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

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

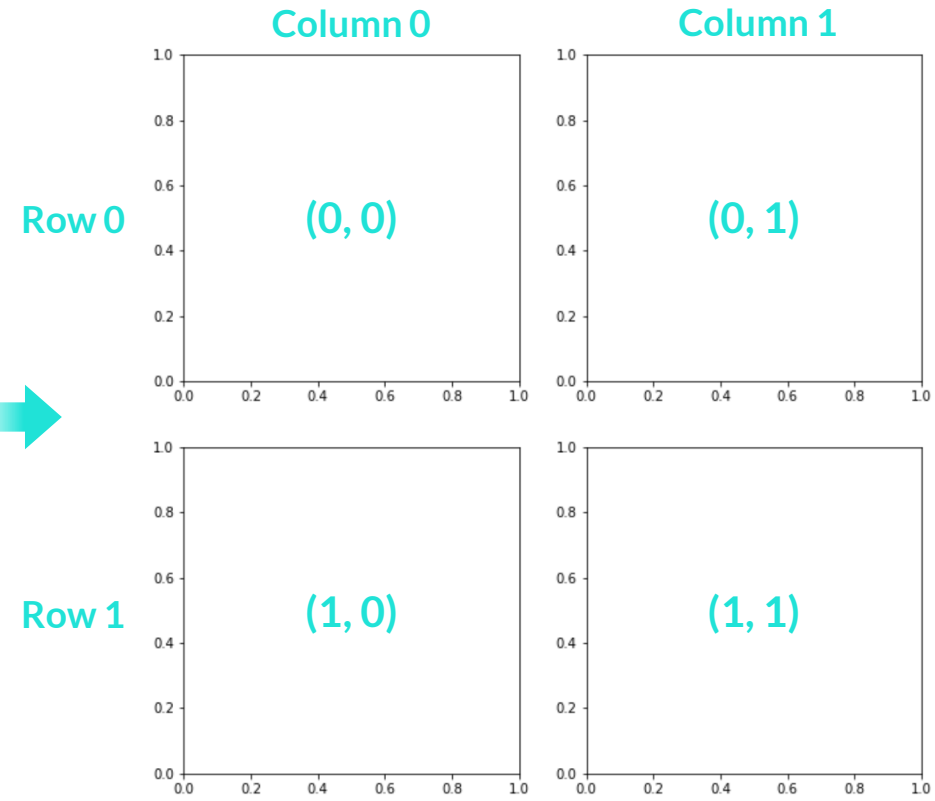
- **fig, ax = plt.subplots(rows, columns)** – this creates a grid with the specified rows & columns

```
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))
```

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



SUBPLOTS

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

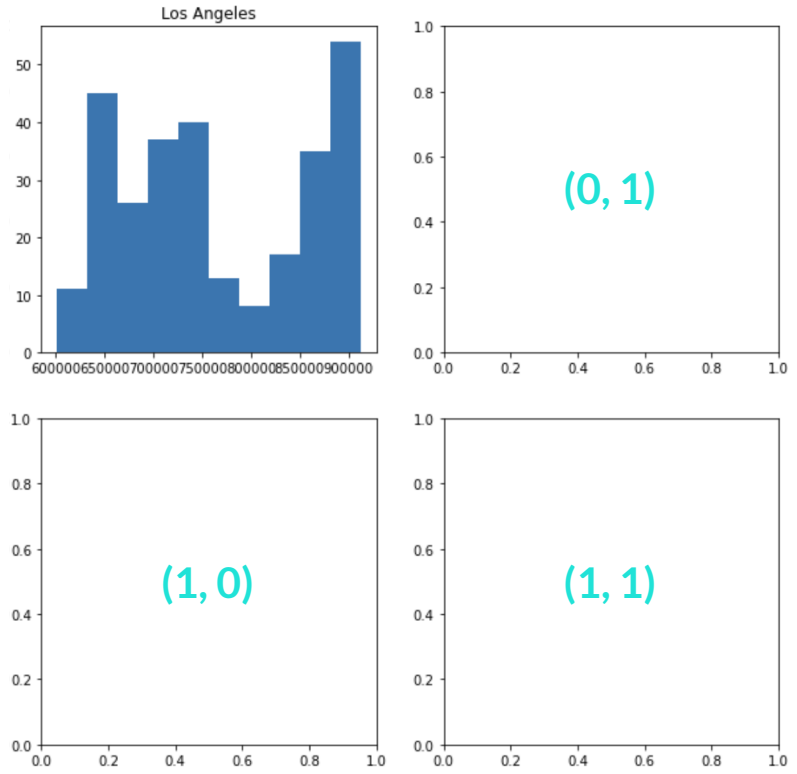
- **fig, ax = plt.subplots(rows, columns)** – this creates a grid with the specified rows & columns

```
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))

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

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



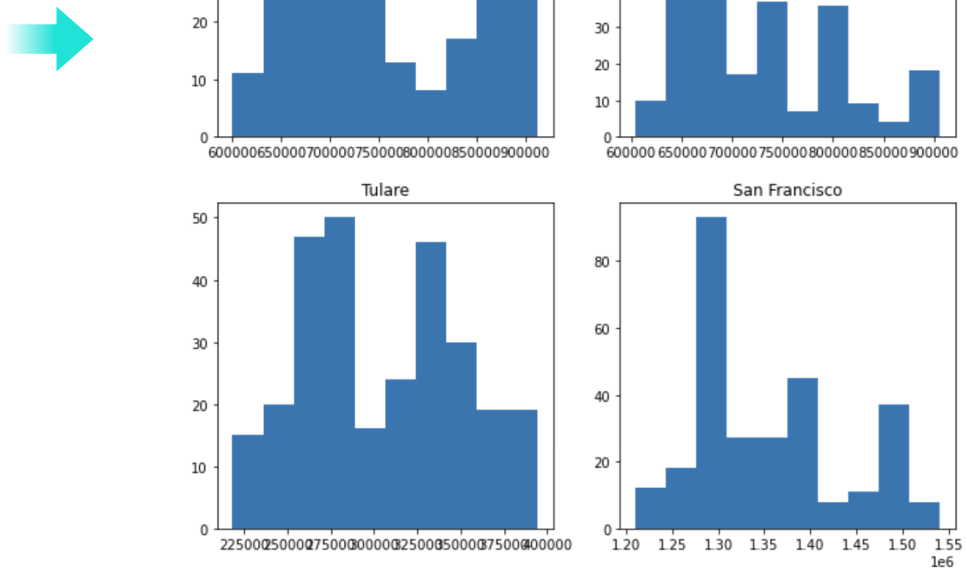
SUBPLOTS

- Subplots
- 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

```
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")
```

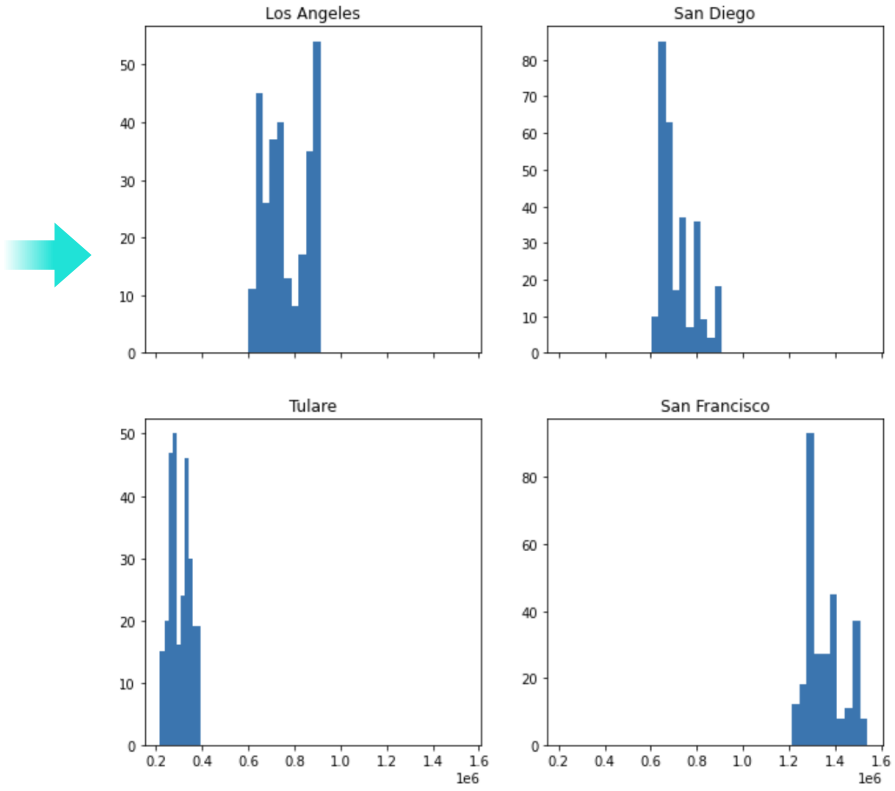


SUBPLOTS

Use the “sharex “& “sharey” arguments to **set the same axis limits** on all the plots

- This is set as “none” by default, but can be set to “all”, “row”, or “col”

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

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")
```

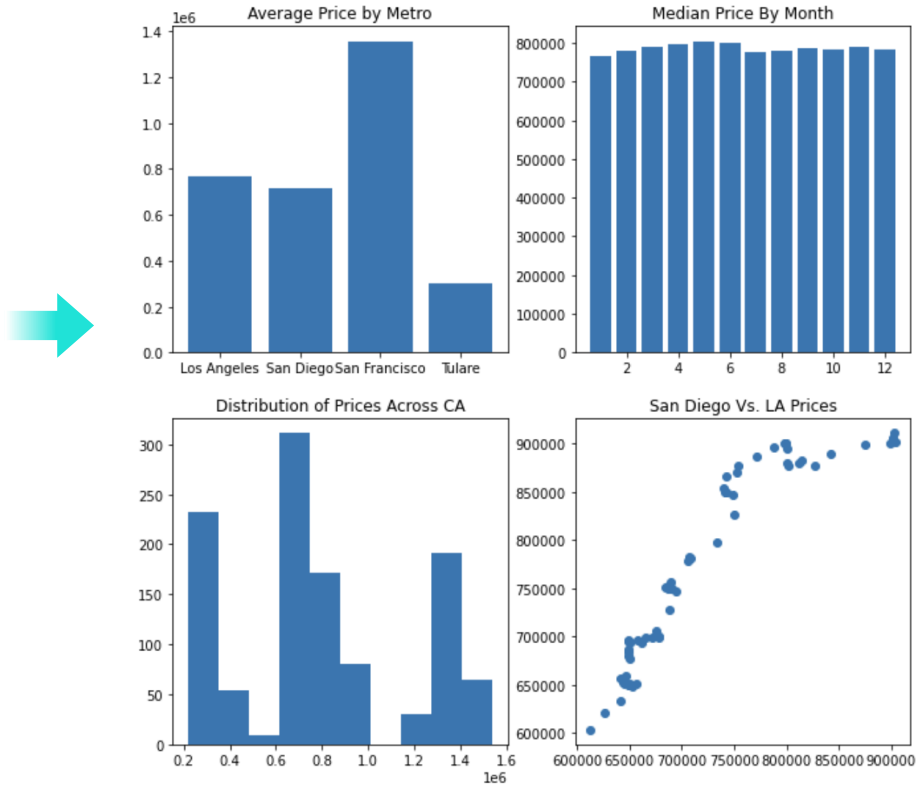


SUBPLOTS

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

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

```
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")
```



ASSIGNMENT: SUBPLOTS

1 **NEW MESSAGE**

September 2024 ,10

From: **Wendy Whiz** *(Data Scientist)*

Subject: **Deeper Exploration**

Hey there,

I want to get a quick read on the distribution of revenue by customer for our top 5 countries – I'm working on a model for a similar client and want to see if the distributions are similar.

Doesn't need to be polished, just need the 5 histograms in a single figure.

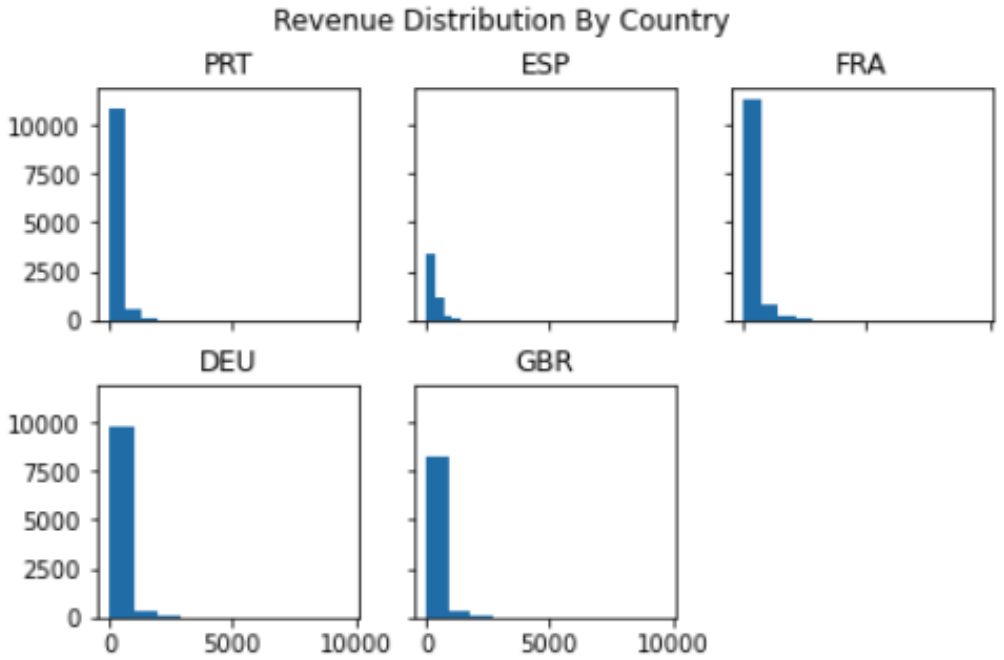
Thanks, and looking forward to working with you more!

Wendy

Section04_assignments.ipynb

Reply Forward

Results Preview



GRIDSPEC

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

You can build layouts with charts of varying sizes by setting a **gridspec** object

- This creates a grid with a specified number of rows & columns

```
import matplotlib.gridspec as gridspec

fig = plt.figure(figsize=(10, 10))
gs = gridspec.GridSpec(ncols=4, nrows=8)
```

	Column 0	Column 1	Column 2	Column 3
Row 0				
Row 1				
Row 2				
Row 3				
Row 4				
Row 5				
Row 6				
Row 7				

GRIDSPEC

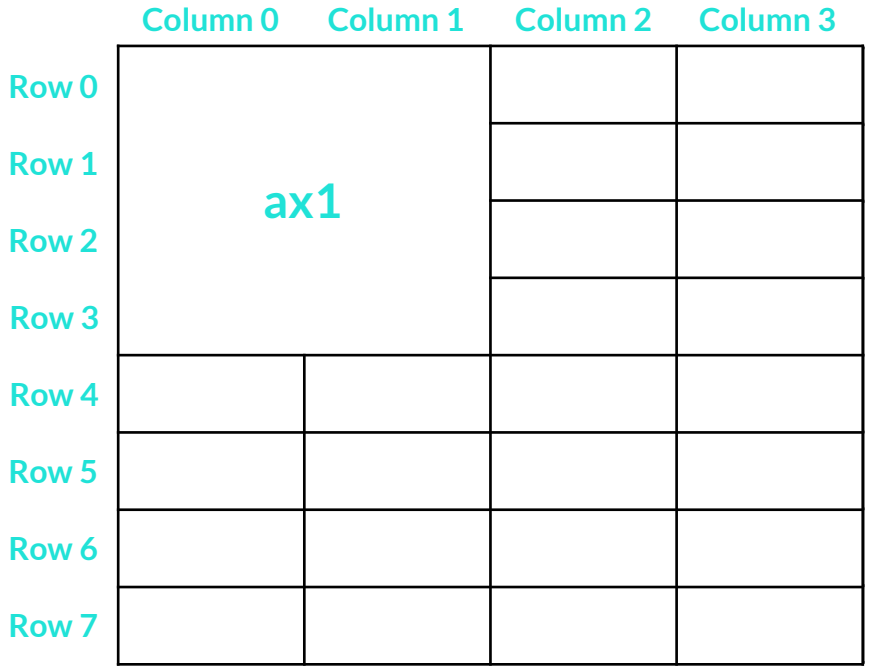
- Subplots
- GridSpec Layouts
- Colors
- Style Sheets
- rcParams
- Saving Figures

You can build layouts with charts of varying sizes by setting a **gridspec** object

- This creates a grid with a specified number of rows & columns
- Each axis, or chart, can then occupy a group of squares in the grid

```
import matplotlib.gridspec as gridspec

fig = plt.figure(figsize=(10, 10))
gs = gridspec.GridSpec(ncols=4, nrows=8)

ax1 = fig.add_subplot(gs[0:4, 0:2])
```

Use a slice to specify the ranges of rows and columns for each axis



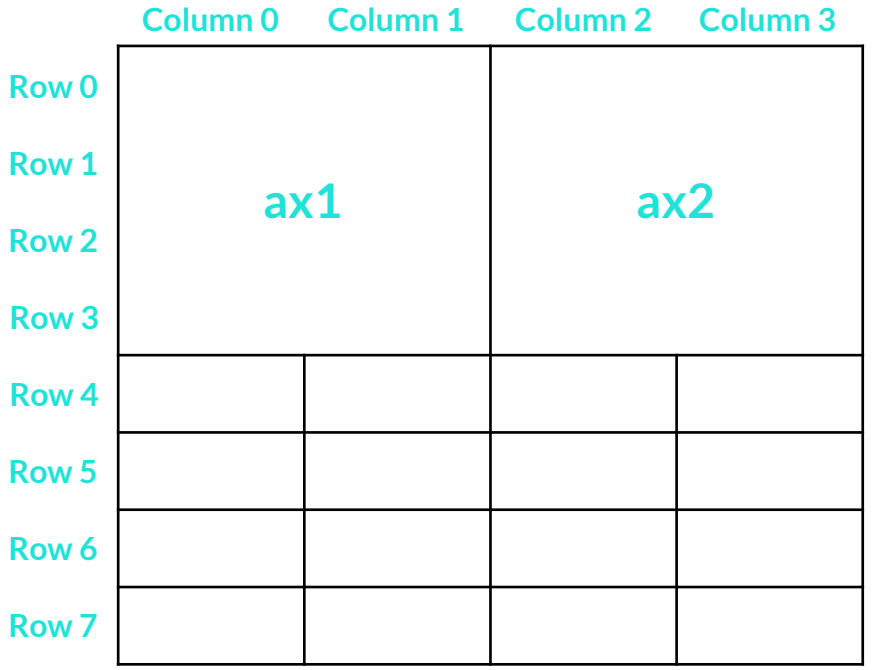
GRIDSPEC

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

You can build layouts with charts of varying sizes by setting a **gridspec** object

