

# Assignment 1

**WEEK 01: THE UNIX SHELL, GIT AND AMAZON EC2**

**WEEK 02: PYTHON**

**WEEK 03: PYTHON LIBRARIES**

**Teacher and teaching assistant :**

David Kofoed Wind,  
Finn Årup Nielsen  
(Rasmus) Malthe Jørgensen  
Ulf Aslak Jensen

# CONTENTS

## WEEK 01

Exercise 1.1-----	3
Exercise 1.2-----	6
Exercise 1.3-----	8
Exercise 1.4-----	11
Exercise 1.5-----	13

## WEEK 02

Exercise 2.1-----	16
Exercise 2.2-----	19
Exercise 2.3-----	21

## WEEK 03

Exercise 3.1-----	25
Exercise 3.2-----	27
Exercise 3.3-----	31
Exercise 3.4-----	35
Exercise 3.5-----	41

# WEEK 01: THE UNIX SHELL, GIT AND AMAZON EC2

## Exercises:

### Exercise 1.1

Write a command that finds the 10 most popular words in a file.

Solution:

#### 1. Formulation

The solution is a shell script 'findWord.sh' which will take the input "cnn\_news.txt". and output the 10 most popular words. First of all, the input file "cnn\_news.txt" contains some text which is going to be the primary input. The text is copied from web news.

In order to do the task, some linux commands such as 'tr', 'sort', 'uniq', 'head' is used. Those commands will be written in the shell script.

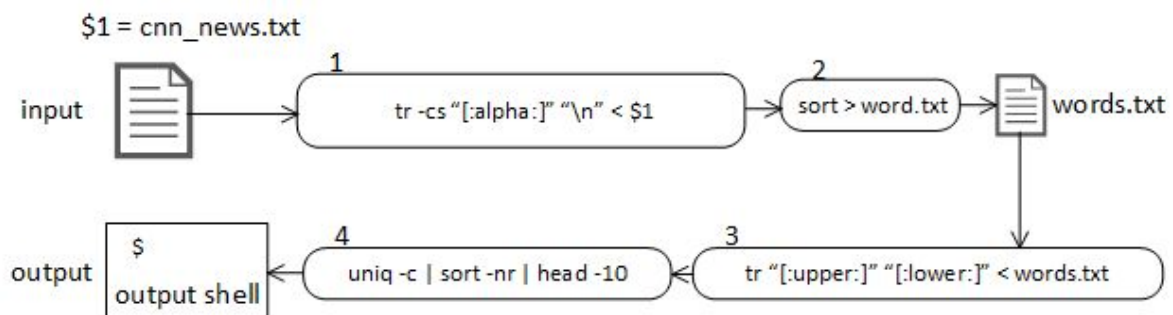


Figure 1.1.1 the commands flow chart

Figure 1.1.1 shows the commands flow chart. If we execute the shell script, it will take a file as the input. Then it will run through the series of commands from No.1 to No.4 as the flow chart shows. Finally the terminal will show the top 10 popular words.

#### 2. Commands description

```
$ tr -cs "[:alpha:]" "\n" < $
```

The command aims to translate the whole set of alphabet characters in the text file, one per line, where a word is taken to be a maximal string of letters. This command is used to truncate the text in to many pieces of words.

```
2.$ sort > words.txt
```

The command sorts all of the words inside the content in alphabet sequence and then output to a new file “words.txt”. The command is used for saving the output of the words and sorts the word as the alphabet sequence

```
3. $ tr "[:upper:]" "[:lower:]" < words.txt
```

This command takes the “words.txt” file and normalizes all words into lower case. It is used to prevent the ‘uniq’ command which would distinguish the capital letters and small letters.

```
4. $ uniq -c | sort -nr | head -10
```

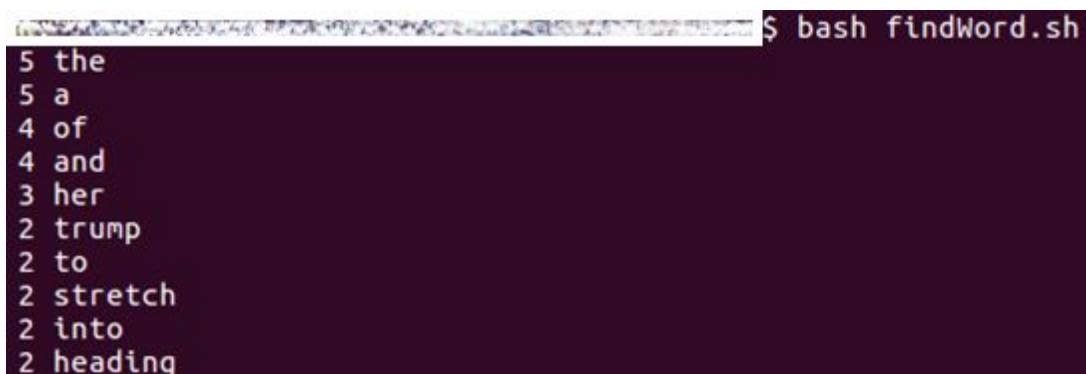
These commands from left to right counts the unique words of the “words.txt”. Then the data would be sorted by the next command as numerical sequence and display in reverse style. The last command would modify the display for top 10 words to full fill the requirement. In next paragraph would show the content of script and the outcome.

### 3. Shellscript

```
1. $ tr -cs "[:alnum:]" "\n" < $1 | sort > words.txt | tr
"[:upper:]" "[:lower:]" < words.txt | uniq -c | sort -nr |
head -10
```

The diagram shows completed commands of the sh script.

### 4. Result



```
$ bash findWord.sh
5 the
5 a
4 of
4 and
3 her
2 trump
2 to
2 stretch
2 into
2 heading
```

Figure 1.1.2 the display of 10 most popular word from “cnn\_news.txt”

Figure 1.1.2 shows the output of the sh script. The first column is the duplicated numbers of word. The second column is the 10 most popular words from “cnn\_news.txt”

in a sequence from top to bottom. The following paragraphs are showing the content of the text file.

## 5. Test

“cnn\_news.txt”

(CNN)Hillary Clinton has advantages heading into the final stretch of the campaign that any presidential candidate would envy: fleet of popular surrogates, mountain of cash and an opponent who is often sidetracked by self-inflicted wounds.

Yet the Democratic nominee enters this home stretch in a dead heat against Donald Trump, according to CNN/ORC poll released Tuesday.

The close contest heading into the fall underscores Clinton's vulnerabilities on trust and honesty -- and her need to summon a relentless and efficient ground game, even if many of her voters are fueled more by revulsion toward Trump than excitement about her.

Figure 1.4 shows test of some popular words

The test shows the script is doing the work properly. The tests simply marked out the several duplicated words in same color and compare with the output.

## Exercise 1.2:

Put this data (<https://www.dropbox.com/s/d5c4x905w4jelbu/cars.txt?dl=0>) into a file and write a command that removes all rows where the price is more than 10,000\$.

## Solution

### 1. Formulation

To solve the exercise a shell script will be used which is called “search.sh”. It will take an argument as the input and then output the list without the rows where the price is more than 10,000\$. The file “cars.txt” is the input for this case. There are couples of ways to achieve the goal. 2 Linux commands ‘gawk’ and ‘cat’ will be used in the shell script ‘search.sh’.

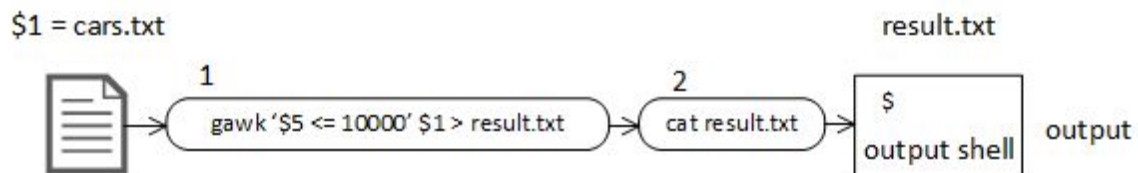


Figure 1.2.1 the command flow chart

Figure 1.2.1 shows the flow chart of the sh file. If we execute the shell script then it will take the input ‘cart.txt’ and then run through the two commands. Finally it will output a new list which does not contain the rows where the prices are higher than 10,000\$. The command will be explained in the paragraphs below.

### 2. Command description

```
1. $ gawk '$5 < 10000' $1 > result.txt
```

First the command will take the input as \$1. Then it will print every line but the \$5 which are higher or equal than 10,000\$. Then it will be printed to a file called ‘result.txt’.

```
2. $ cat result.txt
```

The command is to display the content in side “result.txt”.

### 3. Shell script

```
1. $ gawk '$5 <= 10000' cars.txt > result.txt
2. $ cat result.txt
```

The diagram shows the completed commands.

#### 4. Result

```
$ bash search.
sh
plym    fury    77      73      2500
chevy   nova    79      60      3000
volvo   gl       78      102     9850
Chevy   nova    80      50      3500
fiat     600     65      115     450
honda   accord  81      30      6000
toyota  tercel   82      180     750
chevy   impala   65      85      1550
ford    bronco   83      25      9525
```

Figure 1.2.2 shows the output content

Figure 1.2.2 shows the output content. The fifth column is the '\$5' argument for each line. From the output we can see it has already filtered the any numbers are greater or equals to 10,000\$.

#### 5. Test

"cars.txt"

The file from the link which is the text file "cars.txt" will show a list like below:

1	plym	fury	77	73	2500
2	chevy	nova	79	60	3000
3	ford	mustang	65	45	17000
4	volvo	gl	78	102	9850
5	ford	ltd	83	15	10500
6	Chevy	nova	80	50	3500
7	fiat	600	65	115	450
8	honda	accord	81	30	6000
9	ford	thundbd	84	10	17000
10	toyota	tercel	82	180	750
11	chevy	impala	65	85	1550
12	ford	bronco	83	25	9525

### Exercise 1.3:

Using this file (<https://www.dropbox.com/s/tjv9pyfrd9ztx8r/dict?dl=0>) as a dictionary, write a simple spellchecker that takes input from stdin or a file and outputs a list of words not in the dictionary. One solution gets 721 misspelled words in this Shakespeare file (<https://www.dropbox.com/s/bnku7grfycm8ii6/shakespeare.txt?dl=0>). Consider using the command “comm”.

### Solution:

#### 1.Formulation

In order to do the task, we make a shell script called “spell\_check.sh”. The file “shakespear.txt” and “dict” will be the input file which will go through the script. Finally the script will print out the number of words that are not in the “dict.txt” at the terminal. In order to do the task, some commands such like ‘cat’, ‘tr’, ‘sort’, ‘sed’, ‘comm’ and ‘rm’ are used.

