PRACTICAL 1

Fundamental of python

- Python is a general purpose, dynamic, high level and interpreted programming language. It supports Object Oriented programming approach to develop applications. It is simple and easy to learn and provides lots of high-level data structures.
- Python is easy to learn yet powerful and versatile scripting language which makes it attractive for Application Development.
- Python's syntax and dynamic typing with its interpreted nature, makes it an ideal language for scripting and rapid application development.
- Python supports multiple programming pattern, including object oriented, imperative and functional or procedural programming styles.
- Python is not intended to work on special area such as web programming. That is why it is known as multipurpose because it can be used with web, enterprise, 3D CAD etc.
- We don't need to use data types to declare variable because it is dynamically typed so we can write a=10 to assign an integer value in an integer variable.
- Python makes the development and debugging fast because there is no compilation step included in python development and edit-test-debug cycle is very fast.
- Python provides lots of features that are listed below.

1. **<u>Easy to Learn and Use</u>**
   Python is easy to learn and use. It is developer-friendly and high level programming language.

2. **<u>Expressive Language</u>**
   Python language is more expressive means that it is more understandable and readable.

3. **<u>Interpreted Language</u>**
   Python is an interpreted language i.e. interpreter executes the code line by line at a time. This makes debugging easy and thus suitable for beginners.

4. **Cross-platform Language**
   Python can run equally on different platforms such as Windows, Linux, Unix and Macintosh etc. So, we can say that Python is a portable language.

5. **Free and Open Source**
   Python language is freely available at offical web address.The source-code is also available. Therefore it is open source.

6. **Object-Oriented Language**
   Python supports object oriented language and concepts of classes and objects come into existence.

7. **Extensible**
   It implies that other languages such as C/C++ can be used to compile the code and thus it can be used further in our python code.

8. **Large Standard Library**
   Python has a large and broad library and prvides rich set of module and functions for rapid application development.

9. **GUI Programming Support**
   Graphical user interfaces can be developed using Python.

10. **Integrated**
    It can be easily integrated with languages like C, C++, JAVA etc.

- **Printing a string**

```
print("Hello, Python!")
```

OUTPUT:

```
Hello, Python!
```

- **Defining a variable**

```
var1 = 10      # An integer assignment
var2 = 3.146   # A floating point
var3 = "Hello" # A string
print(var1,' ',var2,' ',var3)
```

OUTPUT:

```
10   3.146   Hello
```

- **Assigning same value to multiple variables**

```
var1 = var2 = var3 = 1

print(var1,' ',var2,' ',var3)
```

OUTPUT:

1 1 1

- **Assigning Different values to variable in a single expression**

```
var1, var2, var3 = 1, 2.5, "john"

print(var1,' ',var2,' ',var3)
```

OUTPUT:

```
1   1   1
1   2.5   john
```

- **String operations**

```
str = 'Hello World!'  # A strin

print(str)          # Prints complete string

print(str[0])       # Prints first character of the string

print(str[2:5])     # Prints characters starting from 3rd to 5th

print(str[2:])      # Prints string starting from 3rd character

print(str[:2])

print(str * 2)      # Prints string two times

print(str + "TEST") # Prints concatenated string
```

OUTPUT:

```
Hello World!
H
llo
llo World!
He
Hello World!Hello World!
Hello World!TEST
```

- **Data types:**

```
list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]  # A list

tuple = ( 'abcd', 786 , 2.23, 'john', 70.2  ) # A tuple. Tuples are immutable, i.e. cannot be edit later

print(list)           # Prints complete list

print(list[0])        # Prints first element of the list

print(tuple[1:3])       # Prints elements starting from 2nd till 3rd
```

OUTPUT:

```
['abcd', 786, 2.23, 'john', 70.2]
abcd
(786, 2.23)
```

Dictionary:

```
tel = {'jack': 4098, 'sape': 4139}

tel['guido'] = 4127

print(tel)

print(tel['jack'])

del tel['sape']

tel['irv'] = 4127

print(tel)

print(tel.keys())

print(sorted(tel.keys()))

print(sorted(tel.values()))

print('guido' in tel)

print('jack' not in tel)
```

OUTPUT:

```
{'jack': 4098, 'sape': 4139, 'guido': 4127}
4098
{'jack': 4098, 'guido': 4127, 'irv': 4127}
dict_keys(['jack', 'guido', 'irv'])
['guido', 'irv', 'jack']
[4098, 4127, 4127]
True
False
```

- **Conditioning and looping:**

```
for i in range(0,10):

    if i%2 == 0:

        print("Square of ",i," is :",i)

    else:

        print(i,"is an odd number")
```

OUTPUT:

```
Square of  0  is : 0
1 is an odd number
Square of  2  is : 2
3 is an odd number
Square of  4  is : 4
5 is an odd number
Square of  6  is : 6
7 is an odd number
Square of  8  is : 8
9 is an odd number
```

- **Built-in Functions**

```
print("Sum of array: ",sum([1,2,3,4]))

print("Length of array: ",len([1,2,3,4]))

print("Absolute value: ",abs(-1234))

print("Round value: ",round(1.2234))



import math as mt      # importing a package

print("Log value: ",mt.log(10))
```

OUTPUT:

```
Sum of array:  10
Length of array:  4
Absolute value:  1234
Round value:  1
Log value:  2.302585092994046
```

Functions:

```
def area(length,width):
    return length*width
are = area(10,20)
print("Area of rectangle:",are)
```

OUTPUT:

Area of rectangle: 200

Different libraries in Python

1. NumPy

- NumPy (stands for Numerical Python).
- It provides an abundance of useful features for operations on n-arrays and matrices in Python.
- The library provides vectorization of mathematical operations on the NumPy array type, which ameliorates performance and accordingly speeds up the execution.
- np.mean(array,axis=0) will return mean along specific axis (0 or 1)
- array.sum() will return the sum of the array
- array.min()will return the minimum value of the array
- array.max(axis=0)will return the maximum value of specific axis
- np.var(array)will return the variance of the array
- np.std(array,axis=1)will return the standard deviation of specific axis
- array.corrcoef()will return the correlation coefficient of the array
- numpy.median(array) will return the median of the array elements

```
import numpy as np
list1 = [0,1,2,3,4]
arr1d = np.array(list1)
```

```
# shape
print('Shape: ', arr1d.shape)

# dtype
print('Datatype: ', arr1d.dtype)

# size
print('Size: ', arr1d.size)

# ndim
print('Num Dimensions: ', arr1d.ndim)
```

```
Shape:  (5,)
Datatype:  int32
Size:  5
Num Dimensions:  1
```

2. SciPy (Commits: 17213, Contributors: 489)

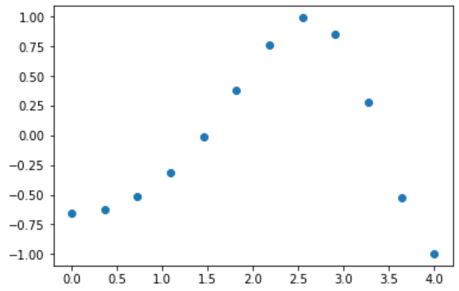
- SciPy is a library of software for engineering and science.
- The main functionality of SciPy library is built upon NumPy, and its arrays thus make substantial use of NumPy.
- It provides efficient numerical routines as numerical integration, optimization, and many others via its specific submodules.

- The functions in all submodules of SciPy are well documented—another coin in its pot.

```
import numpy as np
from scipy import interpolate
import matplotlib.pyplot as plt
x = np.linspace(0, 4, 12)
y = np.cos(x**2/3+4)
print (x,y)
```

```
[0.         0.36363636 0.72727273 1.09090909 1.45454545 1.81818182
 2.18181818 2.54545455 2.90909091 3.27272727 3.63636364 4.        ] [-0.65364362 -0.61966189 -0.51077021 -0.31047698
-0.00715476  0.37976236
  0.76715099  0.99239518  0.85886263  0.27994201 -0.52586509 -0.99582185]
```

```
plt.plot(x, y,'o')
plt.show()
```



3. Pandas (Commits: 15089, Contributors: 762)

- Pandas is a Python package designed to do work with "labeled" and "relational" data simple and intuitive.
- Pandas is a perfect tool for data wrangling.
- It designed for quick and easy data manipulation, aggregation, and visualization.

There are two main data structures in the library:

"Series"—one-dimensional

Series	
A	X0
B	X1
C	X2
D	X3

"Data Frames", two-dimensional

DataFrame				
	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

For example, when you want to receive a new Dataframe from these two types of structures, as a result you will receive such DF by appending a single row to a DataFrame by passing a Series:

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3
4	X0	X1	X2	X3

Here is just a small list of things that you can do with Pandas:

- Easily delete and add columns from DataFrame
- Convert data structures to DataFrame objects
- Handle missing data, represents as NaNs
- Powerful grouping by functionality

```
#create random sequence with numpy
import pandas as pd
```

```
random = np.random.randn(6,4)
```

```
# Create data with date
```

```
dates_m = pd.date_range('20300101', periods=6, freq='M')
df = pd.DataFrame(random,
                  index=dates_m,
                  columns=list('ABCD'))
```

```
df.head(3)
```

	A	B	C	D
2030-01-31	-1.329190	-0.719102	-0.970625	-0.558589
2030-02-28	0.195689	0.819392	-1.282599	-0.345371
2030-03-31	-0.950495	0.661422	-0.234911	0.904661

```
df.tail(3)
```

	A	B	C	D
2030-04-30	1.147502	-0.156842	0.933256	0.366355
2030-05-31	-0.451576	-0.699042	-0.032462	0.744882
2030-06-30	-0.975023	0.163905	-0.859087	1.272341

4.Matplotlib

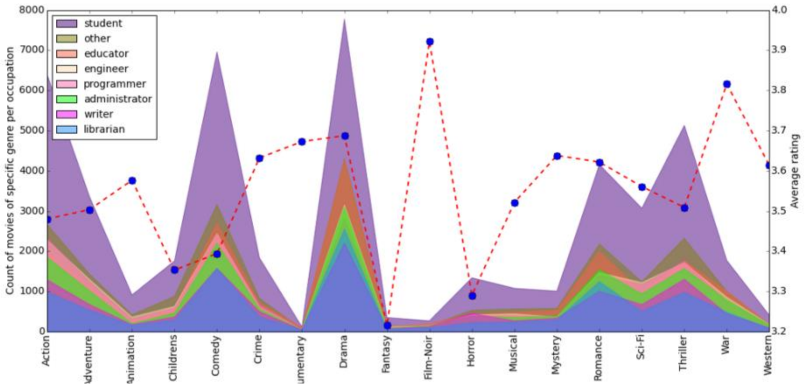
- Another SciPy Stack core package and another Python Library that is tailored for the generation of simple and powerful visualizations with ease is Matplotlib.
- It is a top-notch piece of software which is making Python (with some help of NumPy, SciPy, and Pandas) a cognizant competitor to such scientific tools as MatLab or Mathematica.

