Python Cheat Sheet

Pandas | Numpy | Sklearn
Matplotlib | Seaborn
BS4 | Selenium | Scrapy

by Frank Andrade

Pandas Cheat Sheet

Pandas provides data analysis tools for Python. All of the following code examples refer to the dataframe below.

df =

	col1	col2
A	1	4
B	2	5
C	3	6

axis 1 →

axis 0 →

Getting Started

Import pandas:
```
import pandas as pd
```

Create a series:
```
s = pd.Series([1, 2, 3],
              index=['A', 'B', 'C'],
              name='col1')
```

Create a dataframe:
```
data = [[1, 4], [2, 5], [3, 6]]
index = ['A', 'B', 'C']
df = pd.DataFrame(data, index=index,
                  columns=['col1', 'col2'])
```

Load a dataframe:
```
df = pd.read_csv('filename.csv', sep=',',
                 names=['col1', 'col2'],
                 index_col=0,
                 encoding='utf-8',
                 nrows=3)
```

Selecting rows and columns

Select single column:
```
df['col1']
```

Select multiple columns:
```
df[['col1', 'col2']]
```

Show first n rows:
```
df.head(2)
```

Show last n rows:
```
df.tail(2)
```

Select rows by index values:
```
df.loc['A'] df.loc[['A', 'B']]
```

Select rows by position:
```
df.loc[1] df.loc[1:]
```

Data wrangling

Filter by value:
```
df[df['col1'] > 1]
```

Sort by columns:
```
df.sort_values(['col2', 'col2'],
               ascending=[False, True])
```

Identify duplicate rows:
```
df.duplicated()
```

Identify unique rows:
```
df['col1'].unique()
```

Swap rows and columns:
```
df = df.transpose()
df = df.T
```

Drop a column:
```
df = df.drop('col1', axis=1)
```

Clone a data frame:
```
clone = df.copy()
```

Connect multiple data frames vertically:
```
df2 = df + 5 #new dataframe
pd.concat([df,df2])
```

Merge multiple data frames horizontally:
```
df3 = pd.DataFrame([[1, 7],[8,9]],
                   index=['B', 'D'],
             columns=['col1', 'col3'])
#df3: new dataframe
```

Only merge complete rows (INNER JOIN):
```
df.merge(df3)
```

Left column stays complete (LEFT OUTER JOIN):
```
df.merge(df3, how='left')
```

Right column stays complete (RIGHT OUTER JOIN):
```
df.merge(df3, how='right')
```

Preserve all values (OUTER JOIN):
```
df.merge(df3, how='outer')
```

Merge rows by index:
```
df.merge(df3,left_index=True,
         right_index=True)
```

Fill NaN values:
```
df.fillna(0)
```

Apply your own function:
```
def func(x):
    return 2**x
df.apply(func)
```

Arithmetics and statistics

Add to all values:
```
df + 10
```

Sum over columns:
```
df.sum()
```

Cumulative sum over columns:
```
df.cumsum()
```

Mean over columns:
```
df.mean()
```

Standard deviation over columns:
```
df.std()
```

Count unique values:
```
df['col1'].value_counts()
```

Summarize descriptive statistics:
```
df.describe()
```

Hierarchical indexing

Create hierarchical index:
```
df.stack()
```

Dissolve hierarchical index:
```
df.unstack()
```

Aggregation

Create group object:
```
g = df.groupby('col1')
```

Iterate over groups:
```
for i, group in g:
    print(i, group)
```

Aggregate groups:
```
g.sum()
g.prod()
g.mean()
g.std()
g.describe()
```

Select columns from groups:
```
g['col2'].sum()
g[['col2', 'col3']].sum()
```

Transform values:
```
import math
g.transform(math.log)
```

Apply a list function on each group:
```
def strsum(group):
    return ''.join([str(x) for x in group.value])

g['col2'].apply(strsum)
```

Data export

Data as NumPy array:
```
df.values
```

Save data as CSV file:
```
df.to_csv('output.csv', sep=',')
```

Format a dataframe as tabular string:
```
df.to_string()
```

Convert a dataframe to a dictionary:
```
df.to_dict()
```

Save a dataframe as an Excel table:
```
df.to_excel('output.xlsx')
```

Visualization

Import matplotlib:
```
import matplotlib.pyplot as plt
```

Start a new diagram:
```
plt.figure()
```

Scatter plot:
```
df.plot.scatter('col1', 'col2',
                style='ro')
```

