Web Information Retrieval

Introduction to Search Engines

HeadMind Partners

AI & BLOCKCHAIN

CentraleSupélec

Preamble

In a constantly evolving digital world, search engines play a crucial role in enabling us to explore, analyze, and discover relevant information among the vast amount of data available online. Understanding the inner workings of these engines has become an essential skill for those who aspire to shape the future of information retrieval.

Beyond their apparent simplicity, these systems are actually complex and rely on sophisticated artificial intelligence models and advanced semantic analysis techniques. You will have the opportunity to delve into the underlying architectures that enable search engines to interpret user queries and provide real-time relevant results.

You will work on projects involving emerging technologies, real datasets, and realistic scenarios. This will give you the chance to develop your programming skills, natural language processing expertise, and artificial intelligence model design abilities.

We have selected a web-based data source accessible here: Stack Exchange Data Dump : Stack Exchange, Inc. : Free Download, Borrow, and Streaming : Internet Archive. By making the most of this data, your task for the week is to build a search engine that allows users to find information contained within forum posts. The various tables and metadata available will help you enhance the relevance of your search engine's results.

Table of Contents

Course of events

The labs will take place over 5 days. Here are the objectives of each day:

- **Day 1**: Exploring and Understanding Data
- **Day 2**: Creation of a search engine in Python
- **Day 3 and 4**: Improvement of the search engine by adding semantic features
- **Day 5**: Preparation of the report and other deliverables, oral presentations

Deliverables

Each group will be asked to deliver several things that will be used for the evaluation.

- ❖ Report (for day 5)

Present your work in a synthetic report (less than 10 pages). You should show and discuss your results, explain your choices and the algorithms/strategies you used in this report. You will also find in the different notebooks you will work on some questions that should be answered in the report. The deadline is day 5 (02/06).

- ❖ Oral presentation (for day 5)

Each group will have to prepare an oral presentation of around 10min (+10min discussion) for the last day (02/06). In this presentation you are asked to explain how you built your search engine, discuss your results and your choices. It must be centered on the demonstration of your presentation. Do not hesitate to be critical of your results, and propose potential improvements you have thought of. You can also explain how you worked as a team, the various difficulties you may have faced and how you overcame them.

- ❖ Code (before day 5)

Each group will be asked to create a gitlab repository for the team to put their final work on. By the end of the week this repo should be shared with us.

Special attention will be asked to the daily notebooks. The code should be clear, commented and the results highlighted and discussed. Notebooks should be stored **with the cells' outputs**.

In addition to the daily notebooks, you will be asked to complete a “main” notebook, building your own search engine based on what you will have done each day. The same attention will be asked to this notebook. Its readability, the performance of the search engine and the relevance of your remarks will be assessed.

If you are using your own PC and version of Python, you are asked to add an "Installation Folder" in the ReadMe. It will specify the version of Python used but also the different libraries to be installed via a requirements.txt file in order to run your codes. If you work locally, we strongly advise you to create your own virtual environment.

The ReadMe should summarize the objectives and the context of this project but also specify how to use your code in order to test them.

Data Source

The data source selected for this course is based on the Stack Exchange Q&A forum. This choice is motivated by the intention to confront you with challenges that come when using real data from the Web.

Here is the link to access the raw data:

Stack Exchange Data Dump : Stack Exchange, Inc. : Free Download, Borrow, and Streaming : Internet Archive

You can download the zip format of the Data Science sub-forum (allow about 200 MB of space).

Day by day

Day 1

Get started

First of all, you have to clone the Day 1 Notebook from the gitlab repository that will be provided. Most of the information you will need is already included in the notebook. This document recalls and completes these elements.

The goal of this first day is to go through the data and propose some ideas for the creation of your first search engine.

The data are from the Stack Exchange forum. It is available here :
https://archive.org/details/stackexchange

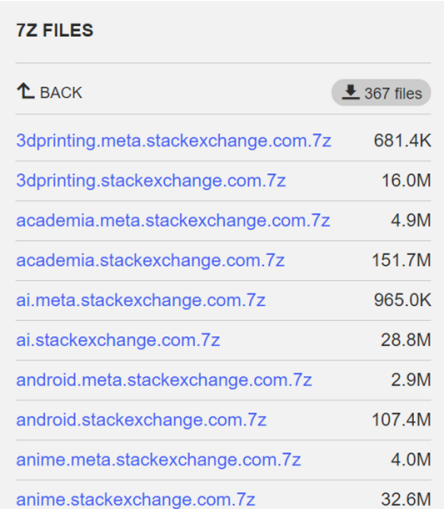
In order to gain time, you can already download the 7zip file of the data science forum (datascience.stackexchange.com.7z). If you work on Colab, we suggest you push it to a specific directory in your drive (it may take a few minutes to upload) and then mount this folder every time you need it from Colab.

