# **Data Visualization**

## **Agenda**

We want to **plot** points & lines on a graph with 2 axis.

Lecture outline:

* **Matplotlib** (1h10)
* Seaborn (20m)

## **Matplotlib**

****

Quoting the documentation:

Python 2D plotting library which produces publication quality figures of hardcopy formats and **interactive environments**.

Matplotlib can be used in **Python scripts**, the IPython shells, the **Jupyter notebook**, web application servers, etc.

👉 [Gallery](https://matplotlib.org/gallery/index.html) & [Repo](https://github.com/matplotlib/matplotlib)

### **Canonical import**

**import** **matplotlib.pyplot** **as** **plt**

### **Dataset**

In the upcoming slides, we will use data from the [US Carbon emissions from electricity production](https://www.kaggle.com/txtrouble/carbon-emissions)dataset.

### **Python script**

Matplotlib can be used in a regular .py python script:

mkdir dataviz-101 && cd $\_

touch carbon.py

In our favorite text editor we can code:

*# carbon.py*

**import** **matplotlib.pyplot** **as** **plt**

years\_x = [1975, 1980, 1985, 1990, 1995, 2000, 2005, 2010, 2015]

total\_y = [1243, 1543, 1619, 1831, 1960, 2310, 2415, 2270, 1918]

plt.plot(years\_x, total\_y)

plt.show() *# Will open a graph window.*

*# Wait for it to be closed to continue script execution*

Back to the terminal we can run:

python carbon.py

💾 Saving to disk with [matplotlib.pyplot.savefig](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.savefig.html)

*# [...]*

plt.savefig('carbon.png')

plt.show()

### **Notebook**

Matplotlib can also be used in a Jupyter Notebook.  
Start a jupyter notebook and create a Carbon notebook.  
Start with the following cell (after a good title):

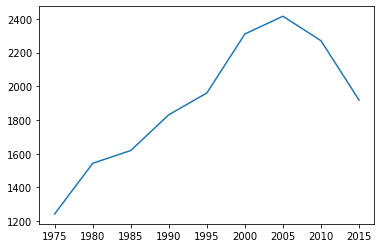
**from** **matplotlib** **import** pyplot **as** plt

years\_x = [1975, 1980, 1985, 1990, 1995, 2000, 2005, 2010, 2015]

total\_y = [1243, 1543, 1619, 1831, 1960, 2310, 2415, 2270, 1918]

plt.plot(years\_x, total\_y)

plt.show()



### **Notebook Tips**

* You can use ipywidgets for interactive graphs

!pip install ipywidgets

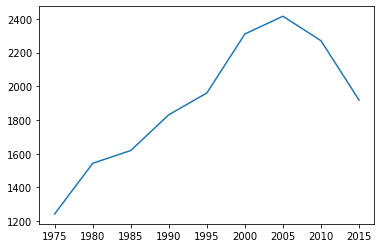
%matplotlib widget *# enable interactivity in your notebook*

%matplotlib inline *# get back to normal mode*

❗️ Warning: with widgets, you might encounter compatibility issues. We recommend you to use it only when needed.

* You don't need to type plt.show() in a notebook context.
  + Make sure to use ; at the final line to avoid printing its output

plt.plot(years\_x, total\_y);



## **Matplotlib Basics**

Methods to call *before* plt.show() to **enrich** the plot.

Let's stick in carbon.py for now and run the code with

python carbon.py

### **Title**

👉 [matplotlib.pyplot.title](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.title.html)

plt.title("USA - CO2 emissions from electricity production")

### **Axis labels**

👉 [matplotlib.pyplot.xlabel](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.xlabel.html) and [matplotlib.pyplot.ylabel](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.ylabel.html)

plt.xlabel("Year")

plt.ylabel("CO2 - M of tons")

### **Axis ticks**

👉 [matplotlib.pyplot.xticks](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.xticks.html) & [matplotlib.pyplot.yticks](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.yticks.html)

For example:

plt.xticks([1975, 1995, 2015], ['start', 1995, 'end'])

plt.yticks([0, 5000])

💡2 more useful methods for axis: [matplotlib.pyplot.xlim](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.xlim.html) & [matplotlib.pyplot.ylim](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.ylim.html)

Let's add two more lines for **Coal** and **Natural gas** production:

coal\_y = [823, 1136, 1367, 1547, 1660, 1927, 1983, 1827, 1352]

gas\_y = [171, 200, 166, 175, 228, 280, 319, 399, 529]

And plot them:

plt.plot(years\_x, coal\_y)

plt.plot(years\_x, gas\_y)

### **Legend**

👉 [matplotlib.pyplot.legend](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.legend.html)

First you need to add **labels** to each plot:

plt.plot(years\_x, total\_y, label="Total")

plt.plot(years\_x, coal\_y, label="Coal")

plt.plot(years\_x, gas\_y, label="Natural Gas")

Then call:

plt.legend(loc="best")

### **Grid**

👉 [matplotlib.pyplot.grid](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.grid.html)

Example:

plt.grid(axis="y", linewidth=0.5)