Bar plot:
```
df.plot.bar(x='col1', y='col2',
            width=0.7)
```

Area plot:
```
df.plot.area(stacked=True,
             alpha=1.0)
```

Box-and-whisker plot:
```
df.plot.box()
```

Histogram over one column:
```
df['col1'].plot.hist(bins=3)
```

Histogram over all columns:
```
df.plot.hist(bins=3, alpha=0.5)
```

Set tick marks:
```
labels = ['A', 'B', 'C', 'D']
positions = [1, 2, 3, 4]
plt.xticks(positions, labels)
plt.yticks(positions, labels)
```

Select area to plot:
```
plt.axis([0, 2.5, 0, 10]) # [from
x, to x, from y, to y]
```

Label diagram and axes:
```
plt.title('Correlation')
plt.xlabel('Nunstück')
plt.ylabel('Slotermeyer')
```

Save most recent diagram:
```
plt.savefig('plot.png')
plt.savefig('plot.png',dpi=300)
plt.savefig('plot.svg')
```

Find practical examples in these guides I made:
- Pandas Guide for Excel Users(link)
- Data Wrangling Guide (link)
- Regular Expression Guide (link)

Made by Frank Andrade frank-andrade.medium.com

NumPy Cheat Sheet

NumPy provides tools for working with arrays. All of the following code examples refer to the arrays below.

NumPy Arrays

Getting Started

Import numpy:

```
import numpy as np
```

Create arrays:

```
a = np.array([1,2,3])
b = np.array([(1.5,2,3), (4,5,6)], dtype=float)
c = np.array([[(1.5,2,3), (4,5,6)],
              [(3,2,1), (4,5,6)]],
             dtype = float)
```

Initial placeholders:

```
np.zeros((3,4)) #Create an array of zeros
np.ones((2,3,4),dtype=np.int16)
d = np.arange(10,25,5)
np.linspace( 0,2, 9)
e = np.full((2,2), 7)
f = np.eye(2)
np.random.random((2,2))
np.empty((3,2))
```

Saving & Loading On Disk:

```
np.save('my_array', a)
np.savez('array.npz', a, b)
np.load('my_array.npy')
```

Saving & Loading Text Files

```
np.loadtxt('my_file.txt')
np.genfromtxt('my_file.csv',
                  delimiter=',')
np.savetxt('myarray.txt', a,
                 delimiter=' ')
```

Inspecting Your Array

```
a.shape
len(a)
b.ndim
e.size
b.dtype #data type
b.dtype.name
b.astype(int) #change data type
```

Data Types

```
np.int64
np.float32
np.complex
np.bool
np.object
np.string_
np.unicode_
```

Array Mathematics

Arithmetic Operations

```
>>> g = a-b
 array([[-0.5, 0. , 0. ],
        [-3. , 3. , 3. ]])
>>> np.subtract(a,b)

>>> b+a
 array([[2.5, 4. , 6. ],
        [ 5. , 7. , 9. ]])
>>> np.add(b,a)

>>> a/b
 array([[ 0.66666667, 1. , 1. ],
        [ 0.25 , 0.4 , 0. 5 ]])
>>> np.divide(a,b)

>>> a*b
 array([[ 1 . 5, 4. , 9. ],
        [ 4. , 10. , 18. ]])
>>> np.multiply(a,b)

>>> np.exp(b)
>>> np.sqrt(b)
>>> np.sin(a)
>>> np.log(a)
>>> e.dot(f)
```

Aggregate functions:

```
a.sum()
a.min()
b.max(axis= 0)
b.cumsum(axis= 1) #Cumulative sum
a.mean()
b.median()
a.corrcoef() #Correlation coefficient
np.std(b) #Standard deviation
```

Copying arrays:

```
h = a.view() #Create a view
np.copy(a)
h = a.copy() #Create a deep copy
```