Objectives

The objectives of this first day are:

- Query the data : explore it using sql queries on the data source web interface(Cheat sheet for SQL basics: SQL Basics).
- Extract the data : understand its format, the various files it is composed of.
- Explore the data: explore the posts, using the library pandas (Cheat sheet: Pandas, documentation: pandas - Python Data Analysis Library (pydata.org)).
- Visualize the data: using matplotlib and seaborn, build some useful visualization (cheat sheets: Matplotlib, Seaborn, documentation: https://matplotlib.org/stable/index.html, https://seaborn.pydata.org/).
- Suggest ideas for a powerful search engine: Using what you have learnt from the data, propose a structure for your search engine. You do not need to code it entirely yet, simply write down some ideas you may already have on how you could build a great search engine, which data to use and how to use it.

- View of a part of the zip files for each dataset:

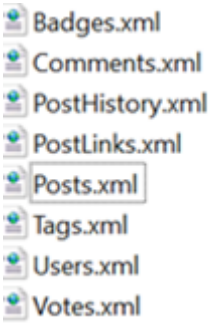


7Z FILES

BACK 367 files

3dprinting.meta.stackexchange.com.7z	681.4K
3dprinting.stackexchange.com.7z	16.0M
academia.meta.stackexchange.com.7z	4.9M
academia.stackexchange.com.7z	151.7M
ai.meta.stackexchange.com.7z	965.0K
ai.stackexchange.com.7z	28.8M
android.meta.stackexchange.com.7z	2.9M
android.stackexchange.com.7z	107.4M
anime.meta.stackexchange.com.7z	4.0M
anime.stackexchange.com.7z	32.6M

- View of the XML tables in one of the zip files above:



- View of first rows of Posts.xml file:


```
<?xml version="1.0" encoding="UTF-8"?>
- <posts>
    <row ContentLicense="CC BY-SA 4.0" CommentCount="4" AnswerCount="1"
      Tags="<dolphins><whistles>" Title="How many names can a dolphin remember?"
      LastActivityDate="2022-06-22T14:29:40.340" OwnerUserId="33" Body="<p><a
      href="https://www.science.org/content/article/dolphins-can-call-each-other-not-
      name-whistle" rel="noreferrer">This fascinating article in Science</a> says that each
      dolphin has it's own name (which is communicated as a &quot;whistle&quot;), and
      other dolphins can recognize each other by their &quot;name&quot;. However, do we
      have a rough idea about how many names a dolphin can know? For example, is there a
      dolphin that is known to recognize 100+ names?</p> " ViewCount="166" Score="10"
      CreationDate="2022-06-22T01:17:59.667" PostTypeId="1" Id="2"/>
    <row ContentLicense="CC BY-SA 4.0" CommentCount="1" AnswerCount="2"
      Tags="<distance><attenuation><methods>" Title="Can we measure distance from the
      animal harmonic calls?" LastActivityDate="2022-06-23T11:46:12.600" OwnerUserId="41"
      Body="<p>Harmonic calls have a fundamental frequency (f0) and integral multiples of
      that frequency as harmonic frequencies. Since the lower frequency sound wave has a
      low attenuation rate and the higher frequency band has high attenuation rate, is there
      any possibility to measure animal distance (with a single mic setup) depending on the
      attenuation rate of different frequency bands, now or in future?</p> " ViewCount="142"
```

Day 2

Get started

Clone the Day 2 Notebook from the same repository as the last time. You will now start to build the search engine. As a reminder, it should help users access information from the posts of the forum. Therefore, you will have to build an index of this post, and then to implement a search method on this index. Finally, you will estimate the relevance of your results using relevant scoring methods.

Objectives

- Design your first search engine with Python
- Build an index for the dataset
- Understand and compare different types of indexation techniques
- Implement the search method using the index
- Estimate the performance of the search engine

Day 3 & 4

Get started

Clone the Day 3 notebook. The goal now is to improve the performance of your search engine by implementing some NLP[1] features. You will work with various AI algorithms and cover several NLP tasks which should help you make the best choices for your search engine to be the most powerful.

This is the key part of making a great search engine, so do not hesitate to try many things. The notebook will guide you through your first steps, but it is up to you to make the best use of all that. It can be very difficult to build a search engine that always provides relevant results, whatever the query or topic… You will probably need to build a combination of several techniques to do so.

In the report and the notebooks (Day 3 and/or Search_engine.ipynb) be very exhaustive and critical about what you have tried, and why it worked or not.

Objectives

- Data Cleaning: Use regular expressions and other techniques to enhance the quality of the data
- Text-specific metadata: Use or create text-specific metadata and discuss how they would be useful for the search engine
- Preprocessing: Preprocess the textual data to make it better suited for the NLP models
- Semantic Similarity: Compare several texts with semantic AI models, compare queries with posts based on their semantic similarity to get the most relevant results
- Text Clustering: Use text clustering models to classify the texts in topics
- Incorporation in the search engine: Integrate the NLP techniques you have implemented to your search engine

[1] NLP for Natural Language Processing is a field of study focused on enabling computers to understand, interpret, and interact with human language.

