MCA II year III Semester

Course Code IT 31L – Practical's

Part - B

Knowledge Representation, Artificial Intelligence, Machine Learning and Deep Learning

Sr. No.	Program Title	Page No.	Remark	Faculty Sign
1	Find the correlation matrix.			
2	Plot the correlation plot on dataset and visualize giving an overview of relationships among data on iris data.			
3	Analysis of covariance: variance (ANOVA), if data have categorical variables on iris data.			
4	Apply linear regression Model techniques to predict the data on any dataset.			
5	Apply logical regression Model techniques to predict the data on any dataset			
6	Clustering algorithms for unsupervised classification			
7	Association algorithms for supervised classification on any dataset			
8	Developing and implementing Decision Tree model on the dataset			
9	Bayesian classification on any dataset			
10	SVM classification on any dataset			
11	Text Mining algorithms on unstructured dataset			
12	Plot the cluster data using python visualizations.			
13	Creating & Visualizing Neural Network for the given data. (Use python)			
14	Recognize optical character using ANN.			
15	Write a program to implement CNN			
16	Write a program to implement CNN			
17	Write a program to implement GAN			
18	Web scraping experiments (by using tools)			

1. **Find the correlation matrix.**

Code: -

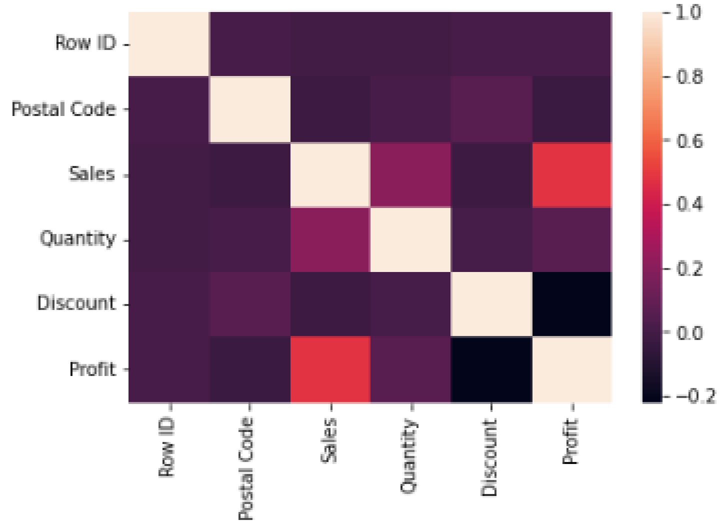
```
import scipy.stats as st
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
superdata=pd.read_excel("superarketstore.xls")
!pip install xlrd
np.corrcoef(superdata ['Sales'], superdata ['Profit'])
superdata.corr()
sns.heatmap(superdata.corr())
```

Output: -

Out[37]:

	Row ID	Postal Code	Sales	Quantity	Discount	Profit
Row ID	1.000000	0.009671	-0.001359	-0.004016	0.013480	0.012497
Postal Code	0.009671	1.000000	-0.023854	0.012761	0.058443	-0.029961
Sales	-0.001359	-0.023854	1.000000	0.200795	-0.028190	0.479064
Quantity	-0.004016	0.012761	0.200795	1.000000	0.008623	0.066253
Discount	0.013480	0.058443	-0.028190	0.008623	1.000000	-0.219487
Profit	0.012497	-0.029961	0.479064	0.066253	-0.219487	1.000000

Out[34]: <AxesSubplot:>



2. **Plot the correlation plot on dataset and visualize giving an overview of relationships among data on iris data.**

Code: -

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn import metrics
sns.set()
iris_data=pd.read_csv('iris.csv')
iris_data
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

150 rows × 5 columns

```
iris_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   species       150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

iris_data.describe()

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

iris_data[iris_data.duplicated()]

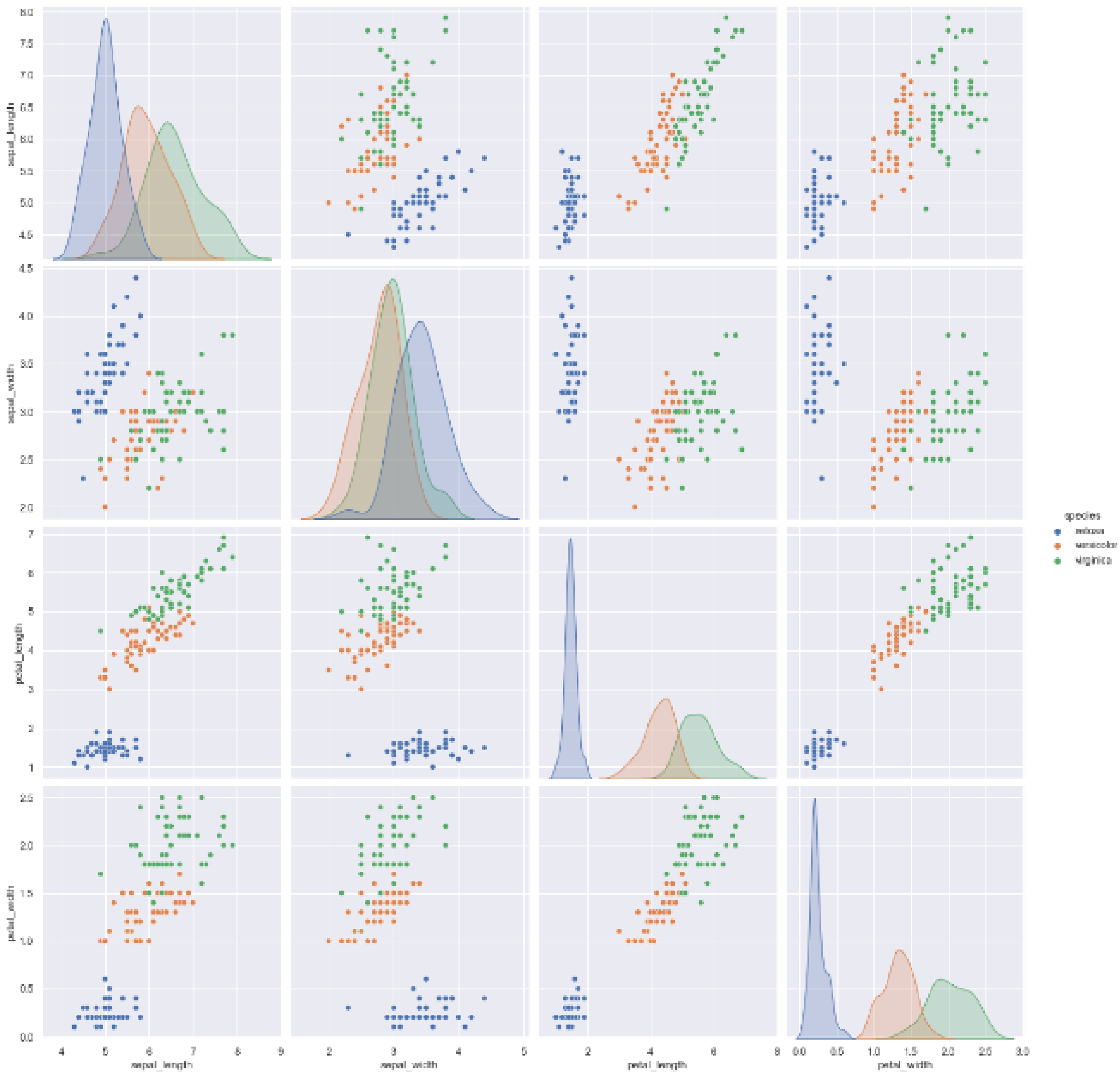
	sepal_length	sepal_width	petal_length	petal_width	species
142	5.8	2.7	5.1	1.9	virginica

iris_data['species'].value_counts()

