Python

Why Python?

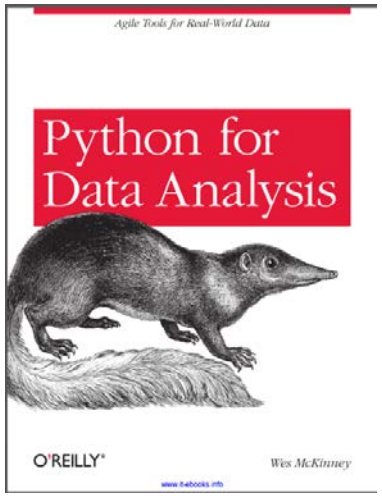
- A high level language, easy to read and write.
- Free and open source
- Widely used in the industry
  - Large community, existing programs and modules
  - LLM coding support works well
- Can be used for any type of analysis.

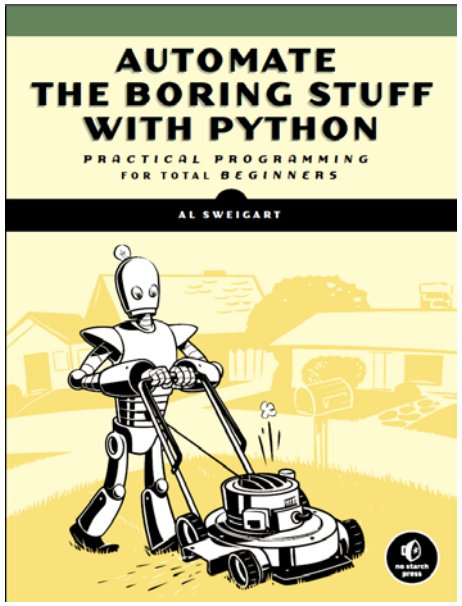
Why Python?

- Can be used for any type of analysis.
  - Data analysis and regressions, ArcGIS, Network analysis
  - But for these tasks we have more specialized tools.

- Frontline development
  - ML

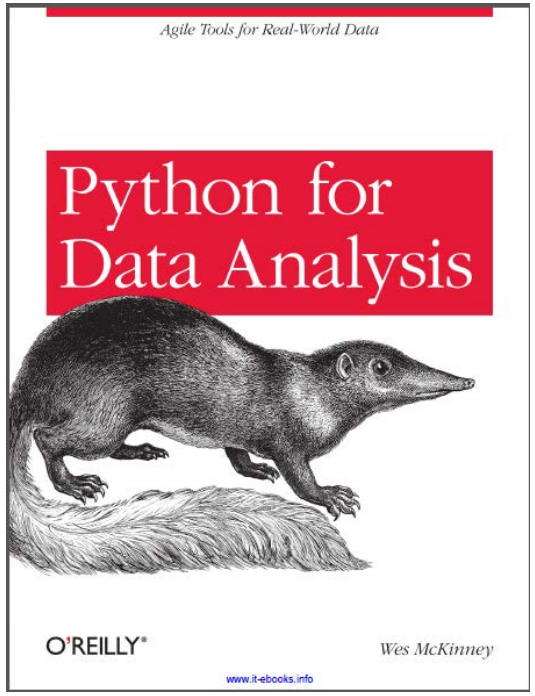
- I use Python to automate boring things.
  - Reformatting and doing simple editing of text files.
  - Merging thousands of files into a one data file.
  - Splitting files that are too long, into files of given size.
  - Webscraping
    - Downloading hundreds of files from the web.
    - Extracting information from those files.
  - …

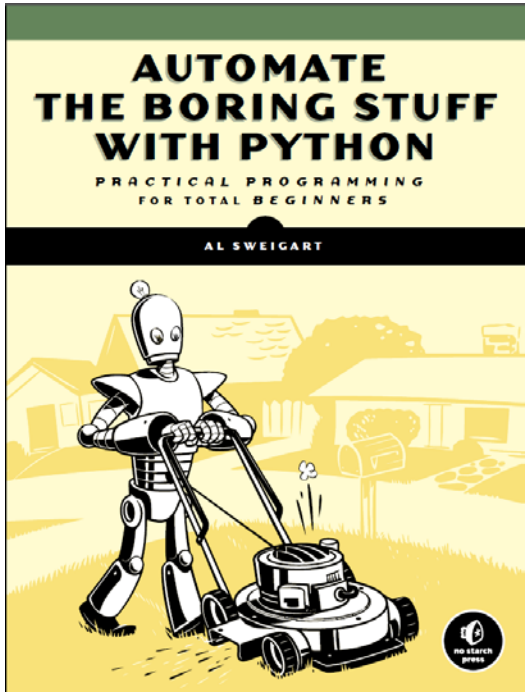




Useful references

[Python for economists]





Topics

1. Getting started: download and installation.
2. Basics
   - Data types
   - Syntax
   - Conditional statements and iteration
   - Functions and exception handling
   - Reading and writing files.
   - System commands and file handling.
3. Working with text
   - Strings and lists.
   - Regular expressions.
   - Dictionaries to count words.
4. Program execution
   - Python program arguments.
   - Interaction with Stata.
5. Next lectures: Neural nets and Web scraping

1. Getting started

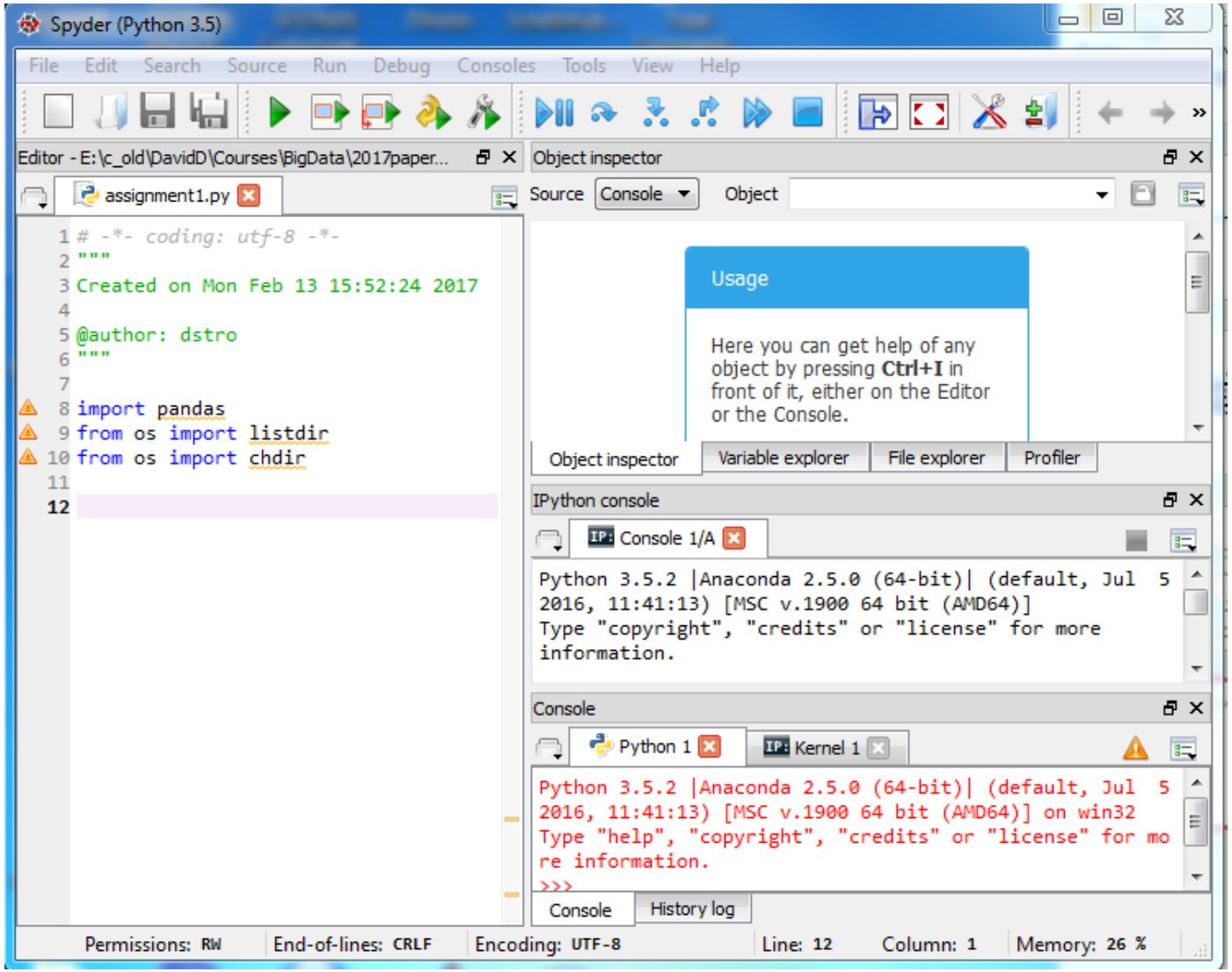
Versions

- **Python version 3.x is NOT** backward compatible with **Python version 2.x**
  - Deliberate choice to handle characters for more languages than just English.
- Many programmers and organisations still use Python ***ver 2.x***
  - Syntax for printing is simple: `print "Hello World!“`
  - `9/5 = 1`
  - `9/5.0 = 1.8`
- In this course where text handling is an important core technique we will only use **version 3.x.**
  - Syntax for printing is still simple: `print("Hello World!")`
  - `9/5 = 1.8`
  - `9/5.0 = 1.8`

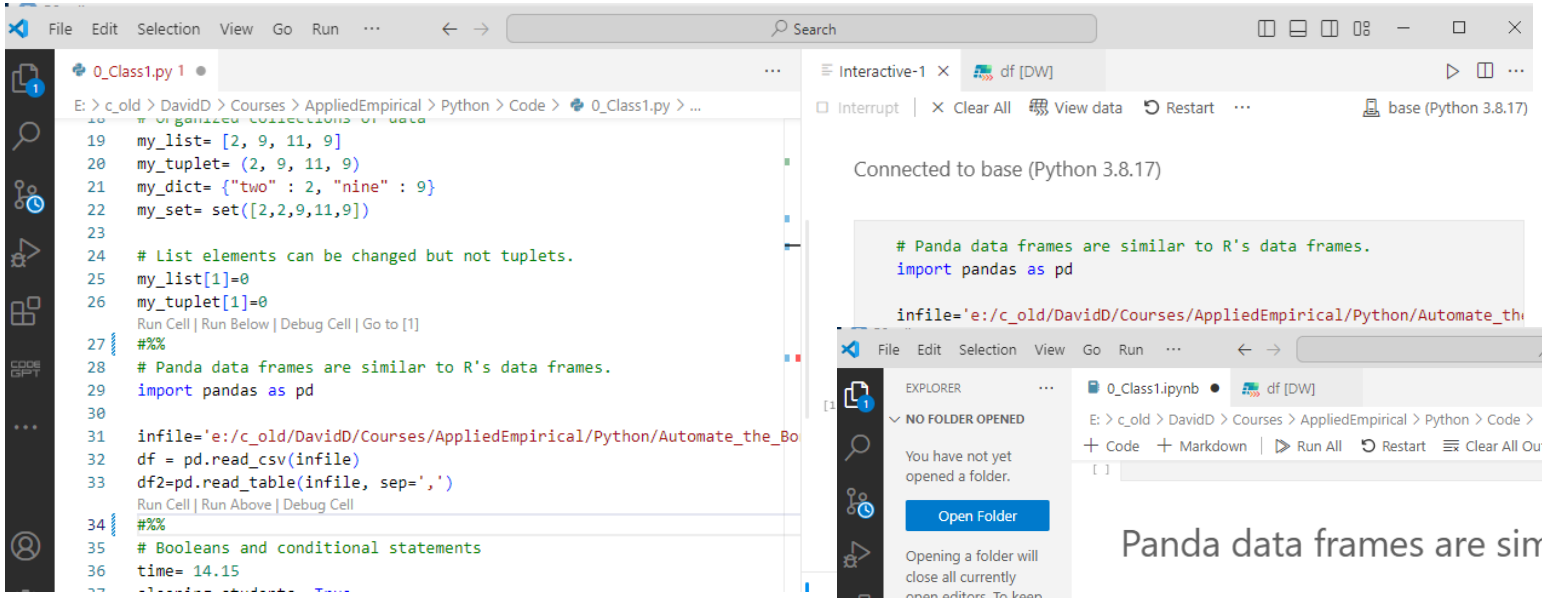
Download

- Download the Anaconda Python distribution at https://www.anaconda.com/download/