### **Styles**

Matplotlib comes with [many **style sheets**](https://matplotlib.org/gallery/style_sheets/style_sheets_reference.html) to customize the look & feel of your graph.

print(plt.style.available)

['Solarize\_Light2', '\_classic\_test\_patch', 'bmh', 'classic', 'dark\_background', 'fast', 'fivethirtyeight', 'ggplot', 'grayscale', 'seaborn', 'seaborn-bright', 'seaborn-colorblind', 'seaborn-dark', 'seaborn-dark-palette', 'seaborn-darkgrid', 'seaborn-deep', 'seaborn-muted', 'seaborn-notebook', 'seaborn-paper', 'seaborn-pastel', 'seaborn-poster', 'seaborn-talk', 'seaborn-ticks', 'seaborn-white', 'seaborn-whitegrid', 'tableau-colorblind10']

👉 [lib/matplotlib/mpl-data/stylelib](https://github.com/matplotlib/matplotlib/tree/master/lib/matplotlib/mpl-data/stylelib) folder on GitHub

### **Using a style**

In a Python script:

*# After `matplotlib` import:*

plt.style.use('seaborn')

In a Notebook, isolate the configuration to *one* plot to not pollute the whole context:

**with** plt.style.context('seaborn'):

*# [...]*

plt.show()

### **Lines**

You have total control over the [matplotlib.lines.Line2D](https://matplotlib.org/api/_as_gen/matplotlib.lines.Line2D.html):

* [color](https://matplotlib.org/api/colors_api.html) (cycled, see plt.rcParams['axes.prop\_cycle'])
* [marker](https://matplotlib.org/gallery/lines_bars_and_markers/marker_reference.html) (default: None)
* [linestyle](https://matplotlib.org/gallery/lines_bars_and_markers/line_styles_reference.html) (default: "-")
* linewidth (default: 1.5)

For example:

plt.plot(years\_x, total\_y, color="#999999", linestyle=':', marker='s')

plt.plot(years\_x, coal\_y, linewidth=3)

### **Figure Size**

plt.figure(figsize=(10,5))

plt.figure(figsize=(10,5))

*# 3 lines plot*

plt.plot(years\_x, total\_y, label='total', c="grey", ls=':', marker='s')

plt.plot(years\_x, coal\_y, label='coal')

plt.plot(years\_x, gas\_y, label='gas')

*# Decoration*

plt.legend()

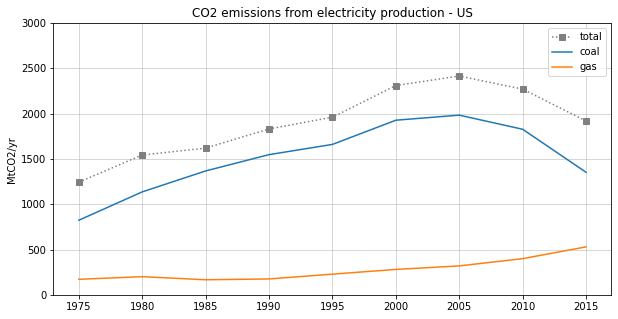
plt.title('CO2 emissions from electricity production - US')

plt.ylim((0,3000))

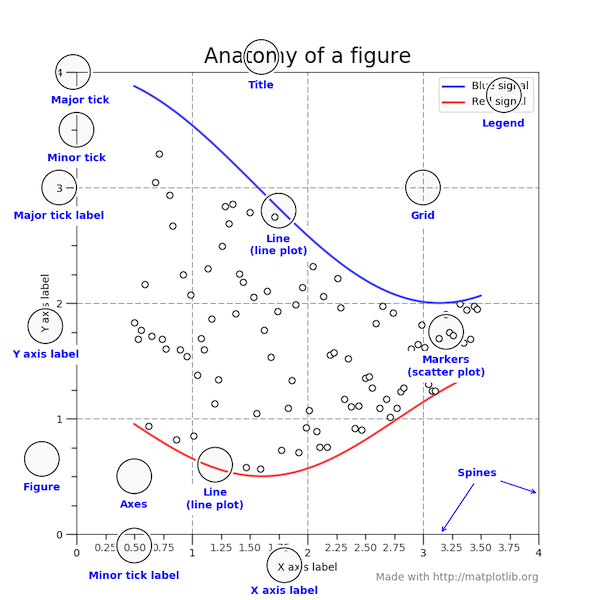
plt.ylabel('MtCO2/yr')

plt.grid(lw=0.5)

plt.show()



## **Matplotlib Advanced**

****

### **Axes vs Axis**

* Axis is the axis of the plot, the thing that gets ticks and tick labels.
* The axes is the area your plot appears in.

👉 [matplotlib.axes](https://matplotlib.org/api/axes_api.html)

You can access the current Axes instance with [matplotlib.pyplot.gca](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.gca.html)

ax = plt.gca()

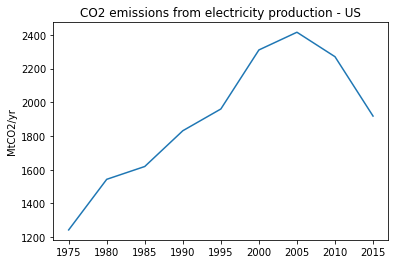
*# Let's take this simple example*

plt.plot(years\_x, total\_y)

plt.ylabel('MtCO2/yr')

plt.title('CO2 emissions from electricity production - US')

plt.show()



*# And compare it with this one*

plt.plot(years\_x, total\_y)

*# Access the ax first*

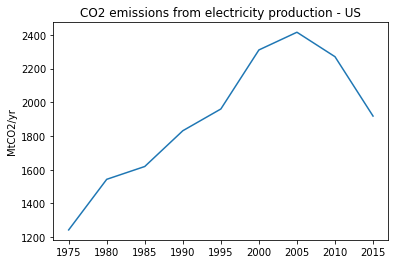
ax = plt.gca()

*# then change its properties*

ax.set\_title('CO2 emissions from electricity production - US')

ax.set\_ylabel('MtCO2/yr')

plt.show()



⚠️ Notice the difference:

* [matplotlib.pyplot.title](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.title.html)
* [matplotlib.axes.Axes.set\_title](https://matplotlib.org/api/_as_gen/matplotlib.axes.Axes.set_title.html)

🤔 Why would we want to access the ax?

* For finetuning / customizing
* When creating multiple subplots
* For integration with other libraries (Pandas etc...)

### **Finetune axes' Spines**

ax.spines

OrderedDict([('left', <matplotlib.spines.Spine at 0x116e400d0>),

('right', <matplotlib.spines.Spine at 0x116e401c0>),

('bottom', <matplotlib.spines.Spine at 0x116e402b0>),

('top', <matplotlib.spines.Spine at 0x116e403a0>)])

You can remove a spine (or set a specific color!) with [Spine.set\_color](https://matplotlib.org/api/spines_api.html#matplotlib.spines.Spine.set_color):

ax.spines['right'].set\_color(**None**)

You can also [Spine.set\_position](https://matplotlib.org/api/spines_api.html#matplotlib.spines.Spine.set_position):

ax.spines['bottom'].set\_position(('axes', 0.5)) *# half of y-axis*

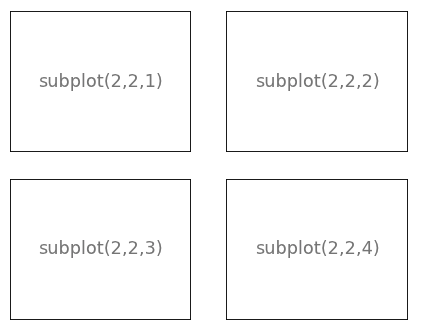
*# or*

ax.spines['bottom'].set\_position(('data', 750)) *# 750 on y-axis*

💡 Useful in a [Math context](https://scipy-lectures.org/intro/matplotlib/auto_examples/exercises/plot_exercise_7.html)

## **Figures, Subplots and Axes**

A **figure** in matplotlib means the whole window in the UI.  
Within this figure there can be several **subplots**Subplots are arranged and numberd in a (nrow, ncol) grid as below



**Subplots** are instances of the **Axes** class

You can also create an axes without a subplot grid, by placing it in absolute position within a figure. 👉 [stack overflow on axes vs. subplot](https://stackoverflow.com/a/43330553/7849552)

### **Multiple subplots on the same figure**

#### **State-based interface**

*# Start a figure*

plt.figure(figsize=(10,3))

*# First subplot*

plt.subplot(1,2,1)

plt.plot(years\_x, coal\_y, label="coal")

plt.plot(years\_x, gas\_y, label = "gas")

plt.title('coal vs. gas')

plt.legend()

*# Second subplot*

plt.subplot(1,2,2)

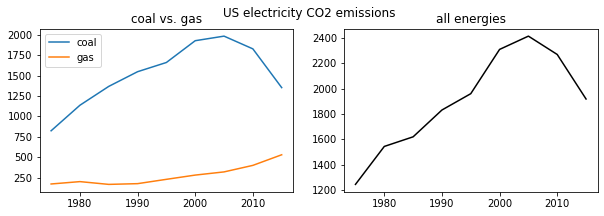
plt.plot(years\_x, total\_y, label="total", c='black')

plt.title("all energies")

*# Global figure methods*

plt.suptitle('US electricity CO2 emissions')

plt.show()



#### **Object-oriented interface**

fig = plt.figure(figsize=(10,3))

*# First subplot*

ax1 = fig.add\_subplot(1,2,1)

ax1.plot(years\_x, coal\_y, label="coal")

ax1.plot(years\_x, gas\_y, label = "gas")

ax1.set\_title('coal vs. gas')

ax1.legend()

*# Second subplot*

ax2 = fig.add\_subplot(1,2,2)

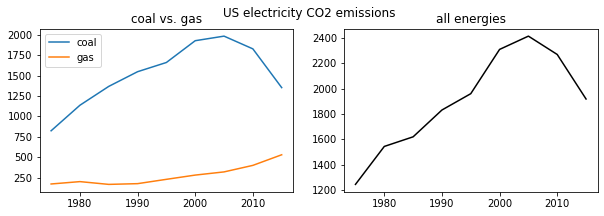
ax2.plot(years\_x, total\_y, c='black')

ax2.set\_title('all energies')

