DATACAMP NOTES

Plot Function:

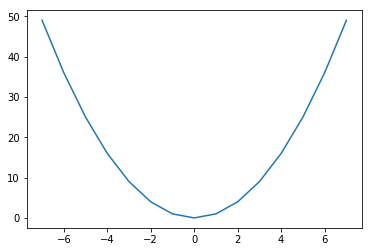
Used to plot the graph using *x* and *y* coordinate lists.

import matplotlib.pyplot as py

x = [z for z in range(-7,8)]

y = [t\*\*2 for t in x]

py.plot(x,y)



Scatter Function:

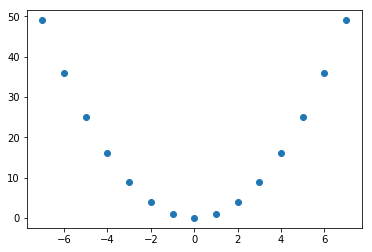
Used to plot the points using *x* and *y* coordinate lists.

import matplotlib.pyplot as py

x = [z for z in range(-7,8)]

y = [t\*\*2 for t in x]

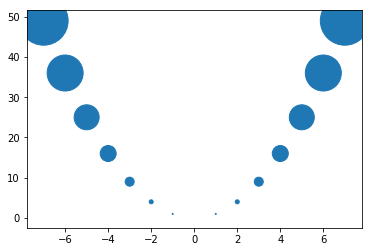
py.scatter(x,y)



Size attribute:

z = [t\*\*4 for t in x]

py.scatter(x,y,s = z)



Text

py.text(x\_pos, y\_pos, text)

#adds *text* at (x\_pos,y\_pos)

Grid

py.grid(True)

Colour

If list then colour goes in cycle

py.scatter(x,y,s = z, c = ['red','yellow','blue'], alpha = 0.8)

If dictionary then as per key value

Bee Swarm Plot

sns.swarmplot(x = 'state', y = 'dem\_share', data = test\_file)

plt.xlabel('States')

plt.ylabel('Share of Votes')



Hist Function:

Labels and Title:

xlab = 'GDP per Capita [in USD]'

ylab = 'Life Expectancy [in years]'

title = 'World Development in 2007'

plt.xlabel(xlab)

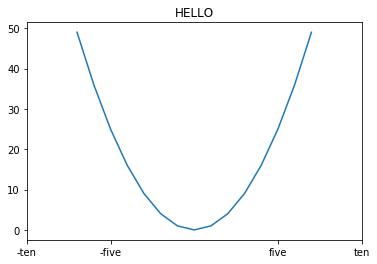
plt.ylabel(ylab)

plt.title(title)

plt.show()

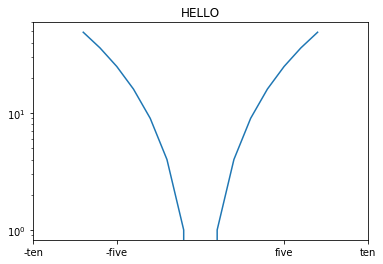
Ticks

py.xticks([-10,-5,5,10],['-ten','-five','five','ten'])



Scale

py.yscale('log')



Importing data from web

To Download Files

from urllib.request import urlretrieve

url = 'https://s3.amazonaws.com/assets.datacamp.com/production/course\_1606/datasets/winequality-red.csv'

urlretrieve(url,filename.csv)

To Open Webpages

from urllib.request import urlopen, Request

request = Request(url)

response = urlopen(request)

response.read()

response.close()

*# method 2*

import requests

r = requests.get(ulr)

html = r.text

Arrange The Html Code Properly

from bs4 import BeautifulSoup

r = requests.get(url)

html\_text = r.text

soup = BeautifulSoup(html\_text)

pretty\_soup = soup.prettify()

Get Title

title = soup.title

Getting Hyperlinks From The Webpage

a\_tags = soup.find\_all(‘a’)

for link in a\_tags:

print(link.get(‘href’)