Day 5

Day 5 will be the day you have to do the oral presentations. As a reminder, it should last around 15min, and be followed by a ~15min Q&A session. In this presentation, you are asked to explain how you worked as a team, what you have tried, what you have done in your final search engine and discuss your results.

You will also have to finish your report and your notebooks. Access to your team's gitlab repository should be granted to us by the end of the day. Don't forget to put your report in it or to send it to us by mail.

Information

We will be available every working day from 26/05 to 02/06 from 9h to 17h.

Here are our contact information:

- vgorce320@headmind.com
- vgillo155@headmind.com
- ebarrere220@headmind.com
- marsaoui463@headmind.com
- fvillenave318@headmind.com

Days of attendance

- Friday 26 (D1) Florent Vivien
- Tuesday 30 (D2) Florent Elise
- Wednesday 31 (D3) : Elise Mehdi
- Thursday 1 (D4): Vivien Mehdi
- Friday 2 (D5 - evaluation) : Valentin Mehdi

Appendix

Cheat sheets

SQL Basics:

datacamp

SQL for Data Science

SQL Basics Cheat Sheet

Learn SQL online at www.DataCamp.com

What is SQL?

SQL stands for "structured query language". It is a language used to query, analyze, and manipulate data from databases. Today, SQL is one of the most widely used tools in data.

> The different dialects of SQL

Although SQL languages all share a basic structure, some of the specific commands and styles can differ slightly. Popular dialects include MySQL, SQLite, SQL Server, Oracle SQL, and more. PostgreSQL is a good place to start —since it's close to standard SQL syntax and is easily adapted to other dialects.

> Sample Data

Throughout this cheat sheet, we'll use the columns listed in this sample table of `airbnb_listings`

airbnb_listings

id	city	country	number_of_rooms	year_listed
1	Paris	France	5	2018
2	Tokyo	Japan	2	2017
3	New York	USA	2	2022

> Querying tables

1. Get all the columns from a table

```
SELECT *
FROM airbnb_listings;
```

2. Return the `city` column from the table

```
SELECT city
FROM airbnb_listings;
```

3. Get the `city` and `year_listed` columns from the table

```
SELECT city, year_listed
FROM airbnb_listings;
```

4. Get the listing `id`, `city`, ordered by the `number_of_rooms` in ascending order

```
SELECT id, city
FROM airbnb_listings
ORDER BY number_of_rooms ASC;
```

5. Get the listing `id`, `city`, ordered by the `number_of_rooms` in descending order

```
SELECT id, city
FROM airbnb_listings
ORDER BY number_of_rooms DESC;
```

6. Get the first 5 rows from the `airbnb_listings` table

```
SELECT *
FROM airbnb_listings
LIMIT 5;
```

7. Get a unique list of cities where there are listings

```
SELECT DISTINCT city
FROM airbnb_lisitings;
```

> Filtering Data

Filtering on numeric columns

1. Get all the listings where `number_of_rooms` is more or equal to 3

```
SELECT *
FROM airbnb_listings
WHERE number_of_rooms >= 3;
```

2. Get all the listings where `number_of_rooms` is more than 3

```
SELECT *
FROM airbnb_listings
WHERE number_of_rooms > 3;
```

3. Get all the listings where `number_of_rooms` is exactly equal to 3

```
SELECT *
FROM airbnb_listings
WHERE number_of_rooms = 3;
```

4. Get all the listings where `number_of_rooms` is lower or equal to 3

```
SELECT *
FROM airbnb_listings
WHERE number_of_rooms <= 3;
```

5. Get all the listings where `number_of_rooms` is lower than 3

```
SELECT *
FROM airbnb_listings
WHERE number_of_rooms < 3;
```

6. Get all the listings with 3 to 6 rooms

```
SELECT *
FROM airbnb_listings
WHERE number_of_rooms BETWEEN 3 AND 6;
```

Filtering on text columns

7. Get all the listings that are based in `'Paris'`

```
SELECT *
FROM airbnb_listings
WHERE city = 'Paris';
```

8. Get the listings based in the `'USA'` and in `'France'`

```
SELECT *
FROM airbnb_listings
WHERE country IN ('USA', 'France');
```

9. Get all the listings where the city starts with `'j'` and where the city does not end in `'t'`

```
SELECT *
FROM airbnb_listings
WHERE city LIKE 'j%' AND city NOT LIKE '%t';
```

Filtering on multiple columns

10. Get all the listings in `'Paris'` where `number_of_rooms` is bigger than 3

```
SELECT *
FROM airbnb_listings
WHERE city = 'Paris' AND number_of_rooms > 3;
```

11. Get all the listings in `'Paris'` OR the ones that were listed after 2012

```
SELECT *
FROM airbnb_listings
WHERE city = 'Paris' OR year_listed > 2012;
```

