Python For Data Science *Cheat Sheet*

Importing Data

Learn Python for data science Interactively at www.DataCamp.com

Importing Data in Python

Most of the time, you'll use either **NumPy** or **pandas** to import your data:

```
>>> import numpy as np
>>> import pandas as pd
```

Help

```
>>> np.info(np.ndarray.dtype)
>>> help(pd.read_csv)
```

Text Files

Plain Text Files

```
>>> filename = 'huck_finn.txt'
>>> file = open(filename, mode='r')
>>> text = file.read()
>>> print(file.closed)
>>> file.close()
>>> print(text)
```

Open the file for reading
Read a file's contents
Check whether file is closed
Close file

Using the context manager `with`

```
>>> with open('huck_finn.txt', 'r') as file:
        print(file.readline())
        print(file.readline())
        print(file.readline())
```

Read a single line

Table Data: Flat Files

Importing Flat Files with numpy

Files with one data type

```
>>> filename = 'mnist.txt'
>>> data = np.loadtxt(filename,
                      delimiter=',',
                      skiprows=2,
                      usecols=[0,2],
                      dtype=str)
```

String used to separate values
Skip the first 2 lines
Read the 1st and 3rd column
The type of the resulting array

Files with mixed data types

```
>>> filename = 'titanic.csv'
>>> data = np.genfromtxt(filename,
                         delimiter=',',
                         names=True,
                         dtype=None)
```

Look for column header

```
>>> data_array = np.recfromcsv(filename)
```

The default `dtype` of the `np.recfromcsv()` function is `None`.

Importing Flat Files with pandas

```
>>> filename = 'winequality-red.csv'
>>> data = pd.read_csv(filename,
                       nrows=5,
                       header=None,
                       sep='\t',
                       comment='#',
                       na_values[""])
```

Number of rows of file to read
Row number to use as col names
Delimiter to use
Character to split comments
String to recognize as NA/NaN

Excel Spreadsheets

```
>>> file = 'urbanpop.xlsx'
>>> data = pd.ExcelFile(file)
>>> df_sheet2 = data.parse('1960-1966',
                           skiprows=[0],
                           names=['Country',
                                  'AAM: War(2002)'])
>>> df_sheet1 = data.parse(0,
                           parse_cols=[0],
                           skiprows=[0],
                           names=['Country'])
```

To access the sheet names, use the `sheet_names` attribute:

```
>>> data.sheet_names
```

SAS Files

```
>>> from sas7bdat import SAS7BDAT
>>> with SAS7BDAT('urbanpop.sas7bdat') as file:
        df_sas = file.to_data_frame()
```

Stata Files

```
>>> data = pd.read_stata('urbanpop.dta')
```

Relational Databases

```
>>> from sqlalchemy import create_engine
>>> engine = create_engine('sqlite://Northwind.sqlite')
```

Use the `table_names()` method to fetch a list of table names:

```
>>> table_names = engine.table_names()
```

Querying Relational Databases

```
>>> con = engine.connect()
>>> rs = con.execute("SELECT * FROM Orders")
>>> df = pd.DataFrame(rs.fetchall())
>>> df.columns = rs.keys()
>>> con.close()
```

Using the context manager `with`

```
>>> with engine.connect() as con:
        rs = con.execute("SELECT OrderID FROM Orders")
        df = pd.DataFrame(rs.fetchmany(size=5))
        df.columns = rs.keys()
```

Querying relational databases with pandas

```
>>> df = pd.read_sql_query("SELECT * FROM Orders", engine)
```

Exploring Your Data

NumPy Arrays

```
>>> data_array.dtype
>>> data_array.shape
>>> len(data_array)
```

Data type of array elements
Array dimensions
Length of array

pandas DataFrames

```
>>> df.head()
>>> df.tail()
>>> df.index
>>> df.columns
>>> df.info()
>>> data_array = data.values
```

Return first DataFrame rows
Return last DataFrame rows
Describe index
Describe DataFrame columns
Info on DataFrame
Convert a DataFrame to an a NumPy array

Pickled Files

```
>>> import pickle
>>> with open('pickled_fruit.pkl', 'rb') as file:
        pickled_data = pickle.load(file)
```

HDF5 Files

```
>>> import h5py
>>> filename = 'H-H1_LOSC_4_v1-815411200-4096.hdf5'
>>> data = h5py.File(filename, 'r')
```

Matlab Files

```
>>> import scipy.io
>>> filename = 'workspace.mat'
>>> mat = scipy.io.loadmat(filename)
```

Exploring Dictionaries

Accessing Elements with Functions

```
>>> print(mat.keys())
>>> for key in data.keys():
        print(key)
meta
quality
strain
>>> pickled_data.values()
>>> print(mat.items())
```

