Seaborn is a powerful Python library that makes it easy to create informative and attractive visualizations. This exercise provides an introduction to Seaborn and teaches you how to visualize your data using plots such as scatter plots, box plots, and bar plots.

### Course Description



Seaborn is a powerful Python library that makes it easy to create informative and attractive data visualizations. This 4-hour course provides an introduction to how you can use Seaborn to create a variety of plots, including scatter plots, count plots, bar plots, and box plots, and how you can customize your visualizations.   
  
You’ll explore this library and create Seaborn plots based on a variety of real-world data sets, including exploring how air pollution in a city changes through the day and looking at what young people like to do in their free time. This data will give you the opportunity to find out about Seaborn’s advantages first hand, including how you can easily create subplots in a single figure and how to automatically calculate confidence intervals.   
  
By the end of this course, you’ll be able to use Seaborn in various situations to explore your data and effectively communicate the results of your data analysis to others. These skills are highly sought-after for data analysts, data scientists, and any other job that may involve creating data visualizations. If you’d like to continue your learning, this course is part of several tracks, including the Data Visualization track, where you can add more libraries and techniques to your skillset.

What is Seaborn, and when should you use it? In this chapter, you will find out! Plus, you will learn how to create scatter plots and count plots with both lists of data and pandas DataFrames. You will also be introduced to one of the big advantages of using Seaborn - the ability to easily add a third variable to your plots by using color to represent different subgroups.

**Introduction to Seaborn**

**50 XP**

**1. Introduction to Seaborn**

Hello! Welcome to this introductory course on Seaborn! My name is Erin Case, and I'll be your instructor.

**2. What is Seaborn?**

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

**3. 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.

**4. 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.

**5. 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.

**6. 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 dot 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 dot show()" function from Matplotlib to show the scatterplot. This plot shows us that taller people tend to have a higher weight.

**7. 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.

**8. 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. 1 Waskom, M. L. (2021). seaborn: statistical data visualization. https://seaborn.pydata.org/

**9. Let's practice!**

I'm excited to dive into Seaborn with you throughout this course. For now, let's practice what you've just learned!

# Making a scatter plot with lists

In this exercise, we'll use a dataset that contains information about 227 countries. This dataset has lots of interesting information on each country, such as the country's birth rates, death rates, and its gross domestic product (GDP). GDP is the value of all the goods and services produced in a year, expressed as dollars per person.

We've created three lists of data from this dataset to get you started. gdp is a list that contains the value of GDP per country, expressed as dollars per person. phones is a list of the number of mobile phones per 1,000 people in that country. Finally, percent\_literate is a list that contains the percent of each country's population that can read and write.

##### Instructions ¼

##### Import Matplotlib and Seaborn using the standard naming convention.

# Import Matplotlib and Seaborn

import matplotlib.pyplot as pyplot

import seaborn as sns

# Import Matplotlib and Seaborn import matplotlib.pyplot as pyplot import seaborn as sns

Create a scatter plot of GDP (gdp) vs. number of phones per 1000 people (phones).

# Import Matplotlib and Seaborn

import matplotlib.pyplot as plt

import seaborn as sns

# Create scatter plot with GDP on the x-axis and number of phones on the y-axis

# Import Matplotlib and Seaborn

import matplotlib.pyplot as plt

import seaborn as sns

# Create scatter plot with GDP on the x-axis and number of phones on the y-axis

sns.scatterplot(x=gdp, y=phones)

# Import Matplotlib and Seaborn

import matplotlib.pyplot as plt

import seaborn as sns

# Create scatter plot with GDP on the x-axis and number of phones on the y-axis

sns.scatterplot(x=gdp, y=phones)

<AxesSubplot:>

Display the plot.

# Import Matplotlib and Seaborn

import matplotlib.pyplot as plt

import seaborn as sns

# Create scatter plot with GDP on the x-axis and number of phones on the y-axis

sns.scatterplot(x=gdp, y=phones)

# Show plot

# Import Matplotlib and Seaborn import matplotlib.pyplot as plt import seaborn as sns # Create scatter plot with GDP on the x-axis and number of phones on the y-axis sns.scatterplot(x=gdp, y=phones) # Show plot plt.show()

* Change the scatter plot so it displays the percent of the population that can read and write (percent\_literate) on the y-axis.
* # Import Matplotlib and Seaborn
* import matplotlib.pyplot as plt
* import seaborn as sns
* # Change this scatter plot to have percent literate on the y-axis
* sns.scatterplot(x=gdp, y=percent\_literate)
* # Show plot
* plt.show()

# Import Matplotlib and Seaborn import matplotlib.pyplot as plt import seaborn as sns # Change this scatter plot to have percent literate on the y-axis sns.scatterplot(x=gdp, y=percent\_literate) # Show plot plt.show()

Alright! While this plot does not show a linear relationship between GDP and percent literate, countries with a lower GDP do seem more likely to have a lower percent of the population that can read and write.

# Making a count plot with a list

In the last exercise, we explored a dataset that contains information about 227 countries. Let's do more exploration of this data - specifically, how many countries are in each region of the world?

To do this, we'll need to use a count plot. Count plots take in a categorical list and return bars that represent the number of list entries per category. You can create one here using a list of regions for each country, which is a variable named region.

##### Instructions

* Import Matplotlib and Seaborn using the standard naming conventions.
* Use Seaborn to create a count plot with region on the y-axis.
* Display the plot.
* # Import Matplotlib and Seaborn
* # Create count plot with region on the y-axis
* \_\_\_\_.\_\_\_\_(y=\_\_\_\_)
* # Show plot

# Import Matplotlib and Seaborn

import matplotlib.pyplot as plt

import seaborn as sns

# Create count plot with region on the y-axis

sns.countplot(y=region)

# Show plot

plt.show()

# Import Matplotlib and Seaborn import matplotlib.pyplot as plt import seaborn as sns # Create count plot with region on the y-axis sns.countplot(y=region) # Show plot plt.show()

**Using pandas with Seaborn**

**50 XP**

**1. 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!

**2. 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.

**3. 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 dot 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?"

**4. 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 dot 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.

**5. "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.

**6. "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.

**7. Let's practice!**

Now it's time to try out using pandas with Seaborn!

# "Tidy" vs. "untidy" data

Here, we have a sample dataset from a survey of children about their favorite animals. But can we use this dataset as-is with Seaborn? Let's use pandas to import the csv file with the data collected from the survey and determine whether it is tidy, which is essential to having it work well with Seaborn.

To get you started, the filepath to the csv file has been assigned to the variable csv\_filepath.

Note that because csv\_filepath is a Python variable, you will not need to put quotation marks around it when you read the csv.

##### Instructions 1/2

Read the csv file located at csv\_filepath into a DataFrame named df

Print the head of df to show the first five rows.

# Import pandas

import pandas as pd

# Create a DataFrame from csv file

\_\_\_\_ = pd.\_\_\_\_(csv\_filepath)

# Print the head of df

print(df.\_\_\_\_)

# Import pandas

import pandas as pd

# Create a DataFrame from csv file

df = pd.read\_csv(csv\_filepath)

# Print the head of df

print(df.head())

# Import pandas

import pandas as pd

# Create a DataFrame from csv file

df = pd.read\_csv(csv\_filepath)

# Print the head of df

print(df.head())

Unnamed: 0 How old are you?

0 Marion 12

1 Elroy 16

2 NaN What is your favorite animal?

3 Marion dog

4 Elroy cat

# Making a count plot with a DataFrame

In this exercise, we'll look at the responses to a survey sent out to young people. Our primary question here is: how many young people surveyed report being scared of spiders? Survey participants were asked to agree or disagree with the statement "I am afraid of spiders". Responses vary from 1 to 5, where 1 is "Strongly disagree" and 5 is "Strongly agree".

