**To read file like csv,excel,json file**

import pandas as pd

frm=pd.read\_csv('iris.csv')

#frm

frm1=pd.read\_excel('Supervison.xls')

#frm1

import numpy as mp

a=mp.array([1,2,3,4,5,6])

print(a)

a.reshape(3,2)

**EDA: Exploratory Data Analysis**

**Plotting**

import pandas as pd

import matplotlib.pyplot as plt

iris = pd.read\_csv("iris.csv")

iris.head(5)

iris.plot(kind='scatter', x='sepal\_length', y='sepal\_width') ;

plt.show()

**import seaborn as sns**

# 2-D Scatter plot with color-coding for each flower type/class.

# Here 'sns' corresponds to seaborn.

sns.set\_style("whitegrid");

sns.FacetGrid(iris, hue="species", size=4) \

.map(plt.scatter, "sepal\_length", "sepal\_width") \

.add\_legend();

plt.show();

# Notice that the blue points can be easily seperated

# from red and green by drawing a line.

# But red and green data points cannot be easily seperated.

# Can we draw multiple 2-D scatter plots for each combination of features?

# How many cobinations exist? 4C2 = 6.

**3D scatter plot** <https://plot.ly/pandas/3d-scatter-plots/>

import plotly

import plotly.express as px

iris = px.data.iris()

fig = px.scatter\_3d(iris, x='sepal\_length', y='sepal\_width', z='petal\_width',

color='species')

fig.show()

**What about 4-D, 5-D or n-D scatter plot?**

**Pair-plot**

#Only possible to view 2D patterns.

plt.close();

sns.set\_style("whitegrid");

sns.pairplot(iris, hue="species", size=3);

plt.show()

**VIOLIN PLOT**

sns.violinplot(x="species", y="petal\_length", data=iris, size=8)

plt.show()

**# MNIST dataset downloaded from Kaggle :**

#https://www.kaggle.com/c/digit-recognizer/data

**# Functions to read and show images**.

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

d0 = pd.read\_csv('train.csv')

print(d0.head(5)) # print first five rows of d0.

# save the labels into a variable l.

l = d0['label']

# Drop the label feature and store the pixel data in d.

d = d0.drop("label",axis=1)

print(d.shape)

print(l.shape)

# display or plot a number.

plt.figure(figsize=(7,7))

idx = 100

grid\_data = d.iloc[idx].as\_matrix().reshape(28,28) # reshape from 1d to 2d pixel array

plt.imshow(grid\_data, interpolation = "none", cmap = "gray")

plt.show()

print(l[idx])

PCA

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

d0 = pd.read\_csv('train.csv')

print(d0.head(5)) # print first five rows of d0.

**# save the labels into a variable l.**

l = d0['label']

# Drop the label feature and store the pixel data in d.

d = d0.drop("label",axis=1)

print(d.shape)

print(l.shape)

labels = l.head(15000)

data = d.head(15000)

print("the shape of sample data = ", data.shape)

**# Standardization in 0…..1 format**

from sklearn.preprocessing import StandardScaler

standardized\_data = StandardScaler().fit\_transform(data)

print(standardized\_data.shape)

print(standardized\_data)

**# Covariance Matrix**

#find the co-variance matrix which is : A^T \* A

sample\_data = standardized\_data

# matrix multiplication using numpy

covar\_matrix = np.matmul(sample\_data.T , sample\_data)

print ( "The shape of variance matrix = ", covar\_matrix.shape)

**#eigen values, eigen vector**

# finding the top two eigen-values and corresponding eigen-vectors

# for projecting onto a 2-Dim space.

from scipy.linalg import eigh

# the parameter 'eigvals' is defined (low value to heigh value)

# eigh function will return the eigen values in asending order

# this code generates only the top 2 (782 and 783) eigenvalues.

values, vectors = eigh(covar\_matrix, eigvals=(782,783))

print(values)

print("Shape of eigen vectors = ",vectors.shape)

# converting the eigen vectors into (2,d) shape for easyness of further computations

**vectors = vectors.T**

print("Updated shape of eigen vectors = ",vectors.shape)

# here the vectors[1] represent the eigen vector corresponding 1st principal eigen vector

# here the vectors[0] represent the eigen vector corresponding 2nd principal eigen vector

# projecting the original data sample on the plane

#formed by two principal eigen vectors by vector-vector multiplication.

import matplotlib.pyplot as plt

new\_coordinates = np.matmul(vectors, sample\_data.T)

print (" resultanat new data points' shape ", vectors.shape, "X", sample\_data.T.shape," = ", new\_coordinates.shape)

# resultanat new data points' shape (2,784) X (784, 15000)= (2, 15000)

import pandas as pd

# **appending label** to the 2d projected data

new\_coordinates = np.vstack((new\_coordinates, labels)).T

# creating a new data frame for ploting the labeled points.

dataframe = pd.DataFrame(data=new\_coordinates, columns=("1st\_principal", "2nd\_principal", "label"))

print(dataframe.head())

# **ploting the 2d data points with seaborn**

import seaborn as sn

sn.FacetGrid(dataframe, hue="label", size=6).map(plt.scatter, '1st\_principal', '2nd\_principal').add\_legend()

plt.show()

**PCA direct**

# initializing the pca

from sklearn import decomposition

pca = decomposition.PCA()

# configuring the parameteres

# the number of components = 2

pca.n\_components = 2

pca\_data = pca.fit\_transform(sample\_data)

# pca\_reduced will contain the 2-d projects of simple data

print("shape of pca\_reduced.shape = ", pca\_data.shape)

# attaching the label for each 2-d data point

pca\_data = np.vstack((pca\_data.T, labels)).T

# creating a new data fram which help us in ploting the result data

pca\_df = pd.DataFrame(data=pca\_data, columns=("1st\_principal", "2nd\_principal", "label"))

#import seaborn as sn

sn.FacetGrid(dataframe, hue="label", size=6).map(plt.scatter, '1st\_principal', '2nd\_principal').add\_legend()

plt.show()

**t-SNE**

t**-SNE** t distributed neighborhood embedding

# TSNE

from sklearn.manifold import TSNE

# Picking the top 1000 points as TSNE takes a lot of time for 15K points

data\_1000 = standardized\_data[0:1000,:]

#print(data\_1000.head())

labels\_1000 = labels[0:1000]

model = TSNE(n\_components=2, random\_state=0)

# configuring the parameteres

# the number of components = 2

# default perplexity = 30

# default learning rate = 200

# default Maximum number of iterations for the optimization = 1000

tsne\_data = model.fit\_transform(data\_1000)

# creating a new data frame which help us in ploting the result data

tsne\_data = np.vstack((tsne\_data.T, labels\_1000)).T

tsne\_df = pd.DataFrame(data=tsne\_data, columns=("Dim\_1", "Dim\_2", "label"))

# Ploting the result of tsne

sn.FacetGrid(tsne\_df, hue="label", size=6).map(plt.scatter, 'Dim\_1', 'Dim\_2').add\_legend()

plt.show()

model = TSNE(n\_components=2, random\_state=0, perplexity=50)

tsne\_data = model.fit\_transform(data\_1000)

# creating a new data fram which help us in ploting the result data

tsne\_data = np.vstack((tsne\_data.T, labels\_1000)).T

tsne\_df = pd.DataFrame(data=tsne\_data, columns=("Dim\_1", "Dim\_2", "label"))

# Ploting the result of tsne

sn.FacetGrid(tsne\_df, hue="label", size=6).map(plt.scatter, 'Dim\_1', 'Dim\_2').add\_legend()

plt.title('With perplexity = 50')

plt.show()

model = TSNE(n\_components=2, random\_state=0, perplexity=50, n\_iter=5000)

tsne\_data = model.fit\_transform(data\_1000)

# creating a new data fram which help us in ploting the result data

tsne\_data = np.vstack((tsne\_data.T, labels\_1000)).T

tsne\_df = pd.DataFrame(data=tsne\_data, columns=("Dim\_1", "Dim\_2", "label"))

# Ploting the result of tsne

sn.FacetGrid(tsne\_df, hue="label", size=6).map(plt.scatter, 'Dim\_1', 'Dim\_2').add\_legend()

plt.title('With perplexity = 50, n\_iter=5000')

plt.show()

model = TSNE(n\_components=2, random\_state=0, perplexity=2)

tsne\_data = model.fit\_transform(data\_1000)

# creating a new data fram which help us in ploting the result data

tsne\_data = np.vstack((tsne\_data.T, labels\_1000)).T

tsne\_df = pd.DataFrame(data=tsne\_data, columns=("Dim\_1", "Dim\_2", "label"))

# Ploting the result of tsne

sn.FacetGrid(tsne\_df, hue="label", size=6).map(plt.scatter, 'Dim\_1', 'Dim\_2').add\_legend()

plt.title('With perplexity = 2')

plt.show()