

Python for Data Science - Cheat Sheet

esuriddick

Jupyter Notebook

Tooltip: Shift+Tab

Run current cell and select next cell: Shift+Enter

Run selected cells: Ctrl+Enter

Run current cell and insert cell below: Alt+Enter

Save and checkpoint: Ctrl+S

Stop loop: 'Kernel' -> 'Restart'

Basics

Data Types

- Numbers (integers vs. floats)
- Strings [single quotes vs. double quotes (later can store single quotes inside)]
- Lists (values are stored between '[' and ']', and it is the Python equivalent to a traditional array)
- Sets (values are stored between '{' and '}')
- Dictionaries (values are stored between '{' and '}', and a <key> is associated to a <value> with ':')
- Tuples (values are stored between '(' and ')', and data included in it is immutable)
- Booleans (True or False)

Variable assignment

<variable-name> = <number/data/variable>

Note: Name of the variable cannot start with a special character or number.

Operators

Symbol	Meaning
==	Equals
>	Greater than
<	Lower than
>=	Greater than or equal
<=	Lower than or equal
and	Both conditions must be true
or	One of the conditions must be true
in	Whether value is in object

Lambda Expression

lambda <parameter>: actions with <parameter>

Functions

Custom

```
def <function-name>(<parameter-01>=<default-value>, ...):  
    <action(s)>  
    return(<output>)
```

Print

```
print('<text/variable>')  
print('My number is: {<variable-to-retrieve-01>}, and my name is: {<variable-to-retrieve-02>}'.format(<variable-to-retrieve-01>=<variable-01>,<variable-to-retrieve-02>=<variable-02>))
```

Append / Add

```
<list/dictionary>.append(<item-to-add>)  
<set>.add(<item-to-add>)
```

Range (create sequence of values)

range(<maximum-number>) (starts at 0)

List (can be used on strings, tuples or lists)

list(<value(s)/object>)

Map (apply function over each value)

list(map(<function>,<object>))

Filter (filters values in an object)

list(filter(<function>,<object>))

Note: Function must be a boolean (0 or 1).

Lower (lowercase every single letter in a string)

<string>.lower()

Upper (uppercases every single letter in a string)

<string>.upper()

Split [removes white space (default) in a string]

<string>.split('<separator>')

Keys (returns keys from a dictionary)

<dictionary>.keys()

Items (returns items from a dictionary)

<dictionary>.items()

Values (returns values from a dictionary)

<dictionary>.values()

Pop (removes permanently value from list)

<list>.pop(<position-of-value>)

Note: By default, the last value is removed. In addition, by assigning the above function to a variable, the excluded value will be stored in the variable.

Unpack tuple

Example: x = [(1,2), (3,4), (5,6)]

for (a,b) in x:

print(a)

print(b)

Maths ('import math' for a lot more)

- abs(<object>)

- max/min(<object>)

Conditional Processing

if <condition(s)>:

 <action(s)>

elif <condition(s)>:

 <action(s)>

else:

 <action(s)>

Loops

For

for <variable> in <list/dictionary/set/tuple>:

 <action(s)>

While

while <variable> <comparison-operator> <value>:

 <action(s)>

 <variable> += <increment-value>

NumPy (Linear Algebra Library)

Library

import numpy as np

Arrays (can do arithmetic operations over these)

my_list = [1,2,3] / my_mat = [[1,2], [3,4], [5,6]]

Vectors: 1-d arrays - np.array(my_list)

Matricces: 2-d arrays - np.array(my_mat)

Arrays Generation

np.zeros (create a vector or matrix of zeros)

np.zeros(<length>) (vector)

np.zeros((<length-01>,<length-02>)) (matrix)

np.ones (create a vector or matrix of ones)

np.ones(<length>) (vector)

np.ones((<length-01>,<length-02>)) (matrix)

np.eye (creates an identity matrix)

np.eye(<dimension>)

np.arange (create a vector - similar to range)

np.arange(<start-value>, <last-index>,
<increments-size>)

np.linspace (creates a vector with evenly spaced numbers in a specified interval)

np.linspace(<start-value>, <last-value>,
<length>)

Random Numbers Generation

Define Seed: np.random.seed(<number>)

Base Form: np.random.<method>

Common methods:

- **Uniform distribution** [0; 1[

rand(<length>) (vector)

rand(<length-01>, <length-02>) (matrix)

- **Standard normal distribution**

randn(<length>) (vector)

randn(<length-01>, <length-02>) (matrix)

- **Random integers** [low-value; high-value[

randint(<low-value>, <high-value>, <length>)
(vector)

randint(<low-value>, <high-value>, size =
(<length-01>, <length-02>)) (matrix)