To get you started, the filepath to the csv file with the survey data has been assigned to the variable csv\_filepath.

Note that because csv\_filepath is a Python variable, you will not need to put quotation marks around it when you read the csv.

##### Instructions

**100 XP**

* Import Matplotlib, pandas, and Seaborn using the standard names.
* Create a DataFrame named df from the csv file located at csv\_filepath.
* Use the countplot() function with the x= and data= arguments to create a count plot with the "Spiders" column values on the x-axis.
* Display the plot.
* # Import Matplotlib, pandas, and Seaborn
* # Create a DataFrame from csv file
* # Create a count plot with "Spiders" on the x-axis
* # Display the plot

# Import Matplotlib, pandas, and Seaborn

import matplotlib.pyplot as  plt

import pandas as pd

import seaborn as sns

# Create a DataFrame from csv file

df = pd.read\_csv(csv\_filepath)

# Create a count plot with "Spiders" on the x-axis

sns.countplot(x='Spiders', data=df)

# Display the plot

plt.show()

# Import Matplotlib, pandas, and Seaborn import matplotlib.pyplot as plt import pandas as pd import seaborn as sns # Create a DataFrame from csv file df = pd.read\_csv(csv\_filepath) # Create a count plot with "Spiders" on the x-axis sns.countplot(x='Spiders', data=df) # Display the plot plt.show()

Awesome! This plot shows us that most young people reported not being afraid of spiders.

**Adding a third variable with hue**

**50 XP**

**1. 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.

**2. 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.

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

**3. 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.

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

**4. 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.

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

**5. 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".

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

**6. Specifying hue colors**

You can also control the colors assigned to each value using the "palette" parameter. This parameter takes in a dictionary, which is a data structure that has key-value pairs. This dictionary should map the variable values to the colors you want to represent the value. Here, we create a dictionary called "hue colors" that maps the value "Yes" to the color black and the value "No" to the color red. When we set hue equal to "smoker" and the palette parameter equal to this dictionary, we have a scatter plot where smokers are represented with black dots and non-smokers are represented with red dots.

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

**7. 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.

**8. 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.

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

**9. 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.

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

**10. Let's practice!**

We'll be using hue a lot in this course, so let's practice what we've learned to round out the first chapter!

# Hue and scatter plots

In the prior video, 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.

For now, we'll look at the relationship between the number of absences they have in school and their final grade in the course, segmented by where the student lives (rural vs. urban area).

##### Instructions 1/2

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).

# Import Matplotlib and Seaborn

import matplotlib.pyplot as plt

import seaborn as sns

# Create a scatter plot of absences vs. final grade

# Show plot

plt.show()

# Import Matplotlib and Seaborn

import matplotlib.pyplot as plt

import seaborn as sns

# Create a scatter plot of absences vs. final grade

sns.scatterplot(x='absences', y='G3', data=student\_data,  hue='location')

# Show plot

plt.show()

# Import Matplotlib and Seaborn import matplotlib.pyplot as plt import seaborn as sns # Create a scatter plot of absences vs. final grade sns.scatterplot(x='absences', y='G3', data=student\_data, hue='location') # Show plot plt.show()

Make "Rural" appear before "Urban" in the plot legend.

# Import Matplotlib and Seaborn

import matplotlib.pyplot as plt

import seaborn as sns

# Change the legend order in the scatter plot

sns.scatterplot(x="absences", y="G3",

                data=student\_data,

                hue="location")

# Show plot

plt.show()

# Import Matplotlib and Seaborn

import matplotlib.pyplot as plt

import seaborn as sns

# Change the legend order in the scatter plot

sns.scatterplot(x="absences", y="G3",

                data=student\_data,

                hue="location", hue\_order=['Rural','Urban'])

# Show plot

plt.show()

# Import Matplotlib and Seaborn import matplotlib.pyplot as plt import seaborn as sns # Change the legend order in the scatter plot sns.scatterplot(x="absences", y="G3", data=student\_data, hue="location", hue\_order=['Rural','Urban']) # Show plot plt.show()

# Hue and count plots

Let's continue exploring our dataset from students in secondary school by looking at a new variable. The "school" column indicates the initials of which school the student attended - either "GP" or "MS".

In the last exercise, we created a scatter plot where the plot points were colored based on whether the student lived in an urban or rural area. How many students live in urban vs. rural areas, and does this vary based on what school the student attends? Let's make a count plot with subgroups to find out.

##### Instructions

* 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".
* Create a count plot with "school" on the x-axis using the student\_data DataFrame.
  + Add subgroups to the plot using "location" variable and use the palette\_colors dictionary to make the location subgroups green and blue.
* # Import Matplotlib and Seaborn
* import matplotlib.pyplot as plt
* import seaborn as sns
* # Create a dictionary mapping subgroup values to colors
* palette\_colors = {\_\_\_\_: "green", \_\_\_\_: "blue"}
* # Create a count plot of school with location subgroups
* # Display plot
* plt.show()
* # Import Matplotlib and Seaborn
* import matplotlib.pyplot as plt
* import seaborn as sns
* # 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', data=student\_data, hue='location', palette=palette\_colors)
* # Display plot
* plt.show()

# Import Matplotlib and Seaborn import matplotlib.pyplot as plt import seaborn as sns # 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', data=student\_data, hue='location', palette=palette\_colors) # Display plot plt.show()

Awesome. Students at GP tend to come from an urban location, but students at MS are more evenly split. Congratulations on finishing Chapter 1!

**Introduction to relational plots and subplots**

**50 XP**

**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".

**2. 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?"

**3. Questions about quantitative variables**

"what's the relationship between the number of absences a student has and their final grade?"

**4. Questions about quantitative variables**

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.

**5. 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, we started to look at subgroups by using the "hue" parameter to visualize each subgroup using a different color on the same plot.

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

**6. Visualizing subgroups**

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

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

**7. 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.

**8. scatterplot() vs. relplot()**

Let's return to our scatter plot of total bill versus tip amount from the tips dataset. On the left, 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".

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

**9. Subplots in columns**

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

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

**10. Subplots in rows**

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

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

**11. 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.

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

**12. 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.

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

**13. Wrapping columns**

Here, we set "col\_wrap" equal to two plots per row.

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

**14. 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.

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

**15. Let's practice!**

Alright! Now it's time to practice what we've learned and create some relational plots!

# Creating subplots with col and row

We've seen in prior exercises that students with more absences ("absences") tend to have lower final grades ("G3"). Does this relationship hold regardless of how much time students study each week?

To answer this, we'll look at the relationship between the number of absences that a student has in school and their final grade in the course, creating separate subplots based on each student's weekly study time ("study\_time").

Seaborn has been imported as sns and matplotlib.pyplot has been imported as plt.

##### Instructions 1/3

[1](javascript:void(0))Modify the code to use relplot() instead of scatterplot().

 [2](javascript:void(0))Modify the code to create one scatter plot for each level of the variable "study\_time", arranged in columns.

 [3](javascript:void(0))Adapt your code to create one scatter plot for each level of a student's weekly study time, this time arranged in rows.

# Change to use relplot() instead of scatterplot()

sns.scatterplot(x="absences", y="G3",

                data=student\_data)

# Show plot

plt.show()

# Change to use relplot() instead of scatterplot() sns.relplot(x="absences", y="G3", data=student\_data) # Show plot plt.show()

# Change to use relplot() instead of scatterplot()

sns.relplot(x="absences", y="G3",

                data=student\_data)

# Show plot

plt.show()

# Change to make subplots based on study time

sns.relplot(x="absences", y="G3",

            data=student\_data,

            kind="scatter")

# Show plot

plt.show()

# Change to make subplots based on study time sns.relplot(x="absences", y="G3", data=student\_data, kind="scatter", col='study\_time') # Show plot plt.show()

# Change this scatter plot to arrange the plots in rows instead of columns

sns.relplot(x="absences", y="G3",

            data=student\_data,

            kind="scatter",

            col="study\_time")

# Show plot

plt.show()

# Change this scatter plot to arrange the plots in rows instead of columns

sns.relplot(x="absences", y="G3",

            data=student\_data,

            kind="scatter",

            row="study\_time")

# Show plot

plt.show()

# Change this scatter plot to arrange the plots in rows instead of columns sns.relplot(x="absences", y="G3", data=student\_data, kind="scatter", row="study\_time") # Show plot plt.show()

Great job! Because these subplots had a large range of x values, it's easier to read them arranged in rows instead of columns.

