**Introduction to Seaborn**

In this lesson, you'll learn how to use Seaborn to create bar charts for statistical analysis.

Seaborn is a Python data visualization library that provides simple code to create elegant visualizations for statistical exploration and insight. Seaborn is based on Matplotlib, but improves on Matplotlib in several ways:

* Seaborn provides a more visually appealing plotting style and concise syntax.
* Seaborn natively understands Pandas DataFrames, making it easier to plot data directly from CSVs.
* Seaborn can easily summarize Pandas DataFrames with many rows of data into aggregated charts.

If you're unfamiliar with Pandas, just know that Pandas is a data analysis library for Python that provides easy-to-use data structures and allows you to organize and manipulate datasets so they can be visualized. To fully leverage the power of Seaborn, it is best to prepare your data using Pandas.

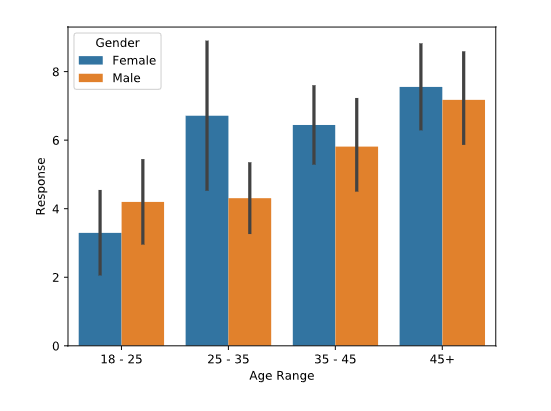
Over the next few exercises, we will explain how Seaborn relates to Pandas and how we can transform massive datasets into easily understandable graphics.

**1.**

The file **script.py** contains code to create a Seaborn visualization. Paste the following code at the very top of **script.py** to import Seaborn so that the code can run successfully:

import seaborn as sns

import codecademylib3\_seaborn  
import pandas as pd  
from matplotlib import pyplot as plt  
  
# Paste import here:  
import seaborn as sns  
  
df = pd.read\_csv('survey.csv')  
sns.barplot(x='Age Range', y='Response', hue='Gender', data=df)  
plt.show()



# Using Pandas For Seaborn

Throughout this lesson, you'll use Seaborn to visualize a Pandas DataFrame.

DataFrames contain data structured into rows and columns. DataFrames look similar to other data tables you may be familiar with, but they are designed specifically to be used with Python.

You can create a DataFrame from a local CSV file (CSV files store data in a tabular format).

To create a DataFrame from a local CSV file you would use the syntax:

df = pd.read\_csv('file\_name.csv')

The code above creates a DataFrame saved to a variable named df. The data inside of the df DataFrame comes from the data in the local CSV file named file\_name.csv.

Once you have prepared and organized a Pandas DataFrame with your chosen dataset, you are ready to plot with Seaborn!

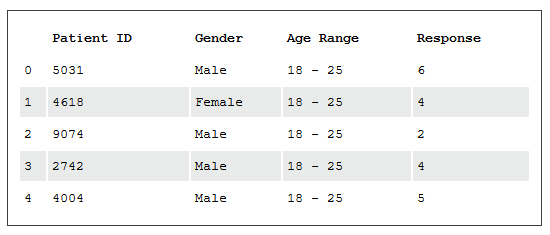
Instructions

**1.**

In **script.py** you can see pd.read\_csv() is used to ingest the data stored in a file named **survey.csv**. If you'd like, you can inspect the contents of **survey.csv** in the file system of your workspace. We will explain the context of **survey.csv** in more detail in the next exercise. For now, focus on the syntax used to create a DataFrame from a CSV file.

Inspect the DataFrame by printing the first 5 rows of df.

import codecademylib3\_seaborn  
import pandas as pd  
from matplotlib import pyplot as plt  
import seaborn as sns  
  
  
df = pd.read\_csv('survey.csv')  
print(df.head())



**Plotting Bars with Seaborn**

Take a look at the file called **results.csv**. You'll plot that data soon, but before you plot it, take a minute to understand the context behind that data, which is based on a hypothetical situation we have created:

Suppose we are analyzing data from a survey: we asked 1,000 patients at a hospital how satisfied they were with their experience. They response was measured on a scale of 1 - 10, with 1 being extremely unsatisfied, and 10 being extremely satisfied. We have summarized that data in a CSV file called **results.csv**.