Arrays Selection and Indexing

Selection (indices are optional)

<array>[<initial-index>:<final-index>+1] (vector)

<array>[<initial-index-01>:<final-index-01>+1,
<initial-index-02>:<final-index-02>+1] (matrix)

Replace Values (affects the original array when using a slice of another array)

<array>[<initial-index>:<final-index>] = <value>

Copy

<new-array>=<array>.copy()

Conditional Selection

<array>[<array> <comparison-operator> <value>]

Arrays Transformation

Data type

<array>.dtype

Shape

<array>.shape (number of rows and columns)

Reshape

<array>.reshape(<length-01>, <length-02>) (vector to matrix)

<array>.reshape(-1) (matrix to vector)

Maths (axis: array=None / columns=0 / rows=1)

- np.mean(<array>, <axis>) (returns the mean)

- np.std(<array>, <axis>) (returns the standard deviation)

- np.max/min(<array>, <axis>)

- np.argmax/argmin(<array>, <axis>) (returns index of the maximum/minimum)

- np.sum(<array>, <axis>) (returns the total sum)

- np.log(<array>) (natural logarithm)

- np.sqrt(<array>) (natural logarithm)

- np.absolute(<array>) (natural logarithm)

- np.sign(<array>) (returns sign of a number)

Pandas (built on top of NumPy)

Library

import pandas as pd

Data Types

- Series: same as a NumPy array, but rows can be indexed by labels and any data type might be used to fill a series
- DataFrames: set of series (each column/row is a series)

Series and DataFrames Generation

Series

```
labels = ['a', 'b', 'c'] & my_data = [10, 20, 30]
arr = np.array(my_data)
d = {'a':10, 'b':20, 'c':30}
pd.Series(data = my_data, index = labels) OR
pd.Series(data = arr, index = labels) OR
pd.Series(d)
```

DataFrames

```
labels = ['a', 'b', 'c', 'd', 'e'] & my_data =
np.random.randn(5,4) & columnLabels = ['W', 'X',
'Y', 'Z']
pd.DataFrame(data = my_data, index = labels,
columns = columnLabels)
```

DataFrames Selection and Indexing

Column creation or deletion (column axis=1)

```
<df-name>[<new-column>] = values / operation
with existing columns / etc.
<df-name>.drop('<column-label>', axis = 1, in-
place = False/True)
```

Note: By default, 'inplace' is set as False and, thus, your dataframe will not be updated with the drop command.

Selection

```
Column: <df-name>['<column-label>'] (extracts a
series)
Columns: <df-name>[['<column-label-01>',
'<column-label-02>', ...]] (extracts a dataframe)
Row: <df-name>.loc['<row-label>'] OR
<df-name>.iloc[<row-index>] (both extract a
series)
```

```
Subset: <df-name>.loc['<row-label>', '<column-
label>'] (returns a value) OR
<df-name>.loc[['<row-label-01>', '<row-label-
02>', ...], ['<column-label-01>', '<column-label-02>',
...]] (returns a dataframe)
```

Conditional Selection

```
<df-name>[<column/subset/dataframe>
<comparison-operator> <value>]
<df-name>[(<condition-01>) & (<condition-02>)
& ...] (single pipe operator for or)
```

Indexing

```
Reset index: <df-name>.reset_index(inplace =
False/True)
Set column as index: <df-
name>.set_index(<column-label>, inplace =
False/True)
```

Note: By default, 'inplace' is set as False and, thus, your dataframe will not be updated with the reset_index or set_index commands.

Missing Values

- Remove rows/columns with NA

```
<df-name>.dropna(axis = 0/1, thresh = <minimum-
number-of-non-NA-values>, inplace = False/True)
```
- Replace NA in rows/columns

```
<df-name>.fillna(value = <replace-NA-value>, axis
= 0/1, inplace = False/True)
```

Groupby

```
<df-name>.groupby('<column-
label>').<aggregate-function>() (e.g., mean, sum,
std, count, max, min, describe, etc.)
```

Merging, Joining and Concatenating

- Concatenate (equivalent to rbind or cbind in R)

```
pd.concat([<df-name-01>, <df-name-02>, ...])
```
- Merge (similar to SQL merging)

```
pd.merge(<left-df>, <right-df>, how =
'left'/'right'/'inner'/'outer', on = ['<column-label-
01>', '<column-label-02>', ...])
```
- Join (similar to merge but does not require a 'on' value)

```
<left-df>.join(<right-df>, how =
'left'/'right'/'inner'/'outer')
```

