Module 7: 1D data

Let's first import basic packages and then load a dataset from `vega_datasets` package. If you don't have `vega_datasets` or `altair` installed yet, use `pip` or `conda` to install them.

```
[2]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

from vega_datasets import data
```

```
[3]: cars = data.cars()
cars.head()
```

Out[3]:

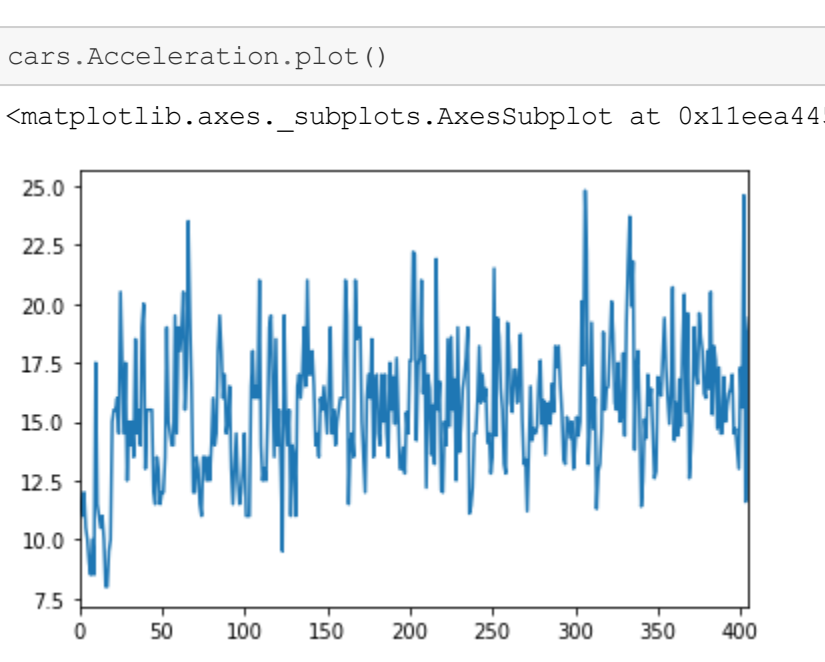
	Name	Miles_per_Gallon	Cylinders	Displacement	Horsepower	Weight_in_lbs	Acceleration	Year	Origin
0	chevrolet chevelle malibu	18.0	8	307.0	130.0	3504	12.0	1970-01-01	USA
1	buick skylark 320	15.0	8	350.0	165.0	3693	11.5	1970-01-01	USA
2	plymouth satellite	18.0	8	318.0	150.0	3436	11.0	1970-01-01	USA
3	amc rebel sst	16.0	8	304.0	150.0	3433	12.0	1970-01-01	USA
4	ford torino	17.0	8	302.0	140.0	3449	10.5	1970-01-01	USA

1D scatter plot

Let's consider the `Acceleration` column as our 1D data. If we ask pandas to plot this series, it'll produce a line graph where the index becomes the horizontal axis.

```
[3]: cars.Acceleration.plot()
```

```
Out[3]: <matplotlib.axes._subplots.AxesSubplot at 0x11eea4450>
```



Because the index is not really meaningful, drawing line between subsequent values is misleading! This is definitely not the plot we want!

It's actually not trivial to use pandas to create an 1-D scatter plot. Instead, we can use `matploblib`'s `scatter` function. We can first create an array with zeros that we can use as the vertical coordinates of the points that we will plot. `np.zeros_like` returns an array with zeros that matches the shape of the input array.

```
[4]: np.zeros_like([1,2,3])
```

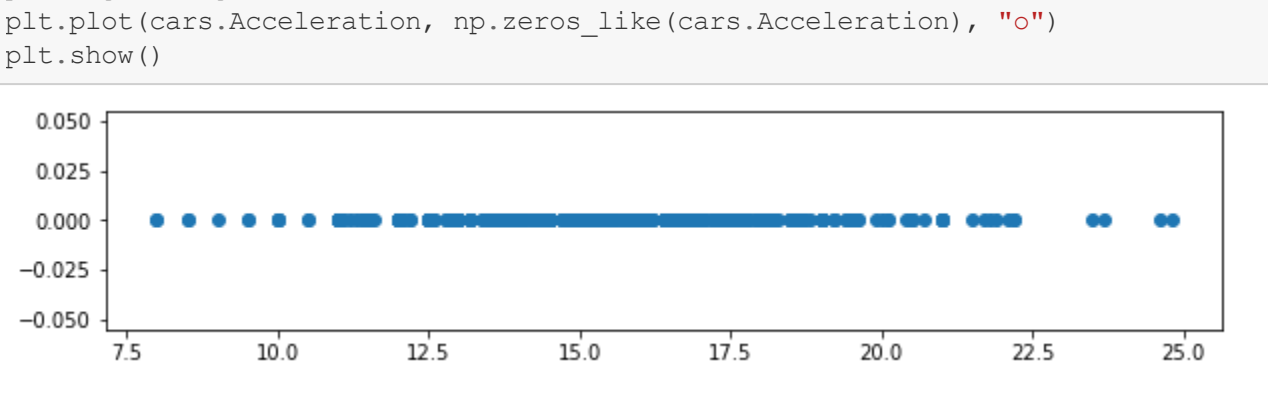
```
Out[4]: array([0, 0, 0])
```

Q: now can you create an 1D scatter plot wit `matplotlib`'s scatter function? Make the figure wide (e.g. set `figsize=(10,2)`) and then remove the y tics.

```
[5]: cars.Acceleration
```

```
Out[5]: 0      12.0
1      11.5
2      11.0
3      12.0
4      10.5
       ...
401    15.6
402    24.6
403    11.6
404    18.6
405    19.4
Name: Acceleration, Length: 406, dtype: float64
```