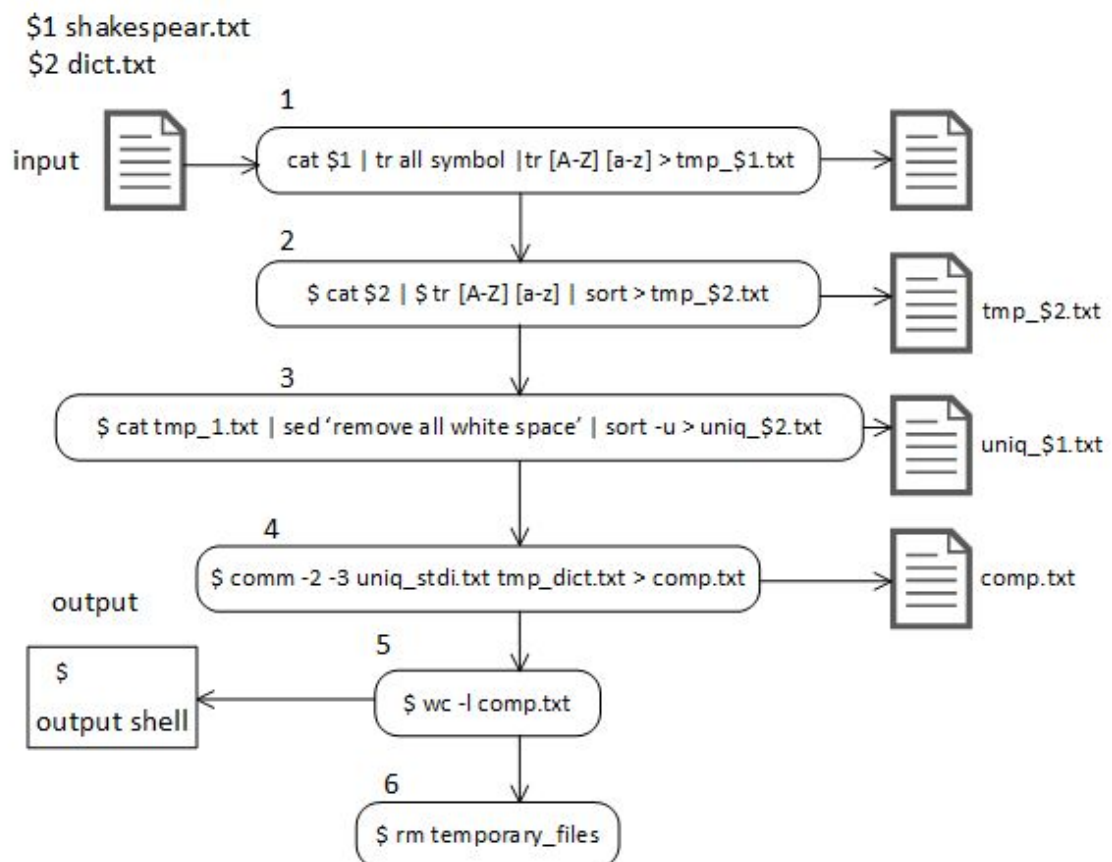


Figure 1.3.1 the flow chart of the shell script.

Figure 1.3.1 shows the flow chart of this shell script. The shell script takes two inputs. The first one is the “shakespear.txt” file and the second one is the “dict.txt”. The script



will take the two inputs and go through the commands from No. 1 to No. 6 in the flow chart. The following paragraphs will explain the commands.

## 2. Command description

```
1. $ cat $1 | tr " " "\n" | tr "," "\n" | tr "." "\n" | tr ":" "\n" | tr "\"\n" | tr "(" "\n" | tr ")" "\n" | tr "\t" "\n" | tr ";" "\n" | tr "[" "\n" | tr "]" "\n" | tr "?" "\n" | tr [A-Z] [a-z] | tr "-" "\n" | tr "'" "\n" | tr "&" "\n" | tr "\!" "\n" > tmp 1.tx
```

This command is used to catch the “shakespear.txt” file and then translate all punctuations and spases into newline symbol. We also normalize all the capital letters to small letters. As a result, it will output a temporary file which contains only one separate word in each line.

```
2. $ cat $2 | tr [A-Z] [a-z] > tmp 2.txt
```

The command is used to catch the “dict” file and normalize all the words from upper case to lower case. Then it outputs to the temporary file “tmp\_2.txt” which contains only lower case characters.

```
3. $ cat tmp 1.txt | sed '/^$/d' | sed 's/^[[:space:]]*/g' | sort -u > uniq 1.txt
```

The command is used to catch the “tmp\_1.txt” file and delete all word begins with ‘\$’. Then it empties all spaces at the beginning of the words. Finally it produces a temporary file called “uniq\_1.txt” which will only contain the pure words in each line.

```
4. $ comm -2 -3 tmp 1.txt tmp2.txt > comp.txt
```

The command is used to compare the two temporary files “tmp\_1.txt” and “tmp\_2.txt” and only show the unique words to “tmp\_2.txt”. Then it outputs to the temporary file “comp.txt”.

```
5. $ wc -l comp.txt
```

The command is used to display the counted number of the words in “comp.txt”.

```
6. $ rm tmp 1.txt tmp 2.txt uniq 1.txt comp.txt
```

The command is used to remove all the temporary files in the end.

## 3. Shell script

```
1. $ cat $1 | tr " " "\n" | tr "," "\n" | tr "." "\n" | tr ":" "\n" | tr "\"\n" | tr "(" "\n" | tr ")" "\n" | tr "\t" "\n" | tr ";" "\n" | tr "[" "\n" | tr "]" "\n" | tr "?" "\n" | tr [A-Z]
```

```
[a-z] | tr "-" "\n" | tr "'" "\n" | tr "&" "\n" | tr "\" "\n" >
tmp 1.txt
2. $ cat $2 | tr [A-Z] [a-z] > tmp 2.txt
3. $ cat tmp 1.txt | sed '/^$/d' | sed 's/^[[:space:]]*/g' | sort
-u > uniq 1.txt
4. $ comm -2 -3 tmp 1.txt tmp2.txt > comp.txt
5. $ wc -l comp.txt
6. $ rm tmp 1.txt tmp 2.txt uniq 1.txt
```

The diagram shows the script with the inputs

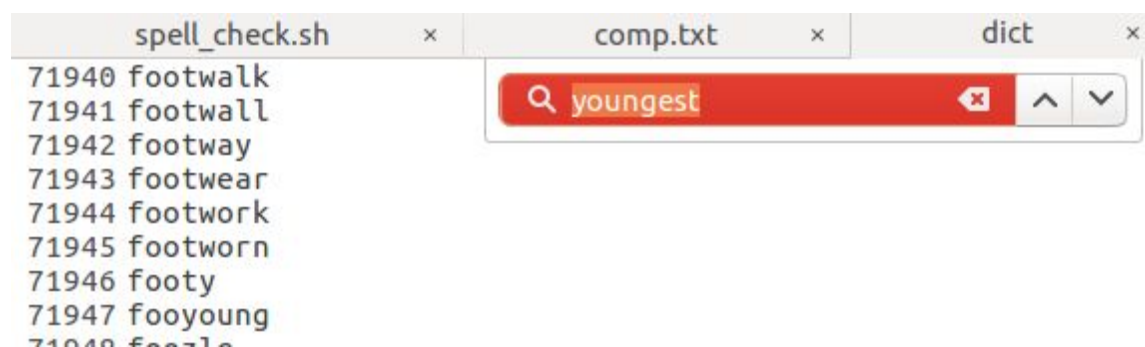
#### 4. Result

```
1. $ bash spell_checker.sh shakepear.txt dict
$ bash spell_check.sh shakespeare.txt dict
721 comp.txt
$ cat -n comp.txt |tail -10
712 worships
713 worthier
714 wrangling
715 wraps
716 wrestled
717 writers
718 writes
719 years
720 yields
721 youngest
```

Figure 1.3.2 the output of the shell script

Figure 1.3.2 shows the number of the different words from “shakespear.txt” to “dict”. This means the script gets the same result as the solution given by the course webpage.

#### 5. Test



```
spell_check.sh x comp.txt x dict x
71940 footwalk
71941 footwall
71942 footway
71943 footwear
71944 footwork
71945 footworn
71946 footy
71947 fooyoung
71948 footy
```

Figure 1.3.3 shows the test

Figure 1.3.3 shows a simple test which is to find the last different word from “dict” file. We can’t find the word in the output file, which means that the script is working as required.

## Exercise 1.4:

Launch a t2.micro instance on Amazon EC2. Log onto the instance, create some files and install some software (for example git). (You have to enter your credit card to make an Amazon account. If you want to make sure you do not spend any money, you can remove your account when you are finished with the exercises. If you really don't want to do this, you can use the GBar instead of Amazon.)

### Solution:

#### 1. Formulation

Amazon EC2 is short for Elastic Cloud 2. It is the Amazon Web Service that provides compute capacity in the server. In order to log onto the EC2 instance, users have to create an account in AWS. There is a good tutorial (<http://docs.aws.amazon.com/gettingstarted/latest/awsgsg-intro/gsg-aws-tutorials.html#tutorials-compute>) for setting the EC2 Virtual Machine instance. The detailed information for setting up the VM will be omitted here.

The requirement is rather easy, which is to create files on the VM and also install some software. Here as for the exercise we would install git. Which also could be a benefit to exercise 1.5. The following pictures are to show the 'mkdir', 'nano' and 'apt-get' in the VM.

#### 2. Solution

```
ubuntu@ip-10-10-10-10:~$ mkdir bigwork2016
ubuntu@ip-10-10-10-10:~$ ls
bigwork2016
```

Figure 1.4.1 the directory made on EC2

```
ubuntu@ip-10-10-10-10:~/repo$ nano ReadMe.txt
ubuntu@ip-10-10-10-10:~/repo$ ls
ReadMe.txt
```

Figure 1.4.2 the file made on EC2

Figure 1.4.1 and 1.4.2 above are showing to make directory and file on the EC2.

```
ubuntu@ip-10-10-10-10:~$ sudo apt-get install git
```

Figure 1.4.3 the command for installing git repository

Figure 1.4.3 shows the installation command on EC2

### 3. Test

```
ubuntu@ip-10-10-10-10:~$ git --help
usage: git [--version] [--help] [-C <path>] [-c name=value]
       [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
       [-p|--paginate|--no-pager] [--no-replace-objects] [--bare]
       [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
       <command> [<args>]
```

Figure 1.4.4 the git is installed

This figure shows the git is installed. By typing the 'git --help' you could test if the git is installed. Now it shows the git is installed.

## Exercise 1.5

Create a few files locally on your computer. Create a new repository on Github and push your files to this repository. Log on to a t2.micro instance on Amazon EC2 and clone your repository there. Make some changes to the files, push them again and pull the changes on your local machine.

(If you did not make an Amazon EC2 account in Exercise 1.4, then you should push your files and pull them on the gbar.)

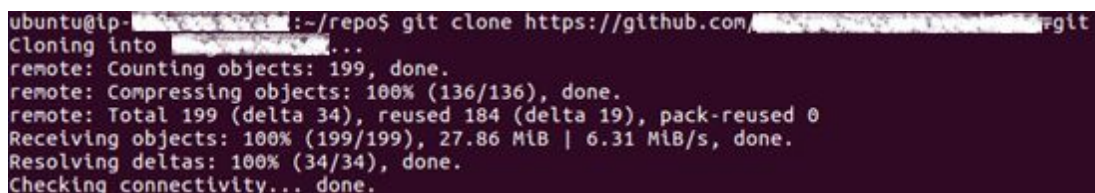
### Solution:

#### 1. Formulation

The exercise is to train the version control between local machine and the server. In order to that some git commands would be used such as 'git clone, add, commit, fetch, push and pull'. The following 5 snap shots will explain how the version control is done step by step.

