CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

Analytical Programming

Analytical Programming

Lecture: Visualizaiton

Ruairí O'Reilly

Introduction to MatPlotLib

- Matplotlib is a **library** for making **graphs** of arrays in Python.
  - Although matplotlib is written primarily in pure Python, it makes heavy use of NumPy and other extension code to provide good performance even for large arrays.
  - http://matplotlib.org/
- ***matplotlib.pyplot*** is a collection of command style functions.
  - Each pyplot function makes some changes to a figure:
    - Create a figure
    - Create a plotting area in a figure
    - Plot some lines in a plotting area
    - Format the plot with labels
  - http://matplotlib.org/api/pyplot_api.html

Using Pandas with Mat

- Pandas offers visualization capabilities and is tightly integrated with Matplotlib.

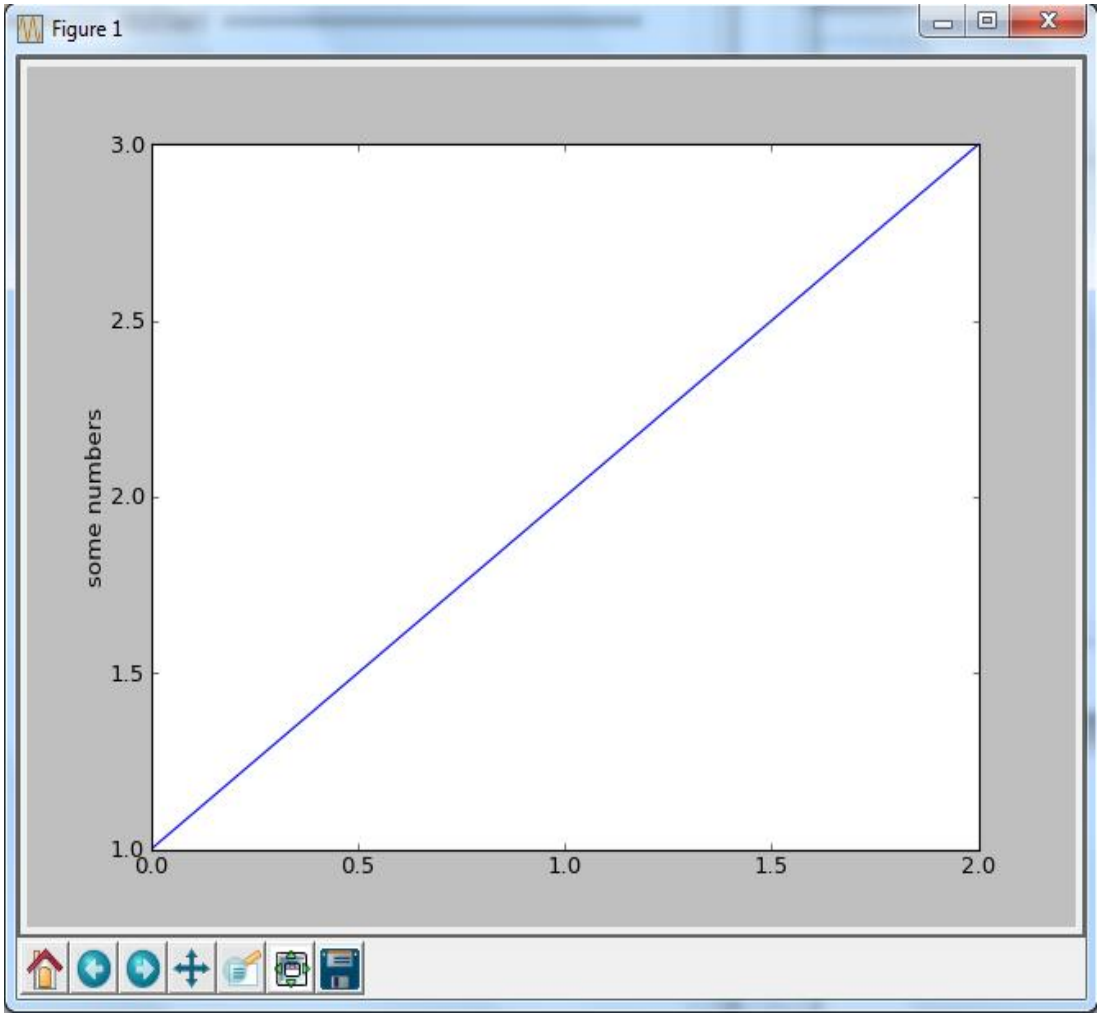
- In the following slides we will show how to generate graphs using both MatPlotLib directly and using Pandas.

Example

```
import matplotlib.pyplot as plt

plt.plot([1,2,3])

plt.show()
```

If the plot method only receives a single list to be plotted then it assumes it is to be plotted on the y axis

The graphical representation is displayed by **show()** function.



Notice the x-axis ranges from 0-2 and the y-axis from 1-3.
If you provide a single list or array to the plot() command, matplotlib assumes it is a sequence of y values, and automatically generates the x values for you.

Since python ranges start with 0, the default x vector has the same length as y but starts with 0. Hence the x data is [0,1,2].

Plot Command

- plot() is a versatile command, and is capable of accepting an **arbitrary number of arguments**.
- For example, to plot x versus y, you can issue the command:
- **plt.plot([1,2,3,4], [1,4,9,16])**
  - The first list is the x coordinates and the second list are the y coordinates

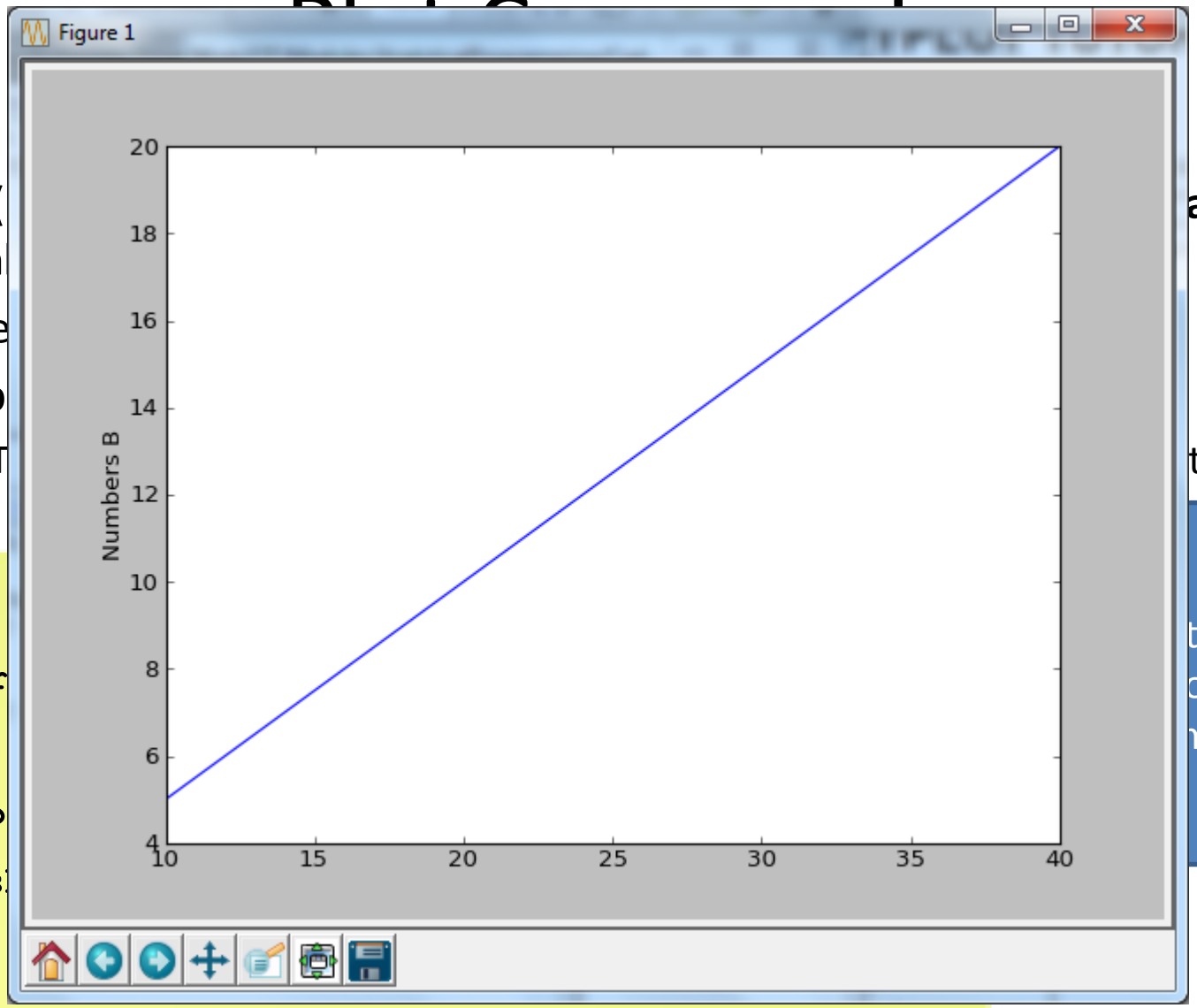
```
import matplotlib.pyplot as plt

plt.plot([10, 20, 30, 40], [5, 10, 15, 20])
plt.show()
```

Plot function uses first list as x coordinates to be mapped

- plot(ary
  num
- For e
- **plt.p**
  - T tes

```
impor

plt.p
plt.s
```

tion uses first
oordinates to
mapped

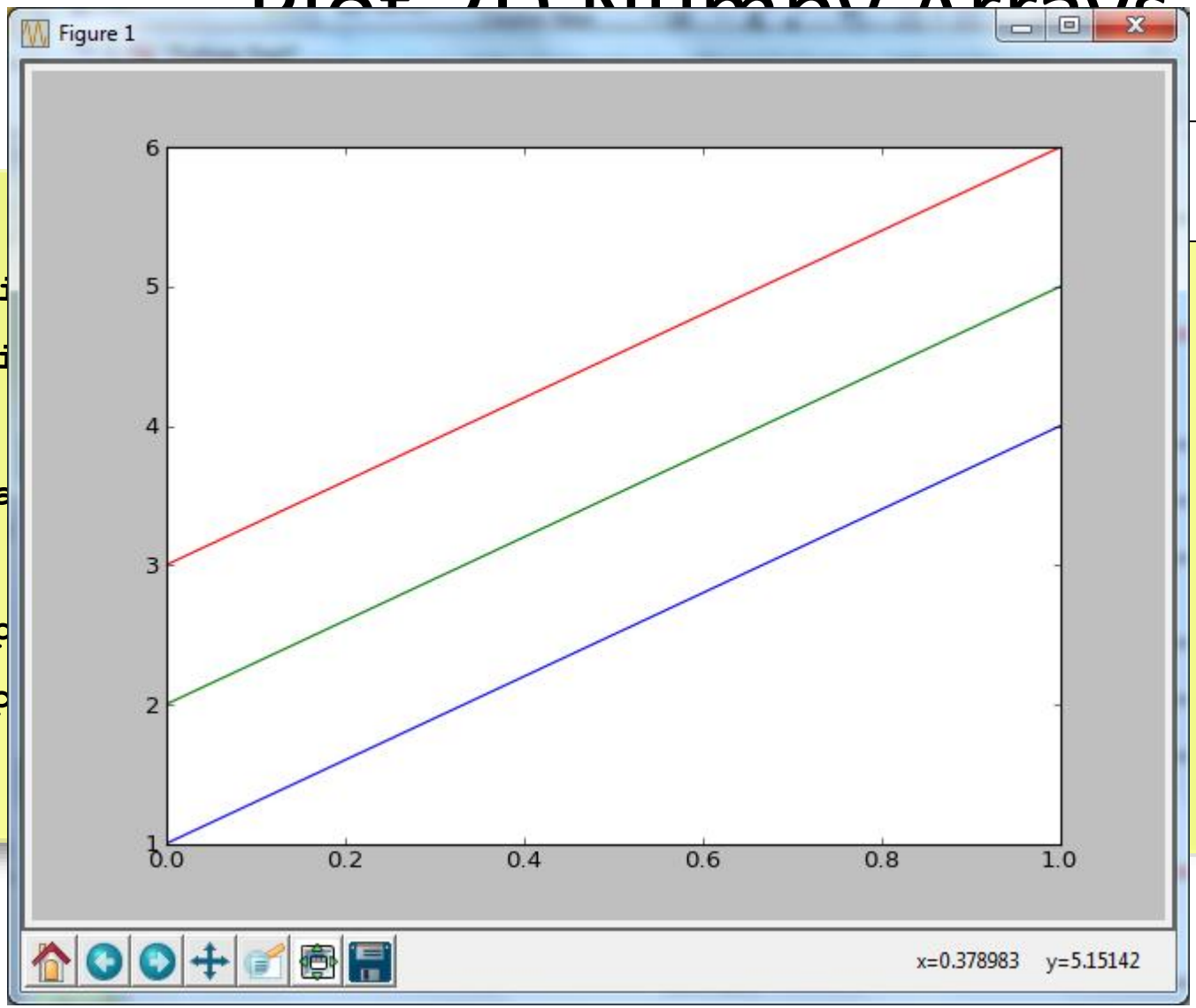


Plot 2D Numpy Arrays

Each **column** in the 2D array is **treated as a separate line**

```
import matplotlib.pyplot as plt
import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6]], float)

plt.plot(arr)
plt.show()
```

Plot 2D Numpy Arrays



is **treated as a**

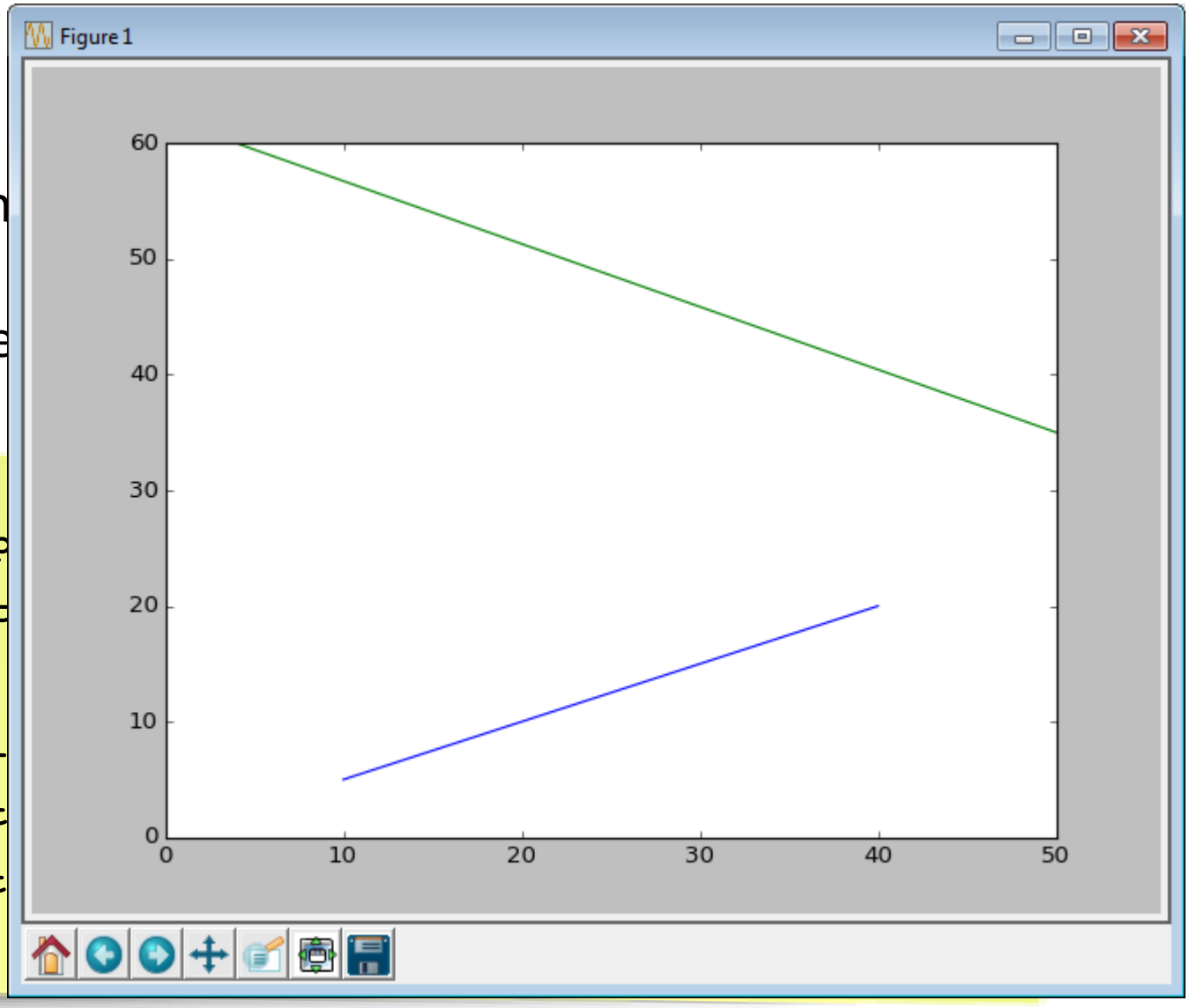
Adding Lines to a Plot

- You can incrementally add additional elements to a plot by using the plot function. Here for example we add a second line to the plot.

```
# plots one line
plt.plot([10, 20, 30, 40], [5, 10, 15, 20])

#plots a second line
plt.plot([4, 50], [60, 35])
plt.show()
```

- You can in
using the
line to the

```
# p
plt

#pl
plt
plt
```



Plotting using a Series and DataFrame

- Visualizing a data in pandas is as simple as calling .plot() on a DataFrame or Series object.

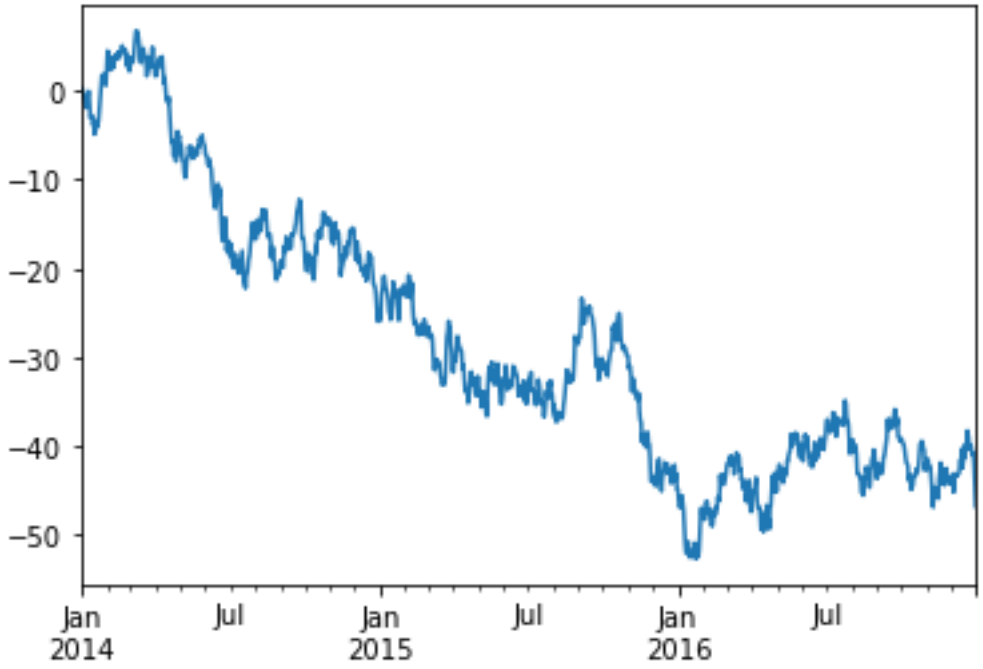
- In the example on the next slide we create a **Series object** that contains some time series data and we then plot it using the plot command.

- Notice, that the:
  - **Index** of the Series becomes the label for the **X axis**
  - **Values** of the Series becomes the **Y values**.

