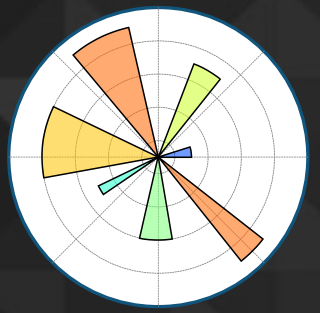
DATA VISUALIZATION WITH MATPLOTLIB & SEABORN



COURSE OUTLINE

1 Intro to Data Visualization

Cover key data visualization best practices for clear communication, with tips for choosing the right chart, formatting it effectively, and using it to tell a story

2 Matplotlib Fundamentals

Introduce the Matplotlib library and use it to build & customize several chart types, including line charts, bar charts, pie charts, scatterplots, and histograms

PROJECT: Visualizing Coffee Industry Data

3 Advanced Customization

Apply advanced customization techniques in Matplotlib, including multi-chart figures, custom layouts & colors, style sheets, and more

PROJECT: Consolidating Coffee Industry Data into a Report

4 Data Viz with Seaborn

Visualize data with Seaborn, another Python library that introduces new chart types and layouts, and interacts will with Matplotlib

PROJECT: Highlighting Insights from the Automotive Auction Industry

WELCOME TO MCG CONSULTING GROUP

THE SITUATION

You've just been hired as an Associate Consultant for **Consulting Group (MCG)**, a multinational firm that provides strategic advice to companies across different industries. Your new role will see you take on projects in the hotel, coffee, automotive, and diamond industries.

THE ASSIGNMENT

Your task is to **effectively visualize data** from these industries to deliver key insights to MCG's clients.

This will range from analyzing hotel customer demographics to understanding the major players in the global coffee industry.

THE OBJECTIVES

- Use Pandas to read & manipulate multiple datasets
- Use Matplotlib to visualize data & communicate insights, and then build reports to consolidate your findings
- Use Seaborn to conduct advanced exploratory analysis and aid the decision-making process

MCG

SETTING EXPECTATIONS

This course covers the **core functionality** for Matplotlib & Seaborn

- *We'll cover chart types, common customization options, and best practices for visualizing and analyzing data*
- *We'll give the tools to use the official documentation to apply any customization option not covered in the course*

We'll focus on creating **static** visuals & dashboards

- *Interactive data visualization with Python will be covered in a separate course*

We'll use **Jupyter Notebooks** as our primary coding environment

- *Jupyter Notebooks are free to use, and the industry standard for conducting data analysis with Python (we'll introduce Google Colab as an alternative, cloud-based environment as well)*

You do **NOT** need to be a Python expert to take this course

- *It is strongly recommended that you complete our Python Foundations and Data Analysis with Pandas courses, or have a solid understanding of basic Python syntax and DataFrame manipulation with the Pandas library*

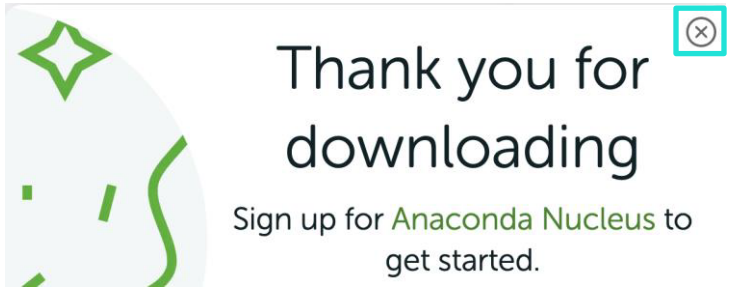
INSTALLATION & SETUP

INSTALLING ANACONDA (MAC)

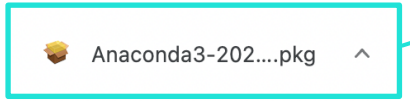
- Installing Anaconda
- Launching Jupyter
- Google Colab

1) Go to **anaconda.com/products/distribution** and click 

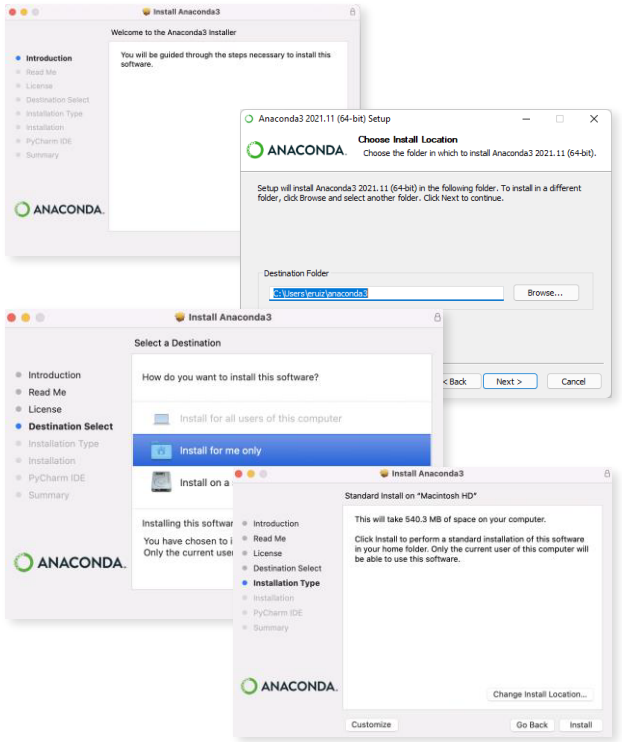
2) Click **X** on the Anaconda Nucleus pop-up
(no need to launch)



3) Launch the downloaded Anaconda **pkg** file



4) Follow the **installation steps**
(default settings are OK)



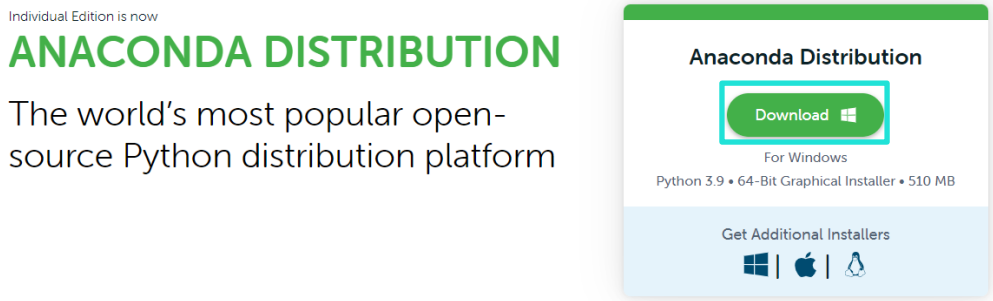
INSTALLING ANACONDA (PC)

Installing Anaconda

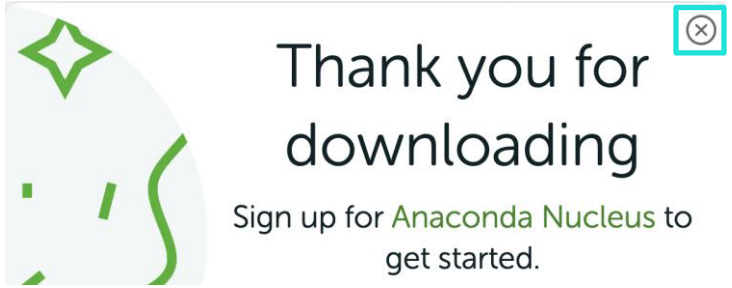
Launching Jupyter

Google Colab

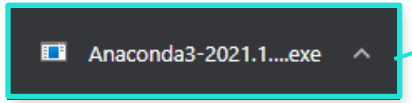
1) Go to **anaconda.com/products/distribution** and click Download



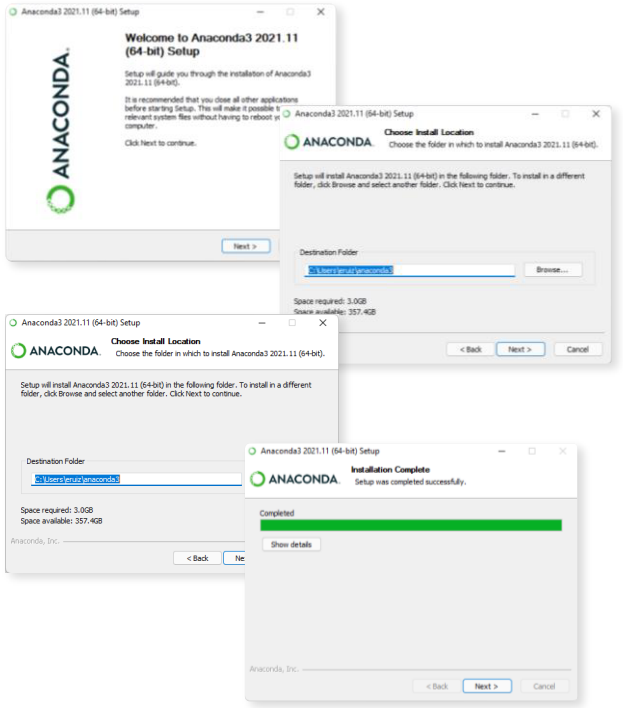
2) Click **X** on the Anaconda Nucleus pop-up
(no need to launch)



3) Launch the downloaded Anaconda **exe** file



4) Follow the **installation steps**
(default settings are OK)



LAUNCHING JUPYTER

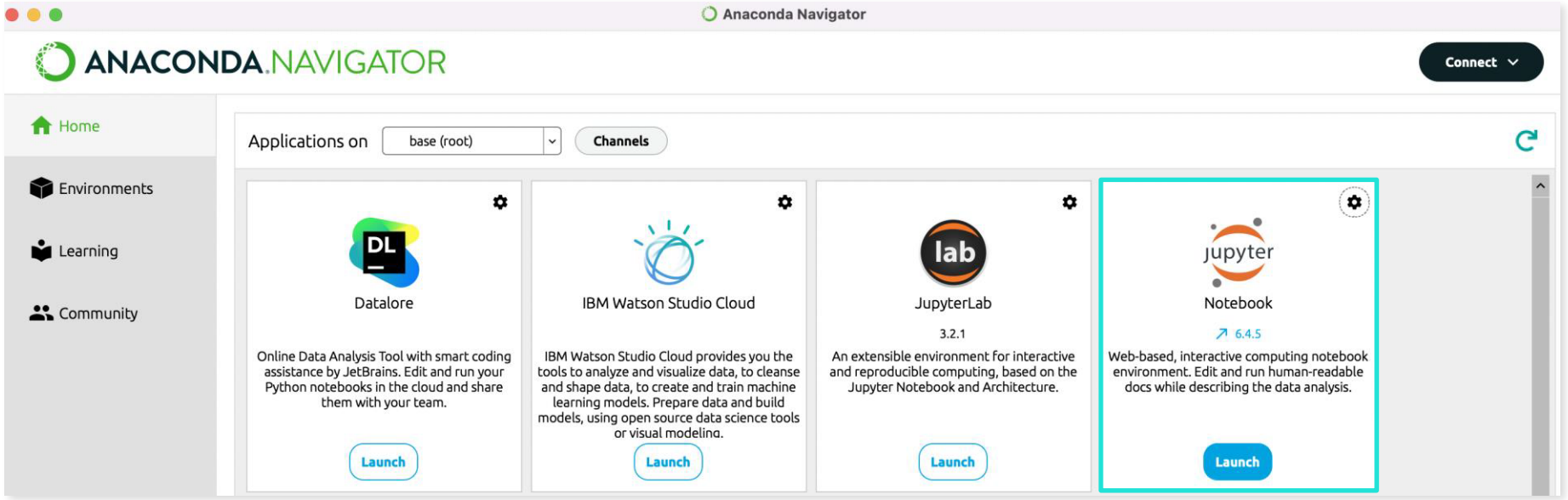
Installing Anaconda

Launching Jupyter

Google Colab

1) Launch **Anaconda Navigator**

2) Find **Jupyter Notebook** and click Launch



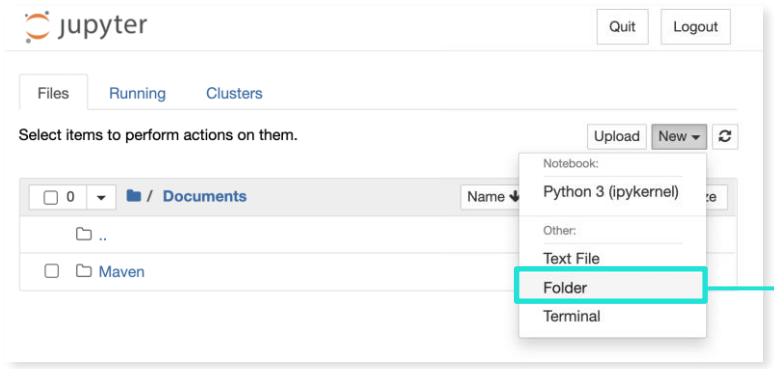
YOUR FIRST JUPYTER NOTEBOOK

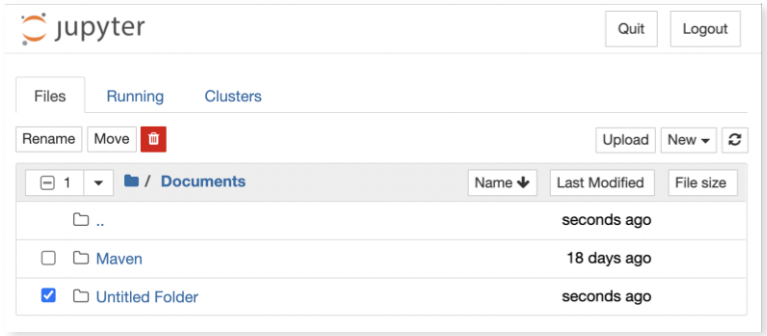
Installing Anaconda

Launching Jupyter

Google Colab

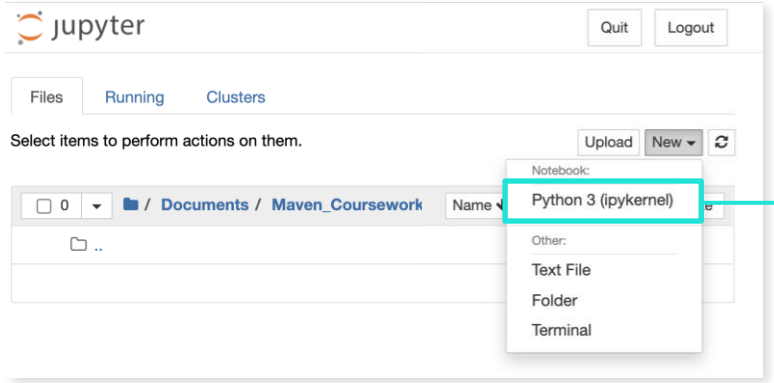
1) Once inside the Jupyter interface, **create a folder** to store your notebooks for the course

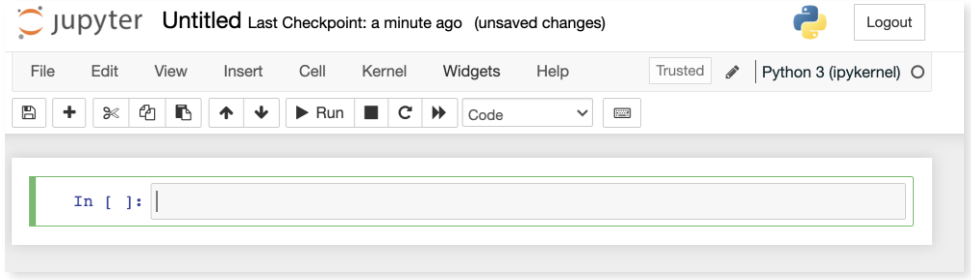




NOTE: *You can rename your folder by clicking "Rename" in the top left corner*

2) Open your new coursework folder and **launch your first Jupyter notebook**!



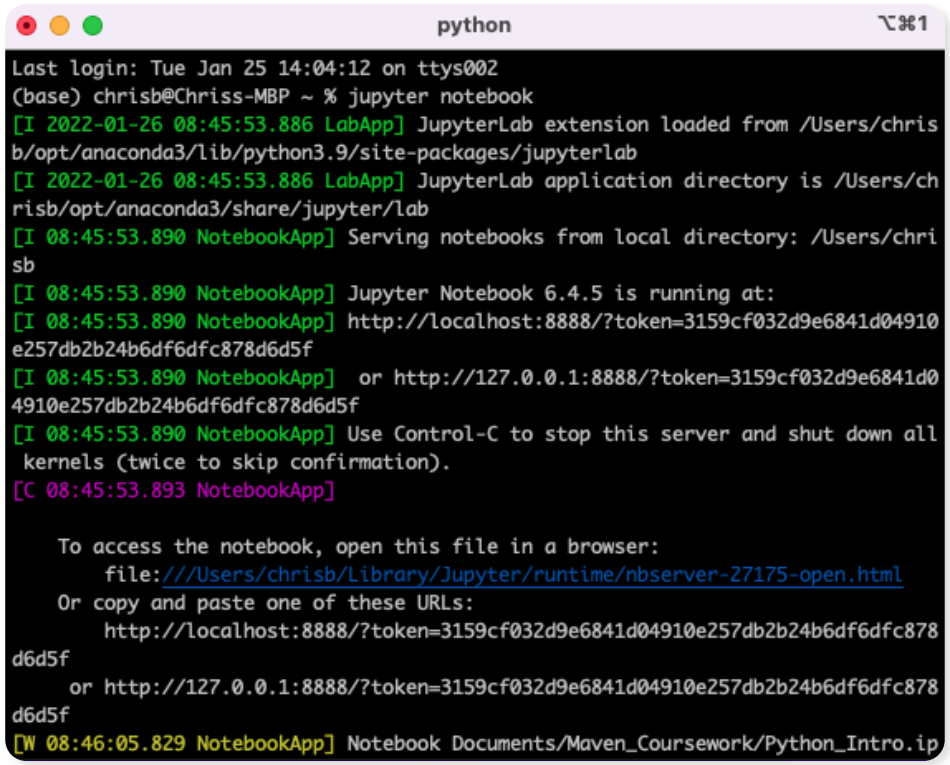


NOTE: *You can rename your notebook by clicking on the title at the top of the screen*

THE NOTEBOOK SERVER

NOTE: When you launch a Jupyter notebook, a terminal window may pop up as well; this is called a **notebook server**, and it powers the notebook interface

- Installing Anaconda
- Launching Jupyter
- Google Colab



If you close the server window, **your notebooks will not run!**

Depending on your OS, and method of launching Jupyter, one may not open. As long as you can run your notebooks, don't worry!

ALTERNATIVE: GOOGLE COLAB

Installing Anaconda

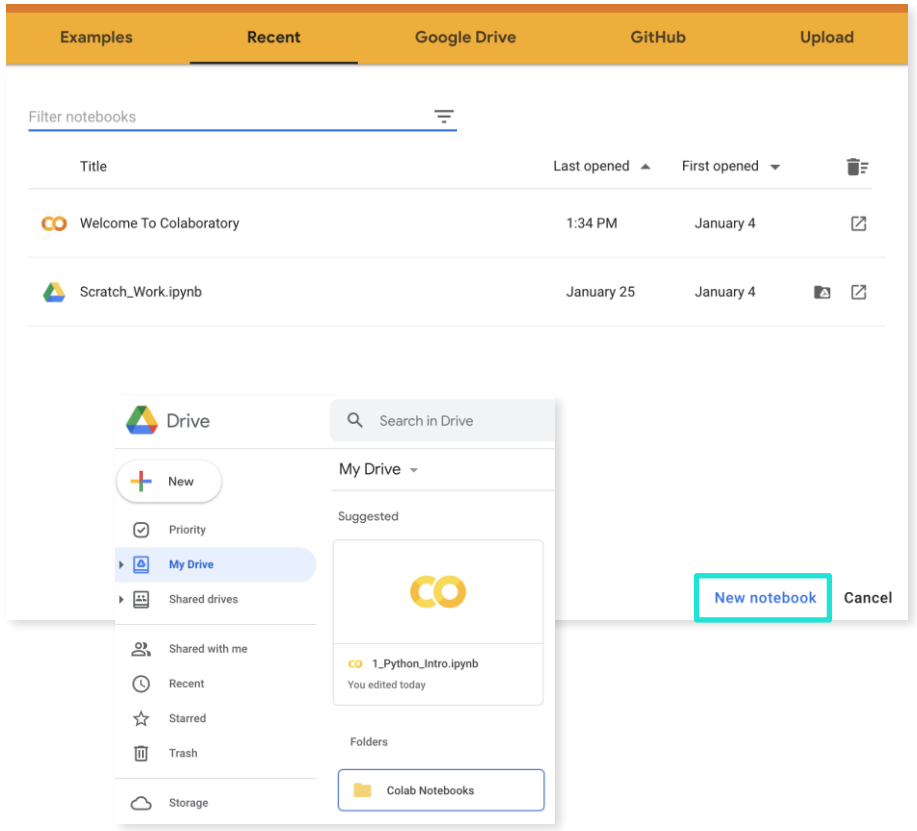
Launching Jupyter

Google Colab

Google Colab is Google's cloud-based version of Jupyter Notebooks

To create a Colab notebook:

1. Log in to a Gmail account
2. Go to **colab.research.google.com**
3. Click "**new notebook**"



Colab is very similar to Jupyter Notebooks *(they even share the same file extension)*; the main difference is that you are connecting to **Google Drive** rather than your machine, so files will be stored in Google's cloud

INTRO TO DATA VISUALIZATION

DATA VISUALIZATION

In this section we'll cover key **data visualization** best practices for clear communication, with tips for choosing the right chart, formatting it effectively, and using it to tell a story

TOPICS WE'LL COVER:

- Data Viz 101
- 3 Key Questions
- Essential Visuals
- Chart Formatting
- Storytelling
- Common Errors

GOALS FOR THIS SECTION:

- Understand the purpose behind visualizing data
- Learn the common chart types and their use cases
- Apply data visualization best practices to create clear and compelling charts
- Address common errors and how to avoid them

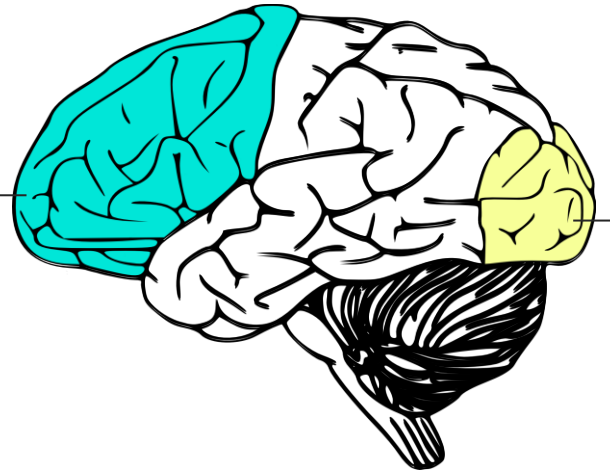
WHY VISUALIZE DATA?