#### 2. Solution

##### git clone on EC2

A terminal window screenshot showing the output of a 'git clone' command. The prompt is 'ubuntu@ip-10-10-10-10:~/repo\$'. The command is 'git clone https://github.com/...'. The output shows the cloning process: 'Cloning into ...', 'remote: Counting objects: 199, done.', 'remote: Compressing objects: 100% (136/136), done.', 'remote: Total 199 (delta 34), reused 184 (delta 19), pack-reused 0', 'Receiving objects: 100% (199/199), 27.86 MiB | 6.31 MiB/s, done.', 'Resolving deltas: 100% (34/34), done.', and 'Checking connectivity... done.'

```
ubuntu@ip-10-10-10-10:~/repo$ git clone https://github.com/...#git
Cloning into ...
remote: Counting objects: 199, done.
remote: Compressing objects: 100% (136/136), done.
remote: Total 199 (delta 34), reused 184 (delta 19), pack-reused 0
Receiving objects: 100% (199/199), 27.86 MiB | 6.31 MiB/s, done.
Resolving deltas: 100% (34/34), done.
Checking connectivity... done.
```

Figure 1.5.1 the git clone is succeeded on EC2

Figure 1.5.1 shows the result of cloning the repository from github to the EC2. Firstly the repository which is created in github should be clone to both local machine and the server. Since the git has already installed on local machine so the part would be omitted.

### git status on EC2

```
ubuntu@ip-10-10-10-10:~$ nano test_EC2.txt
ubuntu@ip-10-10-10-10:~$ git status
On branch master
Your branch is up-to-date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   README.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        test_EC2.txt
```

Figure 1.5.2 the changes on the EC2

Figure 1.5.2 shows the changes on the EC2. The next is to add the changes and commit.

### git add & git commit -m "" on EC2

```
ubuntu@ip-10-10-10-10:~$ git add -A
ubuntu@ip-10-10-10-10:~$ git status
On branch master
Your branch is up-to-date with 'origin/master'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   README.txt
        new file:   test_EC2.txt

ubuntu@ip-10-10-10-10:~$ git commit -m "test from EC2"
[master c8dfebe] test from EC2
2 files changed, 4 insertions(+), 1 deletion(-)
create mode 100644 test_EC2.txt
```

Figure 1.5.3 the changes is added

Figure 1.5.3 shows the changes is added and committed. The next step is to do the 'git fetch' and 'git push' to upload the changes to the repository.

### git fetch & git push



```
$ git fetch origin master
From https://github.com/[redacted]
 * branch                master      -> FETCH_HEAD
ubuntu@[redacted] $ git push origin master
Username for '[redacted]':
Password for '[redacted]':
Counting objects: 8, done.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 570 bytes | 0 bytes/s, done.
Total 4 (delta 0), reused 0 (delta 0)
To https://github.com/bigdata2016/bigwork2016.git
045675c..c8dfebe master -> master
```

Figure 1.5.4 the changes is pushed to the repository

Figure 1.5.4 shows the changes is pushed to the repository from the EC2 to the github. The next step is to fetch and pull from the local machine.

git fetch & git pull

```
$ git fetch origin master
remote: Counting objects: 23, done.
remote: Compressing objects: 100% (21/21), done.
remote: Total 23 (delta 3), reused 22 (delta 2), pack-reused 0
Unpacking objects: 100% (23/23), done.
From https://github.com/[redacted]
 * branch                master      -> FETCH_HEAD
  e435a47..c8dfebe master  -> origin/master
ubuntu@[redacted] $ ls
README.txt  week1  week2  week3  [redacted]
ubuntu@[redacted] $ git pull origin master
From https://github.com/[redacted]
 * branch                master      -> FETCH_HEAD
Merge made by the 'recursive' strategy.
 README.txt              | 4 +-
 test_EC2.txt            | 1 +
```

Figure 1.5.5 the pull command from the local machine

Figure 1.5.5 shows that by the 'git pull' command we can see the updates to the local machine.

## WEEK 02: PYTHON

### Exercises:

#### Exercise 2.1:

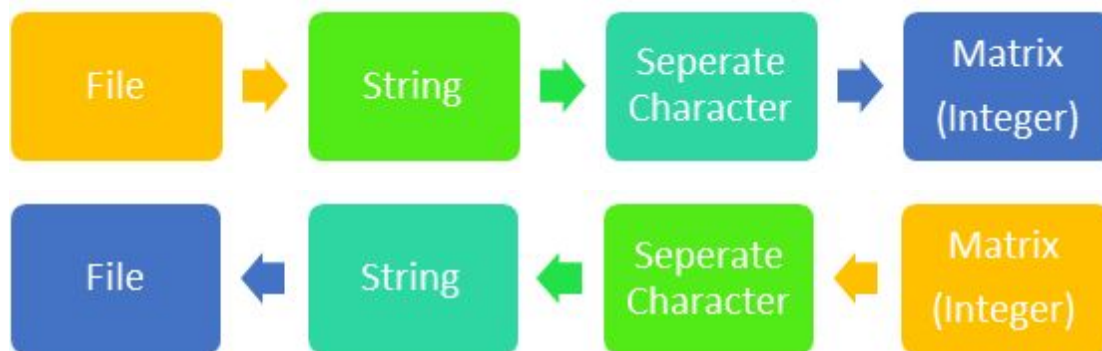
Write a script with two methods. The first method should read in a matrix like the one here and return a list of lists. The second method should do the inverse, namely take, as input, a list of lists and save it in a file with same format as the initial file. The first method should take the file name as a parameter. The second method should take two arguments, the list of lists, and a filename of where to save the output.

#### Solution:

In this exercise we are faced with the following problems:

- File manipulation, including opening, closing, reading and writing
- Memory input and output
- Segmentation of alphabetic string
- Transformation between alphabetic string and integer

The process in both methods can be demonstrated by the following flowchart.



#### Method 1 (Input):

1.The object we are dealing with is illustrated below.

1	0	1	1	3	0
2	0	2	3	4	10
3	8	2	2	0	7

As is shown, the matrix is saved in a text file with space and line break used as delimiter. Obviously, the first step of processing the matrix is to open the file and read the text inside, where functions `open()` and `fileObject.readlines()` are respectively used.



2. Once the matrix is read in, it is temporarily saved as a list of strings, which constitutes of numerical characters and space. For further processing, separating the numbers from each other is necessary. And function `string.split()` is perfect for this purpose.
3. The object we are facing is now a list of lists and every thing seems perfect.

```
[['0', '1', '1', '3', '0'],
 ['0', '2', '3', '4', '10'],
 ['8', '2', '2', '0', '7']]
```

Be careful! Look at each element in the list, you'll find it is a number in quotation. That indicates the list is now made up of numerical characters instead of integers. Hence the last step is the transformation of every element into integers, which requires function `int()`.

Let's have a global view of codes and the result.

```
def mat2lis(filename):
    f = open(filename)
    arr = f.readlines()
    for i in range(len(arr)):
        arr[i] = arr[i].split()
        for j in range(len(arr[i])):
            arr[i][j] = int(arr[i][j])
    f.close()
    return arr
matrix = mat2lis('E:/repo/bigwork2016/week2/ex2.1/wen/R9FVAdXW.txt')
print(matrix)
```

```
[[0, 1, 1, 3, 0], [0, 2, 3, 4, 10], [8, 2, 2, 0, 7]]
```

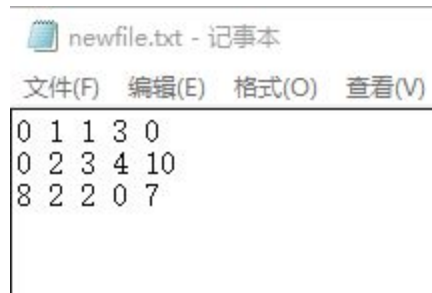
### Method 2 (Output):

This method appoximately does the inverse of Method1. Similarly, we need to open the file first. And then a temporary string should be created. The key step in this method is to append every element in the list as well as space or line break into the temporary string utilizing function `string.append()`. Of course, before each step of appending number we are required to convert it into a character, or a single-character string by function `str()`. Once appending is accomplished, the string can be written into the file using `fileObject.writelines()`.

Codes and the screenshot are attached below.

```
def lis2mat(filename, lis=[[]]):
    f = open(filename, 'w')
    tmp=[]
    for i in range(len(lis)):
        for j in range(len(lis[i])-1):
            tmp.append(str(lis[i][j])+' ')
        tmp.append(str(lis[i][len(lis[i])-1])+'\n')
```

```
f.writelines(tmp)
f.close()
lis2mat('newfile.txt',arr)
```



The screenshot shows a Notepad window with the title 'newfile.txt - 记事本'. The menu bar includes '文件(F)', '编辑(E)', '格式(O)', and '查看(V)'. The text area contains the following matrix:

0	1	1	3	0
0	2	3	4	10
8	2	2	0	7

## Exercise 2.2:

Write a script that takes an integer  $N$ , and outputs all bit-strings of length  $N$  as lists. For example: 3 -> [0,0,0], [0,0,1],[0,1,0],[0,1,1],[1,0,0],[1,0,1],[1,1,0],[1,1,1]. As a sanity check, remember that there are  $2^N$  such lists.

Do not use the bin-function in Python. Do not use strings at all. Do not import anything. Try to solve this using only lists, integers, if-statements, loops and functions.

## Methodology:

In my opinion, this exercise requires more mathematical knowledge than coding skills. Though it is bit-strings that the exercise asks for, we can still regard them as binary numbers with digits from the string on different places. And in this respect we soon find these binary numbers, if transformed into decimal numeral system, exactly constitute a set including all non-negative integers less than  $2^N$ .

From the knowledge of positional notation, we can transform a decimal number into binary numeral system by Euclidean division. Here is an example.

$$\begin{aligned} 6 \div 2 &= 3 \dots 0 \\ 3 \div 2 &= 1 \dots 1 \\ 1 \div 2 &= 0 \dots 1 \\ 6_{10} &= 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 110_2 \end{aligned}$$

For this instance, [1,1,0] is one of the bit-strings we want under  $N=3$ . And we can get all bit-strings required by transforming all integers from 0 to  $2^N-1$  in this way.

## Solution:

1. In order for simplicity of coding, we do not use operator `//` or `%` directly in Euclidean division. Instead, we successively compare the number to be divided with  $2^i$ . ( $i=N-1, \dots, 1, 0$ ) If the denominator is no less than  $2^i$ , the digit on the  $(i+1)$ th place in inverse order of the string would be 1, and the new denominator would be the original denominator minus  $2^i$ ; else the digit would be 0 and the denominator remains. If you are familiar with the knowledge of positional notation, you'll find there is little difference between the idea in this solution and the methodology above. And one important thing is that this can be easily coded by if-statements and for-loops.

2. In this exercise, we get the number  $N$  from the standard input, or the keyboard, which using function `input()`. Since the input would be saved as a string generally, we need to convert it into an integer by function `int()` for further processing. And for the robustness of programming, we want to ensure number  $N$  get from keyboard is a positive integer, hence we utilize an if-else statement to check it.

