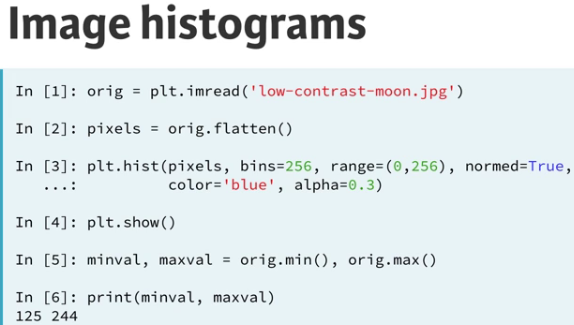
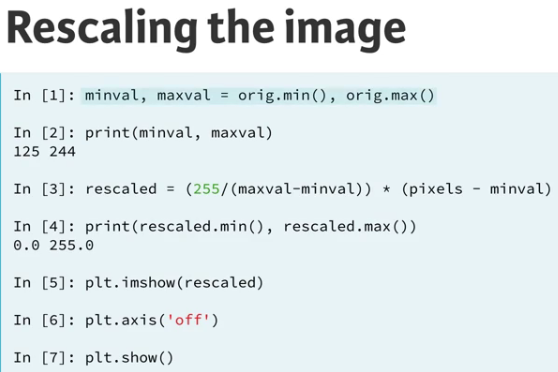


* Alpha makes histogram lines partially transparent.







Flatten the 2-D arrays of the RGB channels into 1-D

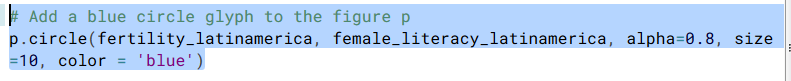
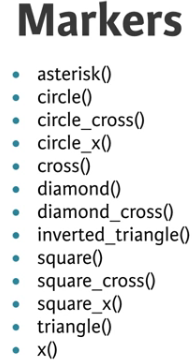
red\_pixels = red.flatten()

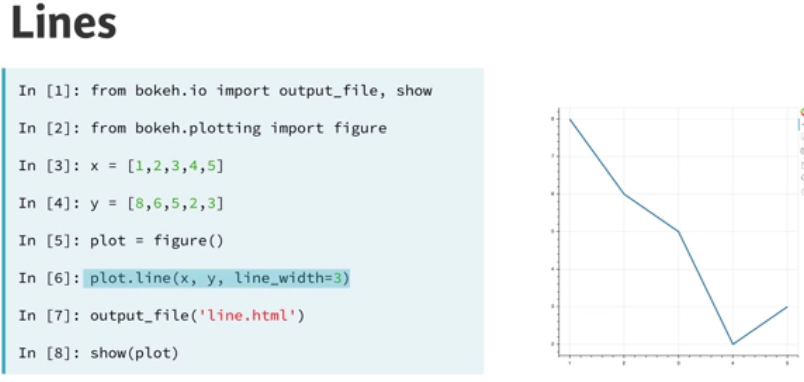
blue\_pixels = blue.flatten()

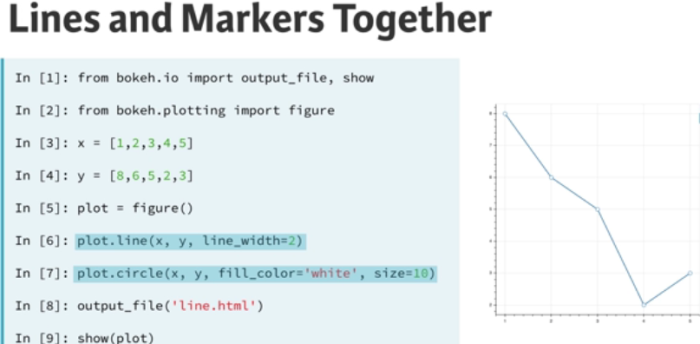
green\_pixels = green.flatten()

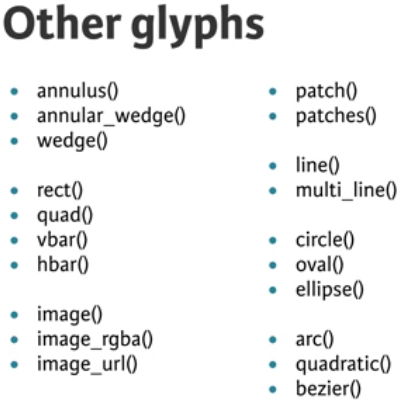
**DATA VISIUALIZATION WITH BOKEH:**

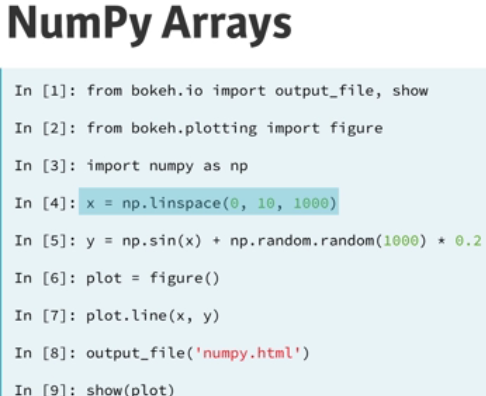


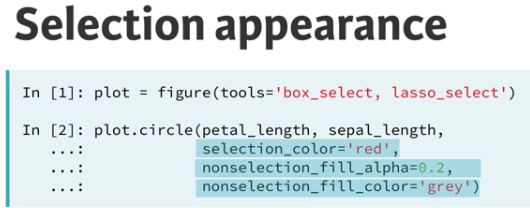
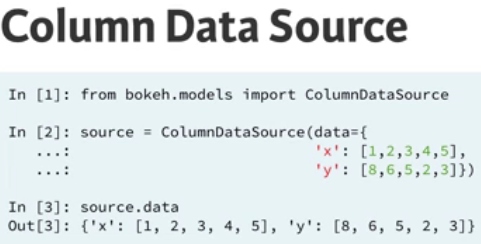