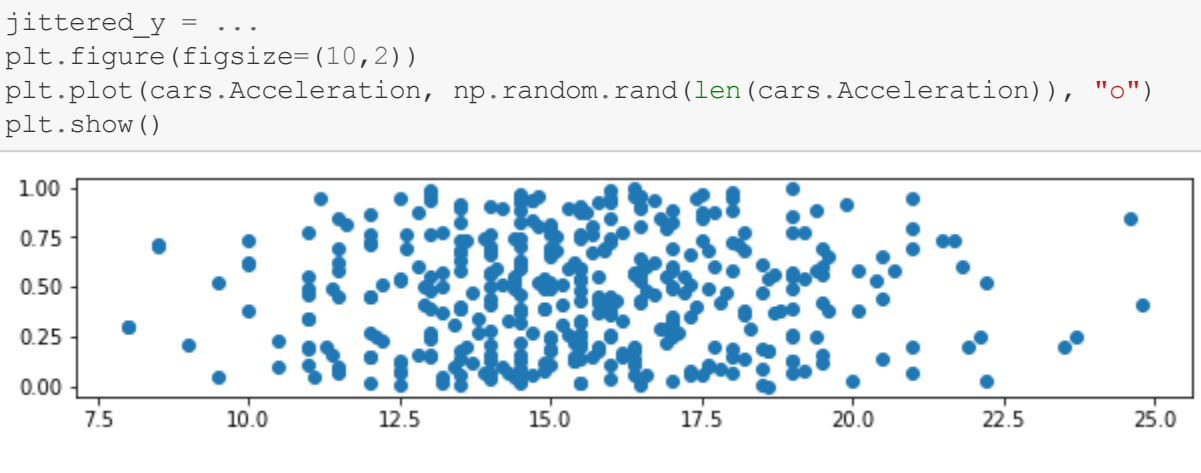
```
[22]: # TODO: put your code here
plt.figure(figsize=(10,2))
plt.plot(cars.Acceleration, np.zeros_like(cars.Acceleration), "o")
plt.show()
```



As you can see, there are lots of occlusions. So this plot cannot show the distribution properly and we would like to fix it. How about adding some jitters? You can use `numpy`'s `random.rand()` function to generate random numbers, instead of using an array with zeros.

Q: create a jittered 1D scatter plot.

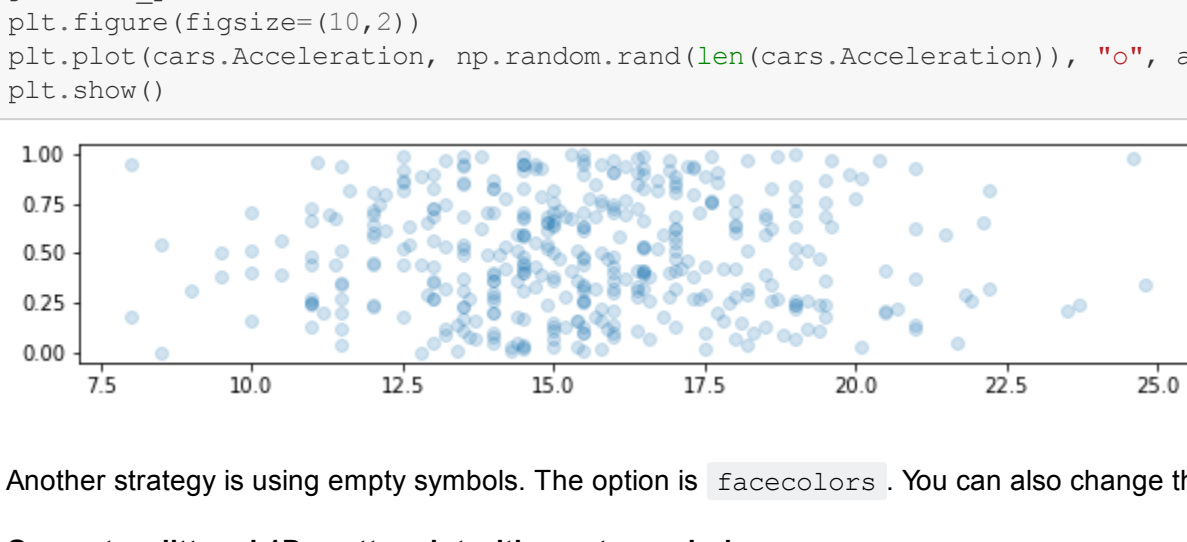
```
[27]: # TODO: put your code here
# jittered_y = ...
# plt ...

jittered_y = ...
plt.figure(figsize=(10,2))
plt.plot(cars.Acceleration, np.random.rand(len(cars.Acceleration)), "o")
plt.show()
```



We can further improve this by adding transparency to the symbols. The transparency option for `scatter` function is called `alpha`. Set it to be 0.2.

Q: create a jittered 1D scatter plot with transparency (alpha=0.2)

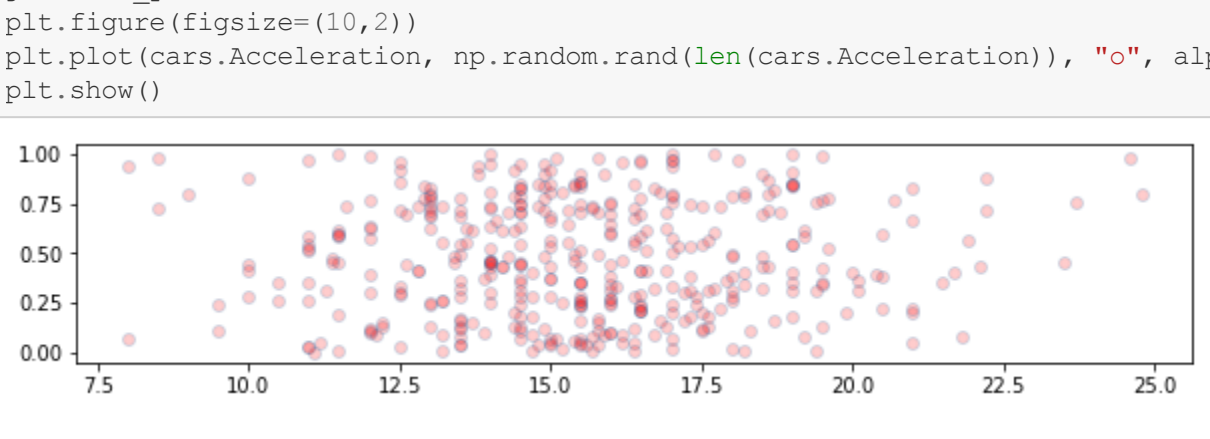
```
[4]: jittered_y = ...
plt.figure(figsize=(10,2))
plt.plot(cars.Acceleration, np.random.rand(len(cars.Acceleration)), "o", alpha=0.2)
plt.show()
```



Another strategy is using empty symbols. The option is `facecolors`. You can also change the stroke color (`edgecolors`).