Codes and the result are as follows.

```
def list_tt (in_put):
    tt = []
    for bit_val in range(2**in_put, 0, -1):
        pattern=[]
        for scale_val in range(in_put, 0, -1):
            if bit_val >2**(scale_val-1):
                bit_val = bit_val-2**(scale_val-1)
                pattern.append(1)
                #print 1
            elif bit_val <= 2**(scale_val-1):
                #print 0
                pattern.append(0)
        tt.append(pattern)
    return tt
print("Enter the natural of input: ")
ip = int(input())
if ip <= 0:
    print("invalid number")
else:
    print (list_tt(ip))

===== RESTART: E:\repo\bigwork2016\week2\ex2.2\ybw.py =====
Enter the natural of input:
4
[[1, 1, 1, 1], [1, 1, 1, 0], [1, 1, 0, 1], [1, 1, 0, 0], [1, 0, 1, 1], [1, 0,
1, 0], [1, 0, 0, 1], [1, 0, 0, 0], [0, 1, 1, 1], [0, 1, 1, 0], [0, 1, 0, 1],
[0, 1, 0, 0], [0, 0, 1, 1], [0, 0, 1, 0], [0, 0, 0, 1], [0, 0, 0, 0]]
>>>
===== RESTART: E:\repo\bigwork2016\week2\ex2.2\ybw.py =====
Enter the natural of input:
-1
invalid number
>>>
```

### Exercise 2.3:

Write a script that takes this file (from this Kaggle competition), extracts the `request_text` field from each dictionary in the list, and construct a bag of words representation of the string (string to count-list).

There should be one row pr. text. The matrix should be  $N \times M$  where  $N$  is the number of texts and  $M$  is the number of distinct words in all the texts.

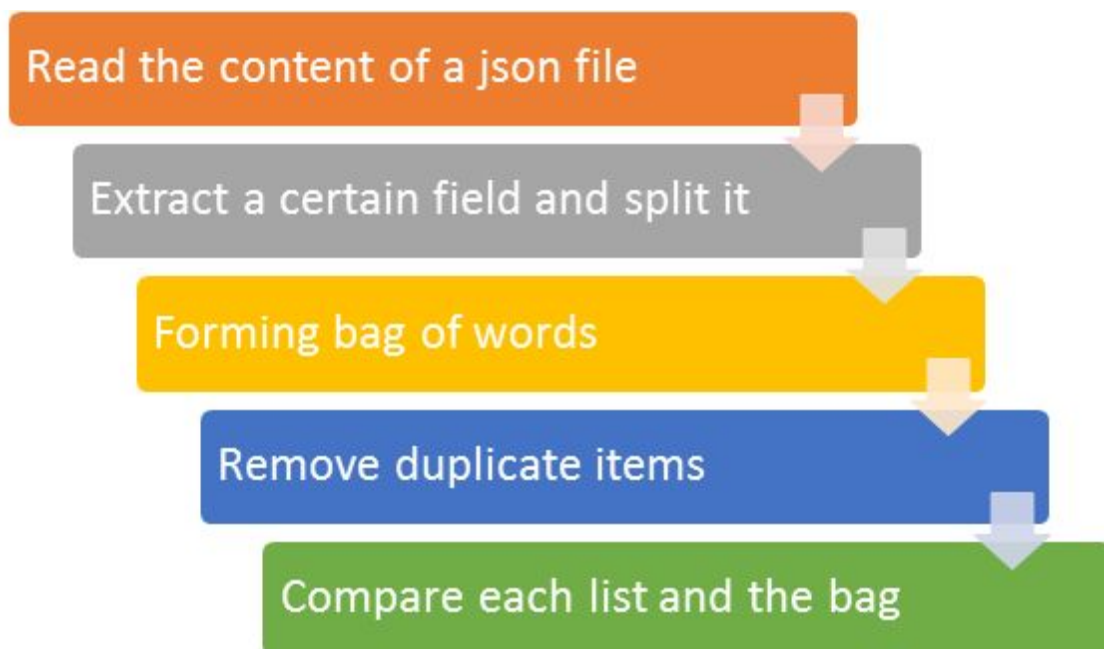
The result should be a list of lists (`[[0,1,0],[1,0,0]]` is a matrix with two rows and three columns).

### Solution:

Main problems this exercise deals with are as follows:

- File manipulation, especially the json file
- Extraction of a certain field from a json file
- Segmentation of alphabetic string
- Removal of duplicate items from a list
- Comparison of two lists

The process can be illustrated by the following flowchart.



1. As usual, the first step of processing is still read in the content from the file, and this time the file is of .json type, hence there will be an extra operation to transform the content saved as strings to json objects. Two key functions are respectively `fileObject.read()` and `json.load()`.

Some codes are demonstrated below.

```
import json
with open('pizza-train.json') as file_i:
    content_string = file_i.read()
    list_dict_json = json.loads(content_string)
    file_i.close()
```

Under this circumstance, *list\_dict\_json* is a dictionary of json objects, each of which has several same fields including *request\_text*. Here is an example of a json object.

```
>>> list_dict_json[0]
{'requester_account_age_in_days_at_request': 0.0, 'requester_upvotes_minus_downvotes_at_retri
eval': 1, 'requester_number_of_comments_at_retrieval': 0, 'requester_number_of_subreddits_at_
request': 0, 'requester_user_flair': None, 'requester_number_of_comments_in_raop_at_retrieval
': 0, 'giver_username_if_known': 'N/A', 'request_text': 'Hi I am in need of food for my 4 chi
ldren we are a military family that has really hit hard times and we have exahusted all means
of help just to be able to feed my family and make it through another night is all i ask i kn
ow our blessing is coming so whatever u can find in your heart to give is greatly appreciated',
'requester_days_since_first_post_on_raop_at_request': 0.0, 'request_id': 't3_l25d7', 'requ
ester_upvotes_minus_downvotes_at_request': 0, 'post_was_edited': False, 'number_of_upvotes_of
_request_at_retrieval': 1, 'requester_number_of_posts_on_raop_at_request': 0, 'requester_numb
er_of_posts_on_raop_at_retrieval': 1, 'requester_number_of_posts_at_retrieval': 1, 'requester
_username': 'nickylvst', 'requester_account_age_in_days_at_retrieval': 792.4204050925925, 're
quest_text_edit_aware': 'Hi I am in need of food for my 4 children we are a military family t
hat has really hit hard times and we have exahusted all means of help just to be able to feed
my family and make it through another night is all i ask i know our blessing is coming so wha
tever u can find in your heart to give is greatly appreciated', 'requester_upvotes_plus_downv
otes_at_request': 0, 'requester_subreddits_at_request': [], 'unix_timestamp_of_request_utc':
1317849007.0, 'requester_number_of_posts_at_request': 0, 'requester_upvotes_plus_downvotes_at
_retrieval': 1, 'requester_days_since_first_post_on_raop_at_retrieval': 792.4204050925925, 'r
equester_received_pizza': False, 'requester_number_of_comments_in_raop_at_request': 0, 'reque
ster_number_of_comments_at_request': 0, 'request_number_of_comments_at_retrieval': 0, 'numbe
r_of_downvotes_of_request_at_retrieval': 0, 'request_title': 'Request Colorado Springs Help Us
Please', 'unix_timestamp_of_request': 1317852607.0}
```

2. Now we want to extract a certain field, *request\_text*, from each object in the dictionary. Our solution is to define a new function called *search\_kw()* to deal with this problem. This function initializes a list and appends it with the content of that certain field of every json object, which requires a for-loop. The function *lower()* can turn all letters in the string to lower case, and *string.replace()* can be used to remove all the punctuations. Obviously, the list returned by the function is exactly what we want.

Some codes are demonstrated below.

```
punctuation = [",", ":", ";", ".", "'", '"', "?", "/", "-", "+", "&",
"(", ")", "!", ""]
def search_kw(*list_dict_json):
    request_txt=[]
    for i in range(len(list_dict_json)):
        temp_txt=(list_dict_json[i]['request_text']).lower()
        for punc in punctuation:
            temp_txt=temp_txt.replace(punc,"")
        request_txt.append(temp_txt)
    return request_txt
request_txt=search_kw(*list_dict_json)
```

Here is an illustration of the list returned.



```
>>> request_txt[0]
'hi i am in need of food for my 4 children we are a military family that has really hit hard
times and we have exahusted all means of help just to be able to feed my family and make it t
hrough another night is all i ask i know our blessing is coming so whatever u can find in you
r heart to give is greatly appreciated'
```

3. Since we are looking forward to the bag of words, it is necessary that each review by audience be seperated items. We utilize functions `string.split()` and `list.append()`, which is similar to those in Ex2.1 or 2.2. Another significant operation is the usage of function `set()`, removing duplicate items in the list and transforming the list into a set. Key codes are attached here.

```
def bag_words(*request_txt):
    word_list=[]
    for i in range(len(request_txt)):
        uniq_word=[]
        uniq_word=set(request_txt[i].split(' '))
        word_list.append(sorted(uniq_word))
    return word_list
bag=bag_words(*request_txt)
```

The effect of operations above is shown like this.

```
>>> bag[0]
['4', 'Hi', 'I', 'a', 'able', 'all', 'am', 'and', 'another', 'appreciated', 'are', 'ask', 'be',
'blessing', 'can', 'children', 'coming', 'exahusted', 'family', 'feed', 'find', 'food', 'f',
or', 'give', 'greatly', 'hard', 'has', 'have', 'heart', 'help', 'hit', 'i', 'in', 'is', 'it',
'just', 'know', 'make', 'means', 'military', 'my', 'need', 'night', 'of', 'our', 'really', 's',
o', 'that', 'through', 'times', 'to', 'u', 'we', 'whatever', 'your']
```

4. Now we can get the bag of words from all reviews. Let's set up a new list, and then successively extend it with items in each bag by the useful function `list.extend()`. Apparently, the final result of this new list includes several duplicate items for many words appear in more than one review. Hence `set()` need to be used one more time. Remember to `list()` it back at in the end. And if you want to put these items in order, `list.sort()` could be your choice. Some codes are demonstrated below.

```
newlist=bag[0]
for i in range(len(bag)):
    newlist.extend(bag[i])
newlist=list(set(newlist))
newlist.sort()
```

