Module 8: Histogram and CDF

A deep dive into Histogram + CDF.

In [173]:
```
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import altair as alt
import pandas as pd
%matplotlib inline
```

`matplotlib` 2.1 has a bug in the histogram. Make sure to have version 2.2.

In [174]:
```
import matplotlib
matplotlib.__version__
```

Out[174]:
```
'2.2.3'
```

The tricky histogram with pre-counted data

Let's revisit the table from the class

Hours	Frequency
0-1	4,300
1-3	6,900
3-5	4,900
5-10	2,000
10-24	2,100

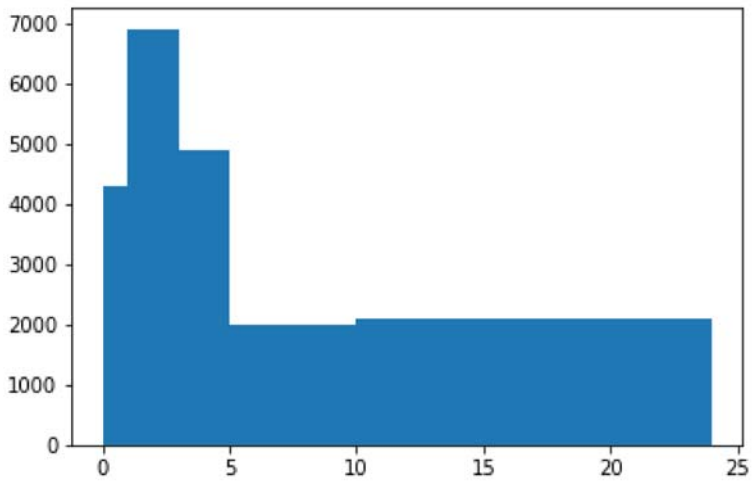
You can draw a histogram by just providing bins and counts instead of a list of numbers. So, let's try that.

In [175]:
```
bins = [0, 1, 3, 5, 10, 24]
data = {0.5: 4300, 2: 6900, 4: 4900, 7: 2000, 15: 2100}
```

Q: Draw histogram using this data. Don't normalize it for now. Useful query: Google search: matplotlib histogram pre-counted (https://www.google.com/search?client=safari&rls=en&q=matplotlib+histogram+already+counted&ie=UTF-8&oe=UTF-8#q=matplotlib+histogram+pre-counted)

In [176]:
```
# TODO: draw a histogram with weighted data.
# Hint: list(an_array) can make the array to list, which is the required input format of
 hist function
val, weight = zip(*[(k, v) for k,v in data.items()])
plt.hist(val, weights=weight, bins=bins)
```

Out[176]:
```
(array([4300., 6900., 4900., 2000., 2100.]),
 array([ 0,  1,  3,  5, 10, 24]),
 <a list of 5 Patch objects>)
```

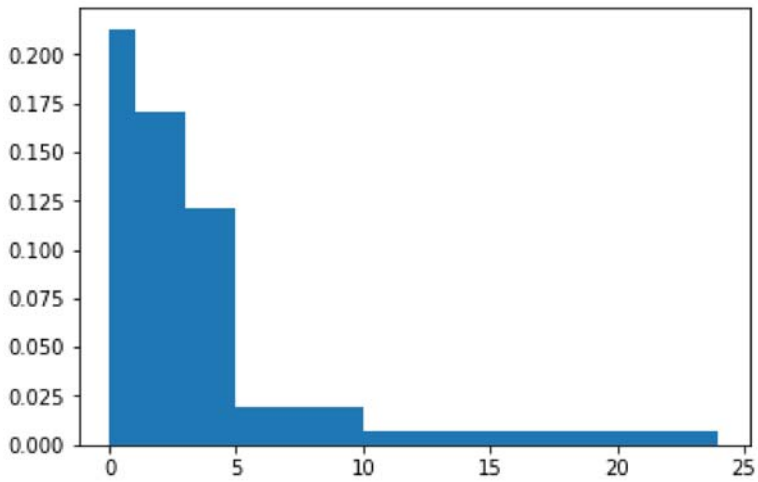


As you can see, the **default histogram does not normalize with binwidth and simply shows the counts**! This can be very misleading if you are working with variable bin width (e.g. logarithmic bins). So please be mindful about histograms when you work with variable bins.

Q: You can fix this by using the `density` option.

In [177]:
```
plt.hist(val, weights=weight, bins=bins, density=True)
```

Out[177]:
```
(array([0.21287129, 0.17079208, 0.12128713, 0.01980198, 0.00742574]),
 array([ 0,  1,  3,  5, 10, 24]),
 <a list of 5 Patch objects>)
```



Let's use an actual dataset

In [178]:
```
import vega_datasets
```

In [179]:
```
movies = vega_datasets.data.movies()
movies.head()
```

Out[179]:

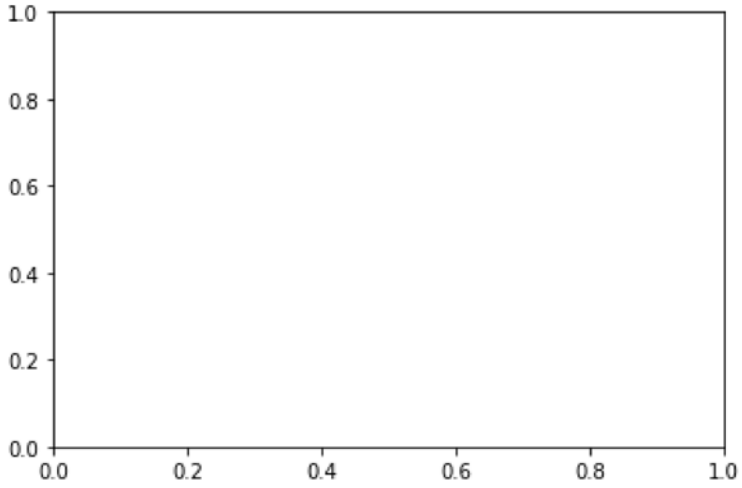
	Creative_Type	Director	Distributor	IMDB_Rating	IMDB_Votes	MPAA_Rating	Major_Genre	Prod
0	None	None	Gramercy	6.1	1071.0	R	None	8000
1	None	None	Strand	6.9	207.0	R	Drama	3000
2	None	None	Lionsgate	6.8	865.0	None	Comedy	2500
3	None	None	Fine Line	NaN	NaN	None	Comedy	3000
4	Contemporary Fiction	None	Trimark	3.4	165.0	R	Drama	1000

Let's plot the histogram of IMDB ratings.