# Creating two-factor subplots

Let's continue looking at the student\_data dataset of students in secondary school. Here, we want to answer the following question: does a student's first semester grade ("G1") tend to correlate with their final grade ("G3")?

There are many aspects of a student's life that could result in a higher or lower final grade in the class. For example, some students receive extra educational support from their school ("schoolsup") or from their family ("famsup"), which could result in higher grades. Let's try to control for these two factors by creating subplots based on whether the student received extra educational support from their school or family.

Seaborn has been imported as sns and matplotlib.pyplot has been imported as plt.

##### Instructions 1/3

**Use relplot() to create a scatter plot with "G1" on the x-axis and "G3" on the y-axis, using the student\_data DataFrame**.

# Create a scatter plot of G1 vs. G3

# Show plot

plt.show()

# Create a scatter plot of G1 vs. G3

sns.relplot(x='G1', y='G3', data=student\_data, kind='scatter')

# Show plot

plt.show()

# Create a scatter plot of G1 vs. G3 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 ("schoolsup"), ordered so that "yes" comes before "no".**

# Adjust to add subplots based on school support

sns.relplot(x="G1", y="G3",

            data=student\_data,

            kind="scatter")

# Show plot

plt.show()

# Adjust to add subplots based on school support

sns.relplot(x="G1", y="G3",

            data=student\_data,

            kind="scatter", col='schoolsup', col\_order=['yes','no'])

# Show plot

plt.show()

# Adjust to add subplots based on school support 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"), ordered so that "yes" comes before "no". This will result in subplots based on two factors.

# Adjust further to add subplots based on family support sns.relplot(x="G1", y="G3", data=student\_data, kind="scatter", col="schoolsup", row='famsup', col\_order=["yes", "no"], row\_order=['yes', 'no']) # Show plot plt.show()

# Adjust further to add subplots based on family support

sns.relplot(x="G1", y="G3",

            data=student\_data,

            kind="scatter",

            col="schoolsup", row='famsup',

            col\_order=["yes", "no"], row\_order=['yes', 'no'])

# Show plot

plt.show()

Fantastic! It looks like the first semester grade does correlate with the final grade, regardless of what kind of support the student received.

**Customizing scatter plots**

**50 XP**

**1. Customizing scatter plots**

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

**2. 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.

**3. 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.

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

**4. 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.

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

**5. 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.

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

**6. 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.

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

**7. Let's practice!**

This is just the beginning of what you can do to customize your Seaborn scatter plots. Make sure to check out the Seaborn documentation for more options like specifying specific sizes or point styles to use in your plots. For now, let's practice what we've learned!

# Changing the size of scatter plot points

In this exercise, we'll explore Seaborn's mpg dataset, which contains one row per car model and includes information such as the year the car was made, the number of miles per gallon ("M.P.G.") it achieves, the power of its engine (measured in "horsepower"), and its country of origin.

What is the relationship between the power of a car's engine ("horsepower") and its fuel efficiency ("mpg")? And how does this relationship vary by the number of cylinders ("cylinders") the car has? Let's find out.

Let's continue to use relplot() instead of scatterplot() since it offers more flexibility.

##### Instructions 1/2

**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 Matplotlib and Seaborn

import matplotlib.pyplot as plt

import seaborn as sns

# Create scatter plot of horsepower vs. mpg

# Show plot

plt.show()

# Import Matplotlib and Seaborn

import matplotlib.pyplot as plt

import seaborn as sns

# Create scatter plot of horsepower vs. mpg

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

# Show plot

plt.show()

# Import Matplotlib and Seaborn import matplotlib.pyplot as plt import seaborn as sns # Create scatter plot of horsepower vs. mpg sns.relplot(x='horsepower', y='mpg', data=mpg, kind='scatter', size='cylinders') # Show plot plt.show()

**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 Matplotlib and Seaborn

import matplotlib.pyplot as plt

import seaborn as sns

# Create scatter plot of horsepower vs. mpg

sns.relplot(x="horsepower", y="mpg",

            data=mpg, kind="scatter",

            size="cylinders",hue='cylinders')

# Show plot

plt.show()

# Import Matplotlib and Seaborn import matplotlib.pyplot as plt import seaborn as sns # Create scatter plot of horsepower vs. mpg sns.relplot(x="horsepower", y="mpg", data=mpg, kind="scatter", size="cylinders",hue='cylinders') # Show plot plt.show()

**Great job! Cars with higher horsepower tend to get a lower number of miles per gallon. They also tend to have a higher number of cylinders.**

# Changing the style of scatter plot points

Let's continue exploring Seaborn's mpg dataset by looking at the relationship between how fast a car can accelerate ("acceleration") and its fuel efficiency ("mpg"). Do these properties vary by country of origin ("origin")?

Note that the "acceleration" variable is the time to accelerate from 0 to 60 miles per hour, in seconds. Higher values indicate slower acceleration.

##### Instruction

* 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 Matplotlib and Seaborn
* import matplotlib.pyplot as plt
* import seaborn as sns
* # Create a scatter plot of acceleration vs. mpg
* # Show plot
* plt.show()
* # Import Matplotlib and Seaborn
* import matplotlib.pyplot as plt
* import seaborn as sns
* # Create a scatter plot of acceleration vs. mpg
* sns.relplot(x='acceleration', y='mpg', data=mpg, kind='scatter', hue='origin', style='origin')
* # Show plot
* plt.show()

# Import Matplotlib and Seaborn import matplotlib.pyplot as plt import seaborn as sns # Create a scatter plot of acceleration vs. mpg sns.relplot(x='acceleration', y='mpg', data=mpg, kind='scatter', hue='origin', style='origin') # Show plot plt.show()

Looks good! Cars from the USA tend to accelerate more quickly and get lower miles per gallon compared to cars from Europe and Japan.

**Introduction to line plots**

**50 XP**

**1. Introduction to line plots**

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

**2. 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.

**3. Air pollution data**

In this video, 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".

**4. Scatter plot**

This is a scatter plot with the average nitrogen dioxide level on the y-axis and the hour of the day on the x-axis. We're tracking the same thing over time, so a line plot would be a better choice.

**5. 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.

**6. 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.

**7. Subgroups by location**

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.

**8. 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.

**9. Turning off line style**

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

**10. 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.

**11. Multiple observations per x-value**

This is the scatter plot, displaying one point per observation.

**12. Multiple observations per x-value**

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.

**13. Multiple observations per x-value**

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.

**14. 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.

**15. Turning off confidence interval**

We can also turn off the confidence interval by setting the "ci" parameter equal to "None".

**16. Let's practice!**

Alright, time to practice what we've learned!

# Interpreting line plots

In this exercise, we'll continue to explore Seaborn's mpg dataset, which contains one row per car model and includes information such as the year the car was made, its fuel efficiency (measured in "miles per gallon" or "M.P.G"), and its country of origin (USA, Europe, or Japan).

How has the average miles per gallon achieved by these cars changed over time? Let's use line plots to find out!