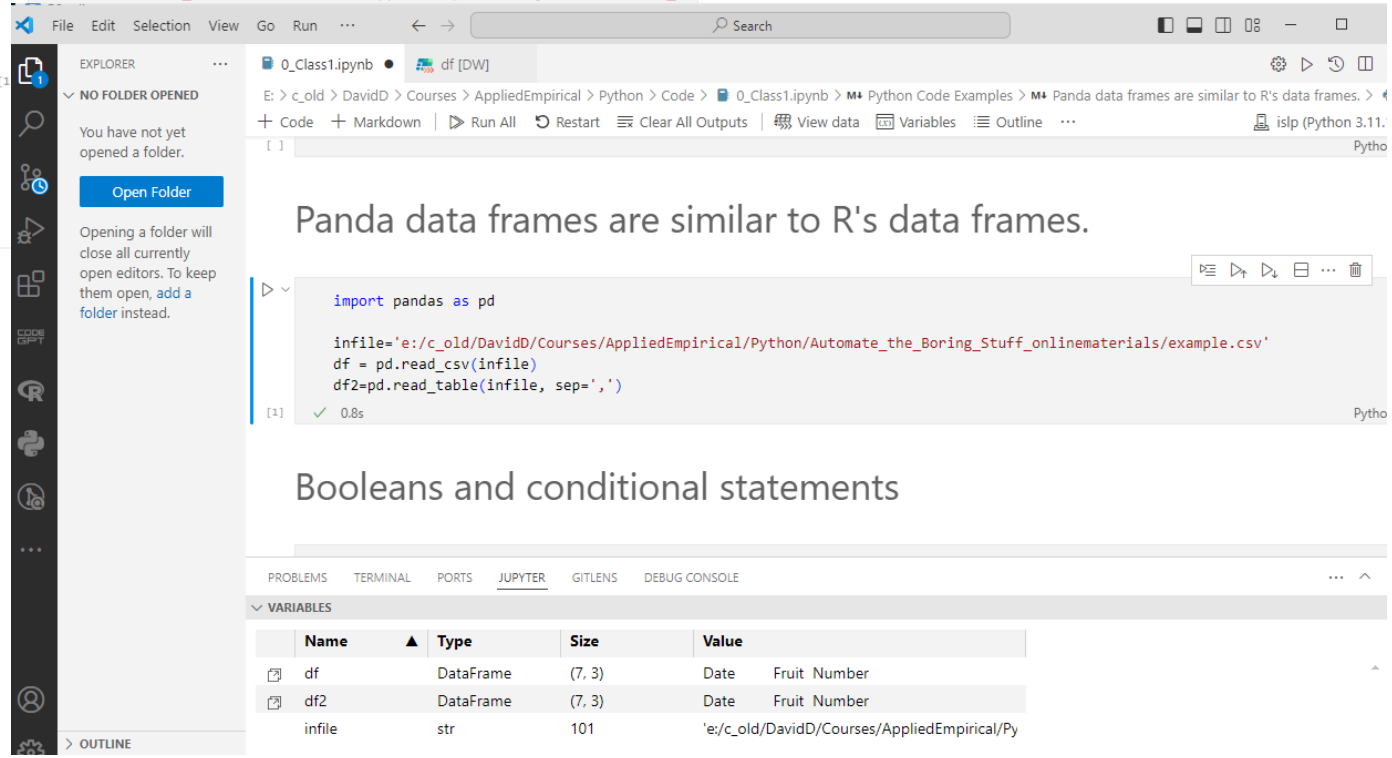
Work in Spyder (included in Anaconda)



Or VS Code

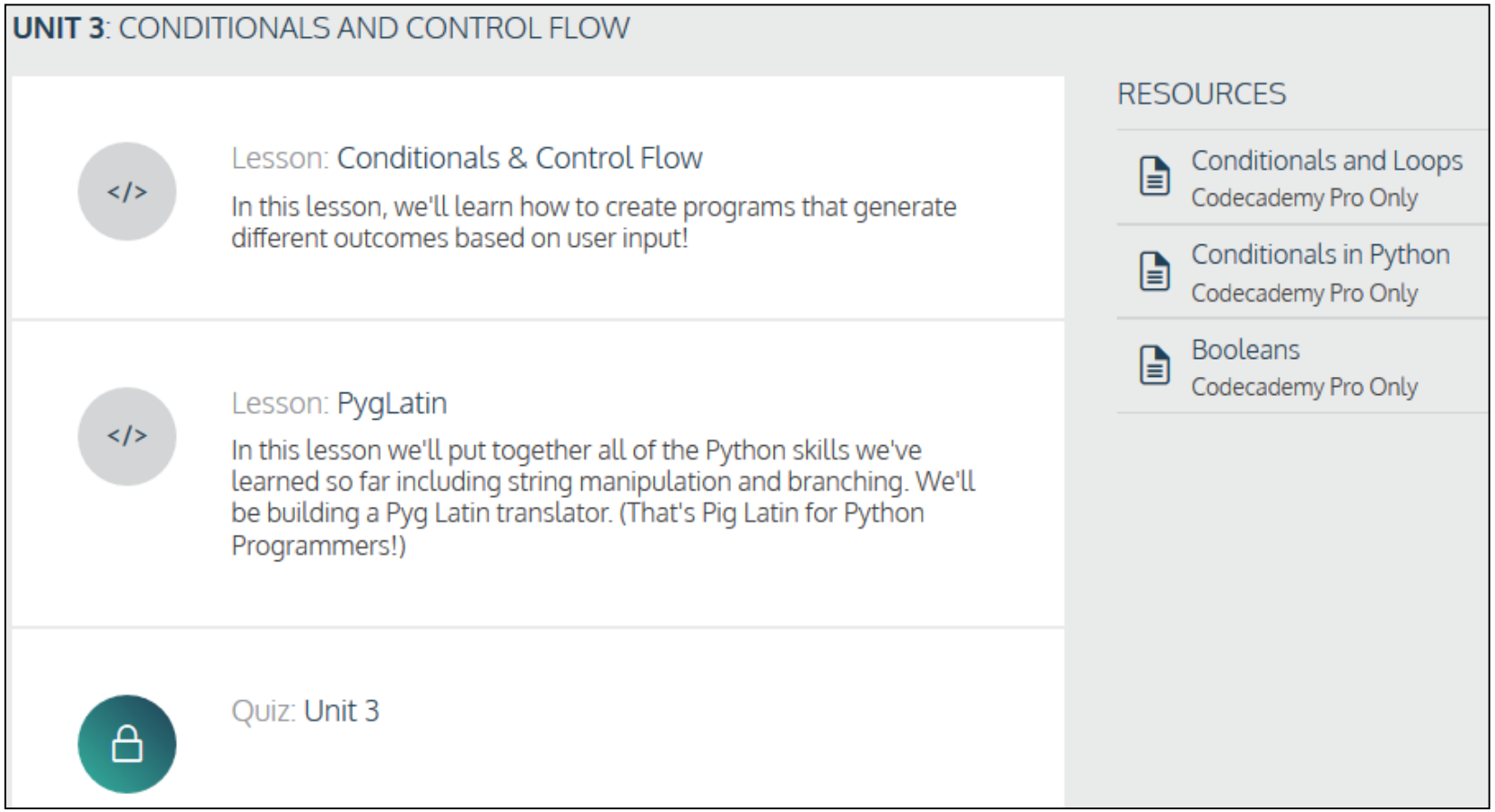

File Edit Selection View Go Run
Search
0_Class1.py 1
E: > c_old > DavidD > Courses > AppliedEmpirical > Python > Code > 0_Class1.py > ...
my_list= [2, 9, 11, 9]
my_tuplet= (2, 9, 11, 9)
my_dict= {"two" : 2, "nine" : 9}
my_set= set([2,2,9,11,9])
# List elements can be changed but not tuplets.
my_list[1]=0
my_tuplet[1]=0
Run Cell | Run Below | Debug Cell | Go to [1]
#%%
# Panda data frames are similar to R's data frames.
import pandas as pd
infile='e:/c_old/DavidD/Courses/AppliedEmpirical/Python/Automate_the_Bo
df = pd.read_csv(infile)
df2=pd.read_table(infile, sep=',')
Run Cell | Run Above | Debug Cell
#%%
# Booleans and conditional statements
time= 14.15

Interactive-1
df [DW]
Interrupt
Clear All
View data
Restart
base (Python 3.8.17)
Connected to base (Python 3.8.17)
# Panda data frames are similar to R's data frames.
import pandas as pd
infile='e:/c_old/DavidD/Courses/AppliedEmpirical/Python/Automate_th


File Edit Selection View Go Run
Search
EXPLORER
NO FOLDER OPENED
You have not yet opened a folder.
Open Folder
Opening a folder will close all currently open editors. To keep them open, add a folder instead.
0_Class1.ipynb
df [DW]
E: > c_old > DavidD > Courses > AppliedEmpirical > Python > Code > 0_Class1.ipynb > Python Code Examples > Panda data frames are similar to R's data frames.
Code
Markdown
Run All
Restart
Clear All Outputs
View data
Variables
Outline
islp (Python 3.11.
Panda data frames are similar to R's data frames.
import pandas as pd
infile='e:/c_old/DavidD/Courses/AppliedEmpirical/Python/Automate_the_Boring_Stuff_onlinematerials/example.csv'
df = pd.read_csv(infile)
df2=pd.read_table(infile, sep=',')
0.8s
Booleans and conditional statements
PROBLEMS
TERMINAL
PORTS
JUPYTER
GITLENS
DEBUG CONSOLE
VARIABLES
Name
Type
Size
Value
df
DataFrame
(7, 3)
Date Fruit Number
df2
DataFrame
(7, 3)
Date Fruit Number
infile
str
101
'e:/c_old/DavidD/Courses/AppliedEmpirical/Py
OUTLINE

Codecademy – your online learning tool

- https://www.codecademy.com/learn/learn-python-3



2. Basics

Basic data types in Python

- Important basic data types are:
  - Integers (int)
  - Floating point numbers (float)
  - String (str)
  - Boolean variables (TRUE, FALSE)
  - None (absence of a value)
- Functions converting data types
  - str(), int(), float()

```
>>> str(0)
'0'
>>> str(-3.14)
'-3.14'
>>> int('42')
42
>>> int('-99')
-99
>>> int(1.25)
1
>>> int(1.99)
1
>>> float('3.14')
3.14
>>> float(10)
10.0
```

Organized collections of data

- Lists my_list = [2, 9, 11, 9]
- Tuplets my_tuplet = (2, 9, 11, 9)
- Dictionaries my_dict = {"two" : 2, "nine" : 9" …
- Sets my_set = set([2,9,11,9])
- Pandas df = pd.read_csv(infile)
  df = pd.read_stata(infile) : reads infile.dta

Note bracket associated with each type.

