Data Mining

Tamás Budavári - budavari@jhu.edu
Class 1

- Python and Jupyter Notebook (previously IPython)
- Location, dispersion, shape
- PDF, CDF, moments

IPython/Jupyter Notebook

Interactive Data Analysis

Load required modules

numpy is the fundamental package for scientific computing with Python.

matplotlib.pyplot is a state-based interface to matplotlib. It provides a MATLAB-like way of plotting. pyplot is mainly intended for interactive plots and simple cases of programmatic plot generation:

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

The **import** comment allows us to use the **numpy** and **matplotlib.pyplot** libraries throughout this notebook.

The **as** comment allows us to simply use **np** as the alias of **numpy** and **plt** as the alias of **matplotlib.pyplot**.

Magic function

Another way to load numpy and matplotlib to work interactively. The following command lets you activate pylab (including matplotlib, numpy and interactive support) at any point during an IPython session.

The symbol **%** is a syntax element for the Magic functions in the Python notebook.

See https://ipython.readthedocs.io/en/stable/interactive/magics.html (https://ipython.readthedocs.io/en/stable/interactive/magics.html) for more details about the Magic functions.

In [2]:
```
%pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

- This is what `%pylab` will do

```
import numpy
import matplotlib
from matplotlib import pylab, mlab, pyplot
np = numpy
plt = pyplot

from IPython.display import display
from IPython.core.pylabtools import figsize, getfigs

from pylab import *
from numpy import *
```

Generate data from common distributions

numpy.random (or np.random) is the main library for doing random sampling in Python.

See https://docs.scipy.org/doc/numpy-1.15.1/reference/routines.random.html (https://docs.scipy.org/doc/numpy-1.15.1/reference/routines.random.html) for a complete guide.

For a few examples:

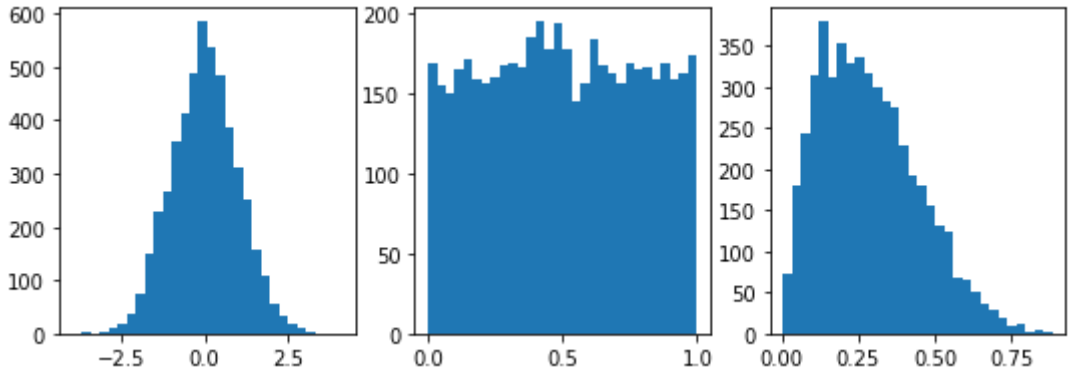
randn(d0, d1, …, dn): Return a sample (or samples) from the “standard normal” distribution.

random([size]): Return random floats in the half-open interval [0.0, 1.0).

beta(a, b[, size]): Draw samples from a Beta distribution.

In [3]:
```
N = 5000;
x_normal = np.random.randn(N) # standard normal distribution
x_random = np.random.random(N) # uniform between 0 and 1
x_beta = np.random.beta(2,5,N) # beta(2,5)

plt.figure(1, figsize=(9, 3))
plt.subplot(131)
plt.hist(x_normal, bins = 30)
plt.subplot(132)
plt.hist(x_random, bins = 30)
plt.subplot(133)
plt.hist(x_beta, bins = 30)
plt.show()
```



Indexing, Slicing and Iterating

In Python, all indices are **zero-based**!

In [4]:
```
N = 10
x = np.arange(N) # retuns a list of numbers starting from 0
x
```

Out[4]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Get the **i-th** element from the array

In [5]:
```
print('The first element is:', x[0])
print('The third element is:', x[2])
print('The last element is %f = %f:' % (x[N-1], x[-1]))
```