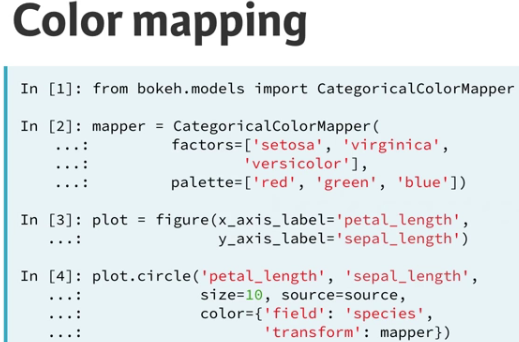
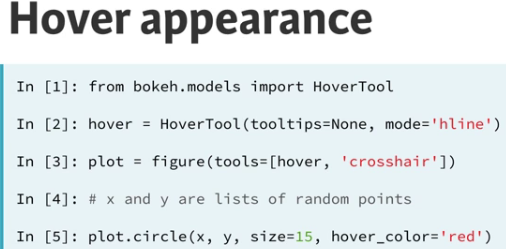


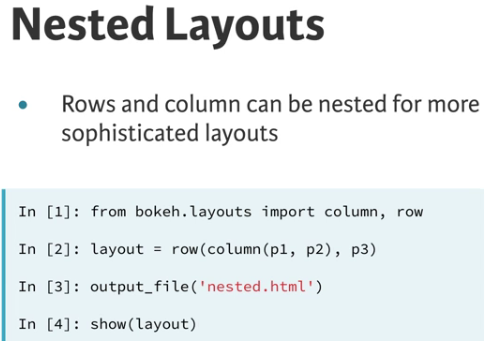
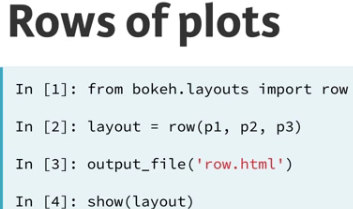












# Import column and row from bokeh.layouts

from bokeh.layouts import row,column

# Make a column layout that will be used as the second row: row2

row2 = column([mpg\_hp, mpg\_weight], sizing\_mode='scale\_width')

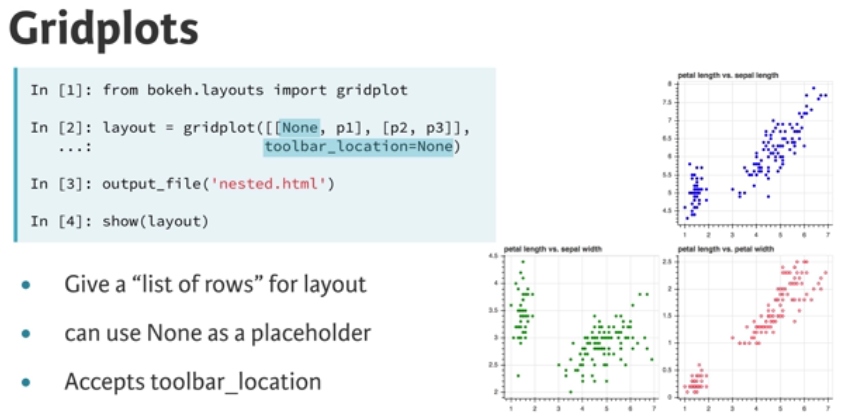
# Make a row layout that includes the above column layout: layout

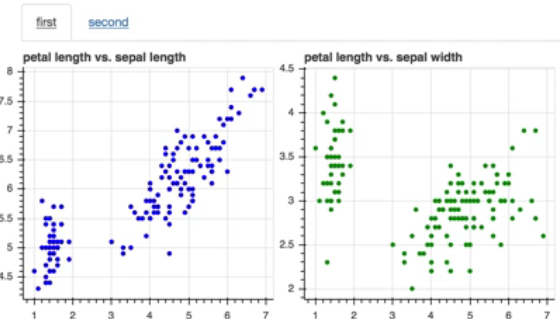
layout = row([avg\_mpg, row2], sizing\_mode='scale\_width')

# Specify the name of the output\_file and show the result

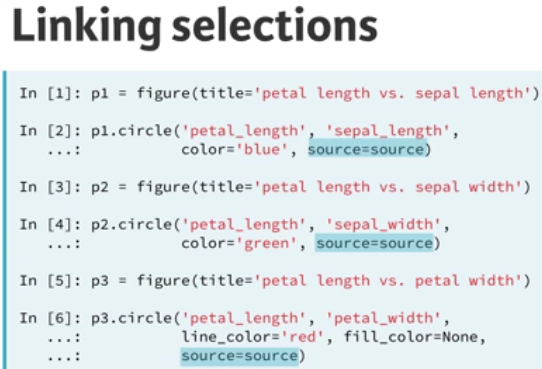
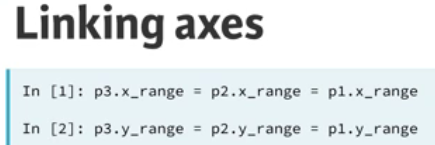
output\_file('layout\_custom.html') show(layout)

**GRIDPLOTS AND TABBED LAYOUTS:**

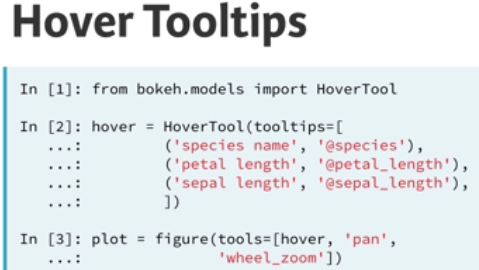
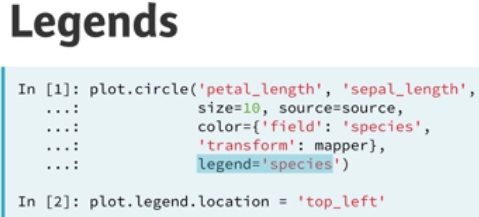




