WELLCOME

DATA SCIENCE Fundamental

EMBARKING ON A JOURNEY INTO DATA SCIENCE

YA MANON

INTRO TO MATPLOTLIB

INTRO TO MATPLOTLIB

In this section we'll introduce the **Matplotlib** library and use it to build & customize several chart types, including line charts, bar charts, pie charts, scatterplots, and histograms… .

TOPICS WE'LL COVER:

- Matplotlib Basics
- Object-Oriented Plotting
- Chart Formatting
- Chart Types

GOALS FOR THIS SECTION:

- Understand the difference between the two primary Matplotlib plotting frameworks
- Identify the key components of an object-oriented plot
- Build different variations of line, bar and pie charts, as well as scatter plots and histograms
- Customize your charts by adding custom titles, labels, legends, annotations and much more!

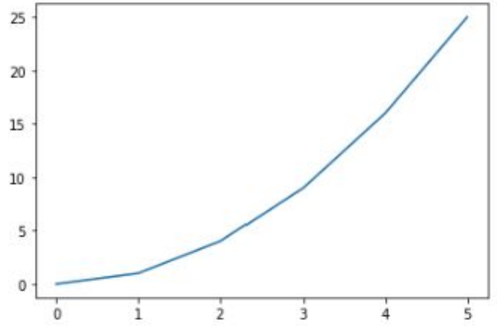
MEET MATPLOTLIB

Matplotlib is an open-source Python library built for data visualization that lets you produce a wide variety of highly customizable charts & graphs

```
import matplotlib.pyplot as plt

plt.plot([0, 1, 4, 9, 16, 25])
```

'plt' is the standard alias for Matplotlib

*The **plot()** function creates a line chart by default, using the index as the x-values and the list elements as the y-values*



COMPATIBLE DATA TYPES

Matplotlib Basics

Object-Oriented Plotting

Chart Formatting

Chart Types

Matplotlib can plot many **data types**, including base Python sequences, NumPy Arrays, and Pandas Series & DataFrames

```
import matplotlib.pyplot as plt
import pandas as pd
```

```
y
```

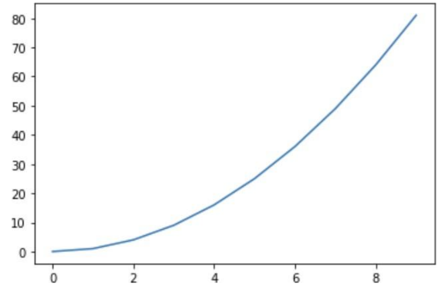
```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

```
df.head(3)
```

	x	y	z
0	0	0	0
1	1	1	1
2	2	4	8

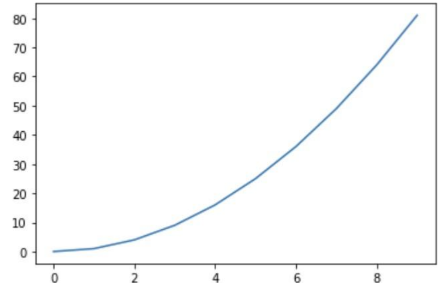
Python List

```
plt.plot(y)
```



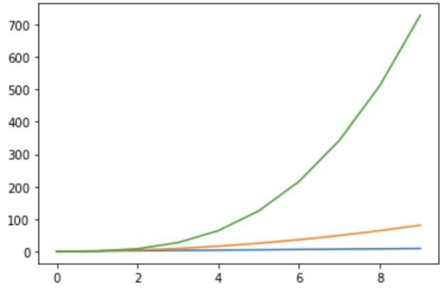
Pandas Series

```
plt.plot(pd.Series(y))
```



Pandas DataFrame

```
plt.plot(df)
```



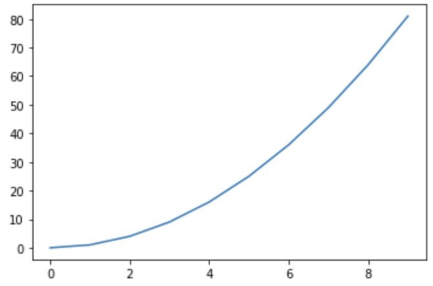
PLOTTING METHODS

Matplotlib has two **plotting methods**, or interfaces:

PyPlot API

Charts are created with the plot() function, and modified with additional functions

```
import matplotlib.pyplot as plt

plt.plot(y)
```

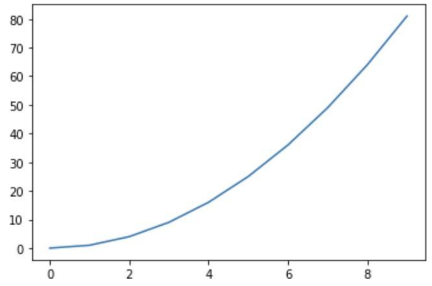


Object-Oriented

Charts are created by defining a plot object, and modified using figure & axis methods

