Practice Final Exam for Programming Tools for the Data Scientist

Name: Hongyi Zheng

Net ID hz2212

Instructions

There are two sections of this test. The first section consists of 10 multiple choice questions, worth 5 points each, for a total of 50 points points. The second section consists of two longer questions worth 25 points each, for a total of 50 points. Therefore, a perfect score on the test would be 100 points.

At the end of this file there are two blank pages, should you need additional space to add answers (see below), although perhaps this is silly for an online test, where you can always add pages anyway. There is also a glossary defining Python and terminal commands that we used in this course and are likely to be on the test. As per the instructions, you are permitted to consult this glossary during the exam.

This test booklet is a .pdf file. You should prepare a .pdf file with your answers to submit to Gradescope in the permitted time period. This .pdf can take one of the following forms:

- If you are able to edit .pdf files, you can revise this test booklet and submit that to GradeScope.
- You can print out the test, fill in your answers in pen or pencil (please make the writing dark enough so it is legible). Then you can scan this file and create a .pdf file. You can submit this. Very clear pictures taken with a cell phone camera are acceptable. However, poor quality photographs that the grader cannot read are not acceptable.
- You can create a .pdf file with a Word processor that contains all the answers in sequence, numbered as appropriate. For example, Microsoft Word has an option to "save as pdf" or "export to pdf" or something like that.

This test has the following time restrictions:

- This test must be completed less than 24 hours after it is released.
- After downloading the test, you have 1 hour and 50 minutes to complete the test. In addition, there is a 30 minute window in case you have technical difficulties submitting the test.
- The above 1 hour and 50 minute window will be modified as a appropriate, if you submitted an accommodation letter through NYU's Moses Center.
- There will be 2 places to submit the final exam on GradeScope.
  - If you submit your test within the above described timeline, you should submit it to Gradescope under **Final Exam – On-Time**.
  - If it is late for any reason, you should submit it to Gradescope under **Final Exam – Late**. If you submit it in this slot, you should email me an explanation.

This test is **partially** open book. The following are permitted:

- You can look at slides from class during the test.
- You can look at the glossary provided as part of the test. This will be the same glossary as is provided with the practice test.
- You can look at UNIX "man" pages, e.g, type "man ls" in Linux to find out about the "ls" command.
- You can use a simple calculator.

- For Part 2 questions, you can test your code in IDLE or jupyter or other standard IDE for Python. For shell scripts, you can test in a linux shell.

Since the test is partially open book, it is in part **closed** book as well. The following are not permitted:

- Do not communicate with others.
- Do not do websearches.
- Do not look at any materials not specifically listed above as being OK. For example, do not look at your notes.

Please (digitally or manually) sign in any way that you can the following statement:
I agree to complete this test, independently, in the appropriate amount of time as per the instructions:

____

Answer all questions on the test. There will be opportunities for partial credit on questions.

Section 1

Answer all 10 multiple choice questions. As per each question, indicate which answers are correct. You can circle the right answers with a pen, or put a dark X or asterisk next to the correct answers. For individual questions, it may be appropriate to either choose a single answer or to choose multiple answers. For example, in sample question **i**, there is a single correct answer, but for sample question **ii**, the correct answer entails choosing more than one of the possibilities enumerated as **a** through **e**. There also may be a different number of options for each question. The correct answer for the following sample questions are indicated with blue Xs.

Sample Question I. Which of the following is the name of a month in the standard Gregorian Calendar?

(a) March **X**

(b) Earth Day

(c) Wednesday

(d) Noon

(e) Springtime

Sample Question II. Which of the following are names months in the standard Gregorian Calendar? Choose all that apply.

(a) March **X**

(b) December **X**

(c) Wednesday

(d) January **X**

(e) Springtime

Part I Questions 1–10

1. In an executable shell script called **smoosh.sh**, $1 should refer to:

   (a) One dollar

   (b) [✓ A variable bound by the string following **smoosh.sh** on the command line.]

   (c) The end of line character

   (d) The environmental PATH variable

2. **sys.argv** is a variable from the **sys** package. In a Python script, **sys.argv** represents which of the following:

   (a) [✓ A list consisting of the name of the script file and the command line arguments.]

