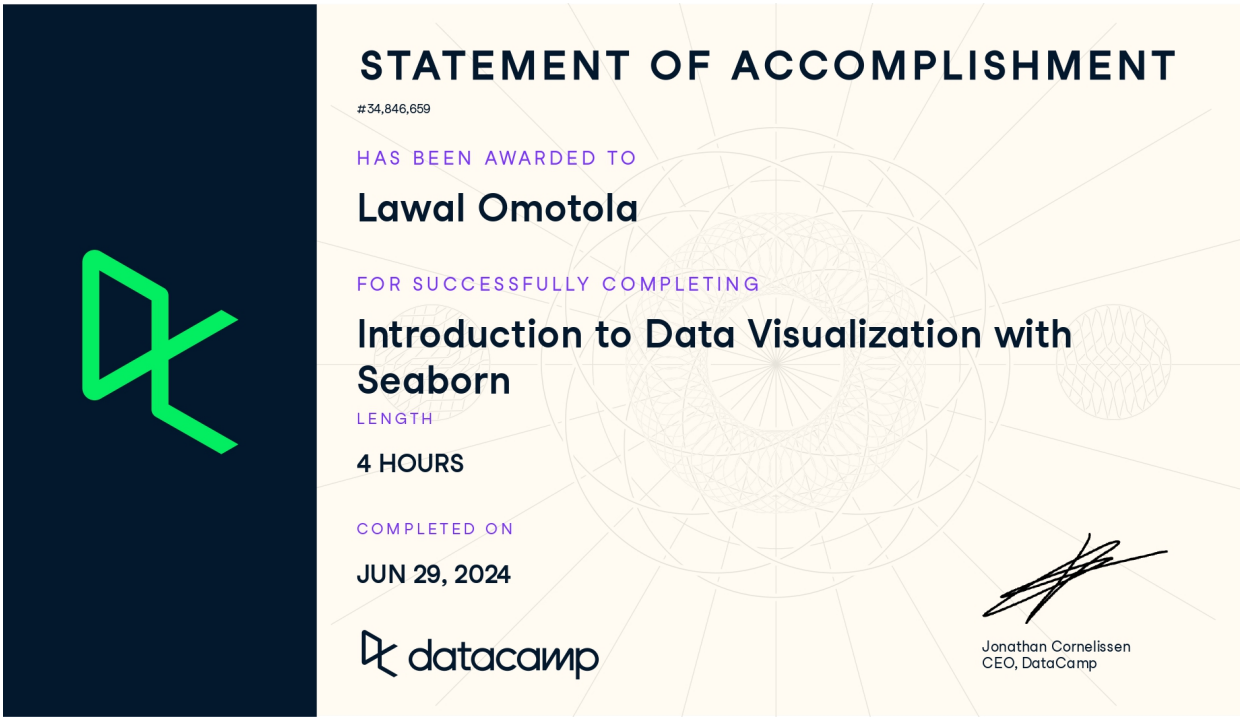
Introduction to Data Visualization with Seaborn

Lawal's Note

2025-10-21

Table of contents



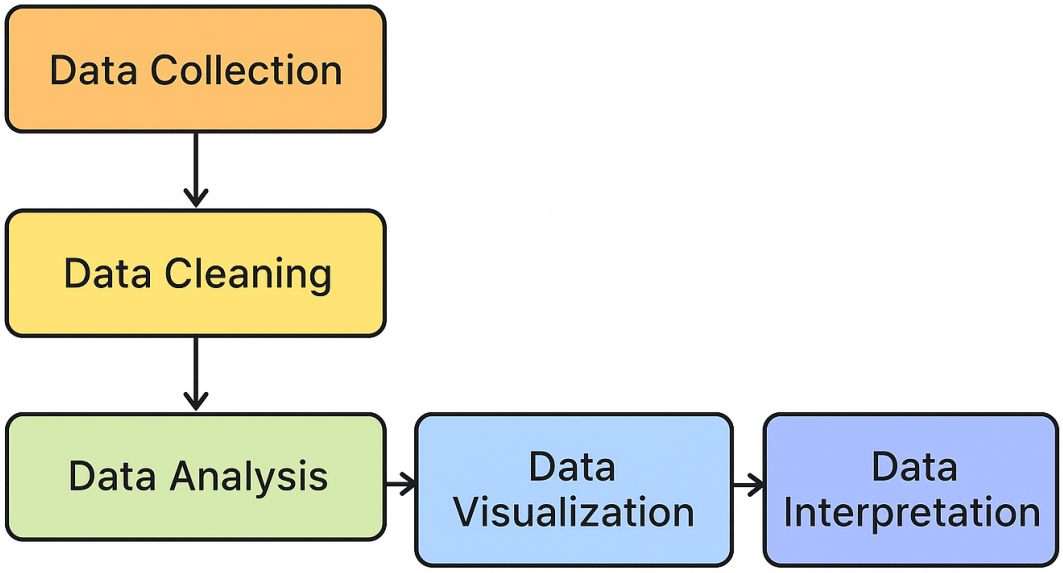
Chapter 1: Introduction to Seaborn

Chapter 1.1: What is Seaborn?

What is Seaborn? Seaborn is a powerful Python library for creating data visualizations. It was developed in order to make it easy to create the most common types of plots. The plot shown here can be created with just a few lines of Seaborn code.

1 Waskom, M. L. (2021). seaborn: statistical data visualization. https://seaborn.pydata.org/

Why is Seaborn useful?



This is a picture of a typical data analysis workflow. Data visualization is often a huge component of both the data exploration phase and the communication of results, so Seaborn will be very useful there.

Advantages of Seaborn

There are several tools that can be used for data visualization, but Seaborn offers several advantages. First, Seaborn's main purpose is to make data visualization easy. It was built to automatically handle a lot of complexity behind the scenes. Second, Seaborn works extremely well with pandas data structures. `pandas` is a Python library that is widely used for data analysis. Finally, it's built on top of `Matplotlib`, which is another Python visualization library. Matplotlib is extremely flexible. Seaborn allows you to take advantage of this flexibility when you need it, while avoiding the complexity that Matplotlib's flexibility can introduce.

Getting started

To get started, we'll need to import the `Seaborn` library. The line `"import seaborn as sns"` will import `Seaborn` as the conventionally used alias `"sns"`. Why `"sns"`? The Seaborn library was apparently named after a character named Samuel Norman Seaborn from the television show `"The West Wing"` - thus, the standard alias is the character's initials (`"sns"`). We also need to import `Matplotlib`, which is the library that Seaborn is built on top of. We do this by typing `"import matplotlib.pyplot as plt"`. `"plt"` is the alias that most people use to refer to `Matplotlib`, so we'll use that here as well.

```
# Import packages
import seaborn as sns
import matplotlib.pyplot as plt
```

Example 1: Scatter plot

Let's now dive into an example to illustrate how easily you can create visualizations using Seaborn. Here, we have data for 10 people consisting of lists of their heights in inches and their weights in pounds. Do taller people tend to weigh more? You can visualize this using a type of plot known as a scatter plot, which you'll learn more about later in the course. Use `"sns.scatterplot()"` to call the scatterplot function from the Seaborn library. Then, specify what to put on the x-axis and y-axis. Finally, call the `"plt.show()"` function from Matplotlib to show the scatterplot. This plot shows us that taller people tend to have a higher weight.

```
# Import packages
import seaborn as sns
import matplotlib.pyplot as plt

# Import dataset
df = pd.read_csv("datasets/data.csv")

sns.scatterplot(x=weights, y=heights, data = df)

# Show plot
plt.show()
```

Example 2: Create a count plot

How many of our observations of heights and weights came from males vs. females? You can use another type of plot - the count plot - to investigate this. Count plots take in a categorical list and return bars that represent the number of list entries per category. Use the `"countplot()"`

function and provide the list of every person's gender. This count plot shows that out of the 10 observations we had in our height and weight scatter plot, 6 were male and 4 were female.

```
# Import packages
import seaborn as sns
import matplotlib.pyplot as plt

# Import dataset
df = pd.read_csv("datasets/data.csv")

sns.countplot(x= 'Gender', data = df)

# Show plot
plt.show()
```

Course Preview

Now, those were a couple of simple examples. Throughout this course, you'll learn to make more complex visualizations such as those pictured here. More importantly, you'll learn when to use each type of visualization in order to most effectively extract and communicate insights using data.

1 Waskom, M. L. (2021). seaborn: statistical data visualization. https://seaborn.pydata.org/

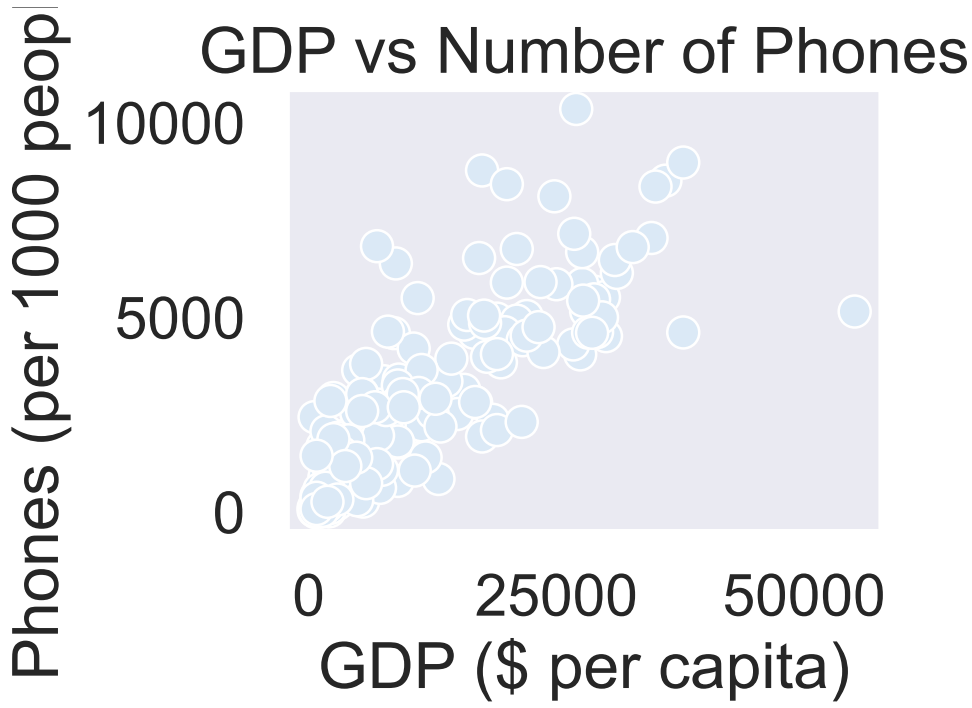
Exercise 1.1

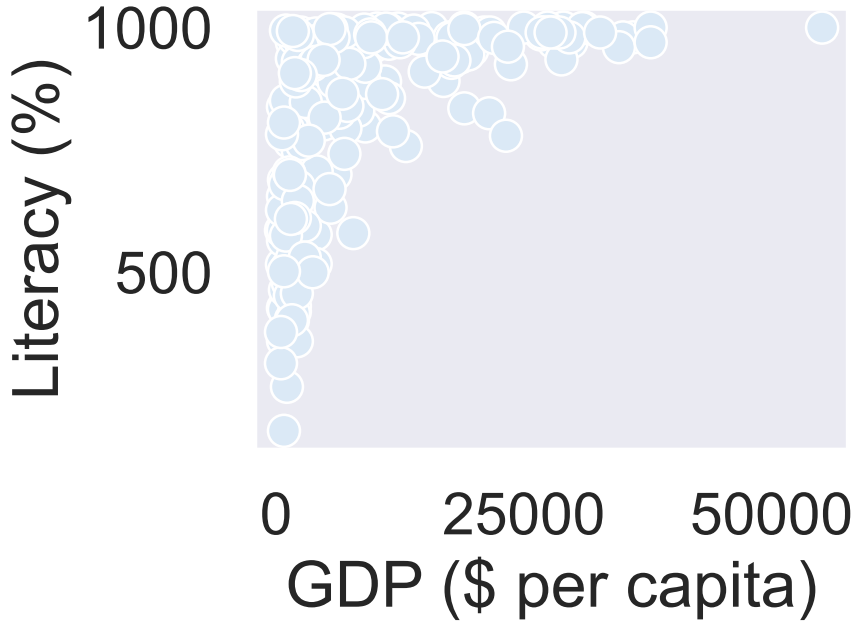
---

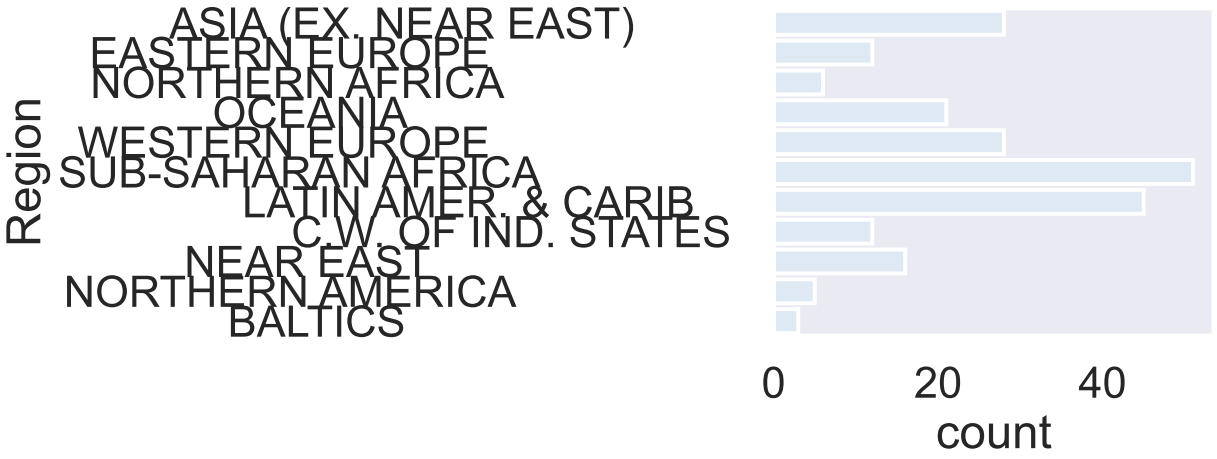
1. Import the relevant packages and load dataset.
2. Convert columns to numeric data types.
3. Create scatter plot with `GDP` on the x-axis and `number of phones` on the y-axis.
4. Change this scatter plot to have percent literate on the y-axis.
5. Create count plot with region on the y-axis.

```
# Import packages
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Load dataset
df = pd.read_csv("datasets/countries-of-the-world.csv")
```

```
# List of columns to convert to numeric
num_cols = [
    'Pop. Density (per sq. mi.)',
    'Coastline (coast/area ratio)',
    'Net migration',
    'Infant mortality (per 1000 births)',
    'GDP ($ per capita)',
    'Literacy (%)',
    'Phones (per 1000)',
    'Arable (%)',
    'Crops (%)',
    'Other (%)',
    'Climate',
    'Birthrate',
    'Deathrate',
    'Agriculture',
    'Industry',
    'Service'
]

# Convert columns to numeric (float), forcing errors to NaN
for col in num_cols:
    df[col] = pd.to_numeric(df[col].astype(str).str.replace(',', '').str.strip(), errors='coerce'


# Create scatter plot with GDP on the x-axis and number of phones on the y-axis
sns.scatterplot(x='GDP ($ per capita)', y='Phones (per 1000)', data=df)
plt.title('GDP vs Number of Phones')
plt.xlabel('GDP ($ per capita)')
plt.ylabel('Phones (per 1000 people)')
plt.show()

# Change this scatter plot to have percent literate on the y-axis
sns.scatterplot(x='GDP ($ per capita)', y='Literacy (%)', data=df)
plt.show()

# Create count plot with region on the y-axis
sns.countplot(y= 'Region', data = df)
# Show plot
plt.show()
```


GDP vs Number of Phones
Phones (per 1000 peop
10000
5000
0
0
25000
50000
GDP ($ per capita)


Literacy (%)
1000
500
0
25000
50000
GDP ($ per capita)



Chapter 1.2: Using pandas with Seaborn

Data scientists commonly use pandas to perform data analysis, so it's a huge advantage that Seaborn works extremely well with pandas data structures. Let's see how this works!

What is pandas?

`pandas` is a python library for data analysis. It can easily read datasets from many types of files including `csv` and `txt` files. pandas supports several types of data structures, but the most common one is the DataFrame object. When you read in a dataset with pandas, you will create a DataFrame.

Working with DataFrames

Let's look at an example. First, import the `pandas` library as `"pd"`. Then, use the `"read_csv"` function to read the `csv` file named `"masculinity.csv"` and create a pandas DataFrame called `"df"`. Calling `"head"` on the DataFrame will show us its first five rows. This dataset contains the results of a survey of adult men. We can see that it has four columns: `"participant_id"`; `"age"`; `"how_masculine"`, which is that person's response to the question `"how masculine or 'manly' do you feel?"`; and `"how_important"`, which is the response to the question `"how important is it to you that others see you as masculine?"`

```
# Import the pandas library
import pandas as pd

# Data provided by the user, extended to 20 rows
masculinity = {
    'participant_id': list(range(1, 21)),
    'age': [35, 42, 29, 51, 37, 30, 45, 55, 25, 33, 40, 48, 28, 39, 47, 53, 26, 34, 43, 50],
    'how_masculine': ['very', 'somewhat', 'very', 'not at all', 'somewhat', 'very', 'somewhat', '
    'how_important': ['very', 'somewhat', 'not very', 'somewhat', 'very', 'very', 'somewhat', 'no
}

# Create the DataFrame
df = pd.DataFrame(masculinity)

# Display the first five rows of the DataFrame
print(df.head())
```

```
   participant_id  age how_masculine how_important
0               1   35          very          very
1               2   42      somewhat      somewhat
2               3   29          very      not very
3               4   51    not at all      somewhat
4               5   37      somewhat          very
```

Using DataFrames with countplot()

Now let's look at how to make a count plot with a DataFrame instead of a list of data. The first thing we'll do is import `pandas`, `Matplotlib` and `Seaborn` as we have in past examples. Then, we'll create a pandas DataFrame called `"df"` from the `masculinity` csv file. To create a count plot with a pandas DataFrame column instead of a list of data, set `x` equal to the name of the column in the DataFrame - in this case, we'll use the `"how_masculine"` column. Then, we'll set the data parameter equal to our DataFrame, `"df"`. After calling `"plt.show"`, we can see that we have a nice count plot of the values in the `"how_masculine"` column of our data. This plot shows us that the most common response to the question `"how masculine or 'manly' do you feel?"` is `"somewhat"`, with `"very"` being the second most common response. Note also that because we're using a named column in the DataFrame, Seaborn automatically adds the name of the column as the x-axis label at the bottom.

```
# Import libraries
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
```

```python
# Data provided by the user, extended to 20 rows
masculinity = {
    'participant_id': list(range(1, 21)),
    'age': [35, 42, 29, 51, 37, 30, 45, 55, 25, 33, 40, 48, 28, 39, 47, 53, 26, 34, 43, 50],
    'how_masculine': ['very', 'somewhat', 'very', 'not at all', 'somewhat', 'very', 'somewhat', '
    'how_important': ['very', 'somewhat', 'not very', 'somewhat', 'very', 'very', 'somewhat', 'no
}

# Create the DataFrame
df = pd.DataFrame(masculinity)

# Create a count plot using the 'how_masculine' column

sns.countplot(x="how_masculine", data=df)

# Display the plot

plt.show()
```

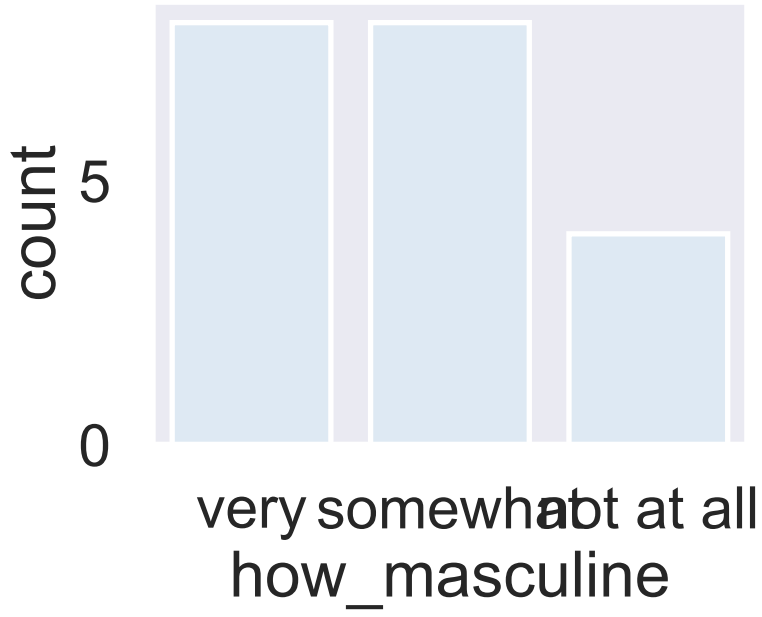


Figure 1: Distribution of Masculinity Perception.

"Tidy" data

Let's pause for an important note here. Seaborn works great with pandas DataFrames, but only if the DataFrame is `"tidy"`. `"Tidy data"` means that each observation has its own row and each variable has its own column. The `"masculinity"` DataFrame shown here is tidy because each row is a survey response with one answer to each survey question in each column. Making a count plot with the `"how masculine"` column works just like passing in a list of that column's values.

"Untidy" data

In contrast, here is an example of an `"untidy"` DataFrame made from the same survey on masculinity. In this untidy DataFrame, notice how each row doesn't contain the same information. Row 0 contains the age categories, rows 1 and 7 contain the question text, and the other rows contain summary data of the responses. This will not work well with Seaborn. Unlike the tidy DataFrame, values in the `"Age"` column don't look like a list of age categories for each observation. Transforming untidy DataFrames into tidy ones is possible, but it's not in scope for this course. There are other DataCamp courses that can teach you how to do this.

Chapter 1.3: Adding a third variable with hue

We saw in the last lesson that a really nice advantage of Seaborn is that it works well with pandas DataFrames. In this lesson, we'll see another big advantage that Seaborn offers: the ability to quickly add a third variable to your plots by adding color.

Tips dataset

To showcase this cool feature in Seaborn, we'll be using Seaborn's built-in tips dataset. You can access it by using the `"load dataset"` function in Seaborn and passing in the name of the dataset. These are the first five rows of the tips dataset. This dataset contains one row for each table served at a restaurant and has information about things like the bill amount, how many people were at the table, and when the table was served. Let's explore the relationship between the `"total_bill"` and `"tip"` columns using a scatter plot.

```python
# Import required libraries
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt

# Load Seaborn's built-in 'tips' dataset
df = sns.load_dataset("tips")

# Display the first five rows
df.head()
```

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

A basic scatter plot

Here is the code to generate it. The total bill per table (in dollars) is on the x-axis, and the total tip (in dollars) is on the y-axis. We can see from this plot that larger bills are associated with larger tips. What if we want to see which of the data points are smokers versus non-smokers? Seaborn makes this super easy.

```python
# Import required libraries
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt

# Load Seaborn's built-in 'tips' dataset
df = sns.load_dataset("tips")

sns.scatterplot(x="total_bill", y="tip", data=df)

# Show the figure
plt.show()
```

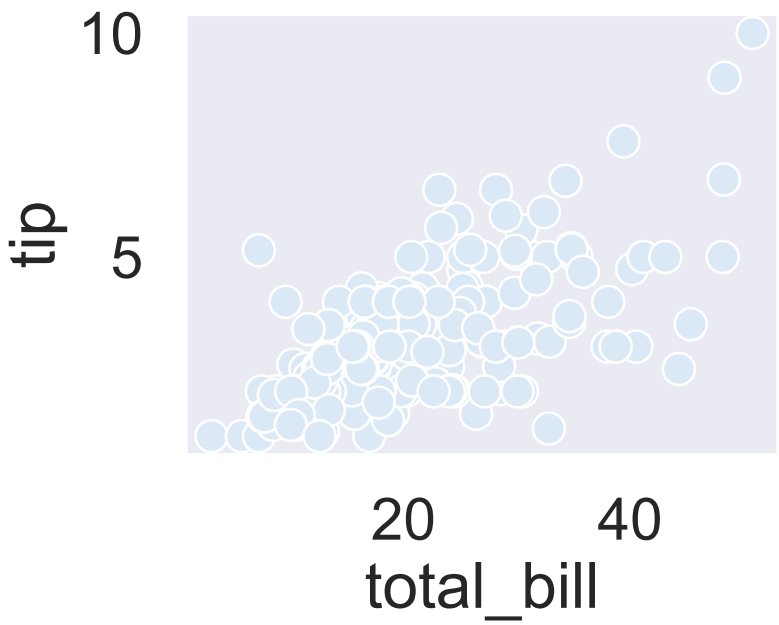


Figure 2: Relationship Between Total Bill and Tip.

A scatter plot with hue

You can set the `"hue"` parameter equal to the DataFrame column name `"smoker"` and then Seaborn will automatically color each point by whether they are a smoker. Plus, it will add a legend to the plot automatically! If you don't want to use pandas, you can set it equal to a list of values instead of a column name.

```
# Import libraries
import seaborn as sns
import matplotlib.pyplot as plt

# Load the built-in 'tips' dataset
df = sns.load_dataset("tips")

# Create scatter plot with 'hue' parameter
sns.scatterplot(x="total_bill", y="tip", hue="smoker", data=df)

# Display the plot
plt.show()
```

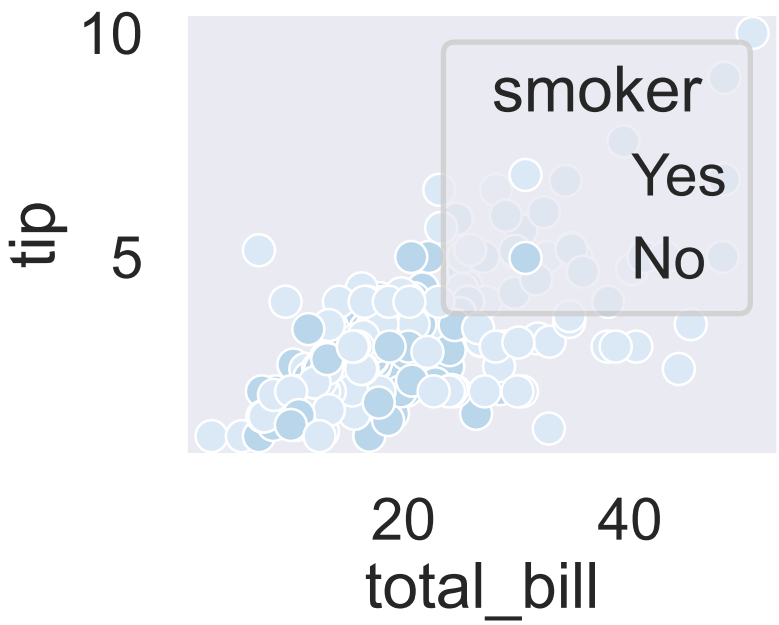


Figure 3: Relationship Between Total Bill and Tip by Smoking Status.

Setting hue order

Hue also allows you to assert more control over the ordering and coloring of each value. The "hue order" parameter takes in a list of values and will set the order of the values in the plot accordingly. Notice how the legend for smoker now lists `"yes"` before `"no"`.

```
# Import libraries
import seaborn as sns
import matplotlib.pyplot as plt

# Load the built-in 'tips' dataset
df = sns.load_dataset("tips")

# Create scatter plot with controlled hue order
sns.scatterplot(
    x="total_bill",
    y="tip",
    hue="smoker",
    hue_order=["Yes", "No"],
    data=df
)
```

```
# Show the plot
plt.show()
```

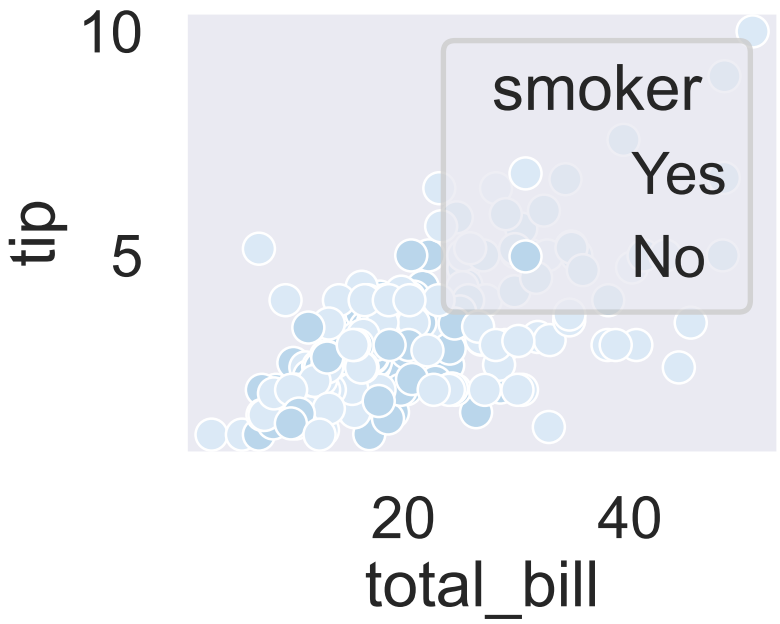


Figure 4: Relationship Between Total Bill and Tip by Smoking Status (Hue Order Set).

Specifying `hue` Colors

You can also control the specific colors assigned to each category by using the **`palette`** parameter. This parameter accepts a **dictionary**, which maps each unique value of the **`hue`** variable to a desired color. For instance, we can create a dictionary called **`hue_colors`** that maps **`"Yes"`** to **black** and **`"No"`** to **red**. Then, by setting **`hue="smoker"`** and **`palette=hue_colors`**, we'll produce a scatter plot where **smokers** are represented by black dots and **non-smokers** by red dots.

```
# Import libraries
import seaborn as sns
import matplotlib.pyplot as plt

# Load the built-in 'tips' dataset
df = sns.load_dataset("tips")

# Define custom color mapping for hue
hue_colors = {"Yes": "black", "No": "red"}
```

```
# Create scatter plot with custom hue colors
sns.scatterplot(
    x="total_bill",
    y="tip",
    hue="smoker",
    palette=hue_colors,
    data=df
)

# Display the plot
plt.show()
```

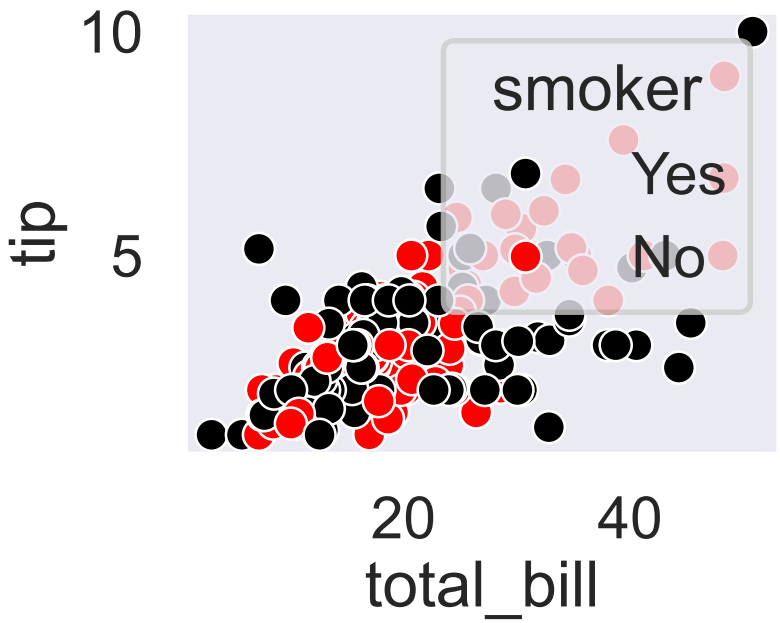
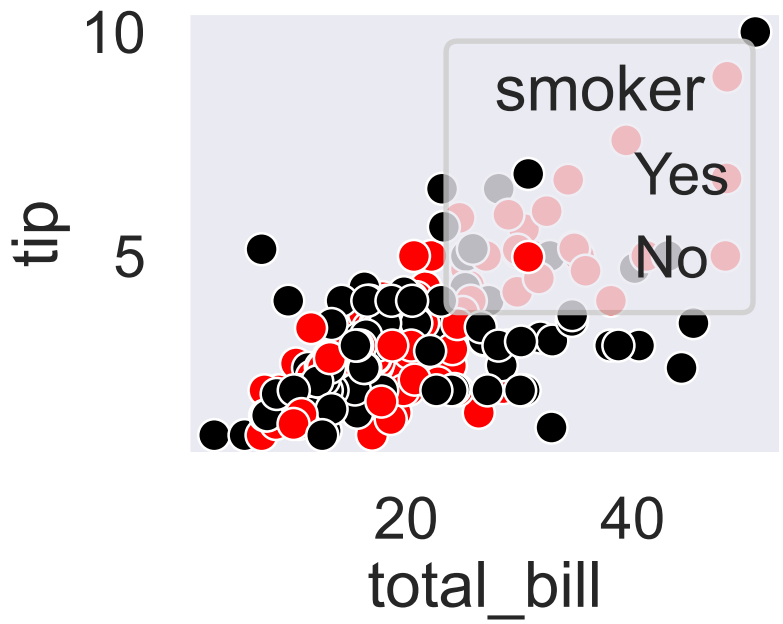


Figure 5: Relationship Between Total Bill and Tip by Smoking Status (Custom Order Set).

Color options

In the last example, we used the words “black” and “red” to define what the hue colors should be. This only works for a small set of color names that are defined by Matplotlib. Here is the list of Matplotlib colors and their names. Note that you can use a single-letter Matplotlib abbreviation instead of the full name. You can also use an HTML color hex code instead of these Matplotlib color names, which allows you to choose any color you want to.

```python
# Import libraries
import seaborn as sns
import matplotlib.pyplot as plt

# Load the built-in 'tips' dataset
df = sns.load_dataset("tips")

# Define custom color mapping for hue
hue_colors = {"Yes": "k", "No": "r"}

# Create scatter plot with custom hue colors
sns.scatterplot(
    x="total_bill",
    y="tip",
    hue="smoker",
    palette=hue_colors,
    data=df
)

# Display the plot
plt.show()
```



Using HTML hex color codes with hue

Here's an example using HTML hex codes. Make sure you put the hex codes in quotes with a pound sign at the beginning.

```
# Define colors using HTML hex codes

hue_colors = {"Yes": "#000000", "No": "#FF0000"}

# Create scatter plot with hex colors

sns.scatterplot(
x="total_bill",
y="tip",
hue="smoker",
palette=hue_colors,
data=df
)

plt.show()
```

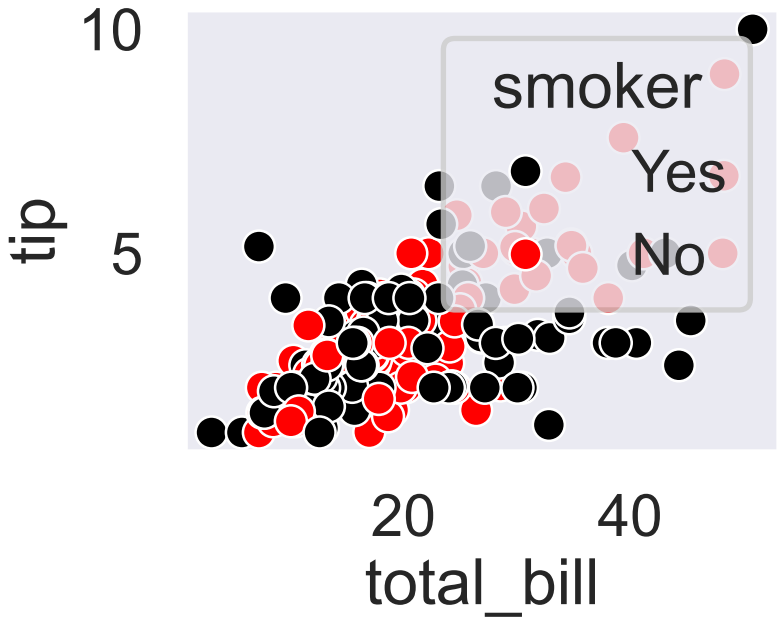


Figure 6: Total Bill vs Tip — Custom Hex Color Palette.

Using hue with count plots

As a final note, hue is available in most of Seaborn's plot types. For example, this count plot shows the number of observations we have for smokers versus non-smokers, and setting `"hue"` equal to `"sex"` divides these bars into subgroups of males versus females. From this plot, we can see that males outnumber females among both smokers and non-smokers in this dataset.

```
# Create count plot with hue for 'sex'
sns.countplot(x="smoker", hue="sex", data=df)

# Show figure
plt.show()
```

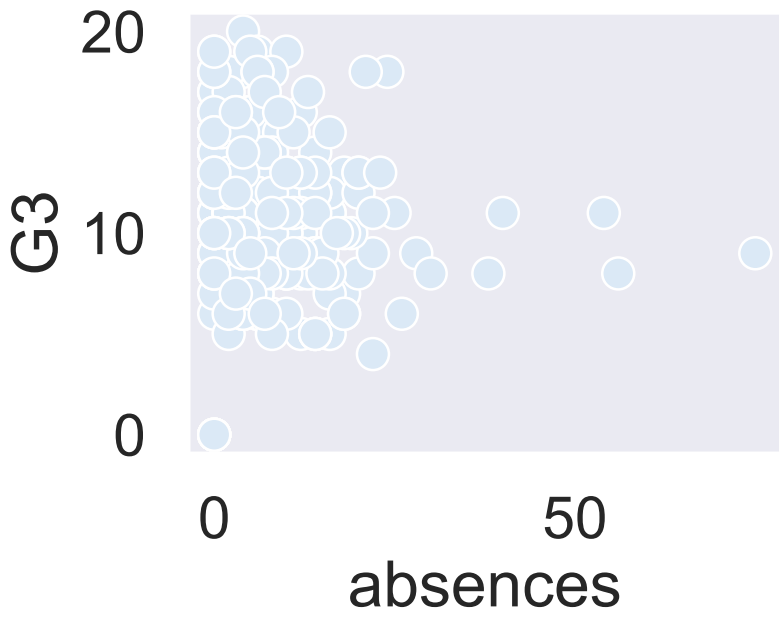
Exercise 1.3

In the prior lesson, we learned how hue allows us to easily make subgroups within Seaborn plots. Let's try it out by exploring data from students in secondary school. We have a lot of information about each student like their age, where they live, their study habits and their extracurricular activities.

1. Create a scatter plot of `absences` vs. `final grade`.
2. Show plot.

```
# Import packages
import matplotlib.pyplot as plt
import seaborn as sns
```

```
import pandas as pd

# Import dataset
student_data = pd.read_csv('datasets/student-alcohol-consumption.csv')

# Create a scatter plot of absences vs. final grade
sns.scatterplot(x= "absences", y="G3", data=student_data)

# Show plot
plt.show()
```

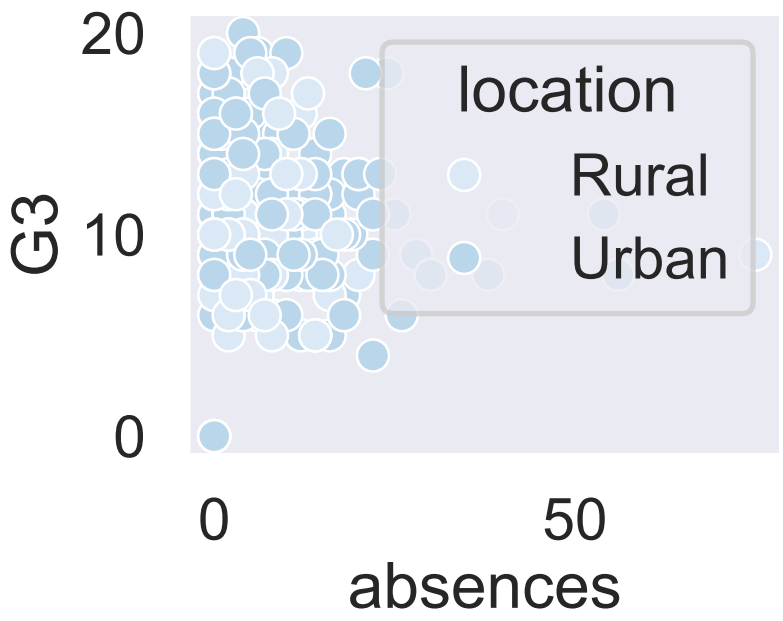


Exercise 1.3.2

---

1. Create a scatter plot with `"absences"` on the x-axis and final grade (`"G3"`) on the y-axis using the DataFrame `student_data`. Color the plot points based on `"location"` (urban vs. rural).
2. Make `"Rural"` appear before `"Urban"` in the plot legend.

```
# Change the legend order in the scatter plot
sns.scatterplot(x="absences", y="G3",
                data=student_data,
                hue="location", hue_order=["Rural", "Urban"])
plt.show()
```



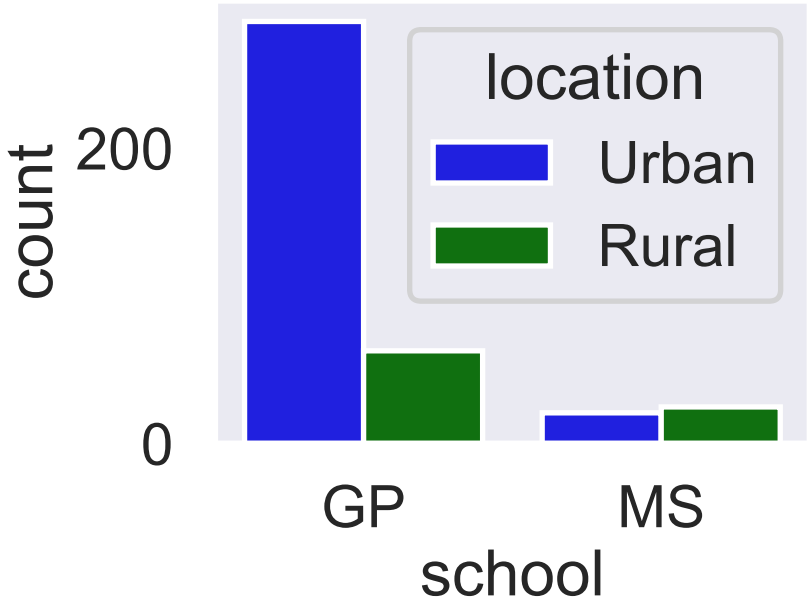
Exercise 1.3.3

---

1. Fill in the `palette_colors` dictionary to map the `"Rural"` location value to the color `"green"` and the `"Urban"` location value to the color `"blue"`.
2. Create a count plot with `"school"` on the x-axis using the `student_data` DataFrame.
3. Add subgroups to the plot using `"location"` variable and use the `palette_colors` dictionary to make the `location` subgroups `green` and `blue`.

```
# Import Matplotlib, Seaborn, and Pandas
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Import dataset
student_data = pd.read_csv('datasets/student-alcohol-consumption.csv')

# Create a dictionary mapping subgroup values to colors
palette_colors = {"Rural": "green", "Urban": "blue"}

# Create a count plot of school with location subgroups
sns.countplot(
    x="school",
    hue="location",
```

```
    data=student_data,
    palette=palette_colors
)

# Display plot
plt.show()
```



Chapter 2.1: Introduction to relational plots and subplots

Many questions in data science are centered around describing the relationship between two quantitative variables. Seaborn calls plots that visualize this relationship `"relational plots"`.

Questions about quantitative variables

So far we've seen several examples of questions about the relationship between two quantitative variables, and we answered them with scatter plots. These examples include: `"do taller people tend to weigh more?"`, `"what's the relationship between the number of absences a student has and their final grade?"` and `"how does a country's GDP relate to the percent of the population that can read and write?"` Because they look at the relationship between two quantitative variables, these scatter plots are all considered relational plots.

Visualizing subgroups

While looking at a relationship between two variables at a high level is often informative, sometimes we suspect that the relationship may be different within certain subgroups. In the last chapter, Section , we started to look at subgroups by using the `"hue"` parameter to visualize each subgroup using a different color on the same plot.

Visualizing subgroups

In this lesson, we'll try out a different method: creating a separate plot per subgroup.

Introducing relplot()

To do this, we're going to introduce a new Seaborn function: `"relplot()"`. `"relplot()"` stands for `"relational plot"` and enables you to visualize the relationship between two quantitative variables using either scatter plots or line plots. You've already seen scatter plots, and you'll learn about line plots later in this chapter. Using `"relplot()"` gives us a big advantage: the ability to create subplots in a single figure. Because of this advantage, we'll be using `"relplot()"` instead of `"scatterplot()"` for the rest of the course.

scatterplot() vs. relplot()

Let's return to our scatter plot of total bill versus tip amount from the tips dataset. In the last lesson, Figure 2, we see how to create a scatter plot with the `"scatterplot"` function. To make it with `"relplot()"` instead, we change the function name to `"relplot()"` and use the `"kind"` parameter to specify what kind of relational plot to use - scatter plot or line plot. In this case, we'll set kind equal to the word `"scatter"`.

```
# Import libraries
import seaborn as sns
import matplotlib.pyplot as plt

# Load the built-in 'tips' dataset
df = sns.load_dataset("tips")


# Change the scatterplot to relplot
sns.relplot(x="total_bill", y="tip",
            data=df,
            kind="scatter")


# Adjust layout and display
plt.show()
```

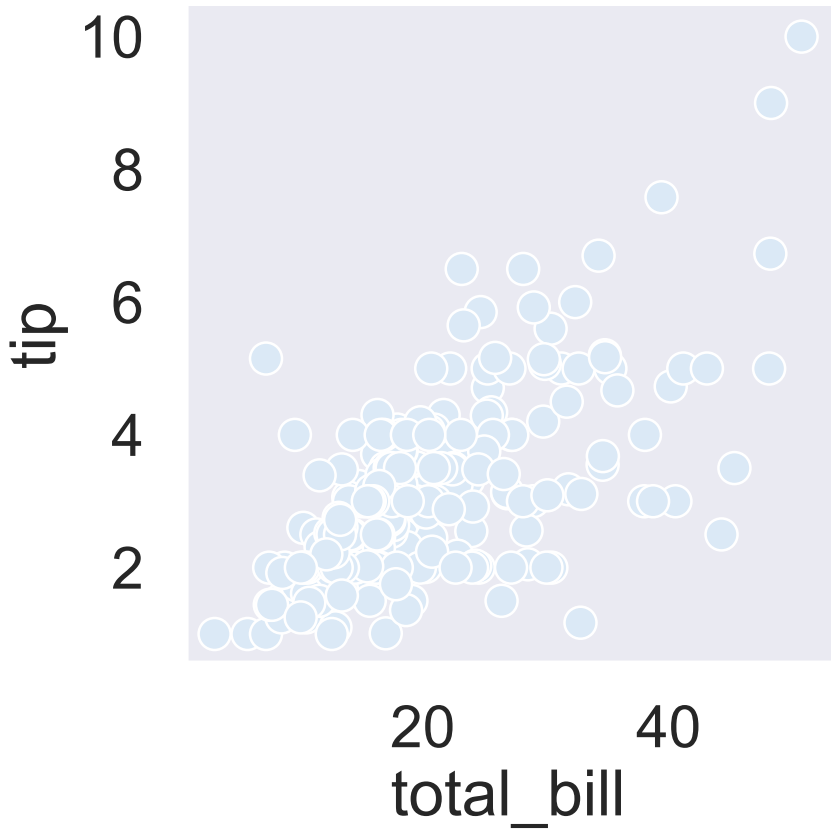


Figure 7: Total bill Vs Tip.

