Python Tutorial

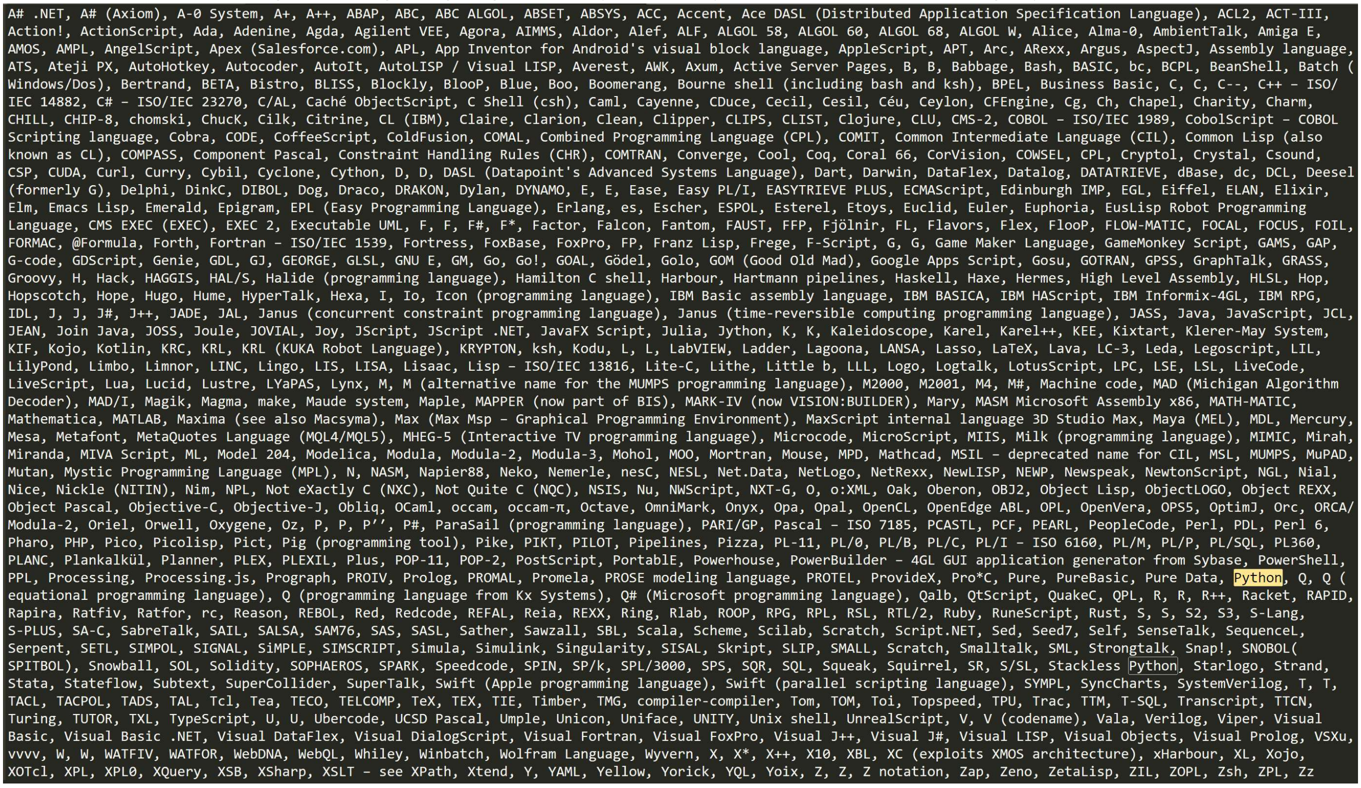
Frederick Ayala-Gómez, Phd Student in Computer Science - ELTE university. Visiting researcher, Aalto university

Agenda

1. Motivation
2. The Python Programming Language

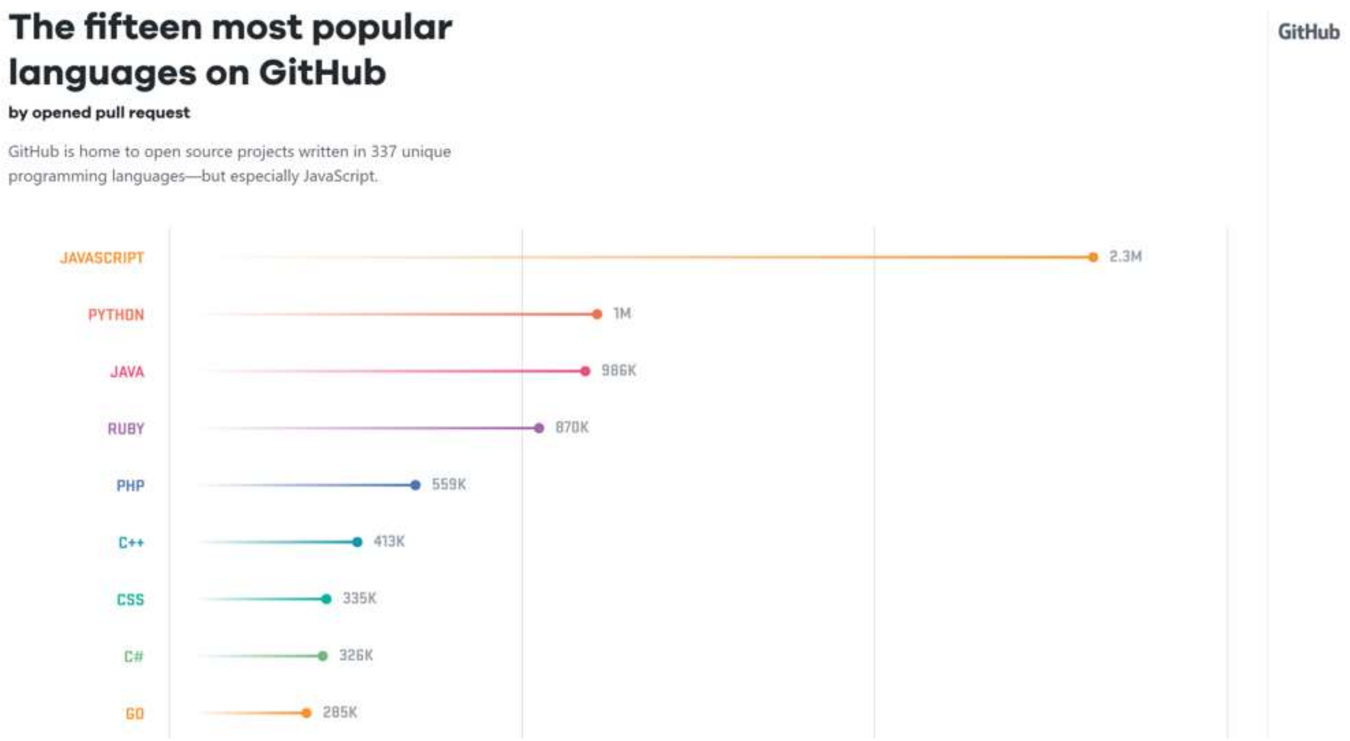
Why to learn Python?