In [180]:
```
plt.hist(movies.IMDB_Rating)
```

```
C:\Users\yyezeret\AppData\Local\Continuum\Anaconda2\envs\dviz\lib\site-packages\numpy\c
ore\_methods.py:32: RuntimeWarning: invalid value encountered in reduce
  return umr_minimum(a, axis, None, out, keepdims, initial)
C:\Users\yyezeret\AppData\Local\Continuum\Anaconda2\envs\dviz\lib\site-packages\numpy\c
ore\_methods.py:28: RuntimeWarning: invalid value encountered in reduce
  return umr_maximum(a, axis, None, out, keepdims, initial)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-180-b218023ece0d> in <module>()
----> 1 plt.hist(movies.IMDB_Rating)

~\AppData\Local\Continuum\Anaconda2\envs\dviz\lib\site-packages\matplotlib\pyplot.py in
 hist(x, bins, range, density, weights, cumulative, bottom, histtype, align, orientatio
n, rwidth, log, color, label, stacked, normed, hold, data, **kwargs)
   3135                      histtype=histtype, align=align, orientation=orientation,
   3136                      rwidth=rwidth, log=log, color=color, label=label,
-> 3137                      stacked=stacked, normed=normed, data=data, **kwargs)
   3138     finally:
   3139         ax._hold = washold

~\AppData\Local\Continuum\Anaconda2\envs\dviz\lib\site-packages\matplotlib\__init__.py
 in inner(ax, *args, **kwargs)
   1865                          "the Matplotlib list!)" % (label_namer, func.__name__),
   1866                          RuntimeWarning, stacklevel=2)
-> 1867             return func(ax, *args, **kwargs)
   1868
   1869         inner.__doc__ = _add_data_doc(inner.__doc__,

~\AppData\Local\Continuum\Anaconda2\envs\dviz\lib\site-packages\matplotlib\axes\_axes.p
y in hist(***failed resolving arguments***)
   6637             # this will automatically overwrite bins,
   6638             # so that each histogram uses the same bins
-> 6639             m, bins = np.histogram(x[i], bins, weights=w[i], **hist_kwargs)
   6640             m = m.astype(float)  # causes problems later if it's an int
   6641             if mlast is None:

~\AppData\Local\Continuum\Anaconda2\envs\dviz\lib\site-packages\numpy\lib\histograms.py
 in histogram(a, bins, range, normed, weights, density)
    700     a, weights = _ravel_and_check_weights(a, weights)
    701
--> 702     bin_edges, uniform_bins = _get_bin_edges(a, bins, range, weights)
    703
    704     # Histogram is an integer or a float array depending on the weights.

~\AppData\Local\Continuum\Anaconda2\envs\dviz\lib\site-packages\numpy\lib\histograms.py
 in _get_bin_edges(a, bins, range, weights)
    353             raise ValueError('`bins` must be positive, when an integer')
    354
--> 355         first_edge, last_edge = _get_outer_edges(a, range)
    356
    357     elif np.ndim(bins) == 1:

~\AppData\Local\Continuum\Anaconda2\envs\dviz\lib\site-packages\numpy\lib\histograms.py
 in _get_outer_edges(a, range)
    240         if first_edge > last_edge:
    241             raise ValueError(
--> 242                 'max must be larger than min in range parameter.')
    243         if not (np.isfinite(first_edge) and np.isfinite(last_edge)):
    244             raise ValueError(

ValueError: max must be larger than min in range parameter.
```



If you run the above cell, you get an error? What's going on?

The problem is that the column contains `NaN` (Not a Number) values, i.e. missing data. The following command check whether each value is a `NaN` and returns the result.

In [181]:
```
movies.IMDB_Rating.isna()
```

```
Out[181]: 0       False
          1       False
          2       False
          3        True
          4       False
          5        True
          6       False
          7       False
          8       False
          9       False
          10      False
          11      False
          12      False
          13       True
          14      False
          15       True
          16      False
          17      False
          18      False
          19      False
          20      False
          21      False
          22      False
          23      False
          24      False
          25       True
          26       True
          27      False
          28      False
          29       True
                  ...
          3171    False
          3172    False
          3173    False
          3174    False
          3175    False
          3176    False
          3177    False
          3178    False
          3179     True
          3180    False
          3181    False
          3182     True
          3183    False
          3184    False
          3185    False
          3186    False
          3187    False
          3188     True
          3189     True
          3190    False
          3191    False
          3192     True
          3193    False
          3194    False
          3195    False
          3196    False
          3197     True
          3198    False
          3199    False
```

```
3200    False
Name: IMDB_Rating, Length: 3201, dtype: bool
```

As you can see there are a bunch of missing rows. You can count them.

In [182]:
```
sum(movies.IMDB_Rating.isna())
```

Out[182]: 213

or drop them.

In [183]:
```
IMDB_ratings_nan_dropped = movies.IMDB_Rating.dropna()
len(IMDB_ratings_nan_dropped)
```

Out[183]: 2988

In [184]:
```
len(IMDB_ratings_nan_dropped) + 213
```

Out[184]: 3201

The `dropna` can be applied to the dataframe too.

Q: drop rows from `movies` dataframe where either `IMDB_Rating` or `IMDB_Votes` is `NaN`.

In [187]:
```
# TODO
movies=movies.dropna(axis=0, subset=['IMDB_Rating','IMDB_Votes'])
```

In [188]:
```
# Both should be zero.
print(sum(movies.IMDB_Rating.isna()), sum(movies.IMDB_Votes.isna()))
```

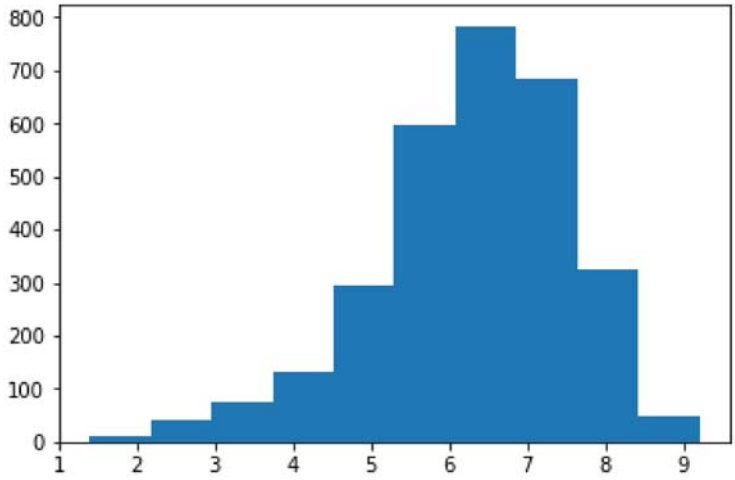
```
0 0
```

How does `matplotlib` decides the bins? Actually `matplotlib`'s `hist` function uses `numpy`'s `histogram` function under the hood.

Q: Plot the histogram of movie ratings (`IMDB_Rating`) using the `plt.hist()` function.

In [190]:
```
#TODO
plt.hist(movies.IMDB_Rating)
```

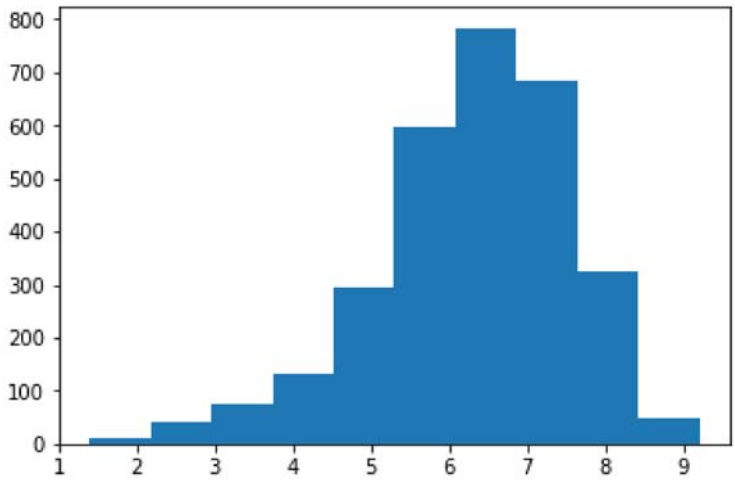
Out[190]:
```
(array([  9.,  39.,  76., 133., 293., 599., 784., 684., 323.,  48.]),
 array([1.4 , 2.18, 2.96, 3.74, 4.52, 5.3 , 6.08, 6.86, 7.64, 8.42, 9.2 ]),
 <a list of 10 Patch objects>)
```



Have you noticed that this function returns three objects? Take a look at the documentation here (http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.hist) to figure out what they are.

To get the returned three objects:

In [194]:
```
n_raw, bins_raw, patches = plt.hist(movies.IMDB_Rating)
```



In [195]:
```
print(n_raw)
print(bins_raw)
```

```
[  9.  39.  76. 133. 293. 599. 784. 684. 323.  48.]
[1.4  2.18 2.96 3.74 4.52 5.3  6.08 6.86 7.64 8.42 9.2 ]
```

Here, `n_raw` contains the values of histograms, i.e., the number of movies in each of the 10 bins. Thus, the sum of the elements in `n_raw` should be equal to the total number of movies.