Subplots in columns

By setting “col” equal to “smoker”, we get a separate scatter plot for smokers and non-smokers, arranged horizontally in columns.

```
# Import libraries
import seaborn as sns
import matplotlib.pyplot as plt

# Load the built-in 'tips' dataset
df = sns.load_dataset("tips")


# Change the scatterplot to relplot
sns.relplot(x="total_bill", y="tip",
            data=df,
            kind="scatter", col = "smoker")


# Adjust layout and display
plt.show()
```

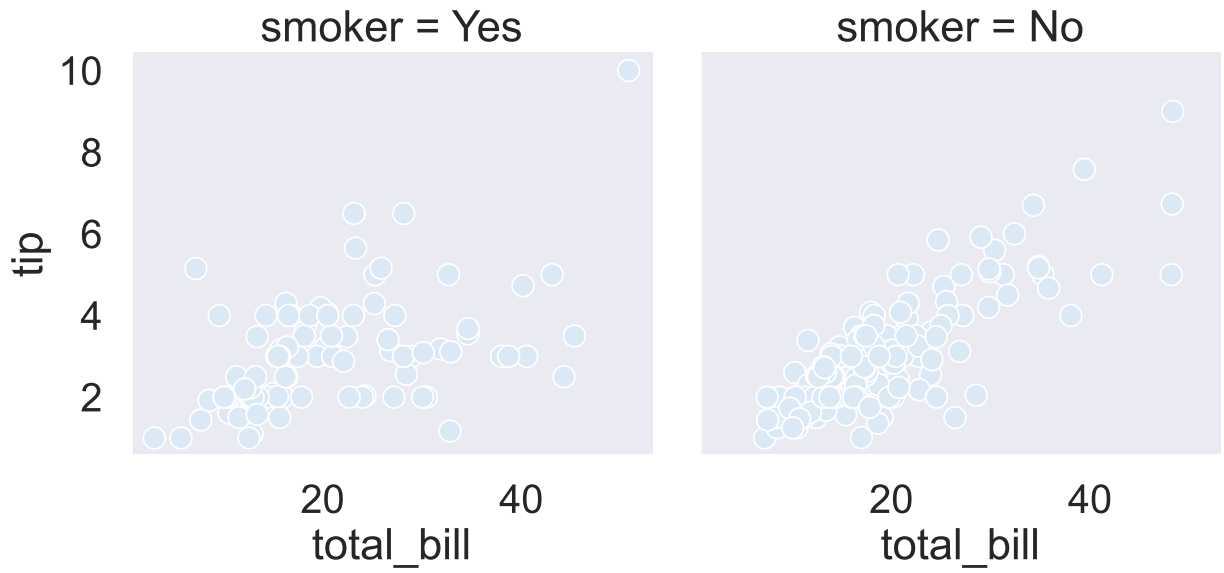


Figure 8: Total bill Vs Tip by Smoking Status.

Subplots in rows

If you want to arrange these vertically in rows instead, you can use the “row” parameter instead of “col”.

```
# Import libraries
import seaborn as sns
import matplotlib.pyplot as plt

# Load the built-in 'tips' dataset
df = sns.load_dataset("tips")


# Change the scatterplot to relplot
sns.relplot(x="total_bill", y="tip",
            data=df,
            kind="scatter", row = "smoker")


# Adjust layout and display
plt.show()
```

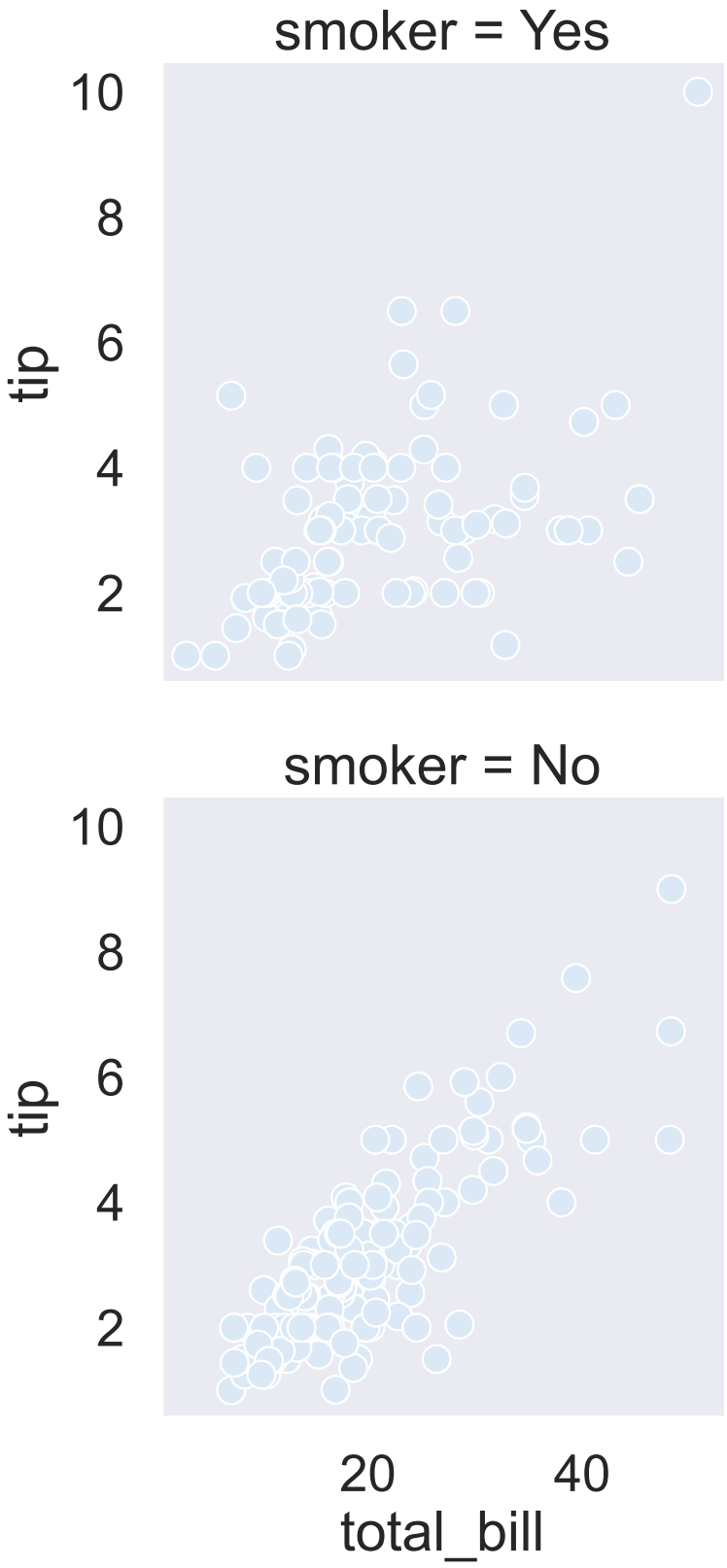


Figure 9: Total bill Vs Tip by Smoking Status.

Subplots in rows and columns

It is possible to use both `"col"` and `"row"` at the same time. Here, we set `"col"` equal to smoking status and `"row"` equal to the `time` of day (lunch or dinner). Now we have a subplot for each combination of these two categorical variables.

```
# Import libraries
import seaborn as sns
import matplotlib.pyplot as plt

# Load the built-in 'tips' dataset
df = sns.load_dataset("tips")


# Change the scatterplot to relplot
sns.relplot(x="total_bill", y="tip",
            data=df,
            kind="scatter", col = "smoker", row = "time")

# Adjust layout and display
plt.show()
```

Figure 10: Total bill Vs Tip by Smoking Status and Time of the day.

Subgroups for days of the week

As another example, let's look at subgroups based on day of the week. There are four subplots here, which can be a lot to show in a single row. To address this, you can use the `"col_wrap"`

parameter to specify how many subplots you want per row. Here, we set "col_wrap" equal to two plots per row.

```
# Import libraries
import seaborn as sns
import matplotlib.pyplot as plt

# Load the built-in 'tips' dataset
df = sns.load_dataset("tips")

sns.relplot(
    x="total_bill",
    y="tip",
    col="time",
    kind="scatter",
    data=df,
    col_wrap=2)

# Adjust layout and display
plt.show()
```

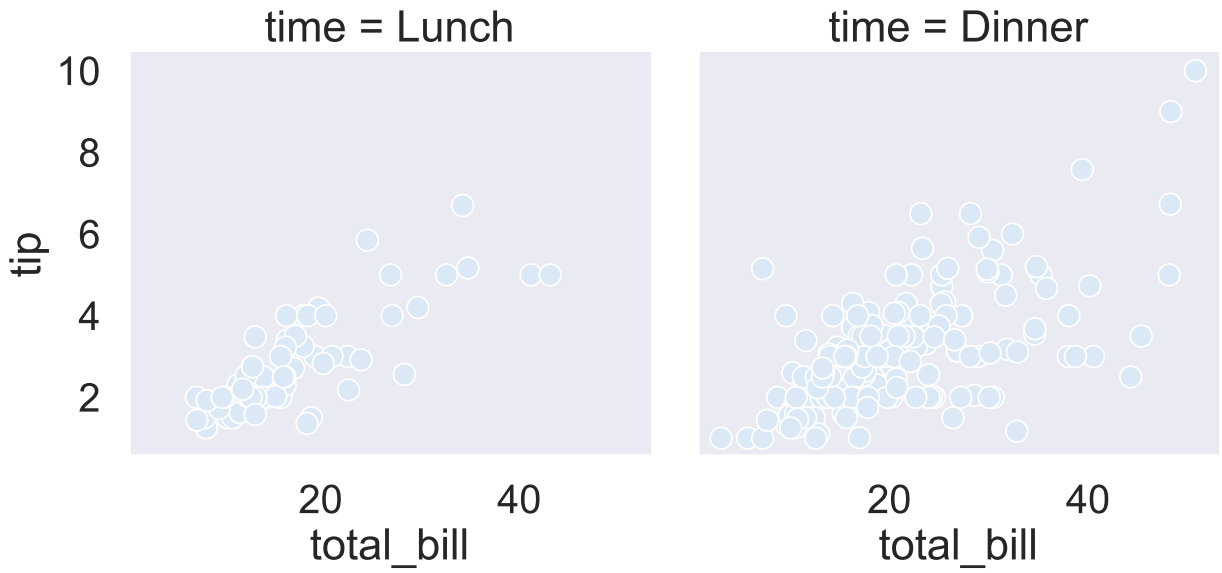


Figure 11: Total bill Vs Tip by Smoking Status and Time of the day.

Ordering columns

We can also change the order of the subplots by using the `"col_order"` and `"row_order"` parameters and giving it a list of ordered values.

```
# Import libraries
import seaborn as sns
import matplotlib.pyplot as plt

# Load the built-in 'tips' dataset
df = sns.load_dataset("tips")

sns.relplot(
x="total_bill",
y="tip",
col="time", # facet columns by time of day
row="smoker", # facet rows by smoking status
col_order=["Dinner", "Lunch"], # order of time categories
row_order=["Yes", "No"], # order of smoking categories
kind="scatter",
data=df,
height=4,
aspect=1.2
)

# Adjust layout and display
plt.show()
```

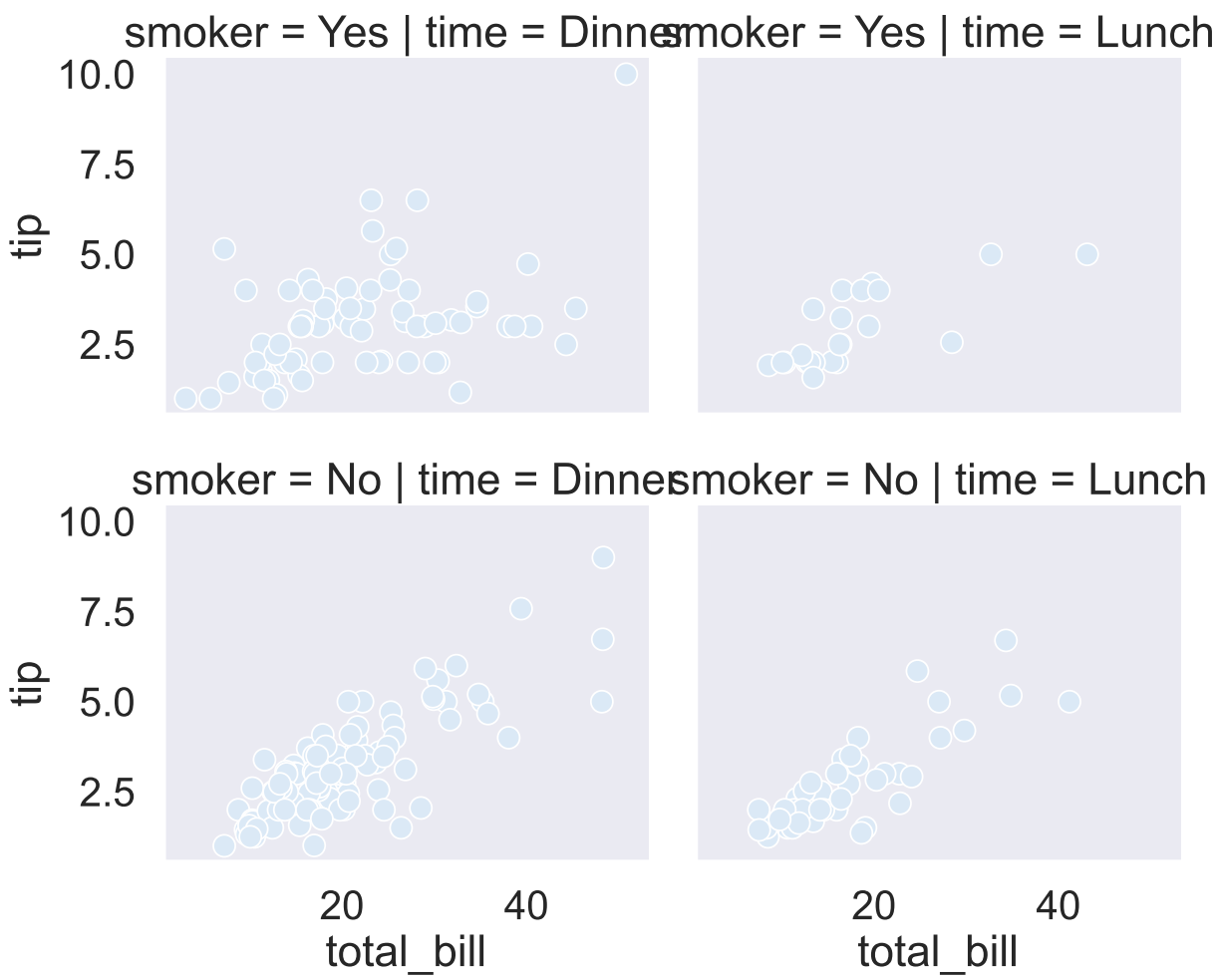


Figure 12: Total bill Vs Tip by Smoking Status and Time of the day.

Exercise 2.1

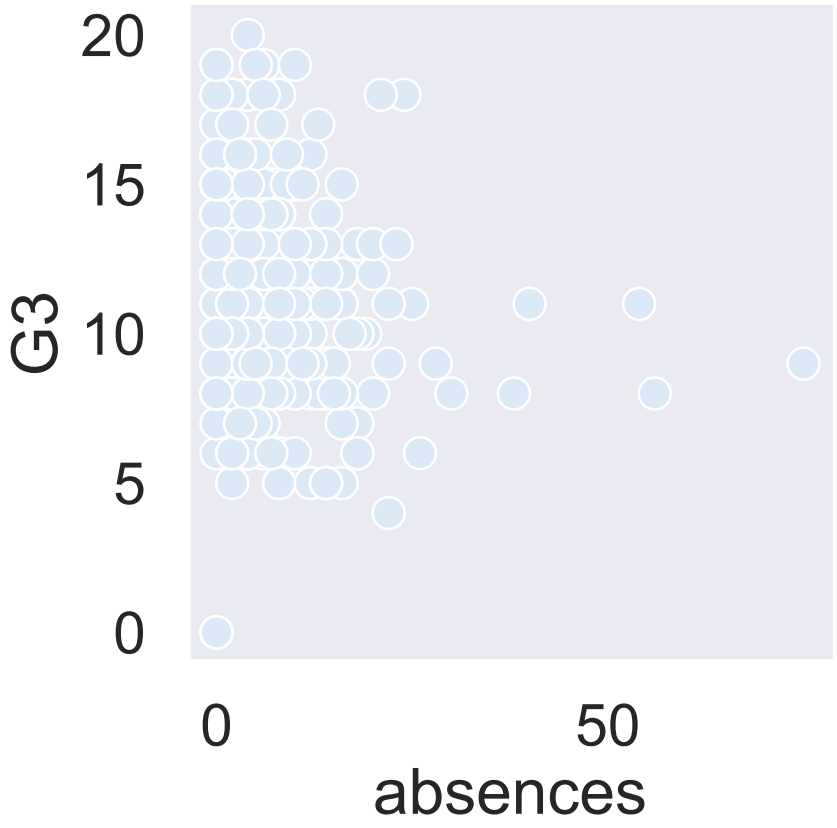
---

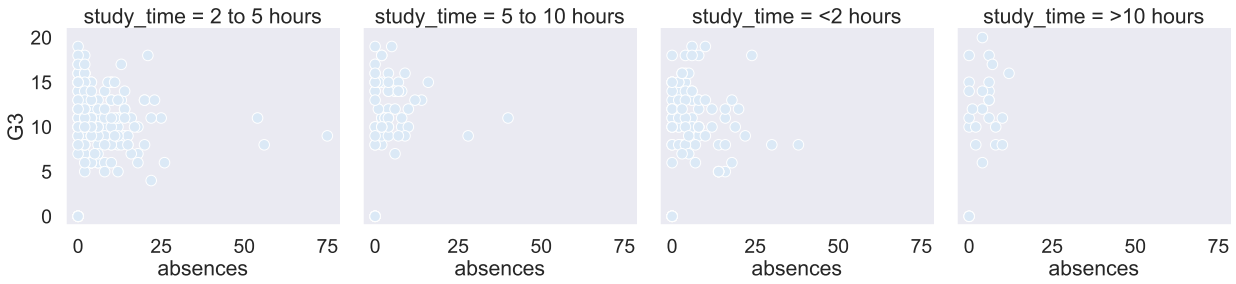
1. Change to use `relplot()` instead of `scatterplot()`.
2. Modify the code to create one scatter plot for each level of the variable `"study_time"`, arranged in columns.
3. Adapt your code to create one scatter plot for each level of a student's weekly study time, this time arranged in rows.
4. Use `relplot()` to create a scatter plot with `"G1"` on the x-axis and `"G3"` on the y-axis, using the `student_data` DataFrame.
5. Create column subplots based on whether the student received support from the school (`"schoolsup"`), ordered so that `"yes"` comes before `"no"`.

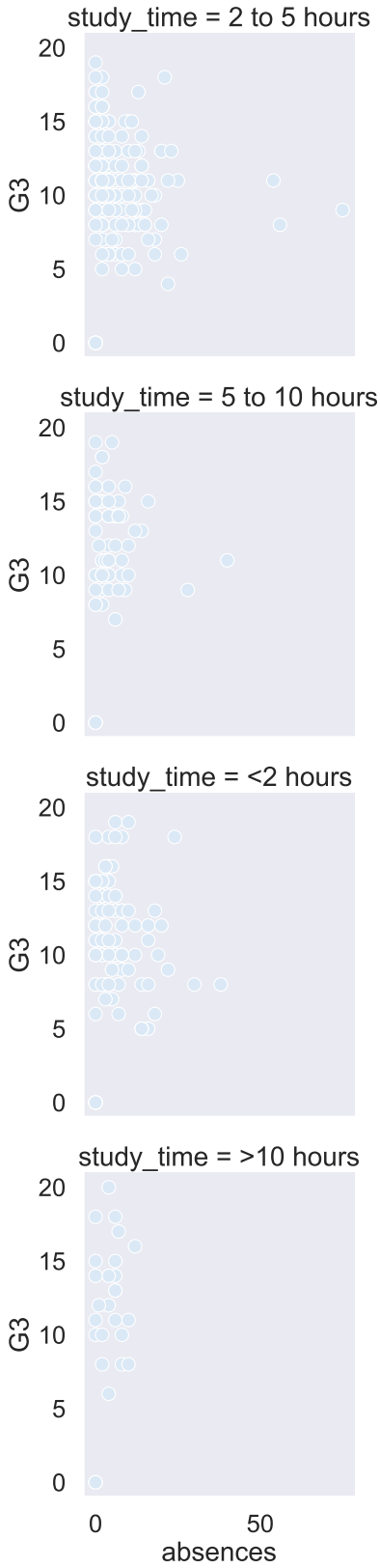
6. Add row subplots based on whether the student received support from the family (`"famsup"`), ordered so that `"yes"` comes before `"no"`. This will result in subplots based on two factors.

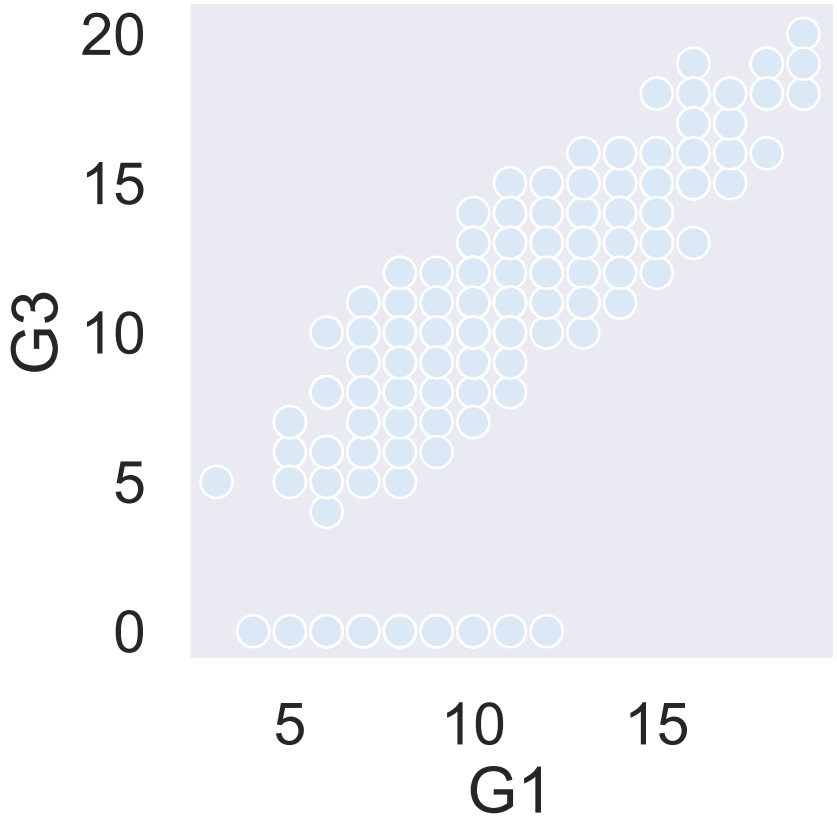
```
# Import packages
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Import dataset
student_data = pd.read_csv('datasets/student-alcohol-consumption.csv')

# Change to use relplot() instead of scatterplot()
sns.relplot(x="absences", y="G3",
                data=student_data, kind="scatter")

# Show plot
plt.show()

# Modify the code to create one scatter plot for each level of the variable "study_time", arrange
sns.relplot(x="absences", y="G3",
            data=student_data,
            kind="scatter", col="study_time")

# Show plot
plt.show()

# Adapt your code to create one scatter plot for each level of a student's weekly study time, thi
sns.relplot(x="absences", y="G3",
            data=student_data,
            kind="scatter",
            row="study_time")

# Show plot
plt.show()

# Use relplot() to create a scatter plot with "G1" on the x-axis and "G3" on the y-axis, using th
sns.relplot(x="G1", y="G3", data=student_data, kind="scatter")

# Show plot
plt.show()

# Create column subplots based on whether the student received support from the school ("schoolsu
sns.relplot(x="G1", y="G3",
            data=student_data,
            kind="scatter", col = "schoolsup", col_order=["yes", "no"])
```

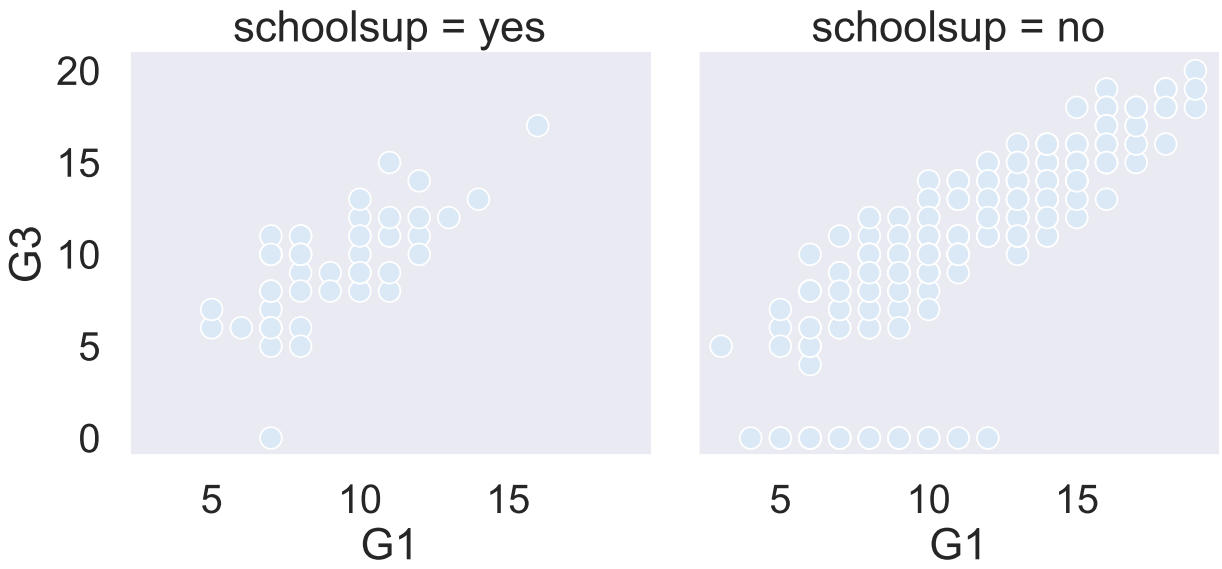
```
# Show plot
plt.show()

# Add row subplots based on whether the student received support from the family ("famsup"), orde
sns.relplot(x="G1", y="G3",
            data=student_data,
            kind="scatter",
            col="schoolsup",
            col_order=["yes", "no"],
            row="famsup",
            row_order=["yes", "no"])

# Show plot
plt.show()
```




study_time = 2 to 5 hours
study_time = 5 to 10 hours
study_time = <2 hours
study_time = >10 hours
G3
20
15
10
5
0
0
25
50
75
absences


study_time = 2 to 5 hours
study_time = 5 to 10 hours
study_time = <2 hours
study_time = >10 hours
G3
0
5
10
15
20
0
50
absences


20
15
10
5
0
G3
5
10
15
G1


schoolsup = yes
schoolsup = no
20
15
10
5
0
G3
5
10
15
G1
5
10
15
G1

Chapter 2.2: Customizing scatter plots

So far, we've only scratched the surface of what we're able to do with scatter plots in Seaborn.

Scatter plot overview

As a reminder, scatter plots are a great tool for visualizing the relationship between two quantitative variables. We've seen a few ways to add more information to them as well, by creating subplots or plotting subgroups with different colored points. In addition to these, Seaborn allows you to add more information to scatter plots by varying the size, the style, and the transparency of the points. All of these options can be used in both the `"scatterplot()"` and `"relplot()"` functions, but we'll continue to use `"relplot()"` for the rest of the course since it's more flexible and allows us to create subplots. For the rest of this lesson, we'll use the tips dataset to learn how to use each customization and cover best practices for deciding which customizations to use.

Subgroups with point size

The first customization we'll talk about is point size. Here, we're creating a scatter plot of total bill versus tip amount. We want each point on the scatter plot to be sized based on the number of people in the group, with larger groups having bigger points on the plot. To do this, we'll set the `"size"` parameter equal to the variable name `"size"` from our dataset. As this example demonstrates, varying point size is best used if the variable is either a quantitative variable or a categorical variable that represents different levels of something, like `"small"`, `"medium"`, and `"large"`. This plot is a bit hard to read because all of the points are of the same color.

```
# Import libraries
import seaborn as sns
import matplotlib.pyplot as plt

# Load the built-in 'tips' dataset
df = sns.load_dataset("tips")


# Change the scatterplot to relplot
sns.relplot(x="total_bill", y="tip",
            data=df,
            kind="scatter", size = "size")


# Adjust layout and display
plt.show()
```

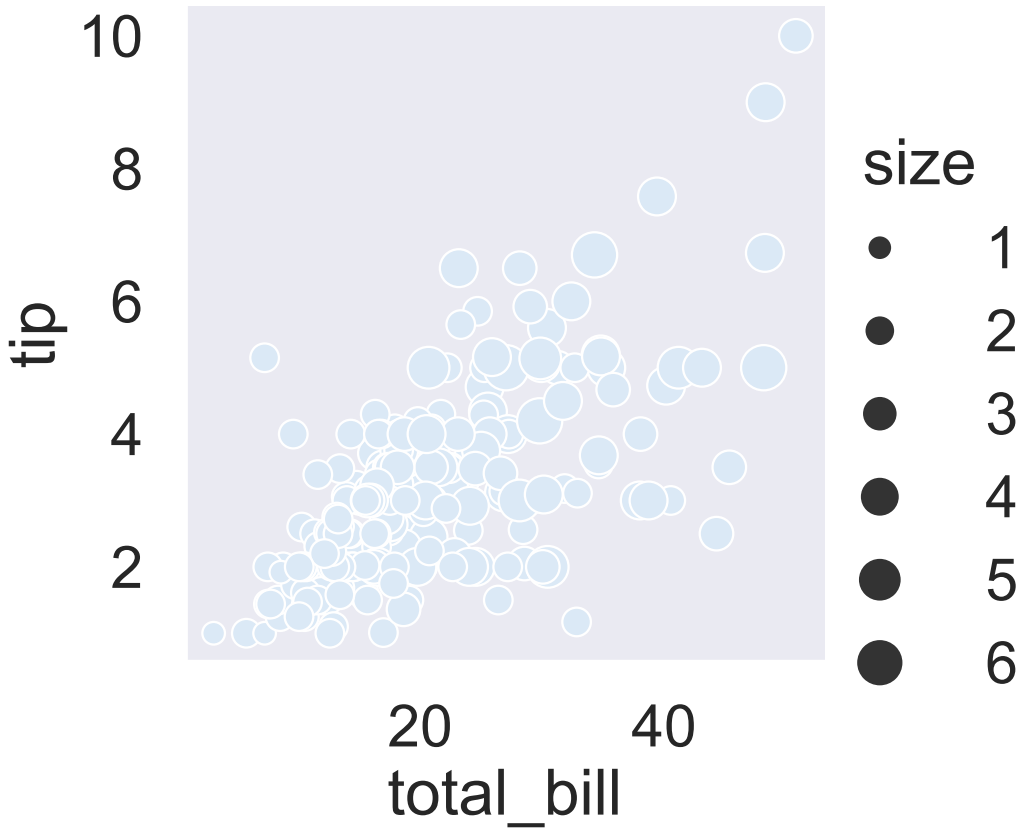


Figure 13: Total bill Vs Tip by Size.

Point size and hue

We can make it easier by using the `"size"` parameter in combination with the `"hue"` parameter. To do this, set `"hue"` equal to the variable name `"size"`. Notice that because `"size"` is a quantitative variable, Seaborn will automatically color the points different shades of the same color instead of different colors per category value like we saw in previous plots. Now larger groups have both larger and darker points, which provides better contrast and makes the plot easier to read.

```
# Import libraries
import seaborn as sns
import matplotlib.pyplot as plt

# Load the built-in 'tips' dataset
df = sns.load_dataset("tips")
```

```
# Change the scatterplot to relplot
sns.relplot(x="total_bill", y="tip",
            data=df,
            kind="scatter", size = "size", hue = "size")


# Adjust layout and display
plt.show()
```

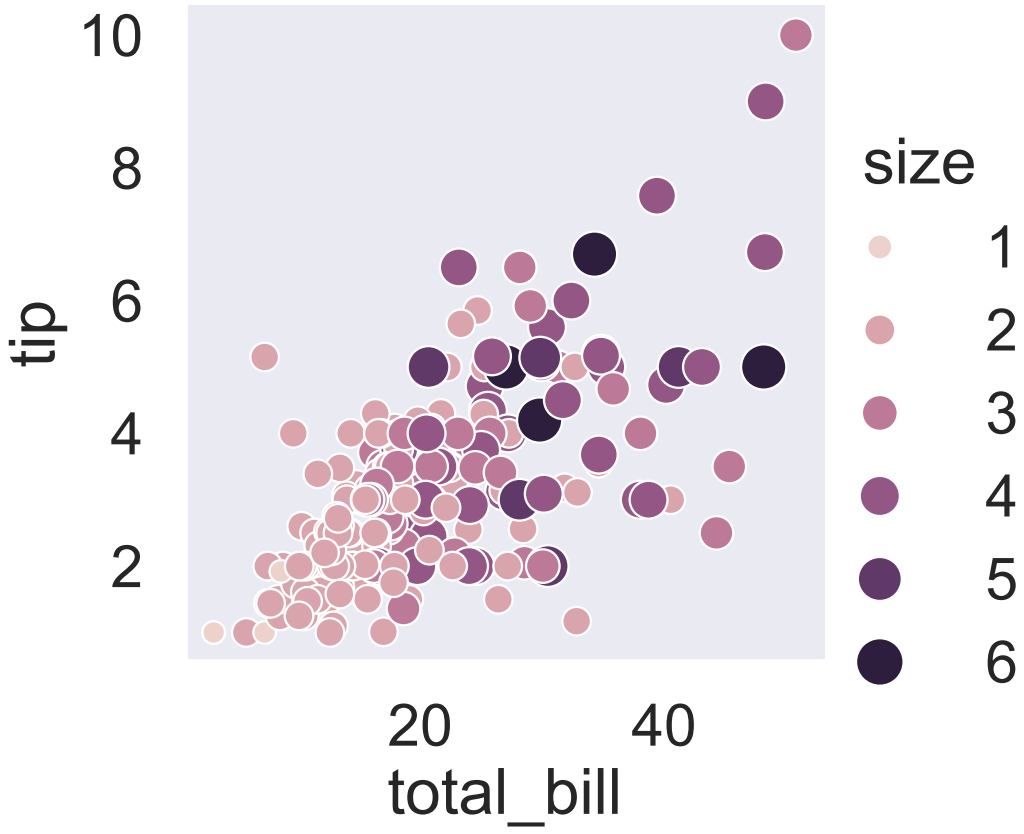


Figure 14: Total bill Vs Tip by Size.

Subgroups with point style

The next customization we'll look at is the point style. Setting the `"style"` parameter to a variable name will use different point styles for each value of the variable. Here's a scatter plot we've seen before, where we use `"hue"` to create different colored points based on smoking status. Setting

"style" equal to "smoker" allows us to better distinguish these subgroups by plotting smokers with a different point style in addition to a different color.

```
# Import libraries
import seaborn as sns
import matplotlib.pyplot as plt

# Load the built-in 'tips' dataset
df = sns.load_dataset("tips")


# Change the scatterplot to relplot
sns.relplot(x="total_bill", y="tip",
            data=df,
            kind="scatter", size = "size", hue = "size",
style = "smoker")


# Adjust layout and display
plt.show()
```

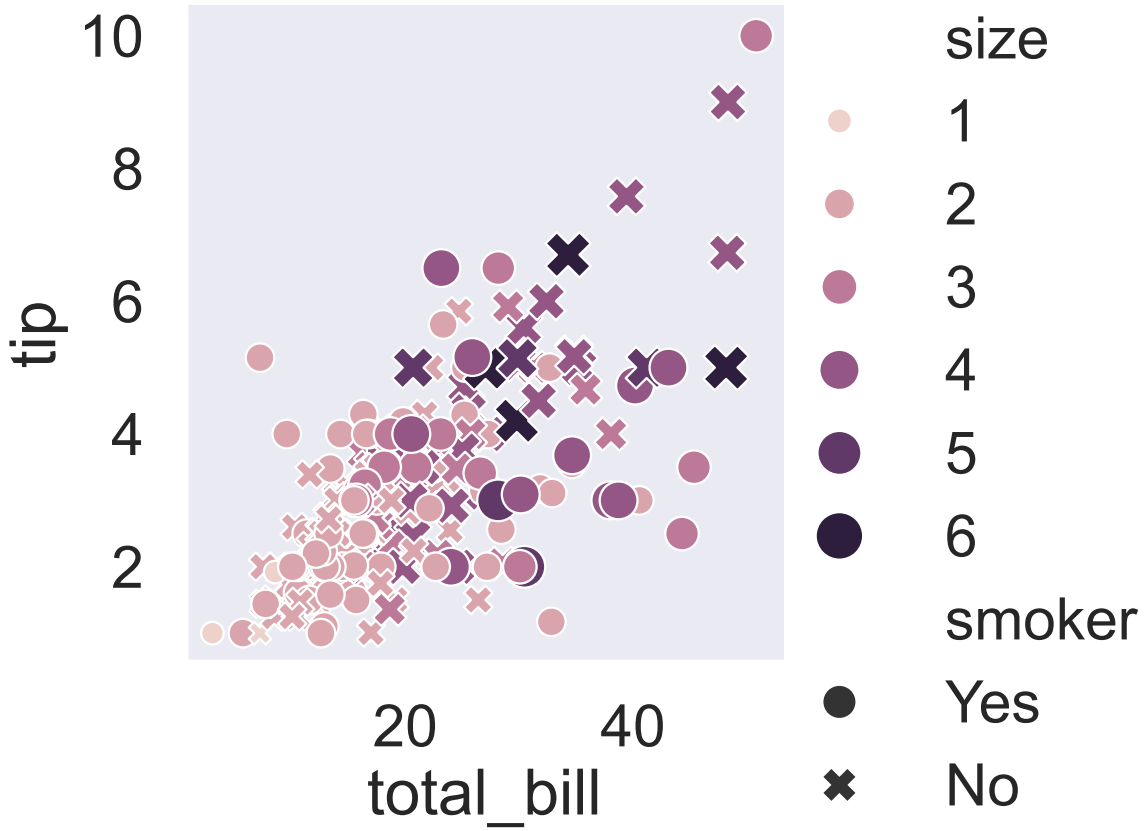


Figure 15: Total bill Vs Tip by Size and Smoking Status.

Changing point transparency

The last customization we'll look at is point transparency. Setting the `"alpha"` parameter to a value between 0 and 1 will vary the transparency of the points in the plot, with 0 being completely transparent and 1 being completely non-transparent. Here, we've set `"alpha"` equal to 0.4. This customization can be useful when you have many overlapping points on the scatter plot, so you can see which areas of the plot have more or less observations.

```
# Import libraries
import seaborn as sns
import matplotlib.pyplot as plt

# Load the built-in 'tips' dataset
df = sns.load_dataset("tips")
```

```
# Change the scatterplot to relplot
sns.relplot(x="total_bill", y="tip",
            data=df,
            kind="scatter", size = "size", hue = "size",
style = "smoker", alpha = 0.4)


# Adjust layout and display
plt.show()
```

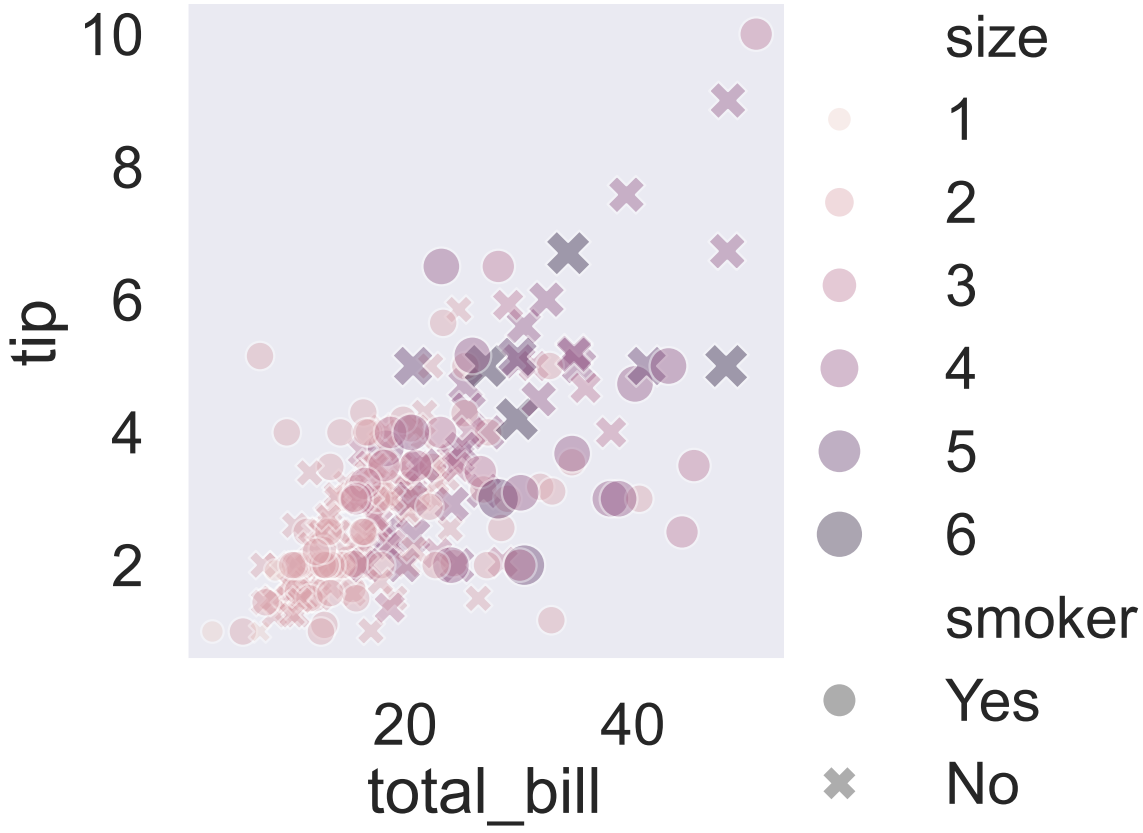


Figure 16: Total bill Vs Tip by Size and Smoking Status.

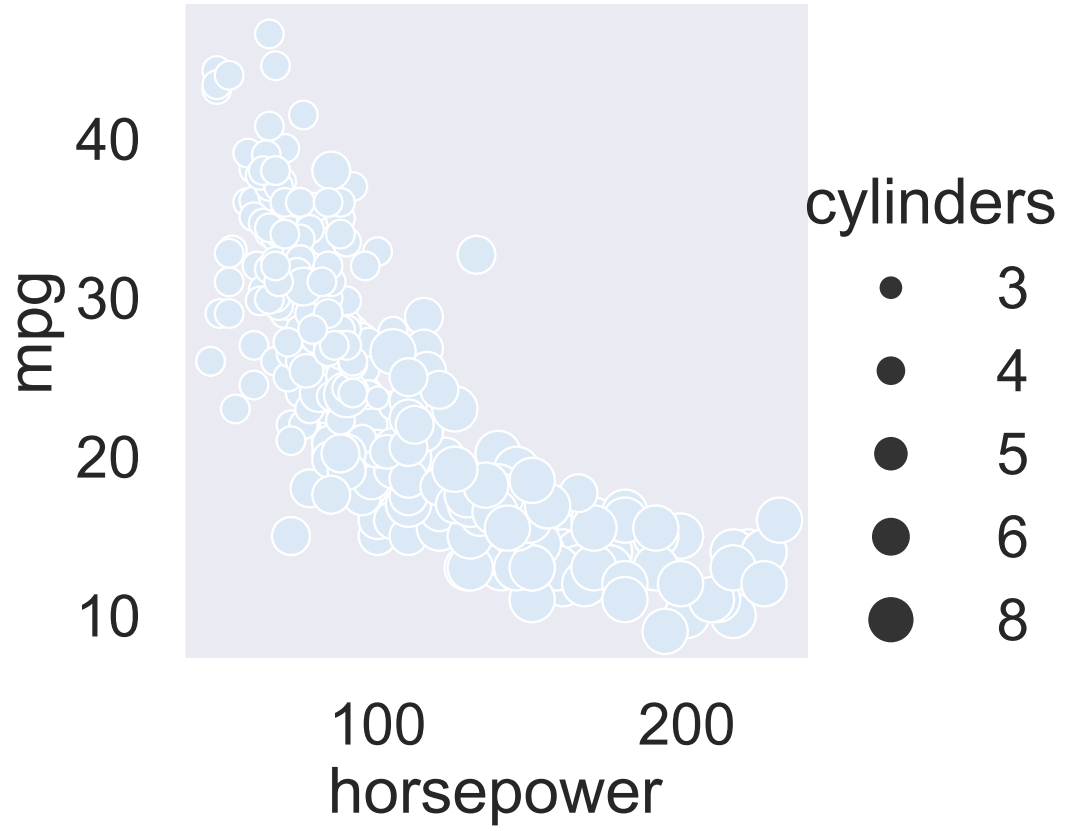
Exercise 2.2

---

1. Use `relplot()` and the `mpg` DataFrame to create a scatter plot with `"horsepower"` on the x-axis and `"mpg"` on the y-axis. Vary the size of the points by the number of cylinders in

the car (`"cylinders"`).

