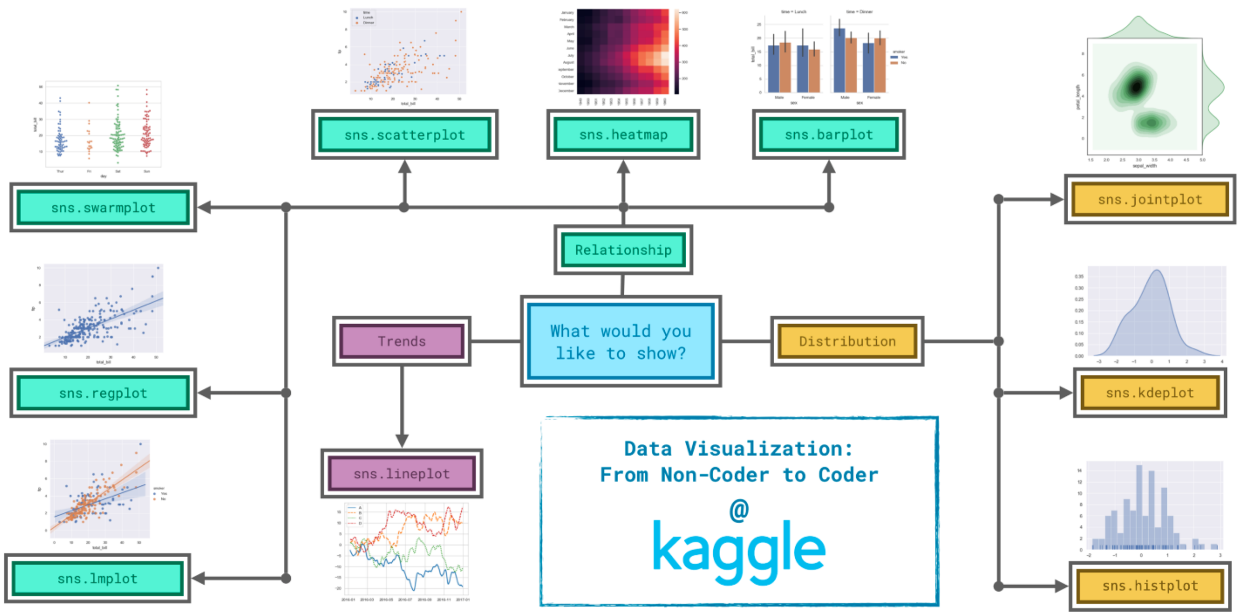
Data Visualization

https://www.kaggle.com/code

https://www.kaggle.com/datasets



Since it's not always easy to decide how to best tell the story behind your data, we've broken the chart types into three broad categories to help with this.

- **Trends** - A trend is defined as a pattern of change.
  - `sns.lineplot` - **Line charts** are best to show trends over a period of time, and multiple lines can be used to show trends in more than one group.
- **Relationship** - There are many different chart types that you can use to understand relationships between variables in your data.
  - `sns.barplot` - **Bar charts** are useful for comparing quantities corresponding to different groups.
  - `sns.heatmap` - **Heatmaps** can be used to find color-coded patterns in tables of numbers.
  - `sns.scatterplot` - **Scatter plots** show the relationship between two continuous variables; if color-coded, we can also show the relationship with a third categorical variable.
  - `sns.regplot` - Including a **regression line** in the scatter plot makes it easier to see any linear relationship between two variables.
  - `sns.lmplot` - This command is useful for drawing multiple regression lines, if the scatter plot contains multiple, color-coded groups.
  - `sns.swarmplot` - **Categorical scatter plots** show the relationship between a continuous variable and a categorical variable.

- **Distribution** - We visualize distributions to show the possible values that we can expect to see in a variable, along with how likely they are.
    - `sns.histplot` - **Histograms** show the distribution of a single numerical variable.
    - `sns.kdeplot` - **KDE plots** (or **2D KDE plots**) show an estimated, smooth distribution of a single numerical variable (or two numerical variables).
    - `sns.jointplot` - This command is useful for simultaneously displaying a 2D KDE plot with the corresponding KDE plots for each individual variable.

Setting up the notebook:

```
import pandas as pd
pd.plotting.register_matplotlib_converters()
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
print("Setup Complete")
```

Load the data

```
# Path of the file to read
fifa_filepath = "../input/fifa.csv"

# Read the file into a variable fifa_data
fifa_data = pd.read_csv(fifa_filepath, index_col="Date", parse_dates=True)
```

Examine the data

```
# Print the first 5 rows of the data
fifa_data.head()
```

Lineplot

```
# Set the width and height of the figure
plt.figure(figsize=(16,6))

# Line chart showing how FIFA rankings evolved over time
sns.lineplot(data=fifa_data)
```

For instance, we use `sns.lineplot` to make line charts. Soon, you'll learn that we use `sns.barplot` and `sns.heatmap` to make bar charts and heatmaps, respectively.

Plot a subset of the data

```
list(spotify_data.columns)
# Set the width and height of the figure
plt.figure(figsize=(14,6))

# Add title
plt.title("Daily Global Streams of Popular Songs in 2017-2018")

# Line chart showing daily global streams of 'Shape of You'
sns.lineplot(data=spotify_data['Shape of You'], label="Shape of You")

# Line chart showing daily global streams of 'Despacito'
sns.lineplot(data=spotify_data['Despacito'], label="Despacito")

# Add label for horizontal axis
plt.xlabel("Date")
```

Setting theme:

```
# Change the style of the figure to the "dark" theme
sns.set_style("dark")
```

Seaborn has five different themes: (1)`"darkgrid"`, (2)`"whitegrid"`, (3)`"dark"`, (4)`"white"`, and (5)`"ticks"`,

