Warm up

Warm-Up:

Can you spot the difference between the two columns? What type of data are in each column?

Gas mileage of different car models	Type of engine in different car models
Average height of men of different ethnicities	Percentage of men above 5’ 7” across all ethnicities
Distribution of the number of leaves on different species of deciduous trees	Color of different species of deciduous trees
Normal distribution of the average income of independent businesses in the UK	Most popular products sold at in independent businesses in the UK
Rate of flow of glucose through the phloem of a plant	Type of macromolecule that flows through the phloem of a plant

Answers

Gas mileage of different car models

Is a quantifiable value

Type of engine in different car models

engine type cannot be quantified, therefore is categorical

Average height of men of different ethnicities

Is an average of a quantifiable value

Percentage of men taller than 5’ 7” versus percentage of men shorter than 5’ 7” across all ethnicities

although it is a percentage, there are two specified categories: taller than 5’ 7” and shorter than 5’ 7”, making it categorical data

Distribution of the number of leaves on different species of deciduous trees

Is given to be a number

Color of different species of deciduous trees

Color cannot be quantified, instead represents traits for the trees to be sorted by

Normal distribution of the average income of independent businesses in the UK

Income is a quantified value

Most popular products sold at in independent businesses in the UK

Products are separate categories

Rate of flow of glucose through the phloem of a plant

Is a rate, a function specifying the change of a certain value over time

Type of macromolecule that flows through the phloem of a plant

Types of macromolecules cannot be quantified, each type is a category

Variables

For instance, you wouldn't add yellow marbles to green marbles to produce a new result; it literally doesn't make any sense! Thus, color is a categorical variable.

TYPES OF VARIABLES

Quantitative Variables—Have numerical values for which arithmetic operations make sense.

Categorical Variables —Places an individual into one of several groups or categories. Includes numerical values for which arithmetic operations do not make sense.

Plan

The lesson:

In this lesson, we'll focus on *categorical variables* only since your projects will all make use of the K-Means algorithm, a simple machine learning algorithm used to cluster data with categorical variables.

Specifically, we can practice analyzing by centering our visualizations around gender, a commonly used categorical variable. We'll come up with a number of different graphs to visualize the relationship between gender and several other variables we might want to analyze.

But first, let's go through commonly-used visualizations that you all could use for whatever you might want to display, and how to access said visualizations in Python.

Just like how we produced visualizations before, we can use functions to just create what we desire. We'll go over each function name and the important parameters that you must know (not every parameter for the functions will be listed), practicing with the same spreadsheet file you all downloaded weeks ago.

Let’s watch this quick video on Seaborn to get some background information:

Pie Charts

Matplot (the framework that Seaborn builds off of) has a function to produce pie-charts.
Here's a bit about that function.

You invoke `pie` from the matplotlib package. Depending on what you imported it as (which is most likely as plt), your function invocation should look something like this:

```
plt.pie(size, colors, explode, labels, shadow, startangle, autopct)
```