The following image demonstrates some words in the bag.

```
[u'00', u'000', u'0000', u'0011011001111000', u'00pm', u'012468', u'02', u'024856', u'0273', u'03', u'04', u'05', u'05ffr', u'06', u'06t2
w', u'07', u'08', u'09', u'0_0', u'0_o', u'0bfhy', u'0cceq8gewaa', u'0lbolul', u'0swi9', u'0x2a00ef6165e1e540', u'0x6d0d49b351cf3399', u'0
x87d437d6283197f9', u'0xa55b439652fb393a', u'10', u'100', u'1000', u'1007', u'1017d', u'102', u'10239146004443176581', u'103', u'103013',
u'104', u'105', u'1050', u'1052', u'106', u'108', u'1080', u'108111747', u'10am', u'10days', u'10k', u'10lb', u'10p', u'10pm', u'10th',
u'10x', u'11', u'110', u'1100', u'1130am', u'115', u'11p', u'11pm', u'11sep', u'11th', u'11yo', u'11yr', u'12', u'120', u'1200', u'121',
u'122', u'124589', u'127', u'12am', u'12hour', u'12hr', u'12oz', u'12pm', u'12th', u'13', u'131', u'1323107399', u'133a31a88fcadd49', u'1
369ic', u'1385474349507543040', u'13mkvq', u'13pexq', u'13th', u'14', u'140', u'1400', u'142', u'14th', u'15', u'150', u'1500', u'1511',
u'151461', u'15945', u'15957703', u'15lbs', u'15p', u'15pm', u'15th', u'16', u'160', u'1600', u'16238', u'163', u'164208037116711', u'16t
h', u'17', u'17044012131257561687', u'170lbs', u'172', u'17th', u'18', u'1800', u'1823', u'186', u'18bitl', u'18th', u'19', u'190', u'1912
8', u'193', u'194', u'1969', u'1984', u'1987', u'1989', u'198kcal', u'1993', u'1994', u'1998', u'19th', u'19u7rr', u'1am', u'1cjhxc', u'1d
iypw', u'1fc8e4', u'1g3me0', u'1ljuds', u'1lk', u'1kxccc', u'1lvmvg', u'1n_my_opinion', u'1pm', u'1st', u'1stgenrex', u'1yr', u'20', u'200',
u'2000', u'2002', u'2006', u'2008', u'2009', u'2010', u'2011', u'2012', u'2013', u'20586466623', u'207', u'20', u'20bux', u'20euros',
u'20gb', u'20hrs', u'20lb', u'20oz', u'20p', u'20s', u'20stuff', u'20th', u'20x30', u'20y', u'20yr', u'21', u'211', u'212', u'214', u'21
9', u'21mdty5tdc8', u'21st', u'22', u'220', u'22193', u'222950261', u'224652260904169', u'227890_1573589896072_1125270824_31806294_2054237
_n', u'22m', u'22nd', u'22yo', u'23', u'23f', u'23p', u'23rd', u'23year', u'24', u'240', u'2400', u'241rqm', u'243', u'24k', u'24th', u'2
5', u'250', u'2500', u'258', u'25th', u'26', u'2637kw3', u'264710_1626693143620_1125270824_31873134_8125879_n', u'2674', u'26th', u'27',
u'270', u'2700', u'270872372947463', u'2736', u'27547768540', u'27th', u'28', u'280', u'285', u'2864270574115', u'2878', u'28iw93n', u'28
th', u'29', u'29424', u'29th', u'2a', u'2am', u'2ao2c', u'2c', u'2cd', u'2day', u'2fet4eut7861', u'2for1', u'2fwatch', u'2hkchis', u'2is
h', u'2k', u'2l', u'2liter', u'2months', u'2morrow', u'2nd', u'2nite', u'2p', u'2pm', u'2py2v', u'2vy4uywbty', u'2x', u'2yr', u'30', u'30
...
```

5. With the help of bag of words, the final result of matrix is now accessible. After initializing a  $N \times M$  matrix, we could verify each item of bag is shown in one certain review using “if a in b” statement. And the corresponding element in the matrix would turn 1 if the result of previous judgement is true. After all the judgement is done, which takes really a long time, we finally get the result.

Some codes are demonstrated below.

```
result=[[0 for col in range(len(newlist))] for row in range(len(bag))]
for i in range(len(bag)):
    for j in range(len(newlist)):
        if newlist[j] in bag[i]:
            result[i][j]=1
```

Have a look at our final result.

Creating the bag of words...

```
(4040L, 12593L)
[[0 0 0 ..., 0 0 0]
 [0 0 0 ..., 0 0 0]
 [0 0 0 ..., 0 0 0]
 ...,
 [0 0 0 ..., 0 0 0]
 [0 0 0 ..., 0 0 0]
 [0 0 0 ..., 0 0 0]]
```



## WEEK 03: PYTHON LIBRARIES

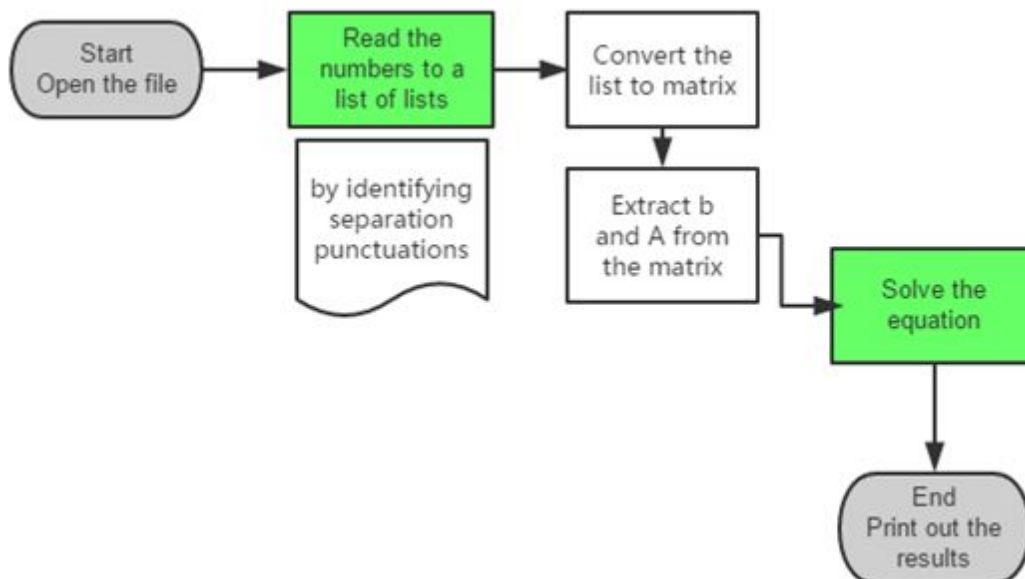
### Exercises:

#### Exercise 3.1 (numpy):

Write a script which reads a matrix from a file like this one and solves the linear matrix equation  $Ax=b$  where  $b$  is the last column of the input-matrix and  $A$  is the other columns. It is okay to use the `solve()`-function from `numpy.linalg`. Does the result make sense?

#### Solution:

In this part, we use jupyter notebook to edit python codes.  
The overall solution flowchart is as below.



1. Firstly, we take a look at the original file. Below is a screenshot of it.

text	0.03 KB
1.	1,2,3,4
2.	6,9,12,7
3.	2,0,9,10

We notice that this matrix is separated by ','. In order to read the matrix from this file, we use the function `map(int,line.split(','))`

Here is the code that shows we have successfully read the matrix.

- **NOTICE:** Now the matrix **mt**'s type is list. We will use **mat()** to convert it later.

```
import numpy
import scipy
myfile = open('C:\\Users\\apple~1\\Desktop\\XAhwshXe.txt') # open the data file
mt = [] # def list mt
mt = [ map(int,line.split(',')) for line in myfile ] # read the numbers into the list
print mt
myfile.close() # close the data file
```

[[1, 2, 3, 4], [6, 9, 12, 7], [2, 0, 9, 10]]

2. Secondly, we want to get **b** and **A**. According to the question, **b** is the last column of the input-matrix(**mt**) and **A** is the other columns.
3. Finally we can solve the linear matrix equation  $Ax=b$  directly using **solve()**-function from **numpy.linalg**.

Below is a screenshot of the rest codes and the result.

```
from numpy import *
mt = mat(mt) # convert list mt to matrix type
b = mt[0:,3:] # get the last column as b
A = mt[0:,:3] # get the rest columns as A
print b
print A
numpy.linalg.solve(A,b) # solve the equation
```

[[ 4]  
[ 7]  
[10]]

[[ 1 2 3]  
[ 6 9 12]  
[ 2 0 9]]

matrix([[-5.09090909],  
[ 1.18181818],  
[ 2.24242424]])

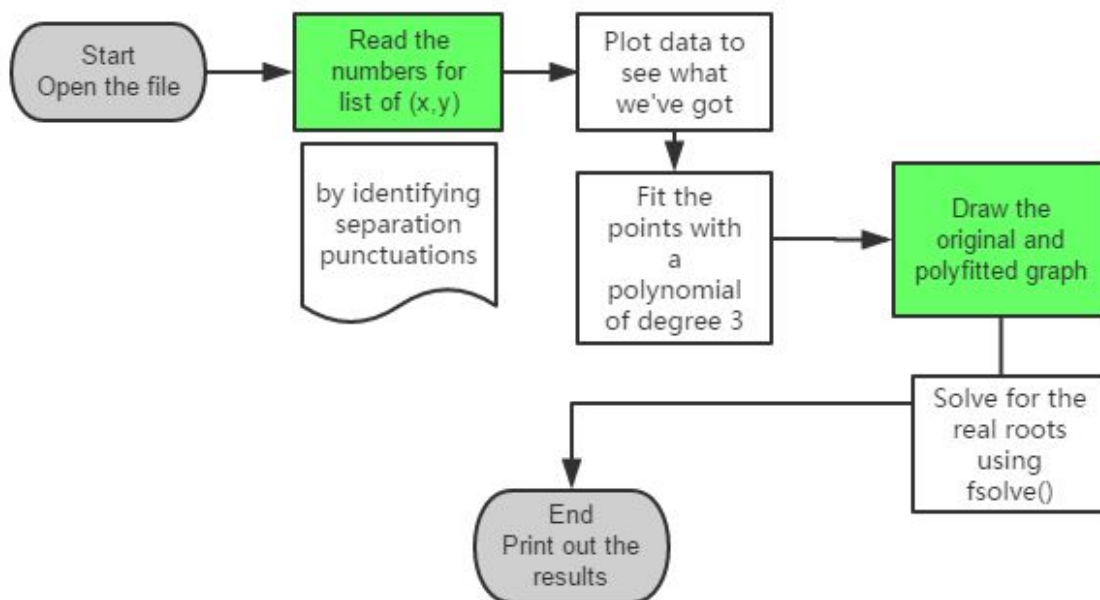
## Exercise 3.2 (scipy):

Write a script that reads in this list of points (x,y), fits/interpolates them with a polynomial of degree 3. Solve for the (real) roots of the polynomial numerically using Scipy's optimization functions (not the root function in Numpy). Does the result make sense (plot something to check).

### Solution:

In this part, we use jupyter notebook to edit python codes.

The overall solution flowchart is as below.



1. Firstly, we take a look at the data file. Below is a screenshot of it, which contains the first 10 points.

text 0.72 KB	
1.	-20 -23555.255109
2.	-19 -20173.3974282
3.	-18 -17132.3837602
4.	-17 -14411.0122397
5.	-16 -11995.5143165
6.	-15 -9866.56079994
7.	-14 -8003.20516776
8.	-13 -6391.33486532
9.	-12 -5011.46272799
10.	-11 -3846.72649659

- After we open the file, the thing we want to do is to load the position data into the points.

According to our preview, we find that each line is a position set of a point.

We use the function `line.strip("\n")` to get each line.

Then we use the function `line.split(' ')` to get the x and y position of each point.

We use `map()` to return a list of x positions and y positions.

Here is a code screenshot that shows we have successfully done the above.