```
setosa        50
versicolor    50
virginica     50
Name: species, dtype: int64
```

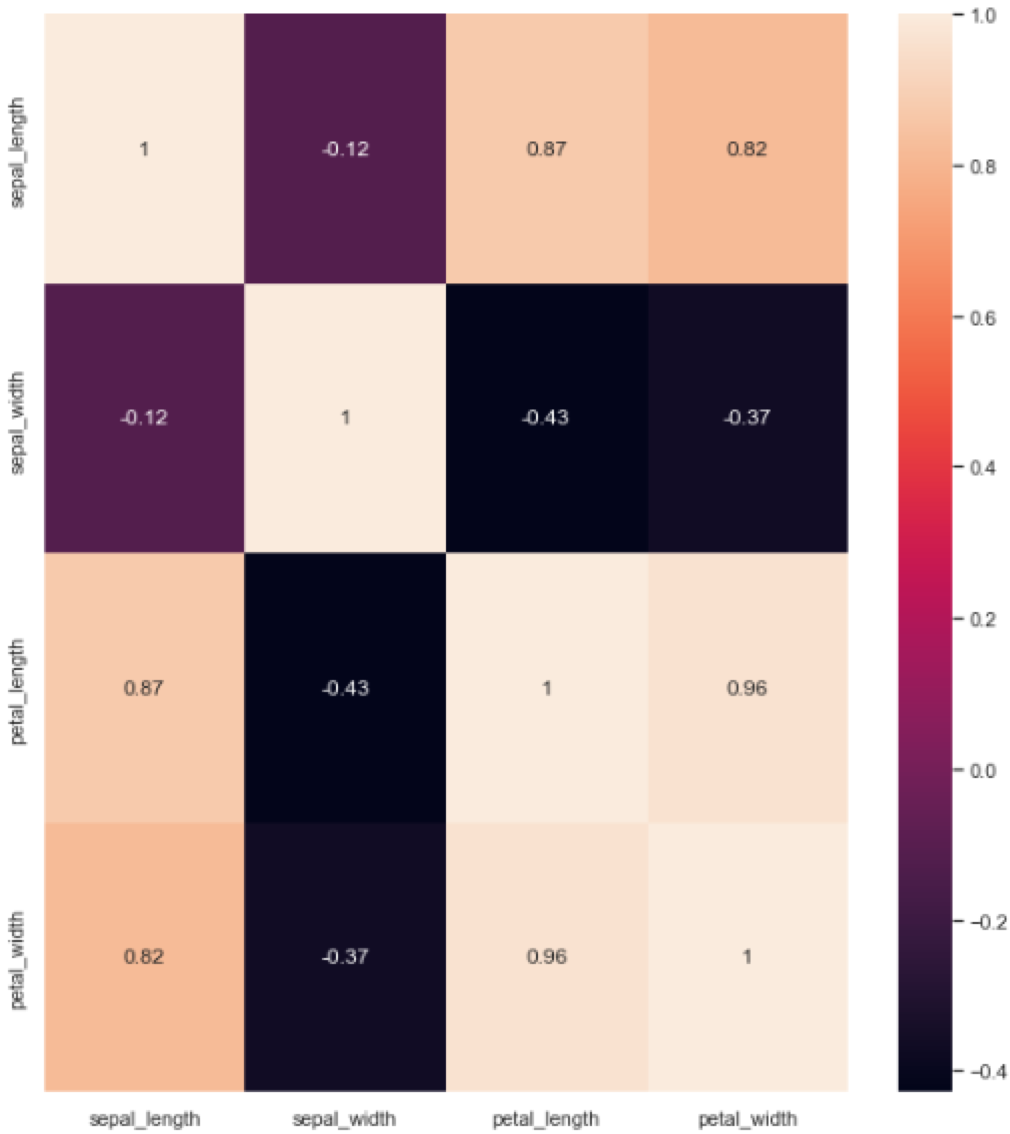
sns.pairplot(iris_data,hue='species',height=4)

```
<seaborn.axisgrid.PairGrid at 0x1a2dc0b35e0>
```




```
plt.figure(figsize=(10,11))
sns.heatmap(iris_data.corr(),annot=True)
plt.plot()
```

[ ]

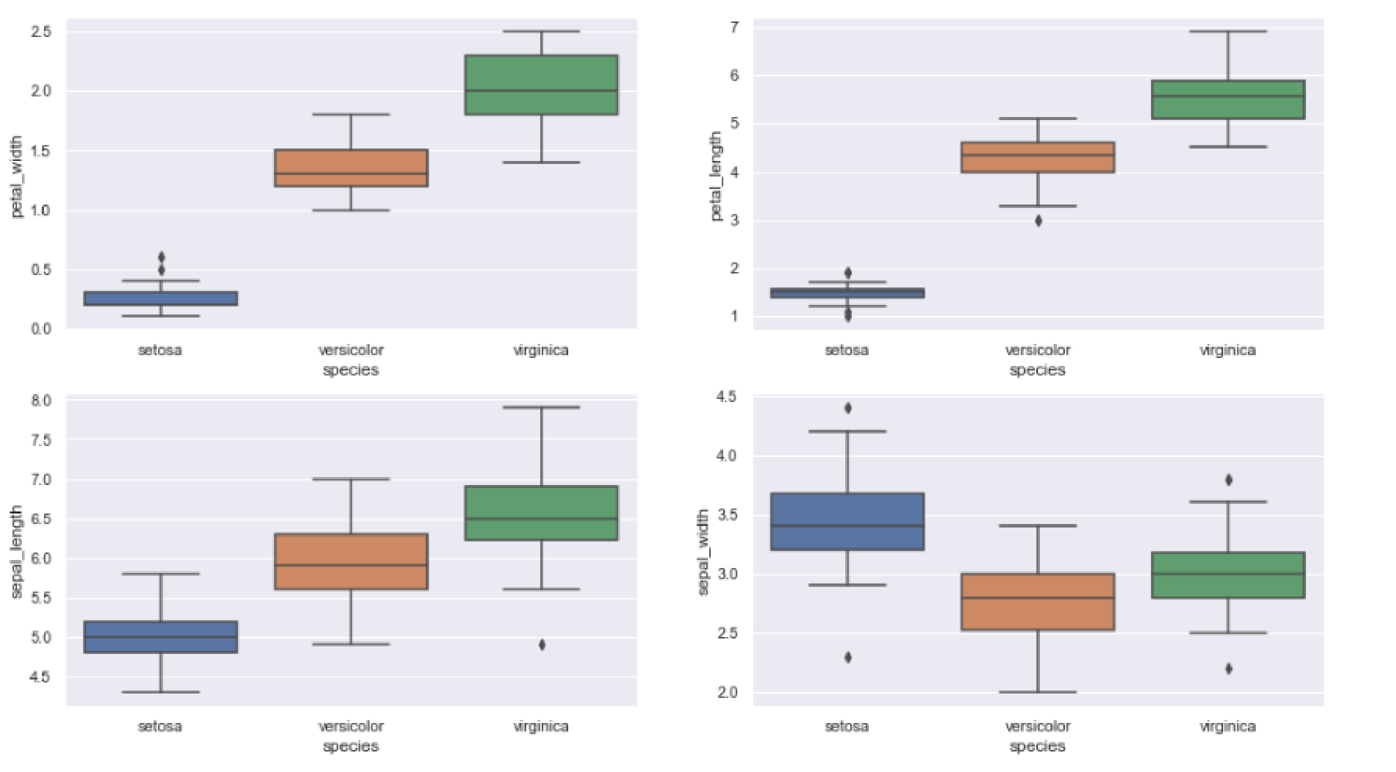