*# Global figure methods*

fig.suptitle('US electricity CO2 emissions')

plt.show()



Instead of

fig = plt.figure()

ax1 = fig.add\_subplot(1,2,1)

ax2 = fig.add\_subplot(1,2,2)

You will often find in the offical docs the shortcut

fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2)

It's called a **Destructuring assignment**

*# Destructuring initialization*

fig, axs = plt.subplots(1, 2, figsize=(10,3)) *# axs is a (1,2) nd-array*

*# First subplot*

axs[0].plot(years\_x, coal\_y, label="coal")

axs[0].plot(years\_x, gas\_y, label = "gas")

axs[0].set\_title('coal vs. gas')

axs[0].legend()

*# Second subplot*

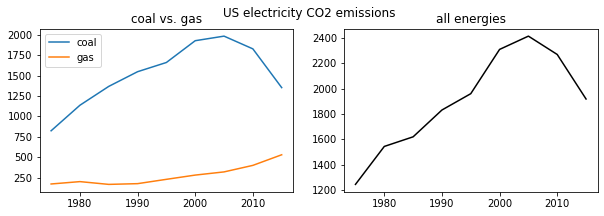
axs[1].plot(years\_x, total\_y, c='black')

axs[1].set\_title('all energies')

*# Global figure methods*

plt.suptitle('US electricity CO2 emissions')

plt.show()



## **What about Pandas?**

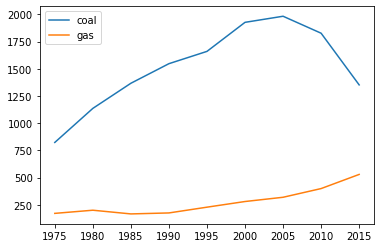
**import** **pandas** **as** **pd**

df = pd.DataFrame({ 'coal': coal\_y, 'gas': gas\_y }, index=years\_x)

df

|  | **coal** | **gas** |
| --- | --- | --- |
| **1975** | 823 | 171 |
| **1980** | 1136 | 200 |
| **1985** | 1367 | 166 |
| **1990** | 1547 | 175 |
| **1995** | 1660 | 228 |
| **2000** | 1927 | 280 |
| **2005** | 1983 | 319 |
| **2010** | 1827 | 399 |
| **2015** | 1352 | 529 |

df.plot();



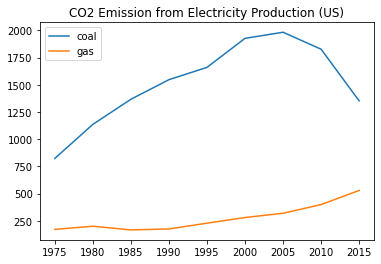
**import** **pandas** **as** **pd**

ax = df.plot()

ax.set\_title('CO2 Emission from Electricity Production (US)')

ax

<AxesSubplot:title={'center':'CO2 Emission from Electricity Production (US)'}>



type(ax)

matplotlib.axes.\_subplots.AxesSubplot

type(ax).\_\_bases\_\_

(matplotlib.axes.\_subplots.SubplotBase, matplotlib.axes.\_axes.Axes)

**⚠️ beware of the two plot methods**:

pandas.DataFrame.plot

df.plot() *# plot all columns against the index*

ax = df.plot() *# this is an Axes, thanks to pandas*

matplotlib.pyplot.plot

plt.plot(df) *# not an Axes (matplotlib.lines.Line2D in our case)*

ax = plt.gca() *# get\_current\_axes method required to access it*

#### **2 plots with pandas**

df1 = pd.DataFrame({ 'coal': coal\_y, 'gas': gas\_y }, index=years\_x)

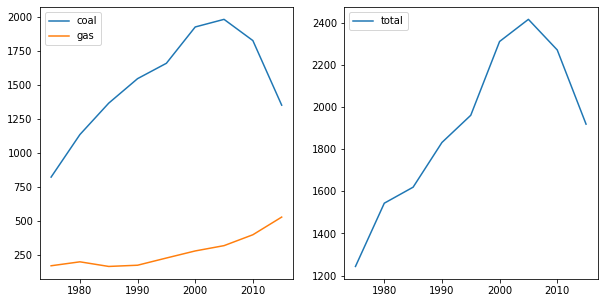
df2 = pd.DataFrame({ 'total': total\_y }, index=years\_x)

fig, (ax1, ax2) = plt.subplots(1,2, figsize=(10,5))

df1.plot(ax=ax1)

df2.plot(ax=ax2)

<AxesSubplot:>



## **Plot types**

Let's explore other plot types than line charts that we can quickly draw with matplotlib

### **Scatter Plot**

Type of plot using Cartesian coordinates (x, y) to display values one (or multiple) set(s) of data.

👉 [matplotlib.pyplot.scatter](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.scatter.html)

Let's plot the relationship between Views & Likes of [Trending Youtube Videos](https://www.kaggle.com/datasnaek/youtube-new) of 2019

[📚 raw csv](https://gist.githubusercontent.com/ssaunier/8044d6a7267223787ed143d0973e3ec6/raw/youtube.csv)

data = pd.read\_csv('https://gist.githubusercontent.com/ssaunier/8044d6a7267223787ed143d0973e3ec6/raw/youtube.csv')

plt.scatter(data['views'], data['likes'], edgecolor='#333333', alpha=0.75)

plt.xscale('log')

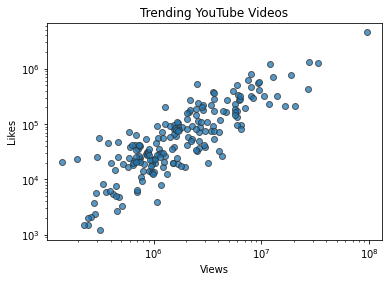
plt.yscale('log')

plt.title('Trending YouTube Videos')

plt.xlabel('Views')

plt.ylabel('Likes')

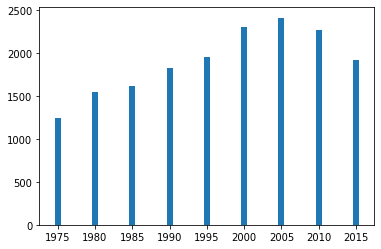
Text(0, 0.5, 'Likes')



### **Bar Plot**

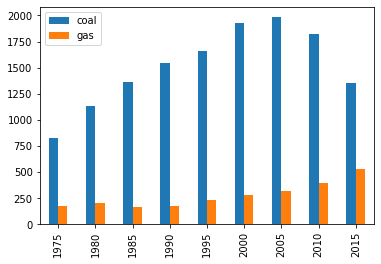
👉 [matplotlib.pyplot.bar](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.bar.html)

plt.bar(years\_x, total\_y);



*# with pandas*

df.plot(kind='bar');



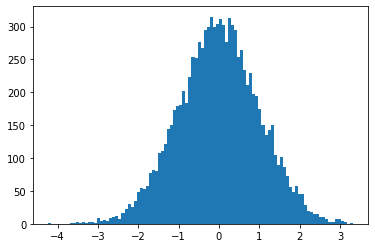
### **Histogram**

A histogram is an accurate representation of the **distribution** of numerical data. We can use the [matplotlib.pyplot.hist](https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.hist.html) function

**import** **numpy** **as** **np**

x = np.random.normal(size=10\_000) *# Randomly pick 10\_000 numbers*

plt.hist(x, bins=100); *# Vertical axis shows the frequencies of each bin.*



## **Seaborn**

****

A Python data visualization library **built on top of matplotlib**. It provides a **high-level interface**for drawing attractive and informative statistical graphics.

Official page: [seaborn.pydata.org](https://seaborn.pydata.org/)

👉 [Gallery](https://seaborn.pydata.org/examples/index.html)

### **Install**

!pip install --quiet seaborn

### **Canonical import**

**import** **matplotlib.pyplot** **as** **plt**

**import** **seaborn** **as** **sns**

### **Loading the tips dataset**

You can preview it on GitHub here: [mwaskom/seaborn-data](https://github.com/mwaskom/seaborn-data)

There are two ways to load the **same** DataFrame:

tips\_df = sns.load\_dataset('tips')

*# tips\_df = pd.read\_csv("https://raw.githubusercontent.com/mwaskom/seaborn-data/master/tips.csv")*

tips\_df.shape

(244, 7)

tips\_df.head(5)

|  | **total\_bill** | **tip** | **sex** | **smoker** | **day** | **time** | **size** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 16.99 | 1.01 | Female | No | Sun | Dinner | 2 |
| **1** | 10.34 | 1.66 | Male | No | Sun | Dinner | 3 |
| **2** | 21.01 | 3.50 | Male | No | Sun | Dinner | 3 |
| **3** | 23.68 | 3.31 | Male | No | Sun | Dinner | 2 |
| **4** | 24.59 | 3.61 | Female | No | Sun | Dinner | 4 |

### **One numeric variable (univariate)**

We can use one of:

* Histogram
* Density plot

In Seaborn, we use [**histplot**](http://seaborn.pydata.org/generated/seaborn.histplot.html)

🤔 Can you plot the distribution of the numeric variable total\_bill?

plt.figure(figsize=(14, 4))

plt.subplot(1, 2, 1)

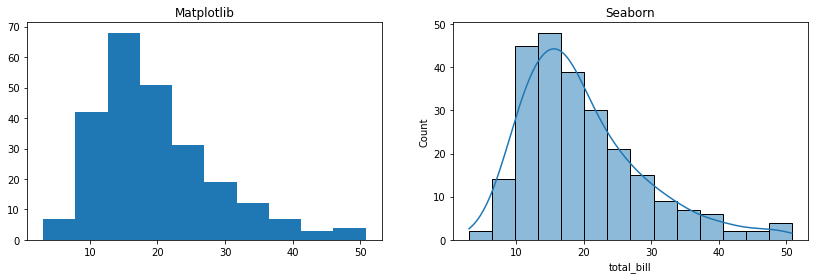
plt.title('Matplotlib')

plt.hist(tips\_df['total\_bill'])

plt.subplot(1, 2, 2)

plt.title('Seaborn')

sns.histplot(tips\_df['total\_bill'], kde=**True**);



### **One categorical variable**

We can use one of:

* Barplot
* Pie/Doughnut charts (👎)

In Seaborn, we use [**countplot**](https://seaborn.pydata.org/generated/seaborn.countplot.html)

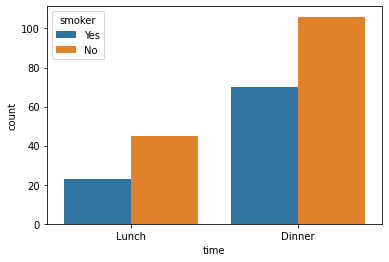
🤔 Are there more smokers than non-smokers in this restaurant clientele?

sns.countplot(x="smoker", data=tips\_df);



🤔 Are there more smokers at Lunch?

sns.countplot(x="time", hue="smoker", data=tips\_df);



### **One Numeric + One Categorical**

We can use one of:

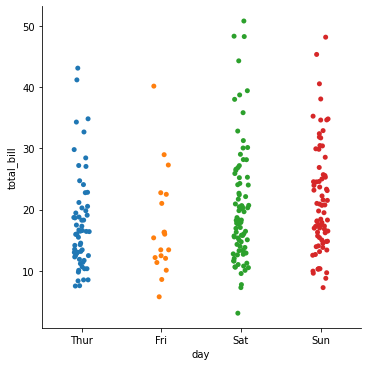
* Scatter plots
* Distribution plots (box, violin, etc.)

In Seaborn, we will use the unified API of [seaborn.catplot](https://seaborn.pydata.org/generated/seaborn.catplot.html)

### **Categorical Scatter plot**

sns.catplot(x='day', y='total\_bill', data=tips\_df)

<seaborn.axisgrid.FacetGrid at 0x116f8eaf0>

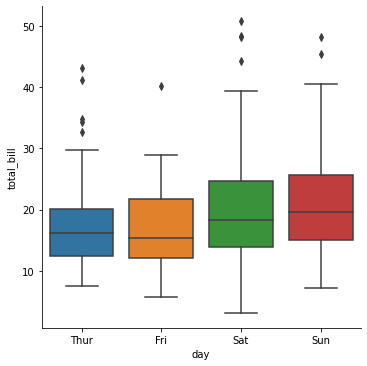


### **Categorical distribution plots**

🤔 **What day** do people spend the most money in average at this restaurant?

Try using a [seaborn.catplot](https://seaborn.pydata.org/generated/seaborn.catplot.html) with kind: bar, box, violin or boxen

sns.catplot(x='day', y='total\_bill', data=tips\_df, kind="box");



### **Two numeric variables (bivariate)**

One can be x, the other can be y allowing to visualize **statistical relationship**.

#### **Scatterplot**

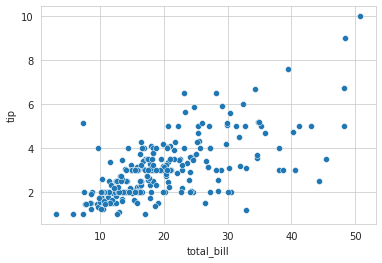
🤔 What is the relationship between the tip and the total\_bill?

Let's use [seaborn.scatterplot](https://seaborn.pydata.org/generated/seaborn.scatterplot.html)

**with** sns.axes\_style('whitegrid'):

*# sns.set(style="whitegrid") for global change*

sns.scatterplot(x="total\_bill", y="tip", data=tips\_df);

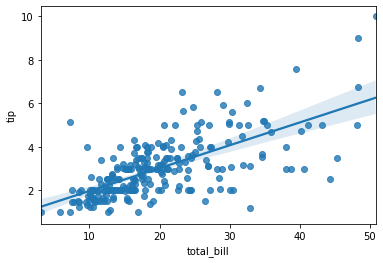


#### **Regression**

You can use [seaborn.regplot](https://seaborn.pydata.org/generated/seaborn.regplot.html)

sns.regplot(x='total\_bill', y='tip', data=tips\_df)

<AxesSubplot:xlabel='total\_bill', ylabel='tip'>



👉 Read more about [visualizing linear relationships](https://seaborn.pydata.org/tutorial/regression.html) in Seaborn's documentation

#### **💪 (3 numerical + 1 categorical) variables in 1 graph ??**

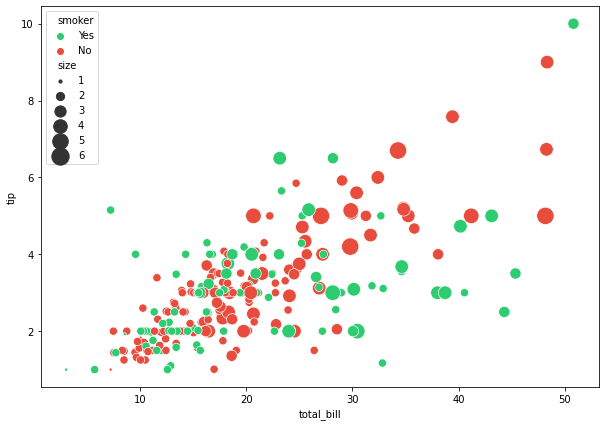
plt.figure(figsize=(10, 7))

sns.scatterplot(x="total\_bill", y="tip", hue='smoker', size="size",

palette=sns.color\_palette(["#2ecc71", "#e74c3c"]),

sizes=(10, 300), data=tips\_df)

<AxesSubplot:xlabel='total\_bill', ylabel='tip'>



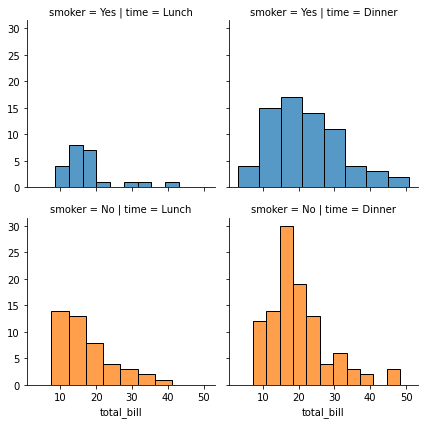
### **Facet grid - lets you plot the graph of your choice by groups**

*# Create a grid*

g = sns.FacetGrid(tips\_df, col="time", row="smoker", hue="smoker")

*# Plot a graph in each grid element*

g.map(sns.histplot, "total\_bill");

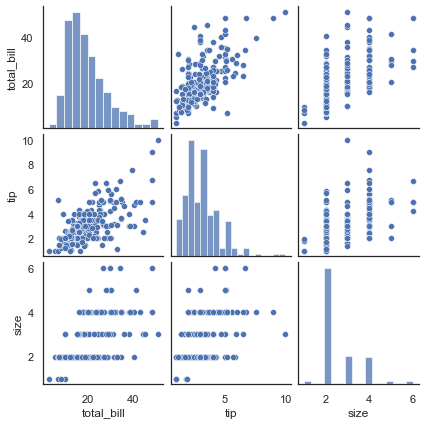


### [**Pair plots**](https://seaborn.pydata.org/generated/seaborn.pairplot.html) **to automatically identify all *correlations* in a DataFrame**

sns.set(style='white')

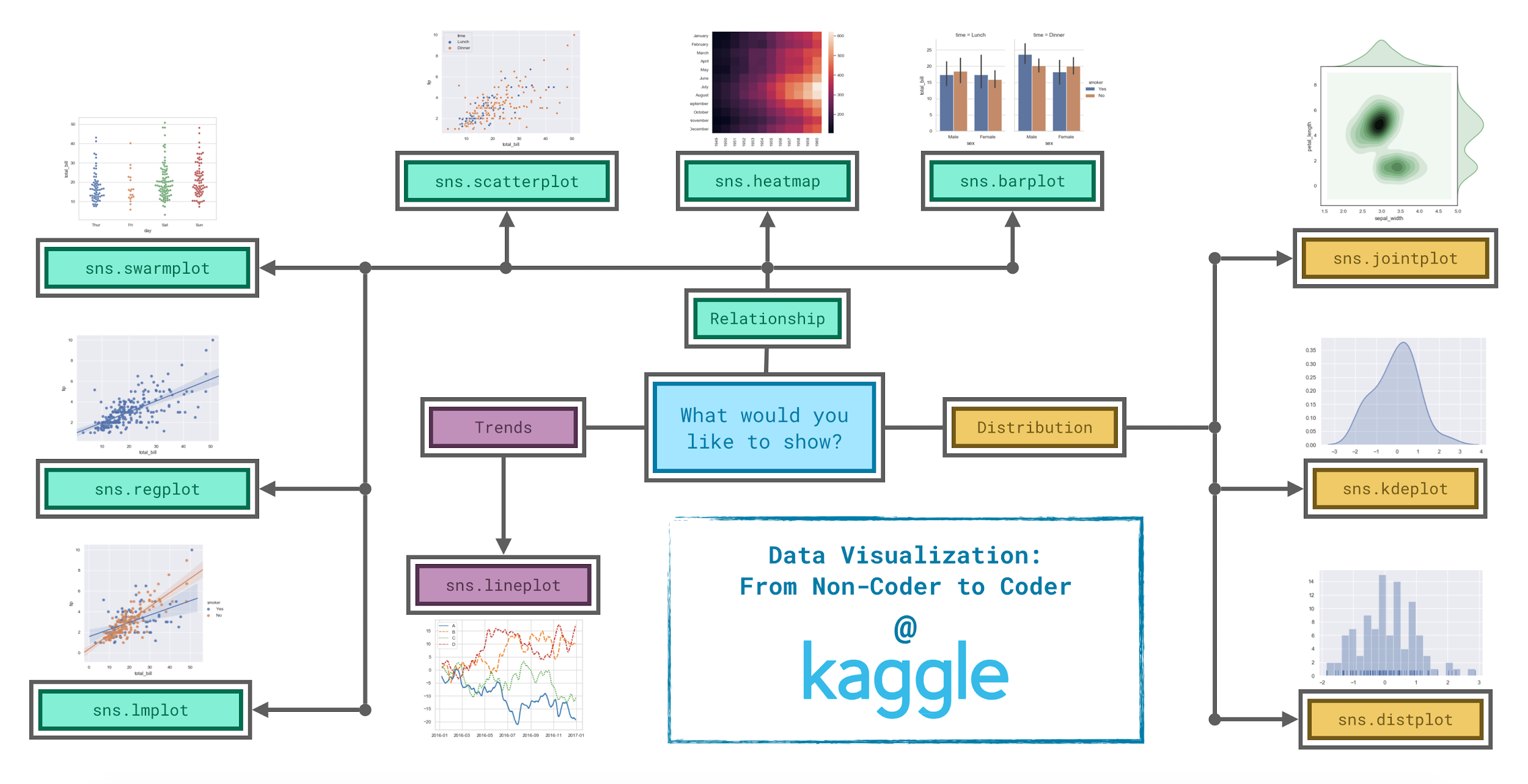
sns.pairplot(tips\_df, height=2)