Tuplets cannot be changed.

Sets are unordered.

Some important operators

- ***Aritmetic***: **+ - * / ****
- ***Assignment***: **=**
- ***Relational*** : **== != < > <= >=**
- **Boolean: or and**

```
time = 14.47
sleeping_students = True
if time > 14.45 and sleeping_students:
    print("Time to take a break")
else:
    print("More Python!!")
```

Augmented Assignment Operators

Augmented assignment statement	Equivalent assignment statement
`spam = spam + 1`	`spam += 1`
`spam = spam - 1`	`spam -= 1`
`spam = spam * 1`	`spam *= 1`
`spam = spam / 1`	`spam /= 1`

Basics

Python	Stata
os.getcwd() # Returns the working directory (wd)	pwd
os.chdir("C:/myfolder/data")	cd c:\myfolder\data
at command prompt: pip install SomePackage # Install the package on your computer	ssc install abc
import SomePackage # Load the SomePackage to your workspace	

- Modules can be imported:
  - `import os`
  - `import sys, random`
- The os. in front of getwd() tells Python to look in the os module for this function.

Macros

replace	Python	Stata
scalars	myNumber = 10 print(myNumber)	scalar x = 10 di “`x’”
strings	myString = "Hello , World!" print(myString)	local x "Hello , World!" di “`x’”

Syntax: indentation

- Python provides no braces to indicate blocks of code for class and function definitions or flow control.
- Blocks of code are denoted by line indentation, all statements within the block must be indented the same amount.
- This works

```
18  if True:
19      print "True"
20  else:
21    print "False"
```

- This generates an error

```
25  if True:
26      print "Answer"
27      print "True"
28  else:
29      print "Answer"
30    print "False"
```

Syntax: multi-line statements

- Statements in Python typically end with a new line.
- Python does, however, allow the use of the line continuation character (\) to denote that the line should continue. For example

```
total = item_one + \
        item_two + \
        item_three
```

- Statements contained within the [], {}, or () brackets do not need to use the line continuation character. For example -

```
days = ['Monday', 'Tuesday', 'Wednesday',
        'Thursday', 'Friday']
```

Syntax

- Multiple Statements on a single Line
  - The semicolon (;) allows multiple statements on the single line given that neither statement starts a new code block, e.g.,

```
import sys; x = 'foo'; sys.stdout.write(x + '\n')
```

- Multiple statement groups as suites
  - Compound or complex statements, such as if, while, def, and class require a header line and a suite.
  - Header lines begin the statement (with the keyword) and terminate with a colon (:) and are followed by one or more lines which make up the suite.

```
if expression :
    suite
elif expression :
    suite
else :
    suite
```

Syntax: the dot notation

- Like Stata, Python uses print(filename)
  to call the function print with the argument filename.
- Unlike Stata, Python uses string.lower() to convert string to lower).

```
text="Hello World!"

text.lower()
'hello world!'
```

```
. local text "Hello World!"

. di strlower("`text'")
hello world!
```

- Object oriented programming
  - text is an *instance* of the *class* string.
  - lower() is a *method* of the string class. Methods are functions defined for a given class/object/instance.
  - Instances can also have attributes, variables defined for class referenced without ().

Syntax: the dot notation

- use dir to list all functions of an object

```
In [5]: dir(text)
Out[5]:
['__add__',
 '__class__',
 '__contains__',
 '__delattr__',
 '__dir__',
 '__doc__',
```

```
'isalnum',
'isalpha',
'isdecimal',
'isdigit',
'isidentifier',
'islower',
'isnumeric',
'isprintable',
'isspace',
'istitle',
'isupper',
'join',
'ljust',
'lower',
'lstrip',
'maketrans',
'partition',
'replace',
```

```
In [8]: text.isalpha()
Out[8]: False
```

Syntax

- Quotation
  - Python accepts single (‘), double (") and triple (''' or """) quotes to denote string literals, as long as the same type of quote starts and ends the string.
- Comments: hash sign (#) used similarly as * in Stata.

Conditional statements: if else

```
my_variabel = False
if not my_variabel:
    print("Not False“)
else:
    print("False")

x = 0
y = 1
if x == y:
    print("x == y“)
else:
    print("x != y“)
```

Iteration

Use **while** and **Boolean conditions,**

or for, with **range()** as a **Boolean condition**

```
time = 14.14
while time < 14.55:
    print("More Python!!")
    time += 0.10

for i in range(0,5):
    print("More Python!!!")
```

```
range(start,stop[,step])
```

Iteration with break and continue

while with **break** (see also **continue** and **sys.exit()**)

```
while True:
    print('Please type your name.')
    name = input()
    if name== 'your name':
        break
print('Thank you!')
```

- *You can use continue and break statements only inside while and for loops.*
- If break statement is inside a nested loop (loop inside another loop), break will terminate the innermost loop.
- The continue statement is used to skip the rest of the code inside a loop for the current iteration only. Loop does not terminate but continues on with the next iteration.

Functions

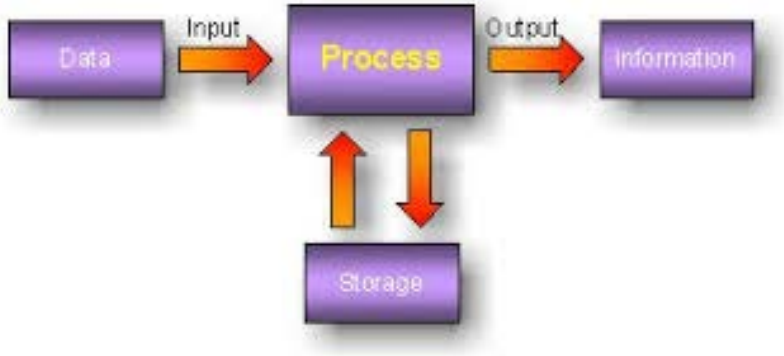
```
1 import random
2 def getAnswer(answerNumber):
3       if answerNumber == 1:
4             return 'It is certain'
5       elif answerNumber == 2:
6             return 'Very doubtful'
7 r = random.randint(1, 2)
8 fortune = getAnswer(r)
9 print(fortune)
```

- def defines function
- arguments passed through (var)
- return values by return statement

- variables in functions are local
- global variables can be accessed through the global command

Recall from lecture 2

- Decompose programs into functions
  - A function is a reusable section of software
    that can be treated as a black box by the rest of the program.
  - Functions should not
    - be more than 1 page (about 60 lines) long,
    - use more than 5 or 6 input parameters,
    - reference outside information.



Exception handling:

```
try:
    lucky_number = int(input("Hello, enter …
    print("Your lucky number is:", lucky_number)
except ValueError:
        print("Sorry, integers only :-(")
```

Exception handling

- You don't want your program to crash when there is a problem with one of a thousand files that you download.
- Errors can be handled with try and except statements.
  - The code that could potentially have an error is put in a try clause.
  - The program execution moves to the start of a following except clause if an error happens.

```
# Exceptions
def div42(divideBy):
    return 42 / divideBy

def div42e(divideBy):
    try:
        return 42 / divideBy
    except ZeroDivisionError:
        print('Error: Invalid argument.')
    print("Program continues")
```

```
In [11]: div42(0)
Traceback (most recent call last):

  File "<ipython-input-11-9345e6c26e48>", line 1,
    div42(0)

  File "<ipython-input-7-66b937553f06>", line 2,
    return 42 / divideBy

ZeroDivisionError: division by zero


In [12]: div42e(0)
Error: Invalid argument.
Program  continues
```

Reading and writing plain text files

1. Call the open() function to return a File object.
2. Call the read() or write() method on the File object.
3. Close the file by calling the close() method on the File object.

```
in_file = open('105-abraham-mi.txt',"r")
text = in_file.read()
in_file.close()

out_file = open('105-abraham-mi2.txt',"w")
out_file.write(text)
out_file.close()
```

105-abraham-mi2.txt

```
<DOC>
<DOCNO>105-abraham-mi-1-19981112</DOCNO>
<TEXT>
 Mr. ABRAHAM. Mr. President, during debate on final
 Competitiveness and Workforce Improvement Act was
 number of documents printed in the Record. These i
 negotiations, whose inclusion I was seeking to hel
 of the key points about these documents is the cha
 that I submitted, these changes were marked by red
 only version I had, which was a copy of a fax, the
 unintelligible, resulting in the printing of a gar
 Accordingly, I ask that the corrected version of t
 Record of the 105th Congress. On the copy of the S
 July 30 version but was deleted in the September 1
 July 30 version and was added in the September 14
</TEXT>
</DOC>

<DOC>
<DOCNO>105-abraham-mi-1-19981112</DOCNO>
<TEXT>
```

Read and write line by line

1. Call the open() function to return a File object.
2. Iterate “for line in in_file:”
3. Close the file by calling the close() method on the File object.

```
in_file = open('105-abraham-mi.txt',"r")
out_file = open('105-abraham-mi2.txt',"w")
for line in in_file:
    wordlist = line.split()
    for word in wordlist:
        out_file.write(word + "\n")
in_file.close()
out_file.close()
```

read_write.py | 105-abraham-mi2.txt

```
1  <DOC>
2  <DOCNO>105-abraham-mi-
3  <TEXT>
4  Mr.
5  ABRAHAM.
6  Mr.
7  President,
8  during
9  debate
```

Read and write line by line

1. Call the open() function to return a File object.
2. Use readline() method.
3. Close the file by calling the close() method on the File object.

```
in_file = open('105-abraham-mi.txt',"r")
out_file = open('105-abraham-mi2.txt',"w")
line = in_file.readline()
while line:
      wordlist = line.split()
      for word in wordlist:
            out_file.write(word + "\n")
      line = in_file.readline()
in_file.close()
out_file.close()
```

read_write.py | 105-abraham-mi2.txt

```
<DOC>
<DOCNO>105-abraham-mi-
<TEXT>
Mr.
ABRAHAM.
Mr.
President,
during
debate
```

readline()