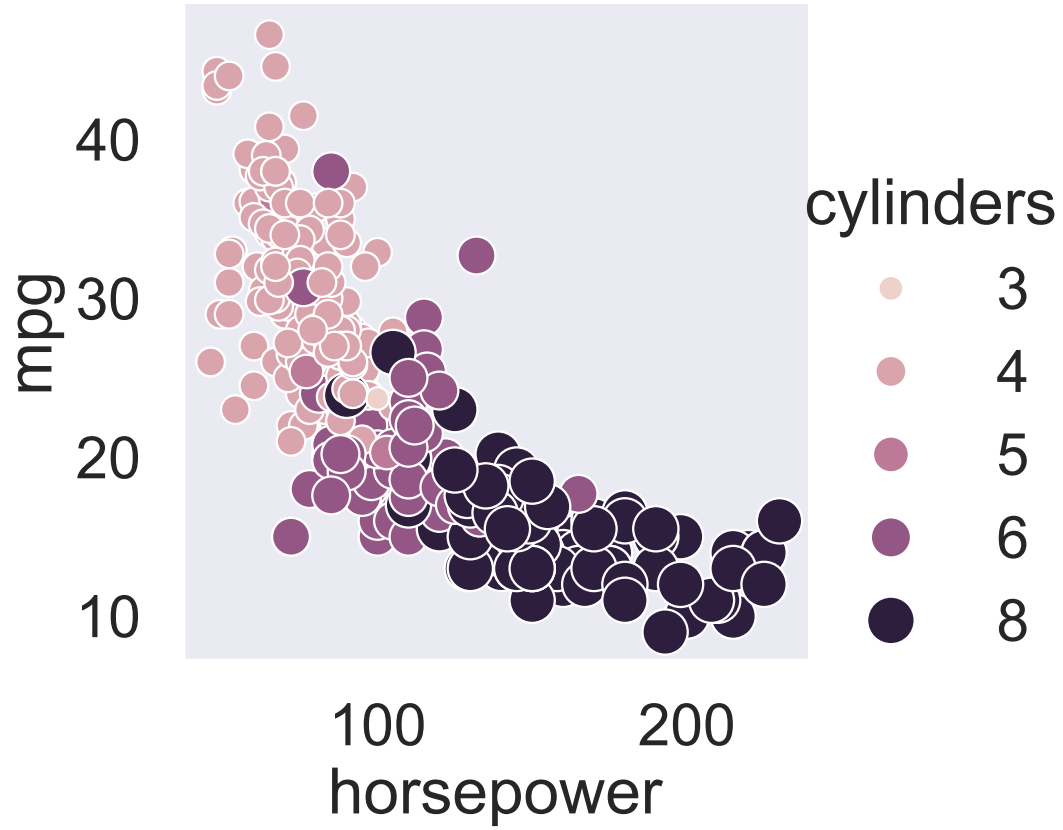
```
# Import packages
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Import dataset
mpg = pd.read_csv("datasets/mpg.csv")

# Create scatter plot of horsepower vs. mpg
sns.relplot(x="horsepower", y="mpg", data=mpg, kind="scatter", size="cylinders")

# Show plot
plt.show()
```



2. To make this plot easier to read, use hue to vary the color of the points by the number of cylinders in the car (`"cylinders"`).

```
# Import packages
import matplotlib.pyplot as plt
```

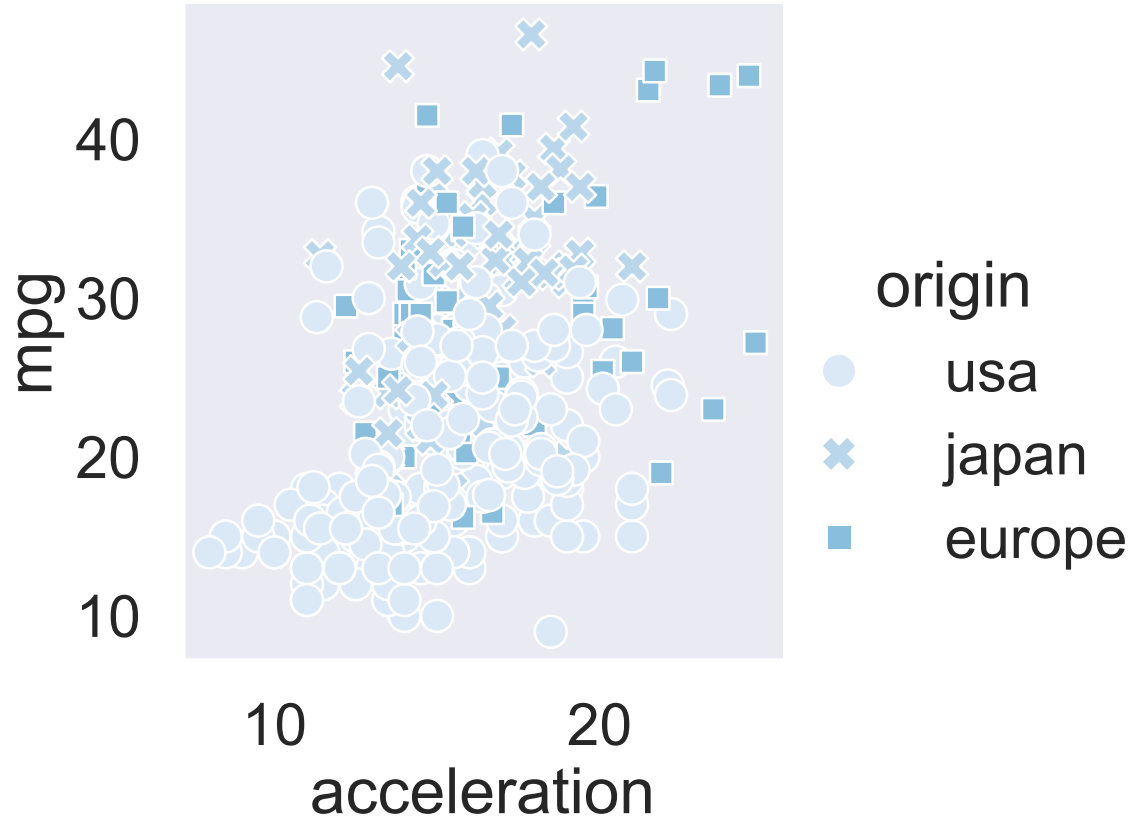
```
import seaborn as sns
import pandas as pd

# Import dataset
mpg = pd.read_csv("datasets/mpg.csv")

sns.relplot(x="horsepower", y="mpg",
            data=mpg, kind="scatter",
            size="cylinders", hue = "cylinders")

# Show plot
plt.show()
```



3. Use `relplot()` and the `mpg` DataFrame to create a scatter plot with `"acceleration"` on the x-axis and `"mpg"` on the y-axis. Vary the style and color of the plot points by country of origin (`"origin"`).

```
# Import packages
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
```

```
# Import dataset
mpg = pd.read_csv("datasets/mpg.csv")

sns.relplot(x="acceleration", y="mpg",
            data=mpg, kind="scatter",
            style="origin", hue = "origin")

# Show plot
plt.show()
```



Chapter 2.3: Introduction to line plots

Hello! In this lesson we'll dive into a new type of relational plot: line plots.

What are line plots?

In Seaborn, we have two types of relational plots: scatter plots and line plots. While each point in a scatter plot is assumed to be an independent observation, line plots are the visualization of choice when we need to track the same thing over time. A common example is tracking the value of a company's stock over time, as shown here.

Air pollution data

In this lesson, we'll be using data on the levels of air pollution in a city. There are many air collection stations around the city, each measuring the nitrogen dioxide level every hour for a single day. Long-term exposure to high levels of nitrogen dioxide can cause chronic lung diseases. Let's begin with the simple case where we have one data point per x-value. Here we have one row per hour over the course of the day with the average nitrogen dioxide level across all the stations in a column called `"NO_2_mean"`.

```
# Import libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Set random seed for reproducibility
np.random.seed(42)

# Define parameters
hours = np.arange(0, 24)
locations = ["North", "South", "East", "West"]
stations = ["Station_A", "Station_B", "Station_C"]

# Generate synthetic NO  data for each location and station
data = []
for loc in locations:
    for st in stations:
        base = np.random.normal(loc=40, scale=5, size=24)
        if loc == "South":
            base += 5  # slightly higher NO  levels in the South
        data.extend([(h, loc, st, v) for h, v in zip(hours, base)])
```

```python
# Create DataFrame
df = pd.DataFrame(data, columns=["hour", "location", "station", "NO_2_mean"])

# Display first few rows
print(df.head())
```

```
   hour location    station  NO_2_mean
0     0    North  Station_A  42.483571
1     1    North  Station_A  39.308678
2     2    North  Station_A  43.238443
3     3    North  Station_A  47.615149
4     4    North  Station_A  38.829233
```

Scatter plot

This is a scatter plot with the **average nitrogen dioxide (NO) level** on the y-axis and the **hour of the day** on the x-axis. Although scatter plots are useful for showing relationships between two variables, we're tracking the *same variable over time*. Therefore, a **line plot** is a better choice for visualizing this pattern clearly.

```python
# Import libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Set random seed for reproducibility
np.random.seed(42)

# Define parameters
hours = np.arange(0, 24)
locations = ["North", "South", "East", "West"]
stations = ["Station_A", "Station_B", "Station_C"]

# Generate synthetic NO  data for each location and station
data = []
for loc in locations:
    for st in stations:
        base = np.random.normal(loc=40, scale=5, size=24)
        if loc == "South":
            base += 5  # slightly higher NO  levels in the South
        data.extend([(h, loc, st, v) for h, v in zip(hours, base)])
```

```
# Create DataFrame
df = pd.DataFrame(data, columns=["hour", "location", "station", "NO_2_mean"])

# Create scatter plot (less ideal for time-based data)
sns.relplot(x="hour", y="NO_2_mean", data=df, kind = "scatter", color="steelblue")
plt.show()
```

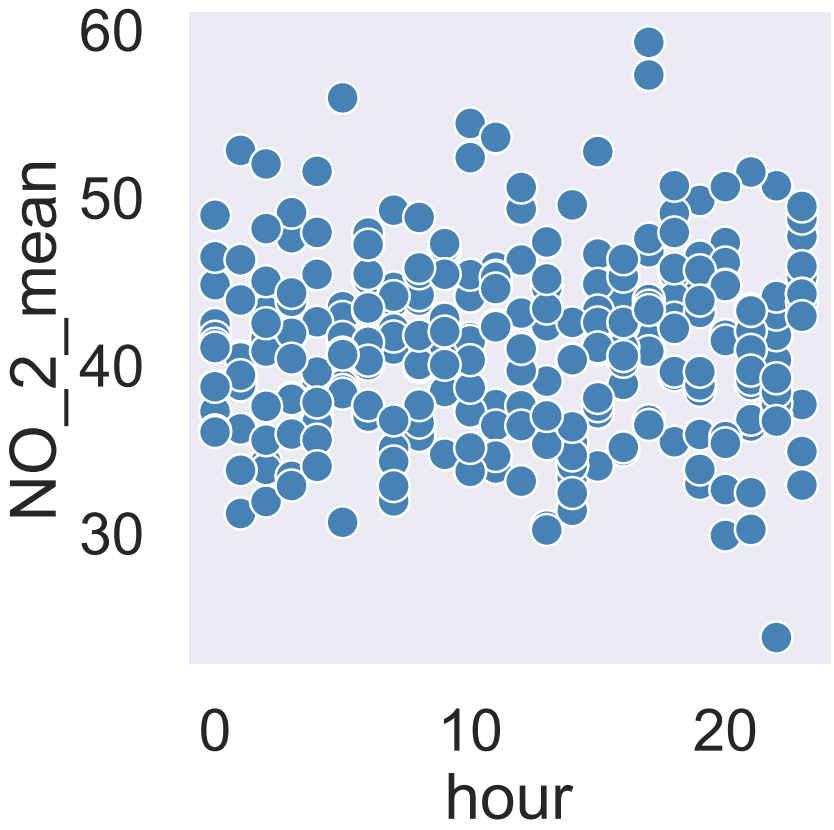


Figure 17: Average Nitrogen Dioxide (NO) Levels by Hour of the Day.

Line plot

By specifying `"kind"` equals `"line"`, we can create a line plot and more easily see how the average nitrogen dioxide level fluctuates throughout the day.

```
# Import libraries
import pandas as pd
```

```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Set random seed for reproducibility
np.random.seed(42)

# Define parameters
hours = np.arange(0, 24)
locations = ["North", "South", "East", "West"]
stations = ["Station_A", "Station_B", "Station_C"]

# Generate synthetic NO  data for each location and station
data = []
for loc in locations:
    for st in stations:
        base = np.random.normal(loc=40, scale=5, size=24)
        if loc == "South":
            base += 5  # slightly higher NO  levels in the South
        data.extend([(h, loc, st, v) for h, v in zip(hours, base)])

# Create DataFrame
df = pd.DataFrame(data, columns=["hour", "location", "station", "NO_2_mean"])

sns.relplot(x="hour", y="NO_2_mean", data=df, kind= "line", color="firebrick")
plt.show()
```

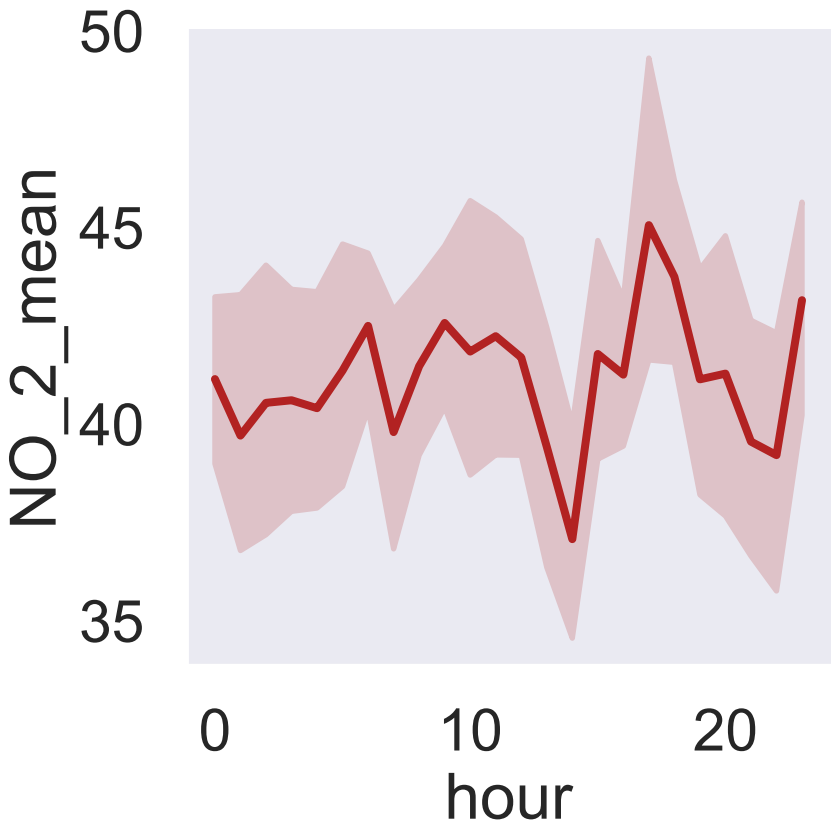


Figure 18: Average Nitrogen Dioxide (NO) Levels by Hour — Line Plot.

Subgroups by location

We can also track subgroups over time with line plots. Here we have the average nitrogen dioxide level for each region (North, South, East, and West) for each hour in the day. Setting the `"style"` and `"hue"` parameters equal to the variable name `"location"` creates different lines for each region that vary in both line style and color. Here, we can see that the South region tends to have slightly higher average nitrogen dioxide levels compared to the other regions.

```
# Import libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Set random seed for reproducibility
```

```
np.random.seed(42)

# Define parameters
hours = np.arange(0, 24)
locations = ["North", "South", "East", "West"]
stations = ["Station_A", "Station_B", "Station_C"]

# Generate synthetic NO  data for each location and station
data = []
for loc in locations:
    for st in stations:
        base = np.random.normal(loc=40, scale=5, size=24)
        if loc == "South":
            base += 5  # slightly higher NO  levels in the South
        data.extend([(h, loc, st, v) for h, v in zip(hours, base)])

# Create DataFrame
df = pd.DataFrame(data, columns=["hour", "location", "station", "NO_2_mean"])

# Create line plot showing subgroup trends over time
sns.relplot(
    x="hour",
    y="NO_2_mean",
    kind="line",
    hue="location",
    style="location",
    markers=True,
    dashes=False,
    data=df
)

# Add title and labels
plt.title("Average Nitrogen Dioxide (NO ) Levels by Hour and Region", fontsize=13)
plt.xlabel("Hour of the Day")
plt.ylabel("NO  Mean (ppb)")

# Show the plot
plt.show()
```

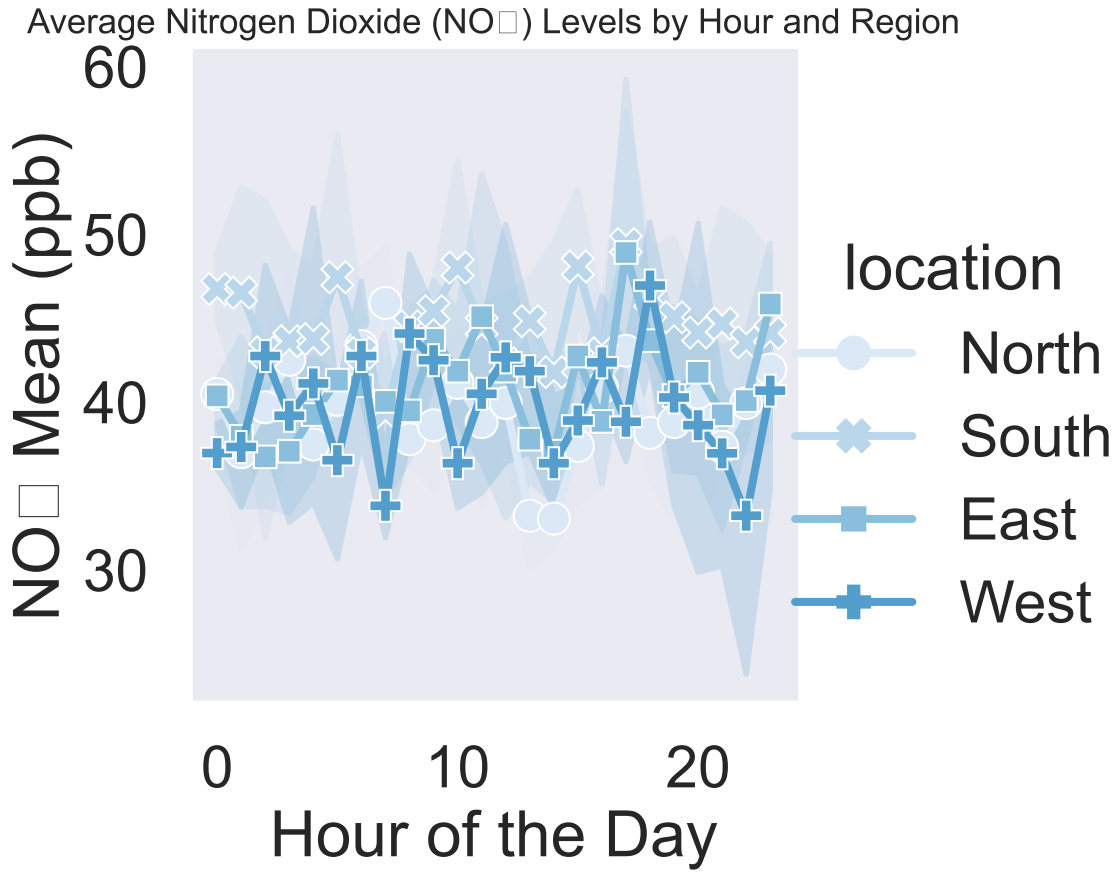


Figure 19: Average Nitrogen Dioxide (NO) Levels by Hour and Region.

Adding markers

Setting the `"markers"` parameter equal to `"True"` will display a marker for each data point. The marker will vary based on the subgroup you've set using the `"style"` parameter.

```
# Import libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Set random seed for reproducibility
np.random.seed(42)
```

```
# Define parameters
hours = np.arange(0, 24)
locations = ["North", "South", "East", "West"]
stations = ["Station_A", "Station_B", "Station_C"]

# Generate synthetic NO  data for each location and station
data = []
for loc in locations:
    for st in stations:
        base = np.random.normal(loc=40, scale=5, size=24)
        if loc == "South":
            base += 5  # slightly higher NO  levels in the South
        data.extend([(h, loc, st, v) for h, v in zip(hours, base)])

# Create DataFrame
df = pd.DataFrame(data, columns=["hour", "location", "station", "NO_2_mean"])

# Create line plot showing subgroup trends over time
sns.relplot(
    x="hour",
    y="NO_2_mean",
    kind="line",
    hue="location",
    style="location",
    markers=True,       # Add markers for each data point
    data=df
)

# Add title and axis labels
plt.title("Average Nitrogen Dioxide (NO ) Levels by Hour and Region", fontsize=13)
plt.xlabel("Hour of the Day")
plt.ylabel("NO  Mean (ppb)")

# Display the plot
plt.show()
```

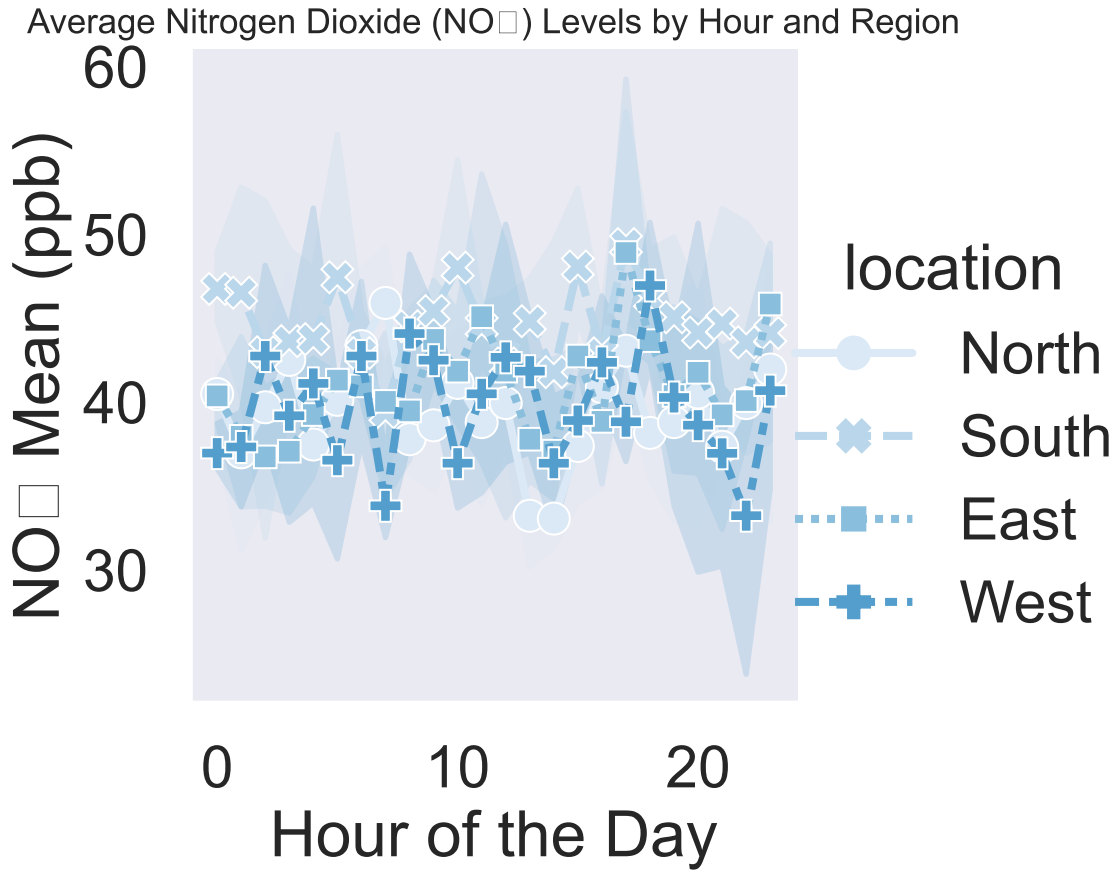


Figure 20: Average Nitrogen Dioxide (NO) Levels by Hour and Region.

Turning off line style

If you don't want the line styles to vary by subgroup, set the `"dashes"` parameter equal to `"False"`.

```
# Import libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Set random seed for reproducibility
np.random.seed(42)
```

```
# Define parameters
hours = np.arange(0, 24)
locations = ["North", "South", "East", "West"]
stations = ["Station_A", "Station_B", "Station_C"]

# Generate synthetic NO  data for each location and station
data = []
for loc in locations:
    for st in stations:
        base = np.random.normal(loc=40, scale=5, size=24)
        if loc == "South":
            base += 5  # slightly higher NO  levels in the South
        data.extend([(h, loc, st, v) for h, v in zip(hours, base)])

# Create DataFrame
df = pd.DataFrame(data, columns=["hour", "location", "station", "NO_2_mean"])

# Create line plot showing subgroup trends over time
sns.relplot(
    x="hour",
    y="NO_2_mean",
    kind="line",
    hue="location",
    style="location",
    markers=True,        # Add markers for each data point
    dashes=False,        # Disable dashed lines for clarity
    data=df
)

# Add title and axis labels
plt.title("Average Nitrogen Dioxide (NO ) Levels by Hour and Region", fontsize=13)
plt.xlabel("Hour of the Day")
plt.ylabel("NO  Mean (ppb)")

# Display the plot
plt.show()
```

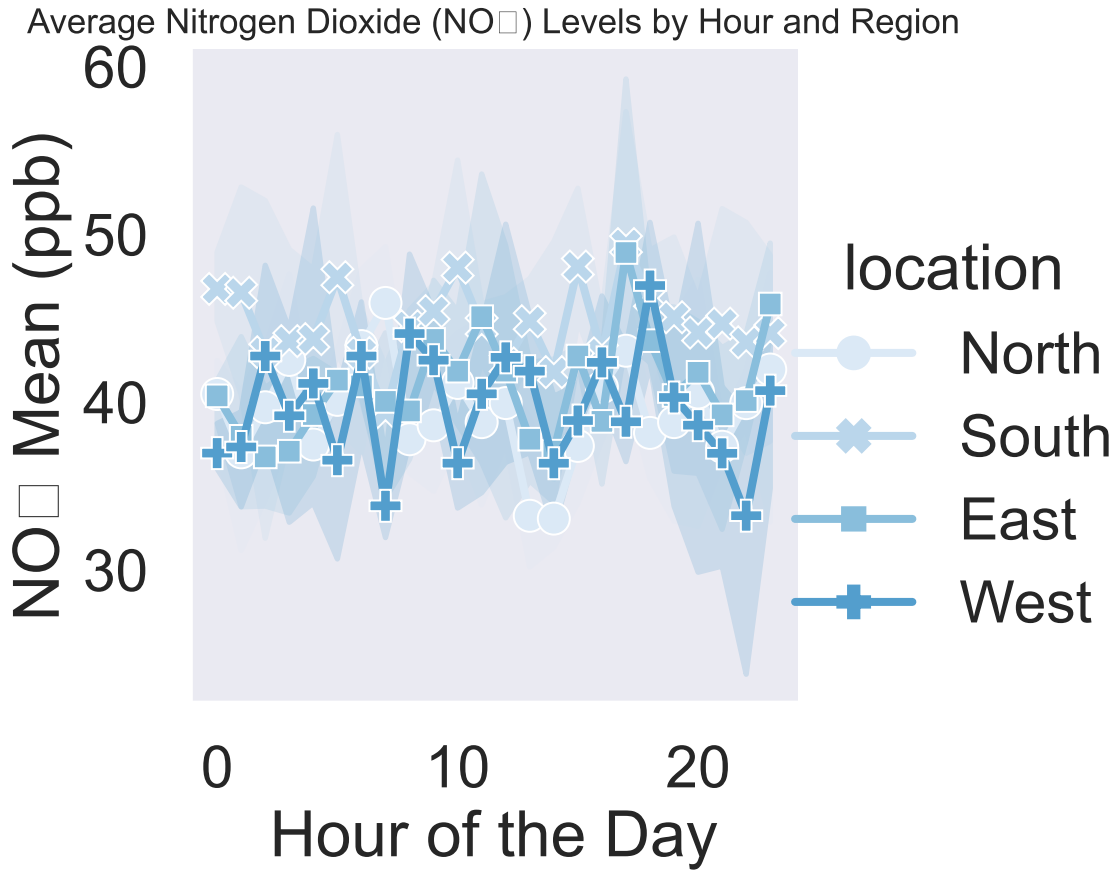


Figure 21: Average Nitrogen Dioxide (NO) Levels by Hour and Region.

Multiple observations per x-value

Line plots can also be used when you have more than one observation per x-value. This dataset has a row for each station that is taking a measurement every hour. This is the scatter plot, displaying one point per observation. This is the line plot. If a line plot is given multiple observations per x-value, it will aggregate them into a single summary measure. By default, it will display the mean. Notice that Seaborn will automatically calculate a confidence interval for the mean, displayed by the shaded region. Assuming the air collection stations were randomly placed throughout the city, this dataset is a random sample of the nitrogen dioxide levels across the whole city. This confidence interval tells us that based on our sample, we can be 95% confident that the average nitrogen dioxide level for the whole city is within this range. Confidence intervals indicate the uncertainty we have about what the true mean is for the whole city. To learn more about confidence intervals, you can check out DataCamp's statistics courses.

Replacing confidence interval with standard deviation

Instead of visualizing a confidence interval, we may want to see how varied the measurements of nitrogen dioxide are across the different collection stations at a given point in time. To visualize this, set the `"ci"` parameter equal to the string `"sd"` to make the shaded area represent the standard deviation, which shows the spread of the distribution of observations at each x value.

```
# Import libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Set random seed for reproducibility
np.random.seed(42)

# Define parameters
hours = np.arange(0, 24)
locations = ["North", "South", "East", "West"]
stations = ["Station_A", "Station_B", "Station_C"]

# Generate synthetic NO  data for each location and station
data = []
for loc in locations:
    for st in stations:
        base = np.random.normal(loc=40, scale=5, size=24)
        if loc == "South":
            base += 5  # slightly higher NO  levels in the South
        data.extend([(h, loc, st, v) for h, v in zip(hours, base)])

# Create DataFrame
df = pd.DataFrame(data, columns=["hour", "location", "station", "NO_2_mean"])

# Create line plot showing subgroup trends over time
sns.relplot(
    x="hour",
    y="NO_2_mean",
    kind="line",
    hue="location",
    style="location",
    markers=True,       # Add markers for each data point
    dashes=False,       # Disable dashed lines for clarity
    data=df,
```

```
    ci = "sd"
)

# Add title and axis labels
plt.title("Average Nitrogen Dioxide (NO ) Levels by Hour and Region", fontsize=13)
plt.xlabel("Hour of the Day")
plt.ylabel("NO  Mean (ppb)")

# Display the plot
plt.show()
```

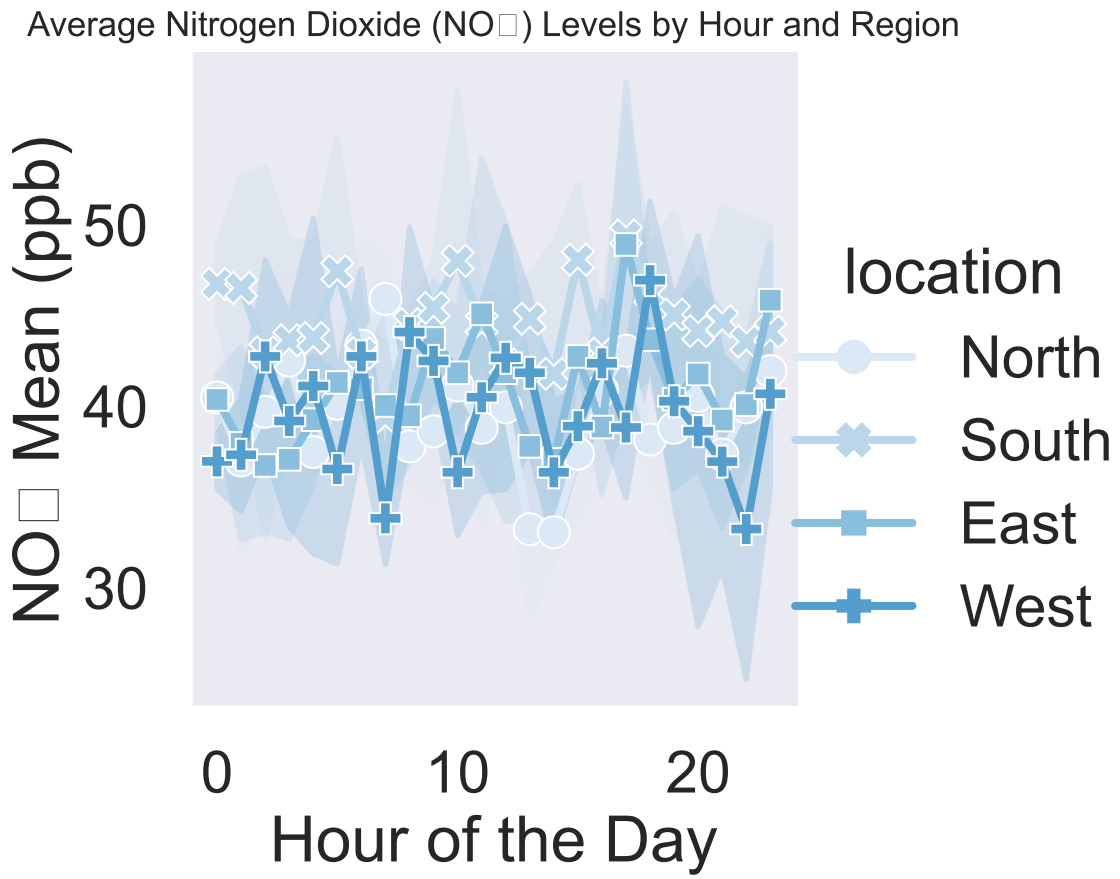


Figure 22: Average Nitrogen Dioxide (NO) Levels by Hour and Region.

Turning off confidence interval

We can also turn off the confidence interval by setting the "ci" parameter equal to `"None"` for old version of seaborn, but `"errorbar"` parameter to `None`, with newer versions of seaborn.

```python
# Import libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Set random seed for reproducibility
np.random.seed(42)

# Define parameters
hours = np.arange(0, 24)
locations = ["North", "South", "East", "West"]
stations = ["Station_A", "Station_B", "Station_C"]

# Generate synthetic NO  data for each location and station
data = []
for loc in locations:
    for st in stations:
        base = np.random.normal(loc=40, scale=5, size=24)
        if loc == "South":
            base += 5  # slightly higher NO  levels in the South
        data.extend([(h, loc, st, v) for h, v in zip(hours, base)])

# Create DataFrame
df = pd.DataFrame(data, columns=["hour", "location", "station", "NO_2_mean"])

# Create line plot showing subgroup trends over time
sns.relplot(
    x="hour",
    y="NO_2_mean",
    kind="line",
    hue="location",
    style="location",
    markers=True,       # Add markers for each data point
    dashes=False,       # Disable dashed lines for clarity
    data=df,
    errorbar=None
)

# Add title and axis labels
plt.title("Average Nitrogen Dioxide (NO ) Levels by Hour and Region", fontsize=13)
plt.xlabel("Hour of the Day")
plt.ylabel("NO  Mean (ppb)")

# Display the plot
```

```
plt.show()
```

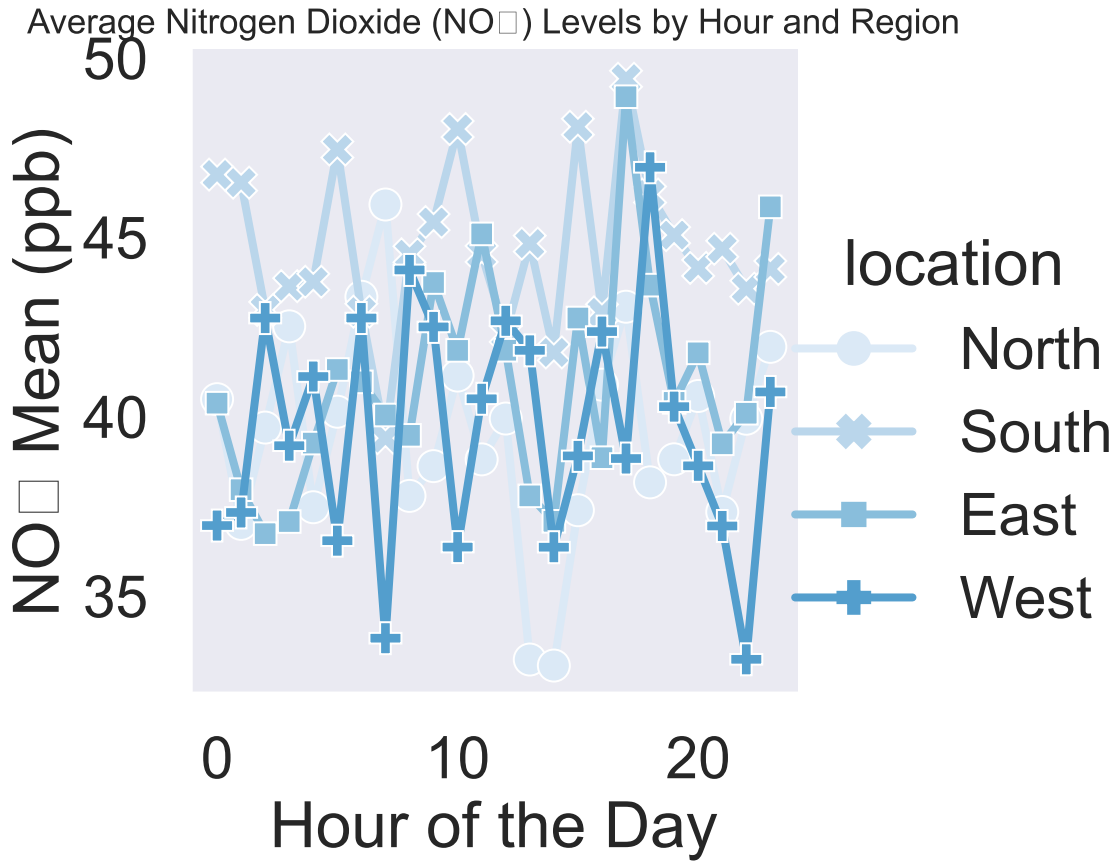


Figure 23: Average Nitrogen Dioxide (NO) Levels by Hour and Region.

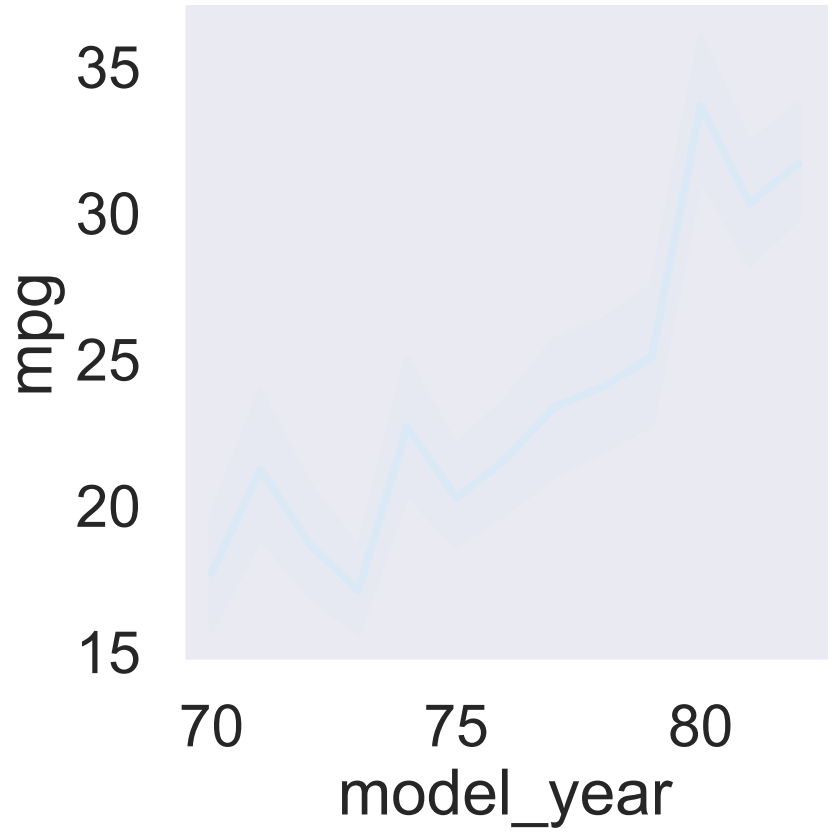
Exercise 2.3

---

1. Use `relplot()` and the mpg DataFrame to create a line plot with `"model_year"` on the x-axis and `"mpg"` on the y-axis.

```
# Import Matplotlib and Seaborn
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Import dataset
mpg = pd.read_csv("datasets/mpg.csv")
```

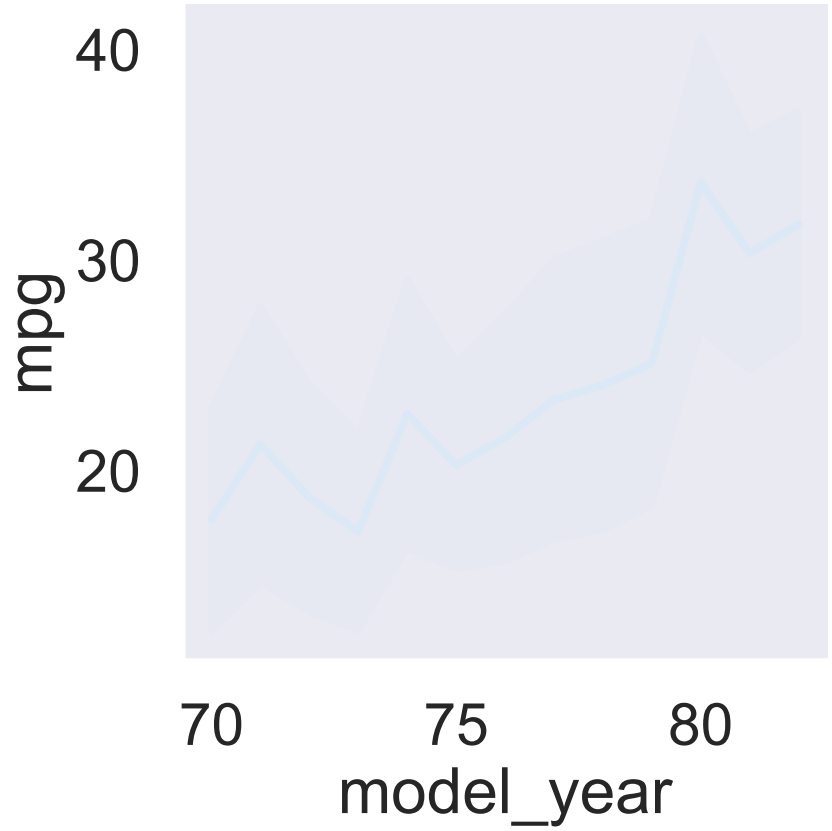
```
# Create line plot
sns.relplot(x="model_year", y="mpg", data=mpg, kind="line")

# Show plot
plt.show()
```



2. Change the plot so the shaded area shows the standard deviation instead of the confidence interval for the mean.

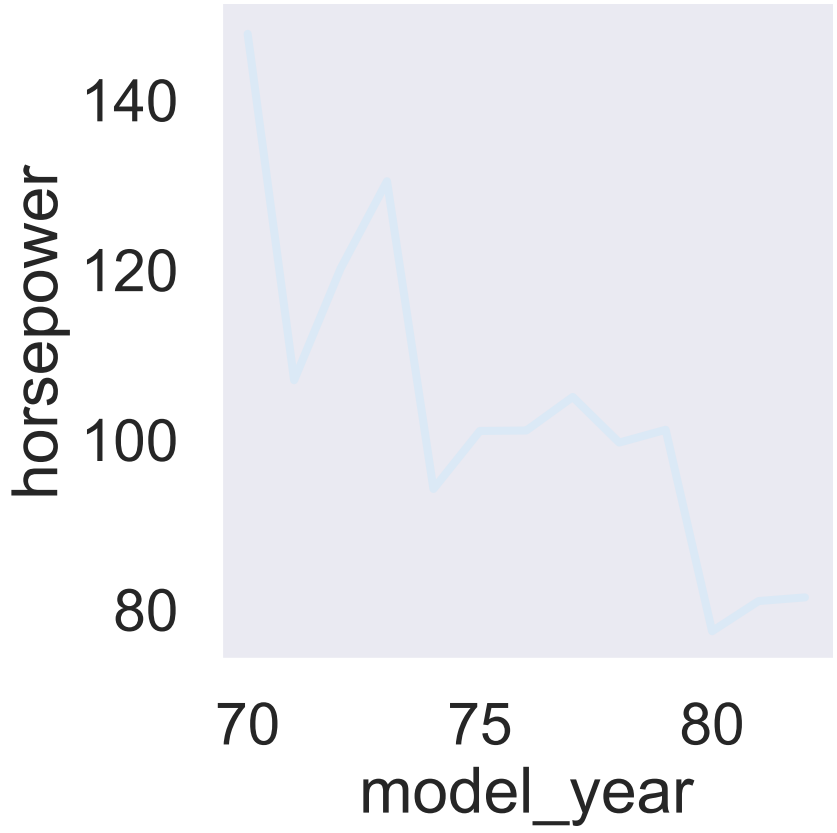
```
# Import Matplotlib and Seaborn
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Import dataset
mpg = pd.read_csv("datasets/mpg.csv")

sns.relplot(x="model_year", y="mpg",
            data=mpg, kind="line", ci="sd")
```

```
# Show plot
plt.show()
```



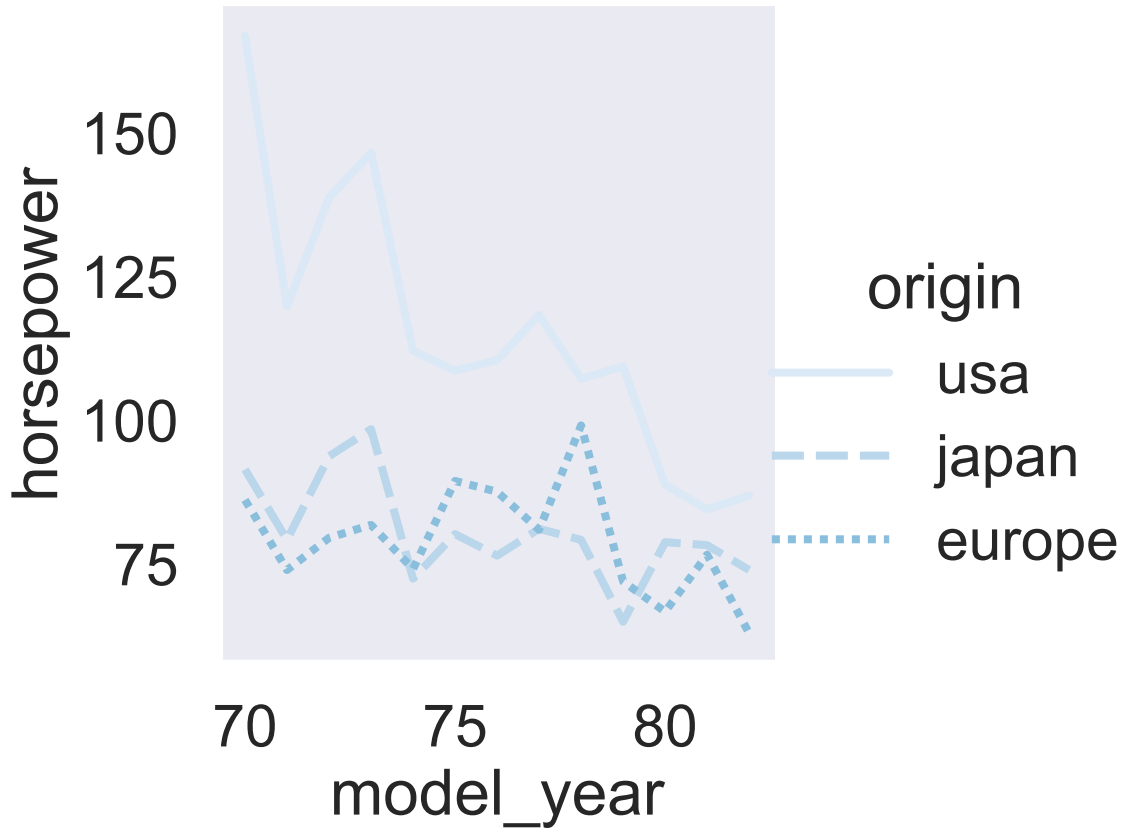
3. Use `relplot()` and the mpg DataFrame to create a line plot with `"model_year"` on the x-axis and `"horsepower"` on the y-axis. Turn off the confidence intervals on the plot.

```
# Import Matplotlib and Seaborn
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Import dataset
mpg = pd.read_csv("datasets/mpg.csv")

sns.relplot(x="model_year", y="horsepower",
            data=mpg, kind="line", errorbar=None)

# Show plot
plt.show()
```



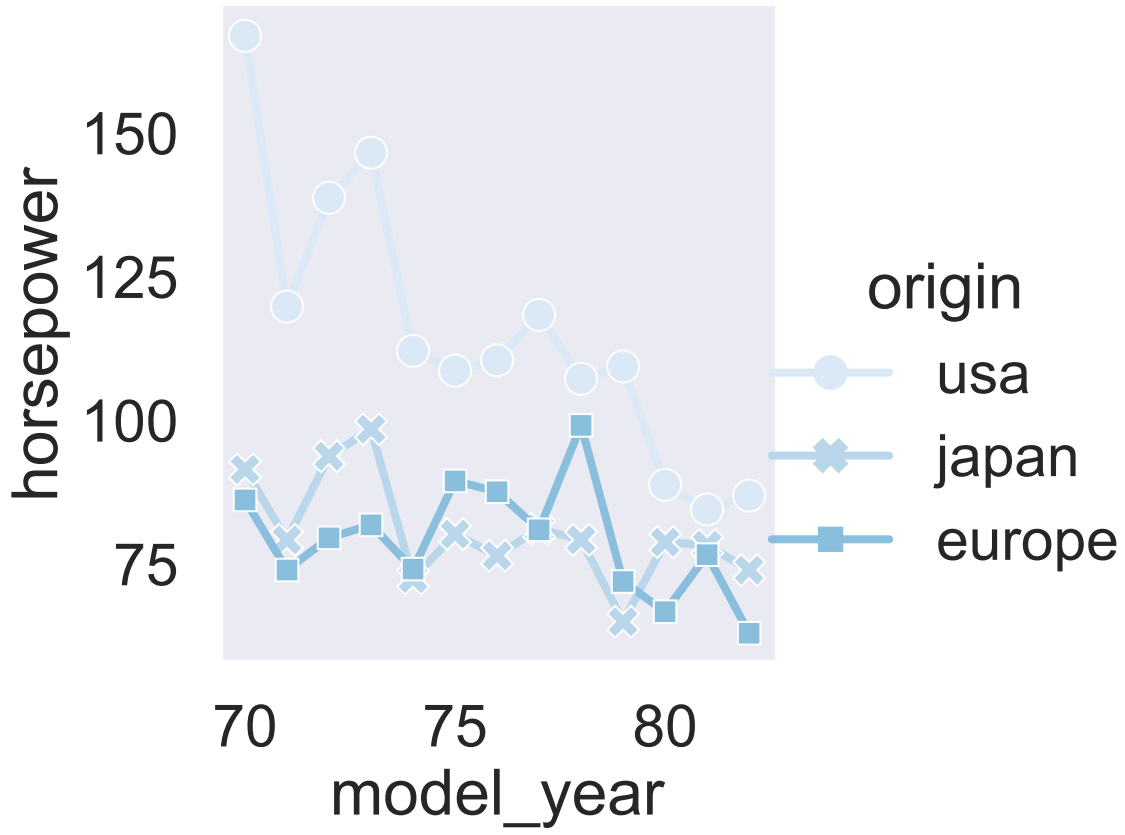
4. Create different lines for each country of origin (`"origin"`) that vary in both line style and color.

```
# Import Matplotlib and Seaborn
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Import dataset
mpg = pd.read_csv("datasets/mpg.csv")

sns.relplot(x="model_year", y="horsepower",
            data=mpg, kind="line",
            errorbar=None, style= "origin", hue="origin")

# Show plot
plt.show()
```



5. Add markers for each data point to the lines.
6. Use the dashes parameter to use solid lines for all countries, while still allowing for different marker styles for each line.

```
# Import Matplotlib and Seaborn
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Import dataset
mpg = pd.read_csv("datasets/mpg.csv")

sns.relplot(x="model_year", y="horsepower",
            data=mpg, kind="line",
            errorbar=None, style="origin",
            hue="origin", markers=True, dashes=False)

# Show plot
plt.show()
```



Chapter 3: Count plots and bar plots

Welcome to Chapter 3! In this chapter, we'll focus on visualizations that involve categorical variables. The first two plots we'll look at are count plots and bar plots.

Categorical plots

Count plots and bar plots are two types of visualizations that Seaborn calls "categorical plots". Categorical plots involve a categorical variable, which is a variable that consists of a fixed, typically small number of possible values, or categories. These types of plots are commonly used when we want to make comparisons between different groups. We began to explore categorical plots in Chapter 1 with count plots. As a reminder, a count plot displays the number of observations in each category. We saw several examples of count plots in earlier chapters, Section , Section , like the number of men reporting that they feel masculine. Most men surveyed here feel`"somewhat"` or `"very"` masculine.

catplot()

Just like we used "relplot()" to create different types of relational plots, in this chapter we'll be using "catplot()" to create different types of categorical plots. "catplot()" offers the same flexibility that "relplot()" does, which means it will be easy to create subplots if we need to using the same "col" and "row" parameters.

countplot() vs. catplot()

To see how "catplot()" works, let's return to the masculinity count plot. On the left, we see how we originally created a count plot with the "countplot()" function.

countplot() vs. catplot()

To make this plot with "catplot()" instead, we change the function name to "catplot()" and use the "kind" parameter to specify what kind of categorical plot to use. In this case, we'll set kind equal to the word "count".

```
# Import libraries
import seaborn as sns
import matplotlib.pyplot as plt


masculinity = {
    'participant_id': list(range(1, 21)),
    'age': [35, 42, 29, 51, 37, 30, 45, 55, 25, 33, 40, 48, 28, 39, 47, 53, 26, 34, 43, 50],
    'how_masculine': ['very', 'somewhat', 'very', 'not at all', 'somewhat', 'very', 'somewhat', '
                      'not at all', 'somewhat', 'very', 'somewhat', 'very', 'not at all', 'somewh
                      'very', 'somewhat', 'very', 'not at all', 'somewhat'],
    'how_important': ['very', 'somewhat', 'not very', 'somewhat', 'very', 'very', 'somewhat', 'no
                      'somewhat', 'very', 'very', 'somewhat', 'not very', 'somewhat', 'very', 've
                      'somewhat', 'not very', 'somewhat', 'very']
}

# Create the DataFrame
df = pd.DataFrame(masculinity)
```

```
# Create count plot using catplot
sns.catplot(
    x="how_masculine",   # categorical variable
    data=df,
    kind="count"         # specify type of plot
)

plt.show()
```

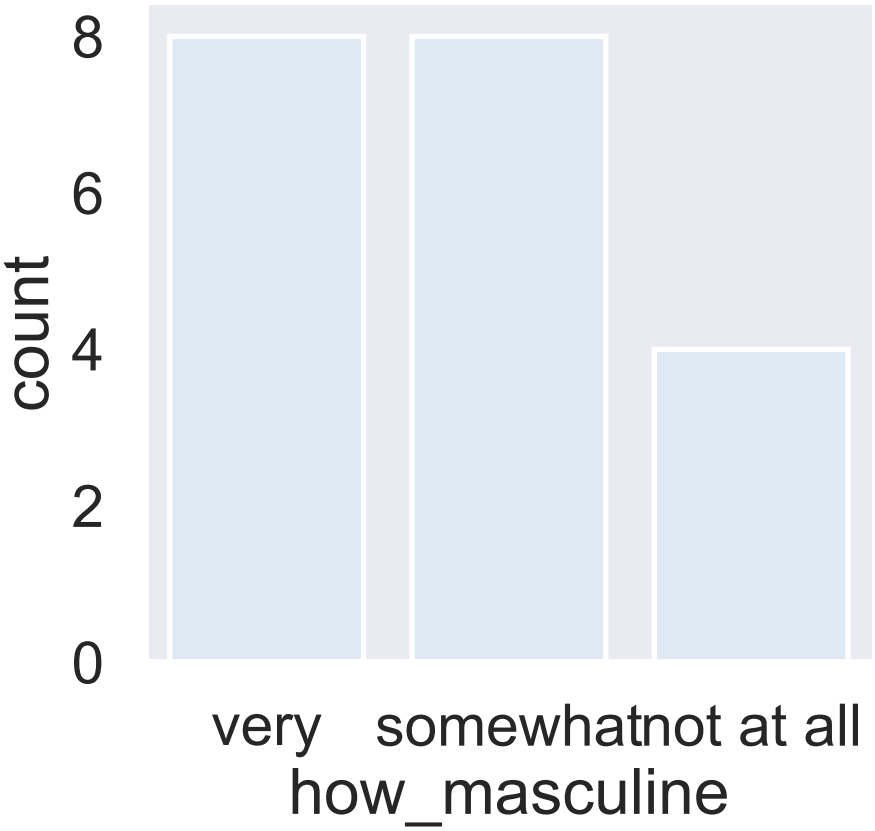


Figure 24: Count of Masculinity Responses Using catplot().

Changing the order

Sometimes there is a specific ordering of categories that makes sense for these plots. In this case, it makes more sense for the categories to be in order from not masculine to very masculine. To change the order of the categories, create a list of category values in the order that you want them

to appear, and then use the "order" parameter. This works for all types of categorical plots, not just count plots.

```
# Import libraries
import seaborn as sns
import matplotlib.pyplot as plt

masculinity = {
    'participant_id': list(range(1, 21)),
    'age': [35, 42, 29, 51, 37, 30, 45, 55, 25, 33, 40, 48, 28, 39, 47, 53, 26, 34, 43, 50],
    'how_masculine': ['very', 'somewhat', 'very', 'not at all', 'somewhat', 'very', 'somewhat', '
                      'not at all', 'somewhat', 'very', 'somewhat', 'very', 'not at all', 'somewh
                      'very', 'somewhat', 'very', 'not at all', 'somewhat'],
    'how_important': ['very', 'somewhat', 'not very', 'somewhat', 'very', 'very', 'somewhat', 'no
                      'somewhat', 'very', 'very', 'somewhat', 'not very', 'somewhat', 'very', 've
                      'somewhat', 'not very', 'somewhat', 'very']
}

# Create the DataFrame
df = pd.DataFrame(masculinity)

# Define custom order for the categories
masculine_order = ["not at all", "somewhat", "very"]

# Create ordered count plot

sns.catplot(
x="how_masculine",
data=df,
kind="count",
order=masculine_order   # specify the category order
)

plt.show()
```

Figure 25: Count of Masculinity Responses Ordered from 'Not at All' to 'Very'.

Bar plots

Bar plots look similar to count plots, but instead of the count of observations in each category, they show the mean of a quantitative variable among observations in each category. This bar plot uses the tips dataset and shows the average bill paid among people who visited the restaurant on each day of the week. From this, we can see that the average bill is slightly higher on the weekends. To create this bar plot, we use `"catplot"`. Specify the categorical variable `"day"` on the x-axis, the quantitative variable `"total bill"` on the y-axis, and set the `"kind"` parameter equal to `"bar"`.

```
# Import libraries
import seaborn as sns
import matplotlib.pyplot as plt

# Load Seaborn's 'tips' dataset
```

```
tips = sns.load_dataset("tips")

# Create bar plot using catplot

sns.catplot(
x="day",
y="total_bill",
data=tips,
kind="bar",          # specify bar plot
palette="pastel"
)

plt.show()
```

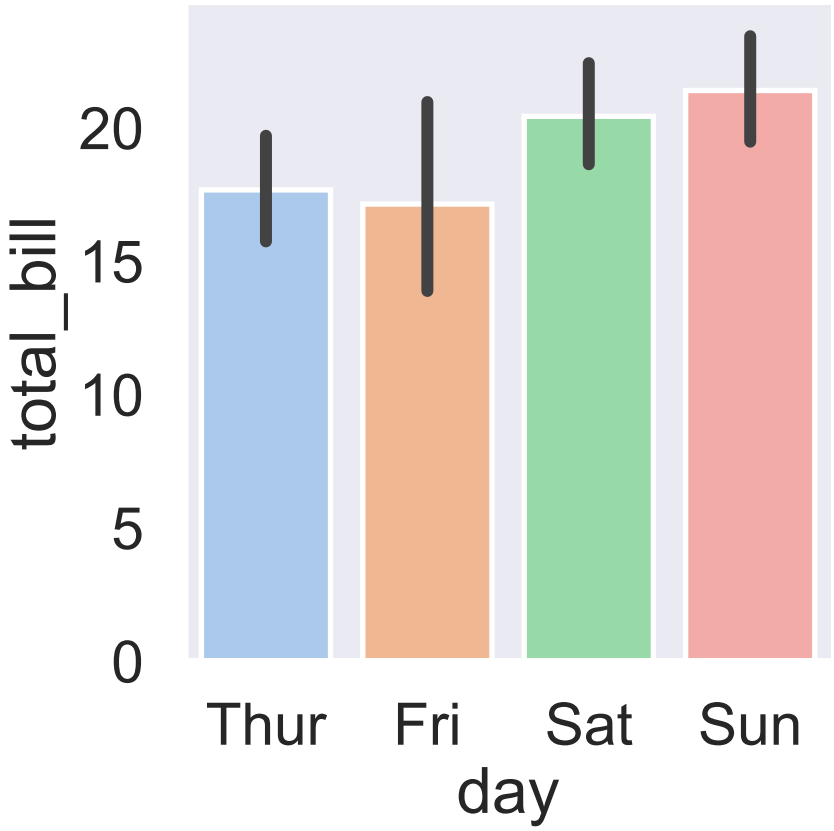


Figure 26: Average Total Bill by Day of Week (Tips Dataset).

Confidence intervals

Notice also that Seaborn automatically shows 95% confidence intervals for these means. Just like with line plots, these confidence intervals show us the level of uncertainty we have about these estimates. Assuming our data is a random sample of some population, we can be 95% sure that the true population mean in each group lies within the confidence interval shown.

Turning off confidence intervals

If we want to turn off these confidence intervals, we can do this by setting the `"errorbar"` parameter equal to `"None"` - just like we did with line plots.

```
# Import libraries
import seaborn as sns
import matplotlib.pyplot as plt

# Load Seaborn's 'tips' dataset

tips = sns.load_dataset("tips")

# Create bar plot using catplot

sns.catplot(
x="day",
y="total_bill",
data=tips,
kind="bar",           # specify bar plot
errorbar = None)

plt.show()
```

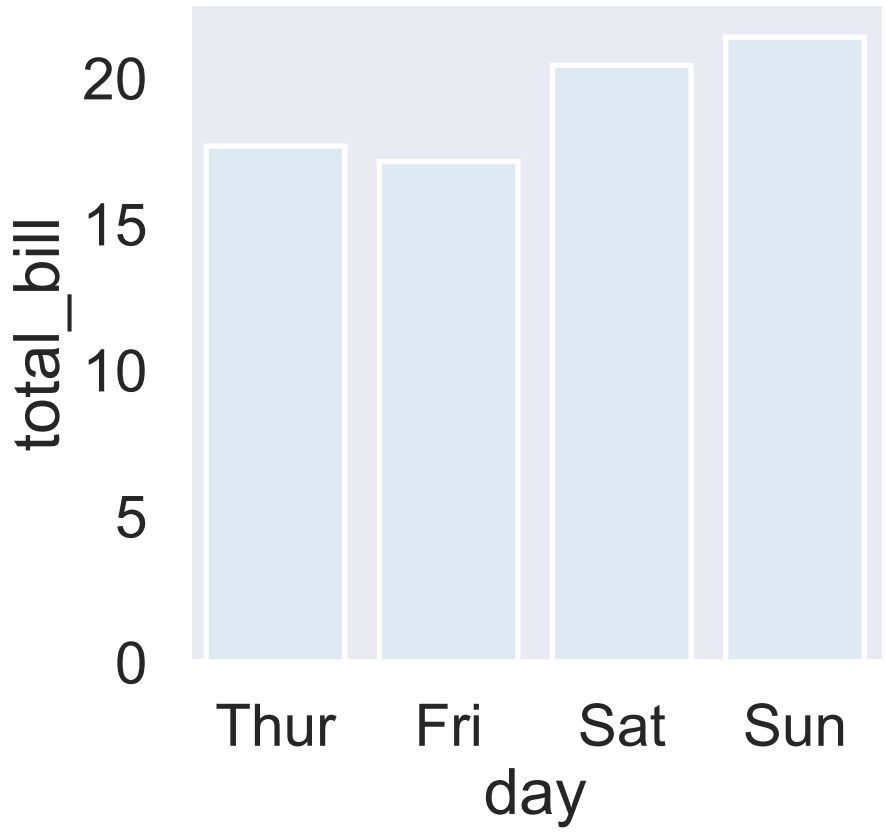


Figure 27: Average Total Bill by Day of Week (Tips Dataset).

Changing the orientation

Finally, you can also change the orientation of the bars in bar plots and count plots by switching the x and y parameters. However, it is fairly common practice to put the categorical variable on the x-axis.

```
# Import libraries
import seaborn as sns
import matplotlib.pyplot as plt

# Load Seaborn's 'tips' dataset

tips = sns.load_dataset("tips")

# Create bar plot using catplot
```

```
sns.catplot(
x="total_bill",
y="day",
data=tips,
kind="bar",          # specify bar plot
errorbar = None)

plt.show()
```

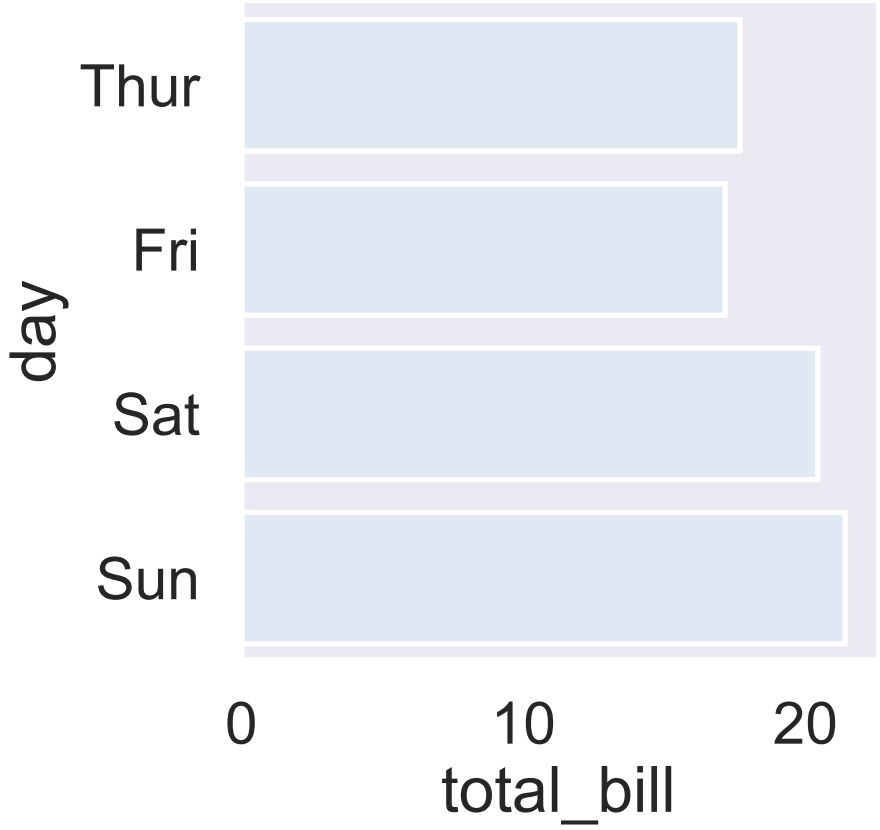


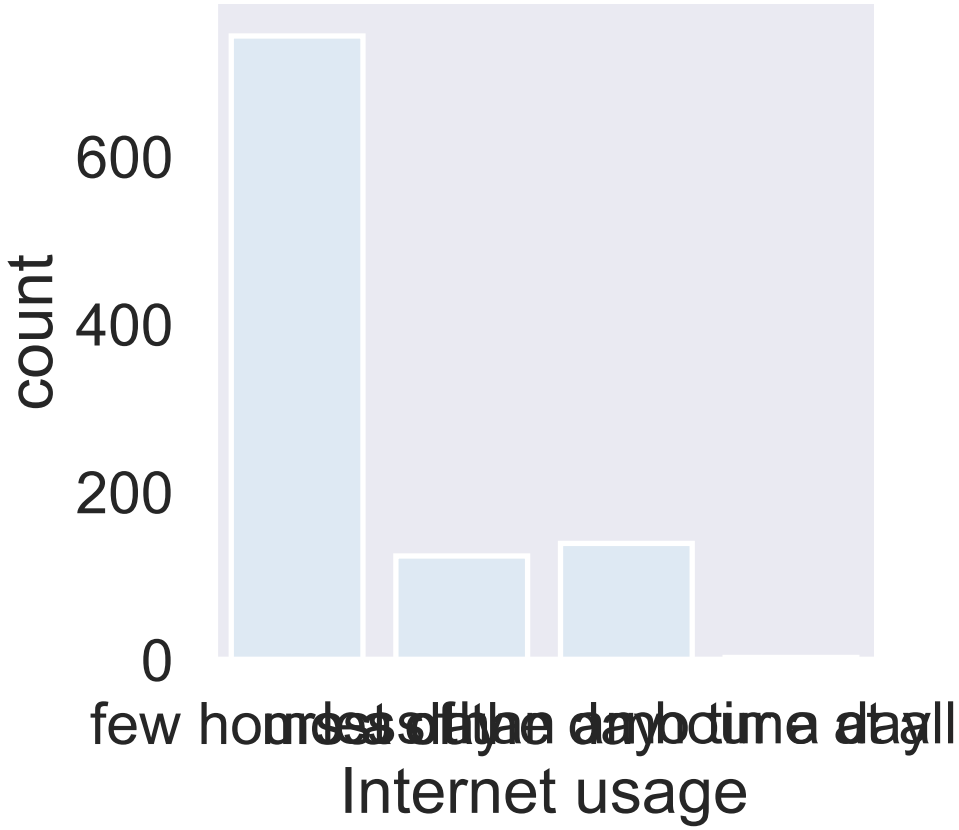
Figure 28: Average Total Bill by Day of Week (Tips Dataset).

Exercise 3.1

---

1. Use `sns.catplot()` to create a count plot using the `survey_data` DataFrame with `"Internet usage"` on the x-axis.

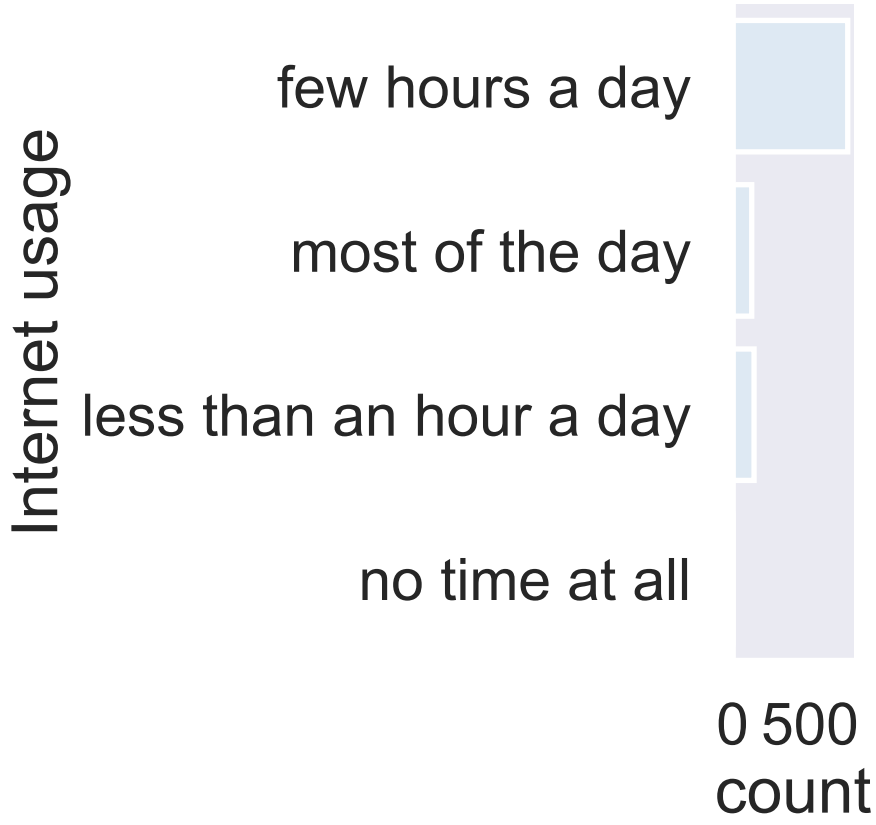
```
# Import packages
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Import dataset
survey_data = pd.read_csv('datasets/young-people-survey-responses.csv')

sns.catplot(x= "Internet usage", data=survey_data, kind="count")

# Show plot
plt.show()
```



2. Make the bars horizontal instead of vertical.

```
# Import packages
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Import dataset
```

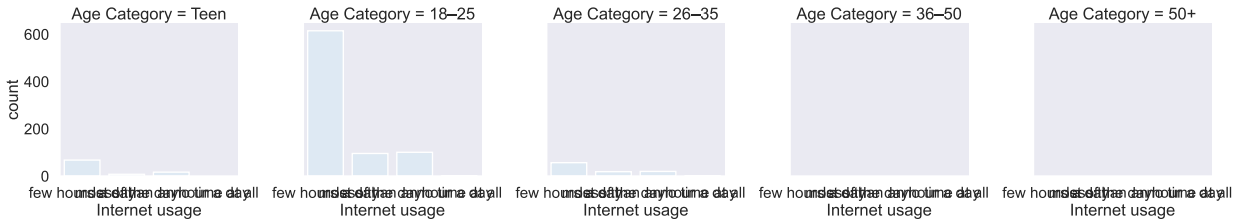
```
survey_data = pd.read_csv('datasets/young-people-survey-responses.csv')

sns.catplot(y= "Internet usage", data=survey_data, kind="count")

# Show plot
plt.show()
```



3. Separate this plot into two side-by-side column subplots based on `"Age Category"`, which separates respondents into those that are younger than 21 vs. 21 and older.

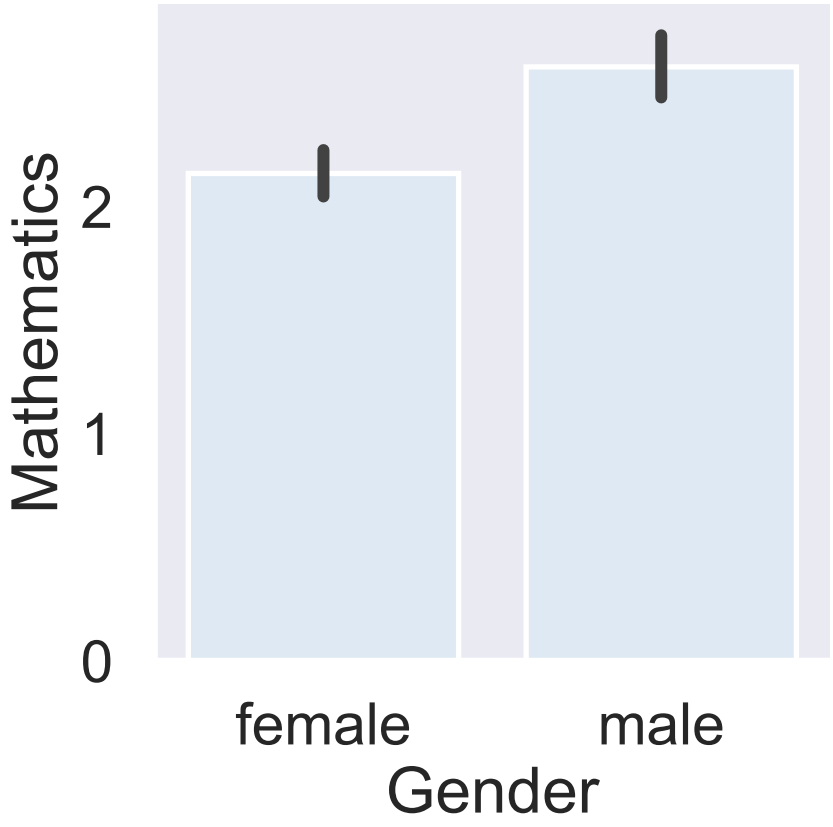
```
# Import packages
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Import dataset
survey_data = pd.read_csv('datasets/young-people-survey-responses.csv')

# Create Age Category groups
bins = [10, 18, 25, 35, 50, 70]
labels = ["Teen", "18-25", "26-35", "36-50", "50+"]
```

```
survey_data["Age Category"] = pd.cut(survey_data["Age"], bins=bins, labels=labels, right=False)


sns.catplot(
    x="Internet usage",
    data=survey_data,
    kind="count",
    col="Age Category")

# Show plot
plt.show()
```



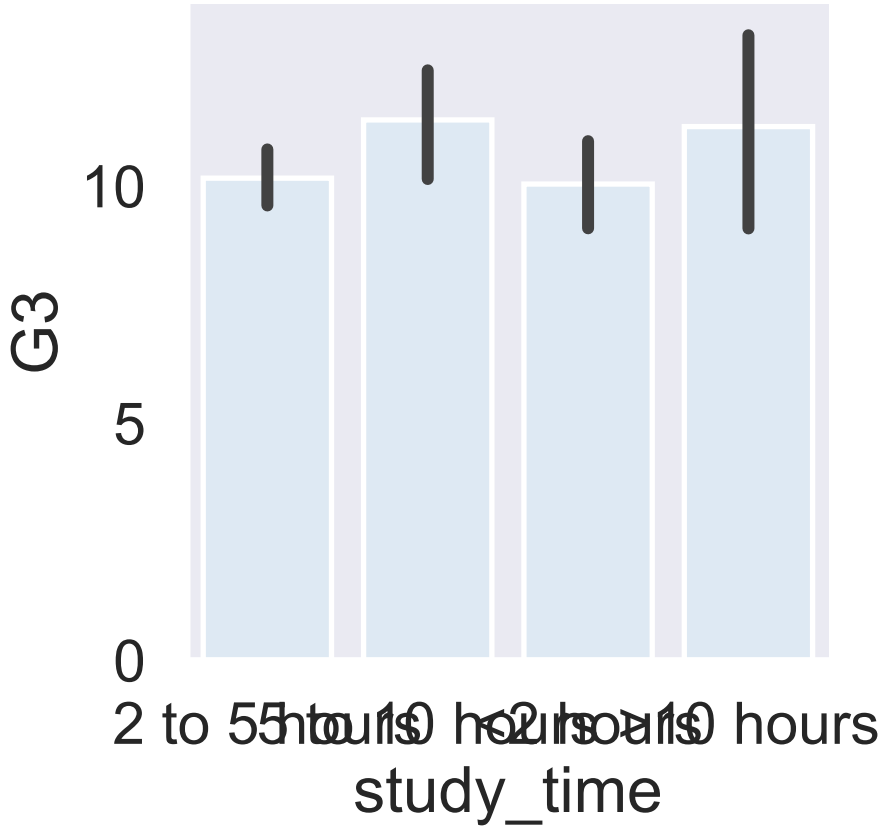
4. Bar plots with percentages. Use the survey_data DataFrame and `sns.catplot()` to create a bar plot with `"Gender"` on the x-axis and `"Mathematics"` on the y-axis, which signifies, students with interest in mathematics.

```
# Import packages
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Import dataset
survey_data = pd.read_csv('datasets/young-people-survey-responses.csv')

sns.catplot(x="Gender", y="Mathematics", data=survey_data, kind="bar")

# Show plot
plt.show()
```



5. Use `sns.catplot()` to create a bar plot with `"study_time"` on the x-axis and final grade (`"G3"`) on the y-axis, using the `student_data` DataFrame.

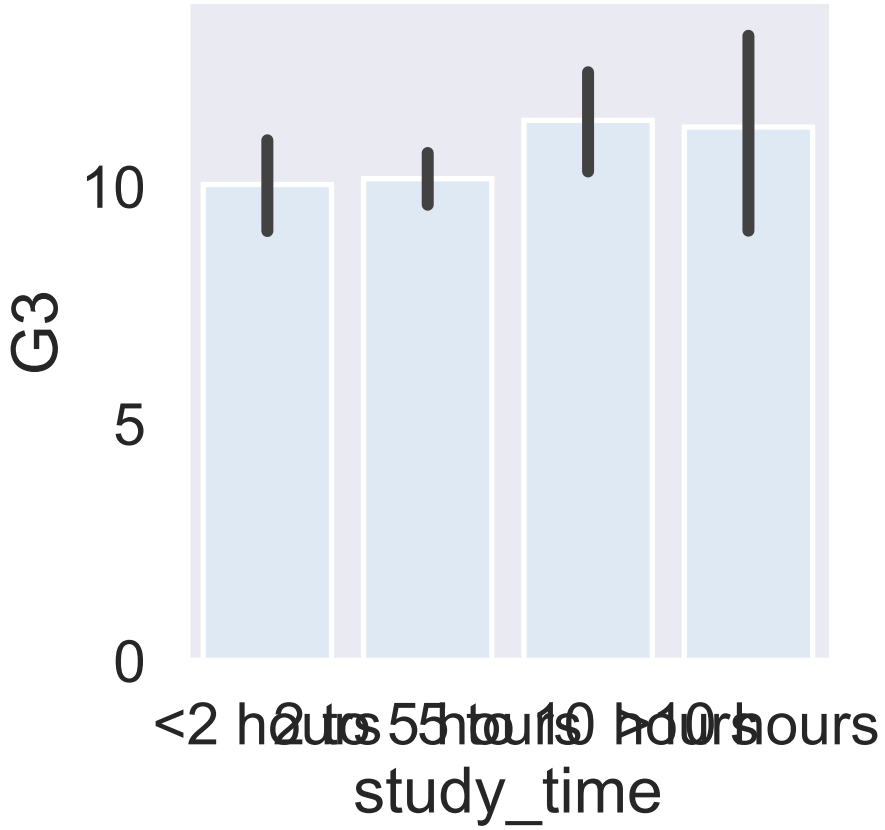
```
# Import packages
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Import dataset
student_data = pd.read_csv('datasets/student-alcohol-consumption.csv')

sns.catplot(x="study_time", y="G3", data=student_data, kind="bar")

# Show plot
plt.show()
```



6. Using the order parameter and the `category_order` list that is provided, rearrange the bars so that they are in order from lowest study time to highest.

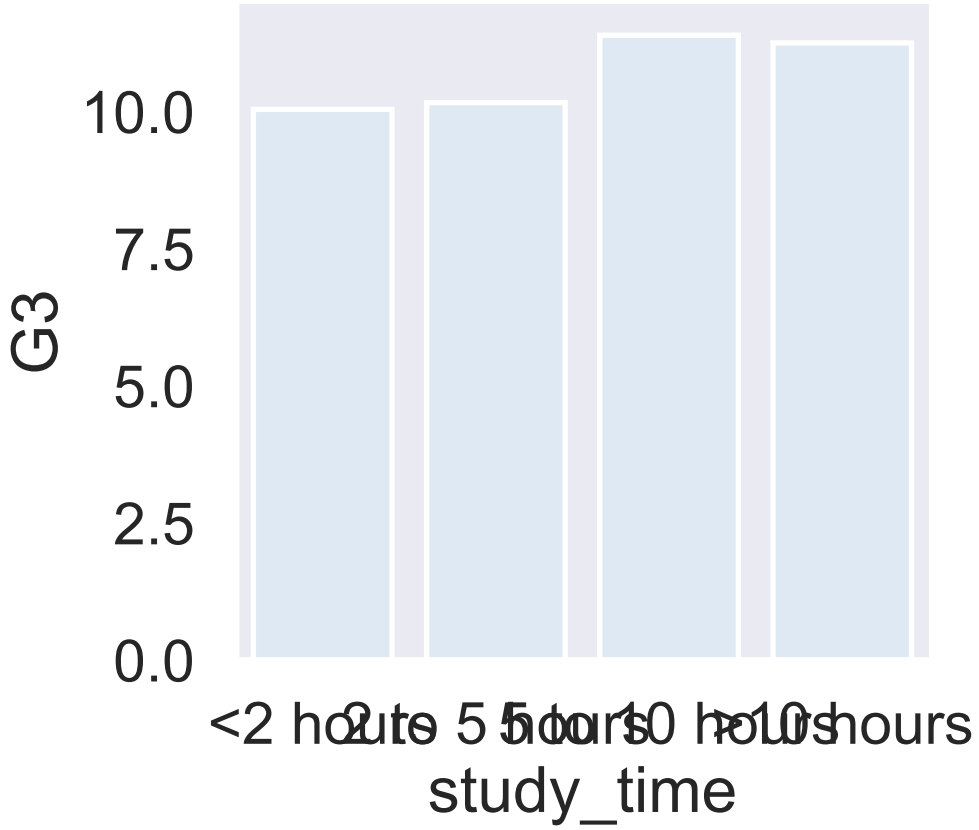
```
# Import packages
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Import dataset
student_data = pd.read_csv('datasets/student-alcohol-consumption.csv')

category_order = ["<2 hours",
                  "2 to 5 hours",
                  "5 to 10 hours",
                  ">10 hours"]

# Rearrange the categories
sns.catplot(x="study_time", y="G3",
            data=student_data,
            kind="bar", order=category_order)
```

```
# Show plot
plt.show()
```



7. Update the plot so that it no longer displays confidence intervals.

```
# Import packages
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Import dataset
student_data = pd.read_csv('datasets/student-alcohol-consumption.csv')

sns.catplot(x="study_time", y="G3",
            data=student_data,
            kind="bar",
            order=category_order, errorbar=None)

# Show plot
plt.show()
```



Chapter 4: Creating a box plot

Hello! In this chapter we'll learn how to create a new type of categorical plot: the box plot.

What is a box plot?

A box plot shows the **distribution of quantitative data**. The colored box represents the **25th to 75th percentile (interquartile range)**, and the line inside the box shows the **median**. The **whiskers** extend to give a sense of the overall spread, while **dots** beyond the whiskers represent **outliers**. Box plots are powerful for comparing distributions of a quantitative variable across different categories. In this example, the box plot uses the **tips dataset** to compare the distribution of **total bill amounts** across the **days of the week**. From this visualization, we can quickly see that the **median total bill is higher on Saturday and Sunday**, but the **spread of the distribution** is also wider — something that's harder to see with simpler plot types.

```
# Load Seaborn's 'tips' dataset
tips = sns.load_dataset("tips")

# Create a box plot using catplot
sns.catplot(
    x="day",
    y="total_bill",
    data=tips,
    kind="box",
    palette="pastel"
)

plt.show()
```

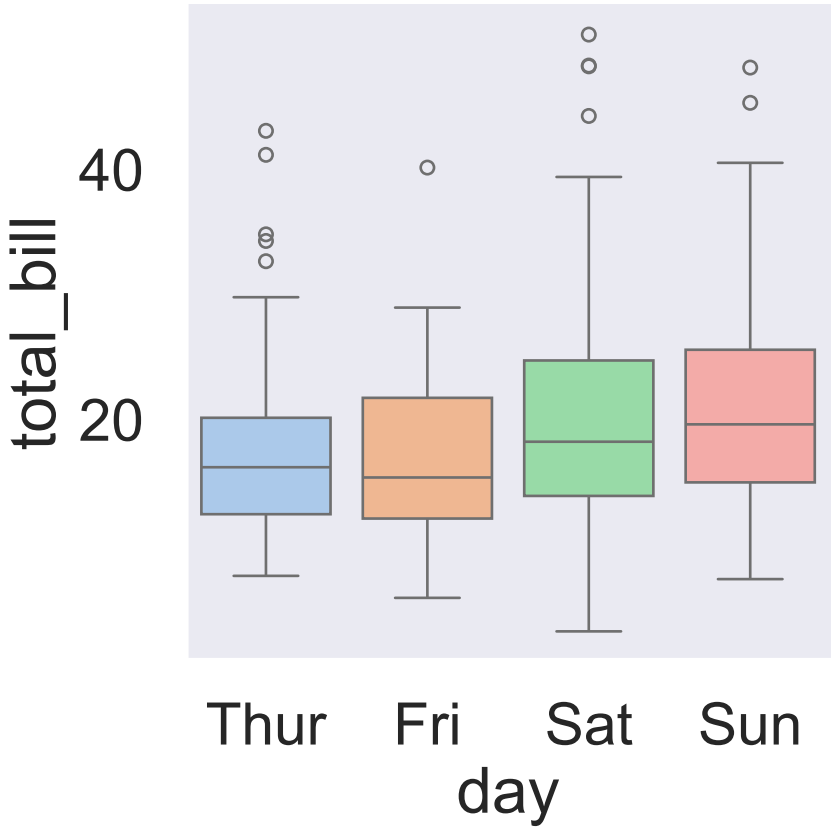


Figure 29: Distribution of Total Bill Amounts by Day (Tips Dataset).

How to create a box plot

Now let's look at how to create a box plot in Seaborn. While Seaborn does have a `"boxplot()"` function, we'll be using the `"catplot()"` function that we introduced in an earlier lesson because it makes it easy to create subplots using the `"col"` and `"row"` parameters. We'll put the categorical variable `"time"` on the x-axis and the quantitative variable `"total bill"` on the y-axis. Here, we want box plots, so we'll specify `kind="box"`. That's it! We have a nice looking box plot. Next, we'll look at different ways to customize this plot.

```
# Load Seaborn's 'tips' dataset
tips = sns.load_dataset("tips")

# Create a box plot using catplot
sns.catplot(
    x="time",
    y="total_bill",
    data=tips,
    kind="box"
)

plt.show()
```

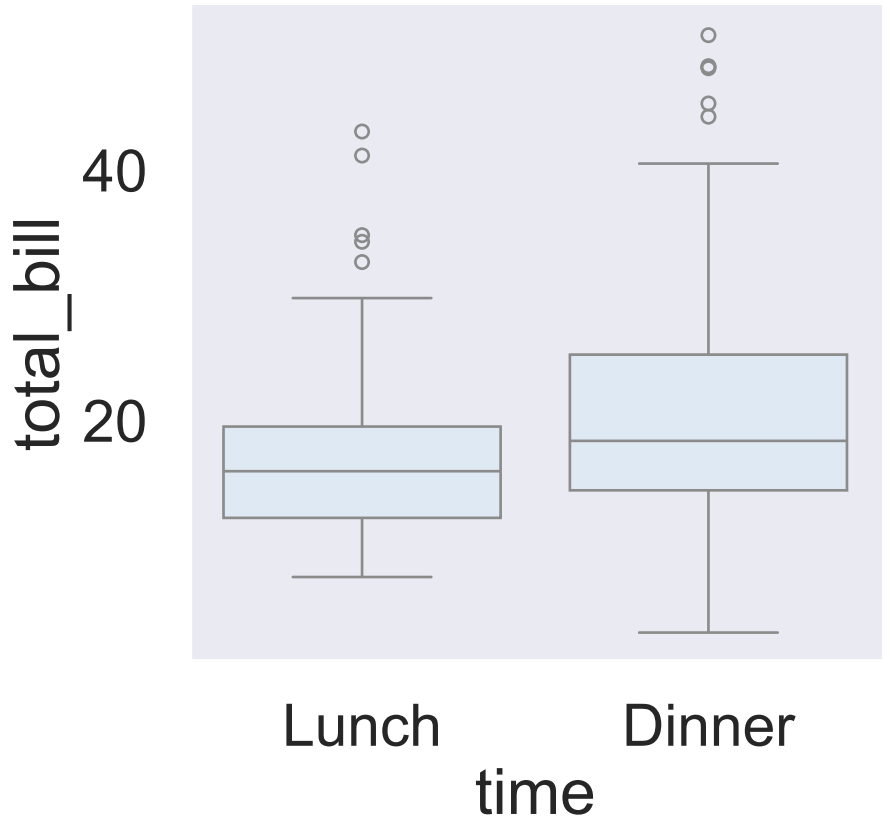


Figure 30: Distribution of Total Bill Amounts by Time (Tips Dataset).

Change the order of categories

As a reminder, `"catplot"` allows you to change the order of the categories using the `"order"` parameter. Here, we specified that `"dinner"` should be shown before "lunch".

```
# Load Seaborn's 'tips' dataset
tips = sns.load_dataset("tips")

# Create a box plot using catplot
sns.catplot(
    x="time",
    y="total_bill",
    data=tips,
    kind="box",
    order = ["Dinner", "Lunch"]
)

plt.show()
```

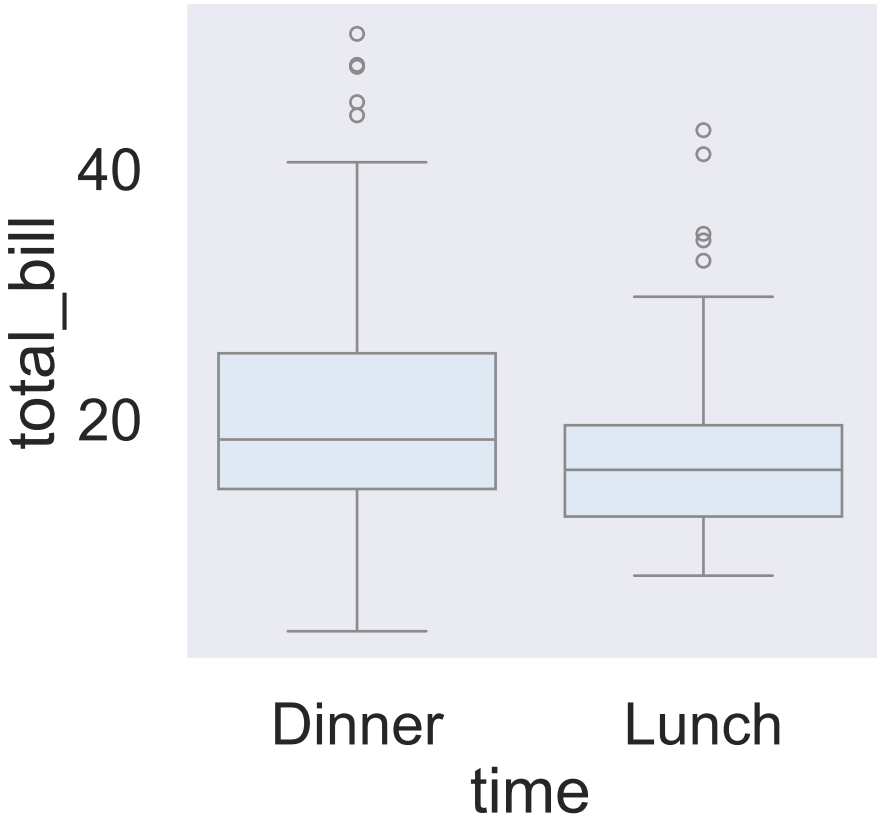


Figure 31: Distribution of Total Bill Amounts by Time Order By Time (Tips Dataset).

Omitting the outliers using `sym`

Occasionally, you may want to omit the outliers from your box plot. You can do this using the `"sym"` parameter. If you pass an empty string into `"sym"`, it will omit the outliers from your plot altogether. `"Sym"` can also be used to change the appearance of the outliers instead of omitting them.

Fixing the TypeError in Seaborn Box Plot

The error occurs because the `sym` argument — which controls outlier symbols — is **not supported** in `sns.catplot(kind="box")` in recent versions of Seaborn.
The `sym` parameter only works with **`sns.boxplot()`**, not with the higher-level `catplot()`.
In addition, In **older versions of Seaborn (< 0.12)**, you could write:

```
sns.boxplot(..., sym="")
```

to hide outlier points ("fliers").
However, starting from Seaborn 0.12+, the library underwent a full internal refactor (to unify plotting with the new object-oriented API). As a result, the `sym` parameter was removed and replaced with `fliersize`.

```
import seaborn as sns
import matplotlib.pyplot as plt

# Load Seaborn's 'tips' dataset
tips = sns.load_dataset("tips")

# Correct usage: replace 'sym' with 'fliersize=0'
sns.boxplot(
    x="time",
    y="total_bill",
    data=tips,
    order=["Dinner", "Lunch"],
    fliersize=0     # hides outlier markers
)

plt.show()
```

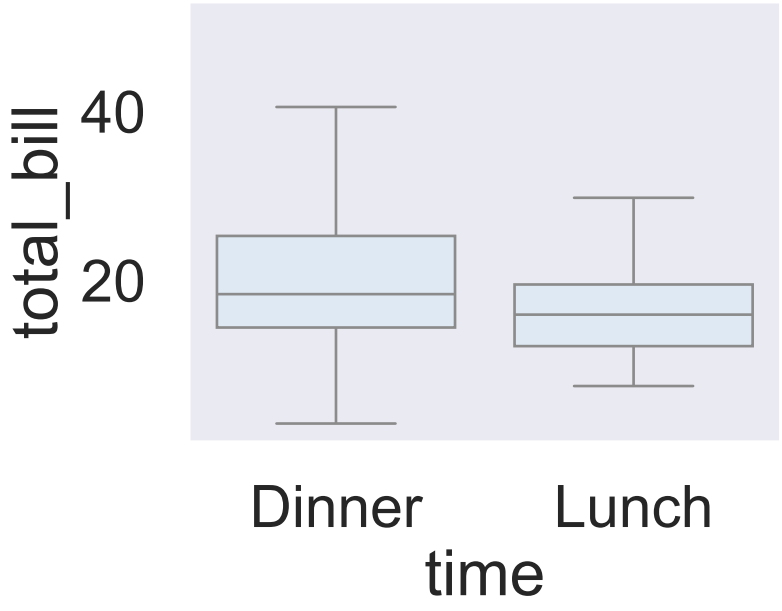


Figure 32: Box plot of total bill by meal time (outliers hidden).

If you want to hide outlier points in a `catplot()`, use the `fliersize=0` argument instead.

Corrected Code:

```
import seaborn as sns
import matplotlib.pyplot as plt

# Load dataset
tips = sns.load_dataset("tips")

# Correct approach: use fliersize=0 to hide outliers in catplot
sns.catplot(
    x="time",
    y="total_bill",
    data=tips,
    kind="box",
    order=["Dinner", "Lunch"],
    fliersize=0   # hides outlier markers
)

plt.title("Box Plot of Total Bill by Meal Time")
plt.show()
```

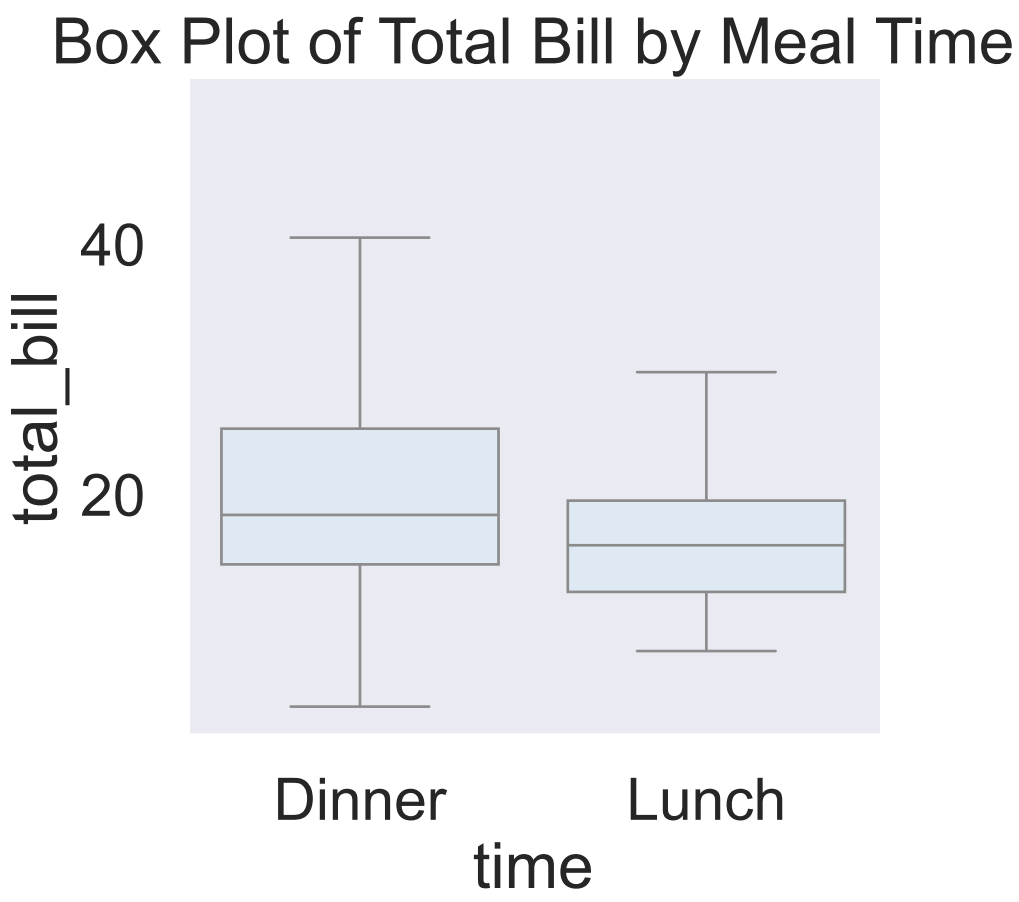


Figure 33: Box plot of total bill by meal time (outliers hidden).

```
# Load Seaborn's 'tips' dataset
tips = sns.load_dataset("tips")

# Create a box plot using catplot
sns.catplot(
    x= "time",
    y="total_bill",
    data=tips,
    kind="box",
    order = ["Dinner", "Lunch"],
    fliersize=0
)

plt.show()
```

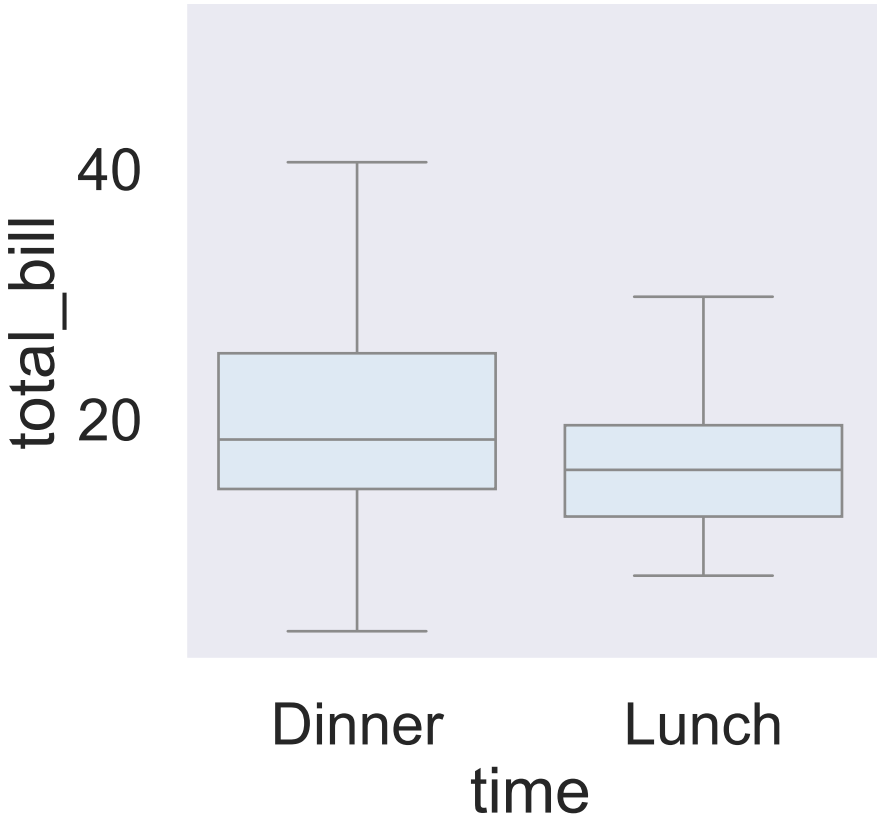


Figure 34: Distribution of Total Bill Amounts by Time Order By Time (Tips Dataset).

Changing the whiskers using `whis`

By default, the whiskers extend to 1.5 times the interquartile range, or `"IQR"`. The IQR is the 25th to the 75th percentile of a distribution of data. If you want to change the way the whiskers in your box plot are defined, you can do this using the `"whis"` parameter. There are several options for changing the whiskers. You can change the range of the whiskers from 1.5 times the IQR (which is the default) to 2 times the IQR by setting `"whis"` equal to 2.0. Alternatively, you can have the whiskers define specific lower and upper percentiles by passing in a list of the lower and upper values. In this example, passing in `"[5, 95]"` will result in the lower whisker being drawn at the 5th percentile and the upper whisker being drawn at the 95th percentile. Finally, you may just want to draw the whiskers at the min and max values. You can do this by specifying the lower percentile as 0 and the upper percentile as 100.

Option 1: Whiskers at 2.0 × IQR

```
# Load Seaborn's 'tips' dataset
tips = sns.load_dataset("tips")

# Create a box plot using catplot, with whiskers at 2.0 x IQR
sns.catplot(
    x="time",
    y="total_bill",
    data=tips,
    kind="box",
    order=["Dinner", "Lunch"],
    whis=2.0
)


plt.show()
```

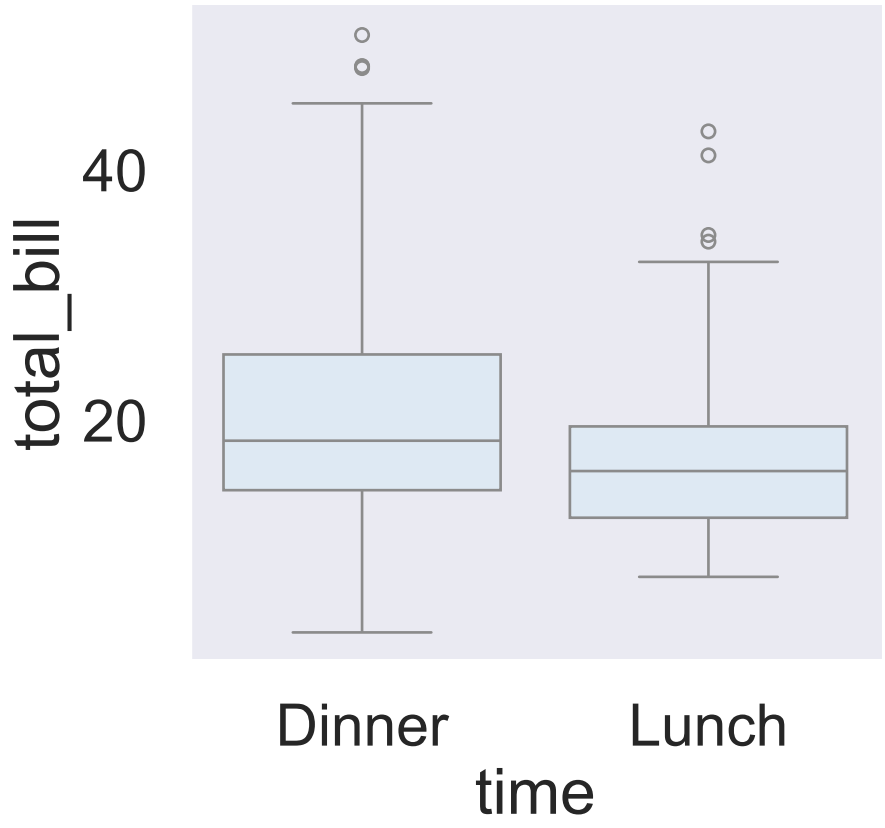


Figure 35: Distribution of Total Bill Amounts by Time Order By Time with whiskers at 2.0 x IQR.

Option 2: Whiskers at 5th and 95th Percentiles

```
# Load Seaborn's 'tips' dataset
tips = sns.load_dataset("tips")

# Create a box plot using catplot, with whiskers at 5th and 95th Percentiles
sns.catplot(
    x="time",
    y="total_bill",
    data=tips,
    kind="box",
    order=["Dinner", "Lunch"],
    whis=[5, 95]
)

plt.show()
```

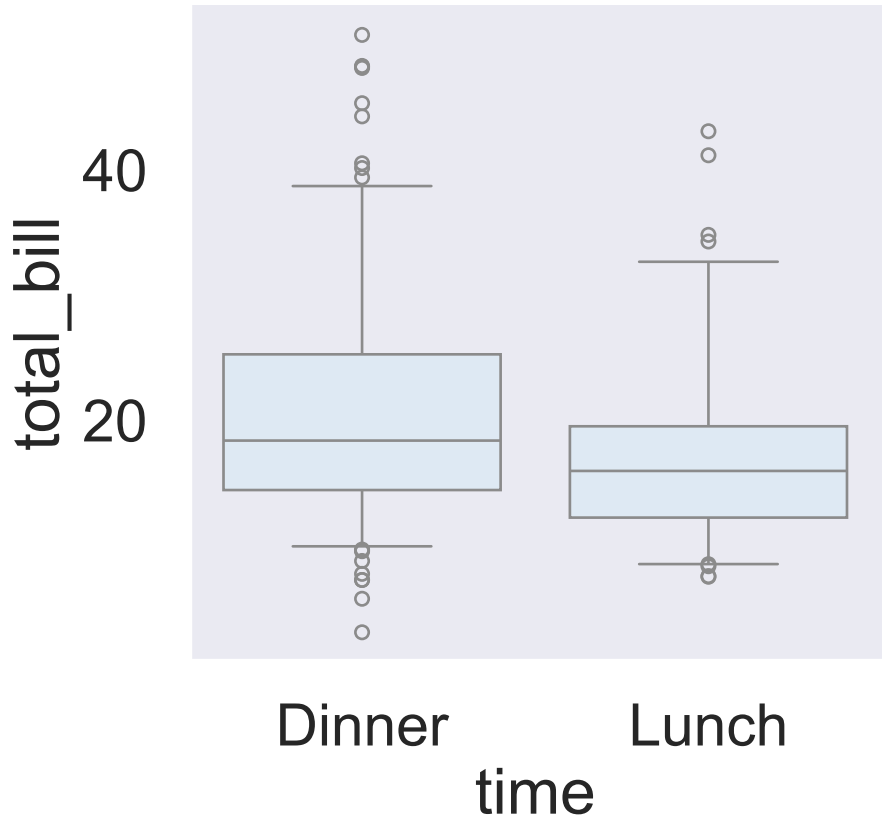


Figure 36: Distribution of Total Bill Amounts by Time Order By Time with Whiskers at 5th and 95th Percentiles.

Changing the whiskers using `whis`

Here's an example where the whiskers are set to the min and max values. Note that there are no outliers, because the box and whiskers cover the entire range of the data.

```
# Load Seaborn's 'tips' dataset
tips = sns.load_dataset("tips")

# Create a box plot using catplot, with whiskers at Min and Max (0th and 100th Percentiles)
sns.catplot(
    x="time",
    y="total_bill",
    data=tips,
    kind="box",
    order=["Dinner", "Lunch"],
    whis=[0, 100]
)

plt.show()
```

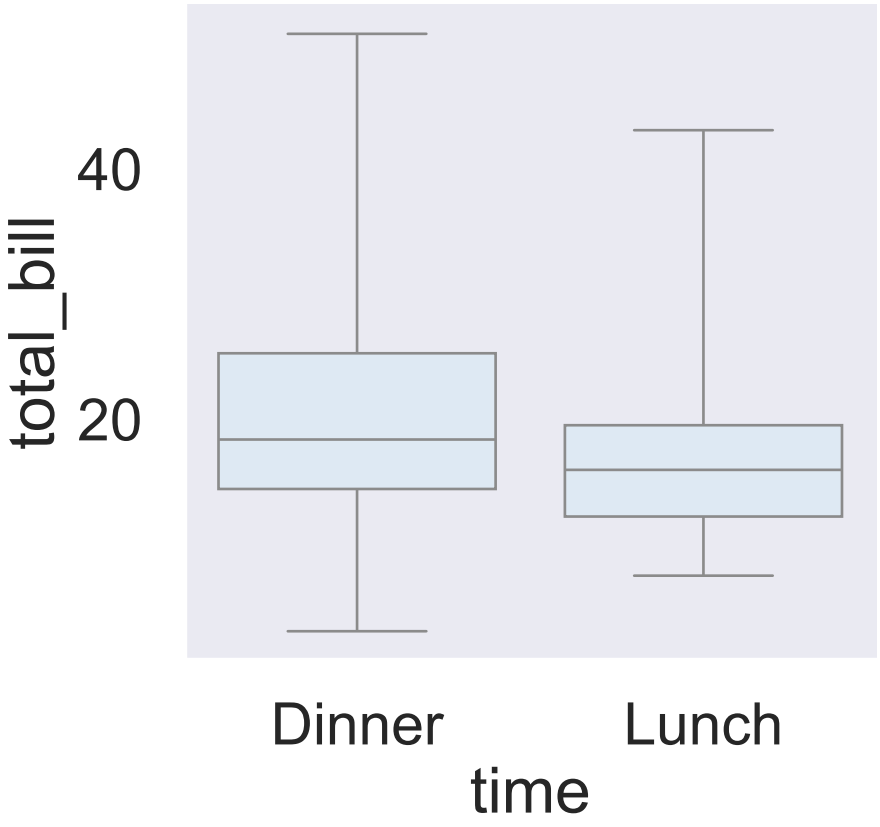


Figure 37: Distribution of Total Bill Amounts by Time Order By Time with Whiskers at Min and Max (0th and 100th Percentiles).

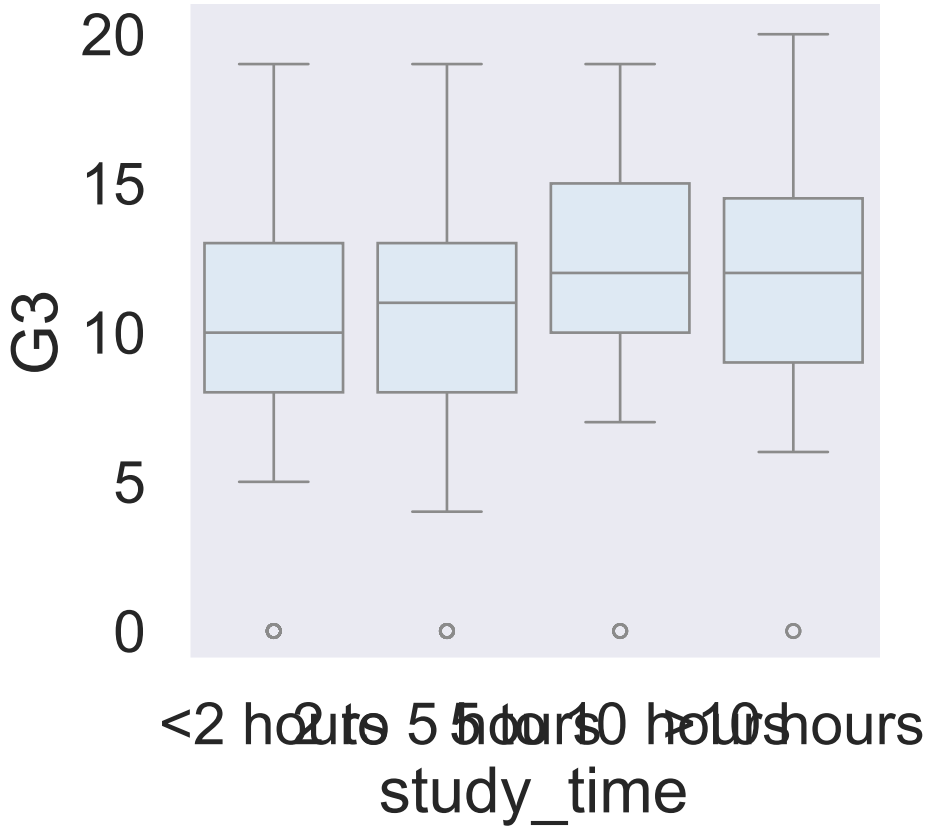
Exercise 4

---

1. Use `sns.catplot()` to create a box plot with the `student_data` DataFrame, putting `"study_time"` on the x-axis and `"G3"` on the y-axis.. Specify the category ordering, using this template, **`study_time_order = ["<2 hours", "2 to 5 hours", "5 to 10 hours", ">10 hours"]`**.

```
# Import packages
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Import dataset
student_data = pd.read_csv('datasets/student-alcohol-consumption.csv')
```

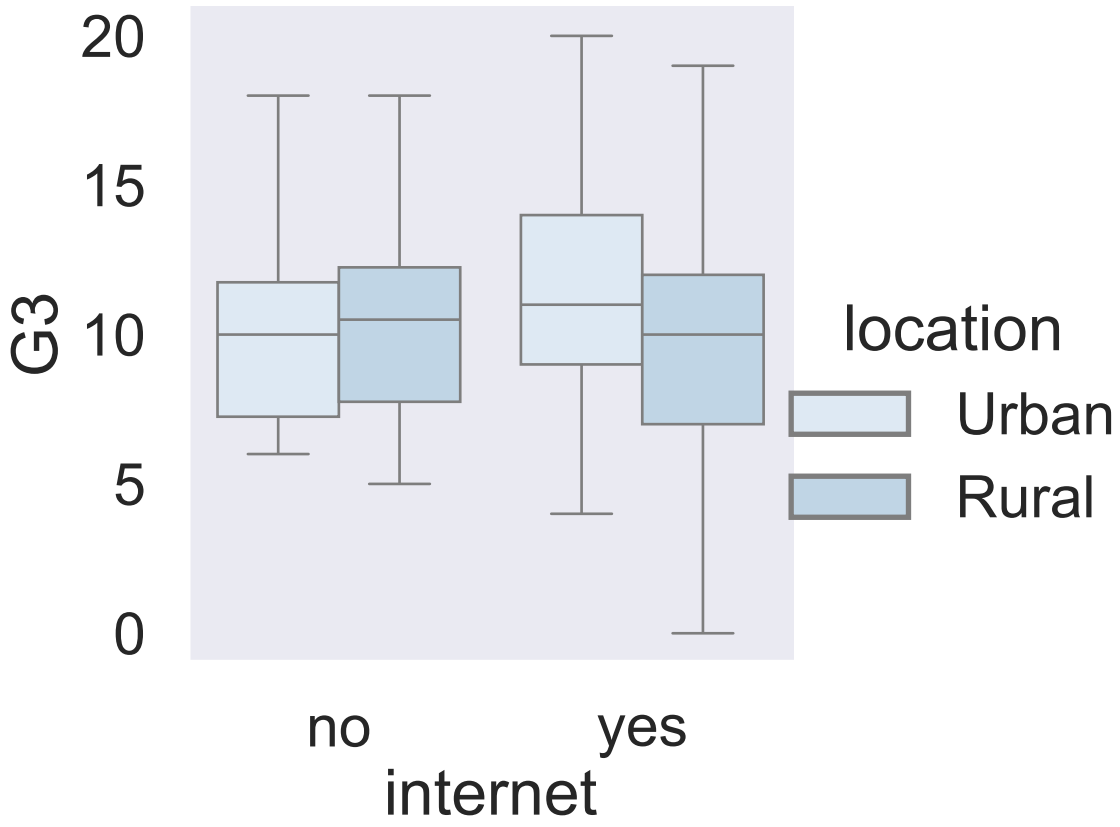
```
# Specify the category ordering
study_time_order = ["<2 hours", "2 to 5 hours",
                    "5 to 10 hours", ">10 hours"]

# Create a box plot and set the order of the categories
sns.catplot(x= "study_time", y="G3", data=student_data, kind="box", order=study_time_order)

# Show plot
plt.show()
```



2. Use `sns.catplot()` to create a box plot with the `student_data` DataFrame, putting `"internet"` on the x-axis and `"G3"` on the y-axis. Add subgroups so each box plot is colored based on `"location"`. Do not display the outliers.

```
# Import packages
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Import dataset
student_data = pd.read_csv('datasets/student-alcohol-consumption.csv')
```

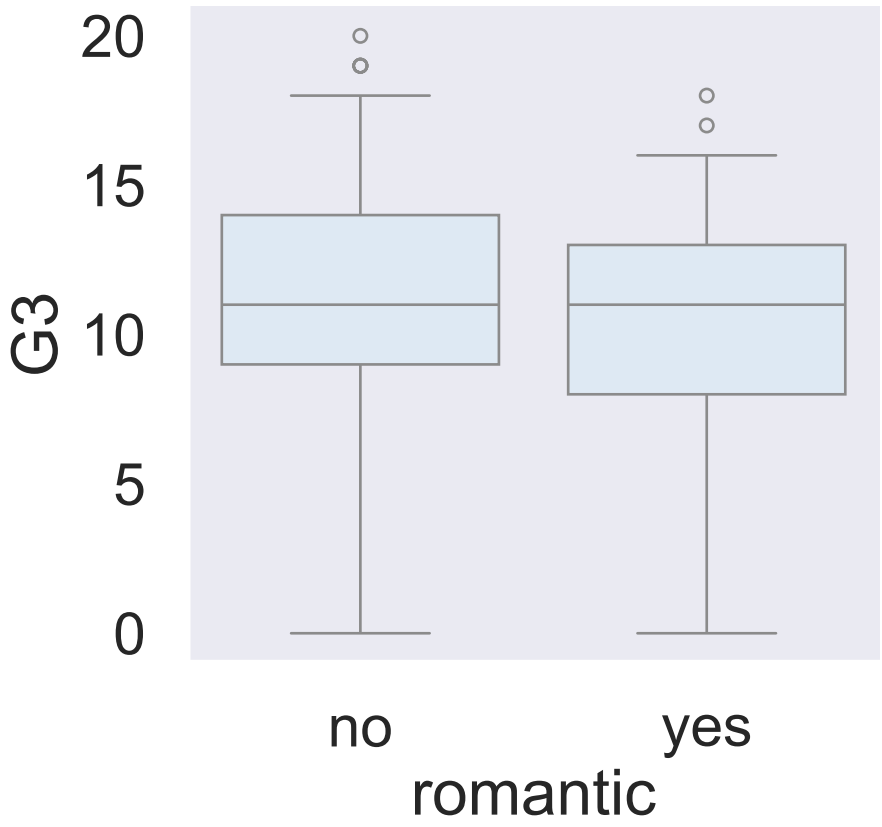
```
# SOLUTION
sns.catplot(x="internet", y="G3", data=student_data, kind="box", hue="location", fliersize=0)

# Show plot
plt.show()
```



3. Use `sns.catplot()` to create a box plot with the `student_data` DataFrame, putting `"romantic"` on the x-axis and `"G3"` on the y-axis. Adjust the code to make the box plot whiskers to extend to `0.5 * IQR`. Recall: the IQR is the interquartile range.

```
# Import packages
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Import dataset
student_data = pd.read_csv('datasets/student-alcohol-consumption.csv')

# Change the code to set the whiskers to extend to the 5th and 95th percentiles.
sns.catplot(x="romantic", y="G3",
```

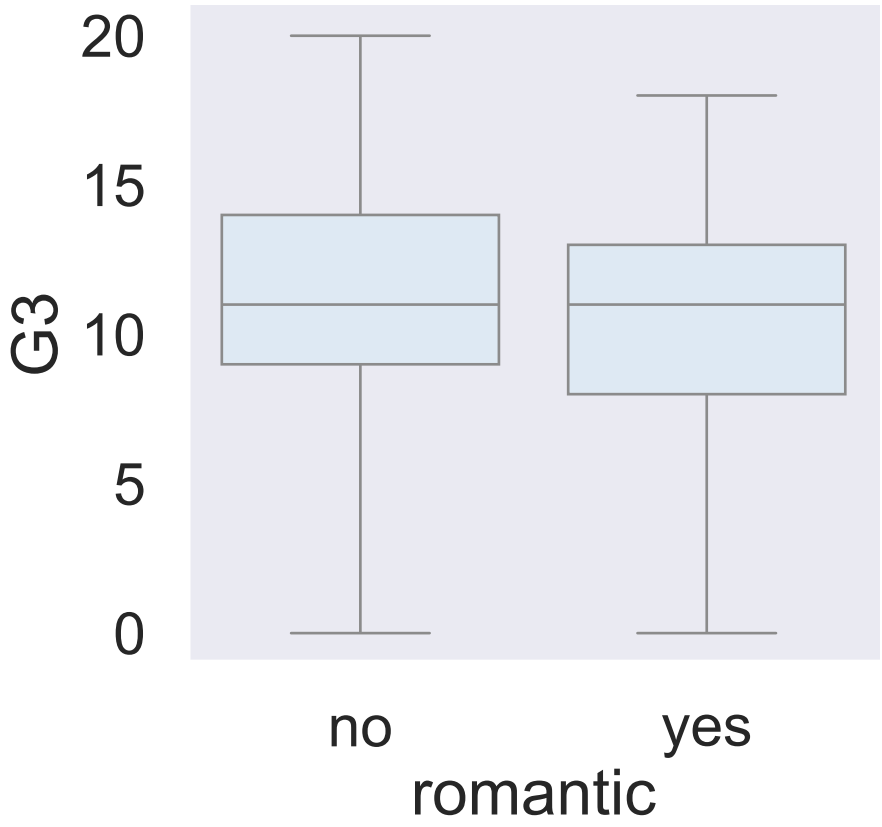
```
            data=student_data,
            kind="box",
            whis=[5, 95])

# Show plot
plt.show()
```



4. Change the code to set the whiskers to extend to the min and max values.

```
# Import packages
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Import dataset
student_data = pd.read_csv('datasets/student-alcohol-consumption.csv')


# Change the code to set the whiskers to extend to the min and max values.
sns.catplot(x="romantic", y="G3",
            data=student_data,
```

```
            kind="box",
            whis=[0, 100])

# Show plot
plt.show()
```



Chapter 4.1: Point plots

Welcome! So far we've seen several types of categorical plots including count plots, bar plots, and box plots. In this lesson, we'll see one final categorical plot: point plots.

What are point plots?

Point plots show the **mean of a quantitative variable** for the observations in each category, displayed as a single point. This point plot uses the **tips dataset** and compares the **average total bill** between **smokers** and **non-smokers**. The **vertical bars** extending above and below each point represent the **95% confidence intervals** for the mean. Just like the confidence intervals we saw in **line plots** and **bar plots**, they show the **uncertainty** in our mean estimates. Assuming the data is a random sample from a larger population, we can be 95% confident that the **true population mean** for each group lies within the interval shown.

```
# Load Seaborn's 'tips' dataset
tips = sns.load_dataset("tips")

# Create point plot using catplot
sns.catplot(
    x="smoker",
    y="total_bill",
    data=tips,
    kind="point"
)


plt.show()
```

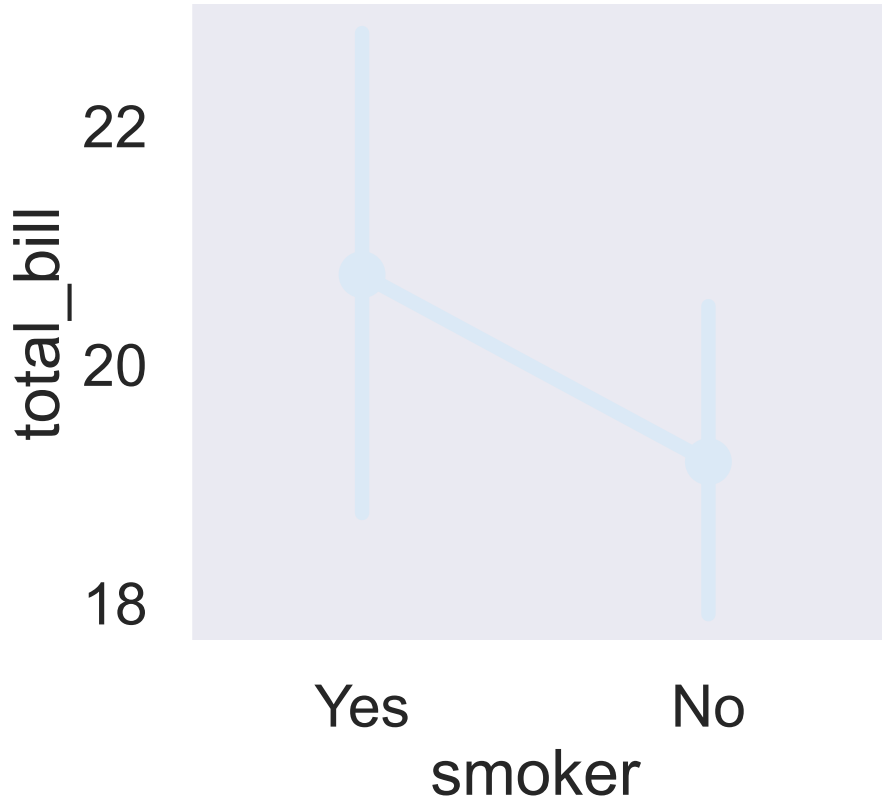


Figure 38: Average Total Bill by Smoking Status (Tips Dataset).

Point plots vs. line plots

You may be thinking: point plots look a lot like line plots. What's the difference? Both line plots and point plots show the mean of a quantitative variable and 95% confidence intervals for the mean. However, there is a key difference. Line plots are relational plots, so both the x- and y-axis are quantitative variables. In a point plot, one axis - usually the x-axis - is a categorical variable, making it a categorical plot.

Point plots vs. bar plots

You may also be thinking: point plots seem to show the same information as bar plots. For each category, both show the mean of a quantitative variable and the confidence intervals for those

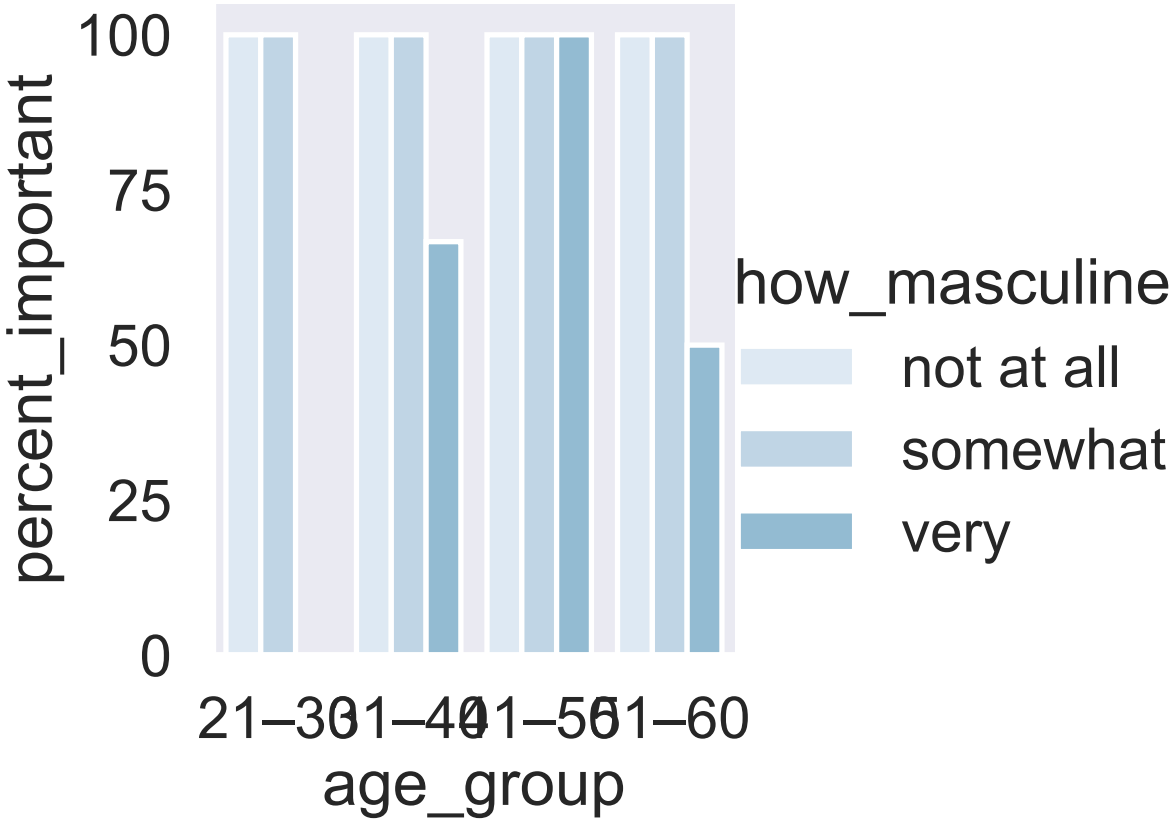
means. When should we use one over the other? Let's look at an example using data from the masculinity survey that we've seen in prior lessons Section and Figure 25.

This bar plot shows the **percent of men per age group** who report thinking that it's important that others see them as masculine, with subgroups based on whether they report **feeling masculine or not**.

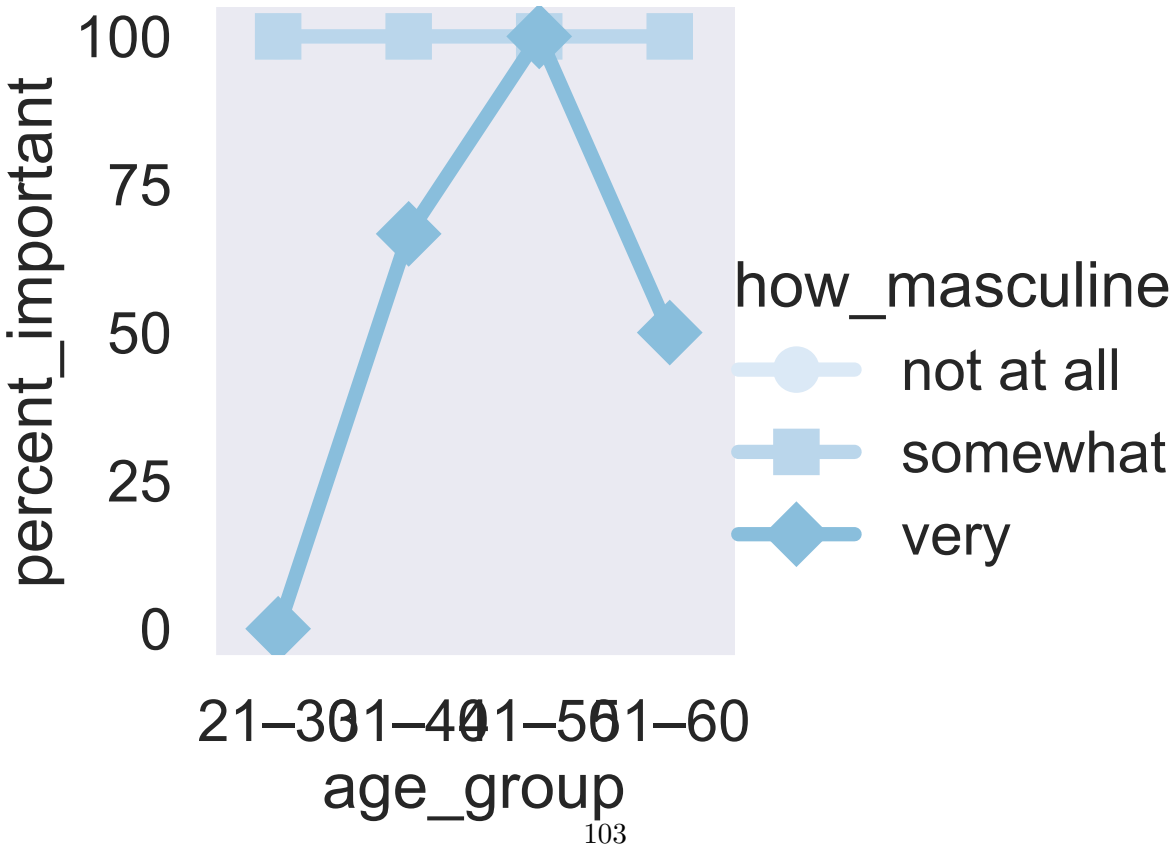
The **point plot** below displays the same data in a different form. In the point plot, it's easier to compare the heights of subgroup points when they're stacked above each other and to see differences in slope between categories — a clearer way to observe how perceptions change across age groups.

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

# Original data
masculinity = {
    'participant_id': list(range(1, 21)),
    'age': [35, 42, 29, 51, 37, 30, 45, 55, 25, 33, 40, 48, 28, 39, 47, 53, 26, 34, 43, 50],
    'how_masculine': ['very', 'somewhat', 'very', 'not at all', 'somewhat', 'very', 'somewhat', '
                      'not at all', 'somewhat', 'very', 'somewhat', 'very', 'not at all', 'somewh
                      'very', 'somewhat', 'very', 'not at all', 'somewhat'],
    'how_important': ['very', 'somewhat', 'not very', 'somewhat', 'very', 'very', 'somewhat', 'no
                      'somewhat', 'very', 'very', 'somewhat', 'not very', 'somewhat', 'very', 've
                      'somewhat', 'not very', 'somewhat', 'very']
}

df = pd.DataFrame(masculinity)

# Create age groups
bins = [20, 30, 40, 50, 60]
labels = ['21-30', '31-40', '41-50', '51-60']
df['age_group'] = pd.cut(df['age'], bins=bins, labels=labels, right=False)

# Compute % of "very" or "somewhat" important by age group & masculinity
df['important_binary'] = df['how_important'].isin(['very', 'somewhat']).astype(int)

summary = (
    df.groupby(['age_group', 'how_masculine'])['important_binary']
      .mean()
      .reset_index(name='percent_important')
)
summary['percent_important'] *= 100
```

```
# --- BAR PLOT ---
sns.catplot(
    x="age_group",
    y="percent_important",
    hue="how_masculine",
    data=summary,
    kind="bar"
)

plt.show()

# --- POINT PLOT ---
sns.catplot(
    x="age_group",
    y="percent_important",
    hue="how_masculine",
    data=summary,
    kind="point",
    markers=["o", "s", "D"]
)


plt.show()
```



(a) Importance of Being Seen as Masculine by Age Group and Self-Perception.



(b)

Figure 39

Creating a point plot

Here's the code to create the point plot we just saw. Since this is a categorical plot, we use `"catplot"` and set `"kind"` equal to `"point"`. Refer to this code chunk Figure 39.

Disconnecting the points

Sometimes we may want to remove the lines connecting each point, perhaps because we only wish to compare within a category group and not between them. To do this, set the `"join"` parameter equal to `False`.

```
# Import libraries
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Data provided by the user, extended to 20 rows
masculinity = {
    'participant_id': list(range(1, 21)),
    'age': [35, 42, 29, 51, 37, 30, 45, 55, 25, 33, 40, 48, 28, 39, 47, 53, 26, 34, 43, 50],
    'how_masculine': ['very', 'somewhat', 'very', 'not at all', 'somewhat', 'very', 'somewhat', '
                      'not at all', 'somewhat', 'very', 'somewhat', 'very', 'not at all', 'somewh
                      'very', 'somewhat', 'very', 'not at all', 'somewhat'],
    'how_important': ['very', 'somewhat', 'not very', 'somewhat', 'very', 'very', 'somewhat', 'no
                      'somewhat', 'very', 'very', 'somewhat', 'not very', 'somewhat', 'very', 've
                      'somewhat', 'not very', 'somewhat', 'very']
}

# Create the DataFrame
df = pd.DataFrame(masculinity)

# Create age groups
bins = [20, 30, 40, 50, 60]
labels = ['21-30', '31-40', '41-50', '51-60']
df['age_group'] = pd.cut(df['age'], bins=bins, labels=labels, right=False)

# --- POINT PLOT ---
sns.catplot(
    x="age_group",
```

```
    y="percent_important",
    hue="how_masculine",
    data=summary,
    kind="point",
    markers=["o", "s", "D"],
    join = False
)

plt.show()
```

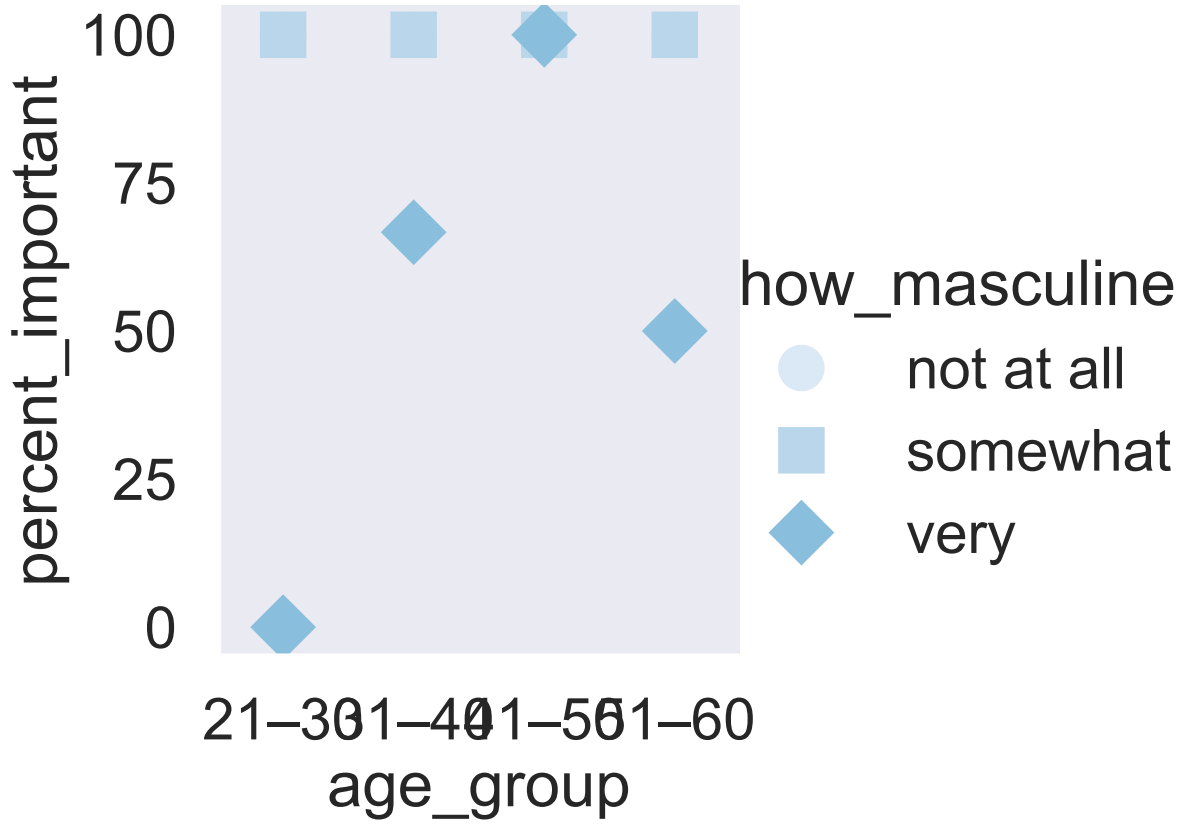


Figure 40: Importance of Being Seen as Masculine by Age Group and Self-Perception — Point Plot.

Displaying the median

Let's return to the point plot using the **tips** dataset and go over a few more ways to customize

your point plots.
Here is the point plot of **average bill comparing smokers to non-smokers**.

To have the points and confidence intervals be calculated for the **median** instead of the mean, we import the `median` function from the **NumPy** library and set the `"estimator"` parameter equal to the NumPy median function.

Why might you want to use the **median** instead of the mean?
The median is more robust to **outliers**, so if your dataset contains extreme values, the median can provide a better measure of the typical observation.

```
# Import libraries

import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Load the built-in 'tips' dataset

tips = sns.load_dataset("tips")

# Create a point plot using catplot with median as the estimator

sns.catplot(
data=tips,
x="smoker",
y="total_bill",
kind="point",
estimator=np.median   # Use median instead of mean
)


# Show the plot
plt.show()
```

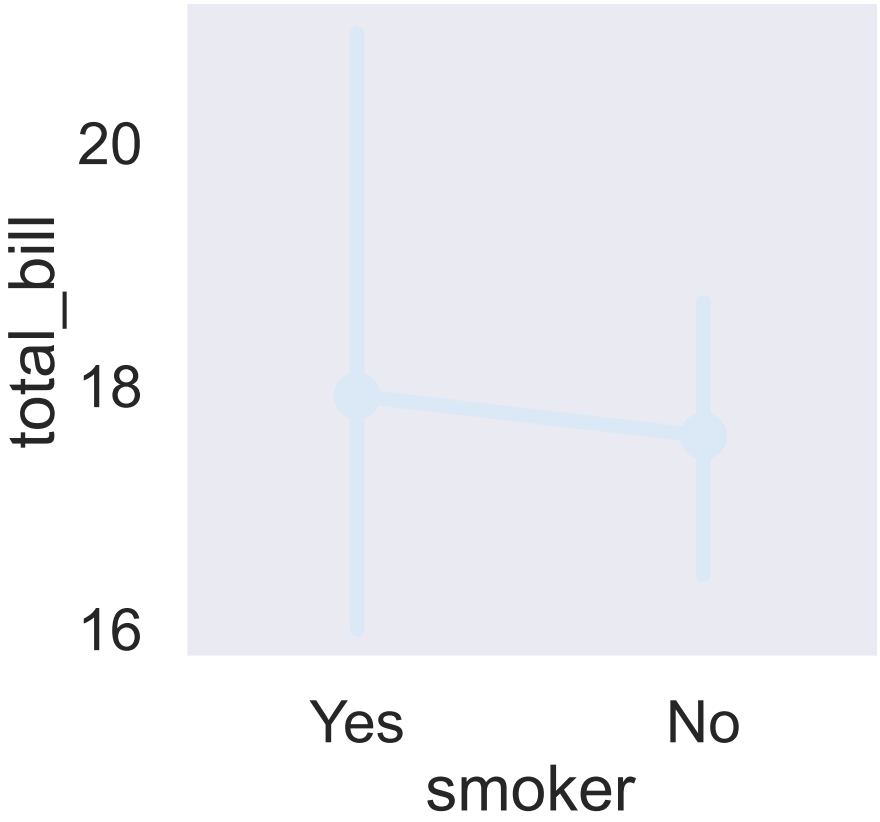


Figure 41: Median total bill by smoking status (tips dataset) using catplot.

Customizing the confidence intervals

You can also customize the way that the confidence intervals are displayed. To add `"caps"` to the end of the confidence intervals, set the `"capsize"` parameter equal to the desired width of the caps. In this case, we chose a width of 0.2.

```
# Import libraries

import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Load the built-in 'tips' dataset

tips = sns.load_dataset("tips")
```

```
# Create a point plot using catplot with median as the estimator

sns.catplot(
data=tips,
x="smoker",
y="total_bill",
kind="point",
estimator=np.median,    # Use median instead of mean
capsize=0.2             # Add small caps to error bars
)

# Add a main title
plt.suptitle("Median Total Bill by Smoking Status", y=1.02, fontsize=13)

# Show the plot
plt.show()
```

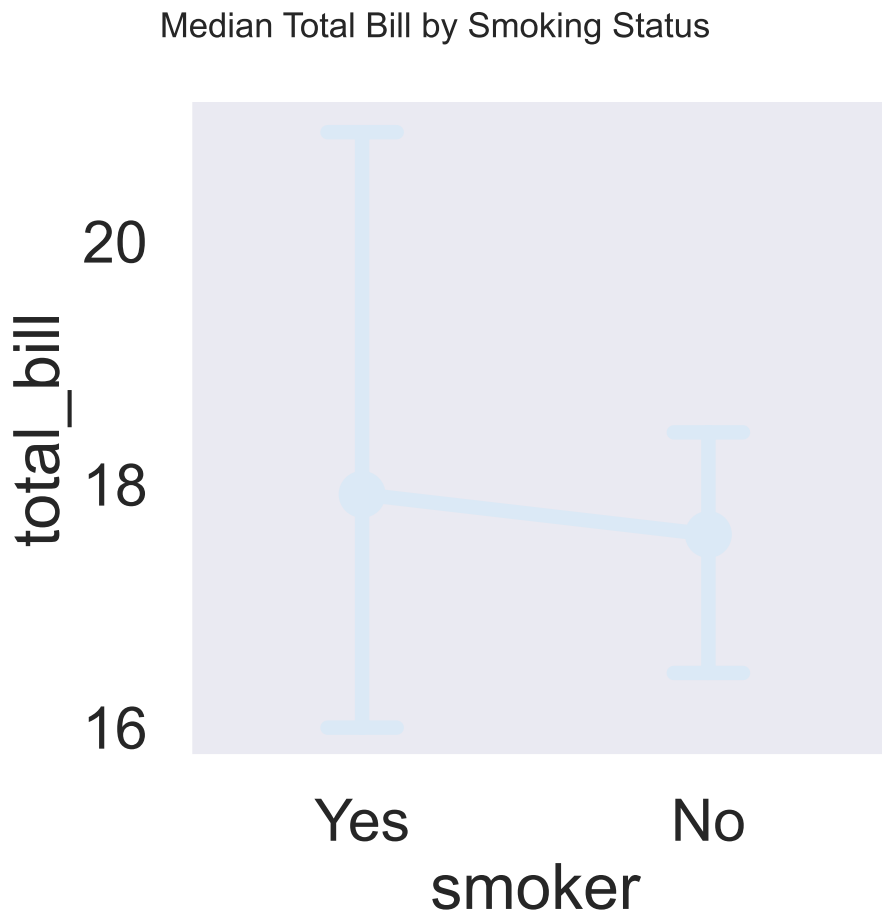


Figure 42: Median total bill by smoking status (tips dataset) using catplot.

Turning off confidence intervals

Finally, like we saw with line plots and bar plots, you can turn the confidence intervals off by setting the "errorbar" parameter equal to `None`.

```
# Import libraries

import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Load the built-in 'tips' dataset
```

```
tips = sns.load_dataset("tips")

# Create a point plot using catplot with median as the estimator

sns.catplot(
data=tips,
x="smoker",
y="total_bill",
kind="point",
estimator=np.median,   # Use median instead of mean
capsize=0.1,           # Add small caps to error bars
errorbar=None
)

# Add a main title
plt.suptitle("Median Total Bill by Smoking Status", y=1.02, fontsize=13)

# Show the plot
plt.show()
```

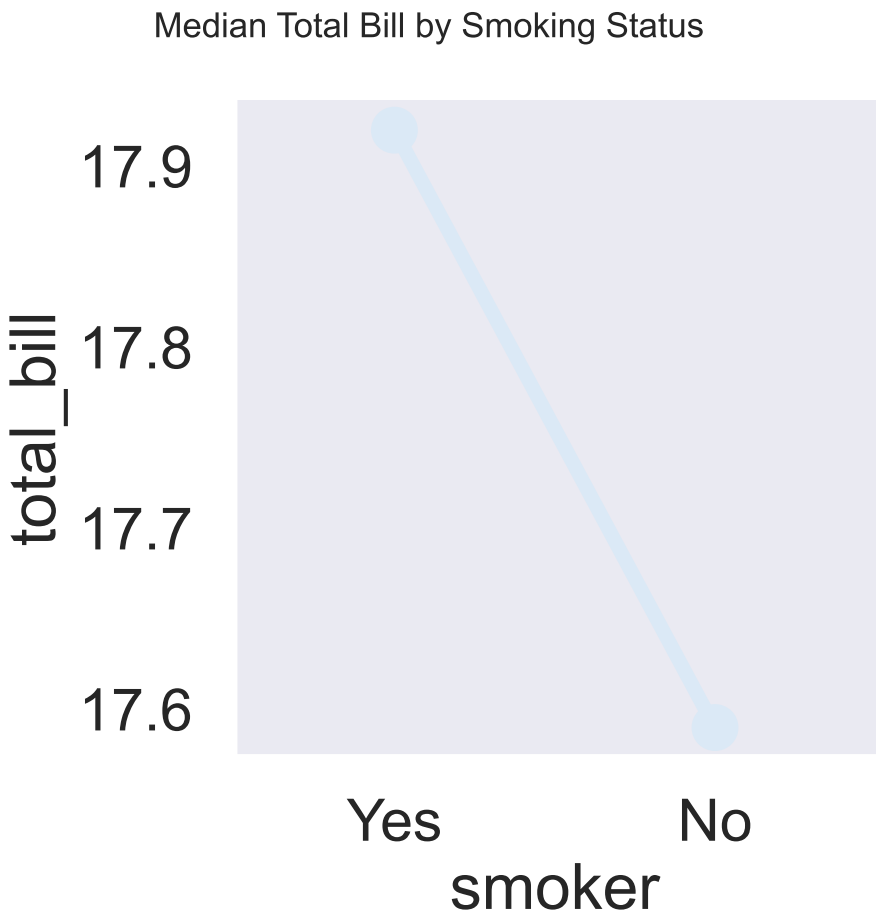


Figure 43: Median total bill by smoking status (tips dataset) using catplot.

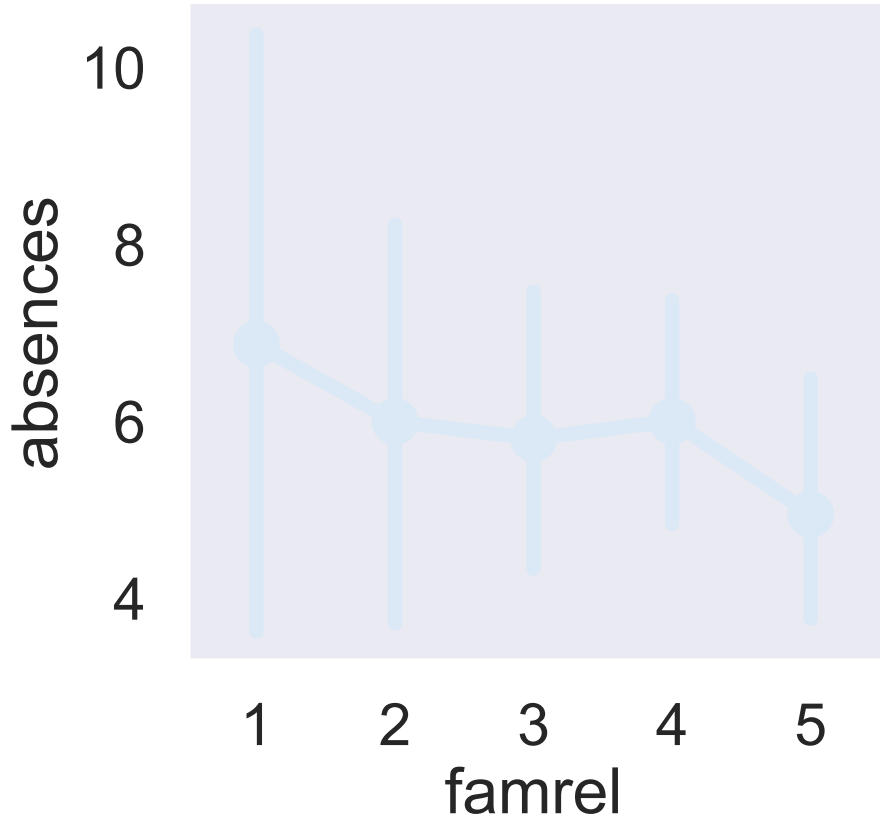
Exercise 4.1

---

1. Use `sns.catplot()` and the `student_data` DataFrame to create a point plot with `"famrel"` on the x-axis and number of absences (`"absences"`) on the y-axis.

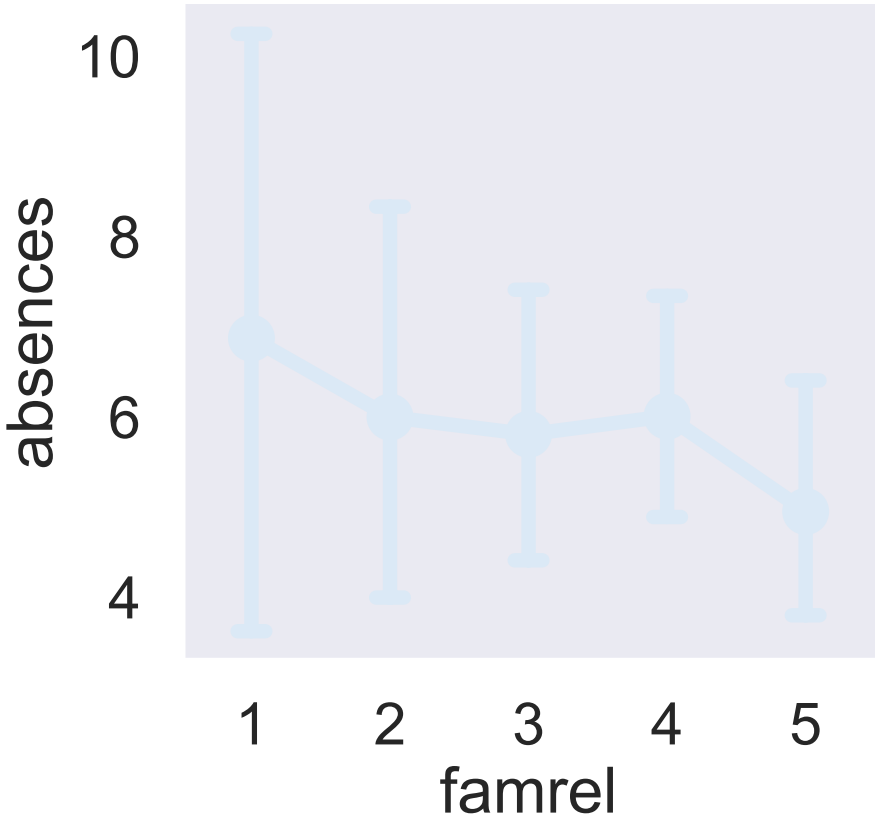
```
# Import packages
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Import dataset
student_data = pd.read_csv('datasets/student-alcohol-consumption.csv')

sns.catplot(x="famrel", y="absences", data=student_data, kind="point")
```

```
# Show plot
plt.show()
```



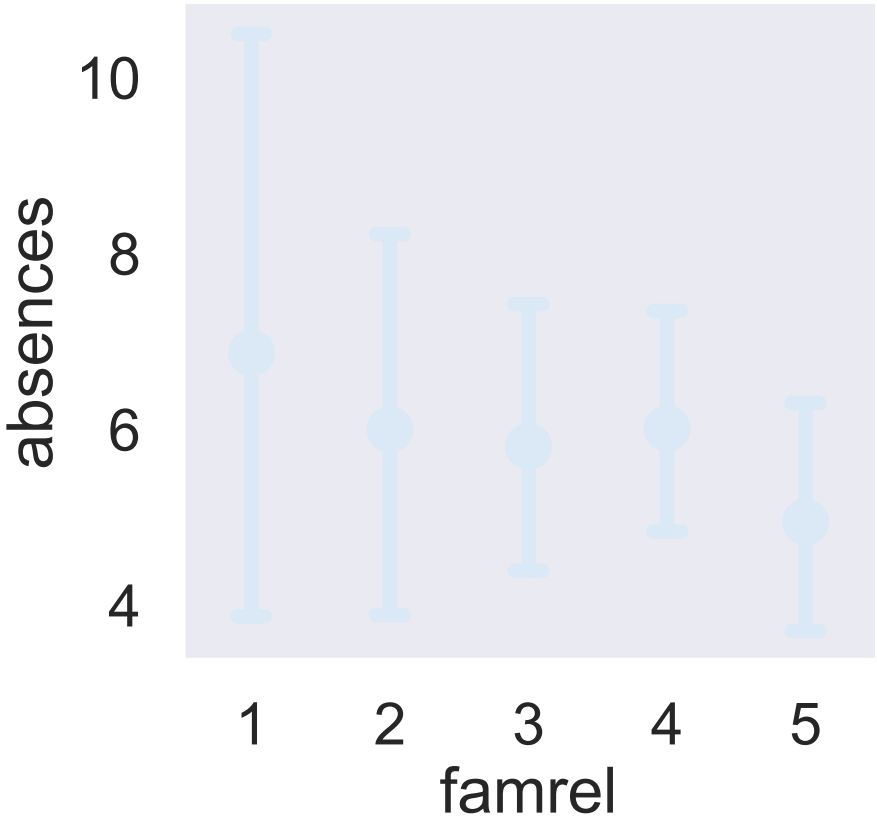
2. Add `"caps"` to the end of the confidence intervals with size 0.2.

```
# Import packages
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Import dataset
student_data = pd.read_csv('datasets/student-alcohol-consumption.csv')

sns.catplot(x="famrel", y="absences",
            data=student_data,
            kind="point", capsize=0.2)

# Show plot
plt.show()
```



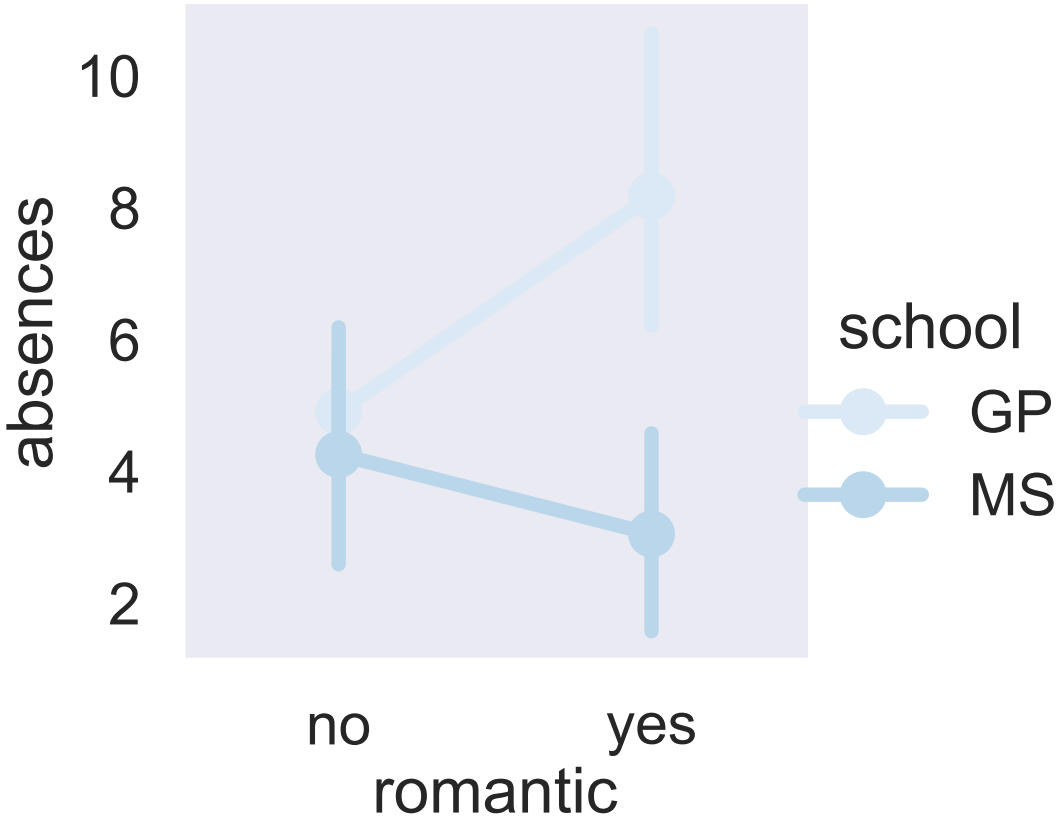
3. Remove the lines joining the points in each category.

```
# Import packages
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Import dataset
student_data = pd.read_csv('datasets/student-alcohol-consumption.csv')

sns.catplot(x="famrel", y="absences",
            data=student_data,
            kind="point",
            capsize=0.2, join=False)

# Show plot
plt.show()
```



4. Use `sns.catplot()` and the `student_data` DataFrame to create a point plot with relationship status (`"romantic"`) on the x-axis and number of absences (`"absences"`) on the y-axis. Color the points based on the school that they attend (`"school"`).

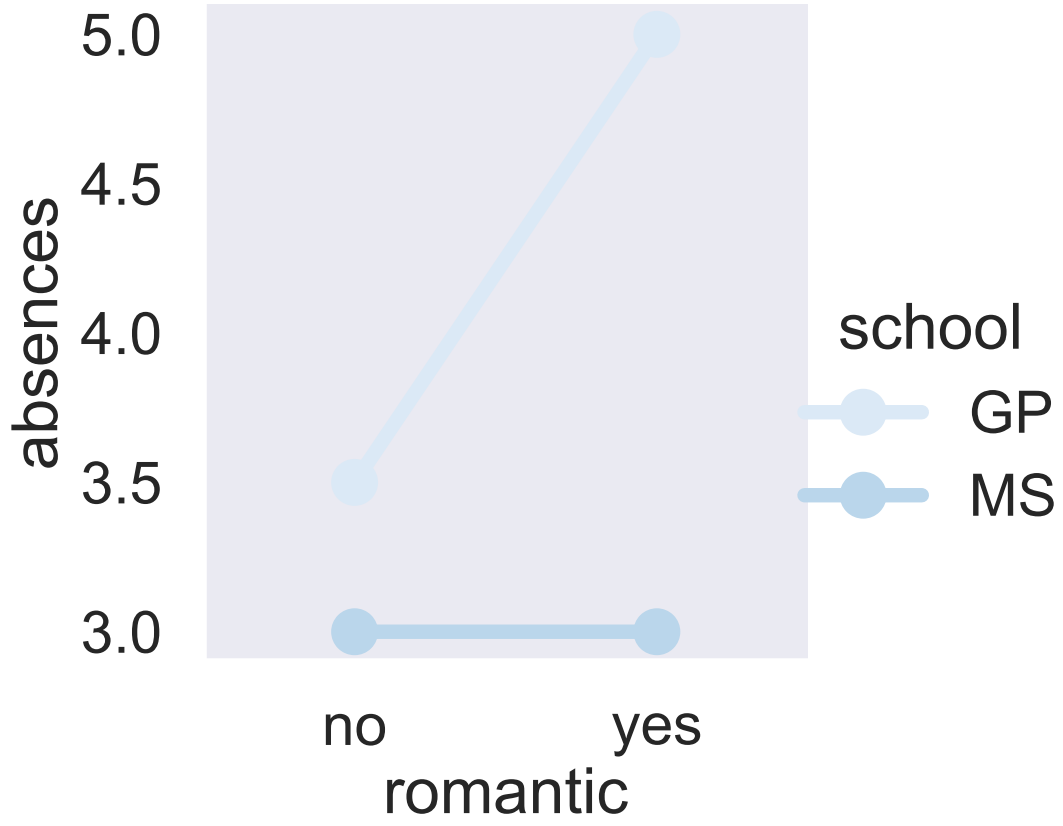
```
# Import packages
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Import dataset
student_data = pd.read_csv('datasets/student-alcohol-consumption.csv')

sns.catplot(x="romantic", y="absences", data=student_data, kind="point", hue="school")

# Show plot
plt.show()
```



5. Since there may be outliers of students with many absences, use the median function that we've imported from `numpy` to display the `median` number of absences instead of the average.

```
# Import packages
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Import dataset
student_data = pd.read_csv('datasets/student-alcohol-consumption.csv')

# Import median function from numpy
from numpy import median

# Plot the median number of absences instead of the mean
sns.catplot(x="romantic", y="absences",
            data=student_data,
            kind="point",
            hue="school",
            errorbar=None, estimator=median)
```

```
# Show plot
plt.show()
```



Chapter 5: Changing plot style and color

So far we've covered how to create a variety of different plot types. Now let's learn how to customize them.

Why customize?

By default, Seaborn plots are pleasing to look at, but there are several reasons you may want to change the appearance. Changing the style of a plot can be motivated by personal preference, but it can also help improve its readability or help orient an audience more quickly to the key takeaway.

Changing the figure style

Seaborn has five preset figure styles which change the background and axes of the plot. You can refer to them by name: **"white", "dark"**, **"whitegrid"**, **"darkgrid"**, and **"ticks"**. To set one of these as the global style for all of your plots, use the **"set style"** function.

Default figure style ("white")

This is a plot, Figure 39, we've seen before, showing the percentage of men reporting that masculinity was important to them, stratified by their age and whether or not they feel masculine. The default style is called "white" and provides clean axes with a solid white background. If we only care about the comparisons between groups or the general trend across age groups instead of the specific values, this is a good choice.

Figure style: "whitegrid"

Changing the style to **"whitegrid"** will add a gray grid in the background. This is useful if you want your audience to be able to determine the specific values of the plotted points instead of making higher level observations.

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

# Set the style to "whitegrid"
sns.set_style("whitegrid")

# Original data
masculinity = {
    'participant_id': list(range(1, 21)),
    'age': [35, 42, 29, 51, 37, 30, 45, 55, 25, 33, 40, 48, 28, 39, 47, 53, 26, 34, 43, 50],
    'how_masculine': ['very', 'somewhat', 'very', 'not at all', 'somewhat', 'very', 'somewhat', '
                      'not at all', 'somewhat', 'very', 'somewhat', 'very', 'not at all', 'somewh
                      'very', 'somewhat', 'very', 'not at all', 'somewhat'],
    'how_important': ['very', 'somewhat', 'not very', 'somewhat', 'very', 'very', 'somewhat', 'no
                      'somewhat', 'very', 'very', 'somewhat', 'not very', 'somewhat', 'very', 've
```

```
                    'somewhat', 'not very', 'somewhat', 'very']
}

# Create age groups
bins = [20, 30, 40, 50, 60]
labels = ['21-30', '31-40', '41-50', '51-60']
df['age_group'] = pd.cut(df['age'], bins=bins, labels=labels, right=False)

# Compute % of "very" or "somewhat" important by age group & masculinity
df['important_binary'] = df['how_important'].isin(['very', 'somewhat']).astype(int)

summary = (
    df.groupby(['age_group', 'how_masculine'])['important_binary']
      .mean()
      .reset_index(name='percent_important')
)
summary['percent_important'] *= 100


# --- BAR PLOT ---
sns.catplot(
    x="age_group",
    y="percent_important",
    hue="how_masculine",
    data=summary,
    kind="bar"
)

plt.show()
```

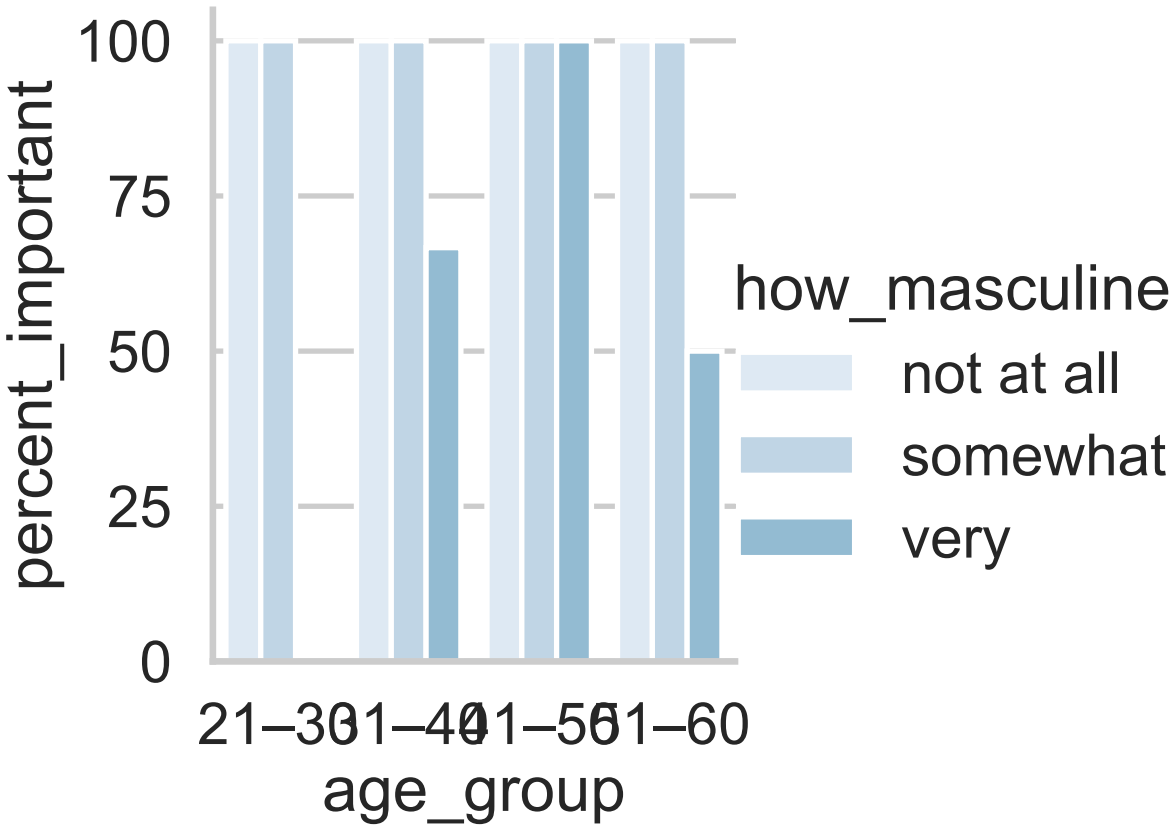


Figure 44: Importance of Being Seen as Masculine by Age Group and Self-Perception.

Other styles

The other styles are variants on these. `"ticks"` is similar to `"white"`, but adds small tick marks to the x- and y-axes. `"dark"` provides a gray background, and `"darkgrid"` provides a gray background with a white grid.

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

# Set the style to "whitegrid"
sns.set_style("whitegrid")

# Original data
masculinity = {
```

```
    'participant_id': list(range(1, 21)),
    'age': [35, 42, 29, 51, 37, 30, 45, 55, 25, 33, 40, 48, 28, 39, 47, 53, 26, 34, 43, 50],
    'how_masculine': ['very', 'somewhat', 'very', 'not at all', 'somewhat', 'very', 'somewhat', '
                      'not at all', 'somewhat', 'very', 'somewhat', 'very', 'not at all', 'somewh
                      'very', 'somewhat', 'very', 'not at all', 'somewhat'],
    'how_important': ['very', 'somewhat', 'not very', 'somewhat', 'very', 'very', 'somewhat', 'no
                      'somewhat', 'very', 'very', 'somewhat', 'not very', 'somewhat', 'very', 've
                      'somewhat', 'not very', 'somewhat', 'very']
}

# Create age groups
bins = [20, 30, 40, 50, 60]
labels = ['21-30', '31-40', '41-50', '51-60']
df['age_group'] = pd.cut(df['age'], bins=bins, labels=labels, right=False)

# Compute % of "very" or "somewhat" important by age group & masculinity
df['important_binary'] = df['how_important'].isin(['very', 'somewhat']).astype(int)

summary = (
    df.groupby(['age_group', 'how_masculine'])['important_binary']
      .mean()
      .reset_index(name='percent_important')
)
summary['percent_important'] *= 100


# --- BAR PLOT ---
sns.catplot(
    x="age_group",
    y="percent_important",
    hue="how_masculine",
    data=summary,
    kind="bar"
)

plt.show()
```

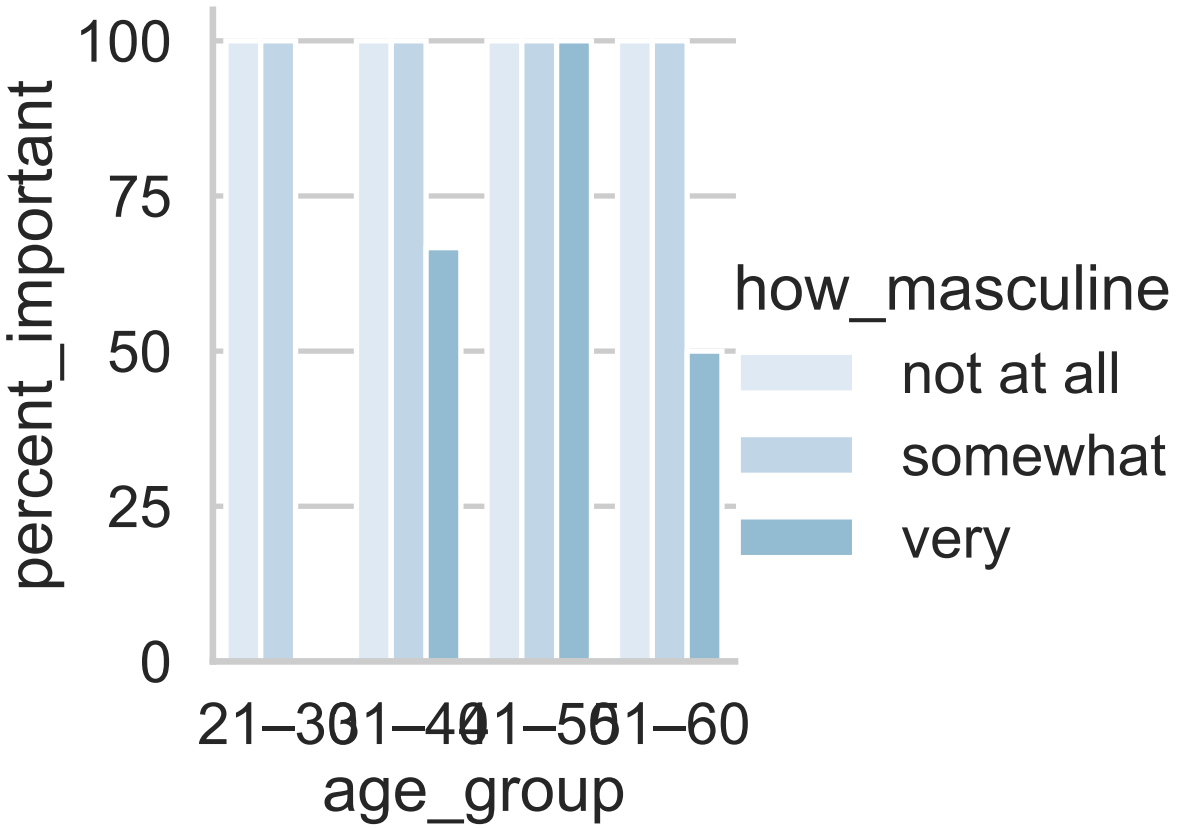


Figure 45: Importance of Being Seen as Masculine by Age Group and Self-Perception.

Changing the palette

You can change the color of the main elements of the plot with Seaborn's `"set palette"` function. Seaborn has many preset color palettes that you can refer to by name, or you can create your own custom palette. Let's see an example.

Diverging palettes

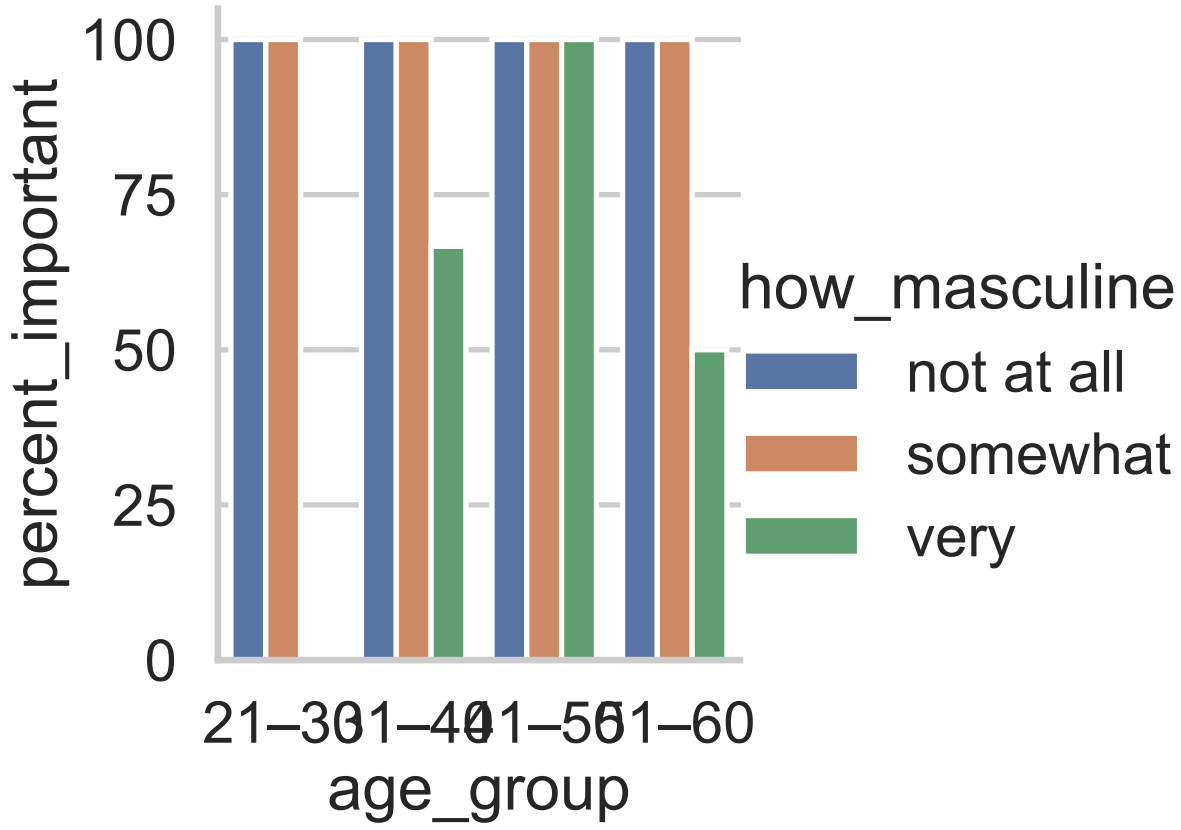
Seaborn has a group of preset palettes called diverging palettes that are great to use if your visualization deals with a scale where the two ends of the scale are opposites and there is a neutral midpoint. Here are some examples of diverging palettes - red/blue and purple/green. Note that if you append the palette name with `"_r"`, you can reverse the palette.

Example (default palette)

To see this in action, let's return to a count plot we've seen before of the responses of men reporting how masculine they feel.

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

# Set the style to "whitegrid"
sns.set_style("whitegrid")

# Set to default palette
sns.set_palette("deep")

# Original data
masculinity = {
    'participant_id': list(range(1, 21)),
    'age': [35, 42, 29, 51, 37, 30, 45, 55, 25, 33, 40, 48, 28, 39, 47, 53, 26, 34, 43, 50],
    'how_masculine': ['very', 'somewhat', 'very', 'not at all', 'somewhat', 'very', 'somewhat', '
                      'not at all', 'somewhat', 'very', 'somewhat', 'very', 'not at all', 'somewh
                      'very', 'somewhat', 'very', 'not at all', 'somewhat'],
    'how_important': ['very', 'somewhat', 'not very', 'somewhat', 'very', 'very', 'somewhat', 'no
                      'somewhat', 'very', 'very', 'somewhat', 'not very', 'somewhat', 'very', 've
                      'somewhat', 'not very', 'somewhat', 'very']
}


# Create age groups
bins = [20, 30, 40, 50, 60]
labels = ['21-30', '31-40', '41-50', '51-60']
df['age_group'] = pd.cut(df['age'], bins=bins, labels=labels, right=False)

# Compute % of "very" or "somewhat" important by age group & masculinity
df['important_binary'] = df['how_important'].isin(['very', 'somewhat']).astype(int)

summary = (
    df.groupby(['age_group', 'how_masculine'])['important_binary']
      .mean()
      .reset_index(name='percent_important')
)
summary['percent_important'] *= 100
```

```
# --- BAR PLOT ---
sns.catplot(
    x="age_group",
    y="percent_important",
    hue="how_masculine",
    data=summary,
    kind="bar"
)

plt.show()
```

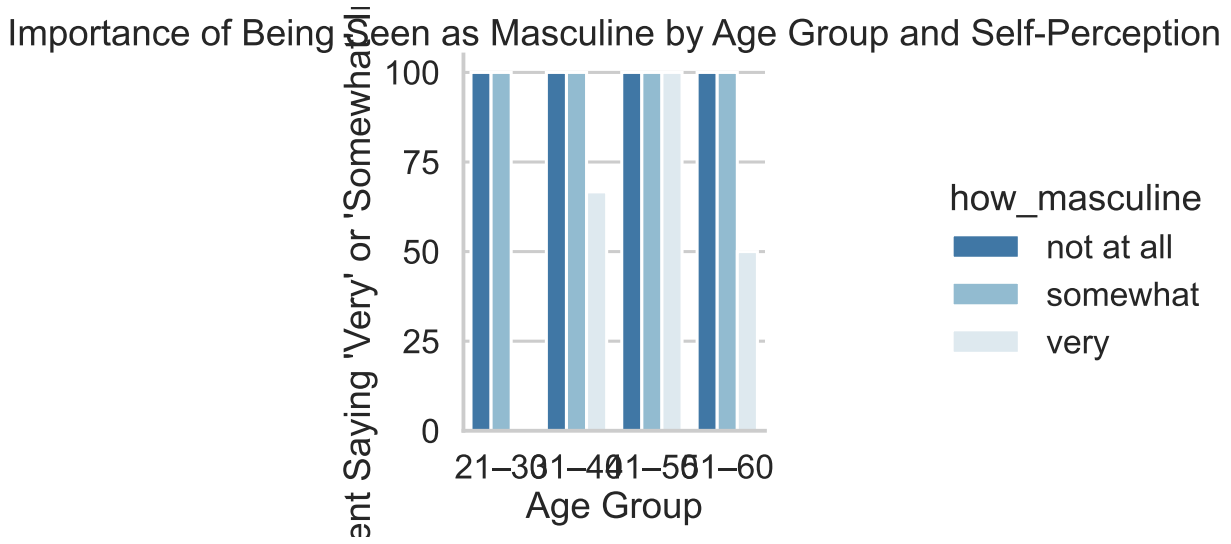


Example (diverging palette)

Setting this plot's palette to red/blue diverging provides a clearer contrast between the men who do not feel masculine and the men who do.

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

# Set the style to "whitegrid"
sns.set_style("whitegrid")

# Set a diverging color palette (reversed red-blue)
sns.set_palette("RdBu_r")

# Original data
masculinity = {
    'participant_id': list(range(1, 21)),
    'age': [35, 42, 29, 51, 37, 30, 45, 55, 25, 33, 40, 48, 28, 39, 47, 53, 26, 34, 43, 50],
    'how_masculine': ['very', 'somewhat', 'very', 'not at all', 'somewhat', 'very', 'somewhat', '
                      'not at all', 'somewhat', 'very', 'somewhat', 'very', 'not at all', 'somewh
                      'very', 'somewhat', 'very', 'not at all', 'somewhat'],
    'how_important': ['very', 'somewhat', 'not very', 'somewhat', 'very', 'very', 'somewhat', 'no
                      'somewhat', 'very', 'very', 'somewhat', 'not very', 'somewhat', 'very', 've
                      'somewhat', 'not very', 'somewhat', 'very']
}

# Create age groups
bins = [20, 30, 40, 50, 60]
labels = ['21-30', '31-40', '41-50', '51-60']
df['age_group'] = pd.cut(df['age'], bins=bins, labels=labels, right=False)

# Compute % of "very" or "somewhat" important by age group & masculinity
df['important_binary'] = df['how_important'].isin(['very', 'somewhat']).astype(int)

summary = (
    df.groupby(['age_group', 'how_masculine'])['important_binary']
      .mean()
      .reset_index(name='percent_important')
)
summary['percent_important'] *= 100

# --- BAR PLOT ---
sns.catplot(
    x="age_group",
    y="percent_important",
    hue="how_masculine",
    data=summary,
    kind="bar"
)
```

```
plt.title("Importance of Being Seen as Masculine by Age Group and Self-Perception")
plt.xlabel("Age Group")
plt.ylabel("Percent Saying 'Very' or 'Somewhat' Important")
plt.show()
```



Sequential palettes

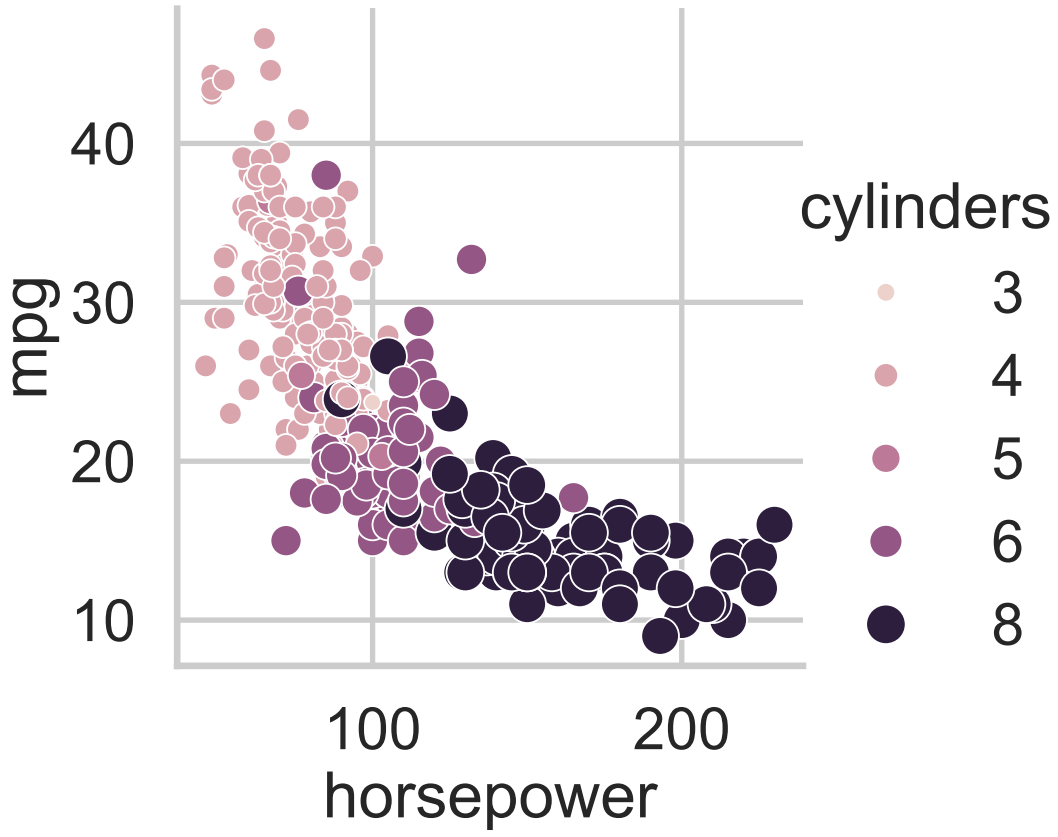
Another group of palettes are called sequential palettes. These are a single color (or two colors blended) moving from light to dark values.

Sequential palette example

Sequential palettes are great for emphasizing a variable on a continuous scale. One example is this plot depicting the relationship between a car's horsepower and its miles per gallon, where points grow larger and darker when the car has more cylinders.

```
# Import packages
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Import dataset
```

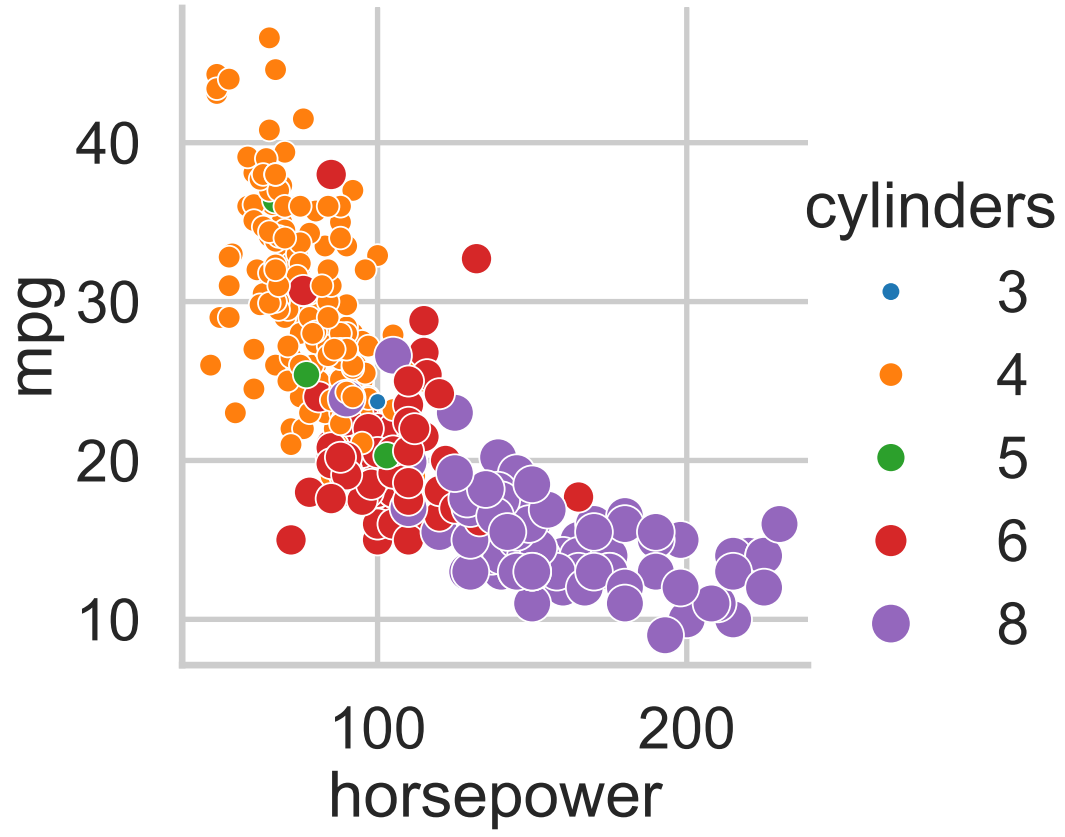
```
mpg = pd.read_csv("datasets/mpg.csv")

# Set a sequential palette (e.g., "Blues")
sns.set_palette("Blues")

# Create scatter plot of horsepower vs. mpg
# Map 'cylinders' to both size and color (hue)
sns.relplot(
    x="horsepower",
    y="mpg",
    data=mpg,
    kind="scatter",
    size="cylinders",
    hue="cylinders",
    sizes=(40, 200)
)

# Show plot
plt.show()
```



Custom palettes

You can also create your own custom palettes by passing in a list of color names... or a list of hex color codes.

```
# Import packages
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Import dataset
mpg = pd.read_csv("datasets/mpg.csv")

# Define a custom palette using hex codes or color names
custom_palette = ["#1f77b4", "#ff7f0e", "#2ca02c", "#d62728", "#9467bd"]

# Create a mapping from unique cylinder values to colors
unique_cylinders = sorted(mpg["cylinders"].unique())
palette_dict = dict(zip(unique_cylinders, custom_palette))

# Create scatter plot of horsepower vs. mpg
sns.relplot(
    x="horsepower",
    y="mpg",
    data=mpg,
    kind="scatter",
    size="cylinders",
    hue="cylinders",
    palette=palette_dict,
    sizes=(40, 200)
)

# Show plot
plt.show()
```



Changing the scale

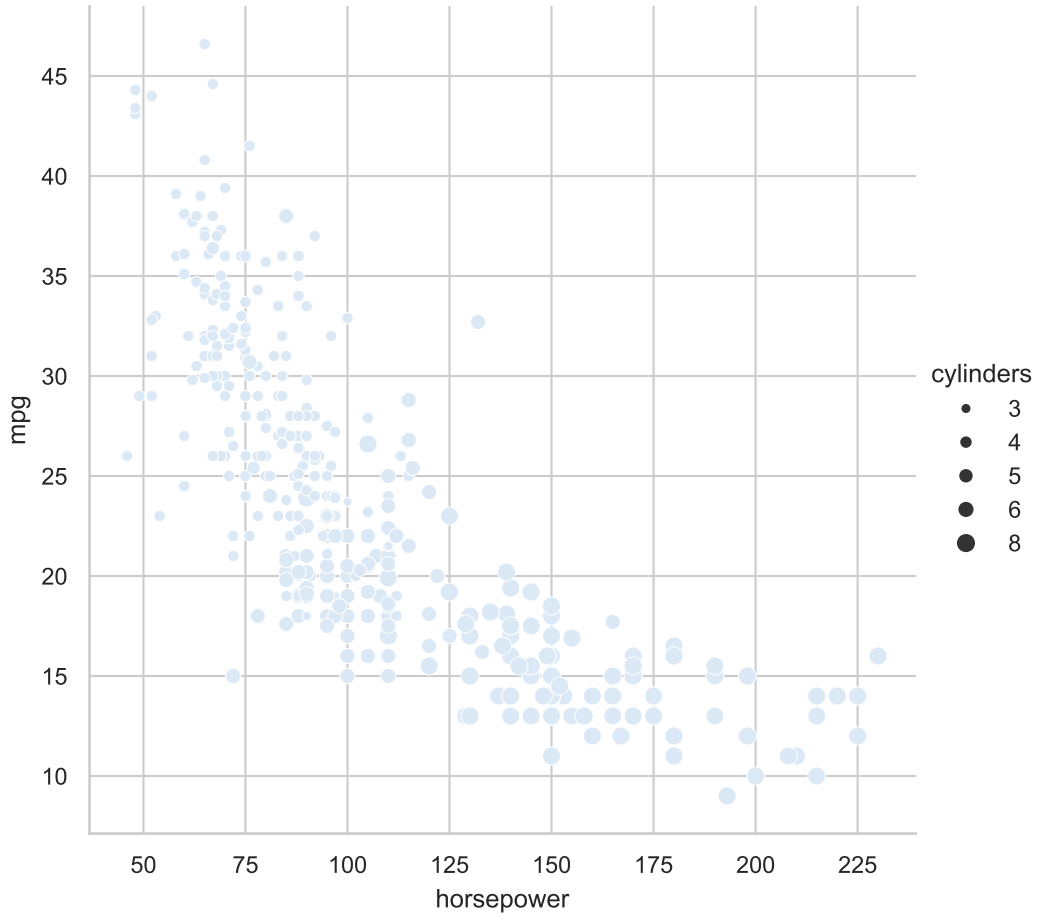
Finally, you can change the scale of your plot by using the `"set context"` function. The scale options from smallest to largest are `"paper"`, `"notebook"`, `"talk"`, and `"poster"`.

Default context: "paper"

The default context is `"paper"`.

```
# Import packages
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
```

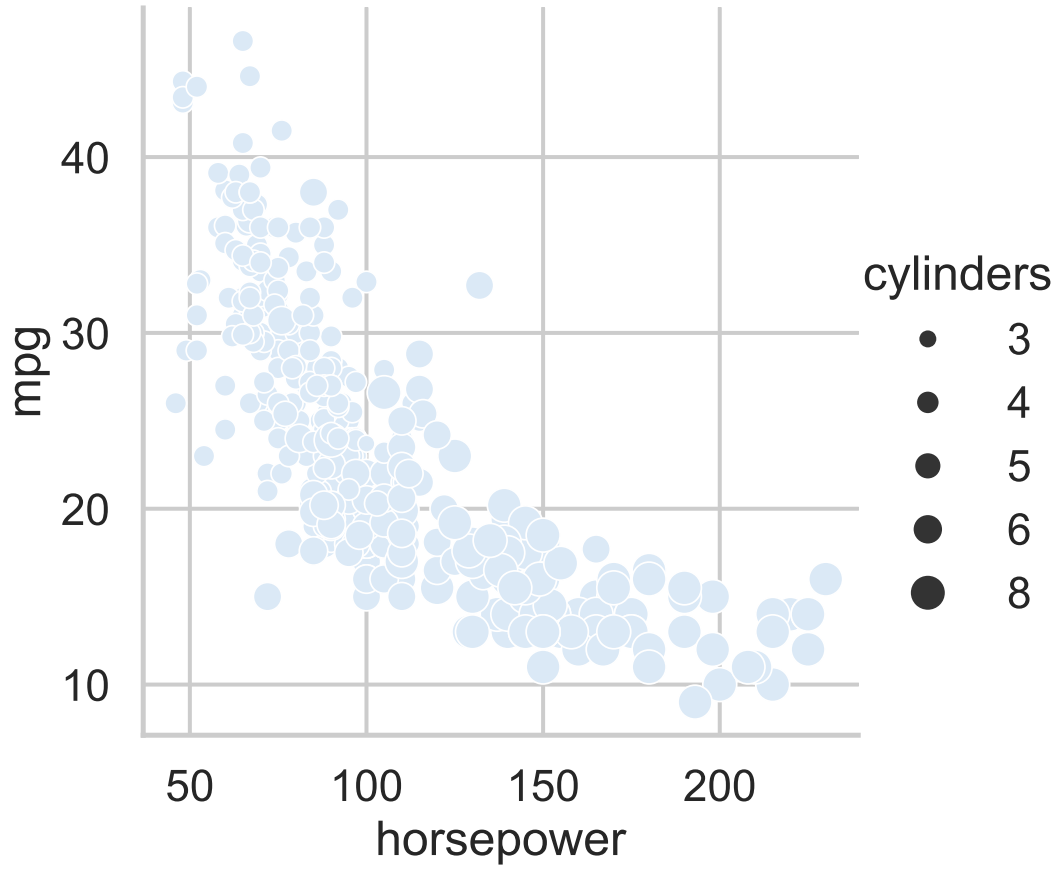
```
# Import dataset
mpg = pd.read_csv("datasets/mpg.csv")

# Set context to "paper" (default)
sns.set_context("paper")

# Create scatter plot of horsepower vs. mpg
sns.relplot(
    x="horsepower",
    y="mpg",
    data=mpg,
    kind="scatter",
    size="cylinders"
)

# Show plot
plt.show()
```



Larger context: "talk"

You'll want to choose a larger scale like `"talk"` for posters or presentations where the audience is further away from the plot.

```
# Import packages
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Import dataset
mpg = pd.read_csv("datasets/mpg.csv")

# Set context to "talk" (larger)
sns.set_context("talk")

# Create scatter plot of horsepower vs. mpg
sns.relplot(
    x="horsepower",
    y="mpg",
    data=mpg,
    kind="scatter",
    size="cylinders"
)

# Show plot
plt.show()
```



Exercise 5

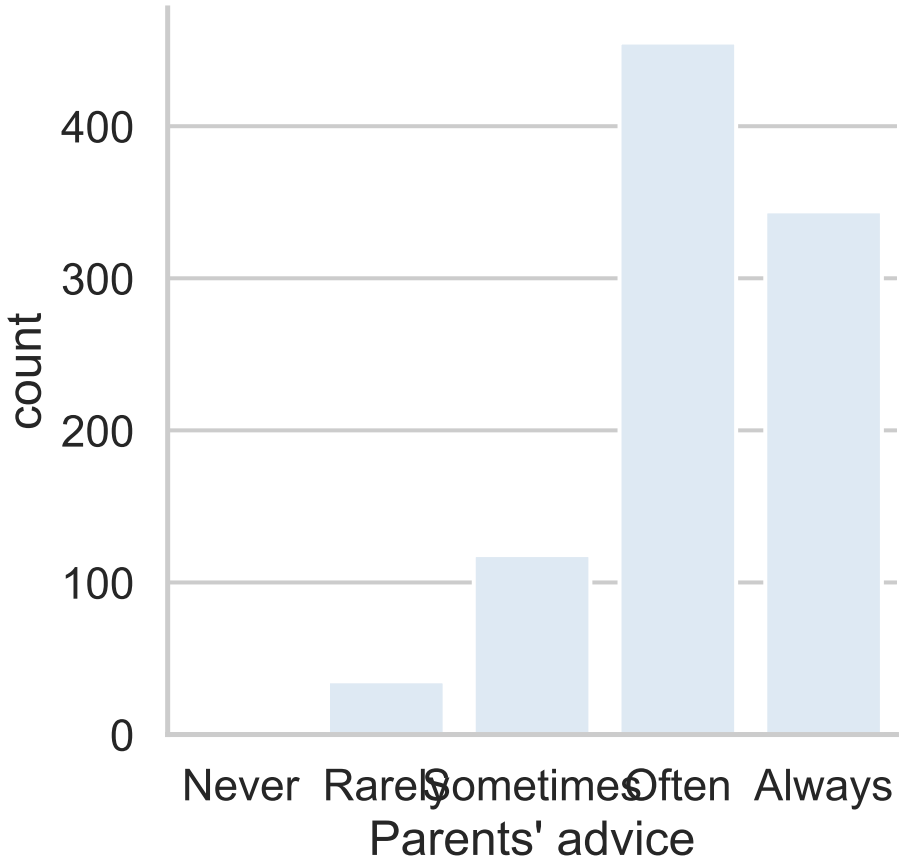
Changing style and palette

---

1. Set the style to `"whitegrid"` to help the audience determine the number of responses in each category: `"Never"`, `"Rarely"`, `"Sometimes"`, `"Often"`, `"Always"`.
2. Create a count plot of survey responses

```
# Import packages
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Import dataset
```

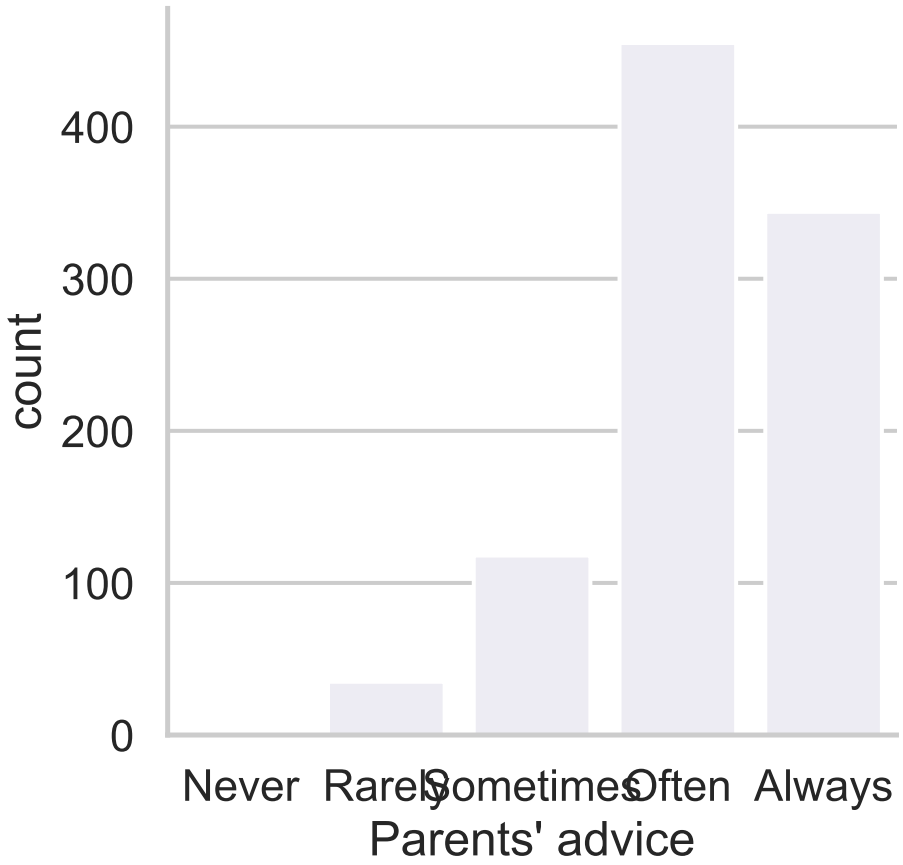
```
survey_data = pd.read_csv('datasets/young-people-survey-responses.csv')

# Set the style to "whitegrid" to help the audience determine the number of responses in each cat
sns.set_style("whitegrid")

# Define the mapping from numerical values to categories
advice_mapping = {
    0: "Never",
    1: "Rarely",
    2: "Sometimes",
    3: "Often",
    4: "Always"
}

# Apply the mapping to the 'Parents' advice' column
survey_data["Parents' advice"] = survey_data["Parents' advice"].map(advice_mapping)

# Create a count plot of survey responses
category_order = ["Never", "Rarely", "Sometimes",
                  "Often", "Always"]

sns.catplot(x="Parents' advice",
            data=survey_data,
            kind="count",
            order=category_order)

# Show plot
plt.show()
```



3. Set the color palette to the sequential palette named `"Purples"`.

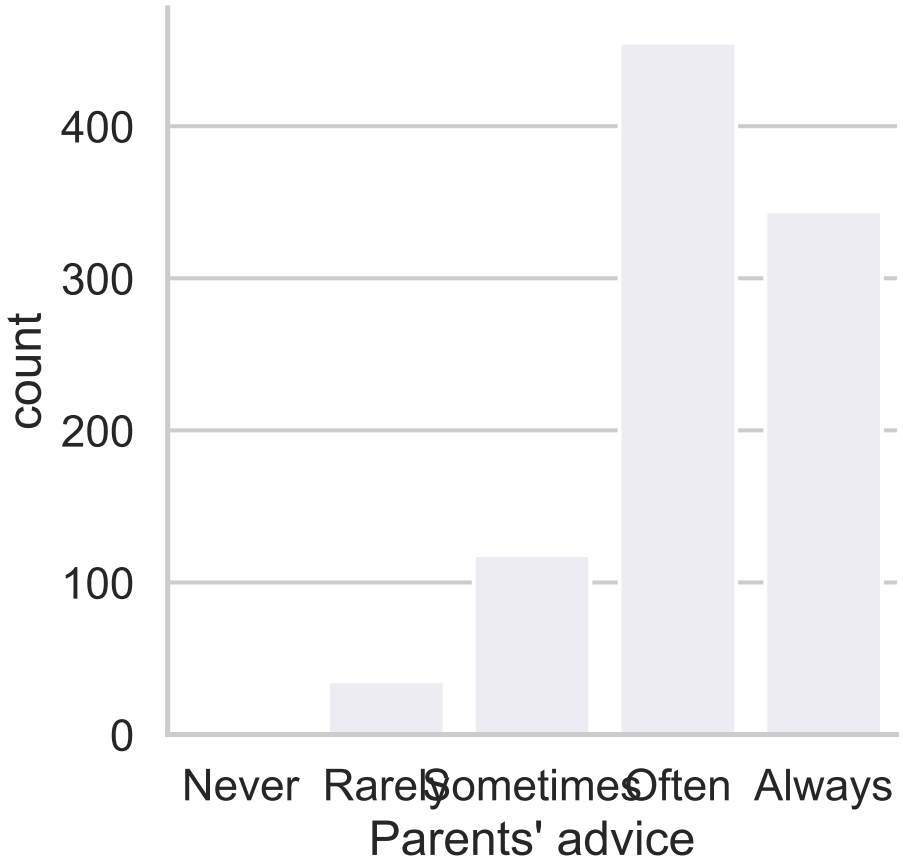
```
# Import packages
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Import dataset
survey_data = pd.read_csv('datasets/young-people-survey-responses.csv')

# Set the color palette to the sequential palette named "Purples"
sns.set_palette("Purples")

# Define the mapping from numerical values to categories
advice_mapping = {
    0: "Never",
    1: "Rarely",
    2: "Sometimes",
    3: "Often",
    4: "Always"
```

```
}

# Apply the mapping to the 'Parents' advice' column
survey_data["Parents' advice"] = survey_data["Parents' advice"].map(advice_mapping)


# Create a count plot of survey responses
category_order = ["Never", "Rarely", "Sometimes",
                  "Often", "Always"]

sns.catplot(x="Parents' advice",
            data=survey_data,
            kind="count",
            order=category_order)

# Show plot
plt.show()
```



4. Set the color palette to the sequential palette named “Purples”.

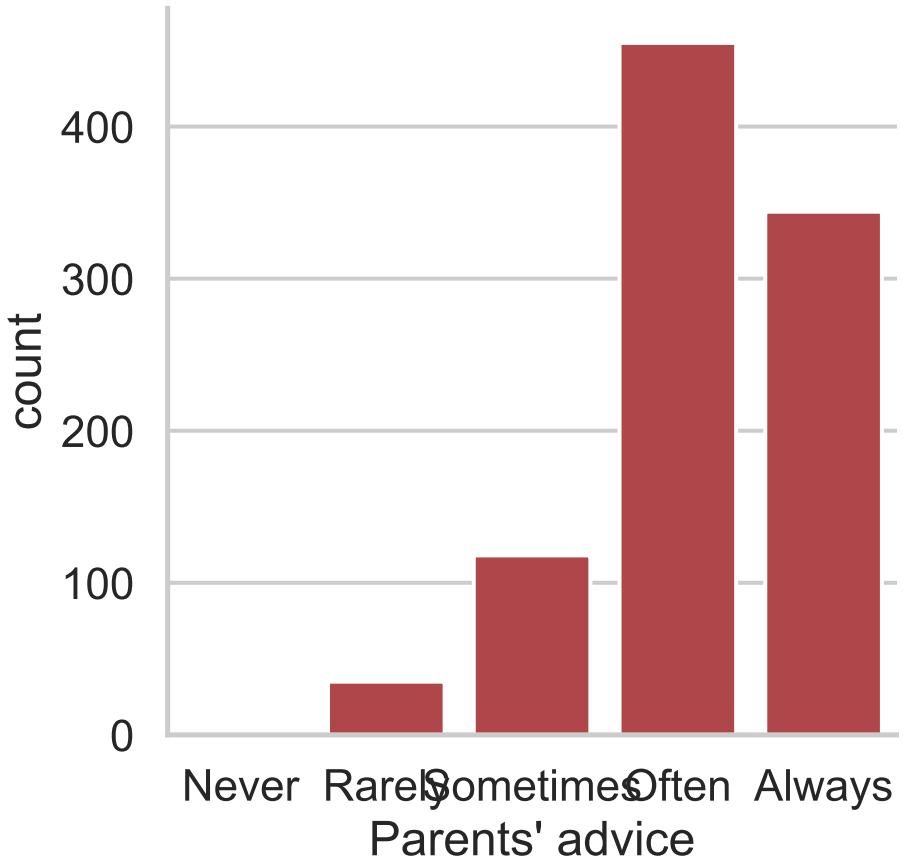
```
# Import packages
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Import dataset
survey_data = pd.read_csv('datasets/young-people-survey-responses.csv')

sns.set_style("whitegrid")
sns.set_palette("Purples")

# Define the mapping from numerical values to categories
advice_mapping = {
    0: "Never",
    1: "Rarely",
    2: "Sometimes",
    3: "Often",
    4: "Always"
}

# Apply the mapping to the 'Parents' advice' column
survey_data["Parents' advice"] = survey_data["Parents' advice"].map(advice_mapping)

# Create a count plot of survey responses
category_order = ["Never", "Rarely", "Sometimes",
                  "Often", "Always"]

sns.catplot(x="Parents' advice",
            data=survey_data,
            kind="count",
            order=category_order)

# Show plot
plt.show()
```



5. Change the color palette to the diverging palette named `"RdBu"`.

```
# Import packages
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Import dataset
survey_data = pd.read_csv('datasets/young-people-survey-responses.csv')

sns.set_style("whitegrid")
sns.set_palette("RdBu")

# Define the mapping from numerical values to categories
advice_mapping = {
    0: "Never",
    1: "Rarely",
    2: "Sometimes",
    3: "Often",
    4: "Always"
```

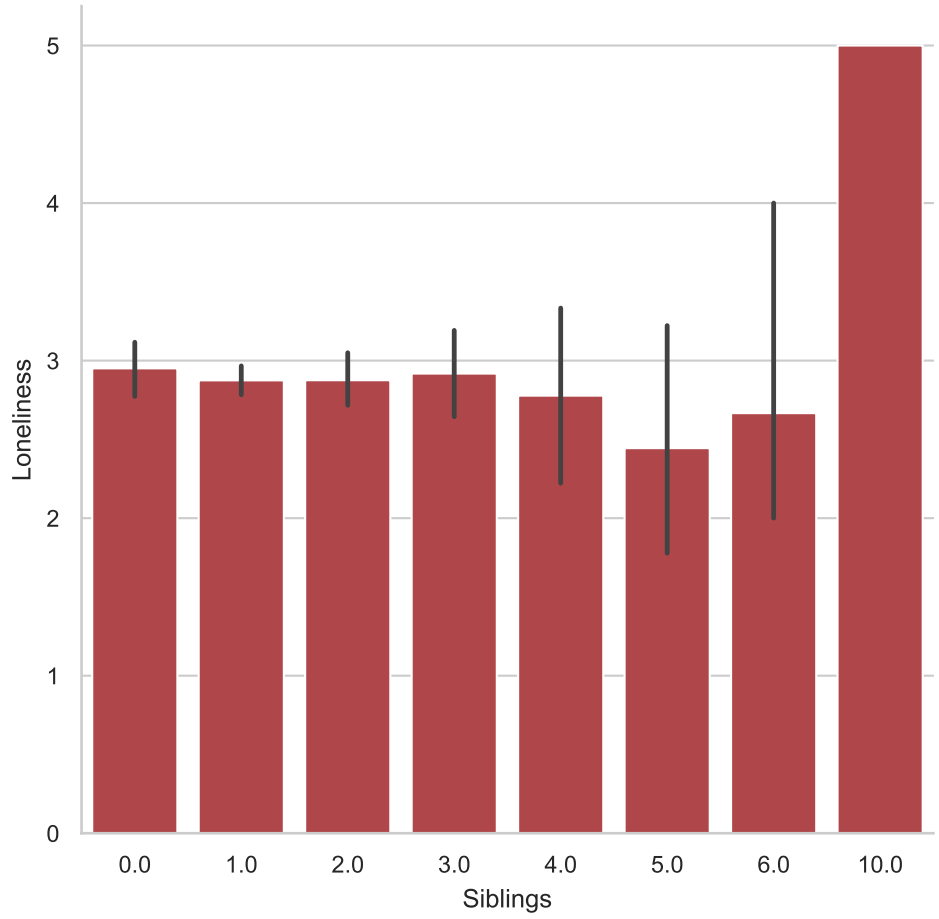
```
}

# Apply the mapping to the 'Parents' advice' column
survey_data["Parents' advice"] = survey_data["Parents' advice"].map(advice_mapping)

# Create a count plot of survey responses
category_order = ["Never", "Rarely", "Sometimes",
                  "Often", "Always"]

sns.catplot(x="Parents' advice",
            data=survey_data,
            kind="count",
            order=category_order)

# Show plot
plt.show()
```



6. Set the scale (`"context"`) to `"paper"`, which is the smallest of the scale options.

```
sns.set_context("paper")
```

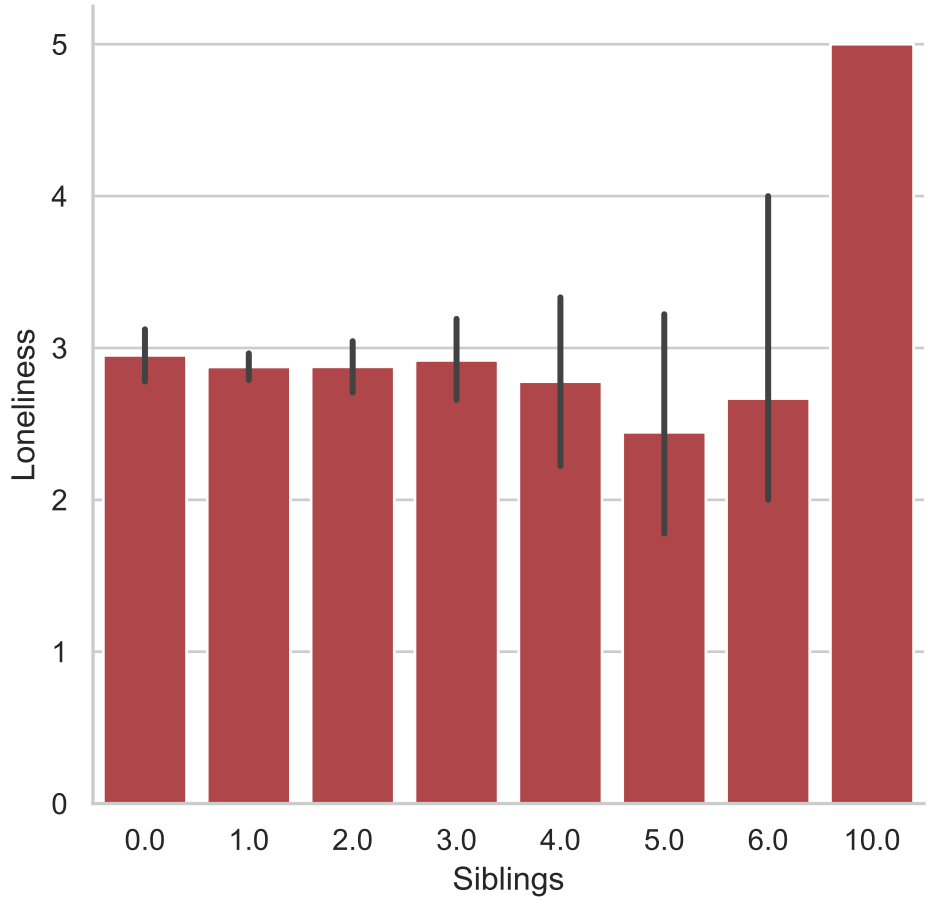
```
# Create bar plot
sns.catplot(x="Siblings", y="Loneliness",
            data=survey_data, kind="bar")

# Show plot
plt.show()
```



7. Change the context to `"notebook"` to increase the scale.

```
# Import packages
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Import dataset
survey_data = pd.read_csv('datasets/young-people-survey-responses.csv')
```

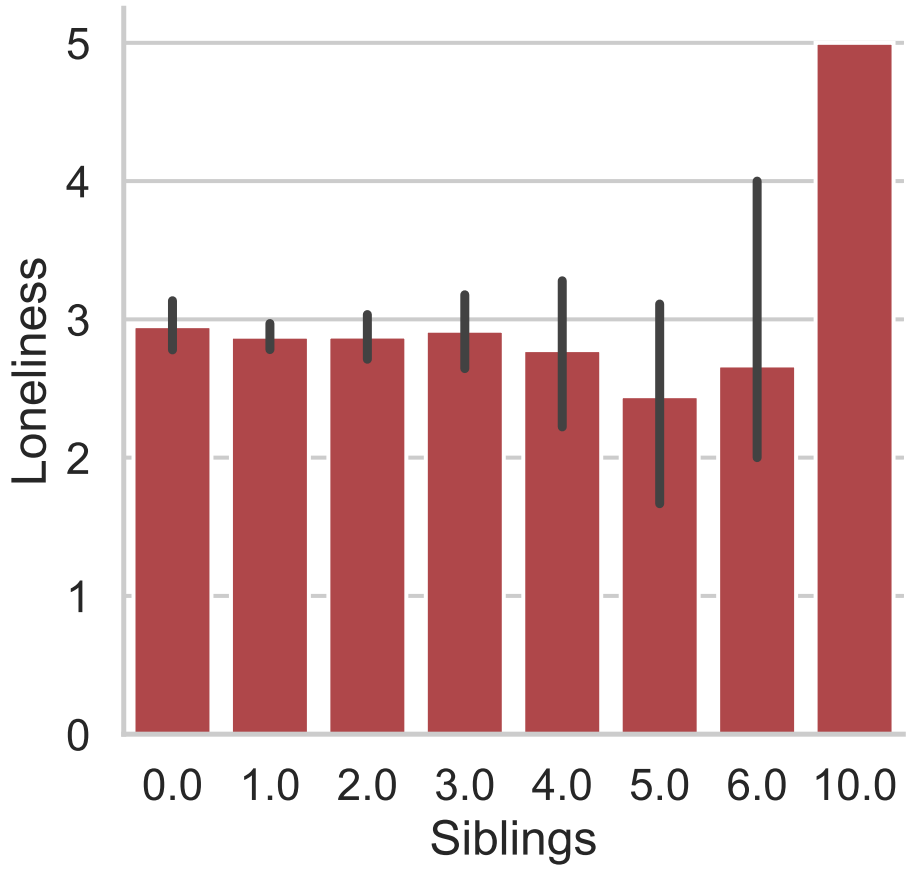
```
sns.set_context("notebook")

# Create bar plot
sns.catplot(x="Siblings", y="Loneliness",
            data=survey_data, kind="bar")

# Show plot
plt.show()
```



8. Change the context to `"talk"` to increase the scale.

```
# Import packages
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Import dataset
survey_data = pd.read_csv('datasets/young-people-survey-responses.csv')
```

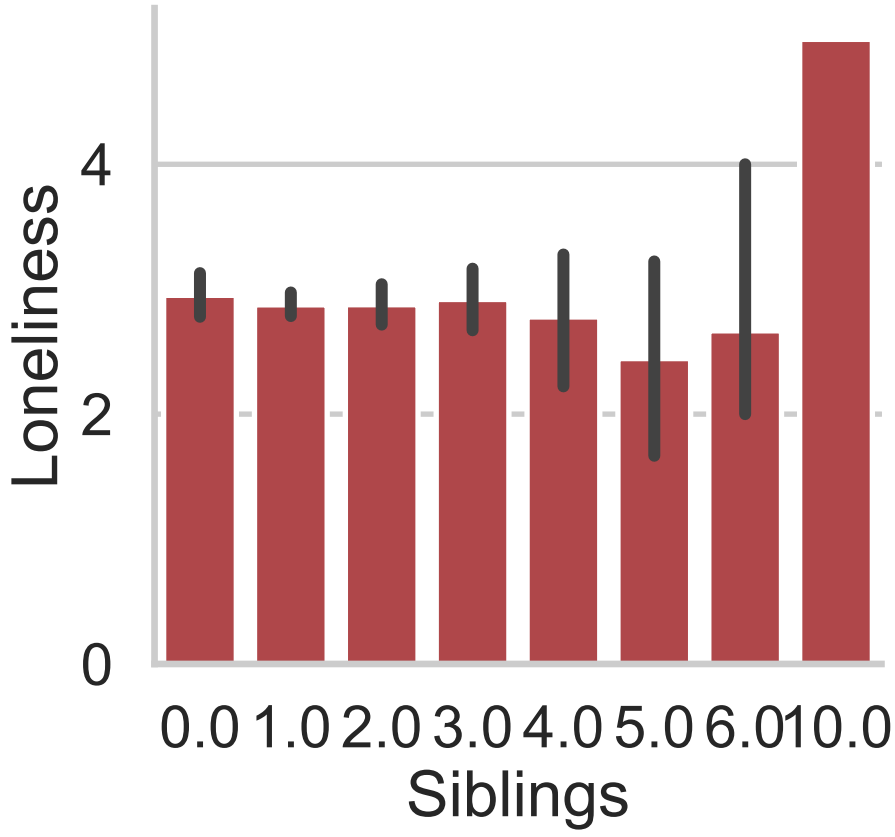
```
sns.set_context("talk")

# Create bar plot
sns.catplot(x="Siblings", y="Loneliness",
            data=survey_data, kind="bar")

# Show plot
plt.show()
```



9. Change the context to `"poster"`, which is the largest scale available.

```
# Import packages
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Import dataset
survey_data = pd.read_csv('datasets/young-people-survey-responses.csv')

sns.set_context("poster")
```

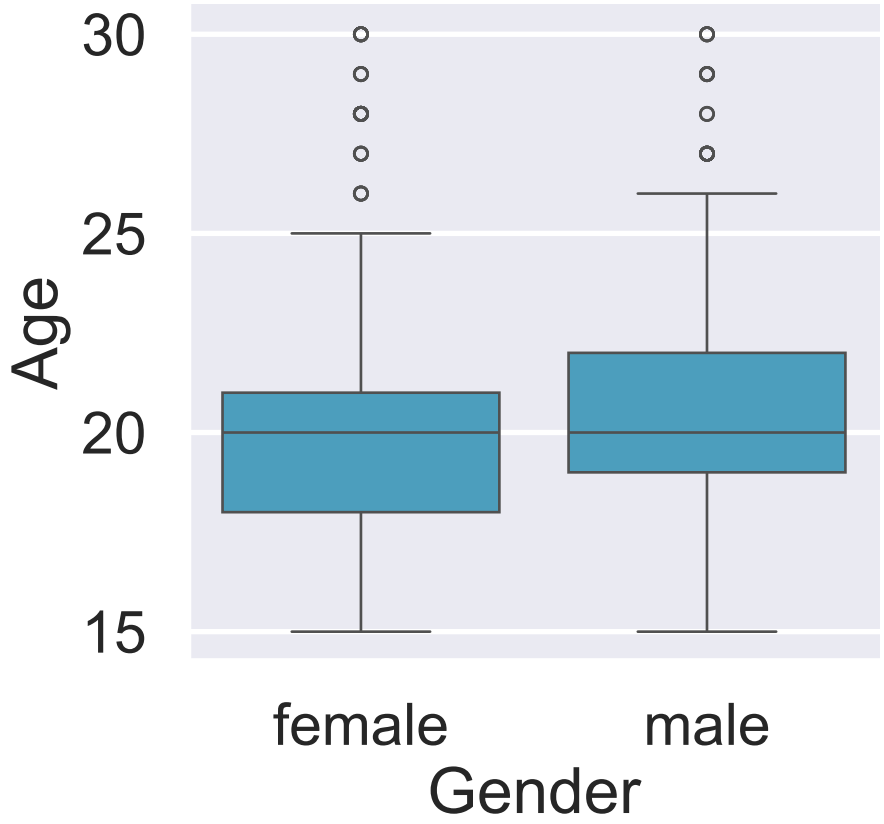
```
# Create bar plot
sns.catplot(x="Siblings", y="Loneliness",
            data=survey_data, kind="bar")

# Show plot
plt.show()
```



10. Set the style to `"darkgrid"` and custom color palette with the hex color codes `"#39A7D0"` and `"#36ADA4"`.

```
# Import packages
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Import dataset
survey_data = pd.read_csv('datasets/young-people-survey-responses.csv')

sns.set_style("darkgrid")
```

```
# Set a custom color palette
sns.set_palette(["#39A7D0", "#36ADA4"])

# Create the box plot of age distribution by gender
sns.catplot(x="Gender", y="Age",
            data=survey_data, kind="box")

# Show plot
plt.show()
```

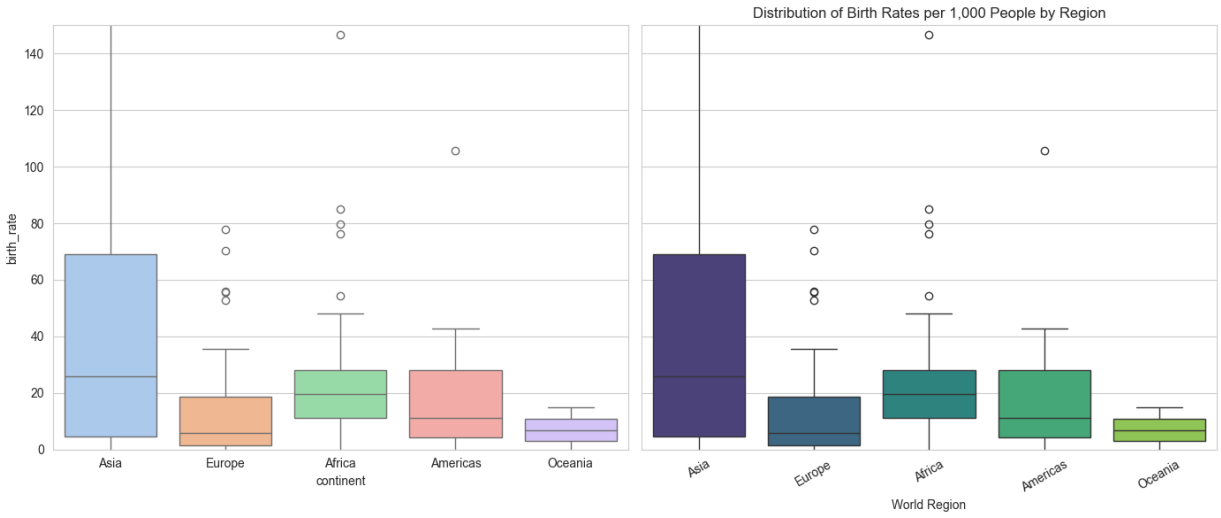


Chapter 6.1: Adding titles and labels: Part 1

Welcome! In the next two lessons, we'll go over one of the most important parts of any data visualization: plot titles and axis labels.

Creating informative visualizations

We create data visualizations to communicate information, and we can't do that effectively without a clear title and informative axis labels. To see this, let's compare two versions of the same visualization. On the left, we see box plots showing the distribution of birth rates for countries in each of 11 regions. On the right, we see the same visualization with three key modifications to make it easier to understand. A title is added, which immediately orients the audience to what they're looking at. The axis labels are more informative, making it clearer that birth rate is measured per one thousand people and birth rates are measured per country in each region. Finally, the x-axis tick labels are rotated to make it clear what each region is called. Let's learn how to make these changes.



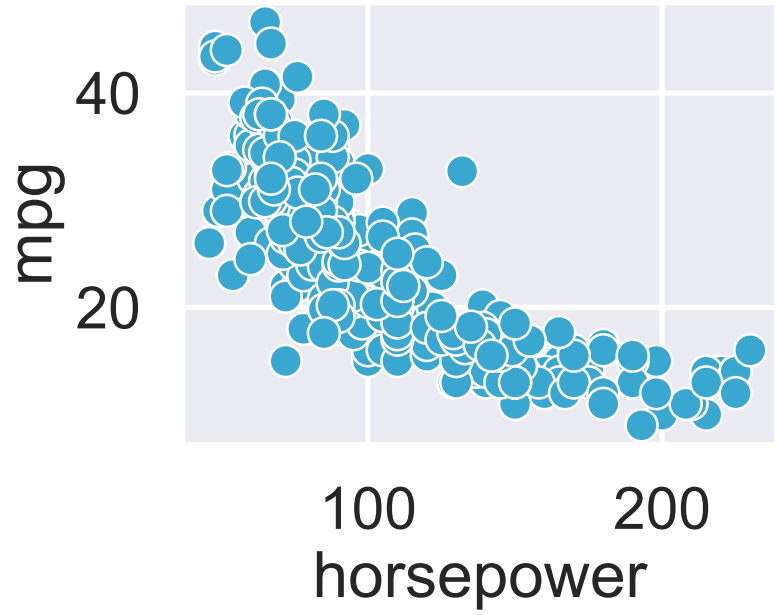
FacetGrid vs. AxesSubplot objects

Before we go into the details of adding a title, we need to understand the underlying mechanism in Seaborn. Seaborn's plot functions create two different types of objects: `FacetGrids` and `AxesSubplots`. To figure out which type of object you're working with, first assign the plot output to a variable. In the documentation, the variable is often named `"g"`, so we'll do that here as well. Write `"type" "g"` to return the object type. This scatter plot is an `AxesSubplot`.

```
# Import packages
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
```

```
# Import dataset
mpg = pd.read_csv("datasets/mpg.csv")

g = sns.scatterplot(x="horsepower", y="mpg", data=mpg)
type(g)
```

```
matplotlib.axes._axes.Axes
```



An Empty FacetGrid

A `FacetGrid` consists of one or more `AxesSubplots`, which is how it supports subplots.

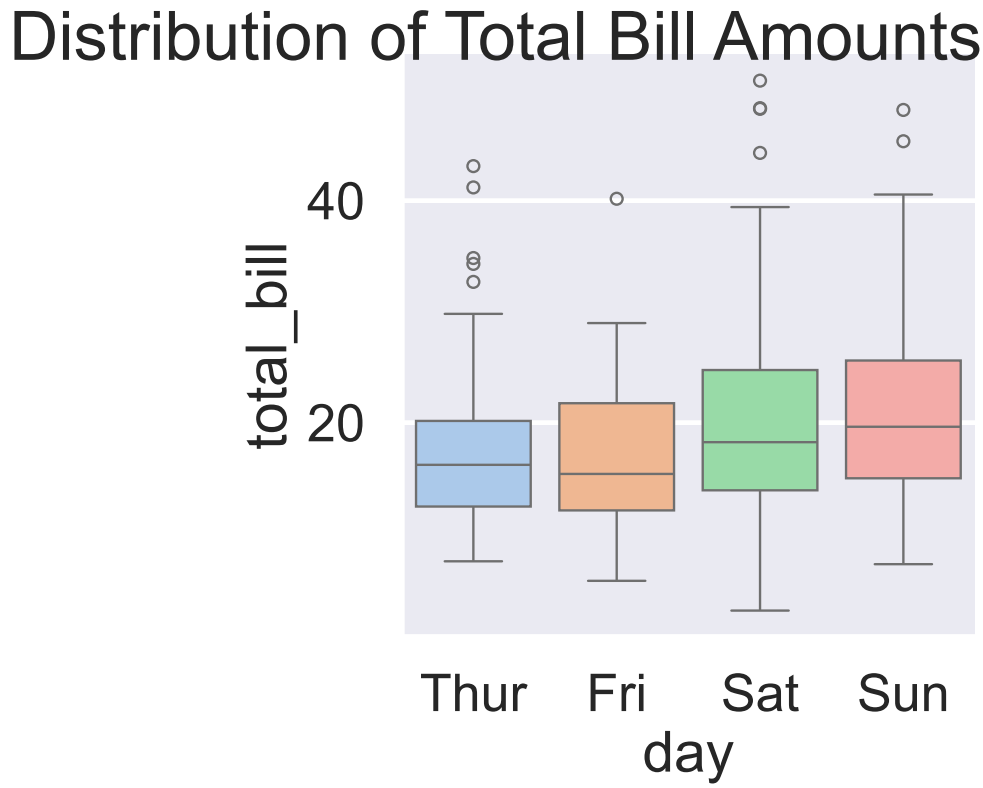
FacetGrid vs. AxesSubplot objects

Recall that `"relplot()"` and `"catplot()"` both support making subplots. This means that they are creating `FacetGrid` objects. In contrast, single-type plot functions like `"scatterplot()"` and `"countplot()"` return a single AxesSubplot object.

Adding a title to FacetGrid

Let's return to our messy plot from the beginning. Recall that `"catplot()"` enables subplots, so it returns a FacetGrid object. To add a title to a `FacetGrid` object, first assign the plot to the variable `"g"`. After you assign the plot to `"g"`, you can set the title using `"g.fig.suptitle"`. This tells Seaborn you want to set a title for the figure as a whole.

```
import seaborn as sns
import matplotlib.pyplot as plt

# Load Seaborn's 'tips' dataset
tips = sns.load_dataset("tips")

# Create a box plot using catplot and assign to variable 'g'
g = sns.catplot(
    x="day",
    y="total_bill",
    data=tips,
    kind="box",
    palette="pastel"
)

# Add a title to the entire figure
g.fig.suptitle("Distribution of Total Bill Amounts by Day")

# Show the plot
plt.show()
```



Adjusting height of title in FacetGrid

Note that by default, the figure title might be a little low. To adjust the height of the title, you can use the "y" parameter. The default value is 1, so setting it to 1.03 will make it a little higher than the default.

```
import seaborn as sns
import matplotlib.pyplot as plt

# Load Seaborn's 'tips' dataset
tips = sns.load_dataset("tips")

# Create a box plot using catplot and assign to variable 'g'
g = sns.catplot(
    x="day",
    y="total_bill",
    data=tips,
    kind="box",
```

```
    palette="pastel"
)

# Add a title to the entire figure
g.fig.suptitle("Distribution of Total Bill Amounts by Day", y=1.03)

# Show the plot
plt.show()
```

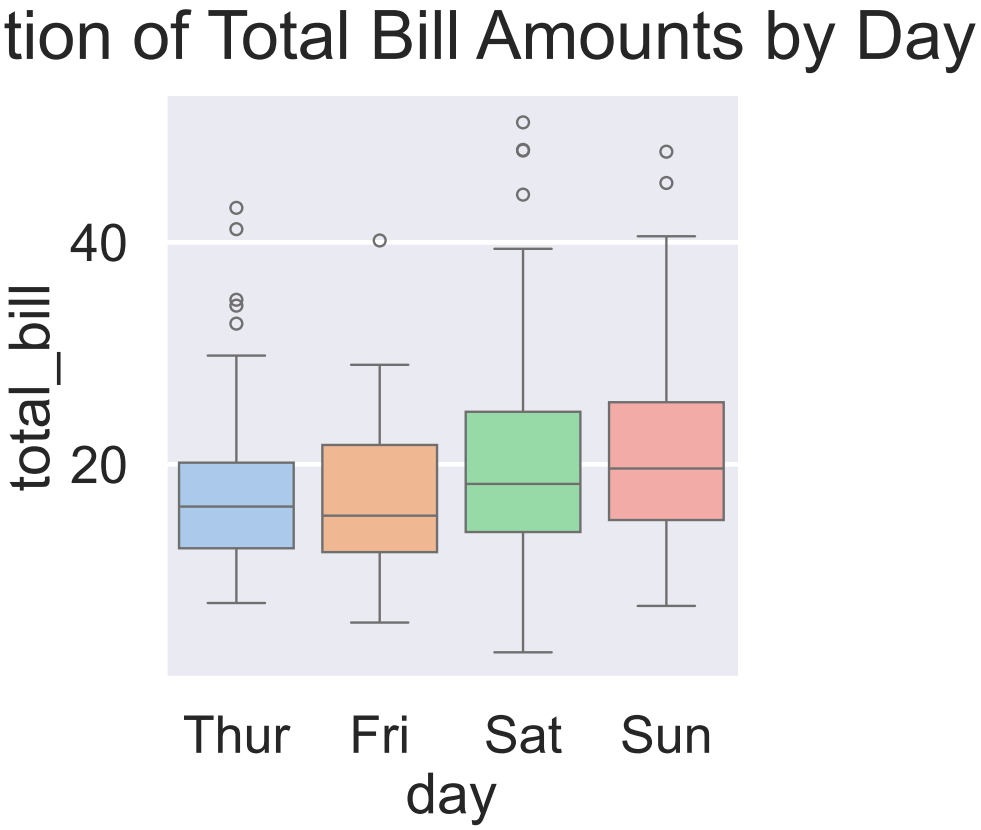


Figure 46: Distribution of Total Bill Amounts by Day.

Exercise 6

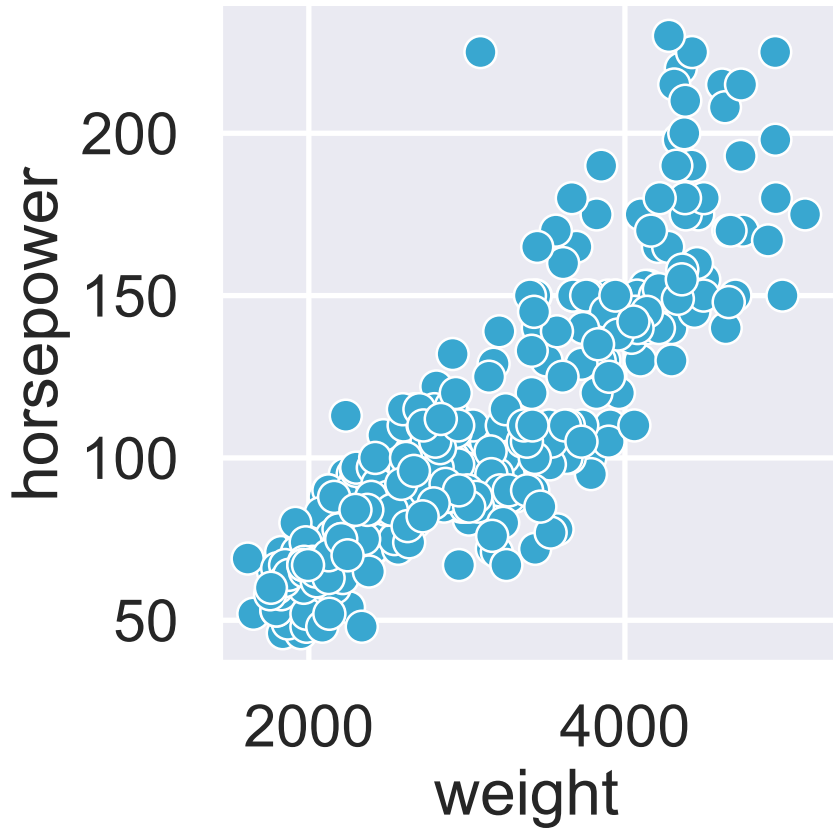
FacetGrids vs. AxesSubplots

---

1. Create scatter plot weight against horsepower of a vehicle.
2. Identify plot type and print type.

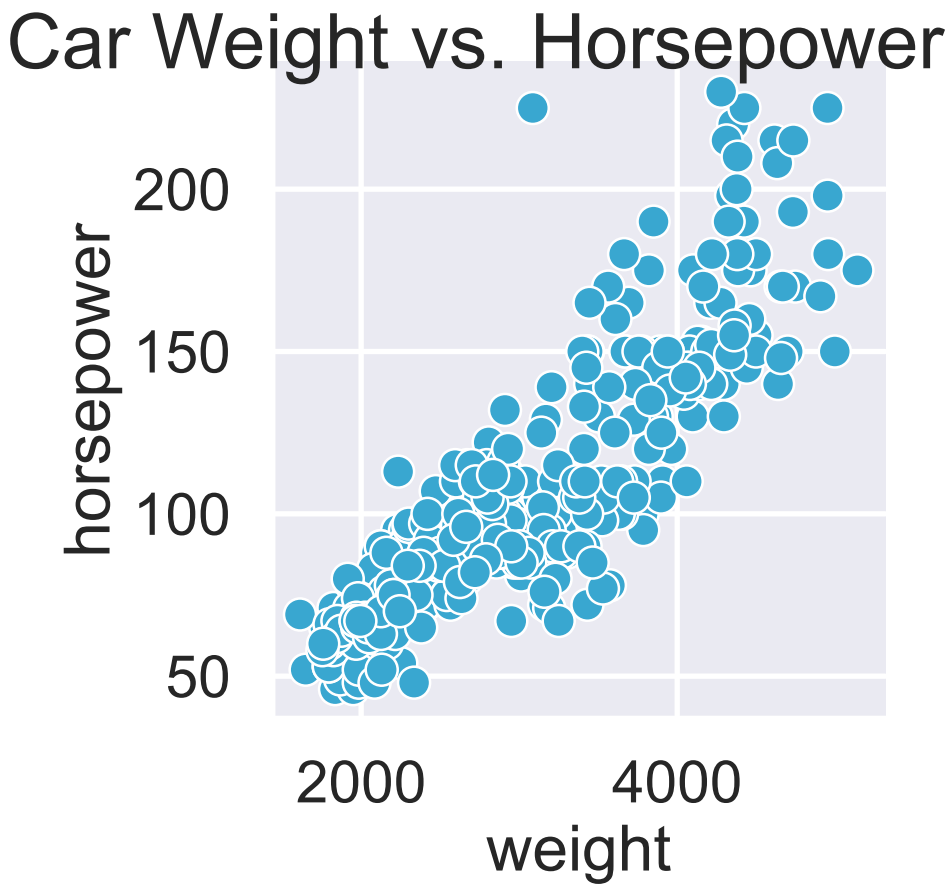
```
# Import packages
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Import dataset
mpg = pd.read_csv("datasets/mpg.csv")

# Create scatter plot
g = sns.relplot(x="weight",
                y="horsepower",
                data=mpg,
                kind="scatter")

# Identify plot type
type_of_g = type(g)

# Print type
print(f'The datatype is: {type_of_g}')
```

```
The datatype is: <class 'seaborn.axisgrid.FacetGrid'>
```



3. Create scatter plot of the above scatter plot and add title.

```
# Import packages
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Import dataset
mpg = pd.read_csv("datasets/mpg.csv")

g = sns.relplot(x="weight",
                y="horsepower",
                data=mpg,
                kind="scatter")

# Add a title "Car Weight vs. Horsepower"
g.fig.suptitle("Car Weight vs. Horsepower")

# Show plot
plt.show()
```



Chapter 6.2: Adding titles and labels: Part 2

Hello! In this lesson, we'll continue learning how to customize plot titles and axis labels.

Adding a title to AxesSubplot

In the last lesson, Section , we learned how to add a title to a `FacetGrid` object using `"g.fig.suptitle"`. To add a title to an AxesSubplot object like that from the `"box plot"` function, assign the plot to a variable and use `"g.set_title"`. You can also use the `"y"` parameter here to adjust the height of the title, Figure 46.

Titles for subplots

Now let's look at what happens if the figure has subplots. Let's say we've divided countries into two groups - group one and group two - and we've set "col" equal to `"Group"` to create a subplot for each group. Since `g` is a `FacetGrid` object, using `"g.fig.suptitle"` will add a title to the figure as a whole. To alter the subplot titles, use `"g.set_titles"` to set the titles for each `AxesSubplot`. If you want to use the variable name in the title, you can use "col name" in braces to reference the column value. Here, we've created subplot titles that display as `"this is group 2"` and `"this is group 1"`.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Set style
sns.set_style("whitegrid")

# Create DataFrame - 3 countries per group
df = pd.DataFrame({
    "Country": [
        "Nigeria", "Kenya", "Ghana",        # Group 1
        "France", "Germany", "Italy"        # Group 2
    ],
    "GDP_per_capita": np.random.randint(2000, 60000, 6),
    "Life_Expectancy": np.random.uniform(50, 85, 6),
    "Group": ["Group 1"] * 3 + ["Group 2"] * 3
})

# Create FacetGrid scatter plots
g = sns.relplot(
    data=df,
    x="GDP_per_capita",
    y="Life_Expectancy",
    col="Group",
    kind="scatter",
    height=4,
    aspect=1,
    hue="Country",
    palette="tab10"
)

# Add a main title for the entire figure
```

```
g.fig.suptitle("Life Expectancy vs GDP per Capita by Group", fontsize=14, y=1.05)

# Add subplot titles dynamically
g.set_titles("This is {col_name}")

# Rotate x-axis labels for clarity (optional)
plt.xticks(rotation=30)

# Show plot
plt.show()
```

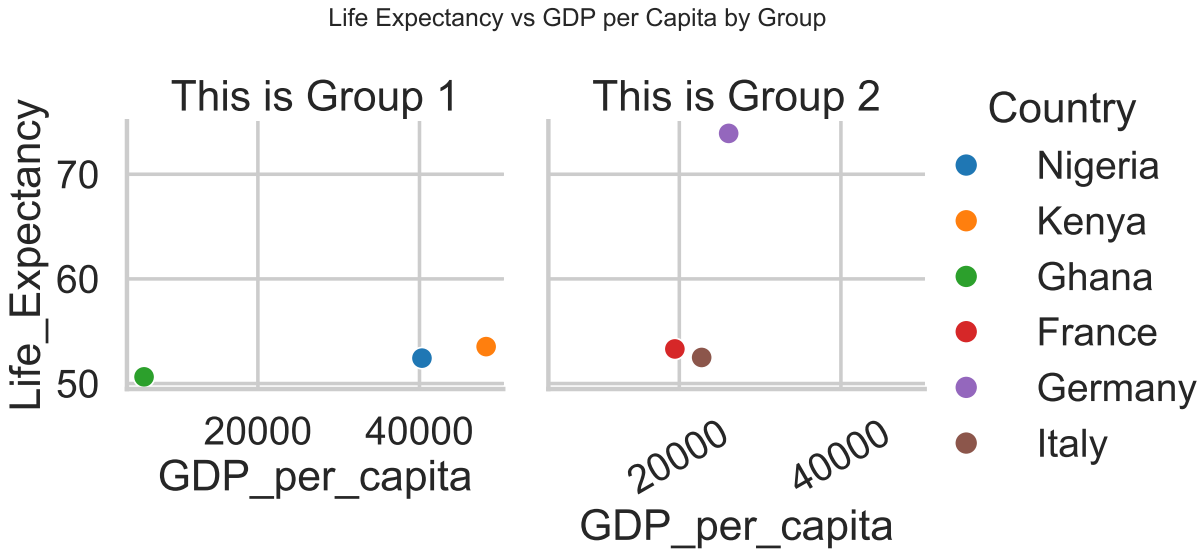


Figure 47: Custom Titles for Subplots Using FacetGrid.

Adding axis labels

To add axis labels, assign the plot to a variable and then call the `"set"` function. Set the parameters `"x label"` and `"y label"` to set the desired x-axis and y-axis labels, respectively. This works with both FacetGrid and AxesSubplot objects.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Apply Seaborn style
```

```
sns.set_style("whitegrid")

# Create DataFrame - 3 countries per group
df = pd.DataFrame({
    "Country": [
        "Nigeria", "Kenya", "Ghana",        # Group 1
        "France", "Germany", "Italy"        # Group 2
    ],
    "GDP_per_capita": np.random.randint(2000, 60000, 6),
    "Life_Expectancy": np.random.uniform(50, 85, 6),
    "Group": ["Group 1"] * 3 + ["Group 2"] * 3
})

# Create FacetGrid scatter plots
g = sns.relplot(
    data=df,
    x="GDP_per_capita",
    y="Life_Expectancy",
    col="Group",
    kind="scatter",
    height=4,
    aspect=1,
    hue="Country",
    palette="tab10"
)

# Add a main title for the entire figure
g.fig.suptitle("Life Expectancy vs GDP per Capita by Group", fontsize=14, y=1.05)

# Add subplot titles dynamically
g.set_titles("This is {col_name}")

# Add axis labels using .set()
g.set(
    xlabel="GDP per Capita (USD)",
    ylabel="Life Expectancy (Years)"
)

# Rotate x-axis labels for clarity (optional)
plt.xticks(rotation=30)

# Show plot
plt.show()
```

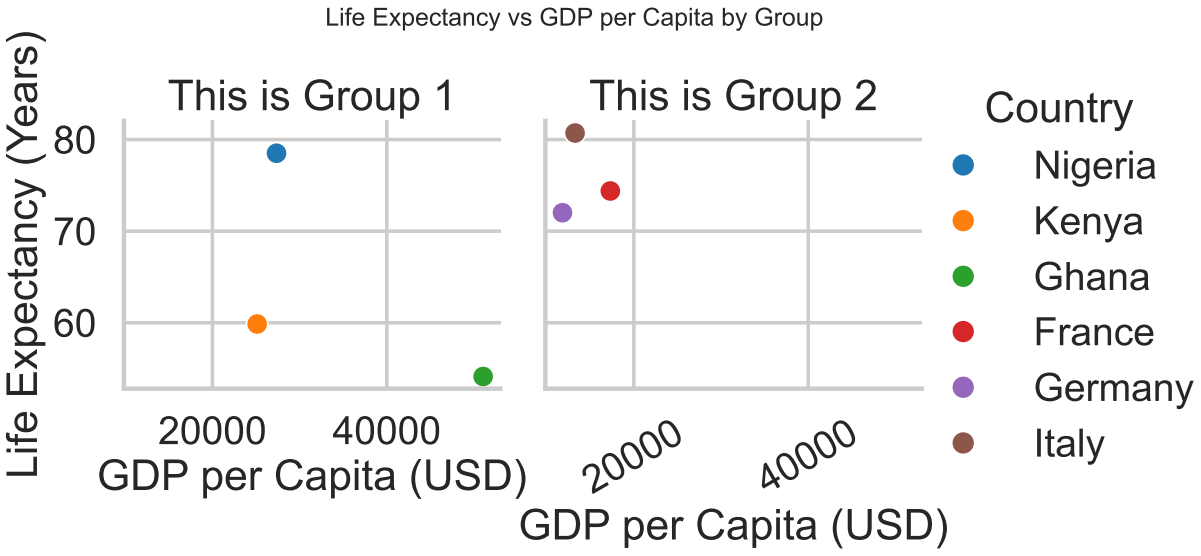


Figure 48: Adding Axis Labels and Subplot Titles Using FacetGrid.

Rotating x-axis tick labels

Sometimes, like in the example we've seen in this lesson, your tick labels may overlap, making it hard to interpret the plot. One way to address this is by rotating the tick labels. To do this, we don't call a function on the plot object itself. Instead, after we create the plot, we call the matplotlib function `"plt.xticks"` and set `"rotation"` equal to 90 degrees. This works with both `FacetGrid` and `AxesSubplot` objects.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Apply Seaborn style
sns.set_style("whitegrid")

# Create dataset
df = pd.DataFrame({
    "Country": [
        "Nigeria", "Kenya", "Ghana", "Ethiopia", "South Africa",
        "Uganda", "Tanzania", "Cameroon", "Zambia", "Rwanda",
        "France", "Germany", "Italy", "Spain", "Sweden",
        "Norway", "Poland", "Greece", "Portugal", "Netherlands"
```

```
    ],
    "GDP_per_capita": np.random.randint(2000, 60000, 20),
    "Life_Expectancy": np.random.uniform(50, 85, 20),
    "Group": ["Group 1"] * 10 + ["Group 2"] * 10
})

# Create FacetGrid scatter plots
g = sns.relplot(
    data=df,
    x="GDP_per_capita",
    y="Life_Expectancy",
    col="Group",
    kind="scatter",
    height=4,
    aspect=1,
    hue="Country"
)

# Add a main title for the entire figure
g.fig.suptitle("Life Expectancy vs GDP per Capita by Group", fontsize=14, y=1.05)

# Add subplot titles dynamically
g.set_titles("This is {col_name}")

# Add axis labels using .set()
g.set(
    xlabel="GDP per Capita (USD)",
    ylabel="Life Expectancy (Years)"
)

# Rotate tick labels for better readability
plt.xticks(rotation=90)

# Adjust layout for spacing
plt.tight_layout()
plt.show()
```

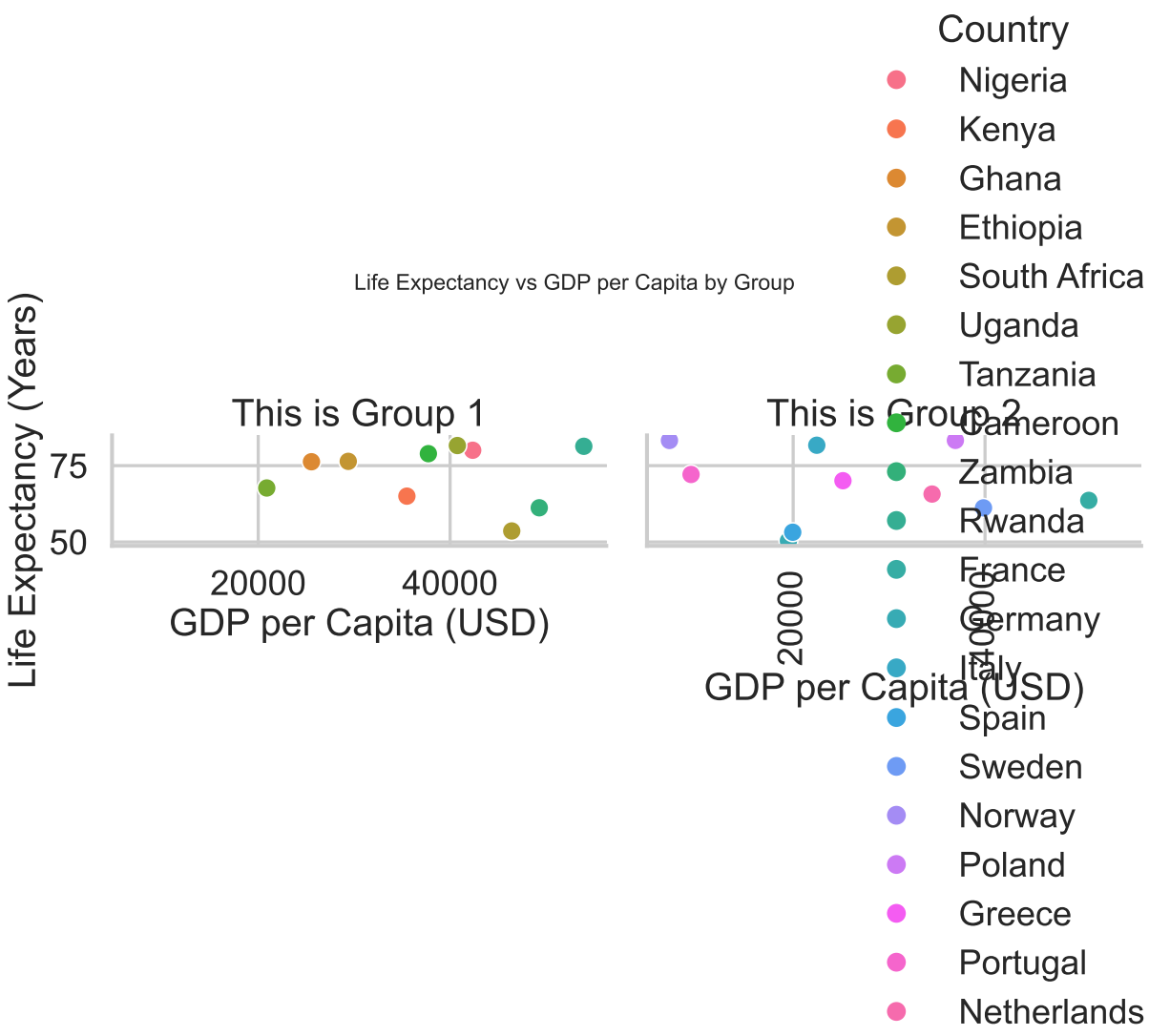


Figure 49: Rotating Tick Labels in FacetGrid Subplots.

Exercise 6

Adding a title and axis labels

---

1. Create line plot with model year in x axis and mean of mpg of countries of origin on y axis.
2. Add a title `"Average MPG Over Time"`.

```
# Import packages
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Load dataset
mpg = pd.read_csv("datasets/mpg.csv")

# Compute average MPG by model year and origin
mpg_mean = (
    mpg.groupby(['origin', 'model_year'])['mpg']
       .mean()
       .reset_index()
)

# Create line plot
g = sns.lineplot(x="model_year", y="mpg",
                 data=mpg_mean,
                 hue="origin")

# Add a title "Average MPG Over Time"
g.set_title("Average MPG Over Time")

# Show plot
plt.show()
```

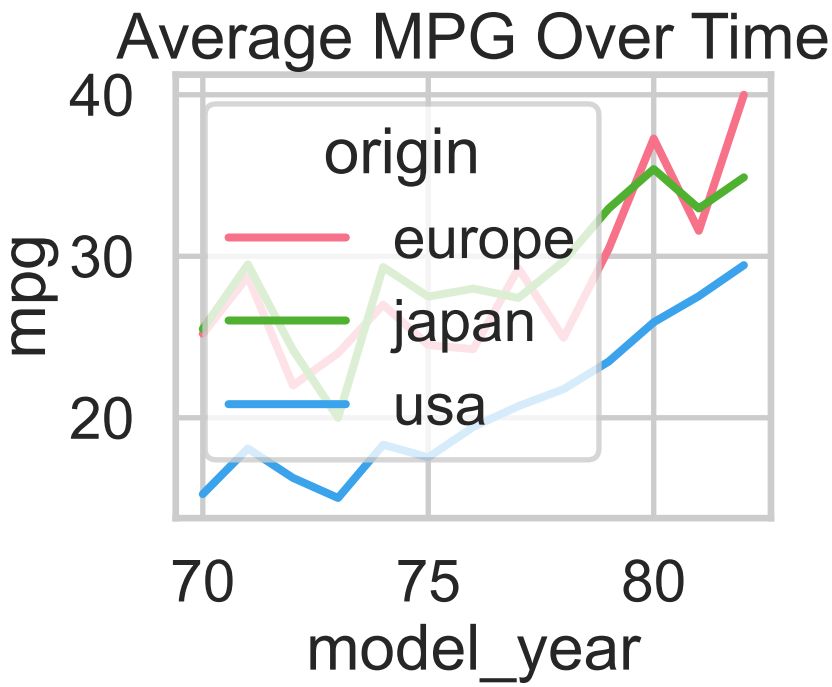
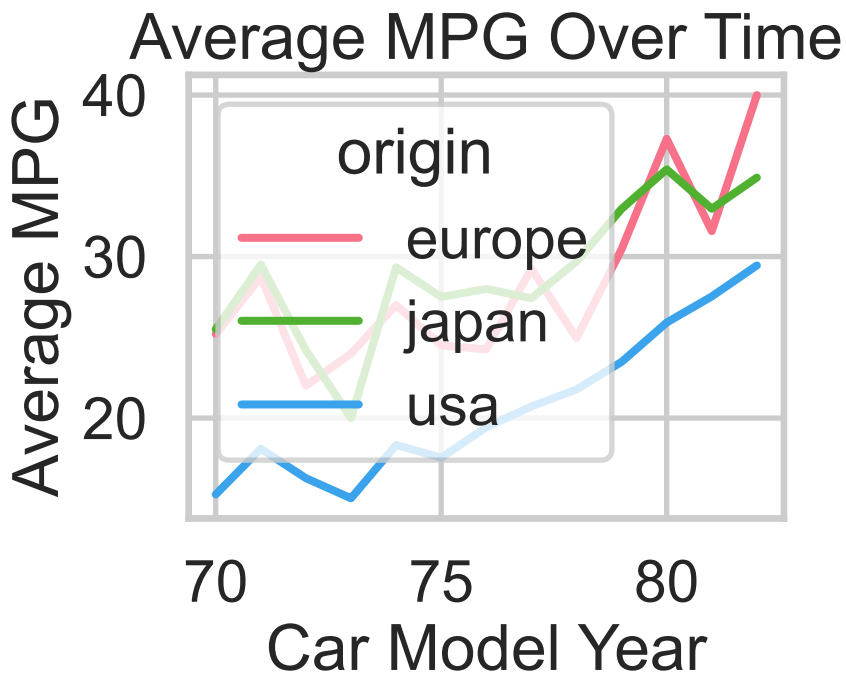


Figure 50: Average Miles per Gallon (MPG) Over Time by Origin.

3. Create a line plot between `model_year` and mean of `mpg`. Add a title `"Average MPG Over Time"`; Add x-axis and y-axis labels

```
# Import packages
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Load dataset
mpg = pd.read_csv("datasets/mpg.csv")

# Compute average MPG by model year and origin
mpg_mean = (
    mpg.groupby(['origin', 'model_year'])['mpg']
       .mean()
       .reset_index()
)

# Create line plot
g = sns.lineplot(x="model_year", y="mpg",
                 data=mpg_mean,
                 hue="origin")
```

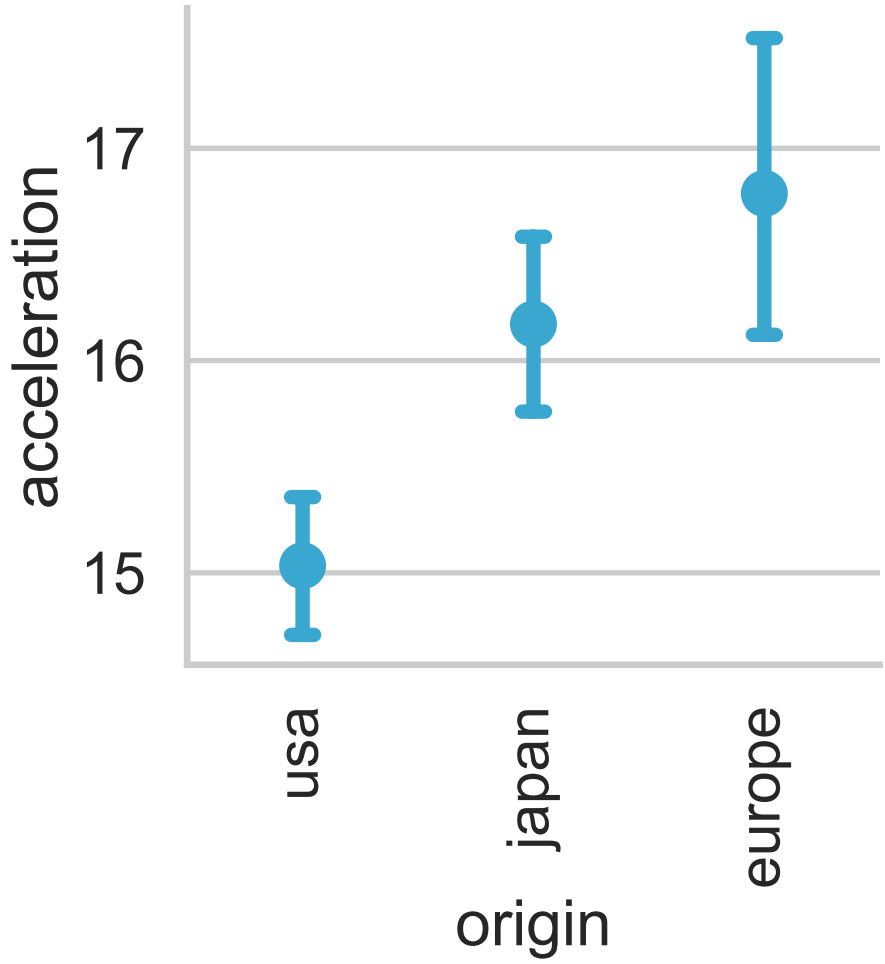
```
# Add a title "Average MPG Over Time"
g.set_title("Average MPG Over Time")

# Add x-axis and y-axis labels
g.set(xlabel="Car Model Year", ylabel="Average MPG" )

# Show plot
plt.show()
```



4. Create point plot, with Countries of origin on the x-axis and acceleration on the y-axis. Rotate x-tick labels to 90 degree.

```
# Import packages
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Import dataset
mpg = pd.read_csv("datasets/mpg.csv")

# Create point plot
sns.catplot(x="origin",
            y="acceleration",
            data=mpg,
            kind="point",
```

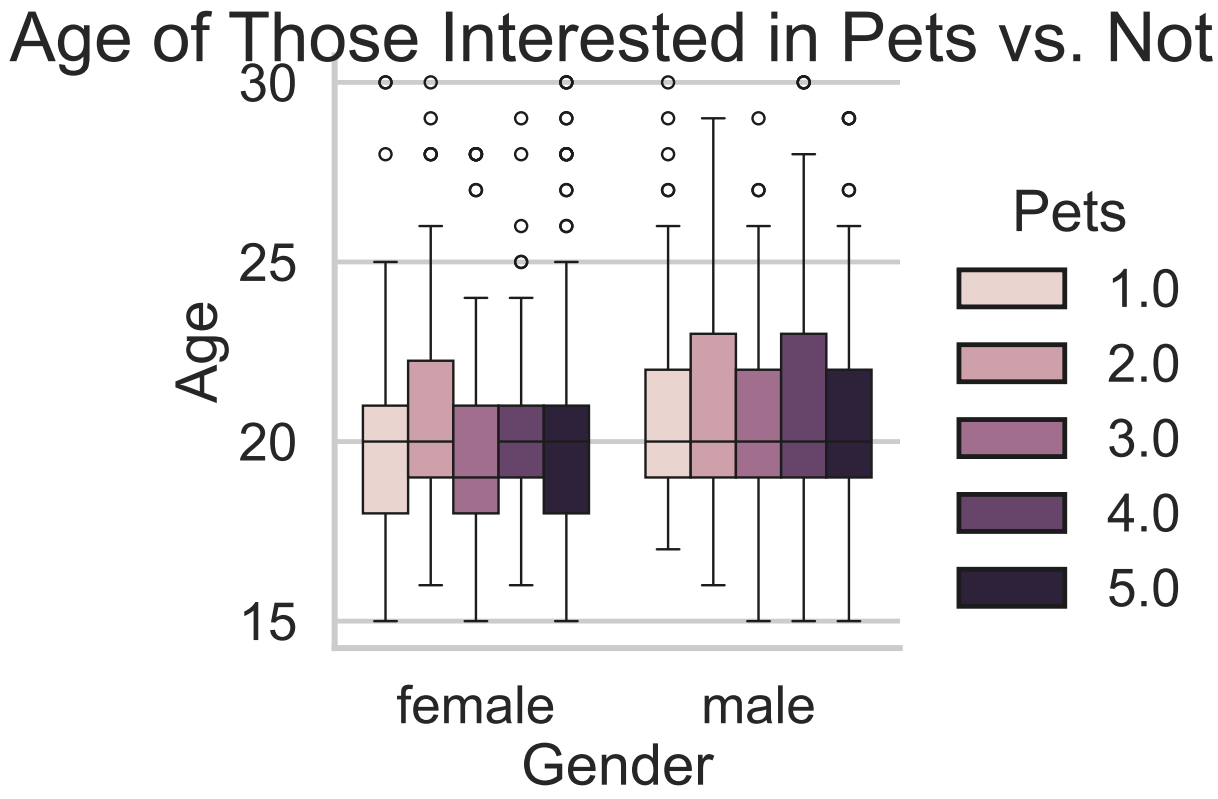
```
            join=False,
            capsize=0.1)

# Rotate x-tick labels to 90 degree
plt.xticks(rotation=90)

# Show plot
plt.show()
```



5. Set palette to `"Blues"`, Create box plot using `catplot`, with `Gender` on the x-axis and `Age` on y-axis, set hue to `Pets`, which means those interested in pets, add title.

```
# Import packages
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
```

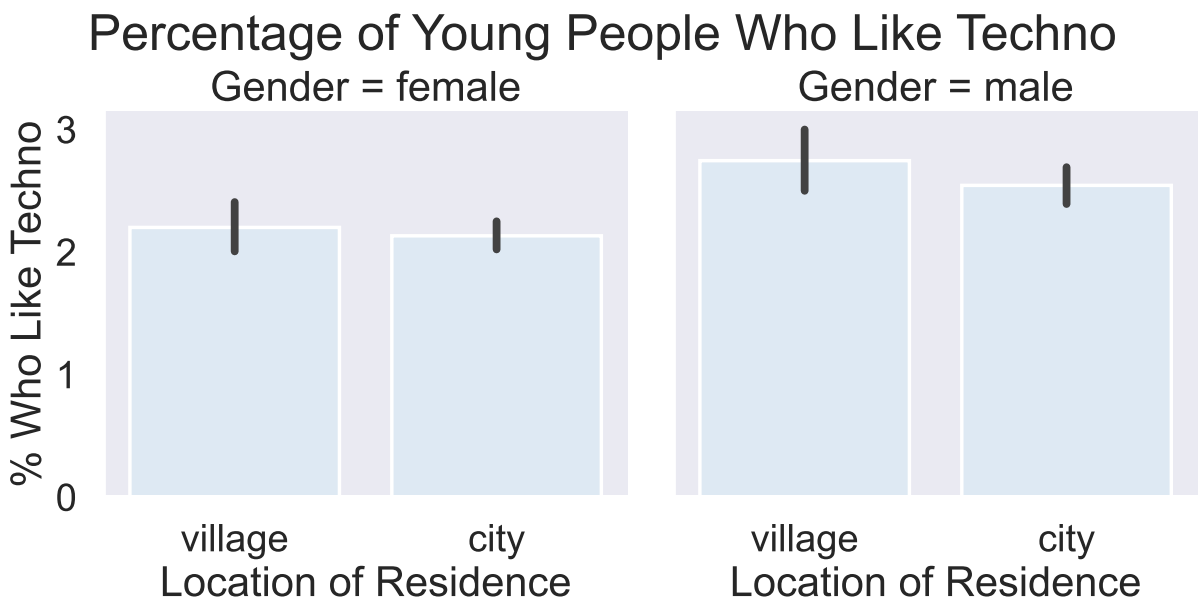
```
# Import dataset
survey_data = pd.read_csv('datasets/young-people-survey-responses.csv')

# Set palette to "Blues"
sns.set_palette("Blues")

# Adjust to add subgroups based on "Interested in Pets"
g = sns.catplot(x="Gender",
                y="Age", data=survey_data,
                kind="box", hue="Pets")

# Set title to "Age of Those Interested in Pets vs. Not"
g.fig.suptitle("Age of Those Interested in Pets vs. Not")

# Show plot
plt.show()
```



6. Create a bar plot using `relplot` and setting x-axis to `Village - town` and y-axis to `Gender` and `col` to `Gender`. Add title and x and y labels.

```
# Import packages
import matplotlib.pyplot as plt
```

```
import seaborn as sns
import pandas as pd

# Import dataset
survey_data = pd.read_csv('datasets/young-people-survey-responses.csv')

# Bar plot with subgroups and subplots
# Set the figure style to "dark".
sns.set_style("dark")

# Adjust the bar plot code to add subplots based on "Gender", arranged in columns.
g = sns.catplot(x="Village - town", y="Techno",
                data=survey_data, kind="bar",
                col="Gender")

# Add the title "Percentage of Young People Who Like Techno" to this FacetGrid plot.
g.fig.suptitle("Percentage of Young People Who Like Techno", y=1.02)
g.set(xlabel="Location of Residence",
      ylabel="% Who Like Techno")

# Label the x-axis "Location of Residence" and y-axis "% Who Like Techno".
plt.show()
```



Reference

Introduction to Data Visualization with Seaborn Course for Associate Data Scientist in Python Carrer Track in DataCamp Inc by Erin Case.