Sorting arrays:

```
a.sort() #Sort an array
c.sort(axis=0)
```

Array Manipulation

Transposing Array:

```
i = np.transpose(b)
i.T
```

Changing Array Shape:

```
b.ravel()
g.reshape(3,-2)
```

Adding/removing elements:

```
h.resize((2,6))
np.append(h,g)
np.insert(a, 1, 5)
np.delete(a,[1])
```

Combining arrays:

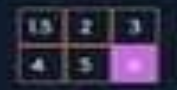
```
np.concatenate((a,d),axis=0)
np.vstack((a,b)) #stack vertically
np.hstack((e,f)) #stack horizontally
```

Splitting arrays:

```
np.hsplit(a,3) #Split horizontally
np.vsplit(c,2) #Split vertically
```

Subsetting

```
b[1,2]
```



Slicing:

```
a[0:2]
```

Boolean Indexing:

```
a[a<2]
```

Scikit-Learn Cheat Sheet

Sklearn is a free machine learning library for Python. It features various classification, regression and clustering algorithms.

Getting Started

The code below demonstrates the basic steps of using sklearn to create and run a model on a set of data.
The steps in the code include loading the data, splitting into train and test sets, scaling the sets, creating the model, fitting the model on the data using the trained model to make predictions on the test set, and finally evaluating the performance of the model.

```
from sklearn import neighbors,datasets,preprocessing
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
iris = datasets.load_iris()
X,y = iris.data[:,:2], iris.target
X_train, X_test, y_train, y_test=train_test_split(X,y)
scaler = preprocessing_StandardScaler().fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
knn = neighbors.KNeighborsClassifier(n_neighbors = 5)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
accuracy_score(y_test, y_pred)
```

Loading the Data

The data needs to be numeric and stored as NumPy arrays or SciPy spare matrix (numeric arrays, such as Pandas DataFrame's are also ok)

```
>>> import numpy as np
>>> X = np.random.random((10,5))
array([[0.21,0.33],
       [0.23, 0.60],
       [0.48, 0.62]])
>>> y = np.array(['A','B','A'])
array(['A', 'B', 'A'])
```

Training and Test Data

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,
random_state = 0)#Splits data into training and test set
```

Preprocessing The Data

Standardization

Standardizes the features by removing the mean and scaling to unit variance.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler().fit(X_train)
standarized_X = scaler.transform(X_train)
standarized_X_test = scaler.transform(X_test)
```

Normalization

Each sample (row of the data matrix) with at least one non-zero component is rescaled independently of other samples so that its norm equals one.

```
from sklearn.preprocessing import Normalizer
scaler = Normalizer().fit(X_train)
normalized_X = scaler.transform(X_train)
normalized_X_test = scaler.transform(X_test)
```

Binarization

Binarize data (set feature values to 0 or 1) according to a threshold.

```
from sklearn.preprocessing import Binarizer
binarizer = Binarizer(threshold = 0.0).fit(X)
binary_X = binarizer.transform(X_test)
```

Encoding Categorical Features

Imputation transformer for completing missing values.

```
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
le.fit_transform(X_train)
```

Imputing Missing Values

```
from sklearn.impute import SimpleImputer
imp = SimpleImputer(missing_values=0, strategy ='mean')
imp.fit_transform(X_train)
```

Generating Polynomial Features

```
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(5)
poly.fit_transform(X)
```

Find practical examples in these guides I made:
- Scikit-Learn Guide (link)
- Tokenize text with Python (link)
- Predicting Football Games (link)

Made by Frank Andrade frank-andrade.medium.com

Data Viz Cheat Sheet

Matplotlib is a Python 2D plotting library that produces figures in a variety of formats.

Matplotlib

Workflow

The basic steps to creating plots with matplotlib are Prepare Data, Plot, Customize Plot, Save Plot and Show Plot.