To plot this data using Matplotlib, you would write the following:

df = pd.read\_csv("results.csv")

ax = plt.subplot()

plt.bar(range(len(df)),

df["Mean Satisfaction"])

ax.set\_xticks(range(len(df)))

ax.set\_xticklabels(df.Gender)

plt.xlabel("Gender")

plt.ylabel("Mean Satisfaction")

That's a lot of work for a simple bar chart! Seaborn gives us a much simpler option. With Seaborn, you can use the sns.barplot() command to do the same thing.

The Seaborn function sns.barplot(), takes at least three keyword arguments:

* data: a Pandas DataFrame that contains the data (in this example, data=df)
* x: a string that tells Seaborn which column in the DataFrame contains otheur x-labels (in this case, x="Gender")
* y: a string that tells Seaborn which column in the DataFrame contains the heights we want to plot for each bar (in this case y="Mean Satisfaction")

By default, Seaborn will aggregate and plot the *mean* of each category. In the next exercise you will learn more about aggregation and how Seaborn handles it.

Instructions

**1.**

Use Pandas to load in the data from **results.csv** and save it to the variable df.

**2.**

Display df using print

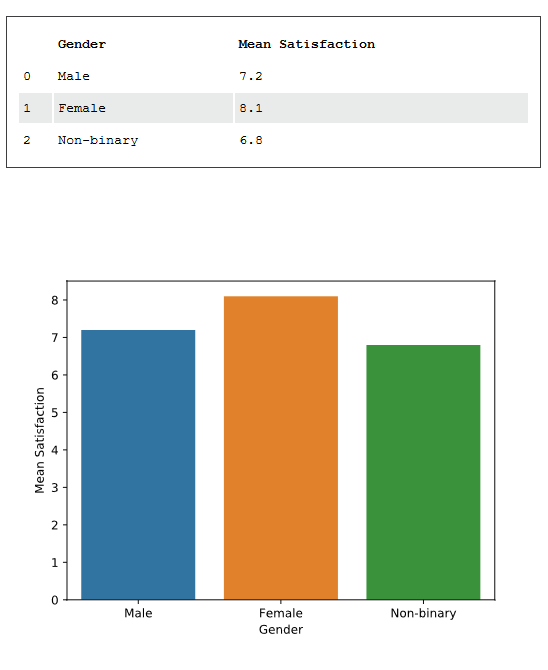
**3.**

Remove all of the # characters from in front of the sns.barplot command and fill in the missing values.

**4.**

Type plt.show() to display the completed bar plot.

import codecademylib3\_seaborn  
import pandas as pd  
from matplotlib import pyplot as plt  
import seaborn as sns  
  
# Load results.csv here:  
df = pd.read\_csv('results.csv')  
print(df)  
  
sns.barplot(  
 data= df,  
 x= "Gender",  
 y= "Mean Satisfaction"  
 )  
plt.show()



**Understanding Aggregates**

Seaborn can also calculate *aggregate statistics* for large datasets. To understand why this is helpful, we must first understand what an *aggregate* is.

An aggregate statistic, or aggregate, is a single number used to describe a set of data. One example of an aggregate is the average, or *mean* of a data set. There are many other aggregate statistics as well.

Suppose we have a grade book with columns student, assignment\_name, and grade, as shown below.

| **student** | **assignment\_name** | **grade** |
| --- | --- | --- |
| Amy | Assignment 1 | 75 |
| Amy | Assignment 2 | 82 |
| Bob | Assignment 1 | 99 |
| Bob | Assignment 2 | 90 |
| Chris | Assignment 1 | 72 |
| Chris | Assignment 2 | 66 |
| ... | ... | ... |

To calculate a student's current grade in the class, we need to *aggregate* the grade data by student. To do this, we'll calculate the average of each student's grades, resulting in the following data set:

| **student** | **grade** |
| --- | --- |
| Amy | 78.5 |
| Bob | 94.5 |
| Chris | 69 |
| ... | ... |

On the other hand, we may be interested in understanding the relative difficulty of each assignment. In this case, we would aggregate by assignment, taking the average of all student's scores on each assignment:

| **assignment\_name** | **grade** |
| --- | --- |
| Assignment 1 | 82 |
| Assignment 2 | 79.3 |
| ... | ... |

In both of these cases, the function we used to aggregate our data was the average or mean, but there are many types of aggregate statistics including:

* Median
* Mode
* Standard Deviation

In Python, you can compute aggregates fairly quickly and easily using Numpy, a popular Python library for computing. You'll use Numpy in this exercise to compute aggregates for a DataFrame.

**1.**

To calculate aggregates using Numpy, you'll first need to import the Numpy library at the top of **script.py**.