Filtering on missing data

12. Return the listings where `number_of_rooms` is missing

```
SELECT *
FROM airbnb_listings
WHERE number_of_rooms IS NULL;
```

13. Return the listings where `number_of_rooms` is not missing

```
SELECT *
FROM airbnb_listings
WHERE number_of_rooms IS NOT NULL;
```

> Aggregating Data

Simple aggregations

1. Get the total number of rooms available across all listings

```
SELECT SUM(number_of_rooms)
FROM airbnb_listings;
```

2. Get the average number of rooms per listing across all listings

```
SELECT AVG(number_of_rooms)
FROM airbnb_listings;
```

3. Get the listing with the highest number of rooms across all listings

```
SELECT MAX(number_of_rooms)
FROM airbnb_listings;
```

4. Get the listing with the lowest number of rooms across all listings

```
SELECT MIN(number_of_rooms)
FROM airbnb_listings;
```

Grouping, filtering, and sorting

5. Get the total number of rooms for each country

```
SELECT country, SUM(number_of_rooms)
FROM airbnb_listings
GROUP BY country;
```

6. Get the average number of rooms for each country

```
SELECT country, AVG(number_of_rooms)
FROM airbnb_listings
GROUP BY country;
```

7. Get the listing with the maximum number of rooms per country

```
SELECT country, MAX(number_of_rooms)
FROM airbnb_listings
GROUP BY country;
```

8. Get the listing with the lowest amount of rooms per country

```
SELECT country, MIN(number_of_rooms)
FROM airbnb_listings
GROUP BY country;
```

9. For each country, get the average number of rooms per listing, sorted by ascending order

```
SELECT country, AVG(number_of_rooms) AS avg_rooms
FROM airbnb_listings
GROUP BY country
ORDER BY avg_rooms ASC;
```

10. For Japan and the USA, get the average number of rooms per listing in each country

```
SELECT country, AVG(number_of_rooms)
FROM airbnb_listings
WHERE country IN ('USA','Japan');
GROUP BY country;
```

11. Get the number of cities per country, where there are listings

```
SELECT country, COUNT(city) AS number_of_cities
FROM airbnb_listings
GROUP BY country;
```

12. Get all the years where there were more than 100 listings per year

```
SELECT year_listed
FROM airbnb_listings
GROUP BY year_listed
HAVING COUNT(id) > 100;
```

Learn Data Skills Online at www.DataCamp.com

Pandas:

Data Wrangling with pandas Cheat Sheet

http://pandas.pydata.org

Tidy Data – A foundation for wrangling in pandas

In a tidy data set:

&

Each **variable** is saved in its own **column**

Each **observation** is saved in its own **row**

Tidy data complements pandas's vectorized operations. pandas will automatically preserve observations as you manipulate variables. No other format works as intuitively with pandas.

M * A

Syntax – Creating DataFrames

	a	b	c
1	4	7	10
2	5	8	11
3	6	9	12

```
df = pd.DataFrame(
        {"a" : [4 ,5, 6],
         "b" : [7, 8, 9],
         "c" : [10, 11, 12]},
       index = [1, 2, 3])
```

Specify values for each column.

```
df = pd.DataFrame(
    [[4, 7, 10],
     [5, 8, 11],
     [6, 9, 12]],
    index=[1, 2, 3],
    columns=['a', 'b', 'c'])
```

Specify values for each row.

		a	b	c
n	v			
d	1	4	7	10
	2	5	8	11
e	2	6	9	12

```
df = pd.DataFrame(
        {"a" : [4 ,5, 6],
         "b" : [7, 8, 9],
         "c" : [10, 11, 12]},
index = pd.MultiIndex.from_tuples(
        [('d',1),('d',2),('e',2)],
          names=['n','v']))
```

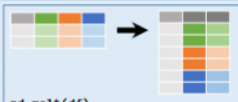
Create DataFrame with a MultiIndex

Method Chaining

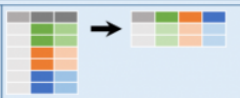
Most pandas methods return a DataFrame so that another pandas method can be applied to the result. This improves readability of code.

```
df = (pd.melt(df)
        .rename(columns={
                'variable' : 'var',
                'value' : 'val'})
        .query('val >= 200')
    )
```

Reshaping Data – Change the layout of a data set

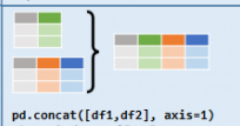


`pd.melt(df)`
Gather columns into rows.



`df.pivot(columns='var', values='val')`
Spread rows into columns.

`pd.concat([df1,df2])`
Append rows of DataFrames



`pd.concat([df1,df2], axis=1)`
Append columns of DataFrames

`df.sort_values('mpg')`
Order rows by values of a column (low to high).

`df.sort_values('mpg',ascending=False)`
Order rows by values of a column (high to low).