Bar chart

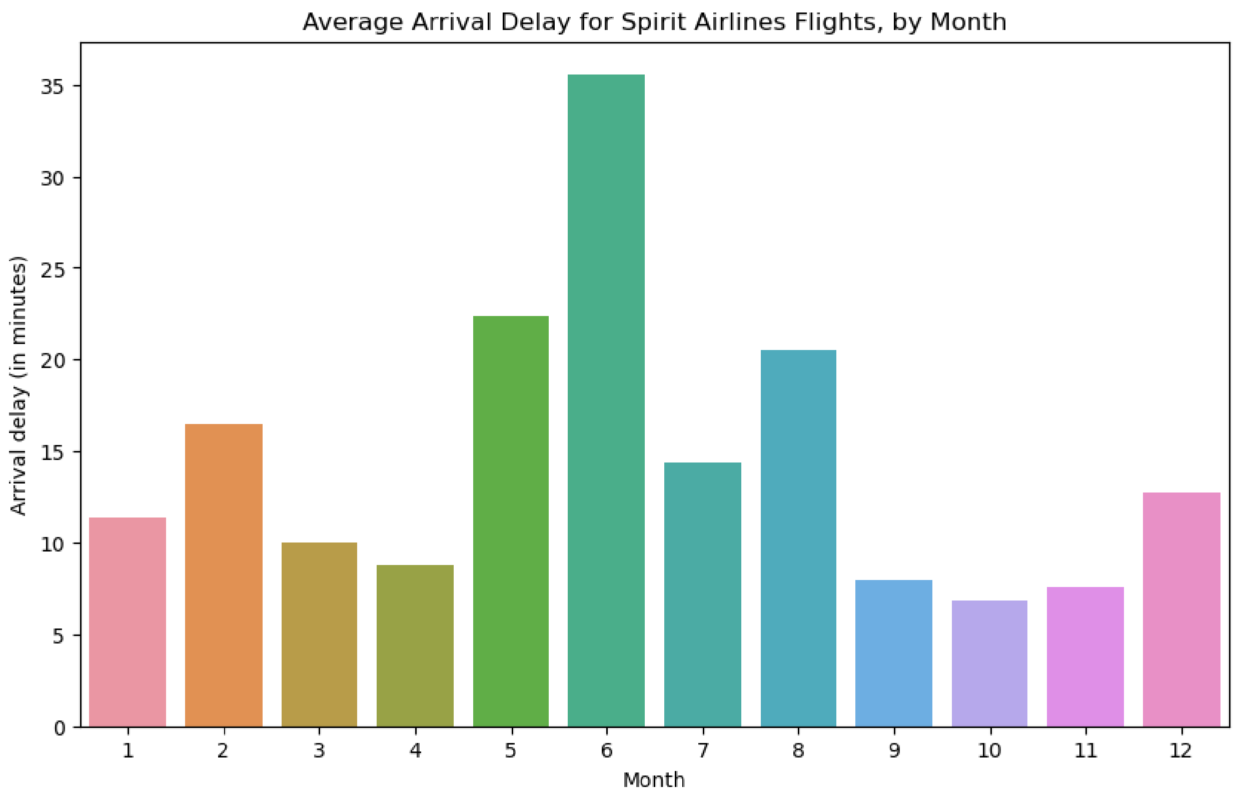
```
# Set the width and height of the figure
plt.figure(figsize=(10,6))

# Add title
plt.title("Average Arrival Delay for Spirit Airlines Flights, by Month")

# Bar chart showing average arrival delay for Spirit Airlines flights by mo
nth
sns.barplot(x=flight_data.index, y=flight_data['NK'])

# Add label for vertical axis
plt.ylabel("Arrival delay (in minutes)")
```

It has three main components:

- `sns.barplot` - This tells the notebook that we want to create a bar chart.
  - *Remember that sns refers to the [seaborn] package, and all of the commands that you use to create charts in this course will start with this prefix.*

- `x=flight_data.index` - This determines what to use on the horizontal axis. In this case, we have selected the column that ***index***es the rows (in this case, the column containing the months).
- `y=flight_data['NK']` - This sets the column in the data that will be used to determine the height of each bar. In this case, we select the `'NK'` column.

Important Note: You must select the indexing column with `flight_data.index`, and it is not possible to use `flight_data['Month']` (*which will return an error*). This is because when we loaded the dataset, the `"Month"` column was used to index the rows. **We always have to use this special notation to select the indexing column.**



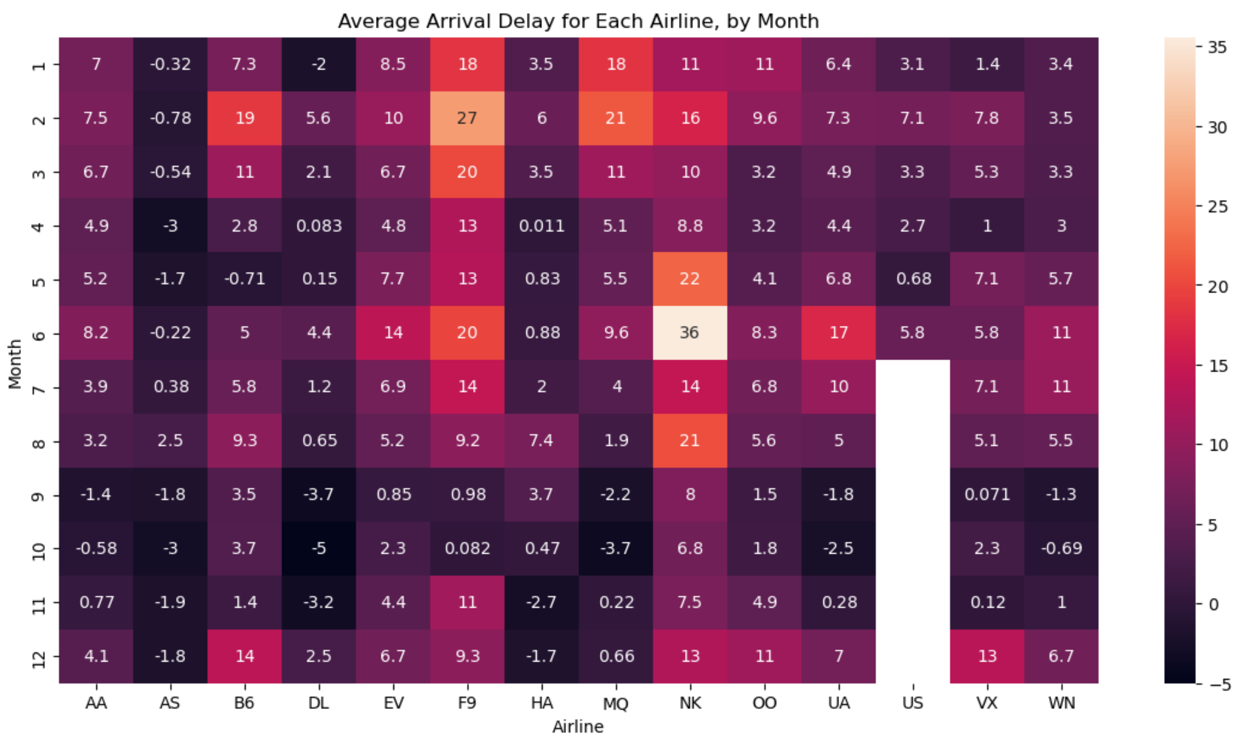
Heatmap

```
# Set the width and height of the figure
plt.figure(figsize=(14,7))

# Add title
plt.title("Average Arrival Delay for Each Airline, by Month")

# Heatmap showing average arrival delay for each airline by month
sns.heatmap(data=flight_data, annot=True)
```

```
# Add label for horizontal axis
plt.xlabel("Airline")
```

This code has three main components:

- `sns.heatmap` - This tells the notebook that we want to create a heatmap.
- `data=flight_data` - This tells the notebook to use all of the entries in `flight_data` to create the heatmap.
- `annot=True` - This ensures that the values for each cell appear on the chart. (*Leaving this out removes the numbers from each of the cells!*)



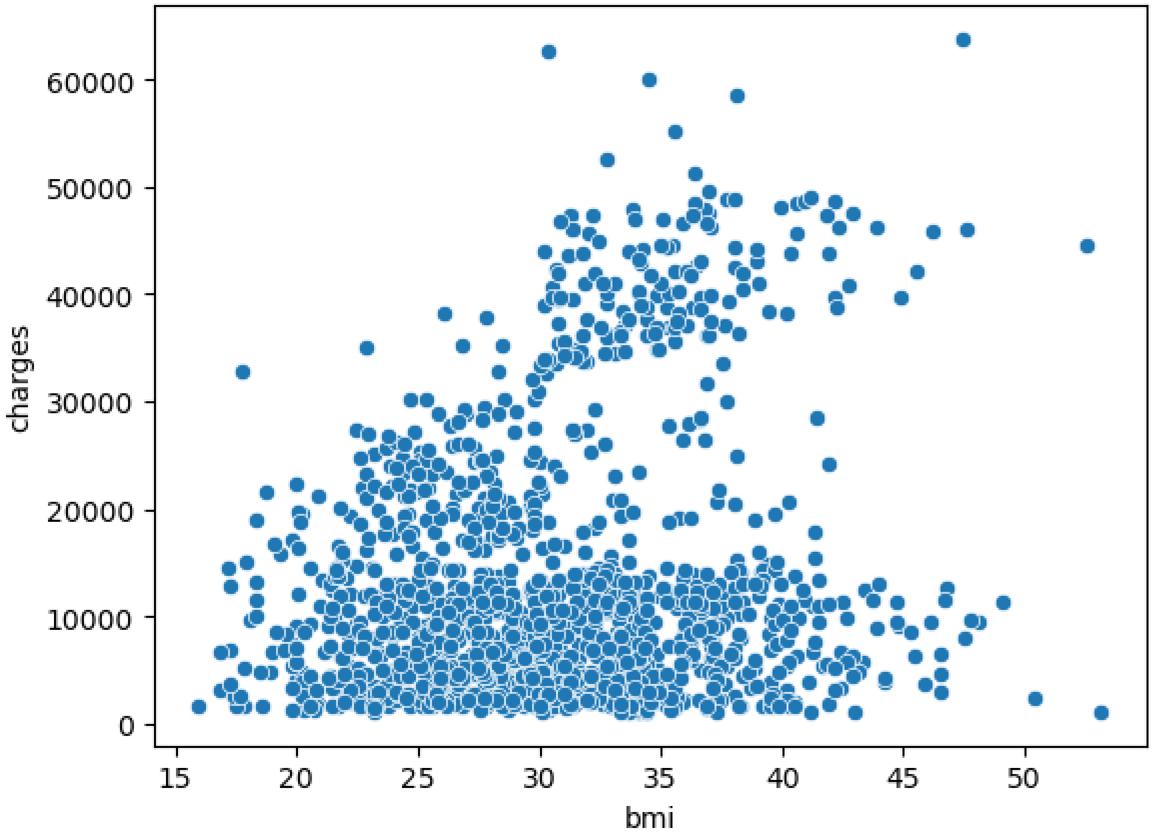
Scatter plots:

Usually, we use scatter plots to highlight the relationship between two continuous variables (like `"bmi"` and `"charges"`)

To create a simple **scatter plot**, we use the `sns.scatterplot` command and specify the values for:

- the horizontal x-axis (`x=insurance_data['bmi']`), and
- the vertical y-axis (`y=insurance_data['charges']`).

