```
import numpy as np
import pandas as pd
from sklearn import preprocessing
import matplotlib.pyplot as plt
import random
from pandas import DataFrame
from pandas import concat
import io



df= pd.read_csv(io.BytesIO(uploaded['GSPC.csv']))
data_csv=df
#how many data we will use
# (should not be more than dataset length )
data_to_use= len(data_csv)
# number of training data
# should be less than data_to_use
train_end =len(data_csv)-458
total_data=len(data_csv)
#most recent data is in the end
#so need offset
start=total_data - data_to_use
#currently doing prediction only for 1 step ahead
steps_to_predict =1
yt = data_csv.iloc [start:total_data ,4] #Close price
yt1 = data_csv.iloc [start:total_data ,1] #Open
yt2 = data_csv.iloc [start:total_data ,2] #High
yt3 = data_csv.iloc [start:total_data ,3] #Low
vt = data_csv.iloc [start:total_data ,5] # volume
```

---

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import math
from scipy import stats
from scipy.stats import norm,t,cauchy,laplace,probplot



col = ['date','citi','sp']
df = pd.read_csv("d-csp0108.txt", delim_whitespace=True, header=0)
```

```
citi = list(df['C'])
sp = list(df['SP'])
c = np.asarray(citi)
sp = np.asarray(sp)
```

---

```
import numpy as np
import random, os, time
import cPickle as pickle
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn import metrics
import matplotlib.pyplot as plt
from scipy.spatial.distance import cdist, pdist

n = 69873
p = 10586
M_train = np.zeros((n,p),dtype='float16')
M_test = np.zeros((n,p),dtype='float16')
with open('u10m.dat') as f:
        for i,line in enumerate(f):
                lst = line.split()
                u_id, m_id, rating = (int(lst[0]) - 1), (int(lst[1]) - 1), int(lst[2])
                if i%5 != 0:
                        M_train[u_id][m_id] = rating
                else:
                        M_test[u_id][m_id] = rating
print "Data loaded"

# d = {"M_train":M_train, "M_test":M_test}
# with open(r"data.pickle","wb") as output_file:
#       pickle.dump(d, output_file)

def initialize_UV(k):
        M = 3.0
        S = 1.0
        mu = np.sqrt(M/k)
        sigma = np.sqrt( np.sqrt(mu**4 + S/k) -mu**2)
        U = np.random.normal(mu,sigma,n*k).reshape(n,k)
        V = np.random.normal(mu,sigma,k*p).reshape(p,k)
        return U,V

def loss(M,U,V,l1,l2):
    loss = l1*np.linalg.norm(U, ord='fro')**2 + l2*np.linalg.norm(V, ord='fro')**2
    M_ = np.dot(U,V.T)
    nz = np.nonzero(M)
    loss += np.sum((M[nz] - M_[nz])**2)
```

```
    return loss

L1 = np.array([6])
L2 = np.array([6])
TestLoss = np.zeros((len(L1),len(L2)))
num = 1
k = 10
I = np.eye(k)
itr = 100

NP_train = M_train!=0
NP_test = M_test!=0
lstsq = np.linalg.lstsq
M_train_copy = M_train.copy()
M_test_copy = M_test.copy()

for l1i, l1 in enumerate(L1):
        for l2j, l2 in enumerate(L2):
                print l1, l2
                TestLoss_values = np.zeros(num)
                for _ in range(num):
                        L_train = []
                        L_test = []
                        U,V = initialize_UV(k)
                        train_loss = loss(M_train,U,V,l1,l2)
                        test_loss = loss(M_test,U,V,0,0)
                        for t in range(itr):
                                start = time.time()
                                print t," Train Loss = ",train_loss," Test Loss = ",test_loss,
                                for i in range(n):
                                        U[i] = lstsq(np.dot(V[NP_train[i]].T,V[NP_train[i]])+l1*I ,
np.dot(V[NP_train[i]].T,M_train[i,NP_train[i]]))[0]
                                for j in range(p):
                                        V[j] = lstsq(np.dot(U[NP_train[:,j]].T,U[NP_train[:,j]])+l2*I ,
np.dot(U[NP_train[:,j]].T,M_train[NP_train[:,j],j]))[0]
                                train_loss = loss(M_train,U,V,l1,l2)
                                test_loss = loss(M_test,U,V,0,0)
                                L_train.append(train_loss)
                                L_test.append(test_loss)
                                if t!=0 and (L_test[-1]>L_test[-2] or (L_test[-2]-L_test[-1])<100):
                                        break
                                # if t!=0 and np.abs(L_train[-1]-L_train[-2])<10:
                                #         break
                                print time.time()-start
                        min_test_loss = L_test[-1]
                        M_pred = np.dot(U,V.T)
                        g = open("U.txt","w")
                        np.savetxt(g, U, fmt='%-7.3f')
                        g.close()
                        g = open("V.txt","w")
                        np.savetxt(g, V, fmt='%-7.3f')
                        g.close()
```

```
                    TestLoss_values[_] = min_test_loss
                    print TestLoss_values

print "Written factor matrices to U.txt and V.txt"
```

```
import pandas as pd
import numpy as np
import random as ran
import matplotlib.pyplot as plt
import time
import math
import scipy
from scipy.stats import norm
user_data=pd.read_csv('ml-100k/u.user', sep='|', header=None, encoding='latin-1')

movies_data=pd.read_csv('ml-100k/u.item', sep='|', header=None, encoding='latin-1')

ratings=pd.read_csv('ml-100k/u.data', sep='\t', header=None, encoding='latin-1')

print("The data has been taken")

nmovies=movies_data.shape[0]
nusers=user_data.shape[0]

ratings_matrix=np.zeros((nusers,nmovies))

train_mat=np.zeros((nusers,nmovies))

for i in range(1,80000):
 train_mat[ratings[0][i]-1][ratings[1][i]-1]=ratings[2][i]

test_mat=np.zeros((nusers,nmovies))

for i in range(80000, 100000):
```

```
 test_mat[ratings[0][i]-1][ratings[1][i]-1]=ratings[2][i]

print("The train test split is 80 20")

init_k=25

print("The initial no of latent features is 25")
#normalization of ratings

train_rate01=(train_mat!=0)
train_rate02=(train_mat==0)

test_rate01=(test_mat!=0)
test_rate02=(test_mat==0)

rat_mean=np.zeros(shape=(nmovies,1))
train_rat_norm=np.zeros(shape=train_mat.shape)

row_mean=0
for i in range(0,nusers):
 row_mean=np.mean(train_mat[i])
 for j in range(0,nmovies):
  train_rat_norm[i][j] =train_mat[i][j]-row_mean
 row_mean=0

train_mat=train_rat_norm

 #def normailze_test_ratings(mat,rate01):
rat_mean=np.zeros(shape=(nmovies,1))
test_rat_norm=np.zeros(shape=test_mat.shape)

for i in range(0,nusers):
 row_mean=np.mean(train_mat[i])
 for j in range(0,nmovies):
  test_rat_norm[i][j]=test_mat[i][j]-row_mean
 row_mean=0

test_mat=test_rat_norm
```

---

```
import numpy as np
import pandas
import matplotlib.pyplot as plt


from sklearn.metrics import mean_squared_error
```

```
from sklearn.preprocessing import MinMaxScaler
import math


#Histori_cols = ['Rainfall', 'SRAD', 'Tmax', 'Tmin', 'Tmean', 'AET', 'RH', 'WT']


data = pandas.read_csv('data_imd_final.txt', header=None, sep = '\t')
data = data.values # now price is np.ndarray
data = data[0:640, 0:8]

print("\n\n\n\n", data.shape[0])

xtrain = data[0:int((data.shape[0])*3/4)]
xtest = data[int((data.shape[0])*3/4):data.shape[0]]
```

---

Crypto-Currency from CSV

---

```
import numpy as np
import pandas
import matplotlib.pyplot as plt

from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Highway
from keras import optimizers
from keras import backend as KBend

from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import MinMaxScaler
import math

price = pandas.read_csv('market-price.csv', header=None)
price = price.values # now price is np.ndarray
price = price[:price.shape[0]-1,1]

pca1 = pandas.read_csv('pca1.csv')
pca1 = pca1.values # now pca1 is np.ndarray
pca1 = pca1[:,0]

pca2 = pandas.read_csv('pca2.csv')
pca2 = pca2.values
pca2 = pca2[:,0]

pca3 = pandas.read_csv('pca3.csv')
pca3 = pca3.values
pca3 = pca3[:,0]

### Creating Nx4 data
```

```
data = np.append(np.reshape(price, (price.shape[0], 1)), np.reshape(pca1, (pca1.shape[0], 1)),
axis=1)
data = np.append(data, np.reshape(pca2, (pca2.shape[0], 1)), axis=1)
data = np.append(data, np.reshape(pca3, (pca3.shape[0], 1)), axis=1)

scaler = MinMaxScaler(feature_range=(0,1))
data = scaler.fit_transform(data)

xtrain = data[0:data.shape[0]*3/4]
xtest = data[data.shape[0]*3/4:data.shape[0]]

dim = xtrain.shape[1]
```

---

```
import requests
from bs4 import BeautifulSoup

#import beautifulsoup4
import csv
import pandas as pd

names=[]
prices=[]
changes=[]
percentChanges=[]
marketCaps=[]
totalVolumes=[]
circulatingSupplys=[]

CryptoCurrenciesUrl = "https://in.finance.yahoo.com/currencies"
r= requests.get(CryptoCurrenciesUrl)
data=r.text
soup=BeautifulSoup(data)

counter = 40
for i in range(40, 404, 14):
  for listing in soup.find_all('tr', attrs={'data-reactid':i}):
    for name in listing.find_all('td', attrs={'data-reactid':i+3}):
      names.append(name.text)
    for price in listing.find_all('td', attrs={'data-reactid':i+4}):
      prices.append(price.text)
    for change in listing.find_all('td', attrs={'data-reactid':i+5}):
      changes.append(change.text)
    for percentChange in listing.find_all('td', attrs={'data-reactid':i+7}):
      percentChanges.append(percentChange.text)

a = pd.DataFrame({"Names": names, "Prices": prices, "Change": changes, "% Change":
percentChanges})

print(a)
```

Importing Data from Bitcoin API..

```
from pycoingecko import CoinGeckoAPI
import numpy as np
import pandas as pd
import json
import requests


cg = CoinGeckoAPI()


def str_column_to_int(dataset, column):
    class_values = [row[column] for row in dataset]
    unique = set(class_values)
    lookup = dict()
    for i, value in enumerate(unique):
        lookup[value] = i
    for row in dataset:
        row[column] = lookup[row[column]]
    return lookup

def str_row_to_int(dataset, row):
    class_values = [column[row] for column in dataset]
    unique = set(class_values)
    lookup = dict()
    for i, value in enumerate(unique):
        lookup[value] = i
    for column in dataset:
        column[row] = lookup[column[row]]
    return lookup

a = {}

x = ['bitcoin','litecoin','ethereum']
df1 = pd.DataFrame()


a = cg.get_coin_market_chart_by_id(id='bitcoin', vs_currencies='usd')['prices']

df2 = pd.DataFrame()

for i in range(3):
 df1 = pd.DataFrame({tic: data[idlist[i]]
                     for tic, data in a.items()})
 df2 = df2.append(df1, ignore_index=True)
```

```
print(df2.values)
```

---

```
from pycoingecko import CoinGeckoAPI
import numpy as np
import pandas as pd

cg = CoinGeckoAPI()

timePeriod = 1257
#timePeriod = 5000

df2= []

a = {}

#x = ['bitcoin','litecoin','ethereum']
x = ['bitcoin']

for coin in x:
 a[coin] = cg.get_coin_market_chart_by_id(id=coin, vs_currency='usd', days=timePeriod)['prices']
 print(pd.DataFrame(a[coin]))
 df1 = pd.DataFrame(a[coin]).values[:,1]
 #df2 = np.append(df2, df1)

print(np.flipud(np.reshape(df1, (timePeriod + 1, 1))))
```

---

Working with a gene-expression data :

```
library(Biobase)
library(breastCancerTRANSBIG)
library(survival)
library(randomForestSRC)
library(pracma)
library(Metrics)
library(stringr)
data(transbig)

library(data.table)
library(quanteda)
library(multiClust)
library(Rtsne)
library(concatenate)
source("gensample.R")


library(GEOquery)
```

```
#library("extract_data.R")

# load series and platform data from GEO

gset <- getGEO("GSE4698", GSEMatrix =TRUE, getGPL=FALSE)

tmpFileName="GSE4698.expression.txt", blnRowNames=TRUE, blnColNames=TRUE)


if (length(gset) > 1) idx <- grep("GPL96", attr(gset, "names")) else idx <- 1
gset <- gset[[idx]]


data1 = pData(gset)
dim(data1)


data1 = data1[, c("Peripheral blast count at relapse diagnosis per microliter:ch1", "Age at relapse
diagnosis [years]:ch1")]

data2 = exprs(gset)
data2 = t(data2)
dim(data1)
dim(data2)

final_data = cbind(data2, data1)
```

---

R Code data extraction from in-build package :

```
-----------------------------------------------------

library(breastCancerTRANSBIG)
library(survival)
library(Biobase)
library(randomForestSRC)
library(pracma)
library(Metrics)

data(transbig)

data1 = pData(transbig)
dim(data1)
data1 = data1[,c(6,7,8,9,20,21)]
data1[,5] = as.integer(data1[,5]/100)

data2 = exprs(transbig)
data2 = t(data2)
```

```
dim(data1)
dim(data2)
#write.csv(data2,file = "part2.csv",row.names = FALSE)
final_data = cbind(data1,data2)
final_data = final_data[complete.cases(final_data),]
```

---

```
x = read.table("datmge.txt", header=TRUE, sep = ",");

x1 = cbind(x[, 2][x[, 1] == "F"], x[, 3][x[, 1] == "F"]);

a1 = x1[,1][x1[,1] > quantile(x1[,1], 0.70)];
b1 = x1[,2][x1[,2] > quantile(x1[,2], 0.7)];

cat("\n\n\n", length(a1), length(b1));

z<-cbind(2*round(a1[(length(a1) - length(b1) + 1):length(a1)]/b1[(length(a1) - length(b1) + 1)], 4),
2*round(b1/b1[(length(a1) - length(b1) + 1)], 4));

muhat1<-min(as.vector(z[,1]));
muhat2<-min(as.vector(z[,2]));

cat("\n\n", muhat1, muhat2);
```

---

```
require(MASS)
require(ks)


sighat1 = 3.898603;
sighat2 = 3.063203;
muhat1  = 12.100000;
muhat2  = 9.000000;



xt = read.table("d-axp3dx-0111.txt", header=TRUE);
x1 = cbind(100*log(1 + xt[,2]), 100*log(1 + xt[,4])); # Auto and cross correlation are almost zero.

#Calculating sub-vector of size 5 from marginals
subvec1 <- split(x1[,1], ceiling(seq_along(x1[,1])/5))
subvec2 <- split(x1[,2], ceiling(seq_along(x1[,2])/5))

#Creating a vector whose element are maximum of corresponding sub-vector
maxvec1 <- as.numeric(sapply(subvec1, max))
maxvec2 <- as.numeric(sapply(subvec2, max))
```

---