   (b) An environmental variable initialized to the system's IP address.

   (c) A list of the files in your current working directory.

   (d) An open port

3. In which of the following Python code snippets, is **every** instance of the string **VARIABLE1** a global variable? Mark all that apply. If only some instances of **VARIABLE1** are global in a particular snippet, don't mark it.

   [✓ (a)]

   ```
   VARIABLE1 = 100
   print(VARIABLE1 * 100)
   ```

   [✓ (b)]

   ```
   VARIABLE1 = 100
   def add_100():
       global VARIABLE1
       VARIABLE1 =  VARIABLE1 + 100
   ```

   (c)

   ```
   VARIABLE1 = 100
   def do_stuff(VARIABLE1):
       VARIABLE1 =  VARIABLE1 + 100
       return(VARIABLE1)
   do_stuff(100)
   ```

   (d)

   ```
   VARIABLE1 = 100
   def do_stuff():
       VARIABLE1 = int(input('Enter an integer: '))
   do_stuff()
   ```

4. Which strings are matched by the regular expression `[01]?[0-9]:[0-6][0-9] ?(AM|PM)?` (Note that there is one space before `?(AM|PM)`})? There are no other spaces in the expression. Mark all choices from a to d that apply.

   (a) 29:69 PM

   (b) 39:59

   (c) [✓ 5:05 AM]

   (d) [✓ 15:05 PM]

5. The first 3 lines of a **.csv** file are:

```
Brand,Color,Number_Holes,shape
Lego,Red,4,rectangle
Lockblocks,Blue,4,square
```

Which of the following statements are true? Mark all that apply.

(a) [✓ The first line represents column labels.]

(b) [✓ The second and third lines each represent 4 feature value pairs. The column labels (first line) determine the feature names.]

(c) The two rows represent items with the same value for the feature **shape**.

(d) [✓ The two rows represent items with the same value for the feature **Number_Holes**.]

6. Starbucks is a coffee store that sells the following sizes of coffee: ”short”, ”tall”, ”grande”, ”venti”, and ”trenta”, listed in order from smallest to largest. The ”short” and ”trenta” sizes are only used by a subset of Starbucks stores. In (United States) fluid ounces, these sizes correspond to 8 ounces, 12 ounces, 16 ounces, 20 ounces and 24 ounces respectively. Thus there is a standard 4 ounces between sizes. Does this system represent a nominal scale, an ordinal scale, an interval scale, or a ratio scale?

(a) nominal

(b) [✓ ordinal]

(c) interval

(d) ratio

7. If a raster graphics pixel consists of the numbers: 0, 0, 0, which color is it? Choose the most informative (best) answer.

(a) [✓ black]

(b) white

(c) grey

(d) either black or white

(e) yellow

8. Which of the following phrase structure trees best represents the meaning of the sentence:
Bird watchers know that telescopes are useful for the observation of birds.

(a) [✓ Figure A]