##### Instructions 1/2

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

# Create line plot

# Show plot

plt.show()

# Import Matplotlib and Seaborn

import matplotlib.pyplot as plt

import seaborn as sns

# Create line plot

sns.relplot(x='model\_year', y='mpg', data=mpg, kind='line')

# Show plot

plt.show()

# Import Matplotlib and Seaborn import matplotlib.pyplot as plt import seaborn as sns # Create line plot sns.relplot(x='model\_year', y='mpg', data=mpg, kind='line') # Show plot plt.show()

**Which of the following is NOT a correct interpretation of this line plot?**

* The average miles per gallon has generally increased over time.
* **The distribution of miles per gallon is smaller in 1973 compared to 1977.**
* The 95% confidence interval for average miles per gallon in 1970 is approximately 16 - 19.5 miles per gallon.
* This plot assumes that our data is a random sample of all cars in the US, Europe, and Japan.
* **Good job. The shaded region represents a confidence interval for the mean, not the distribution of the observations.**

# Visualizing standard deviation with line plots

In the last exercise, we looked at how the average miles per gallon achieved by cars has changed over time. Now let's use a line plot to visualize how the distribution of miles per gallon has changed over time.

Seaborn has been imported as sns and matplotlib.pyplot has been imported as plt.

##### Instructions

* Change the plot so the shaded area shows the standard deviation instead of the confidence interval for the mean.
* # Make the shaded area show the standard deviation
* sns.relplot(x="model\_year", y="mpg",
* data=mpg, kind="line", ci='sd')
* # Show plot
* plt.show()

# Make the shaded area show the standard deviation sns.relplot(x="model\_year", y="mpg", data=mpg, kind="line", ci='sd') # Show plot plt.show()

Excellent. Unlike the plot in the last exercise, this plot shows us the distribution of miles per gallon for all the cars in each year.

# Plotting subgroups in line plots

Let's continue to look at the mpg dataset. We've seen that the average miles per gallon for cars has increased over time, but how has the average horsepower for cars changed over time? And does this trend differ by country of origin?

##### Instructions 1/3

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

# Create line plot of model year vs. horsepower

# Show plot

plt.show()

# Import Matplotlib and Seaborn

import matplotlib.pyplot as plt

import seaborn as sns

# Create line plot of model year vs. horsepower

sns.relplot(x='model\_year', y='horsepower', data=mpg, kind='line', ci=None)

# Show plot

plt.show()

# Import Matplotlib and Seaborn import matplotlib.pyplot as plt import seaborn as sns # Create line plot of model year vs. horsepower sns.relplot(x='model\_year', y='horsepower', data=mpg, kind='line', ci=None) # Show plot plt.show()

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

# Change to create subgroups for country of origin

sns.relplot(x="model\_year", y="horsepower",

            data=mpg, kind="line",

            ci=None)

# Show plot

plt.show()

# Import Matplotlib and Seaborn

import matplotlib.pyplot as plt

import seaborn as sns

# Add markers and make each line have the same style

sns.relplot(x="model\_year", y="horsepower",

            data=mpg, kind="line",

            ci=None, style="origin",

            hue="origin")

# Show plot

plt.show()

# Import Matplotlib and Seaborn import matplotlib.pyplot as plt import seaborn as sns # Change to create subgroups for country of origin sns.relplot(x="model\_year", y="horsepower", data=mpg, kind="line", style='origin', hue='origin', ci=None) # Show plot plt.show()

* Add markers for each data point to the lines.
* 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
* # Add markers and make each line have the same style
* sns.relplot(x="model\_year", y="horsepower",
* data=mpg, kind="line",
* ci=None, style="origin",
* hue="origin")
* # Show plot
* plt.show()

# Import Matplotlib and Seaborn

import matplotlib.pyplot as plt

import seaborn as sns

# Add markers and make each line have the same style

sns.relplot(x="model\_year", y="horsepower",

            data=mpg, kind="line",

            ci=None, style="origin",

            hue="origin", markers=True, dashes=False)

# Show plot

plt.show()

# Import Matplotlib and Seaborn import matplotlib.pyplot as plt import seaborn as sns # Add markers and make each line have the same style sns.relplot(x="model\_year", y="horsepower", data=mpg, kind="line", ci=None, style="origin", hue="origin", markers=True, dashes=False) # Show plot plt.show()

Nice work! Now that we've added subgroups, we can see that this downward trend in horsepower was more pronounced among cars from the USA.

**Count plots and bar plots**

**50 XP**

**1. 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.

**2. 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, like the number of men reporting that they feel masculine. Most men surveyed here feel "somewhat" or "very" masculine.

**3. 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.

**4. 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.

**5. 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".

**6. 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.

**7. 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".

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

**8. 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.

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

**9. Turning off confidence intervals**

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

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

**10. 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.

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

**11. Let's practice!**

We'll introduce more types of categorical plots later in the chapter, but for now, let's practice what we've learned!

# Count plots

In this exercise, we'll return to exploring our dataset that contains the responses to a survey sent out to young people. We might suspect that young people spend a lot of time on the internet, but how much do they report using the internet each day? Let's use a count plot to break down the number of survey responses in each category and then explore whether it changes based on age.

As a reminder, to create a count plot, we'll use the catplot() function and specify the name of the categorical variable to count (x=\_\_\_\_), the pandas DataFrame to use (data=\_\_\_\_), and the type of plot (kind="count").

Seaborn has been imported as sns and matplotlib.pyplot has been imported as plt.

##### Instructions 1/3

Use sns.catplot() to create a count plot using the survey\_data DataFrame with "Internet usage" on the x-axis.

# Create count plot of internet usage

# Show plot

plt.show()

# Create count plot of internet usage sns.catplot(x='Internet usage', data= survey\_data, kind='count') # Show plot plt.show()

# Create count plot of internet usage

sns.catplot(x='Internet usage', data= survey\_data, kind='count')

# Show plot

plt.show()

# Create count plot of internet usage sns.catplot(x='Internet usage', data= survey\_data, kind='count') # Show plot plt.show()

Make the bars horizontal instead of vertical.

# Change the orientation of the plot

sns.catplot( y="Internet usage", data=survey\_data,

            kind="count")

# Show plot

plt.show()

# Change the orientation of the plot sns.catplot( y="Internet usage", data=survey\_data, kind="count") # Show plot plt.show()

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.

# Separate into column subplots based on age category

sns.catplot(y="Internet usage", data=survey\_data,

            kind="count")

# Show plot

plt.show()

# Separate into column subplots based on age category

sns.catplot(y="Internet usage", data=survey\_data,

            kind="count", col='Age Category')

# Show plot

plt.show()

# Separate into column subplots based on age category sns.catplot(y="Internet usage", data=survey\_data, kind="count", col='Age Category') # Show plot plt.show()

Great job! It looks like most young people use the internet for a few hours every day, regardless of their age.

# Bar plots with percentages

Let's continue exploring the responses to a survey sent out to young people. The variable "Interested in Math" is True if the person reported being interested or very interested in mathematics, and False otherwise. What percentage of young people report being interested in math, and does this vary based on gender? Let's use a bar plot to find out.

As a reminder, we'll create a bar plot using the catplot() function, providing the name of categorical variable to put on the x-axis (x=\_\_\_\_), the name of the quantitative variable to summarize on the y-axis (y=\_\_\_\_), the pandas DataFrame to use (data=\_\_\_\_), and the type of categorical plot (kind="bar").

Seaborn has been imported as sns and matplotlib.pyplot has been imported as plt.

##### Instructions

**Use the survey\_data DataFrame and sns.catplot() to create a bar plot with "Gender" on the x-axis and "Interested in Math" on the y-axis.**

# Create a bar plot of interest in math, separated by gender

# Show plot

plt.show()

# Create a bar plot of interest in math, separated by gender

sns.catplot(x='Gender', y='Interested in Math', data=survey\_data, kind='bar')

# Show plot

plt.show()

# Create a bar plot of interest in math, separated by gender sns.catplot(x='Gender', y='Interested in Math', data=survey\_data, kind='bar') # Show plot plt.show()

Excellent. When the y-variable is True/False, bar plots will show the percentage of responses reporting True. This plot shows us that males report a much higher interest in math compared to females.