Q: create a jittered 1D scatter plot with empty symbols.

```
[9]: # TODO: put your code here
jittered_y = ...
plt.figure(figsize=(10,2))
plt.plot(cars.Acceleration, np.random.rand(len(cars.Acceleration)), "o", alpha=0.2, mfc="red")
plt.show()
```

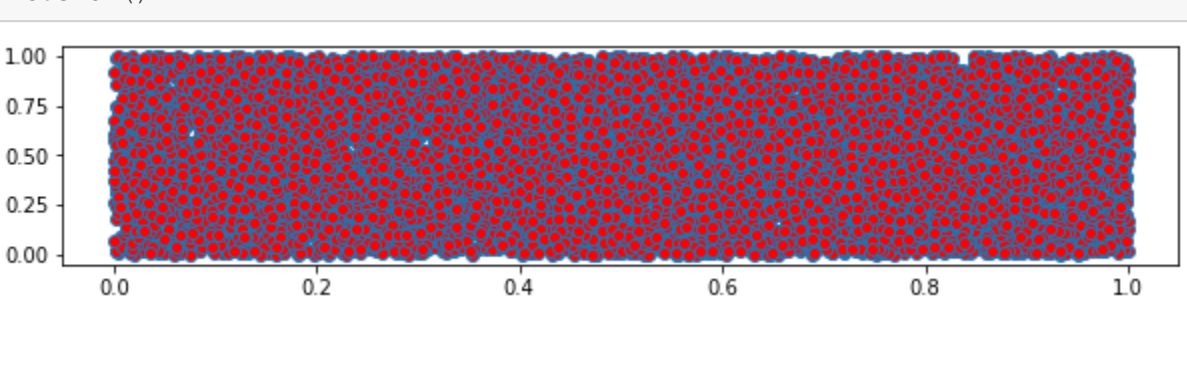


What happens if you have lots and lots of points?

Whatever strategy that you use, it's almost useless if you have too many data points. Let's play with different number of data points and see how it looks.

It not only becomes completely useless, it also take a while to draw the plot itself.

```
[16]: # TODO: play with N and see what happens.
N = 10000
x = np.random.rand(N)
y = np.random.rand(N)
plt.figure(figsize=(10,2))
plt.plot(x,y, "o", mfc="red")
plt.show()
```



Histogram and boxplot

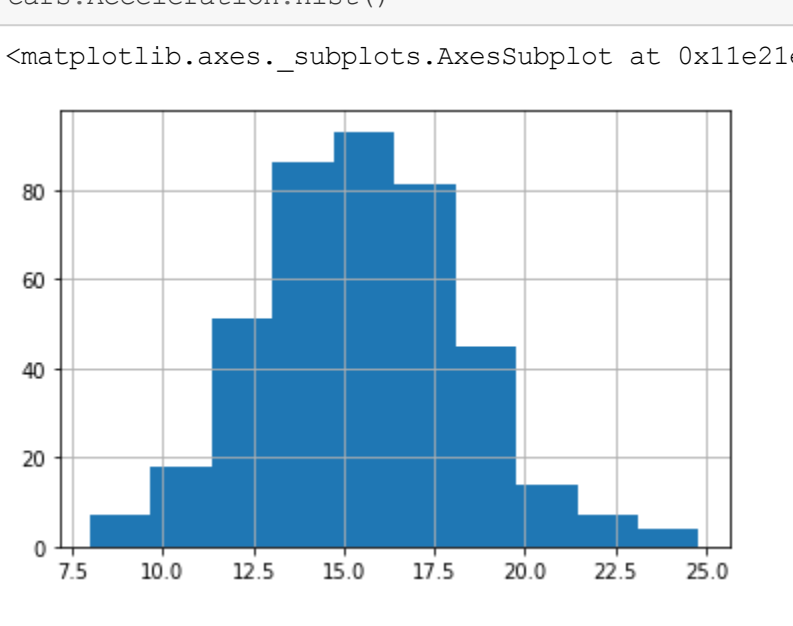
When you have lots of data points, you can't no longer use the scatter plots. Even when you don't have millions of data points, you often want to get a quick summary of the distribution rather than seeing the whole dataset. For 1-D datasets, two major approaches are histogram and boxplot. Histogram is about aggregating and counting the data while boxplot is about summarizing the data. Let's first draw some histograms.

Histogram

It's very easy to draw a histogram with pandas.

```
[17]: cars.Acceleration.hist()
```

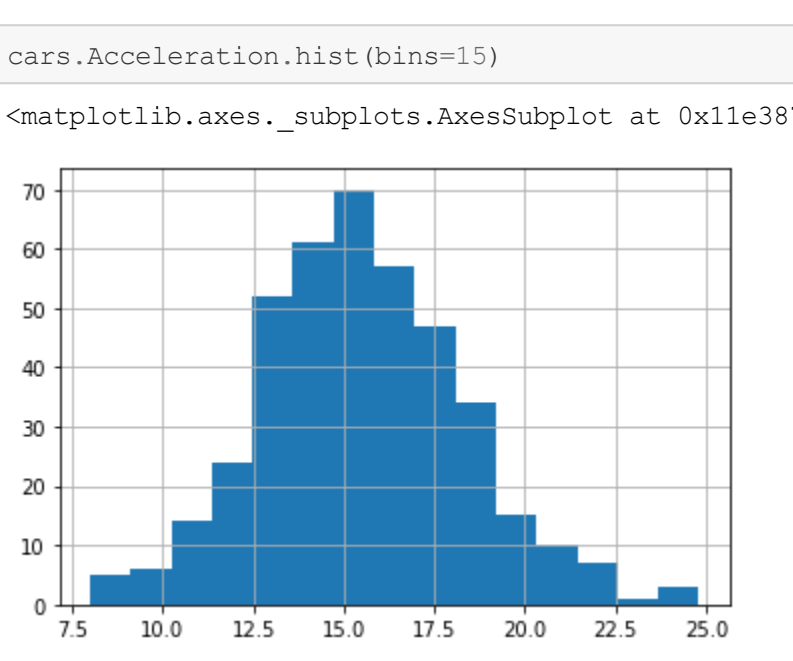
```
[17]: <matplotlib.axes._subplots.AxesSubplot at 0x11e21ebd0>
```