(b) Figure B

(c) Figure C

(d) Figure D

Figure A

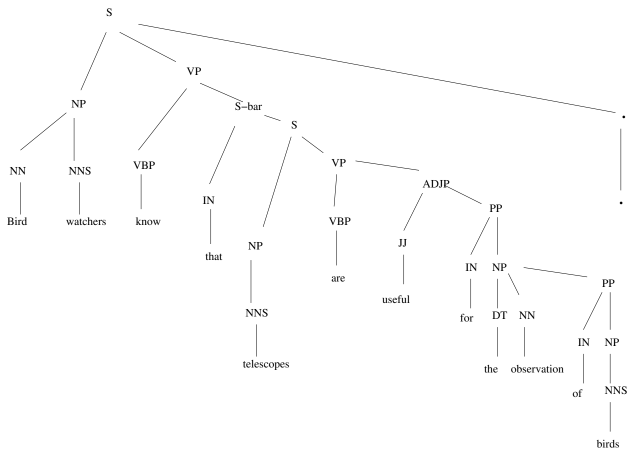


Figure B

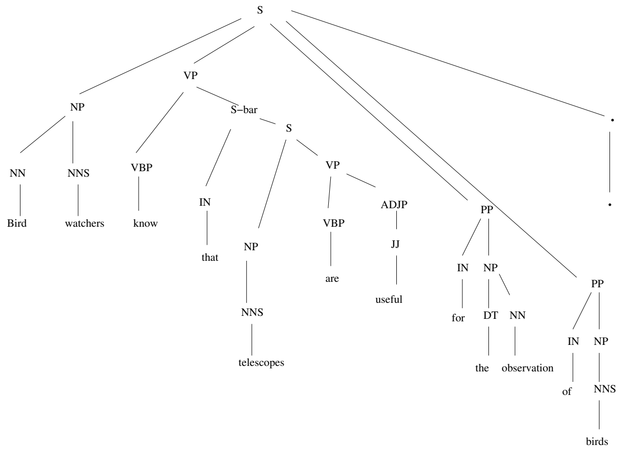
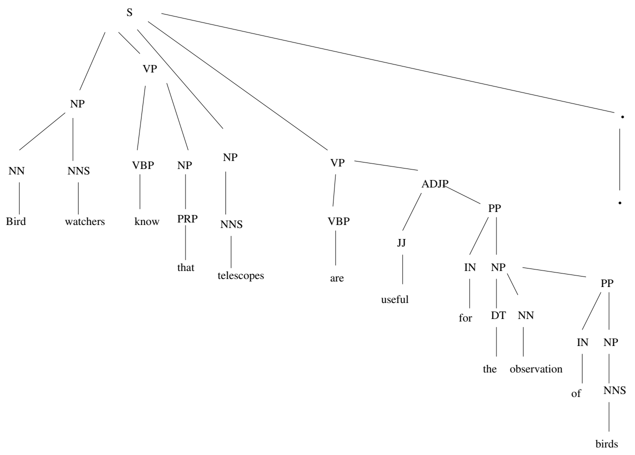


Figure C


S
VP
NP
.
NN
NNS
VBP
NP
NP
VP
ADJP
PRP
PP
Bird
watchers
know
NNS
VBP
JJ
IN
NP
that
telescopes
are
useful
PP
for
DT
NN
IN
NP
the
observation
of
NNS
birds
.

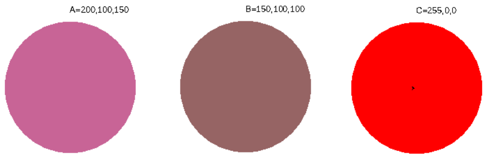
9. The vectors $A = [200, 100, 150]$, $B = [150, 100, 100]$ and $C = [255, 0, 0]$ represent shades of purple, brown and red, as per the image below. Which of the following is true (all numbers are rounded to 2 significant digits).

   (a) The cosine similarity of A and B is .014. The cosine similarity of A and C is .010. The cosine similarity of B and C is .012.

   (b) The cosine similarity of A and B is 6.3. The cosine similarity of A and C is .8.0. The cosine similarity of B and C is 7.7.

   (c) The cosine similarity of A and B is .48. The cosine similarity of A and C is .37. The cosine similarity of B and C is .36.

   [✓ (d) The cosine similarity of A and B is .99. The cosine similarity of A and C is .74. The cosine similarity of B and C is .73.]



10. Calculate the precision, recall and f-measure, given the answer key and system output in the table below. Match your answer to one of the **a-d** choices. Each object is classified either as a **DUCK**, a **GOOSE** or it is not classified at all. An answer is correct if the **DUCK** or **GOOSE** system label matches the answer key. As you know, Precision and Recall have the same numerator (number of correct labels), but different denominators (number of labels in either the system output or the answer key). F-measure is derived from precision and recall.

    (a) Precision 0.71 Recall 0.31 F-measure 0.43

    (b) Precision = 0.71 Recall = 0.31 F-measure = 0.51

    [✓ (c) Precision 0.31 Recall 0.71 F-measure 0.43]

    (d) Precision = 0.31 Recall = 0.71 F-measure = 0.51

Item Number	**System Output**	**Answer Key**
1	Duck	Duck
2	Duck	Duck
3	Duck	
4	Duck	Goose
5	Goose	Goose
6	Goose	Goose
7	Duck	Duck
8	Goose	
9	Duck	
10	Goose	
11		Duck
12	Duck	
13	Duck	
14	Duck	
15	Duck	
16	Duck	
17	Duck	

Section 2

Answer both of the following two questions. Depending on the question, the answer can be a shell script, a Python program, a regular expression or some other programmatic solution to some problem. You can test the program on a linux/Apple/Windows terminal or in an IDE like Idle or jupyter. However, it does not need to be a fully debugged program to get a good score. Remember if your program only does some of these things or if it is correct in principle, but not debugged 100% you will get partial credit.

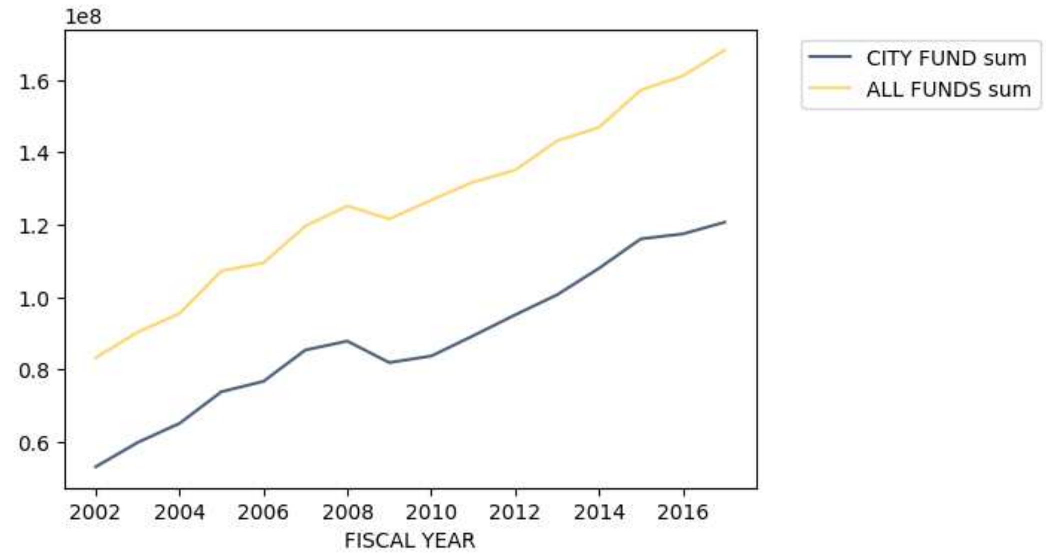
Question 11. Write a Python function that takes a directory as input and returns a dictionary, with words as keys and IDF scores as values. Your program should read in each .txt file in that directory, find the words in the file by splitting each line by spaces. The program should create a dictionary with all the words encountered at least one time as keys. The values of these keys should be the inverse document frequency (IDF) of that word, with respect to that collection of files. For example, if there are 500 text files and all but one of them contains the word *the*, then the IDF for that word would be log(500/499) or about .002. To calculate the (natural) log, I suggest using **math.log(number)**. In summary, you will have to count the number of **.txt** files are in the directory, separate each file into words and use those words to count the number of files that contain each word.

Hint 1: I suggest that you first collect the number of files that each word occurs in and then convert these numbers to IDFs, based on the number of files.

Hint 2: The function *os.listdir(PATH)* returns the list of files in PATH.

Question 12. In "the DataScience Tools/Final" directory, download the file **final12_setup.ipynb**. This downloads a table of information about funding for city agencies. For this exercise only use (.select) the columns corresponding to the labels: 'FISCAL YEAR', 'ALL FUNDS' and 'CITY FUND'. Sort this table by 'FISCAL YEAR'. Then group by 'FISCAL YEAR', using **sum**, so that the sum of the amounts for each row is provided. Then use **.plot** to plot the trends of finances by fiscal year. The result should look like the graph below (the title is not necessary). Then convert this to a **.py** file, just like homework 9.

Financial Trends in the City Budget



The screenshots of the code for Q11,12 are attached to the end of the pdf file.

Glossary of Terms

1. Python’s **os** module includes global variables like *os.linesep* (end of line strings: ’$\backslash n$’ or ’$\backslash r\backslash n$’) and *os.sep* (path separators – forward slash ’/’ or backward slash ’\’). The os module also includes functions that interact with the operating system. *os.getcwd()* returns the current working directory. *os.listdir(PATH)* returns a list of files in PATH; *os.path.isdir(PATH)* returns True if PATH is a directory and False otherwise; *os.path.isfile(PATH)* returns True if PATH is the name of an existing file and False otherwise.

2. File Object Streams – Python objects used for reading files and writing to files.

   - *instream = open(’my_file.txt’,’r’)* sets the variable *instream* to the contents of the file *’my_file.txt’*. *for* loops will treat *instream* as a list of strings, each ending with *os.linesep*. For most applications, it makes sense to remove these.
   - *outstream = open(’my_file.txt’,’w’)* sets the variable outstream to an object that will ultimately be saved as the file *my_file.txt*. The method *outstream.write(string)* will write a string to that file. It is a good idea to include $\backslash n$ anywhere you would like a line break in the file as end of lines are not automatic. $\backslash n$ should be used, rather than os.linesep, even in Windows.
   - *stream.close()* will close an opened stream. This ends the connection between Python and a file. In the case of output streams (like *outstream*), the content of the stream is written to the file.
   - *with open(file,’r’) as instream:* or *with open(file,’w’) as outstream:* starts a block in which a stream is opened. The body of code indented under these statements can read from or write to the stream. After the block ends, the stream is closed.

3. **requests.get(url)** is part of the **requests** library. It produces a input stream containing the content of the url (web address). **requests.get(url).text** (the .text value of that stream object) is the html text from that website. It is possible to obtain text between <p> and </p> or paragraphs and remove all html from that text using other filters. For the class exercises, I wrote such filters using regular expressions and Pythons **re** package.

4. **urllib.request.urlopen** is a command in the **urllib.request** library. It is for obtaining text from websites that we used in conjunction with **BeautifulSoup**.

5. **bs4.BeautifulSoup** (or BeautifulSoup.BeautifulSoup) is a html (and xml) parser that takes two arguments: the input stream produced by **urllib.request.urlopen** ; and ’lxml’ is the name of a library that BeautifulSoup uses to process the html. The soup variable in: **soup = bs4.BeautifulSoup(input_stream)** is an object that contains the website in a sorted form, so it is possible to lookup html fields. We used it to find all the paragraphs, by means of the command **paragraphs = soup.find_all(’p’)**. This

produced a list of paragraph structures and assigns them to the variable **paragraphs**. The text from each paragraph is accessible using **.text**, e.g., **paragraphs[0].text** refers to the text of the first paragraph.

6. **Python re package** includes a variety of ways to use regular expressions. The most basic use is with the command **re.search(REGEXP,TEXT_TO_SEARCH)**. This produces a **re.Match object**. For example, the command
   abc = re.search(’(ABC)+’,’123ABCABCABC123’)
   returns a regexp object representing the substring **ABCABCABC**. The object has several values including abc.group(0) (the whole match) and abc.group(1) (the first instance of ABC). The group number N after 0 refers to a piece of the matching regep, matching the part of the regexp starting at the Nth parenthesis. So the 1st parenthesis is around ’(ABC) and thus, it corresponds to the first match of the string “ABC”. In addition to **MATCH.group(number)**, there is also **MATCH.start(N)** and **MATCH.end(N)**, referring to the beginning and end of the block of text matching group N. In addition to re.search, other re functions include **re.findall**, **re.finditer** and **re.sub**. **re.findall** and **re.finditer** identify multiple instances of a regexp inside a string. **re.sub** substitutes a regexp with a replacement string.

7. Python **csv** package is a package for processing comma separated value files (and tab separated value files). **csv.reader(instream)** returns a list of lists from a **csv** file. Unlike **string.split(’,’)**, it accounts for more complex **csv** structure that is used for spreadsheets, e.g., where columns are surrounded by quotes and a field can contain a comma. There are other commands such as **csv.writer.writerow** that we did not cover in class.

8. Python **sys.exit** and **sys.argv**. **sys.exit** exits Python. **sys.argv** is a list of arguments (strings) taken from command line use of Python. sys.argv[0] is the name of the Python File you are executing and the remaining items in the list are the command line parameters. For example, if you executed the command “python3 do_stuff.py 57 100, sys.argv would be equal to [’do_stuff.py’, ’57’, ’100’].

9. Linux shell commands:

   - **ls** lists the files in a directory. There are numerous flags (e.g., -l) that you can use to provide additional detail or present the results in different ways.
   - **cp**, **mkdir**, **ln -s** are for copying files, making directories and making symbolic links.
   - **chmod** changes the file permissions on a file or directory.
   - **ˆC, kill, fg, bg, ˆZ, &, top, ps, free** are commands that have to do with manipulating jobs (computer processes) or finding out information about jobs.
   - command for printing things to the screen or printing parts of files: echo, cat, grep, cut, sort, uniq

- > and | are operators used in shell commands. | directs the output of the process on its left to be input for the process on its right. > sends the output of the process on its left to the filename on its write, creating a file in the process.
- Shell scripts are executable files that contain shell commands. Very simple shell scripts can be created which are just sequences of shell commands. In addition, command line parameters can be referred to. $1 is the first parameter on the command line; $2 is the second argument and so on.

10. Git commands: git push, git pull, git add, git commit, git clone

11. Short guide to regular expressions

    (a) Repeat operators: *, +, {number}, {minimum,maximum}

    (b) Optional and disjunction: ? and |

    (c) Parentheses () mark scope of repeat, optional and disjunction operators

    (d) Beginning and End of line: ˆand $

    (e) Disjunction (and ranges) for single characters: [a-z], [A-Z], [0-9], [a-zA-Z,0-9], [123abc]

    (f) Negation for single characters [ˆa-z]

    (g) The period . represents any character, e.g., ".*" represents any string. Thus the expression ' ˆA.*Z$' matches any line that starts with A and ends with Z.

12. Short guide to Jupyter Notebooks

    (a) Converting a notebook into a .py file from the command line:
    jupyter nbconvert –to script myfile.ipynb
    creates **myfile.py**
    Next, remove lines in **myfile.py** mentioning **get_ipython** and unnecessary comments.
    Finally, add python command plt.savefig('outfile.png', bbox_inches='tight') after each graph to save a .png file recording that graph.

    (b) Creating modified versions of tables

        i. **.relabeled** – changes the name of column labels

        ii. **.select** (only keep these labels) and **.drop** (drop these labels)

        iii. **.where** in combination with **.** **are.below** (Or ds.are.below), **are.above** and **are.equal** – selects rows with a particular feature constrained to be below, above or equal to a particular value.

    (c) **table.scatter(X)**, **table.plot(X)**, **table.barh(X)**: Compares X against all other column labels. These produce either a scatter plot (scatter), a regression line (plot) or a bar graph (barh).

(d) **.group** – Takes one or more column labels as arguments, plus an optional function. It will group together items with the same labels based on values. The first argument can be a single column label (’Flavor’) or a list of such labels ([’Flavor’,’Price’]). By default it will produce counts, e.g., the ’flavor’, ’vanilla’ occurs 3 times; or that the ’flavor’ ’vanilla’ and ’quality’ ’bad’ occur in combination 2 times. If you provide a function name, e.g., sum, it will apply that function to the values.

13. Operations and Methods for Numpy arrays

    (a) **+** – Adding arrays, numbers,

    (b) ***** – Multiplying numbers with other objects; Multiplying arrays with other objects, component-wise product for same dimension matrices

    (c) **matmul** – matrix multiplication.

14. **Penn Treebank POS tags:**

Tag	Description	Tag	Description
CC	Coordinating conjunction	RB	Adverb
CD	Cardinal number	RBR	Adverb, comparative
DT	Determiner	RBS	Adverb, superlative
EX	Existential there	RP	Particle
FW	Foreign word	SYM	Symbol
IN	Preposition or subordinating conjunction	TO	to
JJ	Adjective	UH	Interjection
JJR	Adjective, comparative	VB	Verb, base form
JJS	Adjective, superlative	VBD	Verb, past tense
LS	List item marker	VBG	Verb, gerund or present participle
MD	Modal	VBN	Verb, past participle
NN	Noun, singular or mass	VBP	Verb, non-3rd person singular present
NNS	Noun, plural	VBZ	Verb, 3rd person singular present
NNP	Proper noun, singular	WDT	Wh-determiner
NNPS	Proper noun, plural	WP	Wh-pronoun
PDT	Predeterminer	WP$	Possessive wh-pronoun
POS	Possessive ending	WRB	Wh-adverb
PRP	Personal pronoun	.#$``"(),:	Punctuation
PRP$	Possessive pronoun		

15. Linguistic Phrase Categories

    (a) Sbar is a unit larger than a sentence, e.g., *that he knows how to fly* is an S-bar consisting of the word *that* and the sentence *he knows how to fly*.

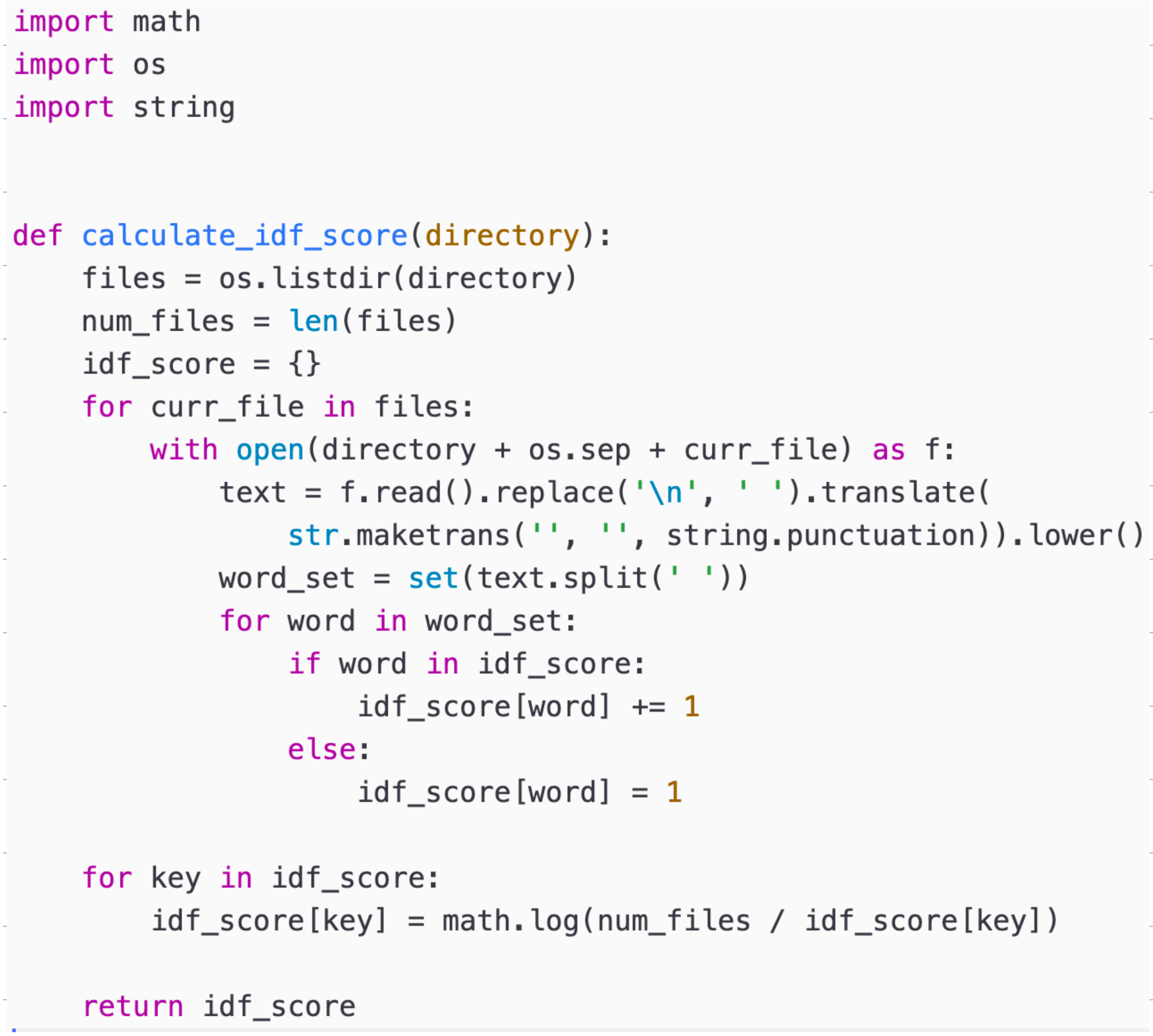
    (b) NP is a noun phrase: *the book about cheese*, *Mary Smith*

    (c) VP is a verb phrase: *knows how to fly*, *laughs at movies*

    (d) PP is a prepostion phrase: *on the roof*, *in the dark*, *at school*.

    (e) ADJP is an adjective phrase: *angry at John*, *very small*, *colorful*.

Question 11



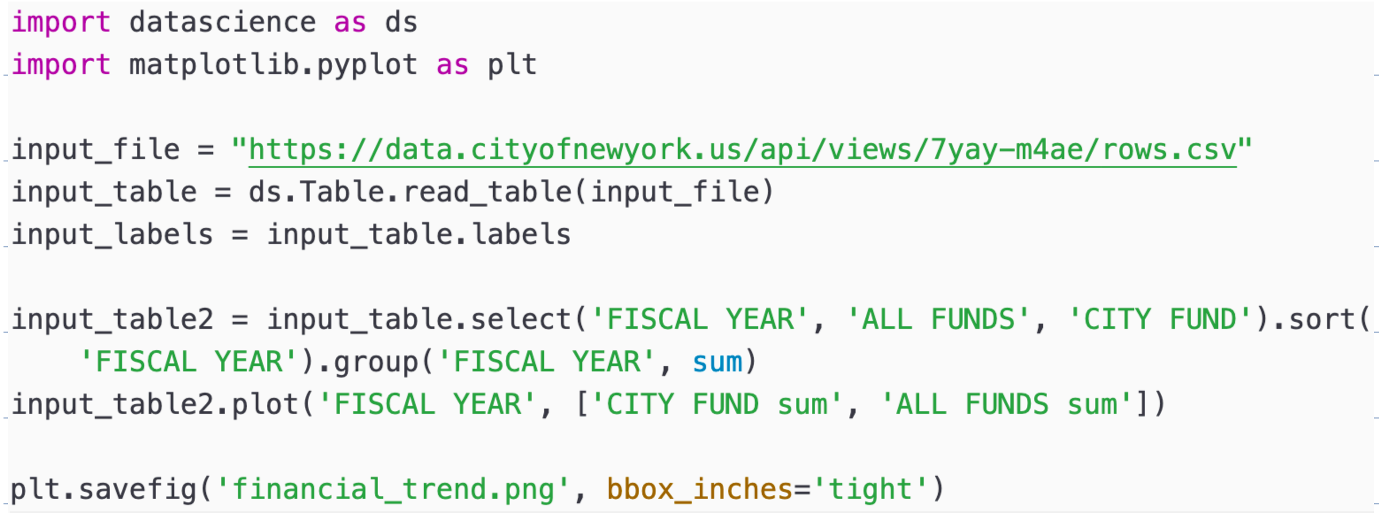
```python
import math
import os
import string


def calculate_idf_score(directory):
    files = os.listdir(directory)
    num_files = len(files)
    idf_score = {}
    for curr_file in files:
        with open(directory + os.sep + curr_file) as f:
            text = f.read().replace('\n', ' ').translate(
                str.maketrans('', '', string.punctuation)).lower()
            word_set = set(text.split(' '))
            for word in word_set:
                if word in idf_score:
                    idf_score[word] += 1
                else:
                    idf_score[word] = 1

    for key in idf_score:
        idf_score[key] = math.log(num_files / idf_score[key])

    return idf_score
```

Question 12



```python
import datascience as ds
import matplotlib.pyplot as plt

input_file = "https://data.cityofnewyork.us/api/views/7yay-m4ae/rows.csv"
input_table = ds.Table.read_table(input_file)
input_labels = input_table.labels

input_table2 = input_table.select('FISCAL YEAR', 'ALL FUNDS', 'CITY FUND').sort(
    'FISCAL YEAR').group('FISCAL YEAR', sum)
input_table2.plot('FISCAL YEAR', ['CITY FUND sum', 'ALL FUNDS sum'])

plt.savefig('financial_trend.png', bbox_inches='tight')
```