```
iris_data.groupby('species').agg(['mean','median'])
```

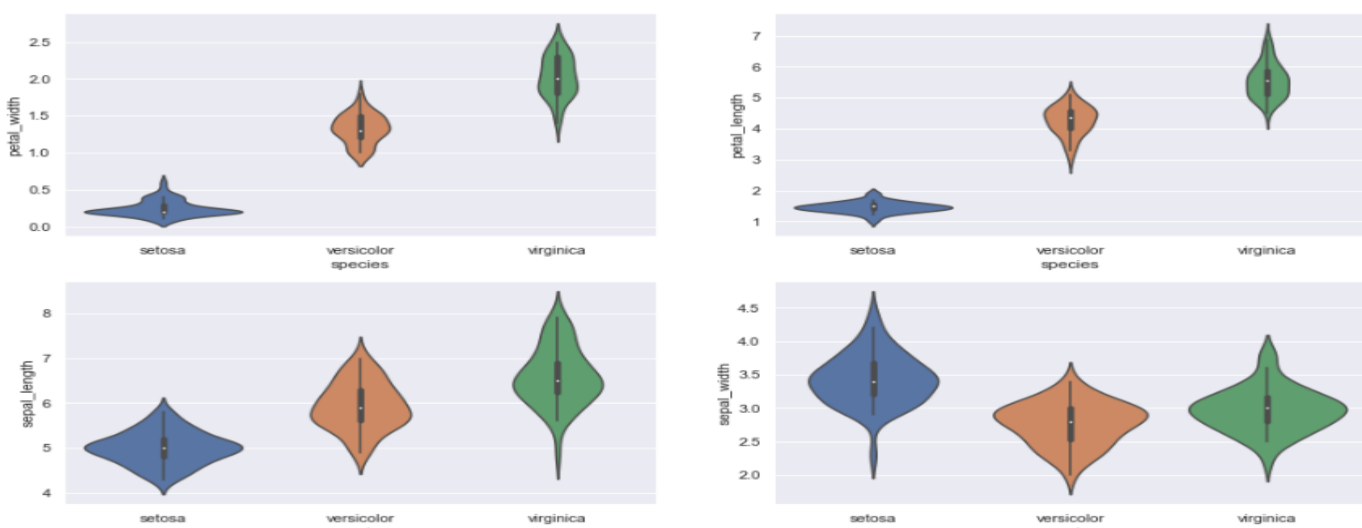
	sepal_length		sepal_width		petal_length		petal_width	
	mean	median	mean	median	mean	median	mean	median
species								
setosa	5.006	5.0	3.428	3.4	1.462	1.50	0.246	0.2
versicolor	5.936	5.9	2.770	2.8	4.260	4.35	1.326	1.3
virginica	6.588	6.5	2.974	3.0	5.552	5.55	2.026	2.0

```
fig, axes = plt.subplots(2, 2, figsize=(16,9))
sns.boxplot( y='petal_width', x= 'species', data=iris_data, orient='v' , ax=axes[0, 0])
sns.boxplot( y='petal_length', x= 'species', data=iris_data, orient='v' , ax=axes[0, 1])
sns.boxplot( y='sepal_length', x= 'species', data=iris_data, orient='v' , ax=axes[1, 0])
```

```
sns.boxplot( y='sepal_width', x= 'species', data=iris_data, orient='v', ax=axes[1, 1])
plt.show()
```




```
fig, axes = plt.subplots(2, 2, figsize=(16,9))
sns.violinplot( y='petal_width', x= 'species', data=iris_data, orient='v' , ax=axes[0, 0])
sns.violinplot( y='petal_length', x= 'species', data=iris_data, orient='v' , ax=axes[0, 1])
sns.violinplot( y='sepal_length', x= 'species', data=iris_data, orient='v' , ax=axes[1, 0])
sns.violinplot( y='sepal_width', x= 'species', data=iris_data, orient='v', ax=axes[1, 1])
plt.show()
```



3. Analysis of covariance: variance (ANOVA), if data have categorical variables on iris data.

Code: -

```
import numpy as np
import pandas as pd
df=pd.read_csv('iris_data.csv')
df.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

4. Apply linear regression Model techniques to predict the data on any dataset.

Code: -

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
data=pd.read_csv('Salary_Data.csv')
X=data.iloc[:,:-1].values
y=data.iloc[:,1].values
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.25,random_state=0)
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
X_train=scaler.fit_transform(X_train)
X_test=scaler.fit_transform(X_test)
from sklearn.linear_model import LinearRegression
regressor=LinearRegression()
regressor.fit(X_train,y_train)
y_pre=regressor.predict(X_test[[0]])
y_pre
```

Output: -

```
Out[17]: array([36569.76758981])
```

5. **Apply logical regression Model techniques to predict the data on any dataset.**

Code: -

```
import pandas as pd
import matplotlib.pyplot as plt
df=pd.read_csv('Social_Network_Ads.csv')
X=df[['Age','EstimatedSalary']]
y=df['Purchased']
from sklearn.linear_model import LogisticRegression
model=LogisticRegression()
model.fit(X,y)
```

```
LogisticRegression()
```

```
Scaled_Age=(df['Age']-df['Age'].min()) / (df['Age'].max()-df['Age'].min())
Scaled_Salary=(df['EstimatedSalary']-df['EstimatedSalary'].min()) / (df['EstimatedSalary'].max()-
df['EstimatedSalary'].min())
X=pd.concat([Scaled_Age,Scaled_Salary],axis=1)
y=df['Purchased']
model_scaled = LogisticRegression()
model_scaled.fit(X,y)
```