Print dictionary keys
Print dictionary keys

Return dictionary values
Returns items in list format of (key, value) tuple pairs

Accessing Data Items with Keys

```
>>> for key in data ['meta'].keys()
        print(key)
Description
DescriptionURL
Detector
Duration
GPSstart
Observatory
Type
UTCstart
>>> print(data['meta']['Description'].value)
```

Explore the HDF5 structure

Retrieve the value for a key

Navigating Your FileSystem

Magic Commands

```
!ls
%cd ..
%pwd
```

List directory contents of files and directories
Change current working directory
Return the current working directory path

`os` Library

```
>>> import os
>>> path = "/usr/tmp"
>>> wd = os.getcwd()
>>> os.listdir(wd)
>>> os.chdir(path)
>>> os.rename("test1.txt",
              "test2.txt")
>>> os.remove("test1.txt")
>>> os.mkdir("newdir")
```

Store the name of current directory in a string
Output contents of the directory in a list
Change current working directory
Rename a file

Delete an existing file
Create a new directory

Python For Data Science *Cheat Sheet*

Matplotlib

Learn Python Interactively at www.DataCamp.com

Matplotlib

Matplotlib is a Python 2D plotting library which produces publication-quality figures in a variety of hardcopy formats and interactive environments across platforms.

matplotlib

1 Prepare The Data

Also see Lists & NumPy

1D Data

```
>>> import numpy as np
>>> x = np.linspace(0, 10, 100)
>>> y = np.cos(x)
>>> z = np.sin(x)
```

2D Data or Images

```
>>> data = 2 * np.random.random((10, 10))
>>> data2 = 3 * np.random.random((10, 10))
>>> Y, X = np.mgrid[-3:3:100j, -3:3:100j]
>>> U = -1 - X**2 + Y
>>> V = 1 + X - Y**2
>>> from matplotlib.cbook import get_sample_data
>>> img = np.load(get_sample_data('axes_grid/bivariate_normal.npy'))
```

2 Create Plot

```
>>> import matplotlib.pyplot as plt
```

Figure

```
>>> fig = plt.figure()
>>> fig2 = plt.figure(figsize=plt.figaspect(2.0))
```

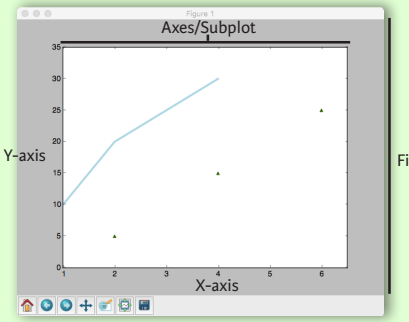
Axes

All plotting is done with respect to an Axes. In most cases, a subplot will fit your needs. A subplot is an axes on a grid system.

```
>>> fig.add_axes()
>>> ax1 = fig.add_subplot(221) # row-col-num
>>> ax3 = fig.add_subplot(212)
>>> fig3, axes = plt.subplots(nrows=2,ncols=2)
>>> fig4, axes2 = plt.subplots(ncols=3)
```

Plot Anatomy & Workflow

Plot Anatomy



Figure

Workflow

The basic steps to creating plots with matplotlib are:

1 Prepare data 2 Create plot 3 Plot 4 Customize plot 5 Save plot 6 Show plot

```
>>> import matplotlib.pyplot as plt
>>> x = [1,2,3,4]                          Step 1
>>> y = [10,20,25,30]
>>> fig = plt.figure()                     Step 2
>>> ax = fig.add_subplot(111)              Step 3
>>> ax.plot(x, y, color='lightblue', linewidth=3)   Step 3, 4
>>> ax.scatter([2,4,6],
               [5,15,25],
               color='darkgreen',
               marker='^')
>>> ax.set_xlim(1, 6.5)
>>> plt.savefig('foo.png')
>>> plt.show()                             Step 6
```

4 Customize Plot

Colors, Color Bars & Color Maps

```
>>> plt.plot(x, x, x, x**2, x, x**3)
>>> ax.plot(x, y, alpha = 0.4)
>>> ax.plot(x, y, c='k')
>>> fig.colorbar(im, orientation='horizontal')
>>> im = ax.imshow(img,
                   cmap='seismic')
```

Markers

```
>>> fig, ax = plt.subplots()
>>> ax.scatter(x,y,marker=".")
>>> ax.plot(x,y,marker="o")
```

Linestyles

```
>>> plt.plot(x,y,linewidth=4.0)
>>> plt.plot(x,y,ls='solid')
>>> plt.plot(x,y,ls='--')
>>> plt.plot(x,y,'--',x**2,y**2,'-.')
>>> plt.setp(lines,color='r',linewidth=4.0)
```