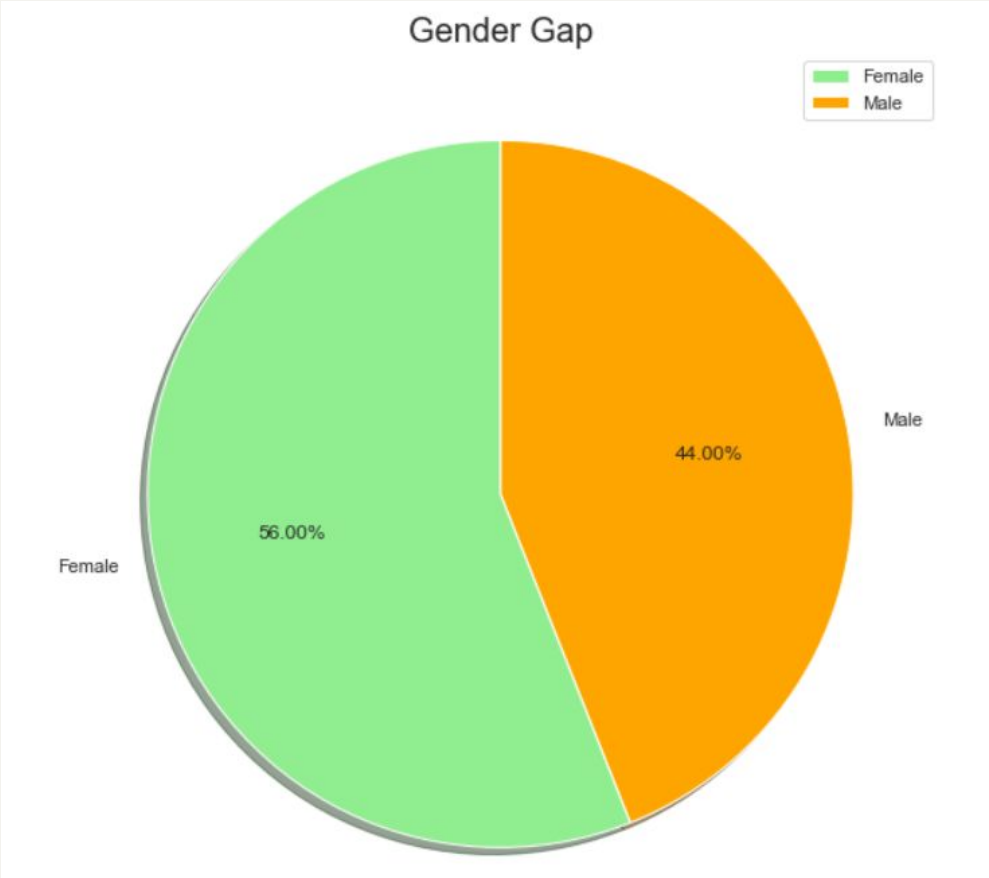
In the same way you created the title, legend, and axis of previous visualizations, you specify that you want to create a pie chart with the `pie` function.
Each parameter of the function specifies an attribute of the pie chart:

- `size` is a specified integer that represents the size of the dataset that is being used
- `colors` is a list of strings, each one representing a color that will be used for the pie chart
- `labels` is a list of strings, each one representing the label for each "slice" of the pie chart
- `explode` is a specified range from 0 to a float that determines the spacing of a slice from the rest of the pie chart
- `shadow` is a boolean value that represents whether or not the pie chart should have a shadow
- `startangle` is a specified float that determines the starting angle of the division between the slices
- `autopct` displays the percent value using Python string formatting
    - For ease of use, let `autopct = '%.2f%%'`. This displays decimals up to their second place (the hundredth's place) in the pie chart.

Pie Charts

```
labels = ['Female', 'Male']
size = data['Gender'].value_counts()
colors = ['lightgreen', 'orange']
explode = [0, 0.001]

plt.rcParams['figure.figsize'] = (9, 9)
plt.pie(size, colors = colors, explode =
explode, labels = labels, shadow = True,
startangle = 90, autopct = '%.2f%%')
```

Now, we can continue as we always have by setting the title, axis, legend, and then finally show our work!

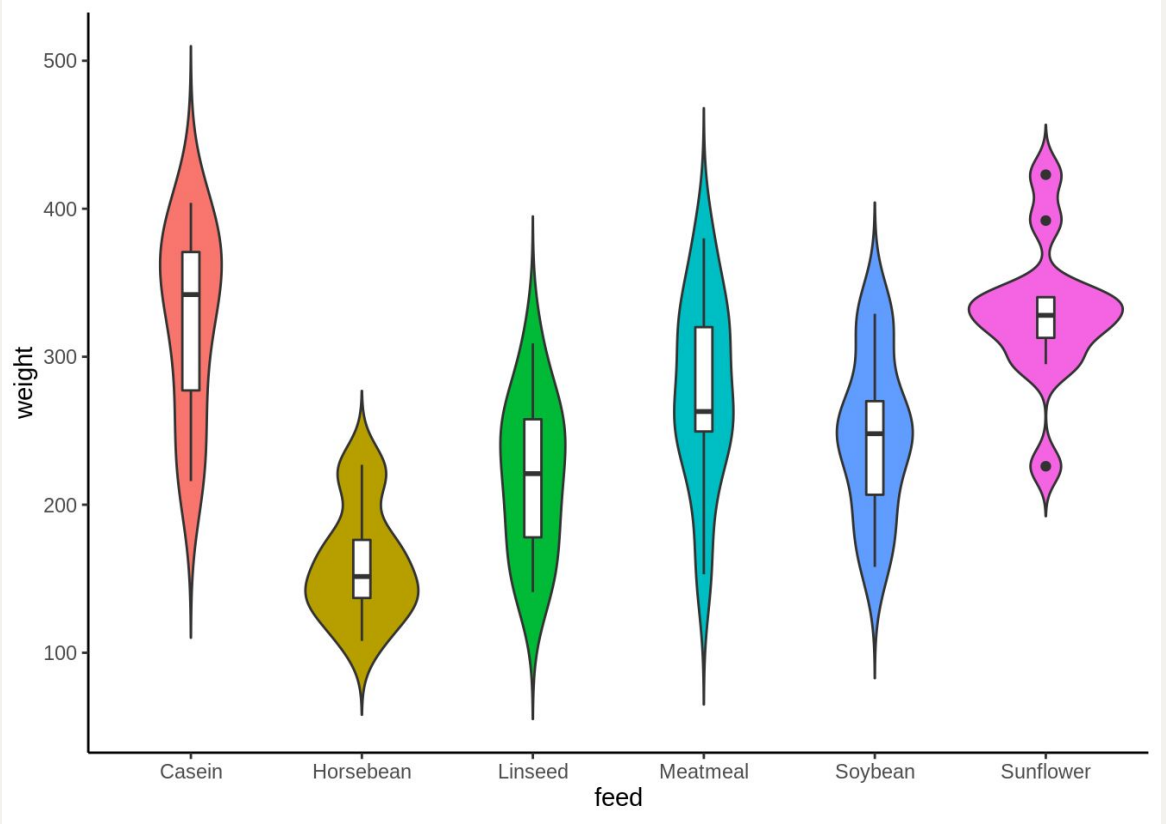


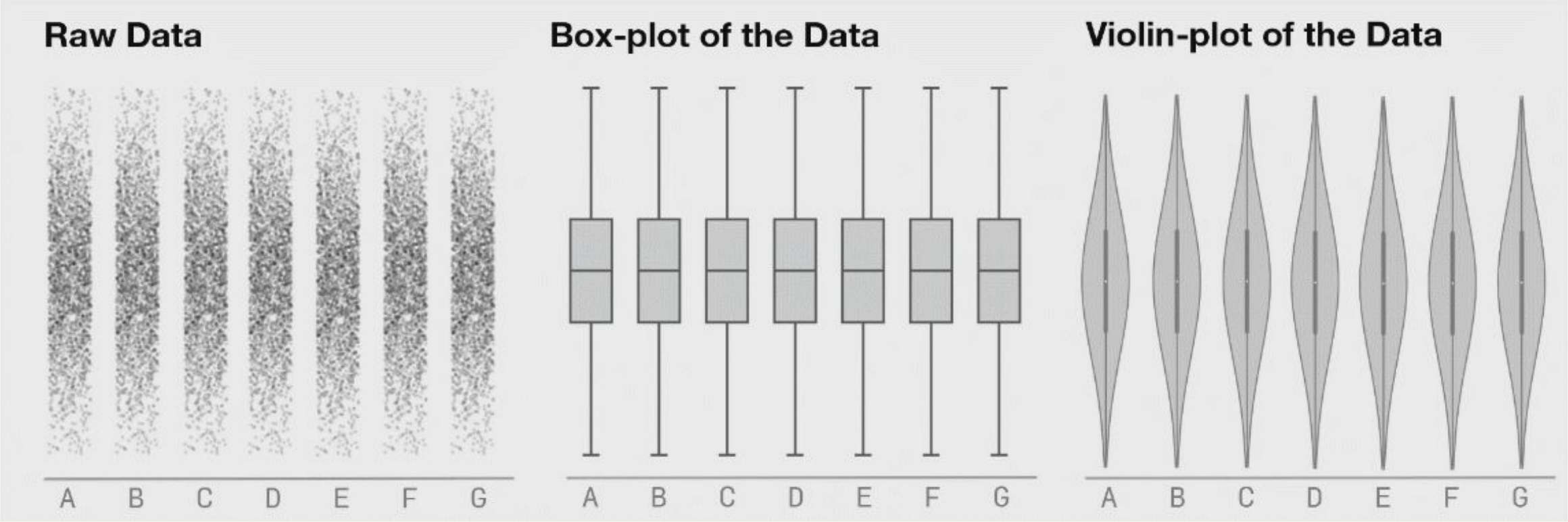
We should end up with something like this. What do you think?

Violin plots

This is probably one that you're unfamiliar with.

A violin plot is used to compare probability distributions. It is similar to a box plot, except is smoothed out to display probability density. Here are a couple of violin plots plotted around box plots to demonstrate their relationship:





This GIF(from autodeskresearch.com) demonstrates how shifts in the density of data(concentration of data points in specific regions of the graph) are not represented on box plots. Violin plots provide a solution for this problem, as shown right.

More about Violin plots

Now that we understand this, let's put it to work in Python.

Unlike the simple pie chart, violinplot must be invoked from the Seaborn package. Depending on what you imported it as (which is most likely as sns), your function invocation should look something like this:

```
sns.violinplot(x, y, palette)
```

Here's what each parameter for the function should be:

- x in this case must be a column or row of data from the dataset that you wish to represent as the x
- y in this case must be a column or row of data from the dataset that you wish to represent as the y
- palette must be set equal to a string denoting the preferred color

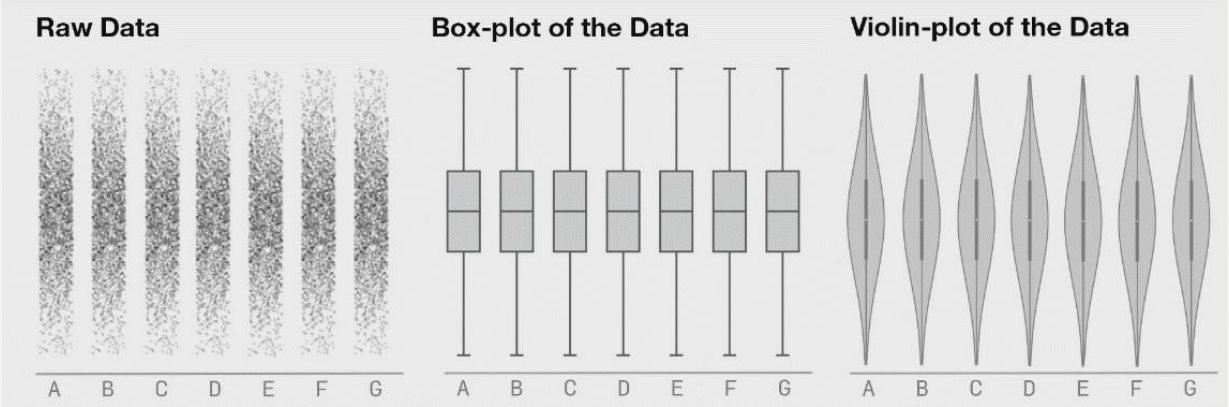
Take some time for yourself to try to make a violin plot of the dataset we have, plotting gender versus annual income. For the palette parameter, you can write palette = 'rainbow'. If you're having trouble, think to yourself: which variable should be represented by which axis? What other lines of code do I have to write to make sure my graph is properly formatted and labeled?

And here’s what we’ve made. A lot cooler than a normal box plot?

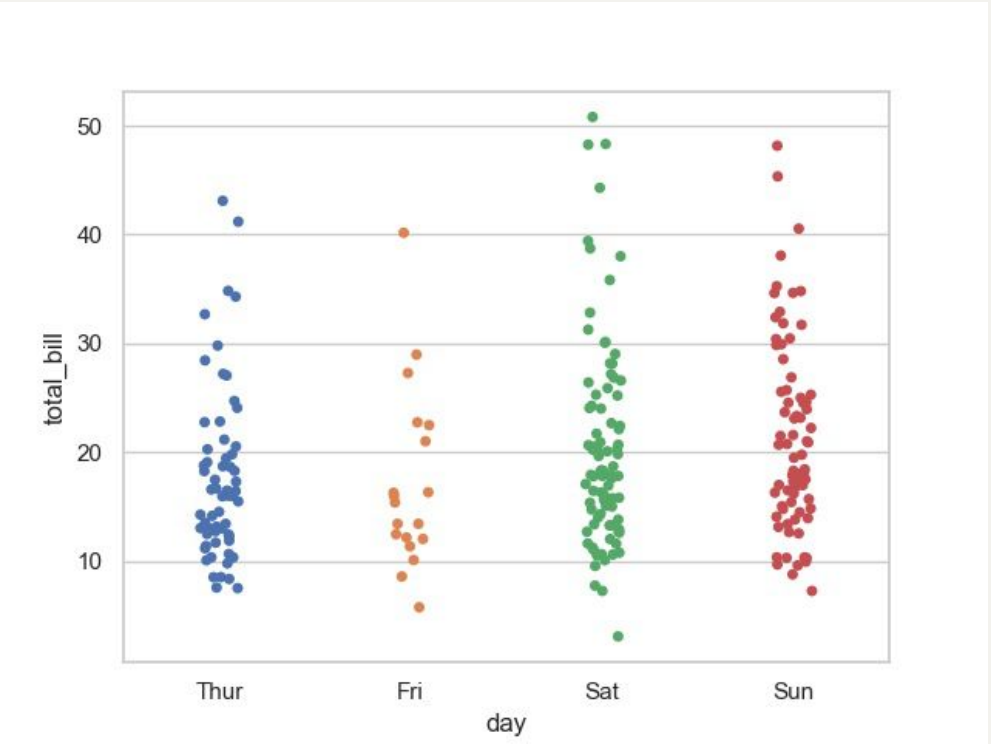
```
## Gender vs Annual Income

plt.rcParams['figure.figsize']
= (18, 7)
sns.violinplot(data['Gender'],
data['Annual Income (k$)'],
palette = 'rainbow')
plt.title('Gender vs Annual
Income', fontsize = 20)
plt.show()
```

3. Strip Plots

The “raw data” graphs are semi-accurate representation of strip plots. Take a look: WHat are some immediate benefits that you can spot from this representation of data? WHy use this instead of a violin plot



This is the last visualization that we’ll cover for today. Remember this image from before?



Well, given that there are few enough data points, it might be helpful to simply plot every single one on the same set of axes as violin plots or box plots. Notice that the horizontal axis is still categorical variables. This means that the horizontal spread on the points is only there to avoid overlap for your viewing convenience. Kind of silly, right?

The main benefit of strip plots is that you can *see every data point, making the data hard to misrepresent*. The main drawback however is that the median, 75th and 25th percentiles, and IQR are all invisible at a glance.

Stripplot

Let's look at the Python function, named `stripplot`.

It's invoked by Seaborn once again, meaning your call should look something like this (if you imported Seaborn as sns):

```
sns.stripplot(x, y, palette, size)
```

Parameters

Similar to your violin plot, here are the parameters:

- x in this case must be a column or row of data from the dataset that you wish to represent as the x
- y in this case must be a column or row of data from the dataset that you wish to represent as the y
- palette must be set equal to a string denoting the preferred color
- size must be set equal to an integer representing the intended size of the plot

How simple is that? Let’s put it into practice by creating a strip plot of Gender vs Age. Since we’re creating a strip plot, we know that gender, our categorical variable, must be the x-axis.

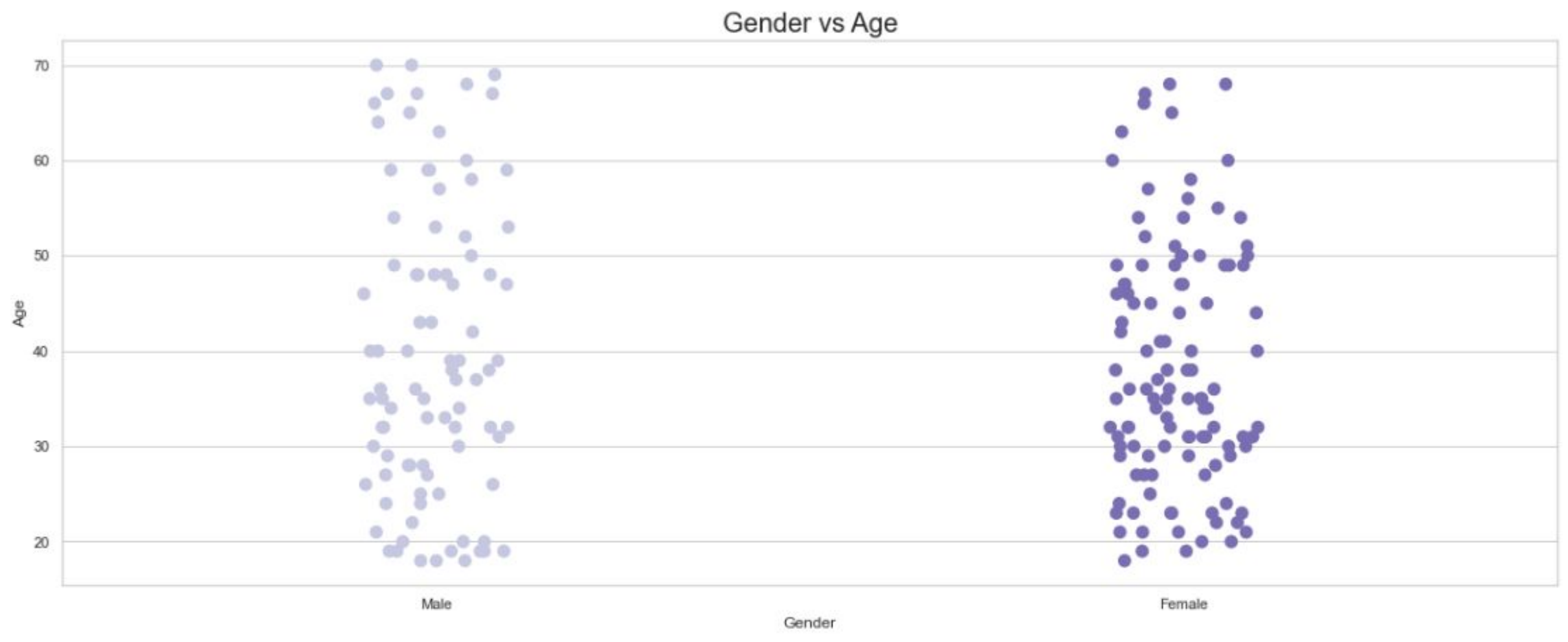
Final Touches

We filled our strip plot function out almost identically to our violin plot, but specified the additional parameter size as 10. As usual, feel free to play around with the parameters to figure out how the visualization changes with different parameters.

```
# Gender vs Age
plt.rcParams['figure.figsize'] = (18, 7)
sns.stripplot(data['Gender'], data['Age'],
palette = 'Purples', size = 10)
```

Next, finish it off as usual with proper labeling and show off your work.

```
plt.title('Gender vs Age', fontsize = 20)
plt.show()
```


Gender vs Age
70
60
50
40
30
20
Age
Male
Female
Gender

Homework

Homework: Using Seaborn, create a violin plot contrasting Gender against Annual Income. Follow this by creating a strip plot contrasting Gender against Age. Interpret the visualizations by drawing your conclusions from the data. This means analyzing factors like the shape of the data (skewed, uniform, etc), spread (dense, sparse, etc), potential outliers, and overall differences between the genders.

Helpful resources

Here are some helpful resources if you were interested in learning more:

The Seaborn website has API documentation that explains all of its features in full detail. Check it out!
https://seaborn.pydata.org/
https://seaborn.pydata.org/tutorial.html

In addition, GeeksForGeeks has an amazing guide on how to utilize almost every feature of Seaborn.
https://www.geeksforgeeks.org/python-seaborn-tutorial/

If you'd prefer video format, this guide (about an hour long) runs through how to use Seaborn. Use the slated video chapters to search for any content you'd like to review or learn more about!
https://www.youtube.com/watch?v=6GUZXDef2U0