Q: Test whether the sum of values in `n_raw` is equal to the number of movies in the `movies` dataset

In [202]:
```
print(sum(n_raw))
print(len(movies))
```

```
2988.0
2988
```

The second returned object (`bins_raw`) is a list containing the edges of the 10 bins: the first bin is [1.4, 2.18], the second [2.18, 2.96], and so on. What's the width of the bins?

In [200]:
```
np.diff(bins_raw)
```

Out[200]:
```
array([0.78, 0.78, 0.78, 0.78, 0.78, 0.78, 0.78, 0.78, 0.78, 0.78])
```

The width is same as the maximum value minus minimum value, divided by 10.

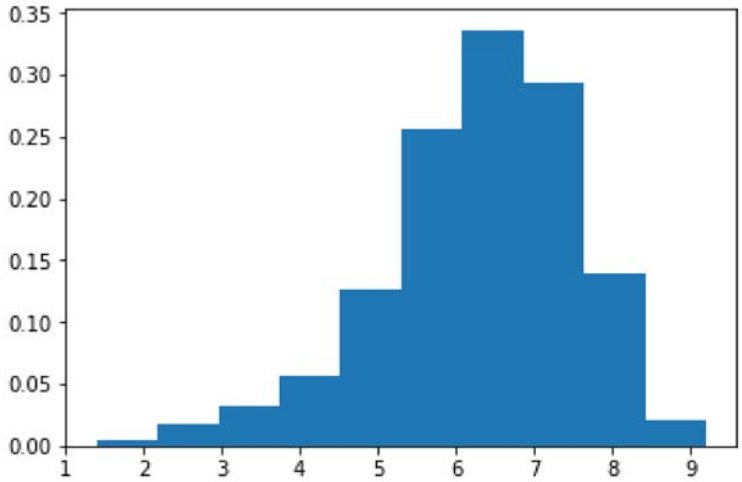
In [203]:
```
min_rating = min(movies.IMDB_Rating)
max_rating = max(movies.IMDB_Rating)
print(min_rating, max_rating)
print( (max_rating-min_rating) / 10 )
```

```
1.4 9.2
0.7799999999999999
```

Now, let's plot a normalized (density) histogram.

In [204]:
```
n, bins, patches = plt.hist(movies.IMDB_Rating, density=True)
print(n)
print(bins)
```

```
[0.0038616  0.0167336  0.03260907 0.05706587 0.12571654 0.25701095
 0.33638829 0.29348162 0.13858854 0.0205952 ]
[1.4  2.18 2.96 3.74 4.52 5.3  6.08 6.86 7.64 8.42 9.2 ]
```



The ten bins do not change. But now `n` represents the density of the data inside each bin. In other words, the sum of the area of each bar will equal to 1.

Q: Can you verify this?

Hint: the area of each bar is calculated as height * width. You may get something like 0.99999999999999978 instead of 1.

In [212]:
```
np.diff(bins)
print(bins)
```

```
[1.4  2.18 2.96 3.74 4.52 5.3  6.08 6.86 7.64 8.42 9.2 ]
```

In [213]:
```
# TODO
print(sum(np.diff(bins)*n))
```

```
1.0
```

Anyway, these data generated from the `hist` function is calculated from `numpy`'s `histogram` function.
https://docs.scipy.org/doc/numpy/reference/generated/numpy.histogram.html
(https://docs.scipy.org/doc/numpy/reference/generated/numpy.histogram.html)

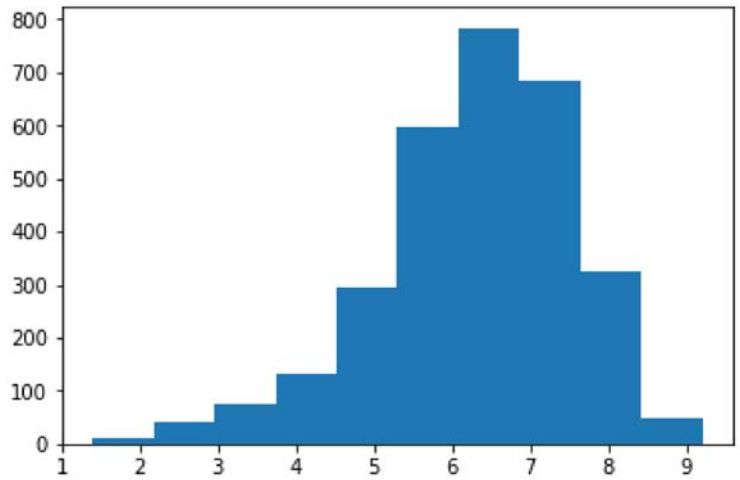
Note that the result of `np.histogram()` is same as that of `plt.hist()`.

In [214]:
```
np.histogram(movies.IMDB_Rating)
```

Out[214]:
```
(array([  9,  39,  76, 133, 293, 599, 784, 684, 323,  48], dtype=int64),
 array([1.4 , 2.18, 2.96, 3.74, 4.52, 5.3 , 6.08, 6.86, 7.64, 8.42, 9.2 ]))
```

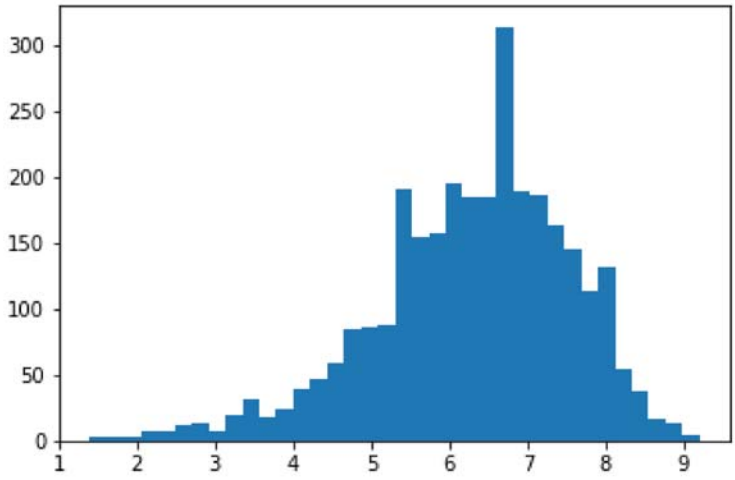
In [215]:
```
plt.hist(movies.IMDB_Rating)
```

Out[215]:
```
(array([  9.,  39.,  76., 133., 293., 599., 784., 684., 323.,  48.]),
 array([1.4 , 2.18, 2.96, 3.74, 4.52, 5.3 , 6.08, 6.86, 7.64, 8.42, 9.2 ]),
 <a list of 10 Patch objects>)
```



If you look at the documentation, you can see that `numpy` uses simply 10 as the default number of bins. But you can set it manually or set it to be `auto`, which is the "Maximum of the `sturges` and `fd` estimators.". Let's try this `auto` option.