# Customizing bar plots

In this exercise, we'll explore data from students in secondary school. The "study\_time" variable records each student's reported weekly study time as one of the following categories: "<2 hours", "2 to 5 hours", "5 to 10 hours", or ">10 hours". Do students who report higher amounts of studying tend to get better final grades? Let's compare the average final grade among students in each category using a bar plot.

Seaborn has been imported as sns and matplotlib.pyplot has been imported as plt.

##### Instructions 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.

# Create bar plot of average final grade in each study category

# Show plot

plt.show()

# Create bar plot of average final grade in each study category

sns.catplot(x='study\_time', y='G3', data=student\_data, kind='bar')

# Show plot

plt.show()

# Create bar plot of average final grade in each study category sns.catplot(x='study\_time', y='G3', data=student\_data, kind='bar') # Show plot plt.show()

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.

# List of categories from lowest to highest

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()

# List of categories from lowest to highest 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()

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

# List of categories from lowest to highest

category\_order = ["<2 hours",

                  "2 to 5 hours",

                  "5 to 10 hours",

                  ">10 hours"]

# Turn off the confidence intervals

sns.catplot(x="study\_time", y="G3",

            data=student\_data,

            kind="bar",

            order=category\_order)

# Show plot

plt.show()

# List of categories from lowest to highest

category\_order = ["<2 hours",

                  "2 to 5 hours",

                  "5 to 10 hours",

                  ">10 hours"]

# Turn off the confidence intervals

sns.catplot(x="study\_time", y="G3",

            data=student\_data,

            kind="bar",

            order=category\_order, ci=None)

# Show plot

plt.show()

# List of categories from lowest to highest

category\_order = ["<2 hours", "2 to 5 hours", "5 to 10 hours", ">10 hours"] # Turn off the confidence intervals sns.catplot(x="study\_time", y="G3", data=student\_data, kind="bar", order=category\_order, ci=None) # Show plot plt.show()

Great work! Students in our sample who studied more have a slightly higher average grade, but it's not a strong relationship.

**Box plots**

**50 XP**

**1. Creating a box plot**

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

**2. What is a box plot?**

A box plot shows the distribution of quantitative data. The colored box represents the 25th to 75th percentile, and the line in the middle of the box represents the median. The whiskers give a sense of the spread of the distribution, and the floating points represent outliers. Box plots are commonly used as a way to compare the distribution of a quantitative variable across different groups of a categorical variable. To see this, let's look at this example. The box plot shown here uses the tips dataset and compares the distribution of the total bill paid per table across the different days of the week. From this box plot we can quickly see that the median bill is higher on Saturday and Sunday, but the spread of the distribution is also larger. This comparison would be much harder to do with other types of visualizations.

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

**3. 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.

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

**4. 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".

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

**5. 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.

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

**6. Changing the whiskers using `whis`**

By default, the whiskers extend to 1 point 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 point 5 times the IQR (which is the default) to 2 times the IQR by setting "whis" equal to 2 point 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.

**7. 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.

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

**8. Let's practice!**

Let's now practice creating and customizing box plots!

# Create and interpret a box plot

Let's continue using the student\_data dataset. In an earlier exercise, we explored the relationship between studying and final grade by using a bar plot to compare the average final grade ("G3") among students in different categories of "study\_time".

In this exercise, we'll try using a box plot look at this relationship instead. As a reminder, to create a box plot you'll need to use the catplot() function and specify the name of the categorical variable to put on the x-axis (x=\_\_\_\_), the name of the quantitative variable to summarize on the y-axis (y=\_\_\_\_), the pandas DataFrame to use (data=\_\_\_\_), and the type of plot (kind="box").

We have already imported matplotlib.pyplot as plt and seaborn as sns.

##### Instructions 1/2

Use sns.catplot() and the student\_data DataFrame to create a box plot with "study\_time" on the x-axis and "G3" on the y-axis. Set the ordering of the categories to study\_time\_order.

# 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

# Show plot

plt.show()

# 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()

# 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()

#### Question

Which of the following is a correct interpretation of this box plot?

##### Possible Answers

* The 75th percentile of grades is highest among students who study more than 10 hours a week.
* There are no outliers plotted for these box plots.
* The 5th percentile of grades among students studying less than 2 hours is 5.0.
* **The median grade among students studying less than 2 hours is 10.0.**

**Correct! The line in the middle of each box represents the median.**

# Omitting outliers

Now let's use the student\_data dataset to compare the distribution of final grades ("G3") between students who have internet access at home and those who don't. To do this, we'll use the "internet" variable, which is a binary (yes/no) indicator of whether the student has internet access at home.

Since internet may be less accessible in rural areas, we'll add subgroups based on where the student lives. For this, we can use the "location" variable, which is an indicator of whether a student lives in an urban ("Urban") or rural ("Rural") location.

Seaborn has already been imported as sns and matplotlib.pyplot has been imported as plt. As a reminder, you can omit outliers in box plots by setting the sym parameter equal to an empty string ("").

##### Instructions

* 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.
* # Create a box plot with subgroups and omit the outliers
* # Show plot
* plt.show()
* # Create a box plot with subgroups and omit the outliers
* sns.catplot(x='internet', y='G3', data=student\_data, kind='box', hue='location', sym='')
* # Show plot
* plt.show()

# Create a box plot with subgroups and omit the outliers sns.catplot(x='internet', y='G3', data=student\_data, kind='box', hue='location', sym='') # Show plot plt.show()

Success! The median grades are quite similar between each group, but the spread of the distribution looks larger among students who have internet access.

# Adjusting the whiskers

In the lesson we saw that there are multiple ways to define the whiskers in a box plot. In this set of exercises, we'll continue to use the student\_data dataset to compare the distribution of final grades ("G3") between students who are in a romantic relationship and those that are not. We'll use the "romantic" variable, which is a yes/no indicator of whether the student is in a romantic relationship.

Let's create a box plot to look at this relationship and try different ways to define the whiskers.

We've already imported Seaborn as sns and matplotlib.pyplot as plt.

##### Instructions 1/3

[1](javascript:void(0))Adjust the code to make the box plot whiskers to extend to 0.5 \* IQR. Recall: the IQR is the interquartile range.

[2](javascript:void(0))Change the code to set the whiskers to extend to the 5th and 95th percentiles.

[3](javascript:void(0))Change the code to set the whiskers to extend to the min and max values.

# Set the whiskers to 0.5 \* IQR

sns.catplot(x="romantic", y="G3",

            data=student\_data,

            kind="box")

# Show plot

plt.show()

# Extend the whiskers to the 5th and 95th percentile

sns.catplot(x="romantic", y="G3",

            data=student\_data,

            kind="box",

            whis=[5,95])

# Show plot

plt.show()

# Extend the whiskers to the 5th and 95th percentile sns.catplot(x="romantic", y="G3", data=student\_data, kind="box", whis=[5,95]) # Show plot plt.show()

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

# Set the whiskers at the min and max values sns.catplot(x="romantic", y="G3", data=student\_data, kind="box", whis=[0, 100]) # Show plot plt.show()

# Set the whiskers at the min and max values

sns.catplot(x="romantic", y="G3",

            data=student\_data,

            kind="box",

            whis=[0, 100])

# Show plot

plt.show()

Fantastic! The median grade is the same between these two groups, but the max grade is higher among students who are not in a romantic relationship.

**Point plots**

**50 XP**

**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.

**2. What are point plots?**

Point plots show the mean of a quantitative variable for the observations in each category, plotted as a single point. This point plot uses the tips dataset and shows the average bill among smokers versus non-smokers. The vertical bars extending above and below the mean represent the 95% confidence intervals for that mean. Just like the confidence intervals we saw in line plots and bar plots, these confidence intervals show us the level of uncertainty we have about these mean 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.