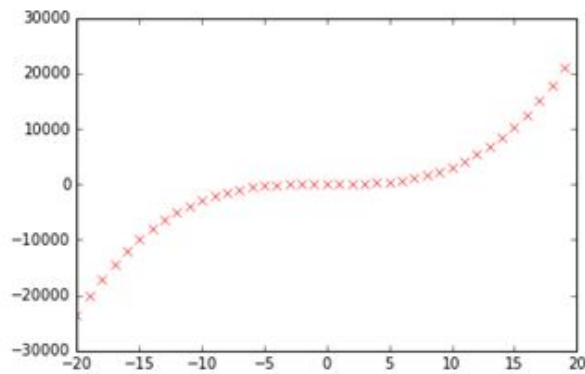
```
import matplotlib.pyplot as plt
mf = open('C:\\Users\\apple\\1\\Desktop\\ENyYffaq.txt')
x = [] # define list x to store the x position of the points
y = [] # define list y to store the y position of the points
for line in mf: # read the positions of the points from the file
    line=line.strip('\n') # each point's position is stored in a line
    trainingSet = line.split(' ') # each point's x and y position is separated with a ' '
    x.append(trainingSet[0]) # a set of x positions
    y.append(trainingSet[1]) # a set of y positions
x= map(int, x) # return a list of x
y= map(float, y) # return a list of y
print x
print y
```

```
import matplotlib.pyplot as plt
mf = open('C:\\Users\\apple\\1\\Desktop\\ENyYffaq.txt')
x = [] # define list x to store the x position of the points
y = [] # define list y to store the y position of the points
for line in mf: # read the positions of the points from the file
    line=line.strip('\n') # each point's position is stored in a line
    trainingSet = line.split(' ') # each point's x and y position is separated with a ' '
    x.append(trainingSet[0]) # a set of x positions
    y.append(trainingSet[1]) # a set of y positions
x= map(int, x) # return a list of x
y= map(float, y) # return a list of y
print x
print y
```

[-20, -19, -18, -17, -16, -15, -14, -13, -12, -11, -10, -9, -8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]  
 [-23555.255109, -20173.3974282, -17132.3837602, -14411.0122397, -11995.5143165, -9866.56079994, -8003.20516776, -6391.33486532, -5011.462727  
 99, -3846.72649659, -2875.44384973, -2084.52334173, -1451.39186881, -962.410182094, -595.527093183, -336.594621263, -104.079980951, -62.3895  
 271886, -11.4248727521, 3.06432478431, 4.2075972256, 6.21613879401, 27.6438528957, 88.1769918966, 204.004215243, 393.974856926, 676.40480070  
 3, 1067.92293554, 1587.79538772, 2254.39823935, 3084.45830648, 4095.6559199, 5307.24683802, 6737.59055266, 8404.59368987, 10323.5165333, 125  
 15.2158242, 14998.1210883, 17788.5813223, 20904.4422828]

- We use self-defined function `plotData()` to see what we've got.

```
%matplotlib inline # to show the printed picture directly inside jupyter notebook
import pylab # to plot data
def plotData(x, y):
    length = len(y)
    pylab.figure(1)
    pylab.plot(x, y, 'rx')
    pylab.xlabel(' ')
    pylab.ylabel(' ')
    pylab.show()
plotData(x,y)
```



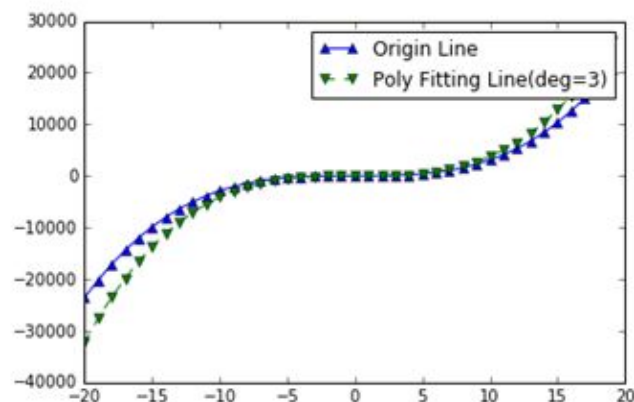
4. We use the function `numpy.polynomial.polynomial.polyfit(x, y, 3)` to fit the points with a polynomial of degree 3.

```
p = numpy.polynomial.polynomial.polyfit(x, y, 3) # fits them with a polynomial of degree 3
pl = numpy.polyld(p) # the estimated parameters
print p
print pl
[ 3.91800201 -2.00736185  1.00106185  2.99999264]
3          2
3.918 x - 2.007 x + 1.001 x + 3
```

5. We draw the original and polyfitted graph in one single graph to compare them.

```
pylab.plot(x, y, 'b^-', label='Origin Line')
pylab.plot(x, pl(x), 'gv--', label='Poly Fitting Line(deg=3)')
pylab.axis()
pylab.legend()
```

<matplotlib.legend.Legend at 0x6f3aa20>



6. Solve for the (real) roots of the polynomial numerically using Scipy's optimization functions `fsolve()`.

```
from scipy.optimize import fsolve

def f(x):
    x0=x.tolist()
    return 3.918*x*x*x-2.007*x*x+1.001*x+3 # the polyfitted 3 degree polynomial
result = fsolve(f, [1,1,1]) # Solve for the real roots of the polynomial
print result                # print the three roots
print f(result)

[-0.69706465 -0.69706465 -0.69706465]
[ 0.  0.  0.]
```

Comparing this result with the graph we plotted before, we come to the conclusion that this result makes sense.



## Exercise 3.3 (pandas):

### TODO 1

Using the movie-lens 1M data and `pandas.read_table` read in all three files (users, ratings, movies) into pandas DataFrames. I recommend giving columns names directly to `read_table` for each case.

Use the data combining tools discussed above to combine these three objects into a single object named `movie_data`

### TODO 2

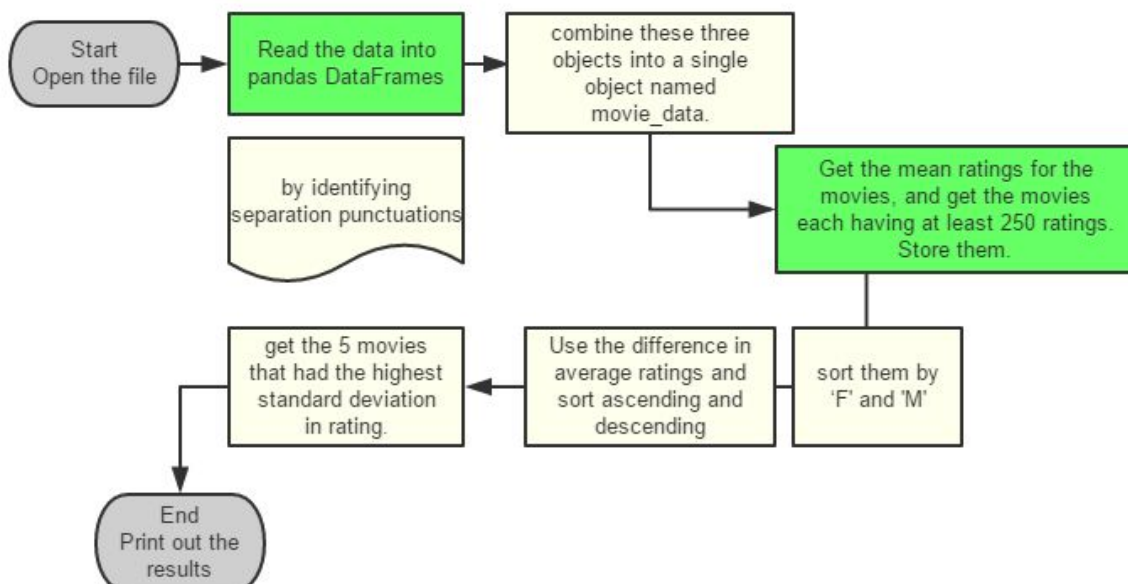
Use the `movie_data` object from the previous exercise and compute the following things:

- The 5 movies with the most number of ratings
- A new object called `active_titles` that is made up of movies each having at least 250 ratings
- For the subset of movies in the `active_titles` list compute the following:
  - The 3 movies with the highest average rating for females. Do the same for males.
  - The 10 movies men liked much more than women and the 10 movies women liked more than men (use the difference in average ratings and sort ascending and descending).
  - The 5 movies that had the highest standard deviation in rating.

### Solution:

In this part, we use jupyter notebook to edit python codes.

The overall solution flowchart is as below.



- After we import pandas as pd, we use `pd.read_table` to read in all three files (users, ratings, movies) into pandas DataFrames.
  - NOTICE:** Remember to initialize the parameter 'names' of each table first. Remember to add "engine='python'".

Then we use `pd.merge` to combine these three objects into a single object named `movie_data`.

  - NOTICE:** `pd.merge` can merge two objects per time, so we use `pd.merge(pd.merge(item1, item2), item3)`

```
import pandas as pd
```

```
unames = ['user_id', 'gender', 'age', 'occupation', 'zip']
users = pd.read_table('C:\\Users\\apple\\1\\Desktop\\ml-lm\\ml-lm\\users.dat', sep='::', header=None, names=unames, engine='python')
rnames = ['user_id', 'movie_id', 'rating', 'timestamp']
ratings = pd.read_table('C:\\Users\\apple\\1\\Desktop\\ml-lm\\ml-lm\\ratings.dat', sep='::', header=None, names=rnames, engine='python')
mnames = ['movie_id', 'title', 'genres']
movies = pd.read_table('C:\\Users\\apple\\1\\Desktop\\ml-lm\\ml-lm\\movies.dat', sep='::', header=None, names=mnames, engine='python')
movie_data = pd.merge(pd.merge(ratings, users), movies)
```

- We get the mean ratings for the movies, get the movies each having at least 250 ratings, and sort them by 'F'.  
So this is the 3 movies with the highest average rating for females.

```
mean_ratings = movie_data.pivot_table('rating', index='title', columns=['gender'], aggfunc='mean')
ratings_by_title = movie_data.groupby('title').size()
# create object "active_titles" that is made up of movies each having at least 250 ratings
active_titles = ratings_by_title.index[ratings_by_title >= 250]
mean_ratings = mean_ratings.ix[active_titles]
# The 3 movies with the highest average rating for females
mean_ratings.sort_values(by='F', ascending=False)[:3]
```

gender	F	M
title		
Close Shave, A (1995)	4.644444	4.473795
Wrong Trousers, The (1993)	4.588235	4.478261
Sunset Blvd. (a.k.a. Sunset Boulevard) (1950)	4.572650	4.464589

Similarly, we get the 3 movies with the highest average rating for males.



```
mean_ratings.sort_values(by='M', ascending=False)[:3]
```

gender	F	M
title		
Godfather, The (1972)	4.314700	4.583333
Seven Samurai (The Magnificent Seven) (Shichinin no samurai) (1954)	4.481132	4.576628
Shawshank Redemption, The (1994)	4.539075	4.560625