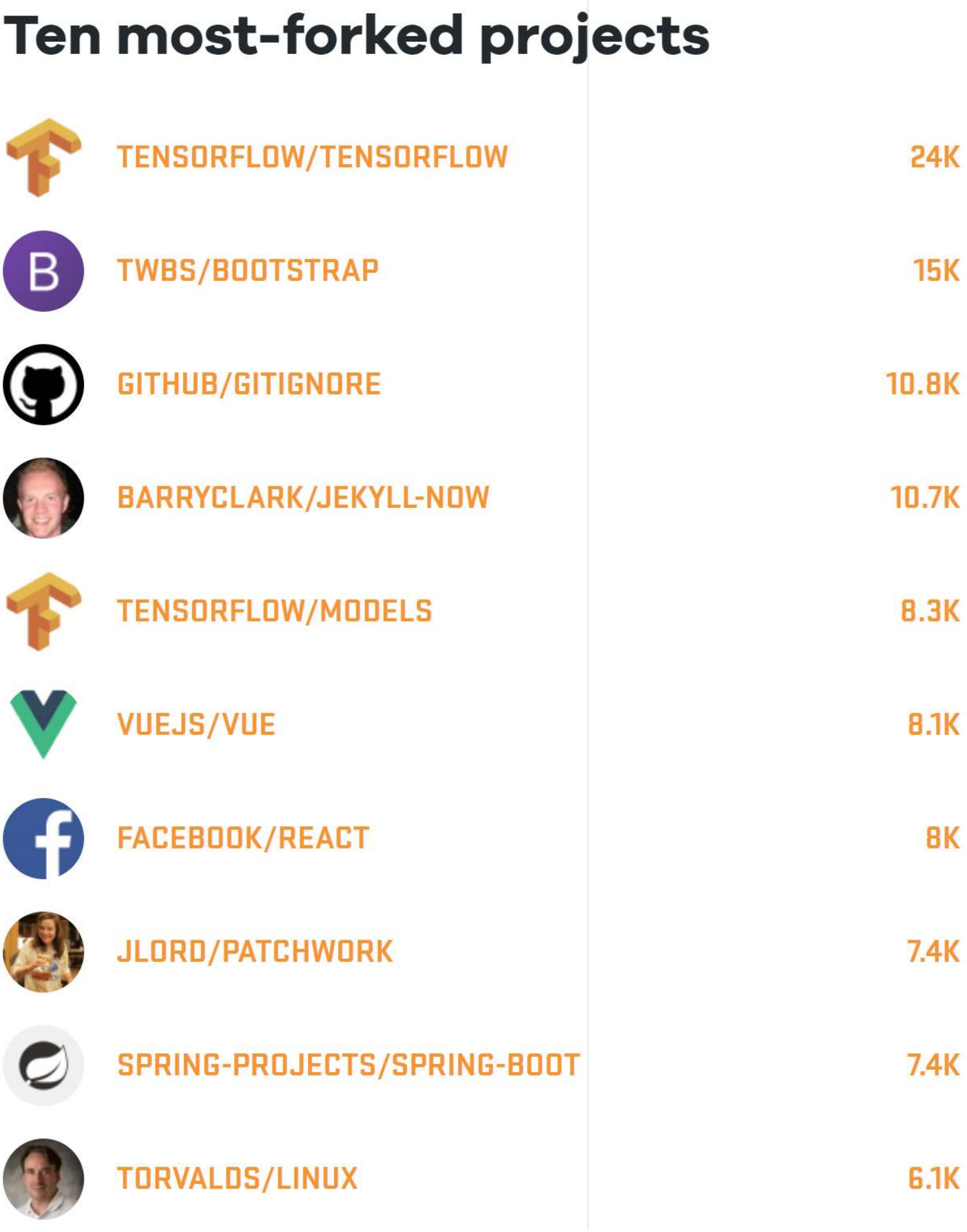
There are several programming languages... why should you care about learning Python?



```
A# .NET, A# (Axiom), A-0 System, A+, A++, ABAP, ABC, ABC ALGOL, ABSET, ABSYS, ACC, Accent, Ace DASL (Distributed Application Specification Language), ACL2, ACT-III,
Action!, ActionScript, Ada, Adenine, Agda, Agilent VEE, Agora, AIMMS, Aldor, Alef, ALF, ALGOL 58, ALGOL 60, ALGOL 68, ALGOL W, Alice, Alma-0, AmbientTalk, Amiga E,
AMOS, AMPL, AngelScript, Apex (Salesforce.com), APL, App Inventor for Android's visual block language, AppleScript, APT, Arc, ARexx, Argus, AspectJ, Assembly language,
ATS, Ateji PX, AutoHotkey, Autocoder, AutoIt, AutoLISP / Visual LISP, Averest, AWK, Axum, Active Server Pages, B, B, Babbage, Bash, BASIC, bc, BCPL, BeanShell, Batch (
Windows/Dos), Bertrand, BETA, Bistro, BLISS, Blockly, BlooP, Blue, Boo, Boomerang, Bourne shell (including bash and ksh), BPEL, Business Basic, C, C, C--, C++ – ISO/
IEC 14882, C# – ISO/IEC 23270, C/AL, Caché ObjectScript, C Shell (csh), Caml, Cayenne, CDuce, Cecil, Cesil, Céu, Ceylon, CFEngine, Cg, Ch, Chapel, Charity, Charm,
CHILL, CHIP-8, chomski, ChucK, Cilk, Citrine, CL (IBM), Claire, Clarion, Clean, Clipper, CLIPS, CLIST, Clojure, CLU, CMS-2, COBOL – ISO/IEC 1989, CobolScript – COBOL
Scripting language, Cobra, CODE, CoffeeScript, ColdFusion, COMAL, Combined Programming Language (CPL), COMIT, Common Intermediate Language (CIL), Common Lisp (also
known as CL), COMPASS, Component Pascal, Constraint Handling Rules (CHR), COMTRAN, Converge, Cool, Coq, Coral 66, CorVision, COWSEL, CPL, Cryptol, Crystal, Csound,
CSP, CUDA, Curl, Curry, Cybil, Cyclone, Cython, D, D, DASL (Datapoint's Advanced Systems Language), Dart, Darwin, DataFlex, Datalog, DATATRIEVE, dBase, dc, DCL, Deesel
(formerly G), Delphi, DinkC, DIBOL, Dog, Draco, DRAKON, Dylan, DYNAMO, E, E, Ease, Easy PL/I, EASYTRIEVE PLUS, ECMAScript, Edinburgh IMP, EGL, Eiffel, ELAN, Elixir,
Elm, Emacs Lisp, Emerald, Epigram, EPL (Easy Programming Language), Erlang, es, Escher, ESPOL, Esterel, Etoys, Euclid, Euler, Euphoria, EusLisp Robot Programming
Language, CMS EXEC (EXEC), EXEC 2, Executable UML, F, F, F#, F*, Factor, Falcon, Fantom, FAUST, FFP, Fjölnir, FL, Flavors, Flex, FlooP, FLOW-MATIC, FOCAL, FOCUS, FOIL,
FORMAC, @Formula, Forth, Fortran – ISO/IEC 1539, Fortress, FoxBase, FoxPro, FP, Franz Lisp, Frege, F-Script, G, G, Game Maker Language, GameMonkey Script, GAMS, GAP,
G-code, GDScript, Genie, GDL, GJ, GEORGE, GLSL, GNU E, GM, Go, Go!, GOAL, Gödel, Golo, GOM (Good Old Mad), Google Apps Script, Gosu, GOTRAN, GPSS, GraphTalk, GRASS,
Groovy, H, Hack, HAGGIS, HAL/S, Halide (programming language), Hamilton C shell, Harbour, Hartmann pipelines, Haskell, Haxe, Hermes, High Level Assembly, HLSL, Hop,
Hopscotch, Hope, Hugo, Hume, HyperTalk, Hexa, I, Io, Icon (programming language), IBM Basic assembly language, IBM BASICA, IBM HAScript, IBM Informix-4GL, IBM RPG,
IDL, J, J, J#, J++, JADE, JAL, Janus (concurrent constraint programming language), Janus (time-reversible computing programming language), JASS, Java, JavaScript, JCL,
JEAN, Join Java, JOSS, Joule, JOVIAL, Joy, JScript, JScript .NET, JavaFX Script, Julia, Jython, K, K, Kaleidoscope, Karel, Karel++, KEE, Kixtart, Klerer-May System,
KIF, Kojo, Kotlin, KRC, KRL, KRL (KUKA Robot Language), KRYPTON, ksh, Kodu, L, L, LabVIEW, Ladder, Lagoona, LANSA, Lasso, LaTeX, Lava, LC-3, Leda, Legoscript, LIL,
LilyPond, Limbo, Limnor, LINC, Lingo, LIS, LISA, Lisaac, Lisp – ISO/IEC 13816, Lite-C, Lithe, Little b, LLL, Logo, Logtalk, LotusScript, LPC, LSE, LSL, LiveCode,
LiveScript, Lua, Lucid, Lustre, LYaPAS, Lynx, M, M (alternative name for the MUMPS programming language), M2000, M2001, M4, M#, Machine code, MAD (Michigan Algorithm