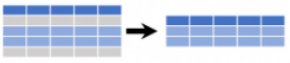
`df.rename(columns = {'y':'year'})`
Rename the columns of a DataFrame

`df.sort_index()`
Sort the index of a DataFrame

`df.reset_index()`
Reset index of DataFrame to row numbers, moving index to columns.

`df.drop(columns=['Length','Height'])`
Drop columns from DataFrame

Subset Observations (Rows)



`df[df.Length > 7]`
Extract rows that meet logical criteria.

`df.drop_duplicates()`
Remove duplicate rows (only considers columns).

`df.head(n)`
Select first n rows.

`df.tail(n)`
Select last n rows.

`df.sample(frac=0.5)`
Randomly select fraction of rows.

`df.sample(n=10)`
Randomly select n rows.

`df.iloc[10:20]`
Select rows by position.

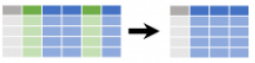
`df.nlargest(n, 'value')`
Select and order top n entries.

`df.nsmallest(n, 'value')`
Select and order bottom n entries.

Logic in Python (and pandas)

<	Less than	!=	Not equal to
>	Greater than	`df.column.isin(values)`	Group membership
==	Equals	`pd.isnull(obj)`	Is NaN
<=	Less than or equals	`pd.notnull(obj)`	Is not NaN
>=	Greater than or equals	`&,|,~,^,df.any(),df.all()`	Logical and, or, not, xor, any, all

Subset Variables (Columns)



`df[['width','length','species']]`
Select multiple columns with specific names.

`df['width']` *or* `df.width`
Select single column with specific name.

`df.filter(regex='regex')`
Select columns whose name matches regular expression *regex*.

regex (Regular Expressions) Examples

`'\.'`	Matches strings containing a period '.'
`'Length$'`	Matches strings ending with word 'Length'
`'^Sepal'`	Matches strings beginning with the word 'Sepal'
`'^x[1-5]$'`	Matches strings beginning with 'x' and ending with 1,2,3,4,5
`'^(?!Species$).*'`	Matches strings except the string 'Species'

`df.loc[:,'x2':'x4']`
Select all columns between x2 and x4 (inclusive).

`df.iloc[:,[1,2,5]]`
Select columns in positions 1, 2 and 5 (first column is 0).

`df.loc[df['a'] > 10, ['a','c']]`
Select rows meeting logical condition, and only the specific columns .

http://pandas.pydata.org/ This cheat sheet inspired by Rstudio Data Wrangling Cheatsheet (https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf) Written by Irv Lustig, Princeton Consultants

Matplotlib:

matplotlib Cheat sheet Version 3.5.0

Quick start

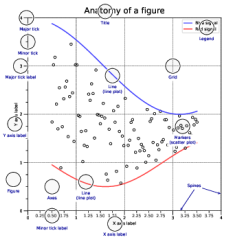
```
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt


X = np.linspace(0, 2*np.pi, 100)
Y = np.cos(X)

fig, ax = plt.subplots()
ax.plot(X, Y, color='green')

fig.savefig("figure.pdf")
fig.show()
```

Anatomy of a figure



Subplots layout

```
subplot[s](rows,cols,…)
fig, axs = plt.subplots(3, 3)

G = gridspec(rows,cols,…)
ax = G[0,:]

ax.inset_axes(extent)

d=make_axes_locatable(ax)
ax = d.new_horizontal('10%')
```

Getting help

- matplotlib.org
- github.com/matplotlib/matplotlib/issues
- discourse.matplotlib.org
- stackoverflow.com/questions/tagged/matplotlib
- https://gitter.im/matplotlib/matplotlib
- twitter.com/matplotlib
- Matplotlib users mailing list

Basic plots

```
plot([X],Y,[fmt],…)
```
X, **Y**, fmt, color, marker, linestyle

```
scatter(X,Y,…)
```
X, **Y**, [s]izes, [c]olors, marker, cmap

```
bar[h](x,height,…)
```
x, **height**, width, bottom, align, color

```
imshow(Z,…)
```
Z, cmap, interpolation, extent, origin

```
contour[f]([X],[Y],Z,…)
```
X, Y, **Z**, levels, colors, extent, origin

```
pcolormesh([X],[Y],Z,…)
```
X, Y, **Z**, vmin, vmax, cmap

```
quiver([X],[Y],U,V,…)
```
X, Y, **U**, **V**, C, units, angles

```
pie(X,…)
```
Z, explode, labels, colors, radius

```
text(x,y,text,…)
```
x, **y**, **text**, va, ha, size, weight, transform

```
fill[_between][x](…)
```
X, Y1, Y2, color, where

Advanced plots

```
step(X,Y,[fmt],…)
```
X, **Y**, fmt, color, marker, where

```
boxplot(X,…)
```
X, notch, sym, bootstrap, widths

```
errorbar(X,Y,xerr,yerr,…)
```
X, **Y**, xerr, yerr, fmt

```
hist(X, bins, …)
```
X, bins, range, density, weights

```
violinplot(D,…)
```
D, positions, widths, vert

```
barbs([X],[Y], U, V, …)
```
X, Y, **U**, **V**, C, length, pivot, sizes

```
eventplot(positions,…)
```
positions, orientation, lineoffsets

```
hexbin(X,Y,C,…)
```
X, **Y**, C, gridsize, bins

