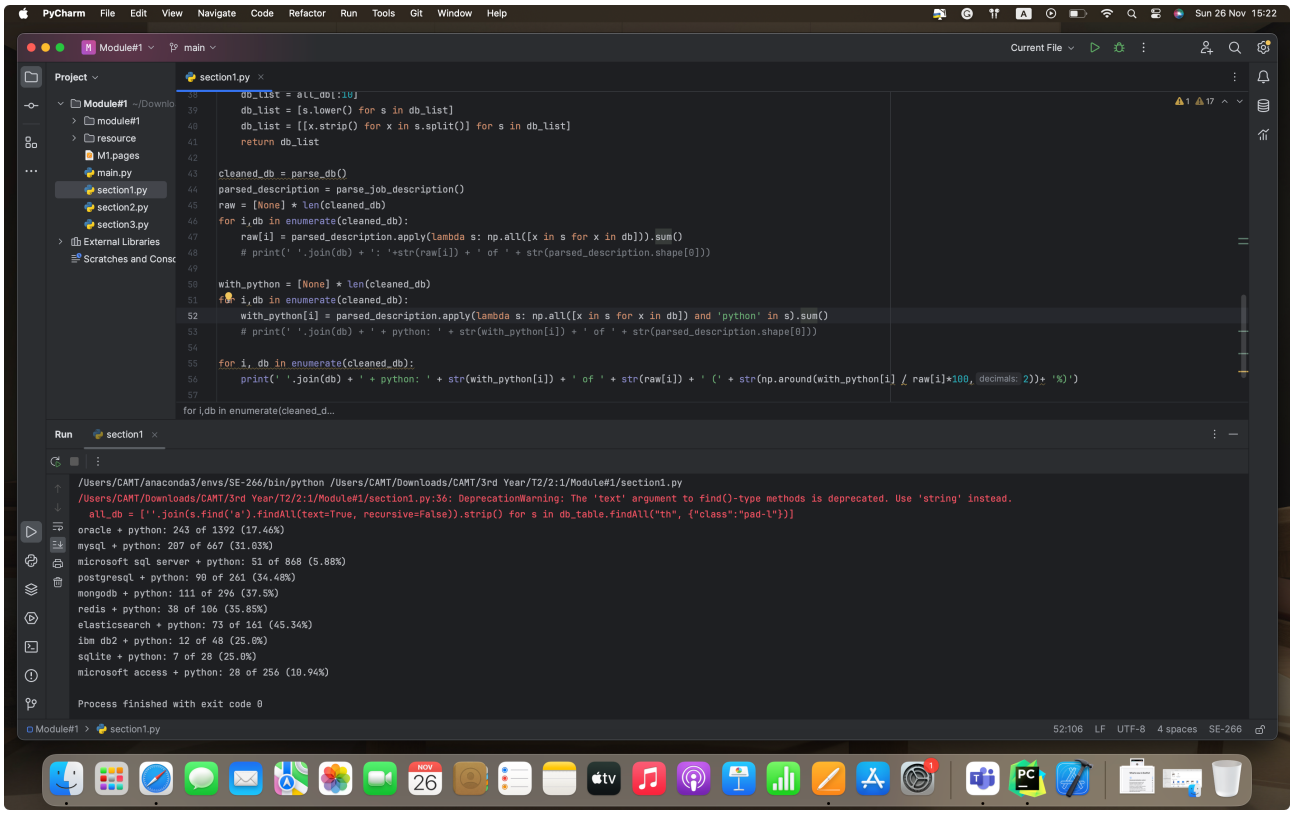
```
oracle + python: 243 of 1392 (17.46%)

mysql + python: 207 of 667 (31.03%)

microsoft sql server + python: 51 of 868 (5.88%)

postgresql + python: 90 of 261 (34.48%)

mongodb + python: 111 of 296 (37.5%)

redis + python: 38 of 106 (35.85%)

elasticsearch + python: 73 of 161 (45.34%)

ibm db2 + python: 12 of 48 (25.0%)

sqlite + python: 7 of 28 (25.0%)

microsoft access + python: 28 of 256 (10.94%)
```