Decoder), MAD/I, Magik, Magma, make, Maude system, Maple, MAPPER (now part of BIS), MARK-IV (now VISION:BUILDER), Mary, MASM Microsoft Assembly x86, MATH-MATIC,
Mathematica, MATLAB, Maxima (see also Macsyma), Max (Max Msp – Graphical Programming Environment), MaxScript internal language 3D Studio Max, Maya (MEL), MDL, Mercury,
Mesa, Metafont, MetaQuotes Language (MQL4/MQL5), MHEG-5 (Interactive TV programming language), Microcode, MicroScript, MIIS, Milk (programming language), MIMIC, Mirah,
Miranda, MIVA Script, ML, Model 204, Modelica, Modula, Modula-2, Modula-3, Mohol, MOO, Mortran, Mouse, MPD, Mathcad, MSIL – deprecated name for CIL, MSL, MUMPS, MuPAD,
Mutan, Mystic Programming Language (MPL), N, NASM, Napier88, Neko, Nemerle, nesC, NESL, Net.Data, NetLogo, NetRexx, NewLISP, NEWP, Newspeak, NewtonScript, NGL, Nial,
Nice, Nickle (NITIN), Nim, NPL, Not eXactly C (NXC), Not Quite C (NQC), NSIS, Nu, NWScript, NXT-G, O, o:XML, Oak, Oberon, OBJ2, Object Lisp, ObjectLOGO, Object REXX,
Object Pascal, Objective-C, Objective-J, Obliq, OCaml, occam, occam-π, Octave, OmniMark, Onyx, Opa, Opal, OpenCL, OpenEdge ABL, OPL, OpenVera, OPS5, OptimJ, Orc, ORCA/
Modula-2, Oriel, Orwell, Oxygene, Oz, P, P, P′′, P#, ParaSail (programming language), PARI/GP, Pascal – ISO 7185, PCASTL, PCF, PEARL, PeopleCode, Perl, PDL, Perl 6,
Pharo, PHP, Pico, Picolisp, Pict, Pig (programming tool), Pike, PIKT, PILOT, Pipelines, Pizza, PL-11, PL/0, PL/B, PL/C, PL/I – ISO 6160, PL/M, PL/P, PL/SQL, PL360,
PLANC, Plankalkül, Planner, PLEX, PLEXIL, Plus, POP-11, POP-2, PostScript, PortablE, Powerhouse, PowerBuilder – 4GL GUI application generator from Sybase, PowerShell,
PPL, Processing, Processing.js, Prograph, PROIV, Prolog, PROMAL, Promela, PROSE modeling language, PROTEL, ProvideX, Pro*C, Pure, PureBasic, Pure Data, Python, Q, Q (
equational programming language), Q (programming language from Kx Systems), Q# (Microsoft programming language), Qalb, QtScript, QuakeC, QPL, R, R, R++, Racket, RAPID,
Rapira, Ratfiv, Ratfor, rc, Reason, REBOL, Red, Redcode, REFAL, Reia, REXX, Ring, Rlab, ROOP, RPG, RPL, RSL, RTL/2, Ruby, RuneScript, Rust, S, S, S2, S3, S-Lang,
S-PLUS, SA-C, SabreTalk, SAIL, SALSA, SAM76, SAS, SASL, Sather, Sawzall, SBL, Scala, Scheme, Scilab, Scratch, Script.NET, Sed, Seed7, Self, SenseTalk, SequenceL,
Serpent, SETL, SIMPOL, SIGNAL, SiMPLE, SIMSCRIPT, Simula, Simulink, Singularity, SISAL, Skript, SLIP, SMALL, Scratch, Smalltalk, SML, Strongtalk, Snap!, SNOBOL(
SPITBOL), Snowball, SOL, Solidity, SOPHAEROS, SPARK, Speedcode, SPIN, SP/k, SPL/3000, SPS, SQR, SQL, Squeak, Squirrel, SR, S/SL, Stackless Python, Starlogo, Strand,
Stata, Stateflow, Subtext, SuperCollider, SuperTalk, Swift (Apple programming language), Swift (parallel scripting language), SYMPL, SyncCharts, SystemVerilog, T, T,
TACL, TACPOL, TADS, TAL, Tcl, Tea, TECO, TELCOMP, TeX, TEX, TIE, Timber, TMG, compiler-compiler, Tom, TOM, Toi, Topspeed, TPU, Trac, TTM, T-SQL, Transcript, TTCN,
Turing, TUTOR, TXL, TypeScript, U, U, Ubercode, UCSD Pascal, Umple, Unicon, Uniface, UNITY, Unix shell, UnrealScript, V, V (codename), Vala, Verilog, Viper, Visual
Basic, Visual Basic .NET, Visual DataFlex, Visual DialogScript, Visual Fortran, Visual FoxPro, Visual J++, Visual J#, Visual LISP, Visual Objects, Visual Prolog, VSXu,
vvvv, W, W, WATFIV, WATFOR, WebDNA, WebQL, Whiley, Winbatch, Wolfram Language, Wyvern, X, X*, X++, X10, XBL, XC (exploits XMOS architecture), xHarbour, XL, Xojo,
XOTcl, XPL, XPL0, XQuery, XSB, XSharp, XSLT – see XPath, Xtend, Y, YAML, Yellow, Yorick, YQL, Yoix, Z, Z, Z notation, Zap, Zeno, ZetaLisp, ZIL, ZOPL, Zsh, ZPL, Zz
```