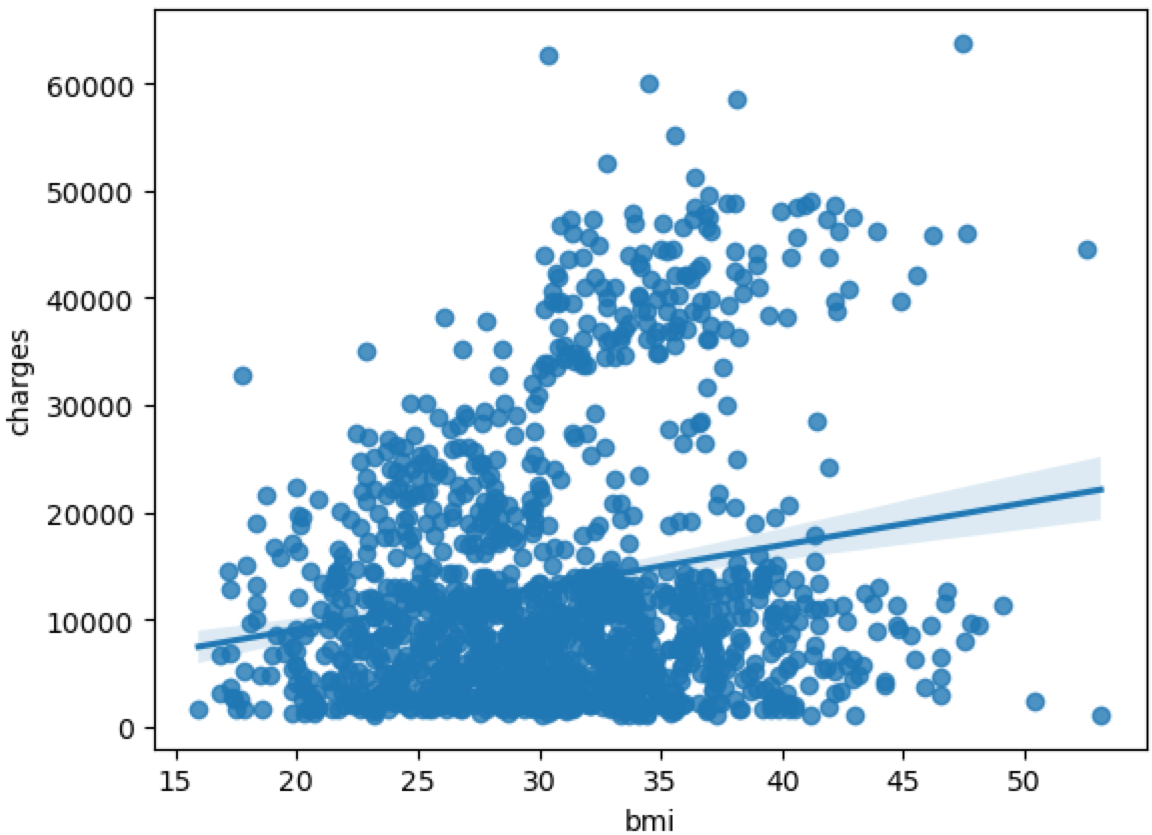
```
sns.scatterplot(x=insurance_data['bmi'], y=insurance_data['charges'])
```



The scatterplot above suggests that body mass index (BMI) and insurance charges are **positively correlated**, where customers with higher BMI typically also tend to pay more in insurance costs. (*This pattern makes sense, since high BMI is typically associated with higher risk of chronic disease.*)

To double-check the strength of this relationship, you might like to add a **regression line**, or the line that best fits the data. We do this by changing the command to `sns.regplot`.

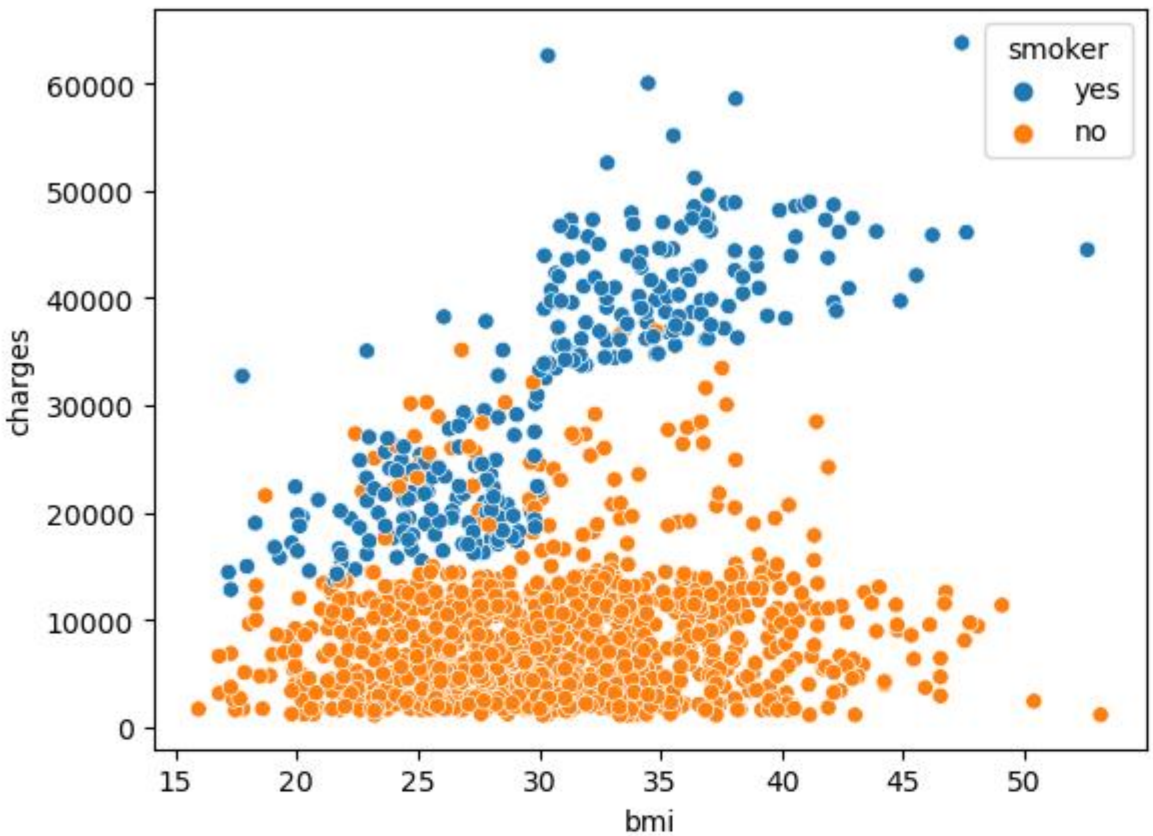
```
sns.regplot(x=insurance_data['bmi'], y=insurance_data['charges'])
```



Color-coded scatter plots

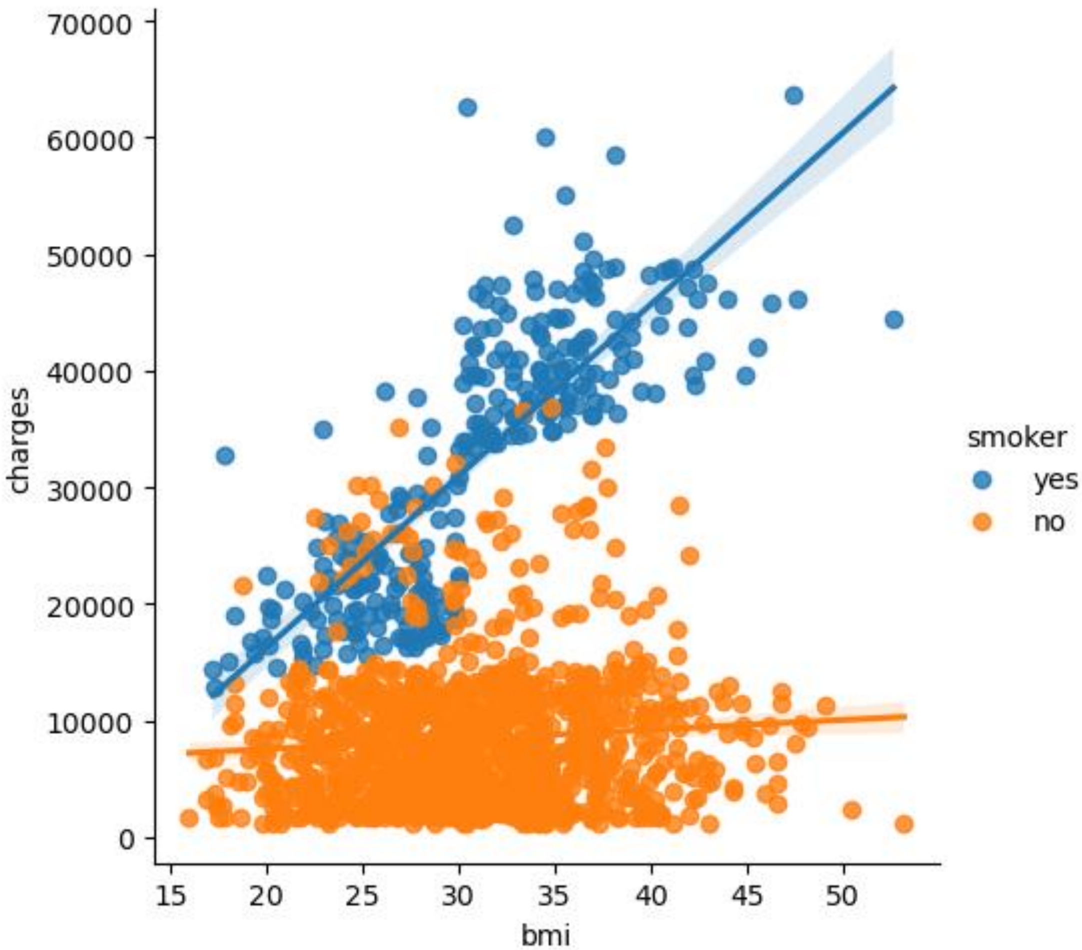
We can use scatter plots to display the relationships between (*not two, but...*) three variables! One way of doing this is by color-coding the points.

```
sns.scatterplot(x=insurance_data['bmi'], y=insurance_data['charges'], hue=
insurance_data['smoker'])
```



To further emphasize this fact, we can use the `sns.lmplot` command to add two regression lines, corresponding to smokers and nonsmokers. (*You'll notice that the regression line for smokers has a much steeper slope, relative to the line for nonsmokers!*)

```
sns.lmplot(x="bmi", y="charges", hue="smoker", data=insurance_data)
```

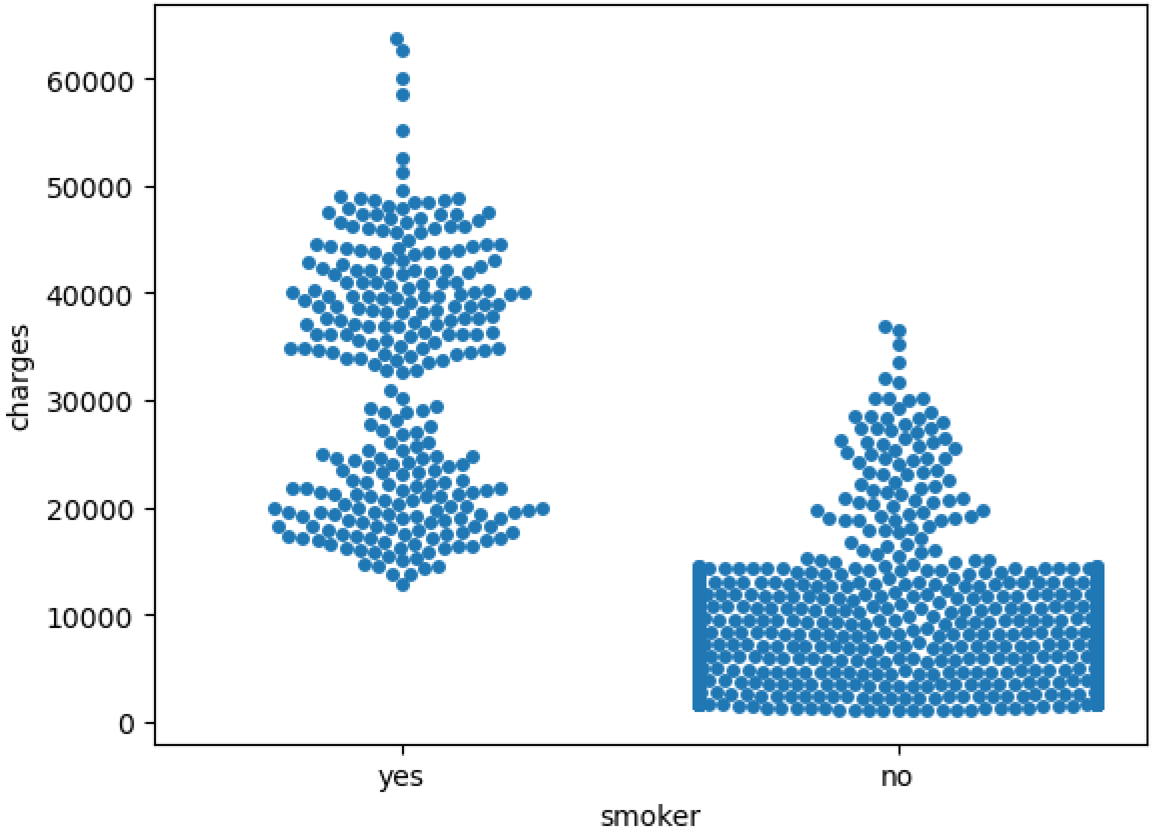


The `sns.lmplot` command above works slightly differently than the commands you have learned about so far:

- Instead of setting `x=insurance_data['bmi']` to select the `'bmi'` column in `insurance_data`, we set `x="bmi"` to specify the name of the column only.
- Similarly, `y="charges"` and `hue="smoker"` also contain the names of columns.
- We specify the dataset with `data=insurance_data`

we can adapt the design of the scatter plot to feature a categorical variable (like `"smoker"`) on one of the main axes. We'll refer to this plot type as a **categorical scatter plot**, and we build it with the `sns.swarmplot` command.

```
sns.swarmplot(x=insurance_data['smoker'],
              y=insurance_data['charges'])
```

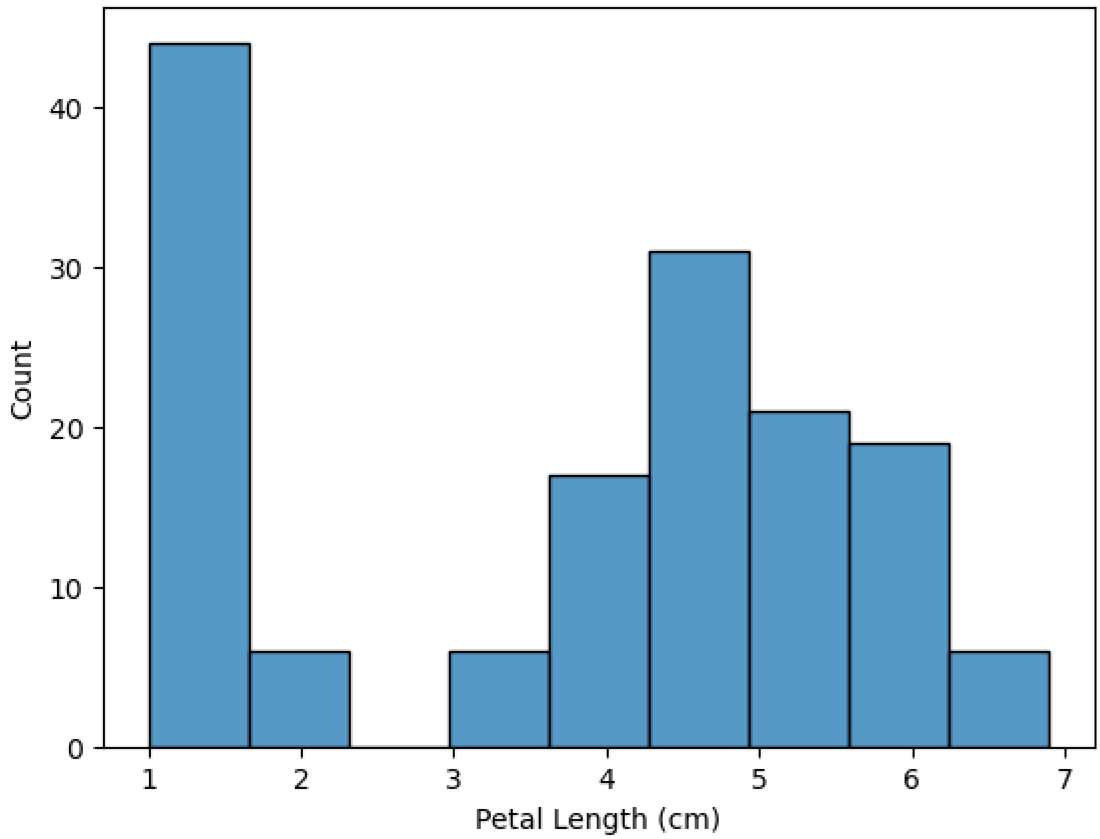


Among other things, this plot shows us that:

- on average, non-smokers are charged less than smokers, and
- the customers who pay the most are smokers; whereas the customers who pay the least are non-smokers.

Histograms

```
# Histogram
sns.histplot(iris_data['Petal Length (cm)'])
```

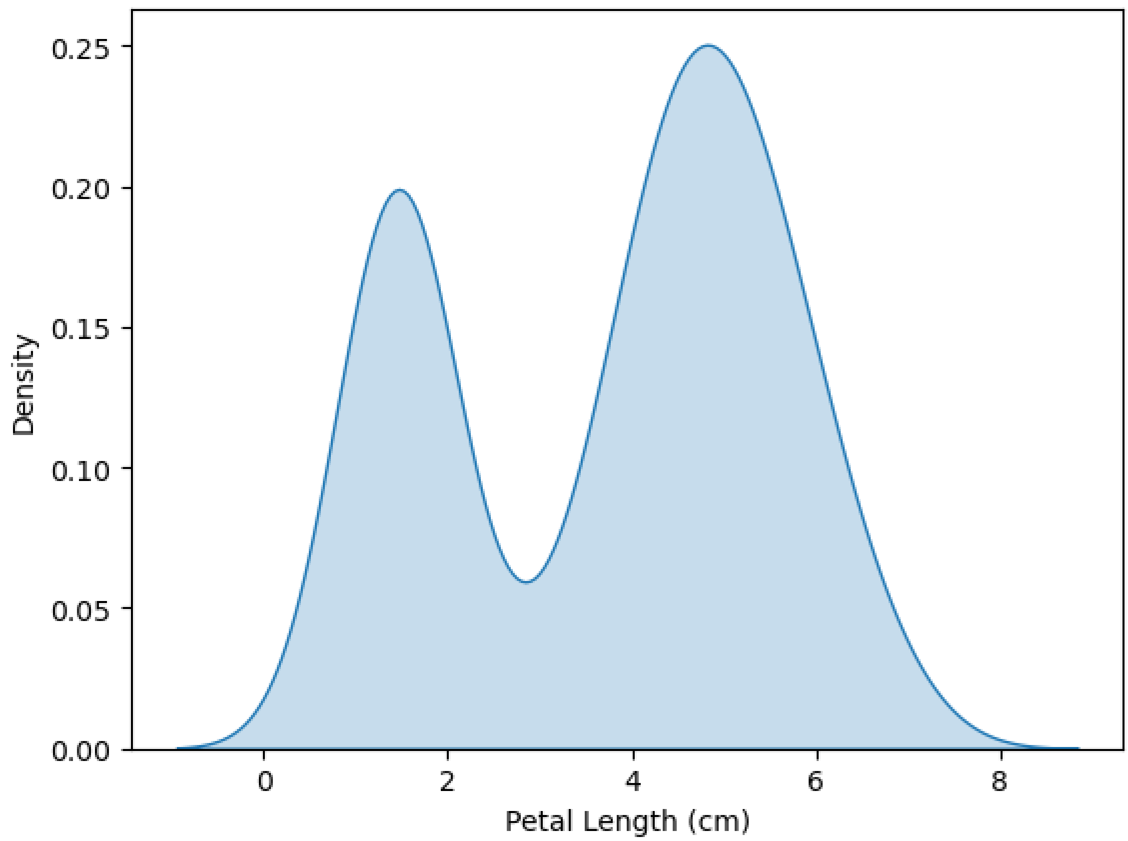


Density plots

The next type of plot is a **kernel density estimate (KDE)** plot. In case you're not familiar with KDE plots, you can think of it as a smoothed histogram.

To make a KDE plot, we use the `sns.kdeplot` command. Setting `shade=True` colors the area below the curve (*and `data=` chooses the column we would like to plot*).

```
# KDE plot
sns.kdeplot(data=iris_data['Petal Length (cm)'], shade=True)
```



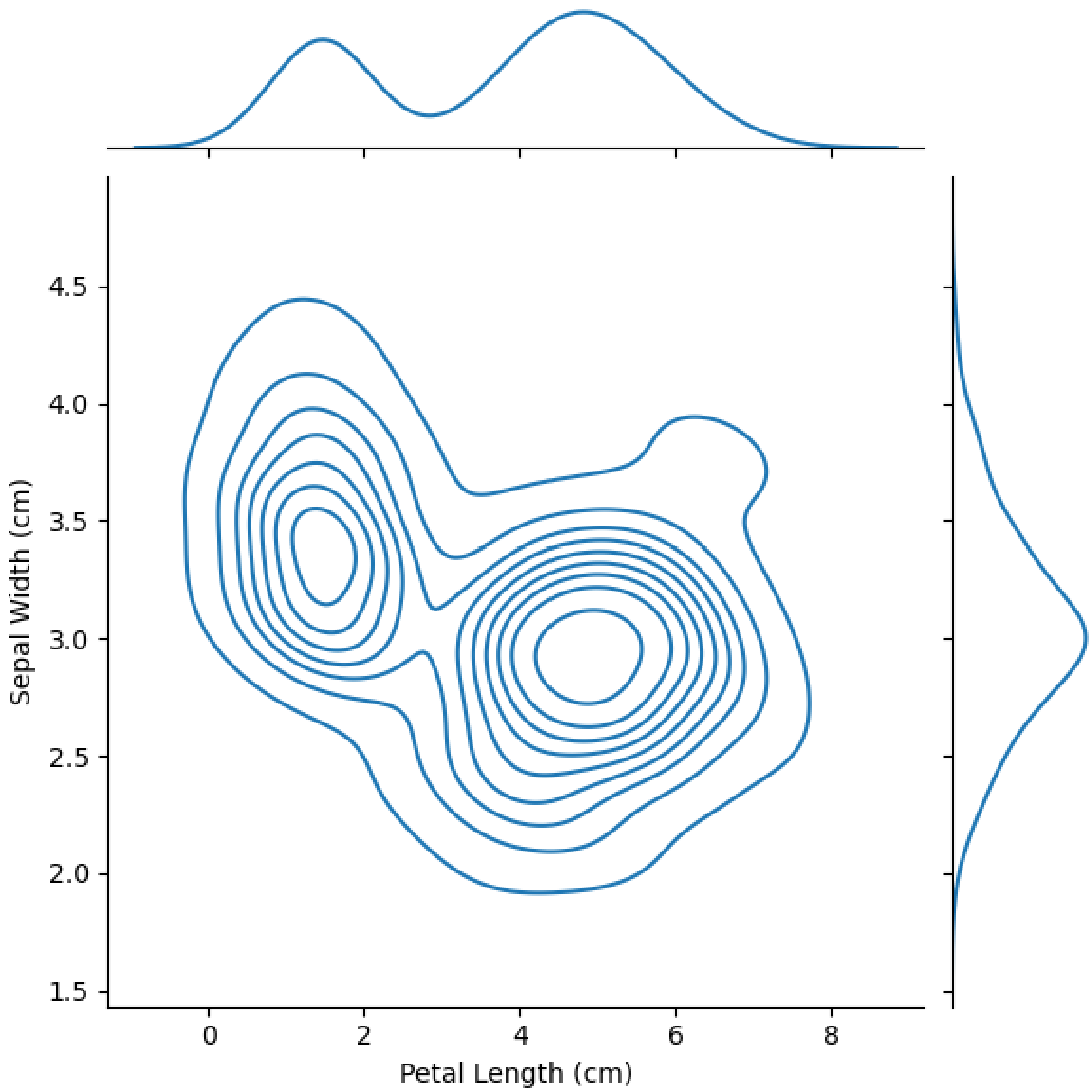
2D KDE plots

We're not restricted to a single column when creating a KDE plot. We can create a **two-dimensional (2D) KDE plot** with the `sns.jointplot` command.

In the plot below, the color-coding shows us how likely we are to see different combinations of sepal width and petal length, where darker parts of the figure are more likely.

In [5]:

```
# 2D KDE plot
sns.jointplot(x=iris_data['Petal Length (cm)'], y=iris_data['Sepal Width (cm)'], kind="kde")
```



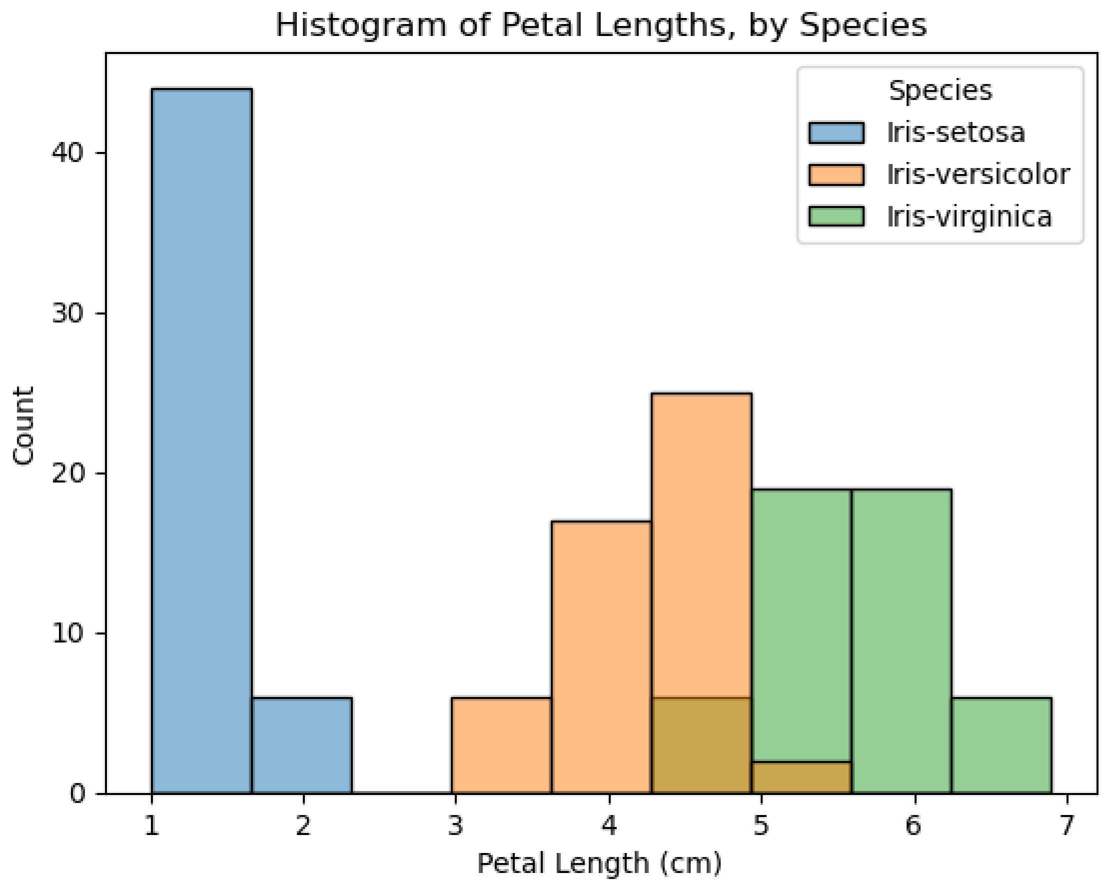
Color-coded plots

We can create three different histograms (one for each species) of petal length by using the `sns.histplot` command (*as above*).

- `data=` provides the name of the variable that we used to read in the data
- `x=` sets the name of column with the data we want to plot
- `hue=` sets the column we'll use to split the data into different histograms

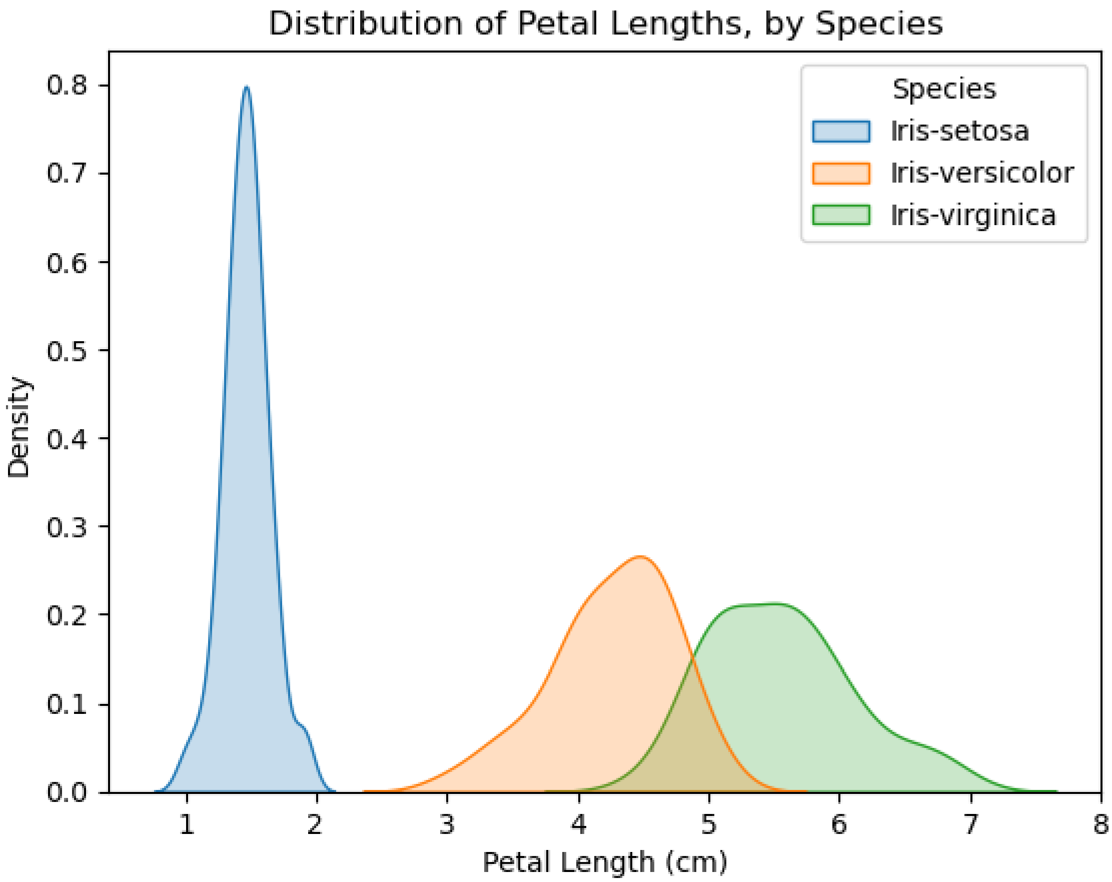
```
# Histograms for each species
sns.histplot(data=iris_data, x='Petal Length (cm)', hue='Species')
```

```
# Add title
plt.title("Histogram of Petal Lengths, by Species")
```



We can also create a KDE plot for each species by using `sns.kdeplot` (*as above*). The functionality for `data`, `x`, and hue are identical to when we used `sns.histplot` above. Additionally, we set `shade=True` to color the area below each curve.

```
# KDE plots for each species
sns.kdeplot(data=iris_data, x='Petal Length (cm)', hue='Species', shade=Tr
ue)

# Add title
plt.title("Distribution of Petal Lengths, by Species")
```



Distribution of Petal Lengths, by Species
Species
Iris-setosa
Iris-versicolor
Iris-virginica
Density
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
1
2
3
4
5
6
7
8
Petal Length (cm)