With a bit of effort you can make just about any visualizations:

- Line plots;
- Scatter plots;
- Bar charts and Histograms;
- Pie charts;

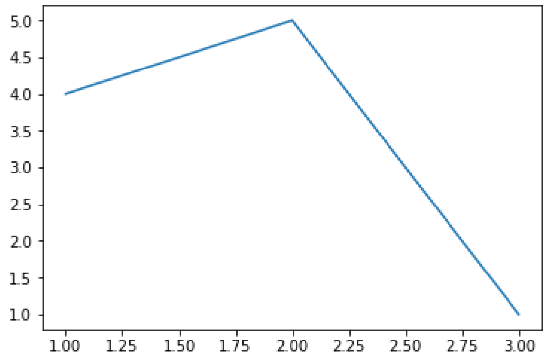
- Stem plots;
- Contour plots;
- Quiver plots;
- Spectrograms.

There are also some additional libraries that can make visualization even easier.



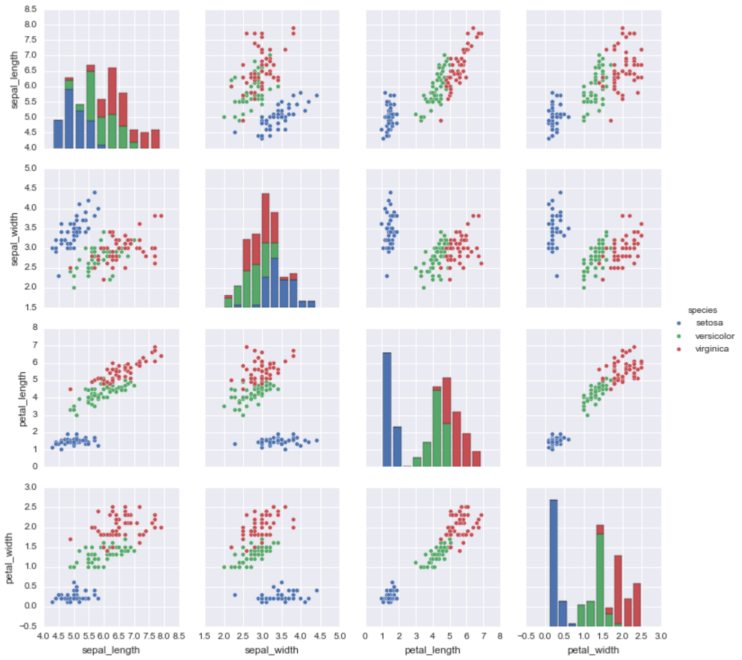

```
import matplotlib
```

```
#Importing pyplot
from matplotlib import pyplot as plt

#Plotting to our canvas
plt.plot([1,2,3],[4,5,1])

#Showing what we plotted
plt.show()
```

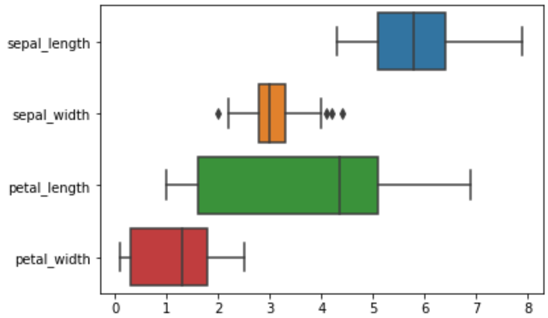


5. Seaborn

- Seaborn is mostly focused on the visualization of statistical models; such visualizations include heat maps, those that summarize the data but still depict the overall distributions.
- Seaborn is based on Matplotlib and highly dependent on that.

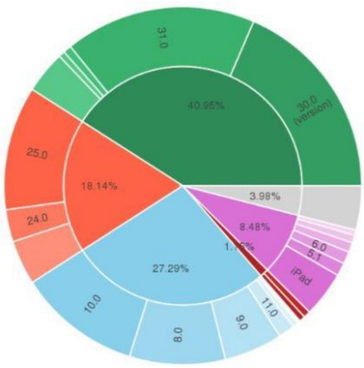



```
import pandas as pd
import seaborn as sb
from matplotlib import pyplot as plt
df = sb.load_dataset('iris')
sb.boxplot(data = df, orient = "h")
plt.show()
```



6. Bokeh

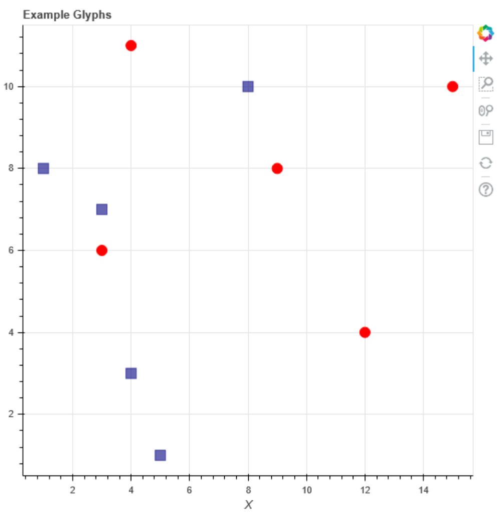
- Another great visualization library is Bokeh, which is aimed at interactive visualizations.
- In contrast to the previous library, this one is independent of Matplotlib.
- The main focus of Bokeh, as we already mentioned, is interactivity and it makes its presentation via modern browsers in the style of Data-Driven Documents (d3.js).



```python
# bokeh basics
from bokeh.plotting import figure
from bokeh.io import show, output_notebook
```

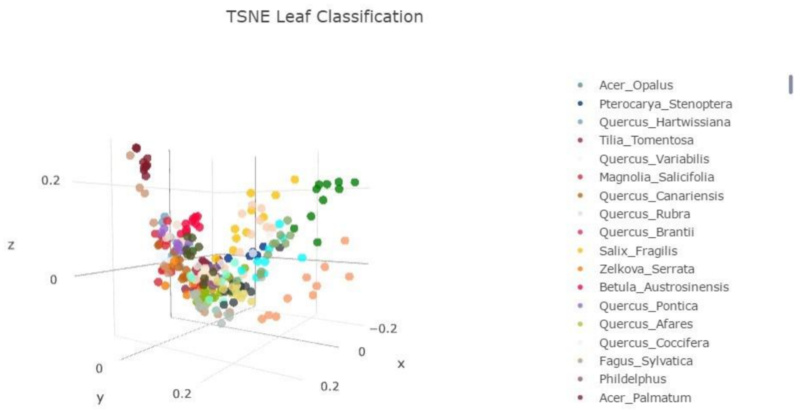
```python
# Create a blank figure with labels
p = figure(plot_width = 600, plot_height = 600,
           title = 'Example Glyphs',
           x_axis_label = 'X', y_axis_label = 'Y')

# Example data
squares_x = [1, 3, 4, 5, 8]
squares_y = [8, 7, 3, 1, 10]
circles_x = [9, 12, 4, 3, 15]
circles_y = [8, 4, 11, 6, 10]

# Add squares glyph
p.square(squares_x, squares_y, size = 12, color = 'navy', alpha = 0.6)
# Add circle glyph
p.circle(circles_x, circles_y, size = 12, color = 'red')

# Set to output the plot in the notebook
output_notebook()
# Show the plot
show(p)
```

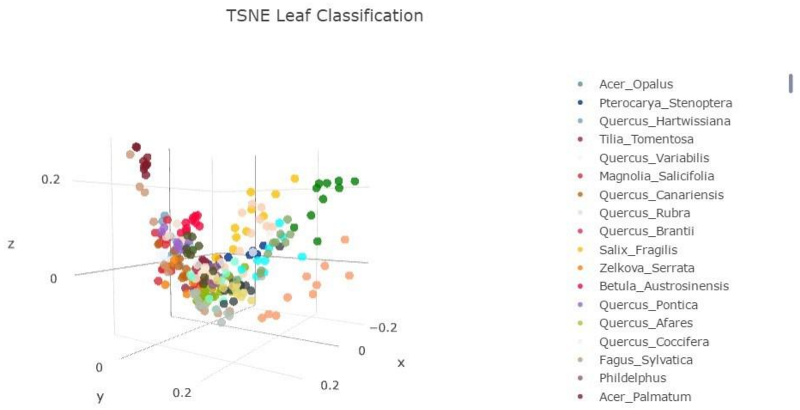
BokehJS 0.12.16 successfully loaded.