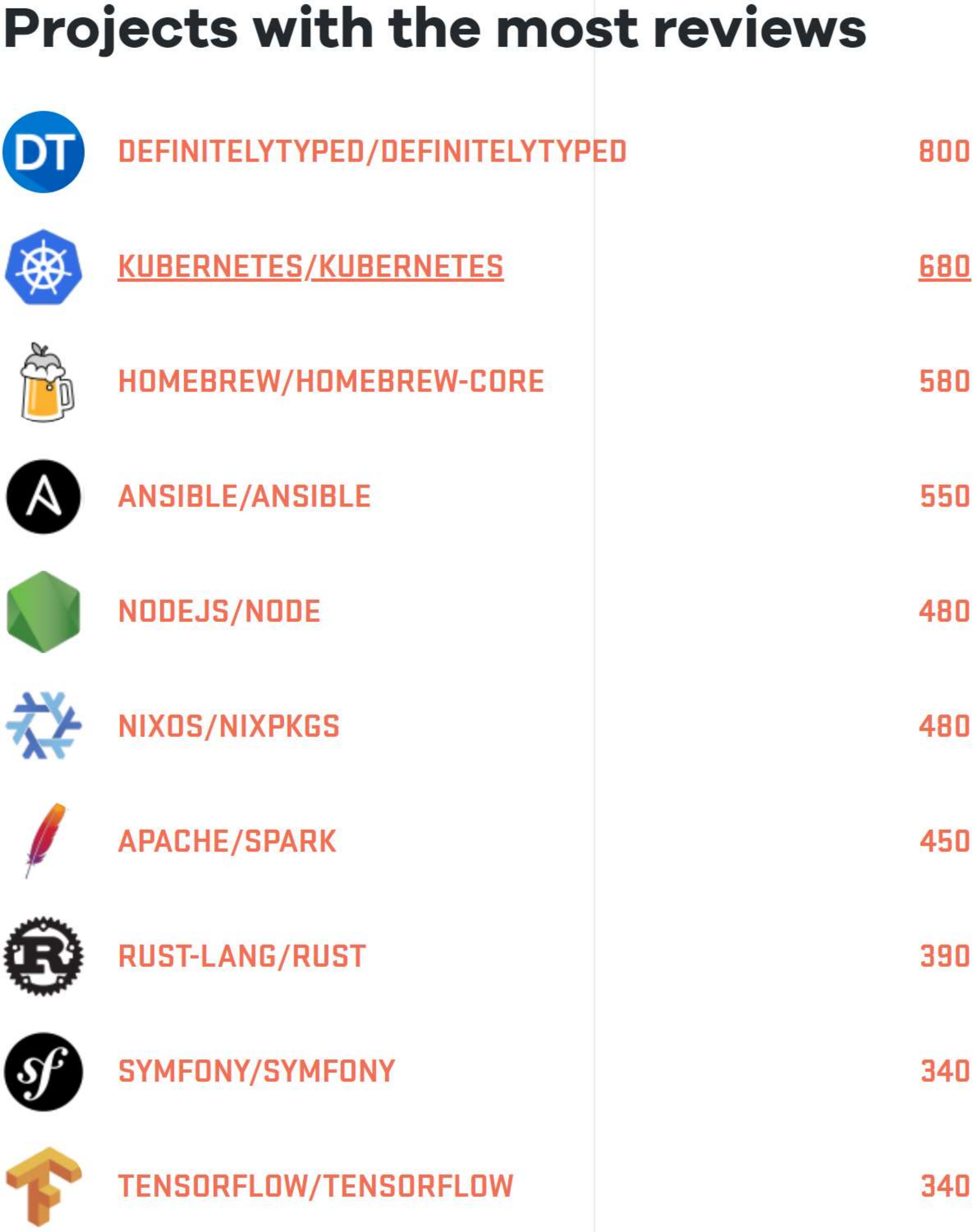
1.- It is widely used in open source projects