You can adjust the bin size, which is the main parameter of the histogram.

```
[18]: cars.Acceleration.hist(bins=15)
```

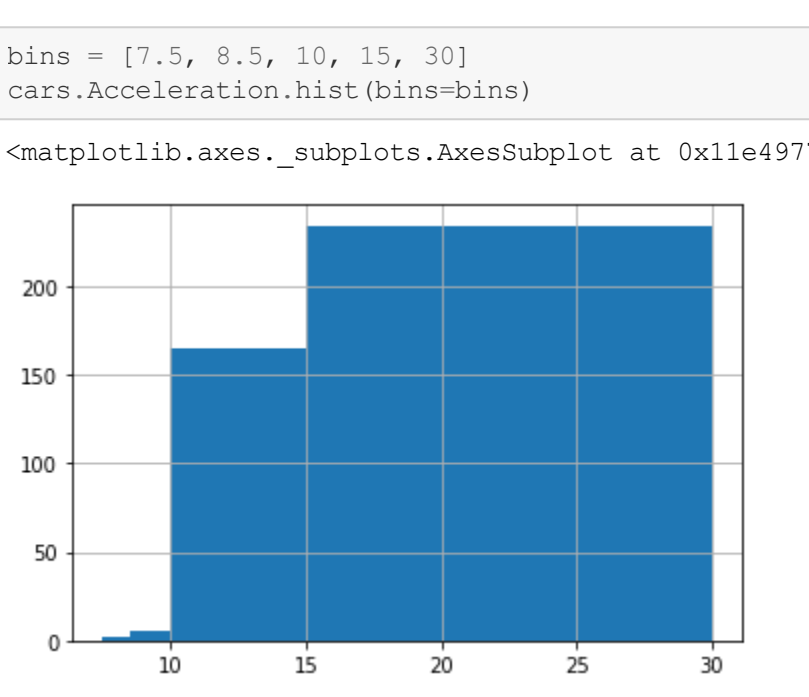
```
[18]: <matplotlib.axes._subplots.AxesSubplot at 0x11e387990>
```



You can even specify the actual bins.

```
[19]: bins = [7.5, 8.5, 10, 15, 30]
cars.Acceleration.hist(bins=bins)
```

```
[19]: <matplotlib.axes._subplots.AxesSubplot at 0x11e497710>
```



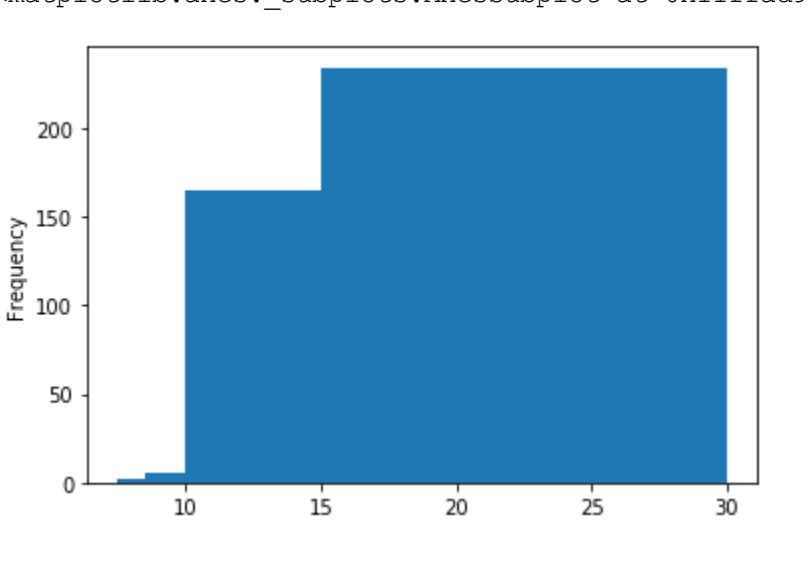
Do you see anything funky going on with this histogram? What's wrong? Can you fix it?

Q: Explain what's wrong with this histogram and fix it.

(hints: do you remember what we discussed regarding histogram? Also [pandas documentation](#) does not show the option that you should use. You should take a look at the `matplotlib`'s documentation.

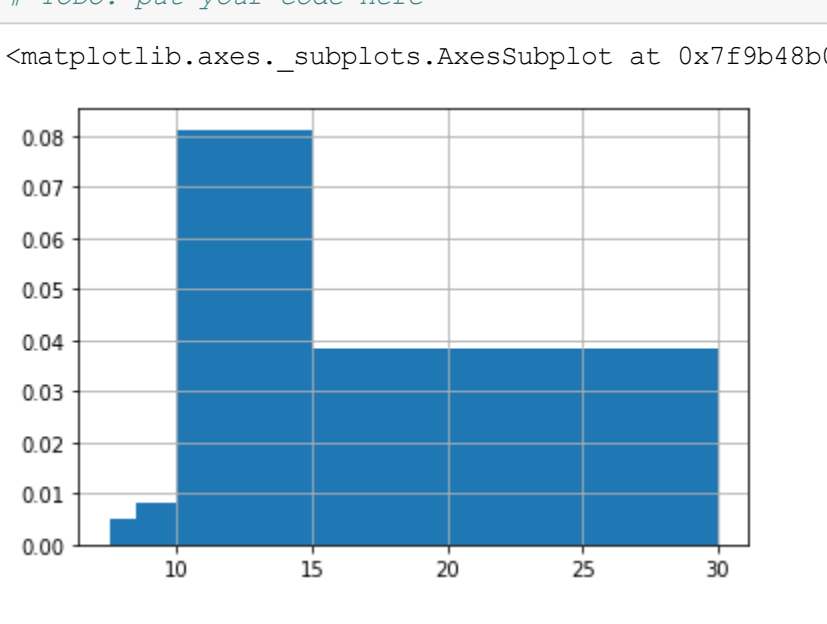
```
[52]: cars["Acceleration"].plot.hist(bins=bins)
```

```
[52]: <matplotlib.axes._subplots.AxesSubplot at 0x11ffad9d0>
```



```
[13]: # TODO: put your code here
```

```
[13]: <matplotlib.axes._subplots.AxesSubplot at 0x7f9b48b0a860>
```