- This creates a grid with a specified number of rows & columns
- Each axis, or chart, can then occupy a group of squares in the grid

```
import matplotlib.gridspec as gridspec

fig = plt.figure(figsize=(10, 10))
gs = gridspec.GridSpec(ncols=4, nrows=8)

ax1 = fig.add_subplot(gs[0:4, 0:2])
ax2 = fig.add_subplot(gs[0:4, 2:4])
```



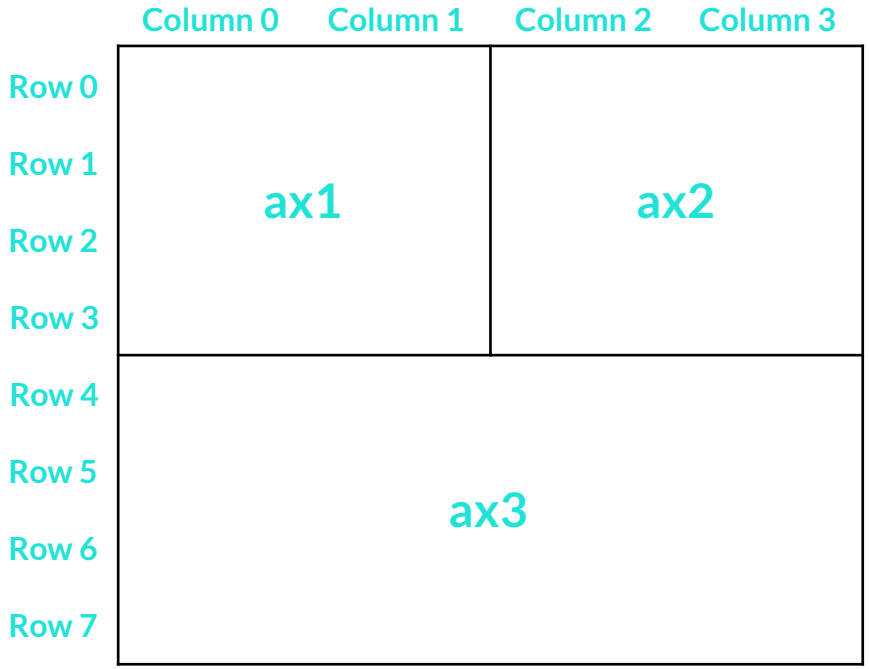
GRIDSPEC

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

You can build layouts with charts of varying sizes by setting a **gridspec** object

- This creates a grid with a specified number of rows & columns
- Each axis, or chart, can then occupy a group of squares in the grid

```
import matplotlib.gridspec as gridspec

fig = plt.figure(figsize=(10, 10))
gs = gridspec.GridSpec(ncols=4, nrows=8)

ax1 = fig.add_subplot(gs[0:4, 0:2])
ax2 = fig.add_subplot(gs[0:4, 2:4])
ax3 = fig.add_subplot(gs[4:, :])
```



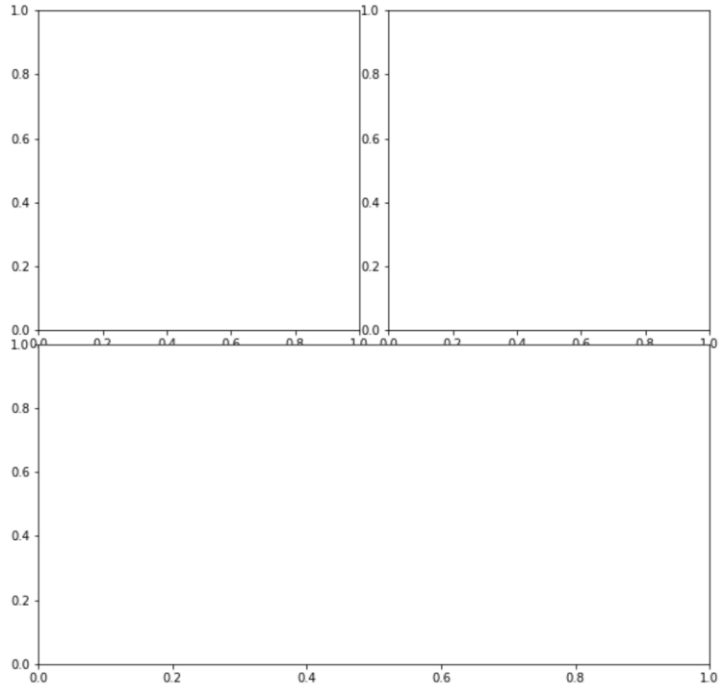
GRIDSPEC

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

You can build layouts with charts of varying sizes by setting a **gridspec** object

- This creates a grid with a specified number of rows & columns
- Each axis, or chart, can then occupy a group of squares in the grid

```
import matplotlib.gridspec as gridspec

fig = plt.figure(figsize=(10, 10))
gs = gridspec.GridSpec(ncols=4, nrows=8)

ax1 = fig.add_subplot(gs[0:4, 0:2])
ax2 = fig.add_subplot(gs[0:4, 2:4])
ax3 = fig.add_subplot(gs[4:, :])
```

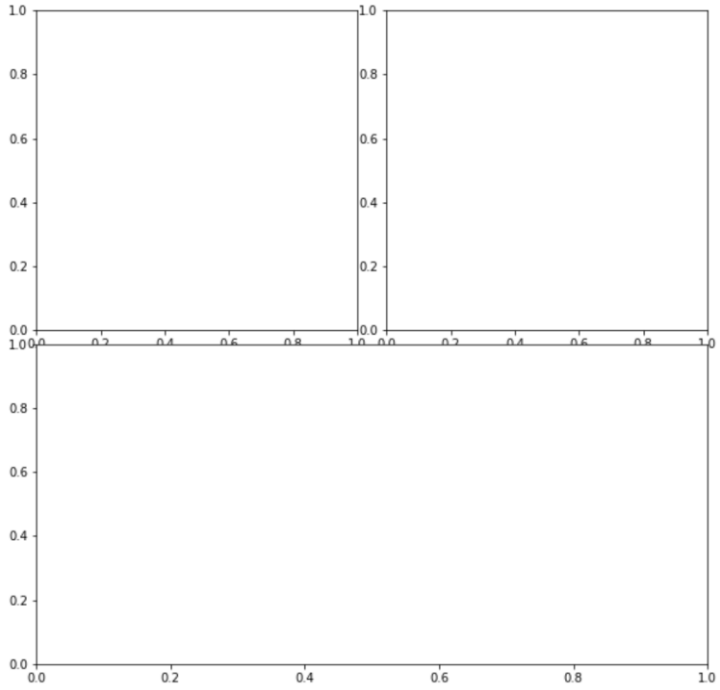



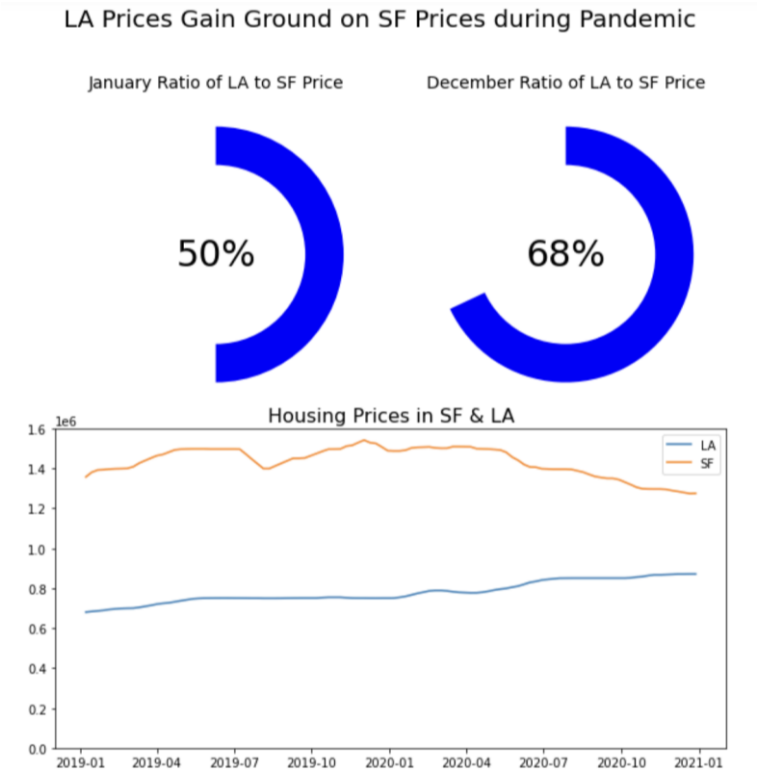
GRIDSPEC

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

You can build layouts with charts of varying sizes by setting a **gridspec** object

- This creates a grid with a specified number of rows & columns
- Each axis, or chart, can then occupy a group of squares in the grid





ASSIGNMENT: GRIDSPEC

NEW MESSAGE

September 2024 ,12

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

Subject: **Revenue Report Format**

Hi there,

Big meeting with our hotel client coming up – we want to propose a report format that will help track their revenue, specifically with respect to their goal to get French customers to surpass German customers.

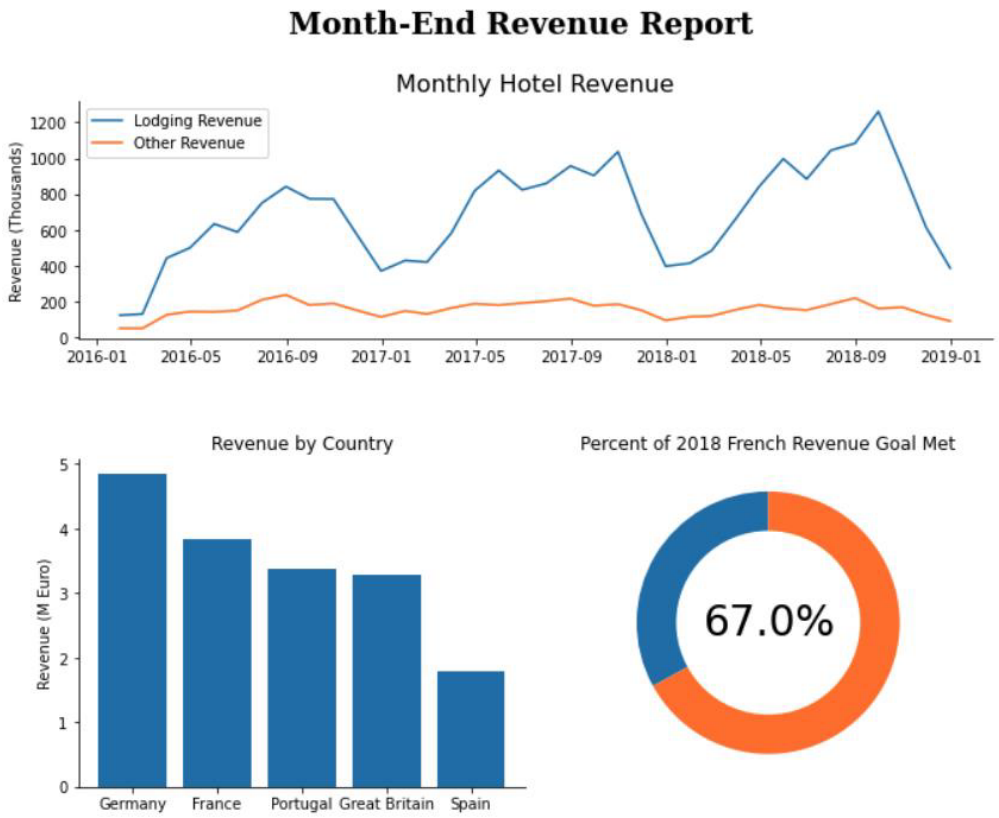
Can you create a figure with a line chart tracking revenue by category, a bar chart with revenue for the top 5 countries, and a chart indicating progress towards our French revenue goal?

Thanks!

section04_assignments.ipynb

Reply Forward

Results Preview



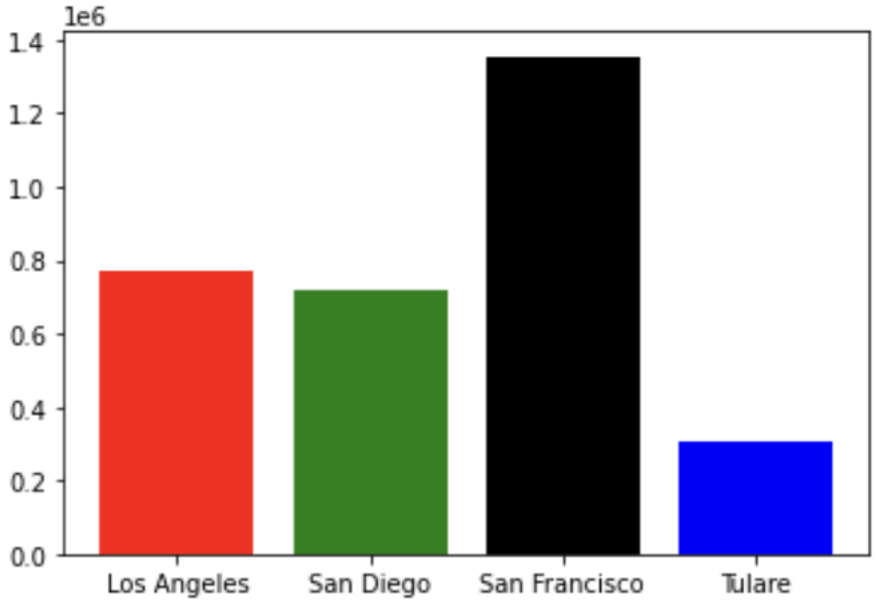
COLORS

You can pass **colors** to a plot by assigning them to a list

	Mean Price
region_name	
Los Angeles	767900.0
San Diego	718008.0
San Francisco	1355496.0
Tulare	303900.0

```
colors = ["red", "green", "black", "blue"]

fig, ax = plt.subplots()

ax.bar(means.index, means.values, color=colors)
```

This assigns each color in the list to each bar in the plot



COLORS

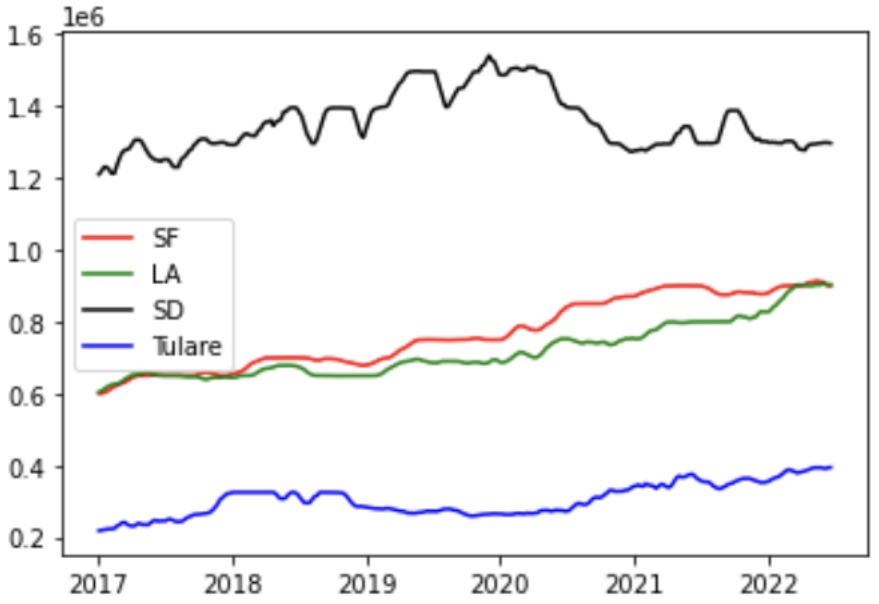
- Subplots
- GridSpec Layouts
- Colors
- Style Sheets
- rcParams
- Saving Figures

You can also loop through a list of colors to pass them to separate series in a plot

```
ca_housing_pivot.head(3)
```

region_name	Los Angeles	San Diego	San Francisco	Tulare
period_begin				
2017-01-02	600558.0	603987.5	1210000.0	218237.5
2017-01-09	600558.0	607487.5	1218250.0	219606.2
2017-01-16	601808.0	612462.5	1230556.2	220975.0

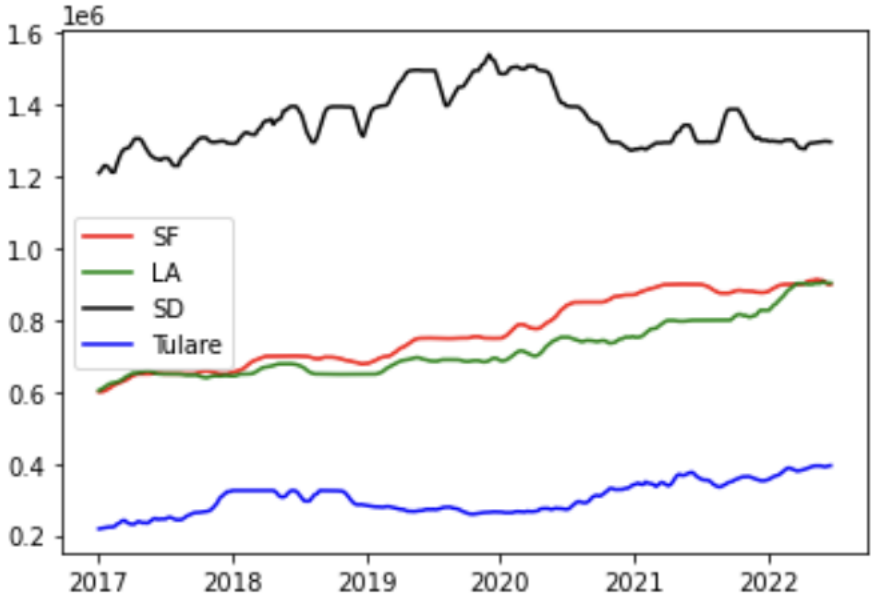
```
fig, ax = plt.subplots()

colors = ["red", "green", "black", "blue"]

for i, color in enumerate(colors):
    ax.plot(
        ca_housing_pivot.index,
        ca_housing_pivot.iloc[:, i],
        c=color,
        label=ca_housing.columns[i]
    )

ax.legend()
```



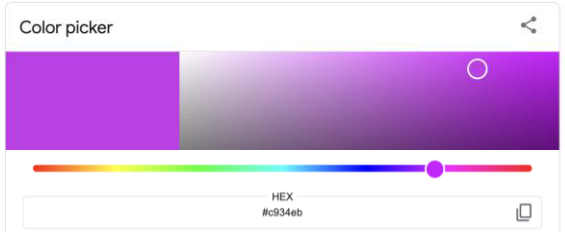
COLORS

Hex codes can be used to supply specific color pantones

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

colors = ["#FF2D00", "#178C36", "#000000", "#1A17D0"]

for i, color in enumerate(colors):
    ax.plot(
        ca_housing_pivot.index,
        ca_housing_pivot.iloc[:, i],
        c=color,
        label=ca_housing.columns[i]
    )

ax.legend()
```



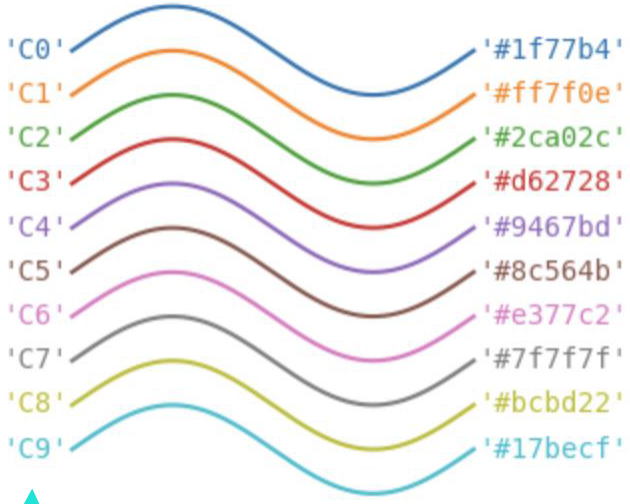
PRO TIP: Sites like Google have helpful hexadecimal color pickers



PRO TIP: COLOR PALETTES

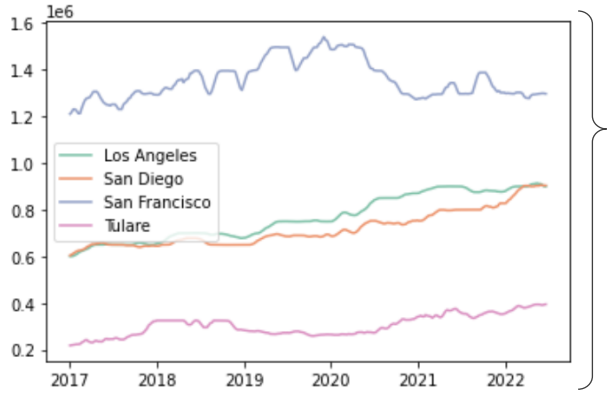
You can also modify the entire **color palette** for the series in a plot

Default Color Map:



Series colors are applied in this sequential order (at +10 series, the cycle repeats)

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

plt.rcParams["axes.prop_cycle"] = plt.cycler("color", plt.cm.Set2.colors)

for i in range(4):
    ax.plot(ca_housing_pivot.index, ca_housing_pivot.iloc[:, i] )

ax.legend(ca_housing_pivot.columns)

plt.show()
```



The "Set2" color map is applied here

rcParams are the underlying settings for Matplotlib charts and can be modified to gain a high level of customization (more on these soon!)

For more on color palettes, visit: ***https://matplotlib.org/3.5.0/tutorials/colors/colormaps.html***

ASSIGNMENT: COLORS

1 **NEW MESSAGE**

September 2024 ,13

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

Subject: **Re: Revenue Report Format**

Hi again,

Love the layout, HATE the colors! Let's show some polish by getting away from the defaults.

Apply the "Set2" colormap to the line chart and look up the national color hex codes for the top 5 countries to use them for the rest of the charts.

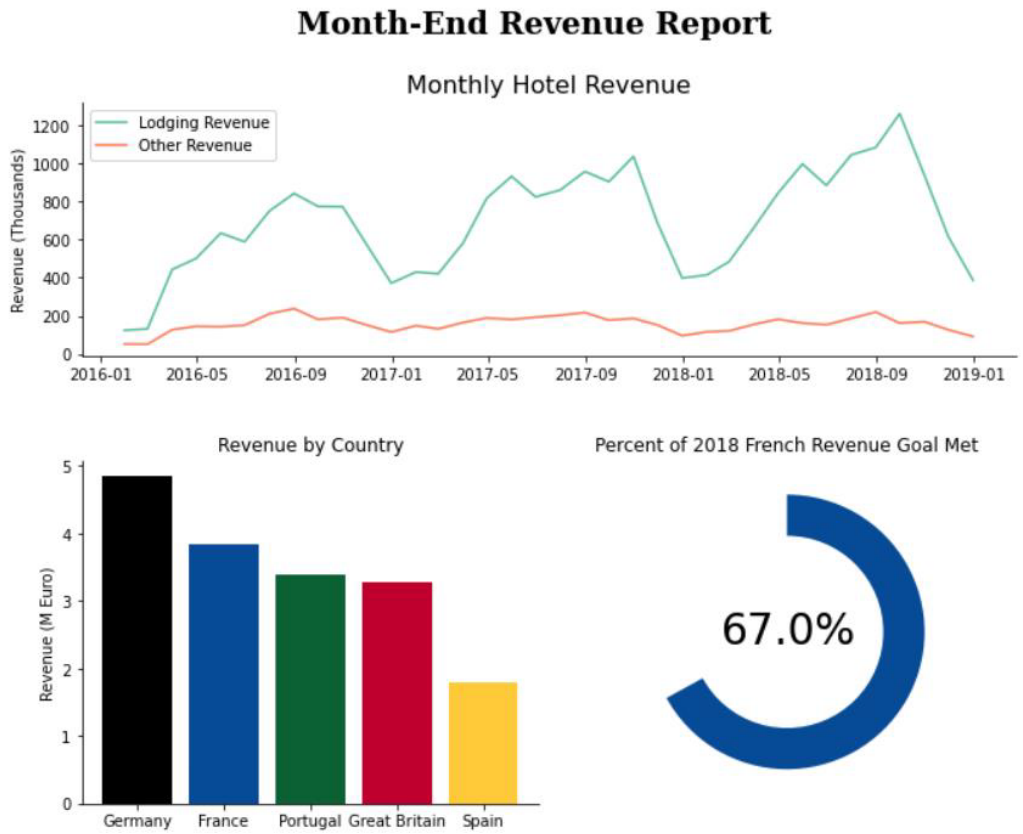
Thanks,

Sarah

section04_assignments.ipynb

Reply Forward

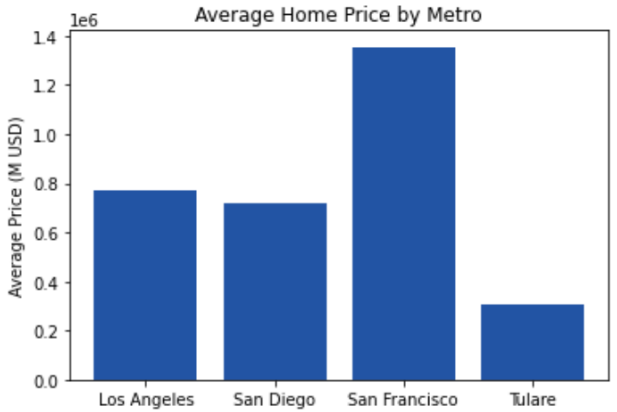
Results Preview



STYLE SHEETS

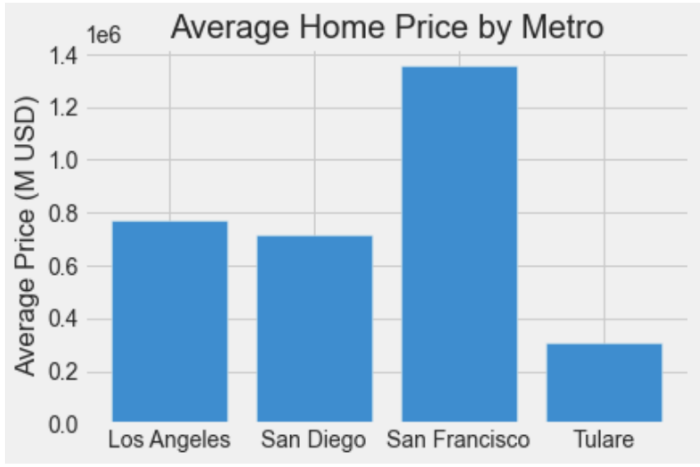
Matplotlib (and Seaborn) have **style sheets** that can be used instead of the default


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

ax.set_title("Average Home Price by Metro")
ax.set_ylabel("Average Price (M USD)")

ax.bar(means.index, means.values)
```




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

fig, ax = plt.subplots()

ax.set_title("Average Home Price by Metro")
ax.set_ylabel("Average Price (M USD)")

ax.bar(means.index, means.values)
```

The style is set in advance



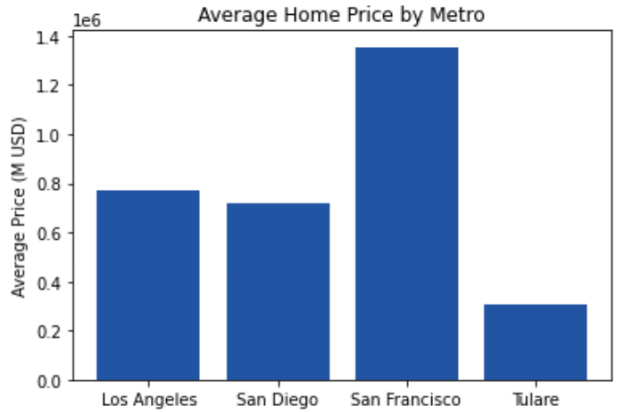
The "fivethirtyeight" style has larger font sizing, and adds gridlines and a background color

STYLE SHEETS

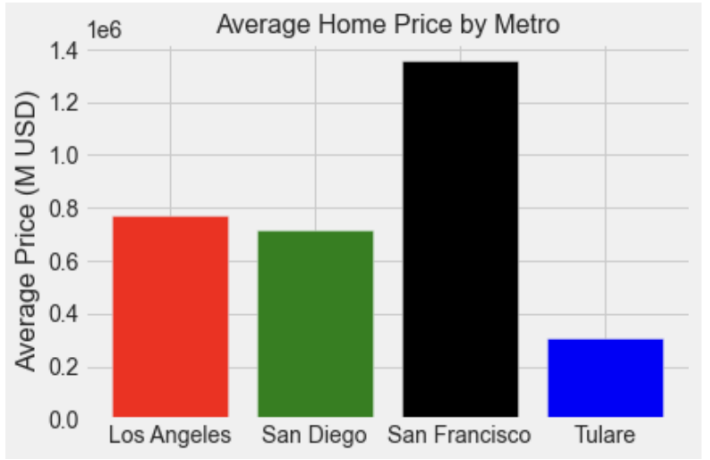
Matplotlib (and Seaborn) have **style sheets** that can be used instead of the default

- You can still customize individual formatting options after setting a style

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

ax.set_title("Average Home Price by Metro")
ax.set_ylabel("Average Price (M USD)")

ax.bar(means.index, means.values)
```




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

fig, ax = plt.subplots()

ax.set_title("Average Home Price by Metro", fontsize=16)
ax.set_ylabel("Average Price (M USD)")

colors = ["red", "green", "black", "blue"]

ax.bar(means.index, means.values, color=colors)
```

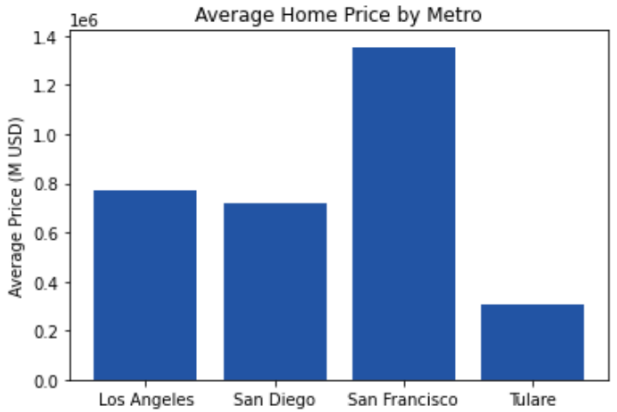


STYLE SHEETS

Matplotlib (and Seaborn) have **style sheets** that can be used instead of the default

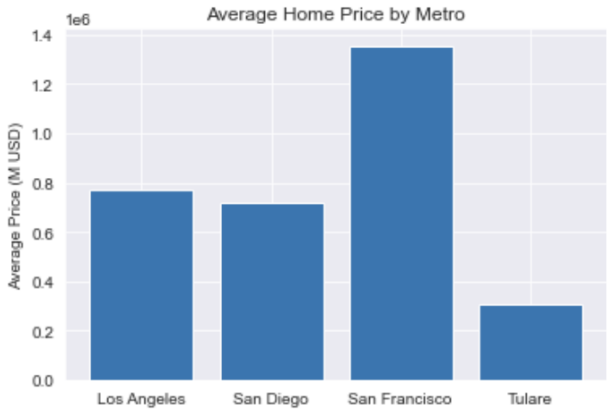
- You can still customize individual formatting options after setting a style

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

ax.set_title("Average Home Price by Metro")
ax.set_ylabel("Average Price (M USD)")

ax.bar(means.index, means.values)
```




```
import seaborn as sns

sns.set_style("darkgrid")

fig, ax = plt.subplots()

ax.set_title("Average Home Price by Metro")
ax.set_ylabel("Average Price (M USD)")

ax.bar(means.index, means.values)
```

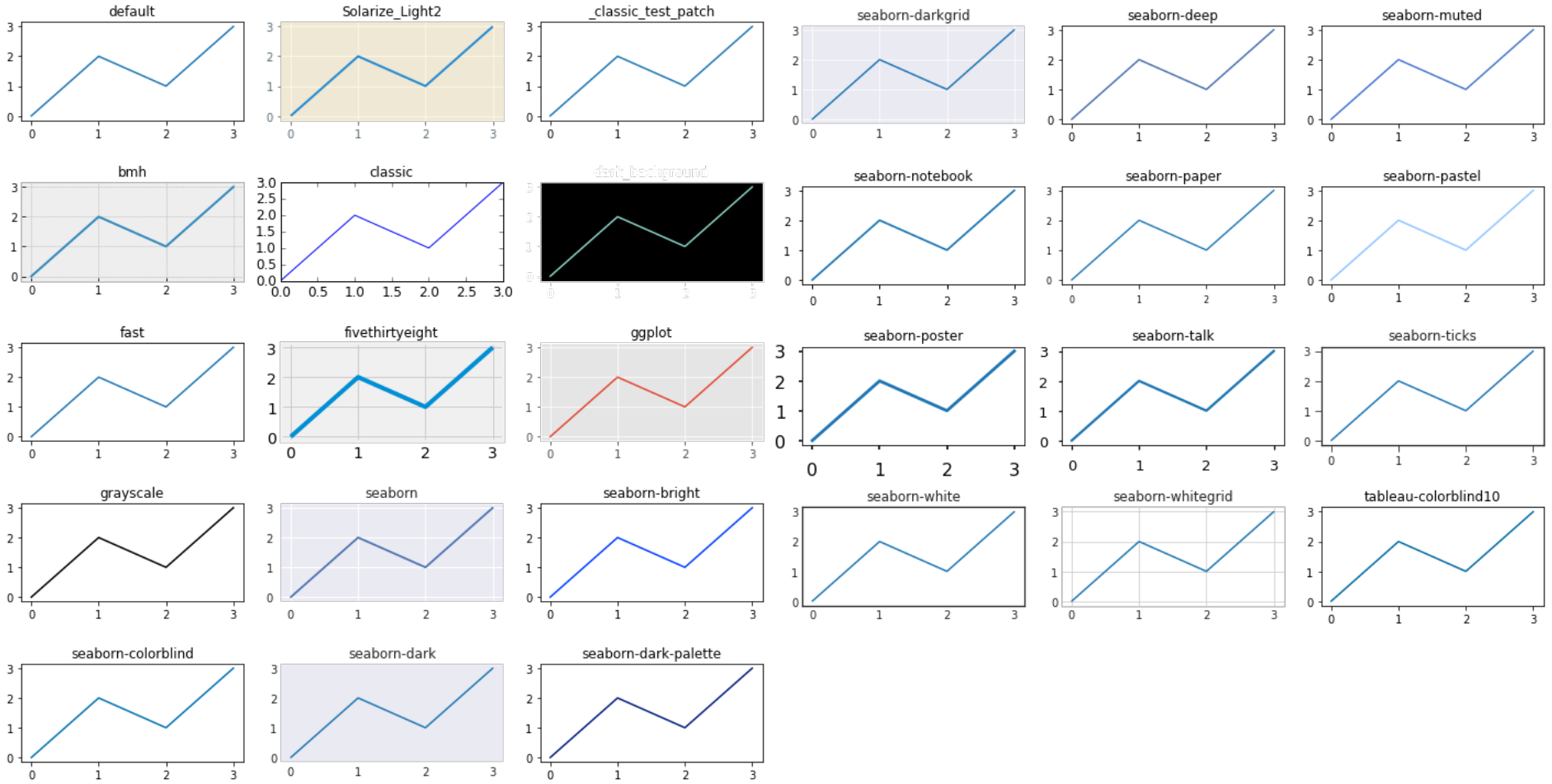
The Seaborn library has additional styles that can be used with Matplotlib charts, like "darkgrid"



ADDITIONAL STYLES

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

These are some of the **additional styles** available in both libraries:



ASSIGNMENT: STYLE SHEETS

NEW MESSAGE

September 2024 ,14

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

Subject: **Re: Re: Revenue Report Format**

Hi,

Layout and colors look great now, but can we spruce up the chart styling?

Use a style sheet of your choice.

Once we've done that it should be ready to ship.

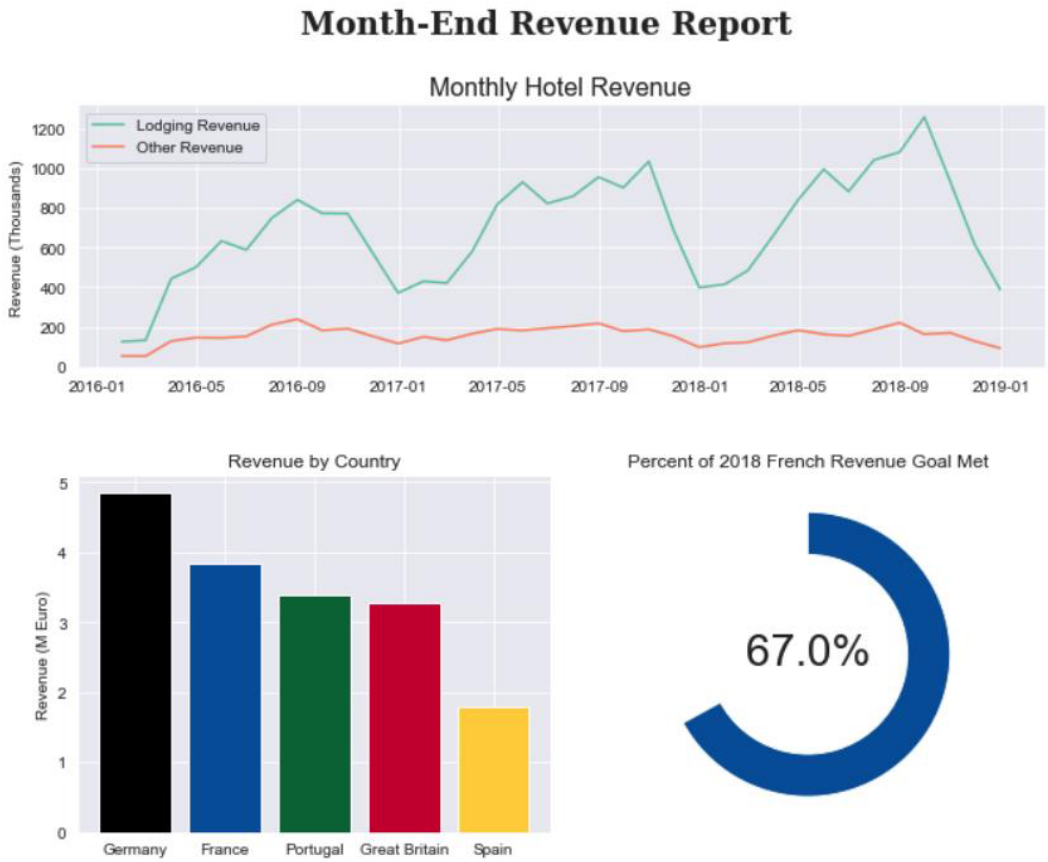
Thx

-S

section04_assignments.ipynb

Reply Forward

Results Preview

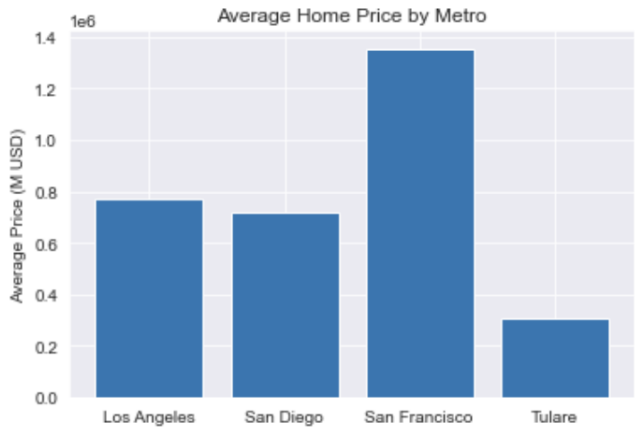


STYLE PARAMETERS

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

Viewing the **parameters** of a style sheet can help format charts properly and provide inspiration for your own formatting changes

```
import seaborn as sns

sns.set_style("darkgrid")

fig, ax = plt.subplots()

ax.set_title("Average Home Price by Metro")
ax.set_ylabel("Average Price (M USD)")

ax.bar(means.index, means.values)
```




```
plt.style.library['seaborn-darkgrid']
```

```
RcParams({'axes.axisbelow': True,
          'axes.edgecolor': 'white',
          'axes.facecolor': '#EAEAF2',
          'axes.grid': True,
          'axes.labelcolor': '.15',
          'axes.linewidth': 0.0,
          'figure.facecolor': 'white',
          'font.family': ['sans-serif'],
          'font.sans-serif': ['Arial',
                              'Liberation Sans',
                              'DejaVu Sans',
                                        'Vera Sans',
                               sans-serif'],
          'grid.color': 'white',
          'grid.linestyle': '-',
          'image.cmap': 'Greys',
          'legend.frameon': False,
          'legend.numpoints': 1,
          'legend.scatterpoints': 1,
          'lines.solid_capstyle': <CapStyle.round: 'round'>,
          'text.color': '.15',
          'xtick.color': '.15',
          'xtick.direction': 'out',
          'xtick.major.size': 0.0,
          'xtick.minor.size': 0.0,
          'ytick.color': '.15',
          'ytick.direction': 'out',
          'ytick.major.size': 0.0,
          'ytick.minor.size': 0.0})
```

PARAMETER GROUPS

There are +300 parameters that can be modified, which fall into **parameter groups**:

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

Parameter Groups	Description	Parameter Examples
axes	Chart-level formatting	axes.spine.top = False, axis.titlesize='Large'
date	Date formatting options	date.autoformatter.month = %Y-%m
figure	Figure-level formatting	figure.figsize = (8.5, 11), figure.facecolor="grey"
font	Font settings	font.size = 16, font.style='helvetica', font.weight='bold'
grid	Gridline settings	grid.linestyle = ':', grid.linewidth = 2
legend	Legend settings	legend.loc = 'lower right', legend.frameon=False
savefig	Saved figure Settings	savefig.dpi = 1000, savefig.format = 'png'
text	Text settings	text.color = 'grey', text.usetex = True
xtick/ytick	X and Y tick settings	xtick.labelcolor='green', ytick.minor.visible = True
boxplot	Settings for boxplots	boxplot.whiskerprops.color = 'orange'
hist	Settings for histograms	hist.bins = 20
lines	Settings for line charts	lines.linewidth = 2, lines.color = 'red',
scatter	Settings for scatterplots	scatter.marker = "+"

For more on rcParams, visit: ***https://matplotlib.org/stable/api/matplotlib_configuration_api.html***

MODIFYING PARAMETERS

There are two ways to **modify parameters**:

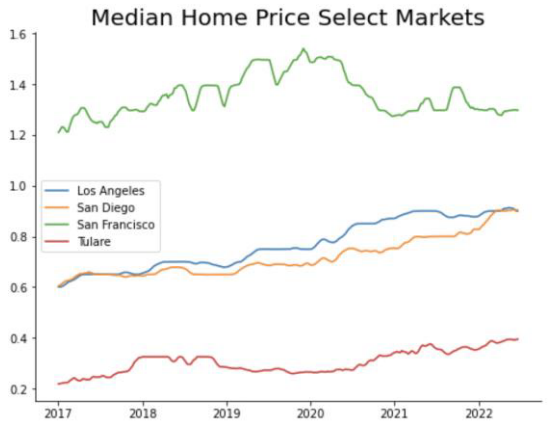
1. You can change individual parameters via **assignment**
2. You can change multiple parameters from the same group with the **rc() function**

```
plt.rc('axes.spines', right=False, top=False)
plt.rc('axes', titlesize=20)
plt.rcParams["figure.figsize"] = (8, 6)
```

Turn off top and right spines

Change default axes title size to 20 Modify figure size to 8"x 6"

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

for col in ca_housing_pivot.columns:
    ax.plot(
        ca_housing_pivot.index,
        ca_housing_pivot.loc[:, col] / 1000000,
        label=col
    )

ax.set_title("Median Home Price Select Markets")

ax.legend()
```



PRO TIP: Modify parameters to avoid having to repeat the same formatting options on each chart

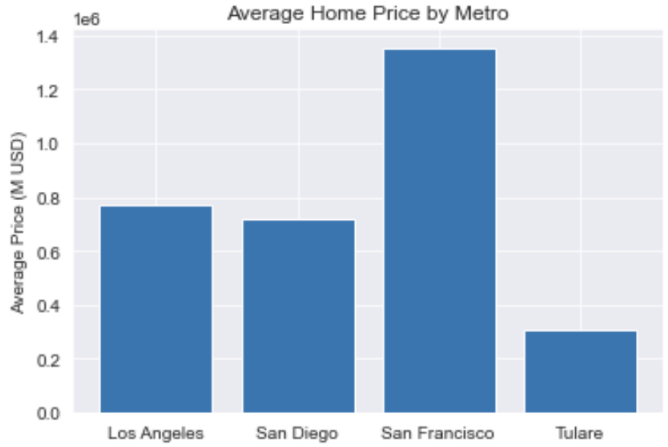
SAVING FIGURES

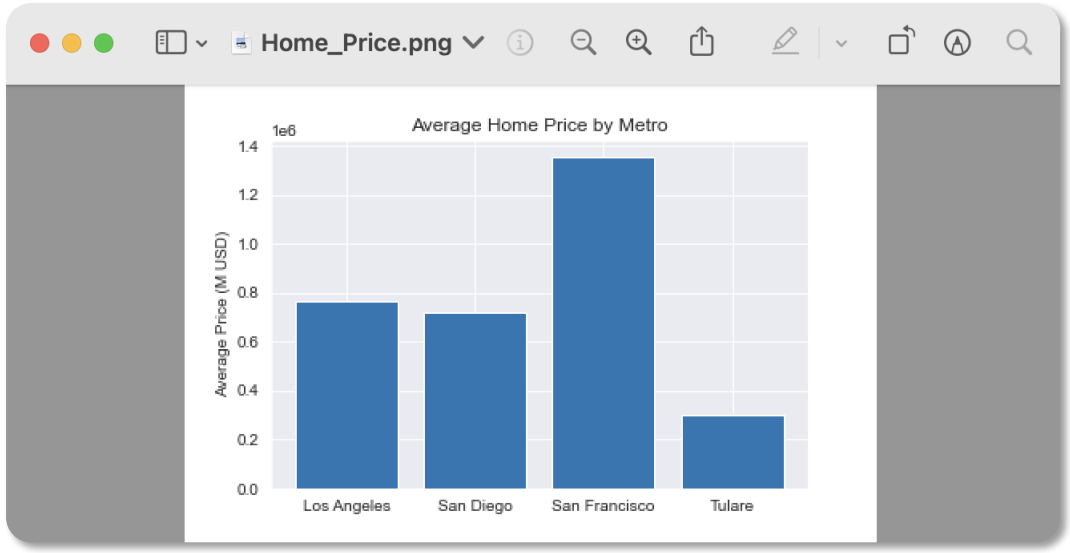
- Subplots
- GridSpec Layouts
- Colors
- Style Sheets
- rcParams
- Saving Figures

The savefig() function will **save figures** as an image file

- Simply specify the desired filename and format

```
import seaborn as sns

sns.set_style("darkgrid")

fig, ax = plt.subplots()

ax.set_title("Average Home Price by Metro")
ax.set_ylabel("Average Price (M USD)")

ax.bar(means.index, means.values)

plt.savefig("Home_Price")
```





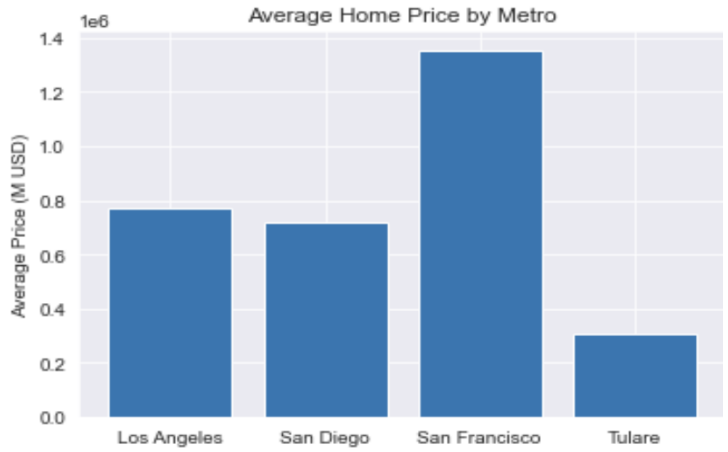
Screenshotting the images with your operating system's snipping tool will often be sufficient for building plots into presentations like this course ;).

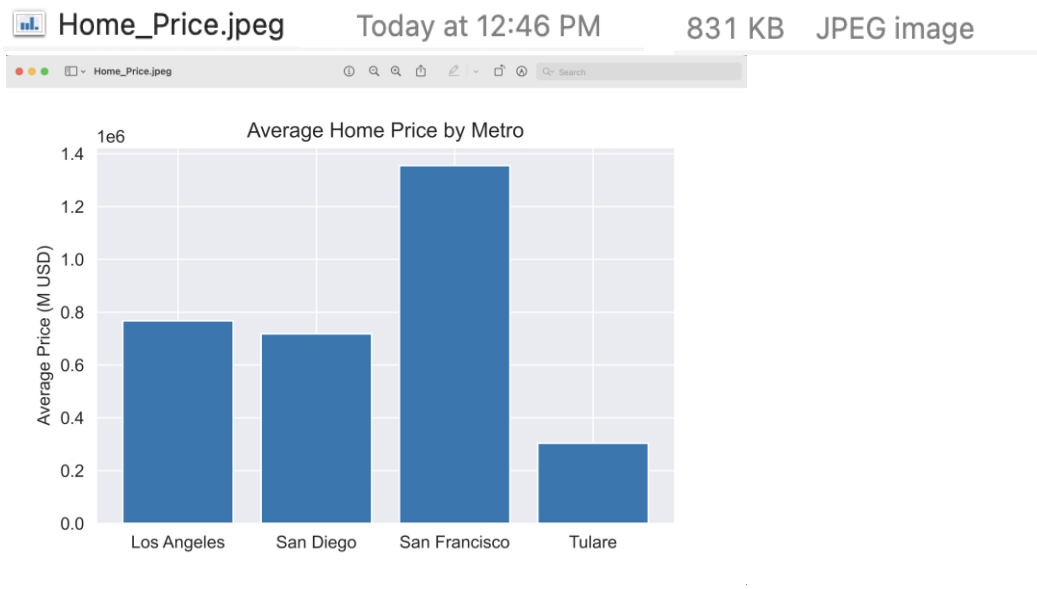
SAVING FIGURES

The savefig() function will **save figures** as an image file

- Simply specify the desired filename and format

```
import seaborn as sns

sns.set_style("darkgrid")

fig, ax = plt.subplots()

ax.set_title("Average Home Price by Metro")
ax.set_ylabel("Average Price (M USD)")

ax.bar(means.index, means.values)

plt.savefig("Home_Price.jpeg", dpi=1000)
```





If no extension in the filename is specified, the file will be saved as a .png. Most systems support .jpg, .jpeg, .svg, and .pdf, among others. The default resolution is 100dpi (pixels per inch)

KEY TAKEAWAYS

Subplots and GridSpec allow us to create multi-chart figures

- *Subplots are equally sized grids, GridSpec allows for custom layouts*

Colors can be set by specifying a colormap or by assigning colors to the data of interest

- *Common color names and hex codes can be used to assign colors to your data*

Set a style to spruce up the default aesthetics, or use rcParams to completely customize your charts

- *Pre-built styles can add some nice aesthetic polish compared to the matplotlib defaults*
- *Understanding how to modify rcParams will allow you full control over chart customization, and reduce the need for manual formatting*

PROJECT 2: CUSTOM LAYOUTS

PROJECT DATA: OVERVIEW

Coffee Production

total_production	Angola	Bolivia (Plurinational State of)	Brazil	Burundi	Ecuador	Indonesia	Madagascar	...	Uganda	Venezuela	Viet Nam	Yemen
1990	50.345	122.777	27285.6286	487.393	1503.815	7441.383	982.447		1955.009	1122.477	1310.288	0.0
1991	79.331	103.536	27293.4934	667.199	2123.824	8493.196	932.513		2088.001	940.704	1437.848	0.0
1992	77.52	120.235	34603.3542	620.238	1185.48	5569.478	1121.684		2185.074	1215.298	2340.447	0.0
1993	32.608	50.823	28166.9786	393.354	2069.007	6743.288	441.859		3141.706	1332.881	3020.216	0.0
1994	76.802	116.944	28192.047	664.143	2375.766	5367.878	641.372		2392.753	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
```

MCG

PROJECT DATA: OVERVIEW

Prices Paid To Growers