Ten most-forked projects

TENSORFLOW/TENSORFLOW	24K
TWBS/BOOTSTRAP	15K
GITHUB/GITIGNORE	10.8K
BARRYCLARK/JEKYLL-NOW	10.7K
TENSORFLOW/MODELS	8.3K
VUEJS/VUE	8.1K
FACEBOOK/REACT	8K
JLORD/PATCHWORK	7.4K
SPRING-PROJECTS/SPRING-BOOT	7.4K
TORVALDS/LINUX	6.1K



Projects with the most reviews

DEFINITELYTYPED/DEFINITELYTYPED	800
KUBERNETES/KUBERNETES	680
HOMEBREW/HOMEBREW-CORE	580
ANSIBLE/ANSIBLE	550
NODEJS/NODE	480
NIXOS/NIXPKGS	480
APACHE/SPARK	450
RUST-LANG/RUST	390
SYMFONY/SYMFONY	340
TENSORFLOW/TENSORFLOW	340

https://octoverse.github.com/ (https://octoverse.github.com/)

2.- Relevant open source databases and big data frameworks have a Python SDK

- PostgreSQL - Psycopg2
- Apache Spark - pyspark
- Apache Flink - flink
- Apache Beam - apache-beam
- Elasticsearch - elasticsearch
- Apple's Turi Create - turi-create

3.- Cloud providers have python SDK too!

- Amazon Web Services - boto3
- Microsoft Azure - azure-sdk-for-python
- Google Cloud - Several python libraries

4.- The python ecosystem has many libraries that are publicly available

Popular packages related to data analysis and data mining:

- **Pandas** - Povides high-performance, easy-to-use data structures and data analysis tools/
- **Matplotlib** - Plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms.
- **SciPy** - For mathematics, science, and engineering.
- **NumPy** - Fundamental package for scientific computing with Python. It contains among other things: a powerful N-dimensional array object; sophisticated (broadcasting) functions; tools for integrating C/C++ and Fortran code; useful linear algebra, Fourier transform, and random number capabilities.
- **TensorFlow** - An open-source machine learning framework for everyone.
- **Scikit-learn** - Simple and efficient tools for data mining and data analysis.

Popular packages related to web development and crawling:

- **Django** - A high-level Python Web framework that encourages rapid development and clean, pragmatic design.
- **Flask** - Micro web framework written in Python.
- **Uvicorn** - Lightning-fast asyncio server, for Python 3.
- **Beautiful Soup** - For parsing HTML and XML documents.
- **Scrapy** - A Fast and Powerful Scraping and Web Crawling Framework

5.- Platform independent

- As long as you have a python enviroment and the dependencies are installed in the OS
- Mobile applications development possible using kiwi or toga

So what is Python?

Reference material: Python Essential Reference (4th Edition) by David Beazley

- Python is an interpreted high-level programming language for general-purpose programming.
- Python programs are executed by an interpreter
  - Usually: **python**
- Often, we use a Python Development Enviroment to simplify the development of Python programs.
  - For example: PyCharm, Jupyter Notebook, Spyder
  - Yes! We can write and run Python in Jupyter Notebook :)
- There are different versions of python. The most popular are: 2.7, 3.5, 3.6
- It is suggested to use python 3. Python 2.7 will be depreciated in the year **2020**

Pyton at a Glance

In [402]:

```
print("Hello World!")
```

```
Hello World!
```

Python is interactive

In [403]:

```
print(10*3)
```

```
30
```

Variables

In [404]:

```
scholarship = 1000 # I am a comment and I just create a variable called balance that co
ntains the number 1000
save_pct = .0 # This is a float
studies_duration = 24  # This is an int
save_each = 3
```

In [405]:

```
scholarship
```

Out[405]:

```
1000
```

Conditionals

In [406]:

```
if save_pct <= 0: # Possible conditions: <, <=, ==, >, >=, !=, is, not
    print("No savings :-O")
elif save_pct <= .02:
    print("Seriously?")
else:
    print("Looking good B-)")
```

```
No savings :-O
```

Functions

- The function help is useful to know more about any function

In [407]:

```
help(help)
```

```
Help on _Helper in module _sitebuiltins object:

class _Helper(builtins.object)
 |  Define the builtin 'help'.
 |  
 |  This is a wrapper around pydoc.help that provides a helpful message
 |  when 'help' is typed at the Python interactive prompt.
 |  
 |  Calling help() at the Python prompt starts an interactive help sessio
n.
 |  Calling help(thing) prints help for the python object 'thing'.
 |  
 |  Methods defined here:
 |  
 |  __call__(self, *args, **kwds)
 |      Call self as a function.
 |  
 |  __repr__(self)
 |      Return repr(self).
 |  
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |  
 |  __dict__
 |      dictionary for instance variables (if defined)
 |  
 |  __weakref__
 |      list of weak references to the object (if defined)
```

In [408]:

```
help(range)
```

```
Help on class range in module builtins:

class range(object)
 |  range(stop) -> range object
 |  range(start, stop[, step]) -> range object
 |  
 |  Return an object that produces a sequence of integers from start (incl
usive)
 |  to stop (exclusive) by step.  range(i, j) produces i, i+1, i+2, ..., j
-1.
 |  start defaults to 0, and stop is omitted!  range(4) produces 0, 1, 2, 
3.
 |  These are exactly the valid indices for a list of 4 elements.
 |  When step is given, it specifies the increment (or decrement).
 |  
 |  Methods defined here:
 |  
 |  __bool__(self, /)
 |      self != 0
 |  
 |  __contains__(self, key, /)
 |      Return key in self.
 |  
 |  __eq__(self, value, /)
 |      Return self==value.
 |  
 |  __ge__(self, value, /)
 |      Return self>=value.
 |  
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |  
 |  __getitem__(self, key, /)
 |      Return self[key].
 |  
 |  __gt__(self, value, /)
 |      Return self>value.
 |  
 |  __hash__(self, /)
 |      Return hash(self).
 |  
 |  __iter__(self, /)
 |      Implement iter(self).
 |  
 |  __le__(self, value, /)
 |      Return self<=value.
 |  
 |  __len__(self, /)
 |      Return len(self).
 |  
 |  __lt__(self, value, /)
 |      Return self<value.
 |  
 |  __ne__(self, value, /)
 |      Return self!=value.
 |  
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for accurate signa
ture.
 |  
 |  __reduce__(...)
 |      helper for pickle
 |  
 |  __repr__(self, /)
 |      Return repr(self).
 |  
 |  __reversed__(...)
 |      Return a reverse iterator.
 |  
 |  count(...)
 |      rangeobject.count(value) -> integer -- return number of occurrence
s of value
 |  
 |  index(...)
 |      rangeobject.index(value, [start, [stop]]) -> integer -- return ind
ex of value.
 |      Raise ValueError if the value is not present.
 |  
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |  
 |  start
 |  
 |  step
 |  
 |  stop
```

In [409]:

```
for month in range(0,studies_duration):
    print(month)
    if month == 5:
        print("Fine I get it")
        break
```

```
0
1
2
3
4
5
Fine I get it
```

Putting all together

In [410]:

```
total_savings = 0
for month in range(0,studies_duration):
    if (month+1) % save_each == 0: # Modulo operation finds the remainder after division of one number by another
        amount_to_save = save_pct * scholarship
        print(f"Saving {amount_to_save} in month {month}")
        total_savings += amount_to_save
print(f"You saved: {total_savings}, great!") # There are multiple ways to format a string
print("You saved: {}, great!".format(total_savings))
print("You saved: %.1f, great!"%(total_savings))
```

```
Saving 0.0 in month 2
Saving 0.0 in month 5
Saving 0.0 in month 8
Saving 0.0 in month 11
Saving 0.0 in month 14
Saving 0.0 in month 17
Saving 0.0 in month 20
Saving 0.0 in month 23
You saved: 0.0, great!
You saved: 0.0, great!
You saved: 0.0, great!
```