Scales

```
ax.set_[xy]scale(scale,…)
```
linear — any values
log — values > 0
symlog — any values
logit — 0 < values < 1

Projections

```
subplot(…,projection=p)
```
p='polar'
p='3d'

p=ccrs.Orthographic()
import cartopy.crs as ccrs

Lines

```
linestyle or ls
capstyle or dash_capstyle
"butt"
"round"
"projecting"
```

Markers

```
markevery
10
[0, -1]
(25, 5)
[0, 25, -1]
```

Colors

```
'Cn'
'x'
'name'
(R,G,B[,A])
'#RRGGBB[AA]'
'x.y'
```

Colormaps

```
plt.get_cmap(name)
```
Uniform: viridis, magma, plasma
Sequential: Greys, YlOrBr, Wistia
Diverging: Spectral, coolwarm, RdGy
Qualitative: tab10, tab20
Cyclic: twilight

Tick locators

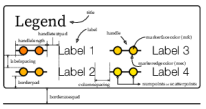
```
from matplotlib import ticker
ax.[xy]axis.set_[minor|major]_locator(locator)
ticker.NullLocator()
ticker.MultipleLocator(0.5)
ticker.FixedLocator([0, 1, 5])
ticker.LinearLocator(numticks=3)
ticker.IndexLocator(base=0.5, offset=0.25)
ticker.AutoLocator()
ticker.MaxNLocator(n=4)
ticker.LogLocator(base=10, numticks=15)
```

Tick formatters

```
from matplotlib import ticker
ax.[xy]axis.set_[minor|major]_formatter(formatter)
ticker.NullFormatter()
ticker.FixedFormatter(['zero', 'one', 'two', …])
ticker.FuncFormatter(lambda x, pos: "[%.2f]" % x)
ticker.FormatStrFormatter('>%d<')
ticker.ScalarFormatter()
ticker.StrMethodFormatter('{x}')
ticker.PercentFormatter(xmax=5)
```

Ornaments

```
ax.legend(…)
```
handles, labels, loc, title, frameon




```
ax.colorbar(…)
```
mappable, ax, cax, orientation

```
ax.annotate(…)
```
text, **xy**, **xytext**, xycoords, textcoords, arrowprops

Event handling

```
fig, ax = plt.subplots()
def on_click(event):
  print(event)
fig.canvas.mpl_connect(
  'button_press_event', on_click)
```

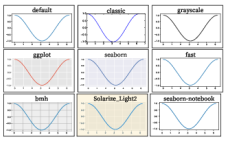
Animation

```
import matplotlib.animation as mpla

T = np.linspace(0, 2*np.pi, 100)
S = np.sin(T)
line, = plt.plot(T, S)
def animate(i):
    line.set_ydata(np.sin(T+i/50))
anim = mpla.FuncAnimation(
    plt.gcf(), animate, interval=5)
plt.show()
```

Styles

```
plt.style.use(style)
```



Quick reminder

```
ax.grid()
ax.set_[xy]lim(vmin, vmax)
ax.set_[xy]label(label)
ax.set_[xy]ticks(ticks, [labels])
ax.set_[xy]ticklabels(labels)
ax.set_title(title)
ax.tick_params(width=10, …)
ax.set_axis_[on|off]()

fig.suptitle(title)
fig.tight_layout()
plt.gcf(), plt.gca()
mpl.rc('axes', linewidth=1, …)
[fig|ax].patch.set_alpha(0)
text=r'$\frac{-e^{i\pi}}{2^n}$'
```

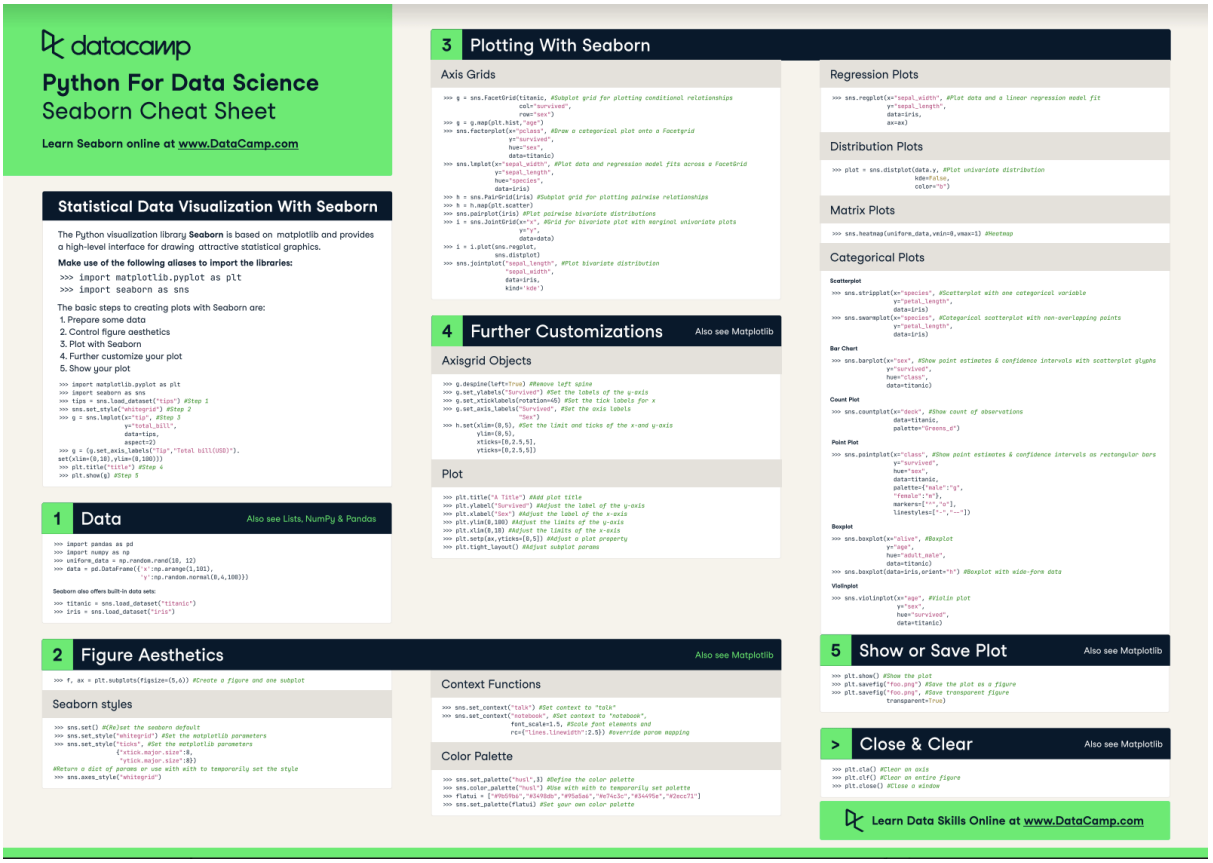
Keyboard shortcuts

ctrl + s	Save	ctrl + w	Close plot
r	Reset view	f	Fullscreen 0/1
f	View forward	b	View back
p	Pan view	o	Zoom to rect
x	X pan/zoom	y	Y pan/zoom
g	Minor grid 0/1	G	Major grid 0/1
l	X axis log/linear	L	Y axis log/linear

Ten simple rules

1. Know your audience
2. Identify your message
3. Adapt the figure
4. Captions are not optional
5. Do not trust the defaults
6. Use color effectively
7. Do not mislead the reader
8. Avoid "chartjunk"
9. Message trumps beauty
10. Get the right tool

Seaborn:



datacamp

Python For Data Science
Seaborn Cheat Sheet

Learn Seaborn online at www.DataCamp.com

Statistical Data Visualization With Seaborn

The Python visualization library **Seaborn** is based on matplotlib and provides a high-level interface for drawing attractive statistical graphics.

Make use of the following aliases to import the libraries:

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

The basic steps to creating plots with Seaborn are:

1. Prepare some data
2. Control figure aesthetics
3. Plot with Seaborn
4. Further customize your plot
5. Show your plot

```
>>> import matplotlib.pyplot as plt
>>> import seaborn as sns
>>> tips = sns.load_dataset("tips") #Step 1
>>> sns.set_style("whitegrid") #Step 2
>>> g = sns.lmplot(x="tip", #Step 3
                   y="total_bill",
                   data=tips,
                   aspect=2)
>>> g = (g.set_axis_labels("Tip","Total bill(USD)").
set(xlim=(0,10),ylim=(0,100)))
>>> plt.title("title") #Step 4
>>> plt.show(g) #Step 5
```

1 Data

Also see Lists, NumPy & Pandas

```
>>> import pandas as pd
>>> import numpy as np
>>> uniform_data = np.random.rand(10, 12)
>>> data = pd.DataFrame({'x':np.arange(1,101),
                         'y':np.random.normal(0,4,100)})
```

Seaborn also offers built-in data sets:

```
>>> titanic = sns.load_dataset("titanic")
>>> iris = sns.load_dataset("iris")
```

2 Figure Aesthetics

Also see Matplotlib

```
>>> f, ax = plt.subplots(figsize=(5,6)) #Create a figure and one subplot
```

Seaborn styles

```
>>> sns.set() #(Re)set the seaborn default
>>> sns.set_style("whitegrid") #Set the matplotlib parameters
>>> sns.set_style("ticks", #Set the matplotlib parameters
                  {"xtick.major.size":8,
                   "ytick.major.size":8})
#Return a dict of params or use with with to temporarily set the style
>>> sns.axes_style("whitegrid")
```