```
import matplotlib.pyplot as plt

fig = plt.figure()

ax = fig.add_subplot()

ax.plot(y)
```

1. *Create the figure object and assign it to the 'fig' variable*
2. *Add a chart, or axis, object to the figure and assign it to the 'ax' variable*
3. *Call the axis plot() method to draw the chart*



We'll mostly focus on the **Object-Oriented** approach, as it provides more clear control over customization

OBJECT-ORIENTED PLOTTING

Object-Oriented plots are built by adding *axes*, or charts, to a *figure*

- The **subplots()** function lets you create the figure and axes in a single line of code
- You can then use figure & axis methods to customize the different elements in the plot

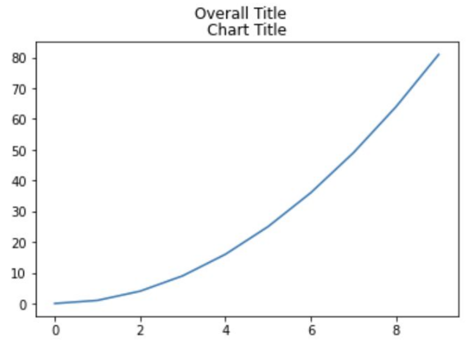
```
fig = plt.figure()

ax = fig.add_subplot()
```

```
fig, ax = plt.subplots()

ax.plot(y)

fig.suptitle("Overall Title")
ax.set_title("Chart Title")
```

Creates the figure and axis

Plots "y"

Adds a title to the figure and axis



We'll start by adding a **single subplot** to each figure for now, but will dive deeper into subplots later in the course!

PLOTTING DATAFRAMES

Matplotlib Basics

Object-Oriented Plotting

Chart Formatting

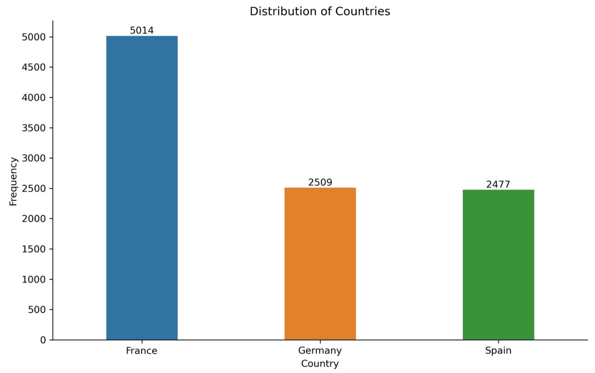
Chart Types

When **plotting Data Frames** using the Object-Oriented interface, Matplotlib will use the index as the x-axis and plot each column as a separate series by default

```python
fig, ax = plt.subplots(figsize = (10, 6), dpi = 100)
sns.barplot(x = df['country'].value_counts().index, y = df['country'].value_counts(), width=.4 )
ax.set_title('Distribution of Countries')
ax.set_ylabel('Frequency')
ax.set_xlabel('Country')
ax.set_yticks(np.arange(0, 5500, 500))
ax.bar_label(ax.containers[0], fontsize=10)
ax.spines[['right', 'top']].set_visible(False)
fig.savefig('distribution_of_countries.png', dpi = 300)
plt.show()
```

```python
df['country'].value_counts()
```

```
country
France     5014
Germany    2509
Spain      2477
Name: count, dtype: int64
```



Exercise

Matplotlib Basics

Object-Oriented Plotting

Chart Formatting

Chart Types

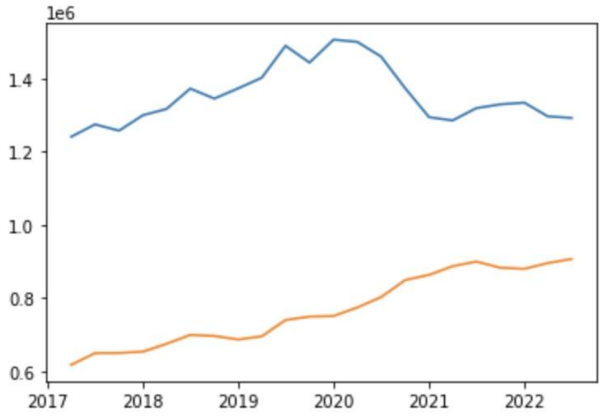
Plotting each series independently allows for improved customization

- **ax.plot(*x-axis series*, *y-series values*)**

```
ca_housing.head()
```

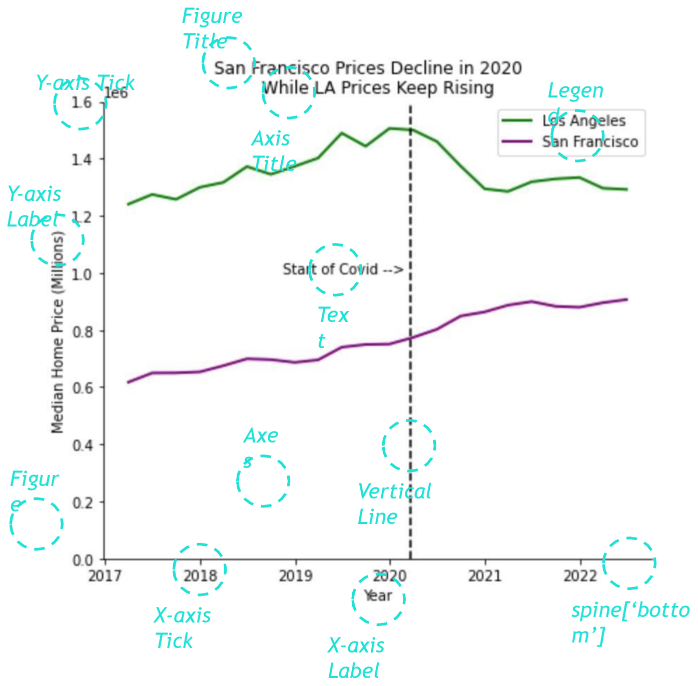
region_name	Los Angeles	San Francisco
period_begin		
2017-03-31	617710.0	1241075.0
2017-06-30	649635.0	1274846.0
2017-09-30	650077.0	1257692.0
2017-12-31	653588.0	1300038.0
2018-03-31	675053.0	1316952.0