Structures

Tuples

In [411]:

```
a_tuple = (0,1,2,3,4,5,6,6)
```

In [412]:

```
print(a_tuple)
```

```
(0, 1, 2, 3, 4, 5, 6, 6)
```

In [413]:

```
print(a_tuple[1])
```

```
1
```

Lists

In [414]:

```
a_list = [0,1,2,3,4,5,4,3,2,1]
print(a_list)
```

```
[0, 1, 2, 3, 4, 5, 4, 3, 2, 1]
```

In [415]:

```
a_list.append(0)
print(a_list)
```

```
[0, 1, 2, 3, 4, 5, 4, 3, 2, 1, 0]
```

In [416]:

```
a_list.remove(0)
print(a_list)
```

```
[1, 2, 3, 4, 5, 4, 3, 2, 1, 0]
```

List indexing

In [417]:

```
# First element
print(a_list[0])
```

```
1
```

In [418]:

```
# Last element
print(a_list[-1])
```

```
0
```

In [419]:

```
# First two elements
print(a_list[:2])
```

```
[1, 2]
```

In [420]:

```
# Last two elements
print(a_list[-2:])
```

```
[1, 0]
```

Dictionaries

In [421]:

```
a_dict = {"key1": 1, "key2": 2, "key3": 3}
print(a_dict)
```

```
{'key1': 1, 'key2': 2, 'key3': 3}
```

In [422]:

```
a_dict["key_1"] = 2
a_dict["key_2"] = "any value"
a_dict["key_3"] = {"another_dict":"wow"}
```

Sets

In [423]:

```
a_set = set(a_list)
print(a_set)
```

```
{0, 1, 2, 3, 4, 5}
```

Functions

In [424]:

```
def add_numbers(a,b):
    return a+b
```

In [425]:

```
print(add_numbers(1,2))
```

```
3
```

List Comprehension

In [426]:

```
my_new_list = [l for l in a_list if l > 2]
```

In [427]:

```
print(my_new_list)
```

```
[3, 4, 5, 4, 3]
```

In [428]:

```
my_new_list = [l*2 for l in a_list if l > 2]
```

In [429]:

```
print(my_new_list)
```

```
[6, 8, 10, 8, 6]
```

Lambda Functions

In [430]:

```
g = lambda x: x*x
```

In [431]:

```
print(g(5))
```

```
25
```

In [432]:

```
help(map)
```

```
Help on class map in module builtins:

class map(object)
 |  map(func, *iterables) --> map object
 |  
 |  Make an iterator that computes the function using arguments from
 |  each of the iterables.  Stops when the shortest iterable is exhausted.
 |  
 |  Methods defined here:
 |  
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |  
 |  __iter__(self, /)
 |      Implement iter(self).
 |  
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for accurate signa
ture.
 |  
 |  __next__(self, /)
 |      Implement next(self).
 |  
 |  __reduce__(...)
 |      Return state information for pickling.
```

In [433]:

```
my_list_tuples = [(0,1),(2,3),(4,5)]
print(my_list_tuples)
```

```
[(0, 1), (2, 3), (4, 5)]
```

In [434]:

```
for i in map(lambda x: x[0] + x[1], my_list_tuples):
    print(i)
```

```
1
5
9
```

I/O operations

In [435]:

```
# Reading a file
with open("C:/Users/frede/languages.txt","r") as my_text:
    for line in my_text:
        print(line)
        break
```

```
A# .NET
```

In [436]:

```
# Writting to a file
with open("C:/Users/frede/count.txt","w") as my_counter:
    for i in range(0,10):
        my_counter.write(str(i))
        my_counter.write("\n")
```

Installing libraries, and managing enviroments

Options

1. Install packages with the package manager globally as a super user or append --user to install in the home directory
   - Using **pip**
   - sudo pip install pandas
   - sudo pip install --upgrade pandas
   - sudo pip uninstall pandas
   - It is not recommended if you are working in multiple projects in the same machine
2. Install the interpreter in your operating system
   - sudo pip install virtualenv
   - Isolate your working enviroments using virtualenv by creating *enviroments*
   - virtualenv -p /usr/bin/python3.6 ~/venv36 # Creates a virtual enviroment using a specific version of python
   - source ~/venv36/bin/activate # Activates the enviroment
   - pip install pandas # This would install pandas in the enviroment
   - deactivate
3. Use Anaconda/Mini-conda
   - Install Anaconda: https://anaconda.org/anaconda/python (https://anaconda.org/anaconda/python)
   - conda create -n env36 python=3.6
   - activate env36 or source activate env36 in linux/mac
   - install packages using conda install instead of pip install. Works better in Windows, as it installs any required DLL

Pandas

Not part of DMS course but it's useful to learn Pandas

In [437]:

```
import pandas as pd
%matplotlib inline
```

In [438]:

```
df_iris = pd.read_csv("iris.data.txt", names=["sepal_length", "sepal_width", "petal_length", "petal_width", "iris_class"])
```

In [439]:

```
df_iris.head()
```

Out[439]:

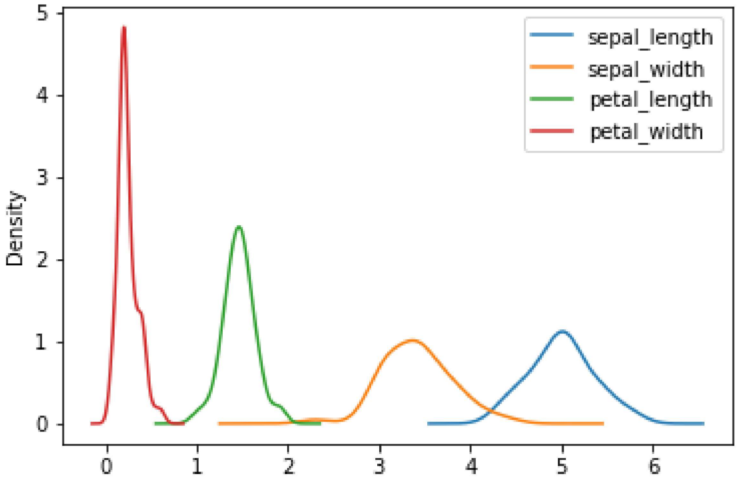
	sepal_length	sepal_width	petal_length	petal_width	iris_class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

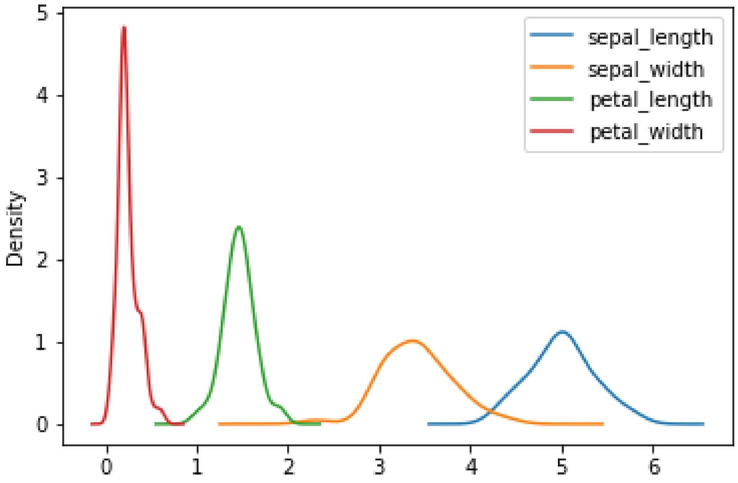
In [440]:

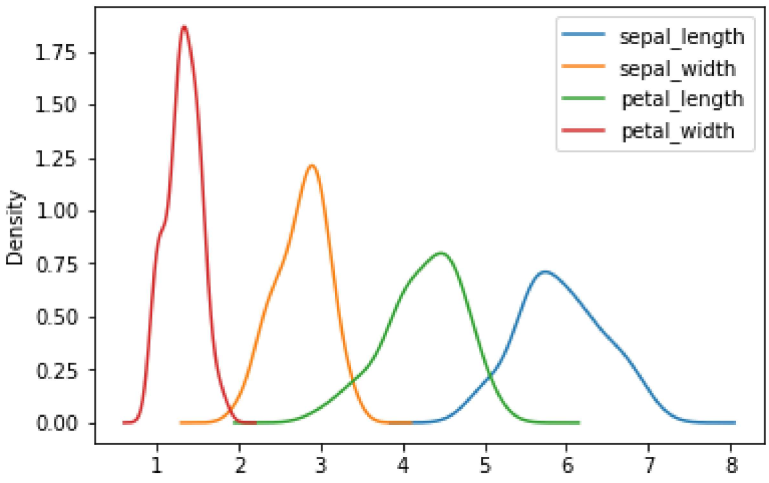
```
df_iris.groupby("iris_class").plot.density()
```

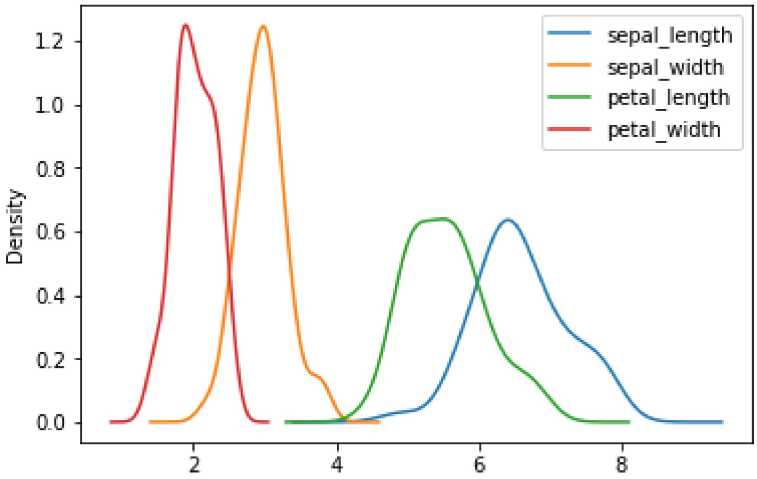
```
Out[440]:

iris_class
Iris-setosa        AxesSubplot(0.125,0.125;0.775x0.755)
Iris-versicolor    AxesSubplot(0.125,0.125;0.775x0.755)
Iris-virginica     AxesSubplot(0.125,0.125;0.775x0.755)
dtype: object
```









In [441]:

```
df_iris.query("sepal_length > 5.8").head()
```

Out[441]:

	sepal_length	sepal_width	petal_length	petal_width	iris_class
50	7.0	3.2	4.7	1.4	Iris-versicolor
51	6.4	3.2	4.5	1.5	Iris-versicolor
52	6.9	3.1	4.9	1.5	Iris-versicolor
54	6.5	2.8	4.6	1.5	Iris-versicolor
56	6.3	3.3	4.7	1.6	Iris-versicolor

In [442]:

```
df_iris.describe()
```

Out[442]:

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

In [443]:

```
df_iris.groupby("iris_class").describe().transpose().to_csv('iris_data_stats.csv')
```