Type the following at the top of your file:

import numpy as np

**2.**

Next, take a minute to understand the data you'll analyze. The DataFrame gradebook contains the complete gradebook for a hypothetical classroom. Use print to examine gradebook.

**3.**

Select all rows from the gradebook DataFrame where assignment\_name is equal to Assignment 1. Save the result to the variable assignment1.

**4.**

Check out the DataFrame you just created. Print assignment1.

**5.**

Now use Numpy to calculate the median grade in assignment1.

Use np.median() to calculate the median of the column grade from assignment1 and save it to asn1\_median.

**6.**

Display asn1\_median using print. What is the median grade on Assignment 1?

import codecademylib3\_seaborn  
import pandas as pd  
from matplotlib import pyplot as plt  
import numpy as np  
  
gradebook = pd.read\_csv("gradebook.csv")  
print(gradebook)  
  
assignment1 = gradebook[gradebook.assignment\_name=="Assignment 1"]  
print(assignment1)  
asn1\_median = np.median(assignment1.grade)  
print(asn1\_median)  
'''  
 student assignment\_name grade  
0 Amy Assignment 1 75  
1 Amy Assignment 2 82  
2 Bob Assignment 1 99  
3 Bob Assignment 2 90  
4 Chris Assignment 1 72  
5 Chris Assignment 2 66  
6 Dan Assignment 1 88  
7 Dan Assignment 2 82  
8 Ellie Assignment 1 91  
9 Ellie Assignment 2 85  
  
 student assignment\_name grade  
0 Amy Assignment 1 75  
2 Bob Assignment 1 99  
4 Chris Assignment 1 72  
6 Dan Assignment 1 88  
8 Ellie Assignment 1 91  
  
88.0  
'''

# Plotting Aggregates

Recall our gradebook from the previous exercise:

| **student** | **assignment\_name** | **grade** |
| --- | --- | --- |
| Amy | Assignment 1 | 75 |
| Amy | Assignment 2 | 82 |
| Bob | Assignment 1 | 99 |
| Bob | Assignment 2 | 90 |
| Chris | Assignment 1 | 72 |
| Chris | Assignment 2 | 66 |
| ... | ... | ... |

Suppose this data is stored in a Pandas DataFrame called df.

The same Seaborn command that you previously learned (sns.barplot()) will plot this data in a bar plot and automatically aggregate the data:

sns.barplot(data=df, x="student", y="grade")

In the example above, Seaborn will aggregate grades by student, and plot the average grade for each student.

Instructions

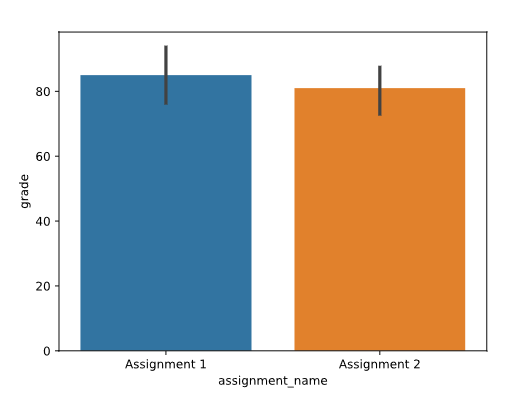
**1.**

Use Seaborn to plot the average grade for each assignment. Take a look at **gradebook.csv** for the column names.

**2.**

Use plt.show() to display the graph.

import codecademylib3\_seaborn  
import pandas as pd  
from matplotlib import pyplot as plt  
import seaborn as sns  
  
gradebook = pd.read\_csv("gradebook.csv")  
  
sns.barplot(data=gradebook, x="assignment\_name", y="grade")  
  
plt.show()



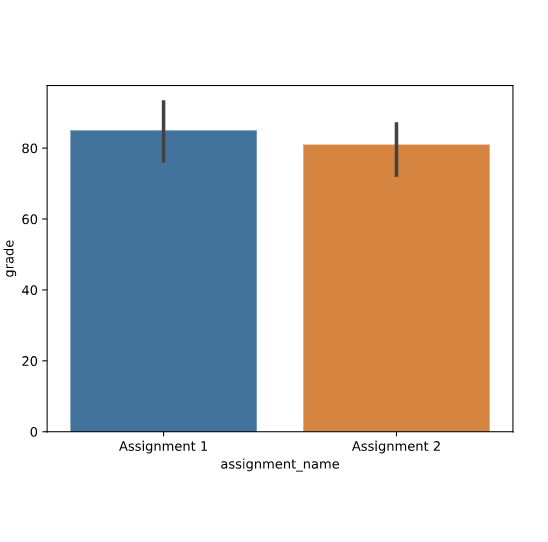
# Modifying Error Bars

By default, Seaborn will place error bars on each bar when you use the barplot() function.