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

**3. Point plots vs. line plots**

You may be thinking: point plots look a lot like line plots. What's the difference?

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

**4. Point plots vs. line plots**

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.

**5. 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.

**6. Point plots vs. bar plots**

This is a bar plot of the percent of men per age group surveyed who report thinking that it's important that others see them as masculine, with subgroups based on whether they report feeling masculine or not. This is the same information, represented as a point plot. In the point plot, it's easier to compare the heights of the subgroup points when they're stacked above each other. In the point plot, it's also easier to look at the differences in slope between the categories than it is to compare the heights of the bars between them.

**7. 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".

**8. 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.

**9. 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.

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

**10. Displaying the median**

To have the points and confidence intervals be calculated for the median instead of the mean, import the median function from the numpy library and set "estimator" 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 has a lot of outliers, the median may be a better statistic to use.

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

**11. 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.

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

**12. Turning off confidence intervals**

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

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

**13. Let's practice!**

Alright! Now that we've covered how to interpret and customize point plots, let's practice what we've learned!

# Customizing point plots

Let's continue to look at data from students in secondary school, this time using a point plot to answer the question: does the quality of the student's family relationship influence the number of absences the student has in school? Here, we'll use the "famrel" variable, which describes the quality of a student's family relationship from 1 (very bad) to 5 (very good).

As a reminder, to create a point plot, use the catplot() function and specify the name of the categorical variable to put on the x-axis (x=\_\_\_\_), the name of the quantitative variable to summarize on the y-axis (y=\_\_\_\_), the pandas DataFrame to use (data=\_\_\_\_), and the type of categorical plot (kind="point").

We've already imported Seaborn as sns and matplotlib.pyplot as plt.

Instructions 1/3

##### 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.

# Create a point plot of family relationship vs. absences

# Show plot

plt.show()

# Create a point plot of family relationship vs. absences

sns.catplot(x='famrel', y='absences', data=student\_data, kind='point')

# Show plot

plt.show()

# Create a point plot of family relationship vs. absences sns.catplot(x='famrel', y='absences', data=student\_data, kind='point') # Show plot plt.show()

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

# Add caps to the confidence interval

sns.catplot(x="famrel", y="absences",

            data=student\_data,

            kind="point", capsize=0.2)

# Show plot

plt.show()

# Add caps to the confidence interval sns.catplot(x="famrel", y="absences", data=student\_data, kind="point", capsize=0.2) # Show plot plt.show()

Remove the lines joining the points in each category.

# Remove the lines joining the points

sns.catplot(x="famrel", y="absences",

            data=student\_data,

            kind="point",

            capsize=0.2, join=False)

# Show plot

plt.show()

# Remove the lines joining the points sns.catplot(x="famrel", y="absences", data=student\_data, kind="point", capsize=0.2, join=False) # Show plot plt.show()

Awesome! While the average number of absences is slightly smaller among students with higher-quality family relationships, the large confidence intervals tell us that we can't be sure there is an actual association here.

# Point plots with subgroups

Let's continue exploring the dataset of students in secondary school. This time, we'll ask the question: is being in a romantic relationship associated with higher or lower school attendance? And does this association differ by which school the students attend? Let's find out using a point plot.

We've already imported Seaborn as sns and matplotlib.pyplot as plt.

##### Instructions 1/3

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").

# Create a point plot that uses color to create subgroups

# Show plot

plt.show()

# Create a point plot that uses color to create subgroups

sns.catplot(x='romantic', y='absences', data=student\_data, kind='point',hue='school')

# Show plot

plt.show()

# Create a point plot that uses color to create subgroups sns.catplot(x='romantic', y='absences', data=student\_data, kind='point',hue='school') # Show plot plt.show()

Turn off the confidence intervals for the plot.

# Turn off the confidence intervals for this plot

sns.catplot(x="romantic", y="absences",

            data=student\_data,

            kind="point",

            hue="school", ci=None)

# Show plot

plt.show()

# Turn off the confidence intervals for this plot sns.catplot(x="romantic", y="absences", data=student\_data, kind="point", hue="school", ci=None) # Show plot plt.show()

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 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", estimator=median,

            ci=None)

# Show plot

plt.show()

# 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", estimator=median, ci=None) # Show plot plt.show()

Good work! It looks like students in romantic relationships have a higher average and median number of absences in the GP school, but this association does not hold for the MS school.

**Changing plot style and color**

**50 XP**

**1. 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.

**2. 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.

**3. 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.

**4. Default figure style ("white")**

This is a plot 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.

**5. 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.

**6. 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.

**7. Other styles**

"dark" provides a gray background,

**8. Other styles**

and "darkgrid" provides a gray background with a white grid.

**9. 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.

**10. 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.

**11. 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.

**12. 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.

**13. 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.

**14. 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.

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

**15. Custom palettes**

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

**16. Custom palettes**

or a list of hex color codes.

**17. 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".

**18. Default context: "paper"**

The default context is "paper".

**19. 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.

**20. Let's practice!**

Now that we've seen how to change the plot style, palette, and scale, let's practice!

##### Instructions 1/3

Set the style to "whitegrid" to help the audience determine the number of responses in each category.

# Set the style to "whitegrid"

# 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()

# Set the style to "whitegrid"

sns.set\_style('whitegrid')

# 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()

sns.set\_style('whitegrid') # 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()

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

# Set the color palette to "Purples"

sns.set\_style("whitegrid")

# 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()

# Set the color palette to "Purples"

sns.set\_style("whitegrid")

sns.set\_palette('Purples')

# 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()

# Set the color palette to "Purples" sns.set\_style("whitegrid") sns.set\_palette('Purples') # 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()

# Change the color palette to "RdBu"

sns.set\_style("whitegrid")

sns.set\_palette("RdBu")

# 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()

# Change the color palette to "RdBu" sns.set\_style("whitegrid") sns.set\_palette("RdBu") # 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()

Awesome work. This style and diverging color palette best highlights the difference between the number of young people who usually listen to their parents' advice versus those who don't.

# Changing the scale

In this exercise, we'll continue to look at the dataset containing responses from a survey of young people. Does the percentage of people reporting that they feel lonely vary depending on how many siblings they have? Let's find out using a bar plot, while also exploring Seaborn's four different plot scales ("contexts").

We've already imported Seaborn as sns and matplotlib.pyplot as plt.

##### Instructions 1/4

[1](javascript:void(0))Set the scale ("context") to "paper", which is the smallest of the scale options.

[2](javascript:void(0))Change the context to "notebook" to increase the scale.

[3](javascript:void(0))Change the context to "talk" to increase the scale.

[4](javascript:void(0))Change the context to "poster", which is the largest scale available.

