DECISIONSTATS.COM

Python for R Users

By

Chandan Routray

As a part of internship at

www.decisionstats.com

Basic Commands

Functions	R	Python
Downloading and installing a package	`install.packages('name')`	`pip install name`
Load a package	`library('name')`	`import name as other_name`

Checking working directory	`getwd()`	`import os`
		`os.getcwd()`

Setting working directory	`setwd()`	`os.chdir()`
List files in a directory	`dir()`	`os.listdir()`
List all objects	`ls()`	`globals()`
Remove an object	`rm('name')`	`del('object')`

Data Frame Creation

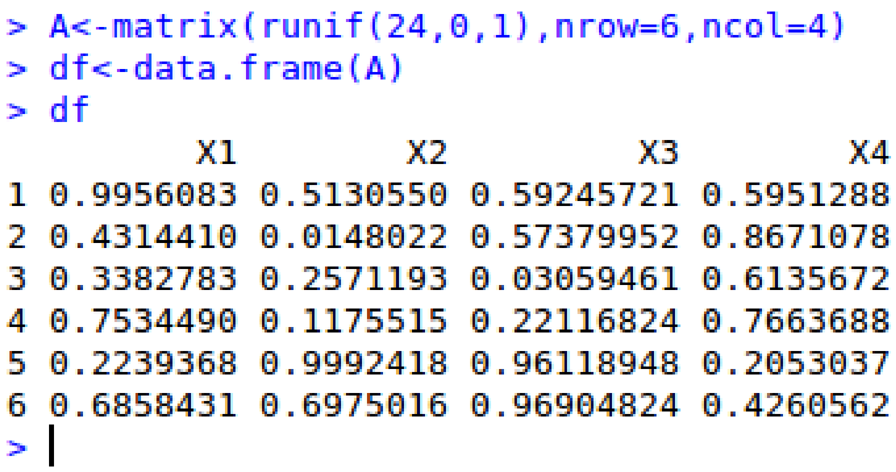
	R	**Python** (Using pandas package*)

Creating a data frame “df” of dimension 6x4 (6 rows and 4 columns) containing random numbers	`A<-`
	`matrix(runif(24,0,1),nrow=6,ncol=4)`
	`df<-data.frame(A)`
	
	Here,
	• *runif function generates 24 random numbers between 0 to 1*
	• *matrix function creates a matrix from those random numbers, nrow and ncol sets the numbers of rows and columns to the matrix*
	• *data.frame converts the matrix to data frame*
		`import numpy as np`
		`import pandas as pd`
		`A=np.random.randn(6,4)`
		`df=pd.DataFrame(A)`
		
		Here,
		• *np.random.randn generates a matrix of 6 rows and 4 columns; this function is a part of numpy** library*
		• *pd.DataFrame converts the matrix in to a data frame*

**To install Pandas library visit: http://pandas.pydata.org/; To import Pandas library type*: import pandas as pd;
***To import Numpy library type*: import numpy as np;

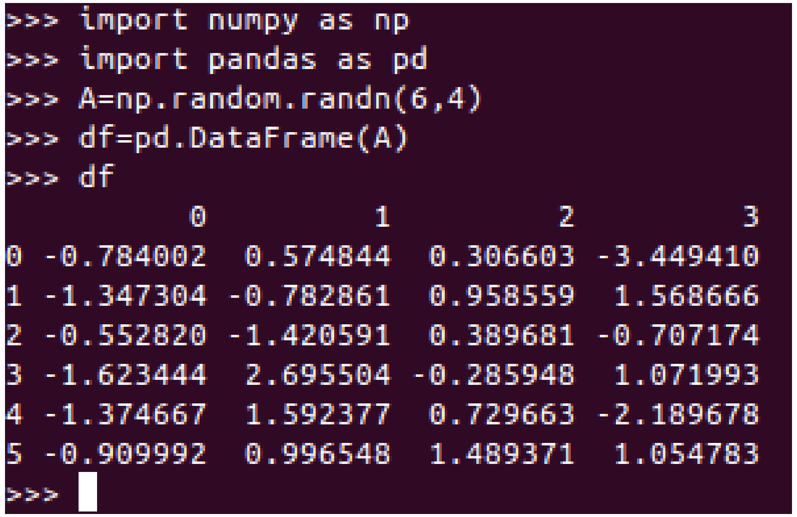
Data Frame Creation

R



```
> A<-matrix(runif(24,0,1),nrow=6,ncol=4)
> df<-data.frame(A)
> df
         X1        X2         X3        X4
1 0.9956083 0.5130550 0.59245721 0.5951288
2 0.4314410 0.0148022 0.57379952 0.8671078
3 0.3382783 0.2571193 0.03059461 0.6135672
4 0.7534490 0.1175515 0.22116824 0.7663688
5 0.2239368 0.9992418 0.96118948 0.2053037
6 0.6858431 0.6975016 0.96904824 0.4260562
> 
```

Python



```
>>> import numpy as np
>>> import pandas as pd
>>> A=np.random.randn(6,4)
>>> df=pd.DataFrame(A)
>>> df
          0         1         2         3
0 -0.784002  0.574844  0.306603 -3.449410
1 -1.347304 -0.782861  0.958559  1.568666
2 -0.552820 -1.420591  0.389681 -0.707174
3 -1.623444  2.695504 -0.285948  1.071993
4 -1.374667  1.592377  0.729663 -2.189678
5 -0.909992  0.996548  1.489371  1.054783
>>> 
```

Data Frame: Inspecting and Viewing Data

	R	**Python** (Using pandas package*)

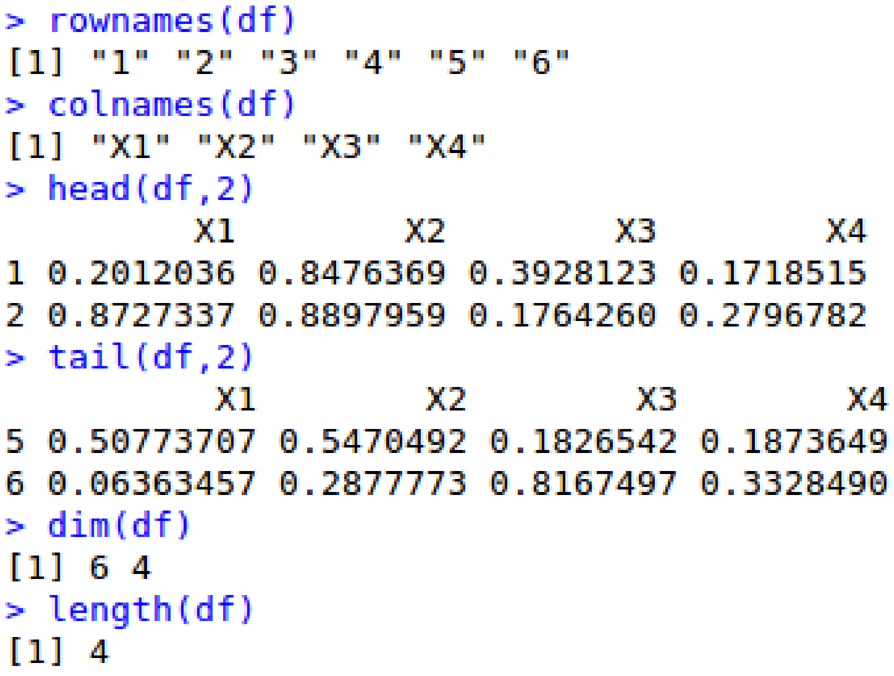
Getting the names of rows and columns of data frame “df”	`rownames(df)`
	returns the name of the rows
	`colnames(df)`
	returns the name of the columns
		`df.index`
		returns the name of the rows
		`df.columns`
		returns the name of the columns

Seeing the top and bottom “x” rows of the data frame “df”	`head(df,x)`
	returns top x rows of data frame
	`tail(df,x)`
	returns bottom x rows of data frame
		df.head(x)
		returns top x rows of data frame
		`df.tail(x)`
		returns bottom x rows of data frame

Getting dimension of data frame “df”	`dim(df)` *returns in this format : rows, columns*	`df.shape` *returns in this format : (rows, columns)*
Length of data frame “df”	`length(df)` *returns no. of columns in data frames*	`len(df)` *returns no. of columns in data frames*

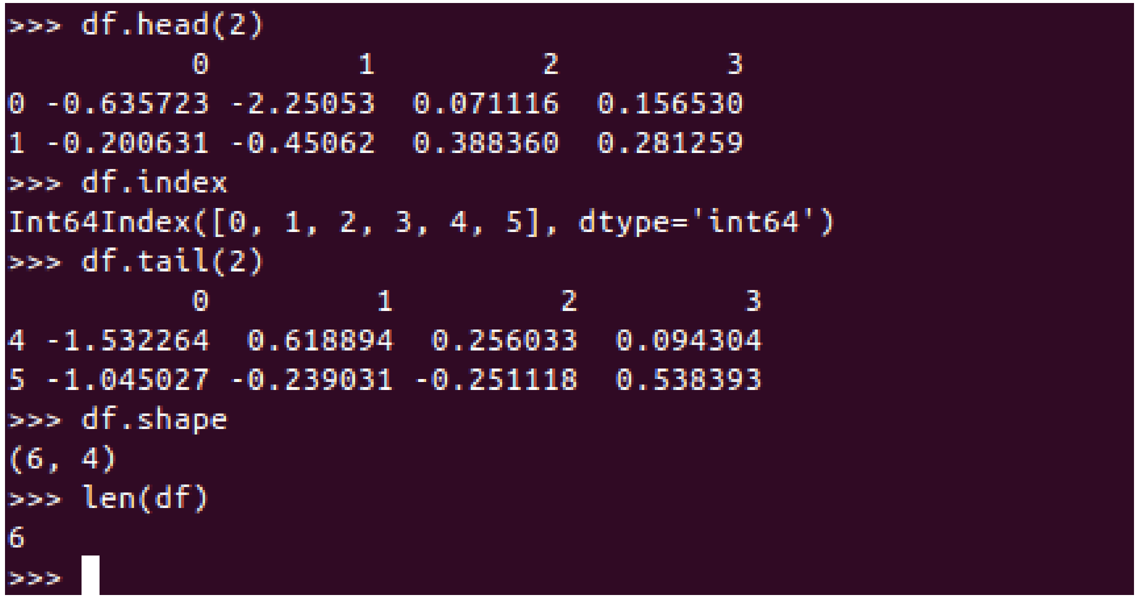
Data Frame: Inspecting and Viewing Data

R



```
> rownames(df)
[1] "1" "2" "3" "4" "5" "6"
> colnames(df)
[1] "X1" "X2" "X3" "X4"
> head(df,2)
         X1        X2        X3        X4
1 0.2012036 0.8476369 0.3928123 0.1718515
2 0.8727337 0.8897959 0.1764260 0.2796782
> tail(df,2)
          X1        X2        X3        X4
5 0.50773707 0.5470492 0.1826542 0.1873649
6 0.06363457 0.2877773 0.8167497 0.3328490
> dim(df)
[1] 6 4
> length(df)
[1] 4
```

Python



```
>>> df.head(2)
          0         1         2         3
0 -0.635723 -2.25053  0.071116  0.156530
1 -0.200631 -0.45062  0.388360  0.281259
>>> df.index
Int64Index([0, 1, 2, 3, 4, 5], dtype='int64')
>>> df.tail(2)
          0         1         2         3
4 -1.532264  0.618894  0.256033  0.094304
5 -1.045027 -0.239031 -0.251118  0.538393
>>> df.shape
(6, 4)
>>> len(df)
6
>>> 
```

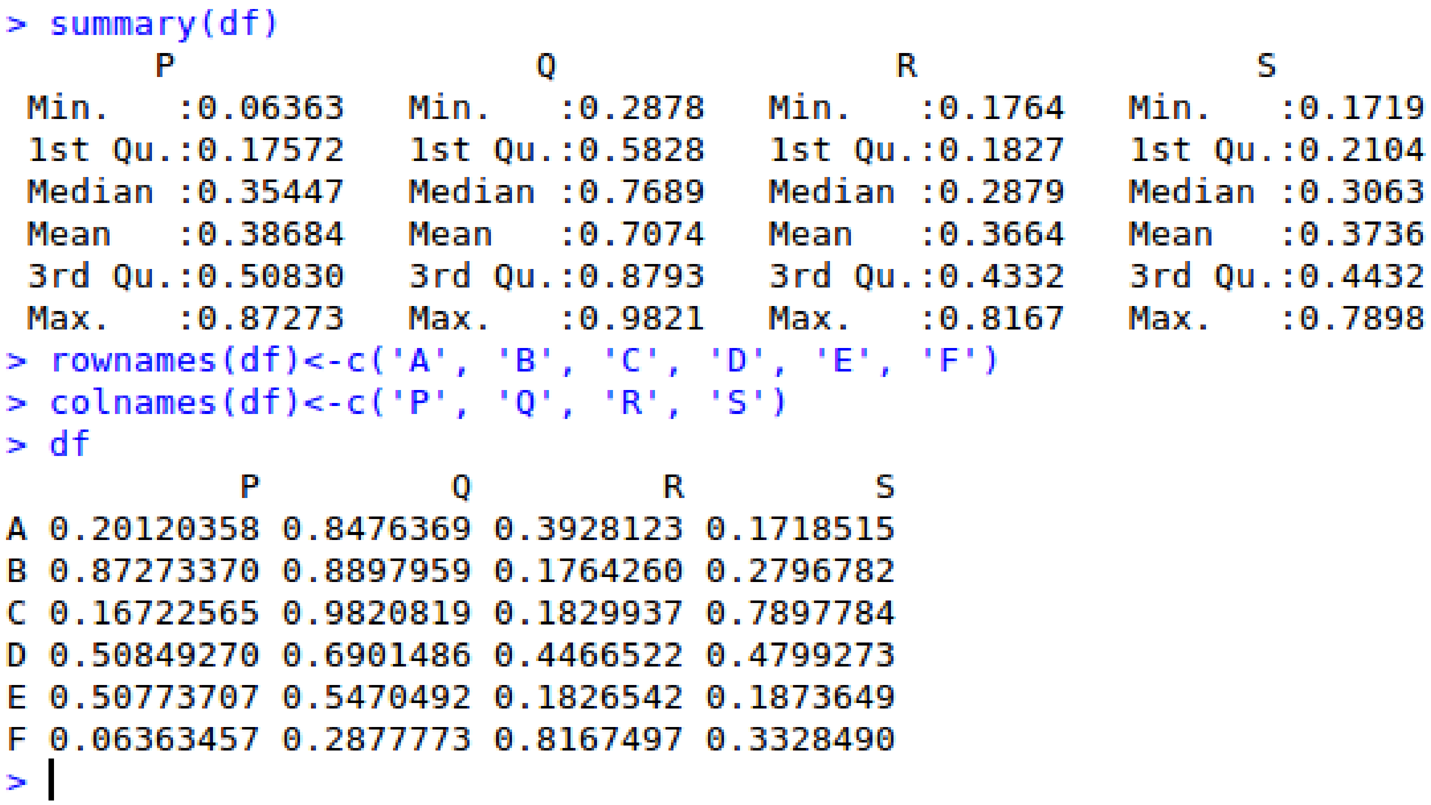
Data Frame: Inspecting and Viewing Data

	R	Python (Using pandas package*)
Getting quick summary(like mean, std. deviation etc.) of data in the data frame “df”	`summary(df)` *returns mean, median , maximum, minimum, first quarter and third quarter*	`df.describe()` *returns count, mean, standard deviation, maximum, minimum, 25%, 50% and 75%*

Setting row names and columns names of the data frame “df”	`rownames(df)=c(“A”, ”B”, “C”, ”D”, “E”, ”F”)`
	set the row names to A, B, C, D and E
	`colnames=c(“P”, ”Q”, “R”, ”S”)`
	set the column names to P, Q, R and S
		`df.index=[“A”, ”B”, “C”, ”D”, “E”, ”F”]`
		set the row names to A, B, C, D and E
		`df.columns=[“P”, ”Q”, “R”, ”S”]`
		set the column names to P, Q, R and S

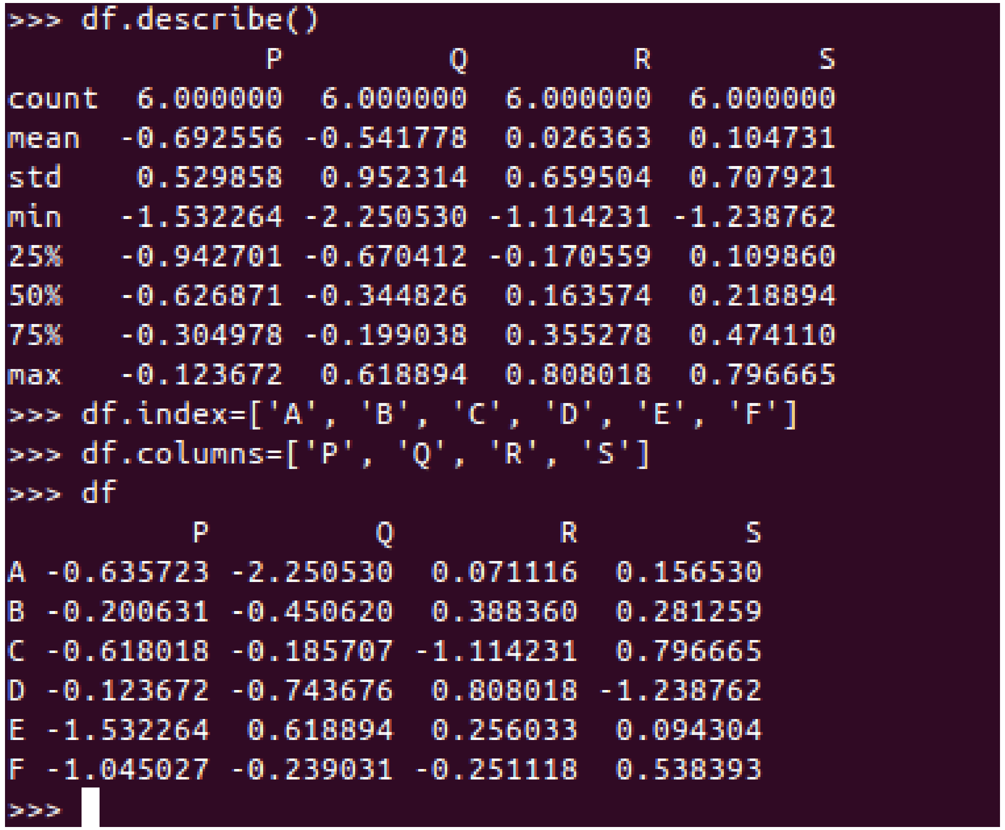
Data Frame: Inspecting and Viewing Data

R



```
> summary(df)
       P                 Q                R                S         
 Min.   :0.06363   Min.   :0.2878   Min.   :0.1764   Min.   :0.1719  
 1st Qu.:0.17572   1st Qu.:0.5828   1st Qu.:0.1827   1st Qu.:0.2104  
 Median :0.35447   Median :0.7689   Median :0.2879   Median :0.3063  
 Mean   :0.38684   Mean   :0.7074   Mean   :0.3664   Mean   :0.3736  
 3rd Qu.:0.50830   3rd Qu.:0.8793   3rd Qu.:0.4332   3rd Qu.:0.4432  
 Max.   :0.87273   Max.   :0.9821   Max.   :0.8167   Max.   :0.7898  
> rownames(df)<-c('A', 'B', 'C', 'D', 'E', 'F')
> colnames(df)<-c('P', 'Q', 'R', 'S')
> df
           P         Q         R         S
A 0.20120358 0.8476369 0.3928123 0.1718515
B 0.87273370 0.8897959 0.1764260 0.2796782
C 0.16722565 0.9820819 0.1829937 0.7897784
D 0.50849270 0.6901486 0.4466522 0.4799273
E 0.50773707 0.5470492 0.1826542 0.1873649
F 0.06363457 0.2877773 0.8167497 0.3328490
> 
```

Python



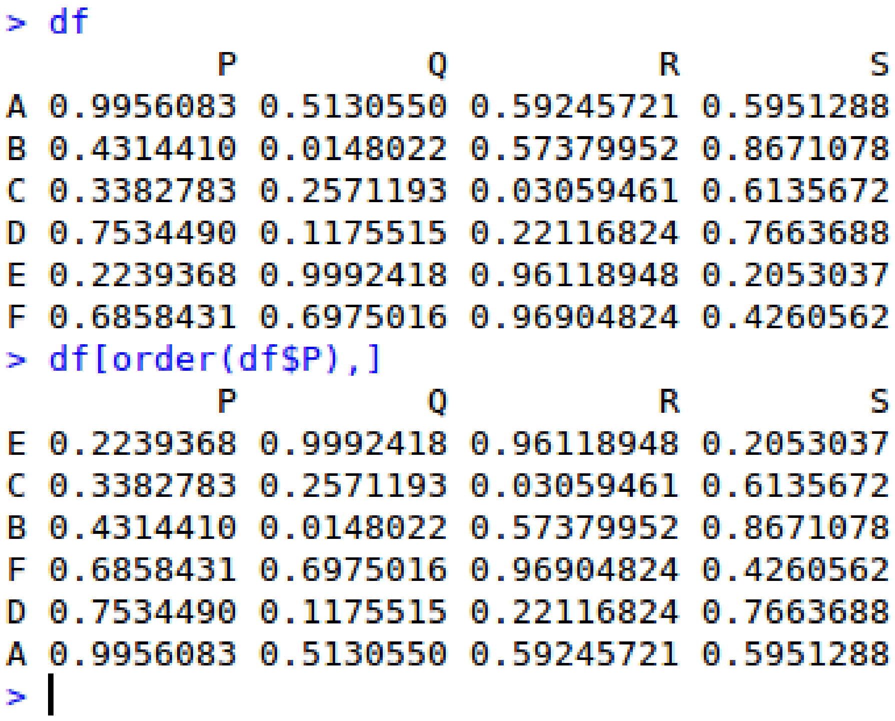
```
>>> df.describe()
              P         Q         R         S
count  6.000000  6.000000  6.000000  6.000000
mean  -0.692556 -0.541778  0.026363  0.104731
std    0.529858  0.952314  0.659504  0.707921
min   -1.532264 -2.250530 -1.114231 -1.238762
25%   -0.942701 -0.670412 -0.170559  0.109860
50%   -0.626871 -0.344826  0.163574  0.218894
75%   -0.304978 -0.199038  0.355278  0.474110
max   -0.123672  0.618894  0.808018  0.796665
>>> df.index=['A', 'B', 'C', 'D', 'E', 'F']
>>> df.columns=['P', 'Q', 'R', 'S']
>>> df
          P         Q         R         S
A -0.635723 -2.250530  0.071116  0.156530
B -0.200631 -0.450620  0.388360  0.281259
C -0.618018 -0.185707 -1.114231  0.796665
D -0.123672 -0.743676  0.808018 -1.238762
E -1.532264  0.618894  0.256033  0.094304
F -1.045027 -0.239031 -0.251118  0.538393
>>> 
```

Data Frame: Sorting Data

	R	**Python** (Using pandas package*)
Sorting the data in the data frame “df” by column name “P”	`df[order(df$P),]`	`df.sort(['P'])`

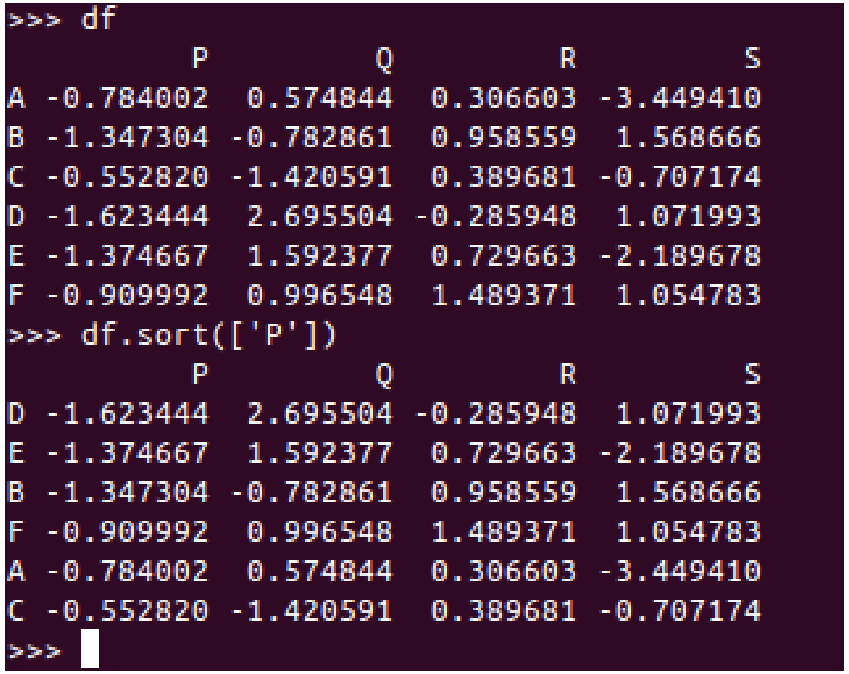
Data Frame: Sorting Data

R



```
> df
          P         Q          R         S
A 0.9956083 0.5130550 0.59245721 0.5951288
B 0.4314410 0.0148022 0.57379952 0.8671078
C 0.3382783 0.2571193 0.03059461 0.6135672
D 0.7534490 0.1175515 0.22116824 0.7663688
E 0.2239368 0.9992418 0.96118948 0.2053037
F 0.6858431 0.6975016 0.96904824 0.4260562
> df[order(df$P),]
          P         Q          R         S
E 0.2239368 0.9992418 0.96118948 0.2053037
C 0.3382783 0.2571193 0.03059461 0.6135672
B 0.4314410 0.0148022 0.57379952 0.8671078
F 0.6858431 0.6975016 0.96904824 0.4260562
D 0.7534490 0.1175515 0.22116824 0.7663688
A 0.9956083 0.5130550 0.59245721 0.5951288
> |
```

Python



```
>>> df
          P         Q         R         S
A -0.784002  0.574844  0.306603 -3.449410
B -1.347304 -0.782861  0.958559  1.568666
C -0.552820 -1.420591  0.389681 -0.707174
D -1.623444  2.695504 -0.285948  1.071993
E -1.374667  1.592377  0.729663 -2.189678
F -0.909992  0.996548  1.489371  1.054783
>>> df.sort(['P'])
          P         Q         R         S
D -1.623444  2.695504 -0.285948  1.071993
E -1.374667  1.592377  0.729663 -2.189678
B -1.347304 -0.782861  0.958559  1.568666
F -0.909992  0.996548  1.489371  1.054783
A -0.784002  0.574844  0.306603 -3.449410
C -0.552820 -1.420591  0.389681 -0.707174
>>>
```

Data Frame: Data Selection

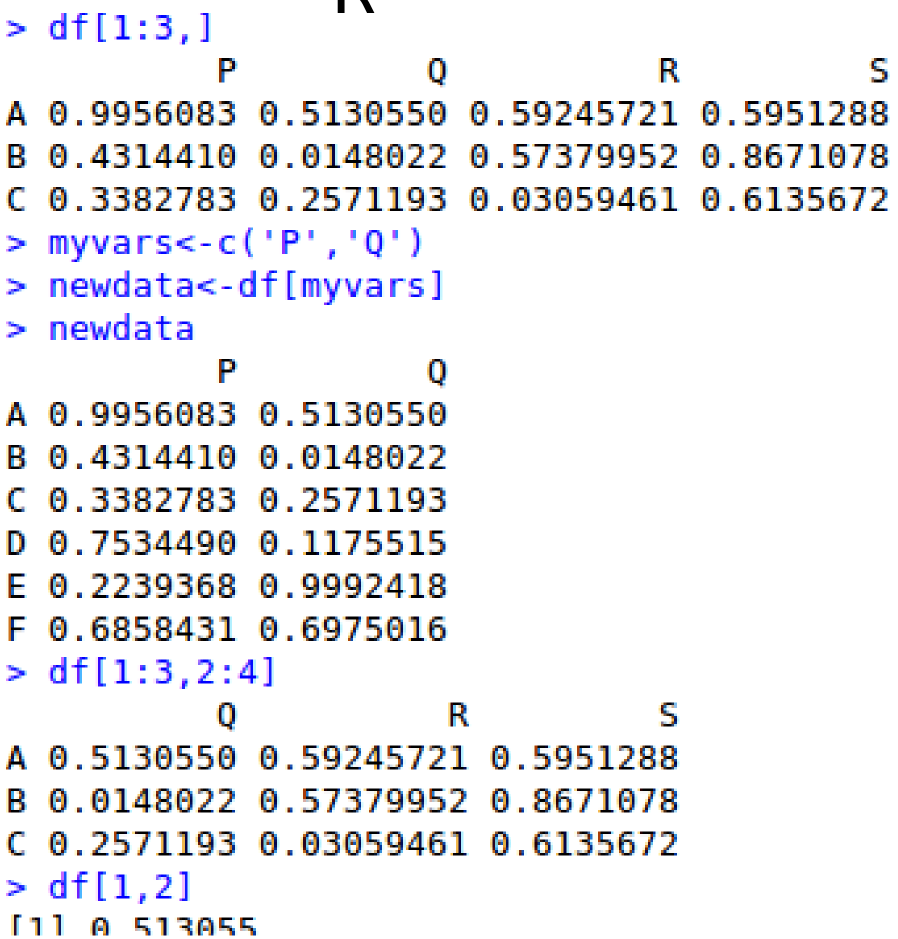
	R	**Python** (Using pandas package*)
Slicing the rows of a data frame from row no. “x” to row no. “y”(including row x and y)	`df[x:y,]`	`df[x-1:y]` *Python starts counting from 0*

Slicing the columns name “x”,”Y” etc. of a data frame “df”	`myvars <- c(“X”,”Y”)`
	`newdata <- df[myvars]`
		`df.loc[:,[‘X’,’Y’]]`

Selecting the the data from row no. “x” to “y” and column no. “a” to “b”	`df[x:y,a:b]`	`df.iloc[x-1:y,a-1,b]`
Selecting the element at row no. “x” and column no. “y”	`df[x,y]`	`df.iat[x-1,y-1]`

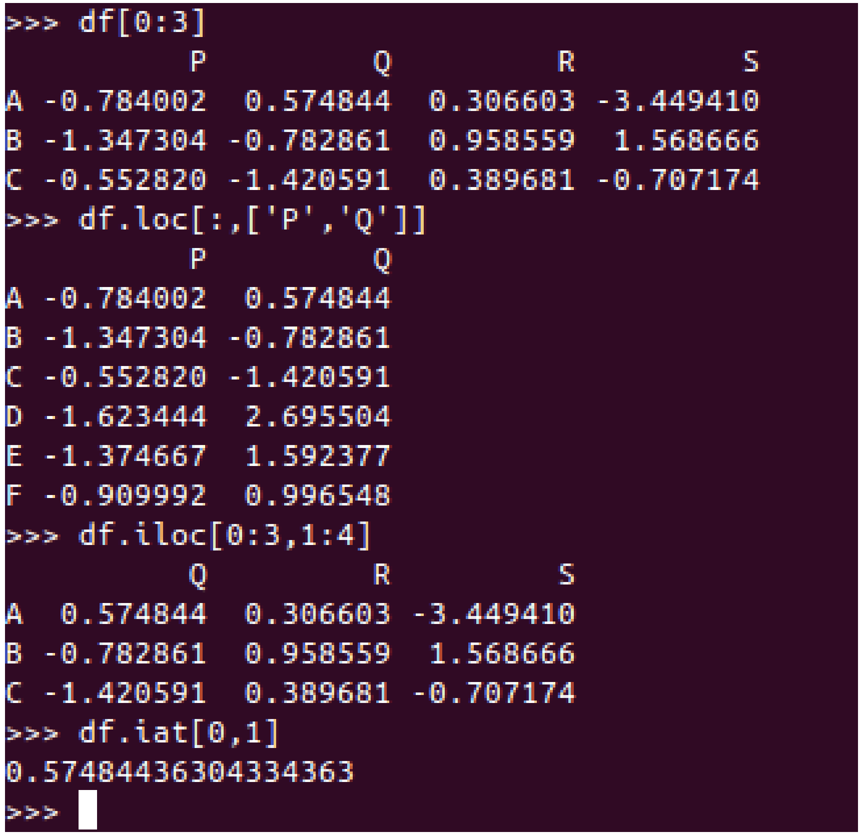
Data Frame: Data Selection

R



```
> df[1:3,]
          P         Q          R         S
A 0.9956083 0.5130550 0.59245721 0.5951288
B 0.4314410 0.0148022 0.57379952 0.8671078
C 0.3382783 0.2571193 0.03059461 0.6135672
> myvars<-c('P','Q')
> newdata<-df[myvars]
> newdata
          P         Q
A 0.9956083 0.5130550
B 0.4314410 0.0148022
C 0.3382783 0.2571193
D 0.7534490 0.1175515
E 0.2239368 0.9992418
F 0.6858431 0.6975016
> df[1:3,2:4]
          Q          R         S
A 0.5130550 0.59245721 0.5951288
B 0.0148022 0.57379952 0.8671078
C 0.2571193 0.03059461 0.6135672
> df[1,2]
[1] 0.513055
```

Python

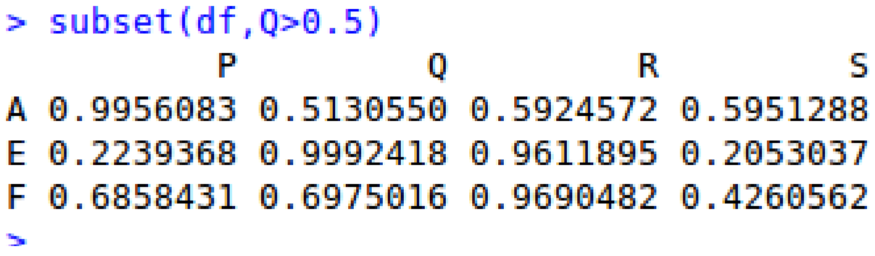


```
>>> df[0:3]
          P         Q         R         S
A -0.784002  0.574844  0.306603 -3.449410
B -1.347304 -0.782861  0.958559  1.568666
C -0.552820 -1.420591  0.389681 -0.707174
>>> df.loc[:,['P','Q']]
          P         Q
A -0.784002  0.574844
B -1.347304 -0.782861
C -0.552820 -1.420591
D -1.623444  2.695504
E -1.374667  1.592377
F -0.909992  0.996548
>>> df.iloc[0:3,1:4]
          Q         R         S
A  0.574844  0.306603 -3.449410
B -0.782861  0.958559  1.568666
C -1.420591  0.389681 -0.707174
>>> df.iat[0,1]
0.57484436304334363
>>> 
```

Data Frame: Data Selection

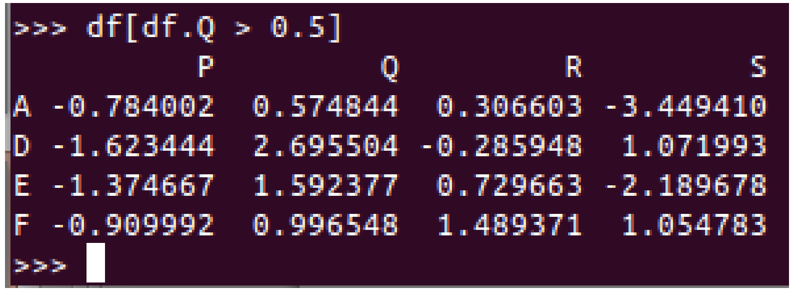
	R	**Python** (Using pandas package*)
Using a single column's values to select data, column name "A"	`subset(df,A>0)` *It will select the all the rows in which the corresponding value in column A of that row is greater than 0*	`df[df.A > 0]` *It will do the same as the R function*

R



```
> subset(df,Q>0.5)
          P         Q         R         S
A 0.9956083 0.5130550 0.5924572 0.5951288
E 0.2239368 0.9992418 0.9611895 0.2053037
F 0.6858431 0.6975016 0.9690482 0.4260562
>
```

Python



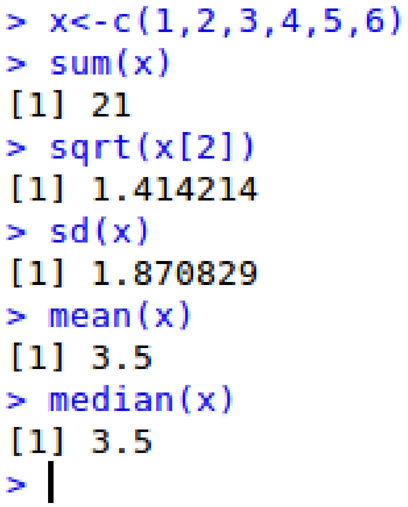
```
>>> df[df.Q > 0.5]
          P         Q         R         S
A -0.784002  0.574844  0.306603 -3.449410
D -1.623444  2.695504 -0.285948  1.071993
E -1.374667  1.592377  0.729663 -2.189678
F -0.909992  0.996548  1.489371  1.054783
>>> 
```

Mathematical Functions

Functions	R	Python *(import math and numpy library)*
Sum	`sum(x)`	`math.fsum(x)`
Square Root	`sqrt(x)`	`math.sqrt(x)`
Standard Deviation	`sd(x)`	`numpy.std(x)`
Log	`log(x)`	`math.log(x[,base])`
Mean	`mean(x)`	`numpy.mean(x)`
Median	`median(x)`	`numpy.median(x)`

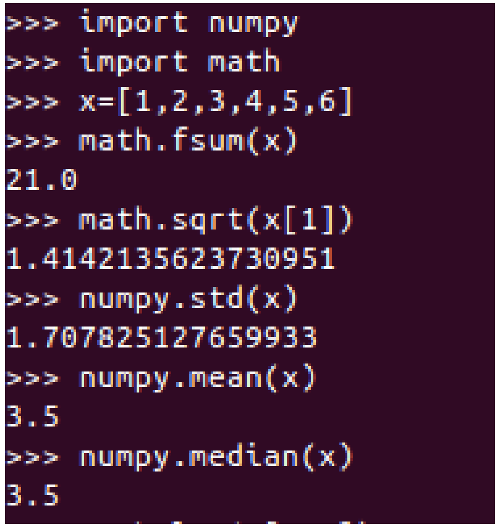
Mathematical Functions

R



```
> x<-c(1,2,3,4,5,6)
> sum(x)
[1] 21
> sqrt(x[2])
[1] 1.414214
> sd(x)
[1] 1.870829
> mean(x)
[1] 3.5
> median(x)
[1] 3.5
> 
```

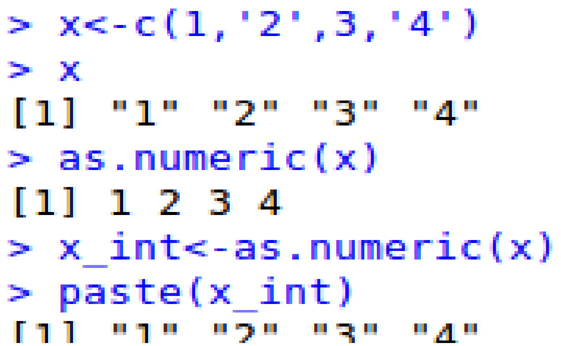
Python



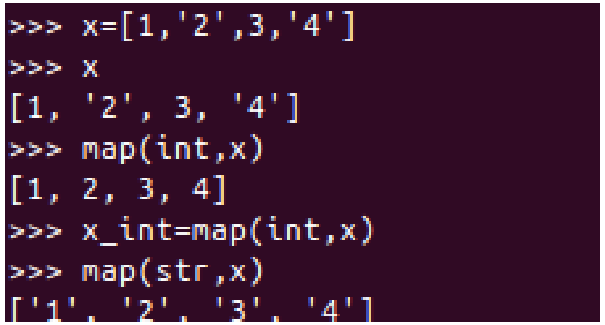
```
>>> import numpy
>>> import math
>>> x=[1,2,3,4,5,6]
>>> math.fsum(x)
21.0
>>> math.sqrt(x[1])
1.4142135623730951
>>> numpy.std(x)
1.707825127659933
>>> numpy.mean(x)
3.5
>>> numpy.median(x)
3.5
```

Data Manipulation

Functions	R	Python *(import math and numpy library)*
Convert character variable to numeric variable	`as.numeric(x)`	For a single value: `int(x), long(x), float(x)` For list, vectors etc.: `map(int,x), map(float,x)`
Convert factor/numeric variable to character variable	`paste(x)`	For a single value: `str(x)` For list, vectors etc.: `map(str,x)`
Check missing value in an object	`is.na(x)`	`math.isnan(x)`
Delete missing value from an object	`na.omit(`*`list`*`)`	`cleanedList = [x for x in list if str(x) != 'nan']`
Calculate the number of characters in character value	`nchar(x)`	`len(x)`



```
> x<-c(1,'2',3,'4')
> x
[1] "1" "2" "3" "4"
> as.numeric(x)
[1] 1 2 3 4
> x_int<-as.numeric(x)
> paste(x_int)
[1] "1" "2" "3" "4"
```

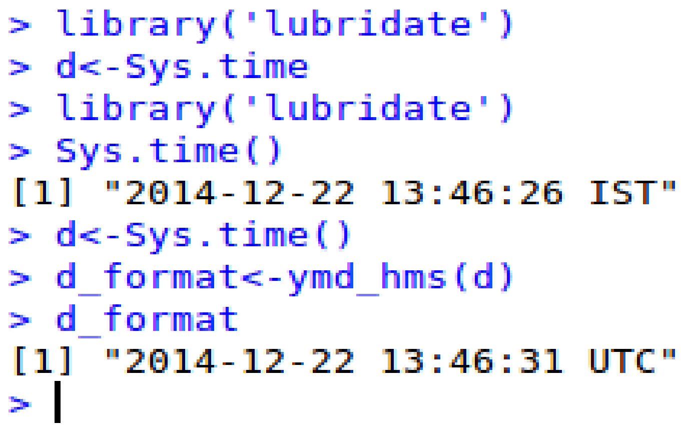


```
>>> x=[1,'2',3,'4']
>>> x
[1, '2', 3, '4']
>>> map(int,x)
[1, 2, 3, 4]
>>> x_int=map(int,x)
>>> map(str,x)
['1', '2', '3', '4']
```

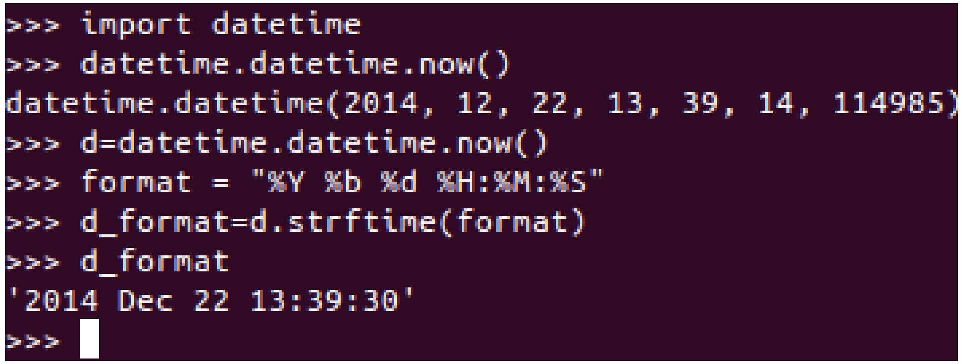
Date & Time Manipulation

Functions	R *(import lubridate library)*	Python *(import datetime library)*
Getting time and date at an instant	`Sys.time()`	`datetime.datetime.now()`

Parsing date and time in format:
YYYY MM DD HH:MM:SS
	`d<-Sys.time()`
	`d_format<-ymd_hms(d)`
		`d=datetime.datetime.now()`
		`format= “%Y %b %d  %H:%M:%S”`
		`d_format=d.strftime(format)`



```
> library('lubridate')
> d<-Sys.time
> library('lubridate')
> Sys.time()
[1] "2014-12-22 13:46:26 IST"
> d<-Sys.time()
> d_format<-ymd_hms(d)
> d_format
[1] "2014-12-22 13:46:31 UTC"
> 
```



```
>>> import datetime
>>> datetime.datetime.now()
datetime.datetime(2014, 12, 22, 13, 39, 14, 114985)
>>> d=datetime.datetime.now()
>>> format = "%Y %b %d %H:%M:%S"
>>> d_format=d.strftime(format)
>>> d_format
'2014 Dec 22 13:39:30'
>>> 
```

Data Visualization

Functions	**R**	**Python** *(import matplotlib library**)*

Scatter Plot variable1 vs variable2	`plot(variable1,variable2)`	`plt.scatter(variable1,variable2)`
		`plt.show()`

Boxplot for Var	`boxplot(Var)`	`plt.boxplot(Var)`
		`plt.show()`

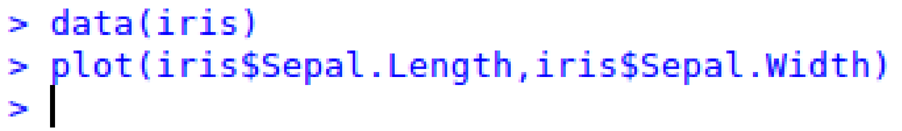
Histogram for Var	`hist(Var)`	`plt.hist(Var)`
		`plt.show()`

Pie Chart for Var	`pie(Var)`	`from pylab import *`
		`pie(Var)`
		`show()`

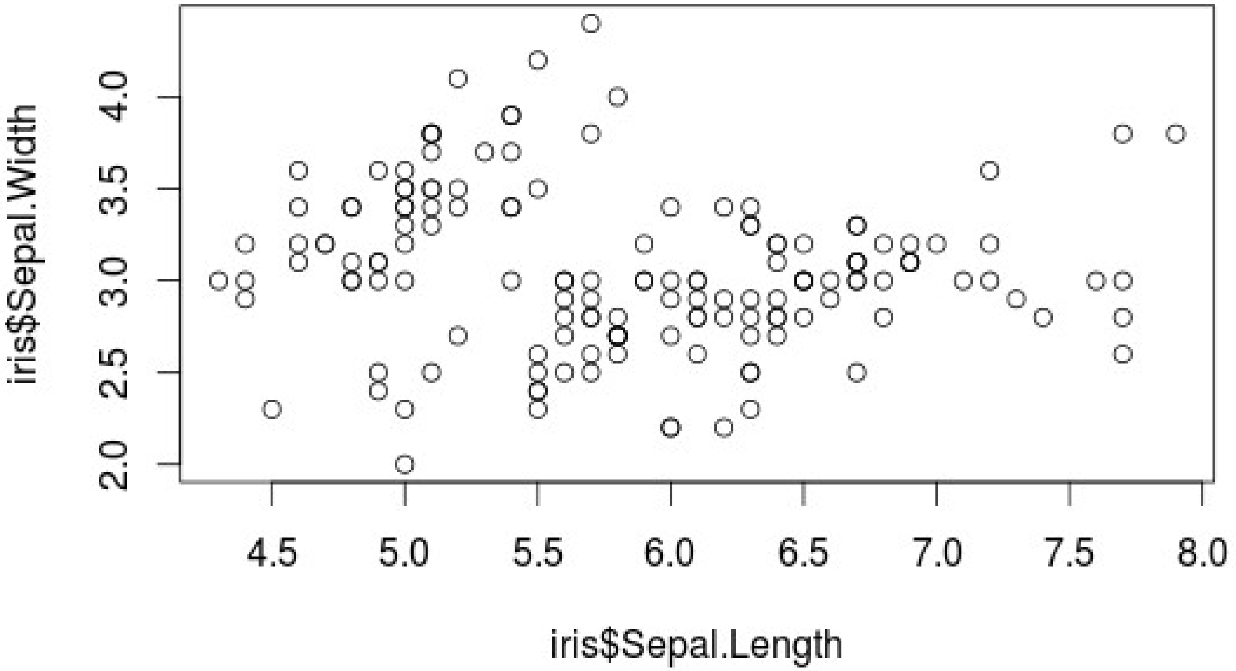
*** To import matplotlib library type: import matplotlib.pyplot as plt*

Data Visualization: Scatter Plot

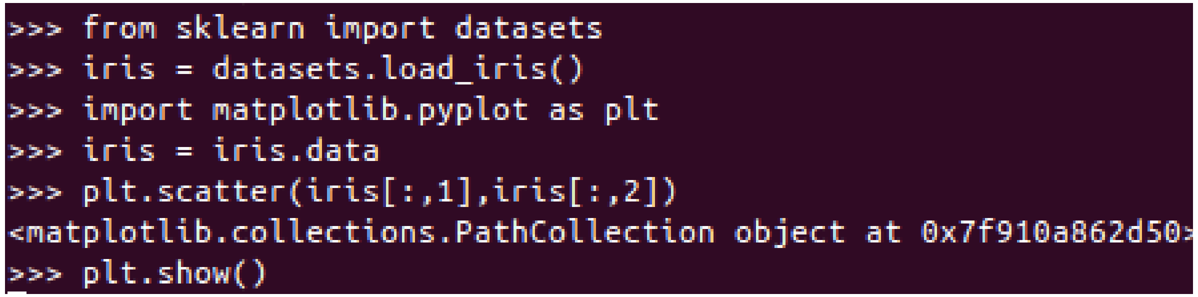
R



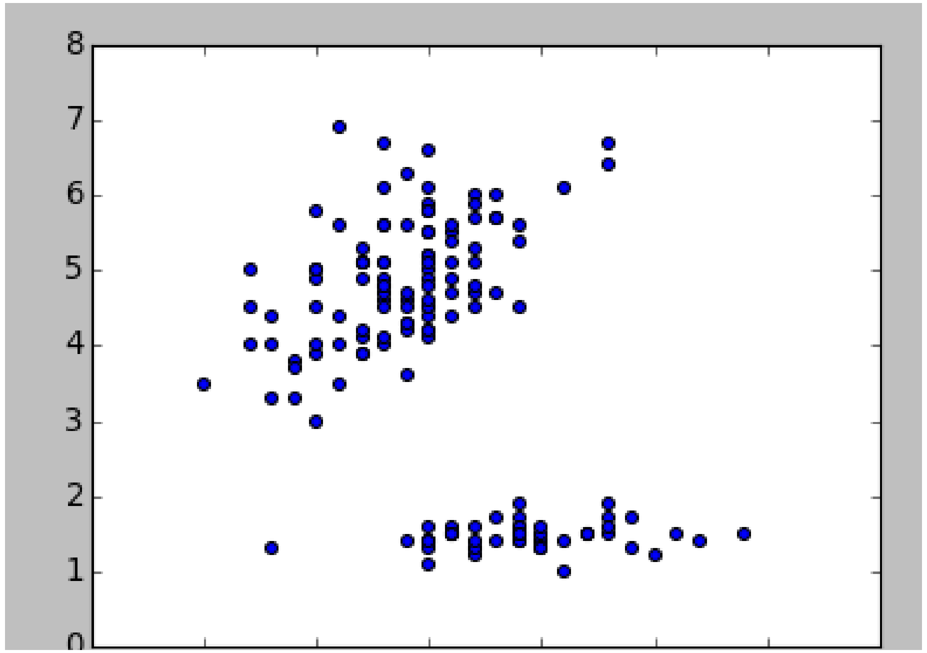
```
> data(iris)
> plot(iris$Sepal.Length,iris$Sepal.Width)
> |
```



Python

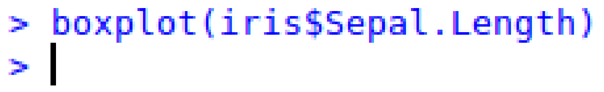


```
>>> from sklearn import datasets
>>> iris = datasets.load_iris()
>>> import matplotlib.pyplot as plt
>>> iris = iris.data
>>> plt.scatter(iris[:,1],iris[:,2])
<matplotlib.collections.PathCollection object at 0x7f910a862d50>
>>> plt.show()
```

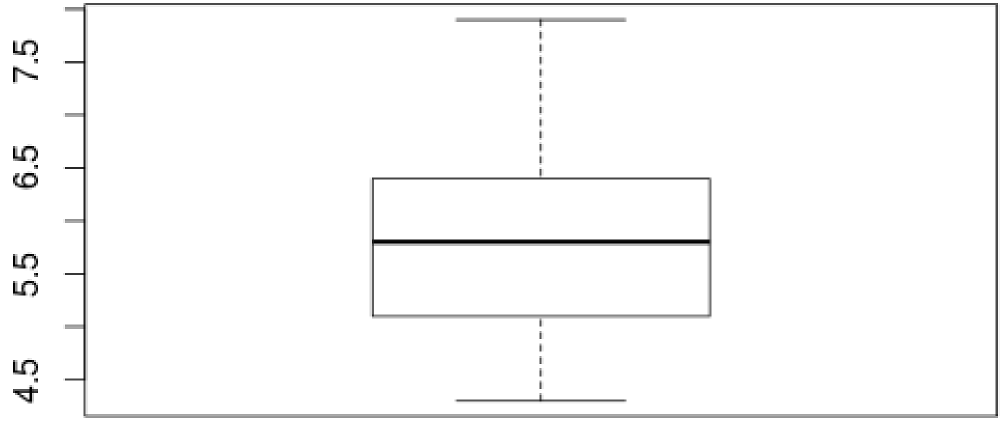


Data Visualization: Box Plot

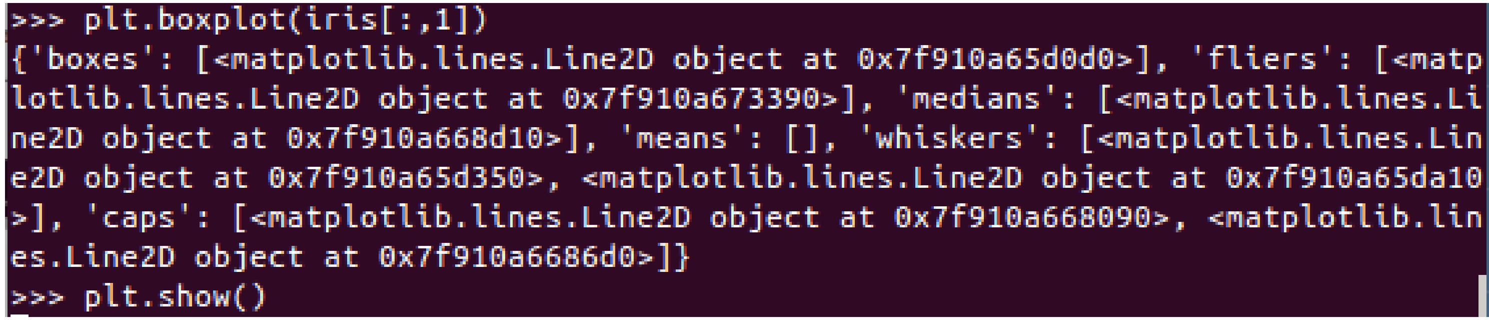
R



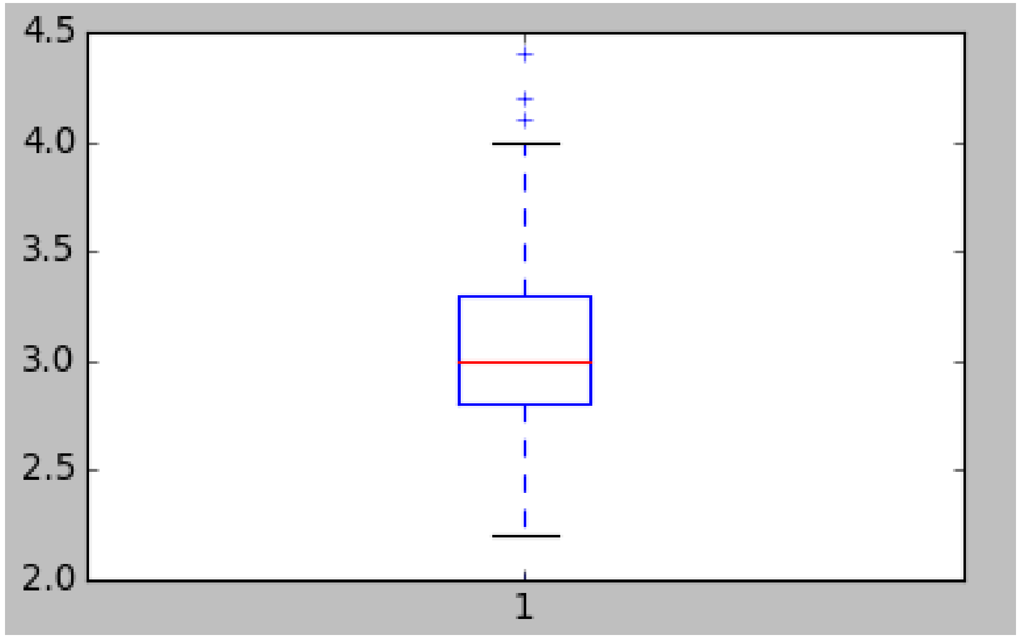
```
> boxplot(iris$Sepal.Length)
> |
```



Python

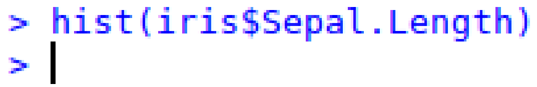


```
>>> plt.boxplot(iris[:,1])
{'boxes': [<matplotlib.lines.Line2D object at 0x7f910a65d0d0>], 'fliers': [<matp
lotlib.lines.Line2D object at 0x7f910a673390>], 'medians': [<matplotlib.lines.Li
ne2D object at 0x7f910a668d10>], 'means': [], 'whiskers': [<matplotlib.lines.Lin
e2D object at 0x7f910a65d350>, <matplotlib.lines.Line2D object at 0x7f910a65da10
>], 'caps': [<matplotlib.lines.Line2D object at 0x7f910a668090>, <matplotlib.lin
es.Line2D object at 0x7f910a6686d0>]}
>>> plt.show()
```



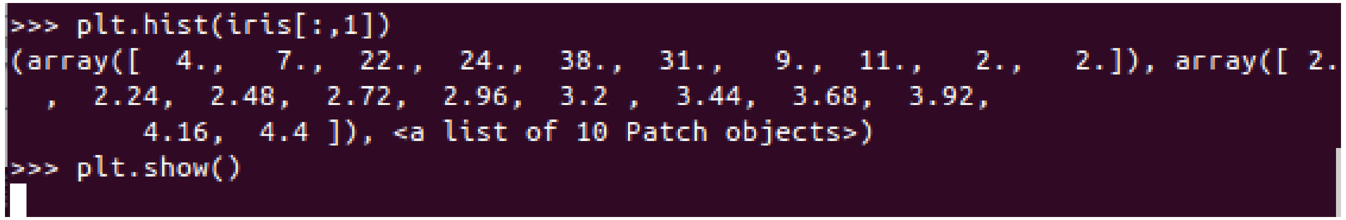
Data Visualization: Histogram

R

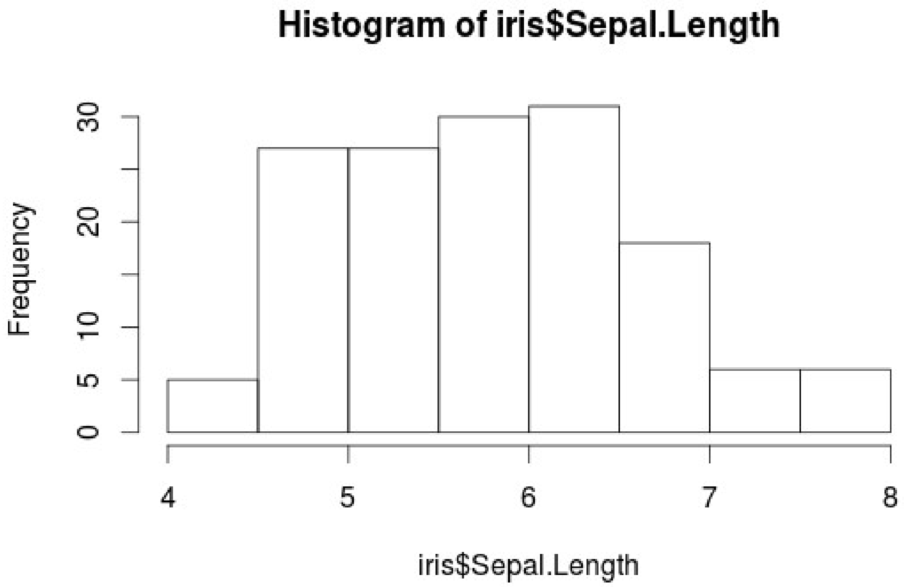


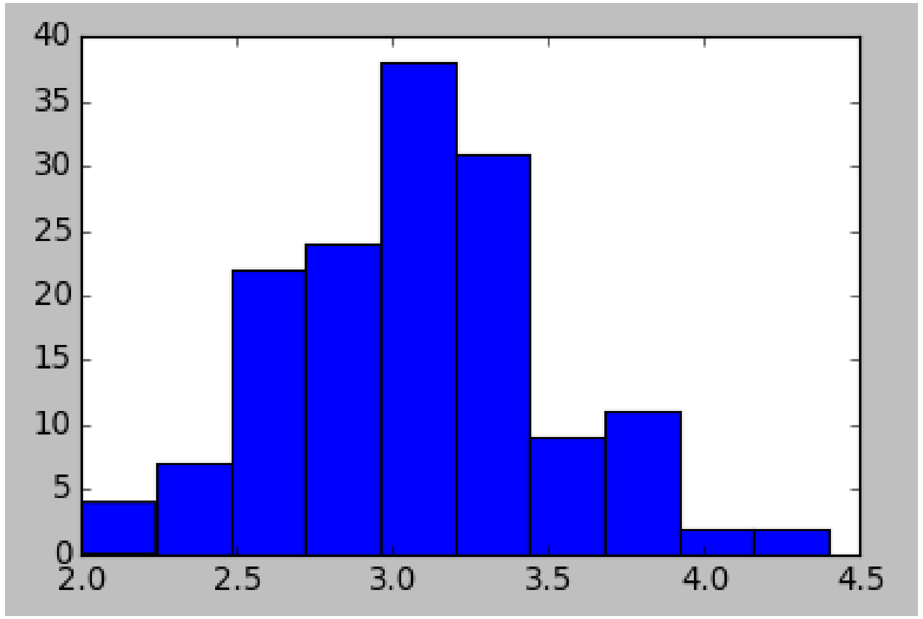
```
> hist(iris$Sepal.Length)
> |
```

Python



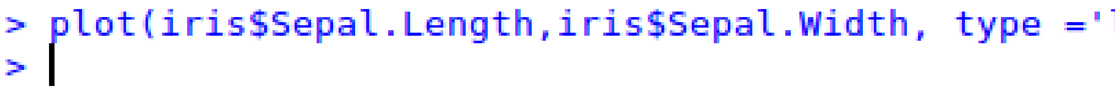
```
>>> plt.hist(iris[:,1])
(array([  4.,   7.,  22.,  24.,  38.,  31.,   9.,  11.,   2.,   2.]), array([ 2.
 ,  2.24,  2.48,  2.72,  2.96,  3.2 ,  3.44,  3.68,  3.92,
       4.16,  4.4 ]), <a list of 10 Patch objects>)
>>> plt.show()
```





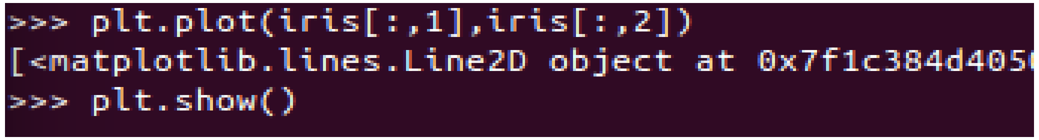
Data Visualization: Line Plot

R

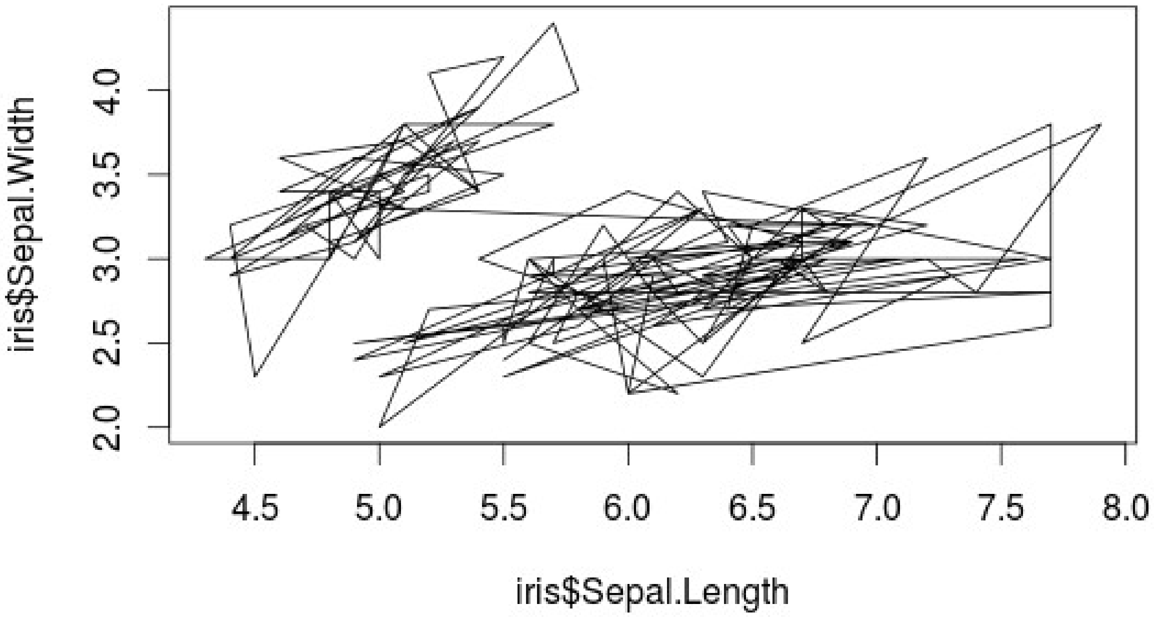


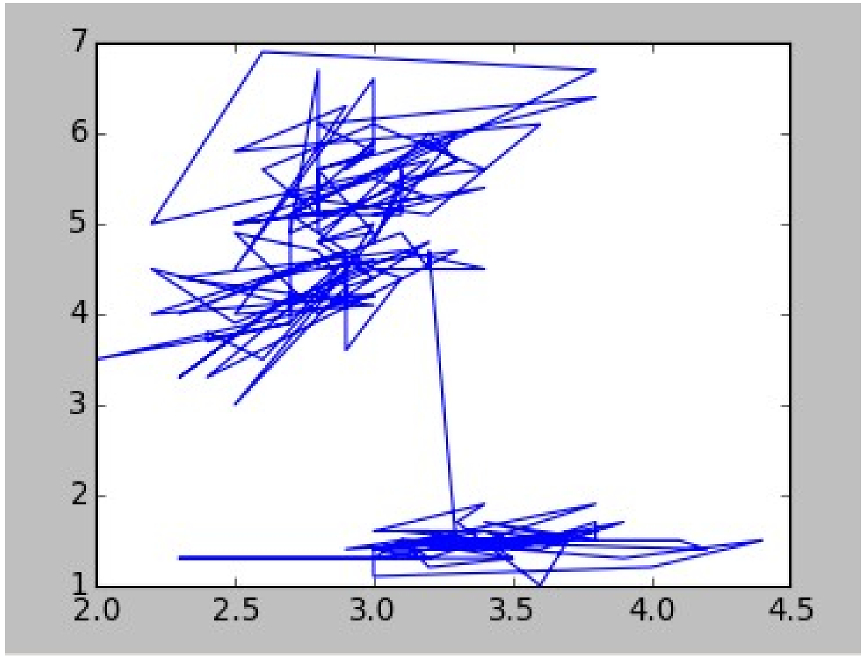
```
> plot(iris$Sepal.Length,iris$Sepal.Width, type ='
> |
```

Python



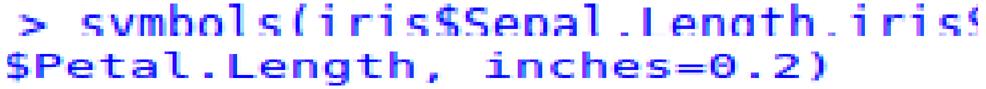
```
>>> plt.plot(iris[:,1],iris[:,2])
[<matplotlib.lines.Line2D object at 0x7f1c384d405
>>> plt.show()
```



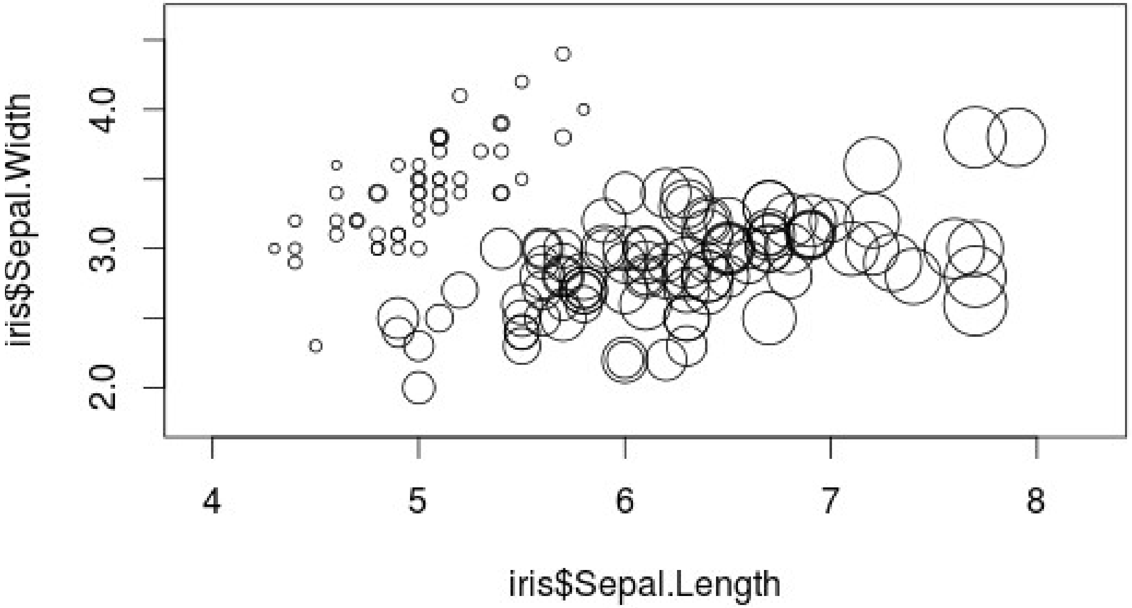


Data Visualization: Bubble

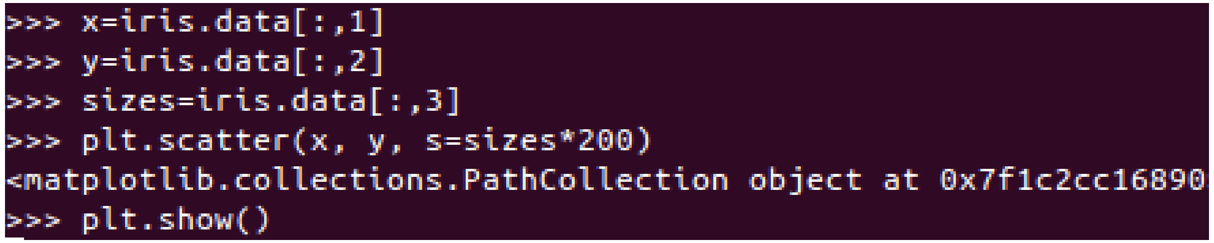
R



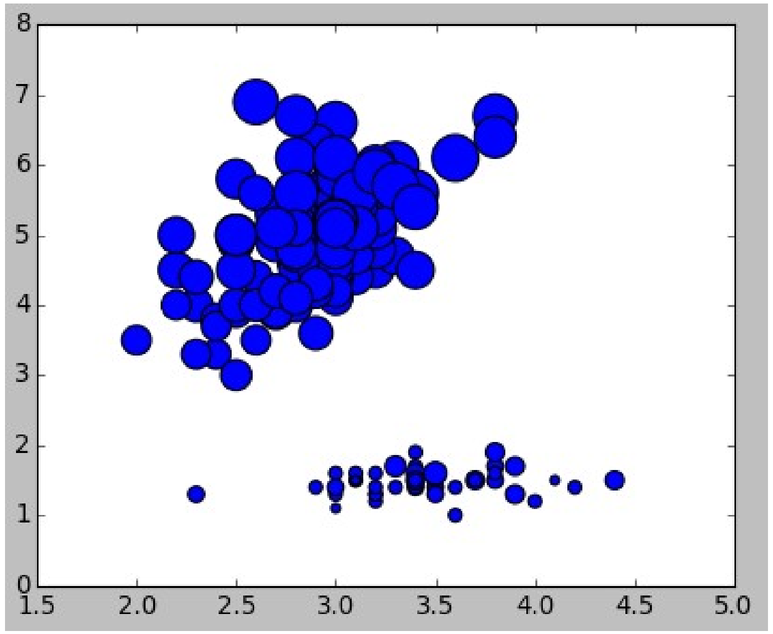
```
> symbols(iris$Sepal.Length,iris$
$Petal.Length, inches=0.2)
```



Python



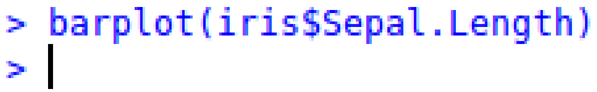
```
>>> x=iris.data[:,1]
>>> y=iris.data[:,2]
>>> sizes=iris.data[:,3]
>>> plt.scatter(x, y, s=sizes*200)
<matplotlib.collections.PathCollection object at 0x7f1c2cc16890
>>> plt.show()
```



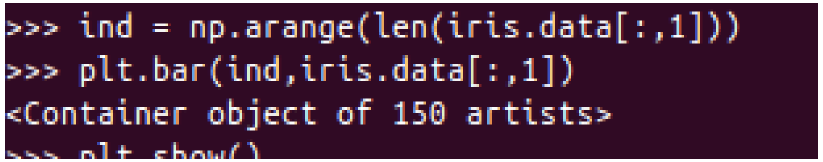
Data Visualization: Bar

R

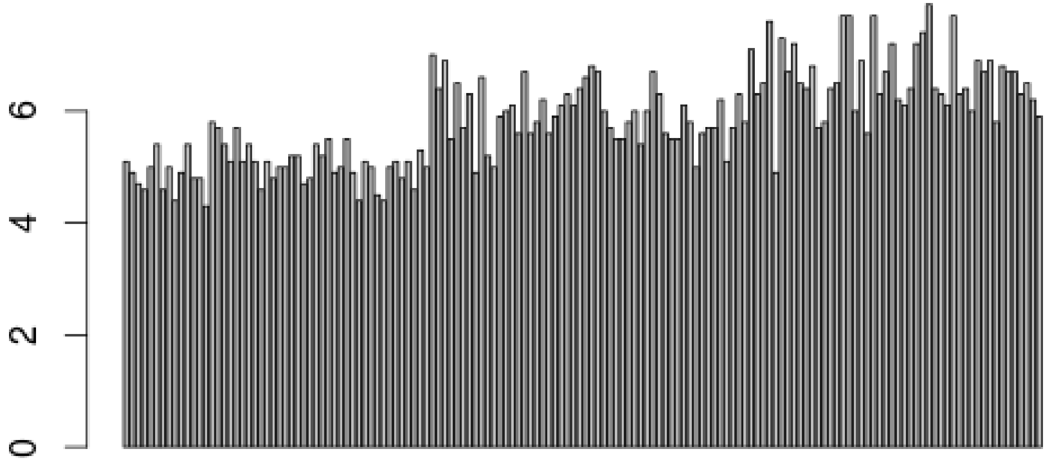
Python

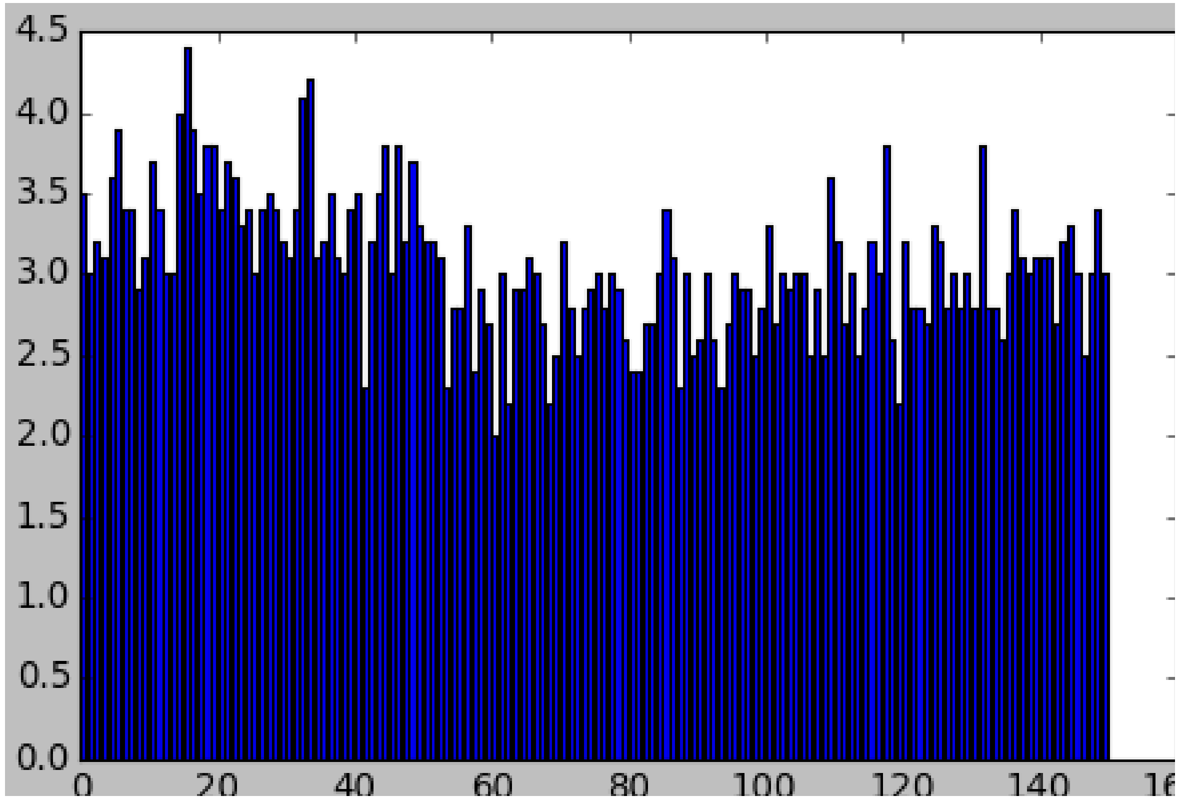


```
> barplot(iris$Sepal.Length)
> |
```



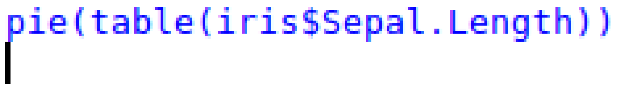
```
>>> ind = np.arange(len(iris.data[:,1]))
>>> plt.bar(ind,iris.data[:,1])
<Container object of 150 artists>
>>> plt.show()
```



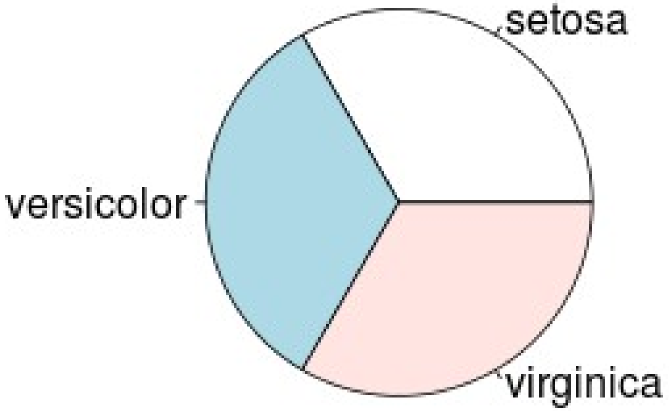


Data Visualization: Pie Chart

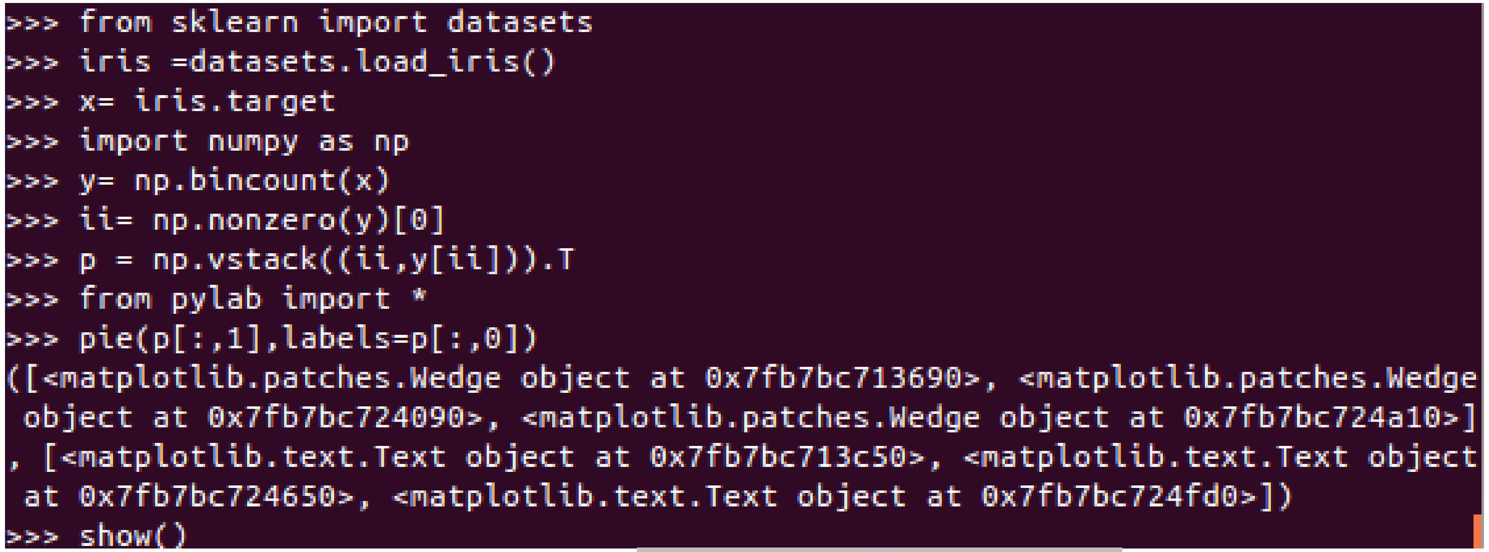
R



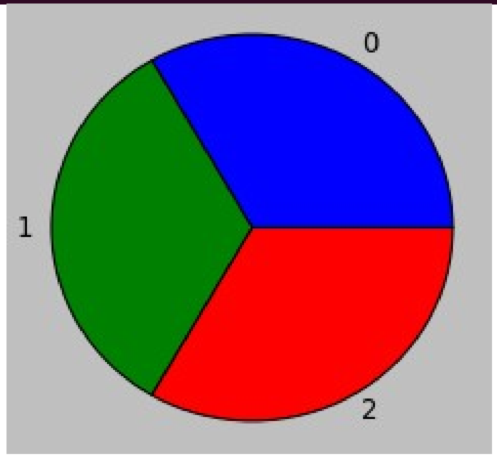
```
pie(table(iris$Sepal.Length))
```



Python



```
>>> from sklearn import datasets
>>> iris =datasets.load_iris()
>>> x= iris.target
>>> import numpy as np
>>> y= np.bincount(x)
>>> ii= np.nonzero(y)[0]
>>> p = np.vstack((ii,y[ii])).T
>>> from pylab import *
>>> pie(p[:,1],labels=p[:,0])
([<matplotlib.patches.Wedge object at 0x7fb7bc713690>, <matplotlib.patches.Wedge
 object at 0x7fb7bc724090>, <matplotlib.patches.Wedge object at 0x7fb7bc724a10>]
, [<matplotlib.text.Text object at 0x7fb7bc713c50>, <matplotlib.text.Text object
 at 0x7fb7bc724650>, <matplotlib.text.Text object at 0x7fb7bc724fd0>])
>>> show()
```



DECISIONSTATS.COM

Thank You

For feedback contact

DecisionStats.com

Coming up

- Data Mining in Python and R (see draft slides afterwards)

Machine Learning: SVM on Iris Dataset

R(Using svm* function)

```
library(e1071)
data(iris)

trainset <-iris[1:149,]
testset <-iris[150,]

svm.model <- svm(Species ~ ., data =
trainset, cost = 100, gamma = 1, type= 'C-
classification')
svm.pred<- predict(svm.model,testset[-5])
svm.pred
```

Output: Virginica

Python(Using sklearn** library)

```
#Loading Library
from sklearn import svm
#Importing Dataset
from sklearn import datasets
#Calling SVM
clf = svm.SVC()
#Loading the package
iris = datasets.load_iris()
#Constructing training data
X, y = iris.data[:-1], iris.target[:-1]
#Fitting SVM
clf.fit(X, y)
#Testing the model on test data
print clf.predict(iris.data[-1])
```

Output: 2, *corresponds to Virginica*

**To know more about svm function in R visit: http://cran.r-project.org/web/packages/e1071/*
*** To install sklearn library visit : http://scikit-learn.org/, To know more about sklearn svm visit: http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html*

Linear Regression: Iris Dataset

R(Using lm* function)

```
data(iris)
total_size<-dim(iris)[1]
num_target<-c(rep(0,total_size))

for (i in 1:length(num_target)){
  if(iris$Species[i]=='setosa'){num_target[i]<-0}
  else if(iris$Species[i]=='versicolor')
{num_target[i]<-1}
  else{num_target[i]<-2}
}
iris$Species<-num_target
train_set <-iris[1:149,]
test_set <-iris[150,]
fit<-lm(Species ~ 0+Sepal.Length+ Sepal.Width+
Petal.Length+ Petal.Width , data=train_set)
coefficients(fit)
predict.lm(fit,test_set)
```

Output: 1.64

Python(Using sklearn** library)

```
from sklearn import linear_model
from sklearn import datasets

iris = datasets.load_iris()
regr = linear_model.LinearRegression()

X, y = iris.data[:-1], iris.target[:-1]
regr.fit(X, y)
print(regr.coef_)
print regr.predict(iris.data[-1])
```

Output: 1.65

**To know more about lm function in R visit: https://stat.ethz.ch/R-manual/R-devel/library/stats/html/lm.html*
*** ** To know more about sklearn linear regression visit : http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html*

Random forest: Iris Dataset

R(Using randomForest* package)

```
library(randomForest)
data(iris)
total_size<-dim(iris)[1]
num_target<-c(rep(0,total_size))

for (i in 1:length(num_target)){
  if(iris$Species[i]=='setosa'){num_target[i]<-0}
  else if(iris$Species[i]=='versicolor')
{num_target[i]<-1}
  else{num_target[i]<-2}}
iris$Species<-num_target
train_set <-iris[1:149,]
test_set <-iris[150,]
iris.rf <- randomForest(Species ~ .,
data=train_set,ntree=100,importance=TRUE,
                        proximity=TRUE)
print(iris.rf)
predict(iris.rf, test_set[-5], predict.all=TRUE)
```

Output: 1.845

Python(Using sklearn** library)

```
from sklearn import ensemble
from sklearn import datasets
clf =
ensemble.RandomForestClassifier(n_estimato
rs=100,max_depth=10)
iris = datasets.load_iris()
X, y = iris.data[:-1], iris.target[:-1]
clf.fit(X, y)
print clf.predict(iris.data[-1])
```

Output: 2

**To know more about randomForest package in R visit: http://cran.r-project.org/web/packages/randomForest/*

*** To know more about sklearn random forest visit : http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html*

Decision Tree: Iris Dataset

R(Using rpart* package)

```
library(rpart)
data(iris)

sub <- c(1:149)

fit <- rpart(Species ~ ., data = iris,
subset = sub)
fit

predict(fit, iris[-sub,], type = "class")
```

Output: Virginica

Python(Using sklearn** library)

```
from sklearn.datasets import load_iris

from sklearn.tree import
DecisionTreeClassifier

clf =
DecisionTreeClassifier(random_state=0)
iris = datasets.load_iris()
X, y = iris.data[:-1], iris.target[:-1]
clf.fit(X, y)
print clf.predict(iris.data[-1])
```

Output: 2, corresponds to virginica

**To know more about rpart package in R visit: http://cran.r-project.org/web/packages/rpart/*

*** To know more about sklearn desicion tree visit : http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html*

Gaussian Naive Bayes: Iris Dataset

R(Using e1071* package)

```
library(e1071)
data(iris)

trainset <-iris[1:149,]

testset <-iris[150,]

classifier<-naiveBayes(trainset[,1:4],
trainset[,5])

predict(classifier, testset[,-5])
```

Output: Virginica

Python(Using sklearn** library)

```
from sklearn.datasets import load_iris

from sklearn.naive_bayes import GaussianNB

clf = GaussianNB()
iris = datasets.load_iris()
X, y = iris.data[:-1], iris.target[:-1]
clf.fit(X, y)
print clf.predict(iris.data[-1])
```

Output: 2, corresponds to virginica

**To know more about e1071 package in R visit: http://cran.r-project.org/web/packages/e1071/*
*** To know more about sklearn Naive Bayes visit : http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html*

K Nearest Neighbours: Iris Dataset

R(Using kknn* package)

```
library(kknn)
data(iris)
trainset <-iris[1:149,]

testset <-iris[150,]
iris.kknn <- kknn(Species~.,
trainset,testset, distance = 1,
    kernel = "triangular")
summary(iris.kknn)
fit <- fitted(iris.kknn)
fit
```

Output: Virginica

Python(Using sklearn** library)

```
from sklearn.datasets import load_iris

from sklearn.neighbors import
KNeighborsClassifier

knn = KNeighborsClassifier()
iris = datasets.load_iris()
X, y = iris.data[:-1], iris.target[:-1]

knn.fit(X,y)
print knn.predict(iris.data[-1])
```

Output: 2, corresponds to virginica

**To know more about kknn package in R visit:*

*** To know more about sklearn k nearest neighbours visit : http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.NearestNeighbors.html*

DECISIONSTATS.COM

Thank You

For feedback please let us know at

ohri2007@gmail.com