

Plotting and Visualization

Prof. Gheith Abandah

Reference

- **Chapter 9: Plotting and Visualization**
- Wes McKinney, **Python for Data Analysis**: Data Wrangling with Pandas, NumPy, and IPython, O'Reilly Media, 2nd Edition, 2018.
 - Material: <https://github.com/wesm/pypop-book>

Plotting and Visualization

- **Making informative visualizations** is one of the most important tasks in data analysis.
- It may be a part of the **exploratory process**: to help identify outliers or needed data transformations, or as a way of generating ideas for models.

Outline

9.1 A Brief **matplotlib** API Primer

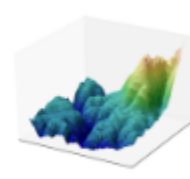
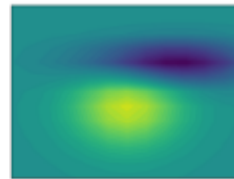
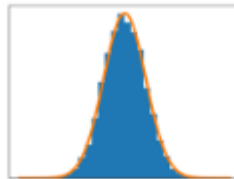
- Figures and Subplots
- Colors, Markers, and Line Styles
- Ticks and Labels
- Saving Plots to File
- matplotlib Configuration

9.2 Plotting with **pandas** and **seaborn**

- Line Plots
- Bar Plots
- Histograms and Density Plots
- Scatter or Point Plots
- Facet Grids and Categorical Data

Matplotlib: MATLAB-style Scientific Visualization

- Matplotlib is a Python **plotting library** which produces publication **quality figures** in a variety of hardcopy formats.



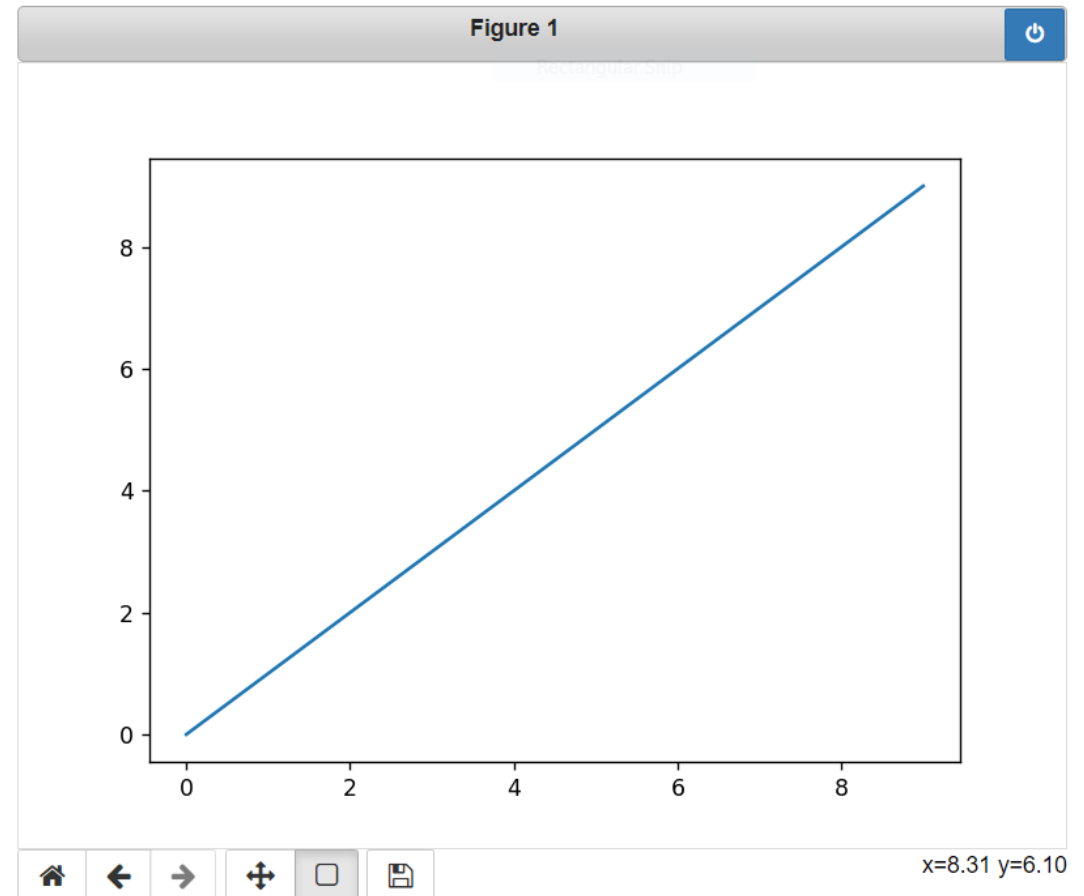
- **Website:** <https://matplotlib.org/>
- Also, check the **tutorial** package website:
<https://matplotlib.org/tutorials/introductory/pyplot.html>

9.1 A Brief matplotlib API Primer

- To set up Jupyter Notebook, run `%matplotlib notebook` (`%matplotlib` in IPython).

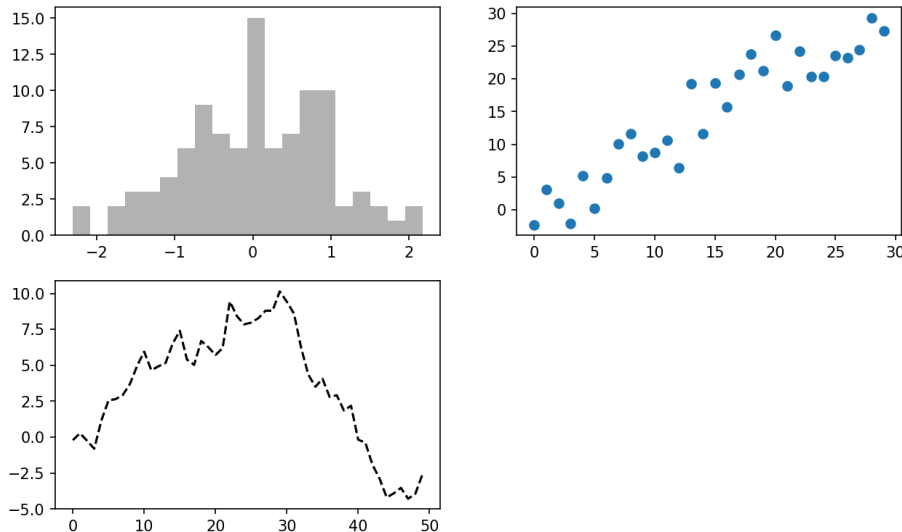
- Simple line plot:

```
import matplotlib.pyplot as plt
import numpy as np
data = np.arange(10)
plt.plot(data)
```



Figures and Subplots

- Plots in matplotlib reside within a **Figure** object.
- You must create one or more **subplots** inside a blank figure.

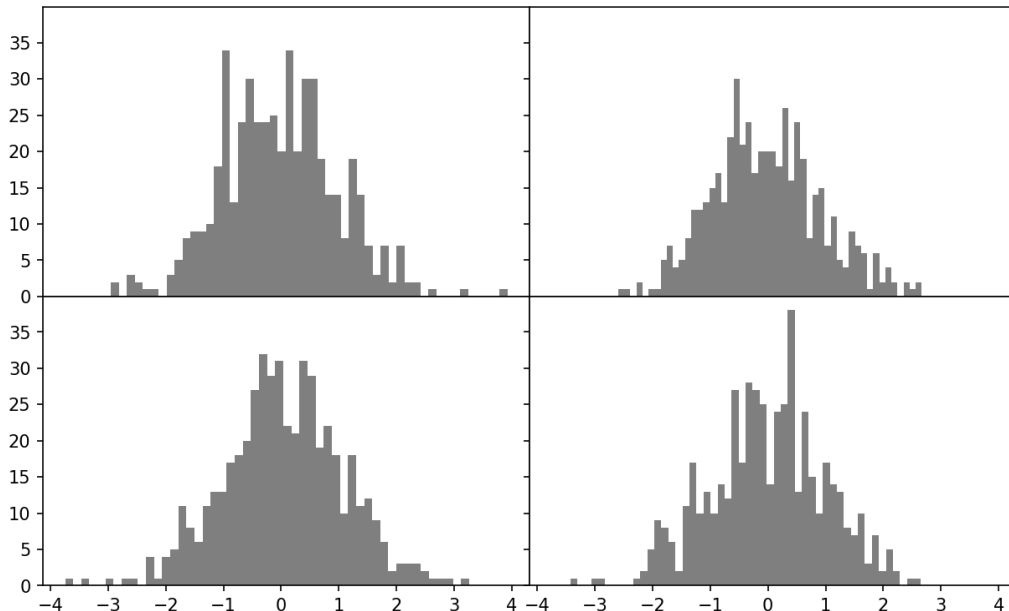


```
fig = plt.figure(figsize=(4, 3))
ax1 = fig.add_subplot(2, 2, 1)
ax2 = fig.add_subplot(2, 2, 2)
ax3 = fig.add_subplot(2, 2, 3)
plt.plot(np.random.randn(50).cumsum
         (), 'k--')

_ = ax1.hist(np.random.randn(100),
             bins=20, color='k', alpha=0.3)
ax2.scatter(np.arange(30),
            np.arange(30) + 3 *
            np.random.randn(30))
```

Figures and Subplots

- Better to create a new figure and return a NumPy **array** containing the created **subplot objects**.



```
fig, axes = plt.subplots(2, 2,  
                        sharex=True, sharey=True)  
for i in range(2):  
    for j in range(2):  
        axes[i, j].hist(  
            np.random.randn(500),  
            bins=50, color='k',  
            alpha=0.5)  
plt.subplots_adjust(wspace=0,  
                   hspace=0)
```

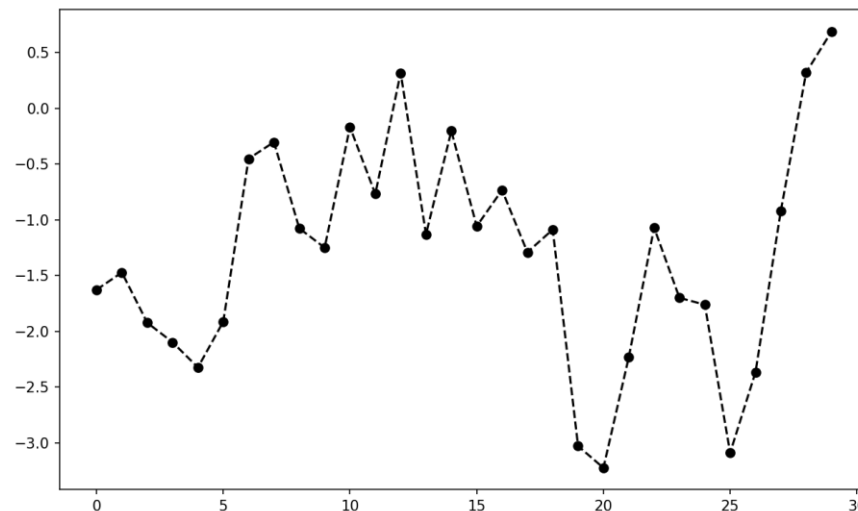

Colors, Markers, and Line Styles

- Specify color and style:

- String
- Explicitly

- Check options using `plt.plot?`

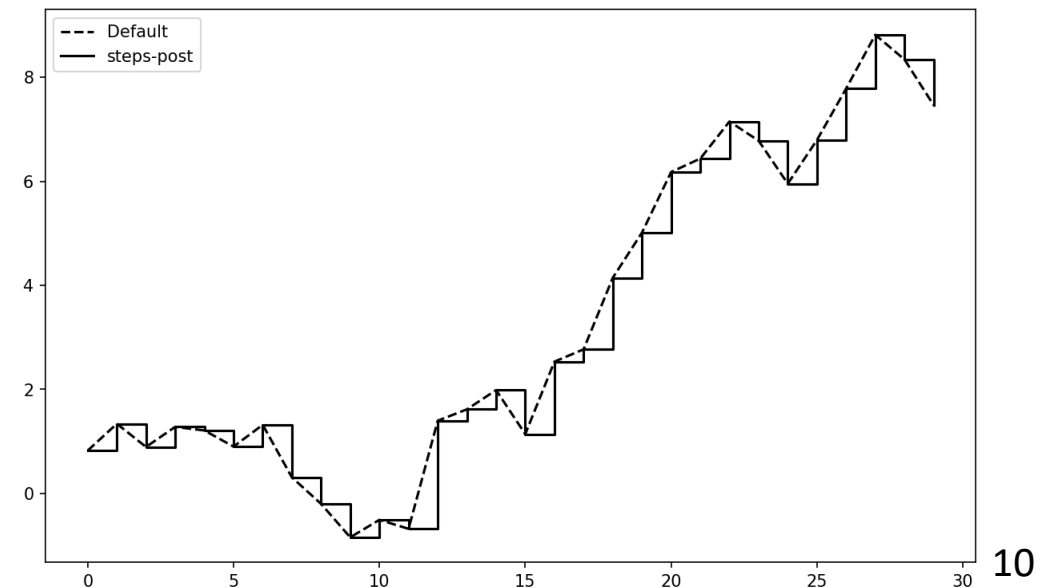
```
from numpy.random import randn
plt.plot(randn(30).cumsum(),
         'ko--')
plt.plot(randn(30).cumsum(),
         color='k', marker='o',
         linestyle='dashed')
```



Colors, Markers, and Line Styles

- You can specify the draw style:
 - Linearly interpolated
 - Step
 - etc.
- You can draw a **legend**.

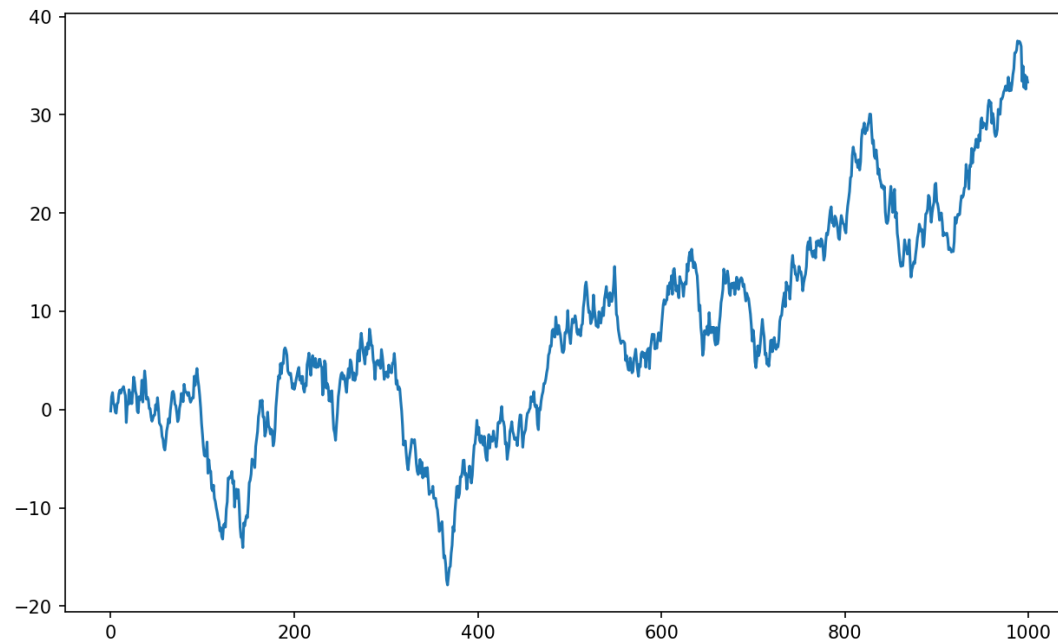
```
data = np.random.randn(30).cumsum()  
plt.plot(data, 'k--',  
         label='Default')  
plt.plot(data, 'k-',  
         drawstyle='steps-post',  
         label='steps-post')  
plt.legend(loc='best')
```



Ticks and Labels

- For plot decoration:
 - **Procedural** pyplot interface
 - **Object-oriented** interface

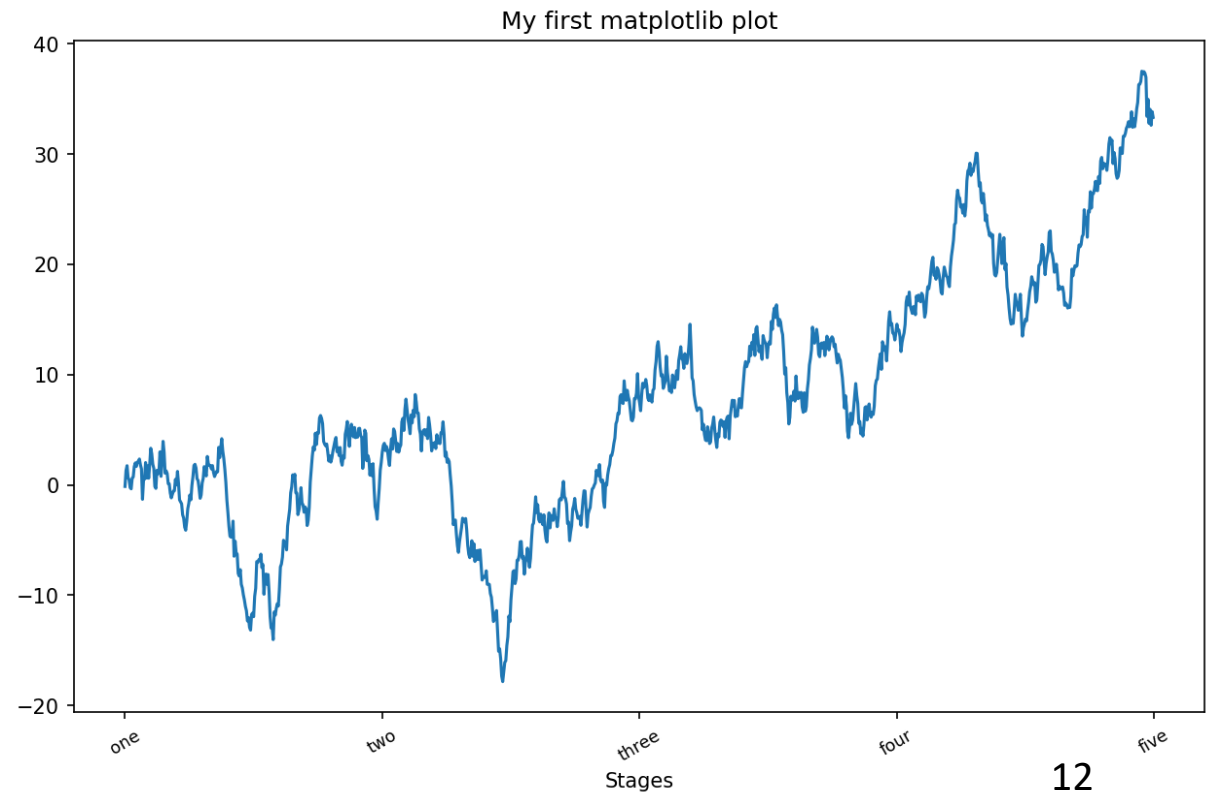
```
fig = plt.figure()  
ax = fig.add_subplot(1, 1, 1)  
ax.plot(np.random.randn(  
1000).cumsum())
```



Ticks and Labels

```
ticks = ax.set_xticks([0, 250, 500,
                      750, 1000])
labels = ax.set_xticklabels(['one',
                             'two', 'three', 'four',
                             'five'], rotation=30,
                             fontsize='small')
ax.set_title(
    'My first matplotlib plot')
ax.set_xlabel('Stages')
# Or:
props = {
    'title': 'My first plot',
    'xlabel': 'Stages'
}
ax.set(**props)
```

- Check [matplotlib.axes](https://matplotlib.org/3.1.1/api/axes_api.html)



Saving Plots to File

- When saving to a file, the **file extension** specifies the **image format**.

```
plt.savefig('figpath.png', dpi=400,  
            bbox_inches='tight')
```

Argument	Description
fname	String containing a filepath or a Python file-like object. The figure format is inferred from the file extension (e.g., .pdf for PDF or .png for PNG)
dpi	The figure resolution in dots per inch; defaults to 100 out of the box but can be configured
facecolor, edgecolor	The color of the figure background outside of the subplots; 'w' (white), by default
format	The explicit file format to use ('png', 'pdf', 'svg', 'ps', 'eps', ...)
bbox_inches	The portion of the figure to save; if 'tight' is passed, will attempt to trim the empty space around the figure

matplotlib Configuration

- matplotlib comes configured with color schemes and defaults geared for publication.
- Can be customized:
 - Programmatically using the **rc** method
 - Configuration:
matplotlib/mpl-data/matplotlibrc.
Customize in your home directory as **.matplotlibrc**

```
plt.rc('figure', figsize=(10, 10))
font_options = {
    'family' : 'monospace',
    'weight' : 'bold',
    'size' : 'small'}
plt.rc('font', **font_options)
```

Outline

9.1 A Brief **matplotlib** API Primer

- Figures and Subplots
- Colors, Markers, and Line Styles
- Ticks and Labels
- Saving Plots to File
- **matplotlib** Configuration

9.2 Plotting with **pandas** and **seaborn**

- Line Plots
- Bar Plots
- Histograms and Density Plots
- Scatter or Point Plots
- Facet Grids and Categorical Data

9.2 Plotting with pandas and seaborn

- YouTube Video from **Kimberly Fessel**

Introduction to Seaborn | How seaborn Python works with matplotlib along with seaborn and pandas

<https://youtu.be/vaf4ir8eT38>

9.2 Plotting with pandas and seaborn

- **matplotlib is low-level tool**; you assemble a plot from its base components.
- Productive plotting is available through:
 - **pandas**
 - **seaborn** (<https://seaborn.pydata.org/>)
- Importing **seaborn** modifies the default matplotlib color schemes.

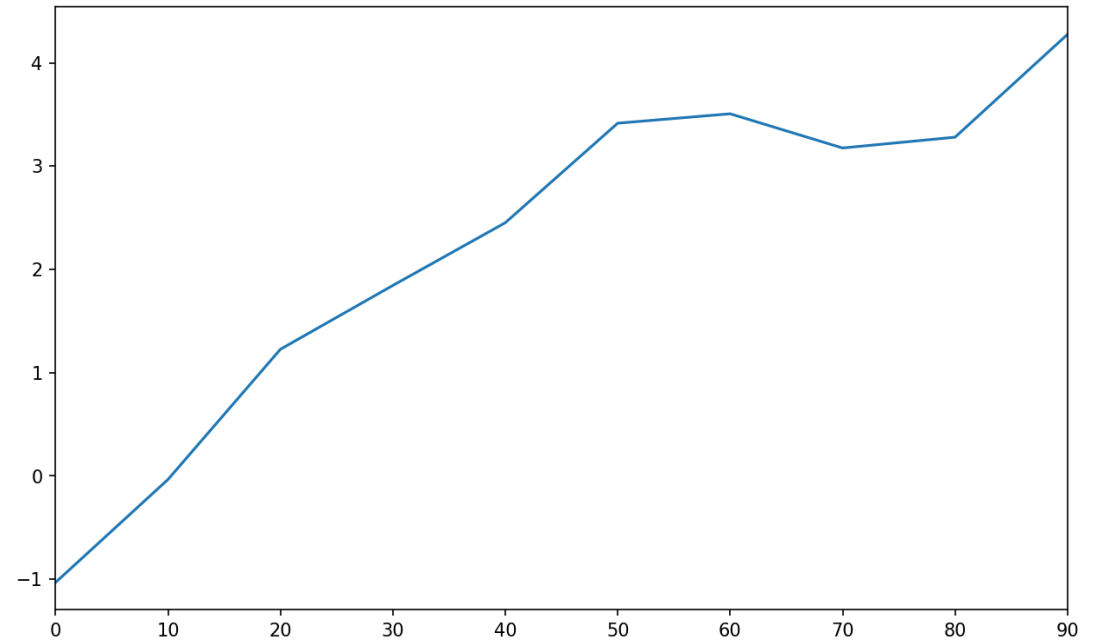
Line Plots

- Series and DataFrame have **plot** for making some basic plot types.

```
s = pd.Series(randn(10).cumsum(),  
              index=np.arange(0, 100, 10))  
s.plot()
```

Options:

`ax, use_index=False, xticks, xlim, yticks, ylim`



Series.plot method arguments

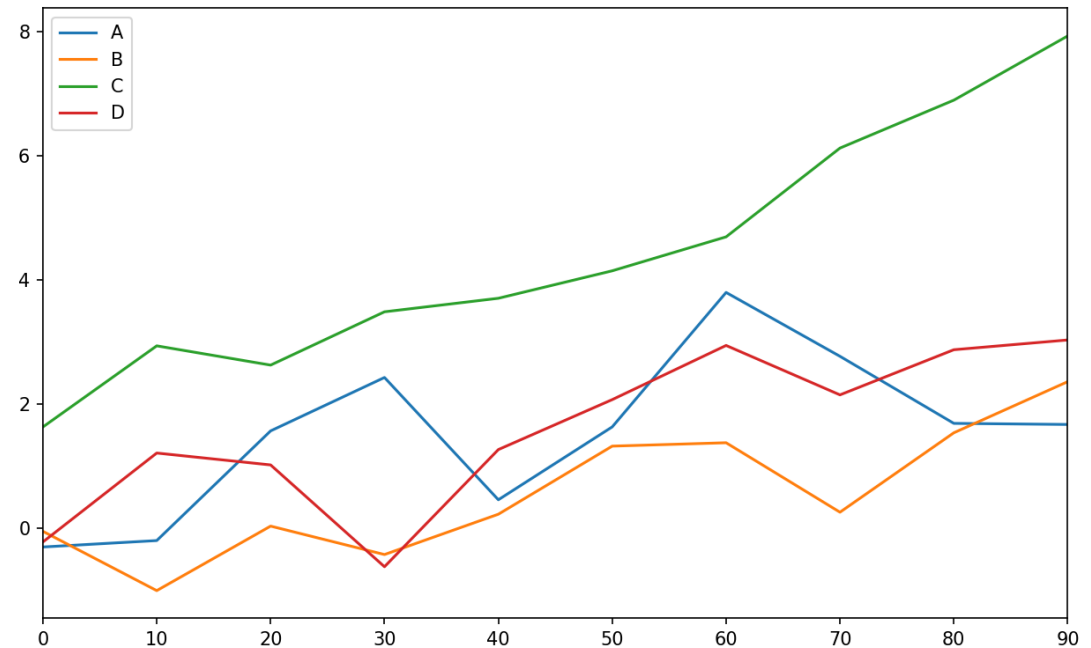
Argument	Description
label	Label for plot legend
ax	matplotlib subplot object to plot on; if nothing passed, uses active matplotlib subplot
style	Style string, like 'ko - - ', to be passed to matplotlib
alpha	The plot fill opacity (from 0 to 1)
kind	Can be 'area', 'bar', 'barh', 'density', 'hist', 'kde', 'line', 'pie'
logy	Use logarithmic scaling on the y-axis
use_index	Use the object index for tick labels
rot	Rotation of tick labels (0 through 360)
xticks	Values to use for x-axis ticks
yticks	Values to use for y-axis ticks
xlim	x-axis limits (e.g., [0, 10])
ylim	y-axis limits
grid	Display axis grid (on by default)

Line Plots

- DataFrame plots each of its **columns** as a **different line** on the same subplot, creating a **legend** automatically.

```
df = pd.DataFrame(randn(10,  
                        4).cumsum(0),  
                  columns=['A', 'B', 'C', 'D'],  
                  index=np.arange(0, 100, 10))  
df.plot()
```

Equivalent to `df.plot.line()`



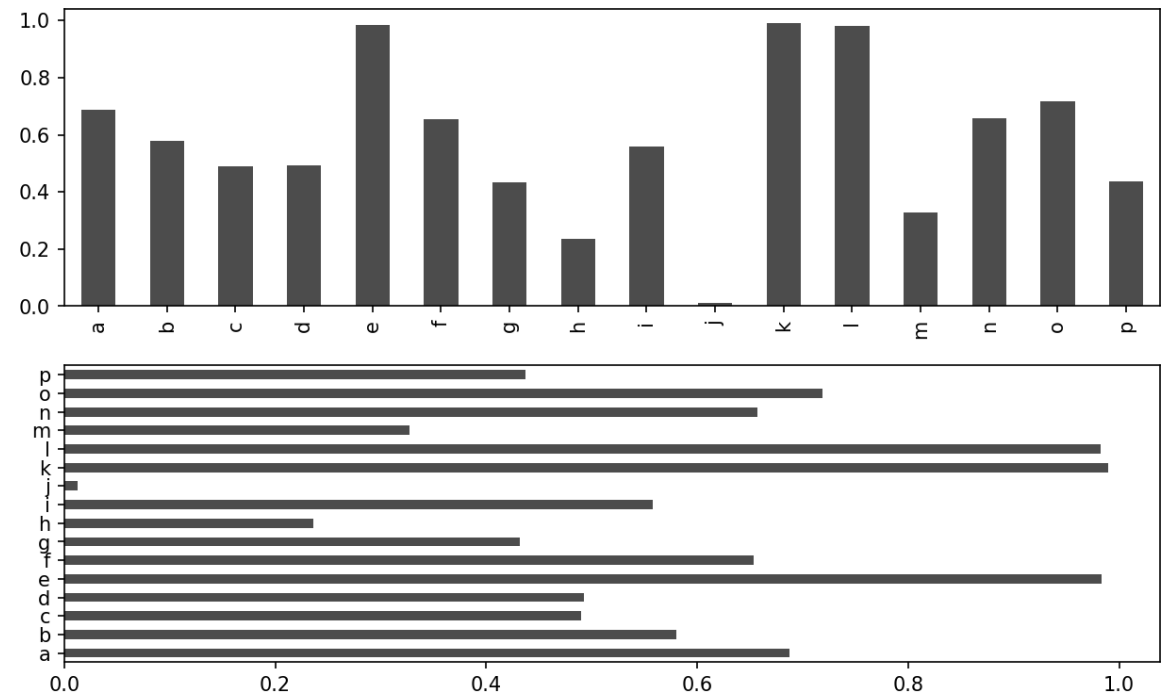
DataFrame-specific plot arguments

Argument	Description
<code>subplots</code>	Plot each DataFrame column in a separate subplot
<code>sharex</code>	If <code>subplots=True</code> , share the same x-axis, linking ticks and limits
<code>sharey</code>	If <code>subplots=True</code> , share the same y-axis
<code>figsize</code>	Size of figure to create as tuple
<code>title</code>	Plot title as string
<code>legend</code>	Add a subplot legend (<code>True</code> by default)
<code>sort_columns</code>	Plot columns in alphabetical order; by default uses existing column order

Bar Plots

- The `plot.bar` and `plot.barh` make vertical and horizontal bar plots.

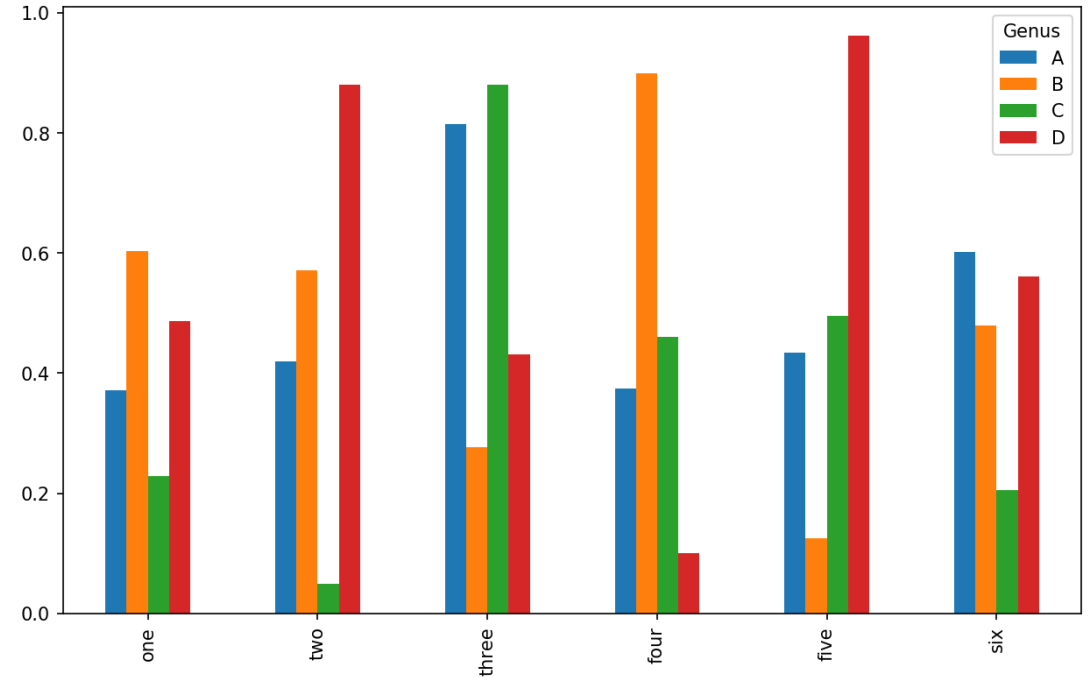
```
fig, axes = plt.subplots(2, 1)
data = pd.Series(rand(16),
                  index=list('abcdefghijklmnop'))
data.plot.bar(ax=axes[0],
              color='k', alpha=0.7)
data.plot.barh(ax=axes[1],
               color='k', alpha=0.7)
```



Bar Plots

- With a DataFrame, bar plots **group** the values in **each row** together in a group in bars, side by side, for each value.

`df.plot.bar()`

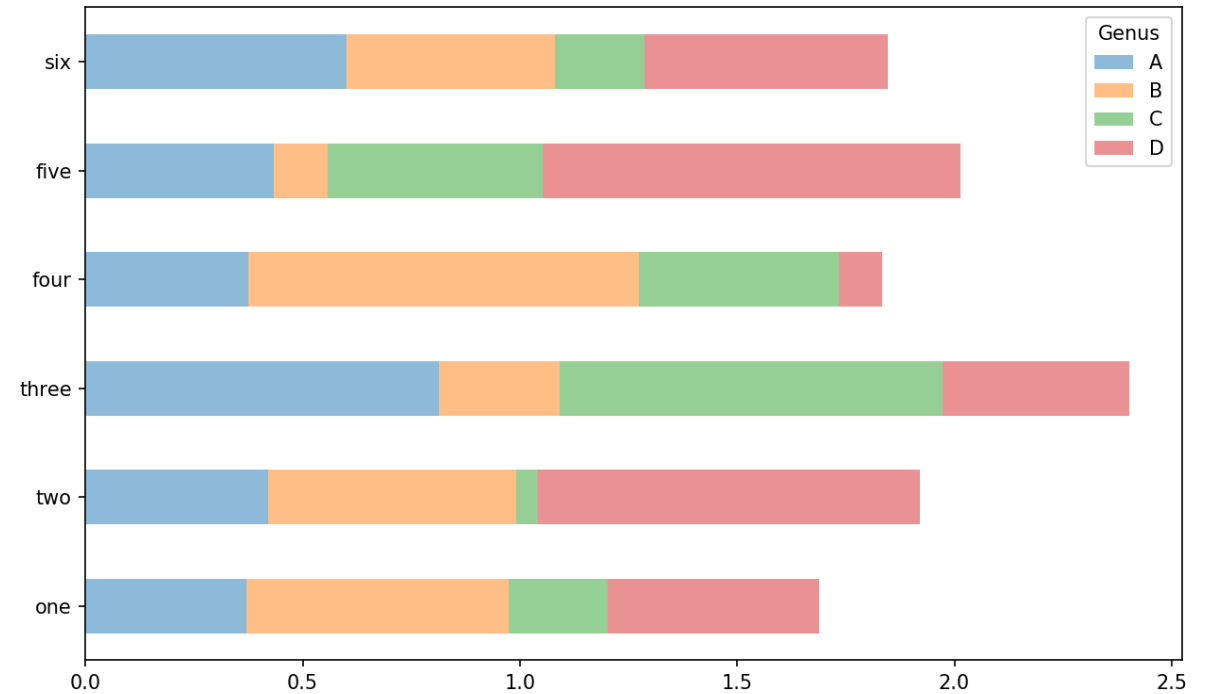


Genus	A	B	C	D
one	0.801554	0.094551	0.469551	0.619210
two	0.208189	0.792578	0.648303	0.260912
three	0.642697	0.847883	0.767702	0.856446
four	0.113493	0.083676	0.283905	0.023767
five	0.220087	0.573322	0.800078	0.514133
six	0.929547	0.272519	0.783754	0.007303

Bar Plots

- We create stacked bar plots from a DataFrame by passing **stacked=True**.

```
df.plot.barh(stacked=True,  
             alpha=0.5)
```

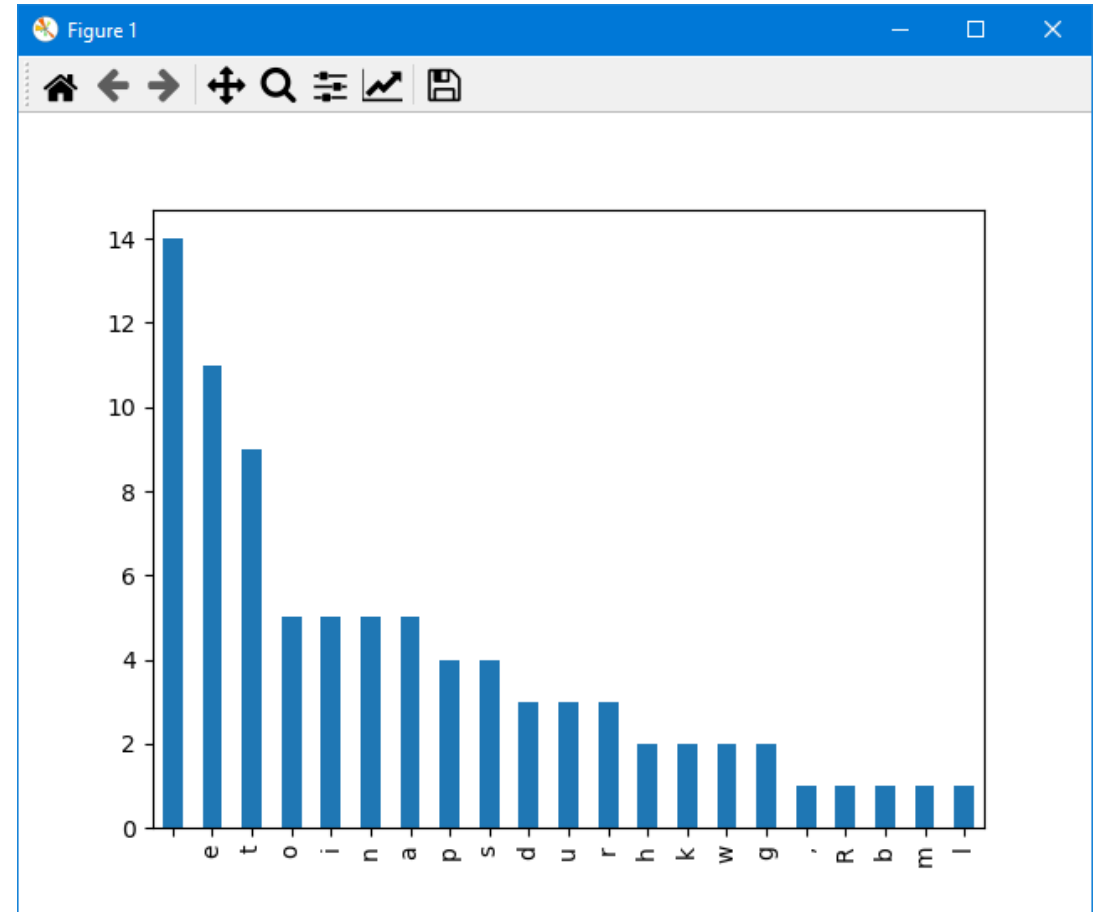


Bar Plots

- A useful recipe for bar plots is to visualize a Series's **value frequency**
`s.value_counts().plot.bar()`.

```
s = pd.Series(list('Returning to  
the tipping dataset used earlier in  
the book, suppose we wanted to  
make'))
```

```
s.value_counts().plot.bar()
```



Bar Plots

- **Example:** Tipping Dataset

Make a stacked bar plot showing the **percentage** of data points for each **party size** on each **day**.

- Hint: use **crosstab**

```
tips = pd.read_csv('tips.csv')
```

```
party_counts = pd.crosstab(  
    tips['day'], tips['size'])
```

party_counts						
size	1	2	3	4	5	6
day						
Fri	1	16	1	1	0	0
Sat	2	53	18	13	1	0
Sun	0	39	15	18	3	1
Thur	1	48	4	5	1	3

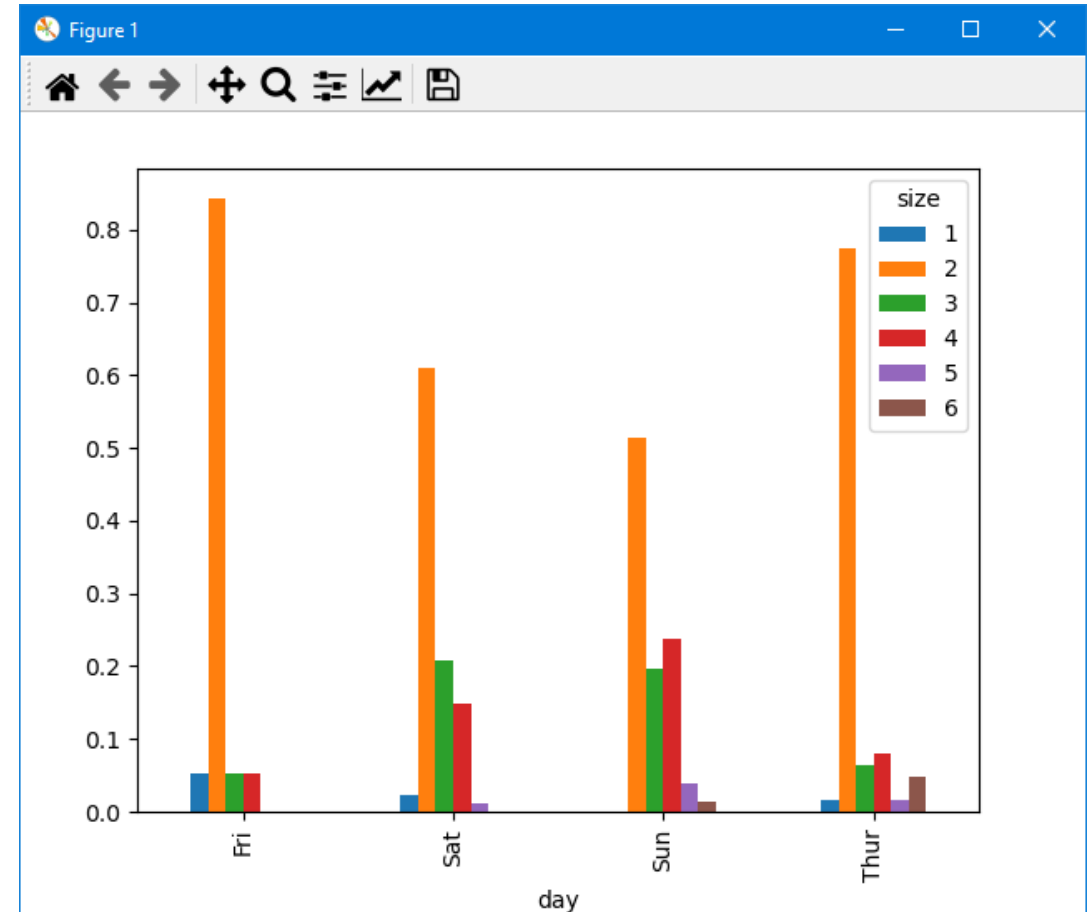
Bar Plots

```
party_pcts = party_counts.div(  
    party_counts.sum(1), axis=0)
```

party_pcts

size	1	2	3	4	5	6
Fri	0.052632	0.842105	0.052632	0.052632	0.000000	0.000000
Sat	0.022989	0.609195	0.206897	0.149425	0.011494	0.000000
Sun	0.000000	0.513158	0.197368	0.236842	0.039474	0.013158
Thur	0.016129	0.774194	0.064516	0.080645	0.016129	0.048387

```
party_pcts.plot.bar()
```



Bar Plots (seaborn)

- **Example:** Use **seaborn** to visualize **tip percent**.

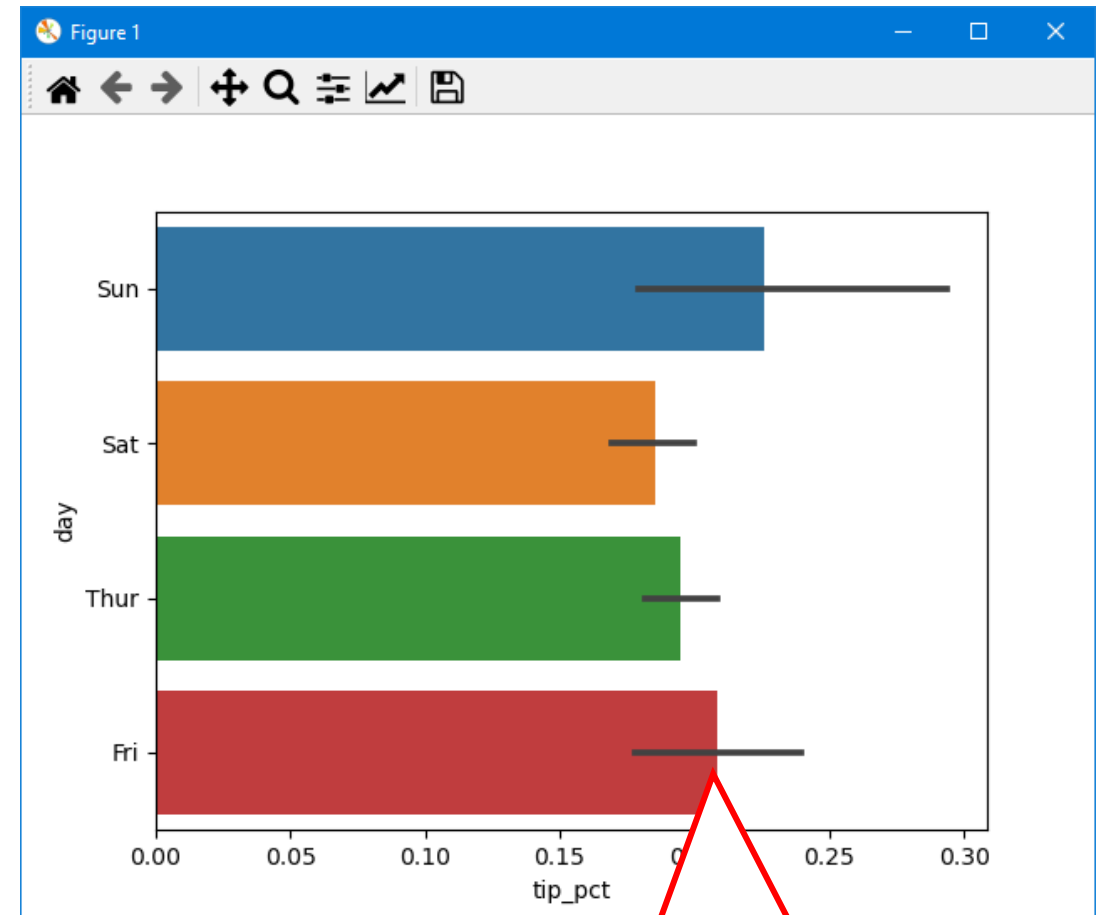
```
import seaborn as sns
```

```
tips['tip_pct'] = tips['tip'] /  
(tips['total_bill'] - tips['tip'])
```

```
tips.head()
```

	total_bill	tip	smoker	day	time	size	tip_pct
0	16.99	1.01	No	Sun	Dinner	2	0.063204
1	10.34	1.66	No	Sun	Dinner	3	0.191244
2	21.01	3.50	No	Sun	Dinner	3	0.199886
3	23.68	3.31	No	Sun	Dinner	2	0.162494
4	24.59	3.61	No	Sun	Dinner	4	0.172069

```
sns.barplot(x='tip_pct', y='day',  
            data=tips, orient='h')
```

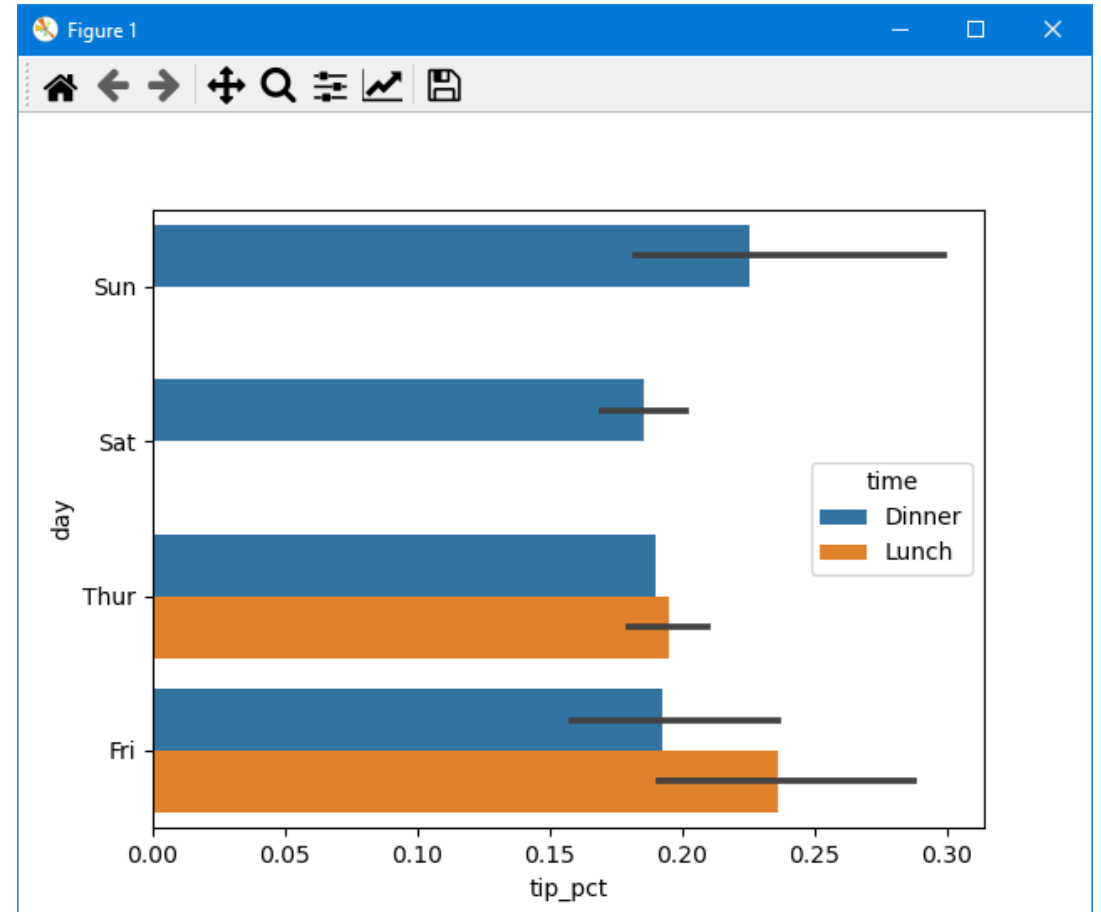


Averages with 95% confidence interval

Bar Plots (seaborn)

- **seaborn.barplot** has a **hue** option that enables us to split by an additional categorical value.

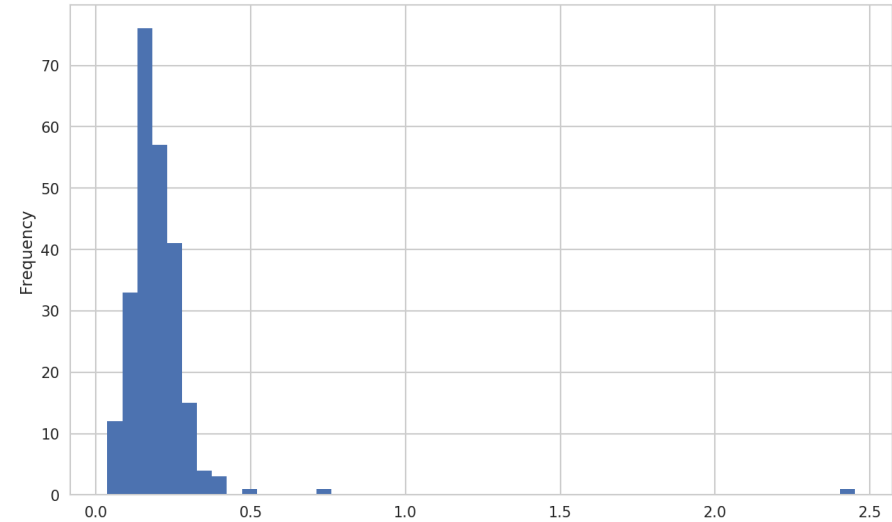
```
sns.barplot(x='tip_pct', y='day',  
            hue='time', data=tips,  
            orient='h')
```



Histograms and Density Plots

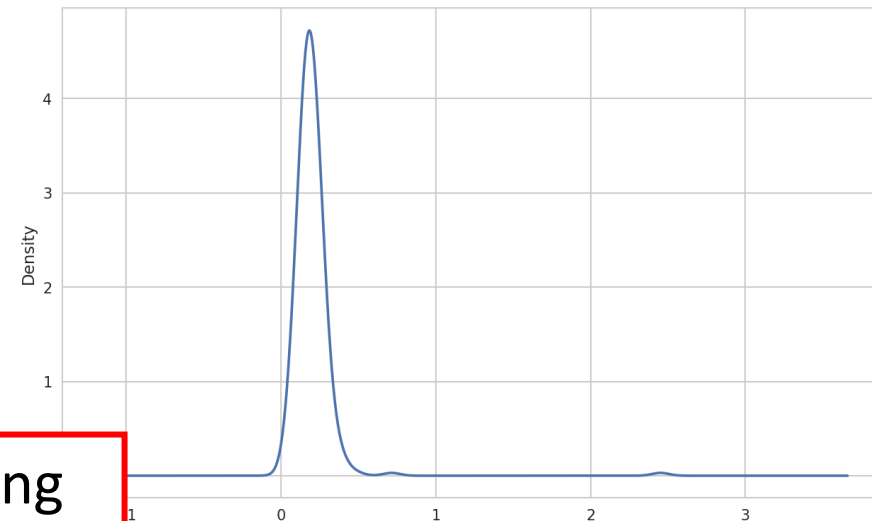
- A **histogram** is a kind of bar plot that gives a discretized display of value frequency.

```
tips['tip_pct'].plot.hist(bins=50)
```



- A **density** plot is formed by computing an estimate of a CPD for the observed data.

```
tips['tip_pct'].plot.density()
```



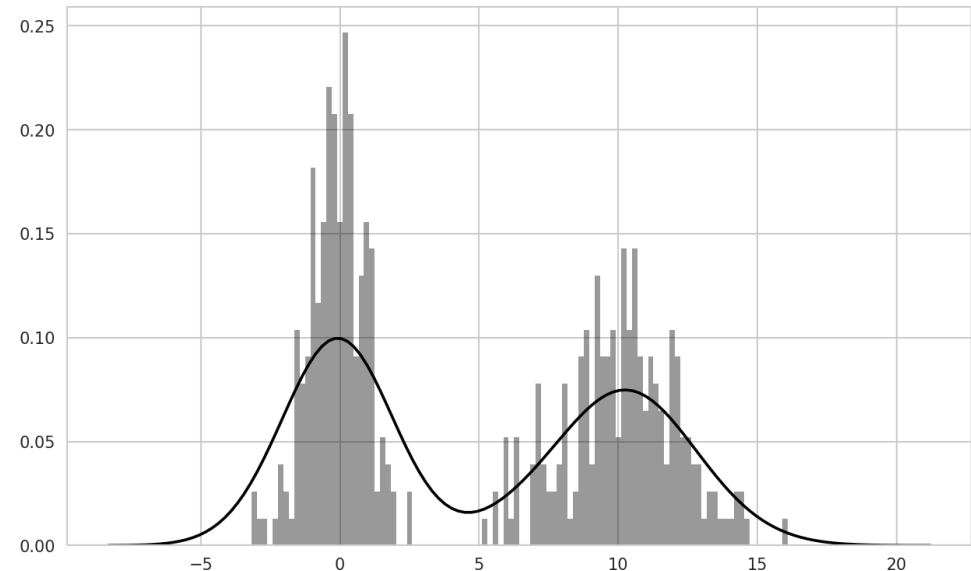
Same as kernel density estimate (KDE) using **plot.kde**

Histograms and Density Plots

- Seaborn's **distplot** can plot both a **histogram** and a **continuous density estimate** simultaneously.
- **Example**: bimodal normal distributions.

```
comp1 = np.random.normal(0, 1,  
                           size=200)  
comp2 = np.random.normal(10, 2,  
                           size=200)
```

```
values = pd.Series(np.concatenate(  
                    [comp1, comp2]))  
sns.distplot(values, bins=100,  
              color='k')
```



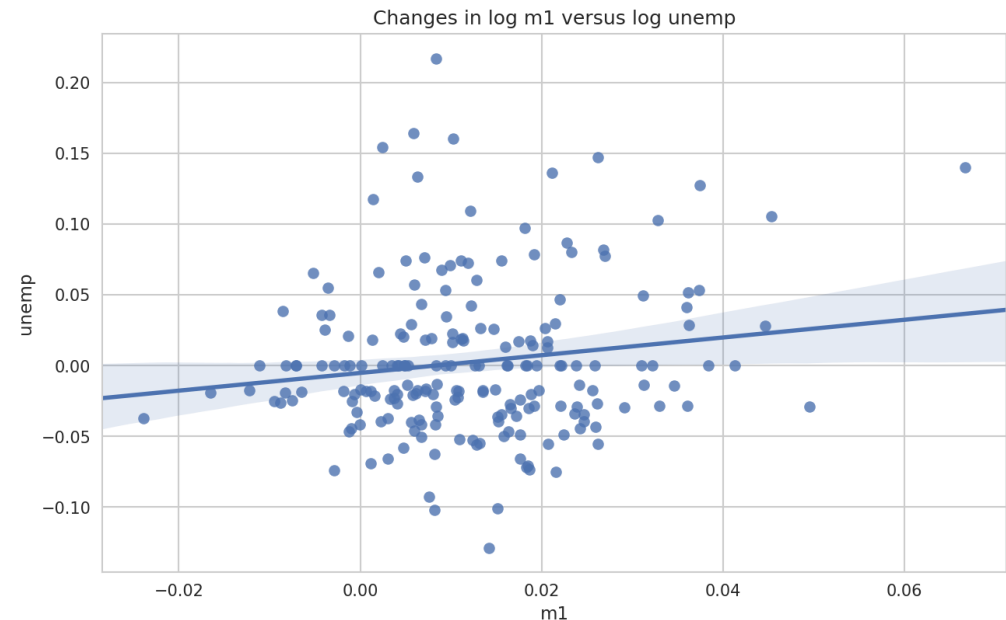
Scatter or Point Plots

- Point plots or **scatter plots** can be a useful way of **examining** the **relationship** between two one-dimensional data series.

- Example:** macrodata.csv

```
macro = pd.read_csv(  
    'macrodata.csv')  
data = macro[['cpi', 'm1',  
    'tbilrate', 'unemp']]  
trans_data = np.log(  
    data).diff().dropna()
```

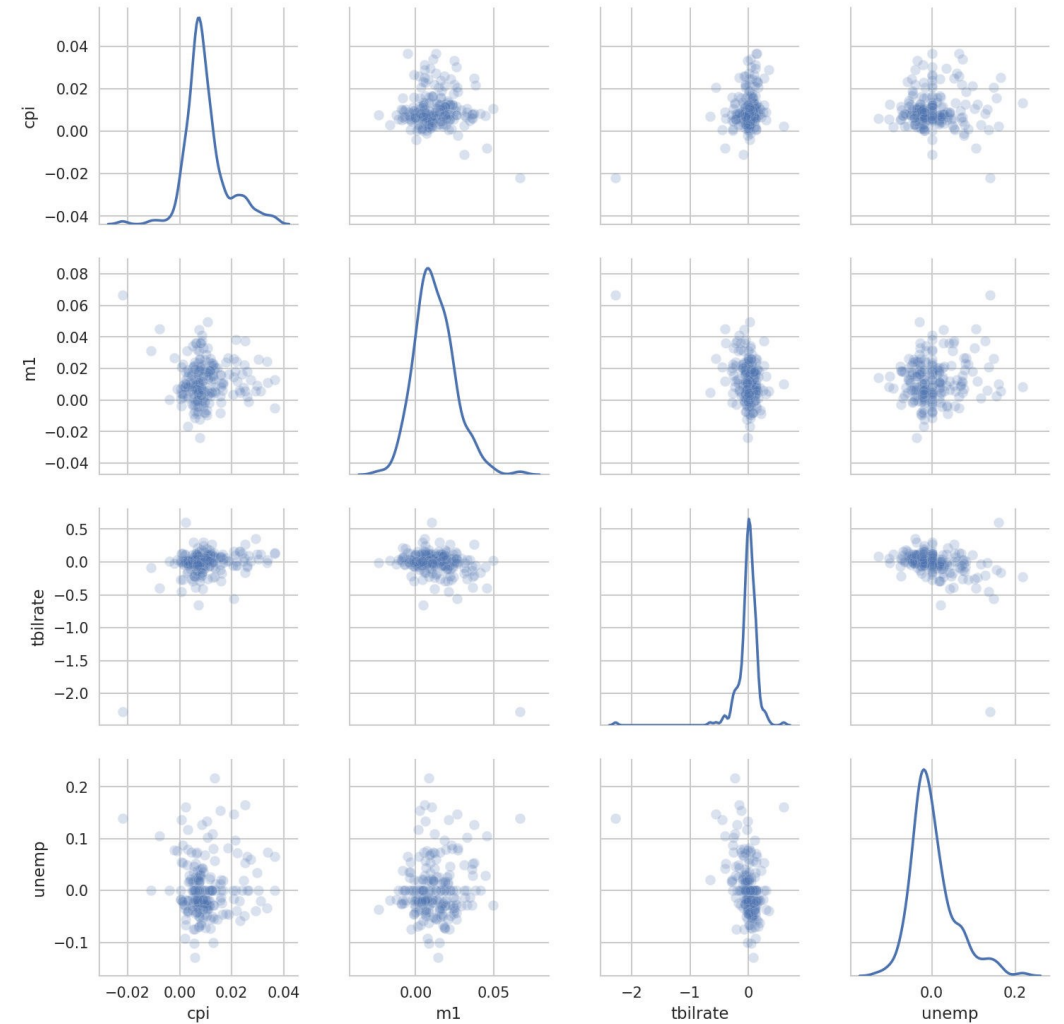
```
sns.regplot('m1', 'unemp',  
            data=trans_data)  
plt.title('Changes in log %s versus  
log %s' % ('m1', 'unemp'))
```



Scatter or Point Plots

- In exploratory data analysis it's helpful to look at all the scatter plots; this is known as **scatter plot matrix**.

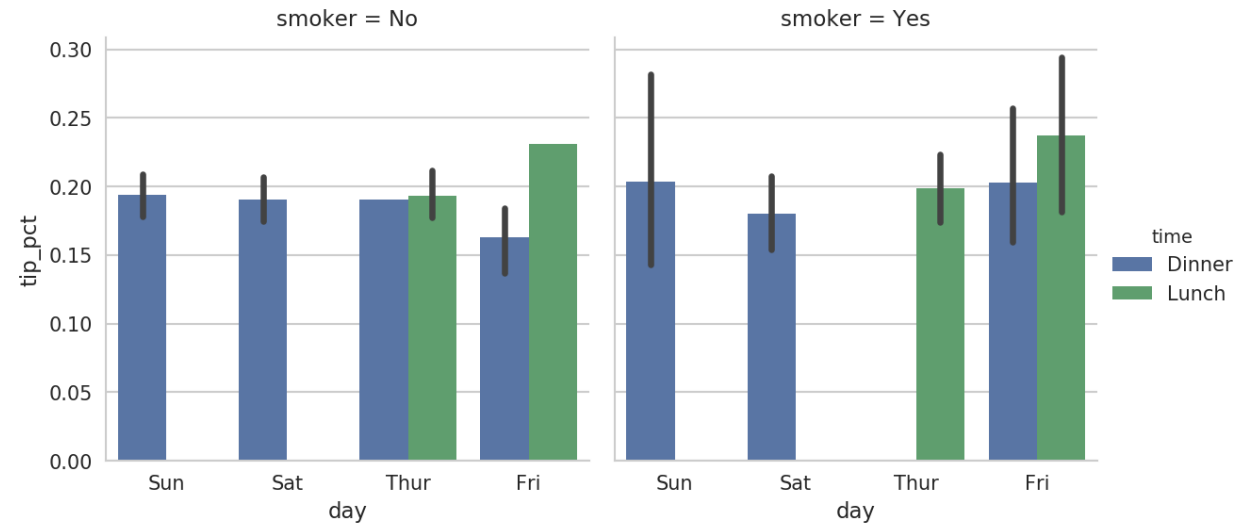
```
sns.pairplot(trans_data,  
              diag_kind='kde',  
              plot_kws={'alpha': 0.2})
```



Facet Grids and Categorical Data

- You can use **facet grids** to visualize data with many categorical variables.
- **Example**: Compare tip percentage with smoking.

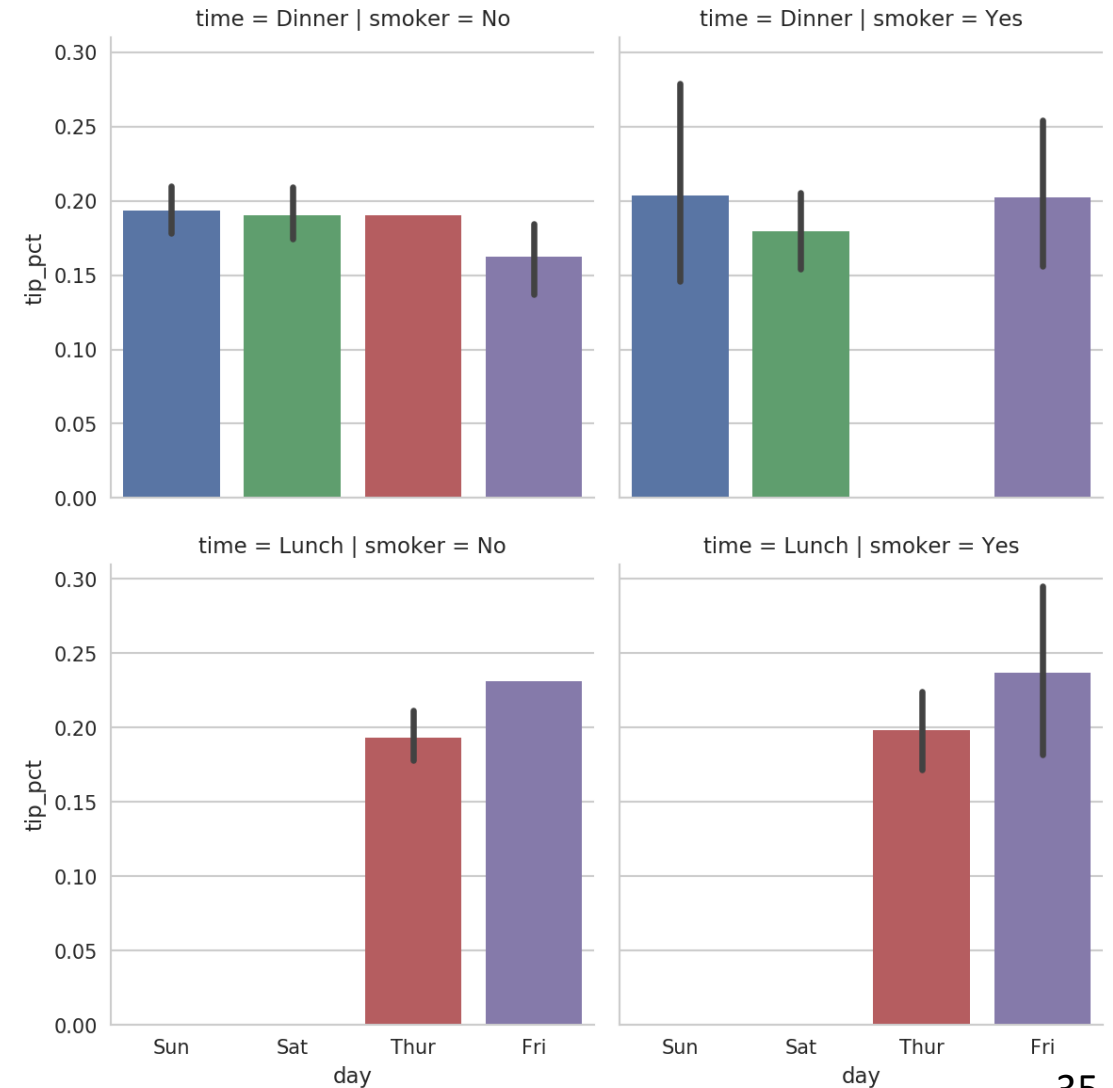
```
sns.factorplot(x='day',  
               y='tip_pct', hue='time',  
               col='smoker', kind='bar',  
               data=tips[tips.tip_pct < 1])
```



Facet Grids and Categorical Data

- **Example:** Show time in a different facet.

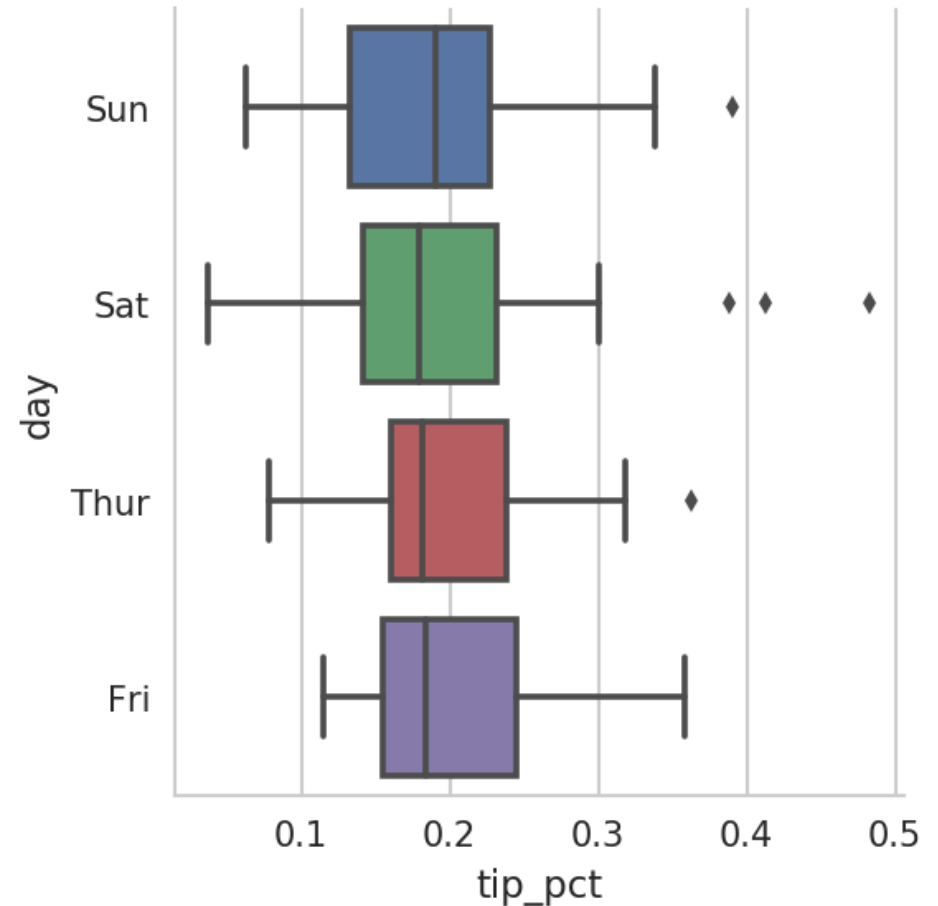
```
sns.factorplot(x='day',  
               y='tip_pct', row='time',  
               col='smoker', kind='bar',  
               data=tips[tips.tip_pct < 1])
```



Facet Grids and Categorical Data

- **Example:** Draw a box plot to show the median, quartiles, and outliers.

```
sns.factorplot(x='tip_pct',  
               y='day', kind='box',  
               data=tips[tips.tip_pct < 0.5])
```



Homework

- Solve the homework on **data wrangling and visualization**.

Summary

9.1 A Brief **matplotlib** API Primer

- Figures and Subplots
- Colors, Markers, and Line Styles
- Ticks and Labels
- Saving Plots to File
- matplotlib Configuration

9.2 Plotting with **pandas** and **seaborn**

- Line Plots
- Bar Plots
- Histograms and Density Plots
- Scatter or Point Plots
- Facet Grids and Categorical Data