Plotting using a Series

```
import numpy as np
import pandas as pd

# generate a random walk time-series
np.random.seed(19)
s = pd.Series(np.random.randn(1096),
              index=pd.date_range('2014-01-01',
                                  '2016-12-31'))
# the cumsum function will return a Series contains the cumulative sum
# of all values in s
walk_ts = s.cumsum()

walk_ts.plot();
```

Plotting usi




```
import numpy as np
import pandas as pd

# generate a random walk time-series
np.random.seed(19)
s = pd.Series(np.random.randn(1096),
              index=pd.date_range('2014-01-01',
                                  '2016-12-31'))
# the cumsum function will return a Series contains the cumulative sum
# of all values in s
walk_ts = s.cumsum()

walk_ts.plot();
```

Plotting using a DataFrame

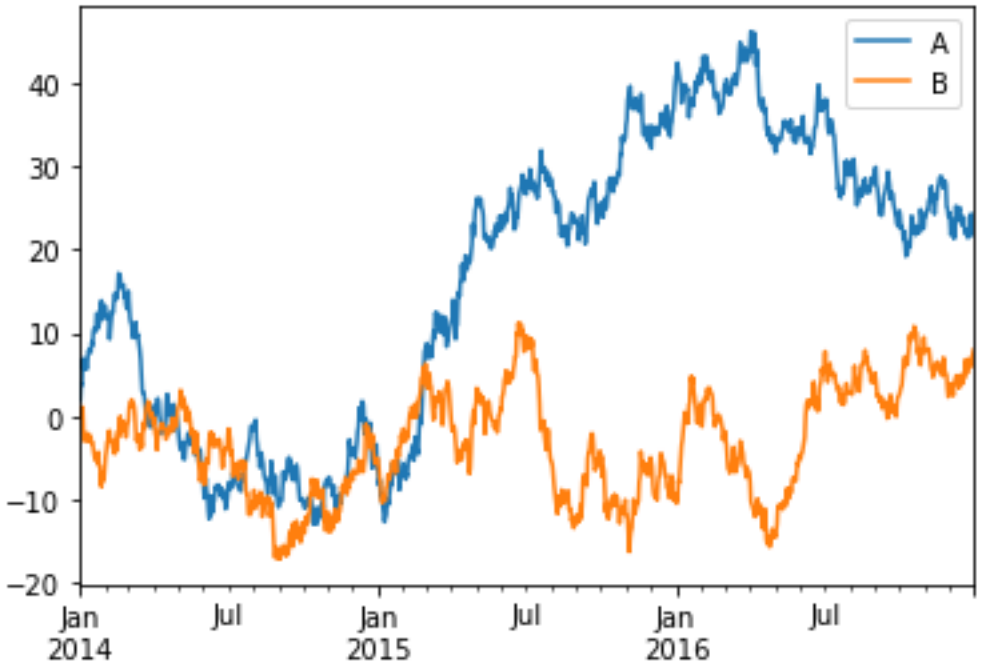
- In the code below we create a DataFrame

```
df = pd.DataFrame(np.random.randn(1096, 2),
                  index=pd.date_range('2014-01-01',
                                      '2014-16-31'),columns=list('AB'))
walk_df = df.cumsum()

walk_df.head()
```

	A	B
2014-01-01	-0.378286	0.881338
2014-01-02	1.489231	1.170020
2014-01-03	1.590698	0.194133
2014-01-04	1.842982	-0.507354
2014-01-05	2.599857	-1.376844

Plotting usi

- In the code below w




```
df = pd.DataFrame(np.random.randn(1096, 2),
                  index=pd.date_range('2014-01-01',
                                      '2014-16-31'),columns=list('AB'))
walk_df = df.cumsum()

walk_df.head()

walk_df.plot()
```

Task

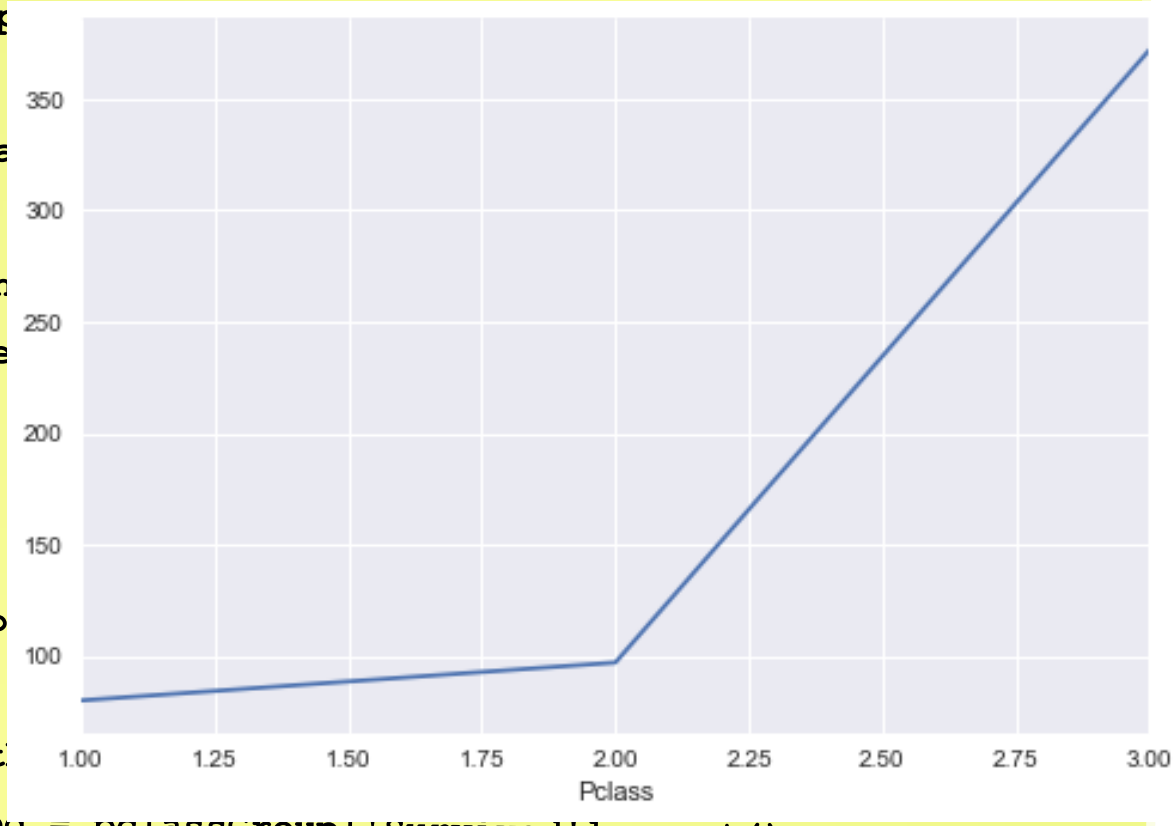
- Returning to our titanic dataset.

- Write a program that will generate a basic graph showing the number of first class, second class and third class passengers that died on the titanic (use groupby functionality in your answer).

```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv("titanic.csv")

# filter the dataframe to only return rows
# where the passenger died
criteria = df['Survived']==0
fatalities = df[criteria]

pclassGroup = fatalities.groupby("Pclass")

# extract the Survived column from the groupby object
classSurived = pclassGroup['Survived'].count()
print (classSurived)

classSurived.plot()
plt.show()
```

```
Pclass
1      80
2      97
3     372
Name: Survived, dtype: int64
```

```
import pandas as pd
import matp

df = pd.rea

# filter th
# where the
criteria =
fatalities

pclassGroup

# extract t
classSurived = pclassGroup['Survived'].count()
print (classSurived)

classSurived.plot()
plt.show()
```




```
Pclass
1    80
2    97
3    372
Name: Survived, dtype: int64
```

Plot Axis Command

- As we mentioned before we can continue to add elements to the plt figure object and alter aspects of the figure.
- We can use the **axis** method to **set the viewable area** of the graph.
- In the case below we fix it so that the x coordinates shown are from 0 to 6 and the y coordinates from 0 to 20.

```
import matplotlib.pyplot as plt


plt.plot([1,2,3,4], [1,4,9,16])
# Sets viewable area to 0-6 on x axis and 0-20 on y axis
plt.axis([0, 6, 0, 20])
plt.show()
```

Plot Axis Command

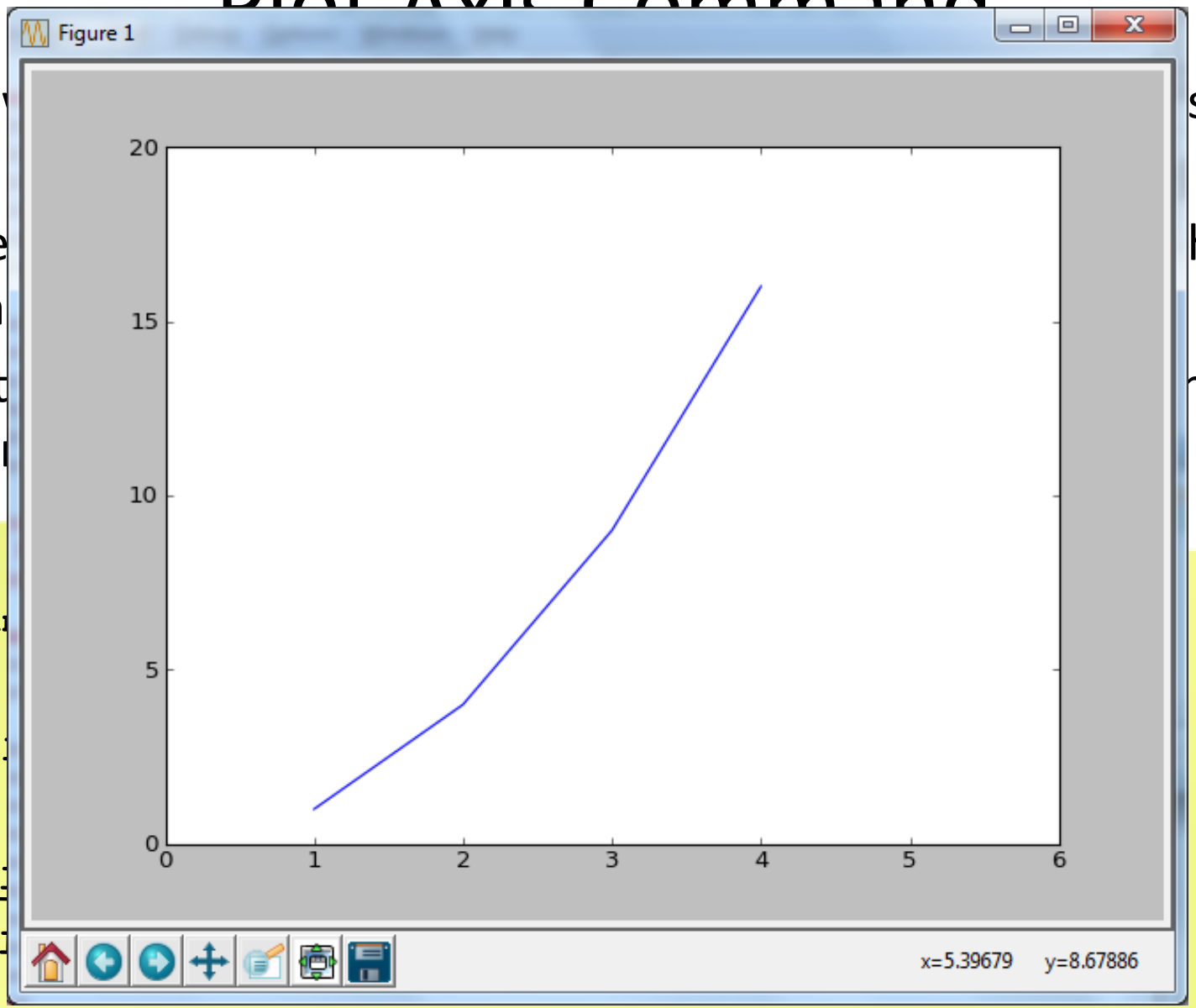
- As
- We
- In t

s to the plt

he gra

n are fro

```
i
p
#
p
p
```



Using the Axis Command with a DataFrame

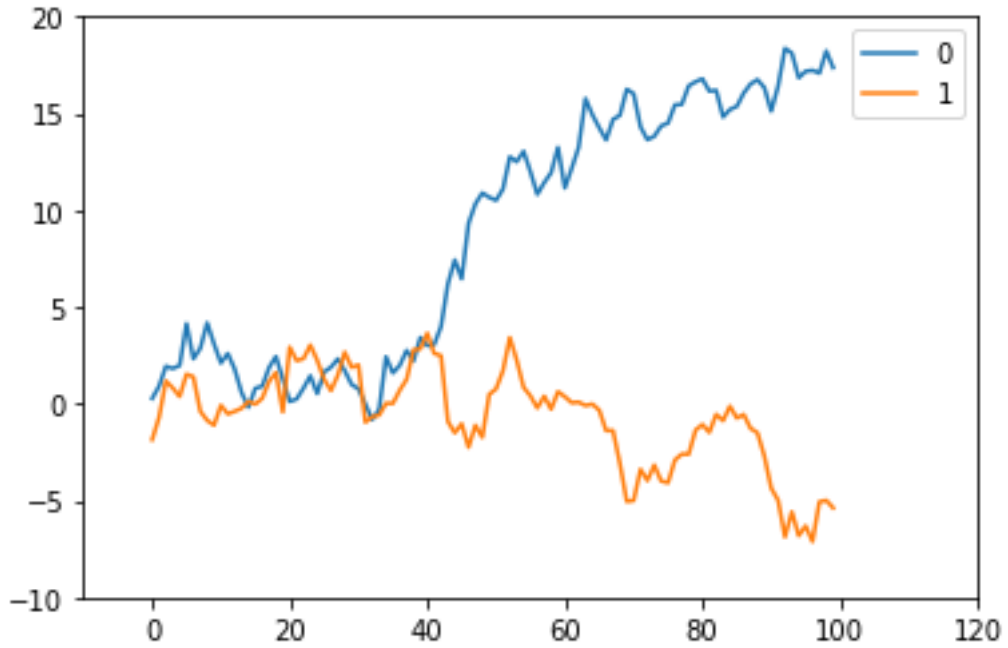
- The DataFrame object is using MatPlotLib when it generates a graph.
- Therefore, the functions we use to alter the plot in MatPlotLib can also be used in conjunction with Pandas.

```
import matplotlib.pyplot as plt
import pandas as pd


np.random.seed(17)
df = pd.DataFrame(np.random.randn(100, 2))


df= df.cumsum()
df.plot()
plt.axis([-10, 120, -10, 20])
plt.show()
```

Using the Axis Command with a DataFrame

- The DataFrame object is using MatPlotLib when it generates a graph.
- Therefore, the functions we use to alter the plot in MatPlotLib can also be used in conjunction with Panda

```
import matplotlib.pyplot a
import pandas as pd


np.random.seed(17)
df = pd.DataFrame(np.rando


df= df.cumsum()
df.plot()
plt.axis([-10, 120, -10, 20])
plt.show()
```



Plot Command and Formatting

- Notice that all information we have plotted to a graph so far has been depicted as lines

- This section will look at adjusting many of the properties of these lines, size, colour, format (dashed)

- plot is a versatile command
  - We can add multiple pairs of x, y coordinates
  - For **every x, y pair** of arguments, there is an **optional third argument** which is the format string that indicates the **color and line type** of the plot.

  - The letters and symbols of the format string are from MATLAB, and you concatenate a **color** string with a **line style string**.

Plot Command and Formatting

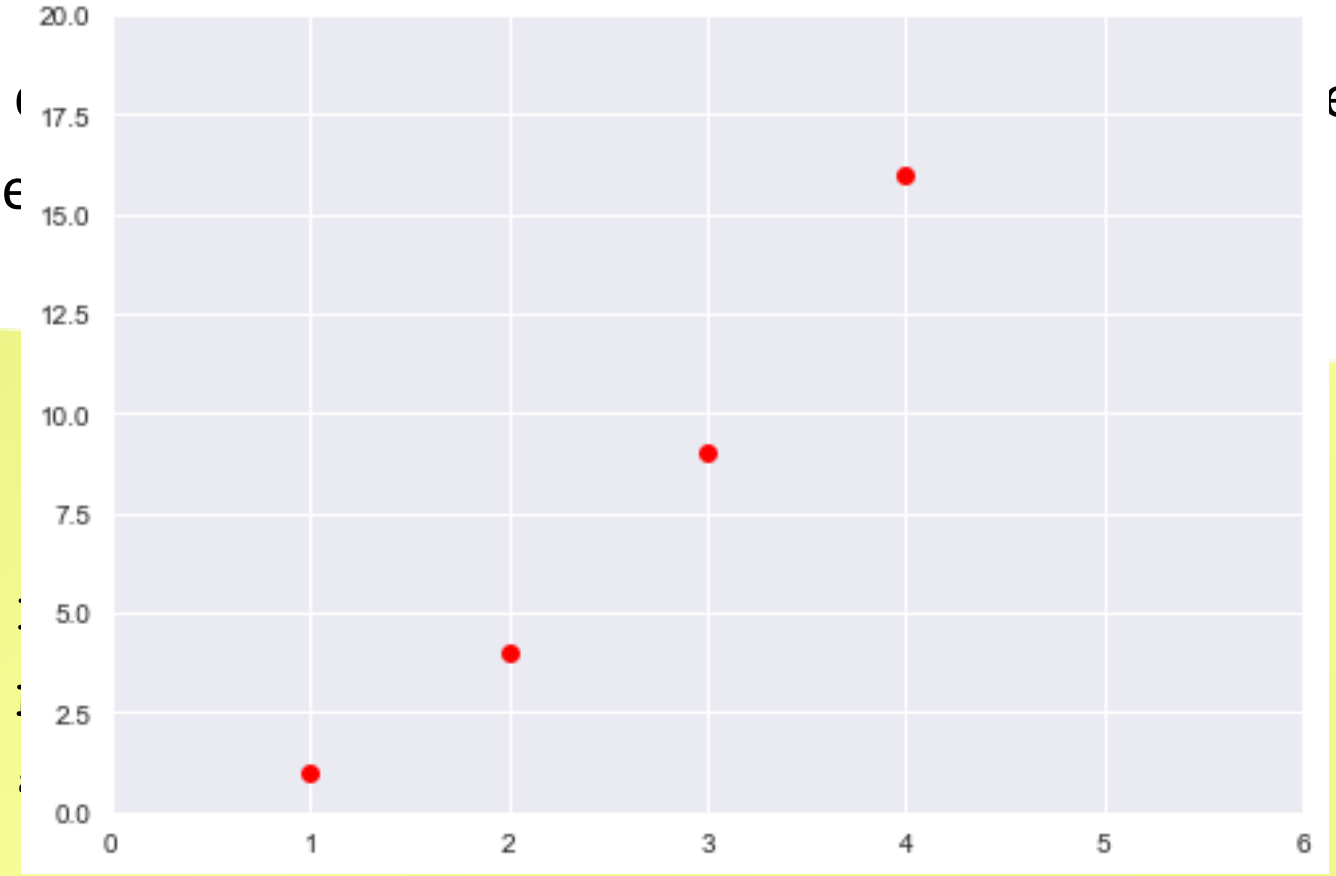
- The default format string is ‘b-‘, which is a solid blue line.
- For example, to plot with red circles, you would issue:

```
import matplotlib.pyplot as plt

plt.plot([1,2,3,4], [1,4,9,16], 'ro')
plt.axis([0, 6, 0, 20])
show()
```

Plot Command and Formatting

- The (
- For e

e.



Plot Command and Formatting

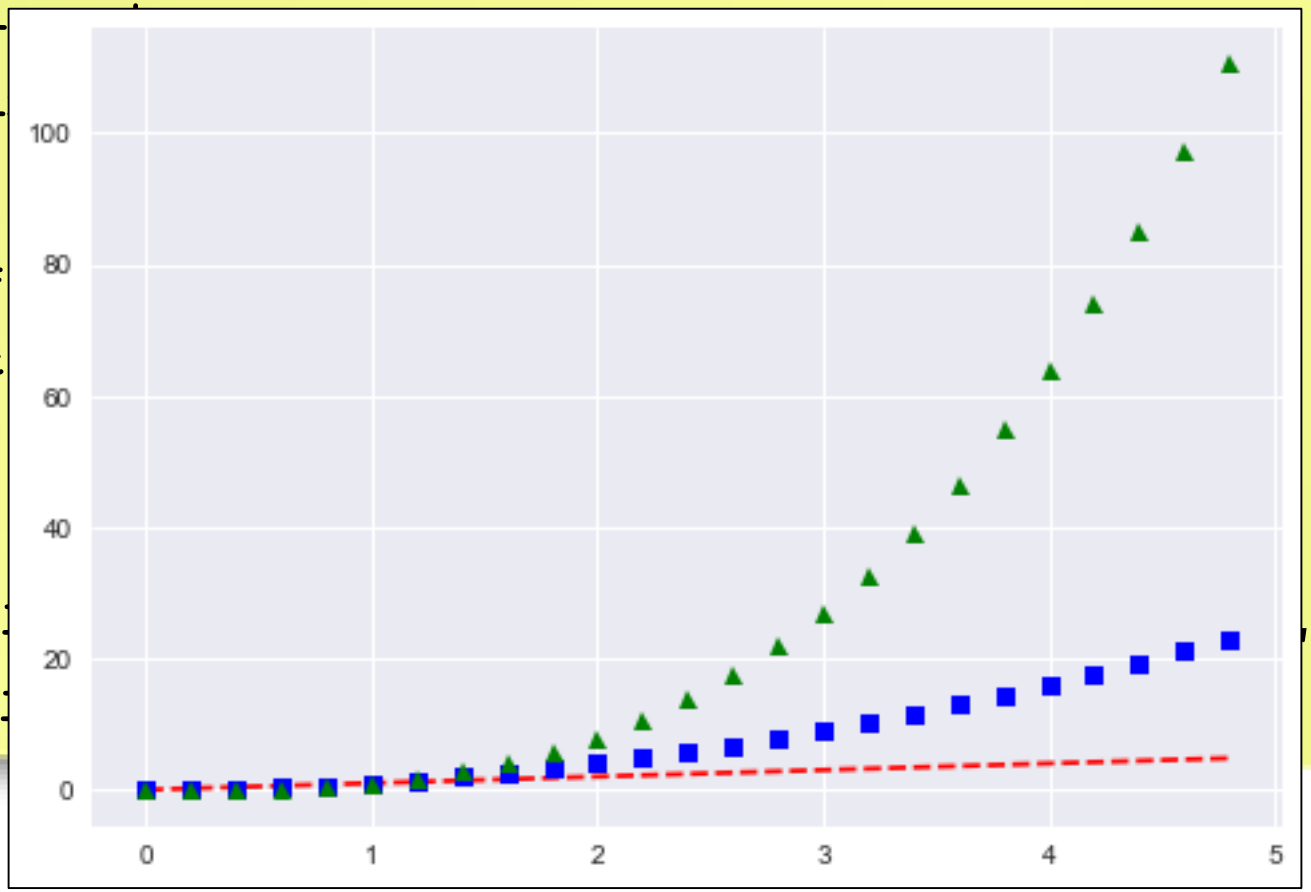
character	color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

character	description
'-'	solid line style
'--'	dashed line style
'-.'	dash-dot line style
':'	dotted line style
'.'	point marker
','	pixel marker
'o'	circle marker
'v'	triangle_down marker
'^'	triangle_up marker
'<'	triangle_left marker
'>'	triangle_right marker
'1'	tri_down marker
'2'	tri_up marker
'3'	tri_left marker
'4'	tri_right marker
's'	square marker
'p'	pentagon marker
'*'	star marker
'h'	hexagon1 marker
'H'	hexagon2 marker
'+'	plus marker
'x'	x marker
'D'	diamond marker
'd'	thin_diamond marker
'|'	vline marker
'_'	hline marker

Plotting Multiple Line using Plot Command

```
import numpy as np
import matplotlib.pyplot as plt

# evenly sampled time at 200ms intervals
t = np.arange(0., 5., 0.2)

# red dashes, blue squares and green triangles
plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')
plt.show()
```