MODULE 1

15-23 25-26 35-36

Source code

266-953881-701000 intro to info retrieve



Created by 642115004 Kitdanai Palee

Mapping(25-26)

```
lang = [['java'], ['python'], ['c'], ['kotlin'], ['swift'], ['rust'], ['ruby'], ['scala'],
['julia'], ['lua']]

parsed_description = parse_job_description()

parsed_db = parse_db()

all_terms = lang + parsed_db

query_map = pd.DataFrame(parsed_description.apply(lambda s: [1 if np.all([d in s for d in db])

                                                    else 0 for db in
all_terms]).values.tolist(), columns=[' '.join(d) for d in all_terms])
```

In this source code is creating a binary mapping that indicates programming languages that contain in each job description.

Output

	java	python	c	kotlin	swift	rust	ruby	scala	julia	lua	oracle	mysql	microsoft sql server	postgresql	mongodb	r
0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	
1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
4	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	

```
java_terms = query_map[query_map['java'] > 0].apply(lambda s: np.where(s==1)[0], axis=1).apply(lambda
s: list(query_map.columns[s]))
```

This line of code is a filters rows where `'java'` is mentioned.

Output

```
10      [java, python, c, oracle, mysql, mongodb]
11                                         [java]
12                            [java, swift, redis]
16                      [java, c, swift, oracle]
19                               [java, python]
                          ...
7576                               [java, oracle]
7578               [java, python, oracle, mysql]
7579                               [java, oracle]
7581                         [java, oracle, mysql]
7582                              [java, mongodb]
Length: 3268, dtype: object
```

Document to be indexed(35-36)

```
import numpy as np

str1 = 'the chosen software developer will be part of a larger engineering team developing software
for medical devices.'

str2 = 'we are seeking a seasoned software developer with strong analytical and technical skills to
join our public sector technology consulting team.'

import nltk

nltk.download('stopwords')

nltk.download('punkt')

from nltk.corpus import stopwords

from nltk.tokenize import word_tokenize

from nltk.stem import PorterStemmer


tokened_str1 = word_tokenize(str1)

tokened_str2 = word_tokenize(str2)


tokened_str1 = [w for w in tokened_str1 if len(w) > 2]

tokened_str2 = [w for w in tokened_str2 if len(w) > 2]


no_sw_str1 = [word for word in tokened_str1 if not word in stopwords.words()]

no_sw_str2 = [word for word in tokened_str2 if not word in stopwords.words()]


ps = PorterStemmer()

stemmed_str1 = np.unique([ps.stem(w) for w in no_sw_str1])

stemmed_str2 = np.unique([ps.stem(w) for w in no_sw_str2])


full_list = np.sort(np.concatenate((stemmed_str1, stemmed_str2)))


print('In Document 1: ' + str(stemmed_str1))

print('============================')

print('In Document 2: ' + str(stemmed_str2))

print('============================')

print(full_list)
```

Created by 642115004 Kitdanai Palee

This sourse code is performs tokenisation for ‘str1’ and ‘str2’, then filters out words with length less than or equal to 2 characters `[w for w in tokened_str if len(w) > 2]`, then stop word removal by removing common stop word. (For example, “and” or “the”), then stemming by using Porter Stemmer to reduce to collapse words with similar meanings into a common base form.(for example, “developer” and “developing” may stemmed to “develop”) and combine and sort `np.sort(np.concatenate((stemmed_str1, stemmed_str2)))`

Output

```
In Document 1: ['chosen' 'develop' 'devic' 'engin' 'larger' 'medic' 'part' 'softwar'
 'team']
============================
In Document 2: ['analyt' 'consult' 'develop' 'join' 'public' 'season' 'sector' 'seek'
 'skill' 'softwar' 'strong' 'team' 'technic' 'technolog']
============================
['analyt' 'chosen' 'consult' 'develop' 'develop' 'devic' 'engin' 'join'
 'larger' 'medic' 'part' 'public' 'season' 'sector' 'seek' 'skill'
 'softwar' 'softwar' 'strong' 'team' 'team' 'technic' 'technolog']
```