```
LogisticRegression()
```

```
def get_scaled(pt):
    age,sal = pt[0],pt[1]
    sc_age=(age-df['Age'].min()) / (df['Age'].max()-df['Age'].min())
    sc_sal=(sal-df['EstimatedSalary'].min()) / (df['EstimatedSalary'].max()-df['EstimatedSalary'].min())
    return sc_age,sc_sal
q1=get_scaled([52,130000])
q2=get_scaled([25,40000])
model_scaled.predict([q1])
```

```
array([1], dtype=int64)
```

```
model_scaled.predict([q2])
```

```
array([0], dtype=int64)
```

```
from sklearn.preprocessing import MinMaxScaler
```

X = df[['Age','EstimatedSalary']]

scaler = MinMaxScaler()

scaler.fit(X)

X_scaled = scaler.transform(X)

X_scaled

```
Out[39]: array([[0.02380952, 0.02962963],
       [0.4047619 , 0.03703704],
       [0.19047619, 0.20740741],
       [0.21428571, 0.31111111],
       [0.02380952, 0.45185185],
       [0.21428571, 0.31851852],
       [0.21428571, 0.51111111],
       [0.33333333, 1.        ],
       [0.16666667, 0.13333333],
       [0.4047619 , 0.37037037],
       [0.19047619, 0.48148148],
       [0.19047619, 0.27407407],
       [0.04761905, 0.52592593],
       [0.33333333, 0.02222222],
       [0.        , 0.4962963 ],
       [0.26190476, 0.48148148],
       [0.69047619, 0.07407407],
       [0.64285714, 0.08148148],
       [0.66666667, 0.0962963 ],
```

model = LogisticRegression()

model.fit(X_scaled,df['Purchased'])

```
LogisticRegression()
```

model.score(X_scaled,df['Purchased'])

```
0.83
```

y_pre=model.predict(X_scaled)

y_act=df['Purchased']

y_pre

```
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1,
       0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0,
       1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0,
       1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0,
       0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1,
       1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1,
       0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0,
       1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1,
       0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0], dtype=int64)
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test=train_test_split(X,y,test_size=0.25)
scaler = MinMaxScaler()
scaler.fit(X_train)
X_train_scaled=scaler.transform(X_train)
model = LogisticRegression()
model.fit(X_train_scaled,y_train)
```

```
LogisticRegression()
```

```
train_score=model.score(X_train_scaled,y_train)
train_score
```

```
0.8266666666666667
```

```
X_test_scaled=scaler.transform(X_test)
test_score=model.score(X_test_scaled,y_test)
test_score
```

```
: 0.88
```

6. **Clustering algorithms for unsupervised classification.**

Code: -

```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
df=pd.read_csv('Mall_Customers_dataset.csv')
df.head()
```

	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

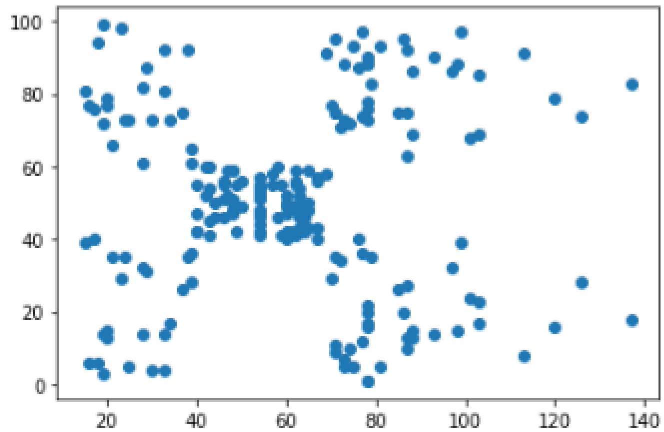
```
X = df[['Annual Income (k$)','Spending Score (1-100)']]
X
```

	Annual Income (k$)	Spending Score (1-100)
0	15	39
1	15	81
2	16	6
3	16	77
4	17	40
...	...	...
195	120	79
196	126	28
197	126	74
198	137	18
199	137	83

200 rows × 2 columns

```
plt.scatter(X['Annual Income (k$)'],X['Spending Score (1-100)'])
```

<matplotlib.collections.PathCollection at 0x2416daabfa0>




```
from sklearn.cluster import KMeans
model = KMeans(n_clusters=5)
model.fit(X)
```

```
KMeans(n_clusters=5)
```

```
model.cluster_centers_
```

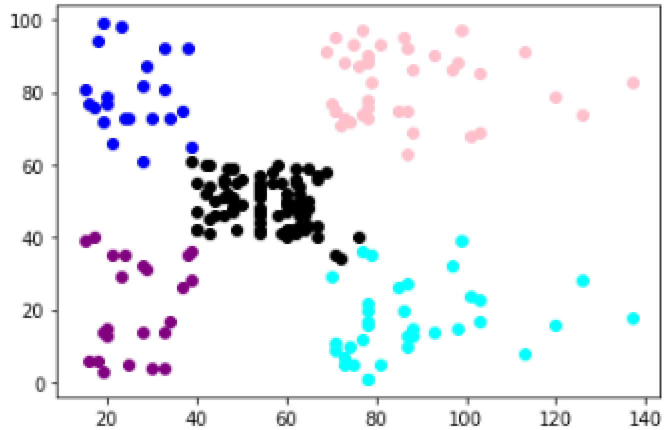
```
array([[55.2962963 , 49.51851852],
       [25.72727273, 79.36363636],
       [86.53846154, 82.12820513],
       [88.2       , 17.11428571],
       [26.30434783, 20.91304348]])
```

```
cluster_number = model.predict(X)
len(cluster_number)
```

```
200
```

```
c0 = X[cluster_number==0]
c1 = X[cluster_number==1]
c2 = X[cluster_number==2]
c3 = X[cluster_number==3]
c4 = X[cluster_number==4]
plt.scatter(c0['Annual Income (k$)'],c0['Spending Score (1-100)'],c='black')
plt.scatter(c1['Annual Income (k$)'],c1['Spending Score (1-100)'],c='blue')
plt.scatter(c2['Annual Income (k$)'],c2['Spending Score (1-100)'],c='pink')
plt.scatter(c3['Annual Income (k$)'],c3['Spending Score (1-100)'],c='cyan')
plt.scatter(c4['Annual Income (k$)'],c4['Spending Score (1-100)'],c='purple')
```

<matplotlib.collections.PathCollection at 0x24170d76e50>



model.inertia_

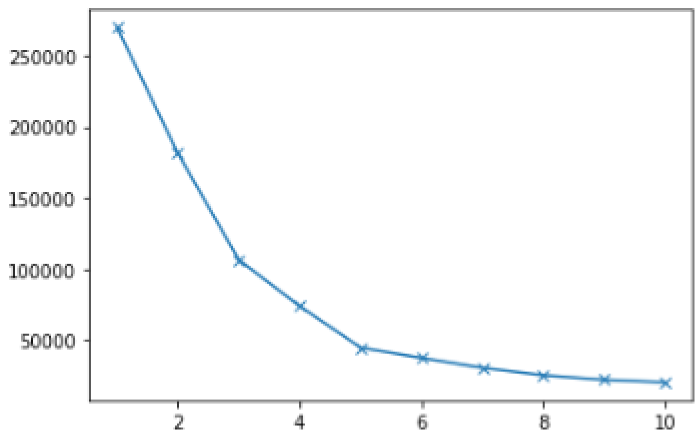
```
44448.45544793369
```