JSON

import json

with open(‘jsonfilename.json, ‘r’) as json\_file:

json\_data = json.load(json\_file)

*# json\_data is a dictionary*

url = 'http://www.omdbapi.com/?apikey=72bc447a&t=the+social+network'

response = requests.get(url)

response.json()

Pandas:

import pandas as pd

d = {'Country':['India', 'Pakistan', 'USA','China'],

'Capital':['New Delhi', 'Lahore', 'Washington', 'Beijing'],'S.No': [1,2,3,4]}

b = pd.DataFrame(d)

b.index = ['IND', 'PAK', 'USA', 'CH']

*#To import csv*

x = pd.read\_csv(filename.csv, index\_col = 0, chunksize = 10)

*# chunksize will determine the number of rows to be read at a time.*

*# use the next(x) to read the next set.*

*# index\_col will assign the column at the specified index as the index of*

*the dataframe or use column name*

*#Other operations*

b['Country','S.No']

prints the Country and S.No column along with the Index column as Series

b[['Country','S.No']]

prints the Country and S.No column along with the Index column as DataFrame

b[1:3]

prints all columns of Pakistan and USA

b.loc['IND']

prints the details of India as a Series

b.loc[['IND']]

prints the details of India as a DataFrame

b.loc[["IND"],['Country']]

prints only the Country column along with Index column of India

*# first row, then column*

b.loc[:,['Country',"S.No"]]

all rows, only Country and S.No column along with Index column

*#b.iloc works in a similar manner (uses index in place of keys)*

b[‘S.No’] > 2

will print Boolean Series of S.No if > 2

b[b[‘S.No’] > 2]

will print only those rows in which S.No > 2

for i in b.index:

print(i)

prints all the indices

for x,y in b.iterrows():

print(x)

print(y)

prints the label and the rest of the table content as Series

for x,y in b.iterrows():

print(x)

print(y[“Country”])

prints the label and the county name

for x,y in b.iterrows():

b.loc[x,"COUNTRY"] = y["Country"].upper()

b.loc[:,"COUNTRY"] = b["Country"].apply(lambda x:x.upper())

constructs a new column COUNTRY with country name in caps

b.head(2)

returns 2 rows from top

b.tail(2)

returns 2 rows from bottom

*# to import excel file*

f = pd.ExcelFile(file)

sheets = f.sheet\_names

z = f.parse(sheetName or index)

Data Cleaning

d.info()

Gives a statistical report of the Data Set

d.columns()

Returns a list of columns of the DataFrame

b['Ozone'].isnull().sum()

Gives the count of the null values in the column Ozone

b.describe()

Gives the mathematical stats of all numeric columns with non null entries

c['S.No'].value\_counts(dropna = False)

Gives the mathematical stats of all columns with mixed data types. If dropna is false, then it gives a count of NaN as well. By default dropna is True

normalize = true 🡪 gives proportions instead of count

b[b.Ozone > 60]['Ozone'].plot('bar')

Gives a bar graph of the Ozone column with entry value > 60

c['New Col'] = c['Country'].str[0]

A new column by the name of New Col is created with the value equal the first char of Country name in each entry

p.XX.notnull()

Gives Boolean false for null in column XX

b.values

Represents the entire dataframe as a numpy array

b.Capital.unique()

Returns the unique values in the Capital column

Concatenating DataFrames

pd.concat([c,c],ignore\_index = True)

-OR-

x = c.append(c)

x.reset\_index()

v = pd.concat(z,axis='rows', keys=[2011,2013], join = ‘inner’)

The items of keys will act as an outer index provided the indices of the two dataframes are in the same order.