Error bars are the small lines that extend above and below the top of each bar. Errors bars visually indicate the range of values that might be expected for that bar.



For example, in our assignment average example, an error bar might indicate what grade we expect an average student to receive on this assignment.



There are several different calculations that are commonly used to determine error bars.

By default, Seaborn uses something called a bootstrapped confidence interval. Roughly speaking, this interval means that "based on this data, 95% of similar situations would have an outcome within this range".

In our gradebook example, the confidence interval for the assignments means "if we gave this assignment to many, many students, we're confident that the mean score on the assignment would be within the range represented by the error bar".

The confidence interval is a nice error bar measurement because it is defined for different types of aggregate functions, such as medians and mode, in addition to means.

If you're calculating a mean and would prefer to use standard deviation for your error bars, you can pass in the keyword argument ci="sd" to sns.barplot() which will represent one standard deviation. It would look like this:

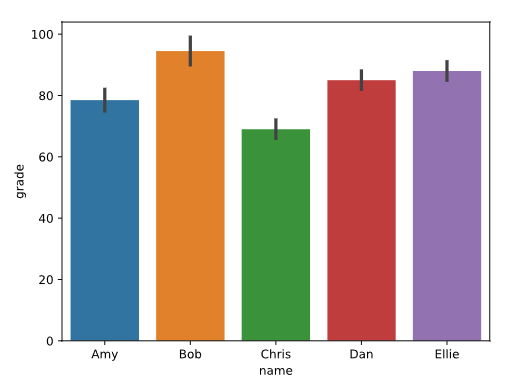
sns.barplot(data=gradebook, x="name", y="grade", ci="sd")

Instructions

**1.**

Modify the bar plot so that the error bars represent one standard deviation, rather than 95% confidence intervals.

import codecademylib3\_seaborn  
import pandas as pd  
from matplotlib import pyplot as plt  
import seaborn as sns  
  
gradebook = pd.read\_csv("gradebook.csv")  
  
sns.barplot(data=gradebook,  
 x="name",  
 y="grade",  
 ci="sd")  
plt.show()



**Calculating Different Aggregates**

In most cases, we'll want to plot the mean of our data, but sometimes, we'll want something different:

* If our data has many outliers, we may want to plot the *median*.
* If our data is categorical, we might want to count how many times each category appears (such as in the case of survey responses).

Seaborn is flexible and can calculate any aggregate you want. To do so, you'll need to use the keyword argument estimator, which accepts any function that works on a list.

For example, to calculate the median, you can pass in np.median to the estimator keyword:

sns.barplot(data=df,

x="x-values",

y="y-values",

estimator=np.median)

Consider the data in **results.csv**. To calculate the number of times a particular value appears in the Response column , we pass in len:

sns.barplot(data=df,

x="Patient ID",

y="Response",

estimator=len)

**1.**

Consider our hospital satisfaction survey data, which is loaded into the Pandas DataFrame df. Use print to examine the data.

**2.**

We'd like to know how many men and women answered the survey. Use sns.barplot() with:

* data equal to df
* x equal to Gender
* y equal to Response
* estimator equal to len

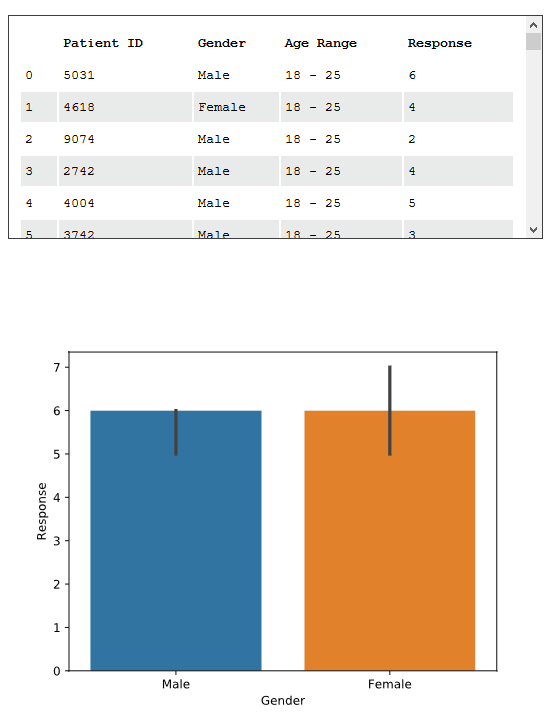
**3.**

Use plt.show() to display the graph.

**4.**

Change sns.barplot() to graph the median Response aggregated by Gender using estimator=np.median.

import codecademylib3\_seaborn  
import warnings  
warnings.filterwarnings('ignore')  
import numpy as np  
import pandas as pd  
from matplotlib import pyplot as plt  
import seaborn as sns  
  
df = pd.read\_csv("survey.csv")  
print(df)  
sns.barplot(data=df, x="Gender", y="Response", estimator=np.median)  
plt.show()



**Aggregating by Multiple Columns**

Sometimes we'll want to aggregate our data by multiple columns to visualize nested categorical variables.

For example, consider our hospital survey data. The mean satisfaction seems to depend on Gender, but it might also depend on another column: Age Range.

We can compare both the Gender and Age Range factors at once by using the keyword hue.

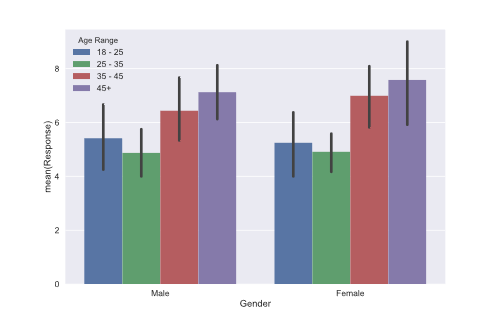
sns.barplot(data=df,

x="Gender",

y="Response",

hue="Age Range")

The hue parameter adds a nested categorical variable to the plot.

**

*Visualizing survey results by gender with age range nested*.

Notice that we keep the same x-labels, but we now have different color bars representing each Age Range. We can compare two bars of the same color to see how patients with the same Age Range, but different Gender rated the survey.

Instructions

**1.**

Use sns.barplot() to create a chart with:

* data equal to df
* x equal to Age Range
* y equal to Response
* hue equal to Gender

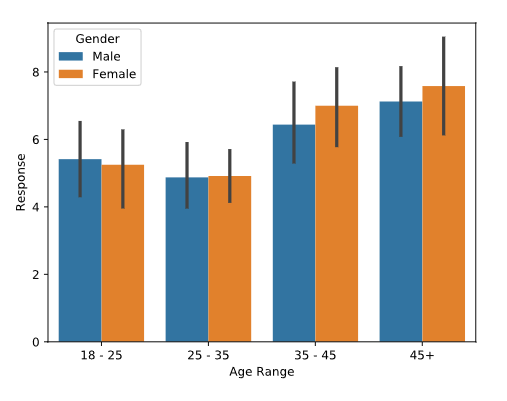
How is this plot different from when hue is "Age Range" and x is "Gender"?

Why might we use one and not the other?

**2.**

Use plt.show() to display the graph.

import codecademylib3\_seaborn  
import pandas as pd  
from matplotlib import pyplot as plt  
import seaborn as sns  
  
df = pd.read\_csv("survey.csv")  
  
sns.barplot(data=df,  
 x="Age Range",  
 y="Response",  
 hue="Gender")  
  
plt.show()



# Review

In this lesson you learned how to extend Matplotlib with Seaborn to create meaningful visualizations from data in DataFrames.

You've also learned how Seaborn creates aggregated charts and how to change the way aggregates and error bars are calculated.

Finally, you learned how to aggregate by multiple columns, and how the hue parameter adds a nested categorical variable to a visualization.

#### To review the seaborn workflow:

##### 1. Ingest data from a CSV file to Pandas DataFrame.

df = pd.read\_csv('file\_name.csv')

##### 2. Set sns.barplot() with desired values for x, y, and set data equal to your DataFrame.

sns.barplot(data=df, x='X-Values', y='Y-Values')

##### 3. Set desired values for estimator and hue parameters.

sns.barplot(data=df, x='X-Values', y='Y-Values', estimator=len, hue='Value')

##### 4. Render the plot using plt.show().

plt.show()

Instructions

Examine the Seaborn graphs in **script.py**. Use this space to practice and modify the graphs.

import codecademylib3\_seaborn  
import pandas as pd  
from matplotlib import pyplot as plt  
import seaborn as sns  
  
df = pd.read\_csv("survey.csv")  
  
sns.barplot(data=df, x="Gender", y="Patient ID", hue="Age Range")  
  
sns.barplot(data=survey, x="Age Range", y="Patient ID", hue="Gender")  
  
plt.show()