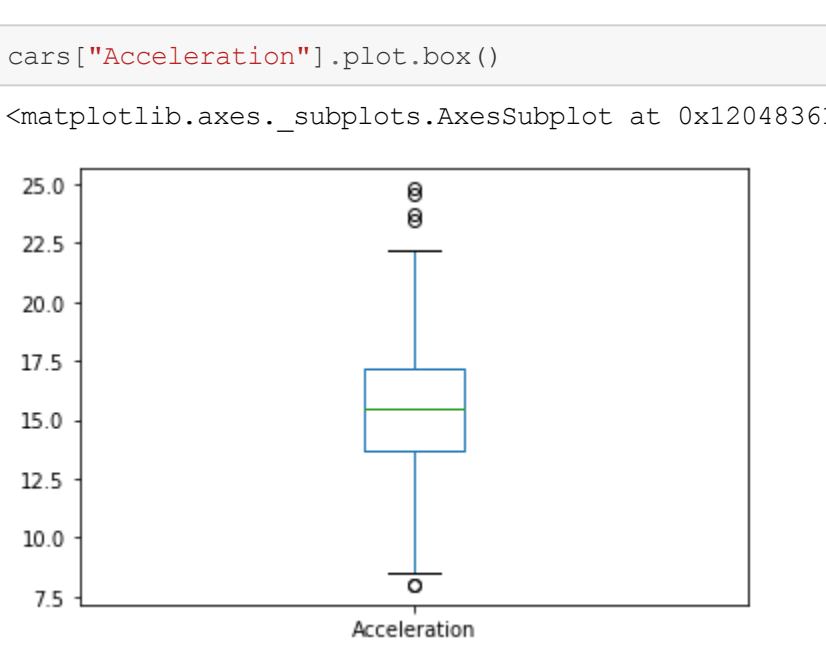
Boxplot

Boxplot can be created with pandas very easily. Check out the `plot` documentation: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.plot.html

Q: create a box plot of `Acceleration`

```
[45]: cars["Acceleration"].plot.box()
```

```
[45]: <matplotlib.axes._subplots.AxesSubplot at 0x120483610>
```



1D scatter plot with Seaborn and Altair

As you may have noticed, it is not very easy to use `matplotlib`. The organization of plot functions and parameters are not very systematic. Whenever you draw something, you should search how to do it, what are the parameters you can tweak, etc. You need to manually tweak a lot of things when you work with `matplotlib`.

There are more systematic approaches towards data visualization, such as the "Grammar of Graphics". This idea of *grammar* led to the famous `ggplot2` (http://ggplot2.tidyverse.org) package in R as well as the Vega & Vega-lite) for the web. The grammar-based approach lets you work with *tidy data* in a natural way, and also lets you approach the data visualization systematically. In other words, they are very cool. ☐

I'd like to introduce two nice Python libraries. One is called `seaborn` (https://seaborn.pydata.org), which is focused on creating complex statistical data visualizations, and the other is called `altair` (https://altair-viz.github.io/) and it is a Python library that lets you *define* a visualization and translates it into vega-lite json.

Seaborn would be useful when you are doing exploratory data analysis; altair may be useful if you are thinking about creating and putting an interactive visualization on the web.

If you don't have them yet, check the installation page of altair. In `conda`,

```
$ conda install -c conda-forge altair vega_datasets jupyterlab
```

Let's play with it.

```
[53]: import seaborn as sns
import altair as alt

# Uncomment the following line if you are using Jupyter notebook
alt.renderers.enable('notebook')
```

```
[53]: RendererRegistry.enable('notebook')
```

```
[54]: cars.head()
```

[54]:

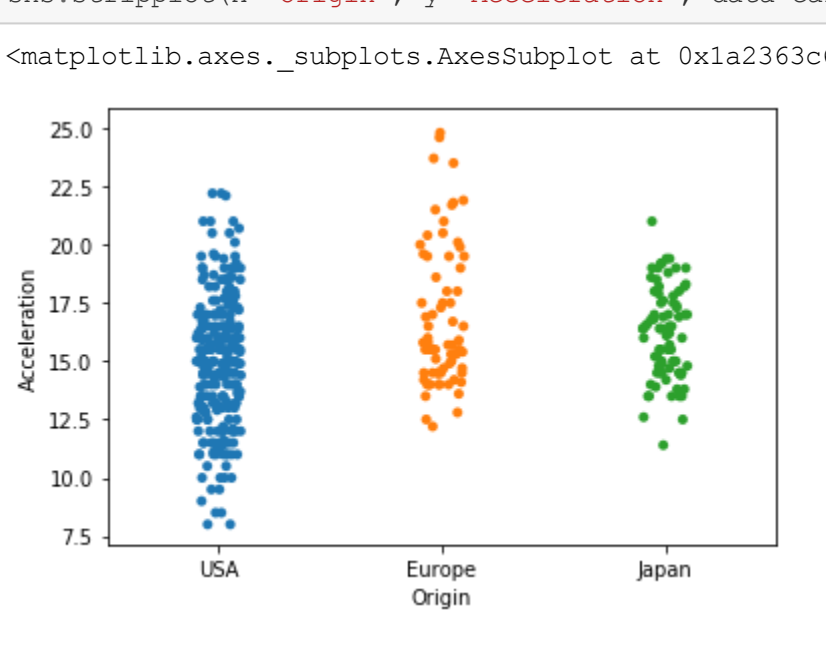
	Name	Miles_per_Gallon	Cylinders	Displacement	Horsepower	Weight_in_lbs	Acceleration	Year	Origin
0	chevrolet chevelle malibu	18.0	8	307.0	130.0	3504	12.0	1970-01-01	USA
1	buick skylark 320	15.0	8	350.0	165.0	3693	11.5	1970-01-01	USA
2	plymouth satellite	18.0	8	318.0	150.0	3436	11.0	1970-01-01	USA
3	amc rebel sst	16.0	8	304.0	150.0	3433	12.0	1970-01-01	USA
4	ford torino	17.0	8	302.0	140.0	3449	10.5	1970-01-01	USA

Beeswarm plots with seaborn

Seaborn has a built-in function to create 1D scatter plots with multiple categories.

```
[55]: sns.stripplot(x='Origin', y='Acceleration', data=cars)
```