```
fig, ax = plt.subplots()

ax.plot(ca_housing.index, ca_housing["San Francisco"])
ax.plot(ca_housing.index, ca_housing["Los Angeles"])
```



FORMATTING OPTIONS

Matplotlib has these **formatting options** for PyPlot and Object-Oriented plots:



Option	Object-Oriented	PyPlot API
Figure Title	fig.suptitle()	plt.suptitle()
Chart Title	ax.set_title()	plt.subtitle()
X-Axis Label	ax.set_xlabel()	plt.xlabel()
Y-Axis Label	ax.set_ylabel()	plt.ylabel()
Legend	ax.legend()	plt.legend()
X-Axis Limit	ax.set_xlim()	plt.xlim()
Y-Axis Limit	ax.set_ylim()	plt.ylim()
X-Axis Ticks	ax.set_xticks()	plt.xticks()
Y-Axis Ticks	ax.set_yticks()	plt.yticks()
Vertical Line	ax.axvline()	plt.axvline()
Horizontal Line	ax.axhline()	plt.axhline()
Text	ax.text()	plt.text()
Spines (borders)	ax.spines['side']	plt.spines['side']

FIGURE SIZE

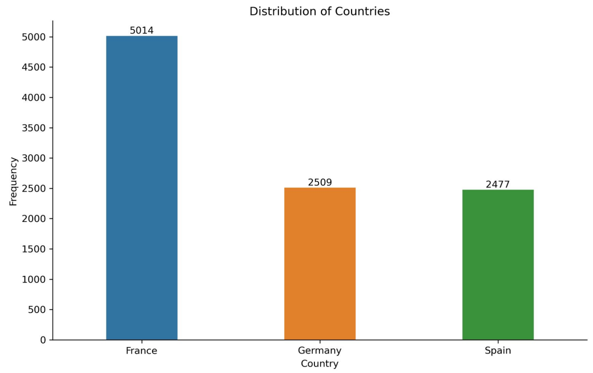
You can adjust the **figure size** with the “figsize” argument

- **figsize=(*width*, *height*)** – the default is 6.4 x 4.8 inches

```
fig, ax = plt.subplots(figsize = (10, 6), dpi = 100)
sns.barplot(x = df['country'].value_counts().index, y = df['country'].value_counts(), width=.4 )
ax.set_title('Distribution of Countries')
ax.set_ylabel('Frequency')
ax.set_xlabel('Country')
ax.set_yticks(np.arange(0, 5500, 500))
ax.bar_label(ax.containers[0], fontsize=10)
ax.spines[['right', 'top']].set_visible(False)
fig.savefig('distribution_of_countries.png', dpi = 300)
plt.show()
```

```
df['country'].value_counts()
```

```
country
France     5014
Germany    2509
Spain      2477
Name: count, dtype: int64
```



TIP: Increasing figure size lets you add whitespace to your visual, which can reduce clutter and add space to crowded axes

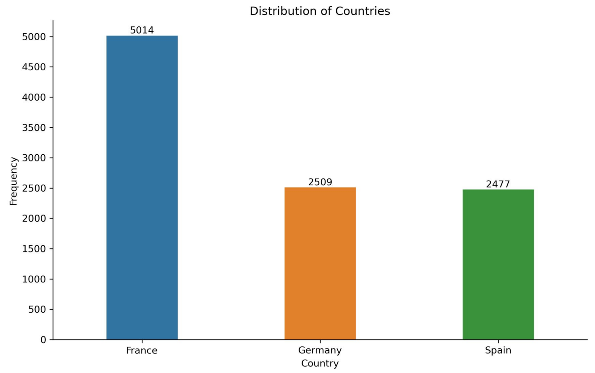
CHART TITLES

The set_title() and set_label() methods let you add **chart titles** and axis labels

```
fig, ax = plt.subplots(figsize = (10, 6), dpi = 100)
sns.barplot(x = df['country'].value_counts().index, y = df['country'].value_counts(), width=.4 )
ax.set_title('Distribution of Countries')
ax.set_ylabel('Frequency')
ax.set_xlabel('Country')
ax.set_yticks(np.arange(0, 5500, 500))
ax.bar_label(ax.containers[0], fontsize=10)
ax.spines[['right', 'top']].set_visible(False)
fig.savefig('distribution_of_countries.png', dpi = 300)
plt.show()
```

```
df['country'].value_counts()
```

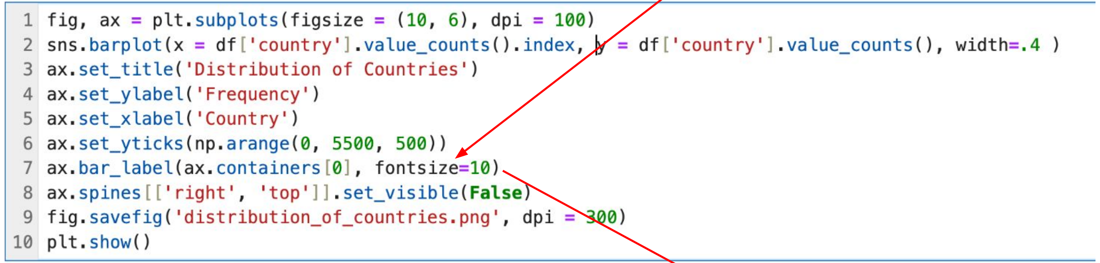
```
country
France     5014
Germany    2509
Spain      2477
Name: count, dtype: int64
```



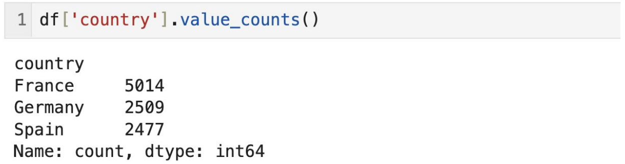
FONT SIZES

You can modify chart **font sizes** with the "fontsize" argument

- You can specify the size in points *(10, 12, etc.)* or relative size *("smaller", "x-large", etc.)*

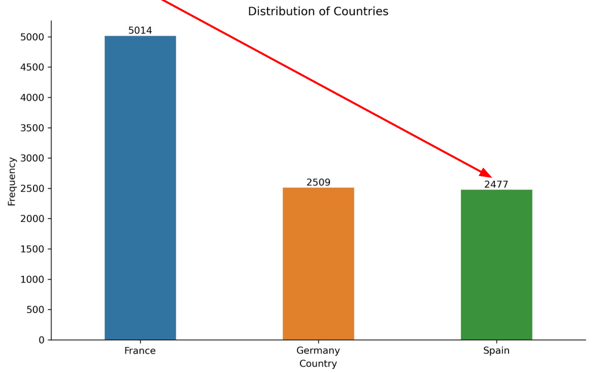


```
fig, ax = plt.subplots(figsize = (10, 6), dpi = 100)
sns.barplot(x = df['country'].value_counts().index, y = df['country'].value_counts(), width=.4 )
ax.set_title('Distribution of Countries')
ax.set_ylabel('Frequency')
ax.set_xlabel('Country')
ax.set_yticks(np.arange(0, 5500, 500))
ax.bar_label(ax.containers[0], fontsize=10)
ax.spines[['right', 'top']].set_visible(False)
fig.savefig('distribution_of_countries.png', dpi = 300)
plt.show()
```



```
df['country'].value_counts()
```

```
country
France     5014
Germany    2509
Spain      2477
Name: count, dtype: int64
```



CUSTOM X-TICKS

Matplotlib Basics

Object-Oriented Plotting

Chart Formatting

Chart Types

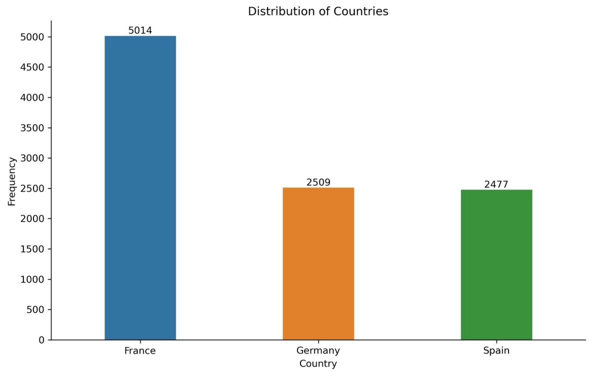
You can apply **custom x-ticks** with the set_xticks() and xticks() functions

- **ax.set_xticks(*iterable*)**

```
fig, ax = plt.subplots(figsize = (10, 6), dpi = 100)
sns.barplot(x = df['country'].value_counts().index, y = df['country'].value_counts(), width=.4 )
ax.set_title('Distribution of Countries')
ax.set_ylabel('Frequency')
ax.set_xlabel('Country')
ax.set_yticks(np.arange(0, 5500, 500))
ax.bar_label(ax.containers[0], fontsize=10)
ax.spines[['right', 'top']].set_visible(False)
fig.savefig('distribution_of_countries.png', dpi = 300)
plt.show()
```

```
df['country'].value_counts()
```

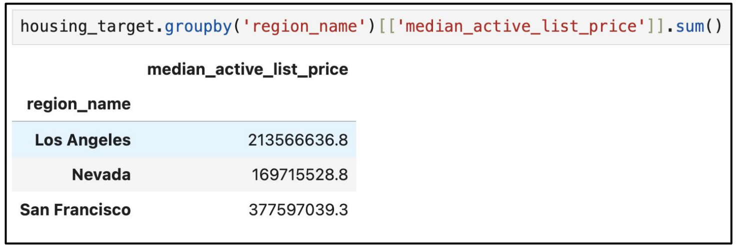
```
country
France     5014
Germany    2509
Spain      2477
Name: count, dtype: int64
```



Bar CHARTS

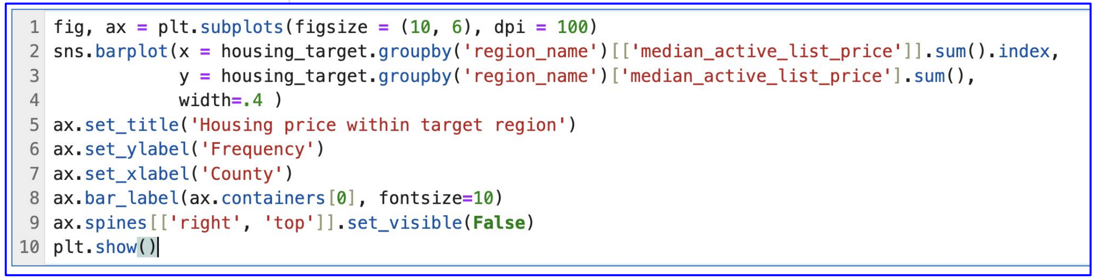
- Matplotlib Basics
- Object-Oriented Plotting
- Chart Formatting
- **Chart Types**

EXAMPLE *Sum price housing by Unit region_name* **Los Angeles, Nevada, San Francisco.**

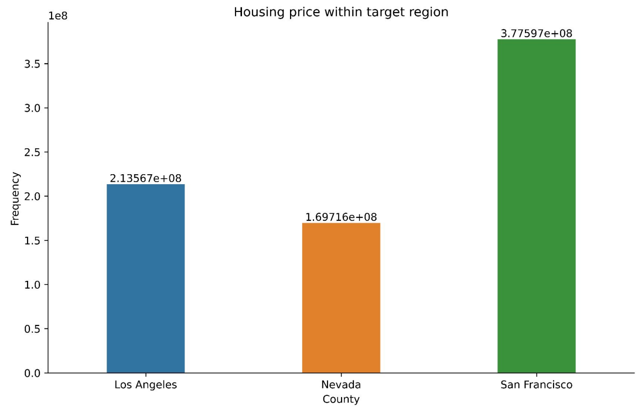


```
housing_target.groupby('region_name')[['median_active_list_price']].sum()
```

region_name	median_active_list_price
Los Angeles	213566636.8
Nevada	169715528.8
San Francisco	377597039.3



```
fig, ax = plt.subplots(figsize = (10, 6), dpi = 100)
sns.barplot(x = housing_target.groupby('region_name')[['median_active_list_price']].sum().index,
            y = housing_target.groupby('region_name')['median_active_list_price'].sum(),
            width=.4 )
ax.set_title('Housing price within target region')
ax.set_ylabel('Frequency')
ax.set_xlabel('County')
ax.bar_label(ax.containers[0], fontsize=10)
ax.spines[['right', 'top']].set_visible(False)
plt.show()
```



PIE CHARTS

Pie charts are used to compare proportions totaling 100%

- **ax.pie(*series values*, *labels=* , *startangle=* , *autopct=*, *pctdistance=*, *explode=*)**

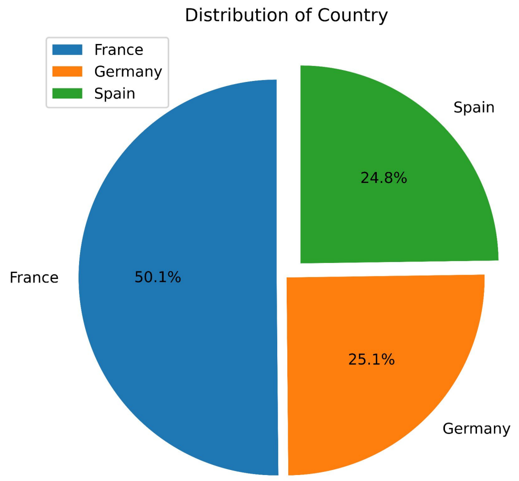
Values in a single column

Labels as the index

```
df['country'].value_counts()
```

```
country
France     5014
Germany    2509
Spain      2477
Name: count, dtype: int64
```

```
fig, ax = plt.subplots( figsize = (6, 6), dpi = 100)
ax.pie(x = df['country'].value_counts(),
       labels = df['country'].value_counts().index,
      autopct='%1.1f%%',
      explode= [0.05, 0, 0.1],
      startangle=90)
ax.set_title('Distribution of Country')
ax.legend(df['country'].value_counts().index, loc='upper left')

plt.show()
```



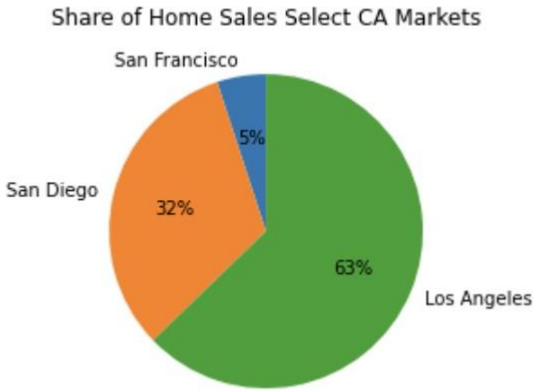
PIE CHARTS

- Matplotlib Basics
- Object Oriented Plotting
- Chart Formatting
- Chart Types

Exercise *Homes Sold by City*

region_name	total_homes_sold
Los Angeles	1580414.0
San Diego	809853.0
San Francisco	126990.0

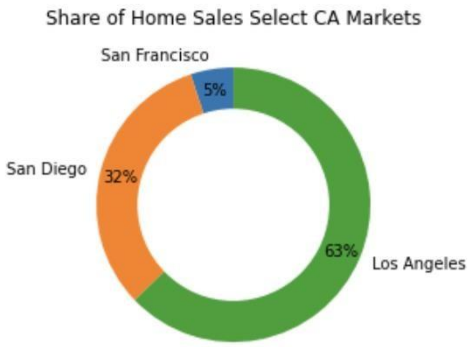
```
fig, ax = plt.subplots()

ax.pie(
    x=sales_totals["total_homes_sold"],
    startangle=90,
    labels=sales_totals.index,
    autopct="%.0f%%"
)

ax.set_title("Share of Home Sales Select CA Markets")
```



DONUT CHARTS

You can create a **donut chart** by adding a "hole" to a pie chart and shifting the labels

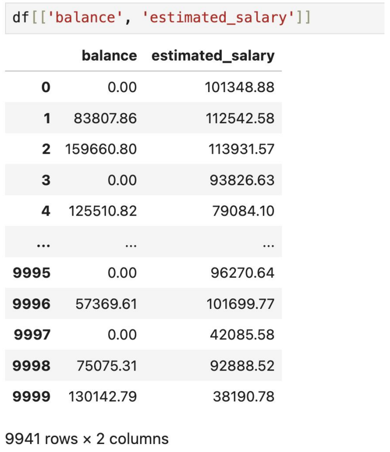
```
fig, ax = plt.subplots()

ax.pie(
    x=sales_totals["total_homes_sold"],
    startangle=90,
    labels=sales_totals.index,
    autopct="%.0f%%",
    pctdistance=.85)


donut_hole = plt.Circle((0, 0), 0.70, fc='white')
fig = plt.gcf()

fig.gca().add_artist(donut_hole)

ax.set_title("Share of Home Sales Select CA Markets")
```



Count plot using seaborn

Countplot show the counts of observations in each categorical bin using bars

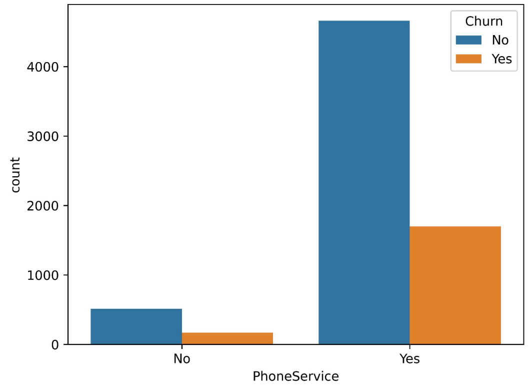
- **seaborn.countplot**(*, x=None, y=None, hue=None, data=None,**)**



```
df[['balance', 'estimated_salary']]
```

	balance	estimated_salary
0	0.00	101348.88
1	83807.86	112542.58
2	159660.80	113931.57
3	0.00	93826.63
4	125510.82	79084.10
...	...	...
9995	0.00	96270.64
9996	57369.61	101699.77
9997	0.00	42085.58
9998	75075.31	92888.52
9999	130142.79	38190.78

9941 rows × 2 columns



```
sns.countplot(x =df['PhoneService'], hue=df['Churn'])
plt.show()
```

HISTOGRAMS

Histograms are used to visualize the distribution of a numeric variable

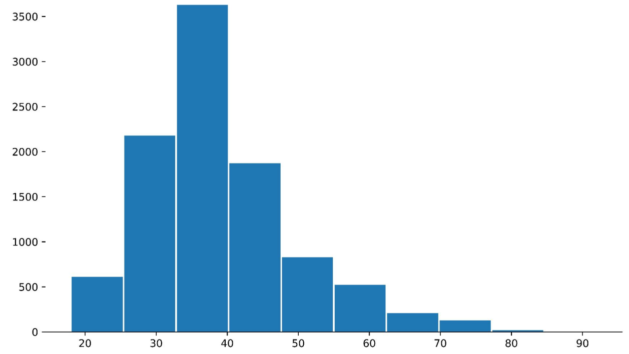
- **ax.hist(*series*, *density=* , *alpha=*, *bins=*)**

  numerical series

```
df['age']
```

```
0       42
1       41
2       42
3       39
4       43
        ..
9995    39
9996    35
```

```
fig, ax = plt.subplots(figsize = (10, 6))
ax.hist(x = df['age'], rwidth = 0.97)
ax.spines['right'].set_visible(False)
ax.spines['left'].set_visible(False)
ax.spines['top'].set_visible(False)
plt.savefig('hist.png', dpi = 1000)
plt.show()
```



SCATTERPLOTS

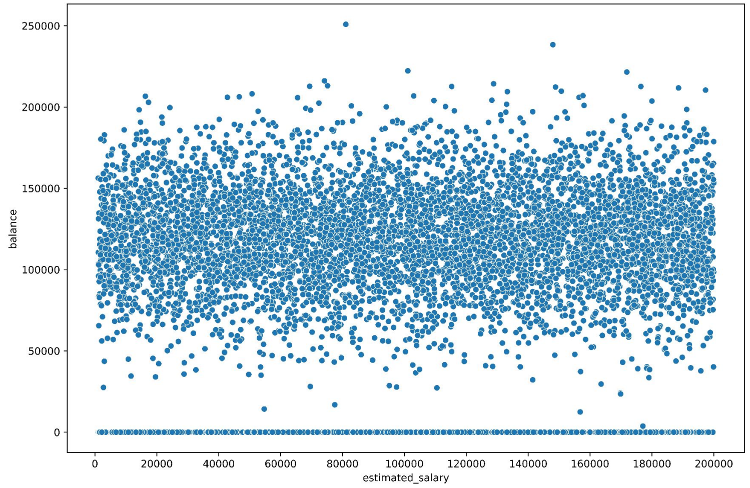
Scatterplots are used to visualize the relationship between numerical variables

- **ax.scatter(*x-axis series*, *y-axis series*, *size=* , *alpha=*)**

```
df[['balance', 'estimated_salary']]
```

	balance	estimated_salary
0	0.00	101348.88
1	83807.86	112542.58
2	159660.80	113931.57
3	0.00	93826.63
4	125510.82	79084.10
...	...	...
9995	0.00	96270.64
9996	57369.61	101699.77
9997	0.00	42085.58
9998	75075.31	92888.52
9999	130142.79	38190.78

9941 rows × 2 columns



ADVANCED CUSTOMIZATION

ADVANCED CUSTOMIZATION

In this section we'll cover **advanced customization** techniques in Matplotlib, including multi-chart figures, custom layouts & colors, style sheets, and more

TOPICS WE'LL COVER:

Subplots	GridSpec Layouts
Colors	Style Sheets
rcParams	Saving Figures

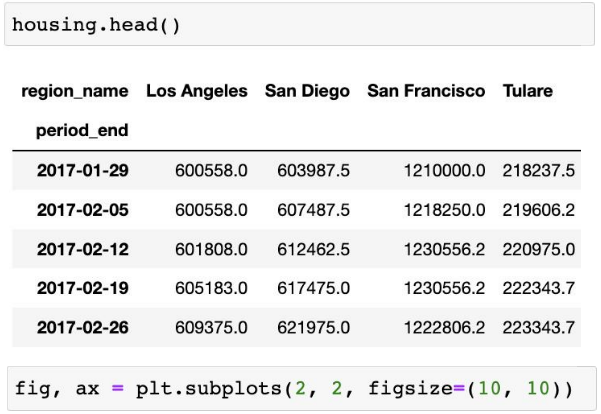
GOALS FOR THIS SECTION:

- Understand how to build multi-chart figures both with subplots and GridSpec layouts
- Learn how to customize chart colors, by leveraging custom colormaps and creating your own!
- Take a look at pre-built stylesheets, and dive into the settings behind them that allow for extreme chart customization

SUBPLOTS

Subplots let you create a grid of equally sized charts in a single figure

- **fig, ax = plt.subplots(rows, columns)** – this creates a grid with the specified rows & columns

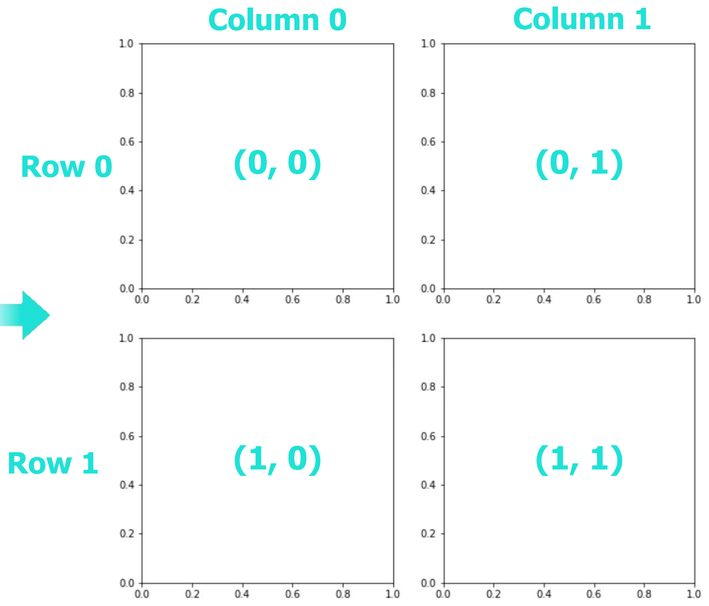


```
housing.head()
```

region_name	Los Angeles	San Diego	San Francisco	Tulare
period_end				
2017-01-29	600558.0	603987.5	1210000.0	218237.5
2017-02-05	600558.0	607487.5	1218250.0	219606.2
2017-02-12	601808.0	612462.5	1230556.2	220975.0
2017-02-19	605183.0	617475.0	1230556.2	222343.7
2017-02-26	609375.0	621975.0	1222806.2	223343.7

```
fig, ax = plt.subplots(2, 2, figsize=(10, 10))
```

This creates a 2 row, 2 column grid that can be populated with individual charts



SUBPLOTS

- **Subplots**
- GridSpec Layouts
- Colors
- Style Sheets
- rcParams
- Saving Figures

Subplots let you create a grid of equally sized charts in a single figure

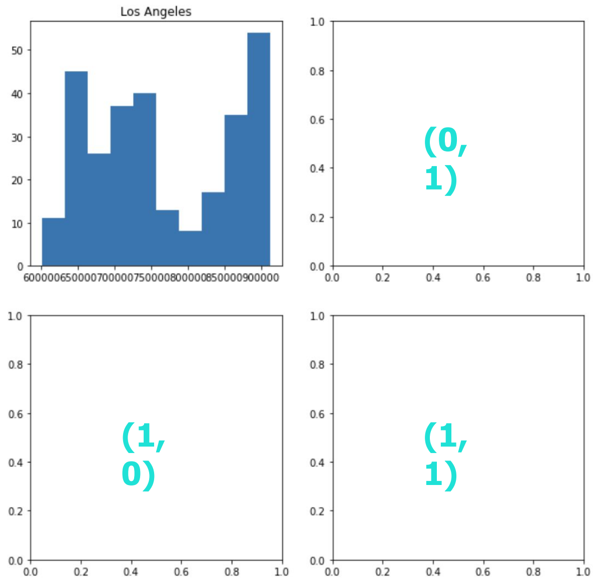
- **fig, ax = plt.subplots(rows, columns)** – this creates a grid with the specified rows & columns

```
housing.head()
```

region_name	Los Angeles	San Diego	San Francisco	Tulare
period_end				
2017-01-29	600558.0	603987.5	1210000.0	218237.5
2017-02-05	600558.0	607487.5	1218250.0	219606.2
2017-02-12	601808.0	612462.5	1230556.2	220975.0
2017-02-19	605183.0	617475.0	1230556.2	222343.7
2017-02-26	609375.0	621975.0	1222806.2	223343.7

```
fig, ax = plt.subplots(2, 2, figsize=(10, 10))

ax[0][0].hist(housing["Los Angeles"])
ax[0][0].set_title("Los Angeles")
```

Specify ax[row][column] to create and modify individual subplots

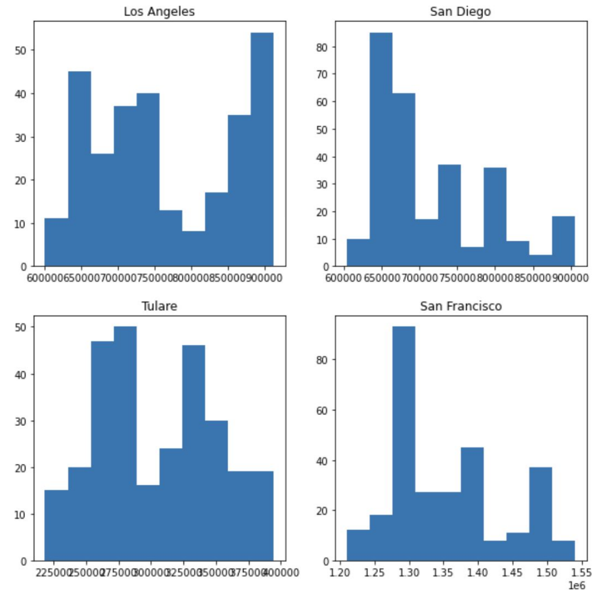


SUBPLOTS

Subplots let you create a grid of equally sized charts in a single figure

- **fig, ax = plt.subplots(rows, columns)** – this creates a grid with the specified rows & columns

```
fig, ax = plt.subplots(2, 2, figsize=(10, 10))

ax[0][0].hist(housing["Los Angeles"])
ax[0][0].set_title("Los Angeles")

ax[0][1].hist(housing["San Diego"])
ax[0][1].set_title("San Diego")

ax[1][0].hist(housing["Tulare"])
ax[1][0].set_title("Tulare")

ax[1][1].hist(housing["San Francisco"])
ax[1][1].set_title("San Francisco")
```



SUBPLOTS

Subplots

GridSpec Layouts

Colors

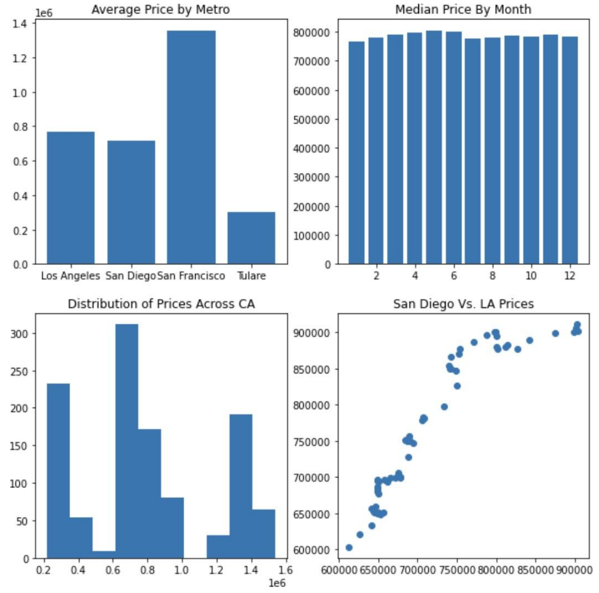
Style Sheets

rcParams

Saving Figures

Subplots can be **any chart type**, and do not have to be the same type

```
fig, ax = plt.subplots(2, 2, figsize=(10, 10))

ax[0][0].bar(
    price_by_region.index,
    price_by_region["median_active_list_price"]
)
ax[0][0].set_title("Average Price by Metro")

ax[0][1].bar(
    price_by_month.index,
    price_by_month["median_active_list_price"]
)
ax[0][1].set_title("Median Price By Month")

ax[1][0].hist(ca_housing["median_active_list_price"])
ax[1][0].set_title("Distribution of Prices Across CA")


ax[1][1].scatter(
    price_by_r_m.loc["San Diego", "median_active_list_price"],
    price_by_r_m.loc["Los Angeles", "median_active_list_price"]
)
ax[1][1].set_title("San Diego Vs. LA Prices")
```

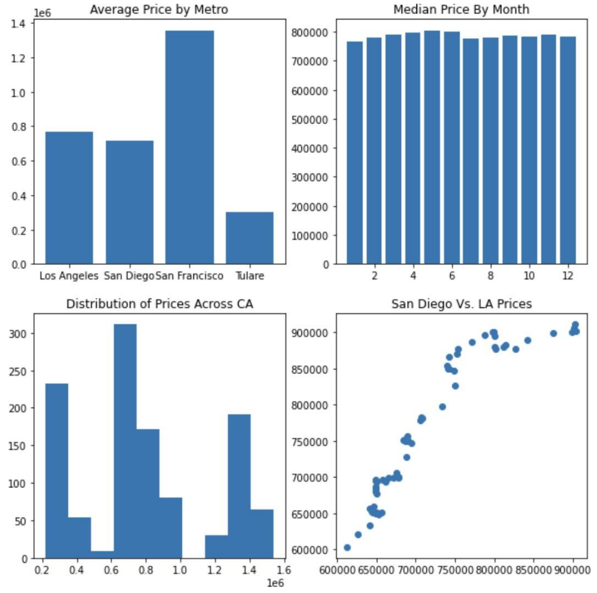


COUNT Plot with Seaborn

- Subplots
- GridSpec Layouts
- Colors
- Style Sheets
- rcParams
- Saving Figures

Subplots can be **any chart type**, and do not have to be the same type

```python
fig, ax = plt.subplots(2, 2, figsize=(10, 10))

ax[0][0].bar(
    price_by_region.index,
    price_by_region["median_active_list_price"]
)
ax[0][0].set_title("Average Price by Metro")

ax[0][1].bar(
    price_by_month.index,
    price_by_month["median_active_list_price"]
)
ax[0][1].set_title("Median Price By Month")

ax[1][0].hist(ca_housing["median_active_list_price"])
ax[1][0].set_title("Distribution of Prices Across CA")


ax[1][1].scatter(
    price_by_r_m.loc["San Diego", "median_active_list_price"],
    price_by_r_m.loc["Los Angeles", "median_active_list_price"]
)
ax[1][1].set_title("San Diego Vs. LA Prices")
```


Average Price by Metro
1e6
Los Angeles
San Diego
San Francisco
Tulare
Median Price By Month
Distribution of Prices Across CA
1e6
San Diego Vs. LA Prices