**LINKING:**



**LEGEND:**

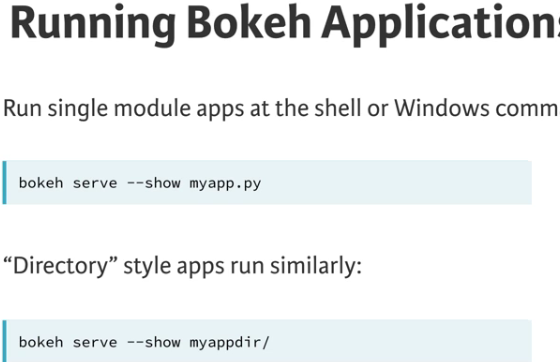
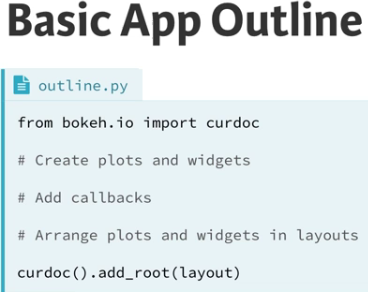
****

* p.legend.background\_fill\_color='lightgray'

# Add the HoverTool object to figure p

* p.add\_tools(hover)

**BASIC APP:**

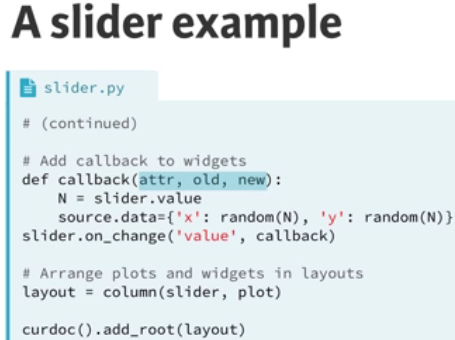
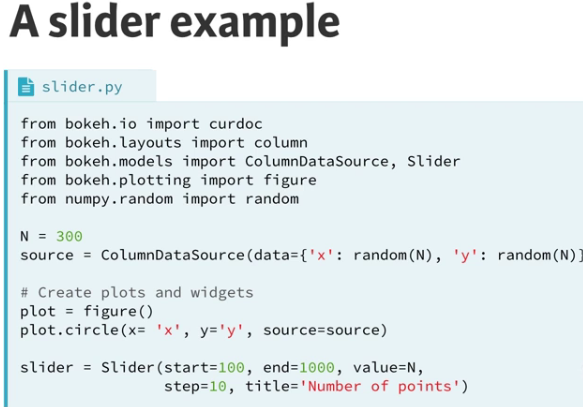
****

# Create a slider: slider

slider = Slider(title='my slider',start=0,end=10,step=0.1,value=2)

# Create a widgetbox layout: layout

layout = widgetbox(slider)



|  |
| --- |
| def callback(attr, old, new): |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Read the current value of the slider: scale |
|  |

|  |
| --- |
| scale = slider.value |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Compute the updated y using np.sin(scale/x): new\_y |
|  |

|  |
| --- |
| new\_y = np.sin(scale/x) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Update source with the new data values |
|  |

|  |
| --- |
| source.data = {'x': x, 'y': new\_y} |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Attach the callback to the 'value' property of slider |
|  |

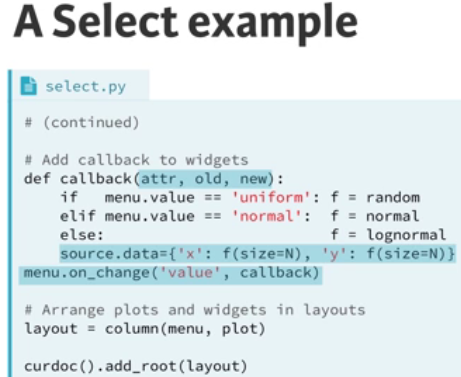
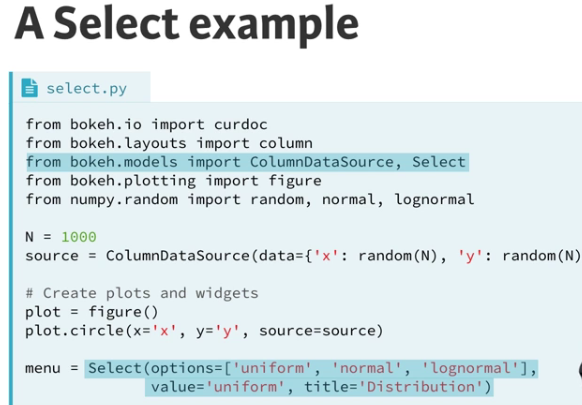
|  |
| --- |
| slider.on\_change('value', callback) |
|  |

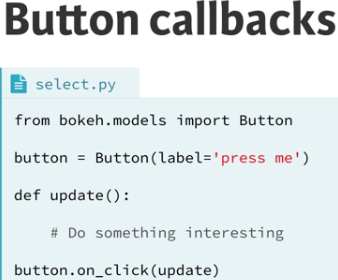
|  |
| --- |
|  |
|  |

|  |
| --- |
| # Create layout and add to current document |
|  |

|  |
| --- |
| layout = column(widgetbox(slider), plot) |
|  |

curdoc().add\_root(layout)





# Add widgetbox(toggle, checkbox, radio) to the current document

curdoc().add\_root(widgetbox(toggle, checkbox, radio))

**STARTING THE APP:**

* **Part1:**

# Import the necessary modules

from bokeh.io import curdoc

from bokeh.models import ColumnDataSource

from bokeh.plotting import figure

# Make the ColumnDataSource: source

source = ColumnDataSource(data={

'x' : data.loc[1970].fertility,

'y' : data.loc[1970].life,

'country' : data.loc[1970].Country,

'pop' : (data.loc[1970].population / 20000000) + 2,

'region' : data.loc[1970].region,

})

# Save the minimum and maximum values of the fertility column: xmin, xmax

xmin, xmax = min(data.fertility), max(data.fertility)

# Save the minimum and maximum values of the life expectancy column: ymin, ymax

ymin, ymax = min(data.life),max(data.life)

# Create the figure: plot

plot = figure(title='Gapminder Data for 1970', plot\_height=400, plot\_width=700,

x\_range=(xmin, xmax),y\_range=(ymin, ymax))

# Add circle glyphs to the plot

plot.circle(x='x', y='y', fill\_alpha=0.8, source=source)

# Set the x-axis label

plot.xaxis.axis\_label ='Fertility (children per woman)'

# Set the y-axis label

plot.yaxis.axis\_label = 'Life Expectancy (years)'