- Using the difference in average ratings and sort ascending and descending, we get the 10 movies men liked much more than women and the 10 movies women liked more than men.

```
# define mean_ratings['diff'] to consider the difference between male and female ratings
mean_ratings['diff'] = mean_ratings['M'] - mean_ratings['F']
# with ascending=False, we get the 10 movies men liked much more than women
sort_by_diff = mean_ratings.dropna().sort_values(by='diff', ascending=False)
sort_by_diff[:10]
```

gender	F	M	diff
title			
Good, The Bad and The Ugly, The (1966)	3.494949	4.221300	0.726351
Kentucky Fried Movie, The (1977)	2.878788	3.555147	0.676359
Dumb & Dumber (1994)	2.697987	3.336595	0.638608
Longest Day, The (1962)	3.411765	4.031447	0.619682
Cable Guy, The (1996)	2.250000	2.863787	0.613787
Evil Dead II (Dead By Dawn) (1987)	3.297297	3.909283	0.611985
Hidden, The (1987)	3.137931	3.745098	0.607167
Rocky III (1982)	2.361702	2.943503	0.581801
Caddyshack (1980)	3.396135	3.969737	0.573602
For a Few Dollars More (1965)	3.409091	3.953795	0.544704

```
# define mean_ratings['diff'] to consider the difference between male and female ratings
mean_ratings['diff'] = mean_ratings['M'] - mean_ratings['F']
# with ascending=True, we get the 10 movies men liked much more than women
sort_by_diff = mean_ratings.dropna().sort_values(by='diff', ascending=True)
sort_by_diff[:10]
```

gender	F	M	diff
title			
Dirty Dancing (1987)	3.790378	2.959596	-0.830782
Jumpin' Jack Flash (1986)	3.254717	2.578358	-0.676359
Grease (1978)	3.975265	3.367041	-0.608224
Little Women (1994)	3.870588	3.321739	-0.548849
Steel Magnolias (1989)	3.901734	3.365957	-0.535777
Anastasia (1997)	3.800000	3.281609	-0.518391
Rocky Horror Picture Show, The (1975)	3.673016	3.160131	-0.512885
Color Purple, The (1985)	4.158192	3.659341	-0.498851
Age of Innocence, The (1993)	3.827068	3.339506	-0.487561
Free Willy (1993)	2.921348	2.438776	-0.482573

4. And we use the following code to get the 5 movies that had the highest standard deviation in rating.

```
movie_data.groupby('title')['rating'].std().ix[active_titles].sort_values(ascending=False)[:5]
```

```
title
Dumb & Dumber (1994)      1.321333
Blair Witch Project, The (1999)  1.316368
Natural Born Killers (1994)    1.307198
Tank Girl (1995)           1.277695
Rocky Horror Picture Show, The (1975)  1.260177
Name: rating, dtype: float64
```

### Exercise 3.4 (scikit-learn):

Last week you read in a dataset for [this Kaggle competition](#) and created a bag-of-words representation on the review strings. Train a logistic regression classifier for the competition using your bag-of-words features (and possibly some of the others) to predict the variable “requester\_received\_pizza”. For this exercise, you might want to work a little bit more on your code from last week. Use 90% of the data as training data and 10% as test data.

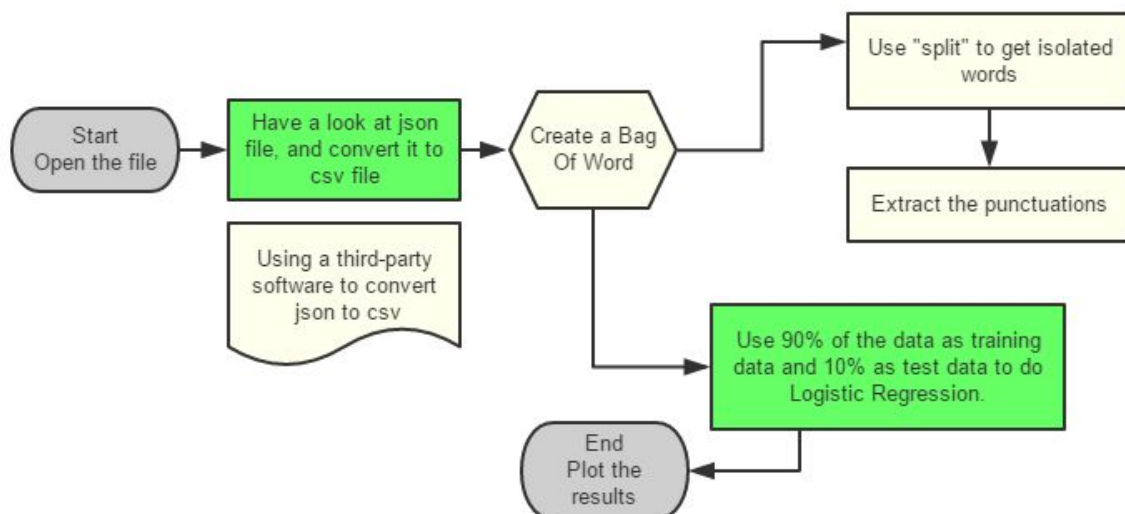
If you don't know anything about machine learning, try to Google a bit and figure out what training and test data is, and how you train a classifier.

How good is your classifier? Discuss the performance of the classifier.

### Solution:

In this part, we use jupyter notebook to edit python codes.

The overall solution flowchart is as below.



1. After we import json, we use `json.load` to load json file, and use `json.dumps` to print it prettily.

```
import json

jsonobject = json.load(file('C:\\Users\\apple\\1\\Documents\\pizza-train.json'))
print json.dumps(jsonobject, sort_keys=True, indent=1)

{
  "request_id": "t3_125dT",
  "request_number_of_comments_at_retrieval": 0,
  "request_text": "Hi I am in need of food for my 4 children we are a military family that has really hit hard times and we have exhausted all means of help just to be able to feed my family and make it through another night is all i ask i know our blessing is coming so whatever u can find in your heart to give is greatly appreciated",
  "request_text_edit_aware": "Hi I am in need of food for my 4 children we are a military family that has really hit hard times and we have exhausted all means of help just to be able to feed my family and make it through another night is all i ask i know our blessing is coming so whatever u can find in your heart to give is greatly appreciated",
  "request_title": "Request Colorado Springs Help Us Please",
  "requester_account_age_in_days_at_request": 0.0,
  "requester_account_age_in_days_at_retrieval": 792.4204050925925,
  "requester_days_since_first_post_on_raop_at_request": 0.0,
  "requester_days_since_first_post_on_raop_at_retrieval": 792.4204050925925,
  "requester_number_of_comments_at_request": 0,
  "requester_number_of_comments_at_retrieval": 0,
  "requester_number_of_comments_in_raop_at_request": 0,
  "requester_number_of_comments_in_raop_at_retrieval": 0,
  "requester_number_of_posts_at_request": 0,
  "requester_number_of_posts_at_retrieval": 1,
}
```

2. We use `len()` and `jsonobject[0].keys()` to have a look at the total length and keys of the json object.

```
len(jsonobject)

4040

jsonobject[0].keys()

[u'requester_number_of_posts_on_raop_at_request',
 u'requester_subreddits_at_request',
 u'requester_number_of_posts_on_raop_at_retrieval',
 u'requester_number_of_comments_at_request',
 u'request_title',
 u'requester_days_since_first_post_on_raop_at_retrieval',
 u'giver_username_if_known',
 u'requester_days_since_first_post_on_raop_at_request',
 u'post_was_edited',
 u'requester_account_age_in_days_at_request',
 u'requester_upvotes_minus_downvotes_at_retrieval',
 u'requester_number_of_posts_at_retrieval',
 u'requester_user_flair',
 u'requester_upvotes_minus_downvotes_at_request',
 u'requester_username',
 u'unix_timestamp_of_request',
 u'requester_upvotes_plus_downvotes_at_request',
 u'unix_timestamp_of_request_utc',
 u'number_of_upvotes_of_request_at_retrieval',
 u'number_of_downvotes_of_request_at_retrieval',
 u'requester_number_of_comments_in_raop_at_retrieval',
 u'request_number_of_comments_at_retrieval',
 u'requester_number_of_posts_at_request',
 u'requester_received_pizza',
 u'requester_number_of_comments_at_retrieval',
 u'request_text',
 u'requester_account_age_in_days_at_retrieval',
 u'requester_upvotes_plus_downvotes_at_retrieval',
 u'request_text_edit_aware',
 u'request_id',
 u'requester_number_of_comments_in_raop_at_request',
 u'requester_number_of_subreddits_at_request']
```



- We use a third-party website to transfer json file into csv file for further implementation. After reading the csv file, we are able to get the request\_text as the following screenshot.

```
import pandas as pd
submissions = pd.read_csv("C:\\Users\\apple\\1\\Desktop\\pizza-train.csv")

print submissions.request_text

0      Hi I am in need of food for my 4 children we a...
1      I spent the last money I had on gas today. Im ...
2      My girlfriend decided it would be a good idea ...
3      It's cold, I'n hungry, and to be completely ho...
4      hey guys:\r\n I love this sub. I think it's gr...
5      Feeling under the weather so I called out off ...
6      We're in Tampa Florida...moving to Ybor on Fri...
7      (Request) I have given a few things on reddit,...
8      Wasnt really sure what to put as the title, un...
9      Austin, Texas\r\n\r\nMy two roommates and I ar...
10     I've been unemployed but working odd jobs. I w...
11     Been a lurker for some time, figured I'd give ...
12     I am a stay at home mom of two kids i live wit...
13     Hey recently moved downtown, but still working...
14     I have never requested a pizza before but I've...
15         You would make me a very happy person.
16     It would be much appreciated, and payed forwar...
17     We are a group of five local community college...
18     -I'm located in Virginia. It would be great to...
19     Been looking for work for 2 weeks since moving...
20     I can't draw.. at all but I'd really love a pi...
21     We're a group of 4 college students, who don't...
22     I hope I'm doing this right. I waited before p...
23     I have an O-Chem midterm in about 3.5 hours (2...
```

