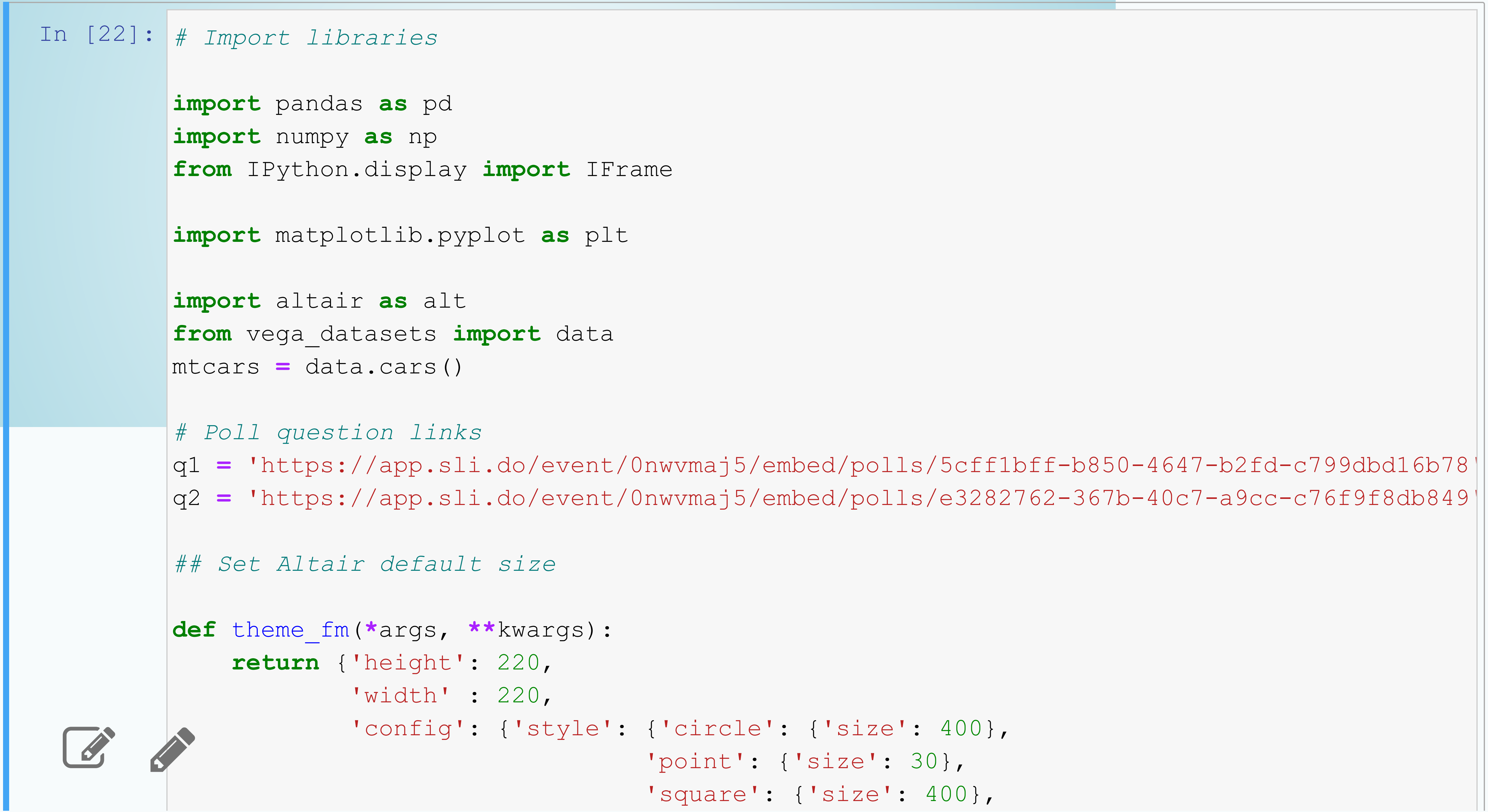


In [22]:

```
# Import libraries

import pandas as pd
import numpy as np
from IPython.display import IFrame

import matplotlib.pyplot as plt

import altair as alt
from vega_datasets import data
mtcars = data.cars()

# Poll question links
q1 = 'https://app.sli.do/event/0nwvmaj5/embed/polls/5cff1bff-b850-4647-b2fd-c799dbd16b78'
q2 = 'https://app.sli.do/event/0nwvmaj5/embed/polls/e3282762-367b-40c7-a9cc-c76f9f8db849'

## Set Altair default size

def theme_fm(*args, **kwargs):
    return {'height': 220,
            'width' : 220,
            'config': {'style': {'circle': {'size': 400},
                                 'point': {'size': 30},
                                 'square': {'size': 400},
```

LEARNING CONTEXT

LEARNING CONTEXT: VISUALIZATION I

- **Academic year**: Block 2 of MDS-V
    - Block 1: Platforms, Programming, Wrangling
    - ~ 120 students in the class

LEARNING CONTEXT: VISUALIZATION I

- **Academic year**: Block 2 of MDS-V
    - Block 1: Platforms, Programming, Wrangling
    - ~ 120 students in the class

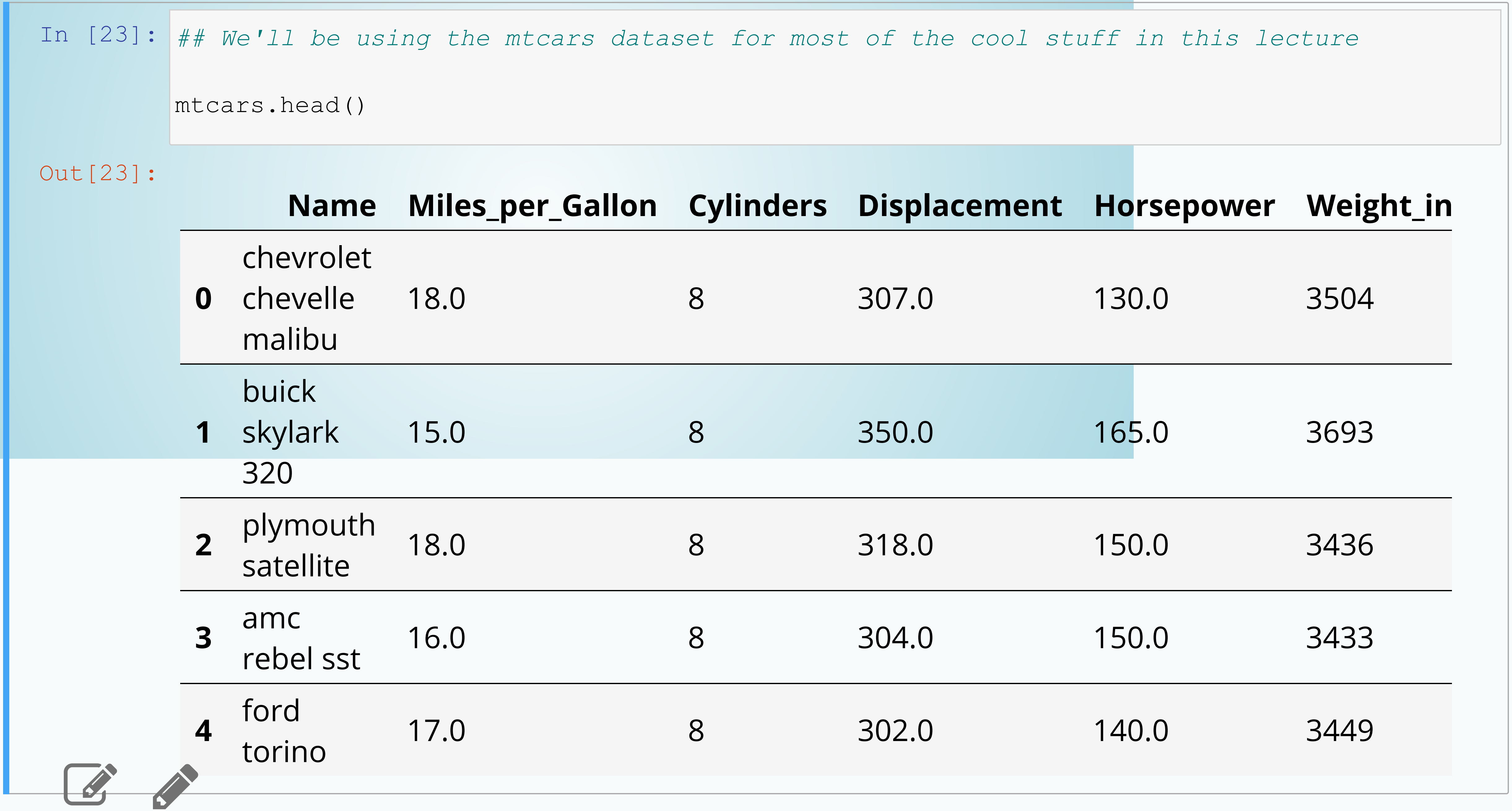
LEARNING CONTEXT: VISUALIZATION I

- **Academic year**: Block 2 of MDS-V
    - Block 1: Platforms, Programming, Wrangling
    - ~ 120 students in the class

LEARNING CONTEXT: VISUALIZATION I

- **Academic year**: Block 2 of MDS-V
    - Block 1: Platforms, Programming, Wrangling
    - ~ 120 students in the class

ALTAIR: DECLARATIVE VISUALIZATION IN PYTHON



```
In [23]: ## We'll be using the mtcars dataset for most of the cool stuff in this lecture

         mtcars.head()
```

Out[23]:

	Name	Miles_per_Gallon	Cylinders	Displacement	Horsepower	Weight_in
0	chevrolet chevelle malibu	18.0	8	307.0	130.0	3504
1	buick skylark 320	15.0	8	350.0	165.0	3693
2	plymouth satellite	18.0	8	318.0	150.0	3436
3	amc rebel sst	16.0	8	304.0	150.0	3433
4	ford torino	17.0	8	302.0	140.0	3449

LEARNING OBJECTIVES

LEARNING OBJECTIVES

- Explain the difference between declarative and imperative syntax

LEARNING OBJECTIVES

- Explain the difference between declarative and imperative syntax
- Describe the 6 components of the visualization grammar

LEARNING OBJECTIVES

- Explain the difference between declarative and imperative syntax
- Describe the 6 components of the visualization grammar
- Construct data visualizations using Altair

LEARNING OBJECTIVES

- Explain the difference between declarative and imperative syntax
- Describe the 6 components of the visualization grammar
- Construct data visualizations using Altair

LEARNING OBJECTIVES

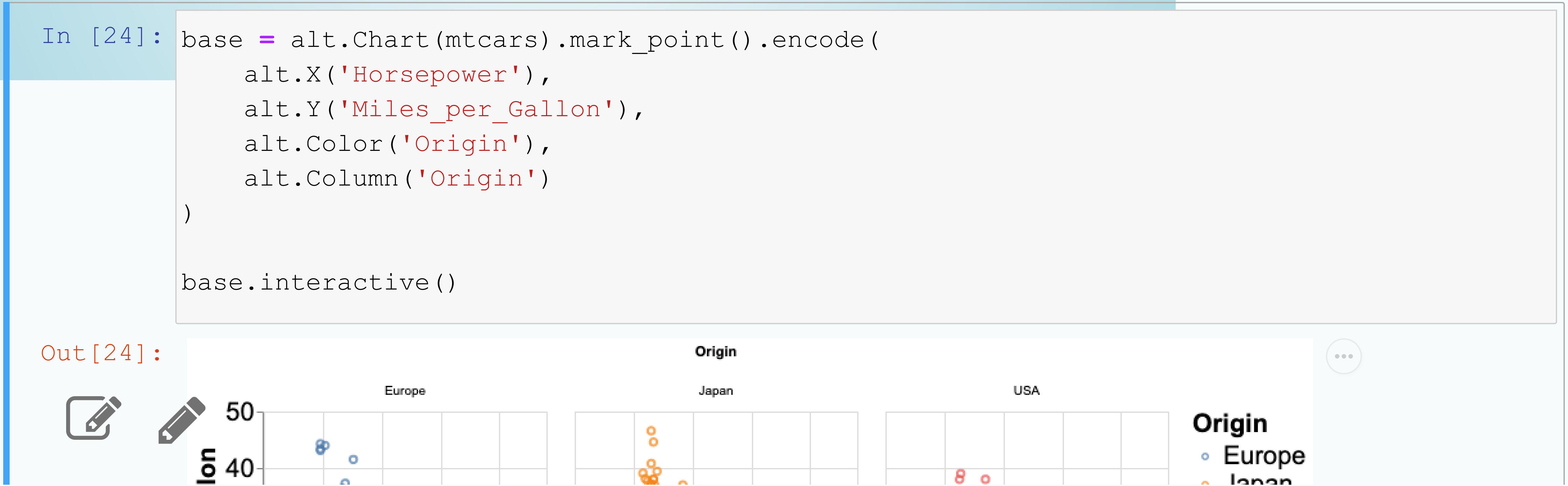
- Explain the difference between declarative and imperative syntax
- Describe the 6 components of the visualization grammar
- Construct data visualizations using Altair

STARTING WITH THE PUNCHLINE!

By the end of lecture today, you will learn how to make this chart using the `mtcars` dataset:

STARTING WITH THE PUNCHLINE!

By the end of lecture today, you will learn how to make this chart using the `mtcars` dataset:



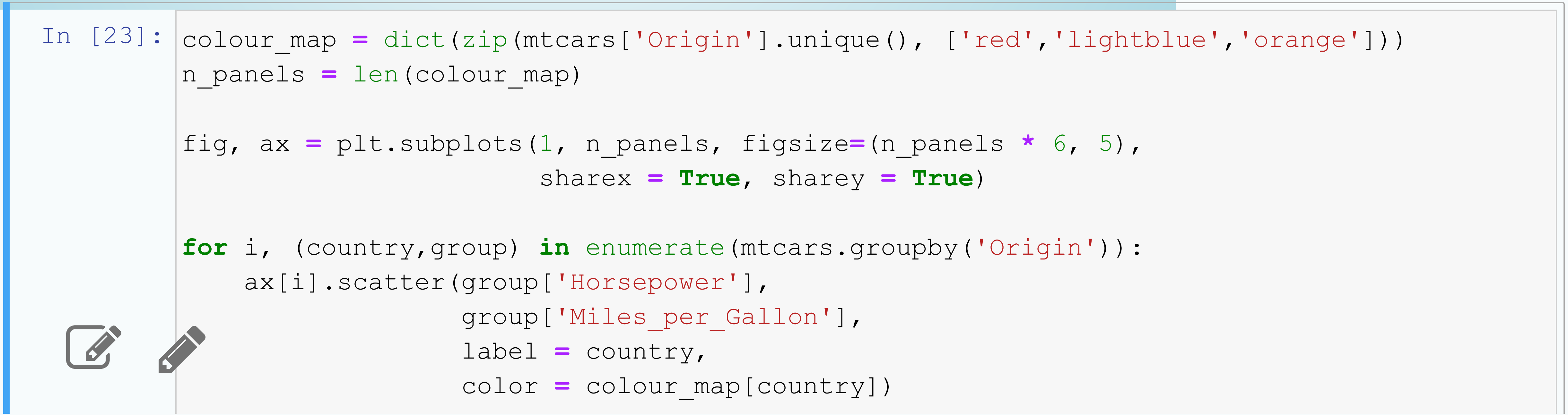
```
In [24]: base = alt.Chart(mtcars).mark_point().encode(
             alt.X('Horsepower'),
             alt.Y('Miles_per_Gallon'),
             alt.Color('Origin'),
             alt.Column('Origin')
         )

         base.interactive()
```

IN MATPLOTLIB:

If you're familiar with `matplotlib`, this should illustrate to you **how** Altair is different - not better or worse, just *differently sane* (h/t Greg Wilson).

IN MATPLOTLIB:

If you're familiar with `matplotlib`, this should illustrate to you **how** Altair is different - not better or worse, just *differently sane* (h/t Greg Wilson).

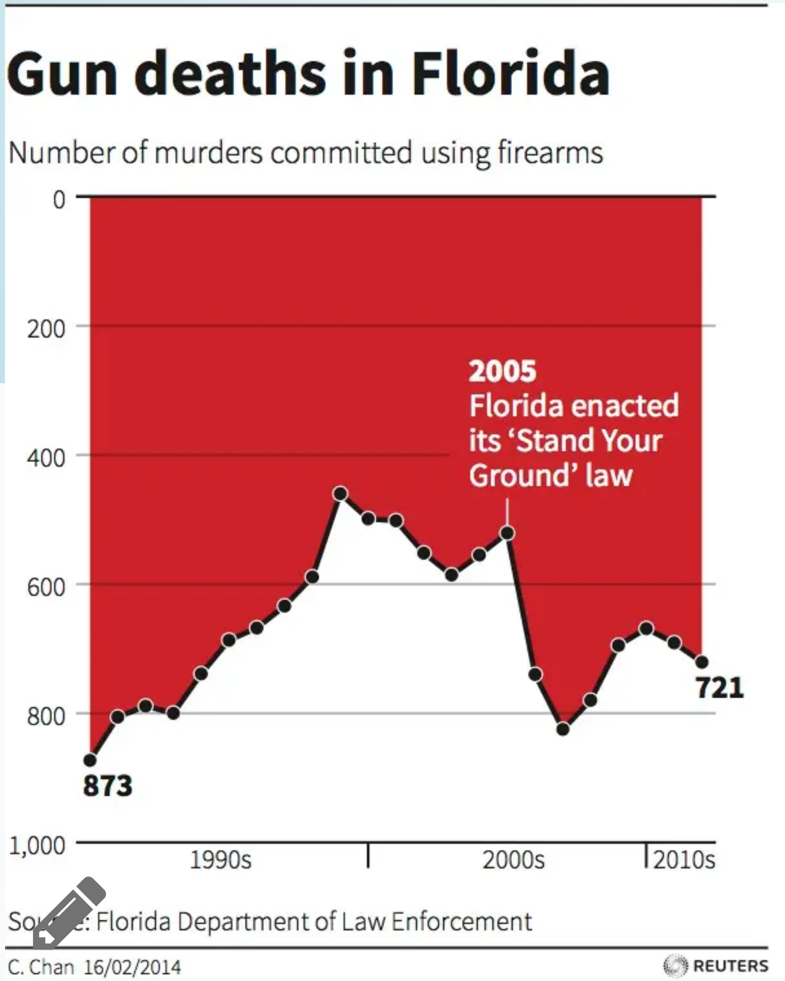


In [23]:

```
colour_map = dict(zip(mtcars['Origin'].unique(), ['red','lightblue','orange']))
n_panels = len(colour_map)

fig, ax = plt.subplots(1, n_panels, figsize=(n_panels * 6, 5),
                       sharex = True, sharey = True)

for i, (country,group) in enumerate(mtcars.groupby('Origin')):
    ax[i].scatter(group['Horsepower'],
                  group['Miles_per_Gallon'],
                  label = country,
                  color = colour_map[country])
```

PART 1: POWER OF DATA VISUALIZATIONS

CASE 1: GUN DEATHS IN FLORIDA



Gun deaths in Florida

Number of murders committed using firearms

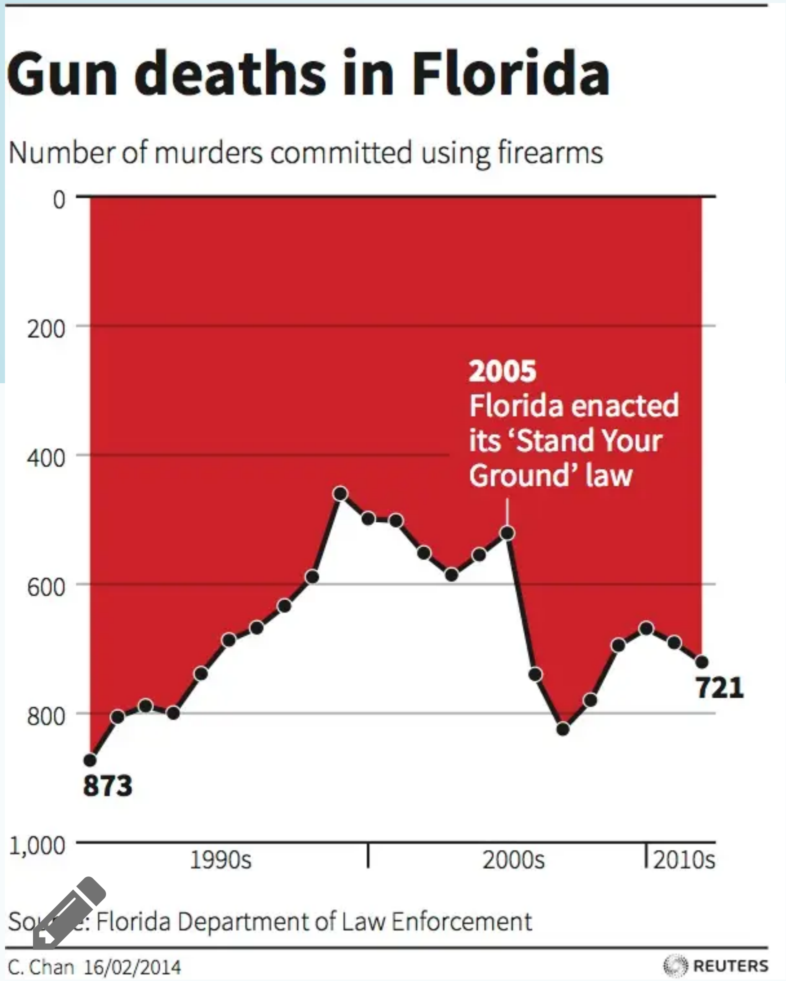
2005 Florida enacted its ‘Stand Your Ground’ law

Source: Florida Department of Law Enforcement

C. Chan 16/02/2014

REUTERS

CASE 1: GUN DEATHS IN FLORIDA

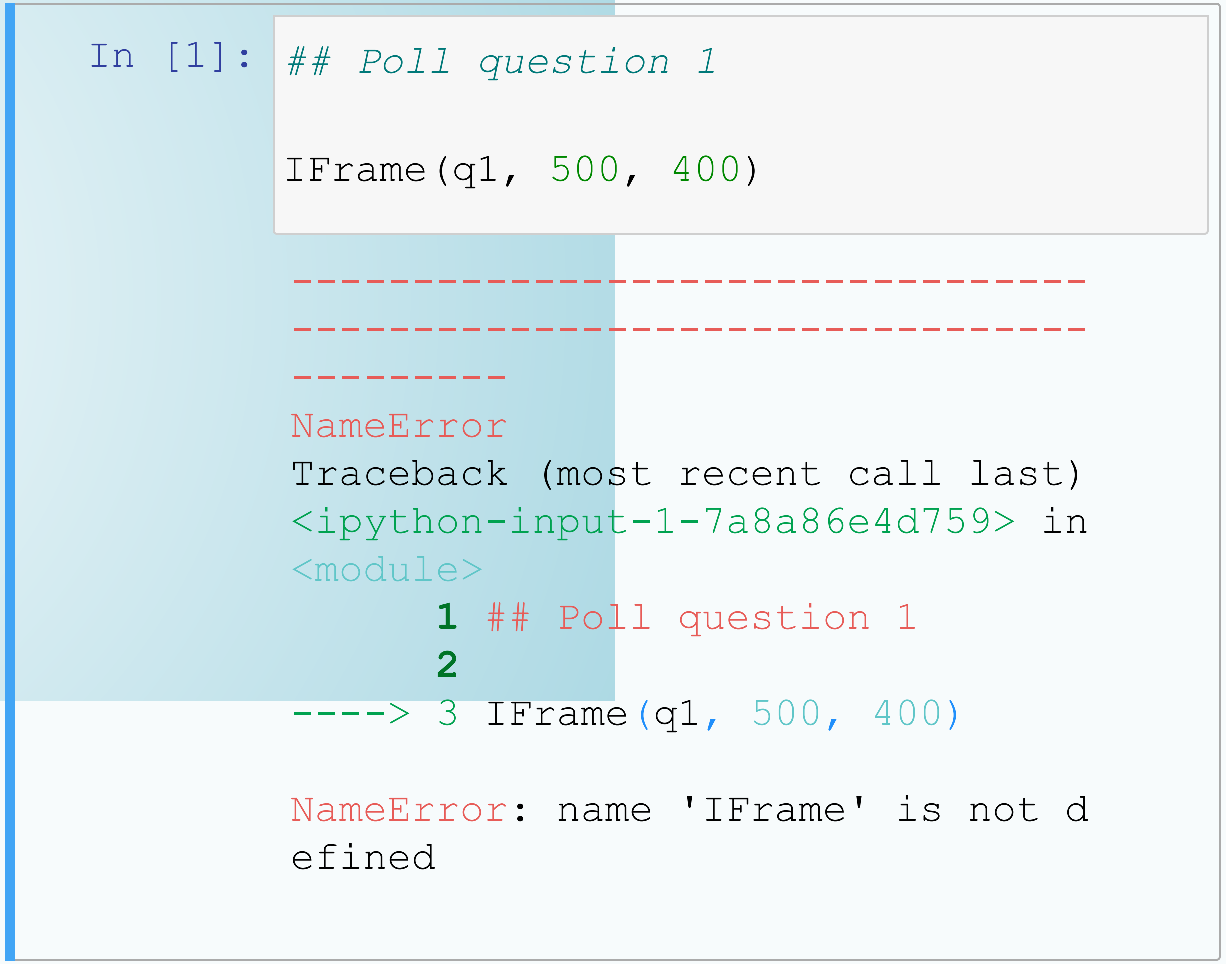


Gun deaths in Florida

Number of murders committed using firearms

Source: Florida Department of Law Enforcement

C. Chan 16/02/2014 REUTERS



```
In [1]: ## Poll question 1

        IFrame(q1, 500, 400)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-7a8a86e4d759> in <module>
      1 ## Poll question 1
      2 
----> 3 IFrame(q1, 500, 400)

NameError: name 'IFrame' is not defined
```

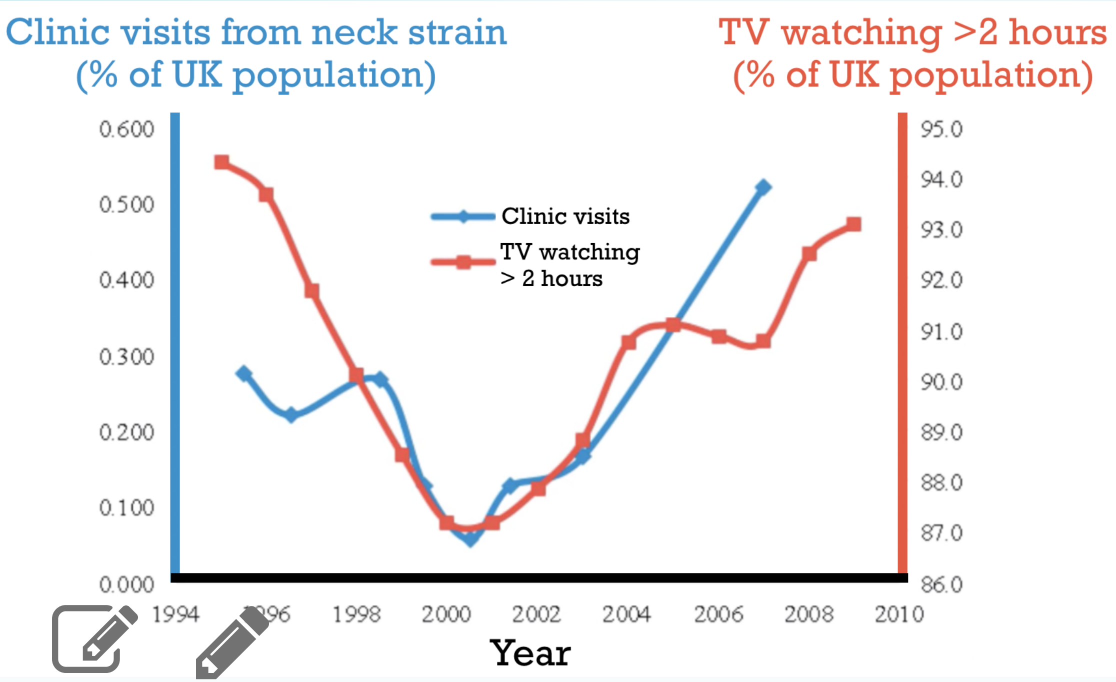
RESULTS!

The correct answer is:

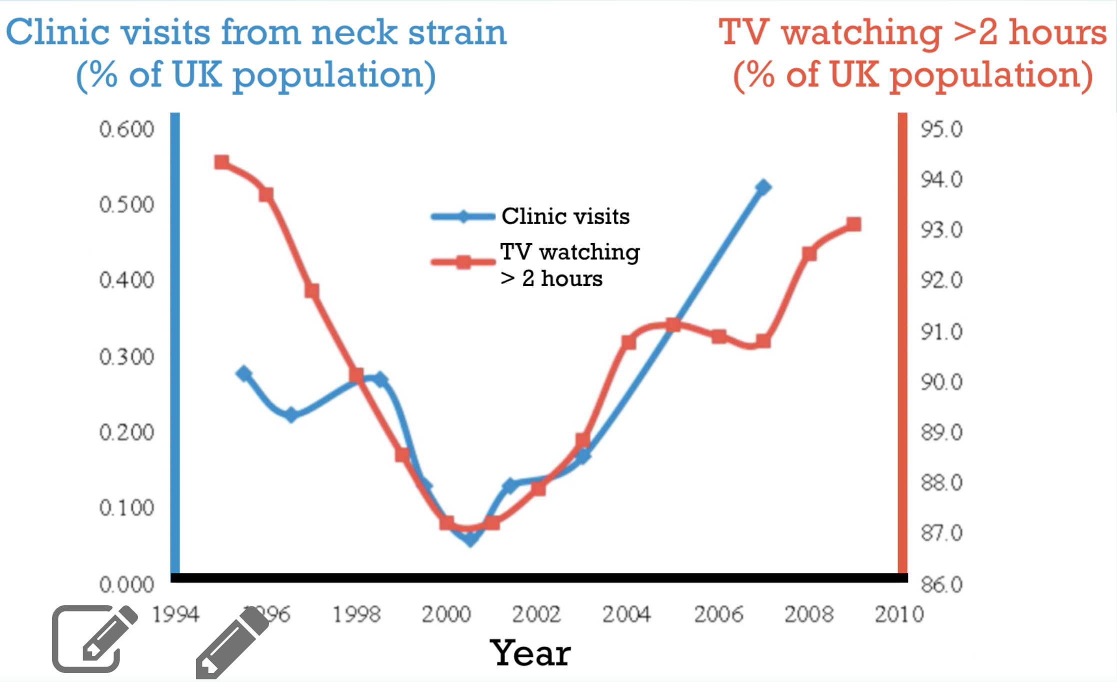
```
## Enter the answer here
```

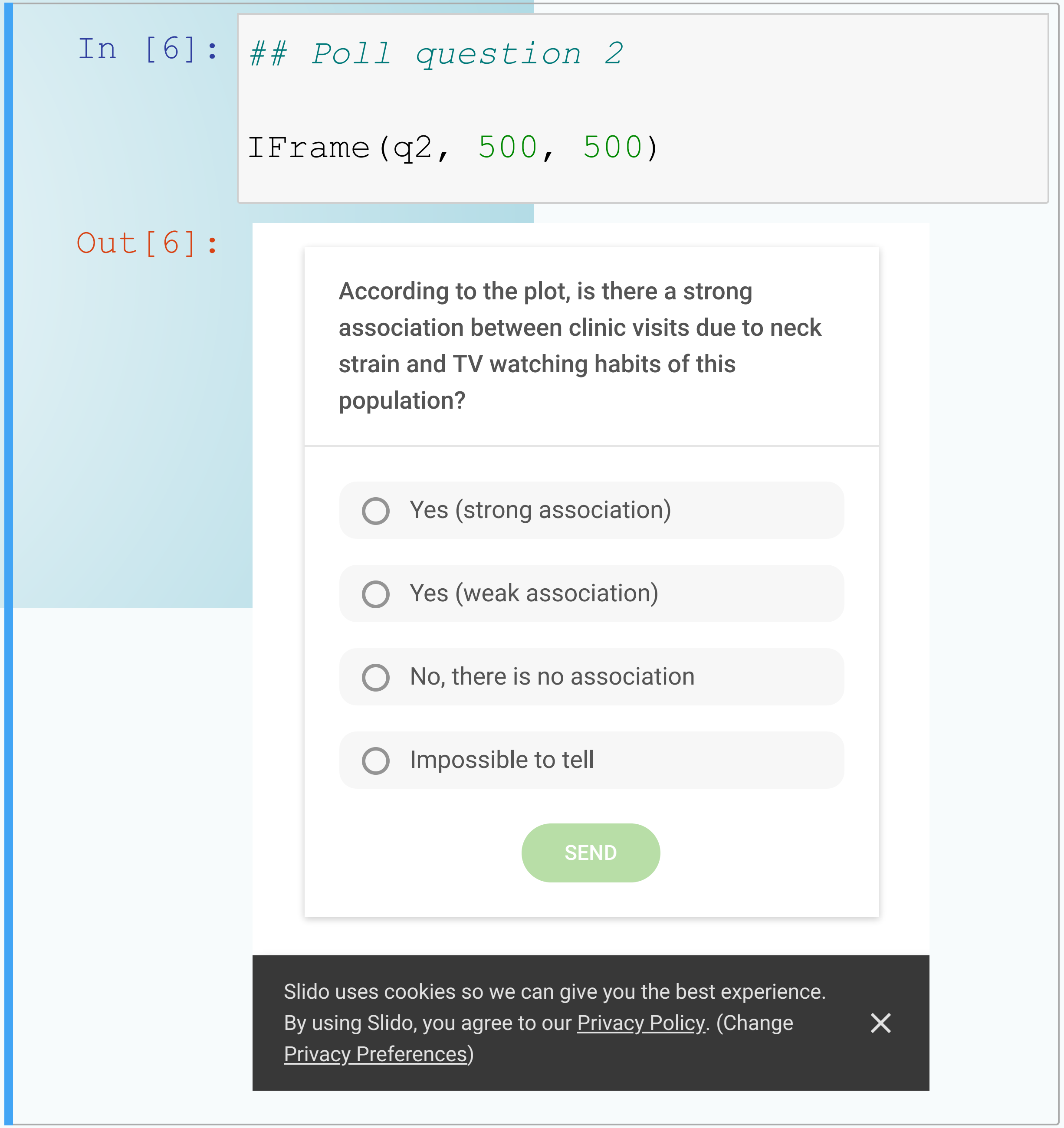
Here is the proper way to visualize the plot above:

CASE 2: CLINIC VISITS FOR NECK INJURIES AND TV-WATCHING HABITS



CASE 2: CLINIC VISITS FOR NECK INJURIES AND TV-WATCHING HABITS





```
In [6]: ## Poll question 2

        IFrame(q2, 500, 500)
```

Out[6]:

According to the plot, is there a strong association between clinic visits due to neck strain and TV watching habits of this population?

○ Yes (strong association)

○ Yes (weak association)

○ No, there is no association

○ Impossible to tell

Slido uses cookies so we can give you the best experience. By using Slido, you agree to our Privacy Policy. (Change Privacy Preferences)

RESULTS!

The correct answer is:

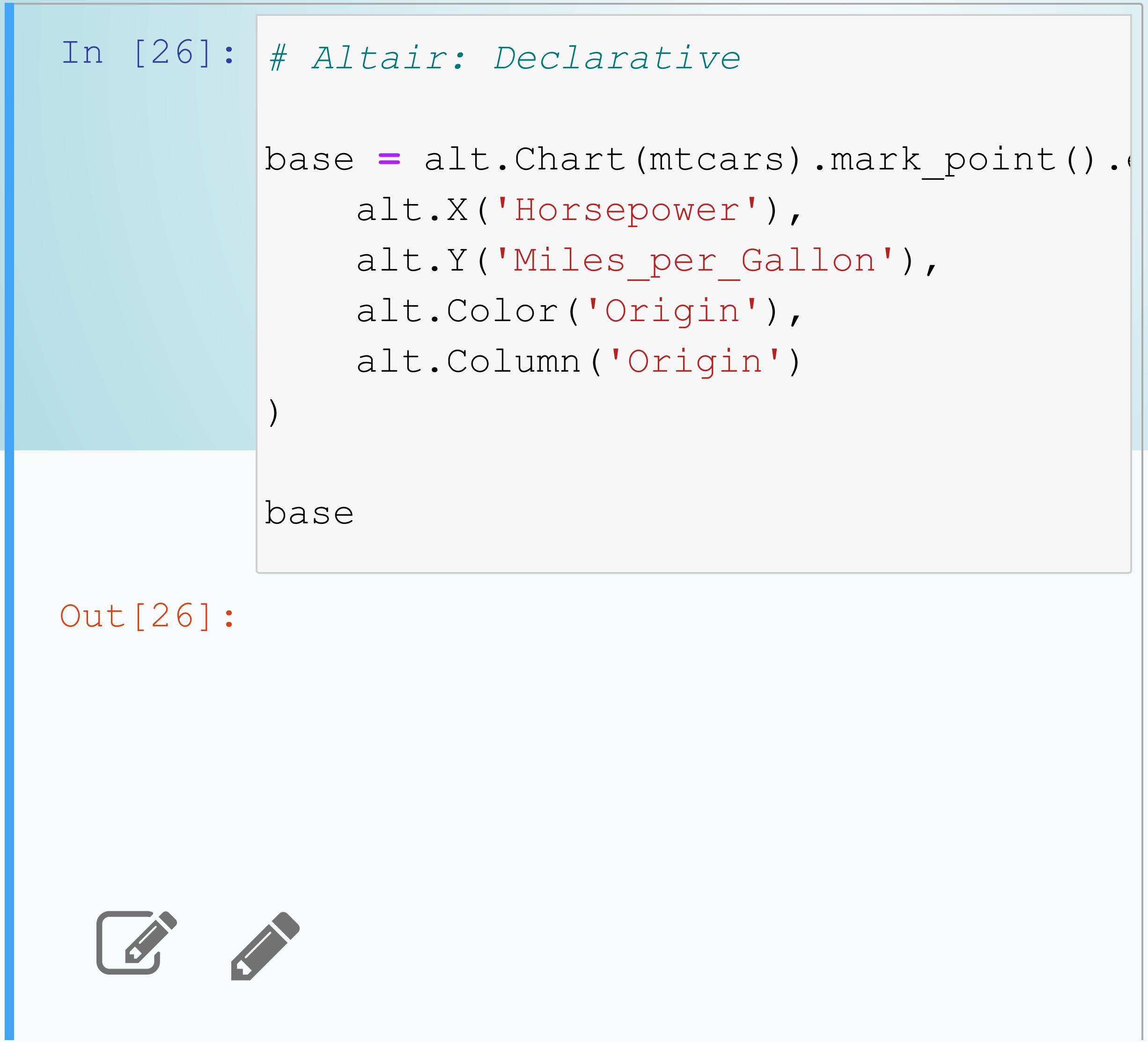
```
## Enter the answer here
```

Here is the actual plot with the real situation:

PART 2: INTRODUCTION TO ALTAIR

WHY DO WE NEED A VISUALIZATION GRAMMAR?

WHY DO WE NEED A VISUALIZATION GRAMMAR?

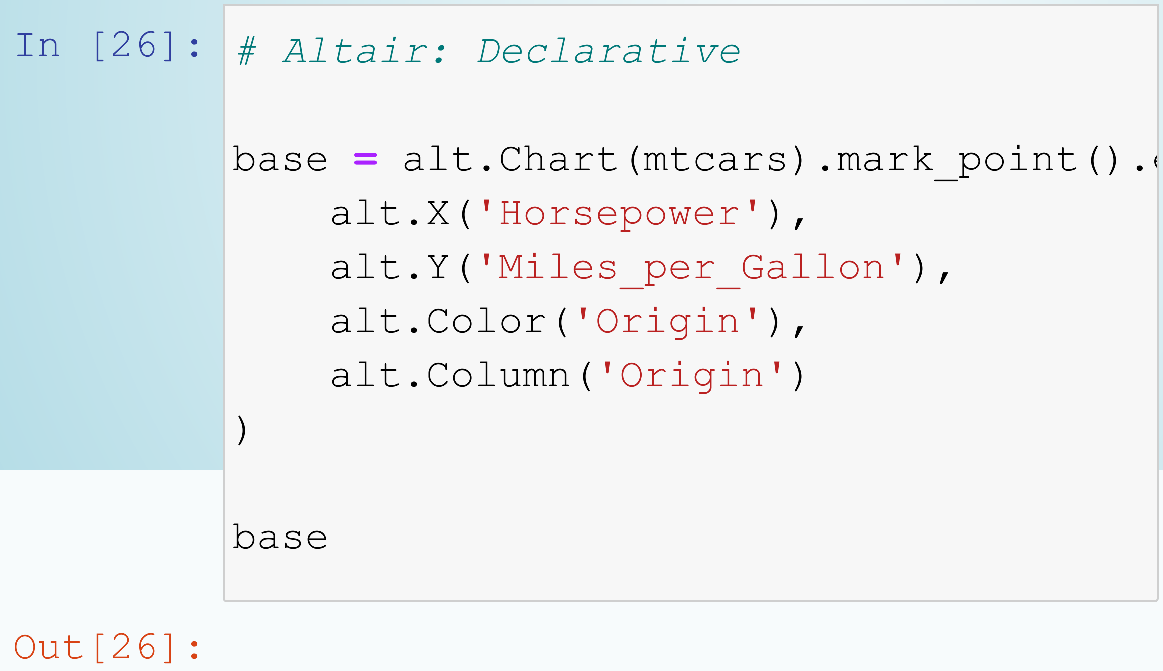


In [26]:

```python
# Altair: Declarative

base = alt.Chart(mtcars).mark_point().e
    alt.X('Horsepower'),
    alt.Y('Miles_per_Gallon'),
    alt.Color('Origin'),
    alt.Column('Origin')
)

base
```

Out[26]:

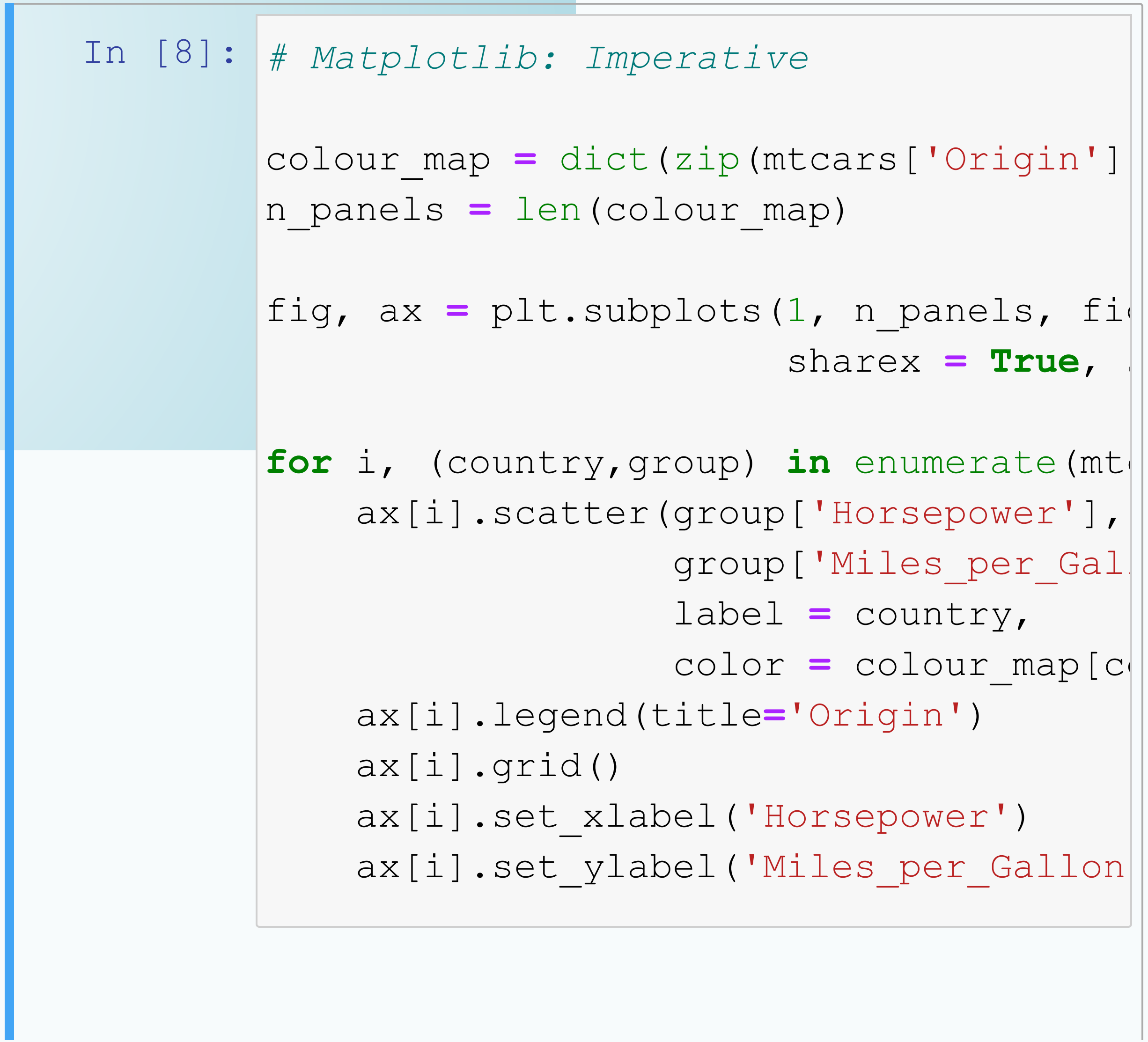
WHY DO WE NEED A VISUALIZATION GRAMMAR?



In [26]:

```
# Altair: Declarative

base = alt.Chart(mtcars).mark_point().e
    alt.X('Horsepower'),
    alt.Y('Miles_per_Gallon'),
    alt.Color('Origin'),
    alt.Column('Origin')
)

base
```

Out[26]:



In [8]:

```
# Matplotlib: Imperative

colour_map = dict(zip(mtcars['Origin']
n_panels = len(colour_map)

fig, ax = plt.subplots(1, n_panels, fi
                       sharex = True, 

for i, (country,group) in enumerate(mt
    ax[i].scatter(group['Horsepower'],
                  group['Miles_per_Gal
                  label = country,
                  color = colour_map[c
    ax[i].legend(title='Origin')
    ax[i].grid()
    ax[i].set_xlabel('Horsepower')
    ax[i].set_ylabel('Miles_per_Gallon
```

1. TABULAR DATA

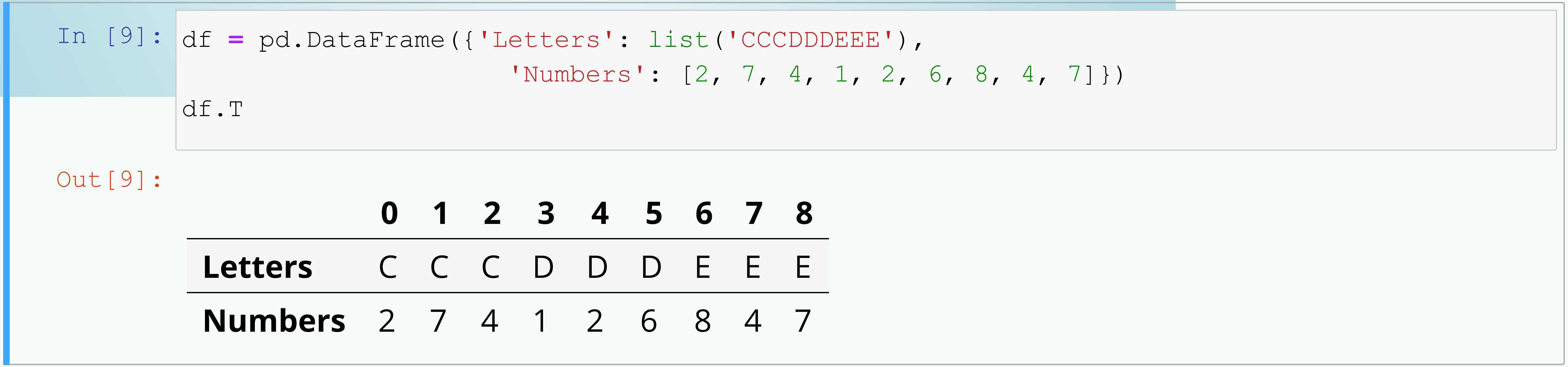
Data in Altair is built around the Pandas DataFrame.

The fundamental object in Altair is the `Chart`. It takes the dataframe as a single argument:

```
chart = alt.Chart(DataFrame)
```

Let's create a simple `DataFrame` to visualize, with a categorical data in the `Letters` column and numerical data in the `Numbers` column:

Let's create a simple `DataFrame` to visualize, with a categorical data in the `Letters` column and numerical data in the `Numbers` column:



In [9]:

```
df = pd.DataFrame({'Letters': list('CCCDDDEEE'),
                     'Numbers': [2, 7, 4, 1, 2, 6, 8, 4, 7]})
df.T
```

Out[9]:

	0	1	2	3	4	5	6	7	8
Letters	C	C	C	D	D	D	E	E	E
Numbers	2	7	4	1	2	6	8	4	7

Let's create a simple `DataFrame` to visualize, with a categorical data in the `Letters` column and numerical data in the `Numbers` column:

In [9]:
```
df = pd.DataFrame({'Letters': list('CCCDDDEEE'),
                   'Numbers': [2, 7, 4, 1, 2, 6, 8, 4, 7]})
df.T
```

Out[9]:

	0	1	2	3	4	5	6	7	8
Letters	C	C	C	D	D	D	E	E	E
Numbers	2	7	4	1	2	6	8	4	7

In [10]:
```
plot = alt.Chart(df)

#plot
```

2. CHART MARKS

Next we can decide what sort of *mark* we would like to use to represent our data.

Here are some of the more commonly used `mark_*()` methods supported in Altair and Vega-Lite; for

2. CHART MARKS

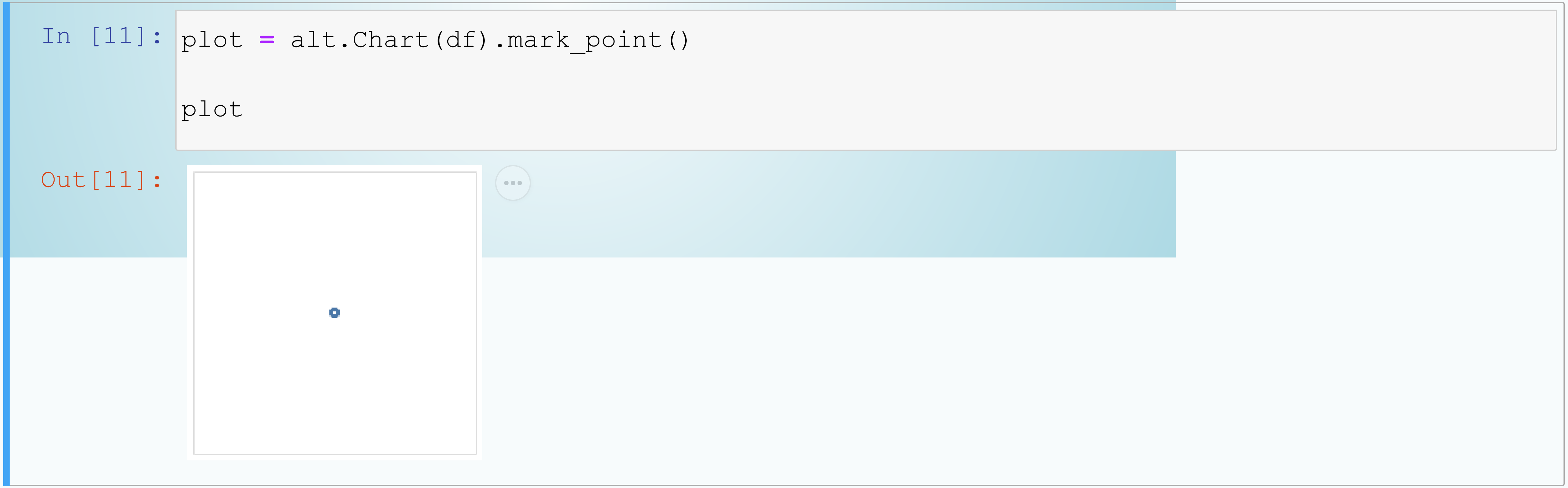
Next we can decide what sort of *mark* we would like to use to represent our data.

Here are some of the more commonly used `mark_*()` methods supported in Altair and Vega-Lite; for

Mark
`mark_area()`
`mark_bar()`
`mark_circle()`, `mark_point`, `mark_square`
`mark_rect()`
`mark_line()`

Let's add a mark_point() to our plot:

Let's add a mark_point() to our plot:

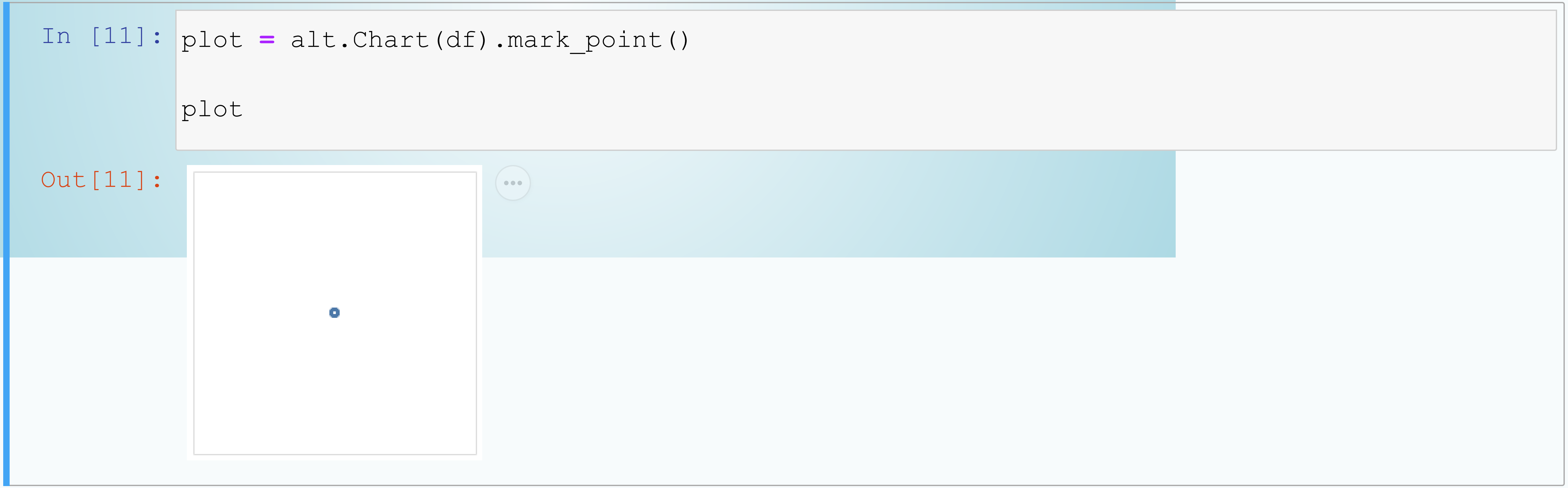


In [11]:

```
plot = alt.Chart(df).mark_point()

plot
```

Out[11]:

Let's add a mark_point() to our plot:



```
plot = alt.Chart(df).mark_point()

plot
```

A visual encoding specifies how a given data column should be **mapped** onto *visual properties* of the visualization.

Some of the more frequently used visual encodings are listed on the right:

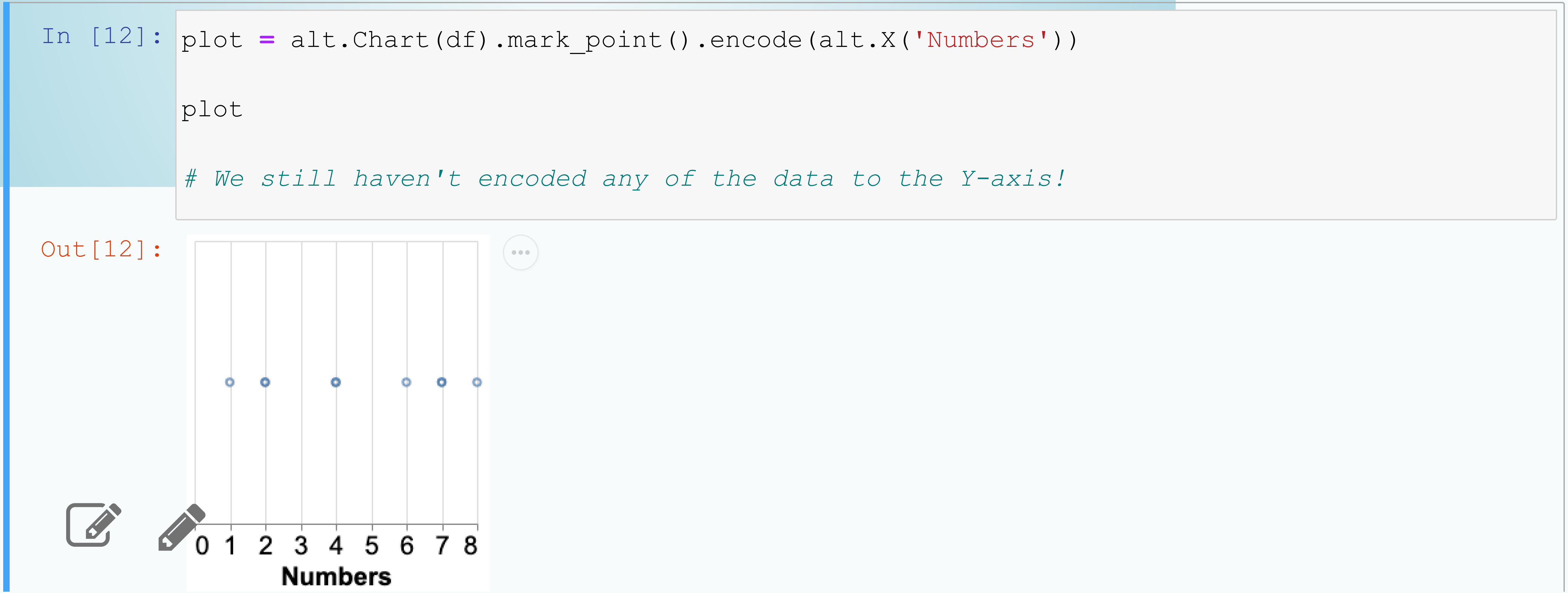
A visual encoding specifies how a given data column should be **mapped** onto *visual properties* of the visualization.

Some of the more frequently used visual encodings are listed on the right:

Encoding	What does it encode?
`X`	x-axis value
`Y`	y-axis value
`Color`	color of the m
`Opacity`	transparency/ of the mark
`Shape`	shape of the r

Let's add an encoding so the data is mapped to the x and y axes:

Let's add an encoding so the data is mapped to the x and y axes:



In [12]:

```
plot = alt.Chart(df).mark_point().encode(alt.X('Numbers'))

plot

# We still haven't encoded any of the data to the Y-axis!
```

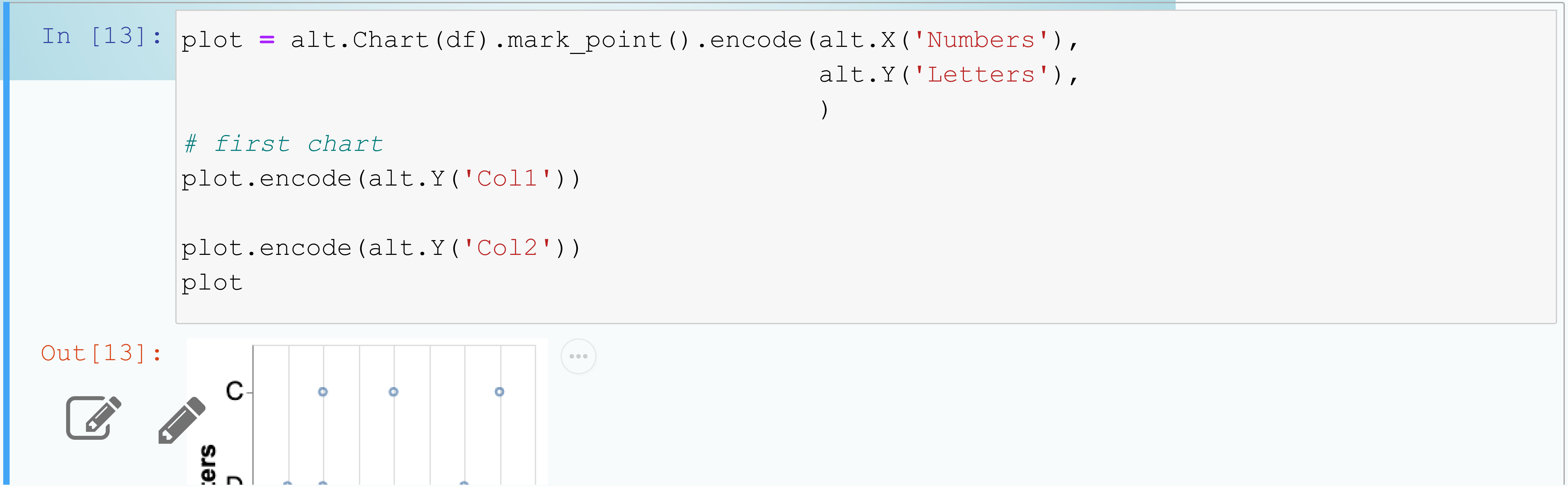
Out[12]:

YOU TRY!

Encode the `Letters` column at the `y` position to make the visualization more useful.

YOU TRY!

Encode the `Letters` column at the `y` position to make the visualization more useful.



In [13]:

```
plot = alt.Chart(df).mark_point().encode(alt.X('Numbers'),
                                         alt.Y('Letters'),
                                         )

# first chart
plot.encode(alt.Y('Col1'))

plot.encode(alt.Y('Col2'))
plot
```

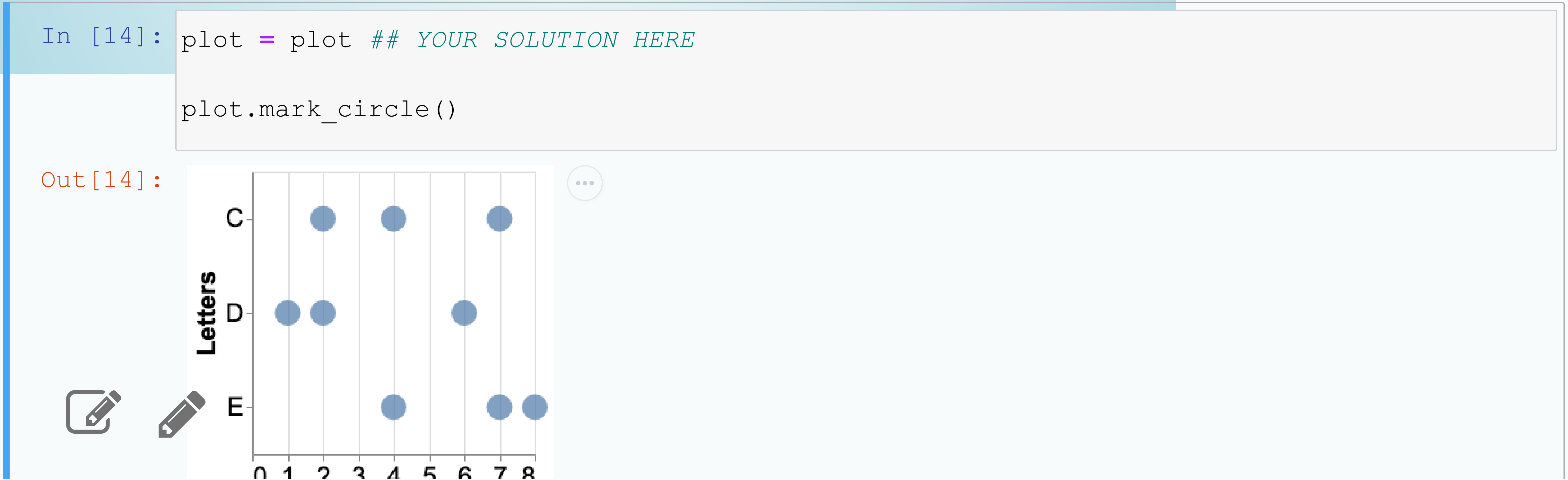
Out[13]:

YOU TRY!

Change the `mark` from `mark_point()` to `mark_circle` or `mark_square`

YOU TRY!

Change the `mark` from `mark_point()` to `mark_circle` or `mark_square`



In [14]:

```
plot = plot ## YOUR SOLUTION HERE

plot.mark_circle()
```

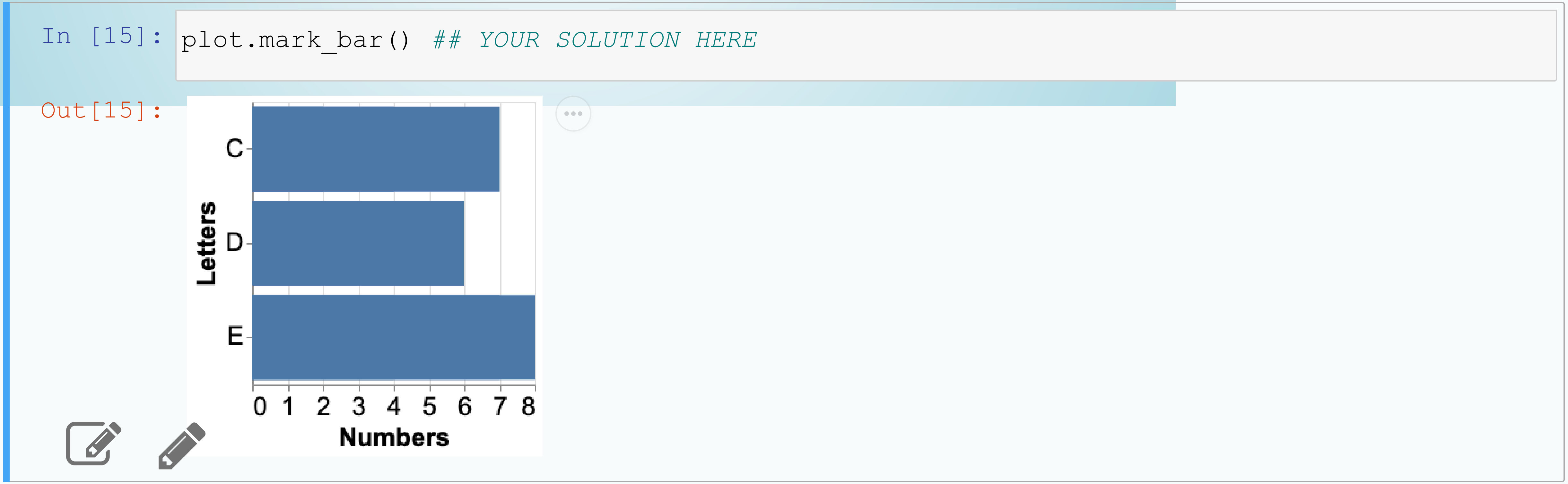
Out[14]:

YOU TRY!

What do you think will happen when you try to change the `mark_circle` to a `mark_bar()`

YOU TRY!

What do you think will happen when you try to change the `mark_circle` to a `mark_bar()`



```
In [15]: plot.mark_bar() ## YOUR SOLUTION HERE
```

Out[15]:

4. TRANSFORMS

Though Altair supports a few built-in data transformations and aggregations, in general I **do not suggest** you use them.

Some reasons why:

5. SCALE

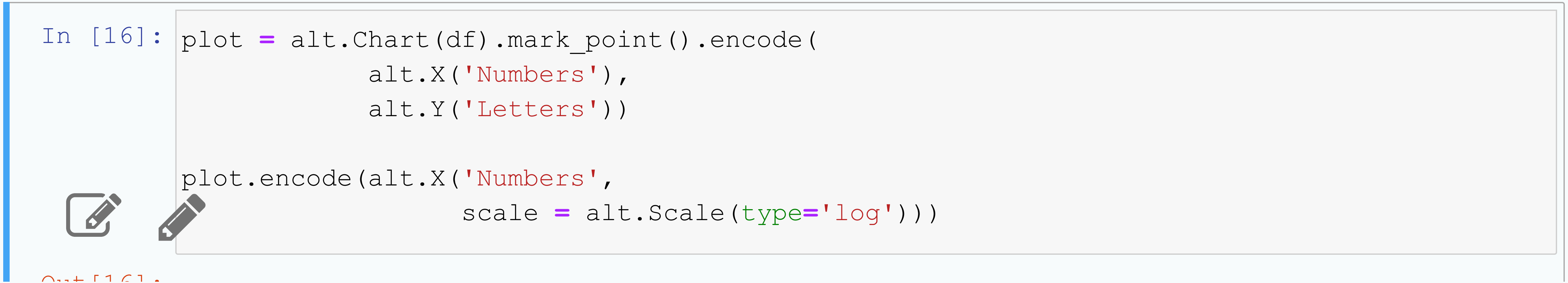
The scale parameter controls axis limits, axis types (`log`, `semi-log`, etc...).

For a complete description of the available options, see the Scales and Guides section of the documentation.

5. SCALE

The scale parameter controls axis limits, axis types (`log`, `semi-log`, etc...).

For a complete description of the available options, see the Scales and Guides section of the documentation.



In [16]:

```
plot = alt.Chart(df).mark_point().encode(
            alt.X('Numbers'),
            alt.Y('Letters'))

plot.encode(alt.X('Numbers',
                  scale = alt.Scale(type='log')))
```

Out[16]:

6. GUIDE

The guides component deals with legends and annotations that "guide" our interpretation of the data. In most cases you will not need to work with this component very much as the defaults are pretty good!

For a complete description of the available options, see the Scales and Guides section of the documentation.

APPLY THE VISUALIZATION GRAMMAR!

APPLY THE VISUALIZATION GRAMMAR!

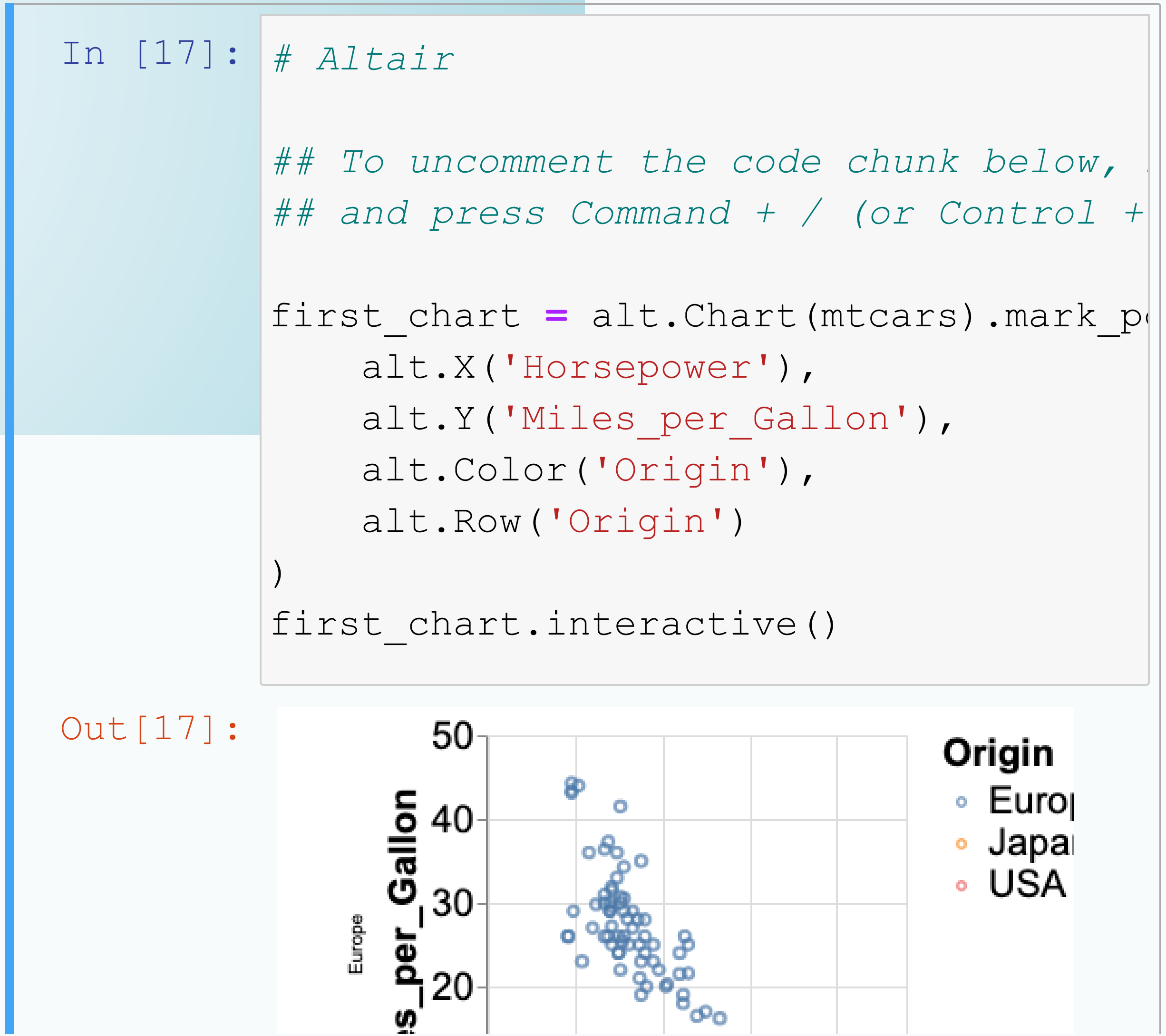
ACTIVITY:

Use the table below to create the visualization we started the lecture with (try not to scroll up to get the code unless you're really stuck!)

APPLY THE VISUALIZATION GRAMMAR!

ACTIVITY:

Use the table below to create the visualization we started the lecture with (try not to scroll up to get the code unless you're really stuck!)



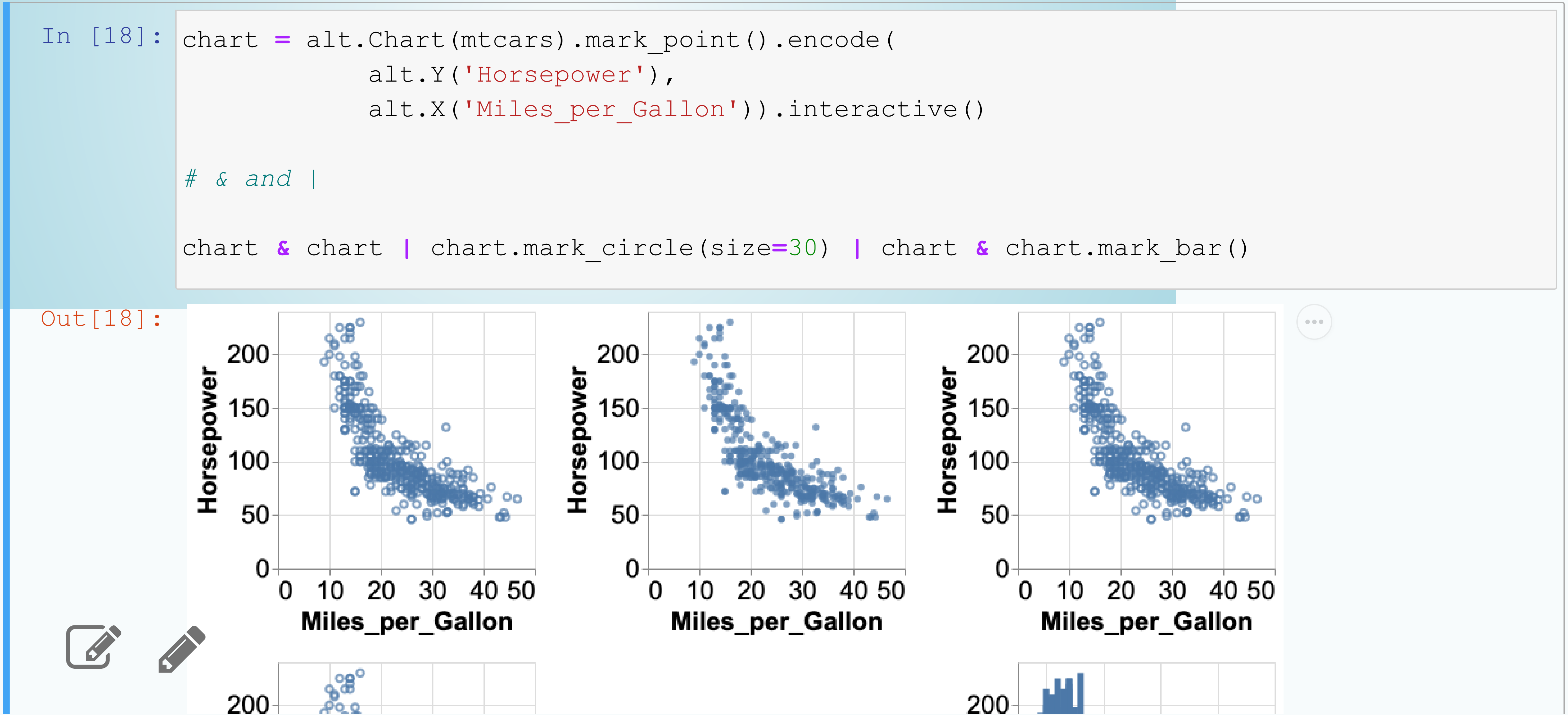
In [17]:

```
# Altair

## To uncomment the code chunk below,
## and press Command + / (or Control +

first_chart = alt.Chart(mtcars).mark_p
    alt.X('Horsepower'),
    alt.Y('Miles_per_Gallon'),
    alt.Color('Origin'),
    alt.Row('Origin')
)
first_chart.interactive()
```

Out[17]:

ONE MORE THING...

ONE MORE THING...



In [18]:

```python
chart = alt.Chart(mtcars).mark_point().encode(
            alt.Y('Horsepower'),
            alt.X('Miles_per_Gallon')).interactive()

# & and |

chart & chart | chart.mark_circle(size=30) | chart & chart.mark_bar()
```

Out[18]:

SUMMARY AND RECAP:

SUMMARY AND RECAP:

1. POWER OF VISUALIZATIONS

- Visualizations can be very effective in communicating complex ideas…
- But they can also be abused
- Responsible use of visualizations

SUMMARY AND RECAP:

1. POWER OF VISUALIZATIONS

- Visualizations can be very effective in communicating complex ideas...
- But they can also be abused
- Responsible use of visualizations

2. VISUALIZATION

SUMMARY AND RECAP:

1. POWER OF VISUALIZATIONS

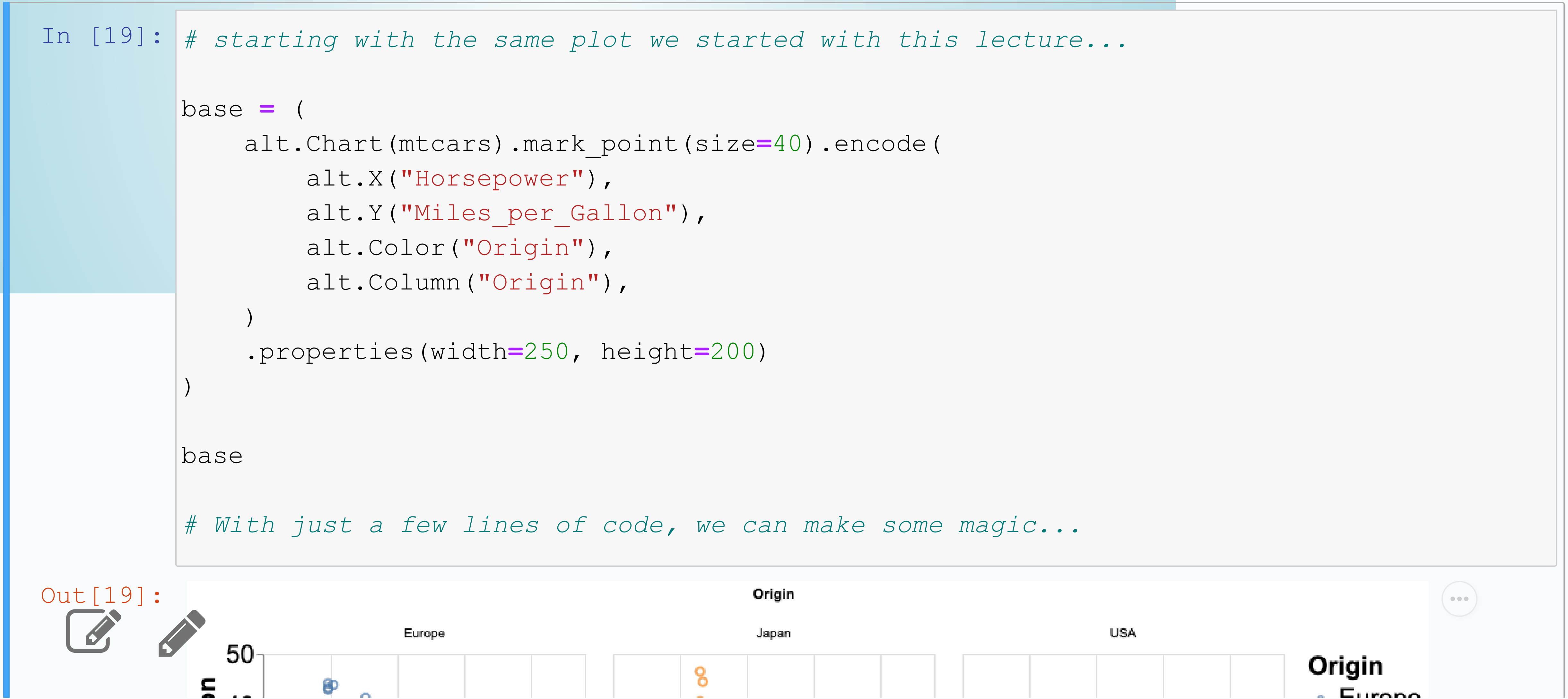
- Visualizations can be very effective in communicating complex ideas…
- But they can also be abused
- Responsible use of visualizations

2. VISUALIZATION

3. INTRODUCTION TO

NEXT CLASS ...

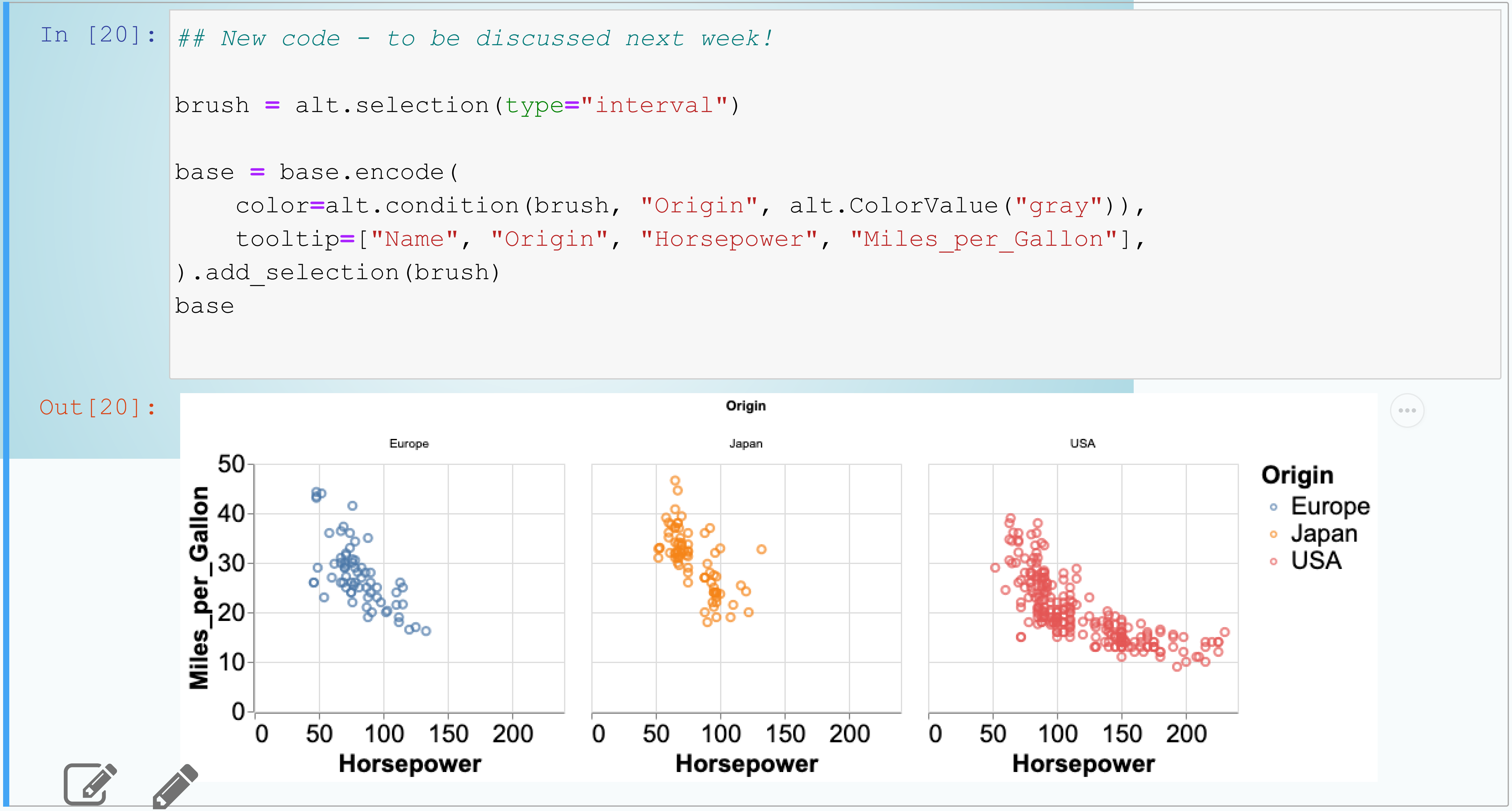
NEXT CLASS ...



In [19]:

```
# starting with the same plot we started with this lecture...

base = (
    alt.Chart(mtcars).mark_point(size=40).encode(
        alt.X("Horsepower"),
        alt.Y("Miles_per_Gallon"),
        alt.Color("Origin"),
        alt.Column("Origin"),
    )
    .properties(width=250, height=200)
)

base

# With just a few lines of code, we can make some magic...
```

Out[19]:



In [20]:

```
## New code - to be discussed next week!

brush = alt.selection(type="interval")

base = base.encode(
    color=alt.condition(brush, "Origin", alt.ColorValue("gray")),
    tooltip=["Name", "Origin", "Horsepower", "Miles_per_Gallon"],
).add_selection(brush)
base
```

Out[20]:

ACKNOWLEDGEMENTS