Functions

- <DataFrame>.**describe()**: returns descriptive statistics, excluding NA values
- <DataFrame>.**info()**: prints a concise summary
- <Series/DataFrame>.**count()**: counts non-NA cells for each column or row
- <DataFrame>.**corr()**: returns the pairwise correlation of columns, excluding NA/null values
- <Series/DataFrame>.**unique()**: returns an array of unique values
- <Series/DataFrame>.**nunique()**: returns the number of unique values
- <Series/DataFrame>.**value_counts()**: returns a series with the unique values and the number of repetitions
- <Series/DataFrame>.**sort_values**(by = '<column-label>', axis = 0/1): sorts the values either ascending (default) or descending ('by' parameter is not required for a Series)
- <Series/DataFrame>.**apply**(<function-name>): apply a (non-)custom function over each row (can use lambda expression)
- <Series/DataFrame>.**isnull**: returns a Series/Dataframe with boolean values indicating whether the value is NA
- <DataFrame>.**columns**: returns an Index object with the list of column names
- <DataFrame>.**index**: returns a RangeIndex object with the characterisation of the index

Pivot Table

```
<df-name>.pivot_table(values = '<column-label-01>',
index = '<column-label-02>', columns = '<column-label-03>')
(you may use lists to indicate more than one column in each parameter)
```

Import / Export Data

Location of workbook: pwd

- CSV/TXT

```
pd.read_csv('<filename>.csv/txt')
```

```
pd.to_csv('<filename>', index = False/True)
```

- Excel

```
pd.read_excel('<filename>.xlsx', sheetname = <sheet-
name>)
```

```
pd.to_excel('<filename>.xlsx', sheet_name = <sheet-
name>)
```

Matplotlib

Examples

Official Website

Library

```
import matplotlib.pyplot as plt
%matplotlib inline (to see plots within Jupyter Notebook)
plt.show() (at the end of a plot code to print the plot when outside of Jupyter Notebook)
```

Functional Form

```
plt.plot(<x-axis-data>, <y-axis-data>)
plt.xlabel(' <x-label-string>')
plt.ylabel(' <y-label-string>')
plt.title(' <title-string>')
```

Subplotting (Functional Form)

```
plt.subplot(<number-of-rows>,<number-of-columns>,<plot-id-01>)
plt.plot(x1,y1)
plt.subplot(<number-of-rows>,<number-of-columns>,<plot-id-02>)
plt.plot(x2,y2)
...
```

Object Oriented (OO)

```
fig = plt.figure()
axes = fig.add_axes([<left-margin-%>,<bottom-margin-%>,<canvas-width-%>,<canvas-height-%>])
axes.set_xlabel(' <x-label-string>')
axes.set_ylabel(' <y-label-string>')
axes.set_title(' <title-string>')
axes.plot(<x-axis-data>, <y-axis-data>)
```

Subplotting (OO)

```
fig = plt.figure()
axes1 = fig.add_axes([<left-margin-%>,<bottom-margin-%>,<canvas-width-%>,<canvas-height-%>])
axes1.set_title(' <title-string>')
axes2 = fig.add_axes([<left-margin-%>,<bottom-margin-%>,<canvas-width-%>,<canvas-height-%>])
axes2.set_title(' <title-string>')
axes1.plot(<x-axis-data>,<y-axis-data>)
axes2.plot(<x-axis-data>,<y-axis-data>)
```

Subplotting (OO + automatic margins/size)

```
fig,axes = plt.subplots(nrows=<number-rows>,<number-columns>)
plt.tight_layout() (to fix overlapping)
```

In order to loop over each graph in the axes array, you can do the following:

```
for current_ax in axes:
    current_ax.plot(<x-axis-data>, <y-axis-data>)
```

Attach plot(s) to specific index:

```
axes[<index>].plot(<x-axis-data>,<y-axis-data>)
```

Figure size and Dots Per Inch

```
fig = plt.figure(figsize=(<width-inches>,<height-inches>),dpi=<number>)
fig,axes = plt.subplots(figsize=(<width-inches>,<height-inches>),dpi=<number>)
```

Customisation

```
axes.plot(<x-axis-data>,<y-axis-data>,<color-or-RGB-code>,<marker>=<style> (options: o, +, *, 1),<markerfacecolour>=<color-or-RGB-code>,<markeredgecolour>=<color-or-RGB-code>,<markeredgewidth>=<number>,<markersize>=<number>,<linestyle>=<style> (options: -, - -, -. , :),<linewidth>=<number>,<alpha>=<number-0-to-1>)
```