# Add the plot to the current document and add a title

curdoc().add\_root(plot)

curdoc().title = 'Gapminder'

* **Part2:**

|  |
| --- |
| # Make a list of the unique values from the region column: regions\_list |
|  |

|  |
| --- |
| regions\_list = data.region.unique().tolist() |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Import CategoricalColorMapper from bokeh.models and the Spectral6 palette from bokeh.palettes |
|  |

|  |
| --- |
| from bokeh.models import CategoricalColorMapper |
|  |

|  |
| --- |
| from bokeh.palettes import Spectral6 |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Make a color mapper: color\_mapper |
|  |

|  |
| --- |
| color\_mapper = CategoricalColorMapper(factors=regions\_list, palette=Spectral6) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Add the color mapper to the circle glyph |
|  |

|  |
| --- |
| plot.circle(x='x', y='y', fill\_alpha=0.8, source=source, |
|  |

|  |
| --- |
| color=dict(field='region', transform=color\_mapper), legend='region') |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Set the legend.location attribute of the plot to 'top\_right' |
|  |

|  |
| --- |
| plot.legend.location = 'top\_right' |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Add the plot to the current document and add the title |
|  |

|  |
| --- |
| curdoc().add\_root(plot) |
|  |

curdoc().title = 'Gapminder'

* **Part3:**

# Import the necessary modules

from bokeh.layouts import widgetbox,row

from bokeh.models import Slider

# Define the callback function: update\_plot

def update\_plot(attr,old,new):

# Set the yr name to slider.value and new\_data to source.data

yr = slider.value

new\_data = {

'x' : data.loc[yr].fertility,

'y' : data.loc[yr].life,

'country' : data.loc[yr].Country,

'pop' : (data.loc[yr].population / 20000000) + 2,

'region' : data.loc[yr].region,

}

source.data = new\_data

# Make a slider object: slider

slider = Slider(title='Year',start=1970,end=2010,step=1,value=1970)

# Attach the callback to the 'value' property of slider

slider.on\_change('value',update\_plot)

# Make a row layout of widgetbox(slider) and plot and add it to the current document

layout = row(widgetbox(slider), plot)

curdoc().add\_root(layout)

* **Part4:**

|  |
| --- |
| def update\_plot(attr, old, new): |
|  |

|  |
| --- |
| # Assign the value of the slider: yr |
|  |

|  |
| --- |
| yr = slider.value |
|  |

|  |
| --- |
| # Set new\_data |
|  |

|  |
| --- |
| new\_data = { |
|  |

|  |
| --- |
| 'x' : data.loc[yr].fertility, |
|  |

|  |
| --- |
| 'y' : data.loc[yr].life, |
|  |

|  |
| --- |
| 'country' : data.loc[yr].Country, |
|  |

|  |
| --- |
| 'pop' : (data.loc[yr].population / 20000000) + 2, |
|  |

|  |
| --- |
| 'region' : data.loc[yr].region, |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| # Assign new\_data to: source.data |
|  |

|  |
| --- |
| source.data = new\_data |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Add title to figure: plot.title.text |
|  |

|  |
| --- |
| plot.title.text = 'Gapminder data for %d' % yr |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Make a slider object: slider |
|  |

|  |
| --- |
| slider = Slider(start=1970, end=2010, step=1, value=1970, title='Year') |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Attach the callback to the 'value' property of slider |
|  |

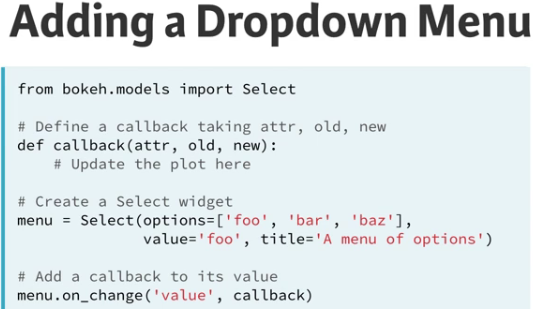
|  |
| --- |
| slider.on\_change('value', update\_plot) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Make a row layout of widgetbox(slider) and plot and add it to the current document |
|  |

|  |
| --- |
| layout = row(widgetbox(slider), plot) |
|  |

curdoc().add\_root(layout)



|  |
| --- |
| def update\_plot(attr, old, new): |
|  |

|  |
| --- |
| # Read the current value off the slider and 2 dropdowns: yr, x, y |
|  |

|  |
| --- |
| yr = slider.value |
|  |

|  |
| --- |
| x = x\_select.value |
|  |

|  |
| --- |
| y = y\_select.value |
|  |

|  |
| --- |
| # Label axes of plot |
|  |

|  |
| --- |
| plot.xaxis.axis\_label = x |
|  |

|  |
| --- |
| plot.yaxis.axis\_label = y |
|  |

|  |
| --- |
| # Set new\_data |
|  |

|  |
| --- |
| new\_data = { |
|  |

|  |
| --- |
| 'x' : data.loc[yr][x], |
|  |

|  |
| --- |
| 'y' : data.loc[yr][y], |
|  |

|  |
| --- |
| 'country' : data.loc[yr].Country, |
|  |

|  |
| --- |
| 'pop' : (data.loc[yr].population / 20000000) + 2, |
|  |

|  |
| --- |
| 'region' : data.loc[yr].region, |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| # Assign new\_data to source.data |
|  |

|  |
| --- |
| source.data = new\_data |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Set the range of all axes |
|  |

|  |
| --- |
| plot.x\_range.start = min(data[x]) |
|  |

|  |
| --- |
| plot.x\_range.end = max(data[x]) |
|  |

|  |
| --- |
| plot.y\_range.start = min(data[y]) |
|  |

|  |
| --- |
| plot.y\_range.end = max(data[y]) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Add title to plot |
|  |

|  |
| --- |
| plot.title.text = 'Gapminder data for %d' % yr |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Create a dropdown slider widget: slider |
|  |

|  |
| --- |
| slider = Slider(start=1970, end=2010, step=1, value=1970, title='Year') |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Attach the callback to the 'value' property of slider |
|  |