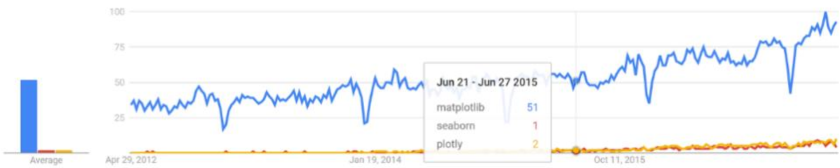


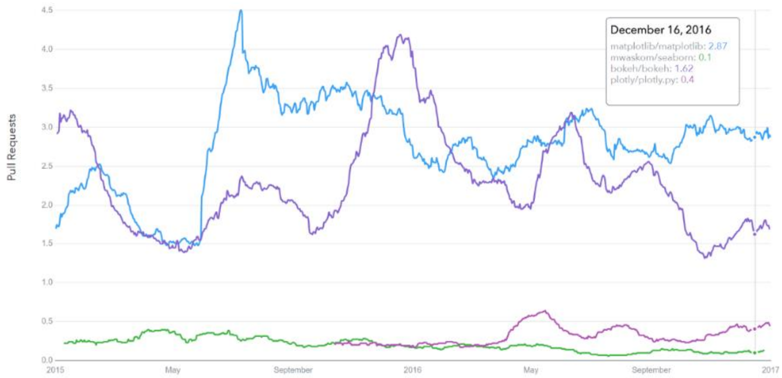
7. Plotly

- It is rather a web-based toolbox for building visualizations, exposing APIs to some programming languages (Python among them).
- There is a number of robust, out-of-box graphics on the plot.ly website.
- In order to use Plotly, you will need to set up your API key.
- The graphics will be processed server side and will be posted on the internet, but there is a way to avoid it.

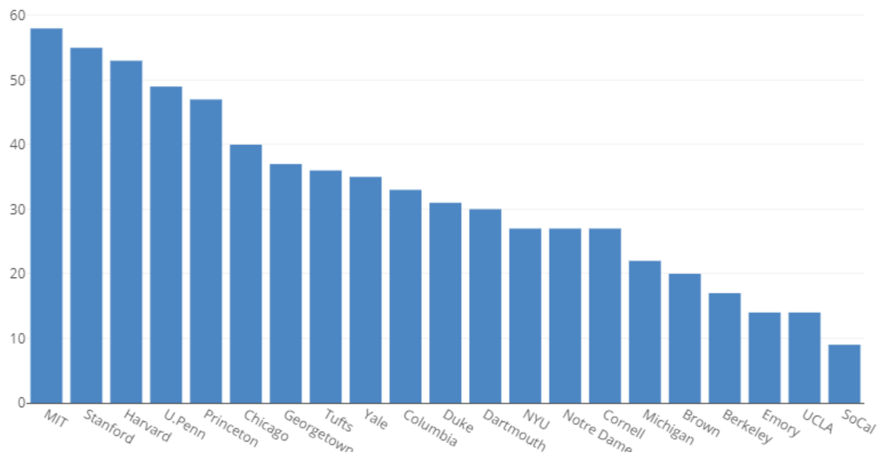









```python
import plotly.plotly as py
import plotly.graph_objs as go

data = [go.Bar(x=df.School,
            y=df.Gap)]

py.iplot(data, filename='jupyter-basic_bar')
```



Machine Learning.

8. SciKit-Learn

- Scikits are additional packages of SciPy Stack designed for specific functionalities like image processing and machine learning facilitation.
- In the regard of the latter, one of the most prominent of these packages is scikit-learn. The package is built on the top of SciPy and makes heavy use of its math operations.
- The scikit-learn exposes a concise and consistent interface to the common machine learning algorithms, making it simple to bring ML into production systems.
- The library combines quality code and good documentation, ease of use and high performance and is de-facto industry standard for machine learning with Python.

```
# Sample Decision Tree Classifier
from sklearn import datasets
from sklearn import metrics
from sklearn.tree import DecisionTreeClassifier
# load the iris datasets
dataset = datasets.load_iris()
# fit a CART model to the data
model = DecisionTreeClassifier()
model.fit(dataset.data, dataset.target)
print(model)
# make predictions
expected = dataset.target
predicted = model.predict(dataset.data)
# summarize the fit of the model
print(metrics.classification_report(expected, predicted))
print(metrics.confusion_matrix(expected, predicted))
```

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
             precision    recall  f1-score   support

          0       1.00      1.00      1.00        50
          1       1.00      1.00      1.00        50
          2       1.00      1.00      1.00        50

avg / total       1.00      1.00      1.00       150

[[50  0  0]
 [ 0 50  0]
 [ 0  0 50]]
```

Deep Learning— TensorFlow / Theano

- In the regard of Deep Learning, one of the most prominent and convenient libraries for Python in this field is Keras, which can function either on top of TensorFlow or Theano. Let's reveal some details about all of them.

9.Theano

Theano is a Python package that defines multi-dimensional arrays similar to NumPy, along with math operations and expressions.

- The library is compiled, making it run efficiently on all architectures.
- Originally developed by the Machine Learning group of Université de Montréal, it is primarily used for the needs of Machine Learning.
- Theano tightly integrates with NumPy on low-level of its operations.
- The library also optimizes the use of GPU and CPU, making the performance of data-intensive computation even faster.
- Efficiency and stability tweaks allow for much more precise results with even very small values, for example, computation of log(1+x) will give cognizant results for even smallest values of x.

```
import theano
from theano import tensor

x = tensor.dscalar()
y = tensor.dscalar()

z = x + y
f = theano.function([x,y], z)
print(f(1.5, 2.5))
```

```
4.0
```

10. TensorFlow

- Coming from developers at Google, it is an open-source library of data flow graphs computations, which are sharpened for Machine Learning.
- It was designed to meet the high-demand requirements of Google environment for training Neural Networks and is a successor of DistBelief, a Machine Learning system, based on Neural Networks.
- However, TensorFlow isn’t strictly for scientific use in border’s of Google—it is general enough to use it in a variety of real-world application.
- The key feature of TensorFlow is their multi-layered nodes system that enables quick training of artificial neural networks on large datasets.
- This powers Google’s voice recognition and object identification from pictures.

```
# Import `tensorflow`
import tensorflow as tf

# Initialize two constants
x1 = tf.constant([1,2,3,4])
x2 = tf.constant([5,6,7,8])

# Multiply
result = tf.multiply(x1, x2)

# Print the result
print(result)
```

```
Tensor("Mul:0", shape=(4,), dtype=int32)
```

Calling data from repository

- **Dataset :-** Downloading a rsfweatherdata2011 dataset from repository.

rsfweatherdata2011 - Microsoft Excel

Home Insert Page Layout Formulas Data Review View

A1 DATE

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	DATE	HOUR-MS	Avg Globa	Avg Globa	Avg Globa	Avg Deck	Dry Bulb Te	DATE AND	Avg Deck I	Avg Globa	Ave Globa	Ave Global 90 South Irradiance (W/ft2)			
2	1/1/2011	1	6.4653	0	-0.5478	-13.2307		########	8.18474	0	0.600647	-0.05089			
3	1/1/2011	2	7.047	0	-0.3776	-12.9523		########	8.68586	0	0.654688	-0.03508			
4	1/1/2011	3	7.1437	0	-0.4596	-12.2292		########	9.98744	0	0.663672	-0.0427			
5	1/1/2011	4	7.0436	0	-0.5842	-11.8457		########	10.67774	0	0.654372	-0.05427			
6	1/1/2011	5	6.8356	0	-0.8721	-11.4127		########	11.45714	0	0.635049	-0.08102			
7	1/1/2011	6	6.8626	0	-0.8528	-11.0883		########	12.04106	0	0.637557	-0.07923			
8	1/1/2011	7	6.2507	0	-0.8124	-10.8942		########	12.39044	0	0.58071	-0.07547			
9	1/1/2011	8	42.8545	3.2397	136.2762	-10.7363		########	12.67466	300.9755	3.981317	12.66049			
10	1/1/2011	9	87.6385	9.5491	134.6766	-9.592		########	14.7344	887.133	8.141891	12.51188			
11	1/1/2011	10	74.9057	9.1773	56.4618	-9.454		########	14.9828	852.592	6.958974	5.245478			
12	1/1/2011	11	115.8546	14.5146	83.2822	-8.9603		########	15.87146	1348.439	10.76325	7.737177			
13	1/1/2011	12	235.9735	28.695	198.1207	-7.5763		########	18.36266	2665.831	21.92268	18.40603			
14	1/1/2011	13	434.1822	51.0432	558.1885	-6.306		########	20.6492	4742.029	40.33689	51.85746			
15	1/1/2011	14	206.5382	23.991	158.4115	-6.589		########	20.1398	2228.818	19.18805	14.71692			
16	1/1/2011	15	151.5942	17.0341	186.8262	-7.084		########	19.2488	1582.507	14.08358	17.35674			
17	1/1/2011	16	75.88	8.2576	147.526	-7.8907		########	17.79674	767.1498	7.049489	13.70563			
18	1/1/2011	17	37.2268	3.4448	118.195	-7.9777		########	17.64014	320.0297	3.458486	10.98069			

rsfweatherdata2011

- **Importing panda library :-**

```
In [10]: import pandas as pd
```

- **Reading and copying data :-** Reading Data from the dataset and copying the data into df.

```
In [5]: df=pd.read_csv('C:/Users/student/Desktop/rsfweatherdata2011.csv')
```

- **To print top elements :-**

```
In [6]: df.head()
```

Out[6]:

	DATE	HOUR-MST	Avg Global PSP (therm-zen cor) [W/m^2]	Avg Global Photometric LI-210 [klux]	Avg Global 90-South PSP [W/m^2]	Avg Deck Dry Bulb Temp [deg C]	Unnamed: 6	DATE AND TIME	Avg Deck Dry Bulb Temp [deg F]
0	1/1/2011	1.0	6.4653	0.0	-0.5478	-13.2307	NaN	1/1/2011 1:00	8.18474
1	1/1/2011	2.0	7.0470	0.0	-0.3776	-12.9523	NaN	1/1/2011 2:00	8.68586
2	1/1/2011	3.0	7.1437	0.0	-0.4596	-12.2292	NaN	1/1/2011 3:00	9.98744
3	1/1/2011	4.0	7.0436	0.0	-0.5842	-11.8457	NaN	1/1/2011 4:00	10.67774
4	1/1/2011	5.0	6.8356	0.0	-0.8721	-11.4127	NaN	1/1/2011 5:00	11.45714

- **To print top 5 rows :-**

```
In [7]: df.head(5)
```

Out[7]:

	DATE	HOUR-MST	Avg Global PSP (therm-zen cor) [W/m^2]	Avg Global Photometric LI-210 [klux]	Avg Global 90-South PSP [W/m^2]	Avg Deck Dry Bulb Temp [deg C]	Unnamed: 6	DATE AND TIME	Avg Deck Dry Bulb Temp [deg F]
0	1/1/2011	1.0	6.4653	0.0	-0.5478	-13.2307	NaN	1/1/2011 1:00	8.18474
1	1/1/2011	2.0	7.0470	0.0	-0.3776	-12.9523	NaN	1/1/2011 2:00	8.68586
2	1/1/2011	3.0	7.1437	0.0	-0.4596	-12.2292	NaN	1/1/2011 3:00	9.98744
3	1/1/2011	4.0	7.0436	0.0	-0.5842	-11.8457	NaN	1/1/2011 4:00	10.67774
4	1/1/2011	5.0	6.8356	0.0	-0.8721	-11.4127	NaN	1/1/2011 5:00	11.45714

- **To print last 2 rows :-**

```
In [8]: df.tail(2)
```

Out[8]:

	DATE	HOUR-MST	Avg Global PSP (therm-zen cor) [W/m^2]	Avg Global Photometric LI-210 [klux]	Avg Global 90-South PSP [W/m^2]	Avg Deck Dry Bulb Temp [deg C]	Unnamed: 6	DATE AND TIME	[I T [de
8758	12/31/2011	23.0	5.5011	0.0	-1.9879	-3.127	NaN	12/31/2011 23:00	26.
8759	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1/0/00 0:00	32.

- **To print top 15 rows of column HOUR-MIST :-**

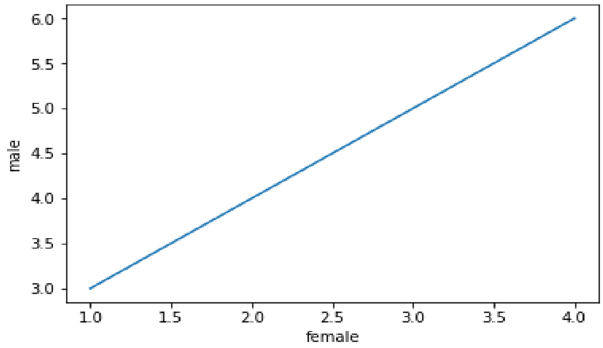
```
In [10]: df['HOUR-MST'].head(15)

Out[10]: 0      1.0
         1      2.0
         2      3.0
         3      4.0
         4      5.0
         5      6.0
         6      7.0
         7      8.0
         8      9.0
         9     10.0
         10    11.0
         11    12.0
         12    13.0
         13    14.0
         14    15.0
         Name: HOUR-MST, dtype: float64
```

- **Line plot :-**

In [11]:
```
import matplotlib.pyplot as plt
```

In [12]:
```
import matplotlib.mlab as mlab
```

In [13]:
```
plt.plot([1,2,3,4],[3,4,5,6])
plt.xlabel('female')
plt.ylabel('male')
plt.show()
```



- **To display data type of eachy column :-**

In [14]:
```
df.dtypes
```

Out[14]:
```
DATE                                              object
HOUR-MST                                         float64
Avg Global PSP (therm-zen cor) [W/m^2]           float64
Avg Global Photometric LI-210 [klux]             float64
Avg Global 90-South PSP [W/m^2]                  float64
Avg Deck Dry Bulb Temp [deg C]                   float64
Unnamed: 6                                       float64
DATE AND TIME                                     object
Avg Deck Dry Bulb Temp [deg F]                   float64
Avg Global Photometric LI-210 [fc]               float64
Ave Global Irradiance (W/ft2)                    float64
Ave Global 90 South Irradiance (W/ft2)           float64
dtype: object
```

- **To check id data has null values**

In [15]:
```
df.isnull().any()
```

Out[15]:
```
DATE                                       True
HOUR-MST                                   True
Avg Global PSP (therm-zen cor) [W/m^2]     True
Avg Global Photometric LI-210 [klux]       True
Avg Global 90-South PSP [W/m^2]            True
Avg Deck Dry Bulb Temp [deg C]             True
Unnamed: 6                                 True
DATE AND TIME                             False
Avg Deck Dry Bulb Temp [deg F]            False
Avg Global Photometric LI-210 [fc]        False
Ave Global Irradiance (W/ft2)             False
Ave Global 90 South Irradiance (W/ft2)    False
dtype: bool
```

- **To describe the data**

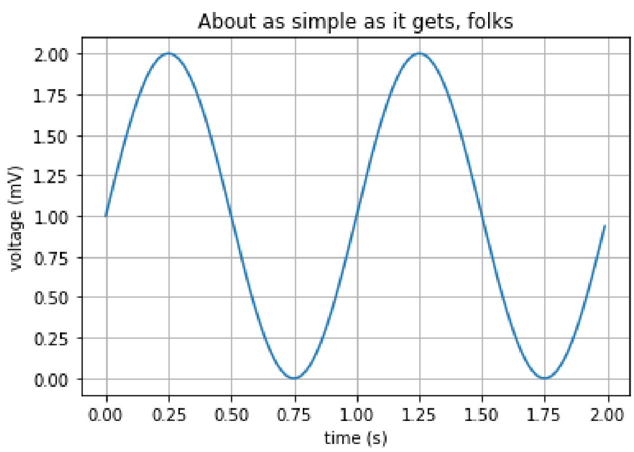
In [16]:
```
df.describe()
```

Out[16]:

	HOUR-MST	Avg Global PSP (therm-zen cor) [W/m^2]	Avg Global Photometric LI-210 [klux]	Avg Global 90-South PSP [W/m^2]	Avg Deck Dry Bulb Temp [deg C]	Unnamed: 6	Avg Dr Tem
count	8759.000000	8759.000000	8759.000000	8759.000000	8759.000000	0.0	8760.0
mean	11.501313	201.286748	21.571919	-213.725429	10.934353	NaN	51.6
std	6.921886	279.534689	30.800580	6142.440740	10.551615	NaN	18.9
min	0.000000	-3.532500	0.000000	-99999.000000	-26.519200	NaN	-15.7
25%	6.000000	4.934900	0.000000	-1.793150	3.559650	NaN	38.4
50%	12.000000	14.978800	0.956400	4.693500	10.617700	NaN	51.1
75%	17.500000	357.039450	38.486800	245.846900	19.114750	NaN	66.4
max	23.000000	1060.963400	120.475100	1260.548300	34.222700	NaN	93.6

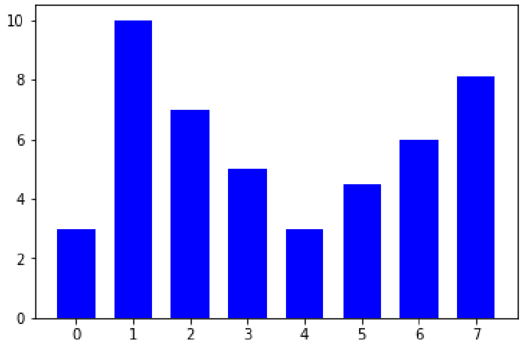
- **Adding elements to line plots :-**

In [25]:
```
import numpy as np
t=np.arange(0.0,2.0,0.01)
s = 1 + np.sin(2*np.pi*t)
plt.plot(t, s)
plt.xlabel('time (s)')
plt.ylabel('voltage (mV)')
plt.title('About as simple as it gets, folks')
plt.grid(True)
plt.savefig("test.png")
plt.show()
```



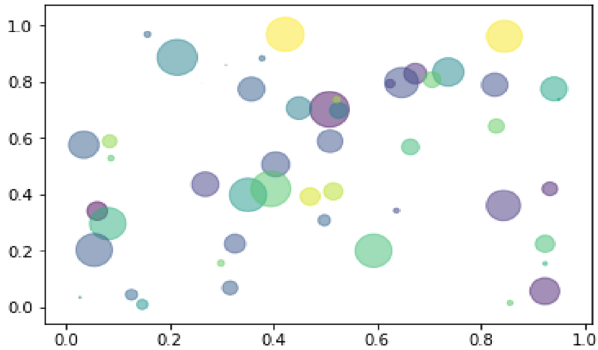
- **Bar Plot :-**

In [23]:
```
y = [3, 10, 7, 5, 3, 4.5, 6, 8.1]
x = range(len(y))
width = 1/1.5
plt.bar(x, y, width, color="blue")
plt.show()
```



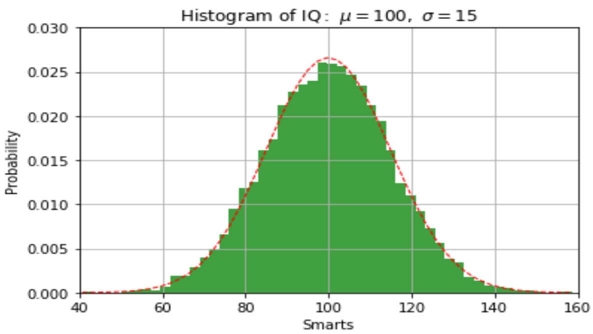
- **Scatter Plot**

In [26]:
```
N = 50
x = np.random.rand(N)
y = np.random.rand(N)
colors = np.random.rand(N)
area = np.pi * (15 * np.random.rand(N))**2  # 0 to 15 point radii

plt.scatter(x, y, s=area, c=colors, alpha=0.5)
plt.show()
```



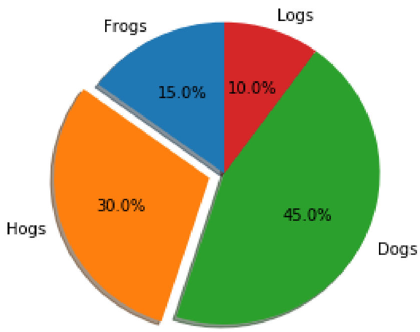
- **Histogram**

```
In [27]: mu, sigma = 100, 15
x = mu + sigma*np.random.randn(10000) # Generate random values with some dis

# the histogram of the data
n, bins, patches = plt.hist(x, 50, normed=1, facecolor='green', alpha=0.75)

# add a 'best fit' line
y = mlab.normpdf( bins, mu, sigma)
l = plt.plot(bins, y, 'r--', linewidth=1)

plt.xlabel('Smarts')
plt.ylabel('Probability')
plt.title(r'$\mathrm{Histogram\ of\ IQ:}\ \mu=100,\ \sigma=15$')
plt.axis([40, 160, 0, 0.03])
plt.grid(True)

plt.show()
```



- **Pie Chart**

```
In [28]: # Pie chart, where the slices will be ordered and plotted counter-clockwise:
labels = 'Frogs', 'Hogs', 'Dogs', 'Logs'
sizes = [15, 30, 45, 10]
explode = (0, 0.1, 0, 0)  # only "explode" the 2nd slice (i.e. 'Hogs')

fig1, ax1 = plt.subplots()
ax1.pie(sizes, explode=explode, labels=labels, autopct='%1.1f%%',
        shadow=True, startangle=90)
ax1.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circl

plt.show()
```



Legends, Subplotting and Annotation

- **Legends:**

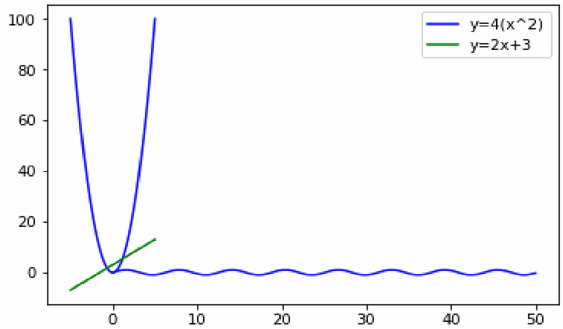
Draw two lots overlaid upon each other with the first plot being a parabola of the form y = 4(x^2) & second being a straight line of form y = 2x+3 in the internal of 15 to 5.

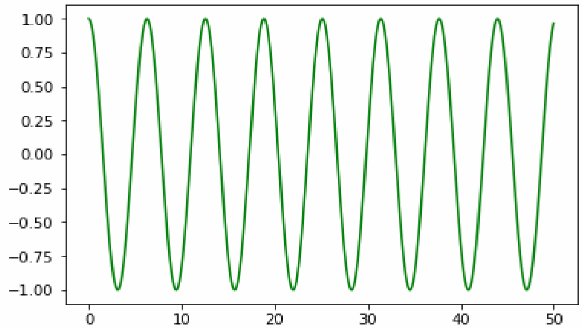
Use colors to differentiate between the plots and use legends to indicate what each plot is doing.

[2]:

```
                                                         #Vandana Negi
import numpy as np
import pandas as pd
%matplotlib inline
%pylab inline
x = linspace(-5,5,100)
plot(x,4*(x*x),'b')
plot(x,(2*x)+3,'g')
legend(['parabola','straight line'])
legend(['y=4(x^2)','y=2x+3'])
x = linspace(0,50,500)
figure(1)
plot(x,sin(x),'b')
figure(2)
plot(x,cos(x),'g')
savefig('C:/Users/Student/Desktop/usine.png')
figure(1)
title('siny')
savefig('C:/Users/Student/Desktop/sine.png')
```

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





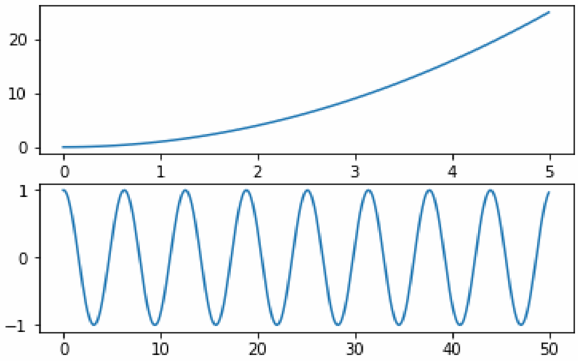
- **Annotation:**

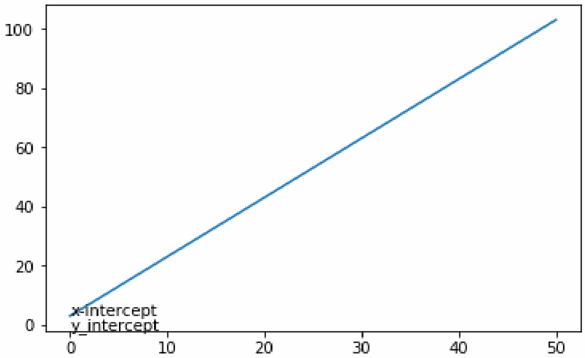
    Draw a line of the form y = x as one figure and another line of the form y = 2x+3.
    Switch back to the first figure, annotate the x & y intercepts.
    Now switch to the second figure and annotate its x & y intercepts and save each of them.

[4]:

```
                                        #Vandana Negi
import numpy as np
import pandas as pd
%matplotlib inline
figure(1)
a=linspace(-5,5,100)
plot(x,x)
figure(2)
plot(x,(2*x)+3)
figure(1)
annotate('origin',xy=(0.0,0.0))
figure(2)
annotate('x-intercept',xy=(0,3))
annotate('y_intercept',xy=(0,-1.5))
savefig('C:/Users/negi/Desktop/usine.png')
figure(1)
savefig('C:/Users/negi/Desktop/ussine.png')
subplot(2,1,1)
subplot(2,1,2)
x=linspace(0,50,500)
plot(x,cos(x))
subplot(2,1,1)
y=linspace(0,5,100)
plot(y,y**2)
```

[5]: [<matplotlib.lines.Line2D at 0x7ff31423ff28>]





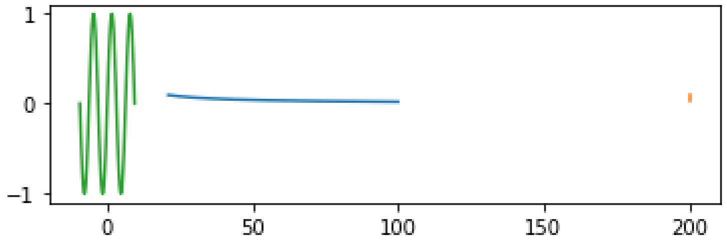
- **Subplotting:**

We know that the pressure, volume &Temperature are held by the equation PV=nRT where nR is a constant. Let us assume nR = 0.01 Joules/Kelvin and T = 200K . V can be in the range from 21cc to 100cc. Draw two different plots as subplots one being the pressure vs volume plot and the other being presume versus Temperature plot.

[7]:

```
                                                        #Vandana Negi
import numpy as np
import pandas as pd
%matplotlib inline
%pylab inline
v=linspace(21,100,500)
subplot(2,1,1)
plot(v,2.0/v)
T=linspace(200,200,500)
plot(T,2.0/v)
x=linspace(-3*pi,3*pi,100)
plot(x,sin(x))
savefig('C:/Users/negi/Desktop/usine.png')
savefig('C:/Users/negi/Desktop/usine.eps')
```

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



NUMPY AND ITS FEATURES

- **Numpy:**

Numpy is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays. If you are already familiar with MATLAB, you might find this tutorial useful to get started with Numpy.

- **Arrays:**

A numpy array is a grid of values, all of the same type, and is indexed by a tuple of nonnegative integers. The number of dimensions is the *rank* of the array; the *shape* of an array is a tuple of integers giving the size of the array along each dimension.

We can initialize numpy arrays from nested Python lists, and access elements using square brackets:

```
import numpy as np
```

```
a = np.array([1, 2, 3])   # Create a rank 1 array  #Loma Attree
print(type(a))            # Prints "<class 'numpy.ndarray'>"
print(a.shape)            # Prints "(3,)"
print(a[0], a[1], a[2])   # Prints "1 2 3"
a[0] = 5                  # Change an element of the array
print(a)                  # Prints "[5, 2, 3]"

b = np.array([[1,2,3],[4,5,6]])    # Create a rank 2 array
print(b.shape)                     # Prints "(2, 3)"
print(b[0, 0], b[0, 1], b[1, 0])   # Prints "1 2 4"
```

```
<class 'numpy.ndarray'>
(3,)
1 2 3
[5 2 3]
(2, 3)
1 2 4
```

Numpy also provides many functions to create arrays:

```
a = np.zeros((2,2))   # Create an array of all zeros  #Loma Attree
print(a)              # Prints "[[ 0.  0.]
                      #          [ 0.  0.]]"

b = np.ones((1,2))    # Create an array of all ones
print(b)              # Prints "[[ 1.  1.]]"

c = np.full((2,2), 7)  # Create a constant array
print(c)               # Prints "[[ 7.  7.]
                       #          [ 7.  7.]]"

d = np.eye(2)         # Create a 2x2 identity matrix
print(d)              # Prints "[[ 1.  0.]
                      #          [ 0.  1.]]"

e = np.random.random((2,2))  # Create an array filled with random values
print(e)                     # Might print "[[ 0.91940167  0.08143941]
                             #               [ 0.68744134  0.87236687]]"
```

```
[[0. 0.]
 [0. 0.]]
[[1. 1.]]
[[7 7]
 [7 7]]
[[1. 0.]
 [0. 1.]]
[[0.28360013 0.2720194 ]
 [0.45414722 0.13032591]]
```

- **Slicing:**

  Similar to Python lists, numpy arrays can be sliced. Since arrays may be multidimensional, you must specify a slice for each dimension of the array:

```python
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])          #Loma Attree

# Use slicing to pull out the subarray consisting of the first 2 rows
# and columns 1 and 2; b is the following array of shape (2, 2):
# [[2 3]
#  [6 7]]
b = a[:2, 1:3]

# A slice of an array is a view into the same data, so modifying it
# will modify the original array.
print(a[0, 1])   # Prints "2"
b[0, 0] = 77     # b[0, 0] is the same piece of data as a[0, 1]
print(a[0, 1])   # Prints "77"
```

```
2
77
```

You can also mix integer indexing with slice indexing. However, doing so will yield an array of lower rank than the original array. Note that this is quite different from the way that MATLAB handles array slicing:

```python
# Create the following rank 2 array with shape (3, 4)
# [[ 1  2  3  4]
#  [ 5  6  7  8]
#  [ 9 10 11 12]]
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

# Two ways of accessing the data in the middle row of the array.
# Mixing integer indexing with slices yields an array of lower rank,
# while using only slices yields an array of the same rank as the
# original array:
row_r1 = a[1, :]    # Rank 1 view of the second row of a
row_r2 = a[1:2, :]  # Rank 2 view of the second row of a
print(row_r1, row_r1.shape)  # Prints "[5 6 7 8] (4,)"
print(row_r2, row_r2.shape)  # Prints "[[5 6 7 8]] (1, 4)"

# We can make the same distinction when accessing columns of an array:
col_r1 = a[:, 1]
col_r2 = a[:, 1:2]
print(col_r1, col_r1.shape)  # Prints "[ 2  6 10] (3,)"
print(col_r2, col_r2.shape)  # Prints "[[ 2]
                             #          [ 6]
                             #          [10]] (3, 1)"
```

- **<u>Integer array indexing:</u>**

When you index into numpy arrays using slicing, the resulting array view will always be a subarray of the original array. In contrast, integer array indexing allows you to construct arbitrary arrays using the data from another array. Here is an example:

```
[5]: a = np.array([[1,2], [3, 4], [5, 6]])          #Loma Attree

     # An example of integer array indexing.
     # The returned array will have shape (3,) and
     print(a[[0, 1, 2], [0, 1, 0]])  # Prints "[1 4 5]"

     # The above example of integer array indexing is equivalent to this:
     print(np.array([a[0, 0], a[1, 1], a[2, 0]]))  # Prints "[1 4 5]"

     # When using integer array indexing, you can reuse the same
     # element from the source array:
     print(a[[0, 0], [1, 1]])  # Prints "[2 2]"

     # Equivalent to the previous integer array indexing example
     print(np.array([a[0, 1], a[0, 1]]))  # Prints "[2 2]"

     [1 4 5]
     [1 4 5]
     [2 2]
     [2 2]
```

One useful trick with integer array indexing is selecting or mutating one element from each row of a matrix:

```
[2 2]

]: a = np.array([[1,2], [3, 4], [5, 6]])                     #Loma Attree

   bool_idx = (a > 2)   # Find the elements of a that are bigger than 2;
                        # this returns a numpy array of Booleans of the same
                        # shape as a, where each slot of bool_idx tells
                        # whether that element of a is > 2.

   print(bool_idx)      # Prints "[[False False]
                        #          [ True  True]
                        #          [ True  True]]"

   # We use boolean array indexing to construct a rank 1 array
   # consisting of the elements of a corresponding to the True values
   # of bool_idx
   print(a[bool_idx])  # Prints "[3 4 5 6]"

   # We can do all of the above in a single concise statement:
   print(a[a > 2])     # Prints "[3 4 5 6]"

   [[False False]
    [ True  True]
    [ True  True]]
   [3 4 5 6]
   [3 4 5 6]
```

- **Datatypes:**

  Every numpy array is a grid of elements of the same type. Numpy provides a large set of numeric datatypes that you can use to construct arrays. Numpy tries to guess a datatype when you create an array, but functions that construct arrays usually also include an optional argument to explicitly specify the datatype. Here is an example:

```
[7]: x = np.array([1, 2])   # Let numpy choose the datatype     #Loma Attree
     print(x.dtype)         # Prints "int64"

     x = np.array([1.0, 2.0])   # Let numpy choose the datatype
     print(x.dtype)             # Prints "float64"

     x = np.array([1, 2], dtype=np.int64)   # Force a particular datatype
     print(x.dtype)                         # Prints "int64"
```

```
int64
float64
int64
```

- **Array math:**

Basic mathematical functions operate elementwise on arrays, and are available both as operator overloads and as functions in the numpy module:

```
[8]: x = np.array([[1,2],[3,4]], dtype=np.float64)              #Loma Attree
     y = np.array([[5,6],[7,8]], dtype=np.float64)

     # Elementwise sum; both produce the array
     # [[ 6.0  8.0]
     #  [10.0 12.0]]
     print(x + y)
     print(np.add(x, y))

     # Elementwise difference; both produce the array
     # [[-4.0 -4.0]
     #  [-4.0 -4.0]]
     print(x - y)
     print(np.subtract(x, y))

     # Elementwise product; both produce the array
     # [[ 5.0 12.0]
     #  [21.0 32.0]]
     print(x * y)
     print(np.multiply(x, y))

     # Elementwise division; both produce the array
     # [[ 0.2         0.33333333]
     #  [ 0.42857143  0.5       ]]
     print(x / y)
     print(np.divide(x, y))

     # Elementwise square root; produces the array
     # [[ 1.          1.41421356]
     #  [ 1.73205081  2.        ]]
     print(np.sqrt(x))
```

```
[[ 6.  8.]
 [10. 12.]]
[[ 6.  8.]
 [10. 12.]]
[[-4. -4.]
 [-4. -4.]]
[[-4. -4.]
 [-4. -4.]]
[[ 5. 12.]
 [21. 32.]]
[[ 5. 12.]
 [21. 32.]]
```

```
 [21. 32.]]
[[ 5. 12.]
 [21. 32.]]
[[0.2        0.33333333]
 [0.42857143 0.5       ]]
[[0.2        0.33333333]
 [0.42857143 0.5       ]]
[[1.         1.41421356]
 [1.73205081 2.        ]]
```

Note that unlike MATLAB, `*` is elementwise multiplication, not matrix multiplication. We instead use the `dot`function to compute inner products of vectors, to multiply a vector by a matrix, and to multiply matrices. `dot` is available both as a function in the numpy module and as an instance method of array objects:

```
 [1.73205081 2.        ]]
```

```
[9]: x = np.array([[1,2],[3,4]])          #Loma Attree
y = np.array([[5,6],[7,8]])

v = np.array([9,10])
w = np.array([11, 12])

# Inner product of vectors; both produce 219
print(v.dot(w))
print(np.dot(v, w))

# Matrix / vector product; both produce the rank 1 array [29 67]
print(x.dot(v))
print(np.dot(x, v))

# Matrix / matrix product; both produce the rank 2 array
# [[19 22]
#  [43 50]]
print(x.dot(y))
print(np.dot(x, y))
```

```
219
219
[29 67]
[29 67]
[[19 22]
 [43 50]]
[[19 22]
 [43 50]]
```

Numpy provides many useful functions for performing computations on arrays; one of the most useful is `sum`:

```
 [43 50]]
```

```
[10]: x = np.array([[1,2],[3,4]])                    #Loma Attree

      print(np.sum(x))  # Compute sum of all elements; prints "10"
      print(np.sum(x, axis=0))  # Compute sum of each column; prints "[4 6]"
      print(np.sum(x, axis=1))  # Compute sum of each row; prints "[3 7]"
```

```
10
[4 6]
[3 7]
```

Apart from computing mathematical functions using arrays, we frequently need to reshape or otherwise manipulate data in arrays. The simplest example of this type of operation is transposing a matrix; to transpose a matrix, simply use the `T` attribute of an array object:

```
[3 7]
```

```
[11]: # We will add the vector v to each row of the matrix x,
      # storing the result in the matrix y
      x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])            #Loma Attree
      v = np.array([1, 0, 1])
      y = np.empty_like(x)   # Create an empty matrix with the same shape as x

      # Add the vector v to each row of the matrix x with an explicit loop
      for i in range(4):
          y[i, :] = x[i, :] + v

      # Now y is the following
      # [[ 2  2  4]
      #  [ 5  5  7]
      #  [ 8  8 10]
      #  [11 11 13]]
      print(y)
```

```
[[ 2  2  4]
 [ 5  5  7]
 [ 8  8 10]
 [11 11 13]]
```

- **<u>Broadcasting:</u>**

Broadcasting is a powerful mechanism that allows numpy to work with arrays of different shapes when performing arithmetic operations. Frequently we have a smaller array and a larger array, and we want to use the smaller array multiple times to perform some operation on the larger array.

For example, suppose that we want to add a constant vector to each row of a matrix. We could do it like this:

```
[12]: # We will add the vector v to each row of the matrix x,
# storing the result in the matrix y
x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])          #Loma Attree
v = np.array([1, 0, 1])
vv = np.tile(v, (4, 1))   # Stack 4 copies of v on top of each other
print(vv)                 # Prints "[[1 0 1]
                          #          [1 0 1]
                          #          [1 0 1]
                          #          [1 0 1]]"
y = x + vv  # Add x and vv elementwise
print(y)  # Prints "[[ 2  2  4
          #          [ 5  5  7]
          #          [ 8  8 10]
          #          [11 11 13]]"
```

```
[[1 0 1]
 [1 0 1]
 [1 0 1]
 [1 0 1]]
[[ 2  2  4]
 [ 5  5  7]
 [ 8  8 10]
 [11 11 13]]
```

This works; however when the matrix `x` is very large, computing an explicit loop in Python could be slow. Note that adding the vector `v` to each row of the matrix `x` is equivalent to forming a matrix `vv` by stacking multiple copies of `v` vertically, then performing elementwise summation of `x` and `VV.`We could implement this approach like this:

```
[13]: # We will add the vector v to each row of the matrix x,
# storing the result in the matrix y
x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])       #Loma Attree
v = np.array([1, 0, 1])
y = x + v  # Add v to each row of x using broadcasting
print(y)  # Prints "[[ 2  2  4]
          #          [ 5  5  7]
          #          [ 8  8 10]
          #          [11 11 13]]"
```

```
[[ 2  2  4]
 [ 5  5  7]
 [ 8  8 10]
 [11 11 13]]
```

What is Machine Learning?

Data science, machine learning and artificial intelligence are some of the top trending topics in the tech world today. Data mining and Bayesian analysis are trending and this is adding the demand for machine learning. This tutorial is your entry into the world of machine learning.

Machine learning is a discipline that deals with programming the systems so as to make them automatically learn and improve with experience. Here, learning implies recognizing and understanding the input data and taking informed decisions based on the supplied data. It is very difficult to consider all the decisions based on all possible inputs. To solve this problem, algorithms are developed that build knowledge from a specific data and past experience by applying the principles of statistical science, probability, logic, mathematical optimization, reinforcement learning, and control theory.

Applications of Machine Learning Algorithms

The developed machine learning algorithms are used in various applications such as −

- Vision processing
- Language processing
- Forecasting things like stock market trends, weather
- Pattern recognition

- Games
- Data mining
- Expert systems
- Robotics

Steps Involved in Machine Learning

A machine learning project involves the following steps −

- Defining a Problem
- Preparing Data
- Evaluating Algorithms
- Improving Results
- Presenting Results

Concepts of Learning

Learning is the process of converting experience into expertise or knowledge.

Learning can be broadly classified into three categories, as mentioned below, based on the nature of the learning data and interaction between the learner and the environment.

- Supervised Learning
- Unsupervised Learning
- Semi-supervised Learning

Similarly, there are four categories of machine learning algorithms as shown below −

- Supervised learning algorithm
- Unsupervised learning algorithm
- Semi-supervised learning algorithm
- Reinforcement learning algorithm

However, the most commonly used ones are **supervised** and **unsupervised learning**.

Supervised Learning

Supervised learning is commonly used in real world applications, such as face and speech recognition, products or movie recommendations, and sales forecasting. Supervised learning can be further classified into two types - **Regression** and **Classification**.

Regression trains on and predicts a continuous-valued response, for example predicting real estate prices.

Classification attempts to find the appropriate class label, such as analyzing positive/negative sentiment, male and female persons, benign and malignant tumors, secure and unsecure loans etc.

In supervised learning, learning data comes with description, labels, targets or desired outputs and the objective is to find a general rule that maps inputs to outputs. This kind of learning data is called **labeled data**. The learned rule is then used to label new data with unknown outputs.

Supervised learning involves building a machine learning model that is based on **labeled samples**. For example, if we build a system to estimate the price of a plot of land or a house based on various features, such as size, location, and so on, we first need to create a database and label it. We need to teach the algorithm what features correspond to what prices. Based on this data, the algorithm will learn how to calculate the price of real estate using the values of the input features.

Supervised learning deals with learning a function from available training data. Here, a learning algorithm analyzes the training data and produces a derived function that can be used for mapping new examples. There are many **supervised learning algorithms** such as Logistic Regression, Neural networks, Support Vector Machines (SVMs), and Naive Bayes classifiers.

Common **examples** of supervised learning include classifying e-mails into spam and not-spam categories, labeling webpages based on their content, and voice recognition.

Unsupervised Learning

Unsupervised learning is used to detect anomalies, outliers, such as fraud or defective equipment, or to group customers with similar behaviors for a sales campaign. It is the opposite of supervised learning. There is no labeled data here.

When learning data contains only some indications without any description or labels, it is up to the coder or to the algorithm to find the structure of the underlying data, to discover hidden patterns, or to determine how to describe the data. This kind of learning data is called **unlabeled data**.

Suppose that we have a number of data points, and we want to classify them into several groups. We may not exactly know what the criteria of classification would be. So, an unsupervised learning algorithm tries to classify the given dataset into a certain number of groups in an optimum way.

Unsupervised learning algorithms are extremely powerful tools for analyzing data and for identifying patterns and trends. They are most commonly used for clustering similar input into logical groups. Unsupervised learning algorithms include Kmeans, Random Forests, Hierarchical clustering and so on.

Semi-supervised Learning

If some learning samples are labeled, but some other are not labeled, then it is semi-supervised learning. It makes use of a large amount of **unlabeled data for training** and a small amount of **labeled data for testing**. Semi-supervised learning is applied in cases where it is expensive to acquire a fully labeled dataset while more practical to label a small subset. For example, it often requires skilled experts to label certain remote sensing images, and lots of field experiments to locate oil at a particular location, while acquiring unlabeled data is relatively easy.

Reinforcement Learning

Here learning data gives feedback so that the system adjusts to dynamic conditions in order to achieve a certain objective. The system evaluates its performance based on the feedback responses and reacts accordingly. The best known instances include self-driving cars and chess master algorithm AlphaGo.

classification: samples belong to two or more classes and we want to learn from already labeled data how to predict the class of unlabeled data. An example of a classification problem would be handwritten digit recognition, in which the aim is to assign each input vector to one of a finite number of discrete categories. Another way to think of classification is as a discrete (as opposed to continuous) form of supervised learning where one has a limited number of categories and for each of the n samples provided, one is to try to label them with the correct category or class.

What is Scikit-learn?

Scikit-learn is an open source Python library for machine learning. The library supports state-of-the-art algorithms such as KNN, XGBoost, random forest, SVM among others. It is built on top of Numpy. Scikit-learn is widely used in kaggle competition as well as prominent tech companies. Scikit-learn helps in preprocessing, dimensionality reduction(parameter selection), classification, regression, clustering, and model selection. If you are a Python programmer or you are looking for a robust library you can use to bring machine learning into a production system then a library that you will want to seriously consider is scikit-learn.

Scikit-learn provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python.

It is licensed under a permissive simplified BSD license and is distributed under many Linux distributions, encouraging academic and commercial use.

The library is built upon the SciPy (Scientific Python) that must be installed before you can use scikit-learn. This stack that includes:

- **NumPy**: Base n-dimensional array package
- **SciPy**: Fundamental library for scientific computing
- **Matplotlib**: Comprehensive 2D/3D plotting
- **IPython**: Enhanced interactive console
- **Sympy**: Symbolic mathematics
- **Pandas**: Data structures and analysis

Extensions or modules for SciPy care conventionally named SciKits. As such, the module provides learning algorithms and is named scikit-learn.

The vision for the library is a level of robustness and support required for use in production systems. This means a deep focus on concerns such as easy of use, code quality, collaboration, documentation and performance.

Although the interface is Python, c-libraries are leverage for performance such as numpy for arrays and matrix operations, LAPACK, LibSVM and the careful use of cython.

Scikit-learn was initially developed by David Cournapeau as a Google summer of code project in 2007.

Later Matthieu Brucher joined the project and started to use it as apart of his thesis work. In 2010 INRIA got involved and the first public release (v0.1 beta) was published in late January 2010.

The project now has more than 30 active contributors and has had paid sponsorship from INRIA, Google, Tinyclues and the Python Software Foundation.

What are the features?

The library is focused on modeling data. It is not focused on loading, manipulating and summarizing data. For these features, refer to NumPy and Pandas. Some popular groups of models provided by scikit-learn include:

- **Clustering**: for grouping unlabeled data such as KMeans.
- **Cross Validation**: for estimating the performance of supervised models on unseen data.
- **Datasets**: for test datasets and for generating datasets with specific properties for investigating model behavior.
- **Dimensionality Reduction**: for reducing the number of attributes in data for summarization, visualization and feature selection such as Principal component analysis.
- **Ensemble methods**: for combining the predictions of multiple supervised models.
- **Feature extraction**: for defining attributes in image and text data.
- **Feature selection**: for identifying meaningful attributes from which to create supervised models.
- **Parameter Tuning**: for getting the most out of supervised models.
- **Manifold Learning**: For summarizing and depicting complex multi-dimensional data.