plt.show()



## **Seaborn - Cheat sheets**

📊 [cheatsheet](https://s3.amazonaws.com/assets.datacamp.com/blog_assets/Python_Seaborn_Cheat_Sheet.pdf)



### **Le Wagon - Cheat Sheet**

*# LINE PLOTS*

plt.plot(x=df.col1, y=df.col2, c='red', ls='--', lw='0.5')

sns.lineplot(data=df, x='col1', y='col2', hue='col3', size='col4')

*# DISTRIBUTIONS*

plt.hist()

sns.histplot()

sns.kdeplot()

sns.jointplot()

*# SCATTER PLOTS*

plt.scatter()

sns.scatterplot()

sns.regplot()

*# COUNT PLOTS*

sns.countplot()

*# CAT PLOTS*

plt.bar() *# eq. plt.plot(kind=‘bar’)*

sns.barplot() *# eq. catplot(kind=“bar”)*

sns.violinplot() *# eq. catplot(kind=“violin”)*

sns.boxplot() *# eq. catplot(kind=“box”)*

*# FACET GRID*

g = sns.FacetGrid(data=df, col='col1')

g.map(plt.hist, 'col2')

*# DATAFRAME-LEVEL MULTI CORRELATIONS*

sns.heatmap(df.corr())

sns.pairplot(hue='')

*## 2D HISTOGRAMS*

plt.hist2d()

plt.colorbar()

sns.jointplot(x,y, kind='kde', data=df)

*## 2D PROJECTION*

plt.contour(X,Y,Z) *# iso lines*

plt.contourf(X,Y,Z=f(X,Y)) *# area colors*

## **Bonus: Plotly**

Building **interactive** graphs (JavaScript!)

👉 [Plot.ly Gallery](https://plot.ly/python/)

!pip install --quiet plotly

*# Canonical import*

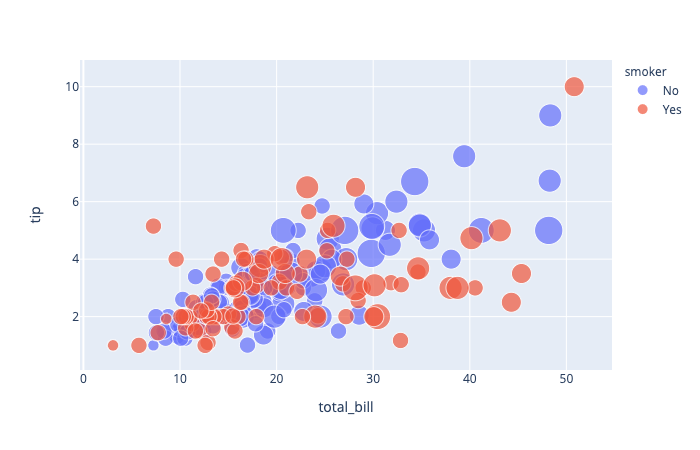
**import** **plotly.express** **as** **px**

### **Scatter Plot**

tips = px.data.tips()

fig = px.scatter(tips, x="total\_bill", y="tip", size="size", color="smoker")

fig.show()



## **Bibliography**

* 📊 [Matplotlib cheatsheet](https://matplotlib.org/cheatsheets/cheatsheets.pdf)
* 📊 [Seaborn cheatsheet](https://s3.amazonaws.com/assets.datacamp.com/blog_assets/Python_Seaborn_Cheat_Sheet.pdf)
* 📚 Claus O. Wilke, [Fundamentals of Data Visualization](https://clauswilke.com/dataviz/) (online book)
* 📚 Yan Holtz, [From data to Viz](https://www.data-to-viz.com/) (website)
  + [Decision Tree](https://www.data-to-viz.com/#explore)
  + [Caveats](https://www.data-to-viz.com/caveats.html)
* 📄 [How to tell a great story with Data Viz](https://www.kdnuggets.com/2021/02/telling-great-data-story-visualization-decision-tree.html)
* 📄 [The many ways to call axes in Matplotlib](https://medium.com/towards-data-science/the-many-ways-to-call-axes-in-matplotlib-2667a7b06e06)

## **Your turn!**