```
The first element is: 0
The third element is: 2
The last element is 9.000000 = 9.000000:
```

In [6]:
```
print("Show every element: %f, %f, ..., %f" % (x[0], x[1], x[N-1]))
print("Show the first five elements:", x[0:5]) # Note: the last digit is excluded, x[5] = 5
print("Show every other element:", x[0:N:2])
print("Reverse the array:", x[::-1])
```

```
Show every element: 0.000000, 1.000000, ..., 9.000000
Show the first five elements: [0 1 2 3 4]
Show every other element: [0 2 4 6 8]
Reverse the array: [9 8 7 6 5 4 3 2 1 0]
```

In [7]:
```
# iterating from 0 to 4
[i*i for i in range(5)]
```

Out[7]:
```
[0, 1, 4, 9, 16]
```

In [8]:
```
# Index out of bounds
for i in range(N-3,N+1):
    print ("%d : \t %r" % (i, x[i]))
```

```
7 :      7
8 :      8
9 :      9

---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-8-6c1d0f137a6e> in <module>()
      1 # Index out of bounds
      2 for i in range(N-3,N+1):
----> 3     print ("%d : \t %r" % (i, x[i]))

IndexError: index 10 is out of bounds for axis 0 with size 10
```

Error handling with exceptions

The try and except block in Python is used to catch and handle exceptions. Python executes code following the try statement as a “normal” part of the program. The code that follows the except statement is the program’s response to any exceptions in the preceding try clause.

In [9]:
```
for i in range(N-3,N+5):
    try:
        print ("%d : \t %r" % (i, x[i]))
    except IndexError as err:
        print (err)
```

```
7 :      7
8 :      8
9 :      9
index 10 is out of bounds for axis 0 with size 10
index 11 is out of bounds for axis 0 with size 10
index 12 is out of bounds for axis 0 with size 10
index 13 is out of bounds for axis 0 with size 10
index 14 is out of bounds for axis 0 with size 10
```

Other common errors include:

IOError: If the file cannot be opened.

ImportError: If python cannot find the module.

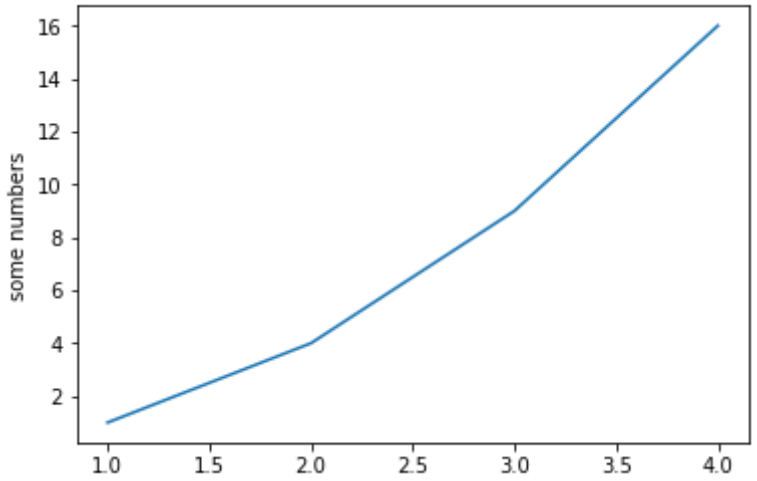
ValueError: Raised when a built-in operation or function receives an argument that has the right type but an inappropriate value.

KeyboardInterrupt: Raised when the user hits the interrupt key (normally Control-C or Delete).

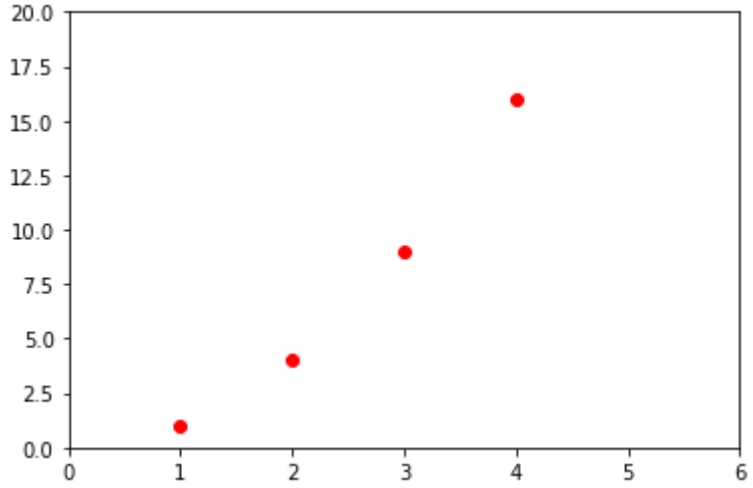
EOFError: Raised when one of the built-in functions (input() or raw_input()) hits an end-of-file condition (EOF) without reading any data.

Plotting

In [10]:
```
# Plot a line
plt.plot([1, 2, 3, 4], [1, 4, 9, 16])
plt.ylabel('some numbers')
plt.show()
```

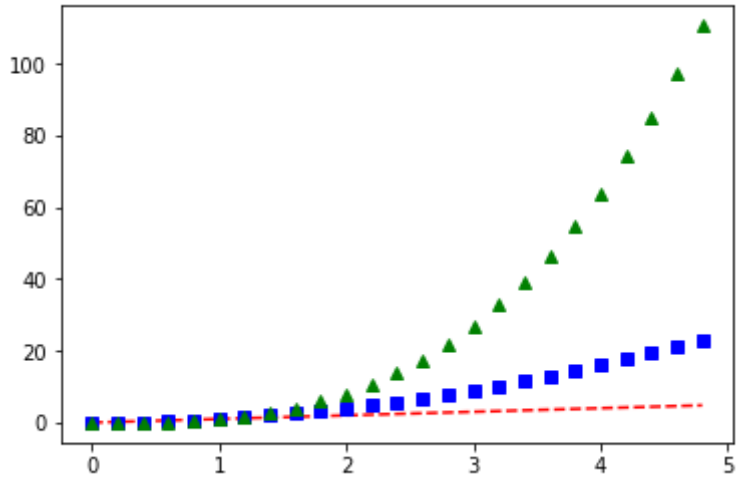


In [11]:
```
# Scatter Plot
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'ro') # The letters and symbols of the format string are from M
ATLAB
plt.axis([0, 6, 0, 20]) # x-axis is from 0 to 6, y-axis is from 0 to 20
plt.show()
```

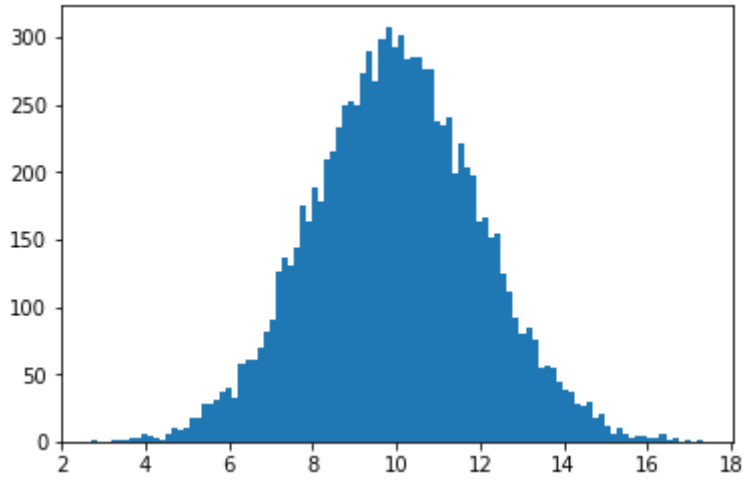


In [12]:
```
# Plot multiple arrays
# evenly sampled time at 200ms intervals
t = np.arange(0., 5., 0.2)

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



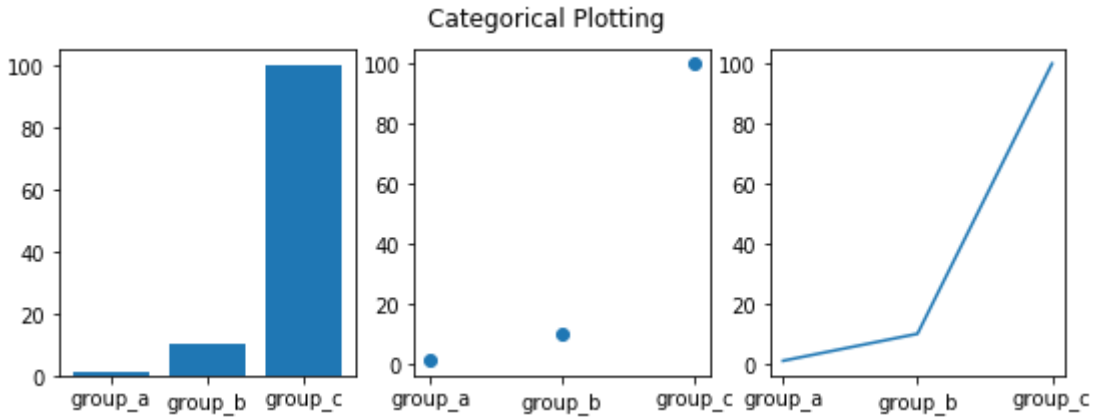
In [13]:
```
# Histogram
x = np.random.randn(10000) * 2 + 10 # normal distribution with mean 10 and variance 4
h = plt.hist(x, bins=100)
```



In [14]:
```
# Plot multiple figures together
names = ['group_a', 'group_b', 'group_c']
values = [1, 10, 100]

plt.figure(1, figsize=(9, 3))

plt.subplot(131)
plt.bar(names, values)
plt.subplot(132)
plt.scatter(names, values)
plt.subplot(133)
plt.plot(names, values)
plt.suptitle('Categorical Plotting')
plt.show()
```



A Little Bit More

- Markdown (http://jupyter-notebook.readthedocs.io/en/stable/examples/Notebook/Working%20With%20Markdown%20Cells.html)
- Built-in magic commands (https://ipython.readthedocs.io/en/stable/interactive/magics.html)
- Tips, Tricks, and Shortcuts (https://www.dataquest.io/blog/jupyter-notebook-tips-tricks-shortcuts/)
- numpy Tutorial (https://docs.scipy.org/doc/numpy-1.15.0/user/quickstart.html)
- scipy Tutorial (https://docs.scipy.org/doc/scipy/reference/tutorial/index.html)
- matplotlib Tutorial (https://matplotlib.org/tutorials/index.html)

In [15]:
```
from IPython.display import YouTubeVideo
YouTubeVideo('-F4WS8o-G2A', width=1000, height=600)
```

Out[15]:

Descriptive Statistics

Data Sets

- For example, a set of N scalar measurements

> $\{x_i\}_{i=1}^{N}$

How to characterize the data?

- Location
- Dispersion
- Shape?

Location

- Mode

> where it peaks
>
> unimodal vs multimodal

- Sample average

> $$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$$
>
> but indexing starts with 0 in Python and most computer languages
>
> $$\bar{x} = \frac{1}{N}\sum_{i=0}^{N-1} x_i$$

- Median

> The number that separates the higher half of the set from the lower half

In [16]:
```
# Generate Data
N = 1000;
x = np.random.randn(N);
```

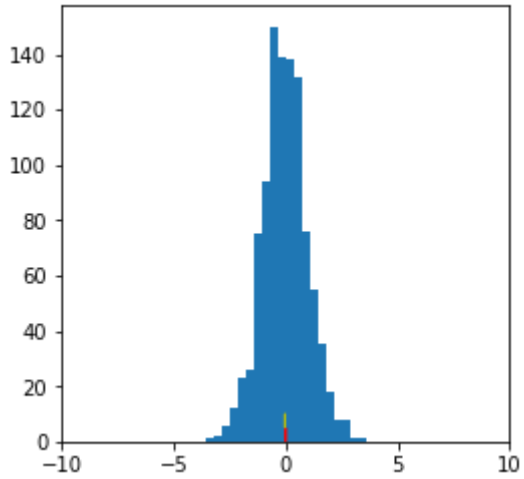
In [17]:
```
# Mean
avg = np.sum(x) / N
avg, np.mean(x)
```

Out[17]: (-0.0070994549819674525, -0.0070994549819674525)

In [18]:
```
# Median
med = np.median(x)
med
```

Out[18]: -0.034413619490687775

In [19]:
```
# Plot
fig = plt.figure(figsize=(4,4))
ax = fig.add_subplot(1,1,1)
ax.hist(x,20)
ax.arrow(avg,0,0,5,color='r')
ax.arrow(med,5,0,5,color='y')
ax.set_xlim(-10, 10);
```



Dispersion

- Sample variance

$$s^2 = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2$$

- Standard deviation

$$s = \sqrt{s^2}$$

Unhomework

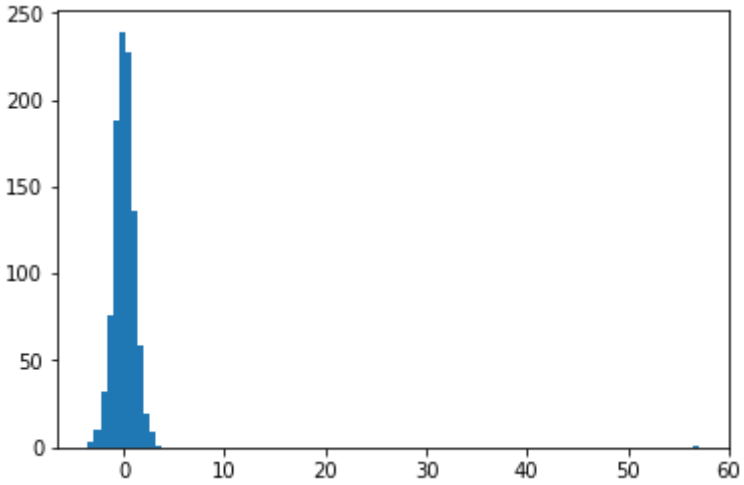
1. Why is $(N-1)$ in the denominator above?

Outliers

- What if just one element is too large, e.g., erroneously becomes $+\infty$
- Sample average $\bar{x} \to +\infty$
- Sample variance explodes, too

Ouch !!

In [20]:
```
# Add one outlier and visulization
x[0] = 57
hist(x,100);
```



Robustness

- Robust against outliers? What fraction can we tolerate?
- Median is more robust than the mean
- Median Absolute Deviation (MAD) for dispersion

In [21]:
```
# Add one outlier and compare the robustness of mean and median
x[0] = -1e9
print ('Average old vs new: %f %f' % (avg, mean(x)))
print ('Median  old vs new: %f %f' % (med, median(x)))
```

```
Average old vs new: -0.007099 -1000000.007037
Median  old vs new: -0.034414 -0.034414
```