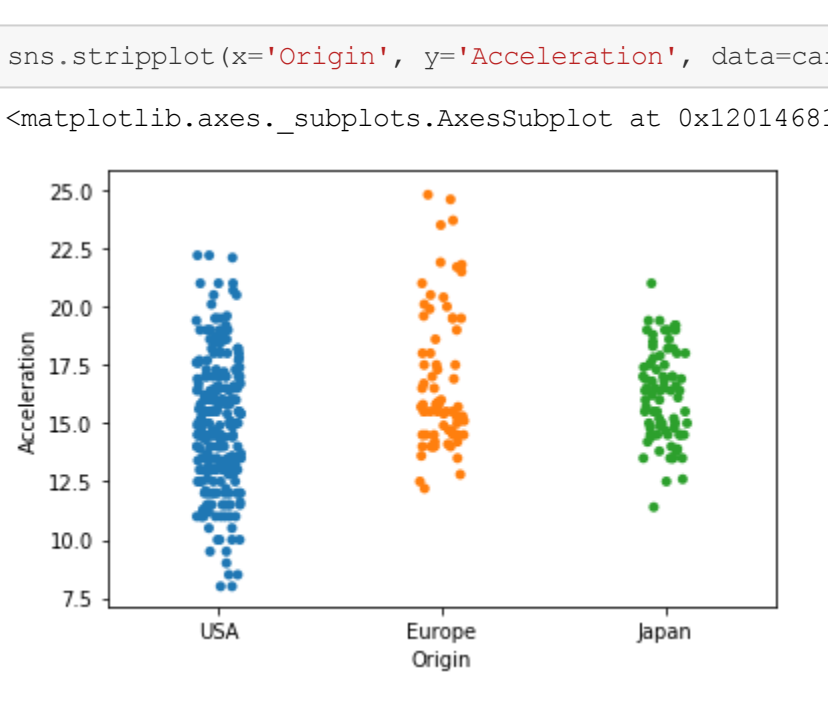
```
[55]: <matplotlib.axes._subplots.AxesSubplot at 0x1a2363c650>
```



And you can easily add jitters or even create a beeswarm plot.

```
[56]: sns.stripplot(x='Origin', y='Acceleration', data=cars, jitter=True)
```

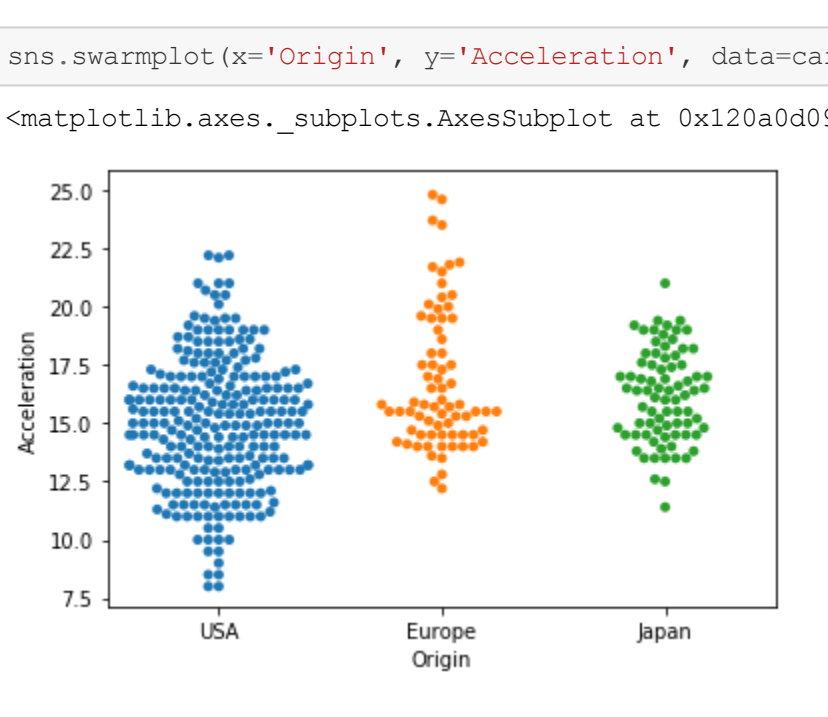
```
[56]: <matplotlib.axes._subplots.AxesSubplot at 0x120146810>
```



Seems like European cars tend to have good acceleration. ☐ Let's look at the beeswarm plot, which is a pretty nice option for fairly small datasets.

```
[57]: sns.swarmplot(x='Origin', y='Acceleration', data=cars)
```

```
[57]: <matplotlib.axes._subplots.AxesSubplot at 0x120a0d090>
```

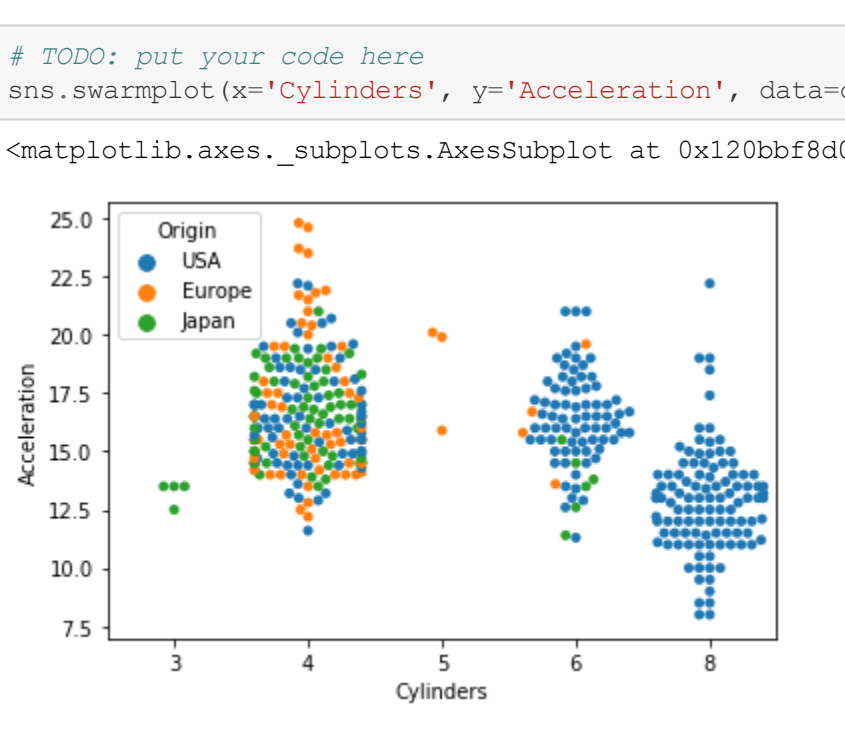


Seaborn also allows you to use colors for another categorical variable. The option is `hue`.