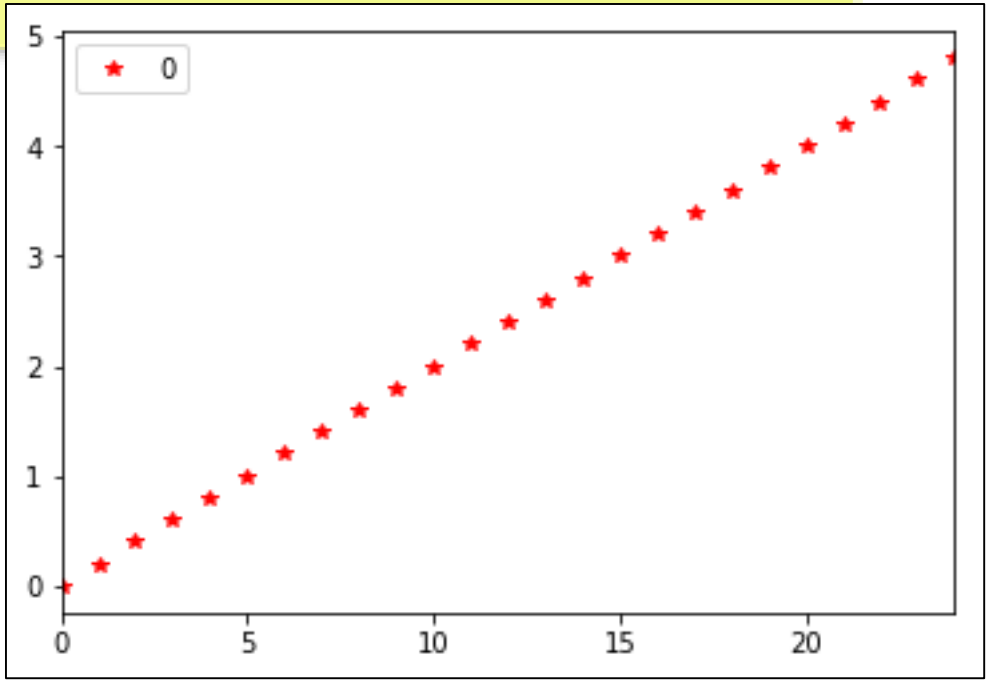
Plotting Multiple Line using Plot Command



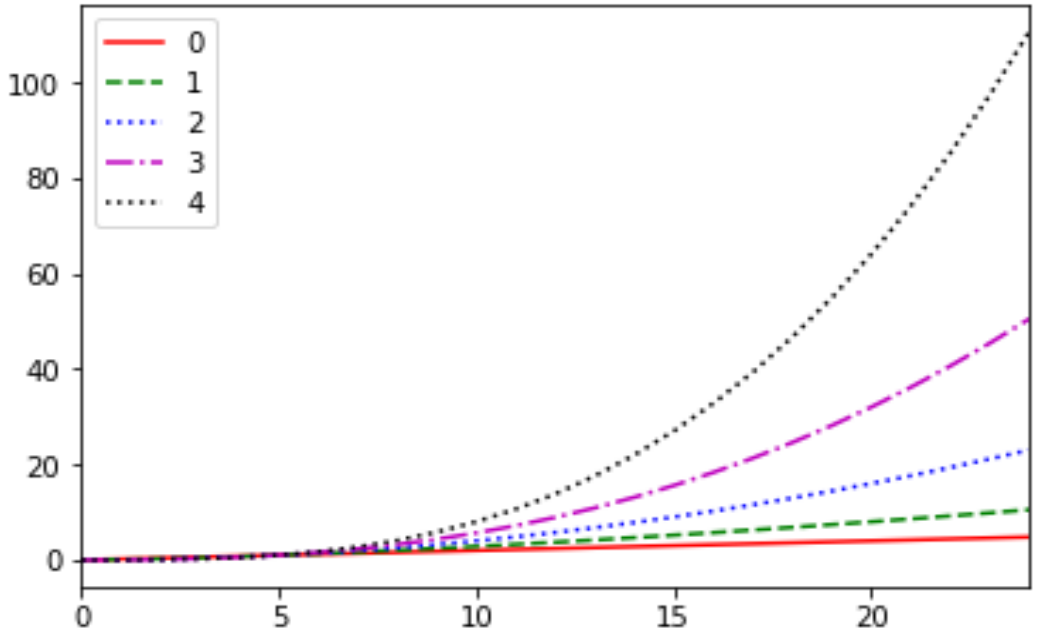
Shortcut Styles in Pandas

```
t = np.arange(0., 5., 0.2)
data = pd.DataFrame({0 : t})
# specifying color and line style for our data
ax = data.plot(style='r*')
```

We can use the style parameter in DataFrames to replicate the shortcut functionality we saw in the previous slides.



```
t = np.arange(0., 5., 0.2)
line_style = pd.DataFrame({0 : t,
                           1 : t**1.5,
                           2 : t**2.0,
                           3 : t**2.5,
                           4 : t**3.0})
ax = line_style.plot(style=['r-', 'g--', 'b:', 'm-.', 'k:'])
```



Notice in this example we provide a list of shortcut styles. One for each column in our DataFrame

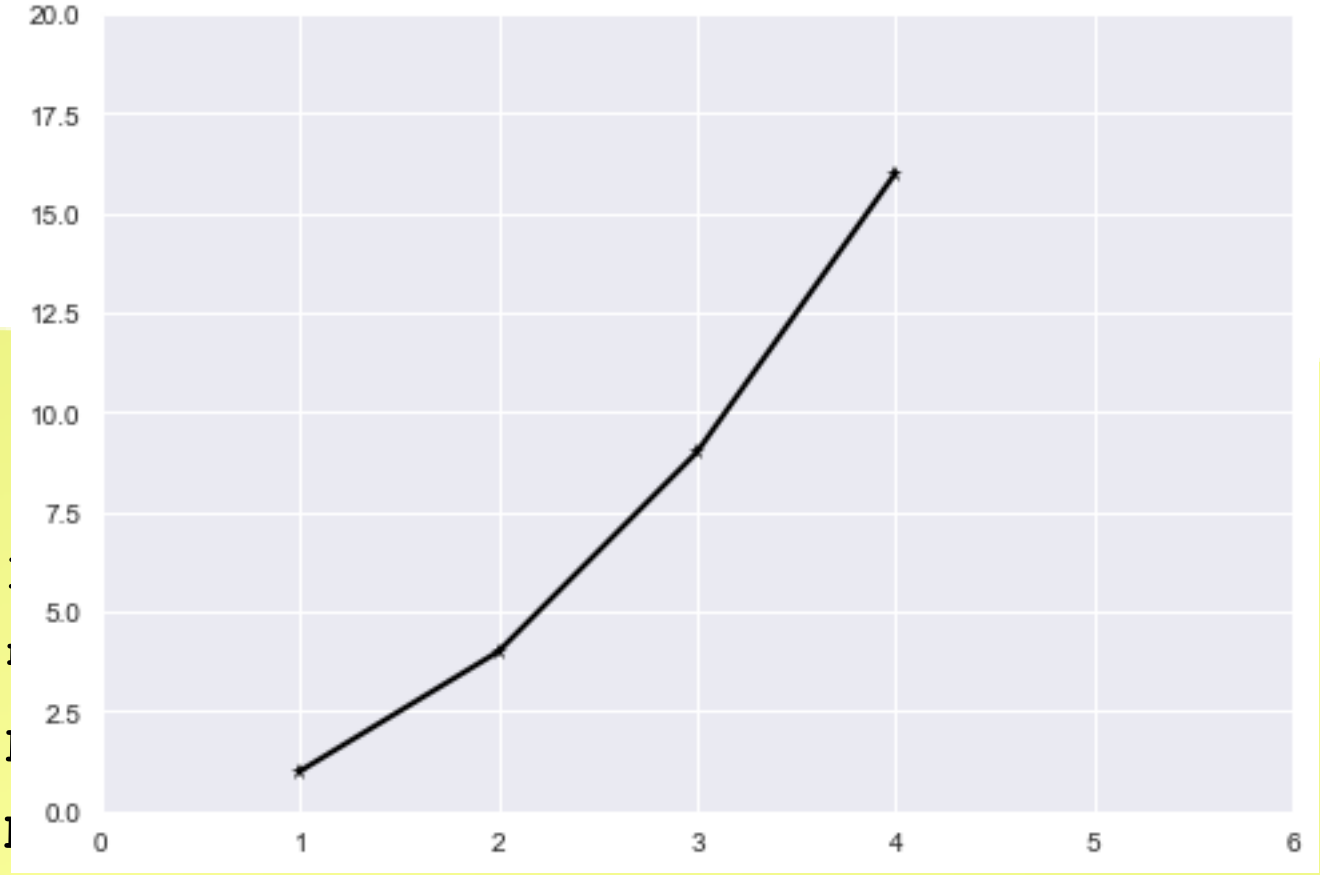
Use keyword arguments when plotting the line

- Lines have many attributes that you can set: linewidth, dash style, etc;
- Aside from the approach in the previous slides we can use keyword arguments when plotting the line
- When using the plot method we can provide additional arguments that control the line
  - plt.plot(x, y, **linewidth**=2.0, **marker** = '*', **linestyle** = '-', **color** = 'black')
  - linewidth, linestyle, color, marker, markersize, markeredgewidth, markeredgecolor, markerfacecolor, etc
  - http://matplotlib.org/api/axes_api.html#matplotlib.axes.Axes.plot

Plot Command and Formatting

```
import matplotlib.pyplot as plt


plt.plot([1,2,3,4], [1,4,9,16], linewidth=2.0,
marker = '*', linestyle = '-', color = 'black')
plt.axis([0, 6, 0, 20])
plt.show()
```

Plot Command and Formatting

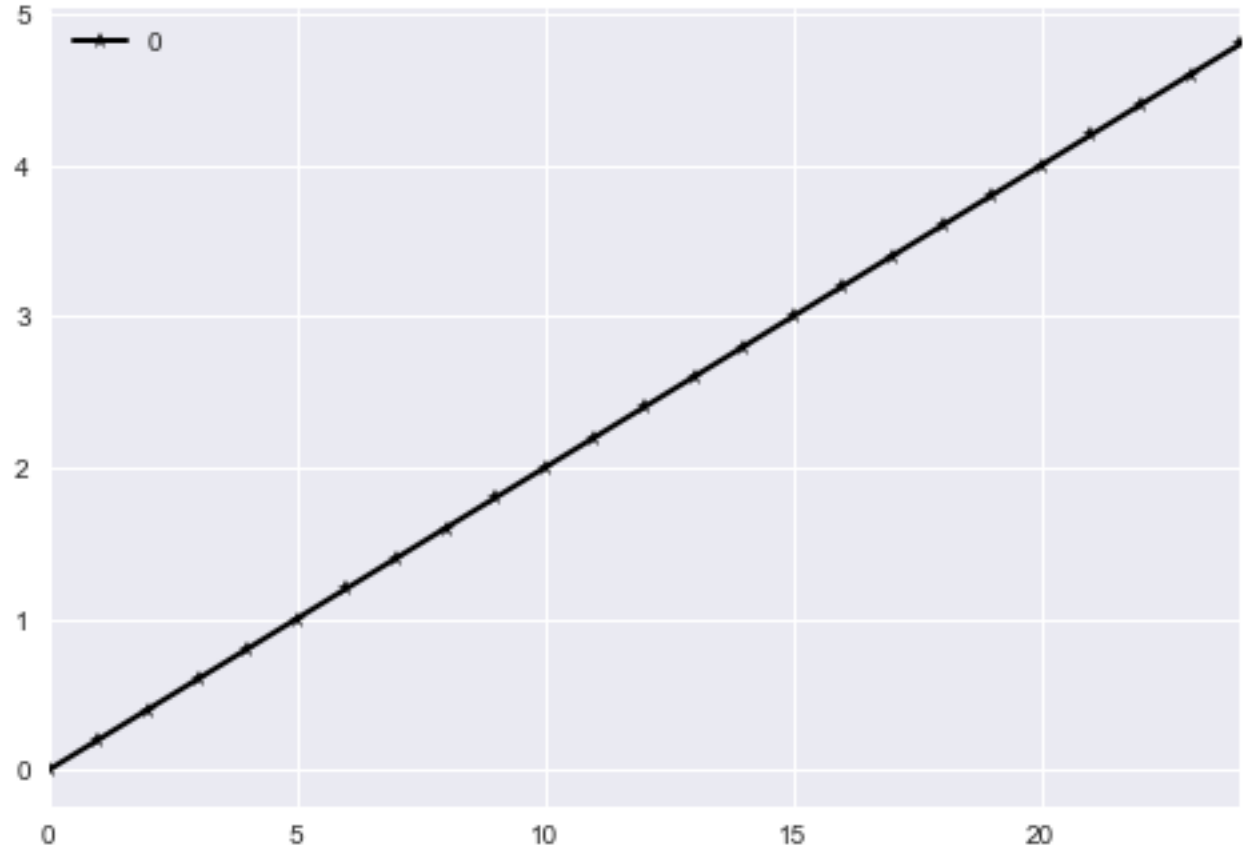


Performing Formatting in Pandas

```
t = np.arange(0., 5., 0.2)

data = pd.DataFrame({0 : t})

# specifying color and line style for our data
data.plot(lw = 2.0, linewidth=2.0, marker = '*',
          linestyle = '-', color = 'black')
```

Performing Formatting in Pandas

```
t =

data

# sp

data
```



Using Text in Figures

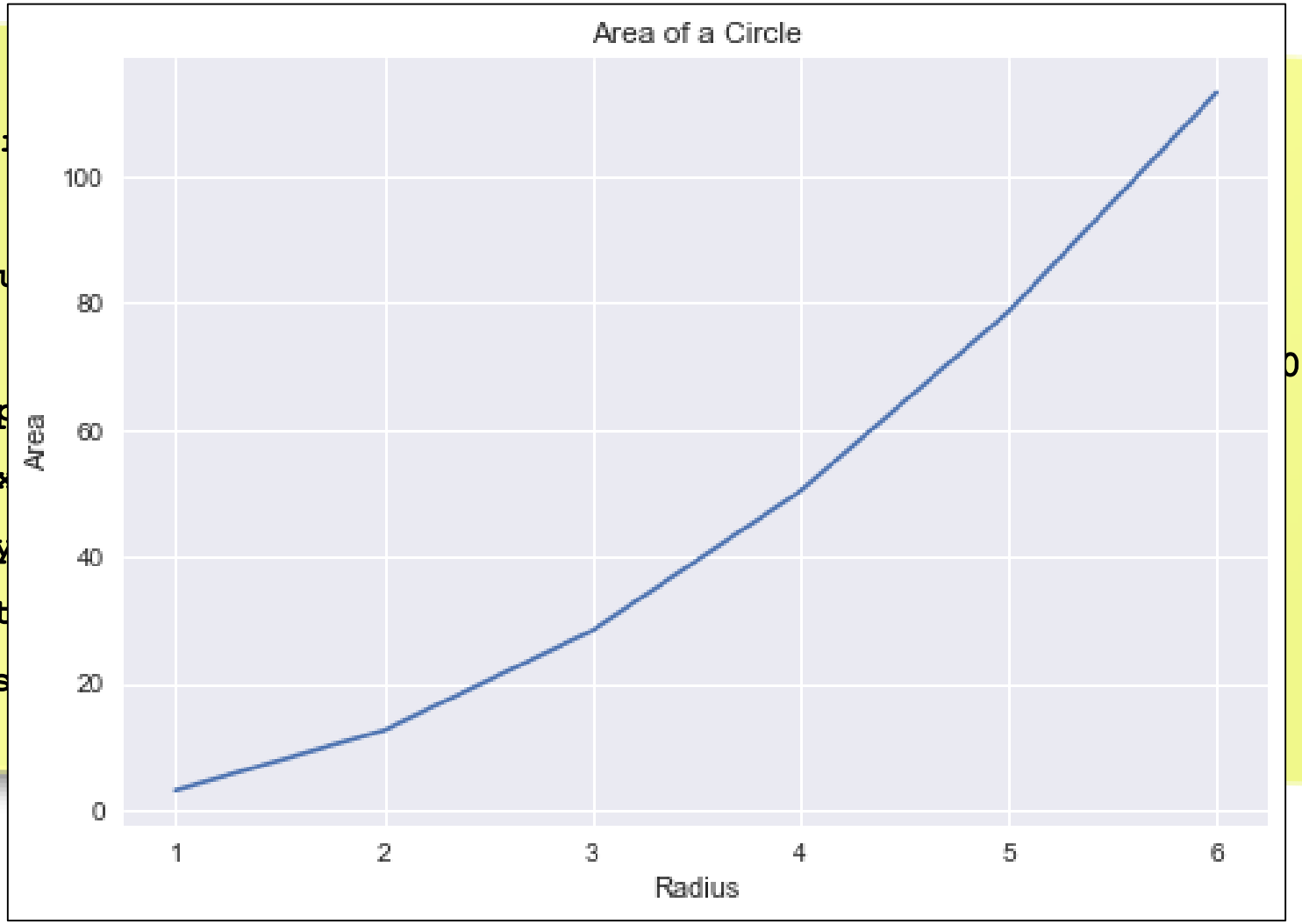
- **xlabel()** - add an axis label to the x-axis;
- **ylabel()** - add an axis label to the y-axis;
- **title()** - add a title to the Axes;
- **text()** - add text at an arbitrary location to the Axes;
- **suptitle()** - add a title to the Figure;

Adding a Title

```
import matplotlib.pyplot as plt


radius = [1.0, 2.0, 3.0, 4.0, 5.0, 6.0]
area = [3.14159, 12.56636, 28.27431, 50.26544, 78.53975, 113.09724]
plt.plot(radius, area)
plt.xlabel('Radius')
plt.ylabel('Area')
plt.title('Area of a Circle')
plt.show()
```

Adding a Title

```
impo

radiu
area
plt.p
plt.x
plt.y
plt.t
plt.s
```




```
09724]
```

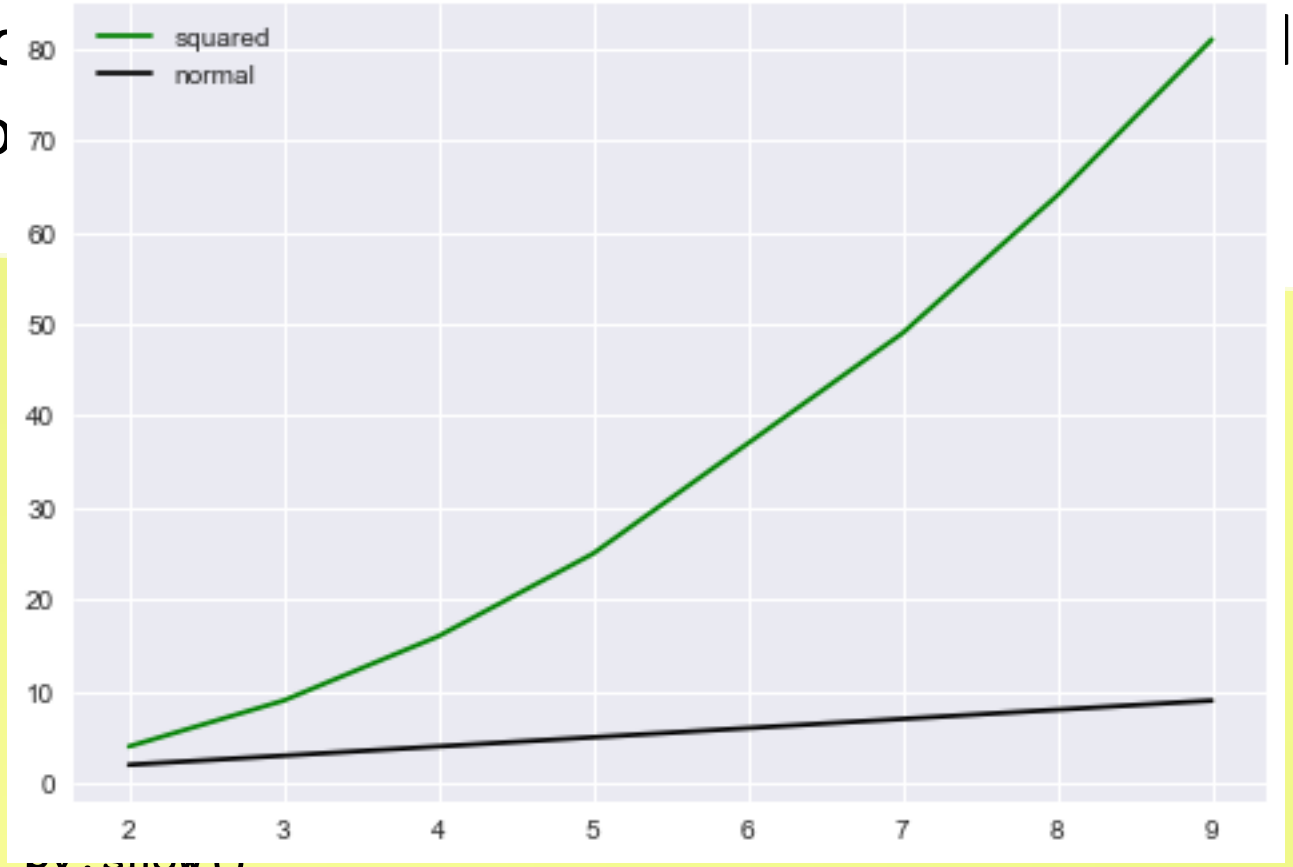
Creating a Legend

- We can use pyplot legend function to create a legend for our graph

```
import numpy as np
import matplotlib.pyplot as py



t = np.array([2, 3, 4, 5, 7, 8, 9], float)
py.plot(t,t**2,'g-', t, t, 'k-')
py.legend(['squared','normal'])
py.show()
```

Creating a Legend

- We c legend for o



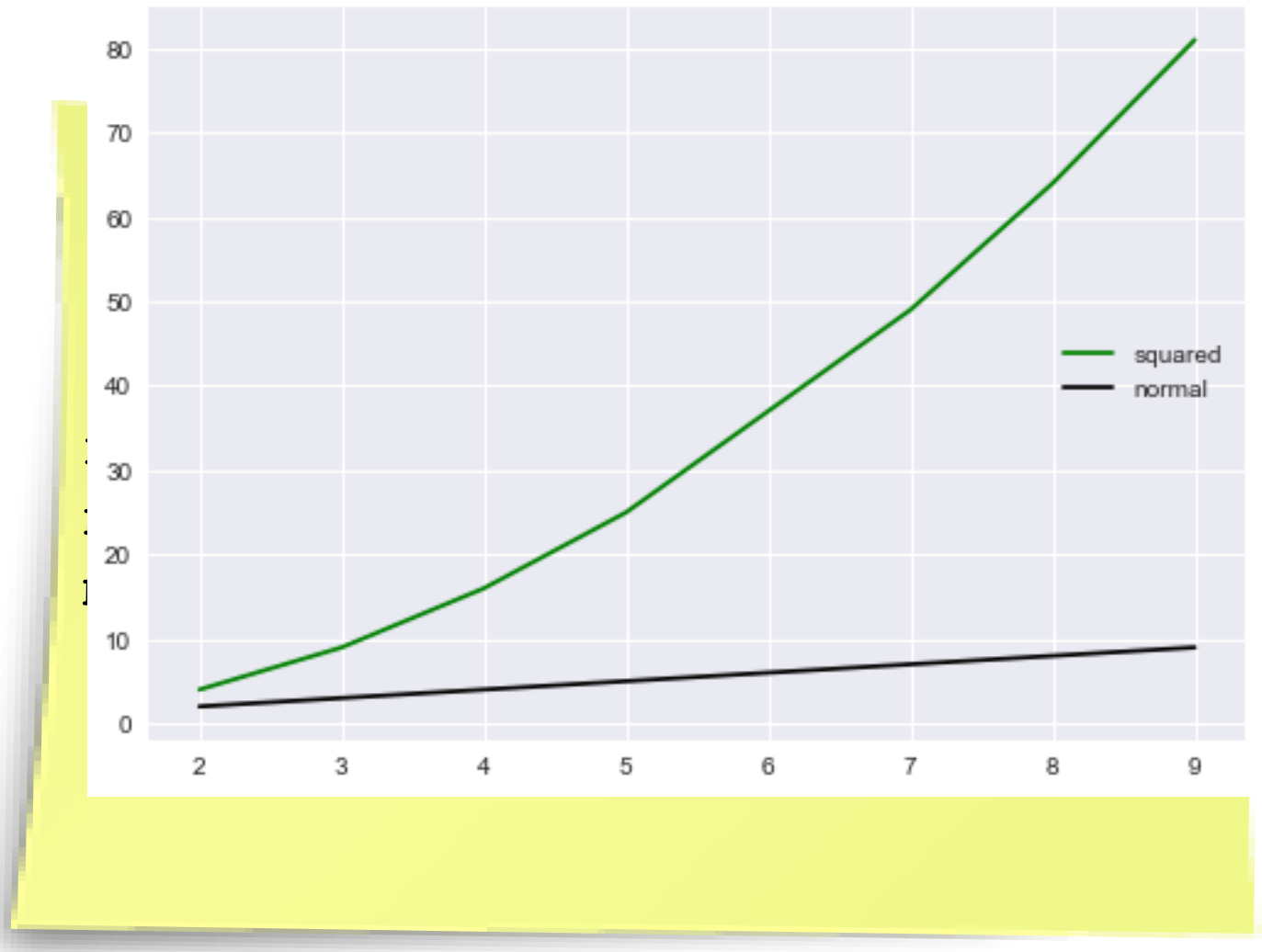

```
py.show()
```

Position of a Legend

- We can position a legend within a graph by providing a location argument:
  - legend(['label1', 'label2', 'label3‘], loc='upper left')
  - You can use the integer value or the String value

Location String	Location Code
‘best’	0
‘upper right’	1
‘upper left’	2
‘lower left’	3
‘lower right’	4
‘right’	5
‘center left’	6
‘center right’	7
‘lower center’	8
‘upper center’	9
‘center’	10

```
t = np.array([2, 3, 4, 5, 7, 8, 9], float)
py.plot(t,t**2,'g-', t, t, 'k-')
py.legend(['squared','normal'], loc = 7)
py.show()
```

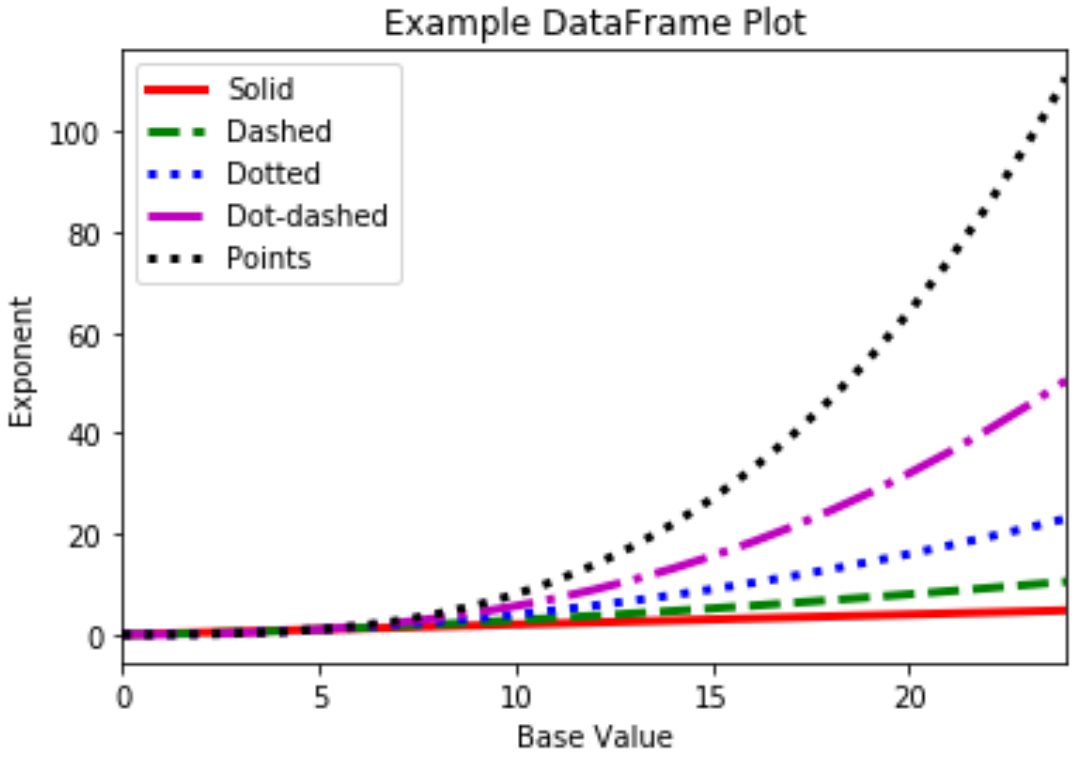

80
70
60
50
40
30
20
10
0
2
3
4
5
6
7
8
9
squared
normal

Legends, Titles, etc with Pandas

```
t = np.arange(0., 5., 0.2)
legend_labels = ['Solid', 'Dashed', 'Dotted', 'Dot-dashed', 'Points']
line_style = pd.DataFrame({0 : t,
                           1 : t**1.5,
                           2 : t**2.0,
                           3 : t**2.5,
                           4 : t**3.0})

ax = line_style.plot(style=['r-', 'g--', 'b:', 'm-.', 'k:'], lw=3, )
plt.title("Example DataFrame Plot")
plt.xlabel('Base Value')
plt.ylabel('Exponent');
plt.legend(legend_labels, loc='upper left')
```

Legends, Titles. etc with Pandas

```
t = np.arange(0., 5., 0.2
legend_labels = ['Solid'
line_style = pd.DataFrame
```



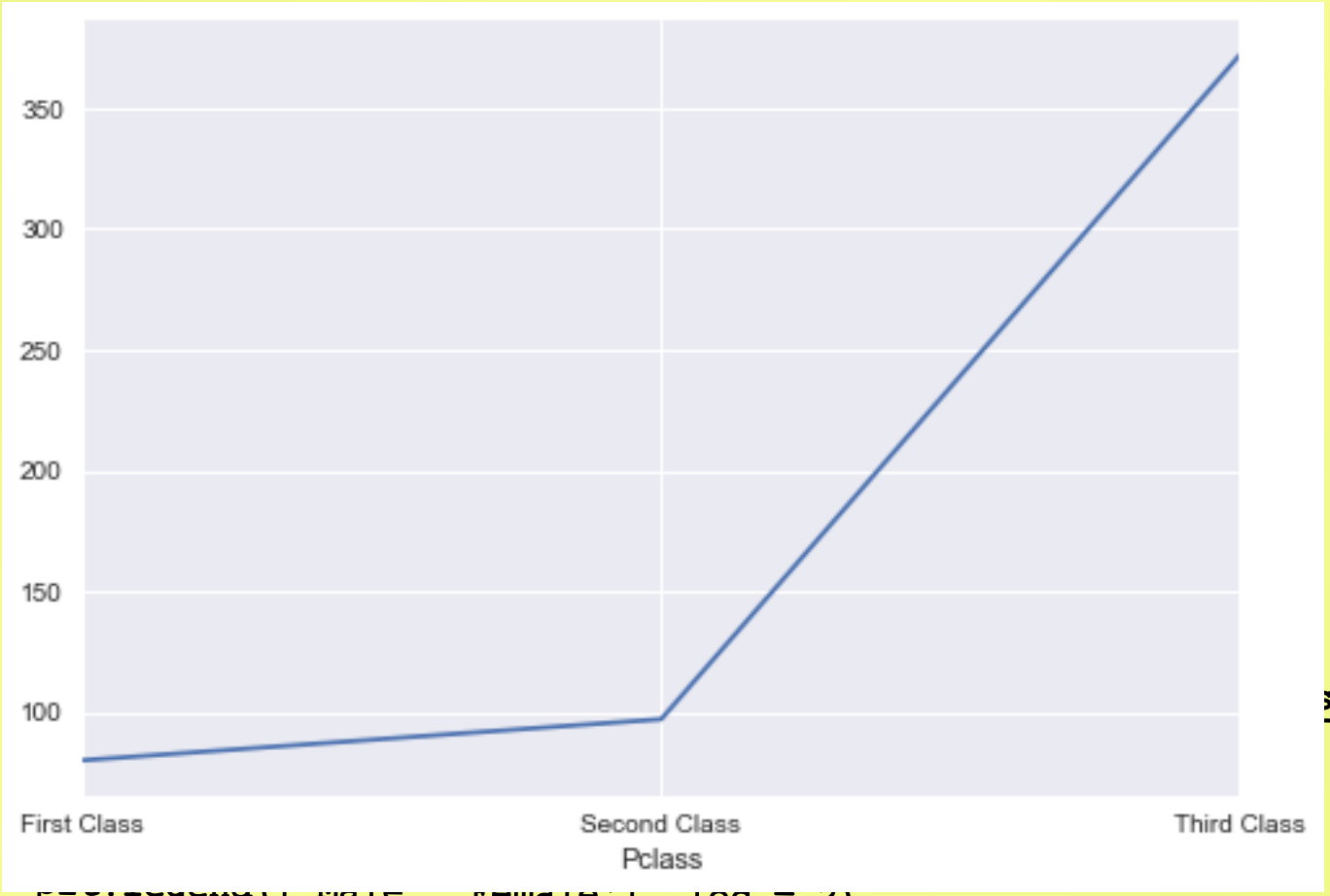

```
ax = line_style.plot(style=['r-', 'g--', 'b:', 'm-.', 'k:'], lw=3, )
plt.title("Example DataFrame Plot")
plt.xlabel('Base Value')
plt.ylabel('Exponent');
plt.legend(legend_labels, loc='upper left')
```

Using xticks – Place labels along x axis

- It can also be very useful to place labels along the x axis of a graph.
- The plt.xticks function allows us to **associate specific values** on the x-axis with **descriptive labels**

```
df = pd.read_csv("titanic.csv")

# filter the dataframe to only return rows
# where the passenger died
criteria = df['Survived']==0
fatalities = df[criteria]

pclassGroup = fatalities.groupby("Pclass")

# extract the Survived column from the groupby object
classSurived = pclassGroup['Survived'].count()
print (classSurived)

labels = ['First Class', 'Second Class', 'Third Class']
plt.xticks([1,2,3], labels)

plt.legend(['Male','Female'], loc = 2)

classSurived.plot()
plt.show()
```

```
df = pd.read_csv("titanic.csv")

# filter the dataframe to only return rows
```

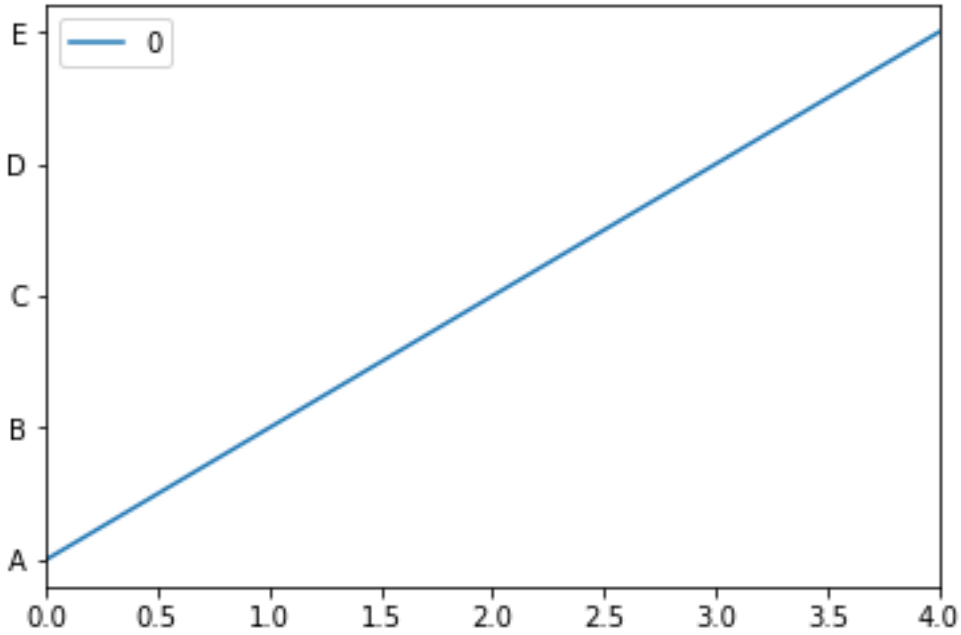



```
ss']
plt.legend(['Male', 'Female'], loc = 2)

classSurived.plot()
plt.show()
```

Using yticks – Place labels along x axis

- There is a similar yticks method that allows us to specify the label values of on y axis. The following example, illustrates it's use with a DataFrame

```
# rename y-axis tick labels to A, B, C, D, and E
ticks_data = pd.DataFrame(np.arange(0,5))
ticks_data.plot()
plt.yticks(np.arange(0, 5), list("ABCDE"));
```



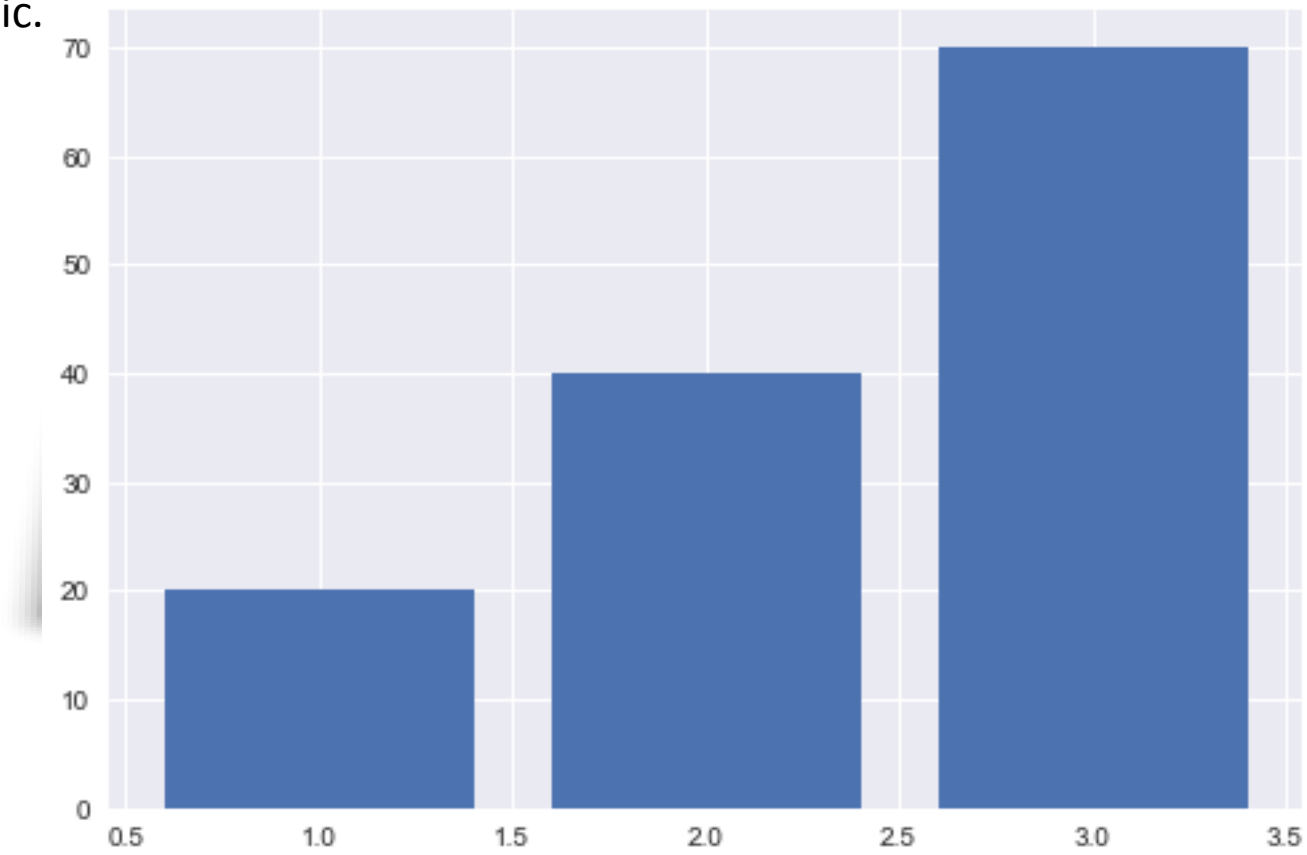
Creating a Bar Graph

- A bar graph would be a better way to depict the information from the titanic dataset.
- Creating a bar graph is relatively straight forward using the **plt.bar** function.

```
# males contains the number of males
# per class that dies on the titanic

plt.bar([1, 2, 3], males)

plt.show()
```

Creating a Bar Graph

- A bar graph would be a better way to depict the information from the titanic dataset.
- Creating a bar graph is relatively straight forward using the **plt.bar** function.
- Below we produce a bar graph for the males from various classes that died on the titanic.



Bar Graphs

- In the next example we incorporate **both the male and female fatalities** on the titanic into the same bar graph (again broken down by Pclass)

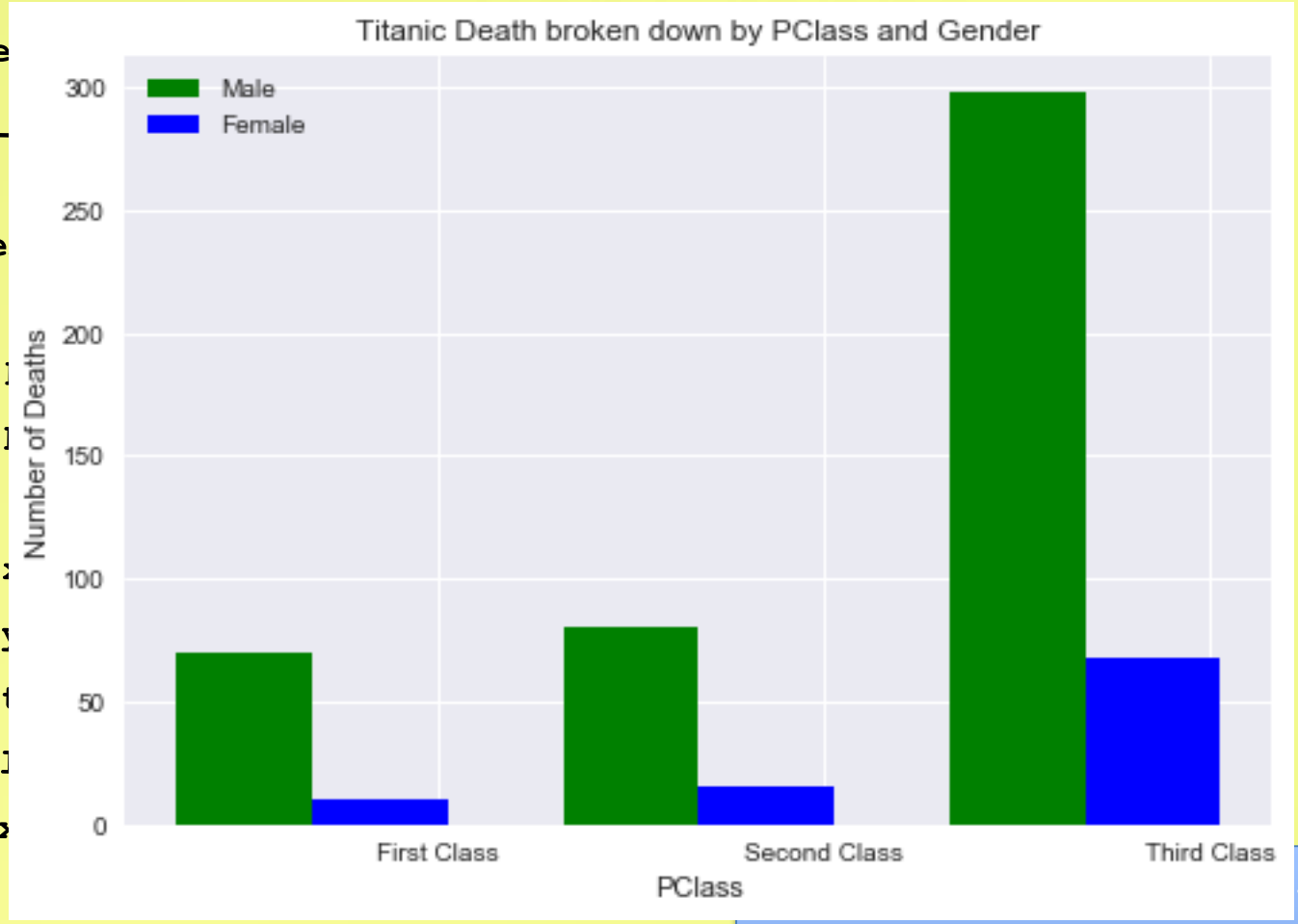
- Just as with the plot function we can **call the bar function multiple times** to add more bars to a given graph