|  |
| --- |
| slider.on\_change('value', update\_plot) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Create a dropdown Select widget for the x data: x\_select |
|  |

|  |
| --- |
| x\_select = Select( |
|  |

|  |
| --- |
| options=['fertility', 'life', 'child\_mortality', 'gdp'], |
|  |

|  |
| --- |
| value='fertility', |
|  |

|  |
| --- |
| title='x-axis data' |
|  |

|  |
| --- |
| ) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Attach the update\_plot callback to the 'value' property of x\_select |
|  |

|  |
| --- |
| x\_select.on\_change('value', update\_plot) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Create a dropdown Select widget for the y data: y\_select |
|  |

|  |
| --- |
| y\_select = Select( |
|  |

|  |
| --- |
| options=['fertility', 'life', 'child\_mortality', 'gdp'], |
|  |

|  |
| --- |
| value='life', |
|  |

|  |
| --- |
| title='y-axis data' |
|  |

|  |
| --- |
| ) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Attach the update\_plot callback to the 'value' property of y\_select |
|  |

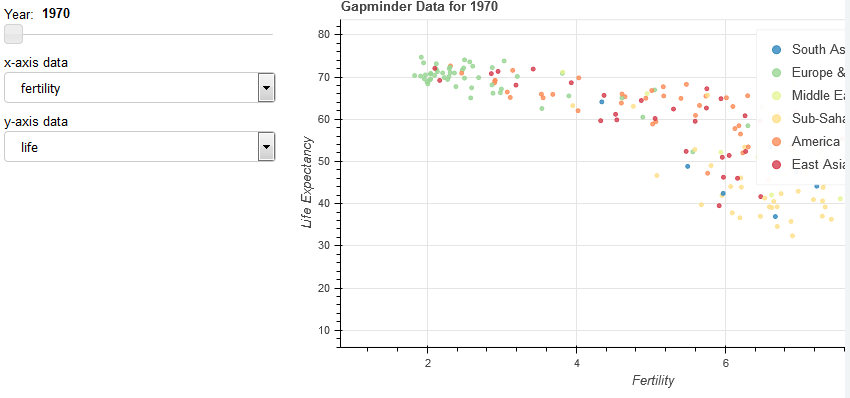
|  |
| --- |
| y\_select.on\_change('value', update\_plot) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Create layout and add to current document |
|  |

|  |
| --- |
| layout = row(widgetbox(slider, x\_select, y\_select), plot) |
|  |

curdoc().add\_root(layout)



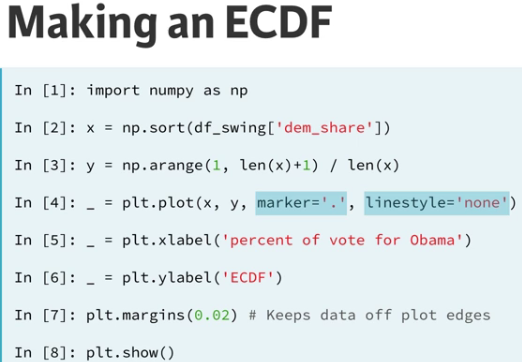
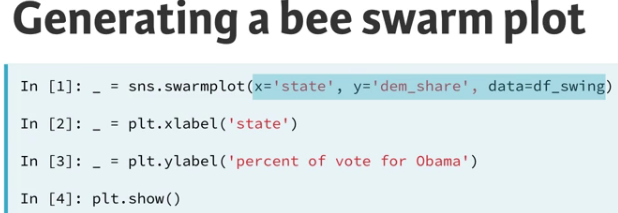
**STATICAL THINKING IN PYTHON:**

* Exploratory Data Analysis (EDA)

Setting bins of a histogram:

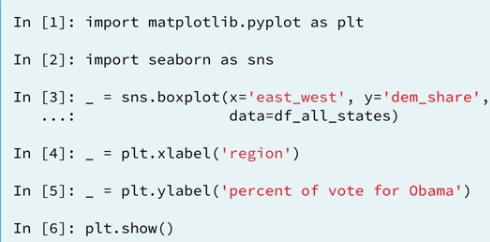
Bin\_ed =[0,10,20,30,40,50,60,70,80]

\_ = plt.hist(df\_S[‘s’] , bins = Bin\_ed # we can set bins as bins = 20



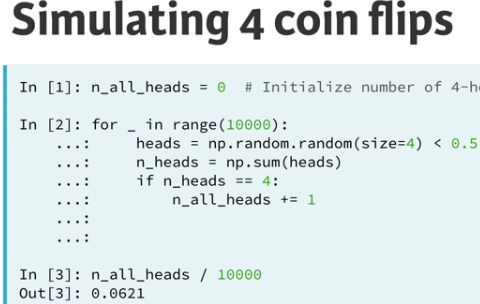
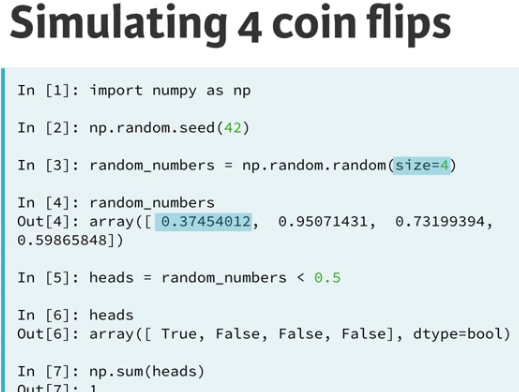
* An outlier can affect the value of mean but not median.

