Module 9: Estimation

In [164]:
```
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import altair as alt
import pandas as pd
import scipy.stats as ss

%matplotlib inline
```

Kernel density estimation

Some resources on KDE: http://yyahnwiki.appspot.com/Kernel%20density%20estimation (http://yyahnwiki.appspot.com/Kernel%20density%20estimation)

Let's import the IMDb data.

In [165]:
```
import vega_datasets

movies = vega_datasets.data.movies()
movies.head()
```

Out[165]:

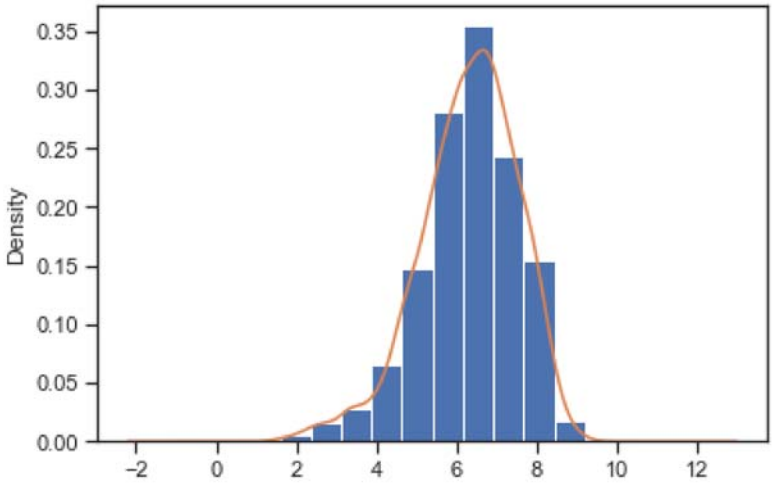
	Creative_Type	Director	Distributor	IMDB_Rating	IMDB_Votes	MPAA_Rating	Major_Genre	Proc
0	None	None	Gramercy	6.1	1071.0	R	None	8000
1	None	None	Strand	6.9	207.0	R	Drama	3000
2	None	None	Lionsgate	6.8	865.0	None	Comedy	2500
3	None	None	Fine Line	NaN	NaN	None	Comedy	3000
4	Contemporary Fiction	None	Trimark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

In [166]:
```
movies = movies.dropna(subset=['IMDB_Rating', 'Rotten_Tomatoes_Rating'])
```

We can plot histogram and KDE using pandas:

```
In [167]: movies['IMDB_Rating'].hist(bins=10, density=True)
          movies['IMDB_Rating'].plot(kind='kde')
```

```
Out[167]: <matplotlib.axes._subplots.AxesSubplot at 0x1400c87bac8>
```

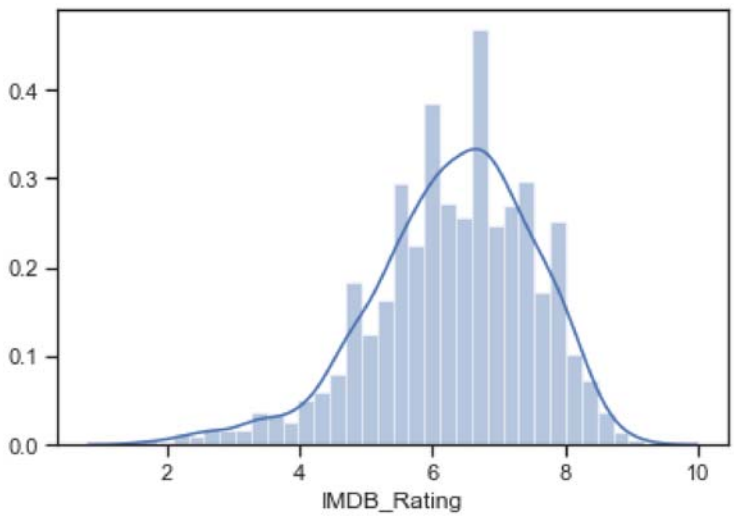


Or using seaborn:

```
In [168]: sns.distplot(movies['IMDB_Rating'])
```

```
C:\Users\yyezeret\AppData\Local\Continuum\Anaconda2\envs\dviz\lib\site-packages\scipy\stats\stats.py:1713: FutureWarning: Using a non-tuple sequence for multidimensional indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will be interpreted as an array index, `arr[np.array(seq)]`, which will result either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

```
Out[168]: <matplotlib.axes._subplots.AxesSubplot at 0x14005b0c400>
```

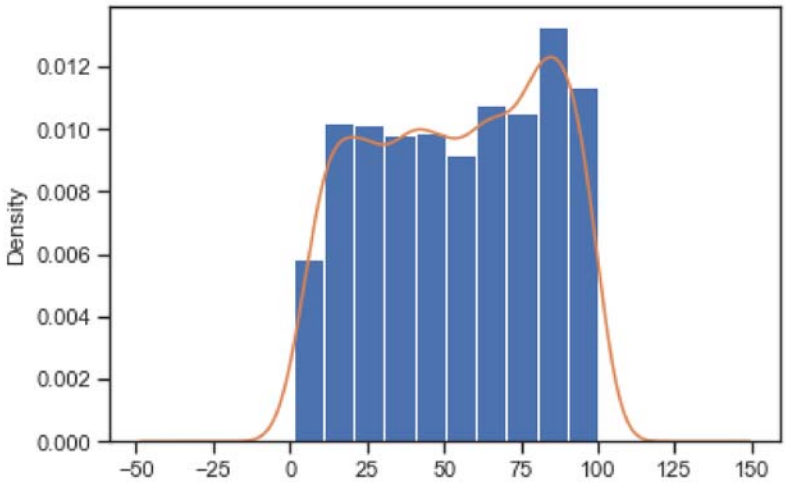


Ah, too many bins.. 😎

Q: Can you plot the histogram and KDE of the `Rotten_Tomatoes_Rating`?

In [169]:
```
# TODO: implement this using pandas
movies['Rotten_Tomatoes_Rating'].hist(bins=10, density=True)
movies['Rotten_Tomatoes_Rating'].plot(kind='kde')
```

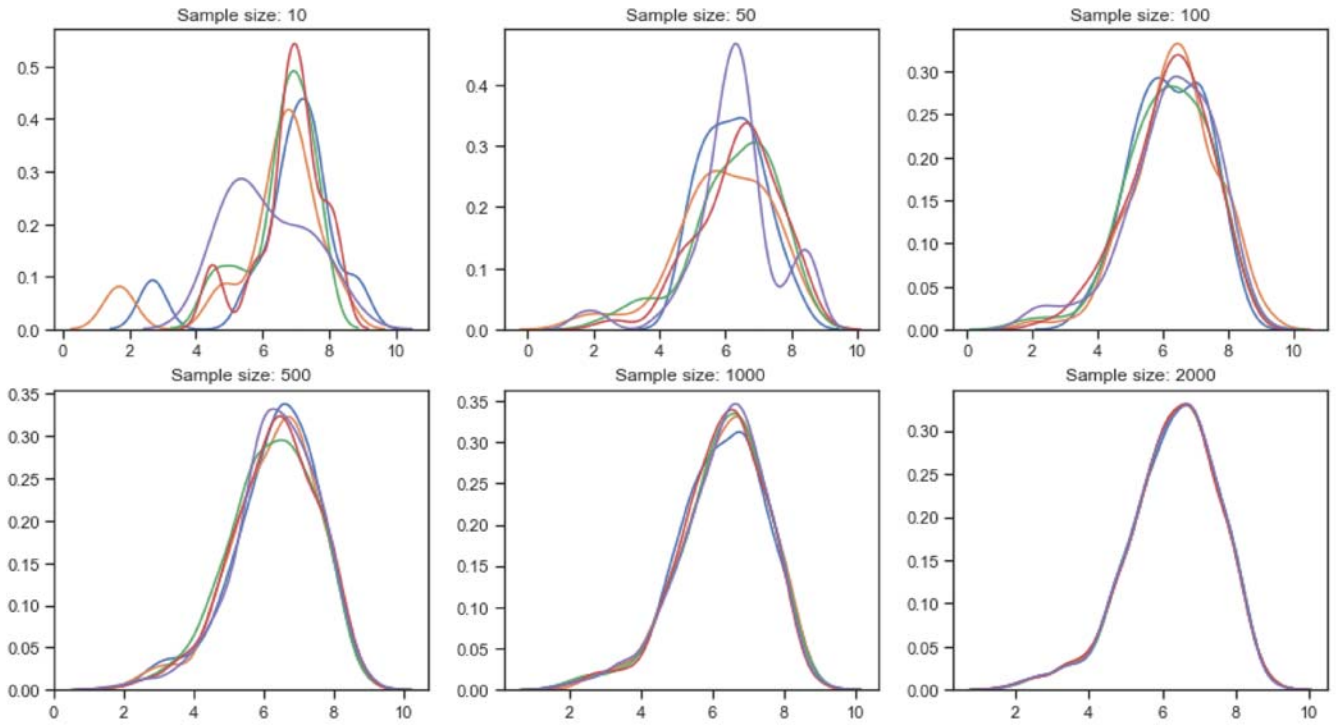
Out[169]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x14005cafe10>
```



We can get a random sample using the pandas' **`sample()`** (http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.sample.html) function. The **`kdeplot()`** (https://stanford.edu/~mwaskom/software/seaborn/generated/seaborn.kdeplot.html) function in seaborn provides many options (like kernel types) to do KDE. Let's sample some data points and see how does KDE plot changes with the size of the samples.

In [170]:
```
f = plt.figure(figsize=(15,8))
plt.xlim(0, 10)

sample_sizes = [10, 50, 100, 500, 1000, 2000]
for i, N in enumerate(sample_sizes, 1):
    plt.subplot(2,3,i)
    plt.title("Sample size: {}".format(N))
    for j in range(5):
        s = movies['IMDB_Rating'].sample(N)
        sns.kdeplot(s, kernel='gau', legend=False)
```

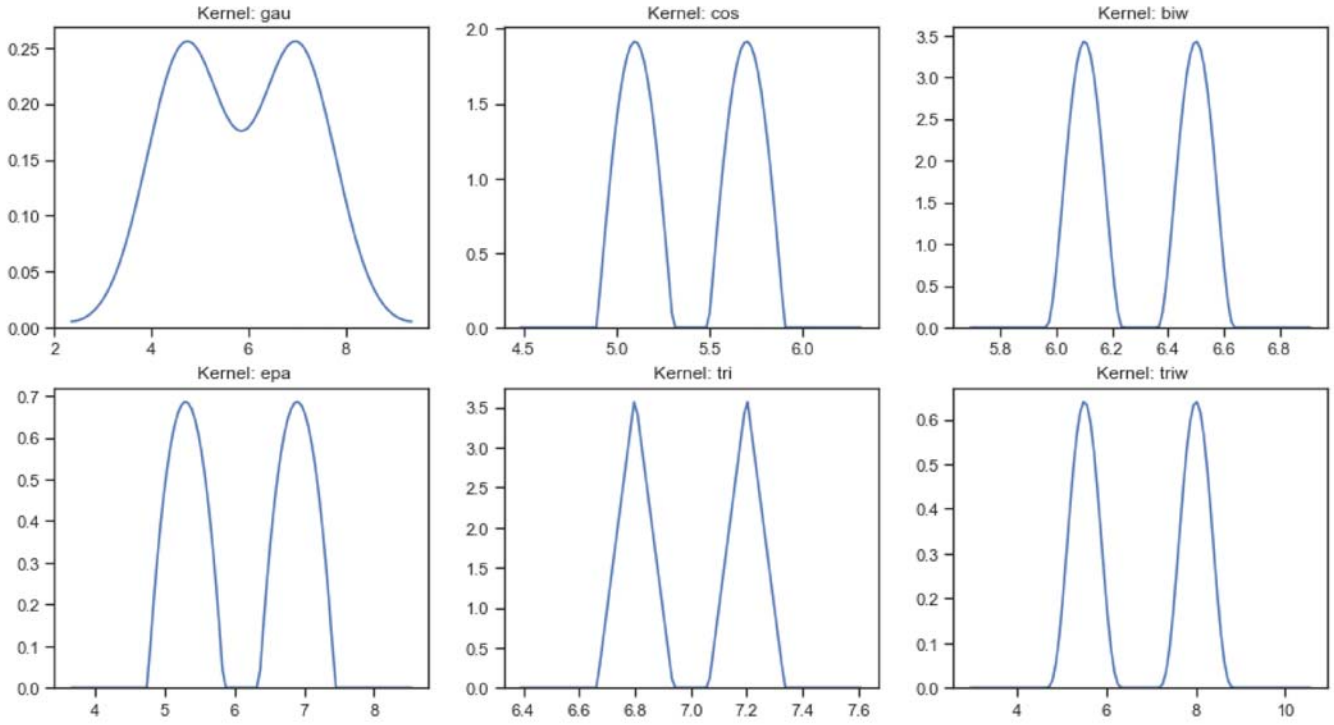
```
C:\Users\yyezeret\AppData\Local\Continuum\Anaconda2\envs\dviz\lib\site-packages\scipy\stats\stats.py:1713: FutureWarning: Using a non-tuple sequence for multidimensional indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will be interpreted as an array index, `arr[np.array(seq)]`, which will result either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



Let's try all kernel types supported by seaborn's `kdeplot()`. Plot the same 2x3 grid with all kernels: https://seaborn.pydata.org/generated/seaborn.kdeplot.html#seaborn.kdeplot (https://seaborn.pydata.org/generated/seaborn.kdeplot.html#seaborn.kdeplot) To see how do the kernels look like, just sample 2 data points and plot them.

In [171]:
```
#Implement Here
#kernel : {‘gau’ | ‘cos’ | ‘biw’ | ‘epa’ | ‘tri’ | ‘triw’ }, optional
f = plt.figure(figsize=(15,8))


kernel= ["gau","cos","biw","epa","tri","triw"]
for i, K in enumerate(kernel, 1):
    plt.subplot(2,3,i)
    plt.title(" Kernel: {}".format(K))
    s = movies['IMDB_Rating'].sample(2)
    sns.kdeplot(s, kernel=K, legend=False)
```

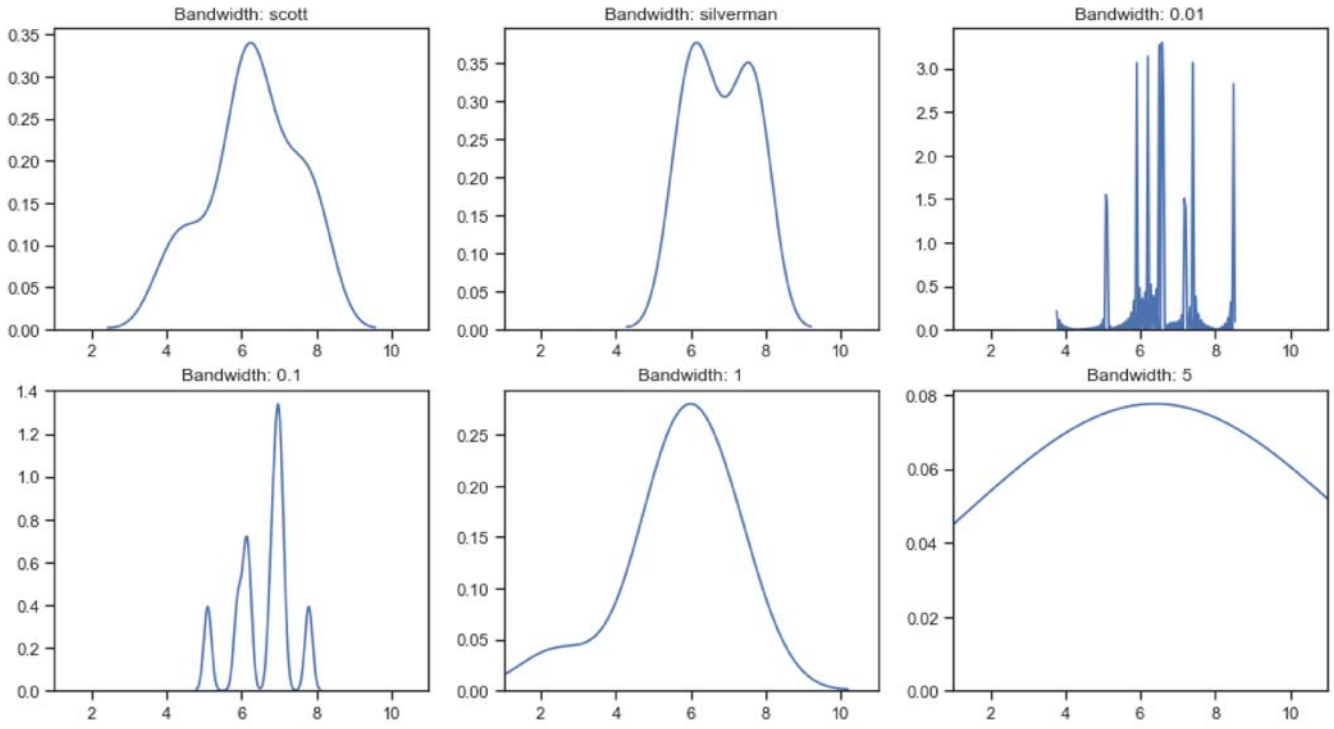
```
C:\Users\yyezeret\AppData\Local\Continuum\Anaconda2\envs\dviz\lib\site-packages\scipy\stats\stats.py:1713: FutureWarning: Using a non-tuple sequence for multidimensional indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will be interpreted as an array index, `arr[np.array(seq)]`, which will result either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



Q: We can also play with the bandwidth option. Make sure to set the `xlim` so that all plots have the same x range, so that we can compare.

In [172]:
```
f = plt.figure(figsize=(15,8))
bw = ['scott', 'silverman', 0.01, 0.1, 1, 5]


for i, B in enumerate(bw, 1):
    plt.subplot(2,3,i)
    plt.title("Bandwidth: {}".format(B))
    s = movies['IMDB_Rating'].sample(10)
    sns.kdeplot(s, bw=B, kernel = 'gau', legend=False)
    plt.xlim(1,11)
```

```
C:\Users\yyezeret\AppData\Local\Continuum\Anaconda2\envs\dviz\lib\site-packages\scipy\stats\stats.py:1713: FutureWarning: Using a non-tuple sequence for multidimensional indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will be interpreted as an array index, `arr[np.array(seq)]`, which will result either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



Q: What's your takeaway? Explain how bandwidth affects the result of your visualization.

A: Since bandwidth controls the proximity of points, according to https://jakevdp.github.io/blog/2013/12/01/kernel-density-estimation/ (https://jakevdp.github.io/blog/2013/12/01/kernel-density-estimation/) the higher the bandwith the lower the density and vice versa. We can see that bw=5 removes significant number of points and does not reflect real distribution of the underlying data. On the other hand, the bw=0.01 is too dense that results in overfitting. Since the underlying distribution seems to be multumodal the bw=silverman depicts it closer to the real distribution then the bw=scott.

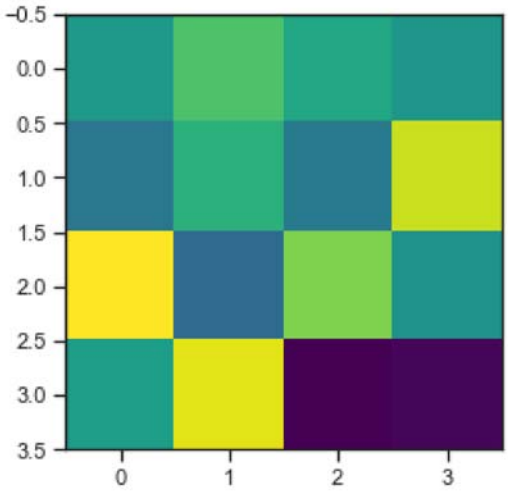
Interpolation

One area where interpolation is used a lot is image processing. Play with it!

https://matplotlib.org/examples/images_contours_and_fields/interpolation_methods.html
(https://matplotlib.org/examples/images_contours_and_fields/interpolation_methods.html)

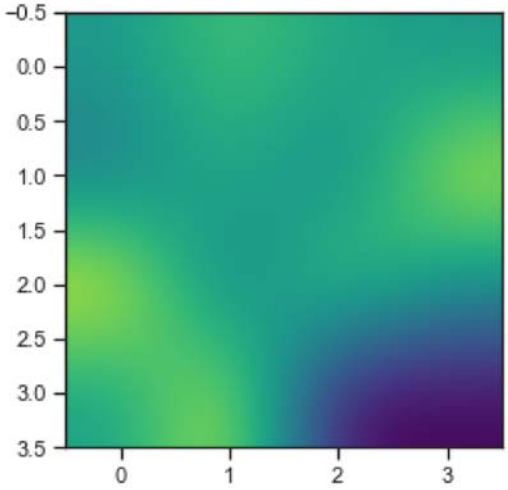
In [173]:
```
methods = [None, 'none', 'nearest', 'bilinear', 'bicubic', 'spline16',
           'spline36', 'hanning', 'hamming', 'hermite', 'kaiser', 'quadric',
           'catrom', 'gaussian', 'bessel', 'mitchell', 'sinc', 'lanczos']
np.random.seed(0)
grid = np.random.rand(4, 4)

plt.imshow(grid, interpolation=None, cmap='viridis')
```

Out[173]:
```
<matplotlib.image.AxesImage at 0x14009bedd68>
```



In [174]:
```
plt.imshow(grid, interpolation='bicubic', cmap='viridis')
```

Out[174]:
```
<matplotlib.image.AxesImage at 0x14009752358>
```



Let's look at some time series data.

In [175]:
```
co2 = vega_datasets.data.co2_concentration()
co2.head()
```

Out[175]:

	Date	CO2
0	1958-03-01	315.70
1	1958-04-01	317.46
2	1958-05-01	317.51
3	1958-07-01	315.86
4	1958-08-01	314.93

In [176]:
```
co2.Date.dtype
```

Out[176]:
```
dtype('O')
```

The `Date` colume is stored as strings. Let's convert it to `datetime` so that we can manipulate.

In [177]:
```
pd.to_datetime(co2.Date).head()
```

Out[177]:
```
0   1958-03-01
1   1958-04-01
2   1958-05-01
3   1958-07-01
4   1958-08-01
Name: Date, dtype: datetime64[ns]
```

In [178]:
```
co2.Date = pd.to_datetime(co2.Date)
```

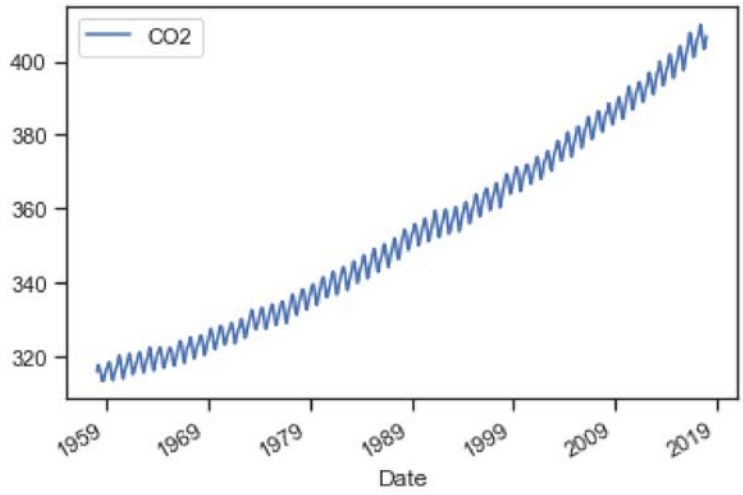
In [179]:
```
co2.set_index('Date', inplace=True)
co2.head()
```

Out[179]:

	CO2
Date	
1958-03-01	315.70
1958-04-01	317.46
1958-05-01	317.51
1958-07-01	315.86
1958-08-01	314.93

In [180]:
```
co2.plot()
```

Out[180]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x14009762160>
```

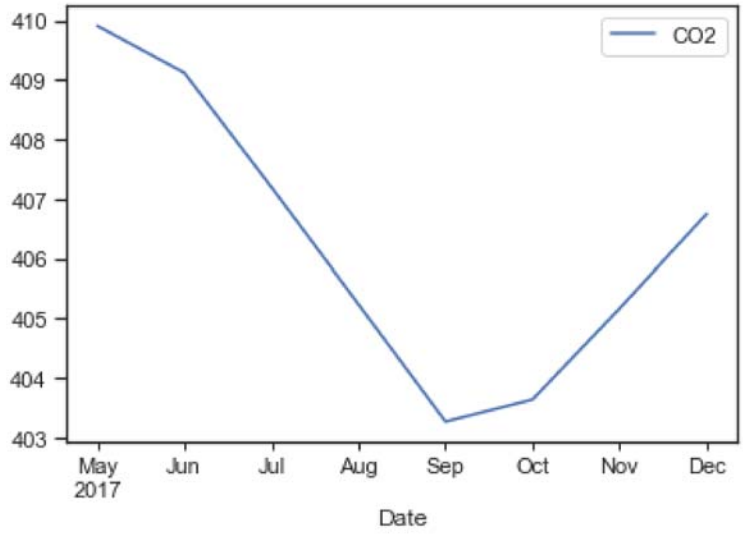


😢

In [181]:
```
recent_co2 = co2.tail(8)
```

In [182]:
```
recent_co2.plot()
```

Out[182]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x140097c0208>
```



This standard line chart above can be considered as a chart with linear interpolation between data points.

The data contains measurements at the resolution of about a month. Let's up-sample the data. This process create new rows that fill the gap between data points. However, because we don't know what to put in the `CO2`, it puts `NaN`.

In [183]:
```
upsampled = recent_co2.resample('D').mean() # D stand for 'byDay'
upsampled
```

Out[183]:

	CO2
Date	
2017-05-01	409.91
2017-05-02	NaN
2017-05-03	NaN
2017-05-04	NaN
2017-05-05	NaN
2017-05-06	NaN
2017-05-07	NaN
2017-05-08	NaN
2017-05-09	NaN
2017-05-10	NaN
2017-05-11	NaN
2017-05-12	NaN
2017-05-13	NaN
2017-05-14	NaN
2017-05-15	NaN
2017-05-16	NaN
2017-05-17	NaN
2017-05-18	NaN
2017-05-19	NaN
2017-05-20	NaN
2017-05-21	NaN
2017-05-22	NaN
2017-05-23	NaN
2017-05-24	NaN
2017-05-25	NaN
2017-05-26	NaN
2017-05-27	NaN
2017-05-28	NaN
2017-05-29	NaN
2017-05-30	NaN
...	...
2017-11-02	NaN
2017-11-03	NaN

	CO2
Date	
2017-11-04	NaN
2017-11-05	NaN
2017-11-06	NaN
2017-11-07	NaN
2017-11-08	NaN
2017-11-09	NaN
2017-11-10	NaN
2017-11-11	NaN
2017-11-12	NaN
2017-11-13	NaN
2017-11-14	NaN
2017-11-15	NaN
2017-11-16	NaN
2017-11-17	NaN
2017-11-18	NaN
2017-11-19	NaN
2017-11-20	NaN
2017-11-21	NaN
2017-11-22	NaN
2017-11-23	NaN
2017-11-24	NaN
2017-11-25	NaN
2017-11-26	NaN
2017-11-27	NaN
2017-11-28	NaN
2017-11-29	NaN
2017-11-30	NaN
2017-12-01	406.75

215 rows × 1 columns

In [198]:
```
upsampled = recent_co2.resample('D').mean()
upsampled.head(10)
```

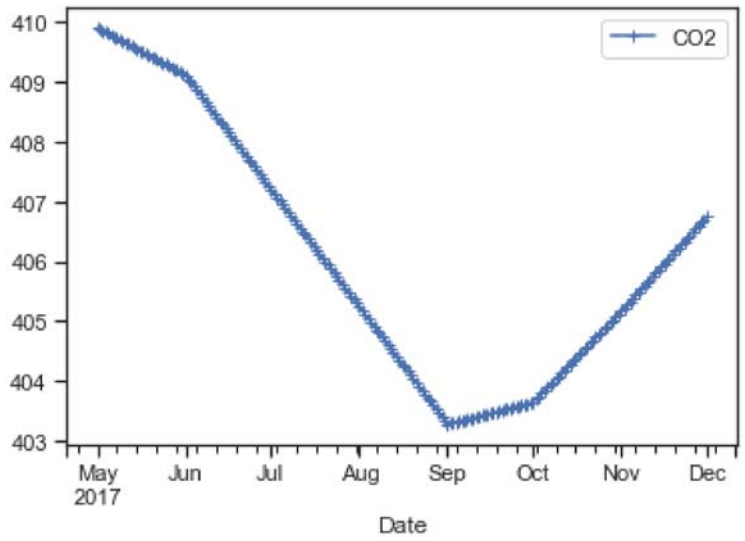
Out[198]:

	CO2
Date	
2017-05-01	409.91
2017-05-02	NaN
2017-05-03	NaN
2017-05-04	NaN
2017-05-05	NaN
2017-05-06	NaN
2017-05-07	NaN
2017-05-08	NaN
2017-05-09	NaN
2017-05-10	NaN

Now we can interpolate and fill the gaps. If we do `linear` interpolation, we get the exactly same plot, but just with more points.

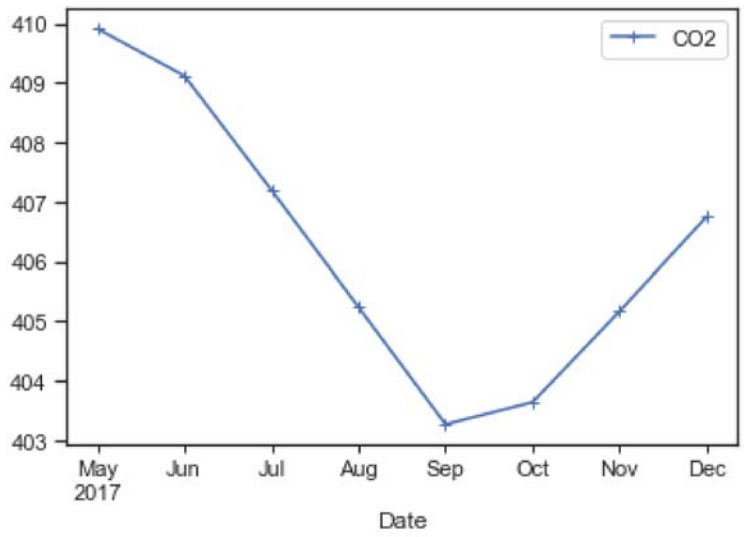
In [185]:
```
recent_co2.resample('D').interpolate(method='linear').plot(style='+-')
```

Out[185]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x140098407f0>
```



In [186]:
```
recent_co2.plot(style='+-')
```

Out[186]: <matplotlib.axes._subplots.AxesSubplot at 0x140098eaeb8>

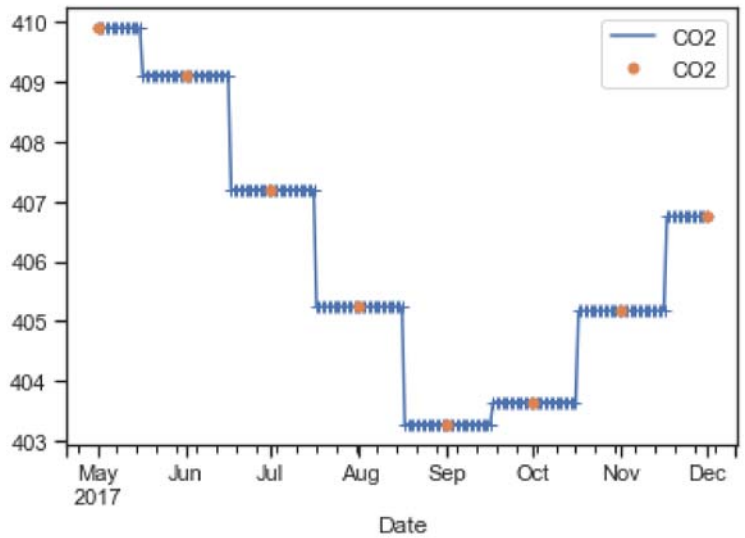


`Nearest` interpolation is just a process of assigning the nearest value to each missing rows.

Q: resample `recent_co2` with 'day' resolution, and then interpolate with `nearest` method, and plot with the `recent_co2`'s actual data points. Useful options are `style='...'`, `ms=...`, and `ax=...`

In [187]:
```
ax = recent_co2.resample('D').interpolate(method='nearest').plot(style='+-') #put first
 plot on ax
recent_co2.plot(style='o', ax=ax, ms=5) #circle: style='o', ms(markersize)
```

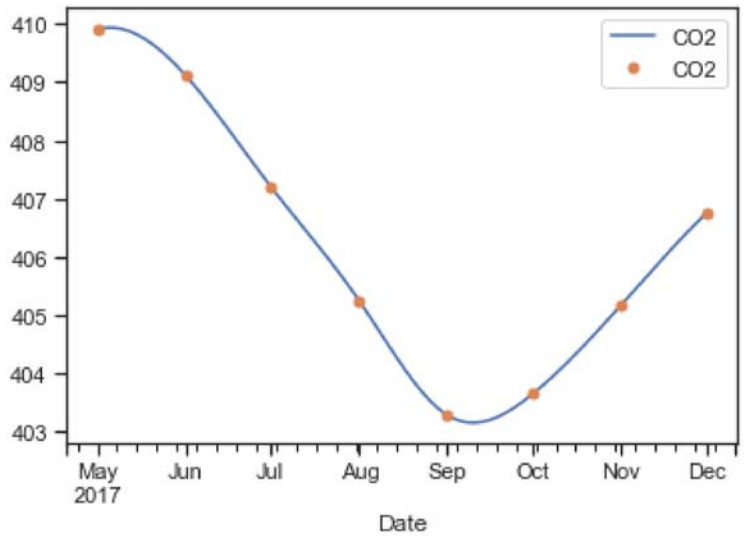
Out[187]: <matplotlib.axes._subplots.AxesSubplot at 0x14009935550>



Q: then let's try `polynomial` interpolation with `order=3`.

In [188]:
```
#put first plot on ax, for pandas method=polinomial need to add order parameter
ax1 = recent_co2.resample('D').interpolate(method='polynomial', order=3).plot()
recent_co2.plot(style='o', ax=ax1, ms=5) #circle: style='o', ms(markersize)
```

Out[188]: <matplotlib.axes._subplots.AxesSubplot at 0x1400931b9e8>



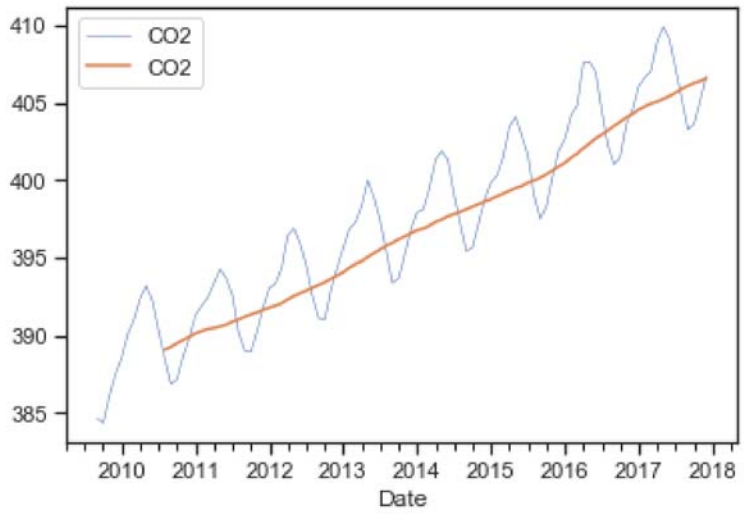
Moving average

Pandas has a nice method called `rolling()`: https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.rolling.html (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.rolling.html)

It lets you do operations on the `rolling` windows. For instance, if you want to calculate the moving average, you can simply

In [189]:
```
ax = co2[-100:].plot(lw=0.5)
co2[-100:].rolling(12).mean().plot(ax=ax)
```

Out[189]: <matplotlib.axes._subplots.AxesSubplot at 0x14009c51e80>

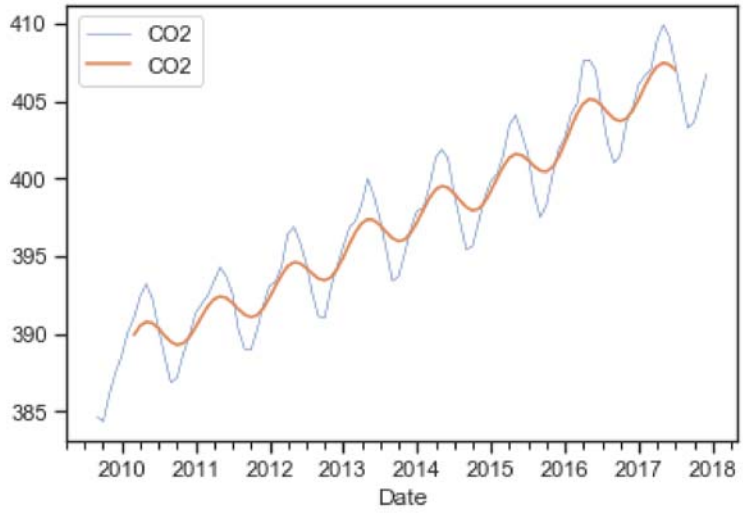


By default, it consider every data point inside each window equally (`win_type=None`) but there are many window types supported by `scipy`. Also by default, the mean value is put at the right end of the window (trailing average).

Q: can you create a plot with `triang` window type and centered average?

In [190]:
```
#*Q: can you create a plot with triang window type and centered average? *
ax = co2[-100:].plot(lw=0.5)
co2[-100:].rolling(12, win_type='triang', center=True).mean().plot(ax=ax)
```

Out[190]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1400a1d3860>
```



Examining relationsips

Remember Anscombe's quartet (https://en.wikipedia.org/wiki/Anscombe%27s_quartet)? Actually, the dataset is not only included in `vega_datasets` but also in `seaborn`.

In [199]:
```
df = sns.load_dataset("anscombe")
df.head(10)
```

Out[199]:

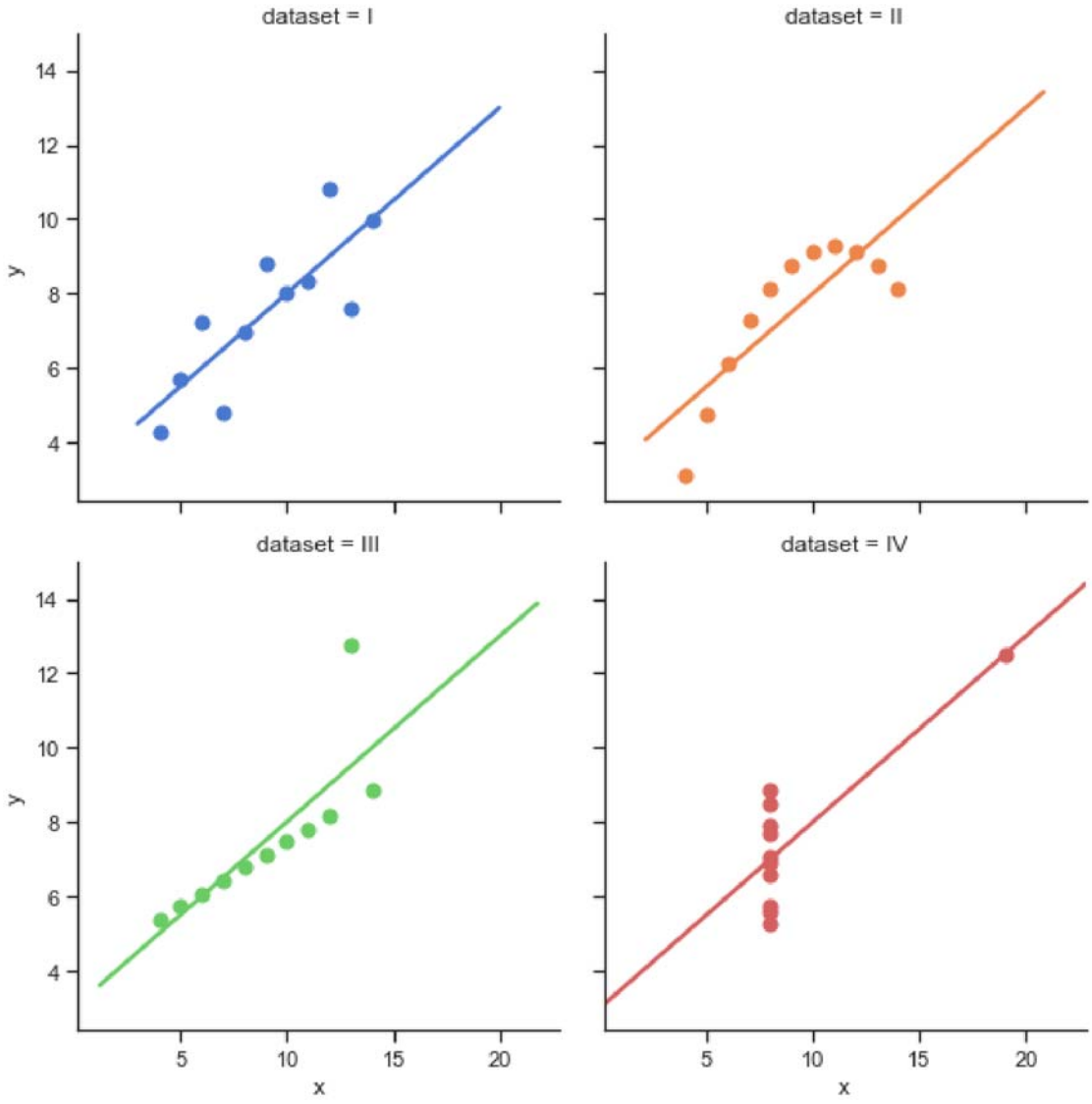
	dataset	x	y
0	I	10.0	8.04
1	I	8.0	6.95
2	I	13.0	7.58
3	I	9.0	8.81
4	I	11.0	8.33
5	I	14.0	9.96
6	I	6.0	7.24
7	I	4.0	4.26
8	I	12.0	10.84
9	I	7.0	4.82

All four datasets are in this single data frame and the 'dataset' indicator is one of the columns. This is a form often called tidy data (http://vita.had.co.nz/papers/tidy-data.pdf), which is easy to manipulate and plot. In tidy data, each row is an observation and columns are the properties of the observation. Seaborn makes use of the tidy form. Using seaborn's `lmplot`, you can very quickly examine relationships between variables, separated by some facets of the dataset.

In [192]:
```
sns.lmplot(x="x", y="y", col="dataset", hue="dataset", data=df,
           col_wrap=2, ci=None, palette="muted", size=4,
           scatter_kws={"s": 50, "alpha": 1})
```

```
C:\Users\yyezeret\AppData\Local\Continuum\Anaconda2\envs\dviz\lib\site-packages\seaborn
\regression.py:546: UserWarning: The `size` paramter has been renamed to `height`; plea
se update your code.
  warnings.warn(msg, UserWarning)
```

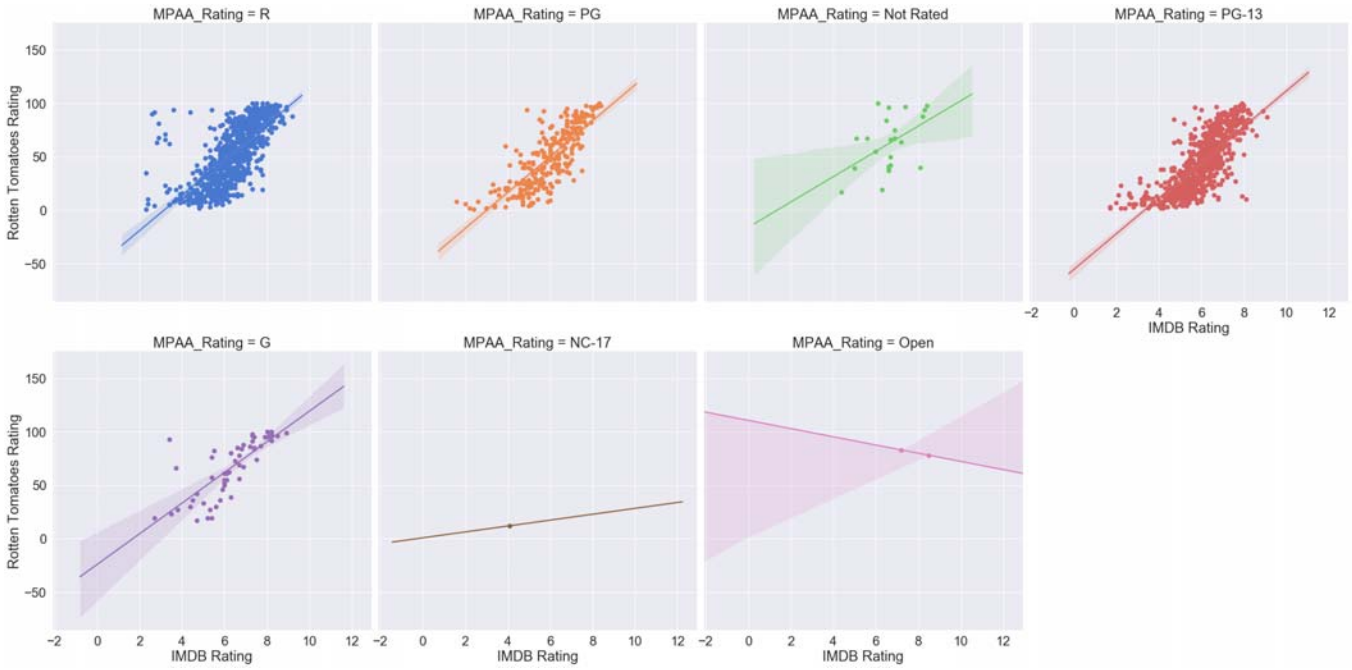
Out[192]: `<seaborn.axisgrid.FacetGrid at 0x14009d5a8d0>`



Q: So let's look at the relationship between `IMDB_Rating` and `Rotten_Tomatoes_Rating` in the `movies` dataset, separated with respect to `MPAA_Rating`. Put 4 plots in a row.

In [193]:
```
sns.set(font_scale=2) # change font size
g=sns.lmplot(x="IMDB_Rating", y="Rotten_Tomatoes_Rating", col="MPAA_Rating", hue="MPAA_R
ating", data=movies,
             col_wrap=4, palette="muted", height=8 ,scatter_kws={"s": 50, "alpha": 1})

g = (g.set_axis_labels("IMDB Rating", "Rotten Tomatoes Rating").fig.subplots_adjust(wspa
ce=.02))
```

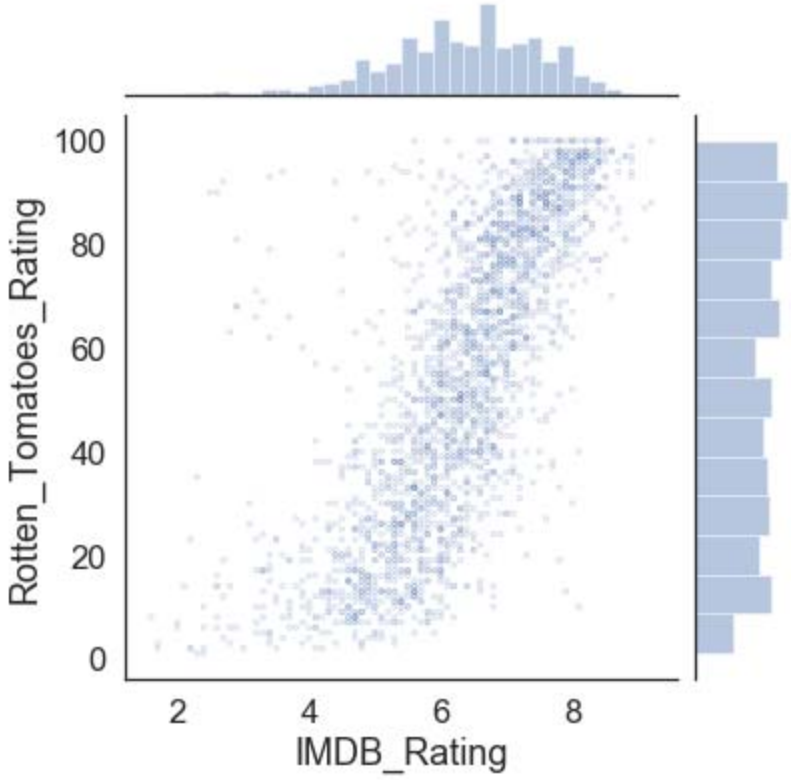
```
C:\Users\yyezeret\AppData\Local\Continuum\Anaconda2\envs\dviz\lib\site-packages\scipy\s
tats\stats.py:1713: FutureWarning: Using a non-tuple sequence for multidimensional inde
xing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this wil
l be interpreted as an array index, `arr[np.array(seq)]`, which will result either in a
n error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



Another useful method for examining relationships is **`jointplot()`** (http://stanford.edu/~mwaskom/software/seaborn/generated/seaborn.jointplot.html), which produces a scatter plot with two marginal histograms.

In [194]:
```
sns.set(font_scale=1.5,style="white", color_codes=True)
g = sns.jointplot(movies['IMDB_Rating'], movies['Rotten_Tomatoes_Rating'], s=5, alpha=0.
2, facecolors='none', edgecolors='b')
```

```
C:\Users\yyezeret\AppData\Local\Continuum\Anaconda2\envs\dviz\lib\site-packages\scipy\s
tats\stats.py:1713: FutureWarning: Using a non-tuple sequence for multidimensional inde
xing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this wil
l be interpreted as an array index, `arr[np.array(seq)]`, which will result either in a
n error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



Hexbin density plot

In 2D, *heatmap* can be considered as a color-based histogram. You divide the space into bins and show the frequency with colors. A common binning method is the hexagonal bin.

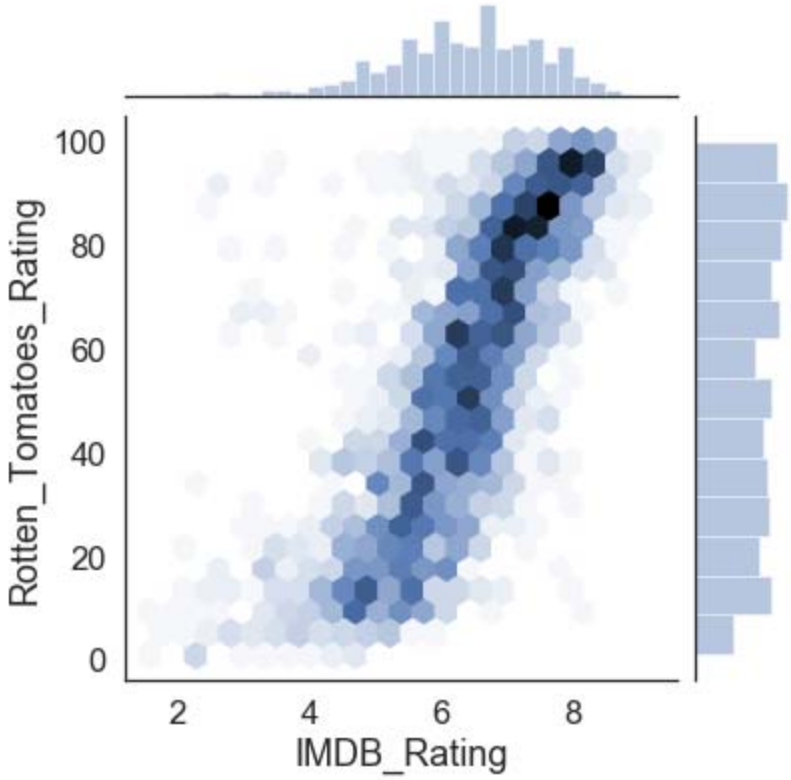
We can again use the **`jointplot()`** (http://stanford.edu/~mwaskom/software/seaborn/generated/seaborn.jointplot.html) and setting the `kind` to be `hexbin`.

Q: Can you create one?

In [195]:
```
sns.jointplot(movies['IMDB_Rating'], movies['Rotten_Tomatoes_Rating'],kind="hexbin")
```

```
C:\Users\yyezeret\AppData\Local\Continuum\Anaconda2\envs\dviz\lib\site-packages\scipy\stats\stats.py:1713: FutureWarning: Using a non-tuple sequence for multidimensional indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will be interpreted as an array index, `arr[np.array(seq)]`, which will result either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

Out[195]:
```
<seaborn.axisgrid.JointGrid at 0x14009944860>
```

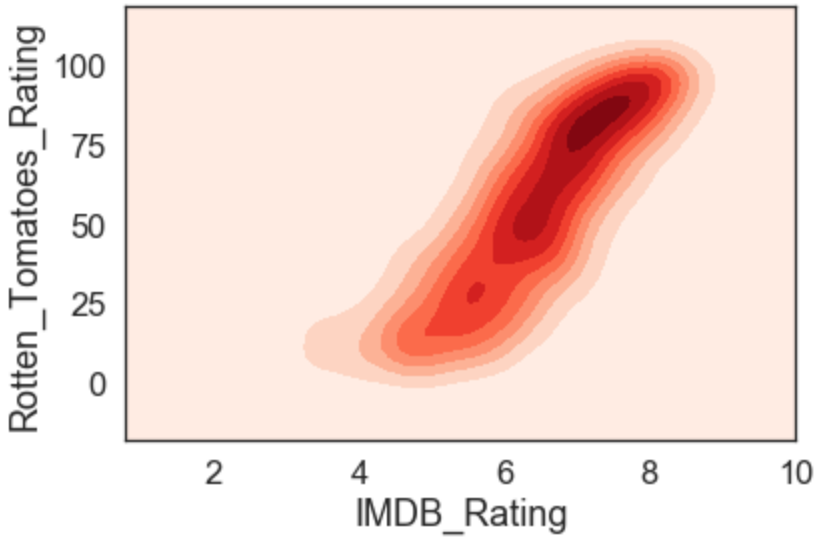


2D KDE We can also do 2D KDE using seaborn's [**`kdeplot()`**] (https://stanford.edu/~mwaskom/software/seaborn/generated/seaborn.kdeplot.html) function. **Q: Can you draw one like this? (this `cmap` is called `Reds`) **

In [196]:
```
sns.kdeplot(movies['IMDB_Rating'], movies['Rotten_Tomatoes_Rating'], cmap='Reds', shade=True)
```

```
C:\Users\yyezeret\AppData\Local\Continuum\Anaconda2\envs\dviz\lib\site-packages\scipy\stats\stats.py:1713: FutureWarning: Using a non-tuple sequence for multidimensional indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will be interpreted as an array index, `arr[np.array(seq)]`, which will result either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

Out[196]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x14005cd6da0>
```



Or again using **`jointplot()`** (http://stanford.edu/~mwaskom/software/seaborn/generated/seaborn.jointplot.html) by setting the `kind` parameter. Look, we also have the 1D marginal KDE plots!

Q: create jointplot with KDE

In [197]:
```
from scipy import stats
sns.set(style="ticks", color_codes=True)
g=sns.jointplot( np.log(movies['IMDB_Votes']),movies['IMDB_Rating'],kind="kde", space=0,
 cmap="Blues")
g = g.annotate(stats.pearsonr,loc="upper right", fontsize=12)
```

```
C:\Users\yyezeret\AppData\Local\Continuum\Anaconda2\envs\dviz\lib\site-packages\scipy\s
tats\stats.py:1713: FutureWarning: Using a non-tuple sequence for multidimensional inde
xing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this wil
l be interpreted as an array index, `arr[np.array(seq)]`, which will result either in a
n error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
C:\Users\yyezeret\AppData\Local\Continuum\Anaconda2\envs\dviz\lib\site-packages\seaborn
\axisgrid.py:1847: UserWarning: JointGrid annotation is deprecated and will be removed
in a future release.
  warnings.warn(UserWarning(msg))
```