```
prices_paid_to_growers.head()
```

prices_paid_to_growers	Colombia	Dominican Republic	El Salvador	Guatemala	Honduras	India	Uganda	Brazil	Ethiopia	India	Togo	Uganda	Other Nations
1990	1.534724	1.458168	1.116194	1.204956	1.11147	1.473558	0.337598	1.199223	1.348565	0.978921	0.645267	0.166486	0.943624
1991	1.48179	1.382845	0.983322	1.270086	1.238947	1.358371	0.654322	0.97115	1.505322	0.897289	0.632307	0.26143	0.964325
1992	1.204656	1.027841	0.682322	0.888099	0.886057	1.191159	0.441397	0.997768	1.351128	0.877945	0.658494	0.197653	0.761219
1993	1.106477	1.172704	0.780397	0.914552	0.828746	1.278669	0.552298	1.167263	1.362442	0.975912	0.499857	0.259737	0.806986
1994	1.898327	2.478234	2.191177	1.662711	1.800576	1.73081	1.666651	2.52911	2.418234	1.246437	0.573784	0.919709	1.585565

```
prices_paid_to_growers.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 29 entries, 1990 to 2018
Data columns (total 13 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Colombia            29 non-null     object
 1   Dominican Republic  29 non-null     object
 2   El Salvador         29 non-null     object
 3   Guatemala           29 non-null     object
 4   Honduras            29 non-null     object
 5   India               29 non-null     object
 6   Uganda              29 non-null     object
 7   Brazil              29 non-null     object
 8   Ethiopia            29 non-null     object
 9   India               29 non-null     object
 10  Togo                29 non-null     object
 11  Uganda              29 non-null     object
 12  Other Nations       29 non-null     float64
dtypes: float64(1), object(12)
memory usage: 3.2+ KB
```

ASSIGNMENT: MID-COURSE PROJECT

NEW MESSAGE
September 2024 ,18

From: **Clarissa Café** *(Coffee Client)*

Subject: **Summary Report**

Hi there,

Sarah told me to reach out directly to you – we loved the work you did on breaking down the industry, but we want to summarize your findings on Brazil into a single figure we can pass around.

Can you combine your findings into a single figure report? We'll also want to modify colors. There are more details in the attached notebook.

Thanks!
Clarissa

section05_coffee_project_part2.ipynb

Reply Forward

Key Objectives

1. Read in data from multiple csv files
2. Reshape the data with Pandas to set up charts
3. Build and customize line charts, bar charts, histograms and more to communicate key insights to our client
4. Modify chart colors to represent national flags
5. Combine modified charts into a single report by leveraging meshgrid and subplots

MCG

DATA VIZ WITH SEABORN

DATA VISUALIZATION WITH SEABORN

In this section we'll cover **data visualization with Seaborn**, another Python library that introduces new chart types and layouts, and interacts well with Matplotlib

TOPICS WE'LL COVER:

- Seaborn Basics
- Chart Formatting
- Chart Types
- FacetGrid Layouts
- Matplotlib Integration

GOALS FOR THIS SECTION:

- Introduce the basics of plotting data with Seaborn
- Build variations of Matplotlib charts like bar charts and histograms, as well as new visuals like boxplots, violin plots, and linear relationship plots
- Create FacetGrid layouts as an alternative to subplots
- Integrate Seaborn plots with Matplotlib objects to get the best of both worlds

MEET SEABORN

- Seaborn Basics
- Chart Formatting
- Chart Types
- FacetGrid Layouts
- Matplotlib Integration

seaborn

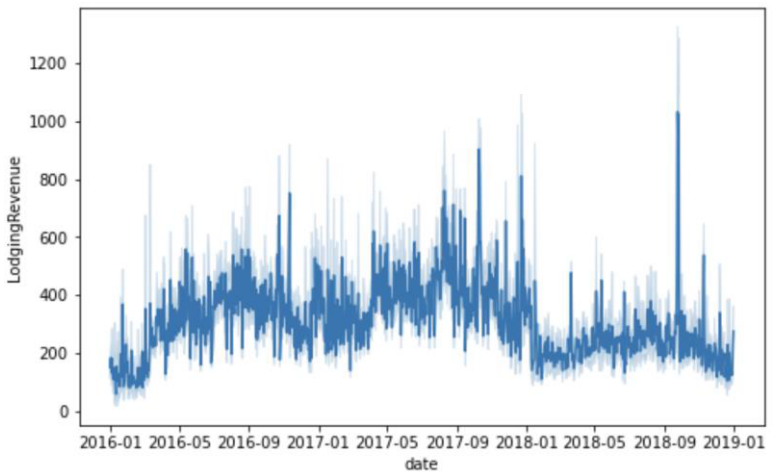
Seaborn is a Python library for built for easily visualizing Pandas DataFrames, taking away some of the "drawing" required when using Matplotlib

You simply need to specify a DataFrame as the "data" argument and set columns as the "x" and "y" axes

Seaborn will automatically aggregate the results!

'sns' is the standard alias for Seaborn

```
import seaborn as sns

sns.lineplot(
    x="date",
    y="LodgingRevenue",
    data=hotels,
)
```



MEET SEABORN

- Seaborn Basics
- Chart Formatting
- Chart Types
- FacetGrid Layouts
- Matplotlib Integration

seaborn

Seaborn is a Python library for built for easily visualizing Pandas DataFrames, taking away some of the "drawing" required when using Matplotlib

```
import seaborn as sns

sns.lineplot(
    x="date",
    y="LodgingRevenue",
    data=hotels,
    estimator=sum,
    ci=None
)
```

You can change the aggregation method and suppress the confidence intervals

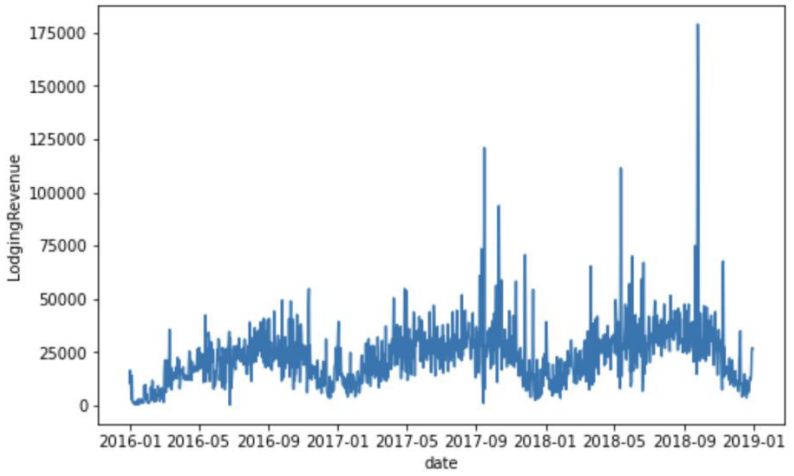
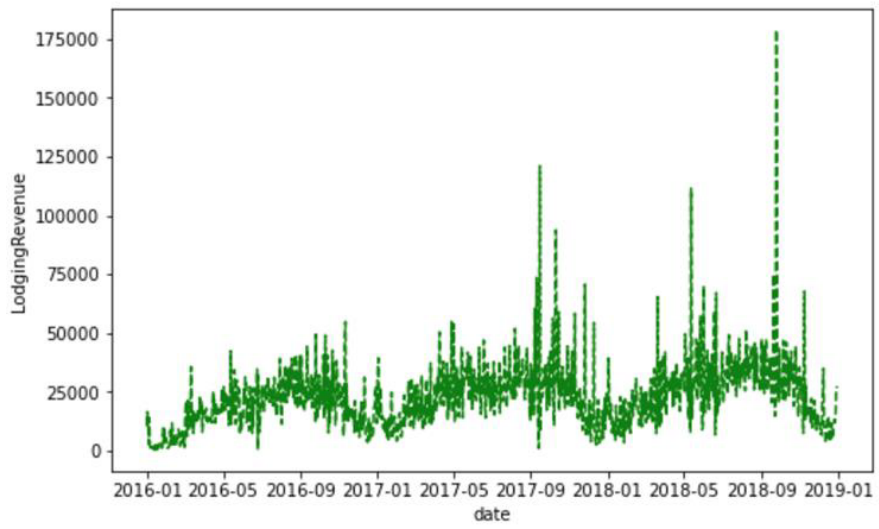


CHART FORMATTING

You can apply **chart formatting** to Seaborn plots using Matplotlib arguments

- These are passed to the Matplotlib object that Seaborn creates internally

```
import seaborn as sns

sns.lineplot(
    x="date",
    y="LodgingRevenue",
    data=hotels,
    estimator=sum,
    ci=None,
    ls='--',
    color='green'
)
```



We'll cover **integration with Matplotlib** later, which is where you'll be able to leverage the chart formatting skills you've learned throughout the course

CHART FORMATTING

Seaborn Basics

Chart Formatting

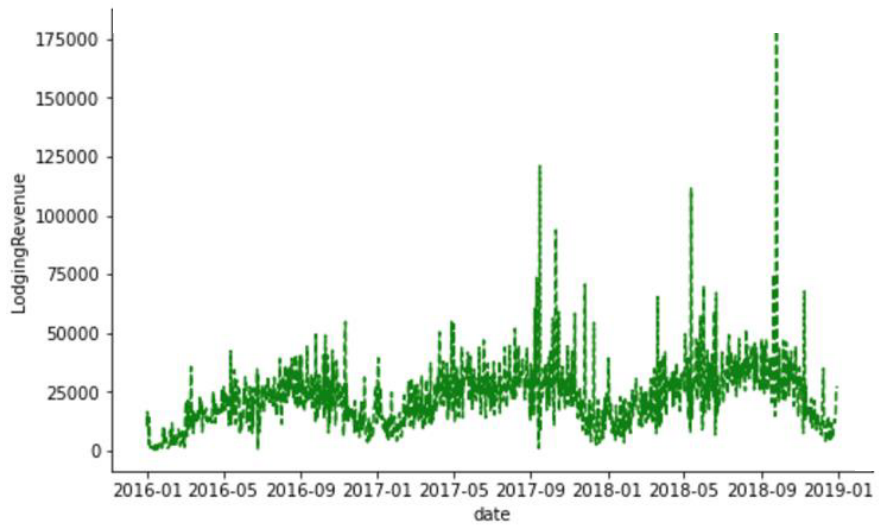
Chart Types

FacetGrid Layouts

Matplotlib Integration

Seaborn still has some useful **chart formatting** functions like despine()

```
import seaborn as sns

sns.lineplot(
    x="date",
    y="LodgingRevenue",
    data=hotels,
    estimator=sum,
    ci=None,
    ls='--',
    color='green'
)

sns.despine()
```



BAR CHARTS

Seaborn Basics
Chart Formatting
Chart Types
FacetGrid Layouts
Matplotlib Integration

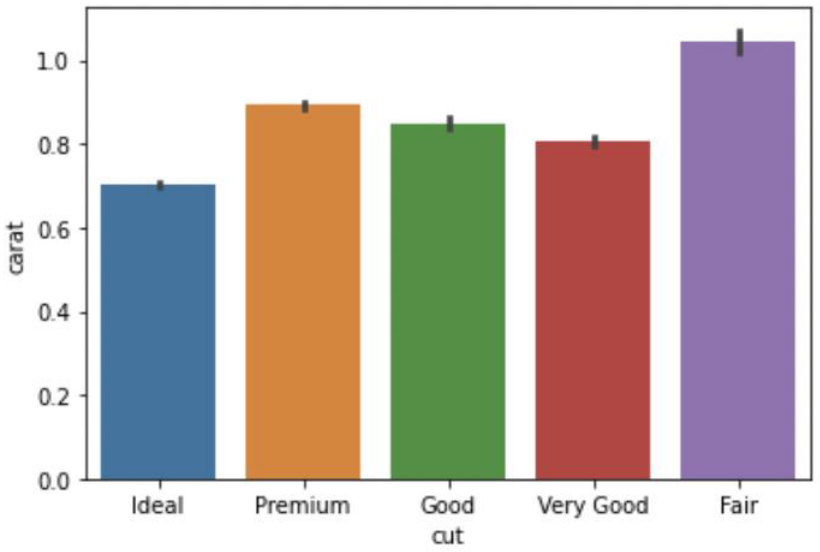
Bar charts can be created in Seaborn with sns.barplot()

- Simply specify the desired category labels and series values as “x” & “y” arguments

```
import seaborn as sns

diamonds = pd.read_csv("Diamonds Prices2022.csv")

diamonds.head()
```

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75

```
sns.barplot(
    x="cut",
    y="carat",
    data=diamonds,
)
```



Note that Seaborn **automatically aggregates the data** for the plot, using unique category values as the labels for the bars, the mean of each category for the bar length, and the column headers as the axis labels

BAR CHARTS

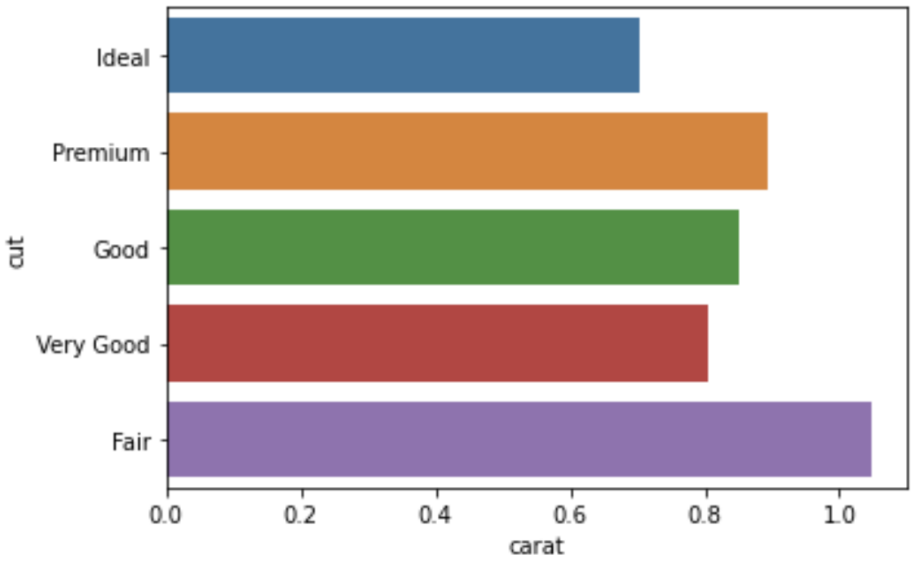
Bar charts can be created in Seaborn with sns.barplot()

- Simply specify the desired category labels and series values as “x” & “y” arguments

```
import seaborn as sns

diamonds = pd.read_csv("Diamonds Prices2022.csv")

diamonds.head()
```

	Unnamed: 0	carat	cut	color	clarity	depth	table	price	x	y	z
0	1	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	2	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	3	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	4	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	5	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75

```
sns.barplot(
    x="carat",
    y="cut",
    data=diamonds,
    ci=None
)
```



To create a horizontal bar chart, specify “x” as the data and “y” as the labels. ci=None will suppress error bars.

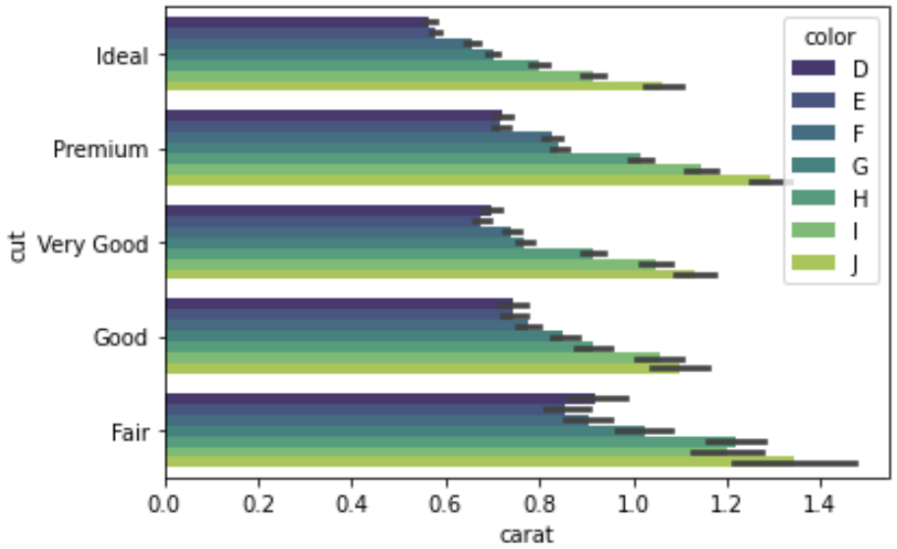
GROUPED BAR CHARTS

Grouped bar charts can be created by specifying a categorical column as “hue”


```
import seaborn as sns

diamonds = pd.read_csv("Diamonds Prices2022.csv")

diamonds.head()
```

	Unnamed: 0	carat	cut	color	clarity	depth	table	price	x	y	z
0	1	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	2	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	3	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	4	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	5	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75

```
sns.barplot(
    x="carat",
    y="cut",
    hue="color",
    data=diamonds.sort_values(by="color"),
    palette='viridis'
)
```

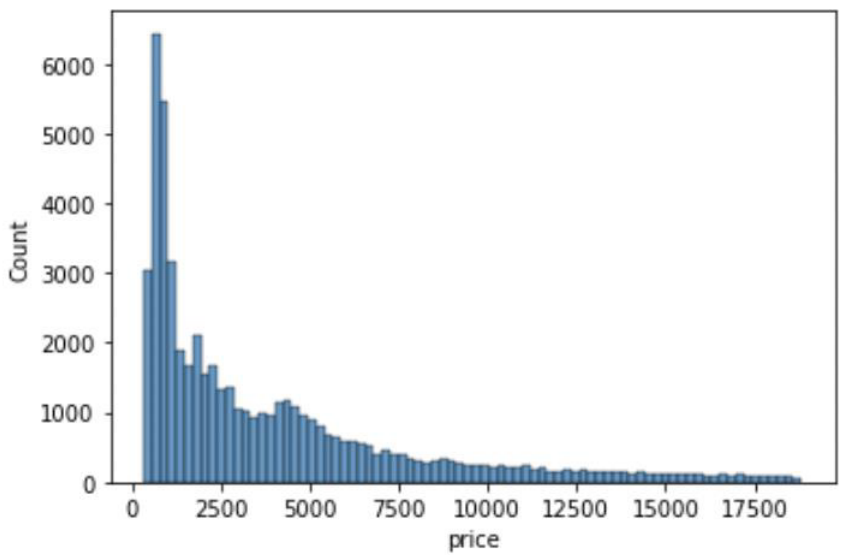


You can also sort the bars by one of the columns, and apply a different color map

HISTOGRAMS

Histograms can be created with sns.histplot() and a single “x” argument

```
sns.histplot(x="price", data=diamonds)
```

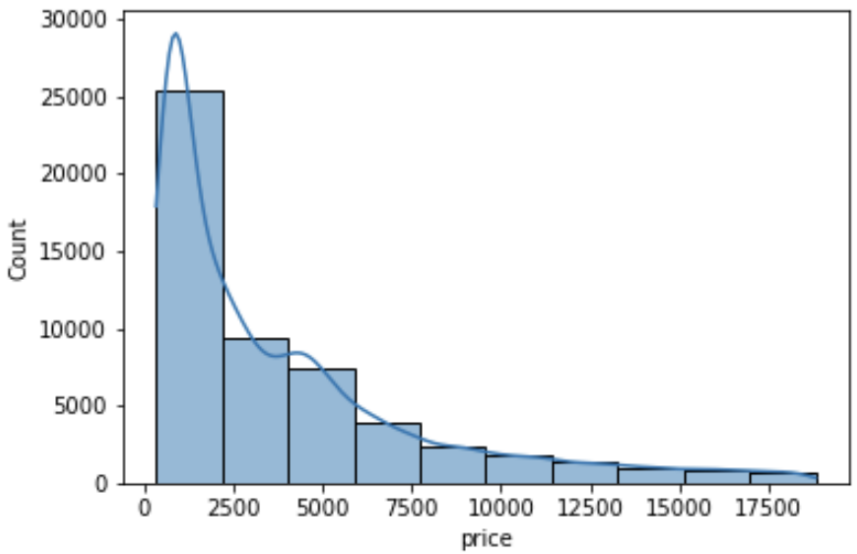


HISTOGRAMS

Histograms can be created with sns.histplot() and a single "x" argument

- You can also specify the number of "bins" and add the kernel density (kde=True)

```
sns.histplot(x="price", bins=10, kde=True, data=diamonds)
```



The default style for Seaborn plots can be nicer than their Matplotlib counterparts, and vice versa, so choose the library the works best for each chart!

ASSIGNMENT: BASIC CHARTS

1 **NEW MESSAGE**

September 20, 2024

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

Subject: **New Charts**

Hi,

Need a few more views on the hotel data using Seaborn.

Can we look at the distribution of lodging revenue for each booking? Only plot customers with less than 1,500 dollars to weed out longer term stays.

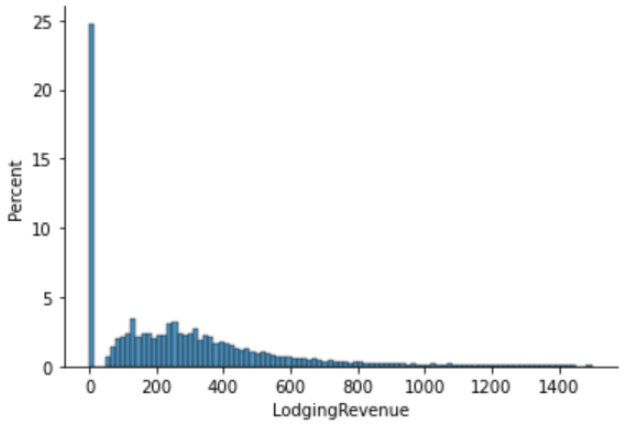
Then, build a bar chart with the average room nights stayed for our top 5 countries.

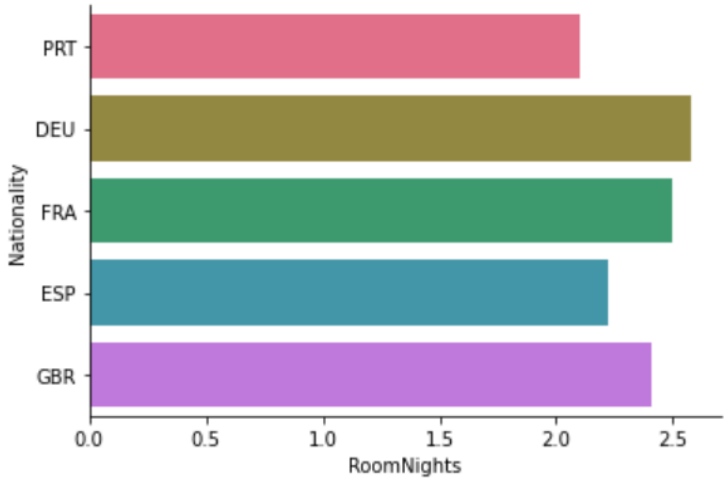
Thanks

section06_assignments.ipynb

Reply Forward

Results Preview



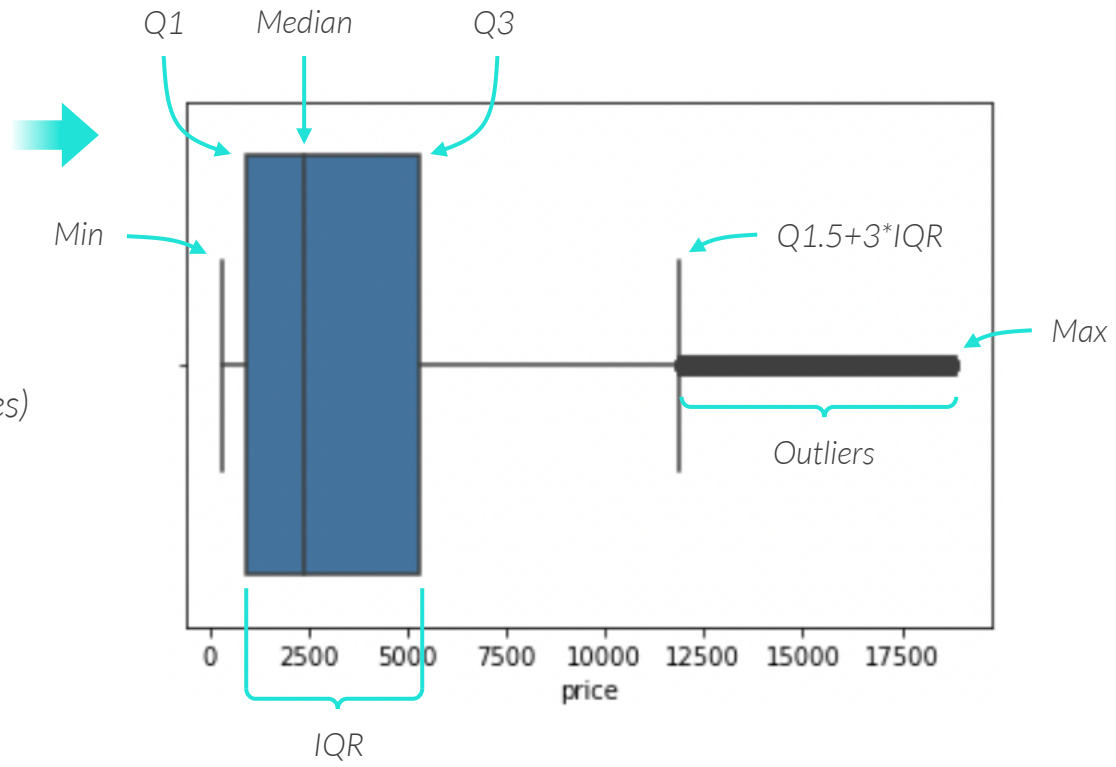


BOXPLOTS

Boxplots can be created with sns.boxplot()

- They visualize the distribution of a variable by plotting key statistics

```
sns.boxplot(x="price", data=diamonds)
```



Boxplot statistics:

- Median (*50th percentile*)
- 1st & 3rd Quartiles (*25th & 75th percentiles*)
- Interquartile Range (IQR)
- Min & Max Values (*or 1.5x the IQR*)
- Outliers

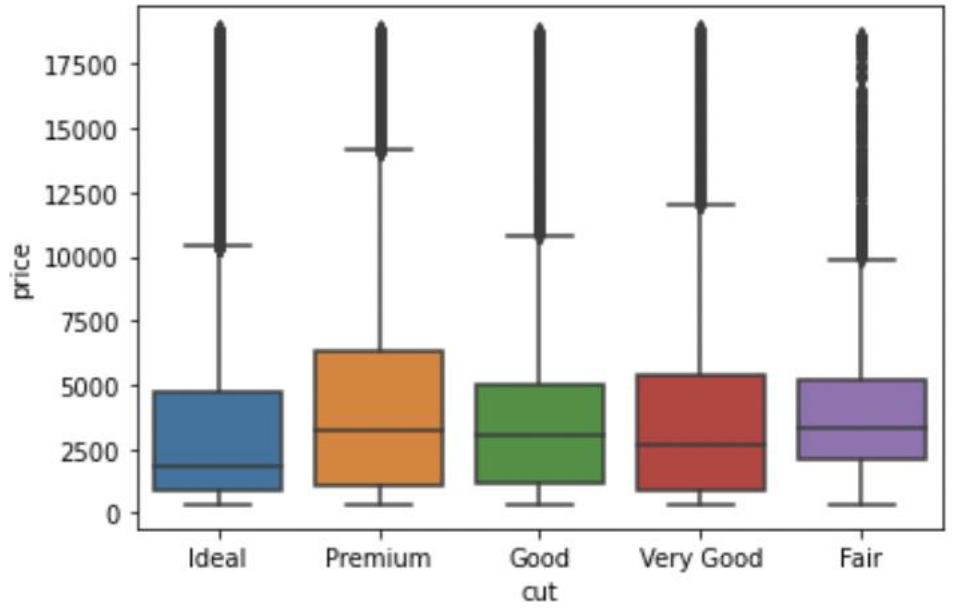
BOXPLOTS

Boxplots can be created with sns.boxplot()

- They visualize the distribution of a variable by plotting key statistics

```
sns.boxplot(x="cut", y="price", data=diamonds)
```

Specify a second axis to create separate boxplots by category

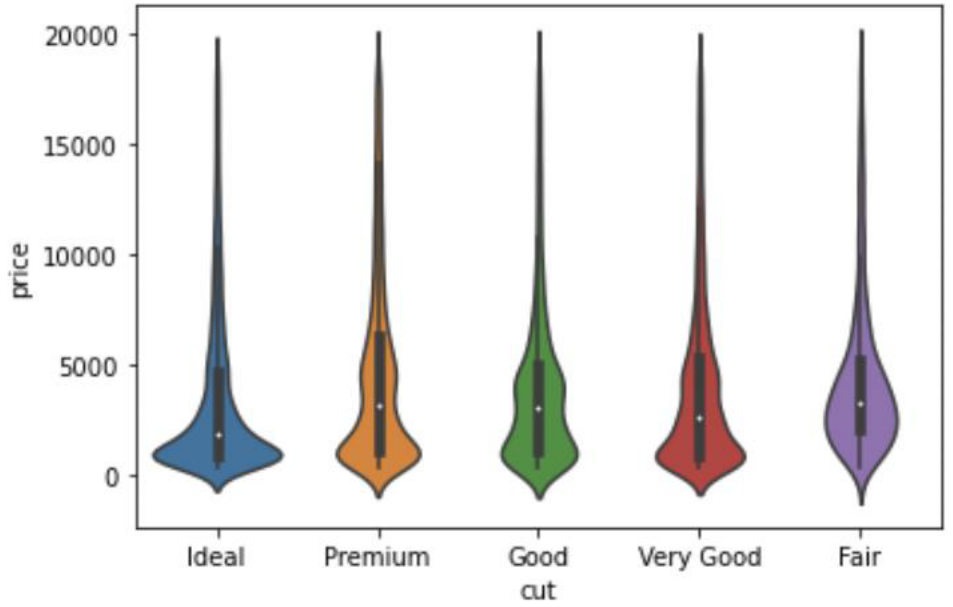


VIOLIN PLOTS

Violin plots can be created with sns.violinplot()

- They are boxplots with symmetrical kernel densities along their sides

```
sns.violinplot(x="cut", y="price", data=diamonds)
```



ASSIGNMENT: BOX & VIOLIN PLOTS

NEW MESSAGE

September 2024 ,24

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

Subject: **Re: New Charts**

Hi,

Let's view the distribution of lodging revenue using a boxplot instead, once again capping the revenue at 1500.

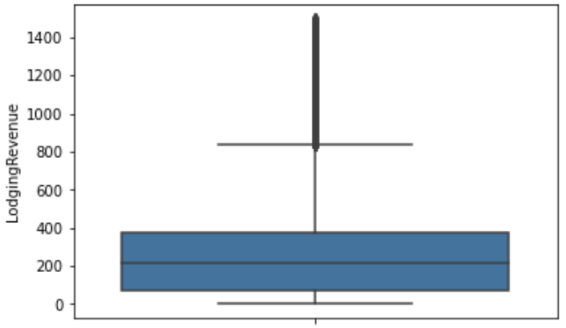
Then filter the data to the top 5 countries and build a violin plot of their lodging revenue, as well as their age distribution.

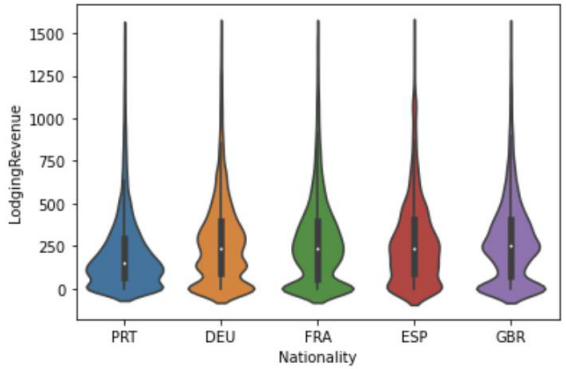
Sarah

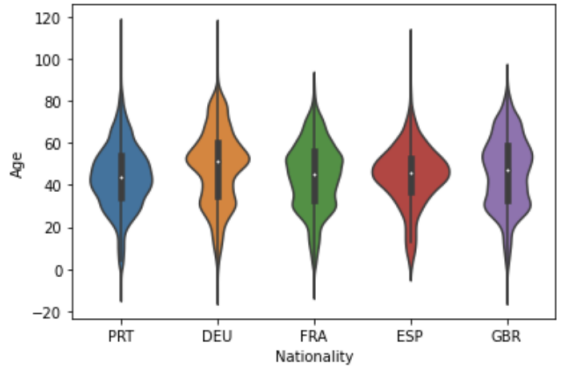
section06_assignments.ipynb

Reply Forward

Results Preview







LINEAR RELATIONSHIP PLOTS

Seaborn has several plots to explore **linear relationships**:

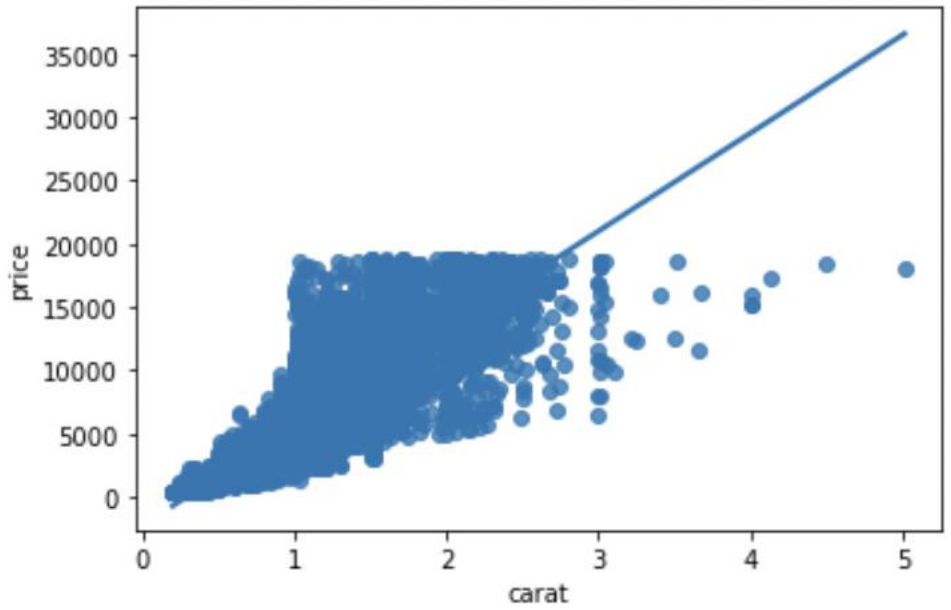
- Seaborn Basics
- Chart Formatting
- Chart Types
- FacetGrid Layouts
- Matplotlib Integration

scatterplot	*Creates a scatterplot*	**sns.scatterplot(*x*, *y*, *data*)**
regplot	*Creates a scatterplot with a fitted regression line*	**sns.regplot(*x*, *y*, *data*)**
lmplot	*Create a scatterplot with a fitted regression line, and can visualize multiple categories using color, or splitting into rows & columns*	**sns.lmplot(*x*, *y*, *hue*, *row*, *col*, *data*)**
jointplot	*Creates a scatterplot and adds the distribution for each variable*	**sns.jointplot(*x*, *y*, *kind*, *data*)**
pairplot	*Creates a matrix of scatterplots comparing multiple variables, and shows the distribution for each one*	**sns.pairplot(*cols*)**

REGPLOT()

sns.regplot() creates a scatterplot with a fitted regression line

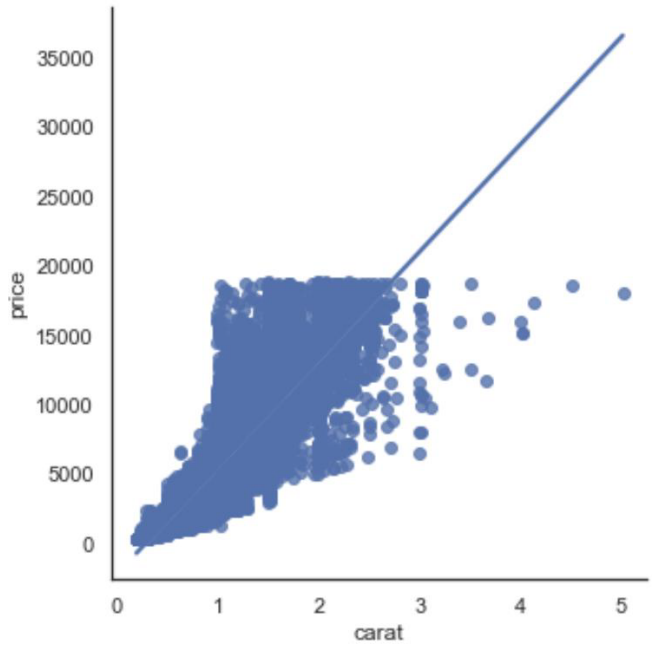

```
sns.regplot(x="carat", y="price", data=diamonds)
```



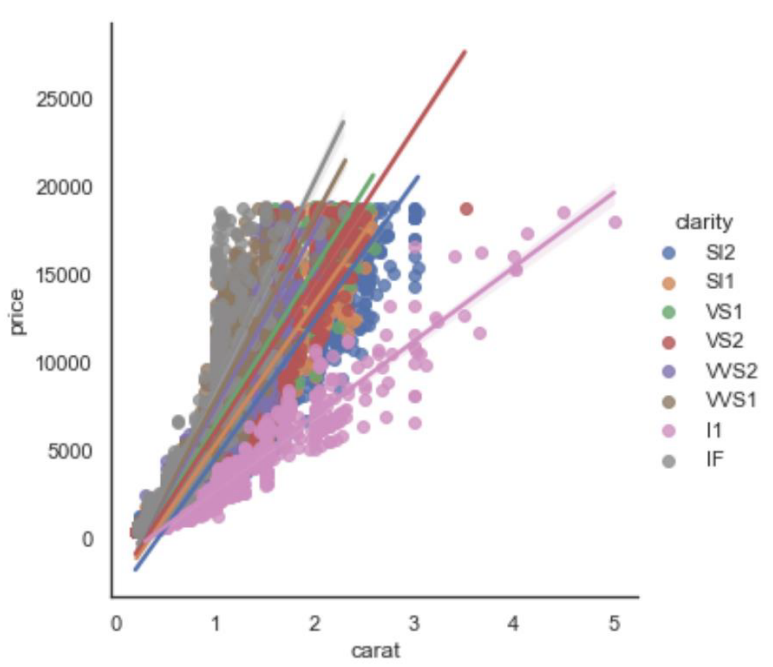
LMPLOT()

sns.lmplot() lets you explore the impact of other variables on the relationship


```
sns.lmplot(x="carat",
           y="price",
           data=diamonds)
```




```
sns.lmplot(x="carat",
           y="price",
           hue="clarity",
           data=diamonds)
```



Specify the 'hue' to create a line for each category in the specified column and set a different color for each category

LMPLOT()

sns.lmplot() lets you explore the impact of other variables on the relationship

Seaborn Basics

Chart Formatting

Chart Types

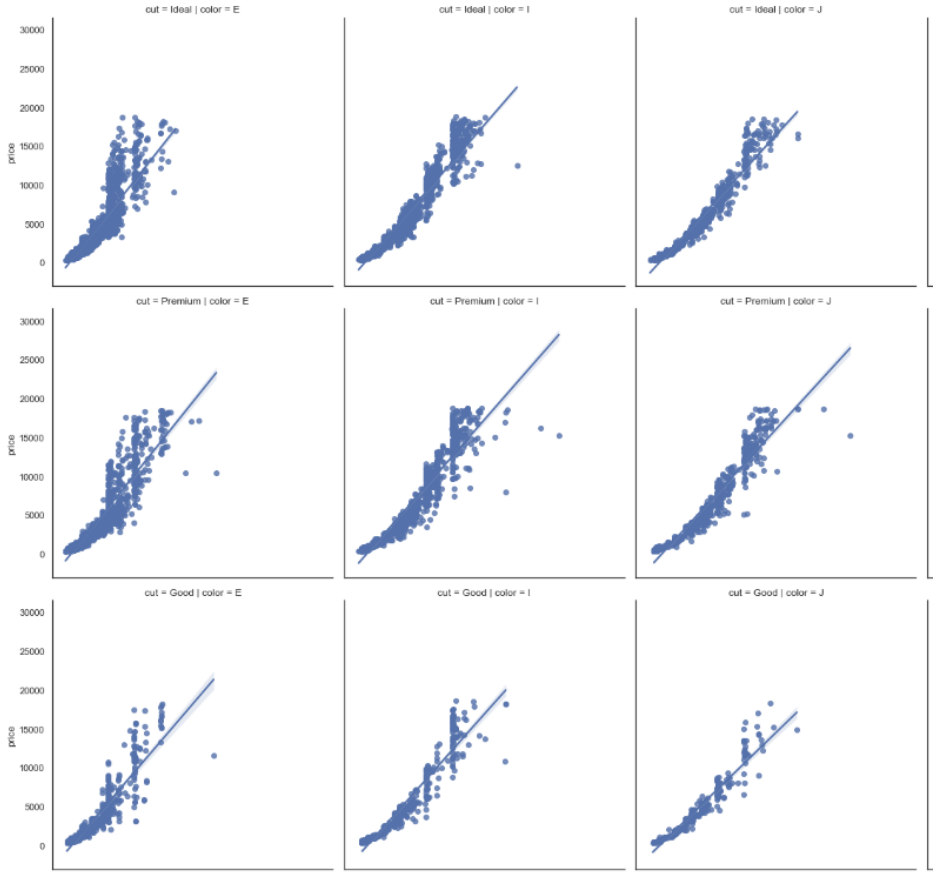
FacetGrid Layouts

Matplotlib Integration


```
sns.lmplot(
    x="carat",
    y="price",
    row="cut",
    col="color",
    data=diamonds
)
```

Specify the 'row' and 'column' to create regression plots for each combination of variables

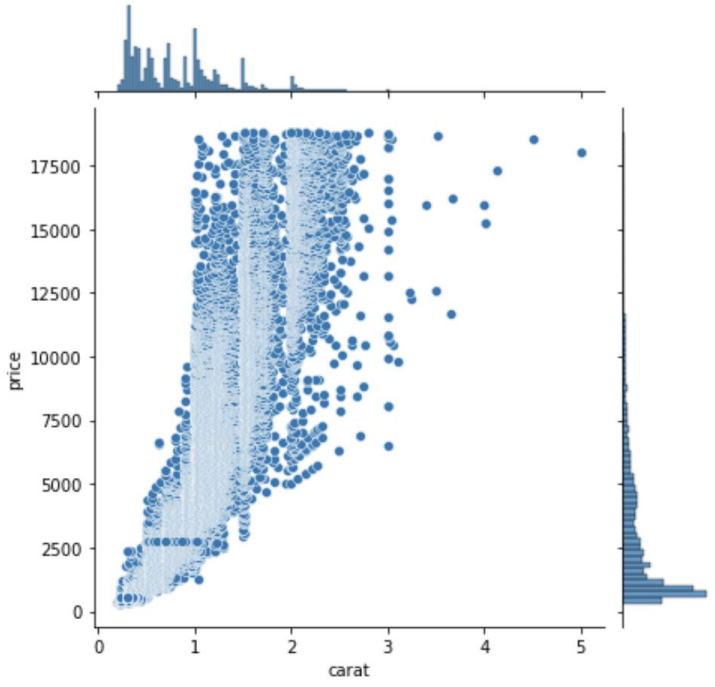
PRO TIP: This type of visual is great for exploring your data, but way too complex for a presentation!



JOINTPLOT()

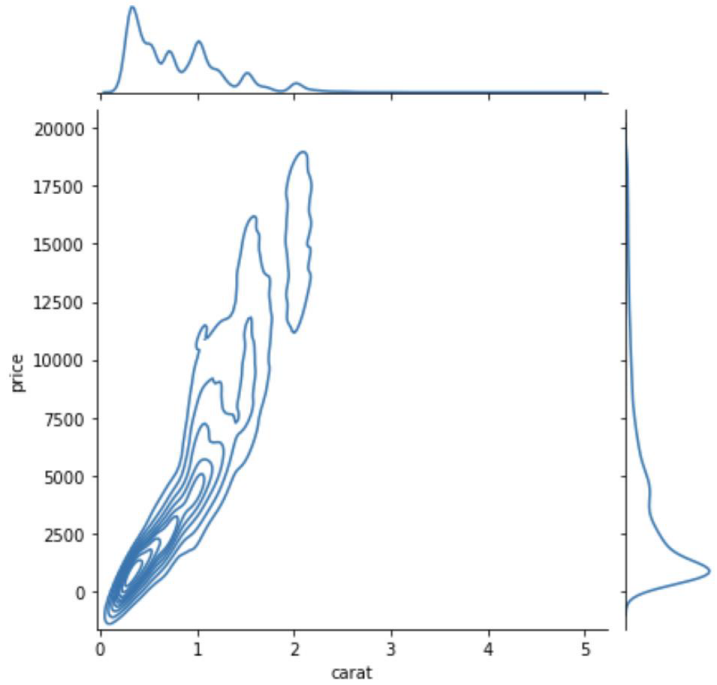
sns.jointplot() creates a scatterplot and adds the distribution of each variable


```
sns.jointplot(x="carat",
              y="price",
              data=diamonds)
```




```
sns.jointplot(x="carat",
              y="price",
              kind='kde',
              data=diamonds)
```

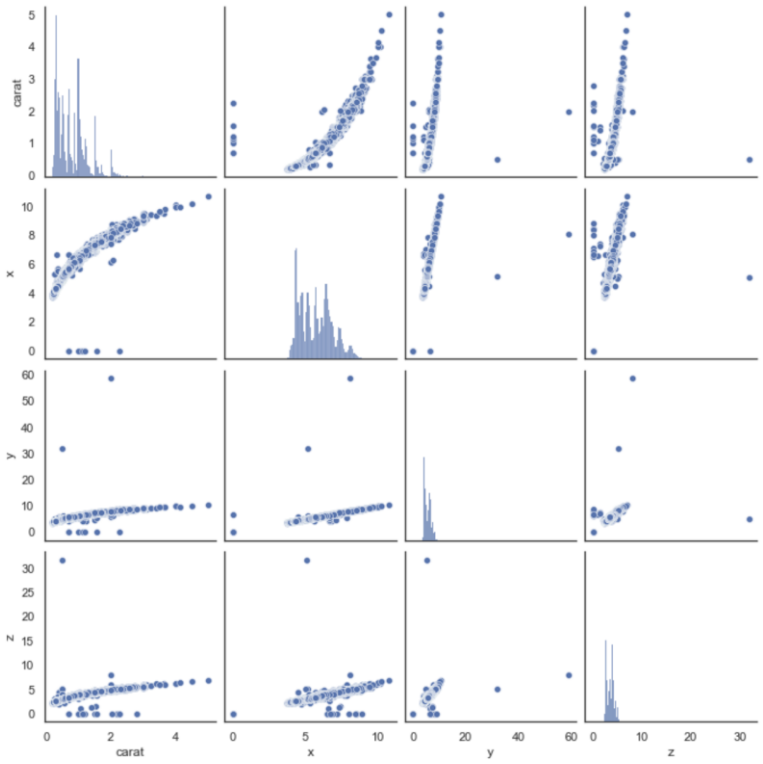
The 'kind' argument has several options like 'kde', which plots the kernel densities, and 'reg', which plots the regression line



PAIRPLOT()

sns.pairplot() creates a matrix of scatterplots comparing multiple variables, and shows the distribution for each one along the diagonal


```
sns.pairplot(diamonds.loc[:, ["carat", "x", "y", "z"]])
```



This lets you see the relationship between a diamond's weight (carat) and its length (x), width (y), and depth (z)

You can see that the weight of the diamond has a positive relationship with height, width, and length, with the relationships being VERY strong for width and depth

ASSIGNMENT: LINEAR RELATIONSHIP PLOTS

1 **NEW MESSAGE**

September 2024 ,26

From: **Wendy Whiz** *(Data Scientist)*

Subject: **More Exploration**

Hi there,

Can you produce charts to explore the relationship between room nights and lodging revenue?

First for all the data and then for each top 5 country.

Can you also produce a pairplot comparing lodging revenue to several key variables? (more details in the notebook)

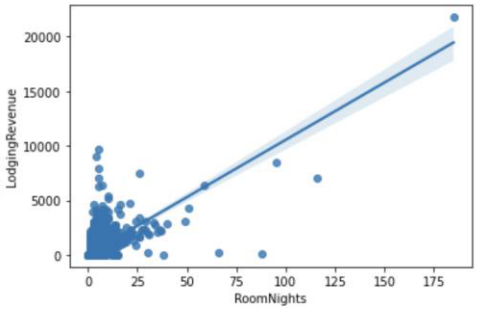
Best,

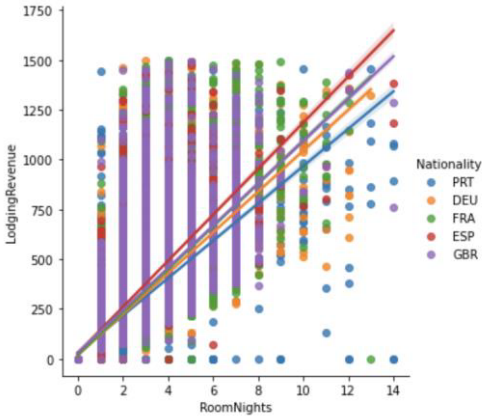
Wendy

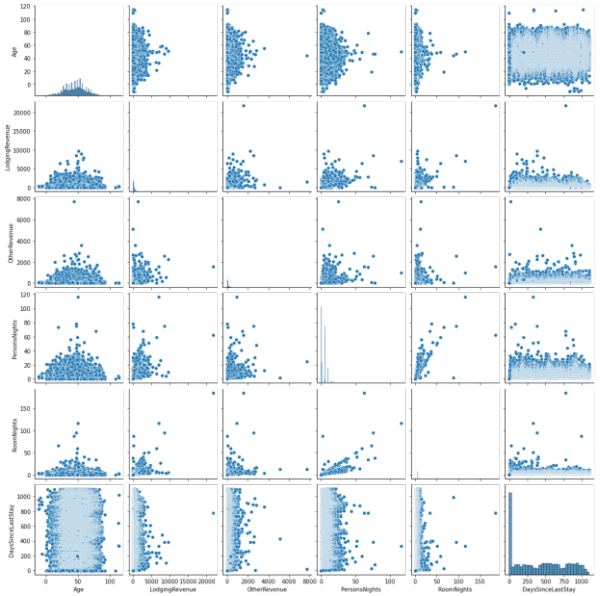
section06_assignments.ipynb

Reply Forward

Results Preview







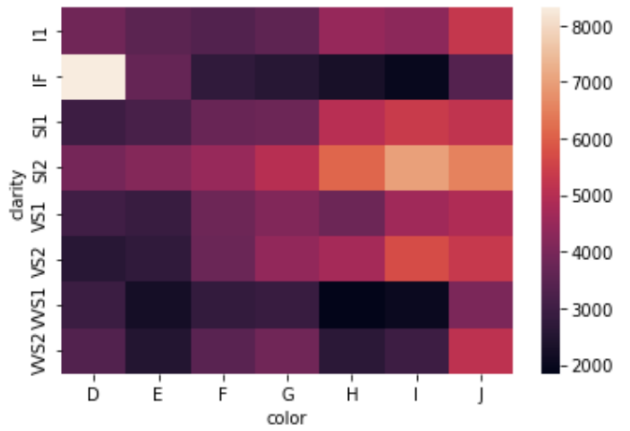
HEATMAPS

Create a **heatmap** to visualize a table of data with sns.heatmap()


```
diamonds_pivot.head()
```

color	D	E	F	G	H	I	J
clarity							
I1	3863.023810	3488.421569	3342.181818	3545.693333	4453.413580	4302.184783	5254.060000
IF	8307.369863	3668.506329	2750.836364	2558.033774	2287.869565	1994.937063	3363.882353
SI1	2976.146423	3161.670787	3713.776266	3774.787449	5032.414945	5355.019663	5186.048000
SI2	3931.101460	4173.826036	4472.625233	5021.684109	6099.895074	7002.649123	6520.958246
VS1	3030.158865	2856.294301	3796.717742	4131.362197	3780.688623	4633.183992	4884.461255

```
sns.heatmap(diamonds_pivot)
```



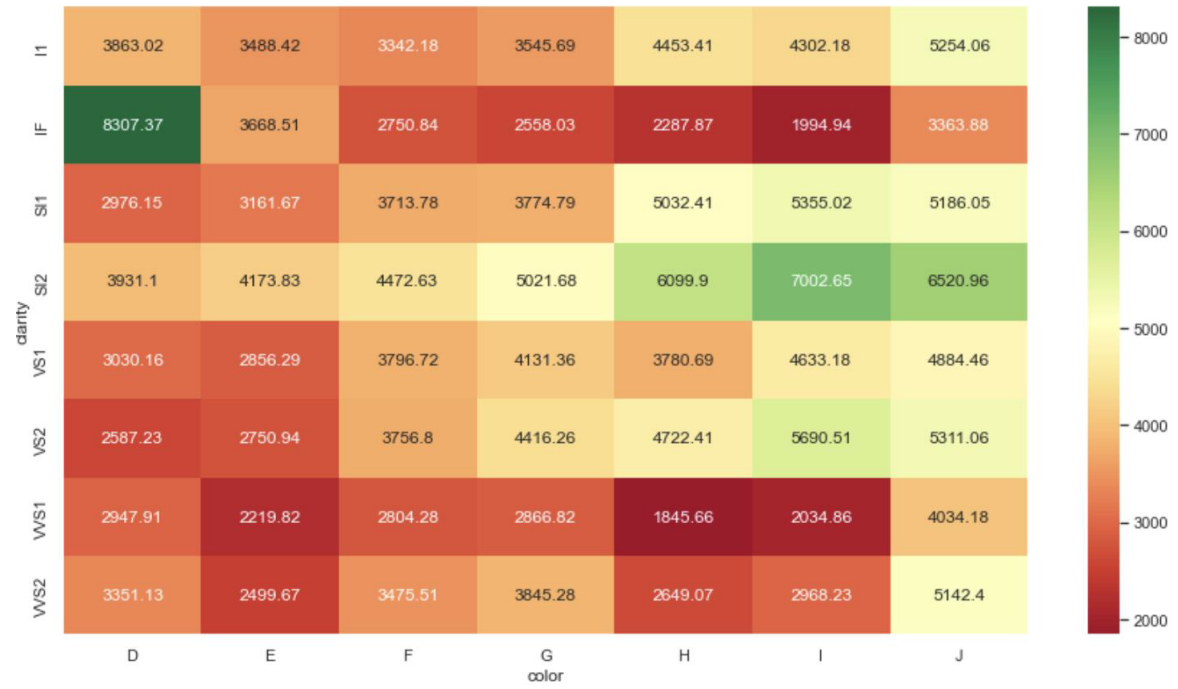
PRO TIP: Pandas' pivot_table method is a great way to set up the data needed for a heat map!

HEATMAPS

Create a **heatmap** to visualize a table of data with sns.heatmap()


```
sns.set(rc = {'figure.figsize':(15,8)})

sns.heatmap(
    diamonds_pivot,
    annot=True,
    fmt='g',
    cmap="RdYlGn"
)
```

You can **modify rcParameters** with sns.set(), but we'll show the syntax for combining Matplotlib and Seaborn shortly!



ASSIGNMENT: HEATMAPS

1 **NEW MESSAGE**

September 2024 ,26

From: **Wendy Whiz** *(Data Scientist)*

Subject: **RE: More Exploration**

Hi there,

Last piece to help me look at features for my modeling work.

Can you build a heatmap with countries as rows and market segment as columns with the mean lodging revenue for each?

Then build a heatmap for a correlation matrix.

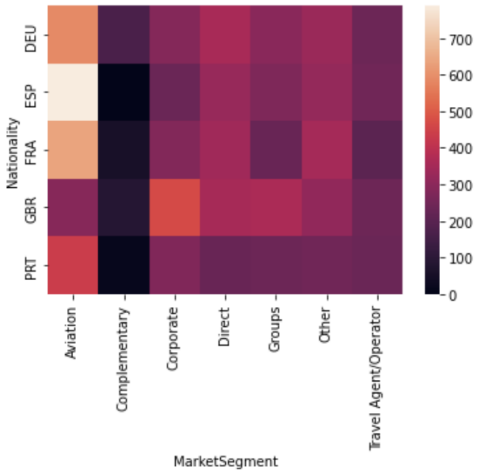
Thanks,

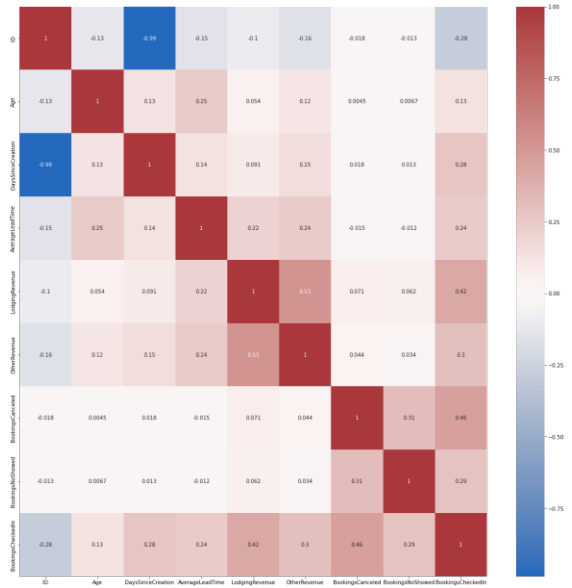
Wendy

section06_assignments.ipynb

Reply Forward

Results Preview





FACETGRID

Seaborn's **FacetGrid** is a convenient alternative to Matplotlib's subplot grids

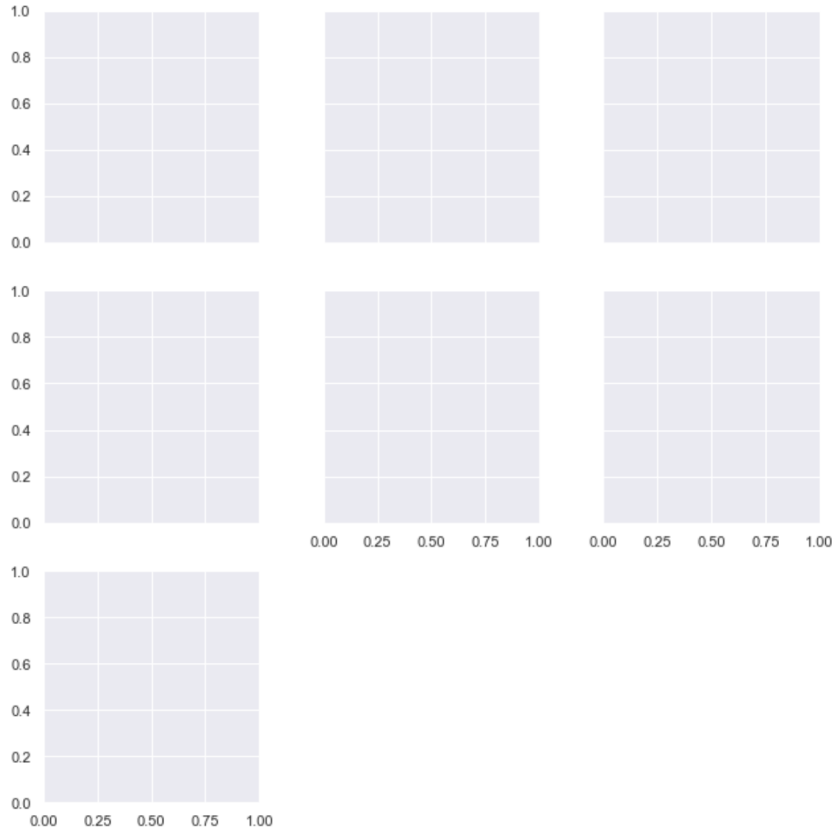
- **sns.FacetGrid(*DataFrame*, *column*, *column wrap*)**


```
import seaborn as sns

diamonds = pd.read_csv("Diamonds Prices2022.csv")

diamonds.head()
```

	Unnamed: 0	carat	cut	color	clarity	depth	table	price	x	y	z
0	1	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	2	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	3	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	4	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	5	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75

```
g = sns.FacetGrid(diamonds, col="color", col_wrap=3)
```

This creates 7 charts, one for each "color", in a grid with 3 columns



FACETGRID

Seaborn's **FacetGrid** is a convenient alternative to Matplotlib's subplot grids

- **sns.FacetGrid(*DataFrame*, *column*, *column wrap*)**

Seaborn Basics

Chart Formatting

Chart Types

FacetGrid Layouts

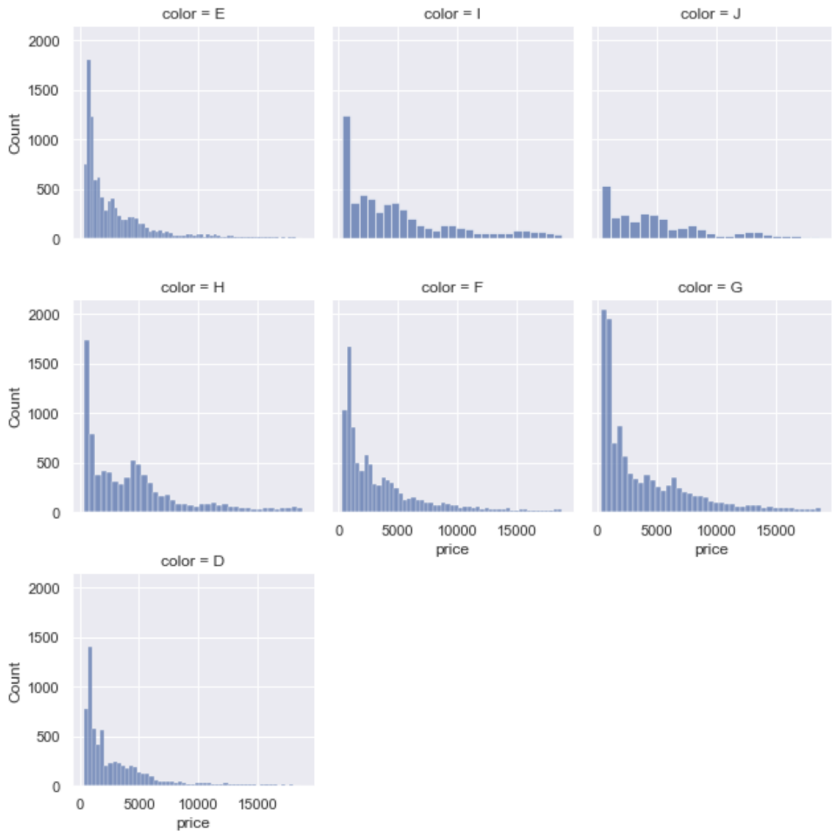
Matplotlib Integration


```
import seaborn as sns

diamonds = pd.read_csv("Diamonds Prices2022.csv")

diamonds.head()
```

	Unnamed: 0	carat	cut	color	clarity	depth	table	price	x	y	z
0	1	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	2	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	3	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	4	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	5	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75

```
g = sns.FacetGrid(diamonds, col="color", col_wrap=3)

g.map_dataframe(sns.histplot, x="price")
```

This plots a histogram of "price" for each "color" in the DataFrame

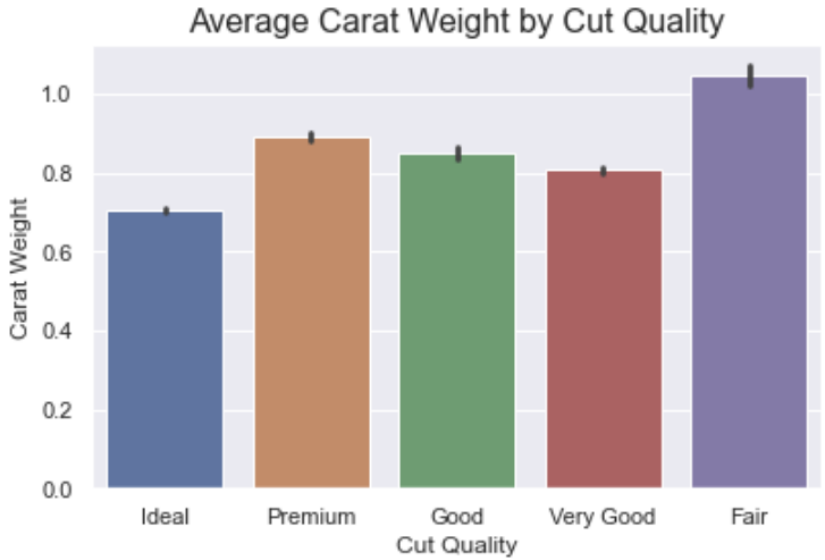


MATPLOTLIB INTEGRATION

You can **build Seaborn plots in Matplotlib objects**, which lets you customize and integrate Seaborn charts as if they were built using Matplotlib


```
fig, ax = plt.subplots(figsize=(6.4, 4))

sns.set_style("darkgrid")

sns.barplot(x="cut",
            y="carat",
            data=diamonds)

ax.set_title("Average Carat Weight by Cut Quality", fontsize=16)
ax.set_xlabel("Cut Quality")
ax.set_ylabel("Carat Weight")
```

This creates a Matplotlib figure and axis, sets a Seaborn style, creates a Seaborn bar chart, and then adds Matplotlib labels

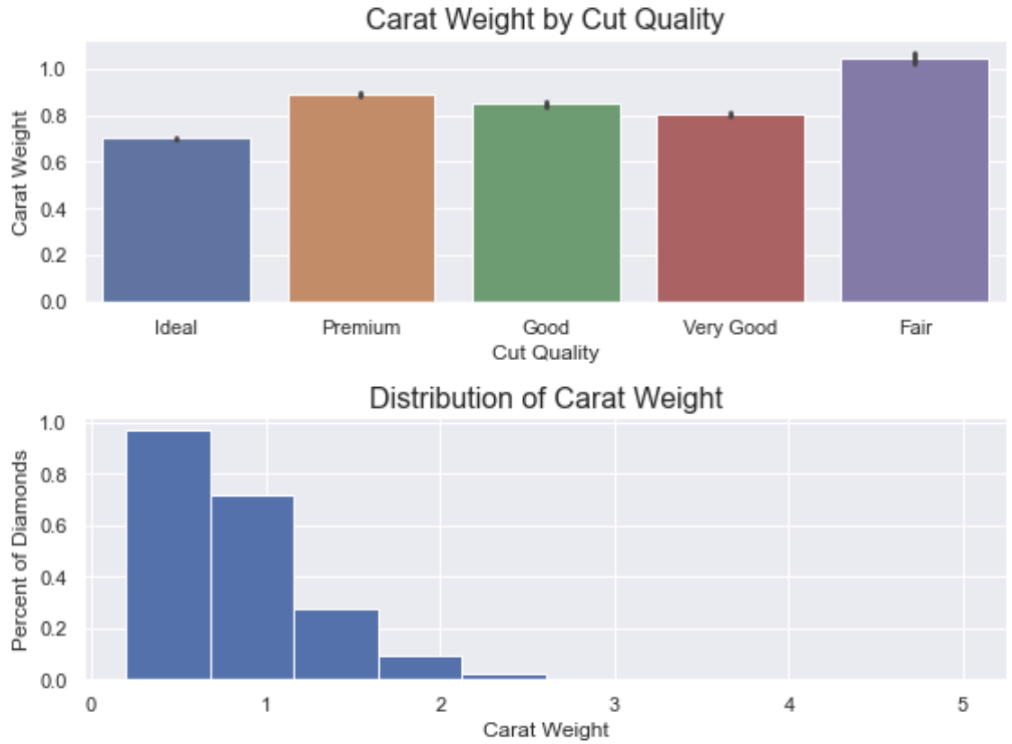


MATPLOTLIB INTEGRATION

You can **build Seaborn plots in Matplotlib objects**, which lets you customize and integrate Seaborn charts as if they were built using Matplotlib


```
fig, ax = plt.subplots(2, 1, figsize=(8, 6))

sns.set_style("darkgrid")

sns.barplot(
    x="cut",
    y="carat",
    data=diamonds,
    ax=ax[0]
)

ax[0].set_title("Carat Weight by Cut Quality", fontsize=16)
ax[0].set_ylabel("Carat Weight")
ax[0].set_xlabel("Cut Quality")

ax[1].hist(diamonds["carat"], density=True)
ax[1].set_title("Distribution of Carat Weight", fontsize=16
ax[1].set_xlabel("Carat Weight")
ax[1].set_ylabel("Percent of Diamonds")

fig.tight_layout()
```

This lets you specify which axes to plot the chart on



KEY TAKEAWAYS

Seaborn is a user-friendly extension of Matplotlib

- *It has a simple interface, nice aesthetics, and works well with Pandas DataFrames*

Seaborn adds **new chart types** that are useful in exploring data

- *Boxplots, violin plots, and linear model plots help profile data and identify relationships between variables*

Seaborn is very **compatible with Matplotlib**

- *Seaborn charts are extensions of Matplotlib objects, so they can be placed in Matplotlib figures*
- *Matplotlib formatting arguments can passed to corresponding Seaborn plotting functions*

PROJECT 3: ADVANCED EDA

PROJECT DATA: USED CARS DATA

```
cars.head()
```

	year	make	model	trim	body	transmission	vin	state	condition	odometer	color	interior	seller	mmr	sellingprice	saledate
0	2015	Kia	Sorento	LX	SUV	automatic	5xyktca69fg566472	ca	5.0	16639.0	white	black	kia motors america, inc	20500	21500	Tue Dec 16 2014 12:30:00 GMT-0800 (PST)
1	2015	Kia	Sorento	LX	SUV	automatic	5xyktca69fg561319	ca	5.0	9393.0	white	beige	kia motors america, inc	20800	21500	Tue Dec 16 2014 12:30:00 GMT-0800 (PST)
2	2014	BMW	3 Series	328i SULEV	Sedan	automatic	wba3c1c51ek116351	ca	4.5	1331.0	gray	black	financial services remarketing (lease)	31900	30000	Thu Jan 15 2015 04:30:00 GMT-0800 (PST)
3	2015	Volvo	S60	T5	Sedan	automatic	yv1612tb4f1310987	ca	4.1	14282.0	white	black	volvo na rep/world omni	27500	27750	Thu Jan 29 2015 04:30:00 GMT-0800 (PST)
4	2014	BMW	6 Series Gran Coupe	650i	Sedan	automatic	wba6b2c57ed129731	ca	4.3	2641.0	gray	black	financial services remarketing (lease)	66000	67000	Thu Dec 18 2014 12:30:00 GMT-0800 (PST)

```
cars.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 558811 entries, 0 to 558810
Data columns (total 16 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
 0   year           558811 non-null  int64
 1   make           548510 non-null  object
 2   model          548412 non-null  object
 3   trim           548160 non-null  object
 4   body           545616 non-null  object
 5   transmission   493458 non-null  object
 6   vin            558811 non-null  object
 7   state          558811 non-null  object
 8   condition      547017 non-null  float64
 9   odometer       558717 non-null  float64
 10  color          558062 non-null  object
 11  interior       558062 non-null  object
 12  seller         558811 non-null  object
 13  mmr            558811 non-null  int64
 14  sellingprice   558811 non-null  int64
 15  saledate       558811 non-null  object
dtypes: float64(2), int64(3), object(11)
memory usage: 68.2+ MB
```

ASSIGNMENT: FINAL PROJECT

NEW MESSAGE

October 2024 ,10

From: **Aaron Auto** *(VP of Fleet Management)*

Subject: **Optimal Fleet Truck Purchase**

Hello,

We need an outside analysis on auto procurement for our fleet of service vehicles. We lease trucks to contractors and other businesses, but a recent spike in demand has meant we're unable to get cars from traditional suppliers.

I want to see an overview of the automotive auction industry, before diving into where we can get Ford F150s for the most affordable price on the market (more details in the notebook).

Thanks

section07_final_project.ipynb

Reply Forward

Key Objectives

1. Read in and manipulate data with Pandas
2. Build summary charts with Matplotlib and Seaborn
3. Leverage Seaborn's advanced chart types to mine insights from the data and make a decision

MCG