**BOXPLOT:**

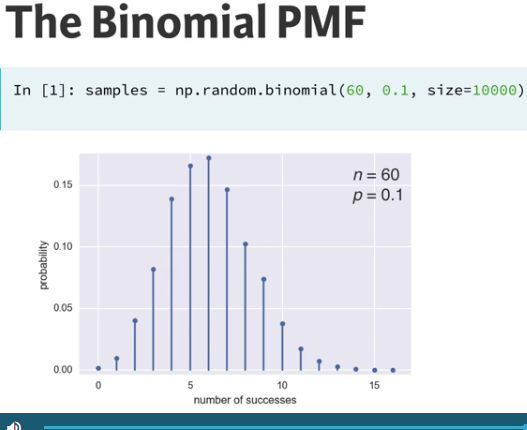


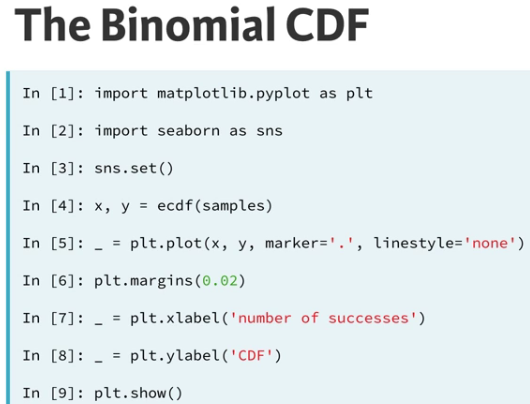
**PERCENTILE:**

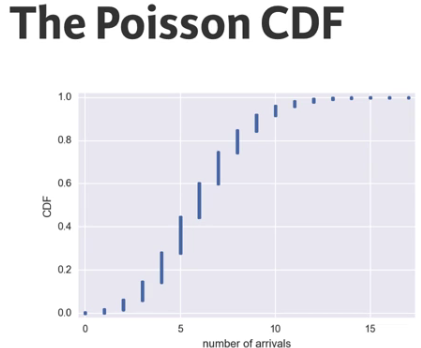
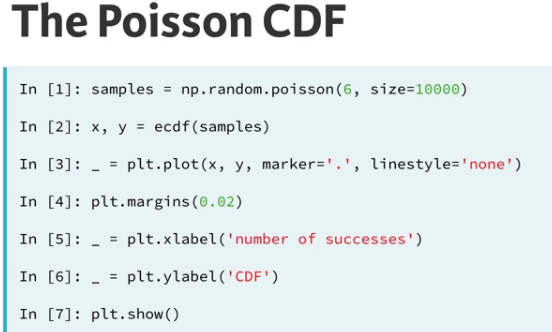
* ptiles\_vers = np.percentile(versicolor\_petal\_length,percentiles)
* np.sqrt( np.var(--)) #variance and its square
* np.std(--) #Standard deviation
* np.cov(--) #Co-variance

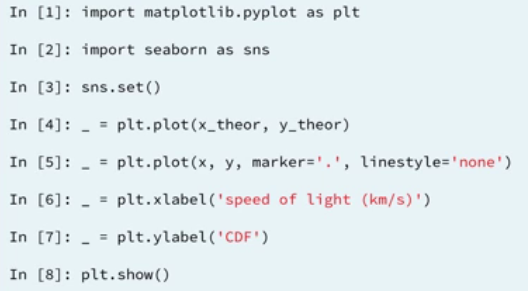
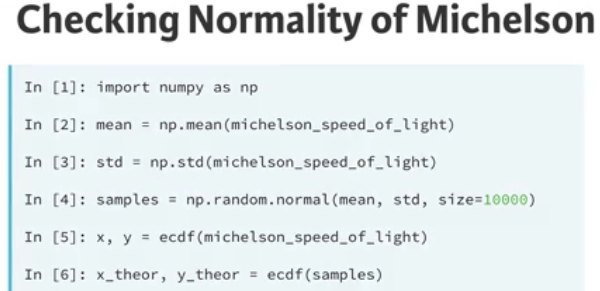


* Initialize an empty array, random\_numbers, of 100,000 entries to store the random numbers. Make sure you use np.empty(100000) to do this.
* random\_numbers = np.empty(100000)

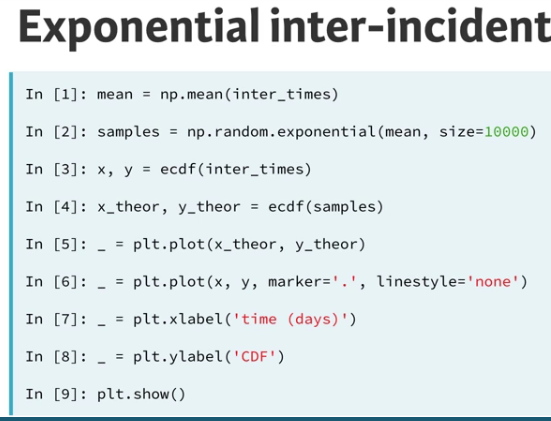




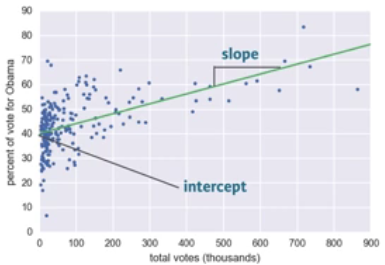
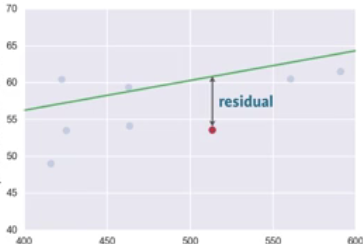




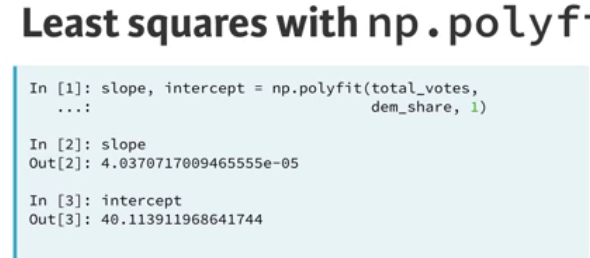
* Draw 100,000 samples from a Normal distribution that has a mean of 20 and a standard deviation of 1:
* samples\_std1=np.random.normal(20,1,size=100000)



**LINEAR REGRESSION:**

**** 

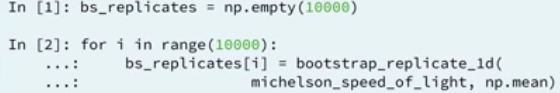
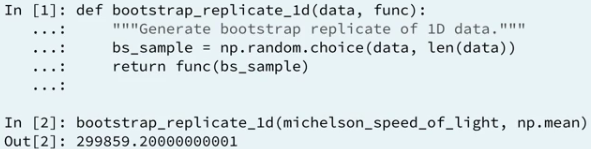
* Intercept is where line crosses y-axis.
* Vertical distance between point and line is residual.
* If data point lie below the line than residual is negative.
* Least Squares – The process of finding the parameters for which the sum of the squares of residuals is minimal.
* 1st and 2nd parameter are x and y values and the 3rd one is degree of polynomial, for linear function its 1.



**BOOTSTRAPPING:**

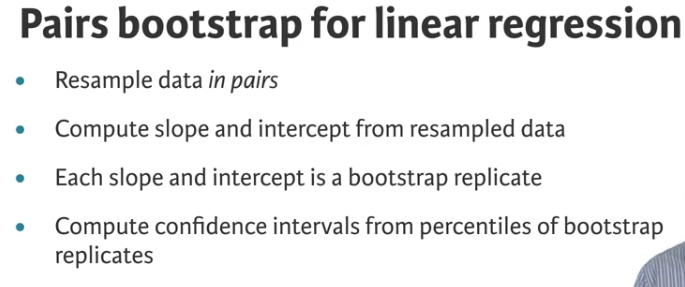
* The use of resampled data to perform statistical inference.

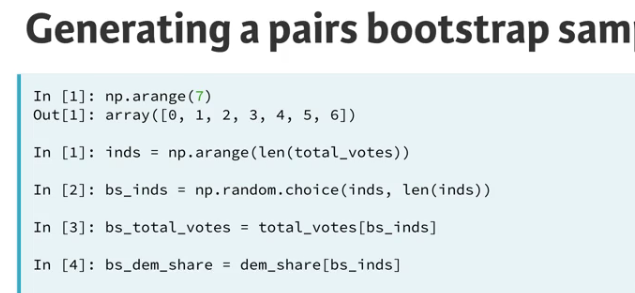
**BOOTSTRAP REPLICATE FUNCTION: MANY BOOTSTARP REPLICATE:**

****

**BOOTSTRAP CONFIDENCE INTERVAL:**

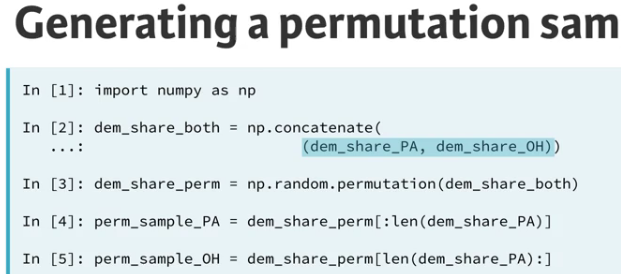
****



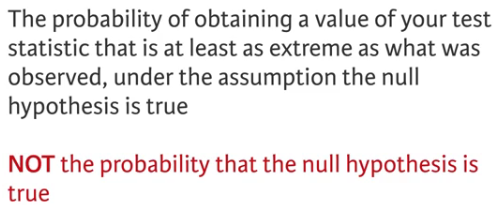


**PERMUTATION:**

* Random reordering of entries in an array

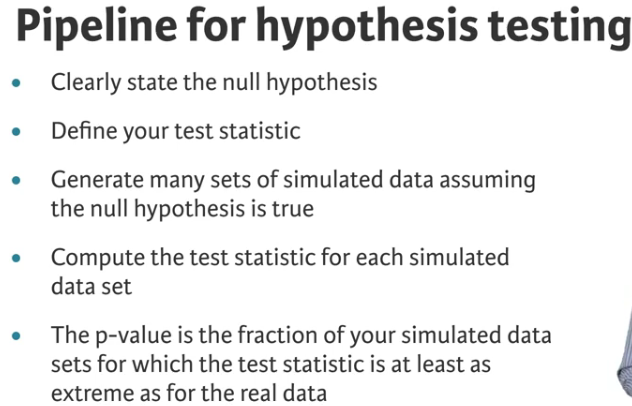
****

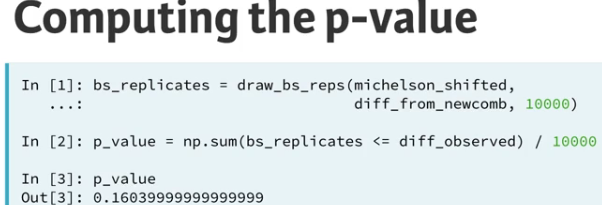
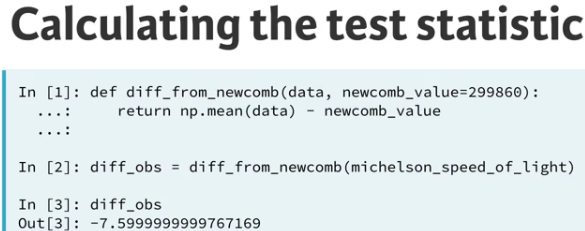
**P-value:**

****

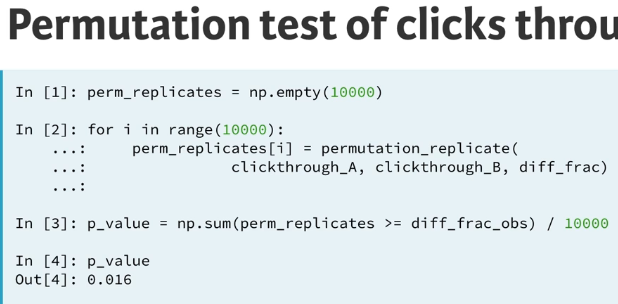
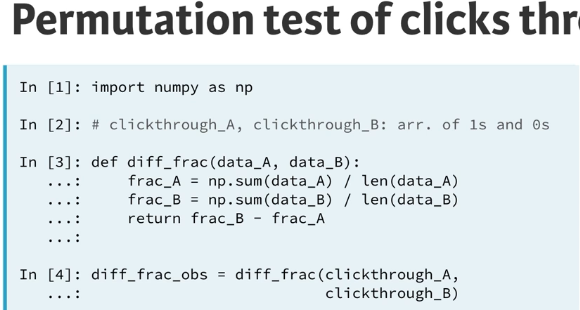
**Statistical Significance:** Determined by the smallness of a p-value.