Data visualization allows you to **bring your data to life**

- The human brain is built to interpret raw data as meaningless numbers and noise
- We need **clear patterns** and **visual cues** to help us quickly make sense of complex information

Prefrontal Cortex

- *Located in the frontal lobe*
- *Responsible for cognitive functioning & problem solving*
- *Helps us make sense of non-visual information (like raw data)*
- *Slow & conscious*



Visual Cortex

- *Located in the occipital lobe*
- *Responsible for visual perception & understanding*
- *Helps us make sense of colors, patterns, shapes, sizes, etc.*
- *Instantaneous & subconscious*

Data visualization puts both our prefrontal and visual cortex to work, combining the power of **cognition** *(slow and conscious)* and **perception** *(instantaneous)*

THE TEN SECOND RULE

In **10 seconds**, what can you learn from the data below?

Coffee		Tea		Soda		Juice	
Units Sold	Profit	Units Sold	Profit	Units Sold	Profit	Units Sold	Profit
10	8.04	10	9.14	10	7.46	8	6.58
8	6.95	8	8.14	8	6.77	8	5.76
13	7.58	13	8.74	13	12.74	8	7.71
9	8.81	9	8.77	9	7.11	8	8.84
11	8.33	11	9.26	11	7.81	8	8.47
14	9.96	14	8.1	14	8.84	8	7.04
6	7.24	6	6.13	6	6.08	8	5.25
4	4.26	4	3.1	4	5.39	19	12.5
12	10.84	12	9.13	12	8.15	8	5.56
7	4.81	7	7.26	7	6.42	8	7.91
5	5.68	5	4.74	5	5.73	8	6.89

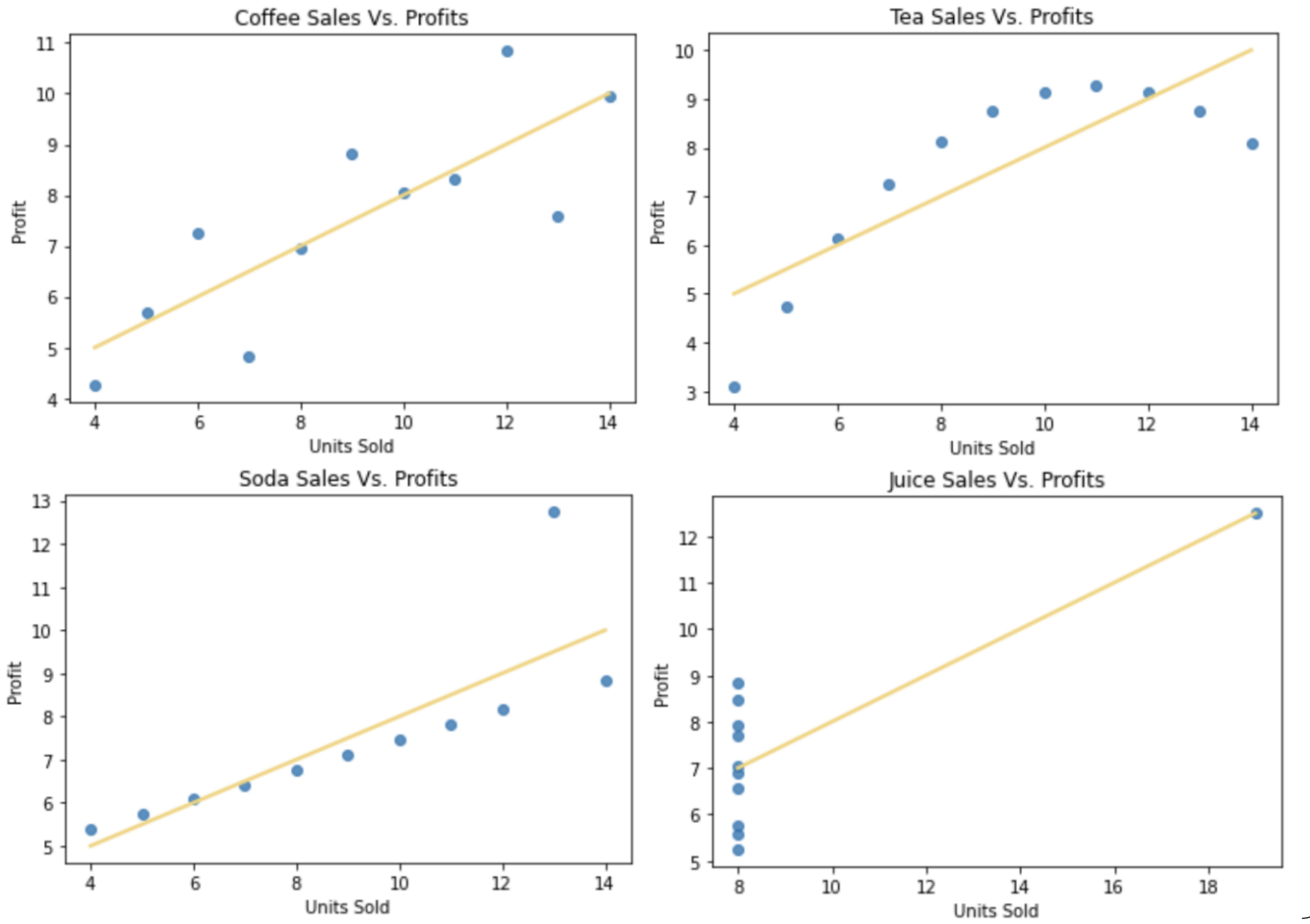
THE TEN SECOND RULE

What if you were given the **averages**?

Coffee		Tea		Soda		Juice	
Units Sold	Profit	Units Sold	Profit	Units Sold	Profit	Units Sold	Profit
10	8.04	10	9.14	10	7.46	8	6.58
8	6.95	8	8.14	8	6.77	8	5.76
13	7.58	13	8.74	13	12.74	8	7.71
9	8.81	9	8.77	9	7.11	8	8.84
11	8.33	11	9.26	11	7.81	8	8.47
14	9.96	14	8.1	14	8.84	8	7.04
6	7.24	6	6.13	6	6.08	8	5.25
4	4.26	4	3.1	4	5.39	19	12.5
12	10.84	12	9.13	12	8.15	8	5.56
7	4.81	7	7.26	7	6.42	8	7.91
5	5.68	5	4.74	5	5.73	8	6.89
9	**7.50**	**9**	**7.50**	**9**	**7.50**	**9**	**7.50**

THE TEN SECOND RULE

What if you **visualize** it?



This is a slight twist on **Anscombe's Quartet**

Despite sharing nearly identical descriptive stats, each series tells a very different visual story

THE 3 KEY QUESTIONS

The **3 key questions** are a great way to help choose the right visual

1 What **type of data** are you working with?

Time-series
Data that spans across continuous time periods

Categorical
Data that can be split up into groups or categories

Numeric
Data with quantitative values, either discrete or continuous

Hierarchical
Data with natural groups and sub-groups

2 What do you want to **communicate**?

Comparison
Compares values over time or across categories

Composition
Breaks down the component parts of a whole

Distribution
Shows the frequency of values within a series

Relationship
Shows the correlation between multiple variables

3 Who is the **end user** and what do they need?

Analyst
Likes to see details and understand what's happening at a granular level

Manager
Wants summarized information with clear, actionable insights

Executive
Needs high-level, clear KPIs to track business health and performance

General Public
Requires engaging visuals and a clear story to follow

ESSENTIAL VISUALS

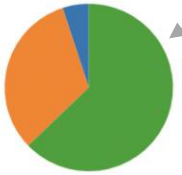
- Data Viz 101
- 3 Key Questions
- Essential Visuals
- Formatting
- Storytelling
- Common Errors

KPI CARD

67%

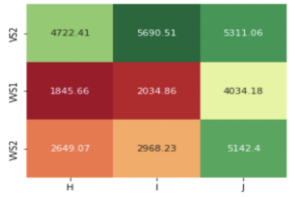
Sometimes simple text works best

PIE CHART



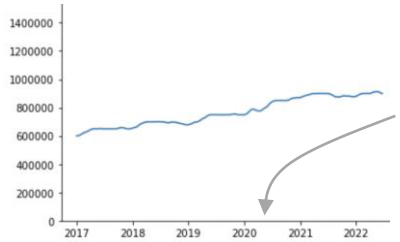
Sort the slices, keep them under ~5, and focus on one

TABLE



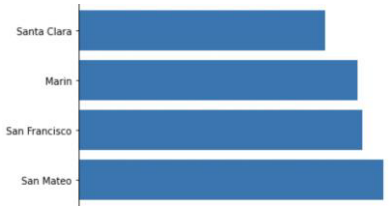
Add a color scale to highlight patterns in the data

LINE CHART



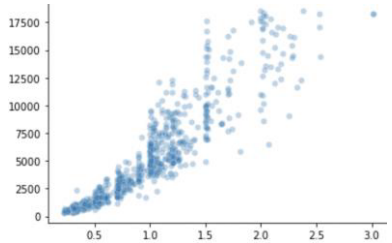
The dates must be continuous

BAR CHART



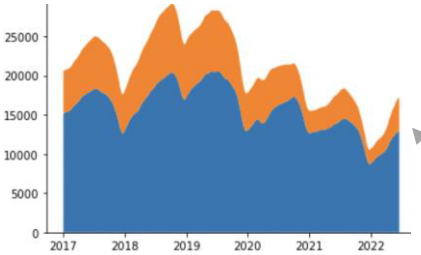
Baseline must start at zero

SCATTER PLOT



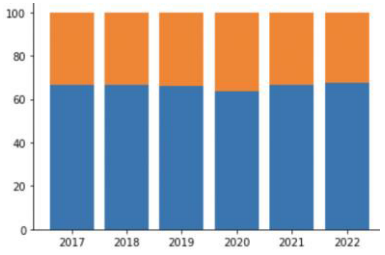
Remember that correlation does not imply causation

AREA CHART

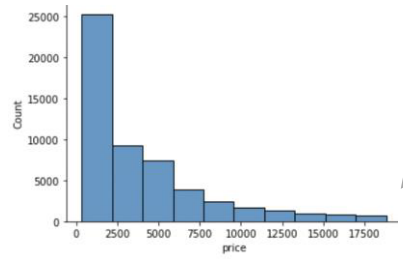


Comparison & composition

%100 STACKED



HISTOGRAM

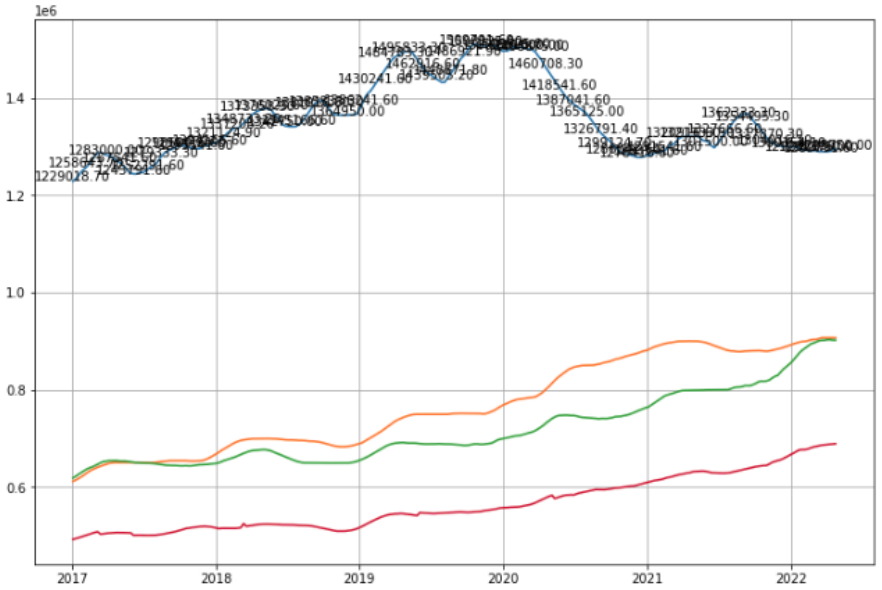


Avoid using too many bins!

CHART FORMATTING

Chart formatting should be used to eliminate noise & facilitate understanding

BEFORE: Cluttered chart



This is the right chart type… so why is it so hard to understand the visual?

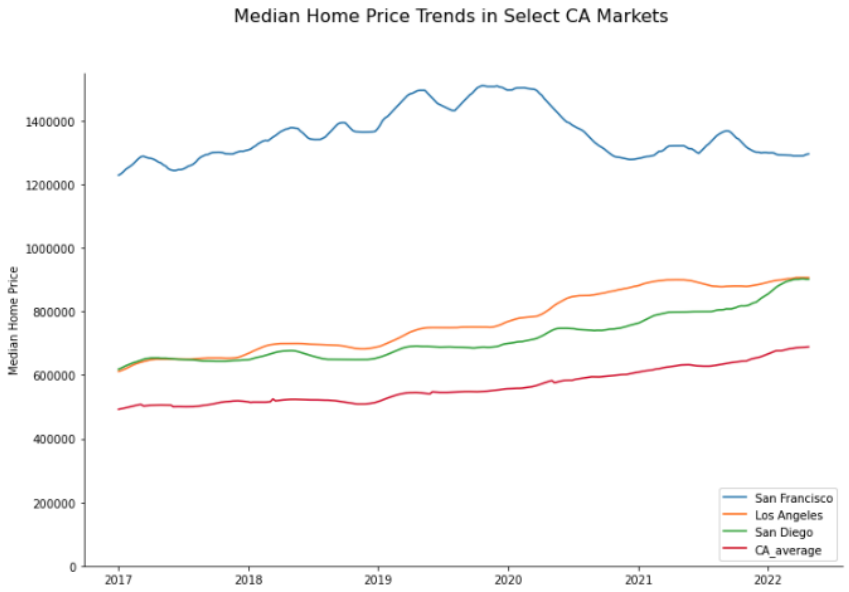
- × The chart border and gridlines are more distracting than useful
- × The vertical axis labels are hard to read and lack context – it's using scientific notation and doesn't start at 0
- × Data labels can help add context, but they just add noise here
- × It's not clear what each line represents

PRO TIP: Be intentional about the formatting you apply – don't just use the default settings!

CHART FORMATTING

Chart formatting should be used to eliminate noise & facilitate understanding

AFTER: Clear chart



PRO TIPS:

- ✓ Remove the chart border & gridlines
- ✓ Format the axis labels clearly
- ✓ Add context with the chart title
- ✓ Create a visual order
- ✓ Make sure the story is clear

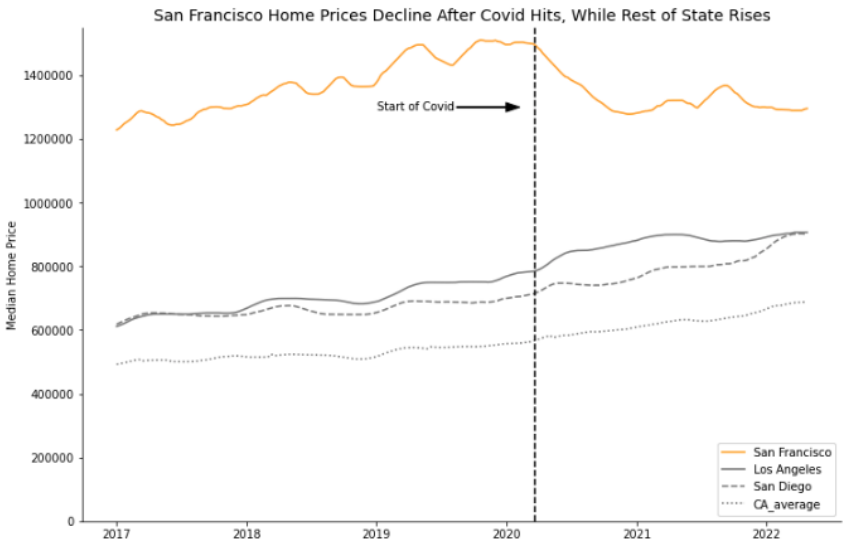
"Perfection is achieved not when there is nothing more to add, ***but when there is nothing left to take away"***

Antoine de Saint-Exupery

STORYTELLING

Descriptive titles and data labels can be used **to tell a clear story** within your visuals

AFTER: Compelling chart



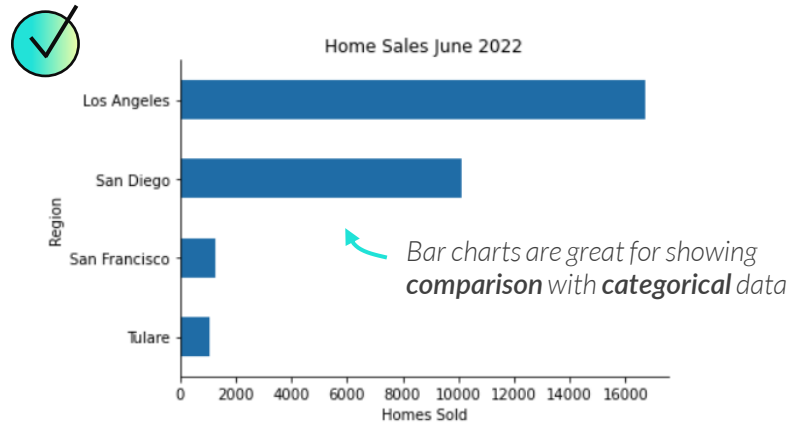
PRO TIPS:

- ✓ Leverage the title to guide the audience toward specific insights
- ✓ Insert text & shapes directly inside the chart
- ✓ Use data labels and annotations to draw attention to the main data points
- ✓ Use color strategically

COMMON ERRORS

1 Choosing the **wrong visual** to represent the type of data

*Using a line chart, which is meant for **time series data**, with categorical data gives the false sense of a trend*



*Bar charts are great for showing **comparison** with **categorical** data*

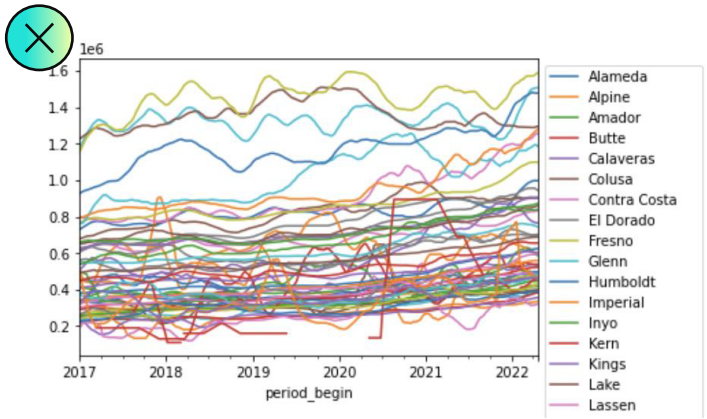
While a tree map can work, comparisons and compositions are harder to make than with a bar or pie chart

*It's best to use them with **hierarchical data***

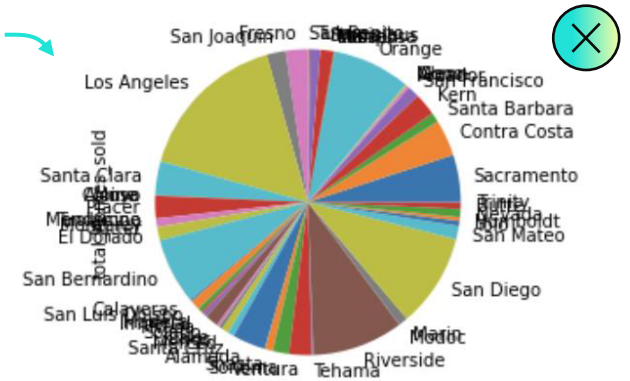
PRO TIP: Don't prioritize variety over effectiveness; use the right chart for the job!

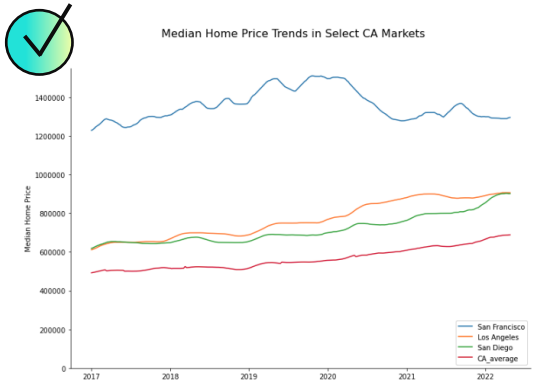
COMMON ERRORS

2 Including **too many series** in a single visual



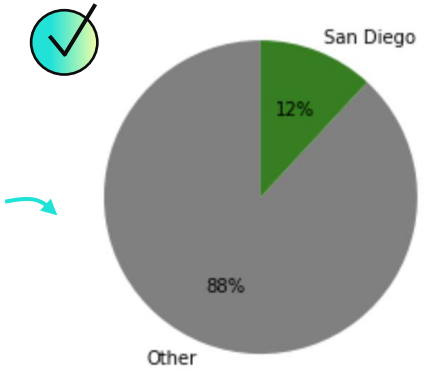
It's hard to focus or extract any valuable information





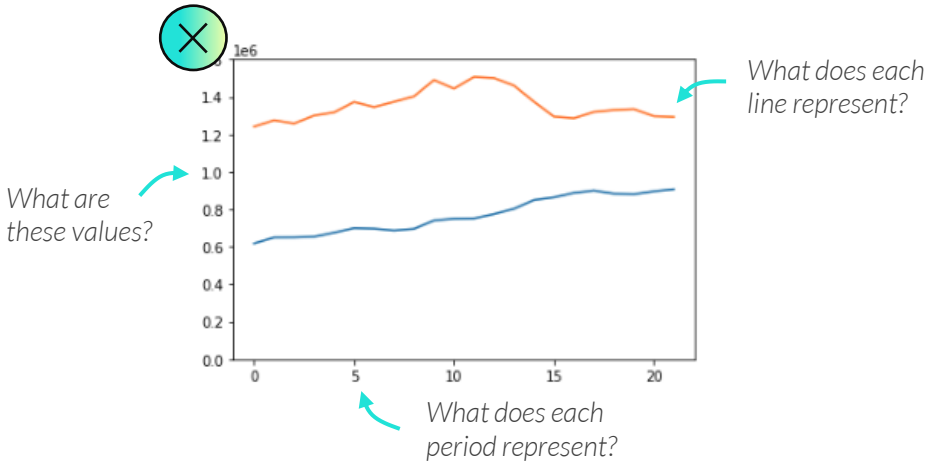
*Try **highlighting** the series you want, or **aggregating** other categories*

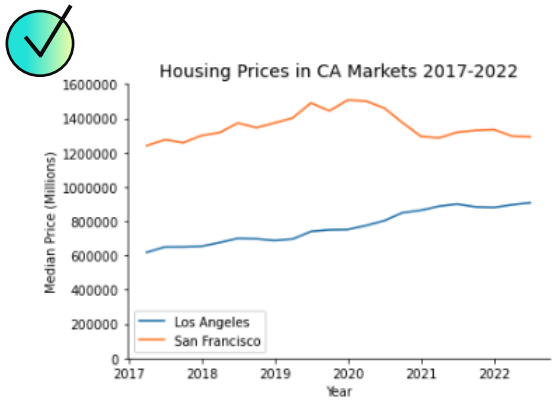
*You can also **group** the other categories into a single series*



COMMON ERRORS

3 Providing **little to no context** with text and labels

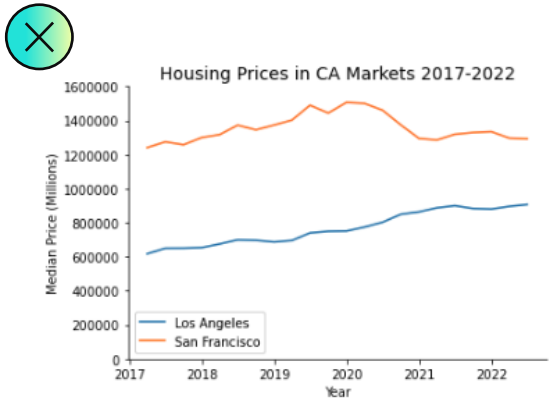


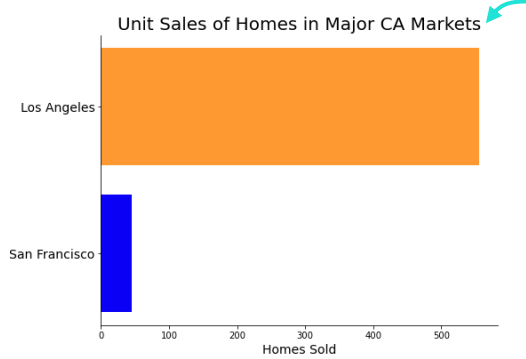


When removing elements from a chart to reduce clutter and noise, remember to **keep all the elements that add understanding**

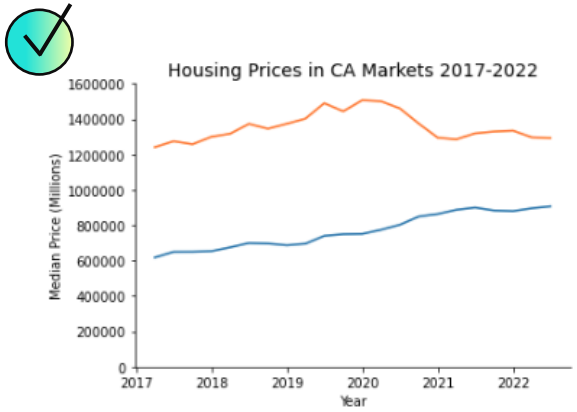
COMMON ERRORS

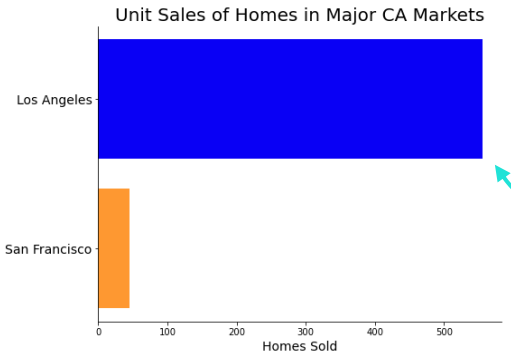
4 Using **inconsistent colors** between related visuals





*Using **different colors for the same series** makes it difficult to associate them visually*





*Using **the same colors consistently** makes them easier to understand, and in some cases allows you to remove the legend*

Consistency gains more importance **as the number of visuals increases**, making it critical for dashboards

KEY TAKEAWAYS

- Always answer the **3 key questions** to choose the right visual
  - *What type of data are you working with? What do you want to communicate? Who is the end user?*
- Do **NOT** prioritize variety over effectiveness
  - *Choose chart types based on how clearly they communicate the data underneath – you can customize later!*
- Eliminate noise and distractions to **facilitate understanding**
  - *"Perfection is achieved not when there is nothing more to add, but when there is nothing left to take away"*
- **Tell a story** with the data to guide the user to the insights
  - *Use titles, strategic labels, and callouts to create a clear narrative*

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 scatterplots and histograms
- Customize your charts by adding custom titles, labels, legends, annotations and much more!

MEET MATPLOTLIB

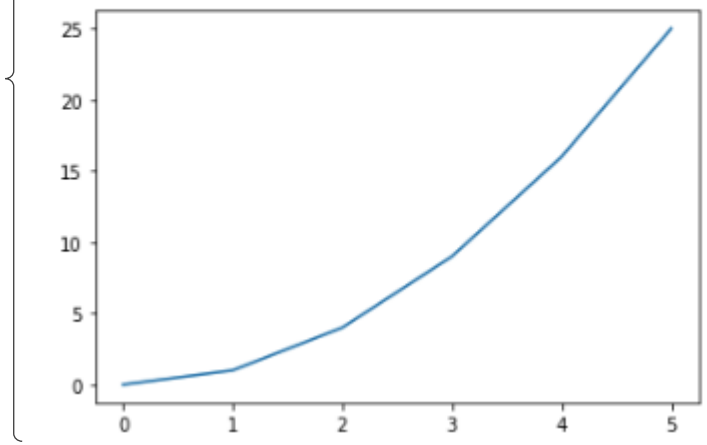
- Matplotlib Basics
- Object-Oriented Plotting
- Chart Formatting
- Chart Types

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 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
```

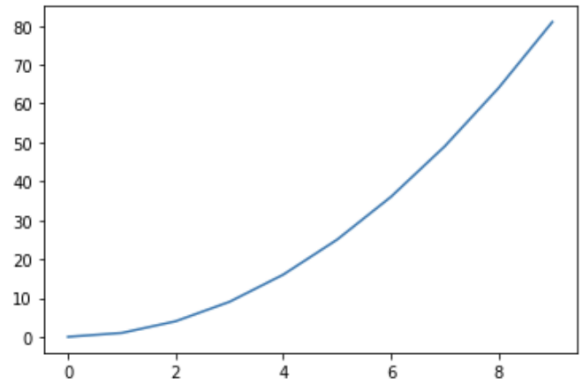
```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

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

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

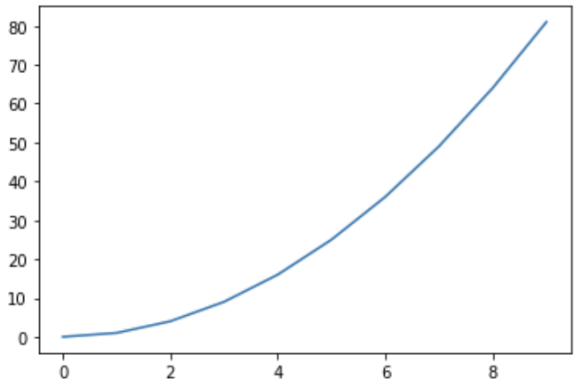
Python List

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



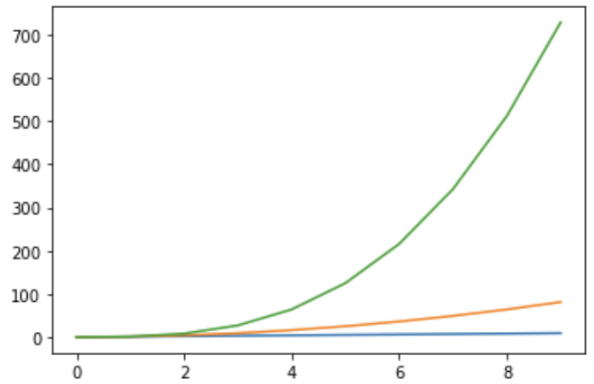
Pandas Series

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



Pandas DataFrame

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



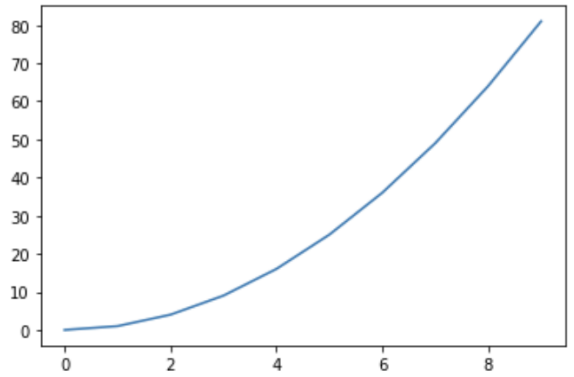
PLOTTING METHODS

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

PyPlot API

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

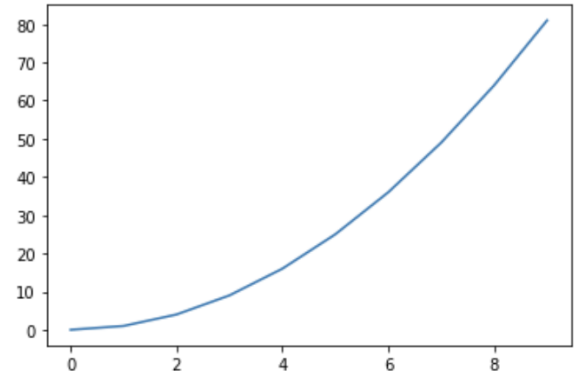
```
import matplotlib.pyplot as plt

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

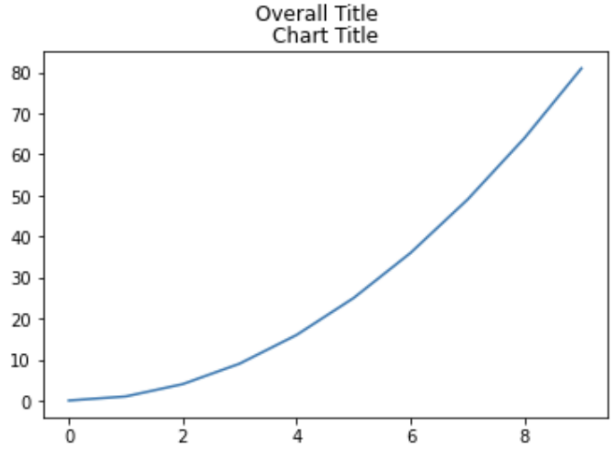
```
fig = plt.figure()

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

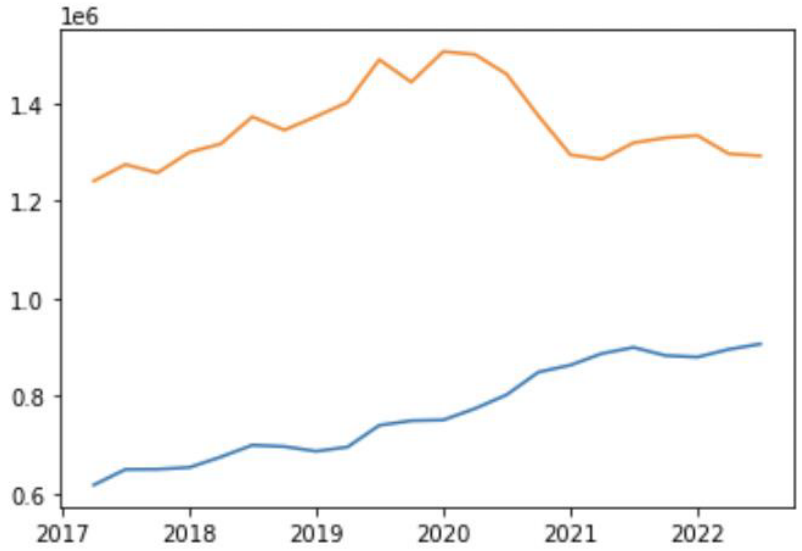
- Matplotlib Basics
- Object-Oriented Plotting
- Chart Formatting
- Chart Types

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

```
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)
```



PLOTTING DATAFRAMES

Matplotlib Basics

Object-Oriented Plotting

Chart Formatting

Chart Types

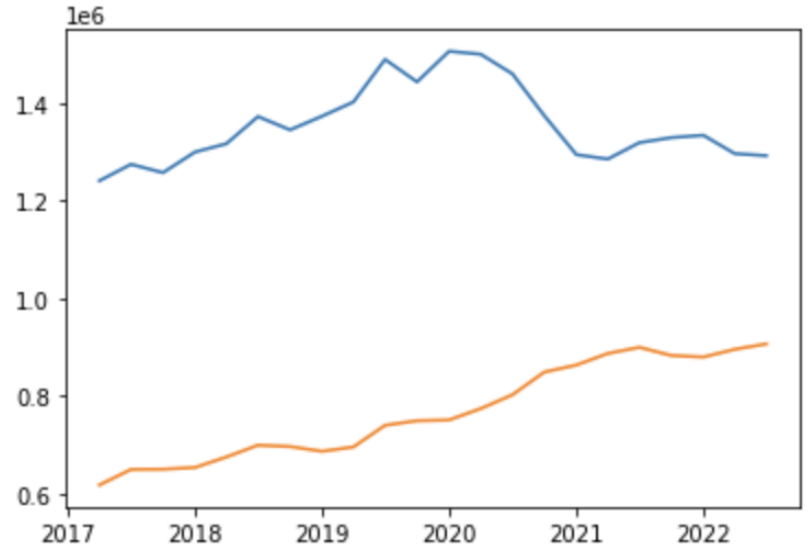
Plotting each series independently allows for improved customization

- **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"])
```



ASSIGNMENT: PLOTTING DATAFRAMES

1 **NEW MESSAGE**

August 2024 ,29

From: **Ian Intern** *(Summer Consultant)*

Subject: **Do you know Matplotlib?**

Hi!

I need someone who knows Matplotlib for help with some client work.

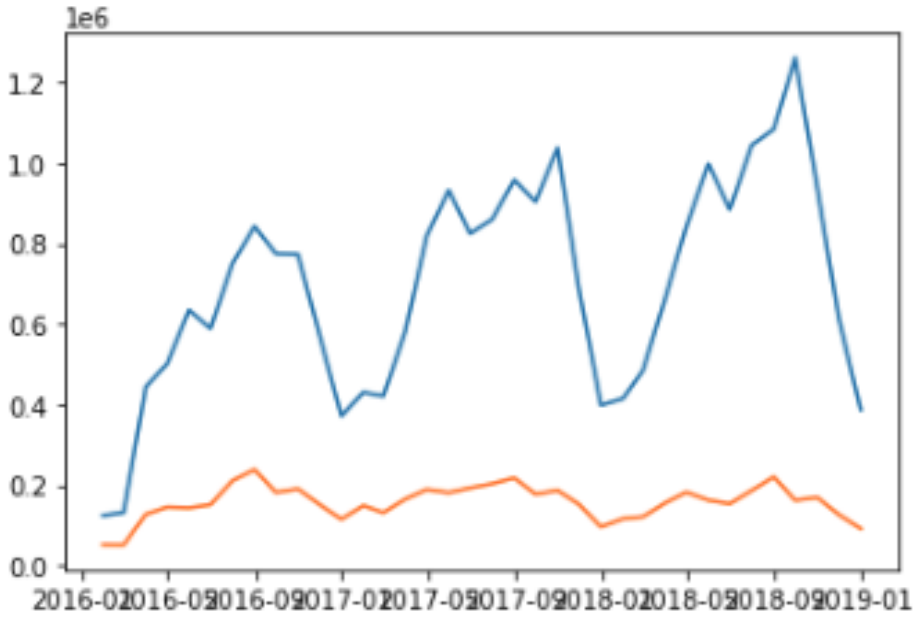
Can you plot Lodging Revenue and Other Revenue over time for our hotel client?

Thanks!

section02_assignments.ipynb

Reply Forward

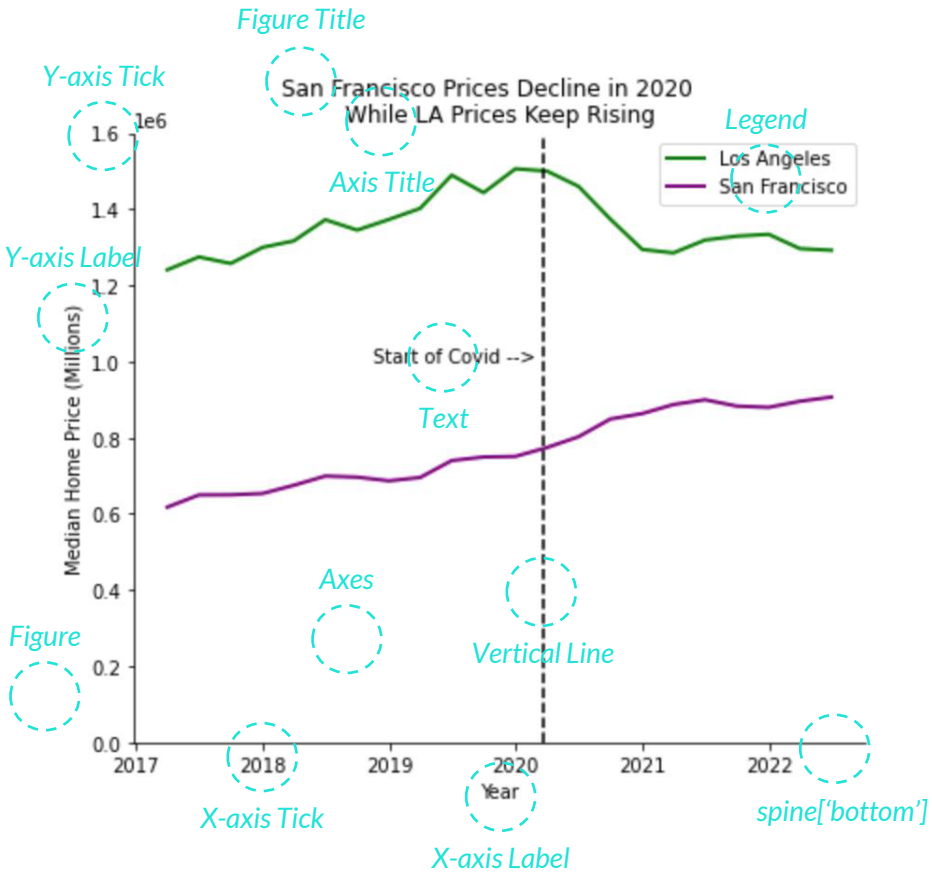
Results Preview



FORMATTING OPTIONS

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

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



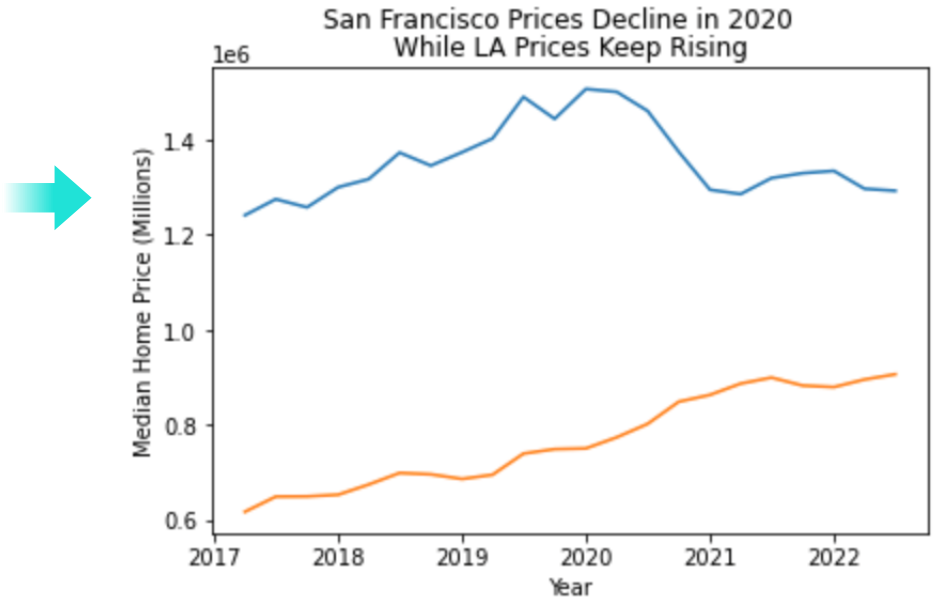
Option	Object-Oriented	PyPlot API
Figure Title	fig.suptitle()	plt.suptitle()
Chart Title	ax.set_title()	plt.subtitle()
X-Axis Label	ax.set_xlabel()	plt.xlabel()
Y-Axis Label	ax.set_ylabel()	plt.ylabel()
Legend	ax.legend()	plt.legend()
X-Axis Limit	ax.set_xlim()	plt.xlim()
Y-Axis Limit	ax.set_ylim()	plt.ylim()
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']

CHART TITLES

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

- fig.suptitle() serves as an overall figure title

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

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

fig.suptitle("San Francisco Prices Decline in 2020")
ax.set_title("While LA Prices Keep Rising")
ax.set_xlabel("Year")
ax.set_ylabel("Median Home Price (Millions)")
```



FONT SIZES

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

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

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

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

fig.suptitle(
    "San Francisco Prices Decline in 2020",
    fontsize=16
)
ax.set_title("While LA Prices Keep Rising")
ax.set_xlabel(
    "Year",
    fontsize=10
)
ax.set_ylabel(
    "Median Home Price (Millions)",
    fontsize=10
)
```

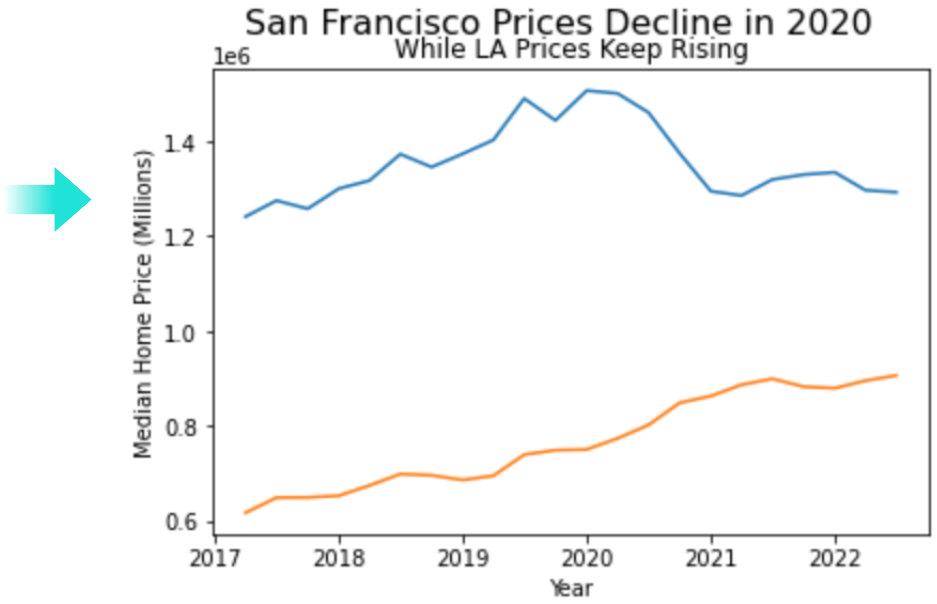


CHART LEGENDS

The legend() method lets you add a **chart legend** to identify each series

- The series labels are used by default, but custom values can also be passed through

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

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

)

fig.suptitle("San Francisco Prices Decline in 2020")
ax.set_title("While LA Prices Keep Rising")
ax.set_xlabel("Year")
ax.set_ylabel("Median Home Price (Millions)")

ax.legend()
```

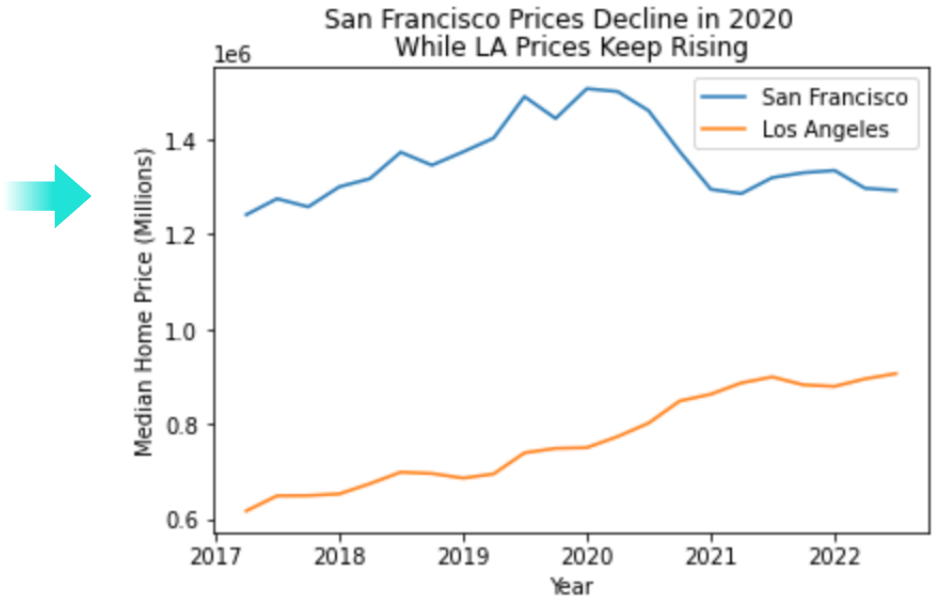
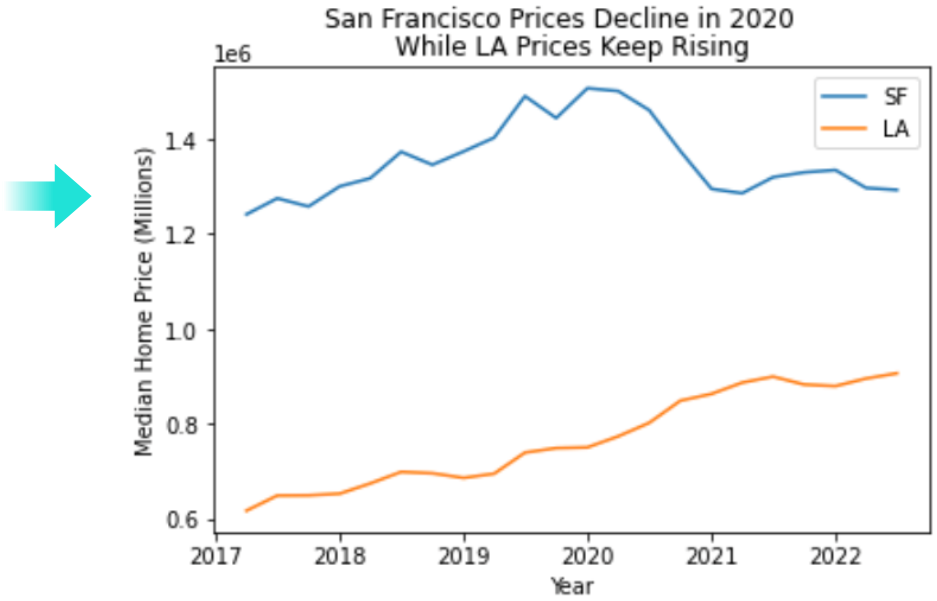


CHART LEGENDS

The legend() method lets you add a **chart legend** to identify each series

- The series labels are used by default, but custom values can also be passed through

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

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

)

fig.suptitle("San Francisco Prices Decline in 2020")
ax.set_title("While LA Prices Keep Rising")
ax.set_xlabel("Year")
ax.set_ylabel("Median Home Price (Millions)")

ax.legend(["SF", "LA"])
```

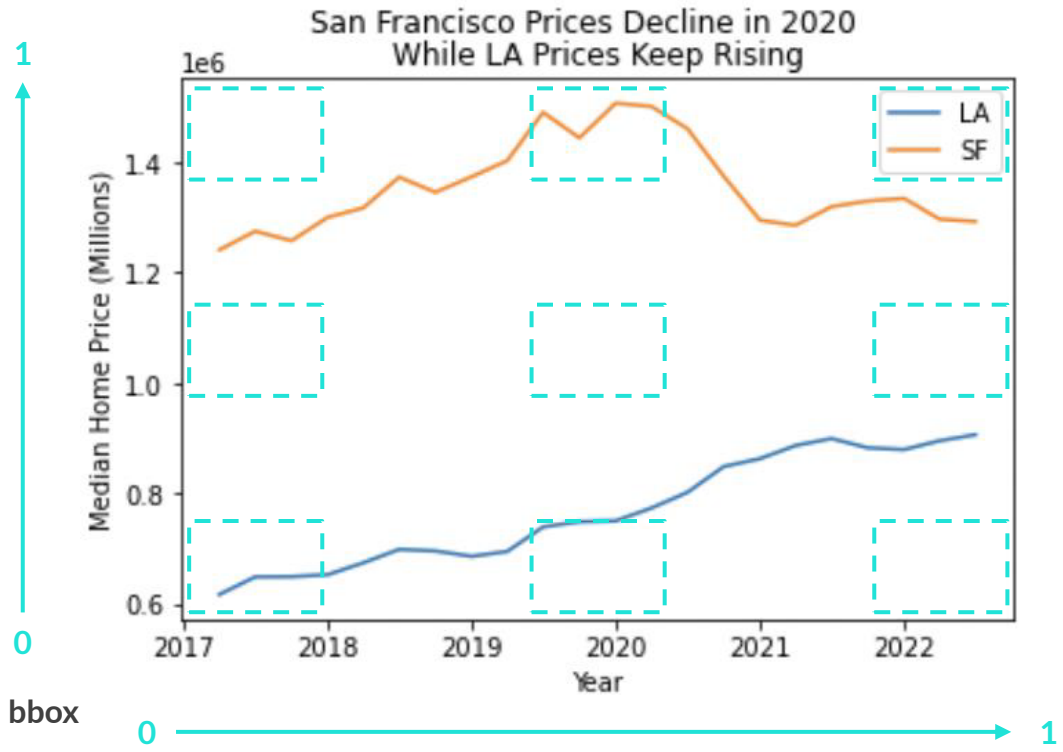


LEGEND LOCATION

You can change the **legend location** with the "loc" or "bbox_to_anchor" arguments

- "loc" lets you set a predetermined location option
- "bbox_to_anchor" lets you set specific (x, y) coordinates

Location Options
best *(default)*
upper right
upper left
upper center
lower right
lower left
lower center
center right
center left
center

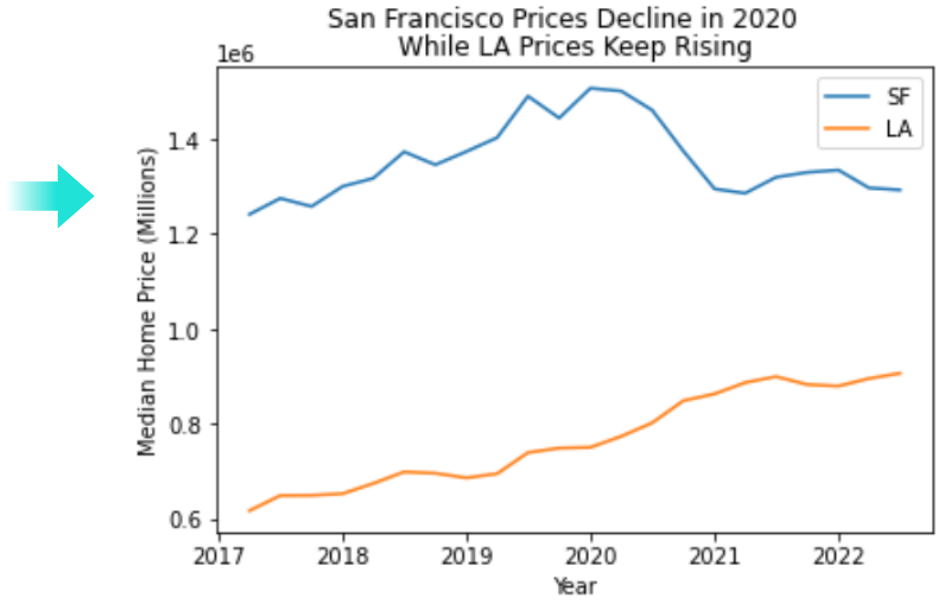


LEGEND LOCATION

You can change the **legend location** with the "loc" or "bbox_to_anchor" arguments

- "loc" lets you set a predetermined location option
- "bbox_to_anchor" lets you set specific (x, y) coordinates

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

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

)

fig.suptitle("San Francisco Prices Decline in 2020")
ax.set_title("While LA Prices Keep Rising")
ax.set_xlabel("Year")
ax.set_ylabel("Median Home Price (Millions)")

ax.legend(["SF", "LA"], loc="best")
```

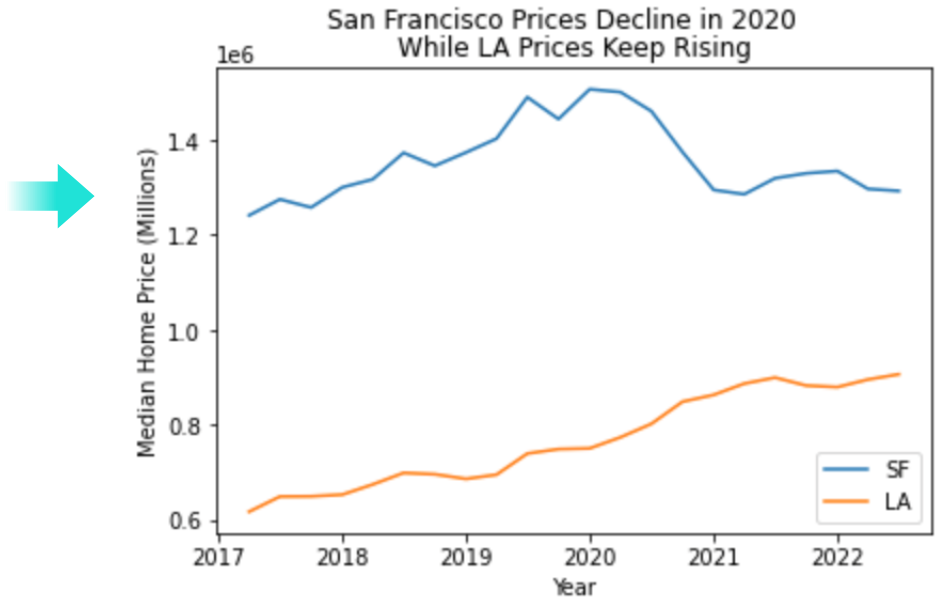


LEGEND LOCATION

You can change the **legend location** with the "loc" or "bbox_to_anchor" arguments

- "loc" lets you set a predetermined location option
- "bbox_to_anchor" lets you set specific (x, y) coordinates

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

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

)

fig.suptitle("San Francisco Prices Decline in 2020")
ax.set_title("While LA Prices Keep Rising")
ax.set_xlabel("Year")
ax.set_ylabel("Median Home Price (Millions)")

ax.legend(["SF", "LA"], loc="lower right")
```

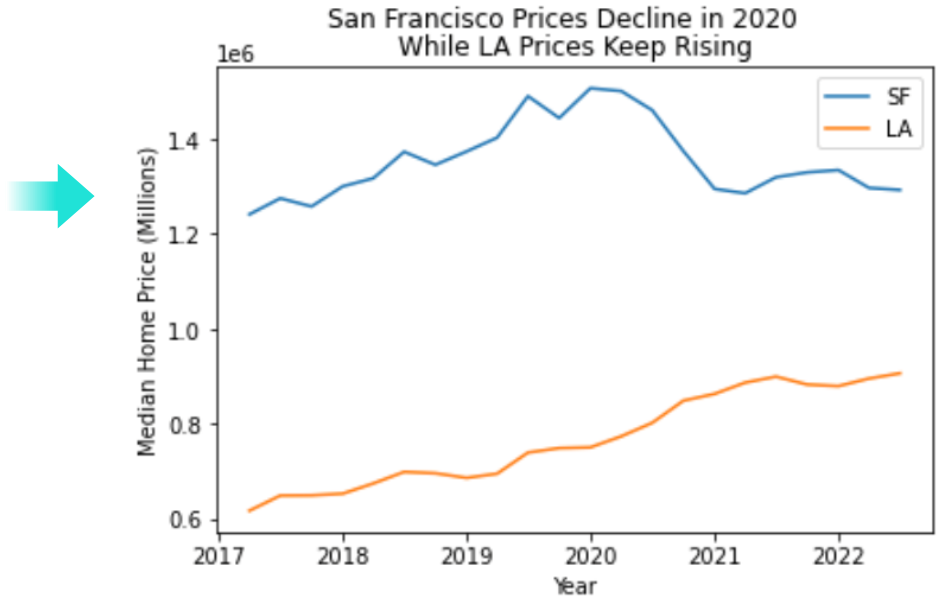


LEGEND LOCATION

You can change the **legend location** with the "loc" or "bbox_to_anchor" arguments

- "loc" lets you set a predetermined location option
- "bbox_to_anchor" lets you set specific (x, y) coordinates

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

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

)

fig.suptitle("San Francisco Prices Decline in 2020")
ax.set_title("While LA Prices Keep Rising")
ax.set_xlabel("Year")
ax.set_ylabel("Median Home Price (Millions)")

ax.legend(["SF", "LA"], bbox_to_anchor=(1, 1))
```

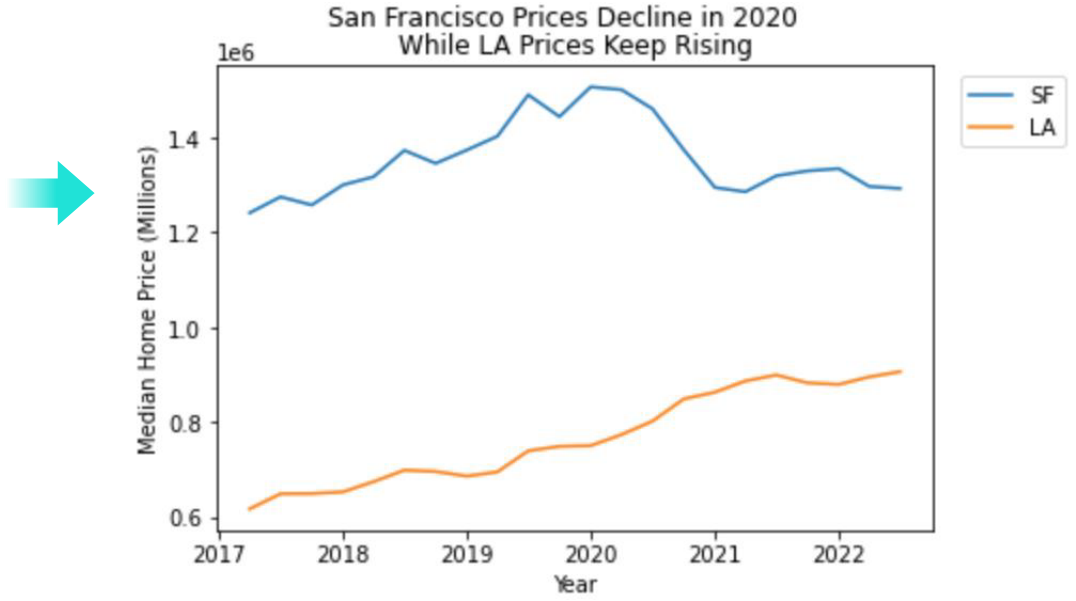


LEGEND LOCATION

You can change the **legend location** with the "loc" or "bbox_to_anchor" arguments

- "loc" lets you set a predetermined location option
- "bbox_to_anchor" lets you set specific (x, y) coordinates

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

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

)

fig.suptitle("San Francisco Prices Decline in 2020")
ax.set_title("While LA Prices Keep Rising")
ax.set_xlabel("Year")
ax.set_ylabel("Median Home Price (Millions)")

ax.legend(["SF", "LA"], bbox_to_anchor=(1.2, 1))
```



Setting coordinates beyond 1 will push the legend outside the chart area

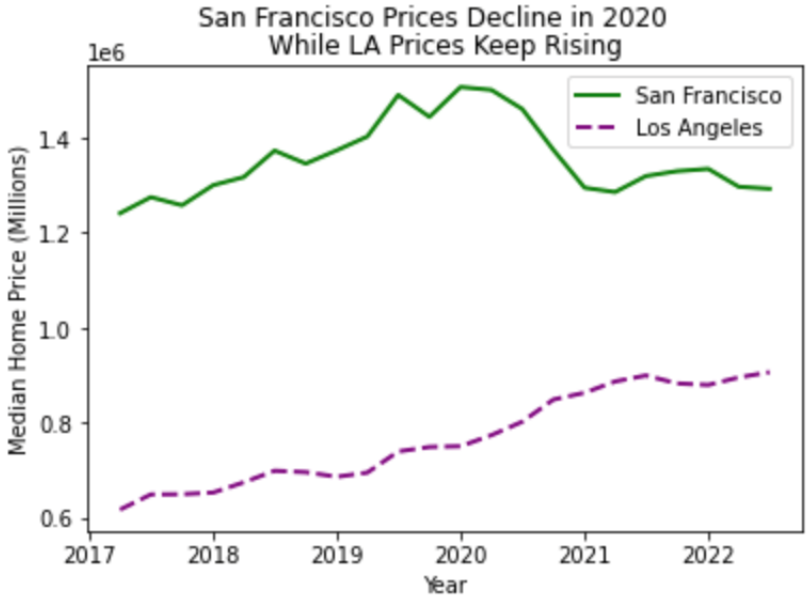
(useful when there is no whitespace!)

LINE STYLE

You can change the **line style** with the "linestyle", "linewidth", and "color" arguments

- Common line styles are "solid", "dashed", or "dotted" *(you can also use "-", "--", or ":")*

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

ax.plot(
    ca_housing.index,
    ca_housing["San Francisco"],
    color="green",
    linewidth=2
)
ax.plot(
    ca_housing.index,
    ca_housing["Los Angeles"],
    color="purple",
    linewidth=2,
    linestyle="--"
)

fig.suptitle("San Francisco Prices Decline in 2020")
ax.set_title("While LA Prices Keep Rising")
ax.set_xlabel("Year")
ax.set_ylabel("Median Home Price (Millions)")

ax.legend(ca_housing.columns)
```



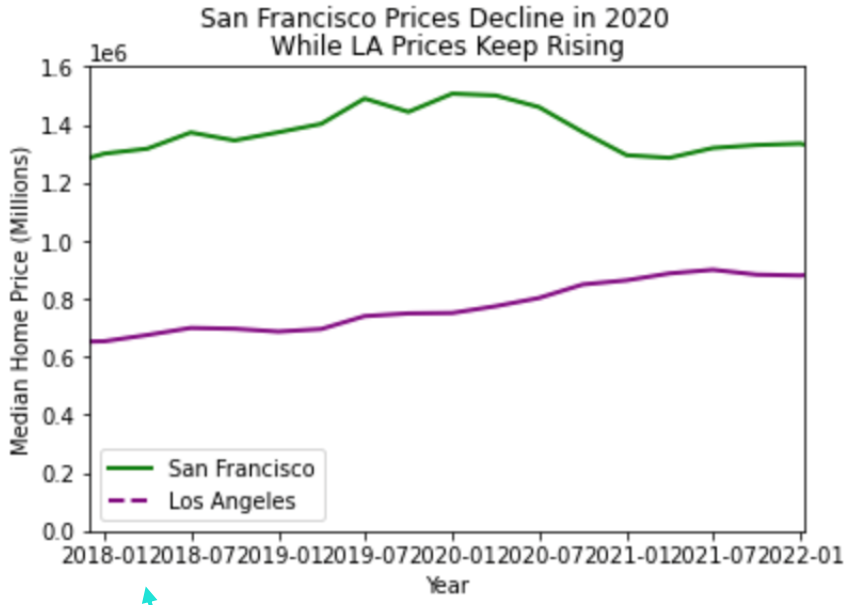
We will dive into colors in depth later, including **changing the default color palette** and **using hex color codes**!

AXIS LIMITS

The set_ylim() and set_xlim() functions let you modify the **axis limits**

- **ax.set_xlim(*lower limit*, *upper limit*)**

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

ax.plot(
    ca_housing.index,
    ca_housing["San Francisco"],
    color="green",
    linewidth=2
)
ax.plot(
    ca_housing.index,
    ca_housing["Los Angeles"],
    color="purple",
    linewidth=2
)

fig.suptitle("San Francisco Prices Decline in 2020")
ax.set_title("While LA Prices Keep Rising")
ax.set_xlabel("Year")
ax.set_ylabel("Median Home Price (Millions)")

ax.legend(ca_housing.columns)

ax.set_xlim(17500, 19000)
ax.set_ylim(0, 1600000)
```



Your date x-axis ticks may change interval size!

PRO TIP: Keeping the base of the y-axis at 0 highlights the true magnitude of change across periods and the differences between series

FIGURE SIZE

Matplotlib Basics

Object-Oriented Plotting

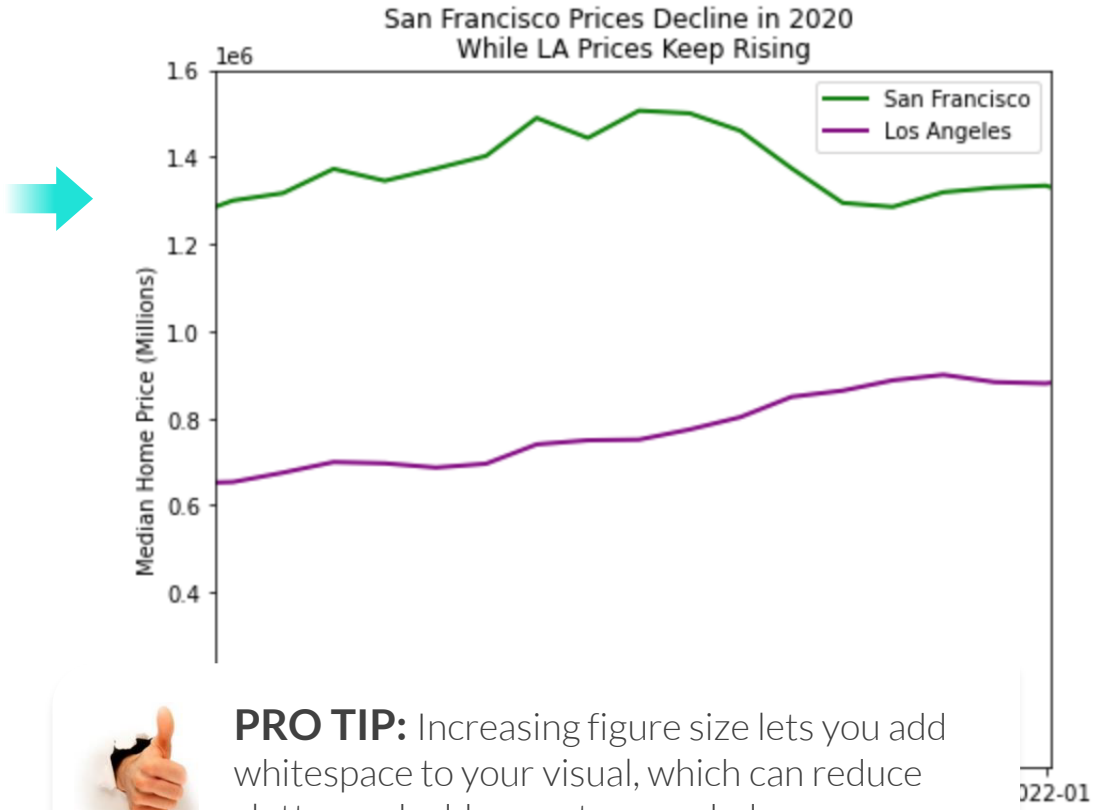
Chart Formatting

Chart Types

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=(7, 6))

ax.plot(
    ca_housing.index,
    ca_housing["San Francisco"],
    color="green",
    linewidth=2
)
ax.plot(
    ca_housing.index,
    ca_housing["Los Angeles"],
    color="purple",
    linewidth=2
)

fig.suptitle(
    "San Francisco Prices Decline in 2020",
    y=.95
)
ax.set_title("While LA Prices Keep Rising")
ax.set_xlabel("Year")
ax.set_ylabel("Median Home Price (Millions)")

ax.legend(ca_housing.columns)

ax.set_xlim(17500, 19000)
ax.set_ylim(0, 1600000)
```



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

CUSTOM X-TICKS

Matplotlib Basics

Object-Oriented Plotting

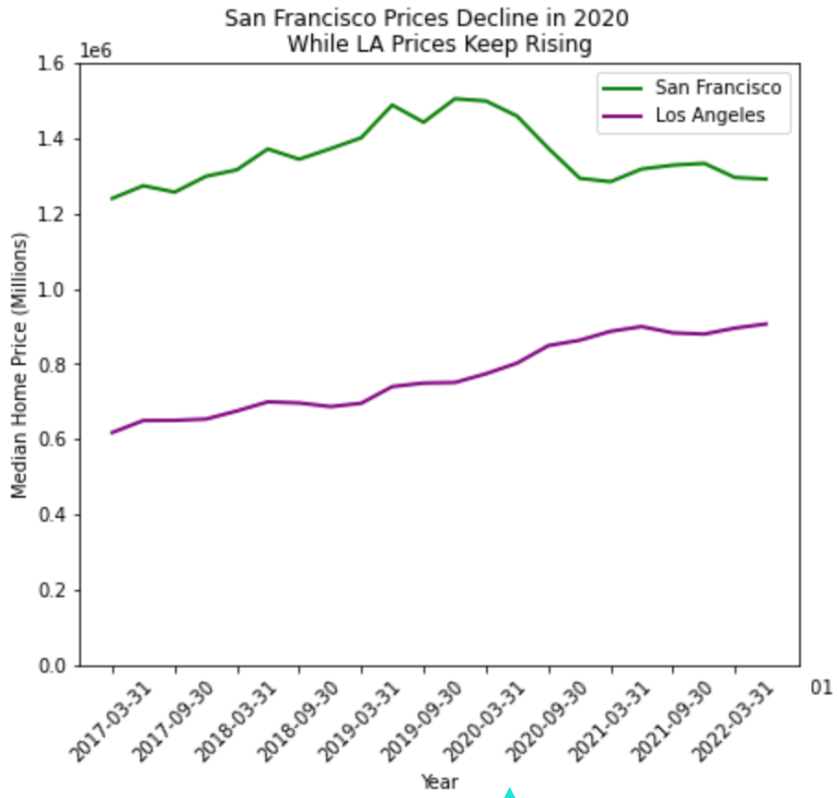
Chart Formatting

Chart Types

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

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

```
fig, ax = plt.subplots(figsize=(7, 6))

ax.plot(
    ca_housing.index,
    ca_housing["San Francisco"],
    color="green",
    linewidth=2
)
ax.plot(
    ca_housing.index,
    ca_housing["Los Angeles"],
    color="purple",
    linewidth=2
)

fig.suptitle(
    "San Francisco Prices Decline in 2020",
    y=.95
)
ax.set_title("While LA Prices Keep Rising")
ax.set_xlabel("Year")
ax.set_ylabel("Median Home Price (Millions)")

ax.legend(ca_housing.columns)
ax.set_ylim(0, 1600000)

ax.set_xticks(ca_housing.index[::2])
plt.xticks(rotation=45)
```



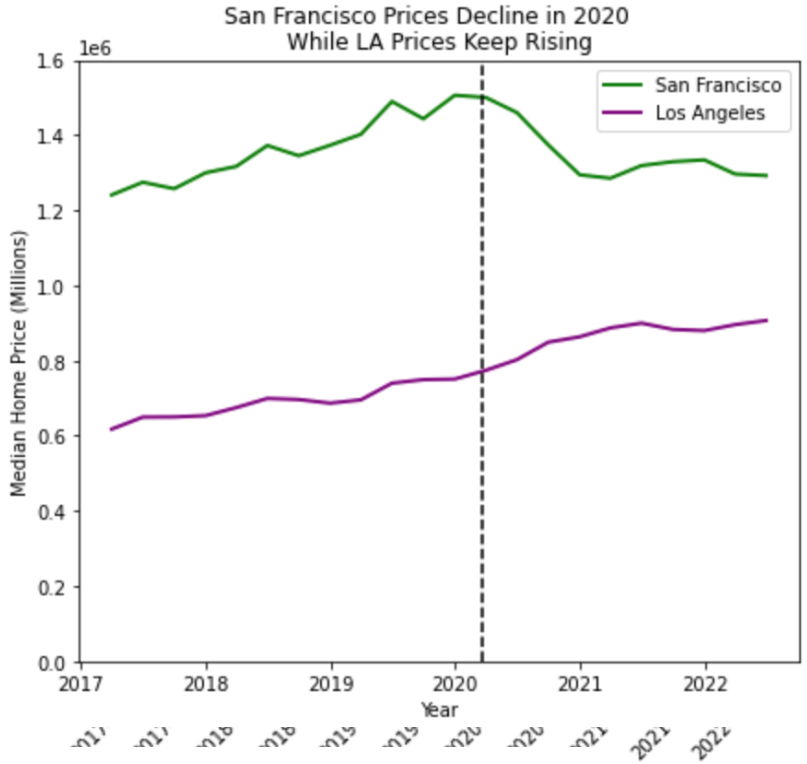
This sets the xticks at every 2nd date from the index and rotates them by 45 degrees

ADDING VERTICAL LINES

You can **add vertical lines** to mark key points with the axvline() function

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

```
fig, ax = plt.subplots(figsize=(7, 6))

ax.plot(
    ca_housing.index,
    ca_housing["San Francisco"],
    color="green",
    linewidth=2
)
ax.plot(
    ca_housing.index,
    ca_housing["Los Angeles"],
    color="purple",
    linewidth=2
)

fig.suptitle(
    "San Francisco Prices Decline in 2020",
    y=.95
)
ax.set_title("While LA Prices Keep Rising")
ax.set_xlabel("Year")
ax.set_ylabel("Median Home Price (Millions)")

ax.legend(ca_housing.columns)
ax.set_ylim(0, 1600000)

ax.axvline(18341, c="black", ls="--")
```



Set the coordinate (in this case days since Jan 1970 ,1) and an optional color and style

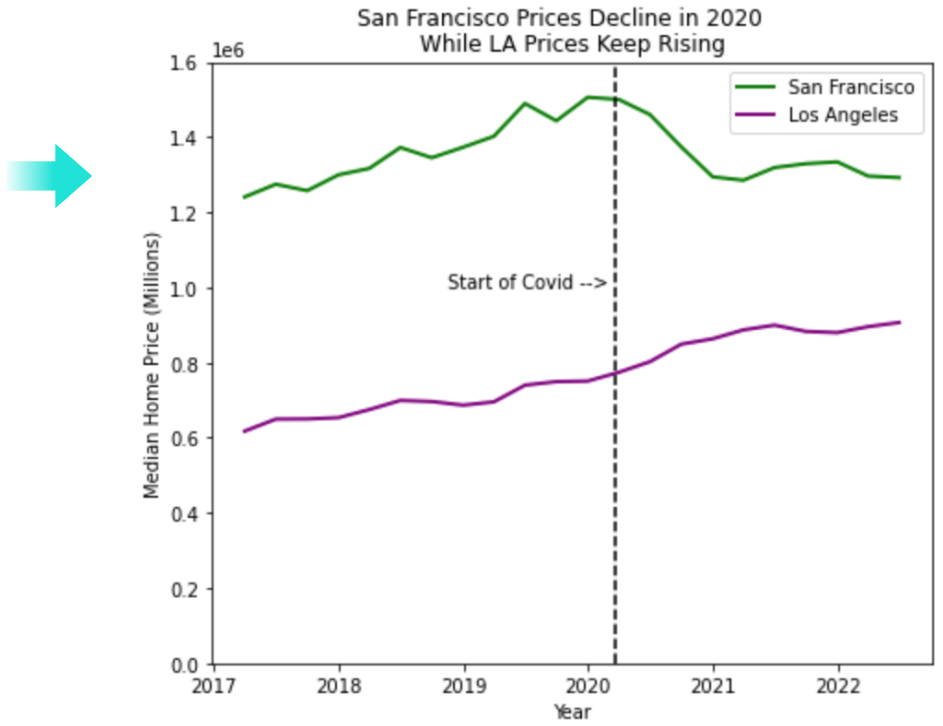
TEXT

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

You can **add text** at specific coordinates with the text() function

- **ax.text(*x-coordinate, y-coordinate, string, additional text formatting*)**

```
fig, ax = plt.subplots(figsize=(7, 6))

ax.plot(
    ca_housing.index,
    ca_housing["San Francisco"],
    color="green",
    linewidth=2
)
ax.plot(
    ca_housing.index,
    ca_housing["Los Angeles"],
    color="purple",
    linewidth=2
)

fig.suptitle(
    "San Francisco Prices Decline in 2020",
    y=.95
)
ax.set_title("While LA Prices Keep Rising")
ax.set_xlabel("Year")
ax.set_ylabel("Median Home Price (Millions)")

ax.legend(ca_housing.columns)
ax.set_ylim(0, 1600000)
ax.axvline(18341, c="black", ls="--")
ax.text(17850, 1000000, "Start of Covid -->")
```



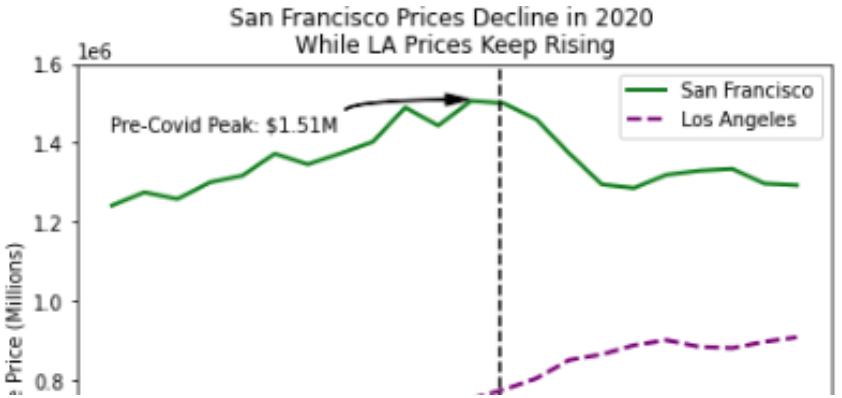
PRO TIP: ANNOTATIONS

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

Annotations are a great way to call-out and label important datapoints

- **ax.annotate(*string*, *datapoint coordinate*, *text coordinate*, *arrow style dictionary*, *text formatting*)**

```
ax.legend(ca_housing.columns[::-1])
ax.set_ylim(0, 1600000)
ax.axvline(18341, c="black", ls="--")
ax.annotate("Pre-Covid Peak: $1.51M ",
            xy=(18255, 1510000),
            xytext=(17250, 1450000),
            arrowprops=dict(
                facecolor="black",
                width=.5,
                headwidth=6,
                connectionstyle="angle3, angleA=290, angleB=0"
            ),
            verticalalignment="center"
)
```



Annotations have many more options that we won't cover in depth, but the documentation has great examples worth looking into!

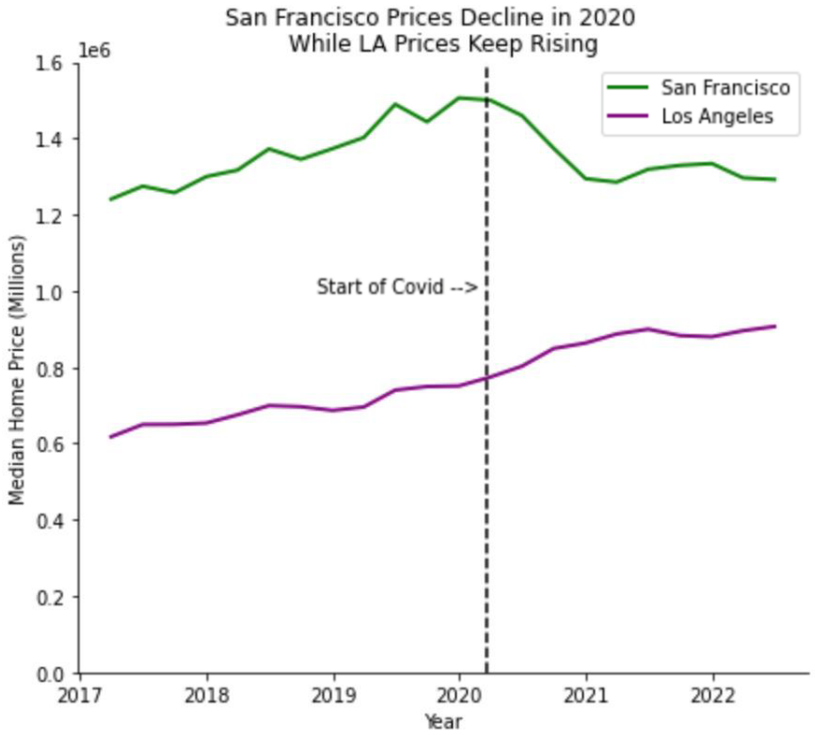
For a more info on annotations, visit: ***https://matplotlib.org/stable/tutorials/text/annotations.html#sphx-glr-tutorials-text-annotations-py***

REMOVING CHART BORDERS

You can **remove specific chart borders** with ax.spines[].set_visible(False)

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

```
fig, ax = plt.subplots(figsize=(7, 6))

ax.plot(
    ca_housing.index,
    ca_housing["San Francisco"],
    color="green",
    linewidth=2
)
ax.plot(
    ca_housing.index,
    ca_housing["Los Angeles"],
    color="purple",
    linewidth=2
)

fig.suptitle(
    "San Francisco Prices Decline in 2020",
    y=.95
)
ax.set_title("While LA Prices Keep Rising")
ax.set_xlabel("Year")
ax.set_ylabel("Median Home Price (Millions)")

ax.legend(ca_housing.columns)
ax.set_ylim(0, 1600000)
ax.axvline(18341, c="black", ls="--")
ax.text(17850, 1000000, "Start of Covid -->")

ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
```



This removes the right and top borders

ASSIGNMENT: CHART FORMATTING

NEW MESSAGE

August 2024 ,30

From: **Ian Intern** *(Summer Consultant)*

Subject: **RE: Final Charts for Client**

Hi there!

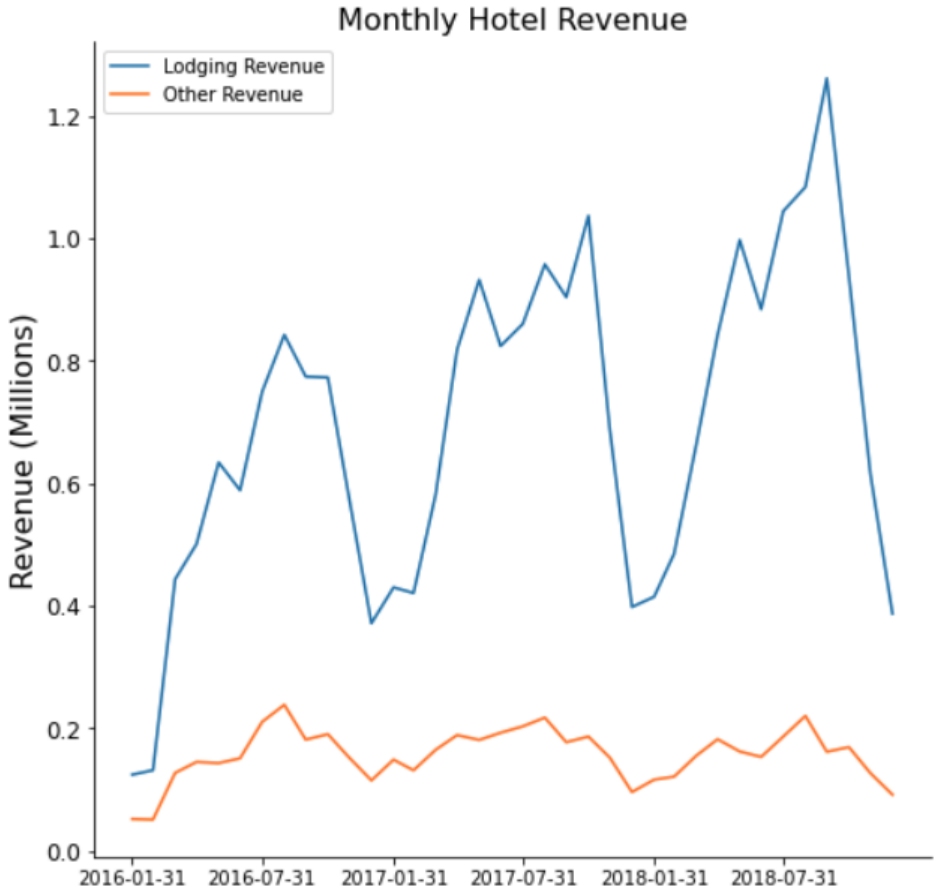
The data you plotted earlier looks good, but can you clean up the chart a little bit? I want it to to look polished for our client. This is my last day in my summer internship and I want to get hired back!

Thanks!

section02_assignments.ipynb

Reply Forward

Results Preview



LINE CHARTS

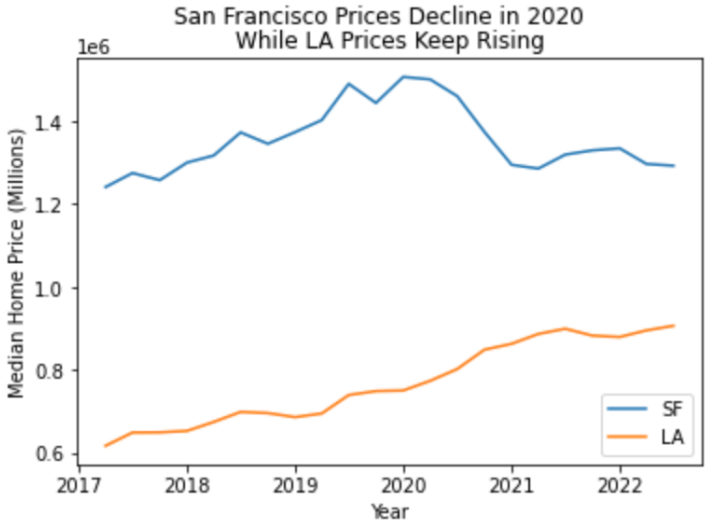
Line charts are used for showing trends over time

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

Column for each series

Dates as the index

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



PRO TIPS

- Pivot tabular data to turn each unique series into a DataFrame column, and set the datetime as the index
- Divide your series by the appropriate units while plotting to simplify the y-axis scale

LINE CHARTS

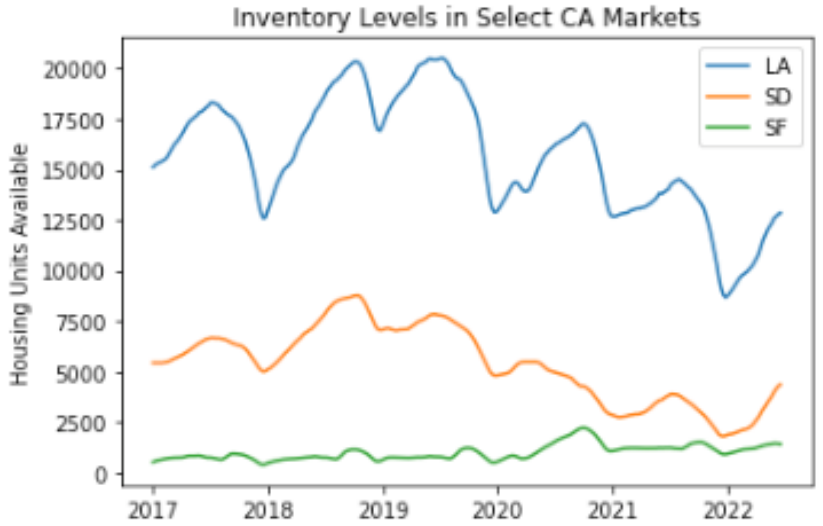
- Matplotlib Basics
- Object-Oriented Plotting
- Chart Formatting
- Chart Types

EXAMPLE *Available Housing Units by Week*

```
inventory.head(3)
```

region_name	Los Angeles	San Diego	San Francisco
period_begin			
2017-01-02	15114.0	5429.0	518.0
2017-01-09	15228.0	5428.0	563.0
2017-01-16	15306.0	5434.0	599.0

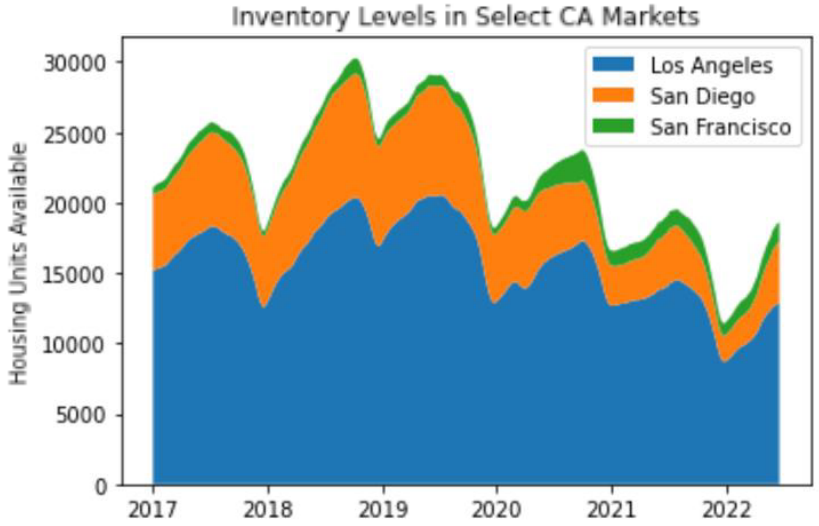
```
fig, ax = plt.subplots()

ax.plot(inventory.index, inventory["Los Angeles"])
ax.plot(inventory.index, inventory["San Diego"])
ax.plot(inventory.index, inventory["San Francisco"])

ax.set_title("Inventory Levels in Select CA Markets")
ax.set_ylabel("Housing Units Available")

ax.legend(["LA", "SD", "SF"])
```



STACKED LINE CHARTS

Use stackplot() to create a **stacked line chart**, which lets you visualize the overall trend over time, as well as its composition by series

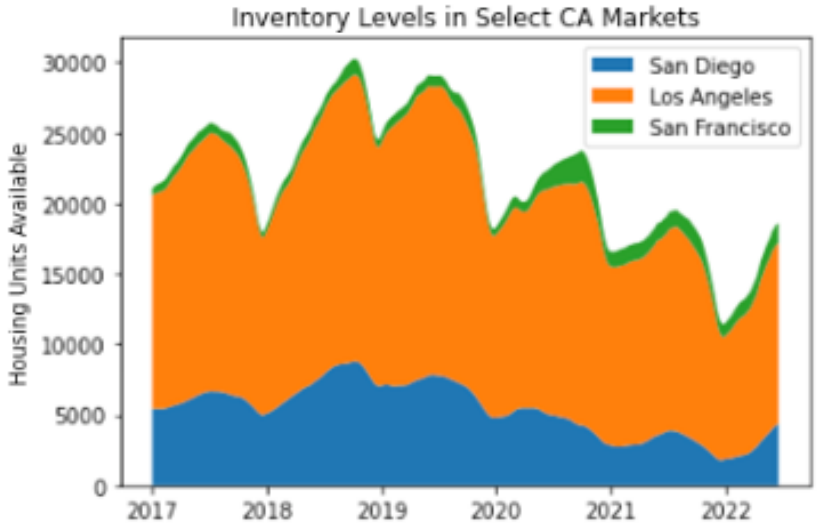

```
fig, ax = plt.subplots()

ax.stackplot(
    inventory.index,
    inventory["Los Angeles"],
    inventory["San Diego"],
    inventory["San Francisco"]
)

ax.set_title("Inventory Levels in Select CA Markets")
ax.set_ylabel("Housing Units Available")
ax.legend(inventory.columns)
```



STACKED LINE CHARTS

Use stackplot() to create a **stacked line chart**, which lets you visualize the overall trend over time, as well as its composition by series

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

ax.stackplot(
    inventory.index,
    inventory["San Diego"],
    inventory["Los Angeles"],
    inventory["San Francisco"]
)

ax.set_title("Inventory Levels in Select CA Markets")
ax.set_ylabel("Housing Units Available")
ax.legend(["San Diego", "Los Angeles", "San Francisco"])
```



PRO TIP: Use the bottom series in the stacked line chart to draw focus to its individual trend – it's the most visible!

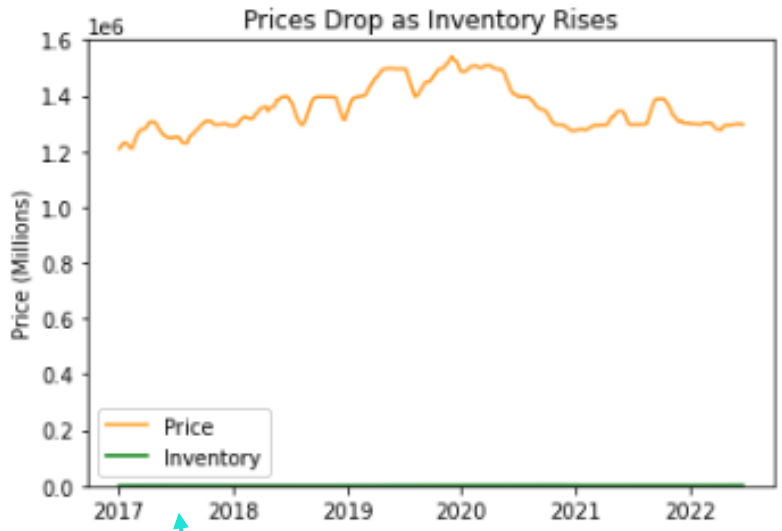
PRO TIP: DUAL AXIS CHARTS

Use twinx() to create a **dual axis chart**, which lets you plot series with values on significantly different scales inside a single visual


```
sf_dual.head(1)
```

period_begin	inventory	median_active_list_price
2017-01-02	518.0	1210000.0

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

ax.plot(
    sf_dual.index,
    sf_dual["median_active_list_price"],
    label="Price",
    c="orange"
)

ax.plot(
    sf_dual.index,
    sf_dual["inventory"],
    label="Inventory",
    c="green"
)

ax.set_title("Prices Drop as Inventory Rises")
ax.set_ylabel("Price (Millions)")
ax.set_ylim(0, 1600000)
ax.legend()
```



The "Inventory" values are so small compared to "Price" that they appear to be 0 when plotted on the same y-axis

PRO TIP: DUAL AXIS CHARTS

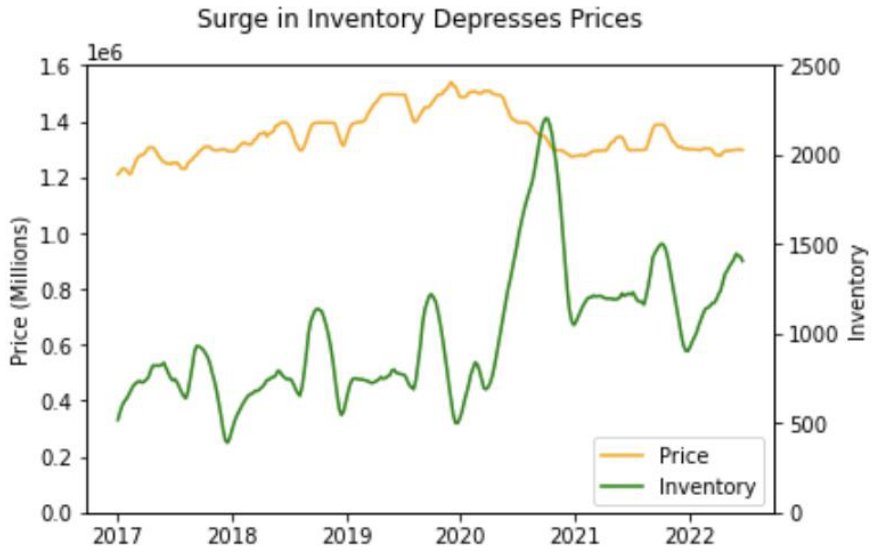
Use twinx() to create a **dual axis chart**, which lets you plot series with values on significantly different scales inside a single visual

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

```
sf_dual.head(1)
```

period_begin	inventory	median_active_list_price
2017-01-02	518.0	1210000.0

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

ax.plot(
    sf_dual.index,
    sf_dual["median_active_list_price"],
    label="Price",
    c="orange"
)

ax.set_ylabel("Price (Millions)")

ax2 = ax.twinx()

ax2.plot(
    sf_dual.index,
    sf_dual["inventory"],
    label="Inventory",
    c="green"
)

ax2.set_ylabel("Inventory")

fig.suptitle("Surge in Inventory Depresses Prices")
fig.legend(bbox_to_anchor=(.9, .88))
```



Create a second axis (ax2) with ax.twinx(), then create the desired plot on ax2

Note that using the figure level legend picks up both series

ASSIGNMENT: LINE CHARTS

NEW MESSAGE

August 2024 ,30

From: **Ian Intern** *(Summer Consultant)*

Subject: **Re: Re: Final Charts for Client**

Hey again,

Great work on those charts!

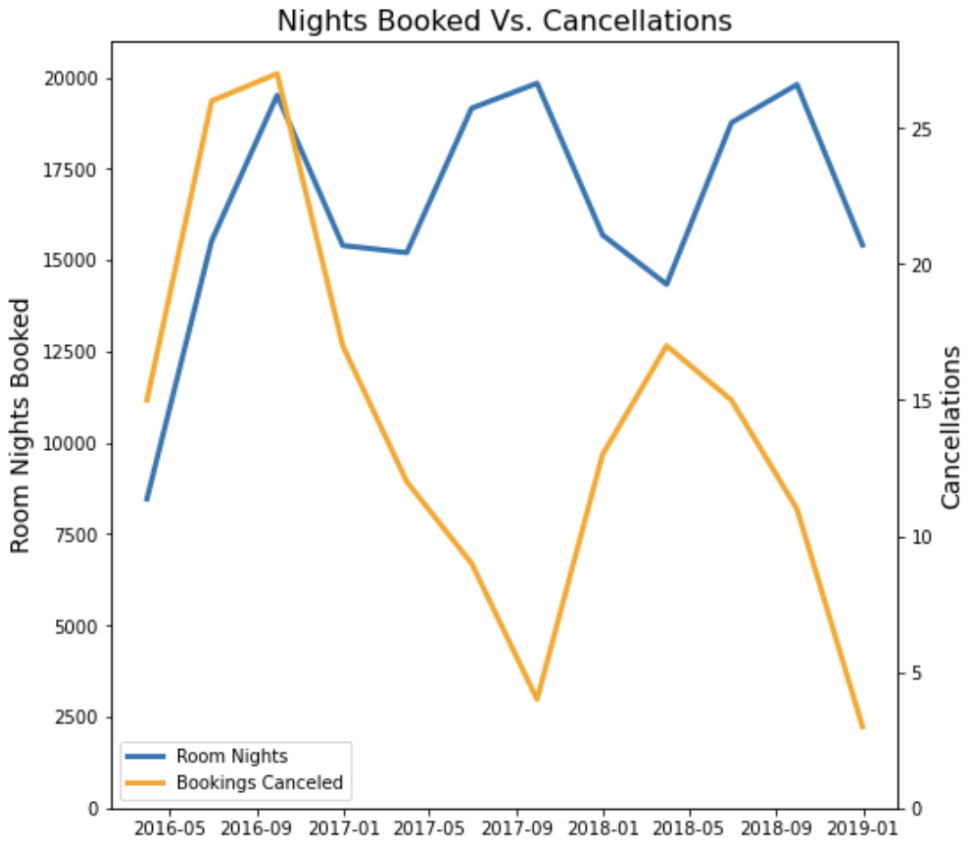
Final request - we want to plot compare room nights booked vs cancellations over time, we might need a dual axis chart to effectively do this. I'm totally checked out, so can you do this? You'll be put in contact with the client soon.

Thanks!

section02_assignments.ipynb

Reply Forward

Results Preview



BAR CHARTS

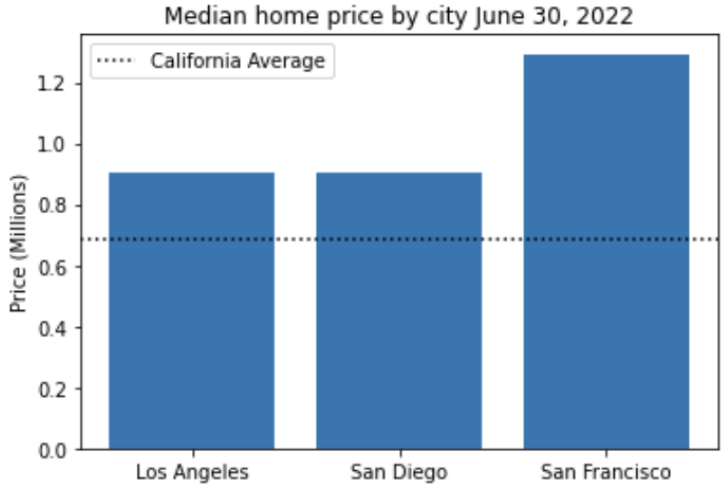
Bar charts are used to compare values across different categories

- **ax.bar(*category labels*, *bar heights*, *formatting options*)**

Values in a single column

Categories as the index

region_name	Price
Los Angeles	906774.0
San Diego	902425.0
San Francisco	1292479.0



PRO TIPS

- Use .groupby() and .agg() to aggregate your data by category and push the labels into the index
- Use Seaborn or the Pandas plot API for grouped bar charts

BAR CHARTS

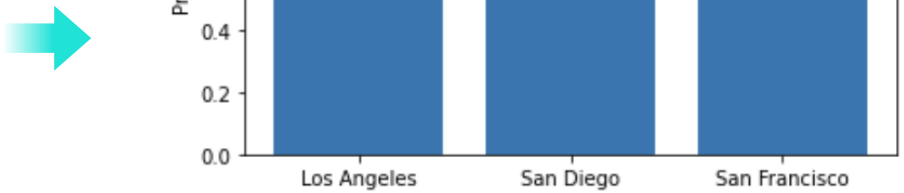
- Matplotlib Basics
- Object-Oriented Plotting
- Chart Formatting
- Chart Types

EXAMPLE *Median Home Price by City*

```
prices
```

region_name	Price
Los Angeles	906774.0
San Diego	902425.0
San Francisco	1292479.0

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

ax.bar(prices.index, prices["Price"] / 1000000)

ax.set_title("Median home price by city June 30, 2022")
ax.set_ylabel("Price (Millions)")
```



PRO TIP: HORIZONTAL LINES

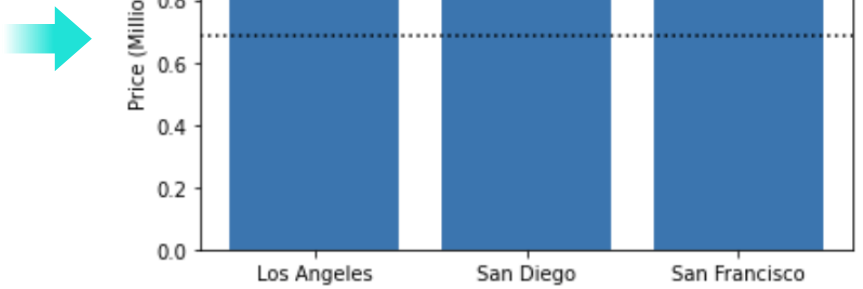
Use axhline() to add a **horizontal line** at a specified y-value on a bar chart

- This will typically be something to benchmark against, like a mean or target

```
mean_housing_price
```

```
687996
```

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

ax.bar(prices.index, prices["Price"] / 1000000)
ax.axhline(mean_housing_price / 1000000, color="black", ls=":")

ax.set_title("Median home price by city June 30, 2022")
ax.set_ylabel("Price (Millions)")

ax.legend(["California Average"])
```

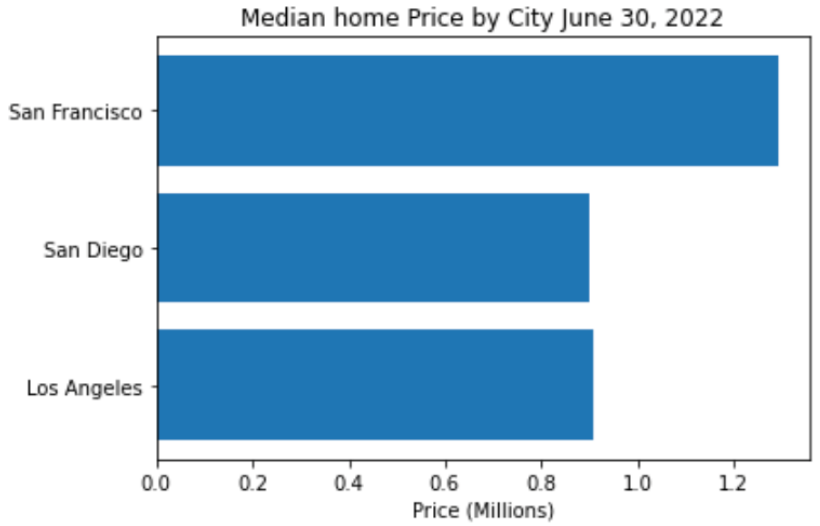


HORIZONTAL BAR CHARTS

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

Use barh() to create a **horizontal bar chart**

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

ax.barh(prices.index, prices["Price"] / 1000000)

ax.set_title("Median home Price by City June 30, 2022")
ax.set_xlabel("Price (Millions)")
```



Note that the Series in a horizontal bar chart are **sorted in the opposite order** as in a vertical bar chart

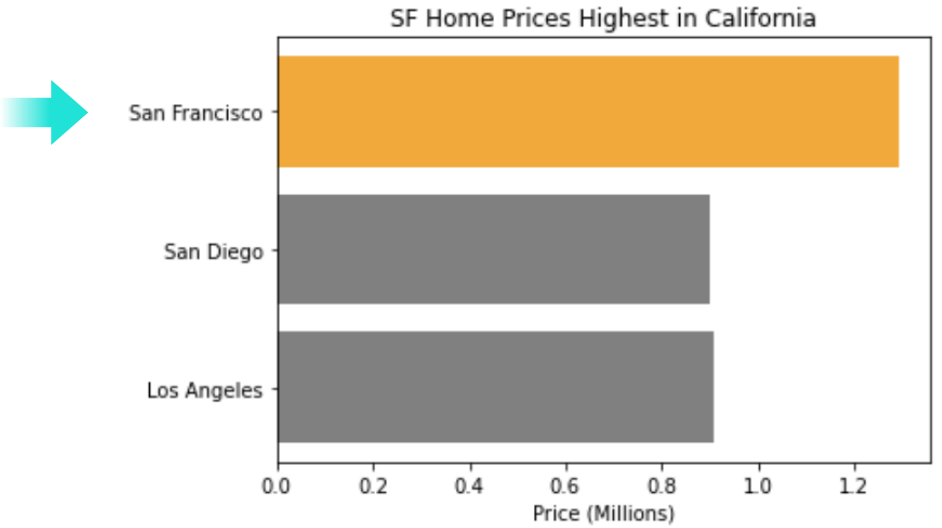
PRO TIP: HIGHLIGHTS

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

Use the “color” argument to **highlight** the series you’d like to focus on

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

ax.barh(
    prices.index, prices["Price"] / 1000000,
    color=["Grey", "Grey", "Orange"]
)

ax.set_title("SF Home Prices Highest in California")
ax.set_xlabel("Price (Millions)")
```

Use a list to specify the color for each Series



ASSIGNMENT: BAR CHARTS

NEW MESSAGE

September 2024 ,1

From: **Sarah Shark** *(Managing Director)*

Subject: **CHARTS NEEDED ASAP**

Hello,

Our hotel client is concerned about our intern's departure.

I need YOU to step up and make sure they're happy with us. Start by taking a quick look at room nights and lodging by country for our top 10 countries by total nights booked.

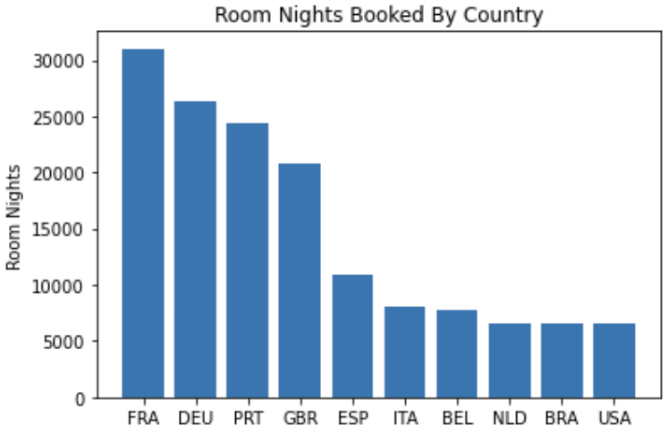
I expect the results in my inbox by morning (more details in the notebook attached).

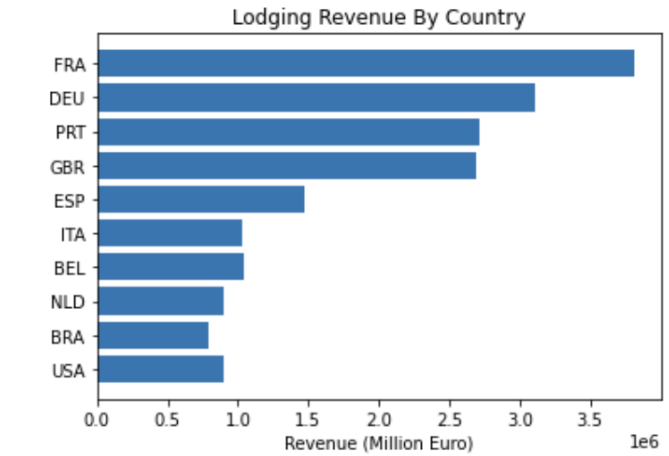
-S

section02_assignments.ipynb

Reply Forward

Results Preview





STACKED BAR CHARTS

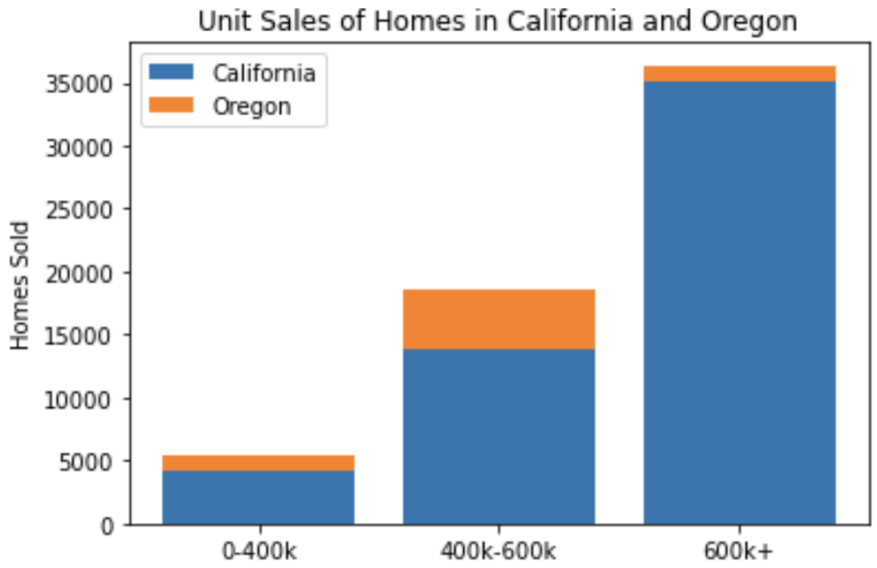
You can create a **stacked bar chart** by setting the "bottom" argument for the second "stacked" series as the values from the bars below it

- This will use those values as the baseline for the stacked bars instead of the x-axis

```
ca_or
```

state	CA	OR
price_range		
0-400k	4267.0	1074.0
400k-600k	13877.0	4711.0
600k+	35147.0	1229.0

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

ax.bar(
    ca_or.index,
    ca_or["CA"],
    label="California"
)

ax.bar(
    ca_or.index,
    ca_or["OR"],
    label="Oregon",
    bottom=ca_or["CA"]
)

ax.set_title("Unit Sales of Homes in CA and OR"
ax.set_ylabel("Homes Sold")

ax.legend()
```



The Oregon bars are plotted by using the California values as their "bottom"

%100 STACKED BAR CHARTS

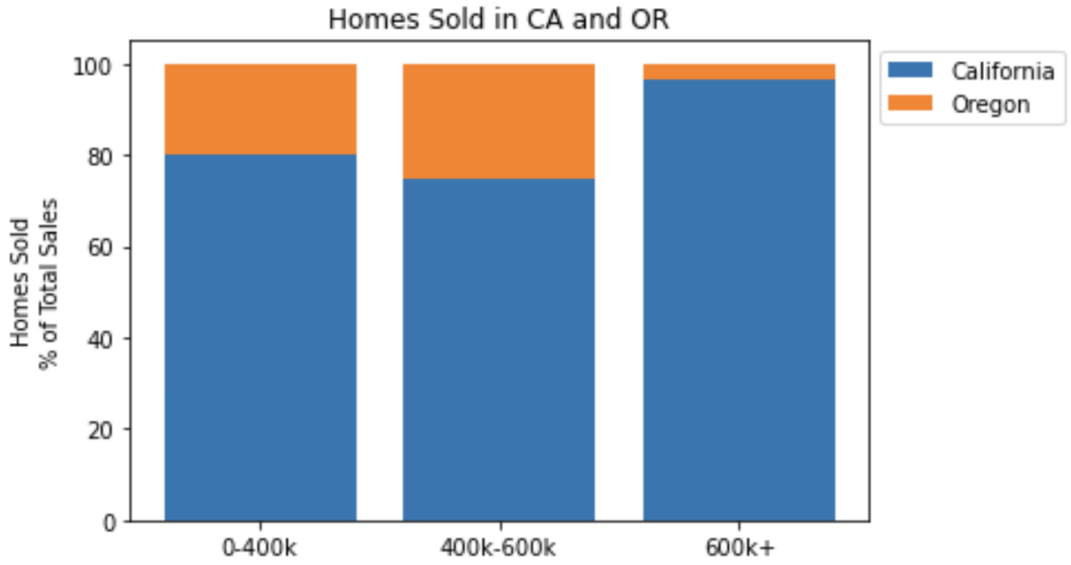
- Matplotlib Basics
- Object Oriented Plotting
- Chart Formatting
- Chart Types

To create a **%100 stacked bar chart**, convert your DataFrame to row-level percentages before plotting

```
ca_or = ca_or.apply(lambda x: x * 100 / sum(x), axis=1)

ca_or
```

state	CA	OR
price_range		
0-400k	79.891406	20.108594
400k-600k	74.655692	25.344308
600k+	96.621399	3.378601

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

ax.bar(
    ca_or.index,
    ca_or["CA"],
    label="California"
)

ax.bar(
    ca_or.index,
    ca_or["OR"],
    label="Oregon",
    bottom=ca_or["CA"]
)

ax.set_title("Homes Sold in CA and OR")
ax.set_ylabel("% of Total Sales")

ax.legend(bbox_to_anchor=(1,1))
```



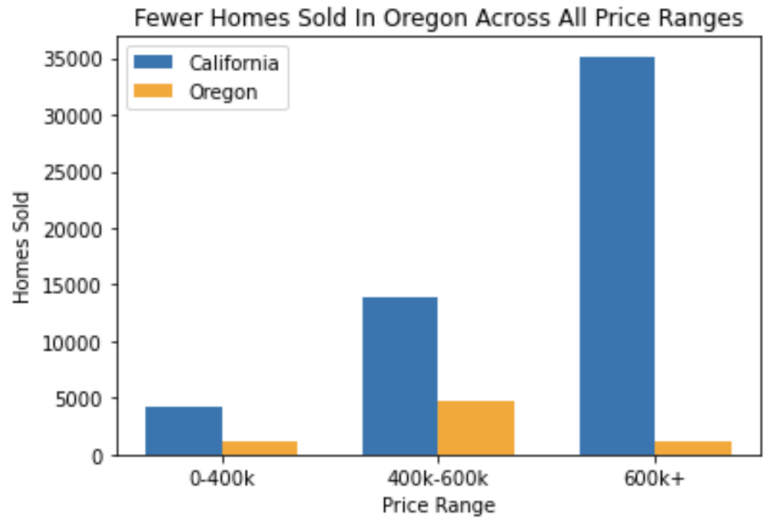
PRO TIP: GROUPED BAR CHARTS

You can create a **grouped bar chart** by reducing the width of each series and shifting them evenly around their corresponding label

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

width=.35
x=np.arange(3)  # length of index

ax.bar(
    x-width/2,
    ca_or["CA"],
    width=width,
    label="California"
)
ax.bar(
    x+width/2,
    ca_or["OR"],
    width=width,
    label="Oregon",
    color="orange"
)
ax.set_title("Fewer Homes Sold In Oregon Across All Price Ranges")
ax.set_xlabel("Price Range")
ax.set_ylabel("Homes Sold")
ax.set_xticks(x)
ax.set_xticklabels(ca_or.index)

ax.legend()
```

This shifts the bars to the left across the x-axis by half their width

This shifts these bars to the right



Grouped bar charts are much easier to create by using **Seaborn** or **Pandas' Matplotlib API**

PRO TIP: COMBO CHARTS

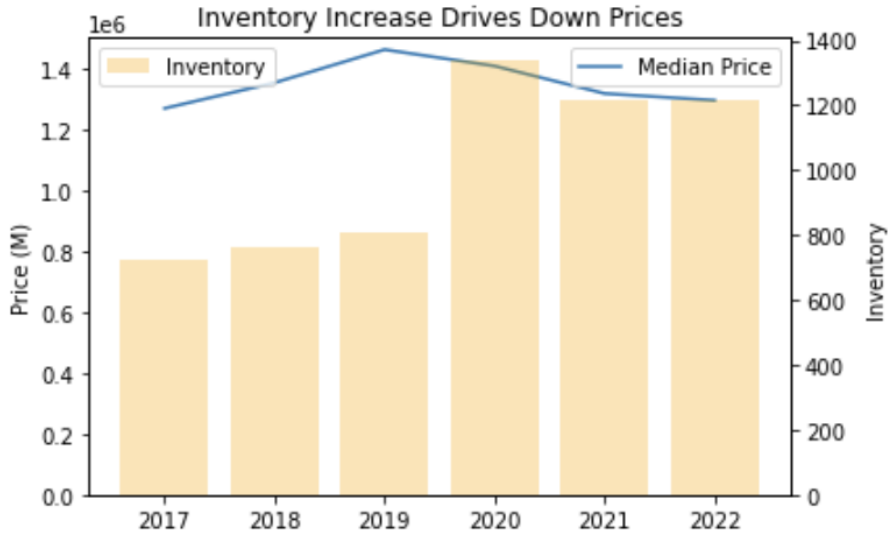
You can create a **combo chart** by specifying different chart types in a dual axis plot

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

```
sf_stats.round().head(2)
```

period_begin	inventory	median_active_list_price
2017-12-31	722.0	1268413.0
2018-12-31	765.0	1352510.0

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

x = np.arange(2017, 2023)
ax.set_title("Inventory Increase Drives Down Prices")
ax.plot(x,
        sf_stats["median_active_list_price"],
        label='Median Price'
       )

ax.set_ylim(0, 1500000)
ax.set_ylabel("Price (M)")
ax.legend(loc="upper right")

ax2 = ax.twinx()

ax2.bar(x,
        sf_stats["inventory"],
        color="Orange",
        label="Inventory",
        alpha=.3
       )
ax2.legend(loc="upper left")
ax2.set_ylabel("Inventory")
```



PRO TIP: Use the “alpha” argument to modify the transparency of each plot (0 is invisible and 1 is solid)

ASSIGNMENT: ADVANCED BAR CHARTS

NEW MESSAGE

September 2024 ,2

From: **Sarah Shark** *(Managing Director)*

Subject: **RE: RE: CHARTS NEEDED ASAP**

Hello,

Nice work...so far. I need some more detailed views on the breakdown of lodging revenue vs. other revenue by country.

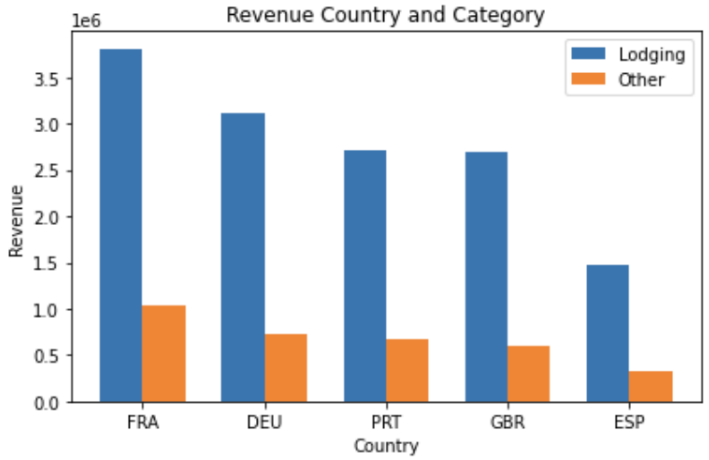
Build a grouped bar chart with the lodging revenue and other revenue for each country. Then, build a %100 stacked bar chart showing how much each revenue category contributes to overall country revenue. Add a reference line at %80 to help illustrate which countries get less than %80 of their revenue from lodging.

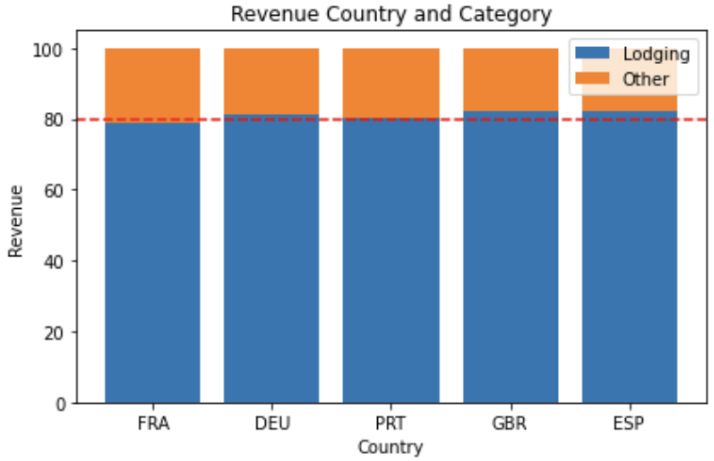
-S

section02_assignments.ipynb

Reply Forward

Results Preview





PIE CHARTS

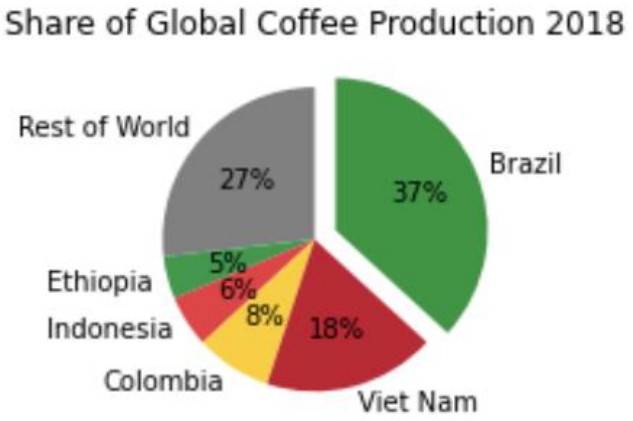
Pie charts are used to compare proportions totaling %100

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

Values in a single column

Labels as the index

	2018
total_production	
Brazil	62924.8836
Viet Nam	31174.0742
Colombia	13857.962
Indonesia	9417.5942
Ethiopia	7776.2234
Rest of World	91572.0006



PRO TIPS

- Keep the number of slices low (<7) to enhance readability – you can group “others” into a single slice
- Use bar charts if you want to compare the categories – pies are for showing how they make up a whole
- Donut charts make great KPI progress trackers

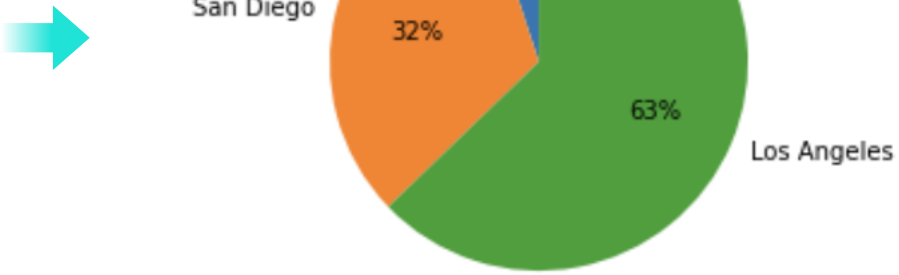
PIE CHARTS

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

EXAMPLE *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")
```