Q: can you create a beeswarm plot where the swarms are grouped by `Cylinders`, y-values are `Acceleration`, and colors represent the `Origin`?

```
[59]: # TODO: put your code here
sns.swarmplot(x='Cylinders', y='Acceleration', data=cars, hue="Origin")
```

```
[59]: <matplotlib.axes._subplots.AxesSubplot at 0x120bbf8d0>
```

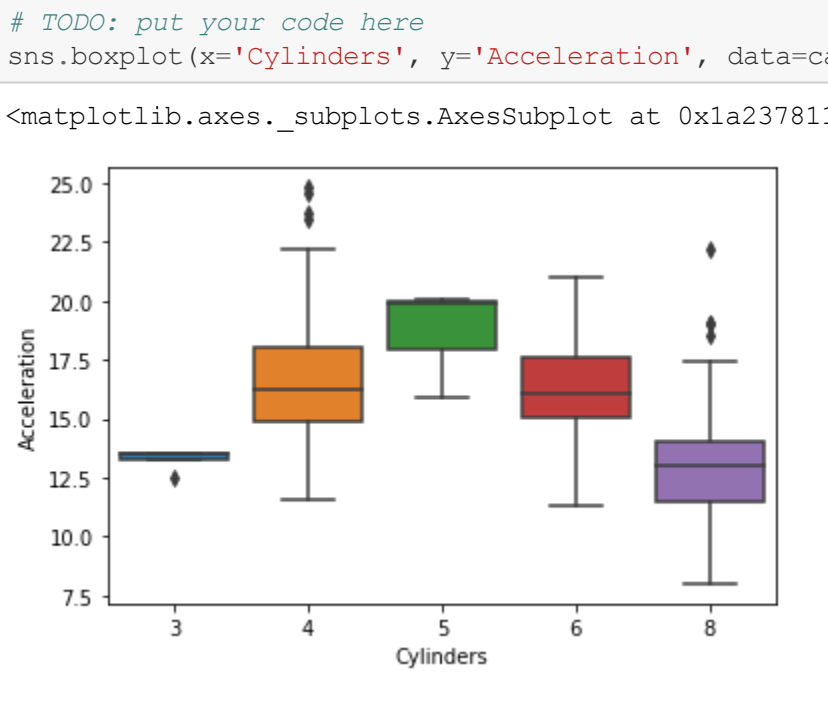


And of course you can create box plots too.

Q: Create boxplots to show the relationships between `Cylinders` and `Acceleration`.

```
[60]: # TODO: put your code here
sns.boxplot(x='Cylinders', y='Acceleration', data=cars)
```

```
[60]: <matplotlib.axes._subplots.AxesSubplot at 0x1a237811d0>
```

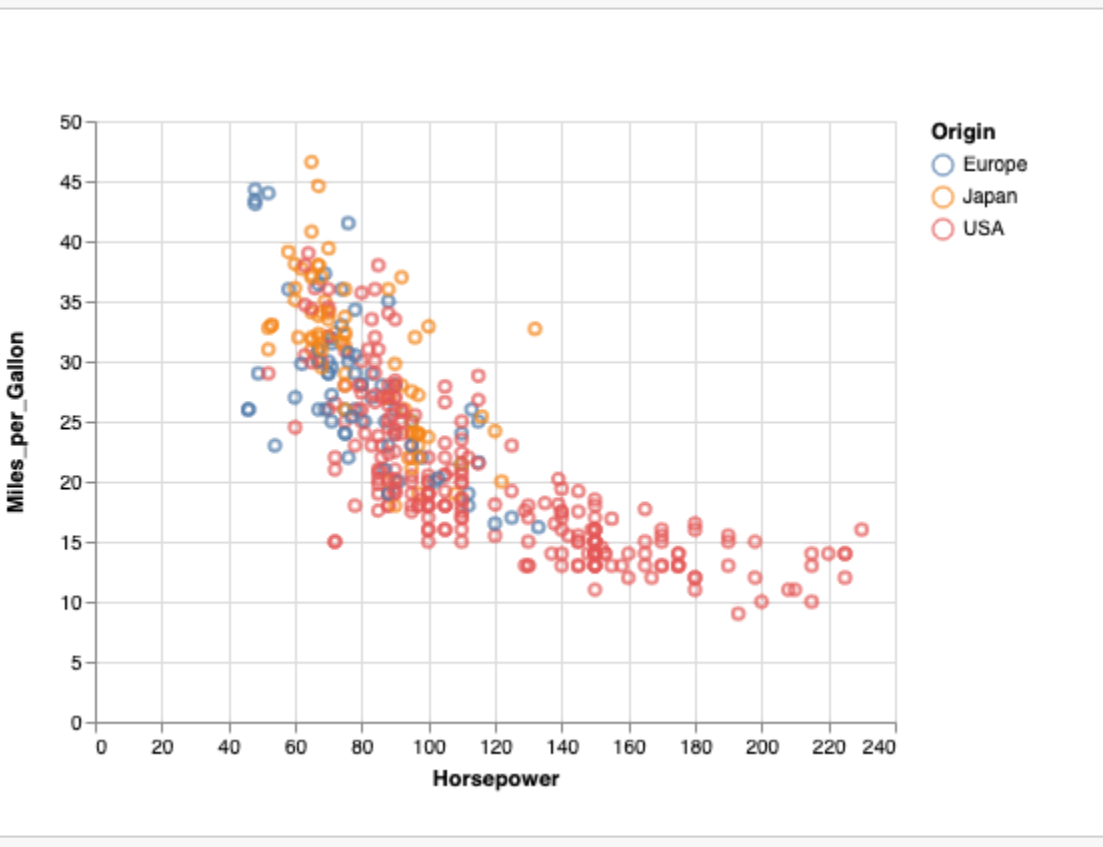


Altair basics

With `altair`, you're thinking in terms of a whole dataframe, rather than vectors for x or vectors for y. Passing the dataset to `Chart` creates an empty plot. If you try to run `alt.Chart(cars)`, it will complain. You need to say what's the visual encoding of the data.