- As with the plot command there are various arguments we can pass to the bar command to control it's style (width of the bar, colour, etc) - http://matplotlib.org/api/pyplot_api.html

```
labels = ['First Class', 'Second Class', 'Third Class']
bar_width = 0.35

index = np.arange(1,4) #[1,2,3]

plt.bar(index, males, bar_width, color = 'g')
plt.bar(index+bar_width, females, bar_width, color = 'b')

plt.xlabel('PClass')
plt.ylabel('Number of Deaths')
plt.title('Titanic Death broken down by PClass and Gender')
plt.legend(['Male','Female'], loc = 2)
plt.xticks(index+0.5, labels)


plt.show()
```

In this example males and females are lists that contains the number of fatalities for males and females broken down by PClass

```
labe
bar_

inde

plt.
plt.

plt.
plt.y
plt.t
plt.l
plt.x

plt.show()
```



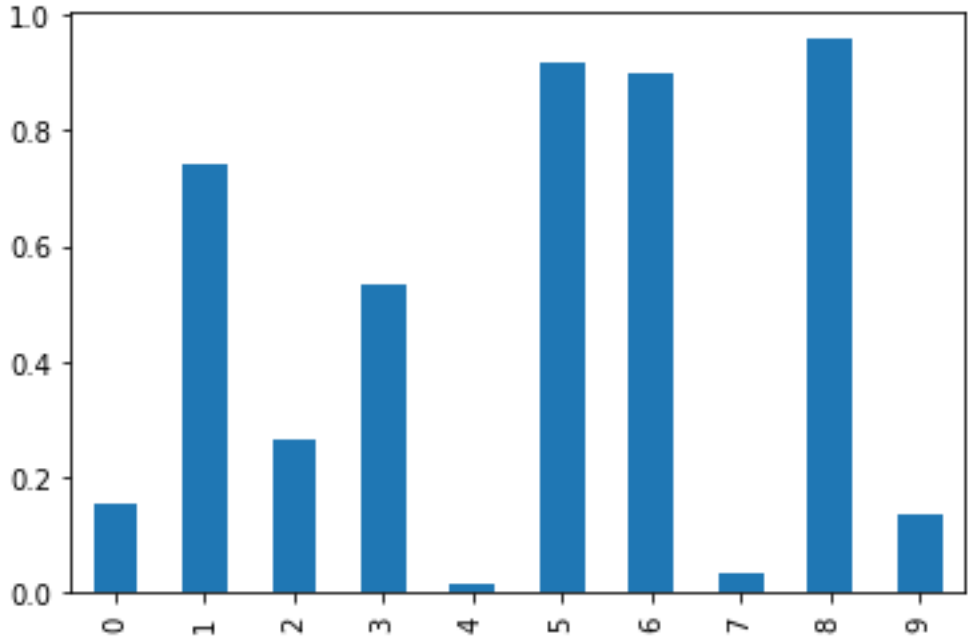
females

are lists that contains the number of fatalities for males and females broken down by PClass

Bar Graphs with Series Object

- The following example shows how we can generate a bar graph using a Series object.
  - X axis becomes the index

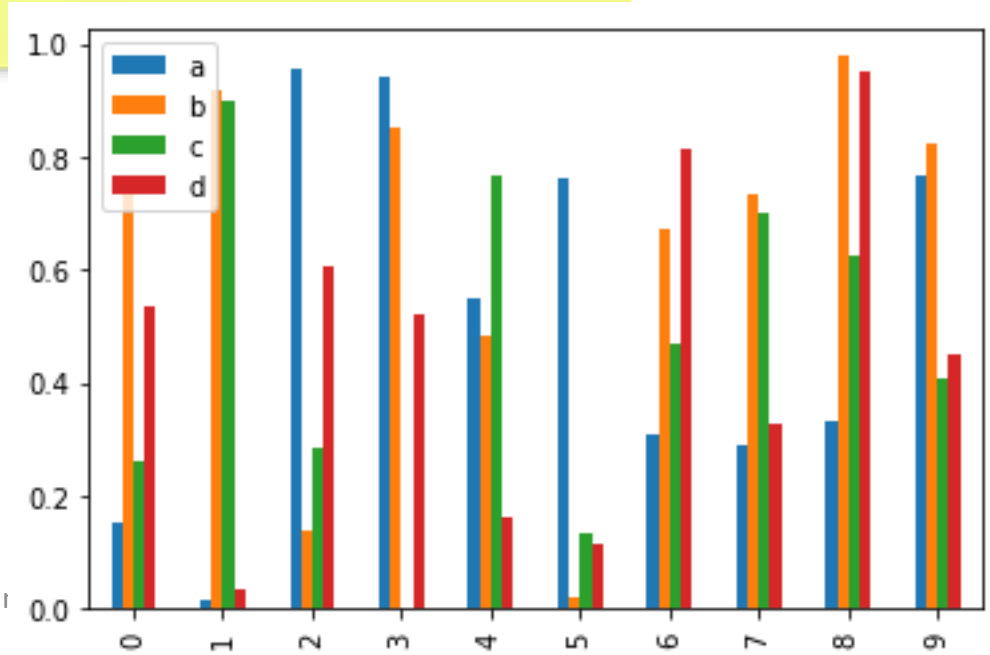
```
np.random.seed(12)
s = pd.Series(np.random.rand(10))
# plot the bar chart
s.plot(kind='bar');
```



Bar Graphs with Pandas

- The following example shows how we can generate a bar graph using a DataFrame object. Notice that each group of bars represents the data in one row

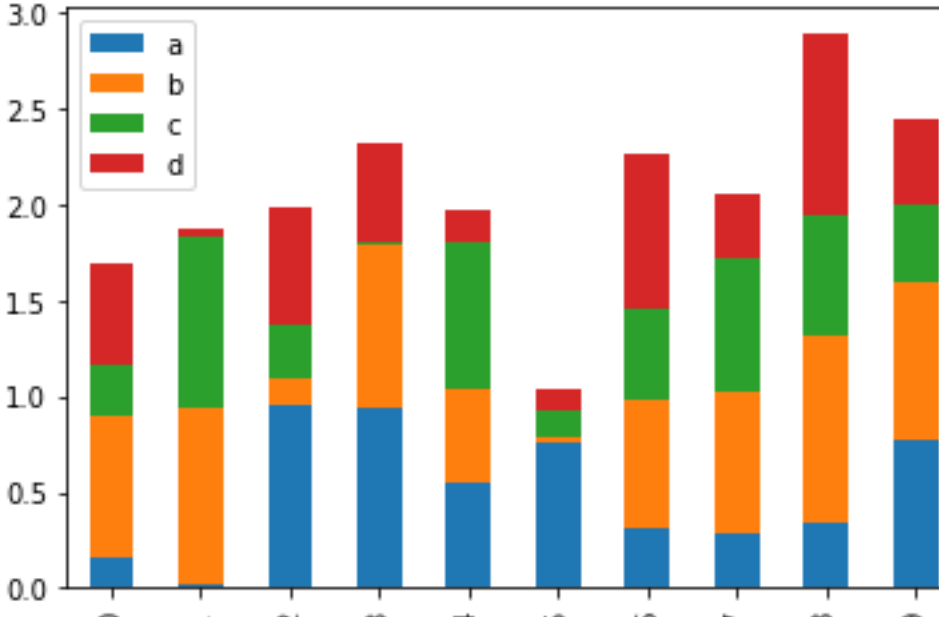
```
np.random.seed(12)
df2 = pd.DataFrame(np.random.rand(10, 4),
                   columns=['a', 'b', 'c', 'd'])
# draw the multi-series bar chart
df2.plot(kind='bar');
```



Bar Graphs with Pandas

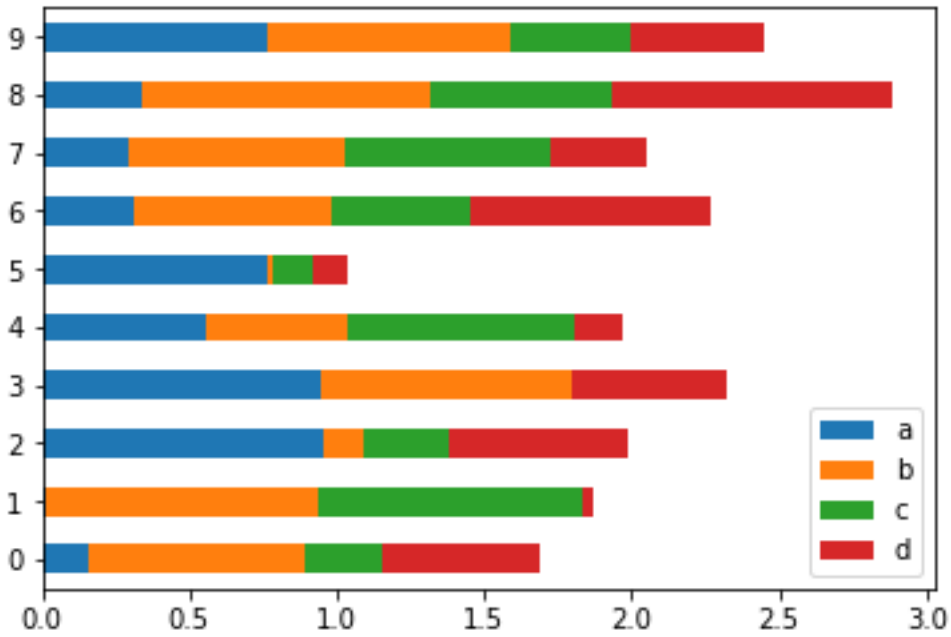
```
np.random.seed(12)
df2 = pd.DataFrame(np.random.rand(10, 4),
                   columns=['a', 'b', 'c', 'd'])
# draw the multi-series bar chart
df2.plot(kind='bar', stacked=True)
```

- A useful parameter we can specify with a bar chart is to stack the bars together.



Bar Graphs with Pandas

```
np.random.seed(12)
df2 = pd.DataFrame(np.random.rand(10, 4),
                   columns=['a', 'b', 'c', 'd'])
# draw the multi-series bar chart
df2.plot(kind='barh', stacked=True)
```



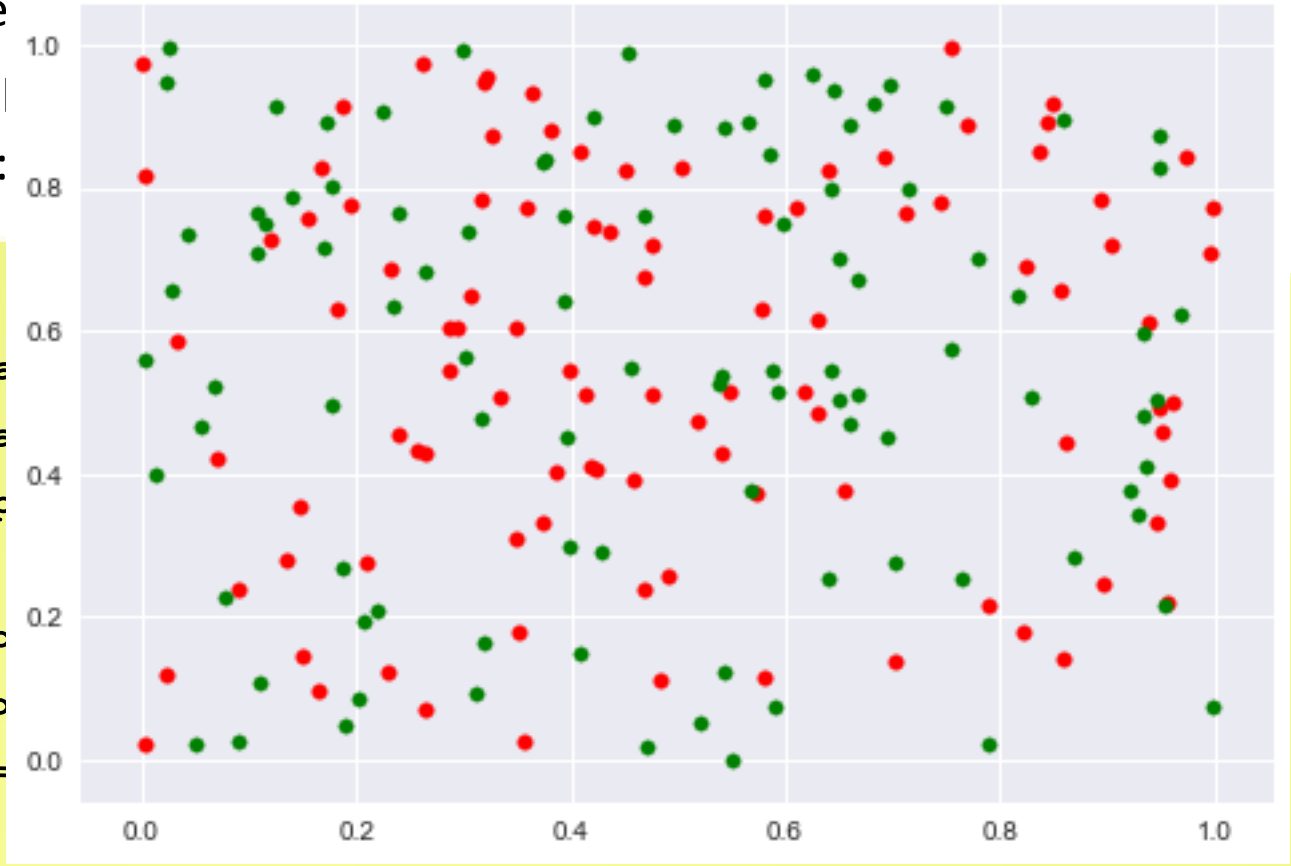
Scatter Plots

- Scatter Plots can be created using matplotlib.**pyplot.scatter**
  - Input data x, y (arrays), s : scalar size or array, optional, default: 20
  - c : color

```
N = 100
x = np.random.rand(N)
y = np.random.rand(N)
area = np.pi * (np.random.rand()*10)

newX = np.random.rand(N)
newY = np.random.rand(N)
newArea = np.pi * (np.random.rand()*10)

plt.scatter(x, y, s=area, color ='r')
plt.scatter(newX, newY, s=newArea, color ='g')
plt.show()
```

Scatter Plots

- Scatte
  - In
  - c :



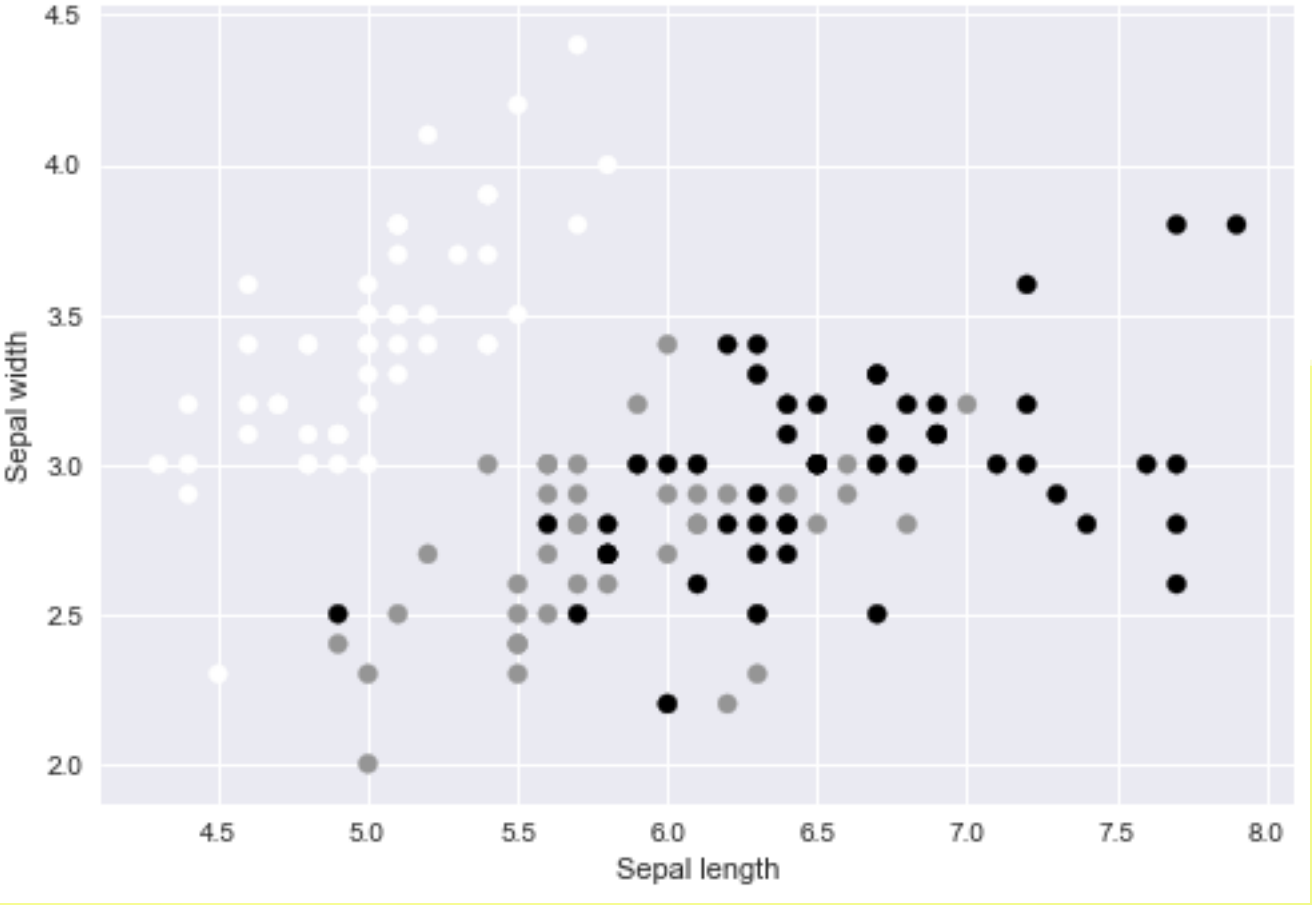
lt: 20

```
N = 100
x = np.ra
y = np.ra
area = np


newX = np
newY = np
newArea =
```

```
plt.scatter(x, y, s=area, color ='r')
plt.scatter(newX, newY, s=newArea, color ='g')
plt.show()
```

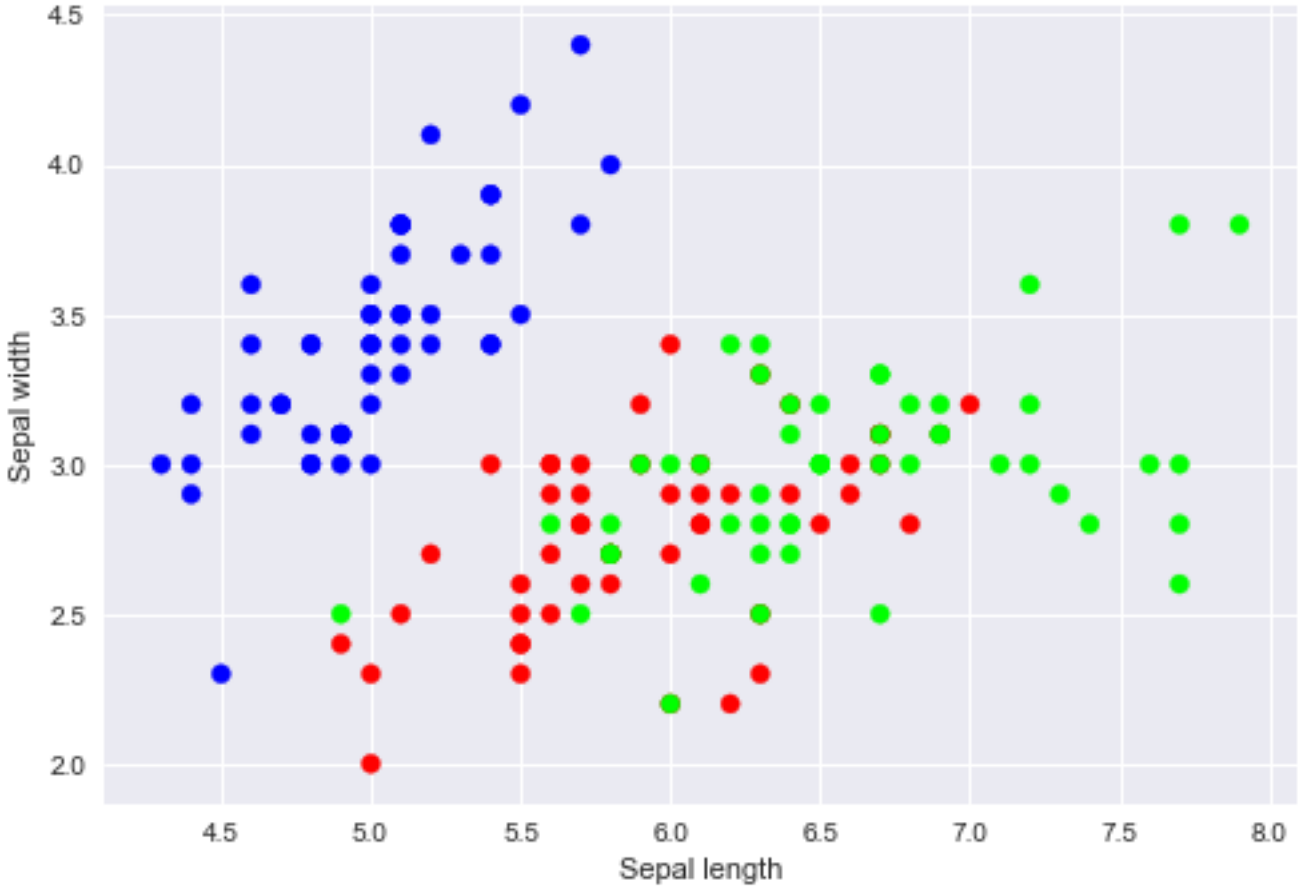
Scatter Plots

```
import matplotlib.pyplot as plt
from sklearn import datasets

# import some data to play with
iris = datasets.load_iris()
X = iris.data[:, :2]  # we only take the first two features.
Y = iris.target

# Plot the training points
plt.scatter(X[:, 0], X[:, 1], c=Y)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.show()
```




```
import matplot
from sklearn i

# import some
iris = dataset
X = iris.data[
Y = iris.targe

# Plot the training points
plt.scatter(X[:, 0], X[:, 1], c=Y)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.show()
```

```
import matplotlib.p
from sklearn import

# import some data
iris = datasets.loa
X = iris.data[:, :2
Y = iris.target

# Plot the training points
plt.scatter(X[:, 0], X[:, 1], c=Y, cmap ='brg')
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.show()
```



