Import Library

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

Basic Line Plot

```
x_values
days = [0, 1, 2, 3, 4, 5, 6]
y_values1
money_spent = [10, 12, 12, 10, 14, 22, 24]
y_values2
money_spent_2 = [11, 14, 15, 15, 22, 21, 12]
assigend to one plot
plt.plot(days, money_spent)
plt.plot(days, money_spent_2)
plt.show()
```

Subplots

```
# Create subplots
plt.subplot(rows, columns, index_of_subplot)
# Example
# First Subplot
plt.subplot(1, 2, 1)
plt.plot(x, y, color='green')
# Second Subplot
plt.subplot(1, 2, 2)
plt.plot(x, y, color='steelblue')
# Format Subplots
plt.subplots_adjust(arguements)
left, right, top, bottom -margin
wspace, hspace horizontal/vertical margin between plots
```

The object that contains all subplots is called *figure*
Always put specific Attributes (color, markers, ...) for a subplot directly under *plt.plot()*

Linestyles

plt.plot(x, y, ***style***=*" "*)

Keywords to put in for **style**:

color= *green*, *#AAAAAA*

linestyle= dotted: **:**, dashed: **--** or **-.**

marker= *o*, ***, *s*, *x*, *d*, *h*

linewidth= 1, 2, ...

Linestyles (cont)

alpha= 0.1 - 1

Boilerplate Styles:

plt.style.use*("fivethirtyeight")*

plt.style.use*("ggplot")*

plt.style.use*("seaborn")*

plt.style.use*("default")*

Legends

```
# Create Legend
plt.legend(["first_line", "second_line", loc=])
# loc Numbercode
1 upper left
2 upper right
3 lower left
4 lower right
5 right
6 center left
7 center right
8 lower center
9 upper center
10 center
```

loc specifies the legends location (if not specified: finds "best" location)

Figures

```
# Create Figure with custom size
plt.figure(figsize=(width, heigth))
plt.plot(x, y)
plt.savefig('tall_and_narrow.png/ .svg/ .pdf')
```

When we’re making lots of plots, it’s easy to end up with**lines that have been plotted and not displayed.** If we’re not careful, these “forgotten” lines will show up in your new plots. In order to be sure that you don’t have any stray lines, you can use the command **plt.close('all')** to clear all existing plots before you plot a new one.

Modify Ticks

```
# Specify subplot to modify
ax1 = plt.subplot( row, column, index)
# Attributes
ax1.set_xticks([1, 2, 4])
ax1.set_yticks([0.1, 0.2, ...])
ax1.set_xticklabels(["Jan", "Feb", "Apr"], rotation=30)
# rotation=degrees rotates the labels
ax1.set_yticklabels(["10%", "20%", ...])
```

We have to do it this way, even if we only have one plot

Axis and Labels

Zoom in or out of the plot:

plt.axis(*x_min*, *x_max*, *y_min*, *y_max*)

Labeling the Axes:

plt.xlabel("*str* ")/ plt.ylabel() / plt.title()

Add Text to Graph

```
plt.text(x_coord, y_coord, "text");
```

Simple Bar Chart

plt.bar(range(len(y_values)), y_values)

We use **range(len(y_values))** to get a tick for each value we want to represent in the Bar Chart

Scatter Plot

```
plt.scatter(x_values, y_values)
```

Side-By-Side Bars

```
# We have to specifiy the location of each Dataset in the Plot using this pattern:
n = ? # Number of specific dataset
t = ? #  Number of datasets
d = ? #  Number of sets of bars
w = 0.8 #  Width of each bar
x_values1 = [t*element + w*n for element in range(d)]
# Get x_values in the middle of both bars
middle_x = [ (a + b) / 2.0 for a, b in zip(x_values1, x_values2)]
```

Stacked Bars

```
# We use the keyword bottom to do this
# The top bar will have bottom set as height
# First Bar
video_game_hours = [1, 2, 2, 1, 2]
plt.bar(range(len(video_game_hours)),
  video_game_hours)
# Second Bar
book_hours = [2, 3, 4, 2, 1]
plt.bar(range(len(book_hours)),
  book_hours,
  bottom=video_game_hours)
# Get each bottom for 3+ bars
sport_hours = np.add(video_game_hours, book_hours)
```

If we want to compare *"different sub-attributes from one attribute"*we can use stacked bar charts. For example:
Attribute: Entertainment hours
Sub-Attributes: Gaming, Reading, ...

Error Bars

```
# Use the keyword yerr to repersent the error range
values = [10, 13, 11, 15, 20]
yerr = [1, 3, 0.5, 2, 4] # singe value possible
plt.bar(y, x, yerr=yerr, capsize=10)
plt.show()
```

If we want to present an uncertainty Range within a Bar Chart we can use Error Bars

Fill Between (Line Plot)

```
x = range(3)
y = [10, 12, 13]
y_lower = [8, 10, 11]
y_upper = [i + 2 for i in y_values]
# Calculate a % deviation
y_lower_bound = [element - (element * error_in_decimal) for element in original_list_of_y_values]
#this is the shaded error
plt.fill_between(x, y_lower, y_upper, alpha=0.2)
#this is the line itself
plt.plot(x, y)
plt.show()
```

Returns a shaded are around the line

Pie Chart

```
payment_names = ["Card Swipe", "Cash", "Apple Pay", "Other"]
payment_freqs = [270, 77, 32, 11]
# Creating Pie Chart
plt.pie(payment_freqs)
plt.axis('equal')
# Two Methods for Labeling
# First Method
plt.legend(payment_names)
# Second Method (directly when creating)
plt.pie(payment_freqs, labels=payment_names)
Show percentages of total in each slice:
plt.pie(payment_freqs, labels=payment_names, autopct='%0.1f%%')
# autopct takes a string formatting instruction
# %d%% -> round to decimal
plt.show()
```

Histogram

```
# Create one Histogram
plt.hist(dataset, range=(0,100), bins=20)
# Specifiy number of bins (default = 10)
# Create multiple Histograms
plt.hist(a, alpha=0.5, normed=True)
plt.hist(b, histtype='step', linewidth=2 normed=True)
```

*# Specify **alpha** for opacity or use **histtype** to draw just the outline*
*# Use **linewidth** to specifiy the linewidth of the outline*
*# Use the keyword **normed** to normalize the histograms*

Normalize divides the x_values by a constat such that the area under the curve sums to 1