Keys works in case of columns also

join = inner is used to concat only those entries which are common to both the dataframes. (default = outer, all entries)

Scanning all files

import glob

file\_names = glob.glob("\*.csv")

*# file\_name will consist of all the files ending with .csv*

*# \* : any arbitrary number of characters*

*# ? : only one character*

*Eg: “file\_?.csv”*

Merging of DataFrame

pd.merge(left = c, right = z, on = 'Country')

*# the ‘on’ attribute is used when the column exists in both the DataFrames and merging is done by comparing the entries in these columns.*

*# if both DataFrames do not consist of same column then left\_on and right\_on are used to compare the entries*

Data Types and Interconversion

c.dtypes

*# returns the datatype of each column*

.apply Attribute

def country\_code(country):

return country[0]

z['Country Code'] = z['Country'].apply(country\_code)

*# adds a Country Code column with values equal to first char of country name*

Delete Duplicate

p.drop\_duplicates()

Missing Data

p['XX'] = p['XX'].fillna('Missing')

*# will fill all NaN entries in XX column with Missing*

mean\_value = p['S.No'].mean()

p['S.No'] = p['S.No'].fillna(mean\_value)

*# will fill all NaN entries in S.No column with mean value of column*

Drop

k = k.drop(columns = 'Country\_y')

Will delete the column Country\_y

df.drop(index = 1, axis = 1)

Will drop the row with index 1

X\_train = X\_train.dropna(how = 'any')

Will drop all rows with any NaN value

b.dropna(subset = ['Capital'], how = 'any')

Will delete the rows in which Capital is NaN

k.dropna(axis = 1)

Will delete the column which has any NaN value

del b['Country']

Saving to a .csv file

b.to\_csv(‘filename.csv’)

b is a DataFrame

Changing Index and Column Name

b.index = ['IND','PAK','USA','CH']

b.columns = ['Country','Capital','S.No','X']

Resample

Used when the index is of date format

df1 = df.Temperature.resample('6H').mean()

df2 = df.Temperature.resample('D').count()

august = df.Temperature['2010-08-01' : '2010-08-31']

Manupulation

b.Capital.str.contains('D')

Returns bool for the index in which the Capital contains ‘D’

Multi indexing or Hierarchical indexing

index1 = ['IND', 'PAK', 'USA', 'CH']

index2 = [1,2,3,4]

b = b.set\_index([index1 , index2])

Will have two index columns

NY\_month1 = sales.loc[('NY',1)]

Accessing is done using tuples

Melting of DataFrame

pd.melt(k\_df,id\_vars=['Response'], value\_vars=['Treatement','Gender'])

*# Refer Documentation*

<https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.melt.html>

.pivot()

k\_df.pivot(index='Gender', columns='Treatement',values='ID')

The Gender column will be assigned as the Index column and the Treatement column will be assigned as the only column. The ID column will be used to fill the values in the table.

.pivot\_table()

*# Used in place of .pivot if any two or more entries have same data.*

k\_df.pivot\_table(index = 'Gender', columns='Treatement',values='ID', aggfunc='count')

Index col🡪 Gender

Columns 🡪 Treatement

Value 🡪 count of number of IDs (if zero then NaN)

*# the aggfunc parameter is used to define the mathematical operation on the values. Eg: mean (default), count, median, std*

*# to replace NaN use fill\_value parameter*

*# margins = True 🡪 adds a subtotal column and row.*

*# margins\_name can be used to change the column/row name of the subtotal row/column*

.groupby()

k\_df.groupby('Gender').max()

Data will be grouped on the basis of Gender (Gender will be the index) and the max value for every column will be showed.

idxmin() and idxmax()

Returns the index label of the min and max values respectively

file.idxmax()

It returns the max value label of all columns

file.Ozone.idxmax()

It returns the max value label of only the Ozone column

file.idxmin().value\_counts()

Gives a count of the indices

unique() and nunique()

k\_df.Response.unique()

Returns all the unique values of the Response column.

k\_df.Response.nunique()

Returns the number of unique values of the Response column.

x = k\_df.groupby('Gender')

x.Treatement.nunique()

Will return number of unique treatments for each gender.

reindex()

k\_df.reindex(['One','Two','Three','Four','Five','Six'])

If more or less indices are used then the table has NaN values or doesn’t show the entries in the respective cases.

