**Python:**

from math import pi  
import numpy  
import numpy as np  
A cheat sheet that covers almost all parts of python required for data analysis and visualization. Please access at following url. <http://www.utc.fr/~jlaforet/Suppl/python-cheatsheets.pdf>

**Lambda Expression:**

Using `def` (old way).

def old\_add(a, b):

return a + b

# Using `lambda` (new way).

new\_add = lambda a, b: a + b

old\_add(10, 5) == new\_add(10, 5)

>> True

**Numpy:**

import numpy as np  
The NumPy library is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.

Some useful numpy commands:

Nan = np.nan  
np.random.random((2,2))  
b.astype(int) type conversion  
dim1size,dim2size..=ndarray1.shape np.where(cond, x, y)  
np.where(matArray>0, 1, -1)  
ndarray1.transpose() or ndarray1.T  
or ndarray1.swapaxes(0,1) => Transpose  
Math.sqrt() works only on scaler so np.sqrt(seq)is solution for vectored data  
np.unique(ndarr1)  
np.in1d(ndarr1, [2,3,6])  
other set methods are intersect1d(), union1d, setdiff1d(), setxor1d()  
np.random.normal(size=(3,3))  
ndarr1.any() at least one value true  
ndarr1.all() all values true  
ndarr1.sort() or np.sort(ndarra1)  
np.concatanate,   
np.append, delete, resize, insert, hstack, vstack etc are other np array functions.  
np.concatenate((a,d),axis=0)  
Axis 0 means column, 1=row

**Scipy:**

from scipy import linalg, sparse  
The SciPy library is one of the core packages for scientific computing that provides mathematical algorithms and convenience functions built on the NumPy extension of Python.

linalg.inv(A) Inverse  
linalg.pinv(C)  
sparse.linalg.inv(I)  
sparse.linalg.norm(I)  
sparse.linalg.spsolve(H,I)  
la, v = linalg.eig(A)  
  
la, v = sparse.linalg.eigs(F,1) Eigenvalues and eigenvectors  
sparse.linalg.svds(H, 2) SVD

**Pandas:**

import pandas as pd  
from pandas import Series, DataFrame  
The Pandas library is built on NumPy and provides easy-to-use data structures.

help(pd.Series.loc), df.iloc([0],[0])  
df.cumsum(), df.rank(), df.count(),  
df.drop('Country', axis=1)  
s.drop(['a', 'c']),s.add(s3, fill\_value=0)  
df.describe(),Df.pct\_change(),df.corr

**Applying Function in Pandas**

f = lambda x: x\*2  
df.apply(f) Apply function  
df.applymap(f) Apply function element-wise

**Dealing with missing Data:**

series.isnull()

series.notisnull()

df.dropna()

Df.dropna(how=’all’)

Df.dropna(axis=1)

Df.dropna(thresh=2)

Df.fillna(1)

Df.fillna(0, inplace=True)

**Index & Column Hierarchy for DF:**

df2=DataFrame(np.arange(16).reshape(4, 4), index=[[‘a’,’a’,’b’,’b’], [1,2,1,2]], columns =[[NY, 'NY', 'LA', ‘SF’], [cold', 'hot', 'hot', ‘cold’]]

For data frames with hierarchy:

Df.sortlevel(1), Df.sum(level = temp, axis =1)

**Pandas data Processing CSV and Excel files:**

df=pd.read\_csv(‘lec25.csv’, header = None), pd.read\_table(‘lec25.csv’, sep = ‘,’ header = None),  
df1.to\_csv(‘myfile.csv’)

Import sys

dframe.to\_csv(sys.stdout, sep = ‘\_’, columns = [0,1,2])

pip install xlrd, openpyxl  
xfile=pd.ExcelFile(‘ExclFile’)  
df=xfile.parse(‘sheet1’)

**JSON with Python**

Import json  
data = json.load(json\_obj)  
json.dump(data)  
df = DataFrame(data[‘diet’])

**HTML/XML with Python**

pip install html5lib, beautiful-soup  
from pandas Import read\_html  
df = pd.io.html.read\_html(url)  
df.columns.values

**DataFrame Operations:**

Merging  
pd.merge(dframe1,dframe2)  
pd.merge(dframe1,dframe2,on=’key’)  
pd.merge(df1,df2,on=’key’,how=’left’)  
pd.merge(f1,f2,on=[‘k1’,’k2’],how=’outer’)

left\_on=[‘Key1’,’Key2’]  
right\_index=True for merging on index.

df1.join(df2)  
Concatanation  
np.concatanate(arr1,arr2,axis=0/1)  
pd.concat([ser1, ser2])  
pd.concat([ser1, ser2],axis=1)=df  
pd.concat(df1,df2,ignore\_index=True)  
ser1.combine\_first(ser2)  
df1.combine\_first(df2)  
Reshaping  
df.stack(), df.unstack(‘col1’), df.unstack().stack(), df.stack(dropna=False)  
Pivoting  
df.pivot(‘row’,‘col’,’value’)  
Duplicates  
df.duplicated(), df.drop\_duplicates()  
df.drop\_duplicates([‘Key1’])  
df.drop\_duplicates([‘Key1’], take\_last=True)  
Mapping  
df[‘city’].map(state\_dict)  
Replace  
ser1.replace([a,b],[x,z])  
ser1.replace({4:np.nan})

**Scikit-Learn:**

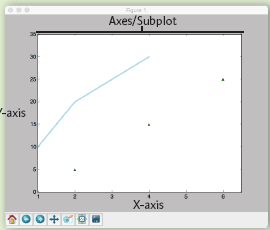
from sklearn.preprocessing import Normalizer

Scikit-learn is an open source Python library that implements a range of machine learning, preprocessing, cross-validation and visualization algorithms using a unified interface  
from sklearn import neighbors, datasets, preprocessing  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import accuracy\_score

iris = datasets.load\_iris()  
X, y = iris.data[:, :2], iris.target  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state=33)  
scaler = preprocessing.StandardScaler().fit(X\_train)  
X\_train = scaler.transform(X\_train)  
X\_test = scaler.transform(X\_test)  
knn = neighbors.KNeighborsClassifier(n\_neighbors=5)  
knn.fit(X\_train, y\_train)  
y\_pred = knn.predict(X\_test)  
accuracy\_score(y\_test, y\_pred)

**Matplotlib:**

Matplotlib is a Python 2D plotting library which produces publication-quality figures in a variety of hardcopy formats and interactive environments across platforms.  
  
import matplotlib.pyplot as plt  
x = [1,2,3,4]  
y = [10,20,25,30]  
fig = plt.figure()  
ax = fig.add\_subplot(111)  
ax.plot(x, y, color='lightblue', linewidth=3)  
ax.scatter([2,4,6],  
[5,15,25],  
color='darkgreen',  
marker='^')  
ax.set\_xlim(1, 6.5)  
plt.savefig('foo.png')  
plt.show()  
  
plt.savefig('foo.png')  
plt.savefig('foo.png', transparent=True)



**Seaborn:**

The Python visualization library Seaborn is based on matplotlib and provides a high-level interface for drawing attractive statistical graphics.

import matplotlib.pyplot as plt  
import seaborn as sns  
  
import matplotlib.pyplot as plt  
import seaborn as sns  
tips = sns.load\_dataset("tips")  
sns.set\_style("whitegrid")  
g = sns.lmplot(x="tip",  
y="total\_bill",  
data=tips,  
aspect=2)  
g = (g.set\_axis\_labels("Tip","Total bill(USD)").  
set(xlim=(0,10),ylim=(0,100)))  
plt.title("title")  
plt.show(g)

**Sampel Tensorflow code:**

import tensorflow as tf  
mnist = tf.keras.datasets.mnist  
  
(x\_train, y\_train),(x\_test, y\_test) = mnist.load\_data()  
x\_train, x\_test = x\_train / 255.0, x\_test / 255.0  
  
model = tf.keras.models.Sequential([  
  tf.keras.layers.Flatten(),  
  tf.keras.layers.Dense(512, activation=tf.nn.relu),  
  tf.keras.layers.Dropout(0.2),  
  tf.keras.layers.Dense(10, activation=tf.nn.softmax)  
])  
model.compile(optimizer='adam',  
              loss='sparse\_categorical\_crossentropy',  
              metrics=['accuracy'])  
  
model.fit(x\_train, y\_train, epochs=5)  
model.evaluate(x\_test, y\_test)

**Sampel OpenCV code:**

import cv2

import numpy as np

from matplotlib import pyplot as plt

img = cv2.imread('watch.jpg',cv2.IMREAD\_GRAYSCALE)

cv2.imshow('image',img)

cv2.waitKey(0)

cv2.destroyAllWindows()

**Data Slicing: -**

**PCA in python: pca.explained\_variance\_ratio\_**

import numpy as np

from sklearn.decomposition import PCA

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.preprocessing import scale

%matplotlib inline

**#Load data set**

data = pd.read\_csv('Big\_Mart\_PCA.csv')

**#convert it to numpy arrays**

X=data.values

**#Scaling the values**

X = scale(X)

pca = PCA(n\_components=44)

pca.fit(X)

**#The amount of variance that each PC explains**

var= pca.explained\_variance\_ratio\_

**#Cumulative Variance explains**

var1=np.cumsum(np.round(pca.explained\_variance\_ratio\_, decimals=4)\*100)

print var1

[ 10.37 17.68 23.92 29.7 34.7 39.28 43.67 46.53 49.27

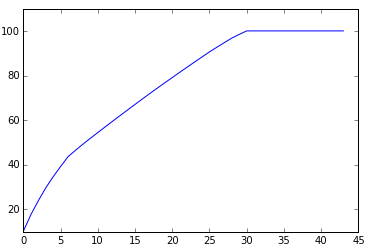
51.92 54.48 57.04 59.59 62.1 64.59 67.08 69.55 72.

74.39 76.76 79.1 81.44 83.77 86.06 88.33 90.59 92.7

94.76 96.78 98.44 100.01 100.01 100.01 100.01 100.01 100.01

100.01 100.01 100.01 100.01 100.01 100.01 100.01 100.01]

plt.plot(var1)



**#Looking at above plot I'm taking 30 variables**

pca = PCA(n\_components=30)

pca.fit(X)

X1=pca.fit\_transform(X)

print X1

**python Dictionary**