Text & Annotations

```
>>> ax.text(1,
            -2.1,
            'Example Graph',
            style='italic')
>>> ax.annotate("Sine",
                xy=(8, 0),
                xycoords='data',
                xytext=(10.5, 0),
                textcoords='data',
                arrowprops=dict(arrowstyle="->",
                                connectionstyle="arc3"),)
```

Mathtext

```
>>> plt.title(r'$sigma_i=15$', fontsize=20)
```

Limits, Legends & Layouts

Limits & Autoscaling

```
>>> ax.margins(x=0.0,y=0.1)                  Add padding to a plot
>>> ax.axis('equal')                         Set the aspect ratio of the plot to 1
>>> ax.set(xlim=[0,10.5],ylim=[-1.5,1.5])    Set limits for x-and y-axis
>>> ax.set_xlim(0,10.5)                      Set limits for x-axis
```

Legends

```
>>> ax.set(title='An Example Axes',          Set a title and x-and y-axis labels
           ylabel='Y-Axis',
           xlabel='X-Axis')
>>> ax.legend(loc='best')                    No overlapping plot elements
```

Ticks

```
>>> ax.xaxis.set(ticks=range(1,5),           Manually set x-ticks
                 ticklabels=[3,100,-12,"foo"])
>>> ax.tick_params(axis='y',                 Make y-ticks longer and go in and out
                   direction='inout',
                   length=10)
```

Subplot Spacing

```
>>> fig3.subplots_adjust(wspace=0.5,         Adjust the spacing between subplots
                         hspace=0.3,
                         left=0.125,
                         right=0.9,
                         top=0.9,
                         bottom=0.1)
>>> fig.tight_layout()                       Fit subplot(s) in to the figure area
```

Axis Spines

```
>>> ax1.spines['top'].set_visible(False)     Make the top axis line for a plot invisible
>>> ax1.spines['bottom'].set_position(('outward',10))   Move the bottom axis line outward
```

3 Plotting Routines

1D Data

```
>>> lines = ax.plot(x,y)                     Draw points with lines or markers connecting them
>>> ax.scatter(x,y)                          Draw unconnected points, scaled or colored
>>> axes[0,0].bar([1,2,3],[3,4,5])           Plot vertical rectangles (constant width)
>>> axes[1,0].barh([0.5,1,2.5],[0,1,2])      Plot horiontal rectangles (constant height)
>>> axes[1,1].axhline(0.45)                  Draw a horizontal line across axes
>>> axes[0,1].axvline(0.65)                  Draw a vertical line across axes
>>> ax.fill(x,y,color='blue')                Draw filled polygons
>>> ax.fill_between(x,y,color='yellow')      Fill between y-values and o
```

Vector Fields

```
>>> axes[0,1].arrow(0,0,0.5,0.5)             Add an arrow to the axes
>>> axes[1,1].quiver(y,z)                    Plot a 2D field of arrows
>>> axes[0,1].streamplot(X,Y,U,V)            Plot 2D vector fields
```

Data Distributions

```
>>> ax1.hist(y)                              Plot a histogram
>>> ax3.boxplot(y)                           Make a box and whisker plot
>>> ax3.violinplot(z)                        Make a violin plot
```

2D Data or Images

```
>>> fig, ax = plt.subplots()
>>> im = ax.imshow(img,                      Colormapped or RGB arrays
                   cmap='gist_earth',
                   interpolation='nearest',
                   vmin=-2,
                   vmax=2)
```

```
>>> axes2[0].pcolor(data2)                   Pseudocolor plot of 2D array
>>> axes2[0].pcolormesh(data)                Pseudocolor plot of 2D array
>>> CS = plt.contour(Y,X,U)                  Plot contours
>>> axes2[2].contourf(data1)                 Plot filled contours
>>> axes2[2]= ax.clabel(CS)                  Label a contour plot
```

5 Save Plot

Save figures

```
>>> plt.savefig('foo.png')
```

Save transparent figures

```
>>> plt.savefig('foo.png', transparent=True)
```

6 Show Plot

```
>>> plt.show()
```

Close & Clear

```
>>> plt.cla()                                Clear an axis
>>> plt.clf()                                Clear the entire figure
>>> plt.close()                              Close a window
```

Data Science Cheat Sheet

Python - Intermediate

KEY BASICS, PRINTING AND GETTING HELP

This cheat sheet assumes you are familiar with the content of our Python Basics Cheat Sheet

`s` - A Python string variable
`i` - A Python integer variable
`f` - A Python float variable
`l` - A Python list variable
`d` - A Python dictionary variable

LISTS

`l.pop(3)` - Returns the fourth item from `l` and deletes it from the list
`l.remove(x)` - Removes the first item in `l` that is equal to `x`
`l.reverse()` - Reverses the order of the items in `l`
`l[1::2]` - Returns every second item from `l`, commencing from the `1`st item
`l[-5:]` - Returns the last 5 items from `l` specific axis

STRINGS

`s.lower()` - Returns a lowercase version of `s`
`s.title()` - Returns `s` with the first letter of every word capitalized
`"23".zfill(4)` - Returns `"0023"` by left-filling the string with `0`'s to make it's length `4`.
`s.splitlines()` - Returns a list by splitting the string on any newline characters.

Python strings share some common methods with lists

`s[:5]` - Returns the first `5` characters of `s`
`"fri" + "end"` - Returns `"friend"`
`"end" in s` - Returns `True` if the substring `"end"` is found in `s`

RANGE

Range objects are useful for creating sequences of integers for looping.

`range(5)` - Returns a sequence from `0` to `4`
`range(2000,2018)` - Returns a sequence from `2000` to `2017`
`range(0,11,2)` - Returns a sequence from `0` to `10`, with each item incrementing by `2`
`range(0,-10,-1)` - Returns a sequence from `0` to `-9`
`list(range(5))` - Returns a list from `0` to `4`

DICTIONARIES

`max(d, key=d.get)` - Return the key that corresponds to the largest value in `d`
`min(d, key=d.get)` - Return the key that corresponds to the smallest value in `d`

SETS

`my_set = set(l)` - Return a `set` object containing the unique values from `l`
`len(my_set)` - Returns the number of objects in `my_set` (or, the number of unique values from `l`)
`a in my_set` - Returns `True` if the value `a` exists in `my_set`

REGULAR EXPRESSIONS

`import re` - Import the Regular Expressions module
`re.search("abc",s)` - Returns a `match` object if the regex `"abc"` is found in `s`, otherwise `None`
`re.sub("abc","xyz",s)` - Returns a string where all instances matching regex `"abc"` are replaced by `"xyz"`

LIST COMPREHENSION

A one-line expression of a for loop

`[i ** 2 for i in range(10)]` - Returns a list of the squares of values from `0` to `9`
`[s.lower() for s in l_strings]` - Returns the list `l_strings`, with each item having had the `.lower()` method applied
`[i for i in l_floats if i < 0.5]` - Returns the items from `l_floats` that are less than `0.5`

FUNCTIONS FOR LOOPING

```
for i, value in enumerate(l):
  print("The value of item {} is {}".
  format(i,value))
```

- Iterate over the list `l`, printing the index location of each item and its value

```
for one, two in zip(l_one,l_two):
  print("one: {}, two: {}".format(one,two))
```

- Iterate over two lists, `l_one` and `l_two` and print each value

```
while x < 10:
  x += 1
```

- Run the code in the body of the loop until the value of `x` is no longer less than `10`

DATETIME

`import datetime as dt` - Import the `datetime` module
`now = dt.datetime.now()` - Assign `datetime` object representing the current time to `now`
`wks4 = dt.datetime.timedelta(weeks=4)` - Assign a `timedelta` object representing a timespan of 4 weeks to `wks4`
`now - wks4` - Return a `datetime` object representing the time 4 weeks prior to `now`
`newyear_2020 = dt.datetime(year=2020, month=12, day=31)` - Assign a `datetime` object representing December 25, 2020 to `newyear_2020`
`newyear_2020.strftime("%A, %b %d, %Y")` - Returns `"Thursday, Dec 31, 2020"`
`dt.datetime.strptime('Dec 31, 2020',"%b %d, %Y")` - Return a `datetime` object representing December 31, 2020

RANDOM

`import random` - Import the `random` module
`random.random()` - Returns a random float between `0.0` and `1.0`
`random.randint(0,10)` - Returns a random integer between `0` and `10`
`random.choice(l)` - Returns a random item from the list `l`

COUNTER

`from collections import Counter` - Import the `Counter` class
`c = Counter(l)` - Assign a `Counter` (dict-like) object with the counts of each unique item from `l`, to `c`
`c.most_common(3)` - Return the 3 most common items from `l`

TRY/EXCEPT

Catch and deal with Errors

`l_ints = [1, 2, 3, "", 5]` - Assign a list of integers with one missing value to `l_ints`

```
l_floats = []
for i in l_ints:
  try:
    l_floats.append(float(i))
  except:
    l_floats.append(i)
```

- Convert each value of `l_ints` to a float, catching and handling `ValueError: could not convert string to float:` where values are missing.