Limits

```
axes.set_xlim([<lower-bound>,<upper-bound>])
axes.set_ylim([<lower-bound>,<upper-bound>])
```

Legend

```
axes.legend() (HERE for options)
```

Save chart

```
fig.savefig(' <filename>.<format>', dpi=<number>)
```

Scatter Plot

```
plt.scatter(<x-axis-data>,<y-axis-data>)
```

Histogram

```
plt.hist(<x-axis-data>,<y-axis-data>)
```

Boxplot

```
plt.boxplot(<x-axis-data>,<y-axis-data>,<vert>=True,<patch_artist>=True)
```

Seaborn

Examples

Official Website

Library

```
import seaborn as sns
%matplotlib inline (to see plots within Jupyter Notebook)
```

Distribution Plots

```
sns.distplot(<Series>, kde=<False/True>,
bins=<number>)
```

Joint Plots

```
sns.jointplot(x=<Series>, y=<Series>,
data=<Dataframe>, kind=<Plot-type>')
Examples for the joint plot: 'hex', 'reg', 'kde', etc.
```

Pair Plots

```
sns.pairplot(<Dataframe>, hue=<'categorical-
variable'>, palette=<'colour-scheme'>)
```

Rug Plots

```
sns.rugplot(<Series>)
```

KDE Plots

```
sns.kdeplot(<Series>)
```

Bar Plots - categorical variable for x

```
sns.barplot(x=<Series>, y=<Series>,
data=<Dataframe>, estimator=<function>)
```

Count Plots - categorical variable for x

```
sns.countplot(x=<Series>, data=<Dataframe>)
```

Box Plots - categorical variable for x

```
sns.boxplot(x=<Series>, y=<Series>,
data=<Dataframe>, hue=<'categorical-variable'>)
```

Violin Plots - categorical variable for x

```
sns.violinplot(x=<Series>, y=<Series>,
data=<Dataframe>, hue=<'categorical-variable'>,
split=<True/False>)
```

Strip Plots - categorical variable for x

```
sns.stripplot(x=<Series>, y=<Series>,
data=<Dataframe>, jitter=<True/False>,
hue=<'categorical-variable'>, split=<True/False>)
```

Swarm Plots - categorical variable for x

```
sns.swarmplot(x=<Series>, y=<Series>,
data=<Dataframe>, color=<color-name>)
```

Factor Plots - categorical variable for x

```
sns.factorplot(x=<Series>, y=<Series>,
data=<Dataframe>, kind=<'Plot-type'>)
Examples for the kind: 'bar', 'violin', etc.
```

Heat map

```
sns.heatmap(<matrix/pivot-table>,
annot=<True/False>, cmap=<'colour-scheme'>,
linecolor=<'colour'>, linewidths=<number>)
```

Cluster map

```
sns.clustermap(<matrix/pivot-table>,
annot=<True/False>, cmap=<'colour-scheme'>,
linecolor=<'colour'>, linewidths=<number>,
standard_scale=<0/1>)
```

Pair Grids

```
g = sns.PairGrid(<Dataframe>)
g.map_diag(<plot-function>)
g.map_upper(<plot-function>)
g.map_lower(<plot-function>)
Examples of plot functions: sns.distplot, sns.kdeplot,
plt.scatter, etc.
```

Facet Grids

```
g = sns.FacetGrid(data=<Dataframe>,
col=<'column-variable'>, row=<'row-variable'>)
g.map(<plot-function>, <'variable-to-separate-per-
column-and-row'>, <additional-arguments>)
Examples of plot functions: sns.distplot, sns.kdeplot,
plt.scatter, etc.
```

Linear Model Plots

```
sns.lmplot(x=<Series>, y=<Series>,
data=<Dataframe>, hue=<'categorical-variable'>,
markers=<'style'> (options: o, +, *, 1))
Instead of 'hue', you could use 'col' or 'row' as if
building a Facet Grid.
```

Design

Font

```
sns.set(font=<'font-name'>)
```

Style

```
sns.set_style(<'style'>)
Style options: white, ticks, darkgrid or whitegrid.
```

Remove Border(s)

```
sns.despine(top=<True/False>, right=<True/False>,
left=<True/False>, bottom=<True/False>)
```

Figure Size

```
plt.figure(figsize=(<width>, <height>))
```

Scale and Context

```
sns.set_context(context=<'format'>,
font_scale=<multiplier-of-format-default-size>)
Examples of formats: paper, notebook, talk or poster.
```

Colour Palettes

Inside of a plot function: palette = <'colour-scheme'>.
Examples of colour palettes can be found [HERE](#).