3 Plotting With Seaborn

Axis Grids

```
>>> g = sns.FacetGrid(titanic, #Subplot grid for plotting conditional relationships
                       col="survived",
                       row="sex")
>>> g = g.map(plt.hist, "age")
>>> sns.factorplot(x="pclass", #Draw a categorical plot onto a Facetgrid
                   y="survived",
                   hue="sex",
                   data=titanic)
>>> sns.lmplot(x="sepal_width", #Plot data and regression model fits across a FacetGrid
               y="sepal_length",
               hue="species",
               data=iris)
>>> h = sns.PairGrid(iris) #Subplot grid for plotting pairwise relationships
>>> h = h.map(plt.scatter)
>>> sns.pairplot(iris) #Plot pairwise bivariate distributions
>>> i = sns.JointGrid(x="x", #Grid for bivariate plot with marginal univariate plots
                      y="y",
                      data=data)
>>> i = i.plot(sns.regplot,
               sns.distplot)
>>> sns.jointplot("sepal_length", #Plot bivariate distribution
                  "sepal_width",
                  data=iris,
                  kind='kde')
```

4 Further Customizations

Also see Matplotlib

Axisgrid Objects

```
>>> g.despine(left=True) #Remove left spine
>>> g.set_ylabels("Survived") #Set the labels of the y-axis
>>> g.set_xticklabels(rotation=45) #Set the tick labels for x
>>> g.set_axis_labels("Survived", #Set the axis labels
                      "Sex")
>>> h.set(xlim=(0,5), #Set the limit and ticks of the x-and y-axis
          ylim=(0,5),
          xticks=[0,2.5,5],
          yticks=[0,2.5,5])
```

Plot

```
>>> plt.title("A Title") #Add plot title
>>> plt.ylabel("Survived") #Adjust the label of the y-axis
>>> plt.xlabel("Sex") #Adjust the label of the x-axis
>>> plt.ylim(0,100) #Adjust the limits of the y-axis
>>> plt.xlim(0,10) #Adjust the limits of the x-axis
>>> plt.setp(ax,yticks=[0,5]) #Adjust a plot property
>>> plt.tight_layout() #Adjust subplot params
```

Context Functions

```
>>> sns.set_context("talk") #Set context to "talk"
>>> sns.set_context("notebook", #Set context to "notebook",
                    font_scale=1.5, #Scale font elements and
                    rc={"lines.linewidth":2.5}) #override param mapping
```

Color Palette

```
>>> sns.set_palette("husl",3) #Define the color palette
>>> sns.color_palette("husl") #Use with with to temporarily set palette
>>> flatui = ["#9b59b6","#3498db","#95a5a6","#e74c3c","#34495e","#2ecc71"]
>>> sns.set_palette(flatui) #Set your own color palette
```

Regression Plots

```
>>> sns.regplot(x="sepal_width", #Plot data and a linear regression model fit
                y="sepal_length",
                data=iris,
                ax=ax)
```

Distribution Plots

```
>>> plot = sns.distplot(data.y, #Plot univariate distribution
                        kde=False,
                        color="b")
```

Matrix Plots

```
>>> sns.heatmap(uniform_data,vmin=0,vmax=1) #Heatmap
```

Categorical Plots

Scatterplot

```
>>> sns.stripplot(x="species", #Scatterplot with one categorical variable
                  y="petal_length",
                  data=iris)
>>> sns.swarmplot(x="species", #Categorical scatterplot with non-overlapping points
                  y="petal_length",
                  data=iris)
```

Bar Chart

```
>>> sns.barplot(x="sex", #Show point estimates & confidence intervals with scatterplot glyphs
                y="survived",
                hue="class",
                data=titanic)
```

Count Plot

```
>>> sns.countplot(x="deck", #Show count of observations
                  data=titanic,
                  palette="Greens_d")
```

Point Plot

```
>>> sns.pointplot(x="class", #Show point estimates & confidence intervals as rectangular bars
                  y="survived",
                  hue="sex",
                  data=titanic,
                  palette={"male":"g",
                  "female":"m"},
                  markers=["^","o"],
                  linestyles=["-","--"])
```

Boxplot

```
>>> sns.boxplot(x="alive", #Boxplot
                y="age",
                hue="adult_male",
                data=titanic)
>>> sns.boxplot(data=iris,orient="h") #Boxplot with wide-form data
```

Violinplot

```
>>> sns.violinplot(x="age", #Violin plot
                   y="sex",
                   hue="survived",
                   data=titanic)
```

5 Show or Save Plot

Also see Matplotlib

```
>>> plt.show() #Show the plot
>>> plt.savefig("foo.png") #Save the plot as a figure
>>> plt.savefig("foo.png", #Save transparent figure
                transparent=True)
```

> Close & Clear

Also see Matplotlib

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

Learn Data Skills Online at www.DataCamp.com