**NHST:** Null Hypothesis significance testing



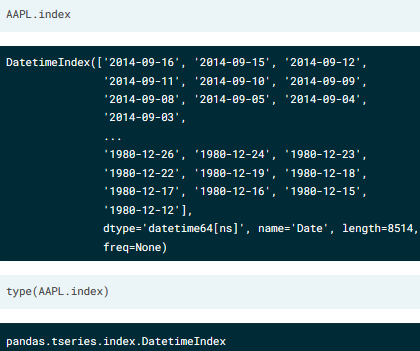
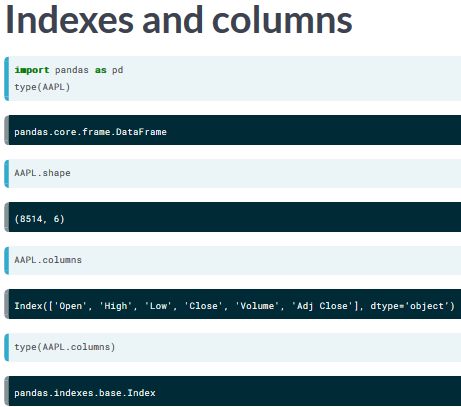


**One sample test**: Compare one set of data to a single number.



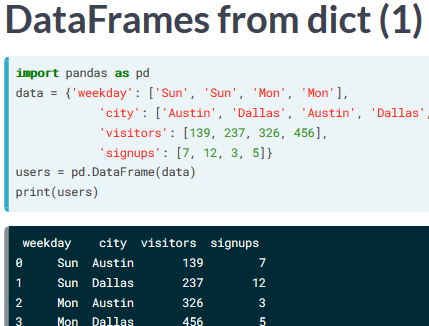
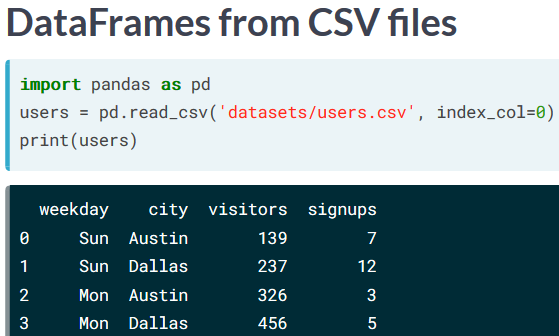
* Construct an array - dems has 153 True entries and 91 False entries.
* dems = np.array([True] \* 153 + [False] \* 91)

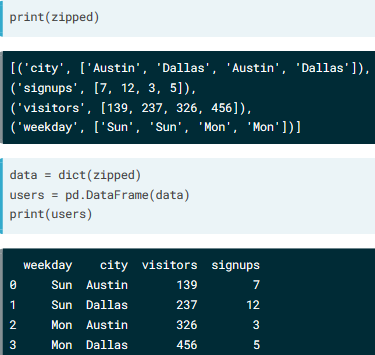
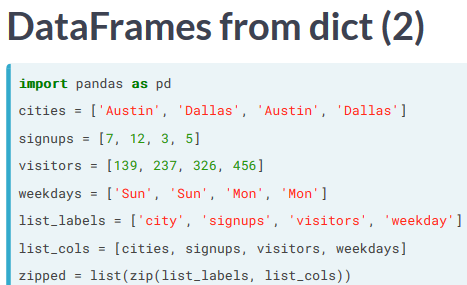
**PANDAS:**

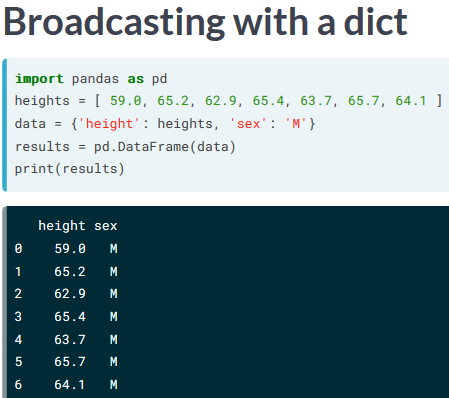
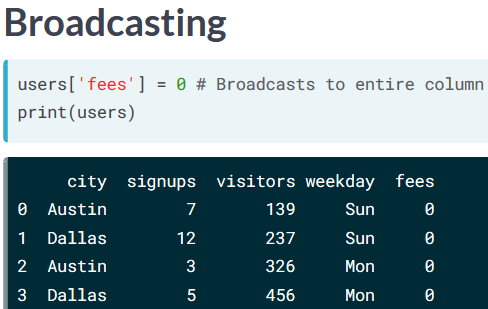


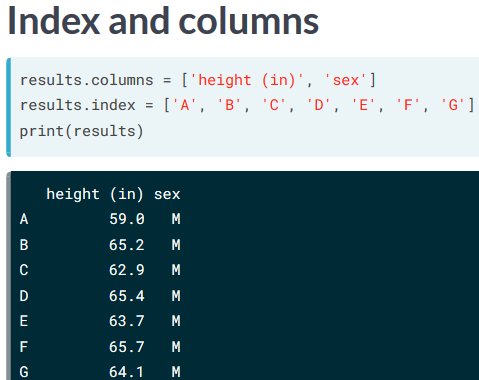


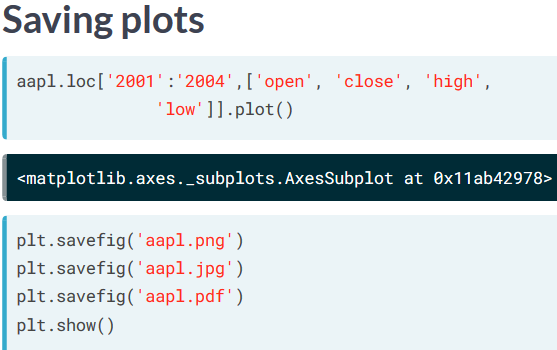
* # Create array of DataFrame values: np\_vals
* np\_vals = df.values











# Plot all columns as subplots

* df.plot(subplots=True)
* plt.show()

# Plot just the Dew Point data

* column\_list1 = ['Dew Point (deg F)']
* df[column\_list1].plot()