- Execute line = in_file.readline() several times. What happens to the line variable?

```
In [68]: dir(in_file)
Out[68]:
['_CHUNK_SIZE',
 'buffer',
 'close',
 'closed',
 'detach',
 'encoding',
 'errors',
 'fileno',
 'flush',
 'isatty',
 'line_buffering',
 'mode',
 'name',
 'newlines',
 'read',
 'readable',
 'readline',
 'readlines',
 'seek',
 'seekable',
 'tell',
 'truncate',
 'writable',
 'write',
 'writelines']
```

Read/write data files with exception handling

```
def read_and_extract_numeric_values():
    file_data = []
    try:
        csv_file = open(file_name,"r")
        for line in csv_file:
            file_data.extend(line.split(","))
    except IOError:
```

Reading pdf files

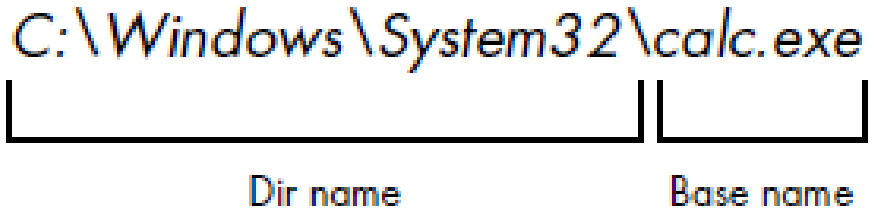
```
import PyPDF2, os
os.chdir('E:/c_old/DavidD/Courses/AppliedEmpirical/Python/Examples')
pdfFileObj = open('JEP.pdf', 'rb')
pdfReader = PyPDF2.PdfFileReader(pdfFileObj)
pdfReader.numPages
pageObj = pdfReader.getPage(0)
pageObj.extractText()
```

Around midnight on March 29, 2014, some Chinese internet night owls noticed that the hazard factor of P-Xylene (PX) had been changed from "low" to "high" on Baidu Encyclopedia—the Chinese equivalent to Wikipedia. The

```
Out[34]: 'Journal of Economic PerspectivesŠVolume 31, Number 1ŠWinter 2017ŠPages
117Œ140\nAround midnight on March 29, 2014, some Chinese internet night owls
noticed \nthat the hazard factor of P-Xylene (PX) had been changed from flowfl to
\nfihighfl on Baidu EncyclopediaŠthe Chinese equivalent to  Wikipedia. The \nnext
morning, hundreds of protestors assembled in MaomingŠa city in southern \nChina™s
industrial heartlandŠwhere a large-scale PX plant was planned. At 8:38 am, a
\nmessage with pictures of the protest was posted on Sina WeiboŠthe Chinese
```

Useful os commands

- os.getcwd()
- os.chdir(*path*)
- os.makedirs(*dir*)
- os.listdir(dir)
- os.path.getsize(*filename*)
- os.path.basename(*path*)
- os.path.dirname(*path*)



Useful os commands

- os.listdir(dir)

```python
def main():
    chdir(in_dir)
    my_files = listdir(in_dir)
    for file_name in my_files:
        print(file_name)
        out_file="csv_" + file_name
        out_handle = open(out_file,"w")
        text_file = open(file_name,"r", encoding='latin-1')
        for line in text_file:
            wordlist = line.split()
            for word in wordlist:
                out_handle.write(word + "\n")
        text_file.close()
        out_handle.close()
if __name__ == "__main__":
    main()
```

105-abraham-mi.txt

```
<DOC>
<DOCNO>105-abraham-mi-1-19981112</DOCNO>
<TEXT>
 Mr. ABRAHAM. Mr. President, during debate on final passage of the Omnibus Appropriations
 bill, in which the American Competitiveness and Workforce Improvement Act was included
 as Title IV of Subdivision C, I asked unanimous consent to have a number of documents
 printed in the Record. These included two documents I received from the Administration
 during the negotiations, whose inclusion I was seeking to help illuminate the meaning of
```

105-abraham-mi.txt | csv_105-abraham-mi.txt

```
<DOC>
<DOCNO>105-abraham-mi-1-19981112</DOCNO>
<TEXT>
Mr.
ABRAHAM.
Mr.
President,
during
debate
on
final
passage
```

Aside: why def main()?

- When Python interpreter reads a source file, it will execute all the code found in it.
- When Python runs the "source file" as the main program, it sets the special variable (__name__) to have a value ("__main__").
- When you execute the main function, it will then read the "if" statement and checks whether __name__ does equal to __main__.
- In Python **"if__name__== "__main__"** allows you to run the Python files either as **reusable modules or standalone programs**.
  - import: __name__= module's filename
    if statement==false, and the script in __main__ will not be executed
  - direct run:__name__=__main__
    if statement == True, and the script in _main_will be executed

Useful os commands

- os.walk()

```
import os
for folderName, subfolders, filenames in os.walk('C:\\delicious'):
    print('The current folder is ' + folderName)
    for subfolder in subfolders:
        print('SUBFOLDER OF ' + folderName + ': ' + subfolder)
    for filename in filenames:
        print('FILE INSIDE ' + folderName + ': '+ filename)
    print('')
```

```
The current folder is C:\delicious
SUBFOLDER OF C:\delicious: cats
SUBFOLDER OF C:\delicious: walnut
FILE INSIDE C:\delicious: spam.txt

The current folder is C:\delicious\cats
FILE INSIDE C:\delicious\cats: catnames.txt
FILE INSIDE C:\delicious\cats: zophie.jpg

The current folder is C:\delicious\walnut
SUBFOLDER OF C:\delicious\walnut: waffles

The current folder is C:\delicious\walnut\waffles
FILE INSIDE C:\delicious\walnut\waffles: butter.txt.
```

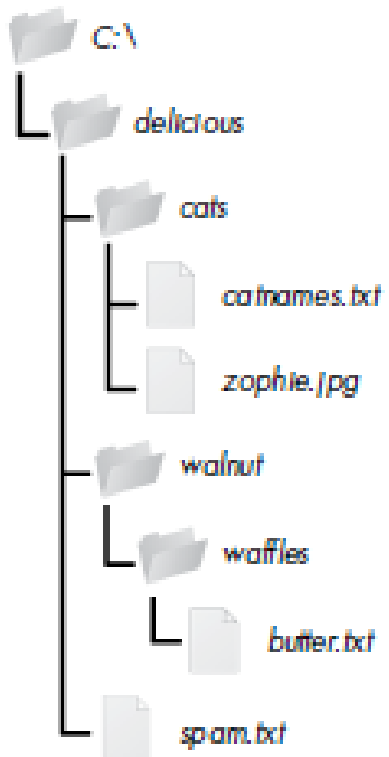


Figure 9-1: An example folder that contains three folders and four files

File management

- shutil module (shell utility), cmp shell command in Stata.
  - shutil.copy(*source*, *destination*)
  - shutil.copytree(*source*, *destination*) will copy the folder at the path *source*, along with all of its files and subfolders, to the folder at the path *destination*.
  - shutil.move(*source*, *destination*) NB overwrites existing files
- os
  - os.unlink(*path*) will delete the file at *path*
  - os.rmdir(*path*) will delete the folder at *path*
  - shutil.rmtree(path) will remove the folder at path,
    and all files and folders it contains will also be deleted.
- send2trash
  - send2trash.send2trash(*path*)
    safer deletes

File management

- shutil.copy(*source*, *destination*)

```
>>> import shutil, os
>>> os.chdir('C:\\')
❶ >>> shutil.copy('C:\\spam.txt', 'C:\\delicious')
'C:\\delicious\\spam.txt'
❷ >>> shutil.copy('eggs.txt', 'C:\\delicious\\eggs2.txt')
'C:\\delicious\\eggs2.txt'
```

- shutil.copytree(*source*, *destination*)

```
>>> shutil.copytree('C:\\bacon', 'C:\\bacon_backup')
'C:\\bacon_backup'
```

- shutil.move(*source*, *destination*)
  NB overwrites existing files

```
>>> import shutil
>>> shutil.move('C:\\bacon.txt', 'C:\\eggs')
'C:\\eggs\\bacon.txt'
```

```
>>> shutil.move('C:\\bacon.txt', 'C:\\eggs')
'C:\\eggs'
```

Reading and writing zipfiles

- Extracting

```
>>> import zipfile, os
>>> os.chdir('C:\\')    # move to the folder with example.zip
>>> exampleZip = zipfile.ZipFile('example.zip')
❶ >>> exampleZip.extractall()
>>> exampleZip.close()
```

- Writing
  - Open in write mode ‘w’
  - compress_type = ZIP_DEFLATED works well on all types of data

```
>>> newZip = zipfile.ZipFile('new.zip', 'w')
>>> newZip.write('spam.txt', compress_type=zipfile.ZIP_DEFLATED)
>>> newZip.close()
```

Example: reading huge zip file by chunk

- Stata cannot read csv file, since it is too large.
- Python: open zip file, read 10,000 rows in chunks, do something, and save.

```
file = 'tls207_part01.zip'
z = zipfile.ZipFile(file)
reader = pd.read_csv(z.open('tls207_part01.csv'), chunksize=10000, encoding='utf-8')
first=1
for chunk in reader:
    x=pd.merge(chunk, pid_no, on='person_id'    , how='inner')
    if first==1:
        tls207_NO=x
        first=0
    else:
        tls207_NO=tls207_NO.append(x)

tls207_NO.to_csv('z:/P2/Log/Issues/Patstat_NO/Data/tls207_NO.csv')
```

3. Working with strings

Strings and Lists – to work with indexing

H	e	l	l	o
0	1	2	3	4
-5	-4	-3	-2	-1

- Hello as a string
  - `my_string  = "Hello"`
- Hello in a list
  - `my_list = ["H","e","l","l","o"]`

Strings and Lists – to work with indexing

```
def main():
    my_string  = "Hello"
    my_list =
  ["H","e","l","l","o"]
    iterate_string(my_string)
    print(4 * "*")
    iterate_list(my_list)
```

```
def iterate_string(my_string):
    i = -1
    while i > -5:
        print(my_string[i])
        i -= 1
def iterate_list(my_list):
    i = 0
    while i < 4:
        print(my_list[i])
        i += 1
if __name__ == "__main__":
    main()
```

Strings and Lists – to work with indexing

```
def main():
    my_string  = "Hello"
    my_list =
  ["H","e","l","l","o"]
    iterate_string(my_string)
    print(4 * "*")
    iterate_list(my_list)
```

```
def iterate_string(my_string):
    i = -1
    while i > -5:
        print(my_string[i])
        i -= 1
def iterate_list(my_list):
    i = 0
    while i < 4:
        print(my_list[i])
        i += 1
if __name__ == "__main__":
    main()
```

```
In [46]: o
l
l
e
****
H
e
l
l
```