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

Example with lineplot

Prepare data

```
x = [2017, 2018, 2019, 2020, 2021]
y = [43, 45, 47, 48, 50]
```

Plot & Customize Plot

```
plt.plot(x,y,marker='o',linestyle='--',
color='g', label='USA')
plt.xlabel('Years')
plt.ylabel('Population (M)')
plt.title('Years vs Population')
plt.legend(loc='lower right')
plt.yticks([41, 45, 48, 51])
```

Save Plot

```
plt.savefig('example.png')
```

Show Plot

```
plt.show()
```

Markers: '.', 'o', 'v', '<', '>'

Line Styles: '-', '--', '-.', ':'

Colors: 'b', 'g', 'r', 'y' #blue, green, red, yellow

Barplot

```
x = ['USA', 'UK', 'Australia']
y = [40, 50, 33]
plt.bar(x, y)
plt.show()
```

Piechart

```
plt.pie(y, labels=x, autopct='%.0f %%')
plt.show()
```

Histogram

```
ages = [15, 16, 17, 30, 31, 32, 35]
bins = [15, 20, 25, 30, 35]
plt.hist(ages, bins, edgecolor='black')
plt.show()
```

Boxplots

```
ages = [15, 16, 17, 30, 31, 32, 35]
plt.boxplot(ages)
plt.show()
```

Scatterplot

```
a = [1, 2, 3, 4, 5, 4, 3 ,2, 5, 6, 7]
b = [7, 2, 3, 5, 5, 7, 3, 2, 6, 3, 2]
plt.scatter(a, b)
plt.show()
```

Subplots

Add the code below to make multple plots with 'n' number of rows and columns.

```
fig, ax = plt.subplots(nrows=1,
                       ncols=2,
                       sharey=True,
                       figsize=(12, 4))
```

Plot & Customize Each Graph

```
ax[0].plot(x, y, color='g')
ax[0].legend()
ax[1].plot(a, b, color='r')
ax[1].legend()
plt.show()
```

Find practical examples in these guides I made:
- Matplotlib & Seaborn Guide (link)
- Wordclouds Guide (link)
- Comparing Data Viz libraries(link)

Seaborn

Workflow

```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
```

Lineplot

```
plt.figure(figsize=(10, 5))
flights = sns.load_dataset("flights")
may_flights=flights.query("month=='May'")
ax = sns.lineplot(data=may_flights,
                  x="year",
                  y="passengers")
ax.set(xlabel='x', ylabel='y',
       title='my_title, xticks=[1,2,3])
ax.legend(title='my_legend,
          title_fontsize=13)
plt.show()
```

Barplot

```
tips = sns.load_dataset("tips")
ax = sns.barplot(x="day",
                 y="total_bill,
                 data=tips)
```

Histogram

```
penguins = sns.load_dataset("penguins")
sns.histplot(data=penguins,
             x="flipper_length_mm")
```

Boxplot

```
tips = sns.load_dataset("tips")
ax = sns.boxplot(x=tips["total_bill"])
```

Scatterplot

```
tips = sns.load_dataset("tips")
sns.scatterplot(data=tips,
                x="total_bill",
                y="tip")
```

Figure aesthetics

```
sns.set_style('darkgrid') #stlyes
sns.set_palette('husl', 3) #palettes
sns.color_palette('husl') #colors
```

Fontsize of the axes title, x and y labels, tick labels and legend:

```
plt.rc('axes', titlesize=18)
plt.rc('axes', labelsize=14)
plt.rc('xtick', labelsize=13)
plt.rc('ytick', labelsize=13)
plt.rc('legend', fontsize=13)
plt.rc('font', size=13)
```

Made by Frank Andrade frank-andrade.medium.com

Web Scraping Cheat Sheet

Web Scraping is the process of extracting data from a website. Before studying Beautiful Soup and Selenium, it's good to review some HTML basics first.

HTML for Web Scraping

Let's take a look at the HTML element syntax.

This is a single HTML element, but the HTML code behind a website has hundreds of them.

HTML code example

```
<article class="main-article">
 <h1> Titanic (1997) </h1>
 <p class="plot"> 84 years later ... </p>
 <div class="full-script"> 13 meters. You ... </div>
</article>
```

The HTML code is structured with "nodes". Each rectangle below represents a node (element, attribute and text nodes)

- "Siblings" are nodes with the same parent.
- A node's children and its children's children are called its "descendants". Similarly, a node's parent and its parent's parent are called its "ancestors".
- it's recommended to find element in this order.
  a. ID
  b. Class name
  c. Tag name
  d. Xpath

Beautiful Soup

Workflow

Importing the libraries

```
from bs4 import BeautifulSoup
import requests
```

Fetch the pages

```
result=requests.get("www.google.com")
result.status_code #get status code
result.headers #get the headers
```

