Data Science and Machine Learning for Engineering Applications

Matplotlib

DataBase and Data Mining Group

Salvatore Greco
Andrea Pasini
Flavio Giobergia
Elena Baralis
Tania Cerquitelli

Matplotlib introduction

PoliTo DBMG

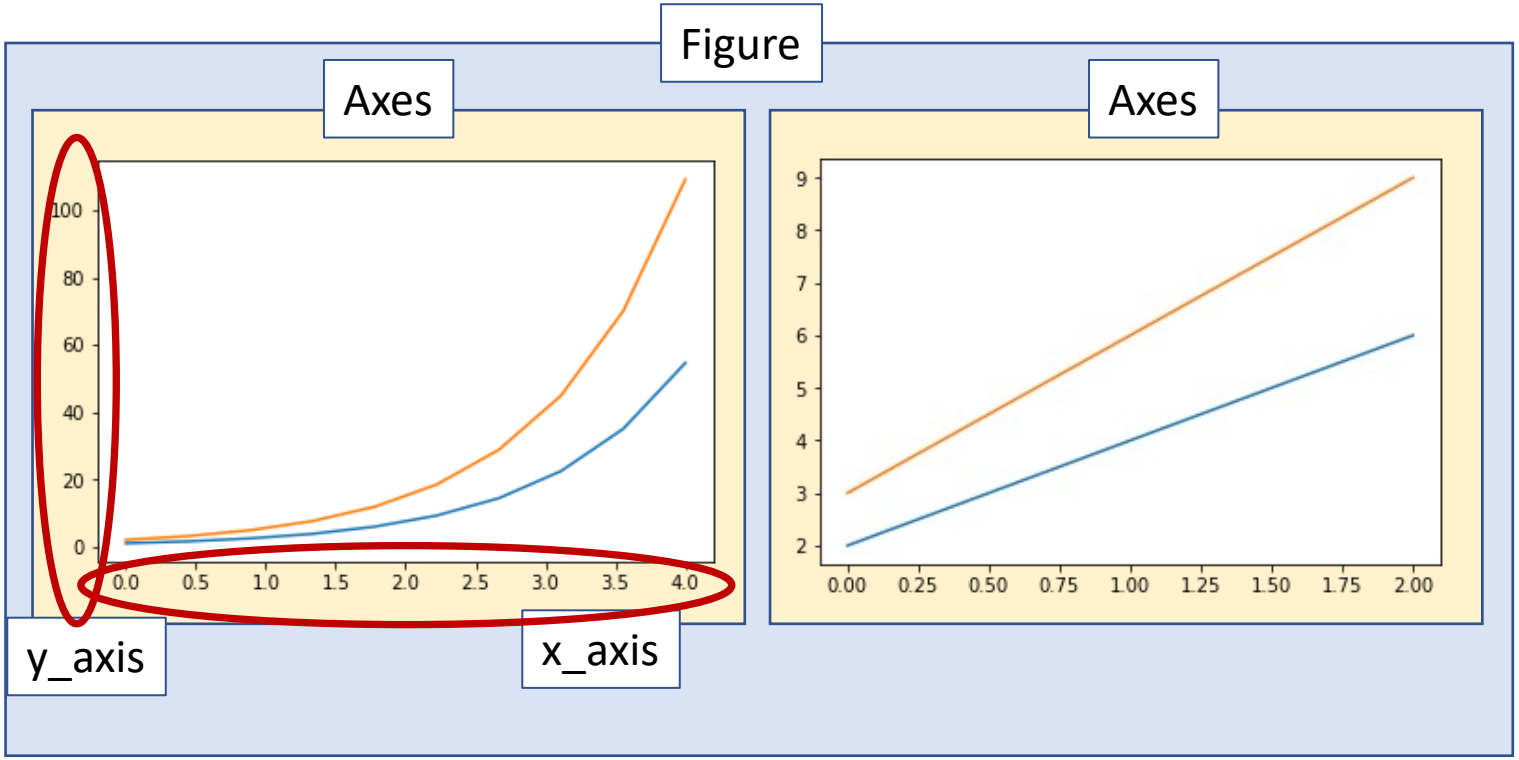
- **Two of the most commonly used graphical libraries are:**
  - **Matplotlib**
    - We present here only a very **short introduction** as the library is fairly large and visualization is not the focus of this course
  - **Seaborn** (data visualization library based on Matplotlib)
    - **Not covered by this course**

Matplotlib introduction

PoliTo DBMG

- **Matplotlib**
  - Set of methods that make matplotlib work like **matlab**
  - It has 2 **interfaces**:
    - **Matlab style plotting (Stateful)** ☹
      - Plotting methods are called from the **pyplot** package
      - They all work on the **current** Figure and Axes
    - **Object oriented (Stateless)** ☺
      - Plot functions are called as **methods** of a specific Figure and Axes
      - This allows modifying **many objects at a time** (the system does not keep a "current object" state)

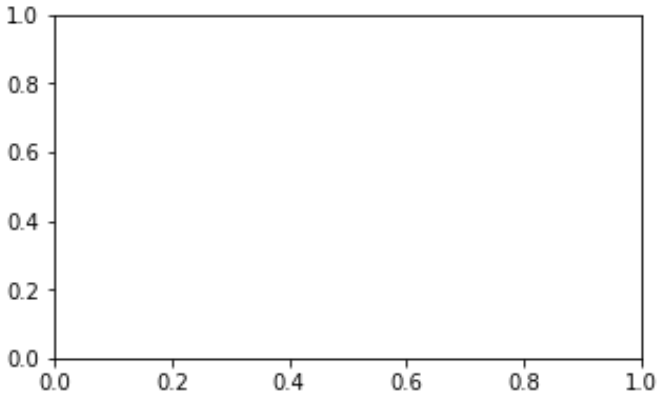
- **Figures and Axes**



Matplotlib introduction

- **Creation of a new figure:**

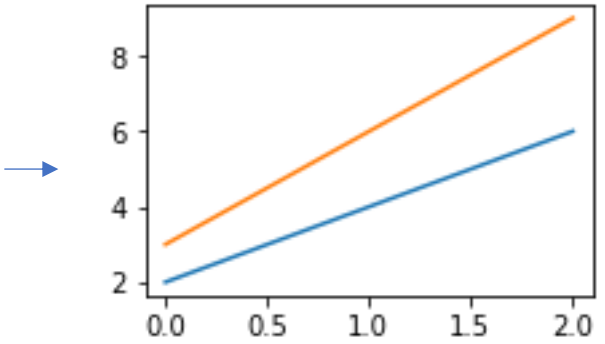
```
import matplotlib.pyplot as plt
fig, ax = plt.subplots(figsize=(5, 3))
plt.show()
```



- Subplots returns a new **Figure** and its **Axes** object
- **figsize** specifies the figure size (width, height) in inches
- By default ax is a single Axes object (1 Figure with a single Axes)

Drawing a line plot (single Axes object)

```
fig, ax = plt.subplots(figsize=(3, 2))
ax.plot([0,1,2],[2,4,6])
ax.plot([0,1,2],[3,6,9])
plt.show()
```

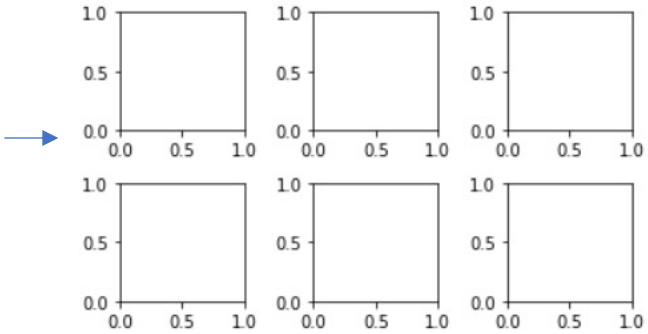


- The plot method of a specific Axes takes as input two lists (or NumPy arrays): **x, y** coordinates of the points
- The default style draws **segments** passing through the specified coordinates
- Subsequent calls of plot add new line to the same Axes

Matplotlib introduction

- **Creation of a new figure:**

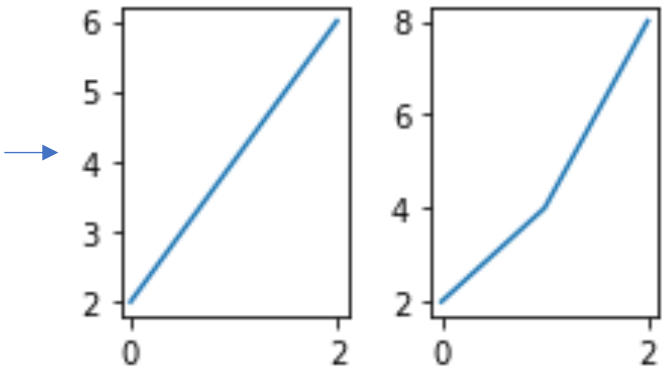
```
fig, ax = plt.subplots(2, 3, figsize=(5, 3))
plt.tight_layout()
plt.show()
```



  - The first two parameters of subplots specify to create a figure with **2 rows, 3 columns** (6 Axes objects)
  - **tight_layout()** is necessary at the end to let the subplots fit the frame size without blank spaces at the borders

- **Drawing a line plot (multiple Axes object)**

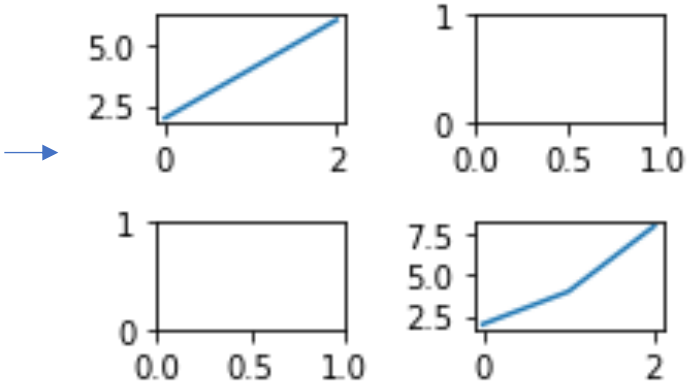
```
fig, ax = plt.subplots(1, 2,
                       figsize=(3, 2))
ax[0].plot([0,1,2],[2,4,6])
ax[1].plot([0,1,2],[3,6,9])
plt.tight_layout()
plt.show()
```



  - The ax object is a **Numpy array** with the created Axes objects
  - It has **shape = (n,)** if the figure has 1 row and n columns

- **Drawing a line plot (multiple Axes object)**

```
fig, ax = plt.subplots(2, 2,
                       figsize=(3, 2))
ax[0, 0].plot([0,1,2],[2,4,6])
ax[1, 1].plot([0,1,2],[3,6,9])
plt.tight_layout()
plt.show()
```



  - It has **shape = (m, n)** if the figure has m rows and n columns

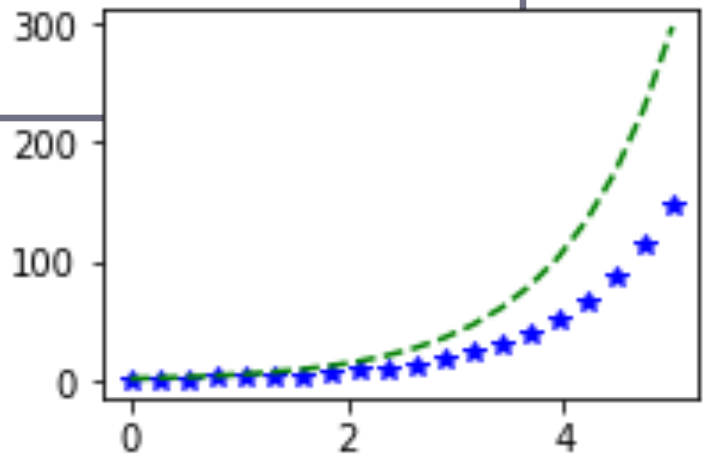
Plot types

- With Matplotlib you can design different plot types
- **The most common are:**
  - Line plot
  - Scatter plot
  - Bar chart

Line plot

- Allows displaying a sequence of points/segments that **share the same properties**
  - E.g. same size, color, width, …

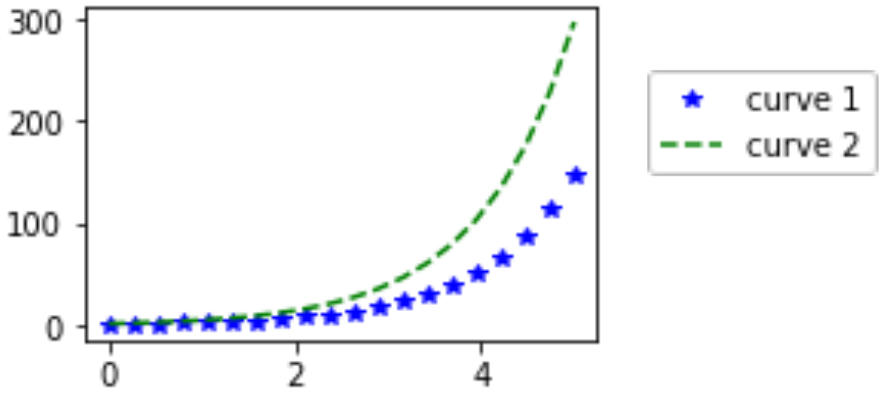
```
x = np.linspace(0, 5, 20)
y = np.exp(x)
fig, ax = plt.subplots(figsize=(3, 2))
ax.plot(x, y, c='blue', linestyle='', marker='*')
ax.plot(x, 2*y, c='green', linestyle='--')
plt.show()
```



Line plot

- Different plots can be associated to **labels** to be displayed in a **legend**

```
x = np.linspace(0, 5, 20)
y = np.exp(x)
fig, ax = plt.subplots(figsize=(3, 2))
ax.plot(x, y, c='blue', linestyle='', marker='*', label='curve 1')
ax.plot(x, 2*y, c='green', linestyle='--', label='curve 2')
ax.legend(loc=(1.1, 0.5))
plt.show()
```



Line plot

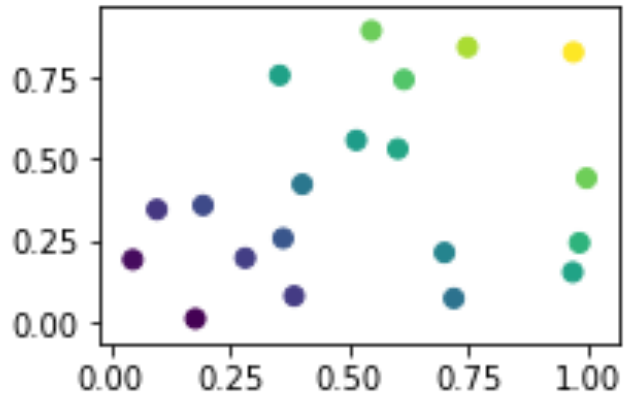
PoliTo DBMG

- **linestyle** specifies the type of line
  - Examples: '-', '--' (or 'dashed'), ':' (or 'dotted')
- **marker** specifies the type of points to be drawn
  - Examples: 'o', '*', '+', '^'
- **c** specifies the color to be applied to markers and segments
  - Examples: 'red', 'orange', 'grey'
  - Examples: '#0F0F6B' (RGB)
  - Examples: (0.5, 1, 0.8, 0.8) (RGBA tuple)

Scatter plot

- Allows displaying a set of points and assign them custom properties
  - E.g. different color, size

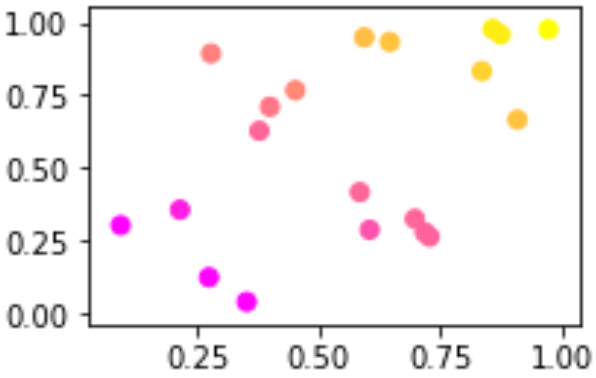
```
x = np.random.rand(20)
y = np.random.rand(20)
colors = x + y     # color as a function of x and y
fig, ax = plt.subplots(figsize=(3, 2))
ax.scatter(x, y, c=colors)
plt.show()
```



Scatter plot

- **c=colors** associate a number (float or integer) to each point
  - In the same sequence as they appear in x, y)
  - These numbers are used to select a color from a specific **colormap**
    - https://matplotlib.org/users/colormaps.html

```
colors = x + y     # color as a function of x and y
fig, ax = plt.subplots(figsize=(3, 2))
ax.scatter(x, y, c=colors, cmap='spring')
plt.show()
```



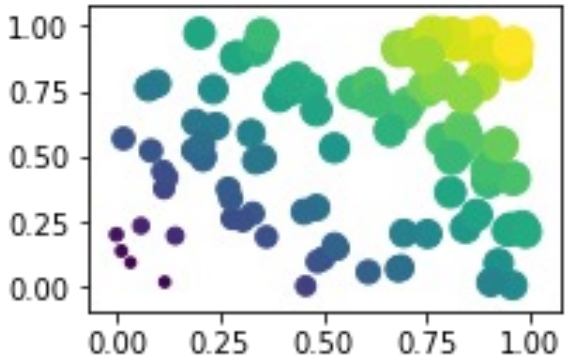
Scatter plot

- **c=colors** associate a number (float or integer) to each point
  - Matplotlib considers the range of values of c to fit the whole range of colors of a colormap
  - c = [101, 120, 50, 60] -> range is 50-120

Scatter plot

- The size of each point can be set with the parameter **s**
- Size is the area in **dpi** (dots per inch)

```
x = np.random.rand(20)
y = np.random.rand(20)
colors = x + y     # color as a function of x and y
area = 100*(x+y)   # size as a function of x, y
fig, ax = plt.subplots(figsize=(3, 2))
ax.scatter(x, y, c=colors, s=area)
plt.show()
```

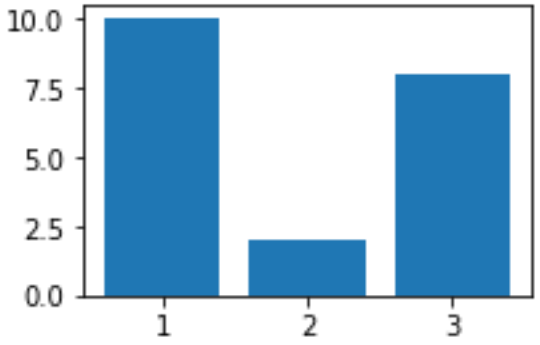


Bar chart

- Allows displaying a sequence of numbers as vertical or horizontal bars

```
height = [10, 2, 8]
x = [1, 2, 3]      # position of the bars, x axis

fig, ax = plt.subplots(figsize=(3, 2))
ax.bar(x, height)
plt.show()
```

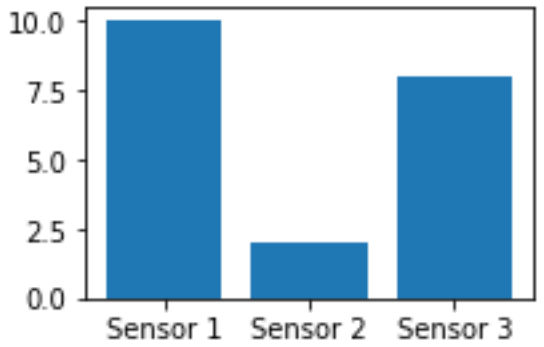


Bar chart

- Ticks on the horizontal axis can be **labeled** with some text

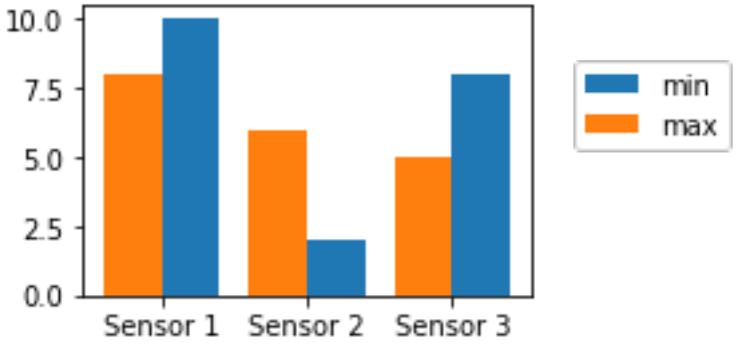
```
height = [10, 2, 8]
x = [1, 2, 3]      # position of the bars, x axis
labels = ['Sensor 1', 'Sensor 2', 'Sensor 3']

fig, ax = plt.subplots(figsize=(3, 2))
ax.bar(x, height, tick_label=labels)
plt.show()
```



Bar chart

- Bars can be grouped

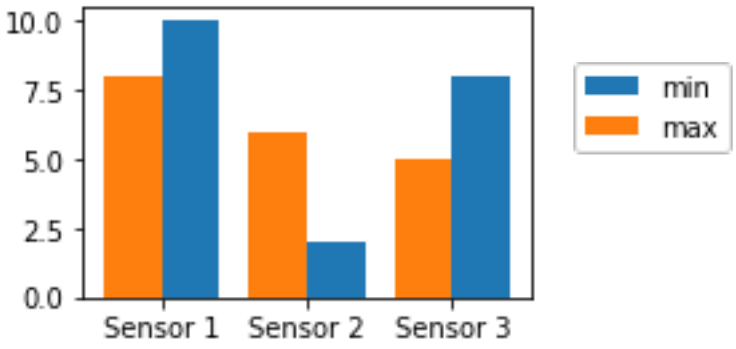



```
height_min = [10, 2, 8]
height_max = [8, 6, 5]
x = np.arange(3)
width = 0.4
labels = ['Sensor 1', 'Sensor 2', 'Sensor 3']

fig, ax = plt.subplots(figsize=(3, 2))
ax.bar(x+width/2, height_min, width=width, label='min')
ax.bar(x-width/2, height_max, width=width, label='max')
ax.set_xticks(x)              # setup positions of x ticks
ax.set_xticklabels(labels)    # set up labels of x ticks
ax.legend(loc=(1.1, 0.5))     # x, y position, in percentage
plt.show()
```

Bar chart

- Bars can be grouped




```
height_min = [10, 2, 8]
height_max = [8, 6, 5]
x = np.arange(3)
width = 0.4
labels = ['Sensor 1', 'Sensor 2', 'Sensor 3']

fig, ax = plt.subplots(figsize=(3, 2))
ax.bar(x+width/2, height_min, width=width, lab
ax.bar(x-width/2, height_max, width=width, lab
ax.set_xticks(x)              # setup positions
ax.set_xticklabels(labels)    # set up labels o
ax.legend(loc=(1.1, 0.5))     # x, y position,
plt.show()
```

However, other libraries might make our life easier!

```
df = pd.DataFrame({
    "min": height_min,
    "max": height_max
  },
  index=labels
)
df.plot.bar()
```

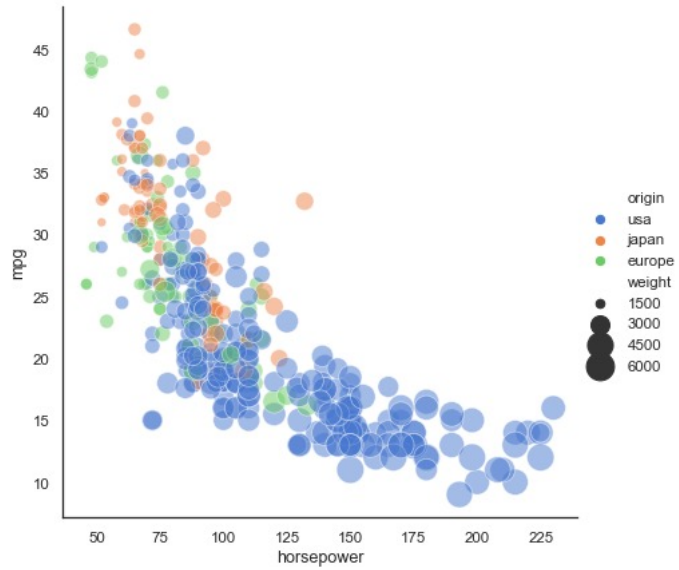
Writing images to file

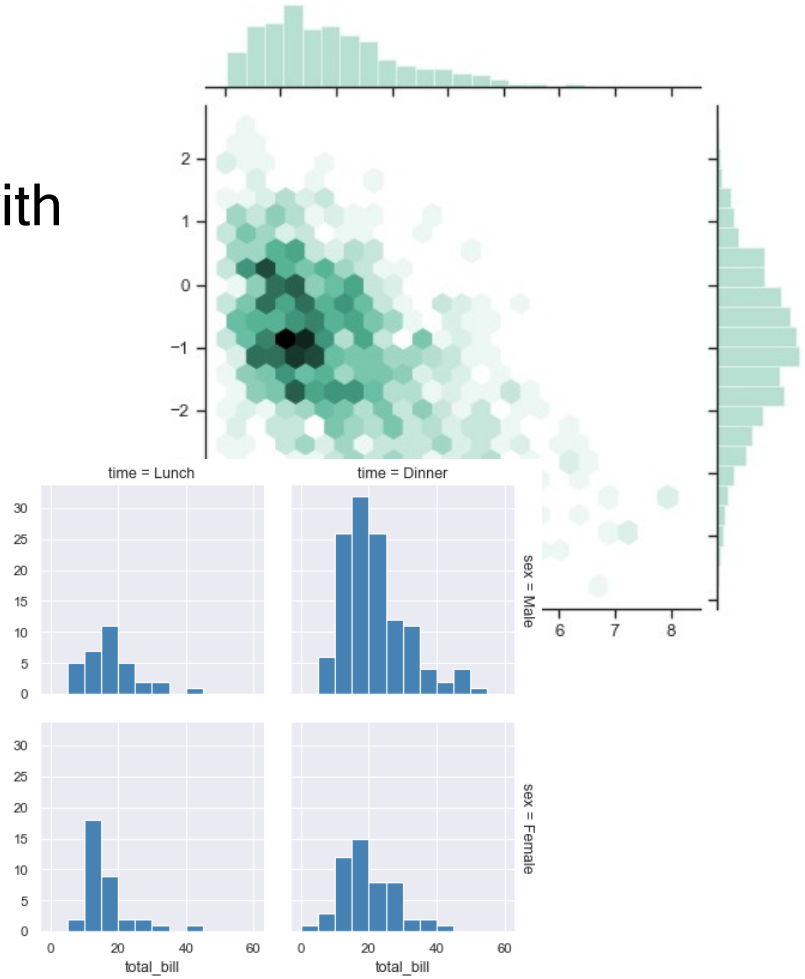
- Generated figures can be **saved** to file with different formats

```
fig, ax = plt.subplots(figsize=(3, 2))
ax.plot([0,1,2],[2,4,6])
ax.plot([0,1,2],[3,6,9])
fig.savefig(“./out/test.png") # or ‘.jpg’, ‘.eps’, ‘.pdf’
```

Seaborn

- Based on Matplotlib
  - High level **interface** for drawing complex chart with attractive visual impact





References

- **Matplotlib website:**
  - https://matplotlib.org/
- **Seaborn website:**
  - https://seaborn.pydata.org/