Useful list commands

- in – not in

```
>>> 'howdy' in ['hello', 'hi', 'howdy', 'heyas']
True
>>> 'howdy' not in spam
False
```

- multiple assignment

```
>>> cat = ['fat', 'black', 'loud']
>>> size, color, disposition = cat
```

- index

```
>>> spam = ['hello', 'hi', 'howdy', 'heyas']
>>> spam.index('hello')
0
```

- spam.append('moose')
- spam.insert(1, 'chicken')
- spam.remove('bat')
  - only first instance removed
- spam.sort
- spam.sort(reverse=True)

Immutable data types

- Strings are immutable: they cannot be changed.

```
In [16]: name = 'Zophie a cat'
    ...: name[7] = 'the'
    ...:
Traceback (most recent call last):

  File "<ipython-input-16-4448876246c3>", line 2, in <module>
    name[7] = 'the'

TypeError: 'str' object does not support item assignment
```

- But it works if we definine a new variable with the same name.

```
In [15]: name = 'Zophie a cat'
    ...: name = name[0:7] + 'the' + name[8:12]
    ...: name
Out[15]: 'Zophie the cat'
```

String commands

- **Literal prefix "\"**
  - \\ means \
  - \t means tab, \n means new line,
  - A *raw string* completely ignores all escape characters and prints any backslash that appears in the string.

```
In [19]: print('Say hi to Bob\'s mother.')
Say hi to Bob's mother.

In [20]: print("Hello there!\nHow are you?\nI\'m doing fine.")
Hello there!
How are you?
I'm doing fine.

In [21]: print(r'That is Carol\'s cat.')
That is Carol\'s cat.
```

- **Multiline Strings with Triple Quotes**

```
print('''Dear Alice,
Eve's cat has been arrested for catnapping, cat burglary, and extortion.
Sincerely,
Bob''')
```

```
Dear Alice,
Eve's cat has been arrested for catnapping, cat burglary, and extortion.
Sincerely,
Bob
```

String commands

- .upper()
- .lower
- .join()
- len()

- x.join(list)
  - join list, separated by x
- .split(x)
  - split list by x (default=space)

```
In [24]: spam = 'Hello world!'
    ...: spam = spam.upper()
    ...: spam
Out[24]: 'HELLO WORLD!'

In [25]: spam = spam.lower()
    ...: spam
    ...:
Out[25]: 'hello world!'

In [30]: len('Hello world!')
Out[30]: 12
```

```
In [26]: ' '.join(['My', 'name', 'is', 'Simon'])
Out[26]: 'My name is Simon'

In [27]: 'ABC'.join(['My', 'name', 'is', 'Simon'])
Out[27]: 'MyABCnameABCisABCSimon'

         'My name is Simon'.split()
         ['My', 'name', 'is', 'Simon']

         spam = '''Dear Alice,
         How have you been?
         Bob'''
         spam.split('\n')
         ['Dear Alice,', 'How have you been?', 'Bob']
```

String concatenation

```
>>> 'Alice' + 'Bob'
'AliceBob'
```

```
for file_name in my_files:
    print(file_name)
    out_file="csv_" + file_name
    out_handle = open(out_file,"w")
```

Regular expression

- Python is better at manipulating strings than Stata.
- import re
  - re.search(pattern, string)
  - re.match()
  - re.findall()
  - re.sub()
  - re.split()

- Syntax: the sequence
  - prog = re.compile(pattern)
  - result = prog.match(string)

is equivalent to

  - result = re.match(pattern, string)

Using re.compile() saves the regular expression object for reuse

- Always put r in front of the regular expression. Otherwise Python cannot parse.

https://pythex.org/

Regular expression cheatsheet

Special characters

`\`	escape special characters
`.`	matches any character
`^`	matches beginning of string
`$`	matches end of string
`[5b-d]`	matches any chars '5', 'b', 'c' or 'd'
`[^a-c6]`	matches any char except 'a', 'b', 'c' or '6'
`R|S`	matches either regex `R` or regex `S`
`()`	creates a capture group and indicates precedence

Quantifiers

`*`	0 or more (append `?` for non-greedy)
`+`	1 or more (append `?` for non-greedy)
`?`	0 or 1 (append `?` for non-greedy)
`{m}`	exactly `m` m occurrences
`{m, n}`	from `m` to `n`. `m` defaults to 0, `n` to infinity
`{m, n}?`	from `m` to `n`, as few as possible

Special sequences

`\A`	start of string
`\b`	matches empty string at word boundary (between `\w` and `\W`)
`\B`	matches empty string not at word boundary
`\d`	digit
`\D`	non-digit
`\s`	whitespace: `[ \t\n\r\f\v]`
`\S`	non-whitespace
`\w`	alphanumeric: `[0-9a-zA-Z_]`
`\W`	non-alphanumeric
`\Z`	end of string
`\g<id>`	matches a previously defined group

Special sequences

`(?iLmsux)`	matches empty string, sets re.X flags
`(?:...)`	non-capturing version of regular parentheses
`(?P...)`	matches whatever matched previously named group
`(?P=)`	digit
`(?#...)`	a comment; ignored
`(?=...)`	lookahead assertion: matches without consuming
`(?!...)`	negative lookahead assertion
`(?<=...)`	lookbehind assertion: matches if preceded
`(?<!...)`	negative lookbehind assertion
`(?(id)yes|no)`	match 'yes' if group 'id' matched, else 'no'

Search, findall, match

- search result in groups

```
>>> phoneNumRegex = re.compile(r'(\d\d\d)-(\d\d\d-\d\d\d\d)')
>>> mo = phoneNumRegex.search('My number is 415-555-4242.')
>>> mo.group(1)
'415'
>>> mo.group(2)
'555-4242'
>>> mo.group(0)
'415-555-4242'
>>> mo.group()
'415-555-4242'
```

- findall creates list of all matches

```
>>> phoneNumRegex = re.compile(r'(\d\d\d)-(\d\d\d)-(\d\d\d\d)') # has groups
>>> phoneNumRegex.findall('Cell: 415-555-9999 Work: 212-555-0000')
[('415', '555', '1122'), ('212', '555', '0000')]
```

- greedy match (.*)
  - match as much text as possible
- non-greedy match (.*?)
  - match as much text as possible

```
>>> nongreedyRegex = re.compile(r'<.*?>')
>>> mo = nongreedyRegex.search('<To serve man> for dinner.>')
>>> mo.group()
'<To serve man>'

>>> greedyRegex = re.compile(r'<.*>')
>>> mo = greedyRegex.search('<To serve man> for dinner.>')
>>> mo.group()
'<To serve man> for dinner.>'
```

Substituting Strings with the sub() Method

- re.sub

```
>>> namesRegex = re.compile(r'Agent \w+')
>>> namesRegex.sub('CENSORED', 'Agent Alice gave the secret documents to Agent Bob.')
'CENSORED gave the secret documents to CENSORED.'
```

Regular expression examples

- re.VERBOSE clarifies long expressions (tells the re.compile() function to ignore whitespace and comments inside the regular expression).

Now instead of a hard-to-read regular expression like this:

```
phoneRegex = re.compile(r'((\d{3}|\(\d{3}\))?(\s|-|\.)?\d{3}(\s|-|\.)\d{4}
(\s*(ext|x|ext.)\s*\d{2,5})?)')
```

you can spread the regular expression over multiple lines with comments like this:

```
phoneRegex = re.compile(r'''(
    (\d{3}|\(\d{3}\))?            # area code
    (\s|-|\.)?                    # separator
    \d{3}                         # first 3 digits
    (\s|-|\.)                     # separator
    \d{4}                         # last 4 digits
    (\s*(ext|x|ext.)\s*\d{2,5})?  # extension
    )''', re.VERBOSE)
```

Dictionaries

- Dictionaries are unordered lists of indexes (*keys)*, and associated values.
  - myCat = {'size': 'fat', 'color': 'gray', 'disposition': 'loud'}
- Methods: keys(), values(), items(),
  - setdefault() set a value in a dictionary for a certain key only if that key does not already have a value.

```
In [45]: for v in myCat.values():
    ...:     print(v)
    ...:
    ...:
gray
fat
loud
```

Using dictionaries to count words

```python
from os import listdir
from os import chdir
import re

in_dir = "E:/c_old/DavidD/Courses/AppliedEmpirical/Python/105-extracted-date"

def main():
    chdir(in_dir)
    my_files = listdir(in_dir)
    count={}
    for file_name in my_files:
        print(file_name)
        text_file = open(file_name,"r", encoding='latin-1')
        for line in text_file:
            wordlist = line.split()
            for word in wordlist:
                word = re.sub(r'\W', '', word)
                word=word.lower()
                count.setdefault(word, 0)
                count[word]+=1
        text_file.close()
    with open('senate_word_freq.txt', 'w') as f:
        f.write('word, frequency\n')
        for key in count:
            f.write(key + ',' + str(count[key]) + '\n')
    f.close()
if __name__ == "__main__":
    main()
```

senate_word_freq.txt

```
word, frequency
doc,8192
docno105abrahammi119981112docno,12
text,8264
mr,12324
abraham,3960
president,8880
during,1214
debate,620
on,12038
final,260
passage,248
of,51808
the,95966
```

\W matches any non-alphanumeric characters

4. Program execution

Python Command Line Arguments

- https://www.tutorialspoint.com/python/python_command_line_arguments.htm

cmd_arguments.py

```python
# -*- coding: utf-8 -*-
"""
Created on Fri Sep 22 11:23:25 2017

@author: dstro
"""

import sys

print('Number of arguments:', len(sys.argv), 'arguments.')
print('Argument List:', str(sys.argv))
second_argument=sys.argv[1]
print('Second Argument:', second_argument)
```

```
e:\c_old\DavidD\Courses\AppliedEmpirical\Python\Code>Python cmd_arguments.py infile 2 3
Number of arguments: 4 arguments.
Argument List: ['cmd_arguments.py', 'infile', '2', '3']
Second Argument: infile
```

Python and Stata

- Python in your master file.
- How to run a Python program in Stata.