.crosstab()

pd.crosstab(k\_df.Gender,k\_df.ID)

Gender will become index and ID will be the column names and value will be count.

.map()

dic = {'A':True,'B':False}

k\_df['New Col'] = k\_df.Treatement.map(dic)

The column New Col consists of True entry if Treatment for that entry is A and False for Treatment B

Numpy

np.random.poisson

x = np.random.poisson(60,size = 10000)

Array of 10000 numbers will be returned with a mean of 60

np.random.bionomial

y = np.random.binomial(5,0.2,100)

Array of length 100 will be returned with numbers ranging from 0-4 with each number’s probability equal to 0.2

np.random.randint()

np.random.randint(2,6,x)

Returns x random integers in the range(2,6)

np.random.random()

np.random.random(2)

Returns 2 random number between [0,1]

np.random.seed(2)

np.random.seed(2)

np.random.random(2)

Generates 2 pseudo random numbers between [0,1]

Supervised Learning:

1. Regression: When we can determine the right answer from a given set of data.

Eg: A dataset consists of price of houses for different sq feet, so we can determine the price of a house of any size by a linear/polynomial graph.

1. Classification: When data can be classified.

Eg: Determining whether an individual from a group is male/ female.

np.polyfit()

a, b = np.polyfit(illiteracy,fertility, deg = 1)

return a 🡪 slope and b🡪 intercept

np.linspace()

a\_vals = np.linspace(0,0.1,200)

Inserts equally spaced 200 numbers between 0 and 0.1

np.random.choice()

np.random.choice(x,5)

Will create a list of 5 items from x chosen at random with replacement

np.random.permutation()

np.random.permutation(x)

Will return a numpy array with elements arranged in random order.

ROC Curve

from sklearn.metrics import roc\_curve

from sklearn.linear\_model import LogisticRegression

log\_r = LogisticRegression()

log\_r.fit(X\_train,Y\_train)

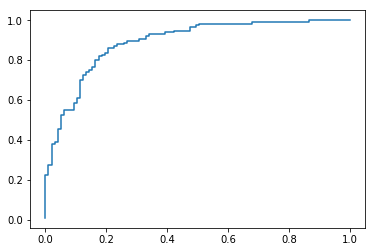
prob = log\_r.predict\_proba(X\_test)[:,1]

*#prob is the probability of each entry being 1*

fpr,tpr,thresholds = roc\_curve(Y\_test,prob)

plt.plot(fpr,tpr)

plt.plot([0,1],[0,1],'k--')



AUC Curve

from sklearn.metrics import roc\_auc\_score

from sklearn.linear\_model import LogisticRegression

logreg = LogisticRegression()

logreg.fit(X\_train,Y\_train)

pred = logreg.predict\_proba(X\_test)[:,1]

roc\_auc\_score(Y\_test,pred)

*# the better the score, the better is the algorithm*

Logistic Regression

from sklearn.linear\_model import LogisticRegression

logreg = LogisticRegression()

logreg.fit(X\_train,Y\_train)

logreg.predict(X\_test)

logreg.score(X\_test,Y\_test)

Confusion Matrix and Classification Report

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import confusion\_matrix, classification\_report

logreg = LogisticRegression()

logreg.fit(X\_train,Y\_train)

pred = logreg.predict(X\_test)

print(confusion\_matrix(Y\_test,pred))

print(classification\_report(Y\_test,pred))



GridSearchCV

from sklearn.model\_selection import GridSearchCV

knn = KNeighborsClassifier()

param\_grid = {'n\_neighbors':np.arange(1,50)}

grid = GridSearchCV(knn,param\_grid,cv = 5)

grid.fit(X,Y)