**Summary Part 1**

The purpose of this section is to give you an idea of how the final web app works in terms of passing information back and forth between the back end and front end. The web template you'll be using at the end of the lesson will already provide the code for sharing information between the back and front ends. Your task will be to wrangle data and set up the plotly visualizations using Python. But it's important to get a sense for how the web app works.

In the video above, the data set was sent from the back end to the front end. This was accomplished by including a variable in the render\_template() function like so:

data = data\_wrangling()

@app.route('/')

@app.route('/index')

**def** **index**():

**return** render\_template('index.html', data\_set = data)

What this code does is to first load the data using the data\_wrangling function from wrangling.py. This data gets stored in a variable called data.

In render\_template, that data is sent to the front end via a variable called data\_set. Now the data is available to the front\_end in the data\_set variable.

In the index.html file, you can access the data\_set variable using the following syntax:

{{ data\_set }}

You can do this because Flask comes with a template engine called **[Jinja](http://jinja.pocoo.org/" \t "_blank)**. Jinja also allows you to put control flow statements in your html using the following syntax:

{% for tuple in data\_set %}

<p>{{tuple}}</p>

{% end\_for %}

The logic is:

1. Wrangle data in a file (aka Python module). In this case, the file is called wrangling.py. The wrangling.py has a function that returns the clean data.
2. Execute this function in routes.py to get the data in routes.py
3. Pass the data to the front-end (index.html file) using the render\_template method.
4. Inside of index.html, you can access the data variable with the squiggly bracket syntax {{ }}

**Summary Part 2**

In the second part, a Plotly visualization was set up on the back-end inside the routes.py file using Plotly's Python library. The Python plotly code is a dictionary of dictionaries. The Python dictionary is then converted to a JSON format and sent to the front-end via the render\_templates method.

Simultaneously a list of ids are created for the plots. This information is also sent to the front-end using the render\_template() method.

On the front-end, the ids and visualization code (JSON code) is then used with the Plotly javascript library to render the plots.

In summary:

1. Python is used to set up a Plotly visualization
2. An id is created associated with each visualization
3. The Python Plotly code is converted to JSON
4. The ids and JSON are sent to the front end (index.html).
5. The front end then uses the ids, JSON, and JavaScript Plotly library to render the plots.

**JavaScript or Python**

You could actually do all of this with only JavaScript. You would read the data, wrangle the data, and then create the plots all using JavaScript; however, to do all of this in JavaScript, you'd need to learn more about JavaScript programming. Instead, you can use the pandas and Python skills you already

have to wrangle data on the back-end.

# Summary Part 3

In part 3, the code iterated through the data set to create a visualization with multiple lines: one for each country.

The original code for a line chart with a single line was:

graph\_one = [go.Scatter(

x = data[0][1],

y = data[0][2],

mode = 'lines',

name = country

)]

To make a visualization with multiple lines, graph\_one will be a list of line charts. This was accomplished with the following code:

graph\_one = []

**for** data\_tuple in data:

graph\_one.append(**go**.Scatter(

x = data\_tuple[1],

y = data\_tuple[2],

mode = 'lines',

name = data\_tuple[0]

))

# Summary Part 4

In the last part, three more visualizations were added to the wrangling Python module. The wrangling included reading in the data, cleaning the data, and preparing the Plotly code. Each visualization's code was appended to a list called figures. These visualizations were then imported into the routes.py file. This figures list was sent from the back end to the front end via the render\_template method. A list of ids were also sent from the back end to the front end.

Then on the front end (index.html), a div was created for each visualization's id. And with help from the JavaScript Plotly library, each visualization was rendered inside appropriate div.

# Beyond a CSV file

Besides storing data in a local csv file (or text, json, etc.), you could also store the data in a database such as a SQL database.

The database could be local to your website meaning that the database file is stored on the same server as your website; alternatively, the database could be stored somewhere else like on a separate database server or with a cloud service like Amazon AWS.

Using a database with your web app goes beyond the scope of this introduction to web development, here are a few resources for using databases with Flask apps:

* [**Tutorial - Using Databases with Flask**](https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-iv-database)
* [**SQL Alchemy**](http://docs.sqlalchemy.org/en/latest/)- a Python toolkit for working with SQL
* [**Flask SQLAlchemy**](http://flask-sqlalchemy.pocoo.org/2.3/) - a Flask library for using SQLAlchemy with Flask