From Stata 16:

- Interactive use
  - python (continues in python after exception)
  - and python: (exits to Stata after exception)

How to run a Python program in Stata

- Use shell commands in Stata

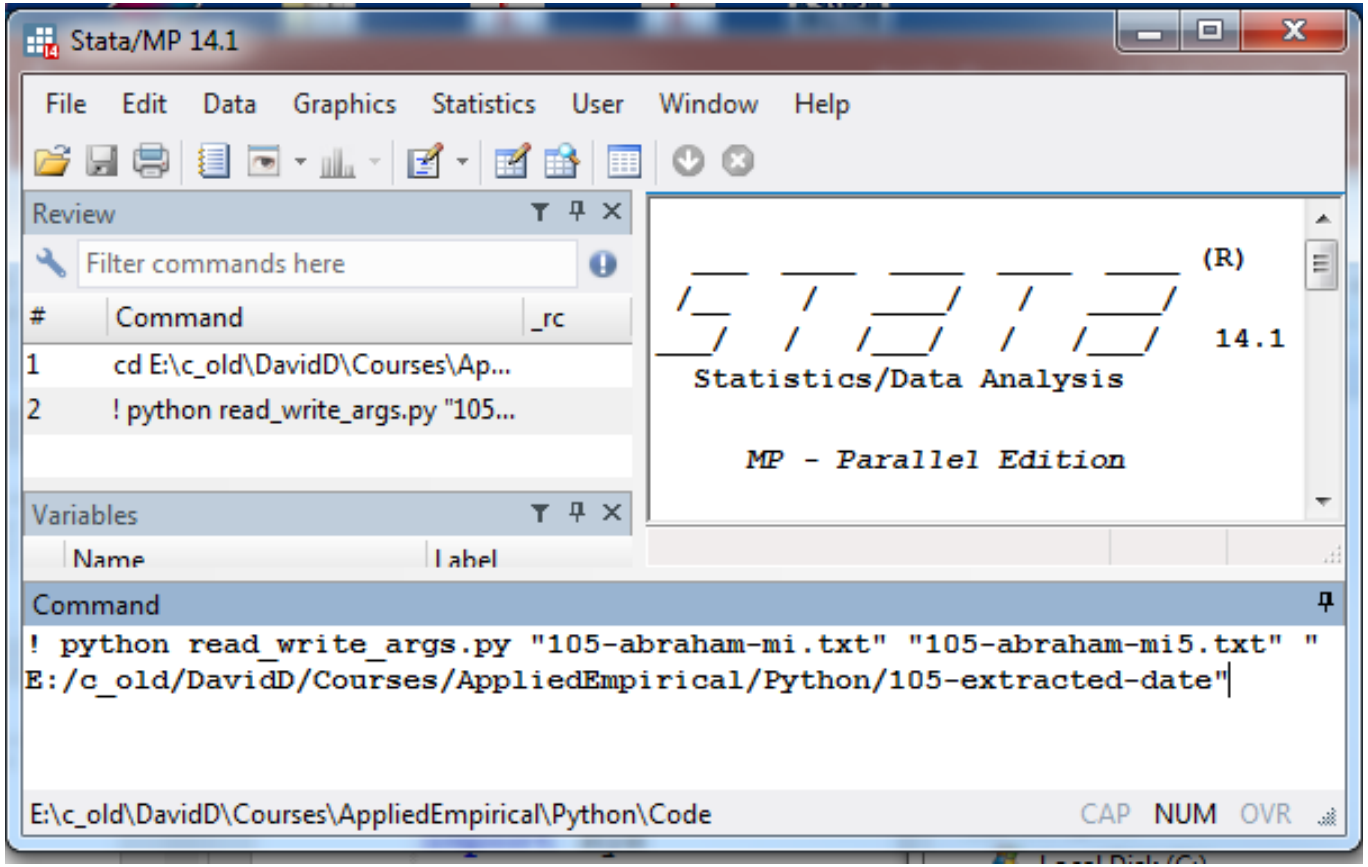
read_write_args.py

```python
from os import chdir
import sys

def main():
    infile=sys.argv[1]
    outfile=sys.argv[2]
    indir=sys.argv[3]

    chdir(indir)
    in_file = open(infile,"r")
    text = in_file.read()
    in_file.close()

    out_file = open(outfile,"w")
    out_file.write(text)
    out_file.close()
if __name__ == "__main__":
    main()
```



How to run a Python in Stata 16

- python: … end statements

```
. do "C:\Users\dstro\AppDat

. python:
------------------------------------------------------------
> exit) ----------------------------------------------------
>>> text="Hello World"
>>> text.lower()
'hello world'
>>> text.split()
['Hello', 'World']
>>> end
------------------------------------------------------------
```

How to run a Python in Stata 16

- In: Stata macros accessible in python, directly and using SFI Macro
- Out: Python results posted to Stata using SFI Scalar

```
local a = 2
local b = 3
python:
from sfi import Scalar
def calcsum(num1, num2):
    res = num1 + num2
    Scalar.setValue("result", res)
calcsum(`a', `b')
end
display result
```

```
. display result
5
```

```
* Using SFI (Stata Function Interface)
local a = 2
local b = 2
python:
from sfi import Macro, Scalar
def calcsum(num1, num2):
  res = num1 + num2
  Scalar.setValue("result", res)
pya = int(Macro.getLocal("a"))
pyb = int(Macro.getLocal("b"))
calcsum(pya, pyb)
end
display result
```

```
. display result
4
```

Python module sfi (Stata Function Interface)

- Provide access to Stata's current dataset, frames, macros, scalars, matrices, value labels, characteristics, global Mata matrices, and more. https://www.stata.com/python/api16/

- In example: Stata variable -> Python list

```
python:
from sfi import Data
wordlist = Data.get(invar)
```

- Out example: Python list -> Stata variable
  - Data.addVarStr(outvar,len) -> gen str outvar = “”
  - Data.store(outvar,None,wordlist) -> replace outvar = wordlist

Python wrapper in Stata

- ado file that calls python

rt_string.do.2019-10-15_154530.bak | test_rm_python.do | rm_pattern.ad

```
clear
adopath + e:\c_old\DavidD\Courses\Appli
set obs 1
gen str word="123#%/hello?-"

rm_pattern word word2 "[^a-z]"
rm_pattern word word3 "\W"
```

	word	word2	word3
1	123w#e%/hello?-	wehello	123wehello

string.do.2019-10-15_154530.bak | test_rm_python.do | rm_pattern.ado

```
program define rm_pattern
    version 16
    args invar outvar pattern
    python: remove_pattern("`invar'", "`outvar'",
    "`pattern'")
end

python:
from sfi import Data
import re
def remove_pattern(invar,outvar,pattern):
    wordlist = Data.get(invar)
    wordlist[:] = [ re.sub(pattern, '', word) for
    word in wordlist]
    fmt=Data.getVarFormat(invar)
    len=int(re.sub(r'\D', '', fmt))
    Data.addVarStr(outvar,len)
    Data.store(outvar, None, wordlist)
end
```

More complex example:
Run Python random forest in Stata

test_rm_python.do | rm_pattern.ado | test_rforest_python.do

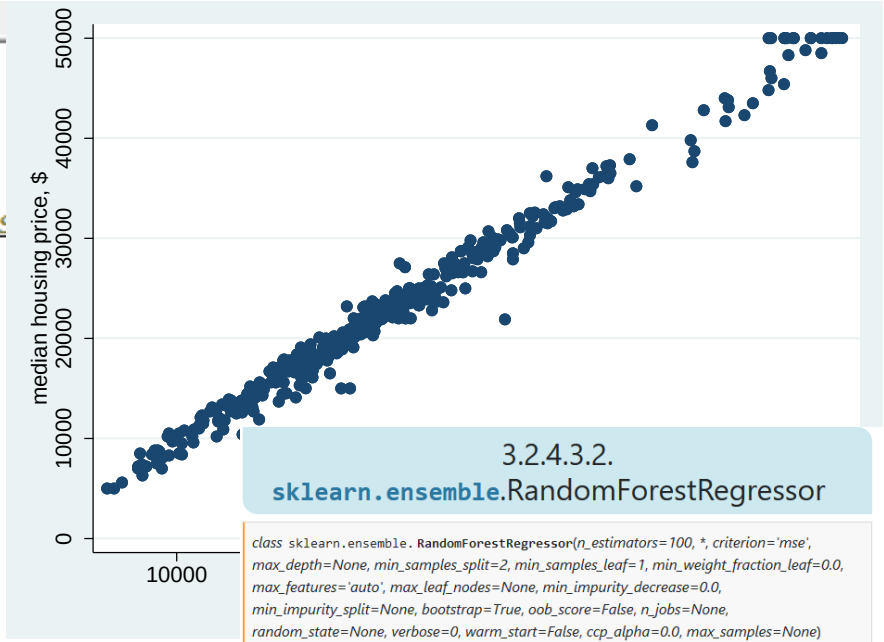
```
/* Call Python random forest predictor from within Stata 16 */
clear
adopath + e:\c_old\DavidD\Courses\AppliedEmpirical\Python\Code

* Load house price data from StatLiv archive (http://lib.stat.cmu.edu/datasets/bos

use http://www.stata-press.com/data/imeus/hprice2a, clear

ds crim-lowstat
local xvars = r(varlist)
di "`xvars'"
sum `xvars'

python:
from sfi import Data
import numpy as np
from sklearn.ensemble import RandomForestRegressor
X = np.array(Data.get("`xvars'"))
y = np.array(Data.get("price"))
rf = RandomForestRegressor()
rf.fit(X,y)
xbhat = rf.predict(X)
Data.addVarFloat("xbhat")
Data.store("xbhat", None, xbhat)
end
```



3.2.4.3.2.
sklearn.ensemble. RandomForestRegressor

class sklearn.ensemble.**RandomForestRegressor**(*n_estimators=100, *, criterion='mse', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, ccp_alpha=0.0, max_samples=None*)

[source]

A random forest regressor.

A random forest is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is controlled with the `max_samples` parameter if `bootstrap=True` (default), otherwise the whole dataset is used to build each tree.

Read more in the User Guide.

Parameters:

n_estimators : *int, default=100*
The number of trees in the forest.

Changed in version 0.22: The default value of `n_estimators` changed from 10 to 100 in 0.22.

criterion : *{"mse", "mae"}, default="mse"*
The function to measure the quality of a split. Supported criteria are

More complex wrapper: pyforest

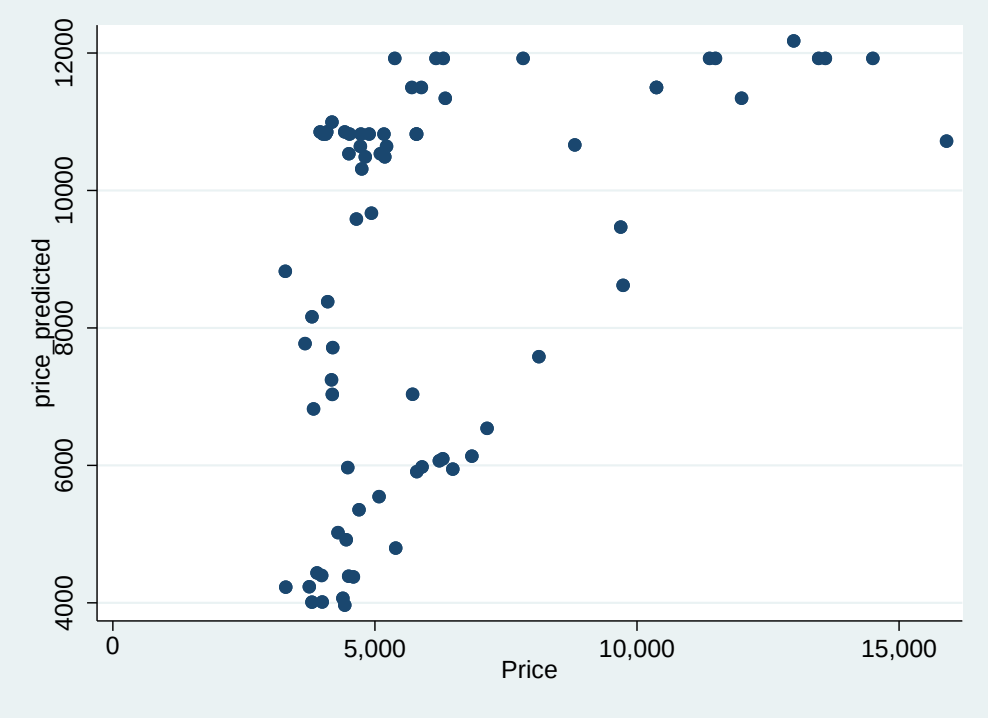
test_rforest_python.do | test_rm_python.do | rm_pattern.ado

```
* Load auto dataset
sysuse auto, clear

* Estimate random forest regression model, predicting price using mpg, trunk, and weight
* Train only on cars with foreign==1 and test on foreign==0
* https://github.com/mdroste/stata-pylearn/blob/master/src/pyforest.ado, row 487-
pyforest price mpg trunk weight, type(regress) training(foreign)
predict price_predicted

* We can also use Stata-like 'if' conditions to specify the training sample, but we won't get out-of-sample RMSE in that case
pyforest price mpg trunk weight if foreign==1, type(regress)
predict price_predicted_2

* Pyforest chooses defaults automatically, but you can use any hyperparameter scikit-learn can
pyforest price mpg trunk weight, type(regress) training(foreign) max_depth(5) min_samples_split(4)
```



pyforest wrapper

```
*=========================================================================
* Program:   pyforest.ado
* Purpose:   Random forest classification and regression in Stata 16+ with
*            Python and scikit-learn. Component of pylearn.
*=========================================================================

program define pyforest, eclass
version 16.0
syntax varlist(min=2) [if] [in] [aweight fweight], ///
[ ///
```

```
#-------------------------------------------------------------------------
# Define Python function: run_random_forest
#-------------------------------------------------------------------------

def run_random_forest(type,vars,n_estimators,criterion,max_depth,min_samples_

        #-----------------------------------------------------------
        # Load data from Stata into Python
        #-----------------------------------------------------------

        # Load into Pandas data frame
        df = DataFrame(Data.get(vars))
```

```
    # Initialize random forest regressor (if model type is regress)
        if type=="regress":
                model = RandomForestRegressor(n_estimators=n_estimators,
```

```
        # Train model on training data
        model.fit(x_insample, y_insample)

        # Get in-sample prediction
        pred_insample = model.predict(x_insample)

        # Get full-sample prediction
        model_predict = model.predict(x)
```

Task 8

1. Construct a Python program that:
   - Converts miles/hour to kilometers/hour
   - Uses the **input – process – output** model
   - Divides the code into ***functions***
   - Has ***error handling*** for the input
   - Starts the execution from a **main()**
   - Has a ***user-friendly*** dialogue

2. Construct a palindrome detector that:
   - Asks the user for a text to test
   - Removes all characters that are not **alphanumeric**
   - Checks the given text **character by character**
   - Accepts ***"Madam, I'm Adam!"*** as a palindrome
   - Divides the code into functions.

Task 8

3. The files in the folders 105-extracted-date and 106-extracted-date contains all speeches by U.S. senators in the 105th and 106th Congress (1997-2000). The name of each file shows the congress-name-state abbreviation. For example, the file "105-akaka-hi.txt" contains all speeches by senator Akaka from Hawaii in the 105th congress (1997-1998). The task is to count the frequency of words used by each senator in each congress. For example, the number of times in the 105th congress that senator Akaka mentioned theword "gun".

a. Write a program that loops over directories and all files in the directories and prints the full file name.

b. In the loop, read the speech files and split the text based on blanks (space) into words that are put in an array.

c. Loop over words in the array. Remove non-alphanumeric characters. Replace upper by lower case letters.

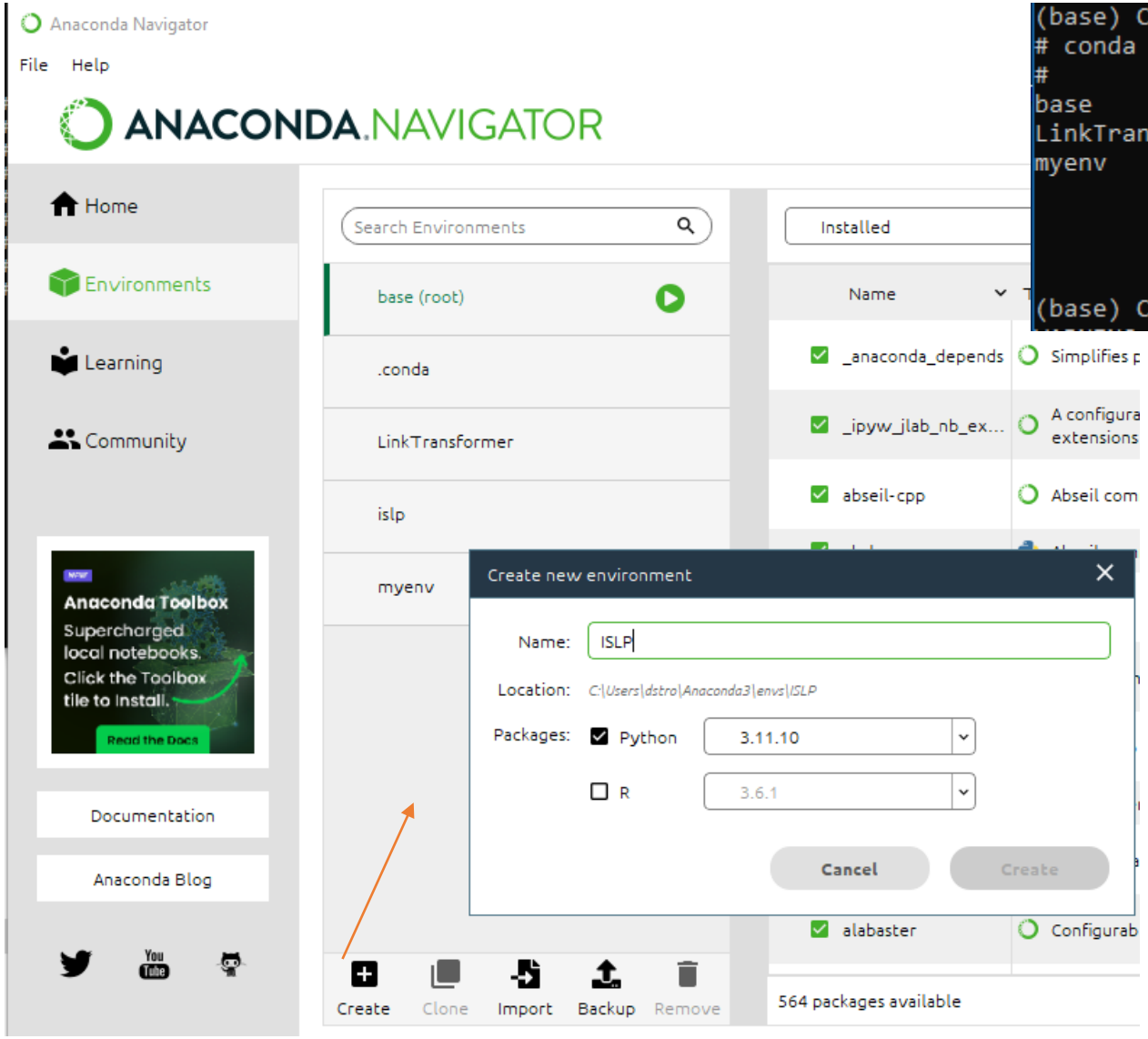
d. Count the frequency of each remaining words. Save the result in a comma-separated text file in "long" format. The first row should contain the variable names: "file_name", "word", "frequency". The following rows should contain the corresponding values.

Python virtual environments

- Version control in Python.
  - Python packages change frequently.
  - Create a virtual environment with the version that you are currently using, save this environment.

- ISLP
  - The ISLP labs are built with ISLP_labs/v2.2. Visit the lab git repo for specific instructions to install the frozen environment.
  - To create a Python conda environment
    - Mac OS X or Linux environment, in a run: conda create --name islp python
    - Windows On windows, create a Python environment called islp in the Anaconda app.
    - Current conda should have this at least 3.9. If not, replace python with python=3.10 or python=3.11.
    - The newest version python=3.12 is not currently supported as some packages, such as torch are not installable. To run python code in this environment, you must activate it:

Python virtual environments



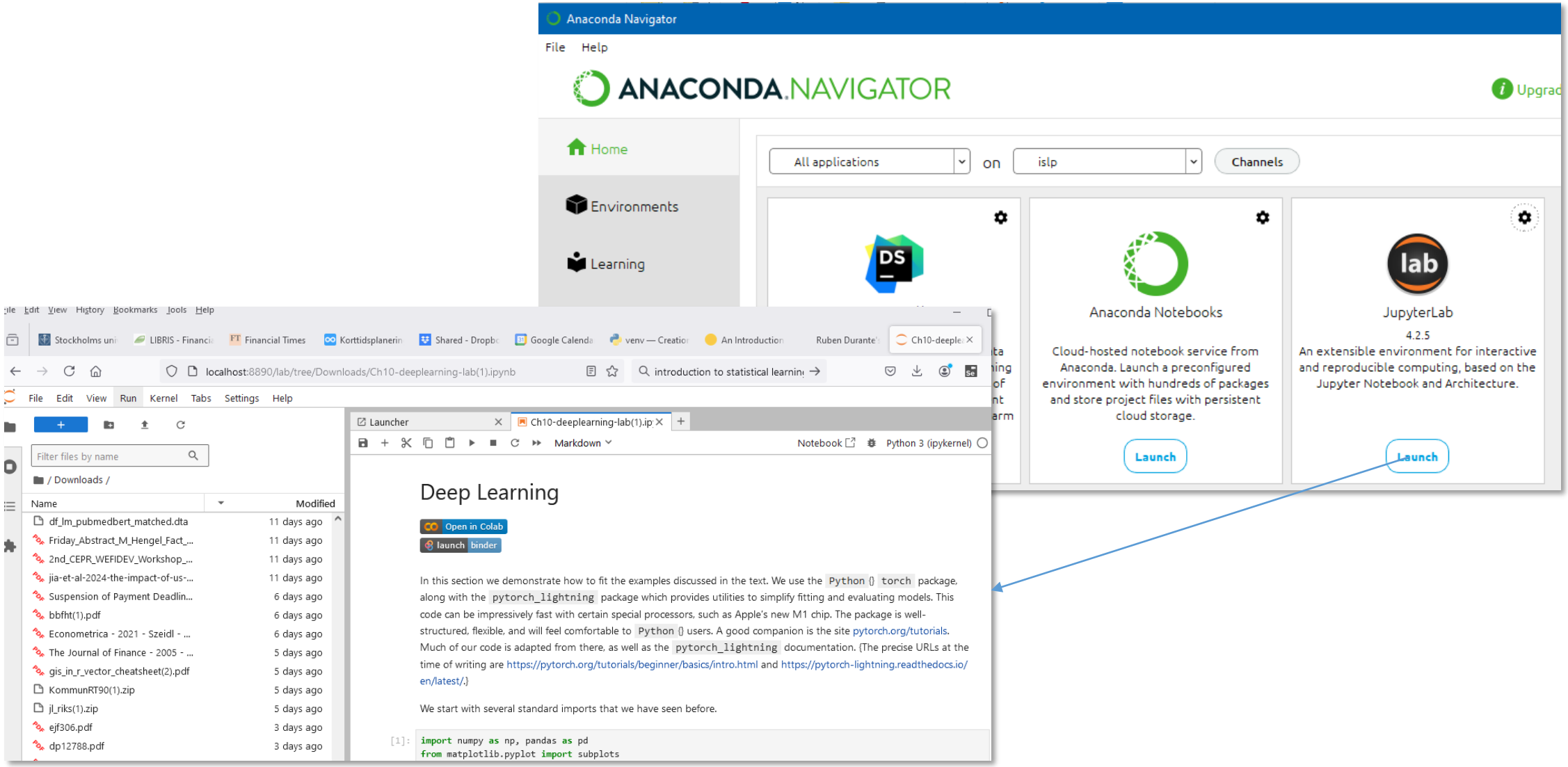

```
(base) C:\>conda env list
# conda environments:
#
base                  *  C:\Users\dstro\Anaconda3
LinkTransformer          C:\Users\dstro\Anaconda3\envs\LinkTransformer
myenv                    C:\Users\dstro\Anaconda3\envs\myenv
                         E:\c_old\DavidD\Courses\AppliedEmpirical\Slides
                         e:\c_old\DavidD\Courses\AppliedEmpirical\Slides


(base) C:\>conda create --name islp python=3.11
```

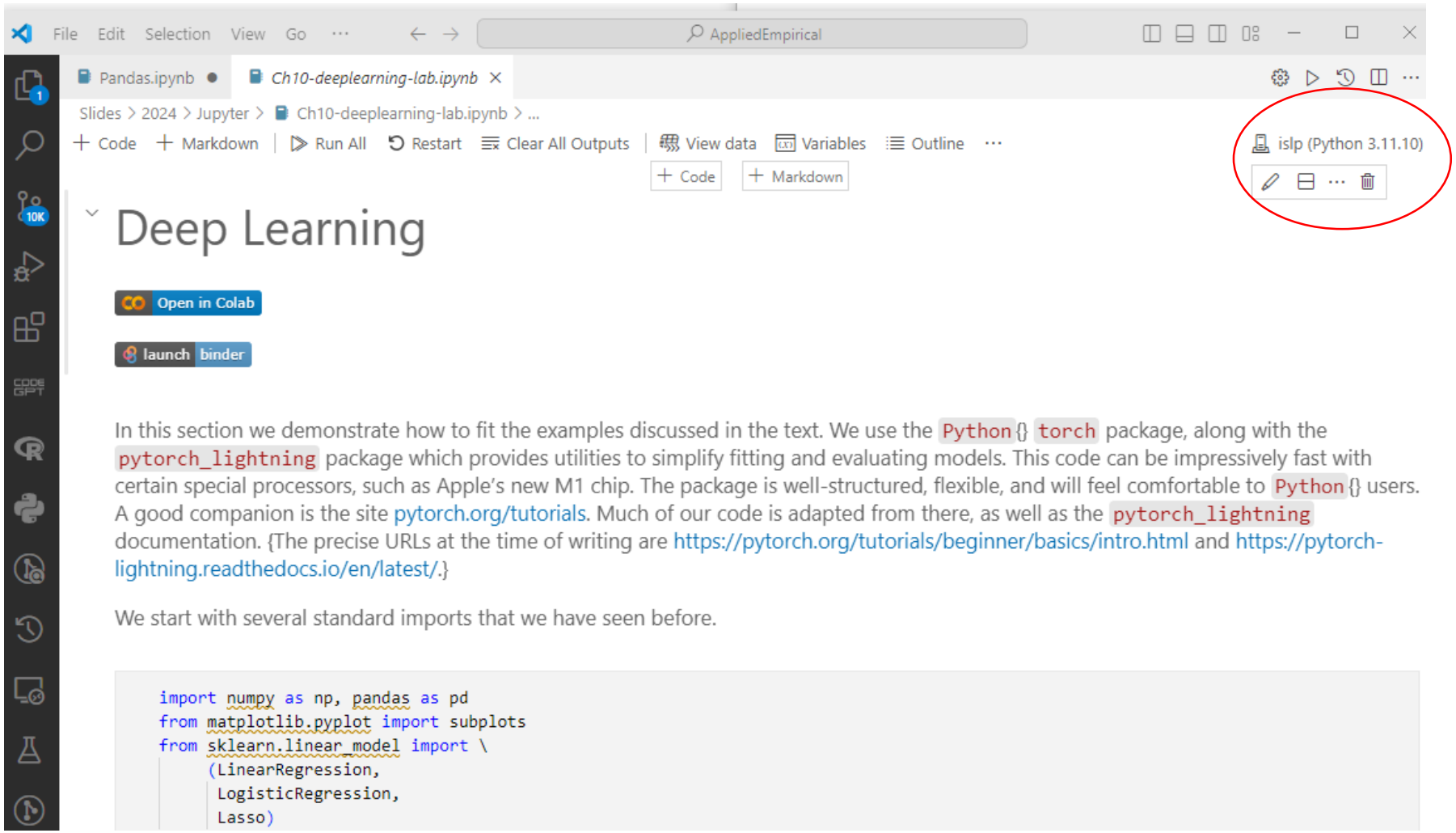
```
(base) C:\>conda activate islp

(islp) C:\>
```

Jupyter Notebook/Lab



Jupyter Notebook in VS Code

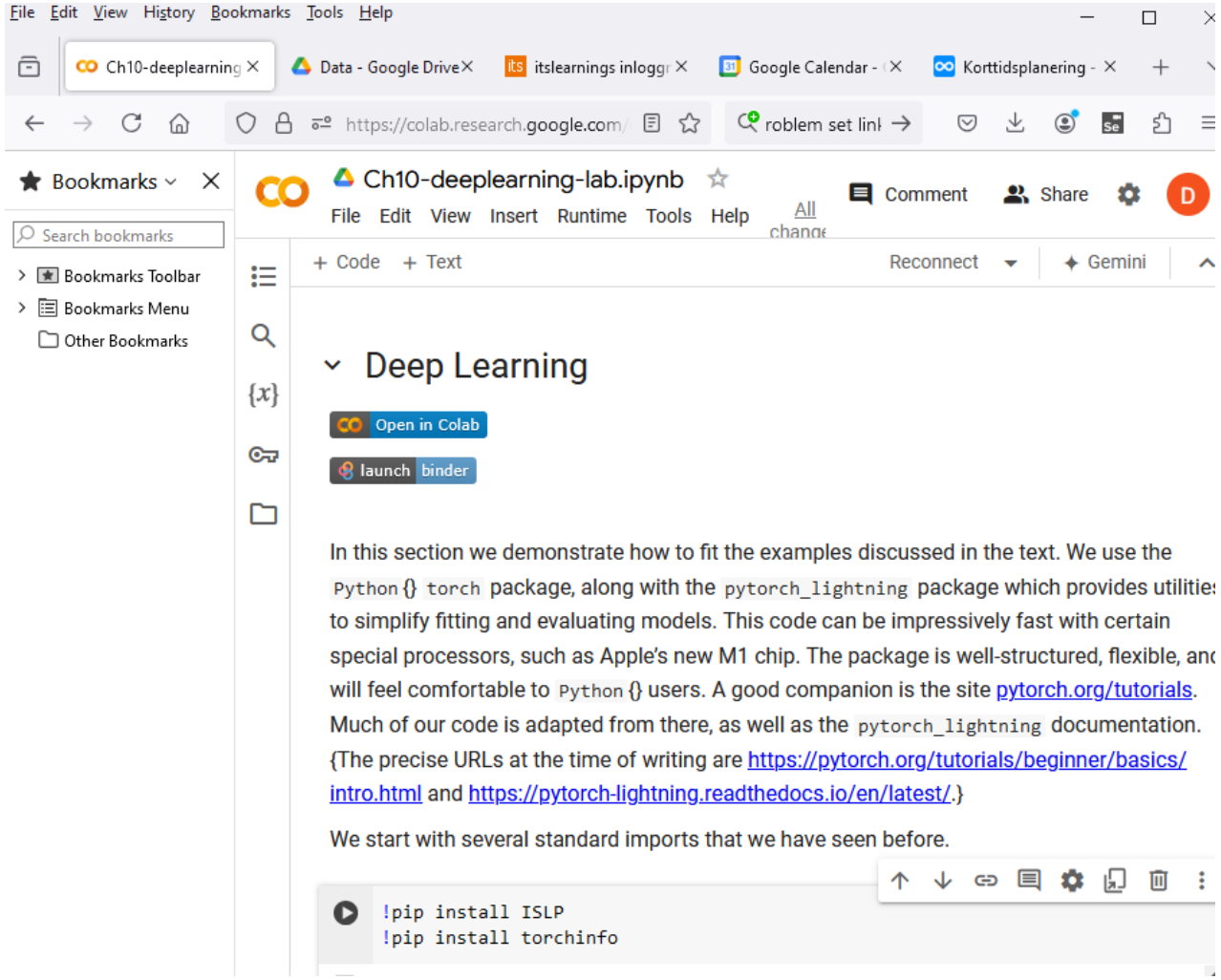


Jupyter Notebooks in Google Colab

Notebooks can be run in [Google Colab]() with a few caveats:

- Labs that use files in the filesystem will require one to mount your Google Drive. See Google's [help]().

- The packages will have to be reinstalled each time a new runtime is started. For most labs, inserting pip install ISLP at the top of the notebook will suffice, though Colab will ask you to restart after installation.



Pandas

Pandas is a powerful open-source data analysis and data manipulation library for Python.

Key features and uses:

- Input/Output:
  - Supports reading from and writing to various file formats, including CSV, Excel, SQL databases, and more.
- Data Structures:
  - DataFrame (two-dimensional), which are used to store and manipulate data efficiently.
- Data Cleaning:
  - Tools for handling missing data, filtering rows or columns, and transforming data into a suitable format for analysis.
- Data Manipulation:
  - Reshape and pivot datasets, merge or join multiple datasets, and perform group operations.
- Data Analysis:
  - Functions for statistical analysis, such as calculating means, medians, variances, and more.
- Integration:
  - Pandas integrates well with other data science libraries like NumPy, Matplotlib, and SciPy, making it a central tool in the Python data analysis ecosystem.

Pandas

- https://pandas.pydata.org/docs/user_guide/10min.html
- Pandasgui
- Data Wrangler extension (VS Code)