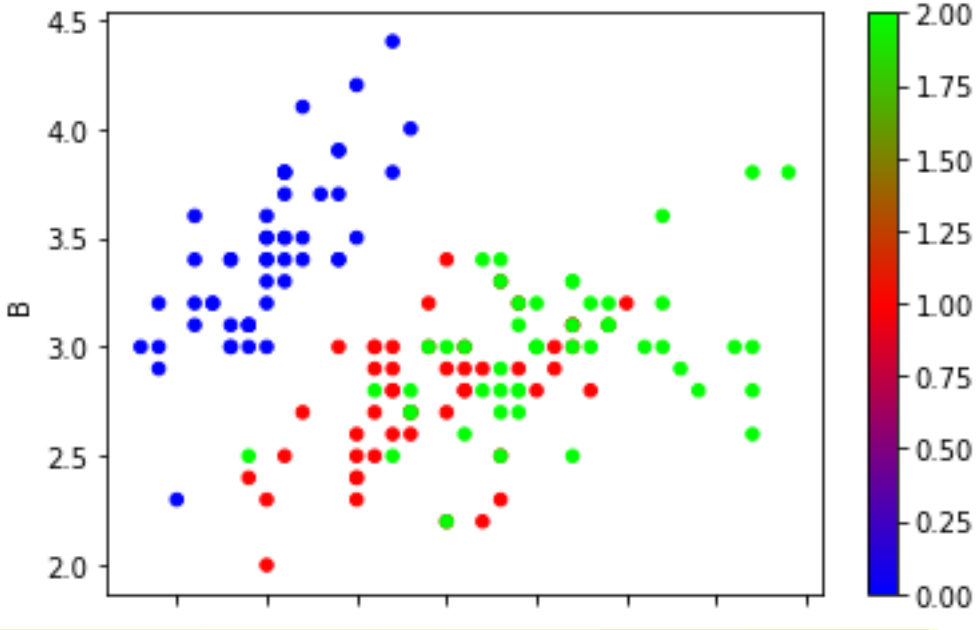
https://matplotlib.org/users/colormaps.html

Scatter Plots with DataFrames

```
import matplotlib.pyplot as plt
from sklearn import datasets

# import some data to play with
iris = datasets.load_iris()
X = iris.data[:, :2]  # we only take the first two features.
Y = iris.target

df = pd.DataFrame(data = X, columns = list("AB"))
df["class"] = Y

df.plot(kind='scatter', x='A', y='B', c=df["class"], cmap='brg')
```

Scatter Plots with DataFrames

```
import matplotlib.pyplot as plt
from sklearn import datasets

# import some data to play with
iris = datasets.load_iris()
X = iris.data[:, :2]  # we only ta
Y = iris.target
#print (iris.columns)

df = pd.DataFrame(data = X, columns = list("AB"))
df["class"] = Y

df.plot(kind='scatter', x='A', y='B', c=df["class"], cmap='brg')
```



Creating a Histogram in Matplotlib

- Histograms are useful for plotting the frequency distribution of numbers across a range of possible values.

- In Python it works by:
  - Taking a list of numbers
  - **Binning** those numbers within a number of ranges
  - And counting the number of occurrences in each bin.

```
import matplotlib.pyplot as plt
import numpy as np

# Generate an Array containing a 1000 random numbers from a
# Gaussian distribution
gaussian_numbers = np.random.normal(size=1000)
plt.hist(gaussian_numbers)

plt.title("Gaussian Histogram")
plt.xlabel("Value")
plt.ylabel("Frequency")

plt.show()
```

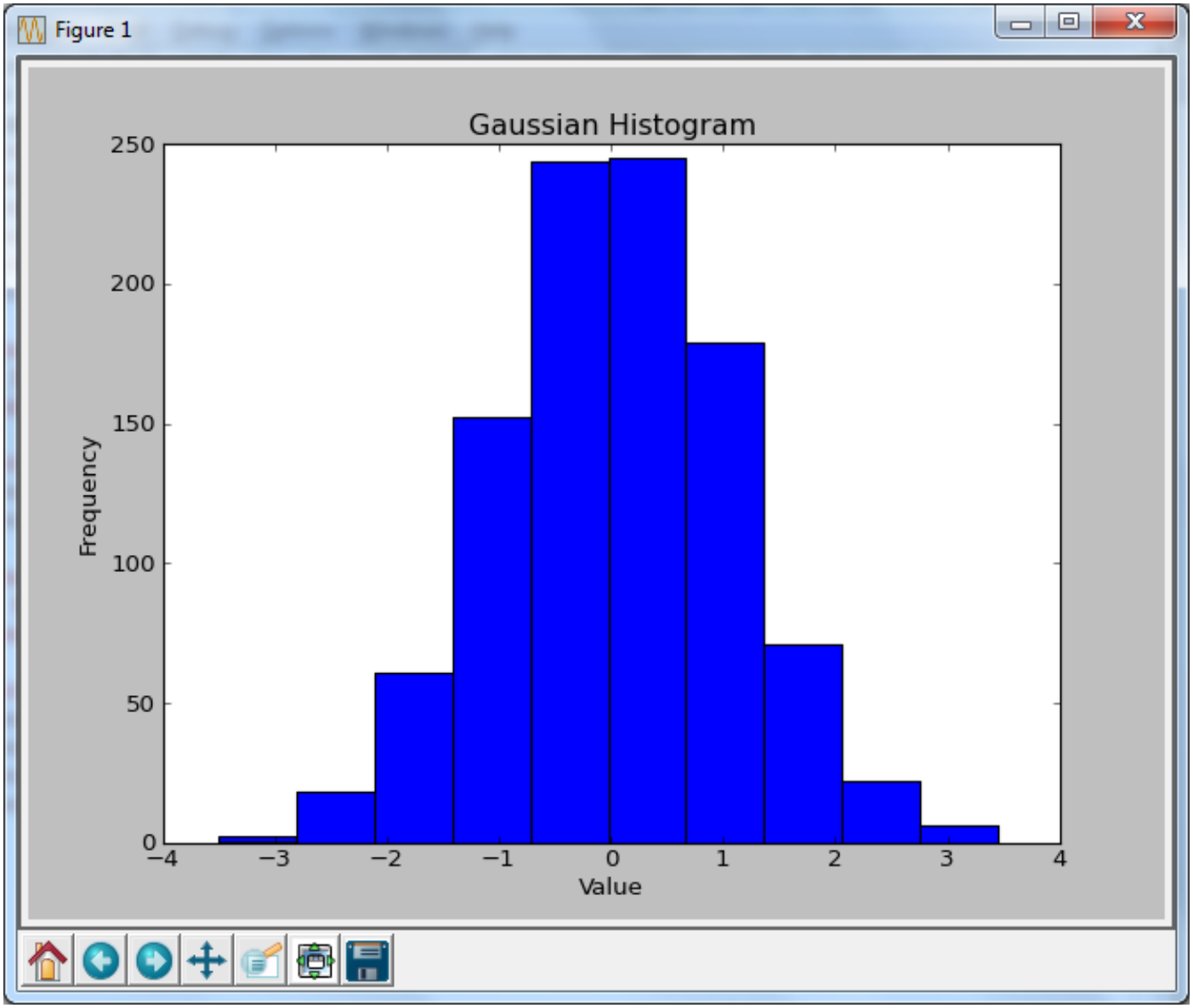


Figure 1
Gaussian Histogram
250
200
150
100
50
0
Frequency
−4
−3
−2
−1
0
1
2
3
4
Value

```
import matplotlib.pyplot as plt
import numpy as np


gaussian_numbers = np.random.normal(size=1000)
plt.hist(gaussian_numbers, bins=40)


plt.title("Gaussian Histogram")
plt.xlabel("Value")
plt.ylabel("Frequency")

plt.show()
```

Notice we can specify the number of bins to use

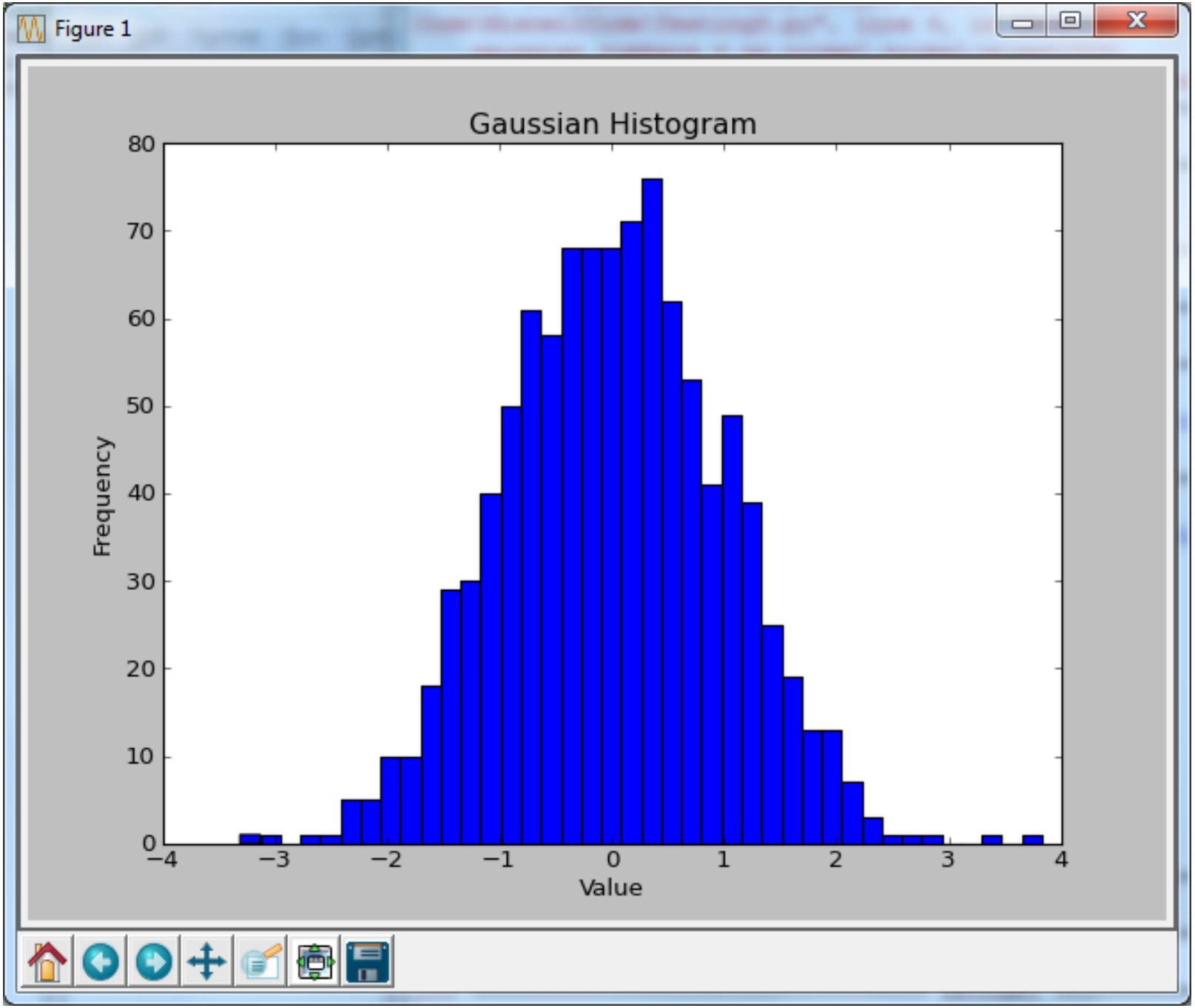
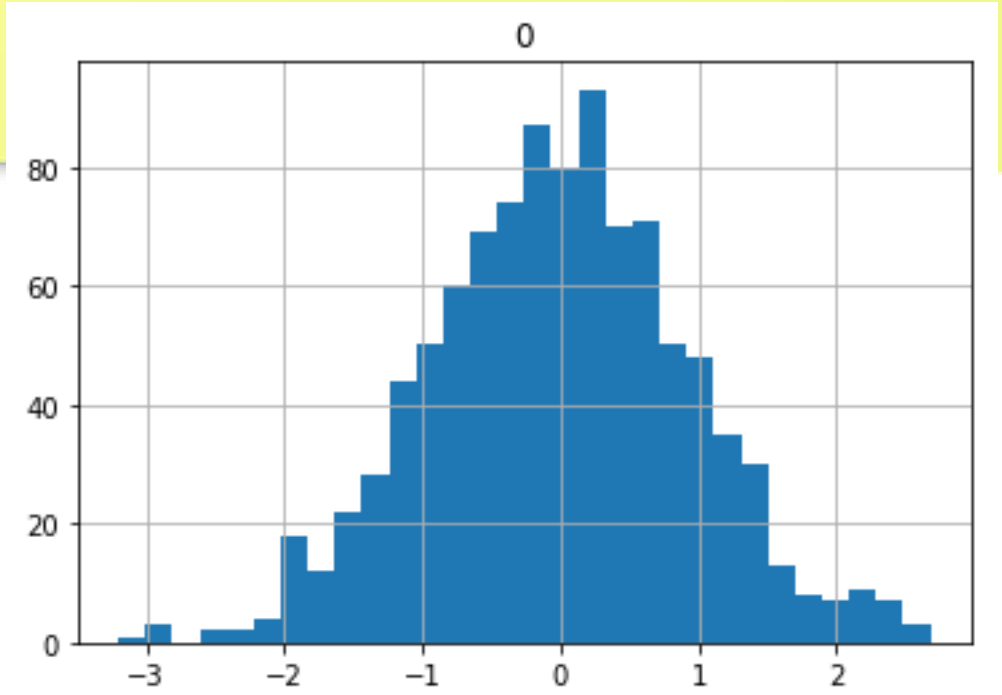


Figure 1
Gaussian Histogram
Frequency
Value
0
10
20
30
40
50
60
70
80
−4
−3
−2
−1
0
1
2
3
4

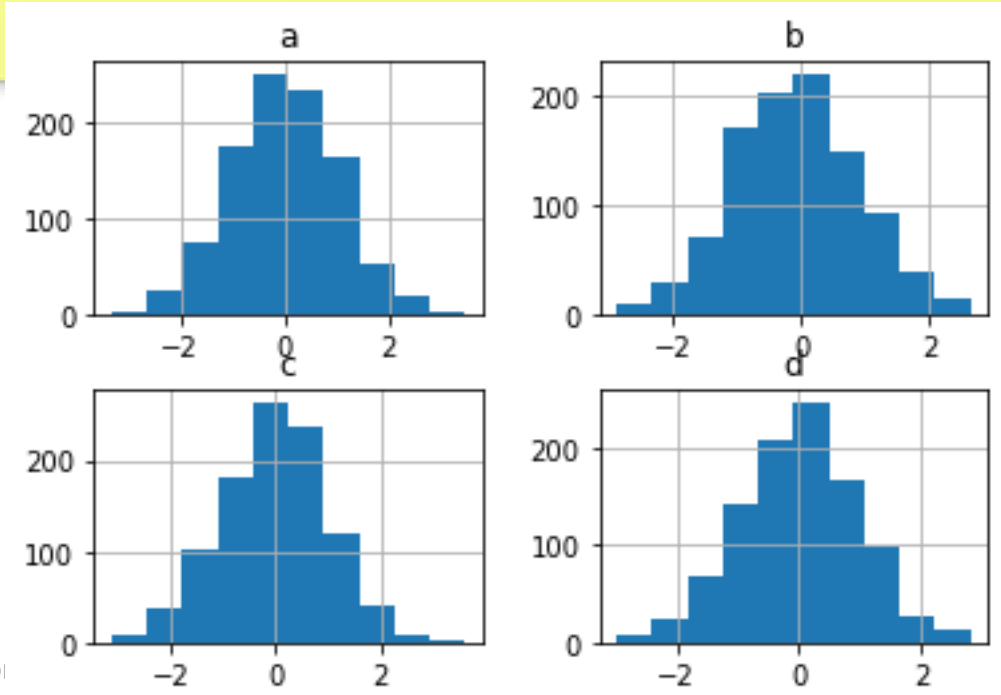
Pandas - Histograms

```
# create a histogram
np.random.seed(10)
# 1000 random numbers
dfh = pd.DataFrame(np.random.randn(1000))
# draw the histogram
dfh.hist(bins=30);
```



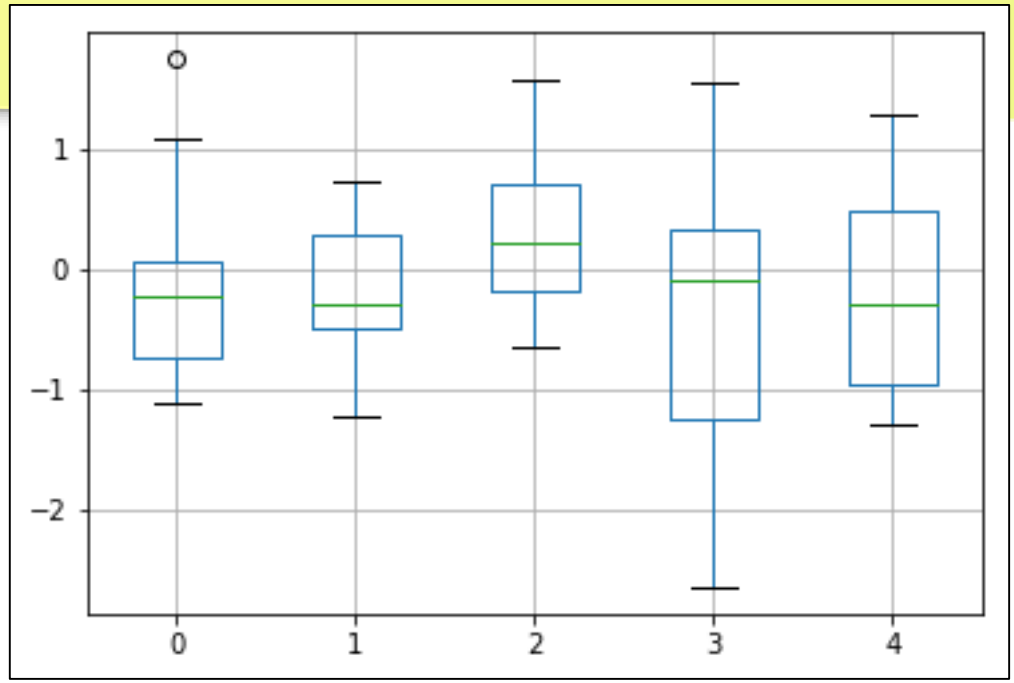
Pandas - Histograms

```
np.random.seed(10)
dfh = pd.DataFrame(np.random.randn(1000, 4),
                   columns=['a', 'b', 'c', 'd'])
# draw the chart.  There are four columns so pandas draws
# four histograms
dfh.hist()
```



Pandas - Boxplots

```
np.random.seed(11)
dfb = pd.DataFrame(np.random.randn(10,5))
# generate the plot
dfb.boxplot()
```

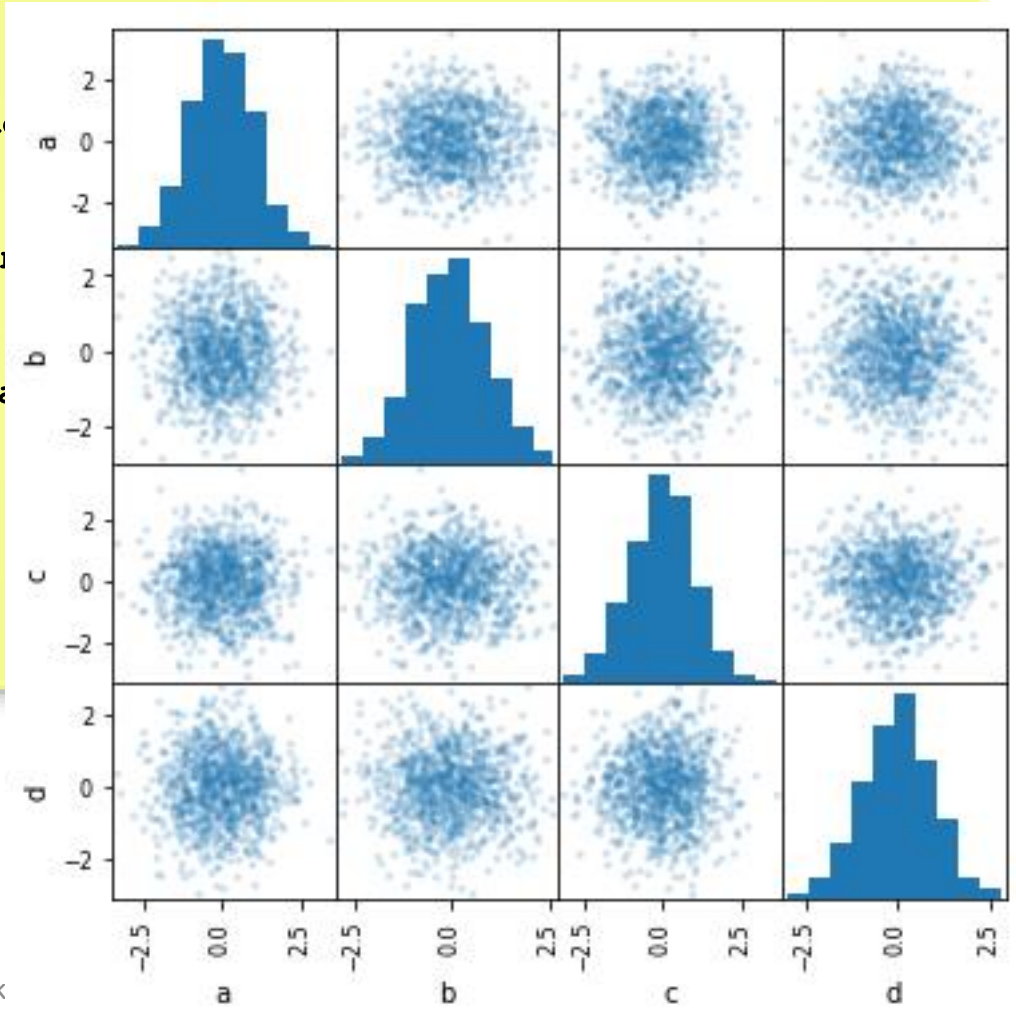


Pandas – Scatter Plot Matrix

```
# create a scatter plot matrix
# import this class
from pandas.plotting import scatter_matrix

# generate DataFrame with 4 columns of 1000 random numbers
np.random.seed(10)
df_spm = pd.DataFrame(np.random.randn(1000, 4),
                      columns=['a', 'b', 'c', 'd'])
# create the scatter matrix
scatter_matrix(df_spm, alpha=0.2, figsize=(6, 6));
```

Pandas – Scatter Plot Matrix

```
# create a scatter plot matrix
# import this class
from pandas.plotting import scatt

# generate DataFrame with 4 colum
np.random.seed(10)
df_spm = pd.DataFrame(np.random.ra
                      columns=['a
# create the scatter matrix
scatter_matrix(df_spm, alpha=0.2,
```



Seaborn

- Matplotlib is a powerful library but is sometimes **unwieldy**. Therefore, constructing attractive plots can take a significant investment of effort and time.
- Seaborn is a Python visualization library which is also based on Matplotlib.
- It provides a high-level interface for rendering attractive statistical graphics.
- It has support for Pandas and NumPy data structures.
- The goal of Seaborn is to create Matplotlib graphs that look more professional and are easier to create.

Seaborn

- Apart from Pandas, the Seaborn library is one of the most **popular** in the Python data science community to create visualizations.

- Like pandas, it does not do any actual plotting itself and is completely reliant on **Matplotlib** for the heavy lifting.

Using Seaborn

- To illustrate the use of Seaborn we will use a dataset that contains information on all employees that are employed by the city of Houston.

```
import pandas as pd
import seaborn as sns


employee = pd.read_csv("employee.csv")
print (employee.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 10 columns):
UNIQUE_ID            2000 non-null int64
POSITION_TITLE       2000 non-null object
DEPARTMENT           2000 non-null object
BASE_SALARY          1886 non-null float64
RACE                 1965 non-null object
EMPLOYMENT_TYPE      2000 non-null object
GENDER               2000 non-null object
EMPLOYMENT_STATUS    2000 non-null object
HIRE_DATE            2000 non-null object
JOB_DATE             1997 non-null object
dtypes: float64(1), int64(1), object(8)
memory usage: 156.3+ KB
None
```

Using Seaborn

- To illustrate the use of Seaborn we will use an employee.csv data file.

	UNIQUE_ID	POSITION_TITLE	DEPARTMENT	BASE_SALARY	RACE	EMPLOYMENT_TYPE	GENDER	EMPLOYMENT_STATUS	HIRE_DATE	JOB_DATE
0	0	ASSISTANT DIRECTOR (EX LVL)	Municipal Courts Department	121862.0	Hispanic/Latino	Full Time	Female	Active	2006-06-12	2012-10-13
1	1	LIBRARY ASSISTANT	Library	26125.0	Hispanic/Latino	Full Time	Female	Active	2000-07-19	2010-09-18
2	2	POLICE OFFICER	Houston Police Department-HPD	45279.0	White	Full Time	Male	Active	2015-02-03	2015-02-03
3	3	ENGINEER/OPERATOR	Houston Fire Department (HFD)	63166.0	White	Full Time	Male	Active	1982-02-08	1991-05-25
4	4	ELECTRICIAN	General Services Department	56347.0	White	Full Time	Male	Active	1989-06-19	1994-10-22

None

Seaborn

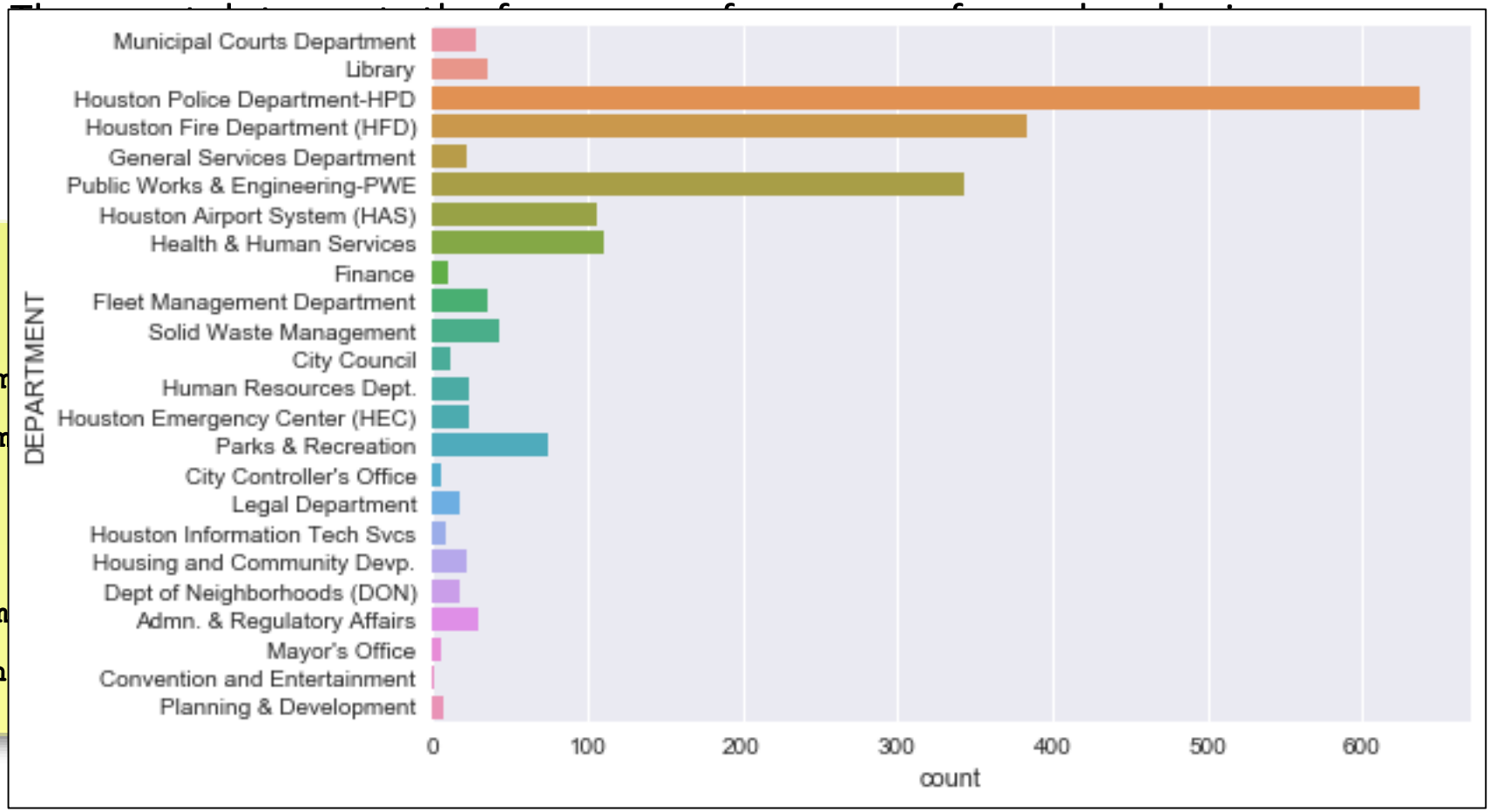
- Typically when using Seaborn you provide it with a **DataFrame** as a 'data' argument and you select **specific columns** from the data to incorporate into your graphical plot.

- The **countplot** counts the frequency of occurrence for each value in a **categorical** feature (column) and depicts it as a horizontal bar graph.
- A count plot can be thought of as a histogram across a categorical, instead of quantitative, variable.
- All seaborn plotting functions have x and y parameters. We could have made a vertical bar plot using x instead of y.

```
import pandas as pd
import seaborn as sns

employee = pd.read_csv("employee.csv")
sns.countplot(y='DEPARTMENT', data=employee)
```

Seaborn

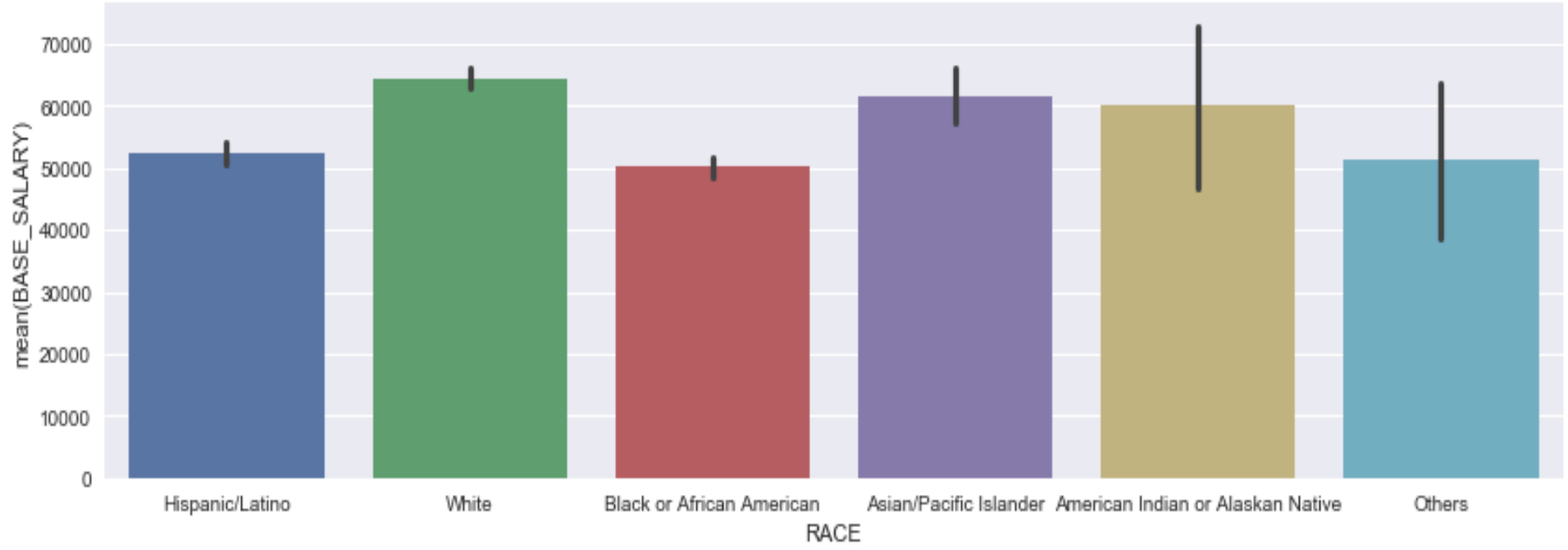
- Typically when using Seaborn you provide it with a DataFrame as a 'data' argument and you select specific columns from the data to incorporate into your graphical plot.
-
-



Seaborn - Barplots

- The bar show us the mean of a numerical variable.
- In this example we use a barplot to find the average salary for each race in our employee dataset.

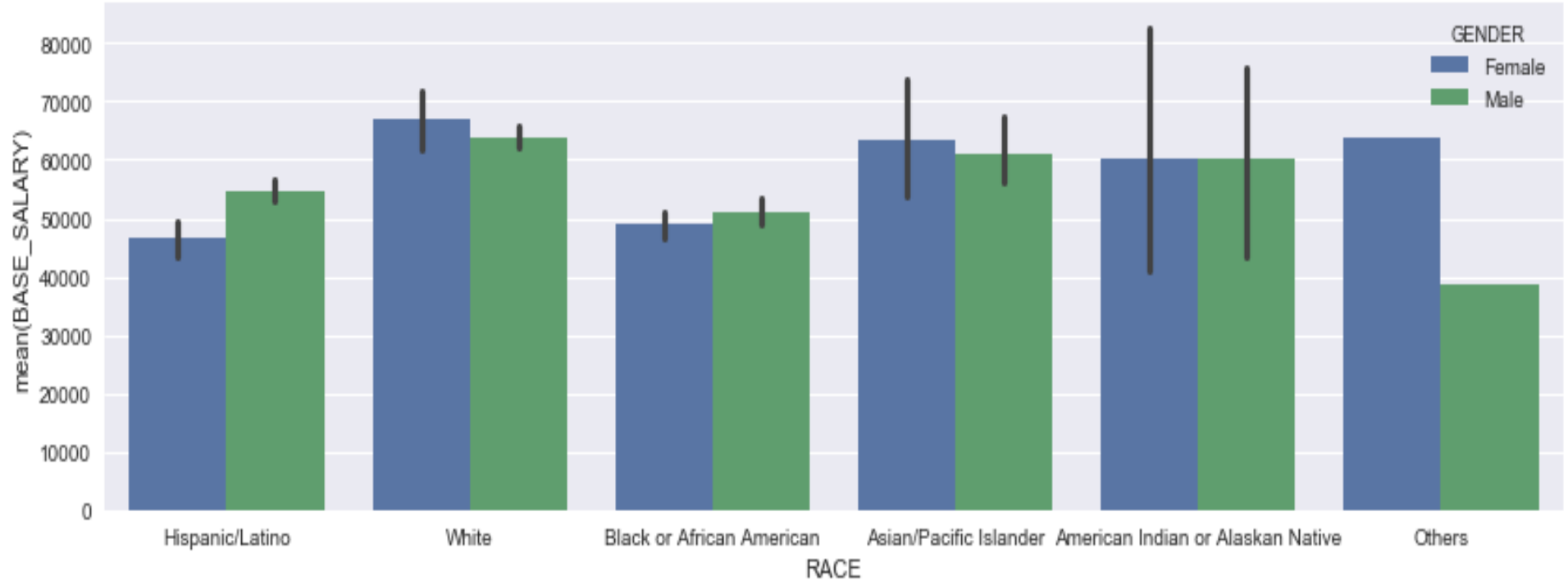
```
ax = sns.barplot(x='RACE', y='BASE_SALARY', data=employee)
ax.figure.set_size_inches(14, 4)
```



Seaborn – Barplots – Hue Parameter

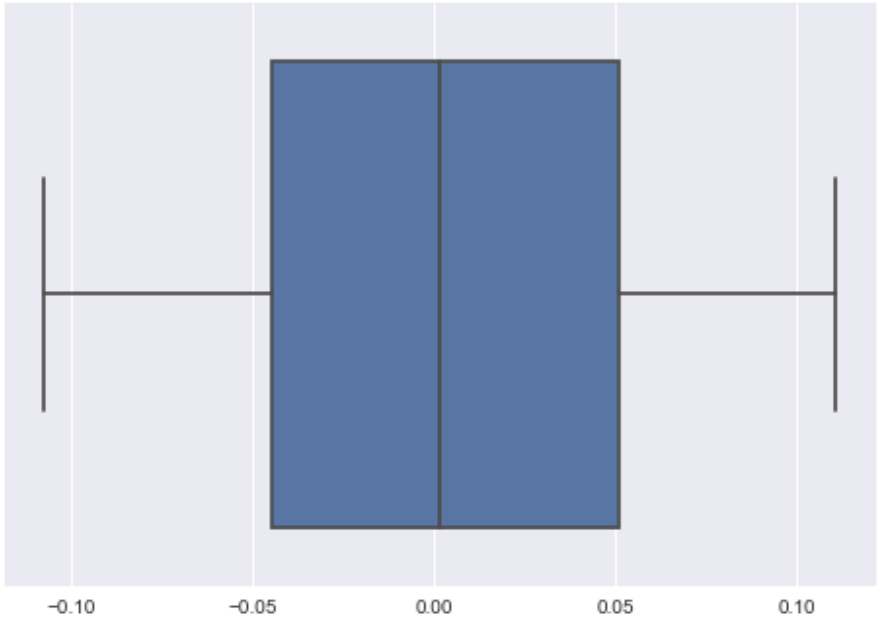
- Seaborn also has the ability to distinguish groups within the data through a third parameter, hue, in most of its plotting functions (The **hue** parameter further splits each of the groups on the *x* axis.).
- Let's find the mean salary by race and gender:

```
ax = sns.barplot(x='RACE', y='BASE_SALARY', data=employee, hue='GENDER')
```



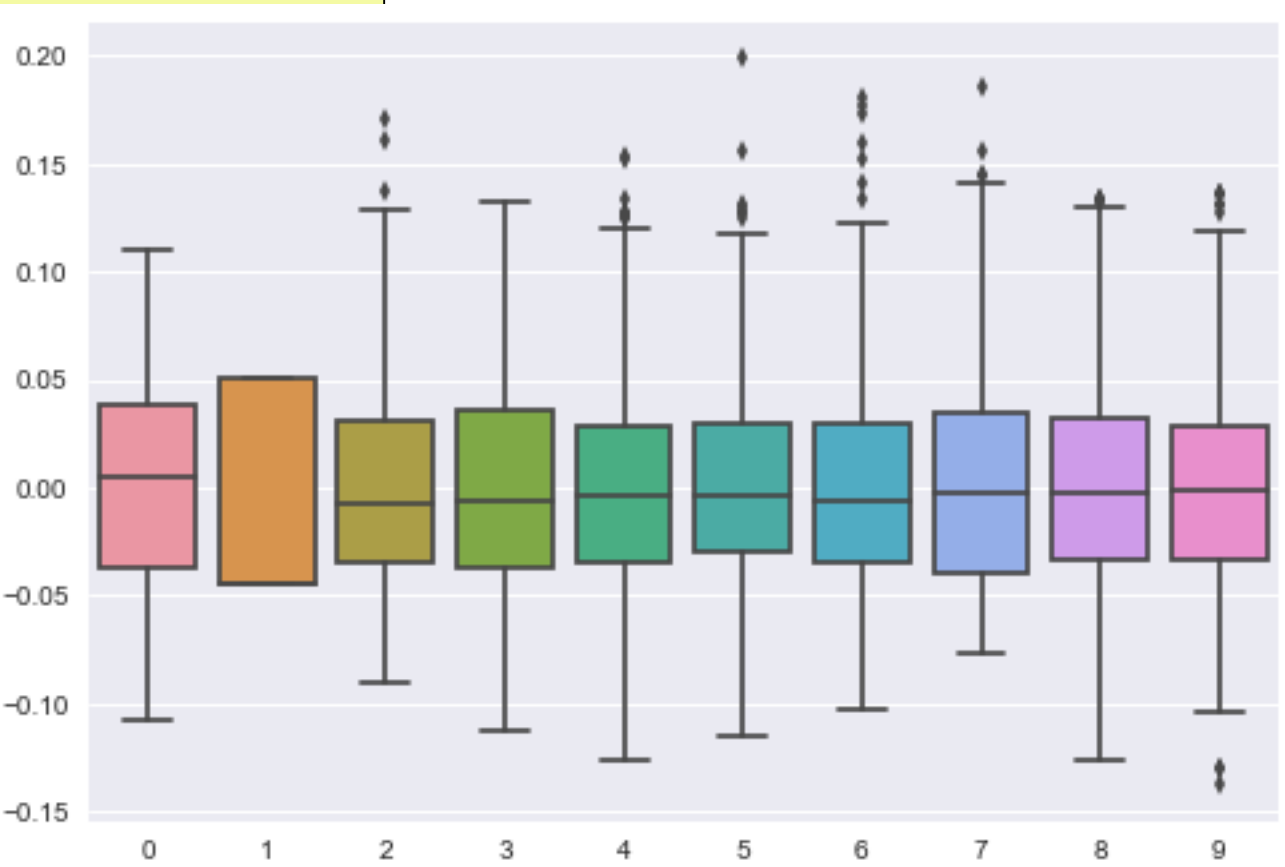
Seaborn - Boxplots

A box plot is another type of plot that seaborn and pandas have in common. Notice, in the example, we have a really simple boxplot where we just want to plot for a single feature (we can directly pass the feature using the x parameter)




```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import datasets

diab = datasets.load_diabetes()
X = diab.data
y = diab.target

sns.boxplot(x=X[:, 1])
plt.show()
```

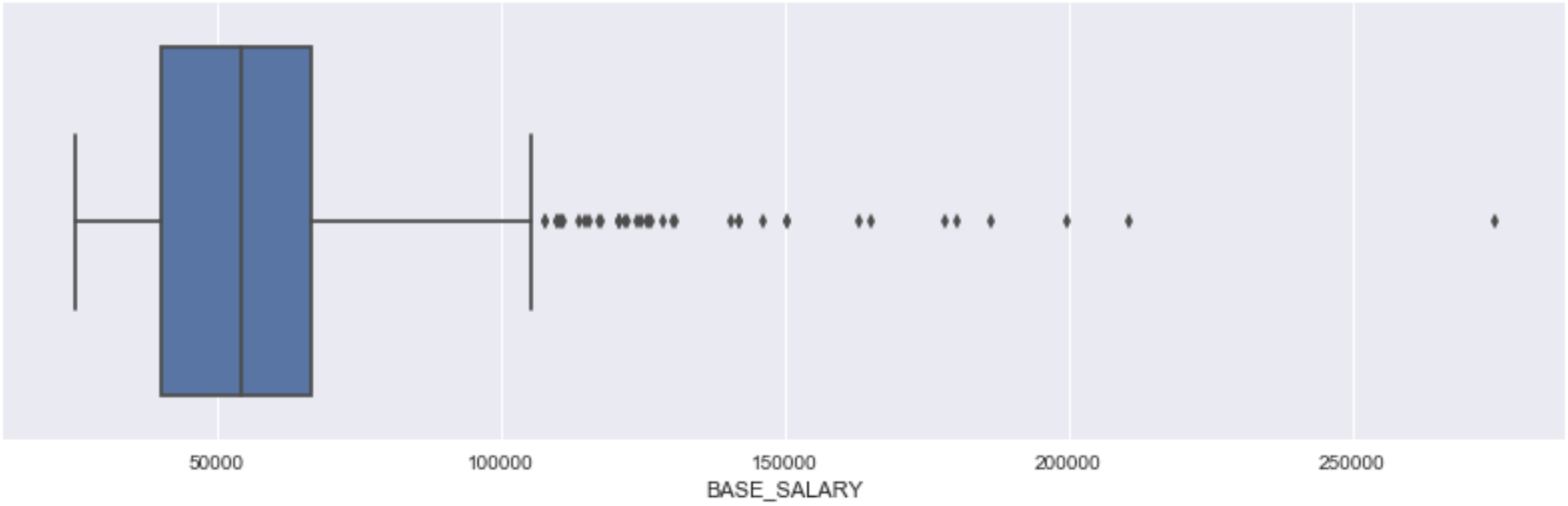
```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import datasets

diab = datasets.load_diabetes()
X = diab.data
y = diab.target

sns.boxplot(data= pd.DataFrame(X))
plt.show()
```



Boxplots

- We can also perform more complex boxplots by specifying a y parameter and even a hue parameter. In the example below we will create a box plot of salary by race and gender with Seaborn

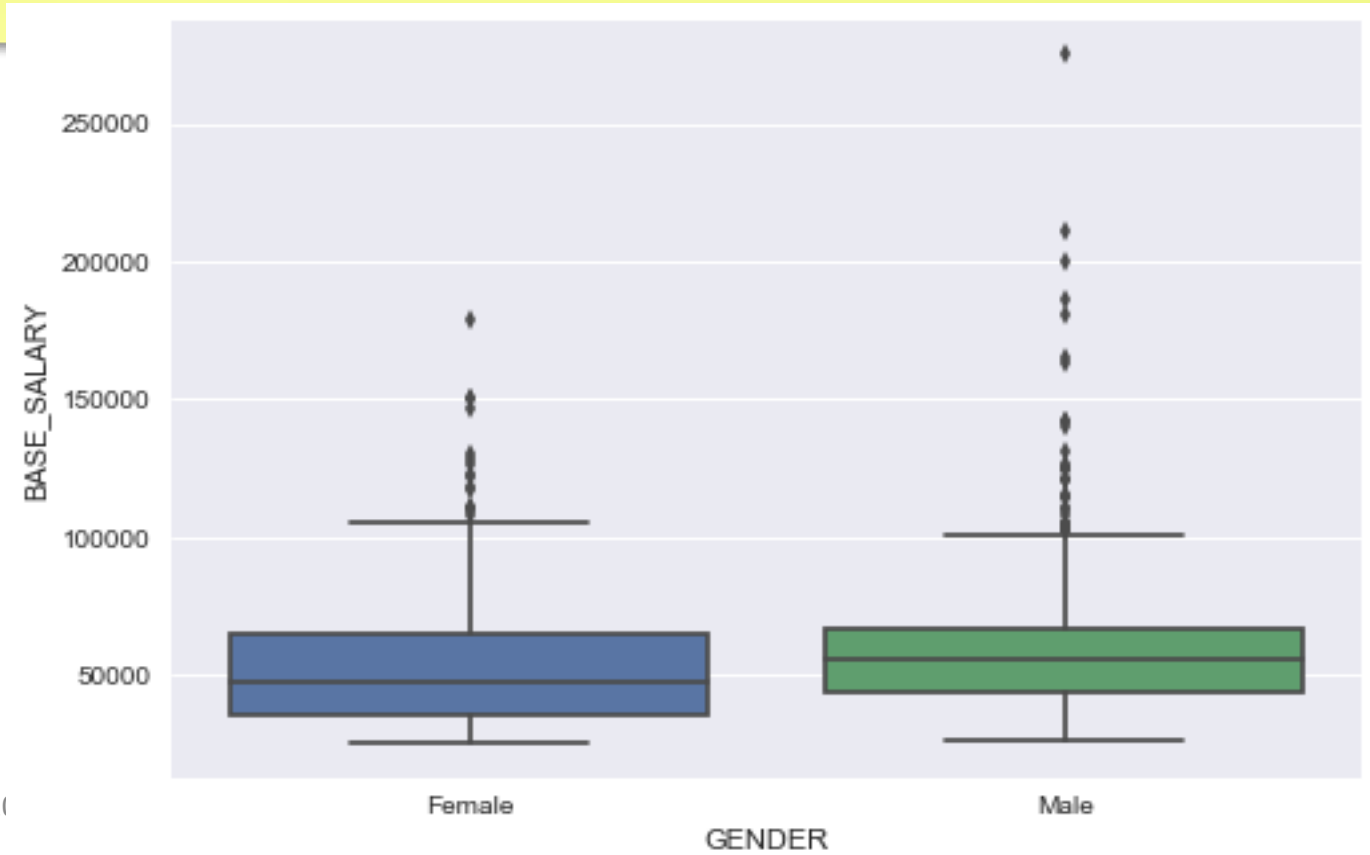
```
ax = sns.boxplot(x='BASE_SALARY', data=employee)
ax.figure.set_size_inches(14, 4)
```



Boxplots

- We can also perform more complex boxplots by specifying a y parameter and even a hue parameter. In the example below we will create a box plot of salary by race and gender with Seaborn

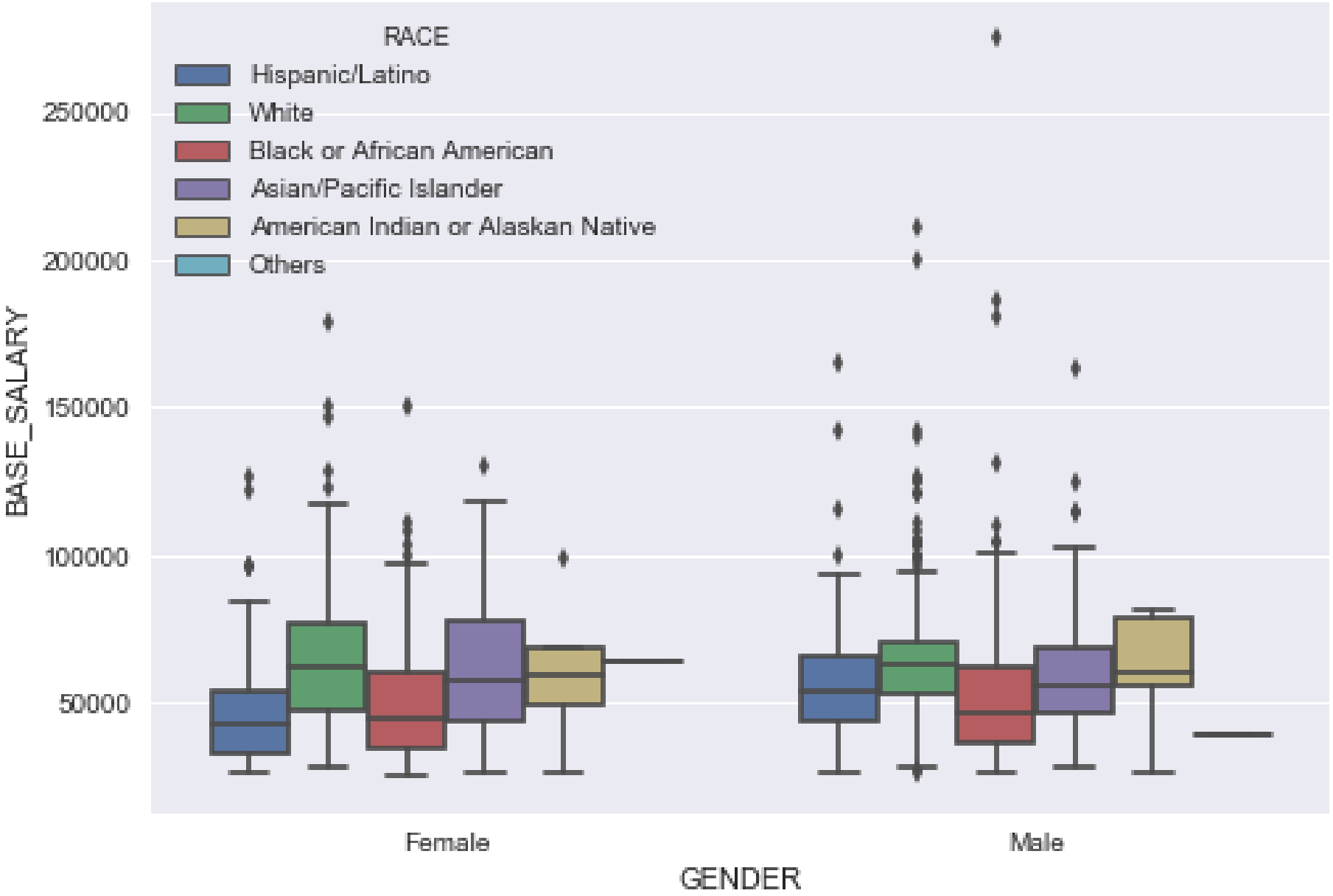
```
ax = sns.boxplot(x='GENDER', y='BASE_SALARY', data=employee)
ax.figure.set_size_inches(14, 4)
```



Boxplots

- We can also perform more complex boxplots by specifying a y parameter and even a hue parameter. In the example below we will create a box plot of salary by race and gender with Seaborn

```
ax = sns.boxplot(x='GENDER', y='BASE_SALARY', data=employee, hue='RACE')
ax.figure.set_size_inches(14, 4)
```


RACE
Hispanic/Latino
White
Black or African American
Asian/Pacific Islander
American Indian or Alaskan Native
Others
250000
200000
150000
100000
50000
BASE_SALARY
Female
Male
GENDER

lmplots

- The lmplot function in seaborn are used to visualize a **linear relationship** as determined through linear regression.
- In the simplest invocation, this function draws a scatterplot of two variables, x and y, and then fits a regression model to the data.
- Before, we illustrate its use we will create a new column in the Employee data that will contains the number of years that each employee has worked.

```
import pandas as pd
import seaborn as sns

employee = pd.read_csv("employee.csv", parse_dates=['HIRE_DATE',
'JOB_DATE'])

days_hired = pd.to_datetime('12-1-2016') - employee['HIRE_DATE']

one_year = pd.Timedelta(1, unit='Y')
employee['YEARS_EXPERIENCE'] = days_hired / one_year

print (employee[['HIRE_DATE', 'YEARS_EXPERIENCE']].head())
```

	HIRE_DATE	YEARS_EXPERIENCE
0	2006-06-12	10.472494
1	2000-07-19	16.369946
2	2015-02-03	1.826184
3	1982-02-08	34.812488
4	1989-06-19	27.452994

lmplots

- You can see from below that that an lm plot plots one variable against another and try to fit a linear regression line. We can see from the linear regression line that there is a gradual increase in base salary as the number of years of experience increases.

```
grid = sns.lmplot(x='YEARS_EXPERIENCE', y='BASE_SALARY',data=employee)
grid.fig.set_size_inches(14, 4)
```

lmplots

- It is possible to change use parameters from the underlying line and scatter plot using matplotlib functions. To do so, set the scatter_kws or the line_kws parameters equal to a dictionary that has the matplotlib parameter as a string paired to the value you want it to be.

```
grid = sns.lmplot(x='YEARS_EXPERIENCE', y='BASE_SALARY',data=employee,
line_kws={"color":"g","alpha":0.5,"lw":8}, scatter_kws={'s':8,
'color':'Red'} )


grid.fig.set_size_inches(14, 4)
```

lmplots

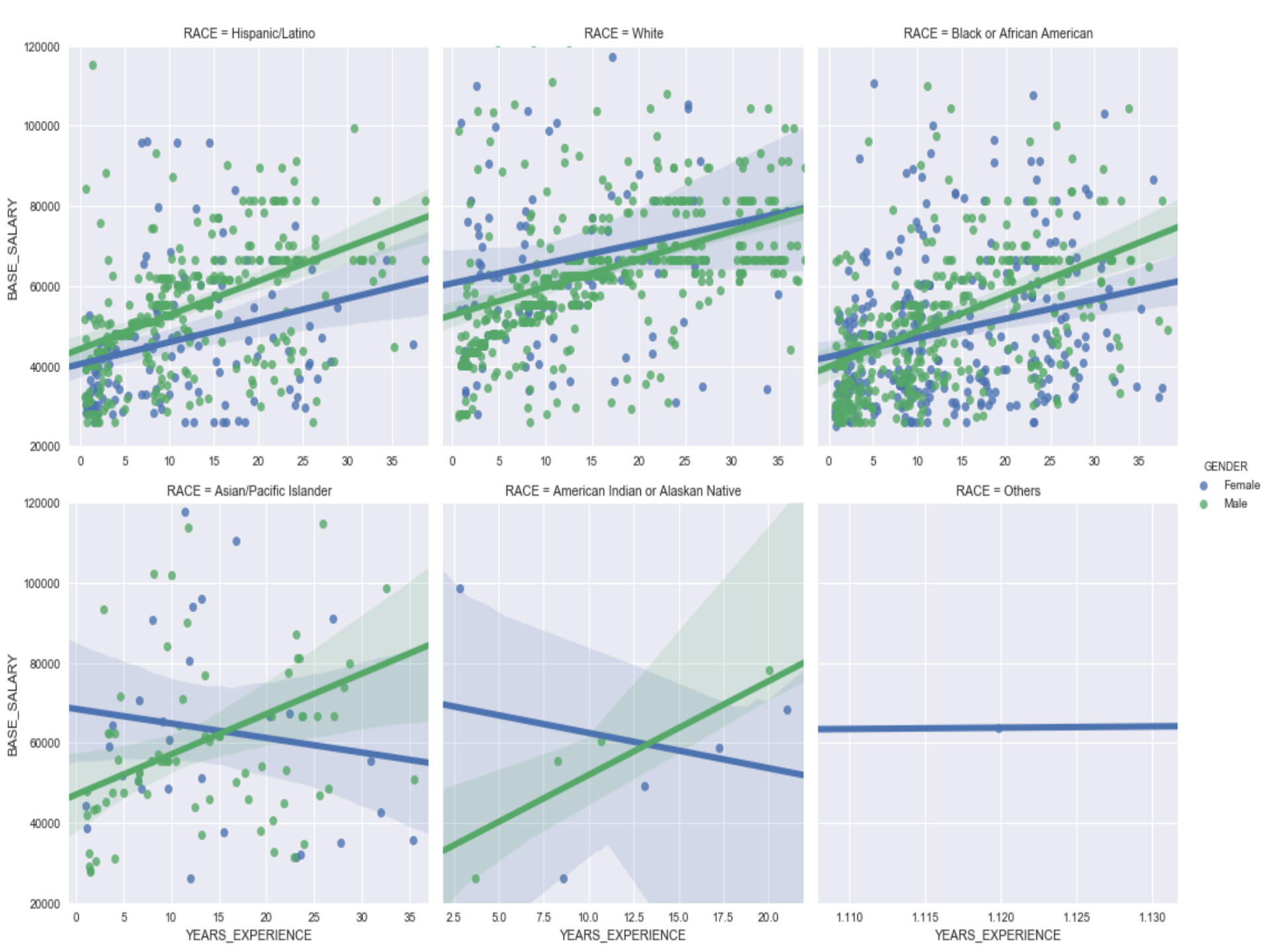
- Notice using lmplots we can also specify a hue that will allow to us to subdivide the results in additional groups. Notice a regression line is drawn for each subgroup.

```
grid = sns.lmplot(x='YEARS_EXPERIENCE', y='BASE_SALARY',
                  hue='GENDER',
                  scatter_kws={'s':8, 'color':'Red'},
                   line_kws={"alpha":0.5,"lw":4},data=employee)
grid.fig.set_size_inches(14, 4)
```

lmplots

- The lmplot has the col parameter that allows us to divide the data further into different groups. For instance, we can create a separate plot for each unique race in the dataset and still fit the regression lines by gender

```
grid = sns.lmplot(x='YEARS_EXPERIENCE', y='BASE_SALARY',
                  hue='GENDER', col='RACE', col_wrap=3,
                   sharex=False,
                  line_kws = {'linewidth':5},
                  data=employee)
grid.set(ylim=(20000, 120000))
```


RACE = Hispanic/Latino
RACE = White
RACE = Black or African American
RACE = Asian/Pacific Islander
RACE = American Indian or Alaskan Native
RACE = Others
BASE_SALARY
YEARS_EXPERIENCE
GENDER
Female
Male

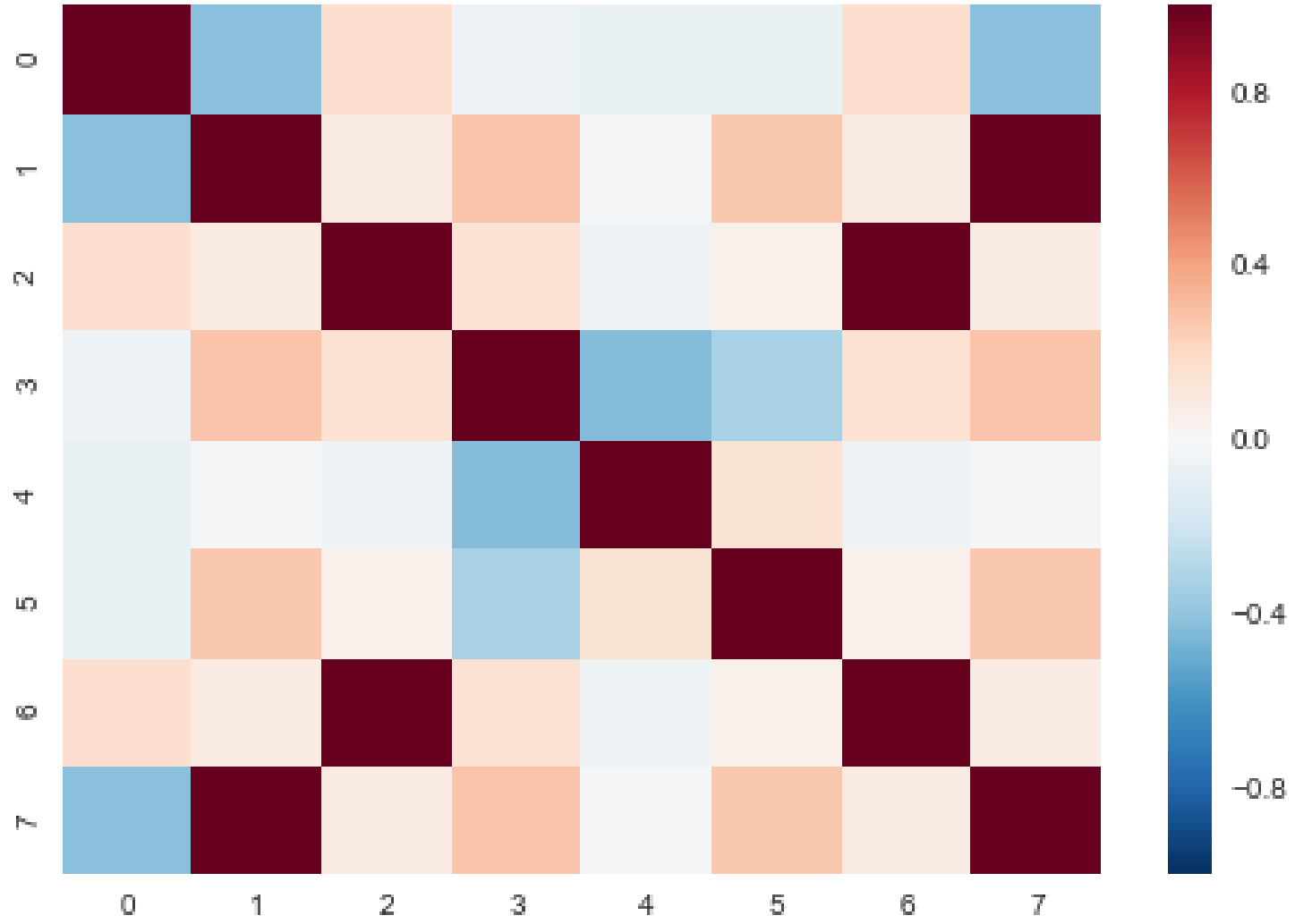
Correlation Matrix

- A correlation matrix is a simple and useful method for examining the correlations between various features in your dataset.

- There are a number of ways of plotting this information but typically you can use the **corr()** function available in a Pandas dataframe object to calculate the correlations and then use Seaborn to create a **heatmap**.

- A **jointplot** is also a useful way of examining the relationship between two variables as it will produce a scatter plot and report the exact Pearson's coefficient values.

```
from sklearn import model_selection
from sklearn.datasets import make_classification
import seaborn as sns
import matplotlib.pyplot as plt

X, y = make_classification(n_samples=500, n_features=8, n_informative=6,
    n_redundant=0, n_repeated=2, n_classes=2, random_state=0,shuffle=False)

df = pd.DataFrame(X)
corrResults = df.corr()

sns.heatmap(corrResults)
plt.show()
```


0
1
2
3
4
5
6
7
0
1
2
3
4
5
6
7
0.8
0.4
0.0
−0.4
−0.8

```
from sklearn import model_selection
from sklearn.datasets import make_classification
import seaborn as sns
import matplotlib.pyplot as plt

X, y = make_classification(n_samples=500, n_features=8, n_informative=6,
    n_redundant=0, n_repeated=2, n_classes=2, random_state=0,shuffle=False)

df = pd.DataFrame(X)
corrResults = df.corr()

sns.heatmap(corrResults)
plt.show()

sns.jointplot(df.iloc[:,2].values, df.iloc[:,6].values)
plt.show()
```

Correlation Matrix

- Joint distribution plots combine information from scatter plots and histograms to give you detailed information for bi-variate distributions and is a useful way of examining the relationship between two variables.