- **Supervised Models**: a vast array not limited to generalized linear models, discriminate analysis, naive bayes, lazy methods, neural networks, support vector machines and decision trees.

Machine Learning Trends

1) Intelligence on the Cloud

Algorithms can help companies unearth insights about their business, but this proposition can be expensive with no guarantees of a bottom-line increase. Companies often deal with having to collect data, hire data scientists and train them to deal with changing databases. Now that more data metrics are becoming available, the cost to store it is dropping thanks to the cloud. There will no longer be the need to manage infrastructure as cloud systems can generate new models as the scale of an operation increases, while also delivering more accurate results. More open-source ML frameworks are coming to the fold, obtaining pre-trained platforms that can tag images, recommend products and perform natural language processing tasks.

2) Quantum Computing Capabilities

Some of the tasks that ML can help companies deal with is the manipulation and classification of large quantities of vectors in high-dimensional spaces. Current algorithms take a large chunk of time to solve these problems, costing companies more to complete their business processes.

Quantum computers are slated to become all the rage soon as they can manipulate high-dimensional vectors at a fraction of the time. These will be able to increase the number of vectors and dimensions that are processed when compared to traditional algorithms in a quicker period of time.

3) Improved Personalization

Retailers are already making waves in developing recommendation engines that reach their target audience more accurately. Taking this a step further, ML will be able to improve the personalization techniques of these engines in more precise ways. The technology will offer more specific data that they can then use on ads to improve the shopping experience for consumers.

4) Data on Data

As the amount of data available increases, the cost of storing this data decreases at roughly the same rate. ML has great potential in generating data of the highest quality that will lead to better models, an improved user experience and more data that helps repeat but improve upon this cycle. Companies such as Tesla add a million miles of driving data to enhance its self-driving capabilities every hour. Its Autopilot feature learns from this data and improves the software that propels these self-driving vehicles forward as the company gathers more data on the possible pitfalls of autonomous driving technology.

Evolution of machine learning

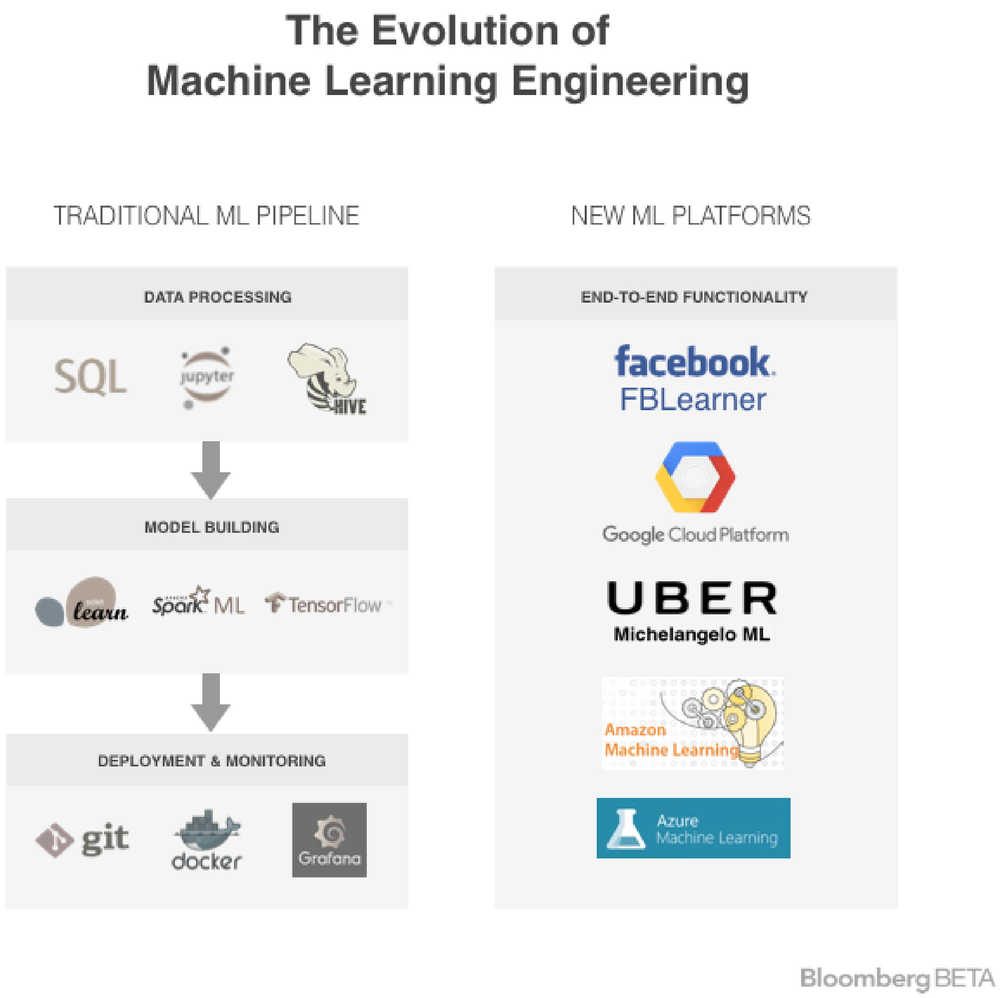
Major tech companies have actively reoriented themselves around AI and machine learning: Google is now “AI-first,” Uber has ML running through its veins and internal AI research labs keep popping up.

They’re pouring resources and attention into convincing the world that the machine intelligence revolution is arriving now. They tout deep learning, in particular, as the breakthrough driving this transformation and powering new self-driving cars, virtual assistants and more.

Despite this hype around the state of the *art*, the state of the *practice* is less futuristic.

Software engineers and data scientists working with machine learning still use many of the same algorithms and engineering tools they did years ago.

That is, traditional machine learning models — not deep neural networks — are powering most AI applications. Engineers still use traditional software engineering tools for machine learning engineering, and they don’t work: The pipelines that take data to model to result end up built out of scattered, incompatible pieces. There is change coming, as big tech companies smooth out this process by building new machine learning-specific platforms with end-to-end functionality.



Keras

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation. *Being able to go from idea to result with the least possible delay is key to doing good research.*

Use Keras if you need a deep learning library that:

- Allows for easy and fast prototyping (through user friendliness, modularity, and extensibility).
- Supports both convolutional networks and recurrent networks, as well as combinations of the two.
- Runs seamlessly on CPU and GPU.

Read the documentation at Keras.io.

Keras is compatible with: **Python 2.7-3.6**.

Guiding principles

- **User friendliness.** Keras is an API designed for human beings, not machines. It puts user experience front and center. Keras follows best practices for reducing cognitive load: it offers consistent & simple APIs, it minimizes the number of user actions required for common use cases, and it provides clear and actionable feedback upon user error.
- **Modularity.** A model is understood as a sequence or a graph of standalone, fully-configurable modules that can be plugged together with as few restrictions as possible. In particular, neural layers, cost functions, optimizers, initialization schemes, activation functions, regularization schemes are all standalone modules that you can combine to create new models.
- **Easy extensibility.** New modules are simple to add (as new classes and functions), and existing modules provide ample examples. To be able to easily create new modules allows for total expressiveness, making Keras suitable for advanced research.
- **Work with Python**. No separate models configuration files in a declarative format. Models are described in Python code, which is compact, easier to debug, and allows for ease of extensibility.

TensorFlow

TensorFlow is a Python library for fast numerical computing created and released by Google.It is a foundation library that can be used to create Deep Learning models directly or by using wrapper libraries that simplify the process built on top of TensorFlow

TensorFlow is an open source library for fast numerical computing.It was created and is maintained by Google and released under the Apache 2.0 open source license. The API is nominally for the Python programming language, although there is access to the underlying C++ API.

Unlike other numerical libraries intended for use in Deep Learning like Theano, TensorFlow was designed for use both in research and development and in production systems, not least RankBrain in Google search and the fun DeepDream project.
It can run on single CPU systems, GPUs as well as mobile devices and large scale distributed systems of hundreds of machines.

. Computation is described in terms of data flow and operations in the structure of a directed graph.

- **Nodes**: Nodes perform computation and have zero or more inputs and outputs. Data that moves between nodes are known as tensors, which are multi-dimensional arrays of real values.
- **Edges**: The graph defines the flow of data, branching, looping and updates to state. Special edges can be used to synchronize behavior within the graph, for example waiting for computation on a number of inputs to complete.
- **Operation**: An operation is a named abstract computation which can take input attributes and produce output attributes. For example, you could define an add or multiply operation.

Machine Learning Applications

Virtual Personal Assistants

Siri, Google Now, Alexa are some of the common examples of virtual personal assistants. These applications assist in finding information, when asked over voice. All that is needed is activating them and asking questions like for example “What are my appointments for today?”, “What are the flights from Delhi to New York”. For answering such queries, the application looks out for the information, recalls your previous queries, and accesses other resources to collect relevant information. You can even tell these assistants to do certain tasks like “Set an alarm for 5.30 AM next morning”, “Remind me to visit Passport office tomorrow at 10.30 am”.

Traffic Congestion Analysis and Predictions

GPS navigation services monitor the user’s location and velocities and use them to build a map of current traffic. This helps in preventing the traffic congestions. Machine learning in such scenarios helps to estimate the regions where congestion can be found based on previous records.

Automated Video Surveillance

Video surveillance systems nowadays are powered by AI and machine learning is the technology behind this that makes it possible to detect and prevent crimes before they occur. They track odd and suspicious behavior of people and sends alerts to human attendants, who can ultimately help accidents and crimes.

Social Media

Facebook continuously monitors the friends that you connect with, your interests, workplace, or a group that you share with someone etc. Based on continuous learning, a list of Facebook users is given as friend suggestions.

Face Recognition

You upload a picture of you with a friend and Facebook instantly recognizes that friend. Machine learning works at the core of Computer Vision, which is a technique to extract useful information from images and videos. Pinterest uses computer vision to identify objects or pins in the images and recommend similar pins to its users.

Email Spam and Malware Filtering

Machine learning is being extensively used in spam detection and malware filtering and the databases of such spams and malwares keep on getting updated so these are handled efficiently.

Online Customer Support

In several websites nowadays, there is an option to chat with customer support representative while users are navigating the site. In most of the cases, instead of a real executive, you talk to a chatbot. These bots extract information from the website and provide it to the customers to assist them. Over a period of time, the chatbots learn to understand the user queries better and serve them with better answers, and this is made possible by machine learning algorithms.

Refinement of Search Engine Results

Google and similar search engines are using machine learning to improve the search results for their users. Every time a search is executed, the algorithms at the backend keep a watch at how the users respond to the results. Depending on the user responses, the algorithms working at the backend improve the search results.

Product Recommendations

If a user purchases or searches for a product online, he/she keeps on receiving emails for shopping suggestions and ads about that product. Based on previous user behavior, on a website/app, past purchases, items liked or added to cart, brand preferences etc., the product recommendations are sent to the user.

Detection of Online frauds

Machine learning is used to track monetary frauds online. For example - Paypal is using ML to prevent money laundering. The company is using a set of tools that helps them compare millions of transactions and make a distinction between legal or illegal transactions taking place between the buyers and sellers.

Thank You