- Then we use *split* function to get isolated words.

```
tokenized_texts = []
for item in submissions["request_text"]:
    tokenized_texts.append(str(item).split(" ")) # split the str by spaces to create isolated words
print tokenized_texts

y', 'luck', 'at', 'getting', 'one.', 'My', 'friend', 'who', 'lives', 'an', 'hour', 'away', 'and', 'our', 'schedules', 'do', 'not', 'let',
'us', 'see', 'each', 'other', 'too', 'much.', 'decided', 'to', 'come', 'down', 'and', 'visit', 'me', 'for', 'the', 'night!', 'I', 'woul
d', 'love', 'to', 'be', 'able', 'to', 'be', 'a', 'good', 'host', 'and', 'order', 'her', 'a', 'pizza', 'to', 'go', 'with', 'some', 'beer!\r
\n\r\nAgain, 'no', 'sob', 'story.', 'just', 'looking', 'to', 'share', 'a', 'pizza', 'with', 'an', 'old', 'friend', ':)'], ['Feeling', 'u
nder', 'the', 'weather', 'so', 'I', 'called', 'out', 'off', 'work', 'today!', 'I', 'hate', 'requesting', 'because', 'I', 'feel', 'like',
'I'm', 'begging', 'so', 'I', 'thought', 'I'd', 'give', 'back!', '\r\n\r\n(I'd', 'offer', 'pizza', 'if', 'today', 'were', 'payday', ':
C)'], ['We're', 'in', 'Tampa', 'Florida...moving', 'to', 'Ybor', 'on', 'Friday.', 'My', 'email', 'is', 'urfstuff@yahoo.com.', 'since',
'I'm', 'having', 'major', 'trouble', 'figuring', 'out', 'reddit's', 'message', 'system.\r\n\r\n\r\nMy', 'two', 'roommates', 'and', 'I',
'have', 'found', 'a', 'new', 'place', 'to', 'live', 'and', 'are', 'moving', 'in', 'on', 'Friday.', 'The', 'rent', 'is', '$1100', 'includi
ng', 'utilities', 'and', 'internet.', 'My', '2', 'roomies', 'have', 'very', 'nicely', 'offered', 'for', 'me', 'to', 'just', 'pay', '$200',
'a', 'month', 'until', 'I', 'have', 'a', 'job.', 'I', 'am', 'so', 'thankful', 'to', 'have', 'them', 'as', 'friends!', 'This', 'month', 'a
ll', 'of', 'us', 'are', 'extremely', 'poor', 'from', 'paying', '$2000', 'to', 'landlord', '(first+last', 'rent+deposit).', 'We', 'also',
'had', 'to', 'pay', '$310', 'for', 'TECO', 'to', 'turn', 'on', 'the', 'electricity.\r\n\r\n\r\nI', 'would', 'just', 'really', 'love', 't
o', 'do', 'something', 'nice', 'for', 'them', 'and', 'pick', 'them', 'up', 'a', 'pizza', 'and', 'a', 'sub/prone.', 'My', 'one', 'roomie',
'doesn't', 'love', 'pizza', 'and', 'she', 'only', 'ever', 'gets', 'a', 'sub/prone', 'haha.', 'Using', 'a', 'few', 'coupons', 'I', 'can',
'get', '8', 'pizza', 'rollers', 'I', 'large', 'pizza', 'and', 'a', 'psone', 'for', '$11', 'from', 'Pizza', 'Hut.', 'If', 'anyone', 'ca
n', 'help', 'me', 'out', 'I', 'would', 'be', 'so', 'grateful!', 'Pizza', 'Hut', 'is', 'just', 'the', 'cheapest', 'if', 'you', 'have', 'so
mething', 'else', 'in', 'mind', 'let', 'me', 'know.', 'I', 'have', 'an', 'interview', 'on', 'Thursday', 'so', 'keep', 'your', 'fingers',
'crossed!\r\n\r\n\r\nThe', 'pic', 'for', 'our', 'huge', 'TECO', 'deposit', ':/\r\nhttp://imgur.com/B0uoo'], ['(Request)', 'I', 'have', 'g
```

- We extract the punctuations to create Bag Of Words.

6. This is our Bag Of Words.

7. We put the words that appear more than 5 times and fewer than 100 into 'counts'.

```
for i, item in enumerate(clean_tokenized):
    for token in item:
        if token in unique_tokens:
            counts.iloc[i][token] += 1
```

```
print(len(unique_tokens)) # total number of unique words
```

6543

```
word_counts = counts.sum(axis=0)
counts = counts.loc[:, (word_counts >= 5) & (word_counts <= 100)]
# we consider words that appears more than 5 times and less than 100
# because some words that appear too often may not have so much value to consider (for example, 'and')
# some words appear for only 1 or 2 times is also not so worthy of consideration
print type(counts)
```

<class 'pandas.core.frame.DataFrame'>

8. We use `train_test_split` to use 90% of the data as training data and 10% as test data to do Logistic Regression.

- **NOTICE:** Here counts' type is DataFrame and we need it to be array, so we have to convert it using `np.array(counts)`.

```
from matplotlib import pyplot
import scipy as sp
import numpy as np
from matplotlib import pylab
import sklearn
from sklearn.datasets import load_files
from sklearn.cross_validation import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import precision_recall_curve, roc_curve, auc
from sklearn.metrics import classification_report
from sklearn.linear_model import LogisticRegression
import time

count_vec = TfidfVectorizer(binary=False, decode_error='ignore',
                             stop_words='english')

average = 0
from sklearn.cross_validation import train_test_split
testNum = 10
for i in range(0, testNum):
    X_train, X_test, y_train, y_test = train_test_split(np.array(counts),
                                                         submissions["requester_received_pizza"],
                                                         test_size=0.1, random_state=i)
    clf = sklearn.linear_model.LogisticRegression() # Use 90% of the data as training data and 10% as test data to do Logistic Regression
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    p = np.mean(y_pred == y_test) # The average probability that the y's prediction is correct
    average += p
print(p)
```

0.710396039604

We can see that we get the average probability that the prediction of  $y$  is correct is 0.710396039604. We think it can be further improved with considering more variant data from the original datasets.

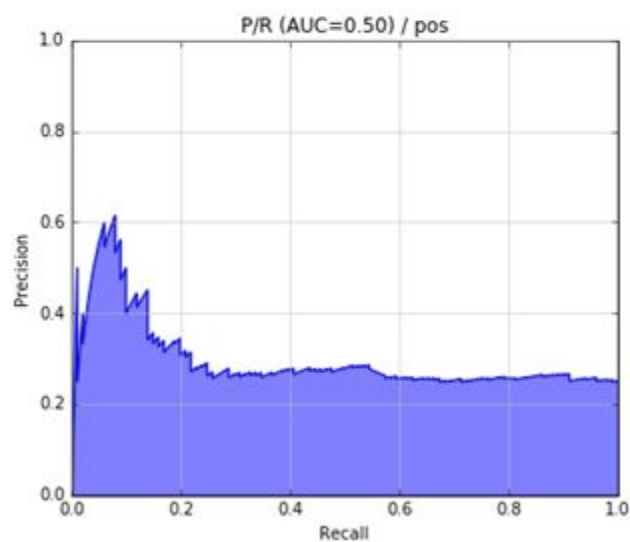
Now we want to plot the results.

```
def plot_pr(auc_score, precision, recall, label=None):
    pylab.figure(num=None, figsize=(6, 5))
    pylab.xlim([0.0, 1.0])
    pylab.ylim([0.0, 1.0])
    pylab.xlabel('Recall')
    pylab.ylabel('Precision')
    pylab.title('P/R (AUC=%0.2f) / %s' % (auc_score, label))
    pylab.fill_between(recall, precision, alpha=0.5)
    pylab.grid(True, linestyle='-', color='0.75')
    pylab.plot(recall, precision, lw=1)
    pylab.show()

answer = clf.predict_proba(X_test)[:,-1]
precision, recall, thresholds = precision_recall_curve(y_test, answer)
report = answer > 0.5
print(classification_report(y_test, report, target_names = ['neg', 'pos']))
print('average precision:', average/len(testNum))
plot_pr(0.5, precision, recall, 'pos')
```

	precision	recall	f1-score	support
neg	0.76	0.89	0.82	303
pos	0.33	0.16	0.21	101
avg / total	0.65	0.71	0.67	404

('average precision:', 0.71039603960396036)



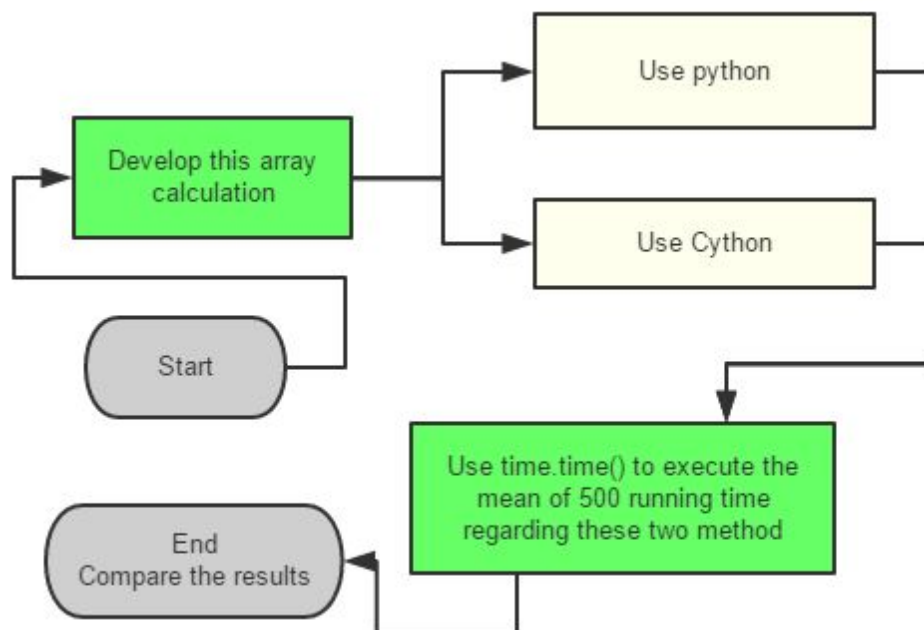


### Exercise 3.5 (cython):

Write a simple Python function for computing the sum  $\frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \dots$  with 10,000 terms (this should be around 1.644), 500 times in a row (to make the execution time measurable). Now compile the code with Cython and see how much speedup you can achieve by this. Remember to declare your variable types.

#### Solution:

In this part, we use jupyter notebook to edit python codes.  
The overall solution flowchart is as below.



#### The python code:

```

def psum(n):
    psum = 0.0
    for a in range(1,n):
        psum = psum+1/float(a*a)
    return psum
psum(10001)
import time
start=time.time()
for i in range(500):
    psum(10001)
finish=time.time()
print (finish-start)/500
  
```

## Screenshot:

```

In [30]: import time
start=time.time()
for i in range(500):
    cysum(10001)
finish=time.time()
print (finish-start)/500
4.07438278198e-05

In [38]: def psum(n):
    psun = 0.0
    for a in range(1,n):
        psun = psun+1/float(a*a)
    return psun

In [39]: psum(10001)
Out[39]: 1.6448340718480652

In [40]: import time
start=time.time()
for i in range(500):
    psum(10001)
finish=time.time()
print (finish-start)/500
0.00893171644211

```

## The Cython code:

load\_extcython

%%cython -a

defcysum(int n):

cdefint a

cdef float sum

sum = 0

for a in range(1,n):

sum = sum+1/float(a\*a)

return sum

import time

start=time.time()

for i in range(500):

cysum(10001)

finish=time.time()

print (finish-start)/500

## Screenshot:

The screenshot shows a Jupyter Notebook with the following content:

```
In [1]: load_ext cython
```

```
In [29]: %%cython -a
def cysum(int n):
    cdef int a
    cdef float sum
    sum = 0
    for a in range(1,n):
        sum = sum+1/float(a*a)
    return sum
```

Out[29]: Generated by Cython 0.24

Yellow lines hint at Python interaction.  
Click on a line that starts with a '+' to see the C code that Cython generated for it.

```
+1: def cysum(int n):
2:     cdef int a
3:     cdef float sum
+4:     sum = 0
+5:     for a in range(1,n):
6:         sum = sum+1/float(a*a)
+7:     return sum
    __pyx_XDECREF(__pyx_r);
    __pyx_t_3 = PyFloat_FromDouble(__pyx_v_sum); if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 7, __pyx_L1_error)
    __pyx_GOTREF(__pyx_t_3);
    __pyx_r = __pyx_t_3;
    __pyx_t_3 = 0;
    goto __pyx_L0;
```

From the output we can see that the python time is 0.00893, while the Cython time decreases to 4.07438e-05.

This is about 200 times faster.