In [216]:
```
_ = plt.hist(movies.IMDB_Rating, bins='auto')
```



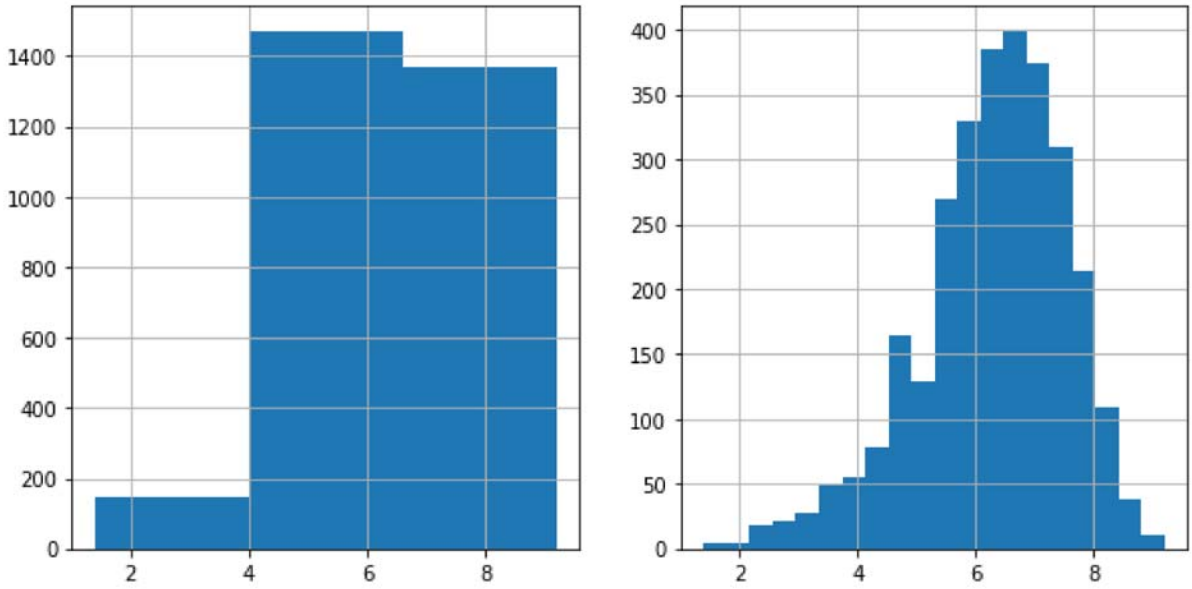
Consequences of the binning parameter

Let's explore the effect of bin size using small multiples. In `matplotlib`, you can use subplot (https://www.google.com/search?client=safari&rls=en&q=matplotlib+subplot&ie=UTF-8&oe=UTF-8) to put multiple plots into a single figure.

For instance, you can do something like:

In [217]:
```
plt.figure(figsize=(10,5))
plt.subplot(1,2,1)
movies.IMDB_Rating.hist(bins=3)
plt.subplot(1,2,2)
movies.IMDB_Rating.hist(bins=20)
```

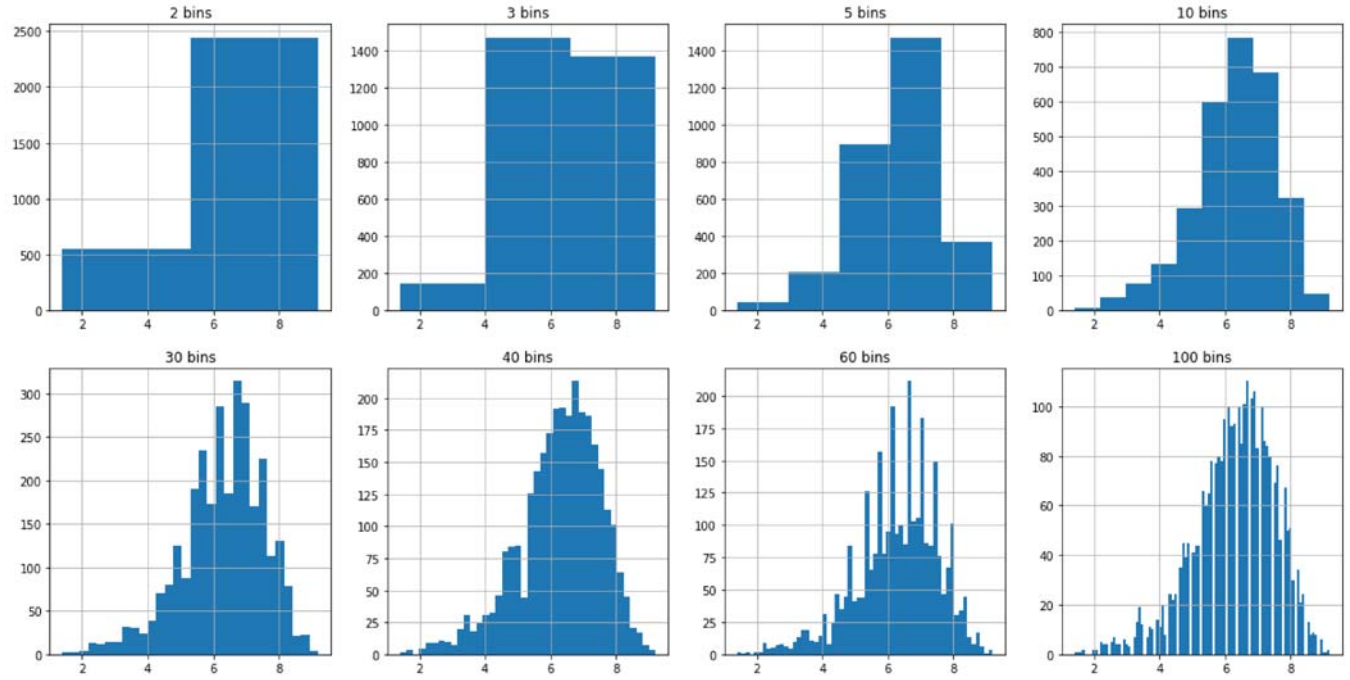
Out[217]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x205939ecdd8>
```



What does the argument in `plt.subplot(1,2,1)` mean? If you're not sure, check out:
http://stackoverflow.com/questions/3584805/in-matplotlib-what-does-the-argument-mean-in-fig-add-subplot111 (http://stackoverflow.com/questions/3584805/in-matplotlib-what-does-the-argument-mean-in-fig-add-subplot111)

Q: ceate 8 subplots (2 rows and 4 columns) with the following `binsizes`.

In [224]:
```
binsizes = [2, 3, 5, 10, 30, 40, 60, 100 ]
plt.figure(figsize=(20,10))
for i, bins in enumerate(binsizes):
    plt.subplot(2,4,i+1) #i needs to start with 1 not 0
    movies.IMDB_Rating.hist(bins=bins)
    plt.title("{} bins".format(bins))
```



Do you see the issues with having too few bins or too many bins? In particular, do you notice weird patterns that emerge from `bins=30`?

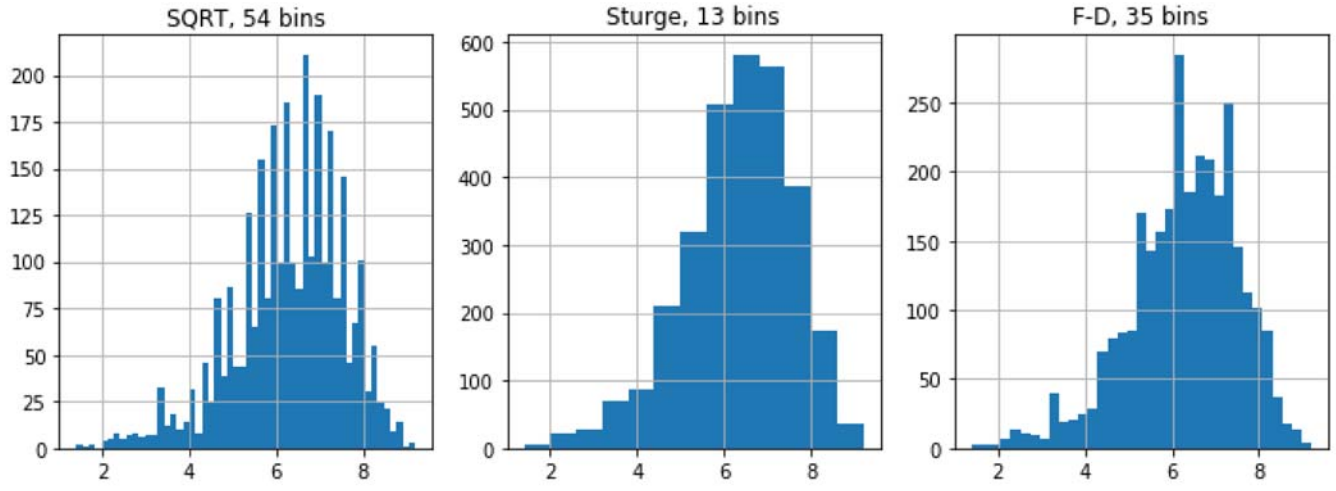
Q: Can you guess why do you see such patterns? What are the peaks and what are the empty bars? What do they tell you about choosing the binsize in histograms?

Formulae for choosing the number of bins.

We can manually choose the number of bins based on those formulae.

In [223]:
```
N = len(movies)

plt.figure(figsize=(12,4))

# Sqrt
nbins = int(np.sqrt(N))

plt.subplot(1,3,1)
plt.title("SQRT, {} bins".format(nbins))
movies.IMDB_Rating.hist(bins=nbins)

# Sturge's formula
nbins = int(np.ceil(np.log2(N) + 1))

plt.subplot(1,3,2)
plt.title("Sturge, {} bins".format(nbins))
movies.IMDB_Rating.hist(bins=nbins)

# Freedman-Diaconis
iqr = np.percentile(movies.IMDB_Rating, 75) - np.percentile(movies.IMDB_Rating, 25)
width = 2*iqr/np.power(N, 1/3)
nbins = int((max(movies.IMDB_Rating) - min(movies.IMDB_Rating)) / width)

plt.subplot(1,3,3)
plt.title("F-D, {} bins".format(nbins))
movies.IMDB_Rating.hist(bins=nbins)
```

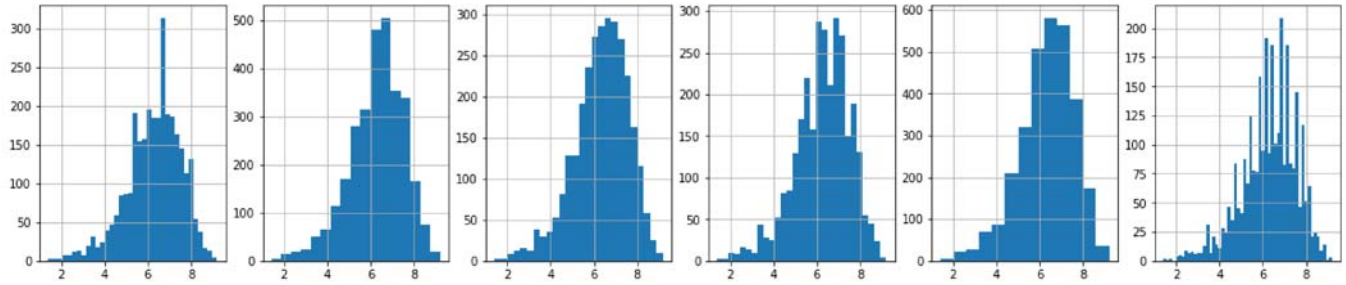
Out[223]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2059570bac8>
```



But we can also use built-in formulae too. Let's try all of them.

In [225]:
```
plt.figure(figsize=(20,4))

plt.subplot(161)
movies.IMDB_Rating.hist(bins='fd')

plt.subplot(162)
movies.IMDB_Rating.hist(bins='doane')

plt.subplot(163)
movies.IMDB_Rating.hist(bins='scott')

plt.subplot(164)
movies.IMDB_Rating.hist(bins='rice')

plt.subplot(165)
movies.IMDB_Rating.hist(bins='sturges')

plt.subplot(166)
movies.IMDB_Rating.hist(bins='sqrt')
```

Out[225]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x205953dae48>
```



Some are decent, but several of them tend to overestimate the good number of bins. As you have more data points, some of the formulae may overestimate the necessary number of bins. Particularly in our case, because of the precision issue, we shouldn't increase the number of bins too much.

So how should we choose the number of bins?

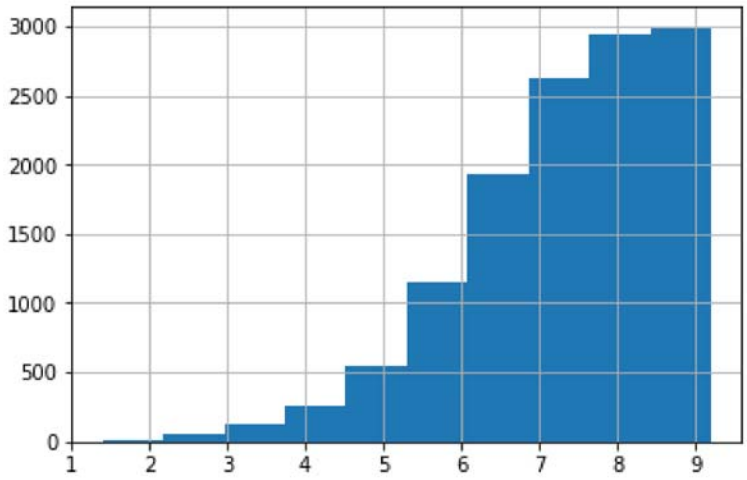
So what's the conclusion? use Scott's rule or Sturges' formula? No, I think the take away is that you should understand how the inappropriate number of bins can mislead you and you should try multiple number of bins to obtain the most accurate picture of the data. Although the 'default' may work in most cases, don't blindly trust it! Don't judge a dataset (maybe more like "a column") based on a single histogram. Try multiple parameters to get the full picture!

CDF (Cumulative distribution function)

Drawing a CDF is very easy. Because it's very common data visualization, histogram has an option called `cumulative`.

```
In [226]: movies.IMDB_Rating.hist(cumulative=True)
```

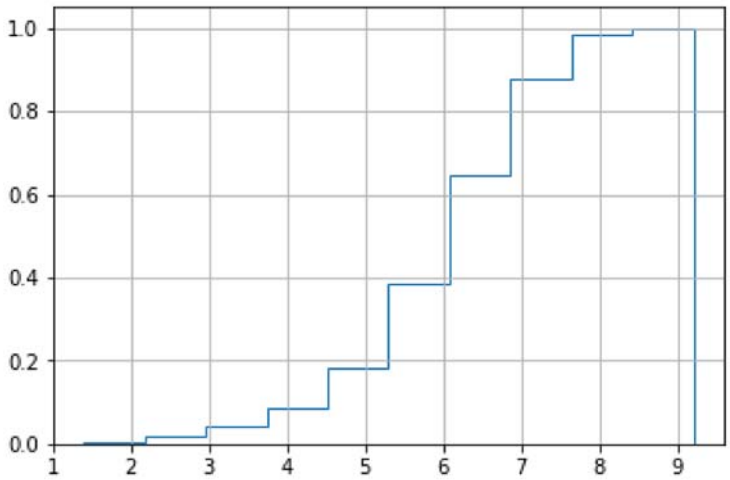
```
Out[226]: <matplotlib.axes._subplots.AxesSubplot at 0x2059528ce10>
```



You can also combine with options such as `histtype` and `density`.

```
In [227]: movies.IMDB_Rating.hist(histtype='step', cumulative=True, density=True)
```

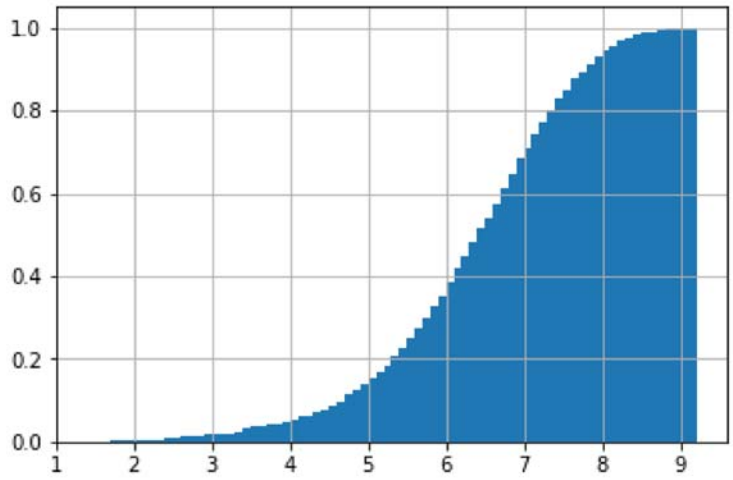
```
Out[227]: <matplotlib.axes._subplots.AxesSubplot at 0x205945d7128>
```



And increase the number of bins.

In [228]:
```
movies.IMDB_Rating.hist(cumulative=True, density=True, bins=1000)
```

Out[228]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x205944d0dd8>
```



This method works fine. By increasing the number of bins, you can get a CDF in the resolution that you want. But let's also try it more manually. First, we should sort all the values.

In [265]:
```
rating_sorted = movies.IMDB_Rating.sort_values()
rating_sorted.head()
```

Out[265]:
```
1247    1.4
406     1.5
1754    1.6
1590    1.7
1515    1.7
Name: IMDB_Rating, dtype: float64
```

We need to know the number of data points,

In [266]:
```
N = len(rating_sorted)
N
```

Out[266]:
```
2988
```

And I think this may be useful for you.

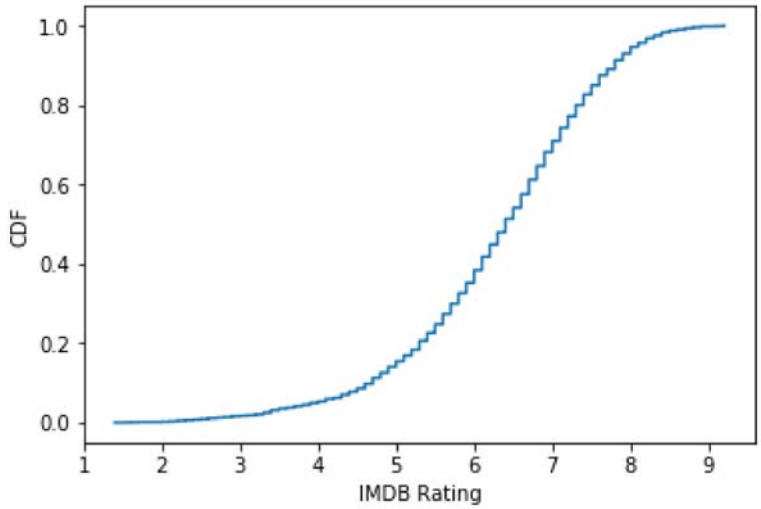
In [267]:
```
n = 50
np.linspace(1/n, 1.0, num=n)
```

Out[267]:
```
array([0.02, 0.04, 0.06, 0.08, 0.1 , 0.12, 0.14, 0.16, 0.18, 0.2 , 0.22,
       0.24, 0.26, 0.28, 0.3 , 0.32, 0.34, 0.36, 0.38, 0.4 , 0.42, 0.44,
       0.46, 0.48, 0.5 , 0.52, 0.54, 0.56, 0.58, 0.6 , 0.62, 0.64, 0.66,
       0.68, 0.7 , 0.72, 0.74, 0.76, 0.78, 0.8 , 0.82, 0.84, 0.86, 0.88,
       0.9 , 0.92, 0.94, 0.96, 0.98, 1.  ])
```

Q: now you're ready to draw a proper CDF. Draw the CDF plot of this data.

In [293]:
```
levels = np.linspace(1. / len(rating_sorted), 1, len(rating_sorted))
plt.xlabel('IMDB Rating')
plt.ylabel('CDF')
plt.step(rating_sorted, levels, where='post')
```

Out[293]: [<matplotlib.lines.Line2D at 0x205986955c0>]



A bit more histogram with altair

As you may remember, you can get a pandas dataframe from `vega_datasets` package and use it to create visualizations. But, if you use `altair`, you can simply pass the URL instead of the actual data.

In [294]:
```
vega_datasets.data.movies.url
```

Out[294]: 'https://vega.github.io/vega-datasets/data/movies.json'

In [282]:
```
import altair as alt
print(alt.renderers.active)
alt.__version__
alt.renderers.enable('notebook')
```

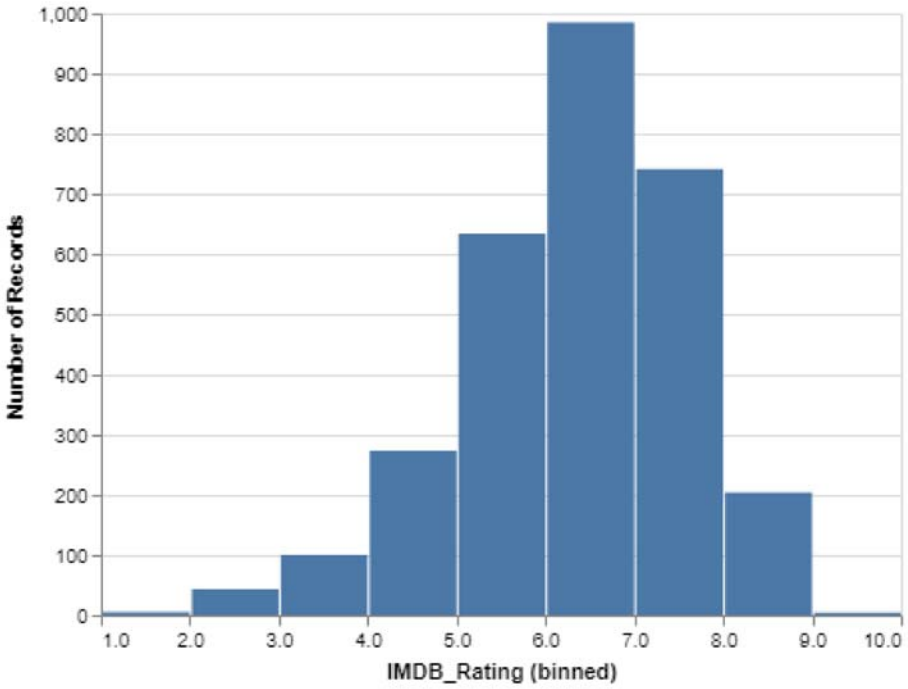
```
default
```

Out[282]: RendererRegistry.enable('notebook')

As mentioned before, in `altair` histogram is not special. It is just a plot that use bars (`mark_bar()`) where X axis is defined by `IMDB_Rating` with bins (`bin=True`), and Y axis is defined by `count()` aggregation function.

In [295]:
```
alt.Chart(vega_datasets.data.movies.url).mark_bar().encode(
    alt.X("IMDB_Rating:Q",  bin=True),
    alt.Y('count()')
)
```

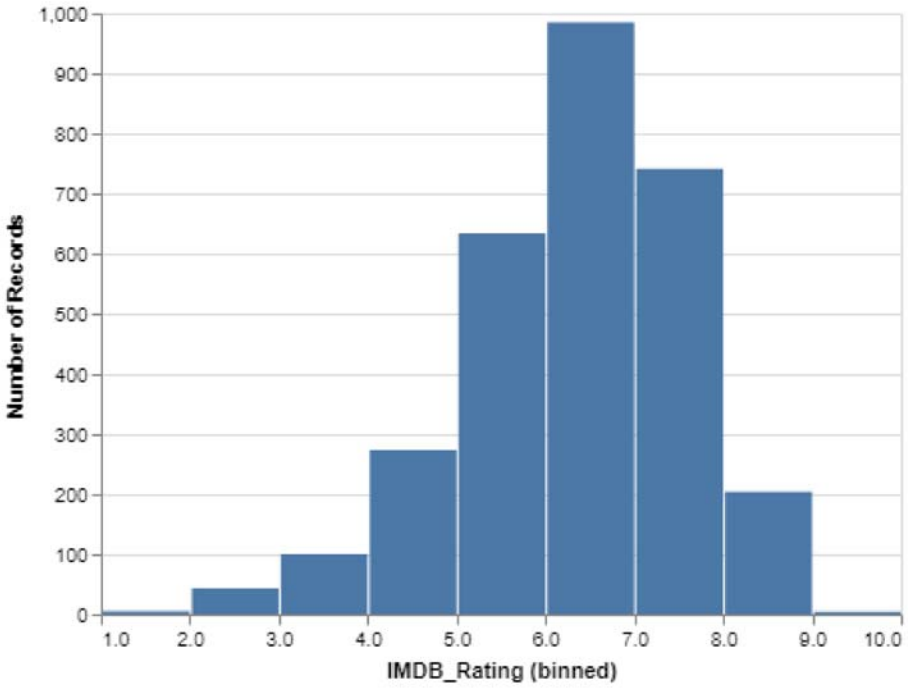
Out[295]:



Have you noted that it is `IMDB_Rating:Q` not `IMDB_Rating`? This is a shorthand for

In [296]:
```
alt.Chart(vega_datasets.data.movies.url).mark_bar().encode(
    alt.X('IMDB_Rating', type='quantitative', bin=True),
    alt.Y(aggregate='count', type='quantitative')
)
```

Out[296]:

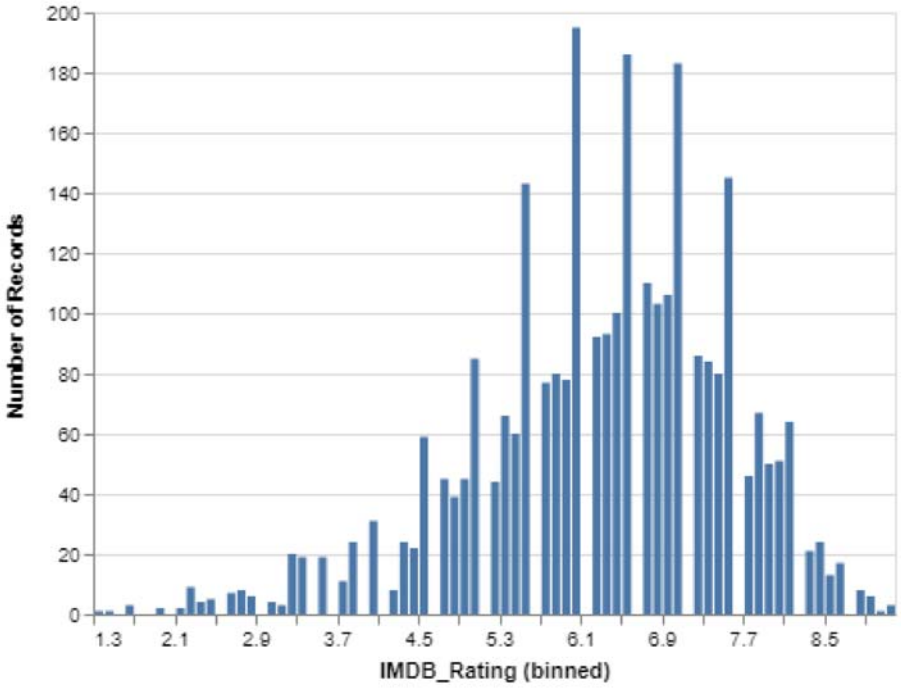


In altair, you want to specify the data types using one of the four categories: quantitative, ordinal, nominal, and temporal. https://altair-viz.github.io/user_guide/encoding.html#data-types (https://altair-viz.github.io/user_guide/encoding.html#data-types)

Although you can adjust the bins in `altair`, it does not encourage you to set the bins directly. For instance, although there is `step` parameter that directly sets the bin size, there are parameters such as `maxbins` (maximum number of bins) or `minstep` (minimum allowable step size), or `nice` (attemps to make the bin boundaries more human-friendly), that encourage you not to specify the bins directly.

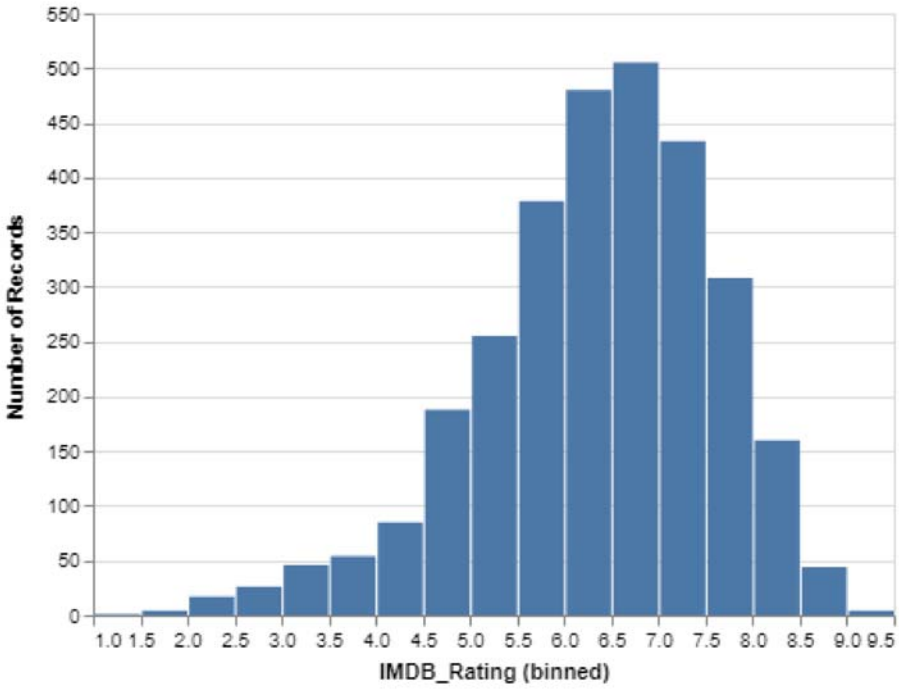
In [297]:
```
from altair import Bin

alt.Chart(vega_datasets.data.movies.url).mark_bar().encode(
    alt.X("IMDB_Rating:Q",  bin=Bin(step=0.1)),
    alt.Y('count()')
)
```

Out[297]:



In [285]:
```
alt.Chart(vega_datasets.data.movies.url).mark_bar().encode(
    alt.X("IMDB_Rating:Q",  bin=Bin(nice=True, maxbins=20)),
    alt.Y('count()')
)
```

Out[285]:



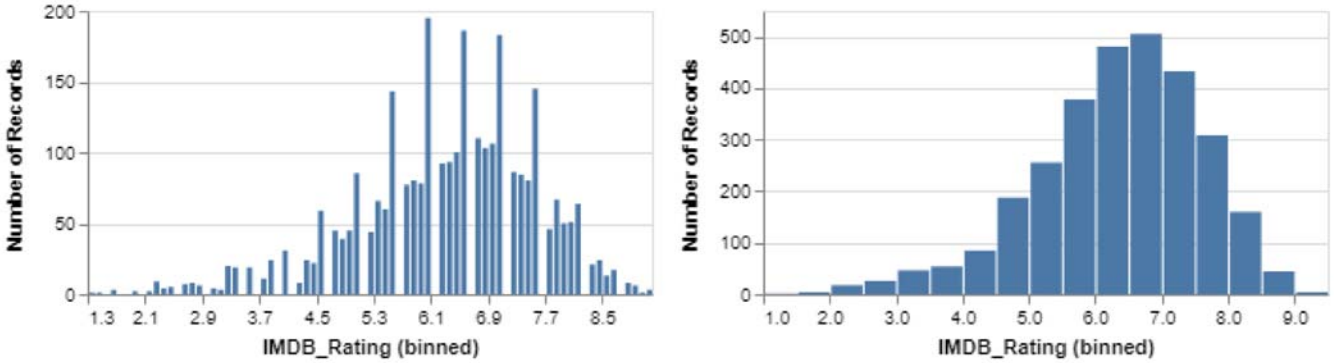
Composing charts in altair

`altair` has a very nice way to compose multiple plots. Two histograms side by side? just do the following.