```
WCSS =[]
for i in range(1,11):
   model = KMeans(n_clusters=i)
   model.fit(X)
   WCSS.append(model.inertia_)
WCSS
```

```
[269981.28000000014,
 181363.59595959607,
 106348.37306211119,
 73679.78903948837,
 44448.45544793369,
 37233.81451071002,
 30273.394312070028,
 25011.839349156595,
 21838.863692828916,
 20022.61156762439]
```

```
plt.plot(range(1,11),WCSS,marker = 'x')
```

```
[<matplotlib.lines.Line2D at 0x24170e91550>]
```



7. **Association algorithms for supervised classification on any dataset.**

Code: -

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy.stats as stats
np.random.seed(12)

races =   ["asian","black","hispanic","other","white"]

voter_race = np.random.choice(a= races,
                              p = [0.05, 0.15 ,0.25, 0.05, 0.5],
                              size=1000)

voter_age = stats.poisson.rvs(loc=18,
                              mu=30,
                              size=1000)

voter_frame = pd.DataFrame({"race":voter_race,"age":voter_age})
groups = voter_frame.groupby("race").groups

asian = voter_age[groups["asian"]]
black = voter_age[groups["black"]]
hispanic = voter_age[groups["hispanic"]]
other = voter_age[groups["other"]]
white = voter_age[groups["white"]]
```

```
stats.f_oneway(asian, black, hispanic, other, white)
```

```
F_onewayResult(statistic=1.7744689357329695, pvalue=0.13173183201930463)
```

```
import statsmodels.api as sm
from statsmodels.formula.api import ols

model = ols('age ~ race',
            data = voter_frame).fit()

anova_result = sm.stats.anova_lm(model, typ=2)
print (anova_result)
```

```
                sum_sq     df        F        PR(>F)
race       1284.123213    4.0  10.1647  4.561324e-08
Residual  31424.995787  995.0      NaN           NaN
```

```
np.random.seed(12)

voter_race = np.random.choice(a= races,
                              p = [0.05, 0.15 ,0.25, 0.05, 0.5],
                              size=1000)

white_ages = stats.poisson.rvs(loc=18,
                              mu=32,
                              size=1000)

voter_age = stats.poisson.rvs(loc=18,
                              mu=30,
                              size=1000)

voter_age = np.where(voter_race=="white", white_ages, voter_age)

voter_frame = pd.DataFrame({"race":voter_race,"age":voter_age})
groups = voter_frame.groupby("race").groups

asian = voter_age[groups["asian"]]
black = voter_age[groups["black"]]
hispanic = voter_age[groups["hispanic"]]
other = voter_age[groups["other"]]
white = voter_age[groups["white"]]
```

```
stats.f_oneway(asian, black, hispanic, other, white)
```

```
F_onewayResult(statistic=10.164699828386366, pvalue=4.5613242113994585e-08)
```

```
model = ols('age ~ race', data = voter_frame).fit()

anova_result = sm.stats.anova_lm(model, typ=2)
print (anova_result)
```

```
               sum_sq     df        F        PR(>F)
race      1284.123213    4.0  10.1647  4.561324e-08
Residual  31424.995787  995.0      NaN           NaN
```

```
race_pairs = []

for race1 in range(4):
    for race2  in range(race1+1,5):
        race_pairs.append((races[race1], races[race2]))

for race1, race2 in race_pairs:
    print(race1, race2)
    print(stats.ttest_ind(voter_age[groups[race1]],
                          voter_age[groups[race2]]))
```

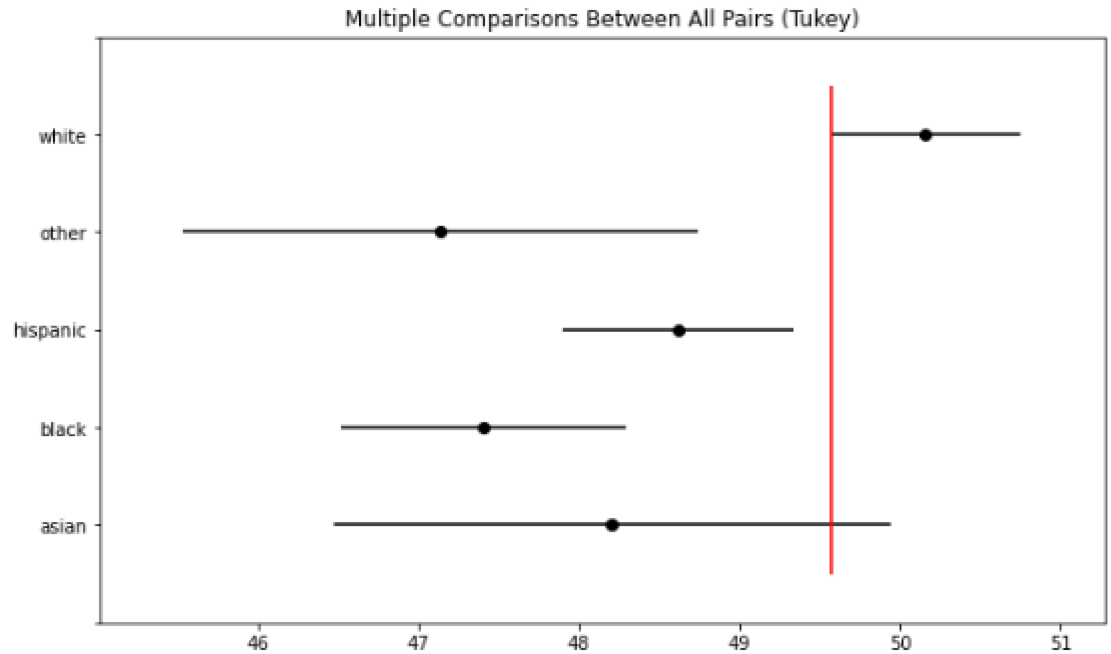
```
asian black
Ttest_indResult(statistic=0.838644690974798, pvalue=0.4027281369339345)
asian hispanic
Ttest_indResult(statistic=-0.42594691924932293, pvalue=0.6704669004240726)
asian other
Ttest_indResult(statistic=0.9795284739636, pvalue=0.3298877500095151)
asian white
Ttest_indResult(statistic=-2.318108811252288, pvalue=0.020804701566400217)
black hispanic
Ttest_indResult(statistic=-1.9527839210712925, pvalue=0.05156197171952594)
black other
Ttest_indResult(statistic=0.28025754367057176, pvalue=0.7795770111117659)
black white
Ttest_indResult(statistic=-5.379303881281835, pvalue=1.039421216662395e-07)
hispanic other
Ttest_indResult(statistic=1.5853626170340225, pvalue=0.11396630528484335)
hispanic white
Ttest_indResult(statistic=-3.516031271411537б, pvalue=0.0004641298649066684)
other white
Ttest_indResult(statistic=-3.763809322077872, pvalue=0.00018490576317593065)
```

```
from statsmodels.stats.multicomp import pairwise_tukeyhsd
```

```
tukey = pairwise_tukeyhsd(endog=voter_age, groups=voter_race, alpha=0.05)

tukey.plot_simultaneous()
plt.vlines(x=49.57,ymin=-0.5,ymax=4.5, color="red")

tukey.summary()
```

