SVM

```
from sklearn.svm import SVC
data=pd.read_csv("iris.csv")
data      df = pd.DataFrame(iris.data,columns=iris.feature_names)        print(df)
df0 = df[:50] # setosa     df1 = df[50:100] # versicolor       df2 = df[100:] # virginica
plt.xlabel('Petal Length')       plt.ylabel('Petal Width')
plt.scatter(df0['petal length (cm)'], df0['petal width (cm)'],color="green",marker='+')
plt.scatter(df1['petal length (cm)'], df1['petal width (cm)'],color="blue",marker='.')
df['target'] = iris.target
df['flower_name'] =df.target.apply(lambda x: iris.target_names[x])       print(df)
x = df.drop(['target','flower_name'], axis='columns')      y = df.target
X_train, X_test, y_train, y_test = train_test_split(x,y, test_size=1)        model = SVC()
model.fit(X_train, y_train)             model.score(X_test, y_test)
```

Supervised learning

```
from sklearn.linear_model import LinearRegression        import random
feature_set = []      target_set = []
rows = 200      limit = 2000
for i in range(0, rows):
   x = random.randint(0, limit)
   y = random.randint(0, limit)
   z = random.randint(0, limit)
feature_set.append([x,y,z])   function = (10*x) + (2*y) + (0*z)
target_set.append(function)      model = LinearRegression()
model.fit(feature_set, target_set)
test_set=[[8,10,0]]
prediction= model.predict(test_set)
print('Prediction : ' ,prediction)
#Unsupervised Learning
from sklearn import datasets
import matplotlib.pyplot as plt
iris_df = datasets.load_iris() print(dir(iris_df))
print(iris_df.feature_names)
print(iris_df.target)
print(iris_df.target_names)
x_axis = iris_df.data[:,0] y_axis = iris_df.data[:,1]
plt.scatter(x_axis, y_axis, c=iris_df.target)     plt.show()
```

K-means

```
from sklearn.cluster import KMeans
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from matplotlib import pyplot as plt
df=pd.read_csv('income.csv')     df.head()
plt.scatter(df['Age'],df['Income($)'])
plt.xlabel('Age')  plt.ylabel('Income($)')
df1 = df[df.cluster==0]
df2 = df[df.cluster==1]
df3 = df[df.cluster==2]
plt.scatter(df1['Age'],df1['Income($)'],color='green')
plt.scatter(df2['Age'],df2['Income($)'],color='red')   plt.scatter(df3['Age'],df3['Income($)'],color='black')
plt.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1],color='purple',marker='*',label='Centroid')
km = KMeans(n_clusters=3)
predicted = km.fit_predict(df[['Age','Income($)']])
df['cluster']=predicted
df.head()
df1 = df[df.cluster==0]
df2 = df[df.cluster==1]
df3 = df[df.cluster==2]
plt.scatter(df1['Age'],df1['Income($)'],color='green')
plt.scatter(df2['Age'],df2['Income($)'],color='red')
plt.scatter(df3['Age'],df3['Income($)'],color='black')
plt.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1],color='purple',marker='*',label='Centroid')
```

Ai appn 12 prac

```
import random
def display(room):
   print(room)
# 1 means dirty location
# 0 means clean location
room = [
[1, 1, 1, 1],
[1, 1, 1, 1],
[1, 1, 1, 1],
[1, 1, 1, 1],
]
print("All the locations in the room are dirty")
display(room)
x=0 # rows
y=0 # cols
while x < 4:
   while y < 4:
      room[x][y] = random.choice([0,1])
      y+=1
   x+=1
   y=0
print("Before cleaning the room the Vacuum cleaner detects all the random dirts in the following locations")
display(room)
x=0
y=0
z=0 #number of rooms cleaned
while x < 4:
   while y < 4:
      if room[x][y] == 1:
         print("Vacuum cleaner is in this location now : ", x, y)
         room[x][y] = 0
         print("Location cleaned : ", x, y)
         z+=1
         y+=1
   x+=1
   y=0
print("Number of locations cleaned = ", z)
Performance=(100-((z/16)*100))
print("Room is clean now")
display(room)
print("Cleaning Performance = ", Performance,"%")
```

RULE based

```
python -m spacy install en_core_web_sm
import spacy
from spacy.matcher import Matcher
nlp=spacy.load('en_core_web_sm')
matcher=Matcher(nlp.vocab)
doc=nlp("I loved dogs but now i love cats more")
pattern=[{'LEMMA':'love'} ,{'POS':'NOUN'}]
matcher.add('DOG_PATTERN',[pattern])
matches=matcher(doc)
for match_id, start, end in matches:
  matched_span=doc[start:end]
  print(matched_span.text)

doc=nlp("2022 Fifa world cup : ITALY Wins")
pattern=[{'IS_DIGIT' :True} ,{'LOWER':'fifa'}, {'LOWER':'world'}, {'LOWER':'cup'}, {'IS_PU
NCT':True}]
matcher.add('FIFA_PATTERN', [pattern])
matches=matcher(doc)
```

Fuzzy

```
A = dict()
B = dict()
C = dict()
A = {"a" : 0.3 , "b" : 0.4 , "c" : 0.6 , "d" : 0.9 , "e" : 0.2}
B = {"a" : 0.2 , "b" : 0.9 , "c" : 0.5 , "d" : 0.7 , "e" : 0.1}
print('Fuzzy Set A : ', A)
print('Fuzzy Set B : ', B)
for A_Key, B_Key in zip(A,B):
  A_Value = A[A_Key]
  B_Value = B[B_Key]
if A_Value > B_Value:   (intersection <)
  C[A_Key] = A_Value
else:
  C[A_Key] = B_Value
print('Union Operation')
print('Fuzzy Set C : ', C)

A = dict()
C = dict()
A = {"a" : 0.3 , "b" : 0.4 , "c" : 0.6 , "d" : 0.9 , "e" : 0.2}
print('Fuzzy Set A : ', A)
for A_Key in A:
  C[A_Key] = 1 - A[A_Key]
print('Complement Operation')
print('Fuzzy Set C : ', C)
```

chatbot

```
<aiml version="1.0.1" encoding="UTF-8">
<category> (doctor_chat.amil)
<pattern>HELLO *</pattern>
<template>
Well, hello patient
</template>
</category>
<category>
<pattern>WHO ARE YOU</pattern>
<template>
I am a Doctor bot
</template>
</category>
</aiml>
(std-startup.xml)
<aiml version="1.0.1" encoding="UFT-8">
<category>
<pattern>LOAD AIML B</pattern>
<template>
<learn>doctor_chat.aiml</learn>
</template>
</category>
</aiml>
(doc.py)
import aiml
kernel = aiml.Kernel()
kernel.learn("std-startup.xml")
kernel.respond("load aiml b")
while True:
input_text = input(">Human: ")
response = kernel.respond(input_text)
print(">Bot: "+response)
```