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.

In [40]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

from vega_datasets import data
```

In [41]:
```
cars = data.cars()
cars.head()
```

Out[41]:

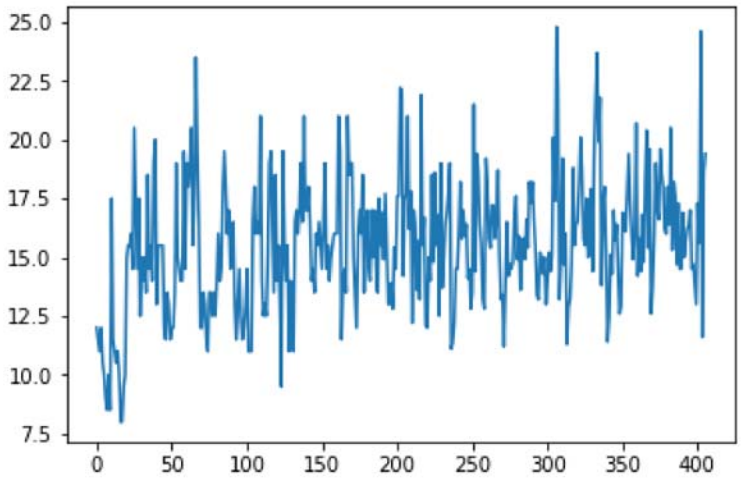
	Acceleration	Cylinders	Displacement	Horsepower	Miles_per_Gallon	Name	Origin	Weight_i
0	12.0	8	307.0	130.0	18.0	chevrolet chevelle malibu	USA	3504
1	11.5	8	350.0	165.0	15.0	buick skylark 320	USA	3693
2	11.0	8	318.0	150.0	18.0	plymouth satellite	USA	3436
3	12.0	8	304.0	150.0	16.0	amc rebel sst	USA	3433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

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.

In [42]:
```
cars.Acceleration.plot()
```

Out[42]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x14fb8498080>
```



It's actually not easy to use pandas to create an 1-D scatter plot. We can use `matploblib`'s `scatter` function though.

We can first create an array with zeros. `np.zeros_like` returns an array with zeros that matches the shape of the input array.

In [43]:
```
np.zeros_like([1,2,3])
```

Out[43]:
```
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.

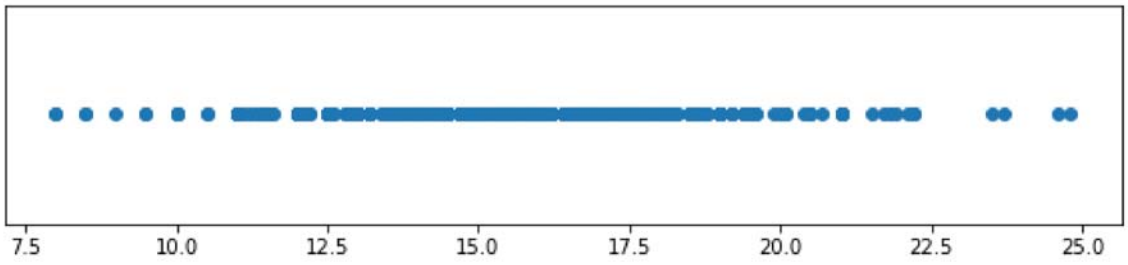
In [44]:
```
#Create x axis
x=cars.Acceleration
```

In [45]:
```
#Create y axis with zeros
y=np.zeros_like(x)
```

In [46]:
```
plt.figure(1, figsize=(10, 2))

#set y ticks to empty
plt.scatter(x, y).axes.get_yaxis().set_ticks([])
```

Out[46]:
```
[]
```

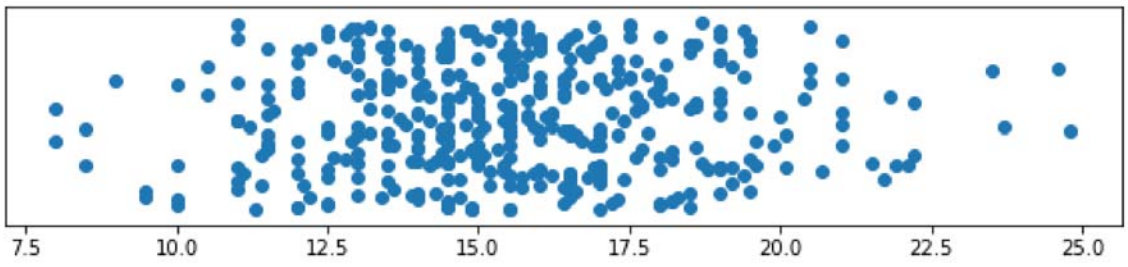


As you can see, there are lots of occlusions. So this plot can be misleading. Let's add some jitters. You can use `numpy`'s `random.rand()` function to generate random numbers.

Q: create a jittered 1D scatter plot.

In [47]:
```
# TODO: put your code here
jittered_y = np.random.rand(len(x))
plt.figure(1, figsize=(10, 2))
plt.scatter(x, jittered_y).axes.get_yaxis().set_ticks([])
```

Out[47]: []

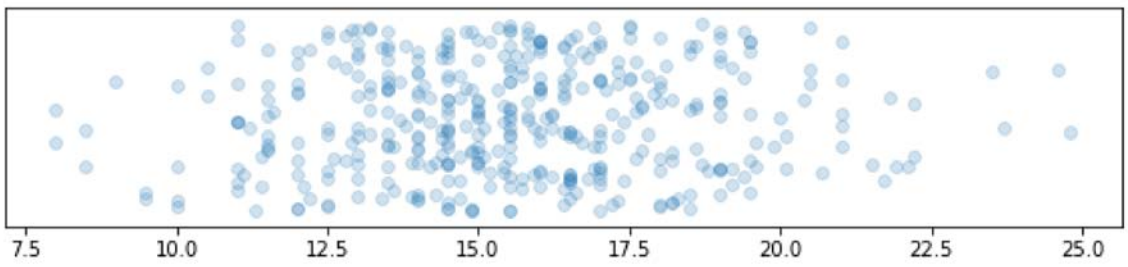


We can further improve this by adding transparency to the symbols. The transparency option for `scatter` function is called `alpha`. Set it to be 0.2.

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

In [48]:
```
# TODO: put your code here
plt.figure(1, figsize=(10, 2))
plt.scatter(x, jittered_y, alpha=0.2).axes.get_yaxis().set_ticks([])
```

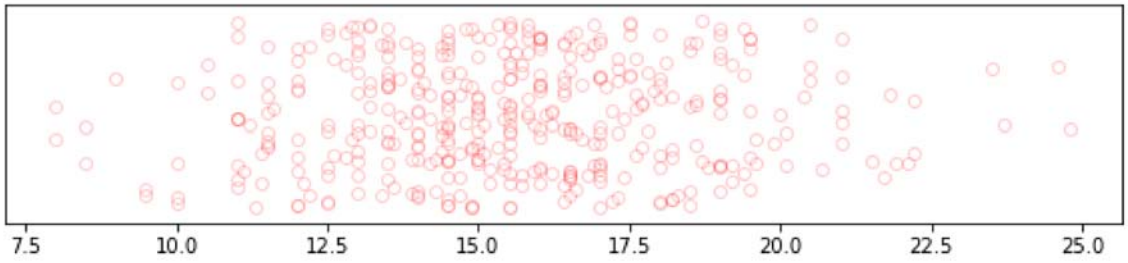
Out[48]: []



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.**

In [50]:
```
plt.figure(1, figsize=(10, 2))
fig1= plt.scatter(x, jittered_y, alpha=0.2, color='w',edgecolors='r')
fig1.axes.get_yaxis().set_ticks([])
#fig1.axes.set_facecolor('#ffffff')
```

Out[50]: []



What happens with lots and lots of points?

Whatever strategy that you use, it's almost useless if you have too many data points.

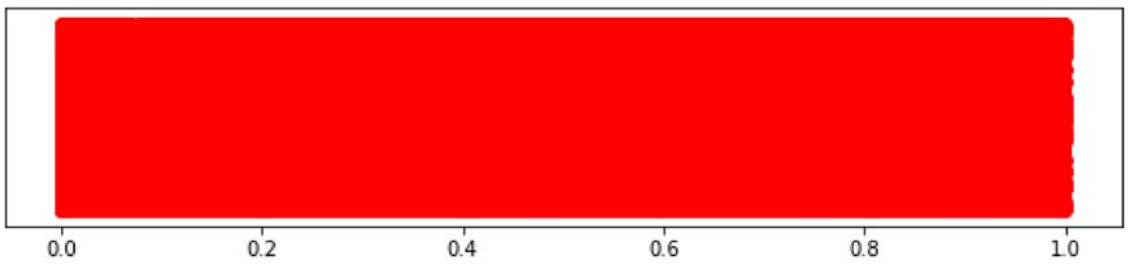
Q: play with different number of data points and see how it looks.

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

In [51]:
```
# TODO: play with N and see what happens.
N = 100000
x = np.random.rand(N)
jittered_y = np.random.rand(N)


# TODO: 1D scatter plot code here
plt.figure(1, figsize=(10, 2))
plt.scatter(x, jittered_y, color='r').axes.get_yaxis().set_ticks([])
```

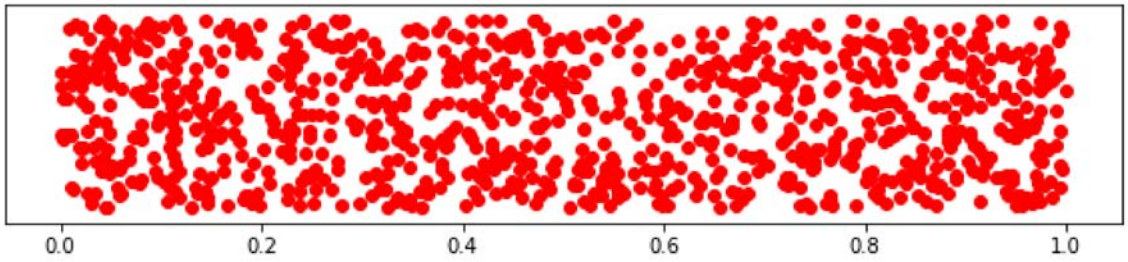
Out[51]: []



In [52]:
```
N = 1000
x = np.random.rand(N)
jittered_y = np.random.rand(N)

plt.figure(1, figsize=(10, 2))
plt.scatter(x, jittered_y, color='r').axes.get_yaxis().set_ticks([])
```

Out[52]: []



Histogram and boxplot

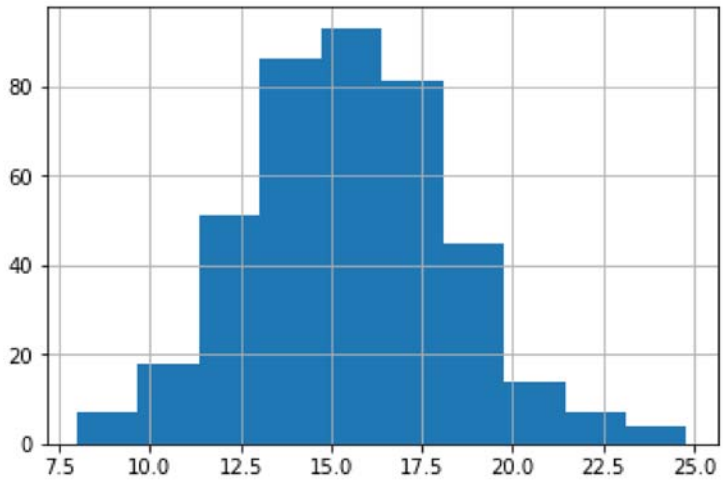
When you have lots of data points, you can't no longer use the scatter plots. Even when you don't have millions of data points, you often want to get a quick summary of the distribution rather than seeing the whole dataset. For 1-D datasets, two major approaches are histogram and boxplot. Histogram is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

In [53]:
```
cars.Acceleration.hist()
```

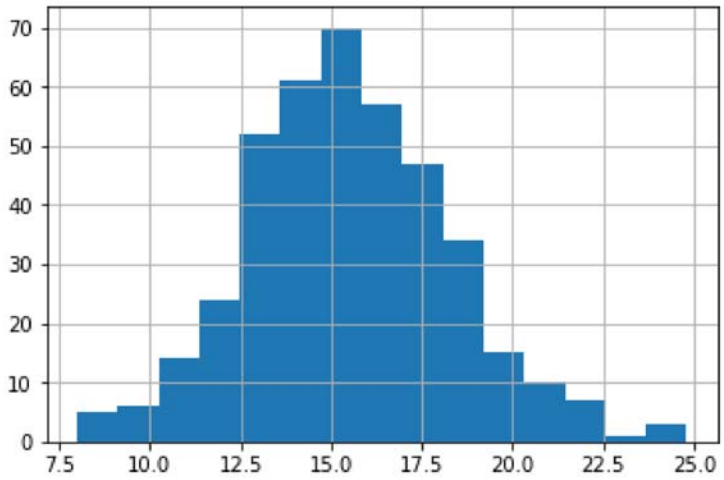
Out[53]: <matplotlib.axes._subplots.AxesSubplot at 0x14fb85d6e48>



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

```
In [54]: cars.Acceleration.hist(bins=15)
```

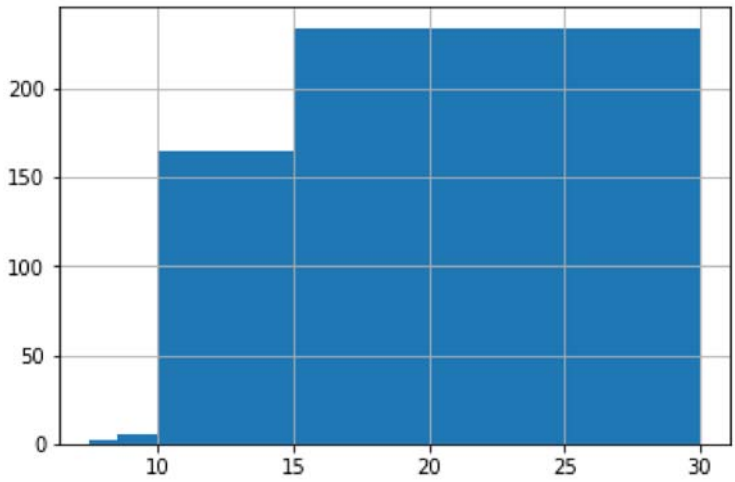
```
Out[54]: <matplotlib.axes._subplots.AxesSubplot at 0x14fb988fd30>
```



You can even specify the actual bins.

```
In [55]: bins = [7.5, 8.5, 10, 15, 30]
         cars.Acceleration.hist(bins=bins)
```

```
Out[55]: <matplotlib.axes._subplots.AxesSubplot at 0x14fb99176d8>
```



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.

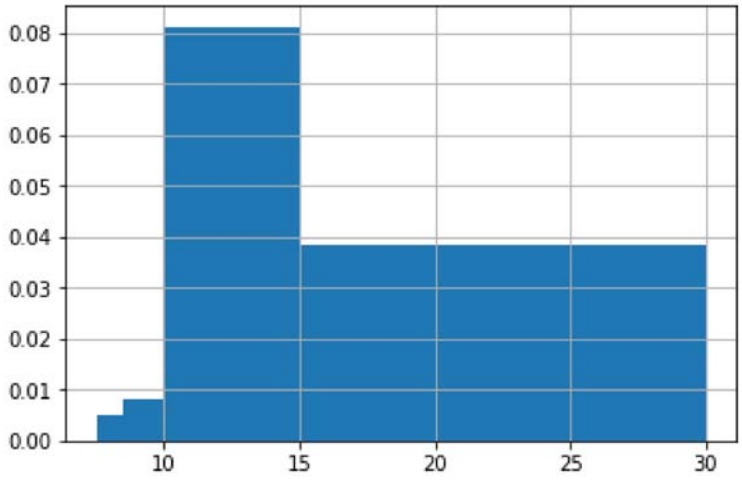
(a hint: pandas documentation (http://pandas.pydata.org/pandas-docs/version/0.22/generated/pandas.DataFrame.hist.html) does not show the option that you should use. You should take a look at the `matplotlib`'s documentation.

The data.cars is not normalized. We will use density = True to form a probability density, i.e., the area (or integral) under the histogram will sum to 1. This is achieved by dividing the count by the number of observations times the bin width and not dividing by the total number of observations.

In [ ]:
```
dataA=cars.Acceleration

dataA.hist(bins=bins, normed=True)
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x14fb9979b70>
```



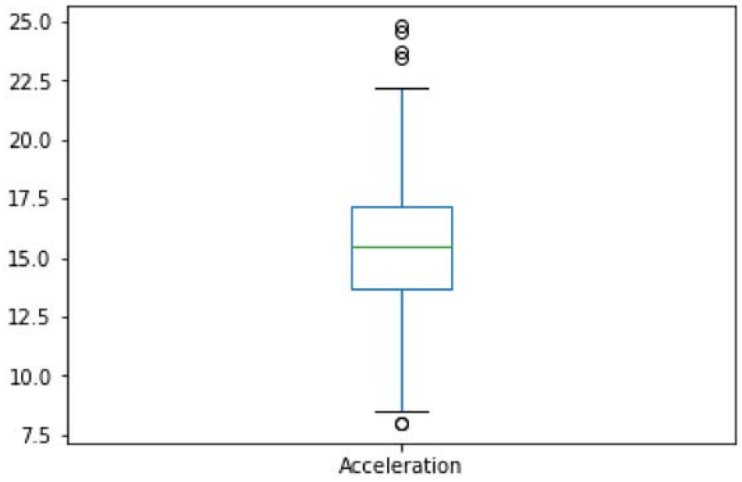
Boxplot

Boxplot can be created with pandas very easily. Check out the `plot` documentation.

Q: create a box plot of `Acceleration`

In [57]:
```
#plt.boxplot(dataA)
#plt.xticks([1], ['Acceleration'])

#Using pandas
df = pd.DataFrame(dataA, columns=['Acceleration'])
boxplot = df.boxplot(grid=False, column=['Acceleration'])
```



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 (https://www.amazon.com/Grammar-Graphics-Statistics-Computing/dp/0387245448)". This idea of *grammar* led to the famous `ggplot2` (http://ggplot2.tidyverse.org (http://ggplot2.tidyverse.org)) package in R as well as the Vega & Vega-lite ((https://vega.github.io)) 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 (https://seaborn.pydata.org)), which is focused on creating complex statistical data visualizations, and the other is called `altair` (https://altair-viz.github.io/ (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, run

```
pip install seaborn altair
```

or

```
pip3 install seaborn altair
```

or

```
conda install seaborn altair
```

depending on your environment.

Let's play with it.

In [58]:
```
import seaborn as sns
import altair as alt
```

In [59]:
```
cars.head()
```

Out[59]:

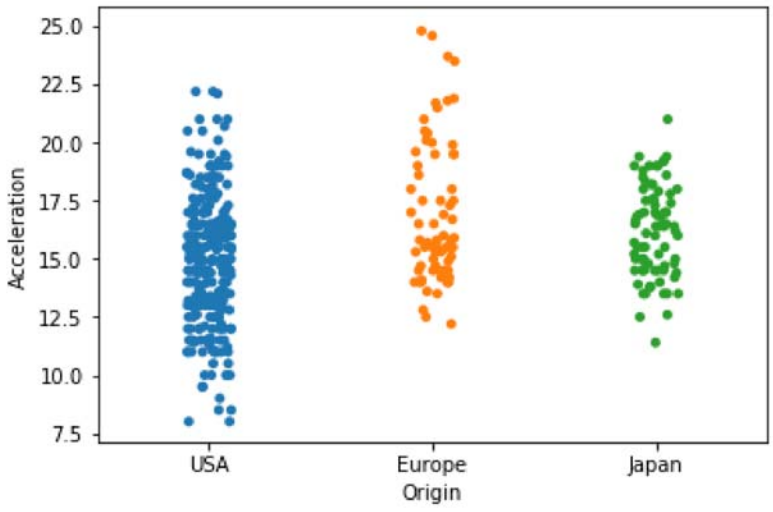
	Acceleration	Cylinders	Displacement	Horsepower	Miles_per_Gallon	Name	Origin	Weight_i
0	12.0	8	307.0	130.0	18.0	chevrolet chevelle malibu	USA	3504
1	11.5	8	350.0	165.0	15.0	buick skylark 320	USA	3693
2	11.0	8	318.0	150.0	18.0	plymouth satellite	USA	3436
3	12.0	8	304.0	150.0	16.0	amc rebel sst	USA	3433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

Beeswarm plots with seaborn

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

In [60]:
```
sns.stripplot(x='Origin', y='Acceleration', data=cars)
```

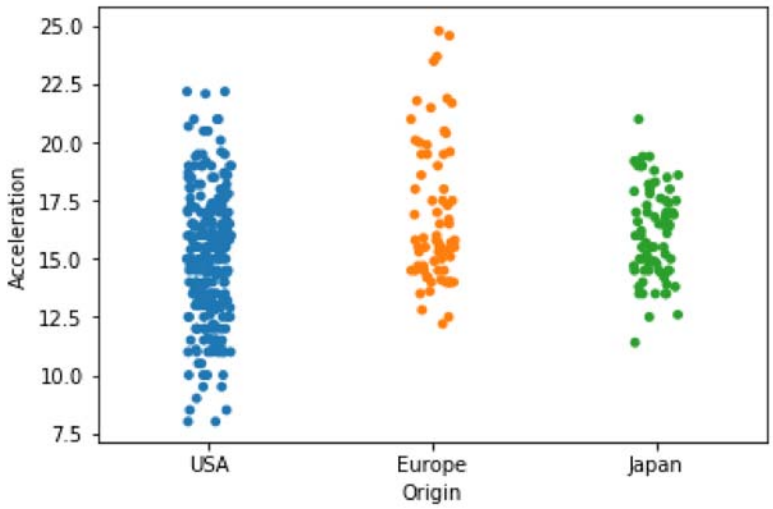
Out[60]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x14fb85d0860>
```



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

```
In [61]: sns.stripplot(x='Origin', y='Acceleration', data=cars, jitter=True)
```

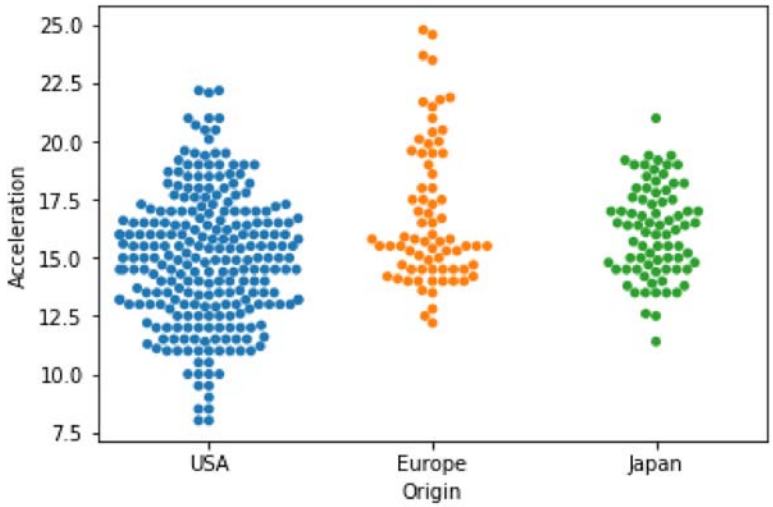
```
Out[61]: <matplotlib.axes._subplots.AxesSubplot at 0x14fb854a978>
```



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.

```
In [62]: sns.swarmplot(x='Origin', y='Acceleration', data=cars)
```

```
Out[62]: <matplotlib.axes._subplots.AxesSubplot at 0x14fb847c0f0>
```

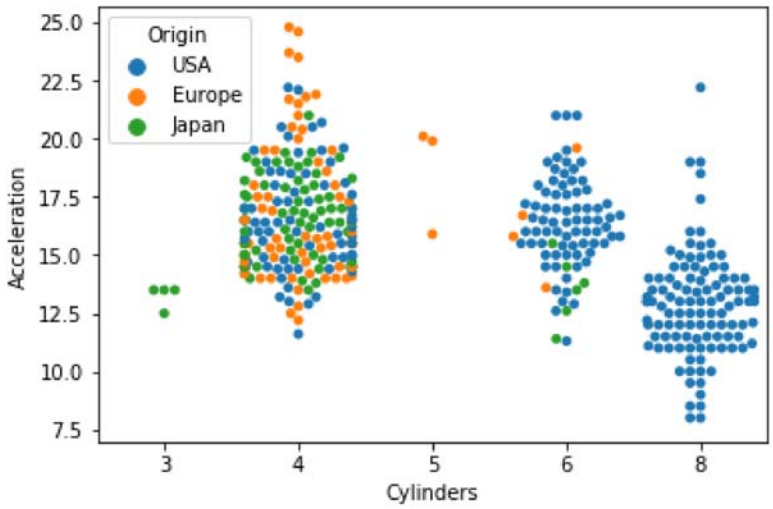


Seaborn also allows you to use colors for another categorical variable. The option is `hue`.

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

```
In [63]: sns.swarmplot(x='Cylinders', y='Acceleration', hue='Origin',data=cars)
```

```
Out[63]: <matplotlib.axes._subplots.AxesSubplot at 0x14fb83ee198>
```

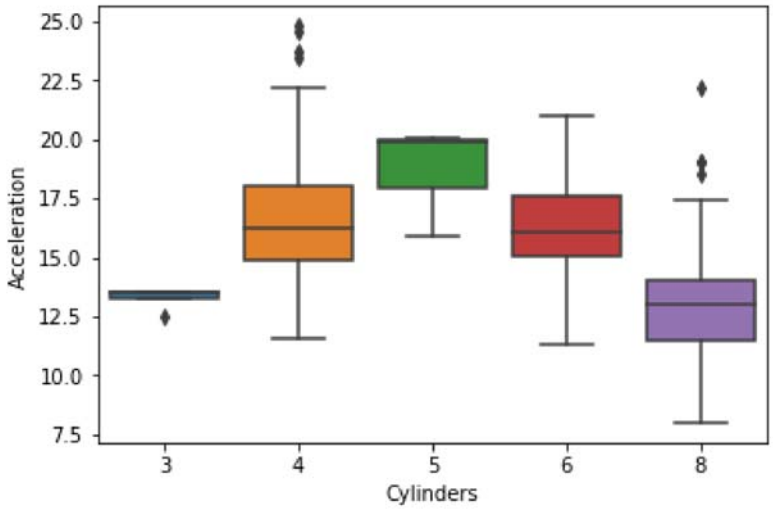


And of course you can create box plots too.

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

```
In [64]: sns.boxplot(x='Cylinders', y='Acceleration',data=cars)
```

```
Out[64]: <matplotlib.axes._subplots.AxesSubplot at 0x14fb83b04e0>
```



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. You then need to say what's the visual encoding of the data. If you try to run `alt.Chart(cars)`, it will complain.

In [65]:
```
import altair as alt
import IPython; IPython.__version__
alt.renderers.enable('notebook')
```

Out[65]:
```
RendererRegistry.enable('notebook')
```

In [66]:
```
alt.Chart(cars).mark_point()
```

```
<vega.vegalite.VegaLite at 0x14fb84d7518>
```

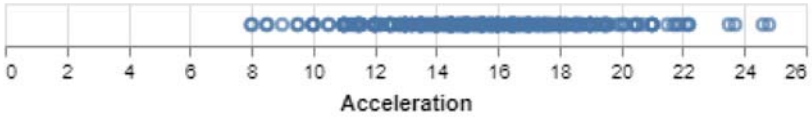
Out[66]:

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.

In [67]:
```
alt.Chart(cars).mark_point().encode(
    x='Acceleration',
)
```

```
<vega.vegalite.VegaLite at 0x14fb83cd6a0>
```

Out[67]:

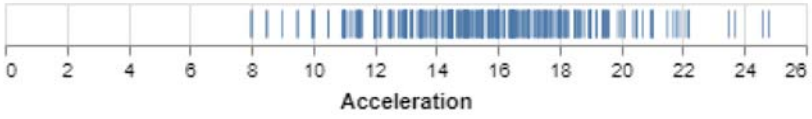


There is another nice mark called `tick`:

In [68]:
```
alt.Chart(cars).mark_tick().encode(
    x='Acceleration',
)
```

```
<vega.vegalite.VegaLite at 0x14fb84154e0>
```

Out[68]:

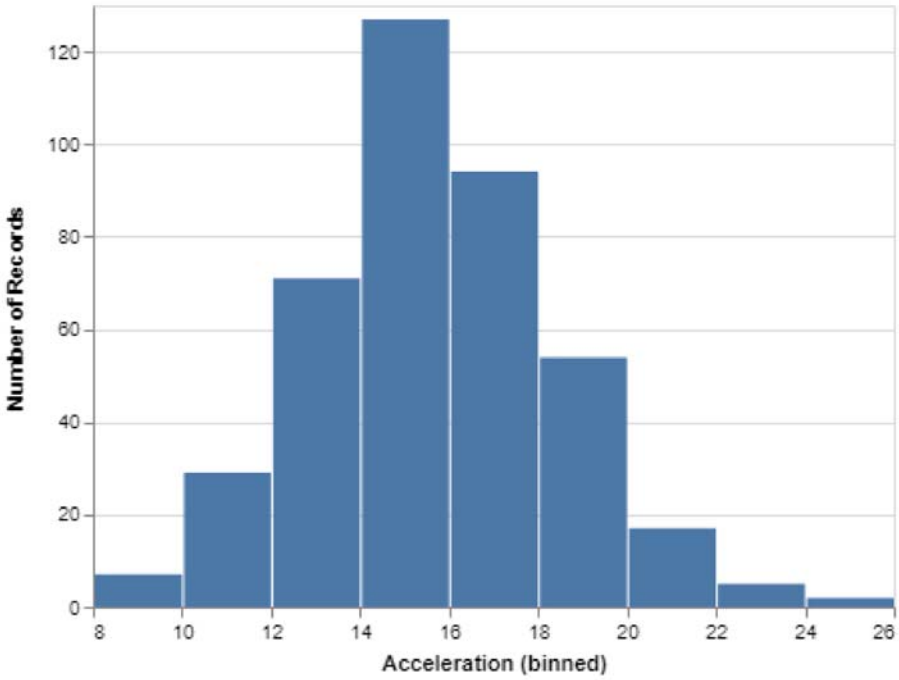


In `altair`, histogram is not a special type of visualization, but simply a plot with bars where a variable is binned and a counting aggregation function is used.

In [69]:
```
alt.Chart(cars).mark_bar().encode(
    x=alt.X('Acceleration', bin=True),
    y='count()'
)
```

```
<vega.vegalite.VegaLite at 0x14fb85a53c8>
```

Out[69]:

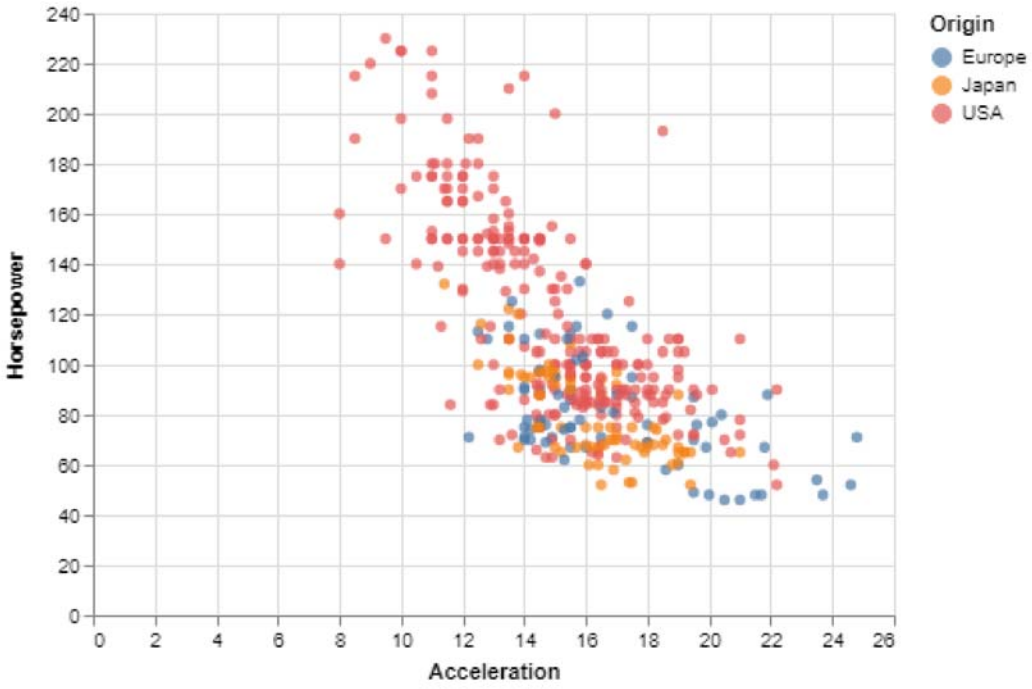


Q: can you create a 2D scatterplot with `Acceleration` and `Horsepower`? Use `Origin` for the colors. Save it to m07.html (https://altair-viz.github.io/getting_started/starting.html#publishing-your-visualization (https://altair-viz.github.io/getting_started/starting.html#publishing-your-visualization))

In [70]:
```
alt.Chart(cars).mark_circle().encode(
    x='Acceleration',
    y='Horsepower',
    color='Origin'
).interactive()
```

```
<vega.vegalite.VegaLite at 0x14fb980beb8>
```

Out[70]:



In [71]:
```
chart=alt.Chart(cars).mark_circle().encode(
    x='Acceleration',
    y='Horsepower',
    color='Origin'
).interactive()
```

In [72]:
```
chart.save('m07_yezerets_helen.html')
```