# Set the context to "paper"

# Create bar plot

sns.catplot(x="Number of Siblings", y="Feels Lonely",

            data=survey\_data, kind="bar")

# Show plot

plt.show()

# Set the context to "paper" sns.set\_context('paper') # Create bar plot sns.catplot(x="Number of Siblings", y="Feels Lonely", data=survey\_data, kind="bar") # Show plot plt.show()

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

# Change the context to "notebook"

sns.set\_context("paper")

# Create bar plot

sns.catplot(x="Number of Siblings", y="Feels Lonely",

            data=survey\_data, kind="bar")

# Show plot

plt.show()

# Change the context to "notebook"

sns.set\_context("notebook")

# Create bar plot

sns.catplot(x="Number of Siblings", y="Feels Lonely",

            data=survey\_data, kind="bar")

# Show plot

plt.show()

# Change the context to "notebook" sns.set\_context("notebook") # Create bar plot sns.catplot(x="Number of Siblings", y="Feels Lonely", data=survey\_data, kind="bar") # Show plot plt.show()

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

 Change the context to "talk"

sns.set\_context("talk")

# Create bar plot

sns.catplot(x="Number of Siblings", y="Feels Lonely",

            data=survey\_data, kind="bar")

# Show plot

plt.show()

# Change the context to "talk" sns.set\_context("talk") # Create bar plot sns.catplot(x="Number of Siblings", y="Feels Lonely", data=survey\_data, kind="bar") # Show plot plt.show()

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

# Change the context to "poster"

sns.set\_context("poster")

# Create bar plot

sns.catplot(x="Number of Siblings", y="Feels Lonely",

            data=survey\_data, kind="bar")

# Show plot

plt.show()

# Change the context to "poster" sns.set\_context("poster") # Create bar plot sns.catplot(x="Number of Siblings", y="Feels Lonely", data=survey\_data, kind="bar") # Show plot plt.show()

Great job! Each context name gives Seaborn's suggestion on when to use a given plot scale (in a paper, in an iPython notebook, in a talk/presentation, or in a poster session).

# Using a custom palette

So far, we've looked at several things in the dataset of survey responses from young people, including their internet usage, how often they listen to their parents, and how many of them report feeling lonely. However, one thing we haven't done is a basic summary of the type of people answering this survey, including their age and gender. Providing these basic summaries is always a good practice when dealing with an unfamiliar dataset.

The code provided will create a box plot showing the distribution of ages for male versus female respondents. Let's adjust the code to customize the appearance, this time using a custom color palette.

We've already imported Seaborn as sns and matplotlib.pyplot as plt.

##### Instructions

* Set the style to "darkgrid".
* Set a custom color palette with the hex color codes "#39A7D0" and "#36ADA4".
* # Set the style to "darkgrid"
* # Set a custom color palette
* # Create the box plot of age distribution by gender
* sns.catplot(x="Gender", y="Age",
* data=survey\_data, kind="box")
* # Show plot
* plt.show()

# Set the style to "darkgrid"

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()

# Set the style to "darkgrid" 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()

Good work! It looks like the median age is the same for males and females, but distribution of females skews younger than the males.

**Adding titles and labels: Part 1**

**50 XP**

**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.

**2. 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.

**3. FacetGrid vs. AxesSubplot objects**

Before we go into the details of adding a title, we need to understand an 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.

**4. An Empty FacetGrid**

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

**5. 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.

**6. 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 dot fig dot suptitle". This tells Seaborn you want to set a title for the figure as a whole.

**7. 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 point 03 will make it a little higher than the default.

**8. Let's practice!**

We'll learn how to add a title to an AxesSubplot object in the next lesson. For now, let's pause and practice what you just learned!

# FacetGrids vs. AxesSubplots

In the recent lesson, we learned that Seaborn plot functions create two different types of objects: FacetGrid objects and AxesSubplot objects. The method for adding a title to your plot will differ depending on the type of object it is.

In the code provided, we've used relplot() with the miles per gallon dataset to create a scatter plot showing the relationship between a car's weight and its horsepower. This scatter plot is assigned to the variable name g. Let's identify which type of object it is.

We've already imported Seaborn as sns and matplotlib.pyplot as plt.

##### Instructions 1/2

Identify what type of object plot g is and assign it to the variable type\_of\_g.

# Create scatter plot

g = sns.relplot(x="weight",

                y="horsepower",

                data=mpg,

                kind="scatter")

# Identify plot type

type\_of\_g = \_\_\_\_

# Print type

print(type\_of\_g)

# 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(type\_of\_g)

# 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(type\_of\_g)

<class 'seaborn.axisgrid.FacetGrid'>

#### Question

We've just seen that sns.relplot() creates FacetGrid objects. Which other Seaborn function creates a FacetGrid object instead of an AxesSubplot object?

##### Possible Answers

* **sns.catplot()**
* sns.scatterplot()
* sns.boxplot()
* sns.countplot()

Great job. catplot() supports creating subplots, so it creates a FacetGrid object.

# Adding a title to a FacetGrid object

In the previous exercise, we used relplot() with the miles per gallon dataset to create a scatter plot showing the relationship between a car's weight and its horsepower. This created a FacetGrid object. Now that we know what type of object it is, let's add a title to this plot.

We've already imported Seaborn as sns and matplotlib.pyplot as plt.

##### Instructions

Add the following title to this plot: "Car Weight vs. Horsepower".

# Create scatter plot

g = sns.relplot(x="weight",

                y="horsepower",

                data=mpg,

                kind="scatter")

# Add a title "Car Weight vs. Horsepower"

# Show plot

plt.show()

# Create scatter plot

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()

# Create scatter plot 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()

Good work! It looks like a car's weight is positively correlated with its horsepower.

**Adding titles and labels: Part 2**

**50 XP**

**1. Adding titles and labels: Part 2**

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

**2. Adding a title to AxesSubplot**

In the last lesson, we learned how to add a title to a FacetGrid object using "g dot fig dot 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 dot set\_title”. You can also use the “y” parameter here to adjust the height of the title.

**3. 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.

**4. Titles for subplots**

Since g is a FacetGrid object, using "g dot fig dot suptitle" will add a title to the figure as a whole.

**5. Titles for subplots**

To alter the subplot titles, use "g dot 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".

**6. 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.

**7. 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 dot xticks" and set "rotation" equal to 90 degrees. This works with both FacetGrid and AxesSubplot objects.

**8. Let's practice!**

And that's it! Now it's time to create some clear and informative visualizations!

# Adding a title and axis labels

Let's continue to look at the miles per gallon dataset. This time we'll create a line plot to answer the question: How does the average miles per gallon achieved by cars change over time for each of the three places of origin? To improve the readability of this plot, we'll add a title and more informative axis labels.

In the code provided, we create the line plot using the lineplot() function. Note that lineplot() does not support the creation of subplots, so it returns an AxesSubplot object instead of an FacetGrid object.

We've already imported Seaborn as sns and matplotlib.pyplot as plt.

##### Instructions 1/2

Add the following title to the plot: "Average MPG Over Time".

# Create line plot

g = sns.lineplot(x="model\_year", y="mpg\_mean",

                 data=mpg\_mean,

                 hue="origin")

# Add a title "Average MPG Over Time"

# Show plot

plt.show()

# Create line plot

g = sns.lineplot(x="model\_year", y="mpg\_mean",

                 data=mpg\_mean,

                 hue="origin")

# Add a title "Average MPG Over Time"

g.set\_title('Average MPG Over Time')

# Show plot

plt.show()

# Create line plot g = sns.lineplot(x="model\_year", y="mpg\_mean", data=mpg\_mean, hue="origin") # Add a title "Average MPG Over Time" g.set\_title('Average MPG Over Time') # Show plot plt.show()

Label the x-axis as "Car Model Year" and the y-axis as "Average MPG".

# Create line plot

g = sns.lineplot(x="model\_year", y="mpg\_mean",

                 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

# Show plot

plt.show()

# Create line plot

g = sns.lineplot(x="model\_year", y="mpg\_mean",

                 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()

# Create line plot g = sns.lineplot(x="model\_year", y="mpg\_mean", 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()

Awesome. The average miles per gallon achieved is increasing over time for all three places of origin, but the USA is always lower than Europe and Japan.

# Rotating x-tick labels

In this exercise, we'll continue looking at the miles per gallon dataset. In the code provided, we create a point plot that displays the average acceleration for cars in each of the three places of origin. Note that the "acceleration" variable is the time to accelerate from 0 to 60 miles per hour, in seconds. Higher values indicate slower acceleration.

Let's use this plot to practice rotating the x-tick labels. Recall that the function to rotate x-tick labels is a standalone Matplotlib function and not a function applied to the plot object itself.