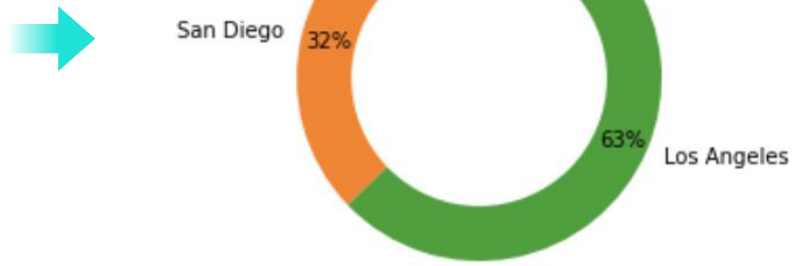


PRO TIP: DONUT CHARTS

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

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")
```



How does this code work?

- It pushes the data labels %85 of the way towards the edge of the pie chart
- Then adds a white circle that covers the center of the pie chart to the figure

ASSIGNMENT: PIE & DONUT CHARTS

NEW MESSAGE

September 2024 ,3

From: **Sarah Shark** *(Managing Director)*

Subject: **UPDATED CHARTS**

Hello,

Our hotel client is looking for a pie/donut chart to represent the share of revenue by country.

Create a pie chart with slices for the top 5 countries by revenue, and a single "other" slice for the rest of the countries.

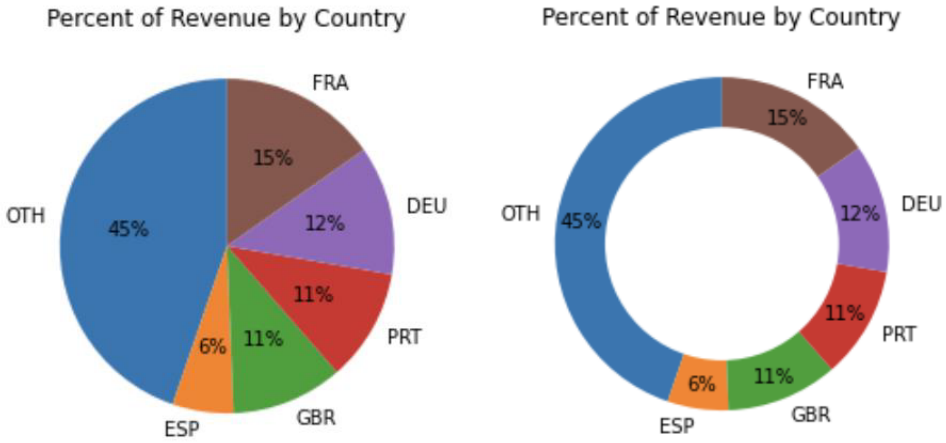
Need it ASAP.

Thx

section02_assignments.ipynb

Reply Forward

Results Preview



SCATTERPLOTS

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

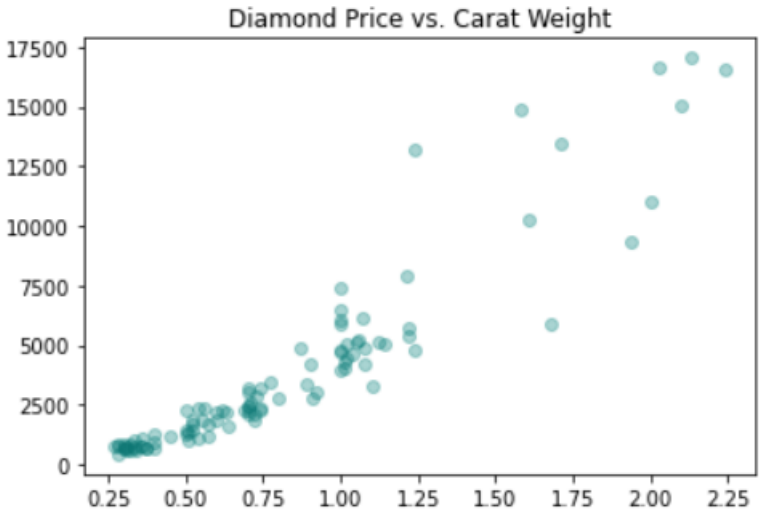
Scatterplots are used to visualize the relationship between numerical variables

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

x-series *y-series*

One row per point

	carat	cut	color	clarity	depth	table	price
0	0.23	Ideal	E	SI2	61.5	55.0	326
1	0.21	Premium	E	SI1	59.8	61.0	326
2	0.23	Good	E	VS1	56.9	65.0	327
3	0.29	Premium	I	VS2	62.4	58.0	334
4	0.31	Good	J	SI2	63.3	58.0	335



PRO TIPS

- Modify the alpha (transparency) level to make overlapping points more visible
- Bubble charts *can* be useful in some cases, but they often add confusion rather than clarity

SCATTERPLOTS

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

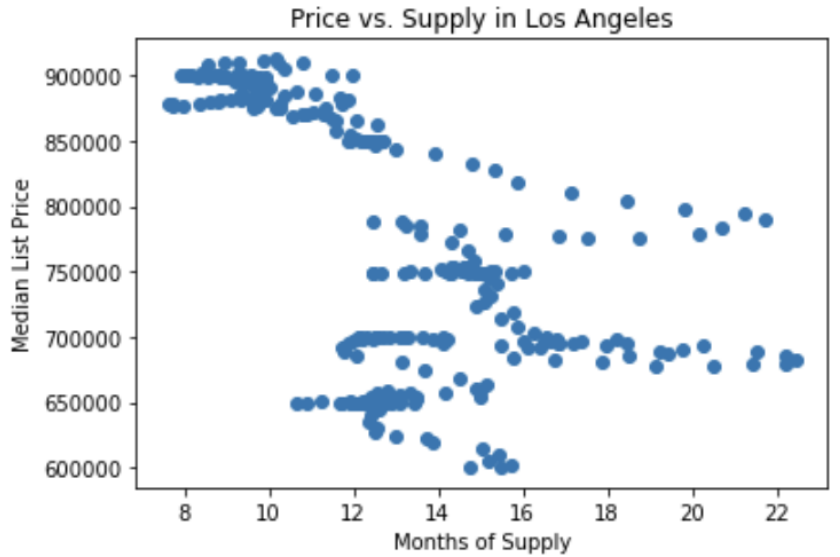
EXAMPLE

Months of Supply vs. Median List Price

```
la_housing.head()
```

	region_name	period_begin	median_active_list_price	months_of_supply	total_homes_sold
4158	Los Angeles	2020-05-04	790098.5	21.743506	2942.0
4182	Los Angeles	2021-07-12	896500.0	9.698174	6724.0
10387	Los Angeles	2021-12-06	877600.0	7.729714	5755.0
20443	Los Angeles	2020-01-20	758475.0	14.850536	4034.0
27113	Los Angeles	2017-05-01	649962.5	12.299351	6310.0

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

ax.scatter(
    la_housing["months_of_supply"],
    la_housing["median_active_list_price"]
)

ax.set_title("Price vs. Supply in Los Angeles")
ax.set_xlabel("Months of Supply")
ax.set_ylabel("Median List Price")
```

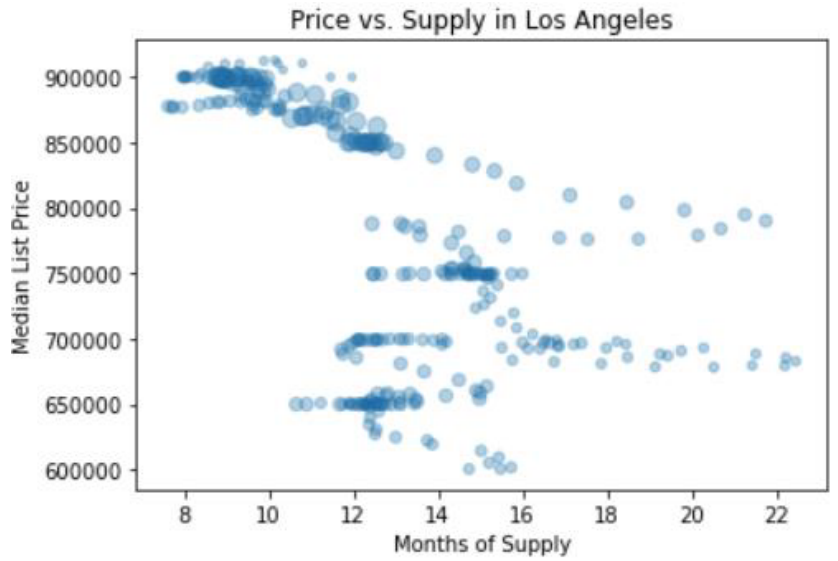


BUBBLE CHARTS

To create a **bubble chart**, specify a third series in the "size" argument of .scatter()

- You may need to apply some arithmetic to adjust the bubble sizes

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

ax.scatter(
    x=la_housing["months_of_supply"],
    y=la_housing["median_active_list_price"],
    s = (la_housing["median_active_list_ppsf_yoy"] * 500),
    alpha=.3
)

ax.set_title("Price vs. Supply in Los Angeles")
ax.set_xlabel("Months of Supply")
ax.set_ylabel("Median List Price")
```



HISTOGRAMS

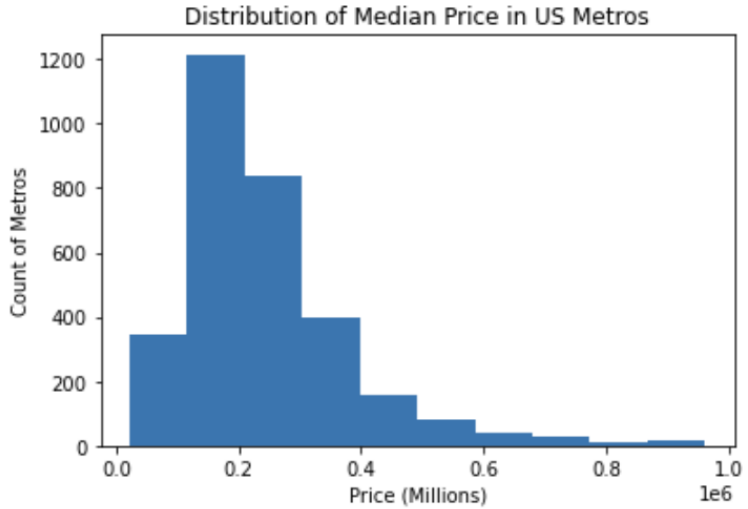
Matplotlib Basics
Object Oriented Plotting
Chart Formatting
Chart Types

Histograms are used to visualize the distribution of a numeric variable

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

numerical series

region_name	median_active_list_price
Abbeville County, SC	201869.897655
Aberdeen, WA metro area	264782.542556
Abilene, TX metro area	205214.942204
Acadia Parish, LA	142864.422978
Ada County, ID	428137.295428



PRO TIPS

- Modify the alpha (transparency) level to plot multiple distributions on the same axis
- Set density=True to use relative frequencies on the y-axis (percent of total)

HISTOGRAMS

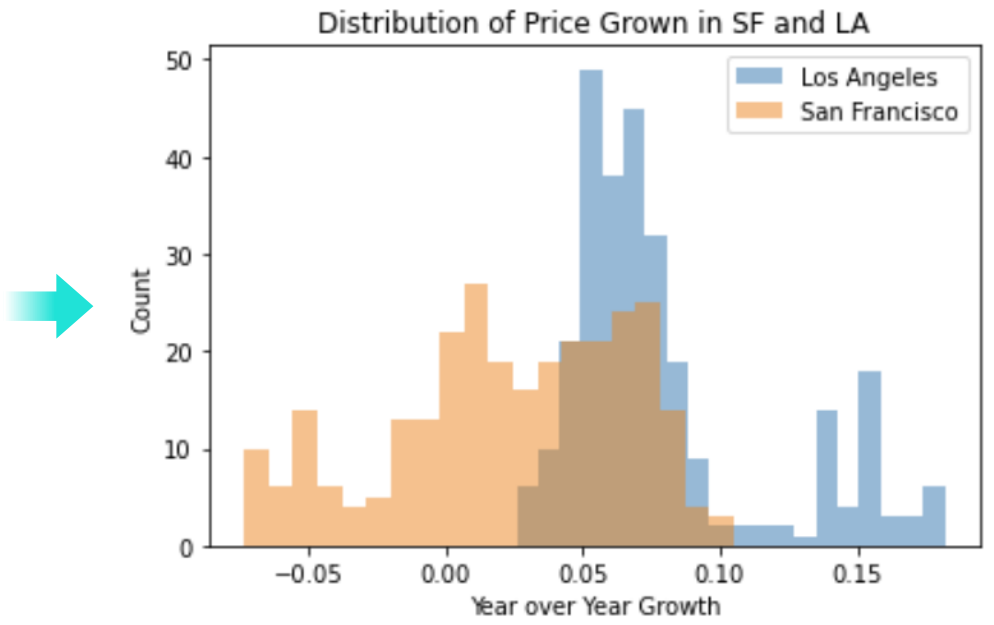
- Matplotlib Basics
- Object Oriented Plotting
- Chart Formatting
- Chart Types

EXAMPLE *Distribution Y-o-Y Growth in Home Price for Calendar Weeks*

```
ca_housing.head(2)
```

region_name	Los Angeles	San Francisco
period_end		
2017-01-29	0.050726	0.068286
2017-02-05	0.050887	0.067319

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

ax.hist(
    ca_housing["Los Angeles"],
    alpha=.5,
    bins=20
)
ax.hist(
    ca_housing["San Francisco"],
    alpha=.5,
    bins=20
)

ax.set_title("Distribution of Price Grown in SF and LA")
ax.set_xlabel("Year over Year Growth")
ax.set_ylabel("Count")

ax.legend(ca_housing.columns)
```



ASSIGNMENT: SCATTERPLOTS & HISTOGRAMS

NEW MESSAGE

September 2024 ,4

From: **Sarah Shark** *(Managing Director)*

Subject: **Additional Customer Profiling**

Not bad rookie – thanks for the quick turnaround.

I need two more charts to help finalize a marketing strategy targeting overseas guests:

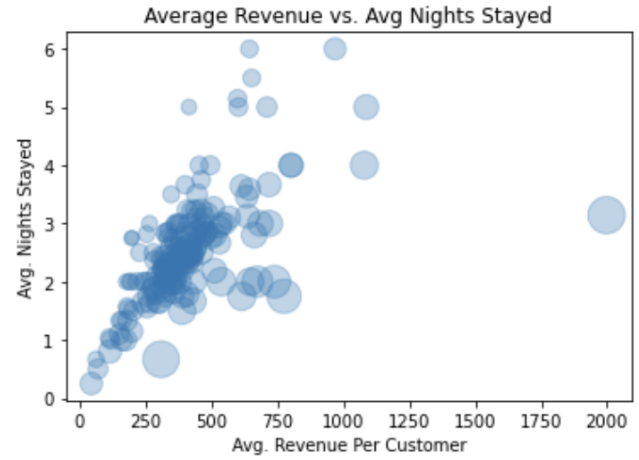
1. A chart comparing average revenue per customer and average nights stayed, with average nightly revenue as the size of the bubbles (you'll need to aggregate the data by country)
2. The distribution of customer ages in France & Germany

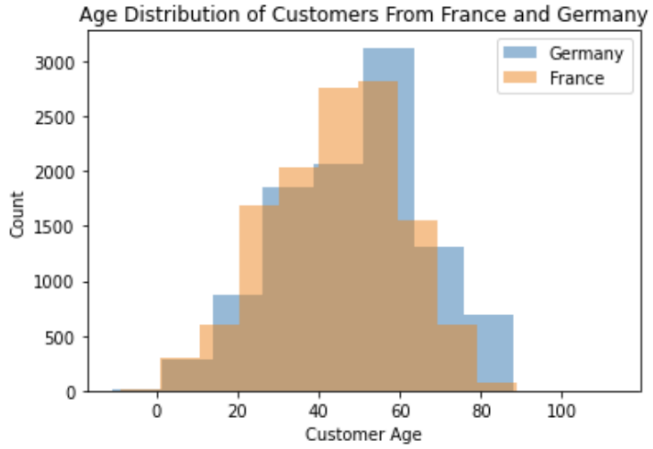
-sent from my yPhone

section02_assignments.ipynb

Reply Forward

Results Preview





KEY TAKEAWAYS

- Matplotlib has **two methods** for plotting data: PyPlot API & Object Oriented
  - *Both can visualize many data types (lists, DataFrames, etc.), but object-oriented plots are easier to fully customize*
- Object Oriented plots are built by adding **axes** to a **figure**
  - *You can layer on different elements to these objects to modify the chart formatting*
- You can create **common chart types** by using Matplotlib functions
  - *Each chart type can be customized further to create more advanced variations*
- Matplotlib's extreme customizability also adds **complexity**
  - *Understanding the anatomy of a Matplotlib figure helps pinpoint how to change every component in your chart*

PROJECT 1: VISUALIZING DATA

PROJECT DATA: COFFEE PRODUCTION

```
coffee_production.head()
```

total_production	Angola	Bolivia (Plurinational State of)	Brazil	...	Venezuela	Viet Nam	Yemen
1990	50.345	122.777	27285.6286		1122.477	1310.288	0.0
1991	79.331	103.536	27293.4934		940.704	1437.848	0.0
1992	77.52	120.235	34603.3542		1215.298	2340.447	0.0
1993	32.608	50.823	28166.9786		1332.881	3020.216	0.0
1994	76.802	116.944	28192.047		988.996	3631.609	0.0

```
coffee_production.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 29 entries, 1990 to 2018
Data columns (total 56 columns):
 #   Column                                Non-Null Count  Dtype
---  ------                                --------------  -----
 0   Angola                                29 non-null     object
 1   Bolivia (Plurinational State of)      29 non-null     object
 2   Brazil                                29 non-null     object
 3   Burundi                               29 non-null     object
 4   Ecuador                               29 non-null     object
 5   Indonesia                             29 non-null     object
 6   Madagascar                            29 non-null     object
 ...
 52  Uganda                                29 non-null     object
 53  Venezuela                             29 non-null     object
 54  Viet Nam                              29 non-null     object
 55  Yemen                                 29 non-null     object
dtypes: object(56)
memory usage: 12.9+ KB
```

PROJECT DATA: COFFEE IMPORTS

```
imports.head()
```

	imports	1990	1991	1992	1993		2011	2012	2013
0	Austria	1880.0	2058.0	2206.0	1836.0		1452.0	1559.0	1555.0
1	Belgium	NaN	NaN	NaN	NaN		5828.0	5668.0	5502.0
2	Belgium/Luxembourg	2015.0	1746.0	1828.0	2063.0	...	NaN	NaN	NaN
3	Bulgaria	268.0	200.0	182.0	397.0		482.0	560.0	609.0
4	Croatia	NaN	NaN	168.0	163.0		391.0	384.0	413.0

```
imports.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 147 entries, 0 to 146
Data columns (total 30 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   imports   147 non-null    object
 1   1990      138 non-null    float64
 2   1991      138 non-null    float64
 3   1992      144 non-null    float64
 4   1993      145 non-null    float64
 5   1994      145 non-null    float64
 ...
 27  2016      118 non-null    float64
 28  2017      118 non-null    float64
 29  2018      118 non-null    float64
dtypes: float64(29), object(1)
memory usage: 34.6+ KB
```

PROJECT DATA: COFFEE PRICES

```
prices.head()
```

	retail_prices	1990	1991	1992		2016	2017	2018
0	Austria	10.816777	10.088300	11.015453		12.450331	13.730684	14.635762
1	Cyprus	6.247241	6.181015	6.335541		11.699779	12.141280	12.781457
2	Denmark	8.410596	8.101545	8.366446	...	10.905077	11.103753	11.699779
3	Finland	6.578366	6.004415	5.430464		8.101545	9.050773	9.359823
4	France	8.233996	7.571744	5.099338		7.196468	7.505519	8.123620

```
prices.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14 entries, 0 to 13
Data columns (total 30 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   retail_prices   14 non-null     object
 1   1990            14 non-null     float64
 2   1991            14 non-null     float64
 3   1992            14 non-null     float64
 4   1993            14 non-null     float64
 5   1994            14 non-null     float64
 ...
 27  2016            14 non-null     float64
 28  2017            14 non-null     float64
 29  2018            14 non-null     float64
dtypes: float64(29), object(1)
memory usage: 3.4+ KB
```

MCG

ASSIGNMENT: MID-COURSE PROJECT

NEW MESSAGE

September 2024 ,7

From: **Sarah Shark** *(Managing Director)*

Subject: **Coffee Industry Deep Dive**

Hi there,

I'm starting to trust you… which is rare. We just got an inquiry from a major coffee trader looking to get an outside view on the coffee industry. They're particularly interested in Brazil's production relative to other nations.

We'll also look at a comparison of importer volume vs the prices they pay to understand if we can unlock margin by diversifying into new markets.

Do well on this and you'll be on promotion track.

section03_coffee_project_part1.ipynb

Reply Forward

Key Objectives

1. Read in data from multiple csv files
2. Reshape the data to prepare it for visualization
3. Build & customize charts to communicate the key insights to the client

MCG