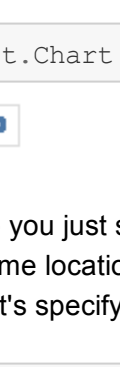
```
[63]: alt.Chart(cars).mark_point().encode(
    x='Horsepower',
    y='Miles_per_Gallon',
    color='Origin',
)
```

[63]:



```
[25]: alt.Chart(cars).mark_point()
```

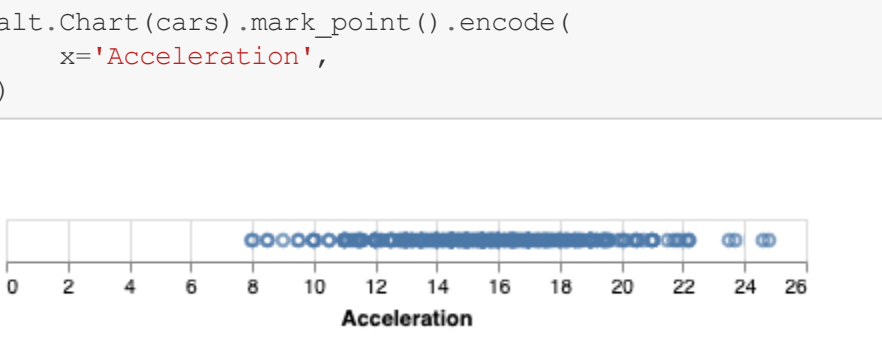
[25]:



So you just see one *point*. But actually this is not a single point. This is every row of the dataset represented as a point at the same location. Because there is no specification about where to put the points, it simply draws everything on top of each other. Let's specify how to spread them across the horizontal axis.

```
[64]: alt.Chart(cars).mark_point().encode(
    x='Acceleration',
)
```

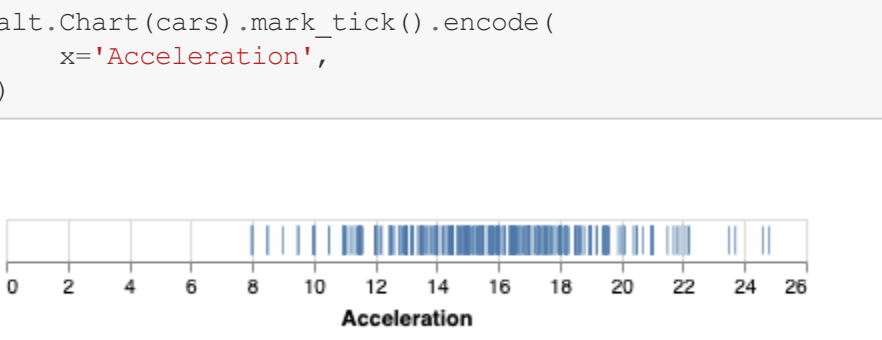
[64]:



There is another nice mark called `tick`:

```
[65]: alt.Chart(cars).mark_tick().encode(
    x='Acceleration',
)
```

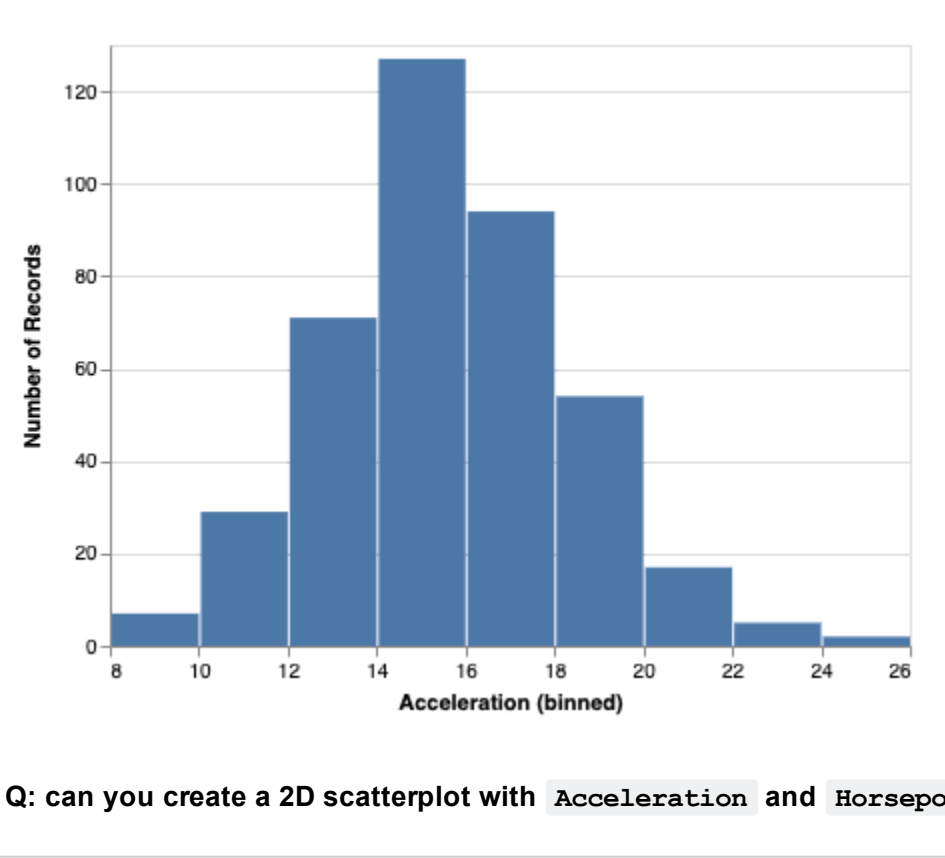
[65]:



In `altair`, histogram is not a special type of visualization, but simply a plot with bars where a variable is binned and a counting aggregation function is used.

```
[67]: alt.Chart(cars).mark_bar().encode(
    x=alt.X('Acceleration', bin=True),
    y='count()'
)
```

[67]:



Q: can you create a 2D scatterplot with `Acceleration` and `Horsepower`? Use `Origin` for the colors.

```
[68]: # TODO: put your code here
alt.Chart(cars).mark_point().encode(
    x='Horsepower',
    y='Miles_per_Gallon',
    color='Origin',
)
```

[68]: