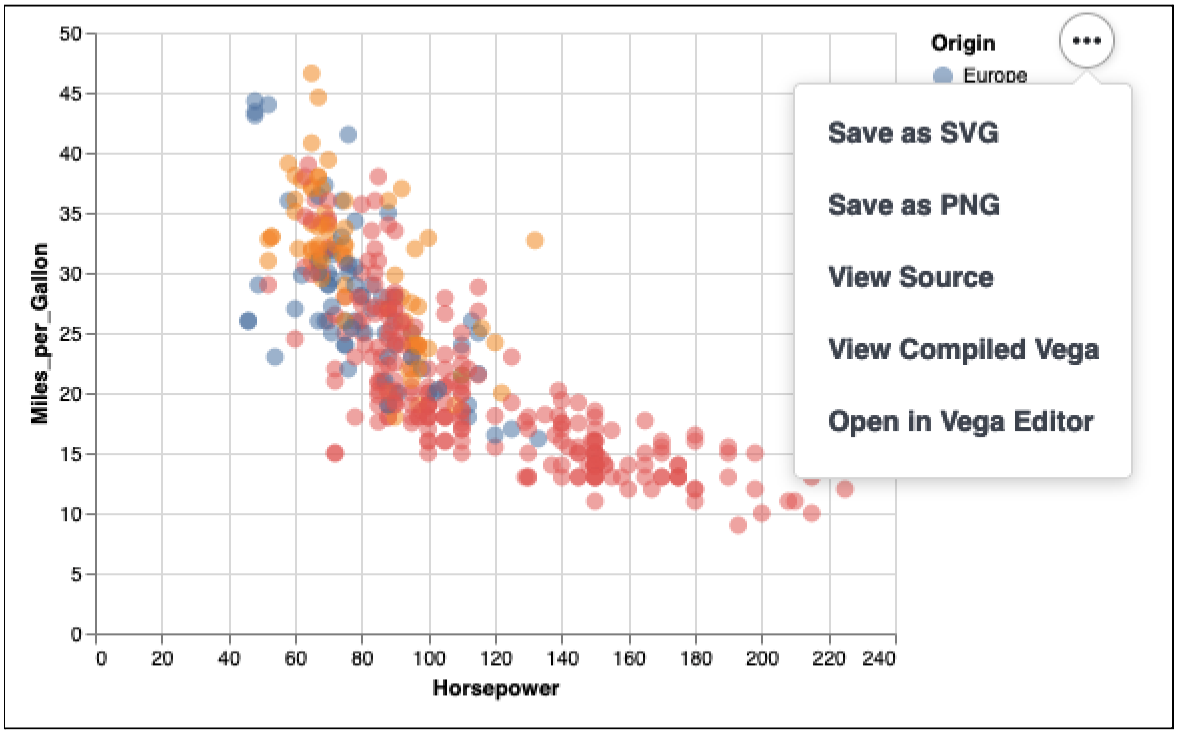
Exporting from Altair to HTML (and other formats)

We often want to export visualizations to other formats (e.g., .png). Especially for interactive work, it's often useful to be able to embed visualizations in a webpage, and Altair provides various ways to do this.

1. Exporting one plot as an image

To save a single chart as an image, there are a few options. The Jupyter notebook included in this folder (on Canvas) provides a runnable example of several of these.

The first option is to manually save a chart after you have generated it in the notebook. To do this, you can click on the three dots (...) at the upper right of the figure, and then choose an option like "Save as PNG" (and then follow the system prompts).



Alternatively, we can directly tell it to save a plot as a file in code. To do so, we can use the `.save()` function provided by the `altair_saver` library.

Basically you create a chart in the usual manner (e.g., `chart = alt.Chart(...)`). You can then use `chart.save()` to save the chart as a file, where the filename you provide will tell it what type of file format to use. (Again, see examples in the Jupyter notebook).

For example, to save the chart as `.png` image, we can provide a filename with the `.png` extension, e.g.,

```
chart.save('chart.png')
```

You can use whatever filename you like, as long as it ends in `.png`. Similarly, we can save it in `.svg` format by providing a filename with a `.svg` extension, e.g.,

```
chart.save('chart.svg')
```

In either case, these will be saved in the same directory as the Jupyter notebook, and you should be able to open them in a standard file browser.