In [298]:
```
chart1 = alt.Chart(vega_datasets.data.movies.url).mark_bar().encode(
    alt.X("IMDB_Rating:Q",  bin=Bin(step=0.1)),
    alt.Y('count()')
).properties(
    width=300,
    height=150
)
chart2 = alt.Chart(vega_datasets.data.movies.url).mark_bar().encode(
    alt.X("IMDB_Rating:Q",  bin=Bin(nice=True, maxbins=20)),
    alt.Y('count()')
).properties(
    width=300,
    height=150
)
```

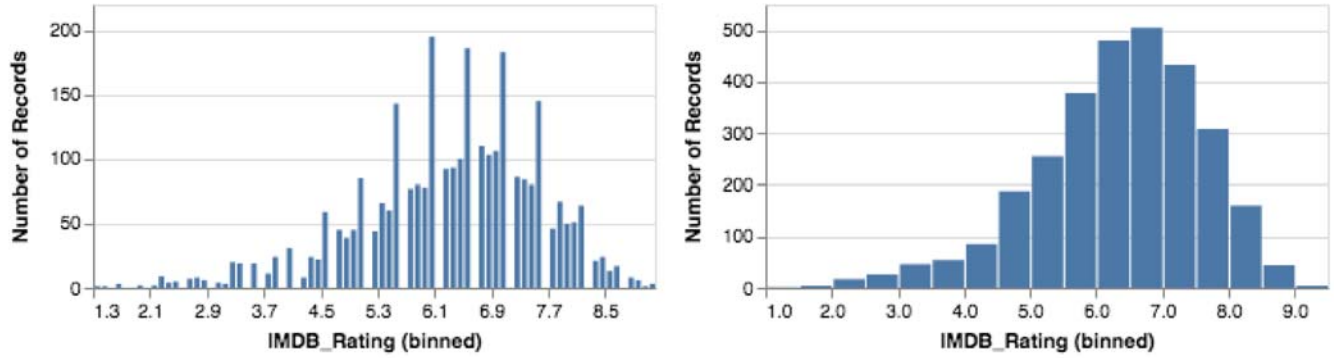
In [299]:
```
chart1 | chart2
```

Out[299]:



In [63]:
```
alt.hconcat(chart1, chart2)
```

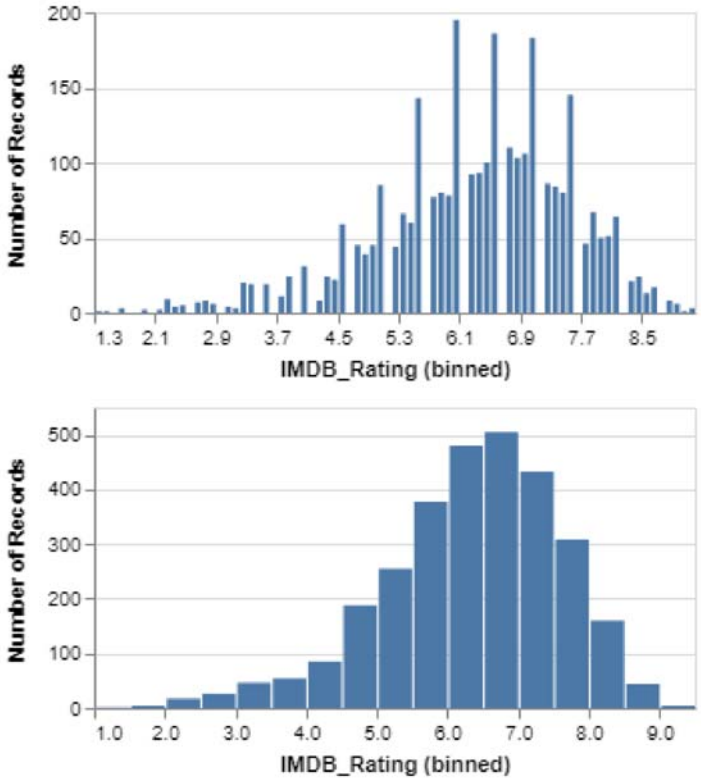
Out[63]:



Vertical commposition?

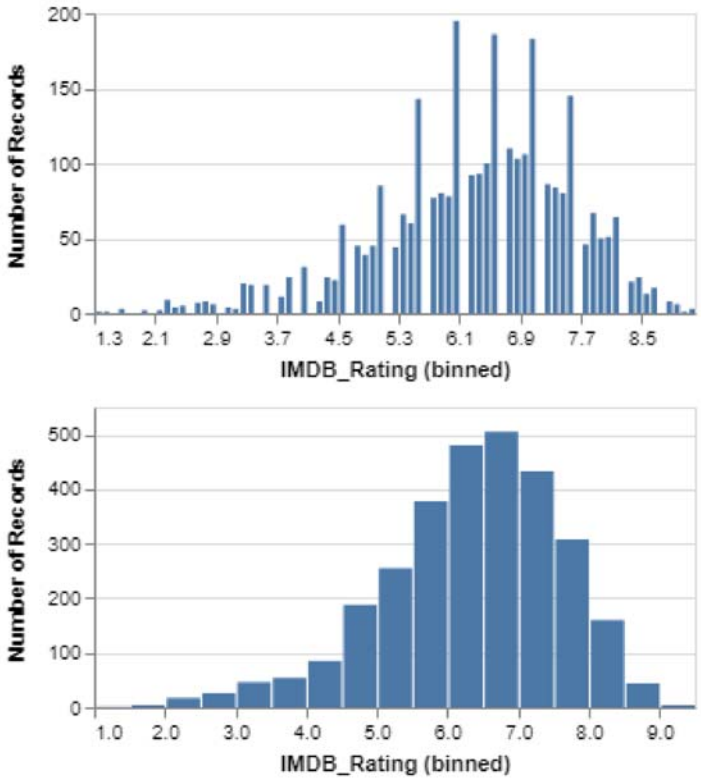
In [300]:
```
alt.vconcat(chart1, chart2)
```

Out[300]:



In [301]:
```
chart1 & chart2
```

Out[301]:



Shall we avoid some repetitions? You can define a *base* empty chart first and then assign encodings later when you put together multiple charts together. Here is an example: https://altair-viz.github.io/user_guide/compound_charts.html#repeated-charts (https://altair-viz.github.io/user_guide/compound_charts.html#repeated-charts)

Q: Using the base chart approach to create a 2x2 chart where the top row shows the two histograms of `IMDB_Rating` with `maxbins`=10 and 50 respectively, and the bottom row shows another two histograms of `IMDB_Votes` with `maxbins`=10 and 50.

In [322]:
```
movies=vega_datasets.data.movies.url

base = alt.Chart().mark_bar().encode(
).properties(
    width=150,
    height=150
).interactive()

chart = alt.vconcat(data=movies)
for x_encoding in ['IMDB_Rating:Q', 'IMDB_Votes:Q']:
    row = alt.hconcat()
    for max_bins in [10,50]:
        row |= base.encode(alt.X(x_encoding,  bin=Bin(nice=True, maxbins=max_bins)), alt
.Y('count()'))
    chart &= row
chart
```

Out[322]: