Practice Final Exam for Programming Tools for the Data Scientist

Name: _____
Net ID _____

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. If you erase a symbolic link, which of the following things happen:

   (a) Both the link and the original file are deleted.

   (b) The link is deleted, but the original file is unchanged.

   (c) You create an anti-link, a name that is specifically not linked to any file.

   (d) Nothing is deleted. All symbolic links and actual files remain in tact.

2. As discussed in class, a Python **stream** has which of the following properties (mark all that apply):

   (a) A stream contains a block of sequential data such as the contents of a current or future file, the text of an email or the content retrieved from a URL.

   (b) A stream includes a pointer to a position in the data it contains so subsequent read and write statements will start at that position.

   (c) Some streams represent input being read, whereas others represent output (e.g., a file) that your program is writing to.

   (d) Each piece of data in a stream is called an "aquatic: object.

3. What 3 number combination should XXX stand for in the command "chmod XXX filename", if you want for both you and your group to be able to read, write or modify the file, but everyone else on the system to be able to read it only. You do not want the file to be executable.

   (a) 661

   (b) 646

   (c) 166

   (d) 664

4. I through IV represent file paths. Assuming that Path I is a symbolic link to Path II, which statement about paths III and IV **must** be True? Choose only one answer.

   (I) /home/user/Lego/4_holes

   (II) /home/user/block/4_holes/Lego

   (III) /home/user/Lego/4_holes/rectangular/transparent/block_153.png

   (IV) /home/user/block/4_holes/Lego/rectangular/transparent/block_153.png

   (a) Paths III and IV point to the same file.

   (b) Paths III and IV are the same length, as measured by the number of forward slashes (/).

   (c) **Lego** and **block** represent users in the same group.

   (d) The paths represent an ontology of lego blocks.

5. Consider the Python command ”line.split(’,’)”, where ”line” is a variable referring to a line in a file, after the newline character is removed from the end. If this line is from a simple .csv file, .split will correctly split the line (row) into a list of fields (columns). If this is a complex .csv file, it may not work correctly. Why?

   (a) The line may be too long.

   (b) A field in the line can contain a comma, provide the beginning and end of a field is marked in some other way, typically with double-quotes (“). These lines will throw off the simple strategy because some commas will be inside fields.

   (c) Commas may be incorrectly recognized due to incompatibilities with the Microsoft Windows Operating System.

   (d) There can be more than one kind of commas in a UTF-8 encoding.

6. The colors labeling the values used in the representation of pixels of a raster graphics files are:

   (a) red, green and blue

   (b) red, yellow and blue

   (c) magenta, cyan and yellow

   (d) white, black, red, yellow, blue

7. If the ascii code for the character ”7” is 55, what is the UTF-8 code for ”7” (in base 10)?

   (a) 155

   (b) 55

   (c) 37

   (d) 67

8. Which of the following phrase structure trees (on the next page) best represents the meaning of the sentence: *The data scientist and the botonist walked three miles with a bag of supplies*.

   (a) Figure A

   (b) Figure B

   (c) Figure C

   (d) Figure D

Figure A

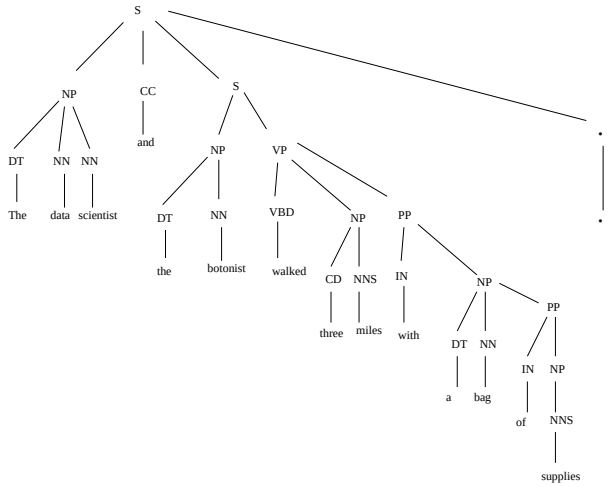


Figure B

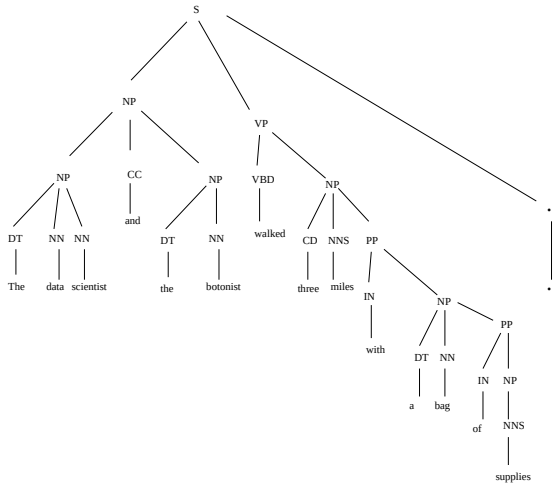


Figure C

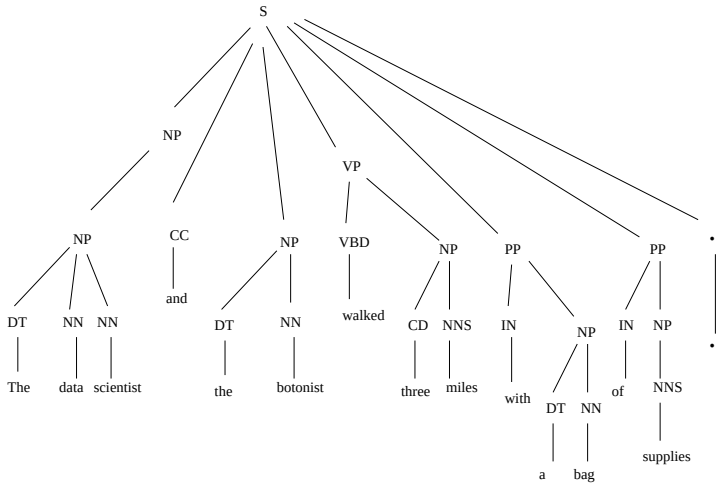
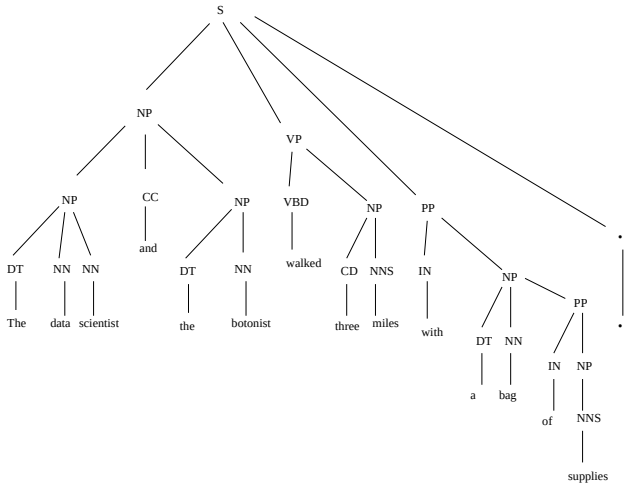


Figure D



9. You are writing a system to automatically classify literature into different genres, based on TF-IDF scores for each word in each document. The word *unicorn* appears 500 times in a single document called *The Evil Unicorn*. The word *unicorn* appears in 20 out of a total of 1000 documents in your sample. What is TF-IDF score for *unicorn* in the vector for *The Evil Unicorn*? Pick the best answer. Assume natural logarithms for IDF and assume that TF is a simple count of the number of instances with no normalization.

   (a) 10.0

   (b) 0.0078

   (c) 10,000

   (d) 1956.01

10. Which of the following statements are true about Supervised Machine Learning? Mark all that apply.

    (a) Systems tend to perform the best when training data is more similar to the test data.

    (b) Systems tend to perform better if there is more training data.

    (c) Systems tend to perform better if the training data is more consistent, e.g., if manual annotation of data with task labels has high inter-annotator agreement scores.

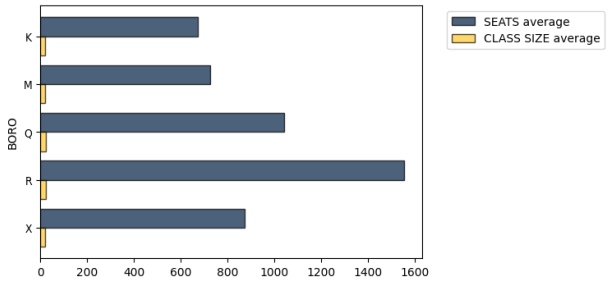
    (d) Different supervised machine learning algorithms perform better for different datasets.

Section 2

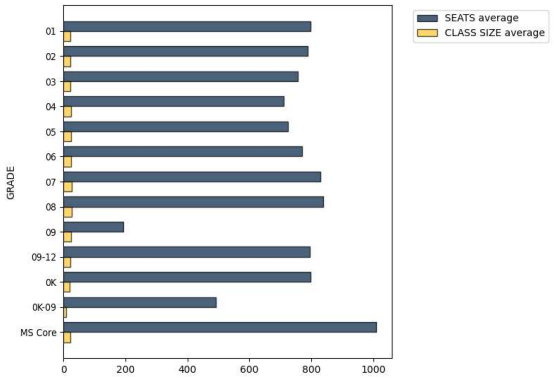
Answer both of the following two questions. Depending on the question, the answer can be a shell script, a Python program, a regular expression or some other programmatic solution to some problem. You can test the program on a linux/Apple/Windows terminal or in an IDE like Idle of Jupyter. However, it 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. In the "DataScienceTools/Final" directory', copy and load the file **practice_final_11_setup.ipynb**. Then create two bar graphs. For the first one, create a table that only includes the labels: **'GRADE'**, **'SEATS'** and **'CLASS SIZE'**. Then group by **'GRADE'** so that the bars represent averages of the grouped amounts. To create the second bar graph, first create a table that includes the features **'BORO'**, **'SEATS'** and **'CLASS SIZE'**. For the bar graph, group by 'BORO' instead of **'GRADE'** and display aveages as before. The images should look like the ones below. Before submitting your code, convert it to a .py file that works from the command line. Then copy the code into the test file that you are going to submit.

'Bar graph for distribution by borough.'



Bar graph for distribution by grade level.'



Question 12. Write a python program that compares two .csv files that are in the form indicated below. The program should work on the command line and take two arguments: a system .csv file and an answer key .csv file. The program should compute precision, recall and f-measure and print these to the screen. Remember the numerator of both precision and recall is the number of correct answers. The denominators are the number of answers in the system output (precision) and the number of answers in the answer key (recall). F-measure is the harmonic mean of precision and recall (2 * precision * recall)/(precision + recall). The answer key and the system output can be of different lengths if not all input elements are classified.

Sample .csv file

```
Item_number,Classification
1,2
2,1
3,0
4,3
5,2
6,0
7,2
```

We will assume that the .csv file format above represents a version of the Iris identification task described in class, except that instead of the 3 species, we will assume four categories: 0 means not an iris (does not belong to any class), and 1, 2 and 3 indicate 3 species of iris. Thus false positives (system incorrectly marks a flower as a type of iris) will cause precision to go down, where as false negatives (system does not recognize a flower as an iris) will cause recall to go down.

The program needs only compare the two .csv files, record the number of times that the system output and answer key agree and the lengths of each (number of flowers total or one less than the number of lines).

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 '$\backslash$'). 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 and 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., **paragrpahs[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.
   - **^C, kill, fg, bg, ^Z, &, 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 expresions
    (a) Repeat operators: *, +, {number}, {minimum,maxium}
    (b) Optional and disjunction: ? and |
    (c) Parentheses () mark scope of repeat, optional and disjunction operators
    (d) Beginning and End of line: ^and $
    (e) Disjunction (and ranges) for single characters: [a-z], [A-Z], [0-9], [a-zA-Z,0-9], [123abc]
    (f) Negation for single characters [^a-z]
    (g) The period . represents any character, e.g., ".*" represents any string. Thus the expression ' ^A.*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*.