Page content

```
content = result.text
```

Create soup

```
soup = BeautifulSoup(content,"lxml")
```

HTML in a readable format

```
print(soup.prettify())
```

Find an element

```
soup.find(id="specific_id")
```

Find elements

```
soup.find_all("a")
soup.find_all("a","css_class")
soup.find_all("a",class_="my_class")
soup.find_all("a",attrs={"class":
                         "my_class"})
```

Get inner text

```
sample = element.get_text()
sample = element.get_text(strip=True,
                          separator=' ')
```

Get specific attributes

```
sample = element.get('href')
```

XPath

We need to learn XPath to scrape with Selenium or Scrapy.

XPath Syntax

An XPath usually contains a tag name, attribute name, and attribute value.

```
//tagName[@AttributeName="Value"]
```

Let's check some examples to locate the article, title, and transcript elements of the HTML code we used before.

```
//article[@class="main-article"]
//h1
//div[@class="full-script"]
```

XPath Functions and Operators

XPath functions

```
//tag[contains(@AttributeName, "Value")]
```

XPath Operators: and, or

```
//tag[(expression 1) and (expression 2)]
```

XPath Special Characters

/	Selects the children from the node set on the left side of this character
//	Specifies that the matching node set should be located at any level within the document
.	Specifies the current context should be used (refers to present node)
..	Refers to a parent node
*	A wildcard character that selects all elements or attributes regardless of names
@	Select an attribute
()	Grouping an XPath expression
[n]	Indicates that a node with index "n" should be selected

Selenium

Workflow

```
from selenium import webdriver
web='www.google.com'
path='introduce chromedriver path'
driver = webdriver.Chrome(path)
driver.get(web)
```

Find an element
```
driver.find_element_by_id('name')
```

Find elements
```
driver.find_elements_by_class_name()
driver.find_elements_by_css_selector
driver.find_elements_by_xpath()
driver.find_elements_by_tag_name()
driver.find_elements_by_name()
```

Quit driver
```
driver.quit()
```

Getting the text
```
data = element.text
```

Implicit Waits
```
import time
time.sleep(2)
```

Explicit Waits
```
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

WebDriverWait(driver, 5).until(EC.element_to_be_clickable((By.ID,
'id_name'))) #Wait 5 seconds until an element is clickable
```

Options: Headless mode, change window size
```
from selenium.webdriver.chrome.options import Options
options = Options()
options.headless = True
options.add_argument('window-size=1920x1080')
driver=webdriver.Chrome(path,options=options)
```

Find practical examples in these guides I made:
- Web Scraping Complete Guide (link)
- Web Scraping with Selenium (link)
- Web Scraping with Beautiful Soup (link)

Made by Frank Andrade frank-andrade.medium.com

Scrapy

Scrapy is the most powerful web scraping framework in Python, but it's a bit complicated to set up, so check my guide or its documentation to set it up.

Creating a Project and Spider
To create a new project, run the following command in the terminal.
```
scrapy startproject my_first_spider
```
To create a new spider, first change the directory.
```
cd my_first_spider
```
Create an spider
```
scrapy genspider example example.com
```

The Basic Template
When you create a spider, you obtain a template with the following content.

```
import scrapy
class ExampleSpider(scrapy.Spider):
    name = 'example'
    allowed_domains = ['example.com']
    start_urls = ['http://example.com/']

    def parse(self, response):
        pass
```

Class

Parse method

The class is built with the data we introduced in the previous command, but the parse method needs to be built by us. To build it, use the functions below.

Finding elements
To find elements in Scrapy, use the response argument from the parse method
```
response.xpath('//tag[@AttributeName="Value"]')
```

Getting the text
To obtain the text element we use text() and either .get() or .getall(). For example:
```
response.xpath('//h1/text()').get()
response.xpath('//tag[@Attribute="Value"]/text()').getall()
```

Return data extracted
To see the data extracted we have to use the yield keyword

```
def parse(self, response):
    title = response.xpath('//h1/text()').get()

    # Return data extracted
    yield {'titles': title}
```

Run the spider and export data to CSV or JSON
```
scrapy crawl example
scrapy crawl example -o name_of_file.csv
scrapy crawl example -o name_of_file.json
```