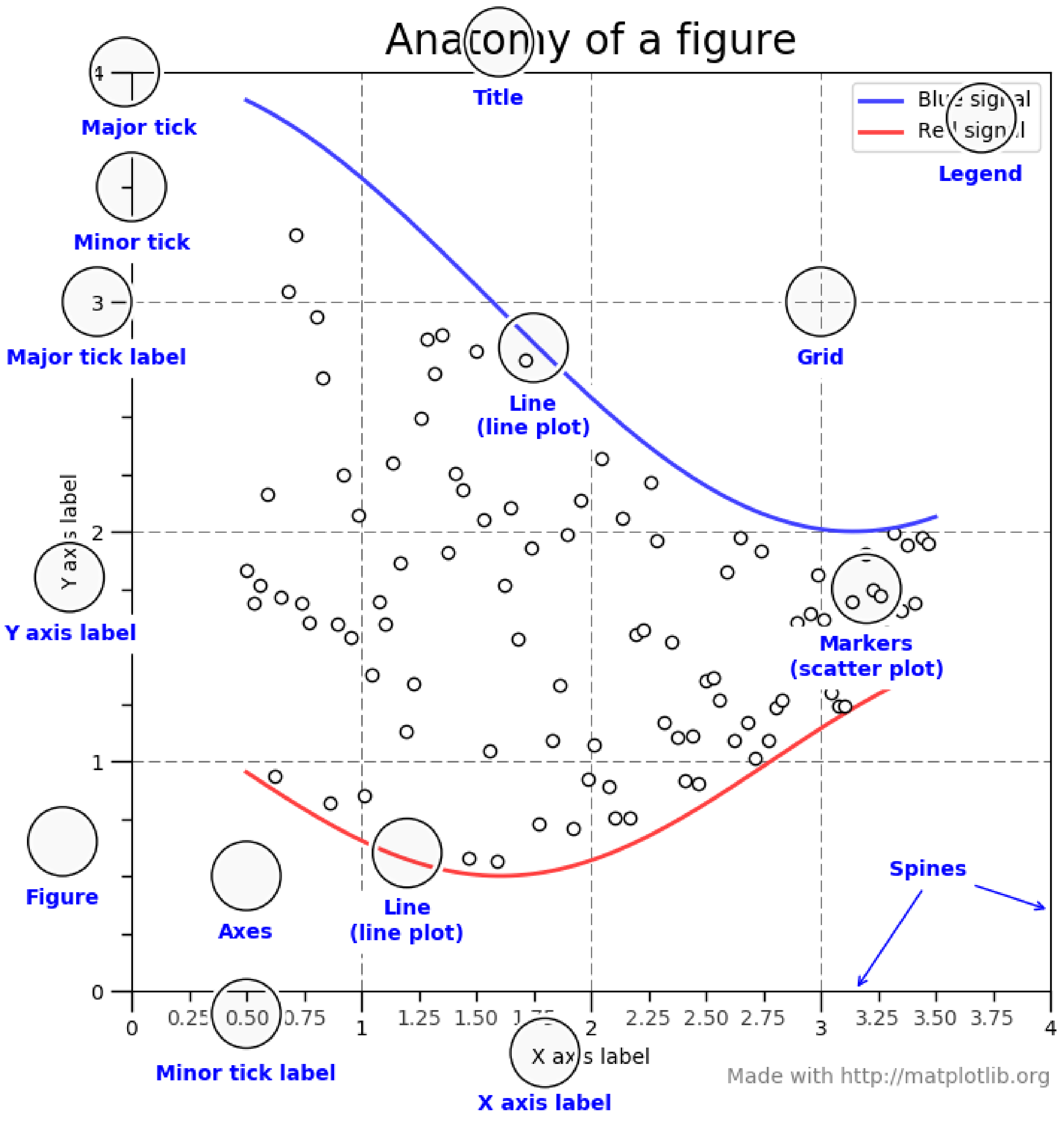
## Matplotlib and Seaborn

Matplotlib and Seaborn are Python graphing libraries that are ubiquitious in the machine learning and data science world. Seaborn is an extension to Matplotlib that provides a high level interface to make prettier plots out of the box than Matplotlib.

### Matplotlib



Matplotlib graphs data onto `Figure`'s, which can have one or more `Axes`. All markings on the figure like the labels and ticks are called `Artist`'s.
Matplotlib has two ways to access its graphing capabilities: an object oriented api and the PyPlot api. The object oriented api allows the user to explicitly create figures and axes for a plot, whereas the pyplot api automatically creates and manages the figure and axes for a plot.
The object oriented access starts by creating figures and axes manually. To create a figure and axes, one would run

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

To plot data onto these axes, we can use the plot method of the Axes class.

```
ax.plot(x, y, label="myFunc")
```

From there, we can set artists like the title and the labels.

```
ax.set_xlabel('x label')
```

However, for most basic graphs, using pyplot will be most convenient. We don't have to manually create and manage plots using subplots(), instead we can just use the call `plt.plot(x=...,y=...)` where pyplot will manage all of our figures for us. We can still edit our figures by acccessing the methods of the plt variable.

```
plt.ylabel('some numbers')
```

## Seaborn

Seaborn is an extension of Matplotlib that allows the user to easily create well styled graphs. It behaves similarly to pyplot, in that you create graphs by calling seaborn functions such as

```
sns.barplot(data=...,x=...,y=...)
```

Using seaborn is very convenient and easy, allowing us to make presentable graphs very fast. There are very few situations that matplotlib alone is preferrable to it.

## Figure Level Plots vs Axes Level Plots

Something that might be confusing when jumping into seaborn is understanding **figure level** plots vs **axes level** plots. Axes level functions plot data onto a Seaborn axes object, which is self contained. Figure-level functions offer a unitary interface to its various axes-level functions. For example, when we worked with displot for all the histogram and density curve plots, those were a figure level plot. However, scatterplot was a axes level plot.

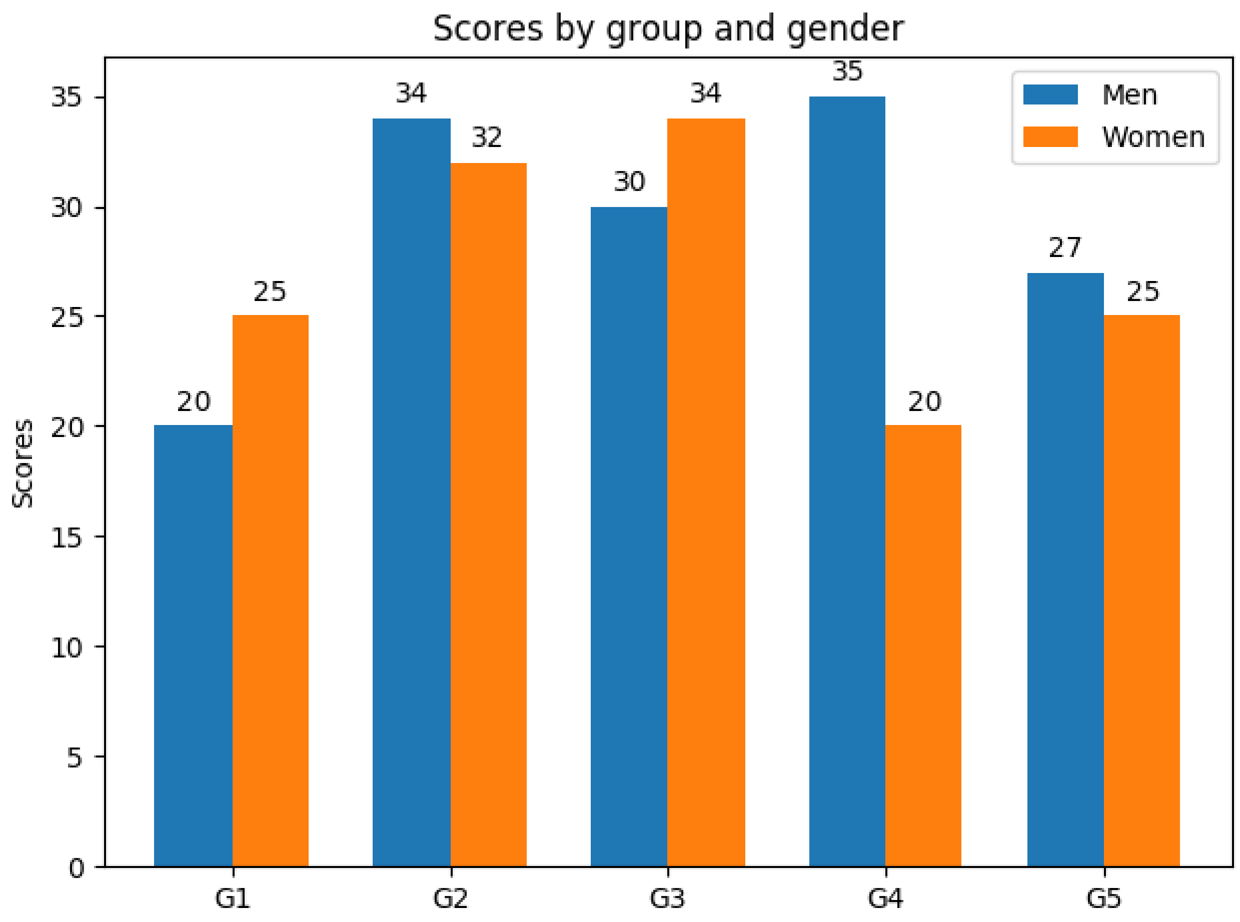
↳ 2 cells hidden

## Facet Grid

Figure functions return a FacetGrid object, which maps a dataset onto multiple axes arrayed in a grid of rows and columns. The resulting object is like a grid of plots, with row*columns total plots. The FacetGrid class is useful when you want to visualize the distribution of a variable or the relationship between multiple variables separately within subsets of your dataset. Having rows and columns allows us to make multiple plots that change which part of the dataset its looking at. For example, on this toy dataset we can see the grid that setting the `row` and `col` parameters create.

[ ] ↳ *4 cells hidden*

## Bar Plots



↳ *2 cells hidden*

## Box Plots

First, it's useful to go over what quartiles are. Quartiles, as the name suggests, split our data up into four equally sized groups (each group contains the same number of datapoints). The first quartile, or lower quartile, acts as the 25th percentile, which means that 25% of our data falls below it. Similarly, the second quartile, which doubles as the median of our data, acts as the 50th percentile while the third quartile, or upper quartile, acts as the 75th percentile.
The Interquartile Range is all the data between the upper and lower quartilea and about 50% of of our data lies in this range.

↳ *8 cells hidden*

## Scatterplots

↳ *5 cells hidden*

## Histograms

A histogram allows us to visualize the distribution of numerical data. To construct one, we first take the range of values of our data and divide it into discrete "bins" or intervals. Then we count the number of times our data falls into each interval. Our bins should be consecutive and non overlapping, and the same size. *italicized text*

↳ *2 cells hidden*

### Density curves

A histogram aims to approximate the underlying probability density function that generated the data by binning and counting observations. Kernel density estimation (KDE) presents a different solution to the same problem. Rather than using discrete bins, a KDE plot smooths the observations with a Gaussian kernel, producing a continuous density estimate. You'll learn how these kernels work in future assignments in this class, so don't worry about it for now.
In seaborn, we can easily add a density curve to our figure level histogram by using the attribute kind="kde", i.e.

```
sns.displot(data=df, x="col_name", kind="kde")
```

For our axes level plot, we can use the `kdeplot` function.

```
sns.kdeplot(data=df, x="x")
```

## Joint Plotting

What if we want to visualize both a bivariate and univariate graph at the same time. Seaborn's `jointplot` can be useful for this. If we have a dataframe df, we can visualize it with the following code:

```
sns.jointplot(data=df, x='', y='', kind = '', hue='')
```

The default behavior (i.e. without supplying the kind variable) will return a scatterplot for the relationship and a histogram for the univariate distributions of x and y.

[ ] ↳ *1 cell hidden*

## Pair plots

We might want to visualize many variables and their relationships in an effort to get an understanding of many different relationships in our data. The pair plot visualizes all pairwise combination of variables for a dataframe.

```
sns.pairplot(data = df)
```

```
penguins = sns.load_dataset("penguins")
sns.pairplot(penguins)
```

## Text and Annotations

Sometimes we may want to annotate our plots to point out interesting features. Matplotlib provides several text functions that help us do this. When given an Axes `ax` and a Figure `fig`, we can perform the following functions.

### `Axes.text`

Adds string `text_to_add` at an arbitrary position (`x`,`y`) on the Axes using the Axes coordinate system.

```
ax.text(x, y, text_to_add)
```

### `Axes.annotate`

Draws an arrow from `xytext` to `xy` on an Axes object that has the string `text` drawn at `xytest`.

```
ax.annotate(text=text, xy=(x, y), xytext=(text_x, text_y),
                    arrowprops=dict(arrowstyle="->", connectionstyle="arc3"))
```

### `Axes.set_title`

Sets the title as `label` for an Axes object.

```
ax.set_title(label)
```

### `Axes.set_ylabel`

Sets the label for the y axis of a given Axes object as ylabel

```
ax.set_ylabel(ylabel)
```

### `Axes.set_xlabel`

Sets the label for the x axis of a given Axes object as xlabel

```
ax.set_xlabel(xlabel)
```

### `Figure.suptitle`

Sets the title for the entire figure as `fig_title`.

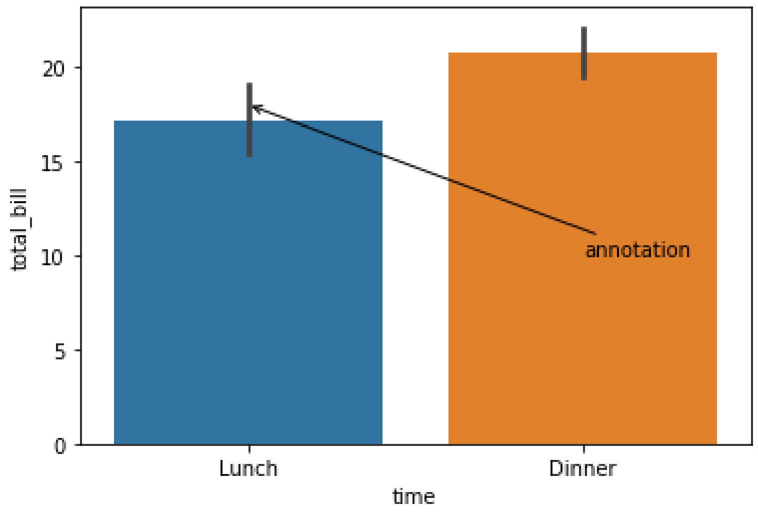
```
fig.suptitle(fig_title)
```

## Text and Annotations in Seaborn

In order to use these functions in Seaborn, we must make sure we are working with the Axes and Figure objects. If we just graphed something with one of Seaborn's axes level functions, we can use the methods above directly on them. If we used a figure level plotting function, all of the Axes objects are in FacetGrid's `.axes` attribute, which is a list of all the Axes objects for each plot in the grid. The figure for a FacetGrid is located in the `.fig` attribute.

```
ax = sns.barplot(data=tips, y="total_bill", x="time")
ax.annotate("annotation",
            xy=(0,18), xycoords='data',
            xytext=(1, 10), textcoords='data',
            arrowprops=dict(arrowstyle="->",
                            connectionstyle="arc3"),
          )
```

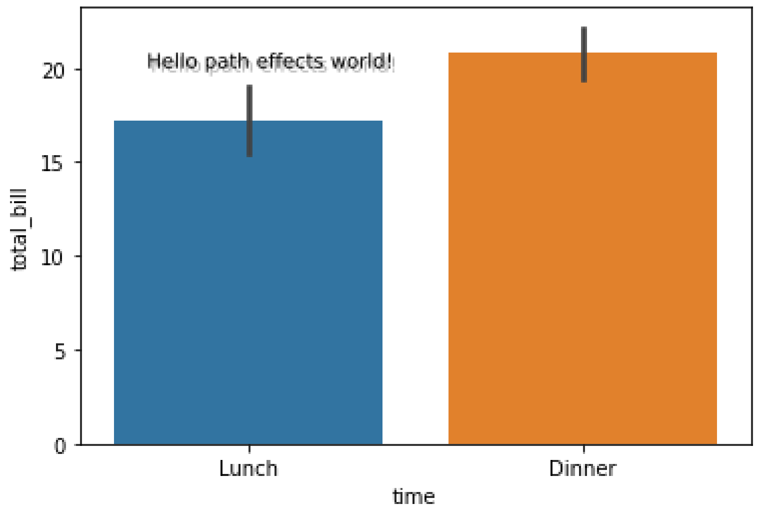
```
Text(1, 10, 'annotation')
```



## Path effects for artists (annotations, text)

Passing in a `path_effects` paramater allows us to create effects. You can find these effects on the Matplotlib website.

```
import matplotlib.patheffects as path_effects
ax = sns.barplot(data=tips, y="total_bill", x="time")
text = plt.text(-0.3,20, 'Hello path effects world!',
                path_effects=[path_effects.withSimplePatchShadow()])
```



## Sliders in Matplotlib

The matplotlib provides useful widgets one of which is the slider. With sliders we can interactively set a value for a variable. This is specially useful when we seek to plot because it adds interaction to our plot and allows us to see the effects of varying a variable.

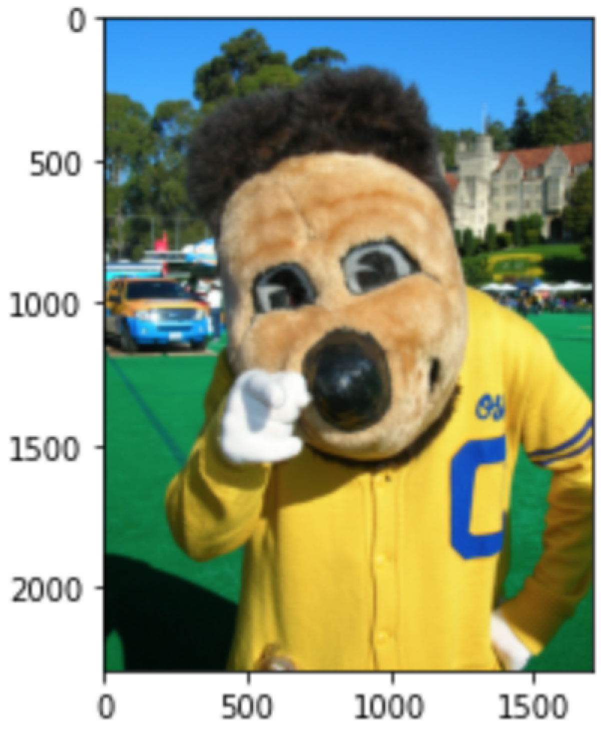
Sliders can be constructed using the following code:

```
# Sliders using matplotlib.widgets
from matplotlib.widgets import Slider, Button, RadioButtons

# First define the range of the parameter:
a_min = 0  # minimum parameter value
a_max = 10 # maximum parameter value
a_init = 1 # default parameter value
slider_axis = plt.axes([0.1, 0.05, 0.8, 0.05])

a_slider = Slider(slider_axis,
                  'a',
                  a_min,
                  a_max,
                  valinit=a_init
                  )

plt.show()
def update(a):
  print(a)
a_slider.on_changed(update)
```

## Working with Images

Images are an important tool for visualization, we can bring images (in png format) into our code using the matplotlib image library. This library provides tools for loading, rescaling, and displaying the image within our code environments.

```
  ...
  [0.         0.29803923 0.10980392 1.        ]
  [0.         0.3647059  0.18039216 1.        ]
  [0.         0.34117648 0.16078432 1.        ]]

 [[0.         0.40784314 0.1764706  1.        ]
  [0.         0.4        0.17254902 1.        ]
  [0.         0.39607844 0.18039216 1.        ]
  ...
  [0.         0.32941177 0.14117648 1.        ]
  [0.         0.3647059  0.18039216 1.        ]
  [0.         0.3137255  0.12941177 1.        ]]]
<matplotlib.image.AxesImage at 0x7f8e44c3c400>
```



Images can be loaded and stored into Image objects provided by the image library. Using these objects we can view the raw RGB data from our png, or we can print these as image data.

```
import matplotlib.image as mpimg
# Importing a local image
img = mpimg.imread('oski_game.png')

'''
matplot allows us to see the image file in its raw form by printing:
'''
print(img)
```

```
'''
matplot also allows us to view the image by calling imshow from pyplot.
'''
plt.imshow(img)
```