Practical -1

Aim: Study about Python learn following Data structure: List, Dictionary, Dataframe, Numpy/scipy package in Python

Description:

Python: Python is an interpreted high-level general-purpose programming language. Its design philosophy emphasizes code readability with its use of significant indentation. Its language constructs as well as its object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects. Python is dynamically-typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly, procedural), object-oriented and functional programming. It is often described as a "batteries included" language due to its comprehensive standard library. Guido van Rossum began working on Python in the late 1980s, as a successor to the ABC programming language, and first released it in 1991 as Python 0.9.0. It supports functional and structured programming methods as well as OOP. It can be used as a scriptinglanguage or can be compiled to byte-code for building large applications. It provides very high-level dynamic data types and supports dynamic type checking. It supports automatic garbage collection. It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

List in Python : Lists are just like dynamically sized arrays, declared in other languages (vector in C++ and ArrayList in Java). Lists need not be homogeneous always which makes it the most powerful tool in Python. A single list may contain DataTypes like Integers, Strings, as well as Objects. Lists are mutable, and hence, they can be altered even after their creation. List in Python are ordered and have a definite count. The elements in a list are indexed according to a definite sequence and the indexing of a list is done with 0 being the first index. Each element in the list has its definite place in the list, which allows duplicating of elements in the list, with each element having its own distinct place and credibility.

Dictionary in Python : Dictionary in Python is an unordered collection of data values, used to store data values like a map, which, unlike other Data Types that hold only a single value as an element, Dictionary holds key:value pair. Key-value is provided in the dictionary to make it more optimized. In Python, a Dictionary can be created by placing a sequence of elements within curly {} braces, separated by ‘comma’. Dictionary holds pairs of values, one being the Key and the other corresponding pair element being its Key:value. Values in a dictionary can be of any data type andcan be duplicated, whereas keys can’t be repeated and must be immutable.

DataFrame: Pandas DataFrame is two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns). A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns. Pandas DataFrame consists of three principal components, the data, rows, and columns.

Numpy: NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more. NumPy is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices. NumPy was created in 2005 by Travis Oliphant. It is an open-source project and you can use it freely. NumPy stands for Numerical Python. Scipy: SciPy is a free and open-source Python library used for scientific computing and technical computing. SciPy contains modules for optimization, linear algebra, integration, interpolation, special functions, FFT, signal and image processing, ODE solvers and other tasks common in science and engineering. SciPy is a scientific computation library that uses NumPy underneath. SciPy stands for Scientific Python. It provides more utility functions for optimization, stats and signal processing. Like NumPy, SciPy is open source so we can use it freely. SciPy was created by NumPy's creator Travis Olliphant.

Code:

```
list = [1,2,34,5,"Bhargav" , ["dr"]]
print(list)
print(list[::-1])
print(list[0])

#append in list
list.append(1)
print(list)
list2 = ["a","b","c"]
list.append(list2)
print(list)
```

```
[1, 2, 34, 5, 'Bhargav', ['dr']]
[['dr'], 'Bhargav', 5, 34, 2, 1]
1
[1, 2, 34, 5, 'Bhargav', ['dr'], 1]
[1, 2, 34, 5, 'Bhargav', ['dr'], 1, ['a', 'b', 'c']]
```

```
#implement dictionary

dict ={
   "a1":"Bhargav",
   "a2":"deep",
   "a3":"dhruv"
}

print(dict)
print(dict["a1"])
```

```
print(dict.get("a2"))

del dict["a3"]
print(dict)
```

```
{'a1': 'Bhargav', 'a2': 'deep', 'a3': 'dhruv'}
Bhargav
deep
{'a1': 'Bhargav', 'a2': 'deep'}
```

```
#impelement dataframes

import pandas as pd
data = {'Name':['Bhargav', 'Deep', 'krish', 'Rohit'],
        'Age':[20, 21,22,35]
      }
data1 = {'Name':['Bhargav', 'Deep', 'krish', 'Rohit'],
        'Age':[20, 21,22,35],
        'Address':['surat', 'surat', 'Allahabad', 'pune'],
        'Qualification':['BE', 'BE', 'MCA', 'PHD']}

D_frame = pd.DataFrame(data)
D_frame2 = pd.DataFrame(data1)
print(D_frame)
print("\n")
print(D_frame2[["Name","Address" ,"Qualification"]])
```

```
      Name  Age
0  Bhargav   20
1     Deep   21
2    krish   22
3    Rohit   35


      Name    Address Qualification
0  Bhargav      surat            BE
1     Deep      surat            BE
2    krish  Allahabad           MCA
3    Rohit       pune           PHD
```

```
# Implement numpy

import numpy as nm
ar = nm.array([[1,2,3],4,5,[11,22]])
print(ar,"\n")
print(ar[:2]) #slice the array

x= ar.view()
print(x,"\n")
```

```
Rs = ar.reshape(2,2)
print(Rs,"\n")

for x in ar:
  print(x)

print('\n')

arr1 = nm.array([1,2,3])
arr2 = nm.array([4,5,6])

arr3 = nm.concatenate((arr1,arr2))
print(arr3,"\n")

arr4 = nm.where(arr1 == 2)
print(arr4)
```

```
[list([1, 2, 3]) 4 5 list([11, 22])]

[list([1, 2, 3]) 4]
[list([1, 2, 3]) 4 5 list([11, 22])]

[[list([1, 2, 3]) 4]
 [5 list([11, 22])]]

[1, 2, 3]
4
5
[11, 22]


[1 2 3 4 5 6]

(array([1]),)
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:4: VisibleDeprecationWarning: Creating an ndarray from ragged
  after removing the cwd from sys.path.
```

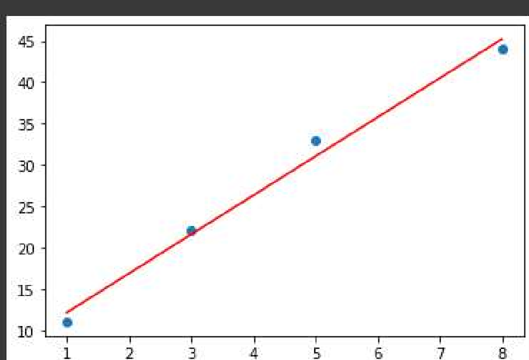
Practical -2

Aim: Implement Linear Regression in Python.

Code:

```
import numpy as np
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt


x=np.array([1,3,5,8]).reshape(-1,1)
y=np.array([11,22,33,44])
print(x,"\n")

a=LinearRegression().fit(x,y)
a.score(x,y)

y_pr = a.predict(x)
print(y_pr,"\n")


plt.scatter(x,y)
plt.plot(x,y_pr,color="r")
plt.show()
```

```
[[1]
 [3]
 [5]
 [8]]

[12.13084112 21.58878505 31.04672897 45.23364486]
```

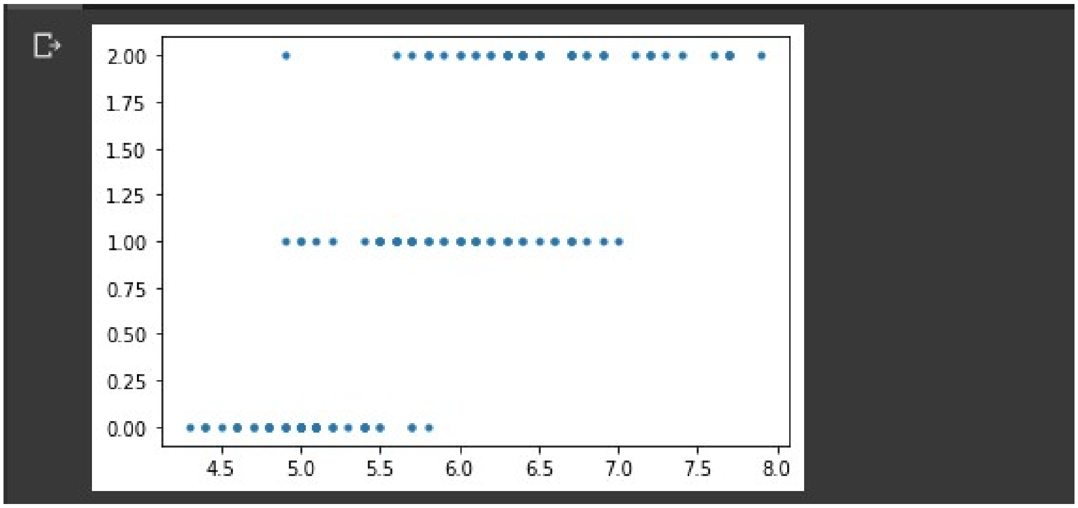


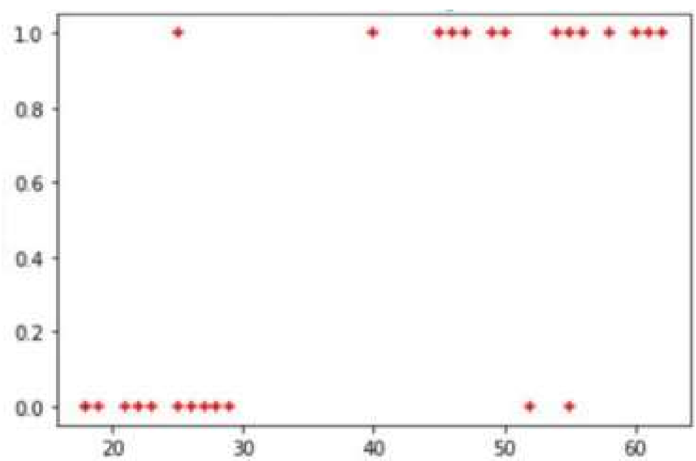
Practical -3

Aim: Implement Logistic Regression in Python

Code:

```
import numpy as np
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
import pandas as pd
from sklearn import datasets
from sklearn.linear_model import LogisticRegression

iris= datasets.load_iris()
X = iris.data[:, :2]
y = iris.target
model=LogisticRegression()
model.fit(X,y)
model.predict(X[:2,:])
model.predict_proba(X[:2,:])
model.score(X,y)
t=X[:,0]
plt.scatter(t,y,s=10)
plt.show()
data=pd.read_csv("/content/drive/MyDrive/ML/ins.csv")
data.rename({'\tbought_insurance':'brought_insurance'})
data.shape
data.head()
plt.title(" Ins vs Age ")
plt.scatter(data['age'],data['bought_ins'],marker='+',color='red')
plt.xlabel(" Age ")
plt.ylabel(" Insurance ")
from sklearn.model_selection import train_test_split
Xtrain,Xtest,ytrain,ytest=train_test_split( data[['age']],data[['bought_ins']],test_size=0.2)
model.fit(Xtrain,ytrain)
y_pred=model.predict(Xtest)
score=model.score(Xtest,ytest)
print(" Accuracy Score is : ",score)
model.coef_
model.intercept_
```


2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0


1.0
0.8
0.6
0.4
0.2
0.0
20
30
40
50
60

Practical -4

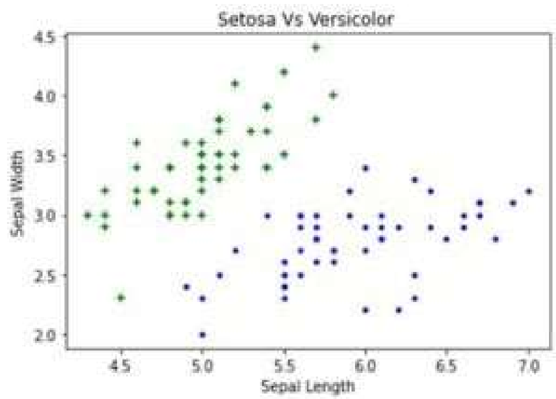
Aim: Implement SVM Classifier in python

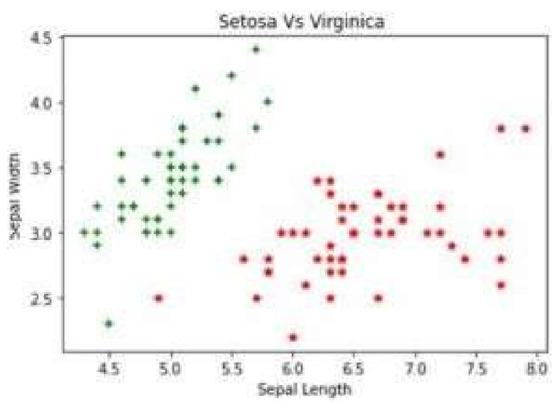
Code:

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC

dataset=load_iris()
dataset.feature_names
dataset.target_names
dataframe=pd.DataFrame(dataset.data,columns=dataset.feature_names)
dataframe['target']=dataset.target
dataframe['flow_name']=dataframe.target.apply(lambda x: dataset.target_names[x])
dataframe.head(5)
dataframe[55:60]
dataframe[105:110]
dataframe_setosa=dataframe[0:50]
dataframe_setosa.head(5)
dataframe_versicolor=dataframe[50:100]
dataframe_versicolor.head(5)
dataframe_virginica=dataframe[100:150]
dataframe_virginica.head(5)


plt.title('Setosa Vs Versicolor ')
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
plt.scatter(dataframe_setosa['sepal length (cm)'], dataframe_setosa['sepal width(cm)'],color="green",marker='+')
plt.scatter(dataframe_versicolor['sepal length (cm)'], dataframe_versicolor['sepal width(cm)'],color="blue",marker='.')


plt.title('Setosa Vs Virginica ')
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
plt.scatter(dataframe_setosa['sepal length (cm)'], dataframe_setosa['sepal width(cm)'],color="green",marker='+')
plt.scatter(dataframe_virginica['sepal length (cm)'], dataframe_virginica['sepal width(cm)'],color="red",marker='*')
X = dataframe.drop(['target','flow_name'], axis='columns')
```

```
y = dataframe['target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
svmModel=SVC()
svmModel.fit(X_train,y_train)
svmModel.score(X_test,y_test)
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	flow_name
100	6.3	3.3	6.0	2.5	2	virginica
101	5.8	2.7	5.1	1.9	2	virginica
102	7.1	3.0	5.9	2.1	2	virginica
103	6.3	2.9	5.6	1.8	2	virginica
104	6.5	3.0	5.8	2.2	2	virginica



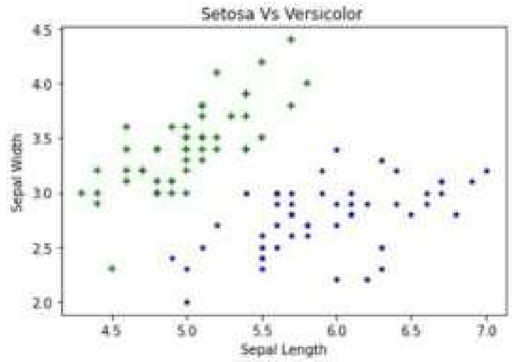


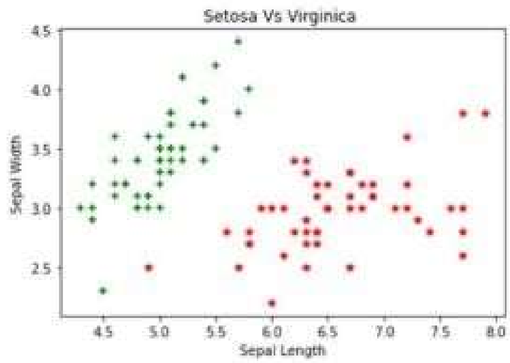
Practical -5

Aim: To implement KNN classifier in Python

Code:

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

dataset=load_iris()
dataset.feature_names
dataset.target_names
dataframe=pd.DataFrame(dataset.data,columns=dataset.feature_names)
dataframe['target']=dataset.target
dataframe['flower_name']=dataframe.target.apply(lambda x: dataset.target_names[x])
dataframe.head(5)
dataframe_setosa=dataframe[0:50]
dataframe_versicolor=dataframe[50:100]
dataframe_virginica=dataframe[100:150]

plt.title('Setosa Vs Versicolor ')
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
plt.scatter(dataframe_setosa['sepal length (cm)'], dataframe_setosa['sepal width(cm)'],color="green",marker='+')
plt.scatter(dataframe_versicolor['sepal length (cm)'], dataframe_versicolor['sepal width(cm)'],color="blue",marker='.')

plt.title('Setosa Vs Virginica ')
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
plt.scatter(dataframe_setosa['sepal length (cm)'], dataframe_setosa['sepal width(cm)'],color="green",marker='+')
plt.scatter(dataframe_virginica['sepal length (cm)'], dataframe_virginica['sepal width(cm)'],color="red",marker='*')

X = dataframe.drop(['target','flower_name'], axis='columns')
y = dataframe['target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model.fit(X_train,y_train)
knn_model.score(X_test,y_test)
```


Setosa Vs Versicolor
Sepal Width
Sepal Length
4.5
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0


Setosa Vs Virginica
Sepal Width
Sepal Length
4.5
4.0
3.5
3.0
2.5
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0

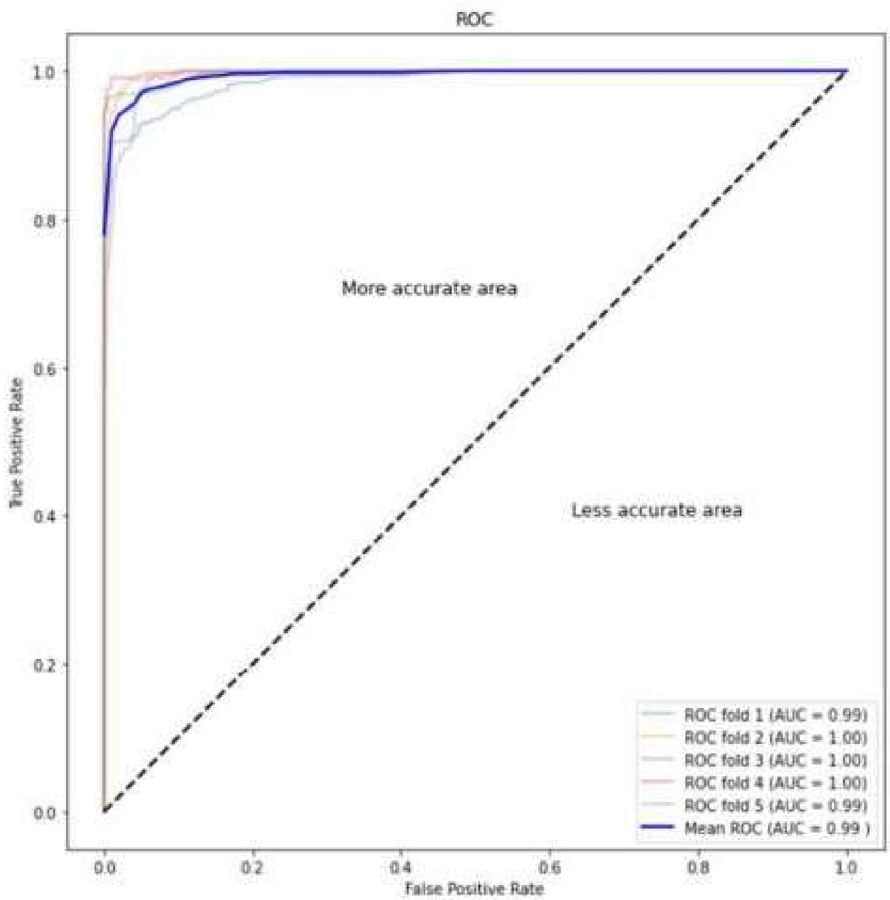
Practical -6

Aim: Study and Implement K-Fold cross validation and ROC

Code:

```
import matplotlib.pylab as plt
from scipy import interp
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_curve,auc
from sklearn.model_selection import StratifiedKFold
import matplotlib.patches as patches
import numpy as np
import pandas as pd
data = pd .read_csv('/content/drive/MyDrive/ML/voice.csv')
print(data.columns)
label_value_count = data.label.value_counts()
print(label_value_count)
print(data.info())
dict = {'label':{'male':1,'female':0}}
data.replace(dict,inplace = True)
x = data.loc[:, data.columns != 'label']
y = data.loc[:,'label']
random_state = np.random.RandomState(0)
clf = RandomForestClassifier(random_state=random_state)
cv = StratifiedKFold(n_splits=5,shuffle=False)
fig1 = plt.figure(figsize=[12,12])
ax1 = fig1.add_subplot(111,aspect = 'equal')
tprs = []
aucs = []
mean_fpr = np.linspace(0,1,100)
i = 1
for train,test in cv.split(x,y):
 prediction = clf.fit(x.iloc[train],y.iloc[train]).predict_proba(x.iloc[test])
 fpr, tpr, t = roc_curve(y[test], prediction[:, 1])
 tprs.append(interp(mean_fpr, fpr, tpr))
 roc_auc = auc(fpr, tpr)
 aucs.append(roc_auc)
 plt.plot(fpr, tpr, lw=2, alpha=0.3, label='ROC fold %d (AUC = %0.2f)' % (i, roc_auc))
 i= i+1
plt.plot([0,1],[0,1],linestyle = '--',lw = 2,color = 'black')
mean_tpr = np.mean(tprs, axis=0)
mean_auc = auc(mean_fpr, mean_tpr)
plt.plot(mean_fpr, mean_tpr, color='blue',
 label=r'Mean ROC (AUC = %0.2f )' % (mean_auc),lw=2, alpha=1)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC')
```

```
plt.legend(loc="lower right")
plt.text(0.32,0.7,'More accurate area',fontsize = 12)
plt.text(0.63,0.4,'Less accurate area',fontsize = 12)
plt.show()
```



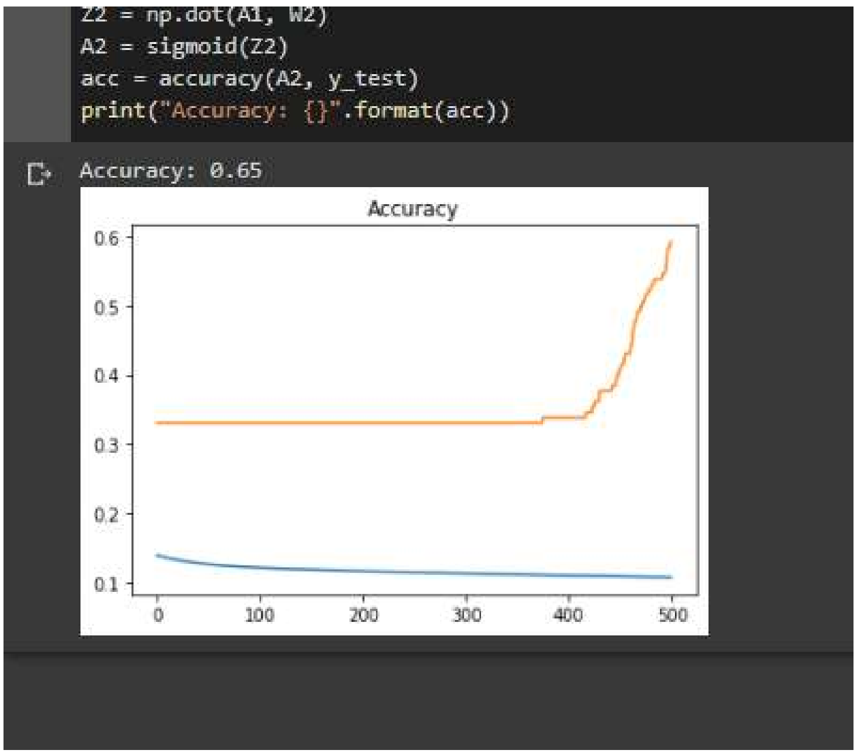
Practical -7

Aim: Implement BPNN Classifier in python.

Code:

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
data = load_iris()
X=data.data
y=data.target
y = pd.get_dummies(y).values
y[:3]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=20, random_state=4)
learning_rate = 0.1
iterations = 500
N = y_train.size
input_size = 4
hidden_size = 2
output_size = 3
results = pd.DataFrame(columns=["mse", "accuracy"])
np.random.seed(10)
W1 = np.random.normal(scale=0.5, size=(input_size, hidden_size))
W2 = np.random.normal(scale=0.5, size=(hidden_size , output_size))
def sigmoid(x):
 return 1 / (1 + np.exp(-x))
def mean_squared_error(y_pred, y_true):
 return ((y_pred - y_true)**2).sum() / (2*y_pred.size)

def accuracy(y_pred, y_true):
 acc = y_pred.argmax(axis=1) == y_true.argmax(axis=1)
 return acc.mean()
for itr in range(iterations):
 Z1 = np.dot(X_train, W1)
 A1 = sigmoid(Z1)
 Z2 = np.dot(A1, W2)
 A2 = sigmoid(Z2)

 mse = mean_squared_error(A2, y_train)
 acc = accuracy(A2, y_train)
 results=results.append({"mse":mse, "accuracy":acc},ignore_index=True )

 E1 = A2 - y_train
 dW1 = E1 * A2 * (1 - A2)
 E2 = np.dot(dW1, W2.T)
```

```
 dW2 = E2 * A1 * (1 - A1)
 W2_update = np.dot(A1.T, dW1) / N
 W1_update = np.dot(X_train.T, dW2) / N
 W2 = W2 - learning_rate * W2_update
 W1 = W1 - learning_rate * W1_update
results.mse.plot(title="Mean Squared Error")
results.accuracy.plot(title="Accuracy")
Z1 = np.dot(X_test, W1)
A1 = sigmoid(Z1)
Z2 = np.dot(A1, W2)
A2 = sigmoid(Z2)
acc = accuracy(A2, y_test)
print("Accuracy: {}".format(acc))
```



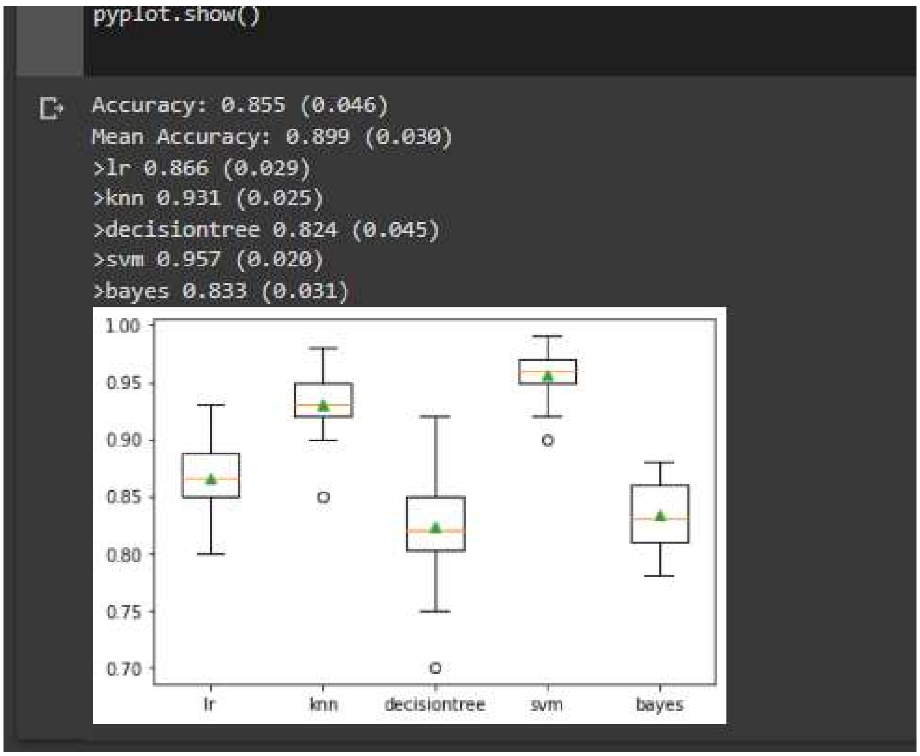
Practical -8

Aim: Study and Implement various Ensemble method of classifier : Bagging,Boosting and Stacking

Code:

```
from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.ensemble import BaggingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from matplotlib import pyplot

#Bagging
X, y = make_classification(n_samples=1000, n_features=20, n_informative=15, n_redundant=5,
random_state=5)
model = BaggingClassifier()
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
n_scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1, error_score='raise')
print('Accuracy: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))

#Boosting
from sklearn.ensemble import GradientBoostingClassifier
X, y = make_classification(n_samples=1000, n_features=20, n_informative=15, n_redundant=5,
random_state=7)
model = GradientBoostingClassifier()
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
n_scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
print('Mean Accuracy: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))
def evaluate_model(model, X, y):
 cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
 scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1, error_score='raise')
 return scores

#Stacking
X, y = make_classification(n_samples=1000, n_features=20, n_informative=15, n_redundant=5,
random_state=1)
models = dict()
models['lr'] = LogisticRegression()
models['knn'] = KNeighborsClassifier()
models['decisiontree'] = DecisionTreeClassifier()
```

```
models['svm'] = SVC()
models['bayes'] = GaussianNB()
results, names = list(), list()
for name, model in models.items():
 scores = evaluate_model(model, X, y)
 results.append(scores)
 names.append(name)
 print('>%s %.3f (%.3f)' % (name, mean(scores), std(scores)))
pyplot.boxplot(results, labels=names, showmeans=True)
pyplot.show()
```

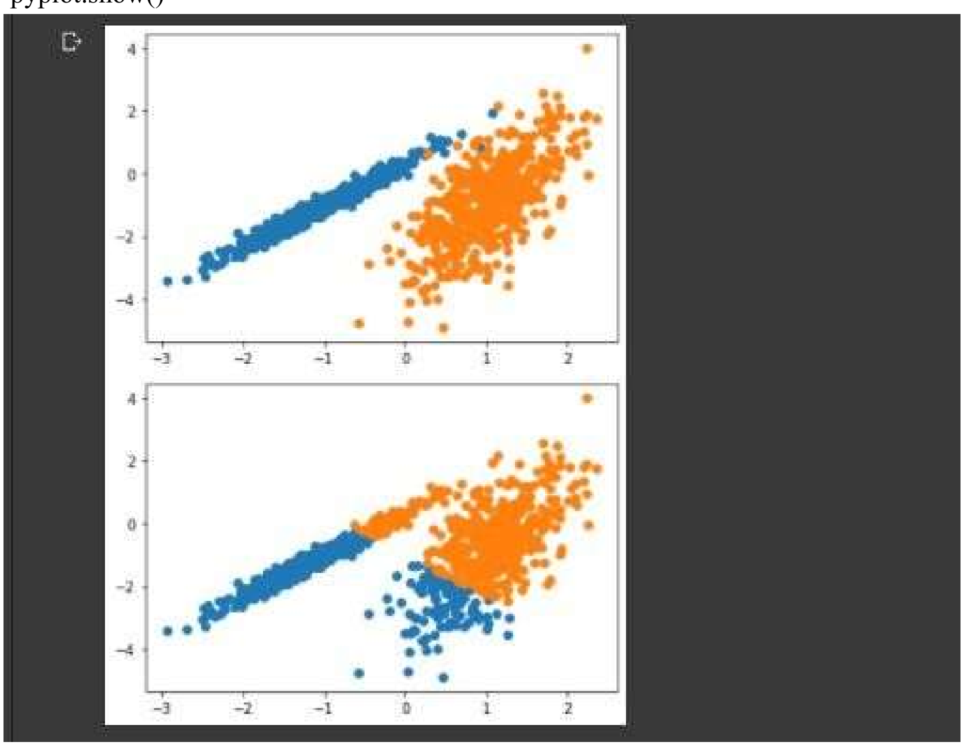


Practical -9

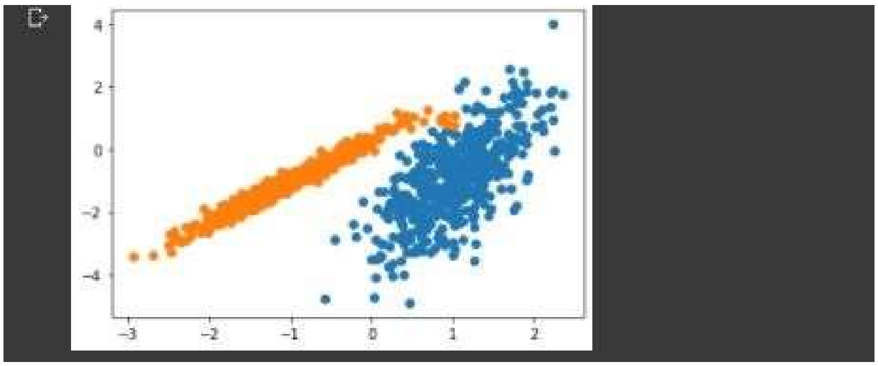
Aim: Implement various Clustering algorithm in python

Code:

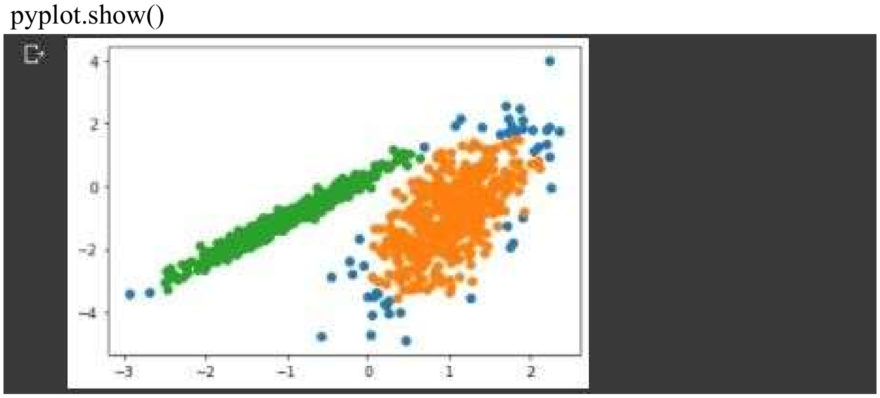
```
from numpy import where
from sklearn.datasets import make_classification
from matplotlib import pyplot
from numpy import unique
X, y = make_classification(n_samples=1000, n_features=2, n_informative=2, n_redundant=0,
n_clusters_per_class=1, random_state=4)
for class_value in range(2):
 row_ix = where(y == class_value)
pyplot.scatter(X[row_ix, 0], X[row_ix, 1])
pyplot.show()

#KMeans
from sklearn.cluster import KMeans
model = KMeans(n_clusters=2)
model.fit(X)
ypredicted = model.predict(X)
clusters = unique(ypredicted)
for cluster in clusters:
 row_ix = where(ypredicted == cluster)
 pyplot.scatter(X[row_ix, 0], X[row_ix, 1])
 pyplot.show()
```



```
#BIRCH
from sklearn.cluster import Birch
model = Birch(threshold=0.01, n_clusters=2)
model.fit(X)
ypredicted = model.predict(X)
clusters = unique(ypredicted)
for cluster in clusters:
 row_ix = where(ypredicted == cluster)
 pyplot.scatter(X[row_ix, 0], X[row_ix, 1])
 pyplot.show()
```

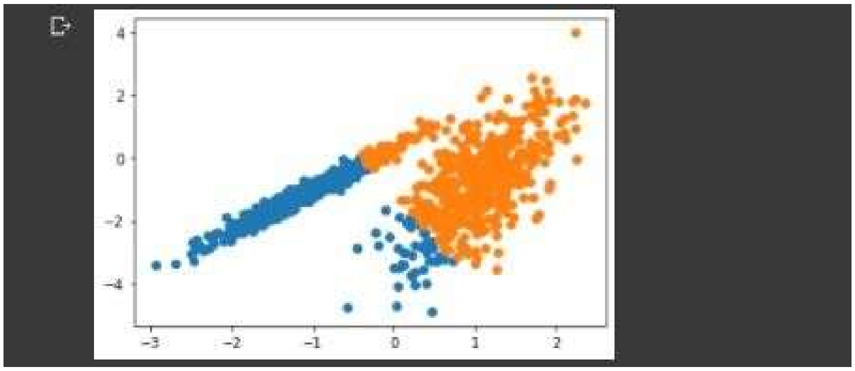


```
#DBSCAN
from sklearn.cluster import DBSCAN
model = DBSCAN(eps=0.30, min_samples=9)
ypredicted = model.fit_predict(X)
clusters = unique(ypredicted)
for cluster in clusters:
 row_ix = where(ypredicted == cluster)
 pyplot.scatter(X[row_ix, 0], X[row_ix, 1])
 pyplot.show()
```

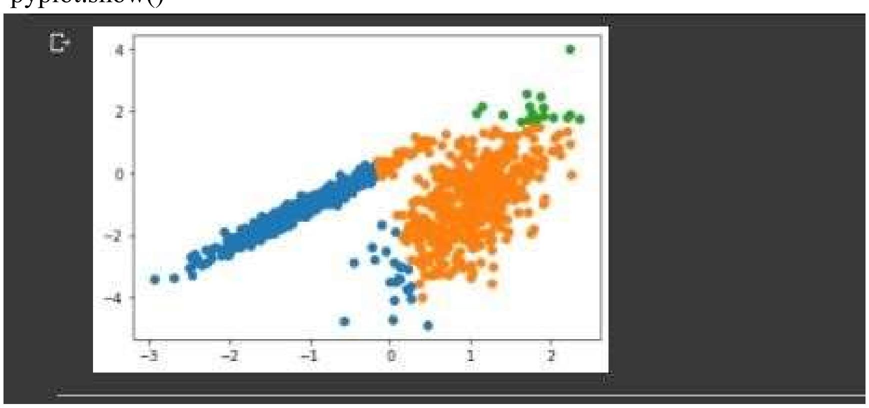


```
#MiniBatchKMeans
from sklearn.cluster import MiniBatchKMeans
model = MiniBatchKMeans(n_clusters=2)
model.fit(X)
ypredicted = model.fit_predict(X)
clusters = unique(ypredicted)
```

```
for cluster in clusters:
 row_ix = where(ypredicted == cluster)
 pyplot.scatter(X[row_ix, 0], X[row_ix, 1])
 pyplot.show()
```



```
#MeanShift
from sklearn.cluster import MeanShift
model = MeanShift()
ypredicted = model.fit_predict(X)
clusters = unique(ypredicted)
for cluster in clusters:
 row_ix = where(ypredicted == cluster)
 pyplot.scatter(X[row_ix, 0], X[row_ix, 1])
 pyplot.show()
```



Practical -10

Aim: Study and Implement various Dimensionality technique like PCA and LDA

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

Code:

```
#PCA
dataset = pd.read_csv('/content/drive/MyDrive/ML/WineQT.csv')
dataset.head(5)
X = dataset.iloc[:, 0:11].values
y = dataset.iloc[:, 11].values
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
from sklearn.decomposition import PCA
pca = PCA(n_components = 2)
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)
explained_variance = pca.explained_variance_ratio_
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
classifier.score(X_test,y_test)
```

```
0.5283842794759825
```

```
#LDA:
dataset = pd.read_csv('/content/drive/MyDrive/ML/WineQT.csv')
dataset.head(5)
X = dataset.iloc[:, 0:11].values
y = dataset.iloc[:, 11].values
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
lda = LDA(n_components = 2)
X_train = lda.fit_transform(X_train, y_train)
X_test = lda.transform(X_test)
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
classifier.score(X_test,y_test)
```

```
0.6637554585152838
```

Practical -11

Aim: Course project on Recommender System, Sentiment Analysis, Image Captioning, Object detection

Recommendation engines are a subclass of machine learning which generally deal with ranking or rating products / users. Loosely defined, a recommender system is a system which predicts ratings a user might give to a specific item. These predictions will then be ranked and returned back to the user.

Sentiment analysis can help you determine the ratio of positive to negative engagements about a specific topic. You can analyze bodies of text, such as comments, tweets, and product reviews, to obtain insights from your audience.

Image Captioning is the process of generating textual description of an image. It uses both Natural Language Processing and Computer Vision to generate the captions. The dataset will be in the form [image → captions]. The dataset consists of input images and their corresponding output captions.

Object Recognition is a technology that lies under the broader domain of Computer Vision. This technology is capable of identifying objects that exist in images and videos and tracking them. Object Recognition also known as Object Detection, has various applications like face recognition, vehicle recognition, pedestrian counting, self-driving vehicles, security systems, and a lot more.

Code:

```
import pandas as pd
from random import randint
import numpy as np
from scipy.sparse import csr_matrix
from scipy.sparse.linalg import svds

def generate_data(n_books = 3000, n_genres = 10, n_authors = 450, n_publishers = 50, n_readers =
30000, dataset_size = 100000):
 d = pd.DataFrame(
 {
 'book_id' : [randint(1, n_books) for _ in range(dataset_size)],
 'author_id' : [randint(1, n_authors) for _ in range(dataset_size)],
 'book_genre' : [randint(1, n_genres) for _ in range(dataset_size)],
 'reader_id' : [randint(1, n_readers) for _ in range(dataset_size)],
 'num_pages' : [randint(75, 700) for _ in range(dataset_size)],
 'book_rating' : [randint(1, 10) for _ in range(dataset_size)],
 'publisher_id' : [randint(1, n_publishers) for _ in range(dataset_size)],
 'publish_year' : [randint(2000, 2021) for _ in range(dataset_size)],
 'book_price' : [randint(1, 200) for _ in range(dataset_size)],
 'text_lang' : [randint(1,7) for _ in range(dataset_size)]
 }
 ).drop_duplicates()
 return d
d = generate_data(dataset_size = 1000)
d.to_csv('data.csv', index = False)
```

```
def normalize(pred_ratings):
 return (pred_ratings - pred_ratings.min()) / (pred_ratings.max() - pred_ratings.min())
def generate_prediction_df(mat, pt_df, n_factors):
 if not 1 <= n_factors < min(mat.shape):
 raise ValueError("Must be 1 <= n_factors < min(mat.shape)")
 u, s, v = svds(mat, k = n_factors)
 s = np.diag(s)
 pred_ratings = np.dot(np.dot(u, s), v)
 pred_ratings = normalize(pred_ratings)
 pred_df = pd.DataFrame(
 pred_ratings,
 columns = pt_df.columns,
 index = list(pt_df.index)
 ).transpose()
 return pred_df

def recommend_items(pred_df, usr_id, n_recs):
 usr_pred = pred_df[usr_id].sort_values(ascending = False).reset_index().rename(columns =
{usr_id : 'sim'})
 rec_df = usr_pred.sort_values(by = 'sim', ascending = False).head(n_recs)
 return rec_df
if___name___== '__main__':
 PATH = '/content/data.csv'
 df = pd.read_csv(PATH)
 print(df.shape)
 pt_df = df.pivot_table(
 columns = 'book_id',
 index = 'reader_id',
 values = 'book_rating'
 ).fillna(0)
 mat = pt_df.values
 mat = csr_matrix(mat)
 pred_df = generate_prediction_df(mat, pt_df, 10)
 print(recommend_items(pred_df, 5, 5))
```

```
import pandas as pd
from random import randint
import numpy as np
from scipy.sparse import csr_matrix
from scipy.sparse.linalg import svds

def generate_data(n_books = 3000, n_genres = 10, n_authors = 450, n_publishers = 50, n_readers =
30000, dataset_size = 100000):
 d = pd.DataFrame(
 {
 'book_id' : [randint(1, n_books) for _ in range(dataset_size)],
 'author_id' : [randint(1, n_authors) for _ in range(dataset_size)],
 'book_genre' : [randint(1, n_genres) for _ in range(dataset_size)],
 'reader_id' : [randint(1, n_readers) for _ in range(dataset_size)],
 'num_pages' : [randint(75, 700) for _ in range(dataset_size)],
 'book_rating' : [randint(1, 10) for _ in range(dataset_size)],
 'publisher_id' : [randint(1, n_publishers) for _ in range(dataset_size)],
 'publish_year' : [randint(2000, 2021) for _ in range(dataset_size)],
 'book_price' : [randint(1, 200) for _ in range(dataset_size)],
 'text_lang' : [randint(1,7) for _ in range(dataset_size)]
 }
 ).drop_duplicates()
 return d
```

```
[6] d = generate_data(dataset_size = 1000)
    d.to_csv('data.csv', index = False)
```


```
[6] d = generate_data(dataset_size = 1000)
    d.to_csv('data.csv', index = False)
```


```
def normalize(pred_ratings
 return (pred_ratings - pred_ratings.min()) / (pred_ratings.max() - pred_ratings.min())
```

```
[8] def generate_prediction_df(mat, pt_df, n_factors):
     if not 1 <= n_factors < min(mat.shape):
      raise ValueError("Must be 1 <= n_factors < min(mat.shape)")
     u, s, v = svds(mat, k = n_factors)
     s = np.diag(s)
     pred_ratings = np.dot(np.dot(u, s), v)
     pred_ratings = normalize(pred_ratings)
     pred_df = pd.DataFrame(
      pred_ratings,
      columns = pt_df.columns,
      index = list(pt_df.index)
     ).transpose()
     return pred_df
```

```
if __name__ == '__main__':
 PATH = '/content/data.csv'
 df = pd.read_csv(PATH)
 print(df.shape)
 pt_df = df.pivot_table(
  columns = 'book_id',
  index = 'reader_id',
  values = 'book_rating'
 ).fillna(0)
 mat = pt_df.values
 mat = csr_matrix(mat)
 pred_df = generate_prediction_df(mat, pt_df, 10)
print(recommend_items(pred_df, 5, 5))

(1000, 10)
```