We've already imported Seaborn as sns and matplotlib.pyplot as plt.

##### Instructions

Rotate the x-tick labels 90 degrees.

# Create point plot

sns.catplot(x="origin",

            y="acceleration",

            data=mpg,

            kind="point",

            join=False,

            capsize=0.1)

# Rotate x-tick labels

# Show plot

plt.show()

# Create point plot sns.catplot(x="origin", y="acceleration", data=mpg, kind="point", join=False, capsize=0.1) # Rotate x-tick labels plt.xticks(rotation=90) # Show plot plt.show()

**Fantastic. Since higher values indicate slower acceleration, it looks like cars from Japan and Europe have significantly slower acceleration compares to the USA.**

**Putting it all together**

**50 XP**

**1. Putting it all together**

In this course, we've learned a great deal about how to create effective data visualizations in Seaborn. In this lesson, we'll review what we've learned and connect the pieces together to form a cohesive picture of how to use Seaborn for future projects.

**2. Getting started**

The first thing to recall is simply how to import Seaborn and its related library, Matplotlib. To do this, write "import seaborn as sns" and "import matplotlib dot pyplot as plt". Recall also that at the end of your data visualization code, you'll call "plt dot show" to show the visualization.

**3. Relational plots**

After you've imported the appropriate libraries, the next thing to do is to choose what type of plot you want to create. Relational plots are plots that show the relationship between two quantitative variables. Examples of relational plots that we've seen in this course are scatter plots and line plots. You can create a relational plot using "relplot()" and providing it with the x-axis variable name, y-axis variable name, the pandas tidy DataFrame, and the type of plot (either scatter or line).

**4. Categorical plots**

Categorical plots are another type of plot. These describe the distribution of a quantitative variable within categories given by a categorical variable. Examples of categorical plots we've seen are bar plots, count plots, box plots, and point plots. You can create a categorical plot using "catplot()" and providing it with the x-axis variable name, y-axis variable name (if applicable), the pandas tidy DataFrame, and the type of plot (either bar, count, box, or point).

**5. Adding a third variable (hue)**

If we want to add a third dimension to our plots, we can do this in one of two ways. Setting the "hue" parameter to a variable name will create a single plot but will show subgroups that are different colors based on that variable's values.

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

**6. Adding a third variable (row/col)**

Alternatively, you can use "relplot()" and "catplot()"’s "col" and "row" parameters to graph each subgroup on a separate subplot in the figure.

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

**7. Customization**

Once you have the basic plot created, you might want to customize the plot's appearance to improve its readability. You can change the background of the plot using "set\_style", the color of the main elements using "set\_palette", and the scale of the plot using "set\_context".

**8. Adding a title**

Finally, every plot should be given an informative title and axis labels. Recall the two types of plot objects - FacetGrids and AxesSubplots - and the way to add a title to each of them.

**9. Final touches**

Also recall how to use the "set" function with the "xlabel" and "ylabel" parameters to provide custom x- and y-axis labels, and how to use "plt.xticks" with the "rotation" parameter to rotate the x-tick labels.

**10. Let's practice!**

And that's it! You're now equipped to make impressive and effective data visualizations with Seaborn. Let's practice putting all of these steps together in the final exercises of this course.

# Box plot with subgroups

In this exercise, we'll look at the dataset containing responses from a survey given to young people. One of the questions asked of the young people was: "Are you interested in having pets?" Let's explore whether the distribution of ages of those answering "yes" tends to be higher or lower than those answering "no", controlling for gender.

##### Instructions

* Set the color palette to "Blues".
* Add subgroups to color the box plots based on "Interested in Pets".
* Set the title of the FacetGrid object g to "Age of Those Interested in Pets vs. Not".
* Make the plot display using a Matplotlib function.
* # Set palette to "Blues"
* \_\_\_\_
* # Adjust to add subgroups based on "Interested in Pets"
* g = sns.catplot(x="Gender",
* y="Age", data=survey\_data,
* kind="box", hue=\_\_\_\_)
* # Set title to "Age of Those Interested in Pets vs. Not"
* \_\_\_\_
* # Show plot
* plt.\_\_\_\_
* # 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='Interested in 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()

# 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='Interested in 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()

Good job. After controlling for gender, it looks like the age distributions of people who are interested in pets are similar than those who aren't.

# Bar plot with subgroups and subplots

In this exercise, we'll return to our young people survey dataset and investigate whether the proportion of people who like techno music ("Likes Techno") varies by their gender ("Gender") or where they live ("Village - town"). This exercise will give us an opportunity to practice the many things we've learned throughout this course!

We've already imported Seaborn as sns and matplotlib.pyplot as plt.

##### Instructions

* Set the figure style to "dark".
* Adjust the bar plot code to add subplots based on "Gender", arranged in columns.
* Add the title "Percentage of Young People Who Like Techno" to this FacetGrid plot.
* Label the x-axis "Location of Residence" and y-axis "% Who Like Techno".
* # Set the figure style to "dark"
* sns.set\_style('dark')
* # Adjust to add subplots per gender
* g = sns.catplot(x="Village - town", y="Likes Techno",
* data=survey\_data, kind="bar",
* hue='Gender', col='Gender')
* # Add title and axis labels
* g.fig.suptitle("Percentage of Young People Who Like Techno", y=1.02)
* g.set(xlabel="Location of Residence",
* ylabel="% Who Like Techno")
* # Show plot
* plt.show()
* # Show plot
* plt.show()
* # Set the figure style to "dark"
* sns.set\_style('dark')
* # Adjust to add subplots per gender
* g = sns.catplot(x="Village - town", y="Likes Techno",
* data=survey\_data, kind="bar",
* hue='Gender', col='Gender')
* # Add title and axis labels
* g.fig.suptitle("Percentage of Young People Who Like Techno", y=1.02)
* g.set(xlabel="Location of Residence",
* ylabel="% Who Like Techno")
* # Show plot
* plt.show()

Amazing work! I hope you enjoyed this introductory course about Seaborn!

**Well done! What's next?**

**50 XP**

**1. Well done! What's next?**

Congratulations on completing this introduction to Seaborn! Let's discuss the next steps you can take to build upon the skills that you've learned in this course.

**2. Where does Seaborn fit in?**

Seaborn is a powerful data visualization tool that allows you to create attractive and informative visualizations with just a few lines of code. Let's return to this diagram of the data analysis workflow to see where Seaborn fits in.

**3. Where does Seaborn fit in?**

As we've seen in our examples, Seaborn is great for both the initial exploration of your data and communicating the results at the end of your data analysis.

**4. Next Steps: Explore and communicate results**

In this course, we've covered the most common data visualizations used for data exploration. DataCamp has other visualization courses if you want to learn even more. For example, Seaborn also supports more advanced visualizations and analyses like linear regressions. We also learned that Seaborn was built on top of Matplotlib and practiced how to use some Matplotlib functions to customize Seaborn plots. Here, too, there are many more customizations that Matplotlib supports if you wish to learn more.

**5. Next steps: Gather data**

You can also learn more about the other steps of the data analysis workflow. If you wish to learn more about how to gather your data, explore courses on importing data in Python and SQL.

**6. Next steps: Transform and clean**

In this course, we learned that Seaborn works extremely well with tidy pandas DataFrames. There is more to learn here about how to get your data into pandas DataFrames, clean it, and transform it into a tidy format.

**7. Next steps: Analyze and build models**

Finally, I encourage you to learn more about statistical analysis. For example, for bar plots, Seaborn automatically calculates confidence intervals for each bar value. There is a lot to learn here about how these confidence intervals are calculated and how to interpret them.

**8. Congratulations!**

Though there is always more to learn, we've covered a great deal in this introduction to Seaborn. Congratulations on completing the course! I hope you enjoyed it and feel confident using Seaborn in the future for your data visualization needs.