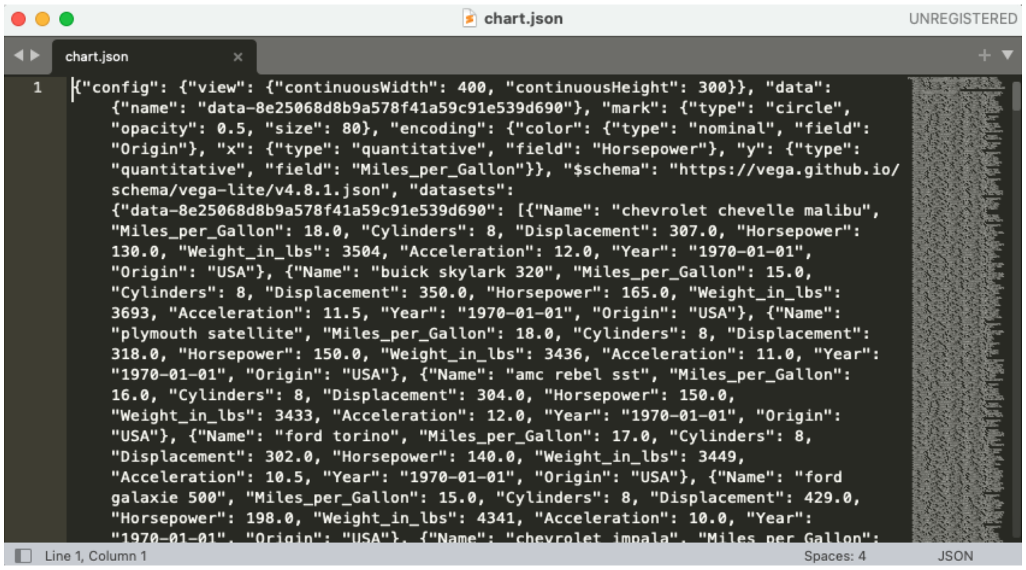
2. Exporting to .json

Alternatively, we can export the entire chart (including the data that it represents) as a single `.json` object, and save it as a `.json` file. As mentioned previously, `.json` is a format that can store many types of data using key-value pairs (and other simple data types). The reason we might want to do this, is that visualizations in `.json` format can be read in by tools like Vega.

The way to do this is exactly the same. Just use the .json extension.

```
carplot.save('chart.json')
```

You will similarly find this file in the same directory as the Jupyter notebook, and you should be able to open it and read it with any text editor (e.g., Sublime Text). It should look something like this:



```
{"config": {"view": {"continuousWidth": 400, "continuousHeight": 300}}, "data":
    {"name": "data-8e25068d8b9a578f41a59c91e539d690"}, "mark": {"type": "circle",
    "opacity": 0.5, "size": 80}, "encoding": {"color": {"type": "nominal", "field":
    "Origin"}, "x": {"type": "quantitative", "field": "Horsepower"}, "y": {"type":
    "quantitative", "field": "Miles_per_Gallon"}}, "$schema": "https://vega.github.io/
    schema/vega-lite/v4.8.1.json", "datasets":
    {"data-8e25068d8b9a578f41a59c91e539d690": [{"Name": "chevrolet chevelle malibu",
    "Miles_per_Gallon": 18.0, "Cylinders": 8, "Displacement": 307.0, "Horsepower":
    130.0, "Weight_in_lbs": 3504, "Acceleration": 12.0, "Year": "1970-01-01",
    "Origin": "USA"}, {"Name": "buick skylark 320", "Miles_per_Gallon": 15.0,
    "Cylinders": 8, "Displacement": 350.0, "Horsepower": 165.0, "Weight_in_lbs":
    3693, "Acceleration": 11.5, "Year": "1970-01-01", "Origin": "USA"}, {"Name":
    "plymouth satellite", "Miles_per_Gallon": 18.0, "Cylinders": 8, "Displacement":
    318.0, "Horsepower": 150.0, "Weight_in_lbs": 3436, "Acceleration": 11.0, "Year":
    "1970-01-01", "Origin": "USA"}, {"Name": "amc rebel sst", "Miles_per_Gallon":
    16.0, "Cylinders": 8, "Displacement": 304.0, "Horsepower": 150.0,
    "Weight_in_lbs": 3433, "Acceleration": 12.0, "Year": "1970-01-01", "Origin":
    "USA"}, {"Name": "ford torino", "Miles_per_Gallon": 17.0, "Cylinders": 8,
    "Displacement": 302.0, "Horsepower": 140.0, "Weight_in_lbs": 3449,
    "Acceleration": 10.5, "Year": "1970-01-01", "Origin": "USA"}, {"Name": "ford
    galaxie 500", "Miles_per_Gallon": 15.0, "Cylinders": 8, "Displacement": 429.0,
    "Horsepower": 198.0, "Weight_in_lbs": 4341, "Acceleration": 10.0, "Year":
    "1970-01-01", "Origin": "USA"}, {"Name": "chevrolet impala", "Miles_per_Gallon":
```

As you can see, the whole thing is one big dictionary, starting with the “config” key, which itself points to a dictionary. You can also see that it contains the data as an element (“data-8e25068d8b9a578f41a59c91e539d690”, though your file might be different), which has been encoded as a list, with each row encoded as a dictionary (key-value pairs for each column – Name, Cylinders, etc.). This is not the most compact or easy-to-read format, but it is highly portable (can be read by many services).

We’ll come back to the `.json` file in a second.

3. Exporting to HTML

Altair also provides a way to create a simple HTML webpage that embeds a single chart. We again include this step in the Jupyter notebook, and it is as simple as using a filename with the `.html` extension, e.g.,

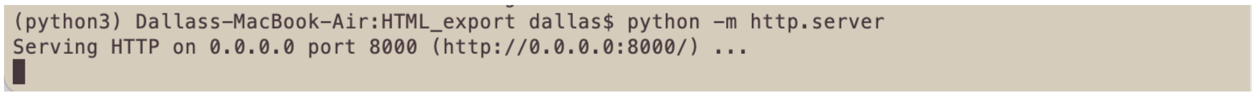
```
carplot.save('chart.html')
```

Again, this should create a chart.html file in the same directory as the Jupyter notebook.

To view this file in a web browser, you can start a simple web server on your computer, which should allow you to display the file. To do so, use the terminal to navigate to where the .html file is, and run:

```
python -m http.server
```

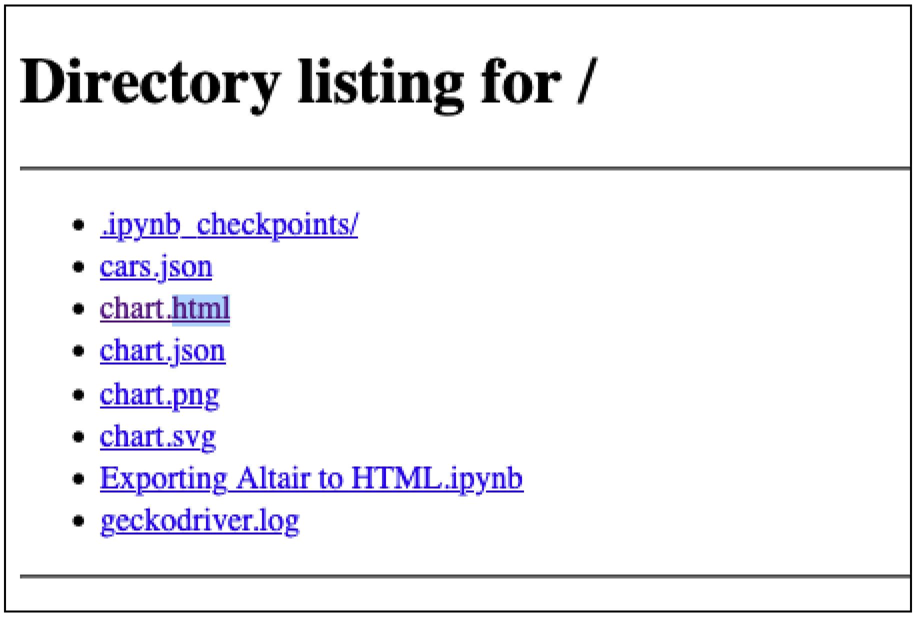
That should show you something like this:



```
(python3) Dallass-MacBook-Air:HTML_export dallas$ python -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

You should then be able to open your web browser and tell it to go to the local page, which should be `localhost:8000`

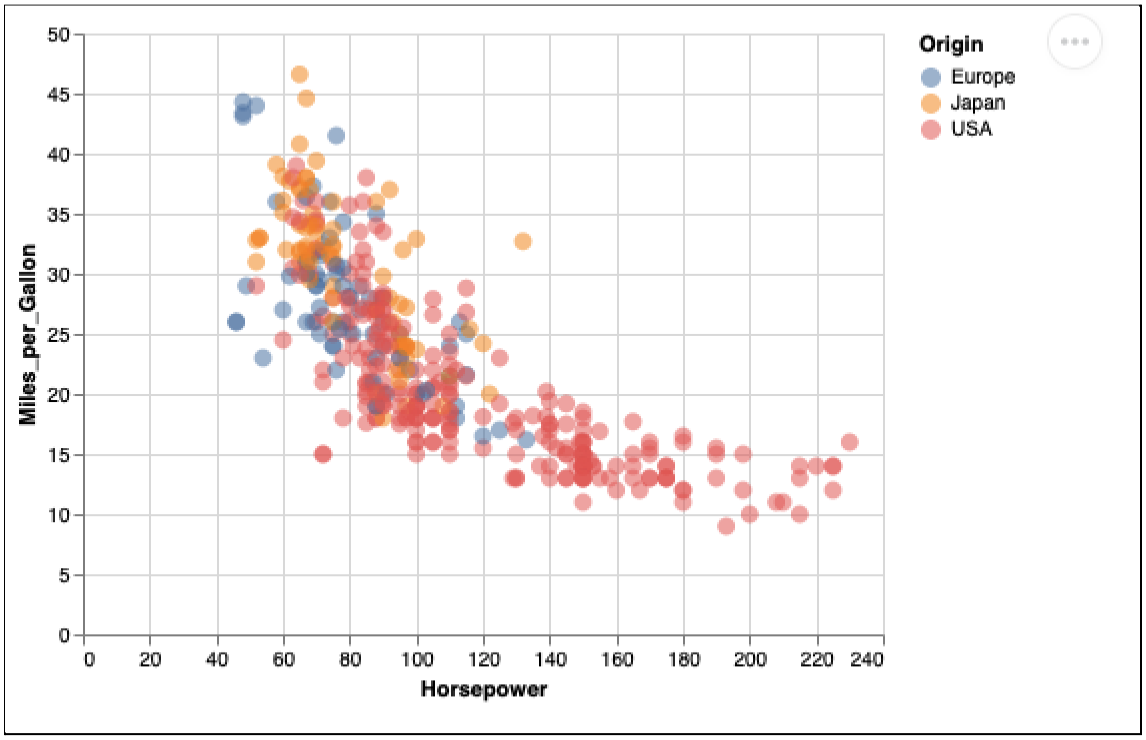
That should display a simple directory, with links, as shown below:



Directory listing for /

- .ipynb_checkpoints/
- cars.json
- chart.html
- chart.json
- chart.png
- chart.svg
- Exporting Altair to HTML.ipynb
- geckodriver.log

If you click on the `chart.html` file (or whatever you called it) in your browser, it should load a new page and show the visualization (see below).



To shut down the local web server, use Ctrl+C in the terminal.

4. Working with .json and HTML

You can also open the `.html` file in a text editor, and see what it's doing. You should see a file that looks something like this:

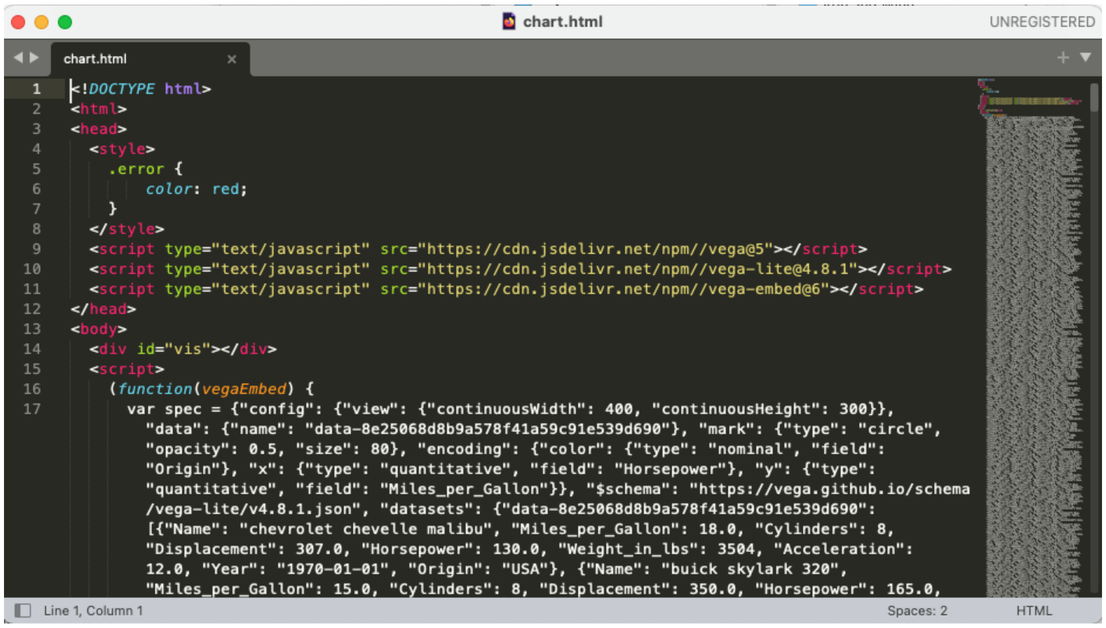


chart.html

UNREGISTERED

```
<!DOCTYPE html>
<html>
<head>
  <style>
    .error {
        color: red;
    }
  </style>
  <script type="text/javascript" src="https://cdn.jsdelivr.net/npm//vega@5"></script>
  <script type="text/javascript" src="https://cdn.jsdelivr.net/npm//vega-lite@4.8.1"></script>
  <script type="text/javascript" src="https://cdn.jsdelivr.net/npm//vega-embed@6"></script>
</head>
<body>
  <div id="vis"></div>
  <script>
    (function(vegaEmbed) {
      var spec = {"config": {"view": {"continuousWidth": 400, "continuousHeight": 300}},
        "data": {"name": "data-8e25068d8b9a578f41a59c91e539d690"}, "mark": {"type": "circle",
        "opacity": 0.5, "size": 80}, "encoding": {"color": {"type": "nominal", "field":
        "Origin"}, "x": {"type": "quantitative", "field": "Horsepower"}, "y": {"type":
        "quantitative", "field": "Miles_per_Gallon"}}, "$schema": "https://vega.github.io/schema
        /vega-lite/v4.8.1.json", "datasets": {"data-8e25068d8b9a578f41a59c91e539d690":
        [{"Name": "chevrolet chevelle malibu", "Miles_per_Gallon": 18.0, "Cylinders": 8,
        "Displacement": 307.0, "Horsepower": 130.0, "Weight_in_lbs": 3504, "Acceleration":
        12.0, "Year": "1970-01-01", "Origin": "USA"}, {"Name": "buick skylark 320",
        "Miles_per_Gallon": 15.0, "Cylinders": 8, "Displacement": 350.0, "Horsepower": 165.0,
```

Line 1, Column 1 Spaces: 2 HTML

Basically, the first part is standard HTML. It also loads some javascript libraries, and then calls a function (`vegaEmbed`), which your web browser uses to convert the `.json` specification into a visualization.

To have more control over the output (for example, to put two visualizations on the same webpage), we can export the visualizations to `.json` files, and manually write or edit a basic HTML webpage.

4.1 Creating a basic HTML page

We can create an extremely basic webpage with the following html code:

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<p>Hello World</p>
</body>
</html>
```

If you save that as an html file (which you should put in the same directory as the Jupyter notebook for this example), you should be able to open it and see the text in the body (Hello World).

4.2 Using Vega-Lite

To be able to embed Altair visualizations in a webpage, we need to tell it to load certain javascript libraries, specifically those for Vega-Lite. We can do this by inserting the following code between `<head></head>` tags (there will be a complete example at the end):

```
<script type="text/javascript"
src="https://cdn.jsdelivr.net/npm//vega@5"></script>
<script type="text/javascript"
src="https://cdn.jsdelivr.net/npm//vega-lite@4.8.1"></script>
<script type="text/javascript"
src="https://cdn.jsdelivr.net/npm//vega-embed@6"></script>
```

4.3 Embedding a visualization

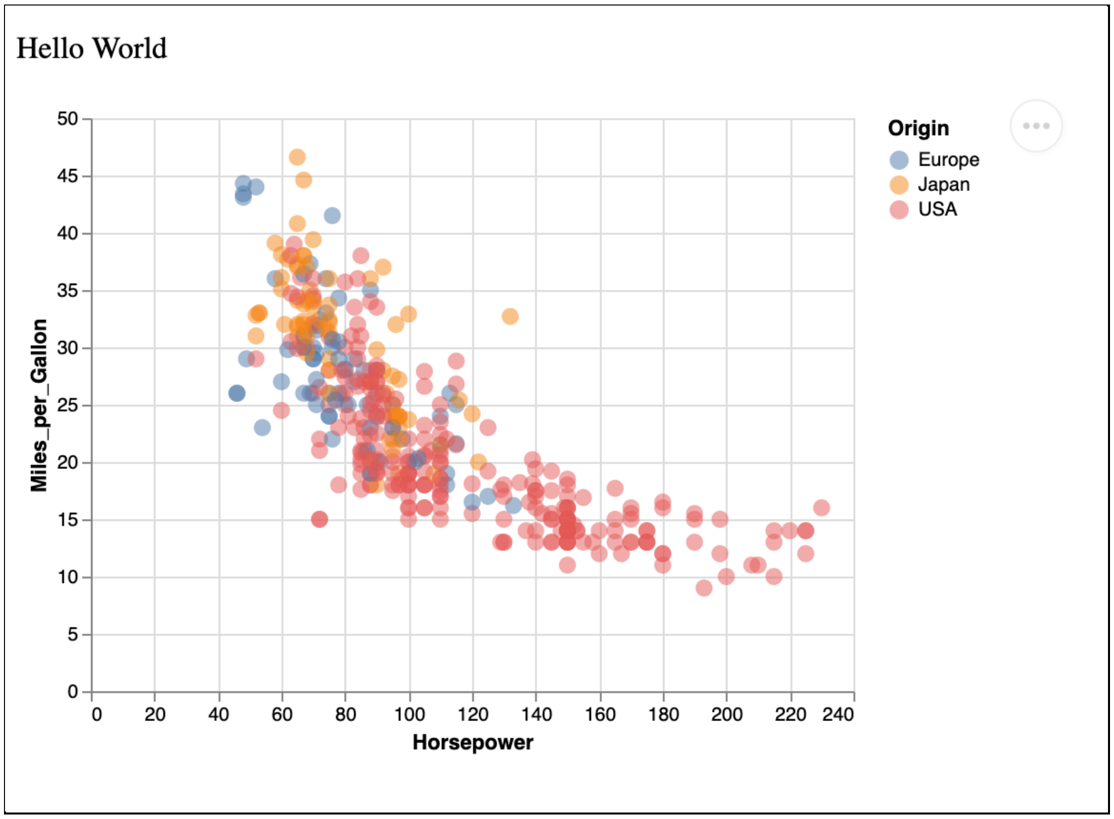
Finally, we'll embed our visualization that we saved as a `.json` file into the web page, putting it between the `<body></body>` tags. To do so, we first need to add a `<div>`, and give it an `id` (which can be anything we like), e.g.:

```
<div id="viz1"></div>
```

We then need to call a Vega script, and tell it to use that div to load the `.json` file which contains our chart (which we exported from the Jupyter notebook). We can do that with the following code (again, between the `<body></body>` tags):

```
<script>
  vegaEmbed("#viz1", "./chart.json");
</script>
```

If you combine all of these together (see below), you should then be able to start the web server (`python -m http.server`) and your new file should display both the Hello World and the visualization, like so:


Hello World

Here is the complete example html:

```
<!DOCTYPE html>
<html>
<head>
      <script type="text/javascript"
src="https://cdn.jsdelivr.net/npm//vega@5"></script>
      <script type="text/javascript"
src="https://cdn.jsdelivr.net/npm//vega-lite@4.8.1"></script>
      <script type="text/javascript"
src="https://cdn.jsdelivr.net/npm//vega-embed@6"></script>
</head>
<body>
<p>Hello World</p>
<div id="viz1"></div>
<script>
  vegaEmbed("#viz1", "./chart.json");
</script>
</body>
</html>
```

4.5 Adding multiple visualizations

Once you've got the basic example working, it is easy to do more. Simply create a new div for each additional visualization you want to include (giving each one a unique id), and add a corresponding VegaEmbed script, eg.,

Create a new div with a new id:
```
<div id="combined"></div>
```

and add a new vegaEmbed command linking it to a `.json` file
```
    vegaEmbed("#combined", "./combined.json");
```

4.6 More efficient .json files with Data Pointers

One problem with the basic approach is that altair embedded the entire dataset into the `.json` file, which is more than we need. If we're making multiple visualizations with the same data, this means we'll be saving the data in every `.json` file.

To avoid this, we can change how we specify the data for an altair Chart so that it uses **data pointers**.

To do so, rather than loading the data into a pandas dataframe, and then passing that to Altair, we can pass the URL (or file system location) of the data directly to Altair. (Please see the example in the Jupyter notebook).

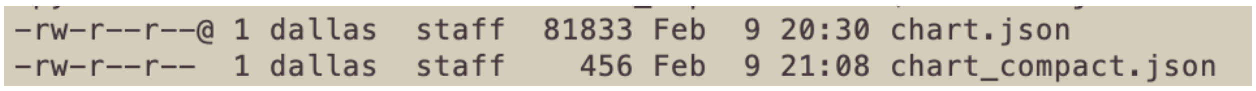
That is, when we create a chart, use

```
alt.Chart(url_of_data)
```

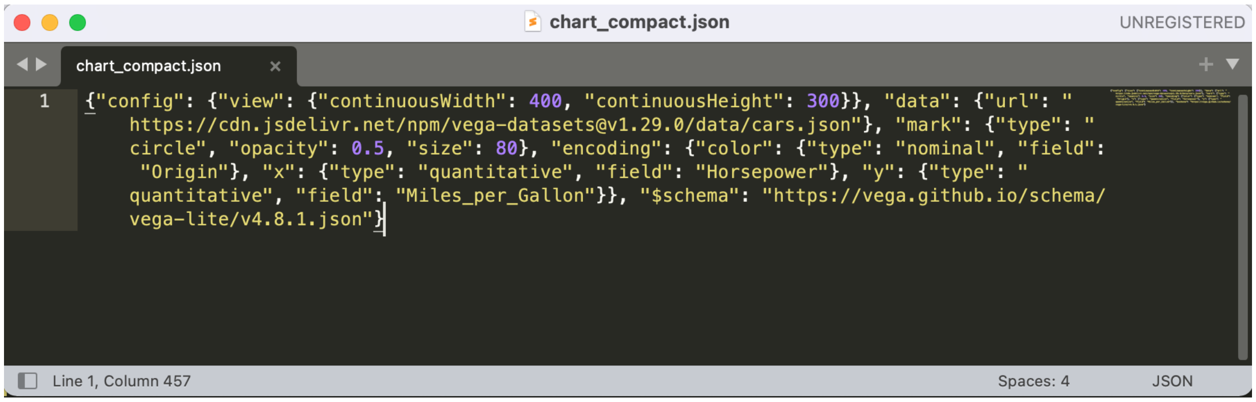
rather than

```
data = pd.load_json(url_of_data)
alt.Chart(data)
```

This is less efficient in some ways, as it will need to pull the data each time you create the visualization. It also means that the web server will need to be able to access the remote data in order to work. The advantage is that we're not embedding the entire dataset in the file, and so the `.json` file will be much smaller (see Jupyter notebook).


```
-rw-r--r--@ 1 dallas  staff  81833 Feb  9 20:30 chart.json
-rw-r--r--  1 dallas  staff    456 Feb  9 21:08 chart_compact.json
```

And if we look at the resulting `.json` file, we can see that it is much more compact, with only a URL, instead of the complete data:



```json
{"config": {"view": {"continuousWidth": 400, "continuousHeight": 300}}, "data": {"url": "
    https://cdn.jsdelivr.net/npm/vega-datasets@v1.29.0/data/cars.json"}, "mark": {"type": "
    circle", "opacity": 0.5, "size": 80}, "encoding": {"color": {"type": "nominal", "field":
     "Origin"}, "x": {"type": "quantitative", "field": "Horsepower"}, "y": {"type": "
    quantitative", "field": "Miles_per_Gallon"}}, "$schema": "https://vega.github.io/schema/
    vega-lite/v4.8.1.json"}
```