Multiple Comparison of Means - Tukey HSD, FWER=0.05

group1	group2	meandiff	p-adj	lower	upper	reject
asian	black	-0.8032	0.9	-3.4423	1.836	False
asian	hispanic	0.4143	0.9	-2.1011	2.9297	False
asian	other	-1.0645	0.8852	-4.2391	2.11	False
asian	white	1.9547	0.175	-0.4575	4.3668	False
black	hispanic	1.2175	0.2318	-0.386	2.821	False
black	other	-0.2614	0.9	-2.7757	2.253	False
black	white	2.7579	0.001	1.3217	4.194	True
hispanic	other	-1.4789	0.4391	-3.863	0.9053	False
hispanic	white	1.5404	0.004	0.3468	2.734	True
other	white	3.0192	0.0028	0.7443	5.2941	True



8. Developing and implementing Decision Tree model on the dataset

Code: -

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
data=pd.read_csv('Salary_Data.csv')
data.head()
```

	YearsExperience	Salary
0	1.1	39343.0
1	1.3	46205.0
2	1.5	37731.0
3	2.0	43525.0
4	2.2	39891.0

```
X=data[['YearsExperience']]
y=data['Salary']
from sklearn.tree import DecisionTreeRegressor
regressor = DecisionTreeRegressor(random_state=0)
regressor.fit(X,y)
```

```
DecisionTreeRegressor(random_state=0)
```

```
regressor.predict([[6.5]])
```

```
array([91738.])
```

9. Bayesian classification on any dataset.

Code: -

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
df=pd.read_csv('iris_data.csv')
df.columns=['sepal_length','sepal_width','petal_length','petal_width','species']
col_names=list(df.columns)
predictors=col_names[0:4]
target=col_names[4]
from sklearn.model_selection import train_test_split
train,test=train_test_split(df,test_size=0.3,random_state=0)

from sklearn.naive_bayes import GaussianNB
Gmodel=GaussianNB()
Gmodel.fit(train[predictors],train[target])
train_Gpred=Gmodel.predict(train[predictors])
test_Gpred=Gmodel.predict(test[predictors])

train_acc_gau=np.mean(train_Gpred==train[target])
test_acc_gau=np.mean(test_Gpred==test[target])
print ("train_acc_gau=",train_acc_gau)
print ("test_acc_gau=",test_acc_gau)

from sklearn.naive_bayes import MultinomialNB
Mmodel=MultinomialNB()
Mmodel.fit(train[predictors],train[target])
train_Mpred=Mmodel.predict(train[predictors])
test_Mpred=Mmodel.predict(test[predictors])

train_acc_multi=np.mean(train_Mpred==train[target])
test_acc_multi=np.mean(test_Mpred==test[target])

print ("train_acc_multi=",train_acc_gau)
print ("test_acc_multi=",test_acc_gau)
```

```
train_acc_gau= 0.9428571428571428
test_acc_gau= 1.0
train_acc_multi= 0.9428571428571428
test_acc_multi= 1.0
```

10. SVM classification on any dataset

Code: -

```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
df=pd.read_csv('Social_Network_Ads.csv')
df.head()
```

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

```
X=df[['Age','EstimatedSalary']]
y=df['Purchased']
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.23, random_state=91)

from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler()
scaler.fit(X_train)
X_train_scaled=scaler.transform(X_train)
X_test_scaled=scaler.transform(X_test)

from sklearn.svm import SVC
model_lin = SVC(kernel='linear')
model_lin.fit(X_train_scaled,y_train)
model_lin.score(X_test_scaled,y_test)
```

```
0.8043478260869565
```

```
model_poly = SVC(kernel='poly')
model_poly.fit(X_train_scaled,y_train)
model_poly.score(X_test_scaled,y_test)
```

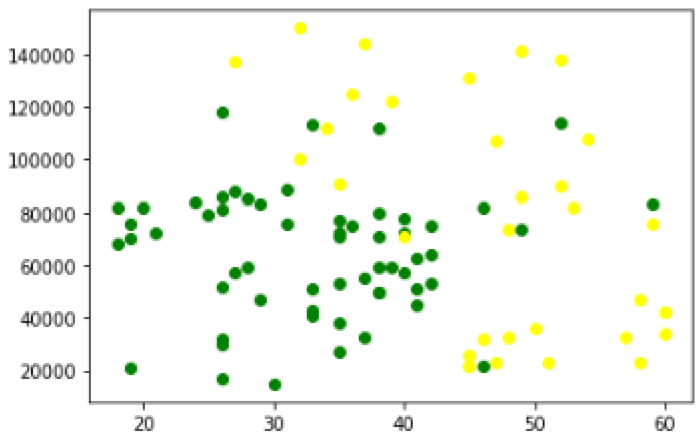
```
0.8913043478260869
```

model_rbf = SVC(kernel='rbf')
model_rbf.fit(X_train_scaled,y_train)
model_rbf.score(X_test_scaled,y_test)

```
0.8913043478260869
```

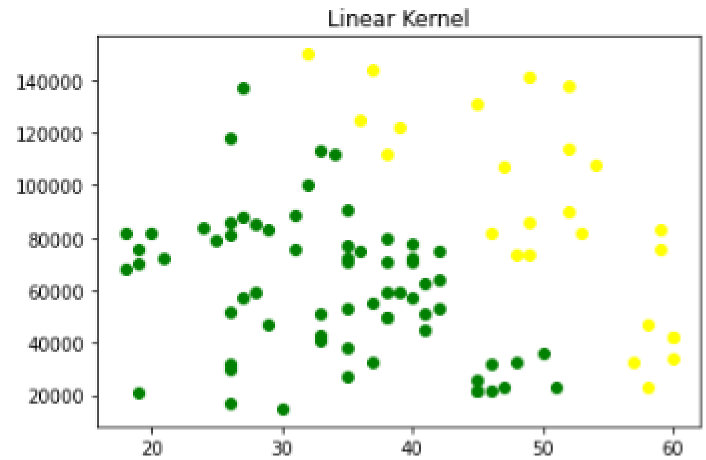
class_0_act = X_test[y_test==0]
class_1_act = X_test[y_test==1]
plt.scatter(class_0_act['Age'],class_0_act['EstimatedSalary'],c='green')
plt.scatter(class_1_act['Age'],class_1_act['EstimatedSalary'],c='yellow')

```
<matplotlib.collections.PathCollection at 0x2579ef68cd0>
```



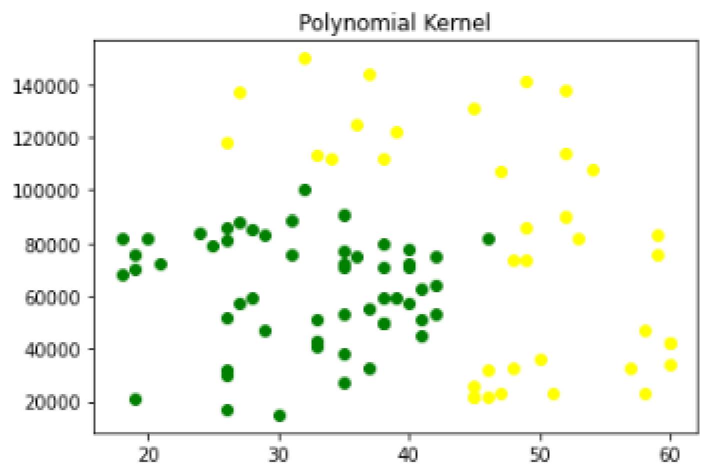
y_pre = model_lin.predict(X_test_scaled)
class_0_pre = X_test[y_pre==0]
class_1_pre = X_test[y_pre==1]
plt.scatter(class_0_pre['Age'],class_0_pre['EstimatedSalary'],c='green')
plt.scatter(class_1_pre['Age'],class_1_pre['EstimatedSalary'],c='yellow')
plt.title('Linear Kernel')

```
Text(0.5, 1.0, 'Linear Kernel')
```



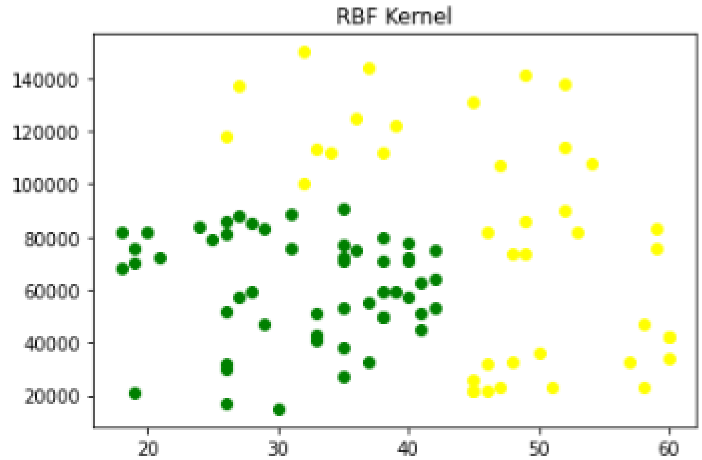

```
y_pre = model_poly.predict(X_test_scaled)
class_0_pre = X_test[y_pre==0]
class_1_pre = X_test[y_pre==1]
plt.scatter(class_0_pre['Age'],class_0_pre['EstimatedSalary'],c='green')
plt.scatter(class_1_pre['Age'],class_1_pre['EstimatedSalary'],c='yellow')
plt.title('Polynomial Kernel')
```

```
Text(0.5, 1.0, 'Polynomial Kernel')
```




```
y_pre = model_rbf.predict(X_test_scaled)
class_0_pre = X_test[y_pre==0]
class_1_pre = X_test[y_pre==1]
plt.scatter(class_0_pre['Age'],class_0_pre['EstimatedSalary'],c='green')
plt.scatter(class_1_pre['Age'],class_1_pre['EstimatedSalary'],c='yellow')
plt.title('RBF Kernel')
```

```
Text(0.5, 1.0, 'RBF Kernel')
```



import numpy as np

plot_data = []

for x in range(0,100,1):

    for y in range(0,100,1):

        plot_data.append([x,y])

plot_data=np.array(plot_data)/100

plot_data

```
array([[0.  , 0.  ],
       [0.  , 0.01],
       [0.  , 0.02],
       ...,
       [0.99, 0.97],
       [0.99, 0.98],
       [0.99, 0.99]])
```

plot_data.shape

```
(10000, 2)
```

y_plot = model_lin.predict(plot_data)

class_0 = plot_data[y_plot==0]

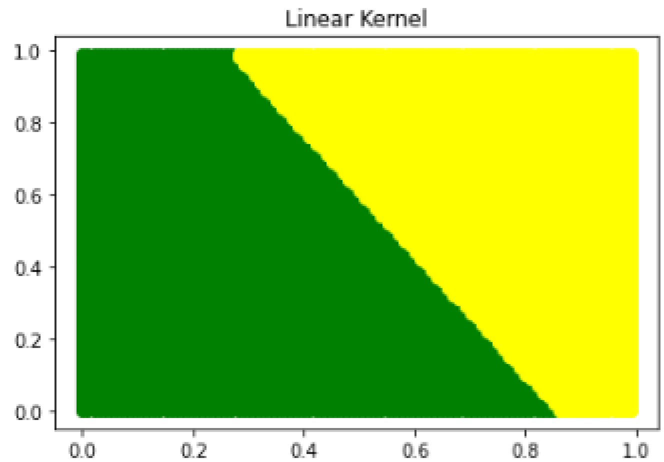
class_1 = plot_data[y_plot==1]

plt.scatter(class_0[:,0],class_0[:,1],c='green')

plt.scatter(class_1[:,0],class_1[:,1],c='yellow')

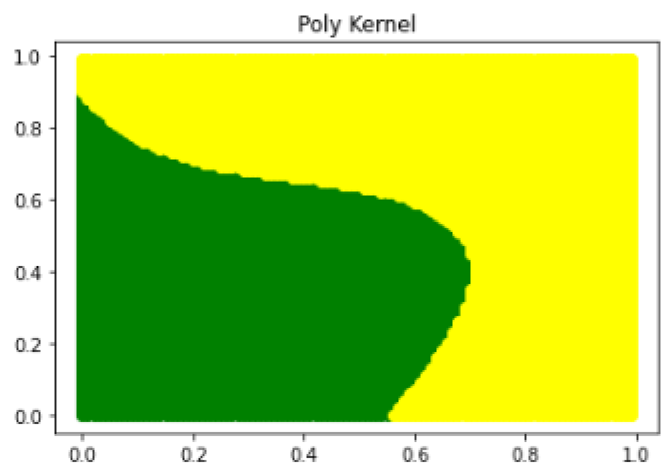
plt.title('Linear Kernel')

```
Text(0.5, 1.0, 'Linear Kernel')
```



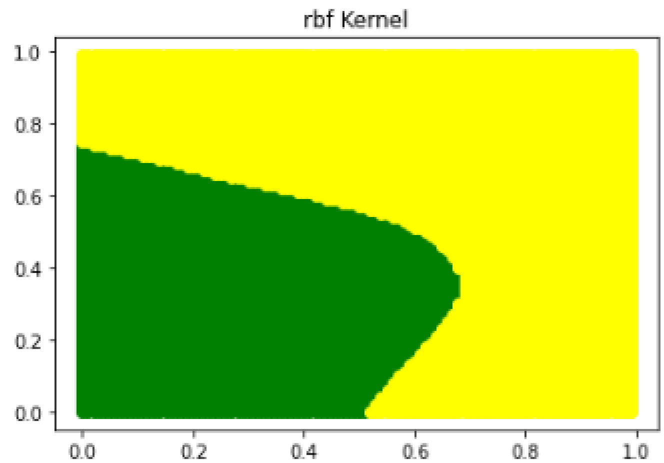

```
y_plot = model_poly.predict(plot_data)
class_0 = plot_data[y_plot==0]
class_1 = plot_data[y_plot==1]
plt.scatter(class_0[:,0],class_0[:,1],c='green')
plt.scatter(class_1[:,0],class_1[:,1],c='yellow')
plt.title('Poly Kernel')
```

```
Text(0.5, 1.0, 'Poly Kernel')
```




```
y_plot = model_rbf.predict(plot_data)
class_0 = plot_data[y_plot==0]
class_1 = plot_data[y_plot==1]
plt.scatter(class_0[:,0],class_0[:,1],c='green')
plt.scatter(class_1[:,0],class_1[:,1],c='yellow')
plt.title('rbf Kernel')
```

```
Text(0.5, 1.0, 'rbf Kernel')
```



pts = np.array([[25,60000],[50,120000]])

pts_scaled = scaler.transform(pts)

pts_scaled

```
array([[0.16666667, 0.33333333],
       [0.76190476, 0.77777778]])
```

y = model_rbf.predict(pts_scaled)

y

```
array([0, 1], dtype=int64)
```

11. Text Mining algorithms on unstructured dataset

Code: -

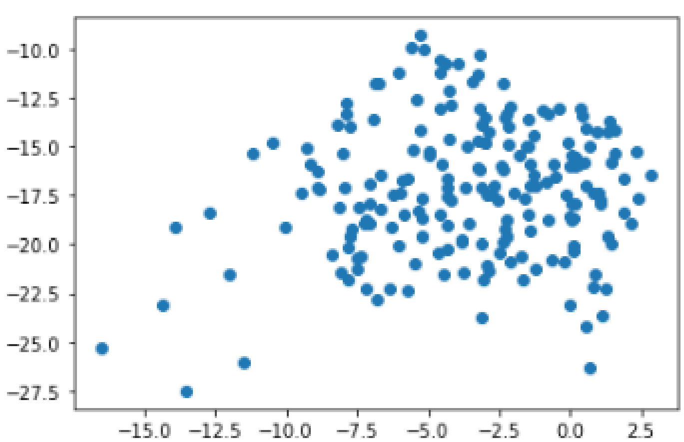
```
from sklearn.datasets import load_digits
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
import numpy as np
data = load_digits().data
pca = PCA(2)
df = pca.fit_transform(data)
df.shape
```

```
(1797, 2)
```

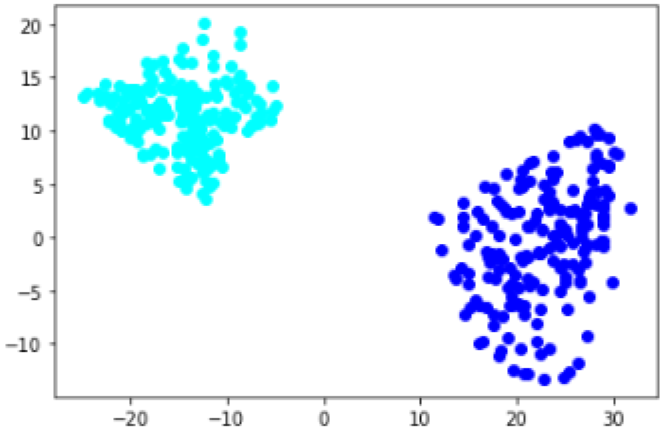
```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters= 10)
label = kmeans.fit_predict(df)

print(label)
```

```
[1 7 3 ... 3 2 9]
```

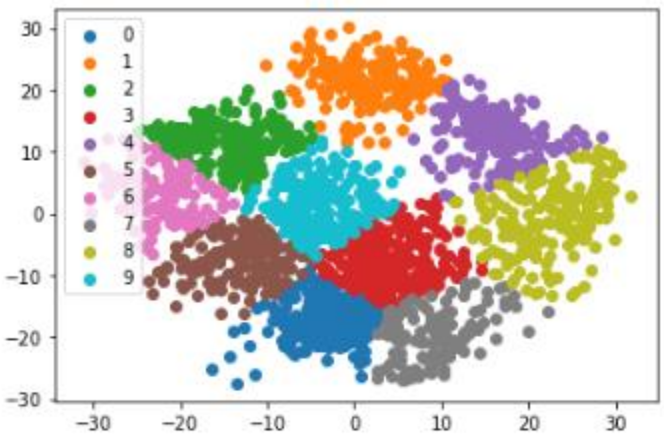
```
import matplotlib.pyplot as plt

filtered_label0 = df[label == 0]

plt.scatter(filtered_label0[:,0] , filtered_label0[:,1])
plt.show()
```



```
filtered_label2 = df[label == 2]

filtered_label8 = df[label == 8]

plt.scatter(filtered_label2[:,0] , filtered_label2[:,1] , color = 'cyan')
plt.scatter(filtered_label8[:,0] , filtered_label8[:,1] , color = 'blue')
plt.show()
```

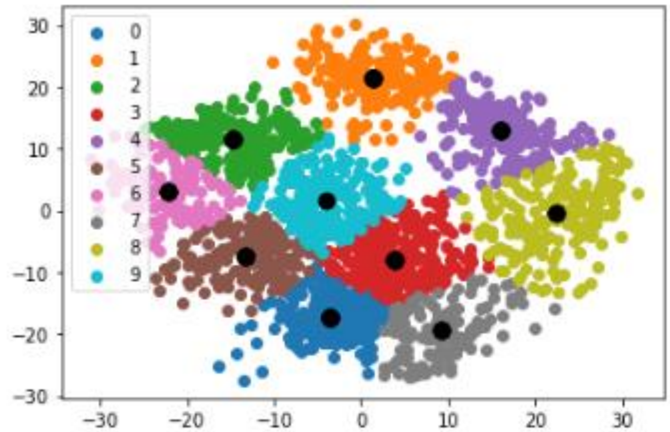


```
u_labels = np.unique(label)


for i in u_labels:
    plt.scatter(df[label == i , 0] , df[label == i , 1] , label = i)
plt.legend()
plt.show()
```



```
centroids = kmeans.cluster_centers_
u_labels = np.unique(label)
```

```
for i in u_labels:
    plt.scatter(df[label == i , 0] , df[label == i , 1] , label = i)
    plt.scatter(centroids[:,0] , centroids[:,1] , s = 80, color = 'k')
plt.legend()
plt.show()
```



12. . Plot the cluster data using python visualizations.

Code: -

```
import tensorflow as tf
from tensorflow import keras
from matplotlib.pyplot import title
from tensorflow.keras.models import Sequential,Model
from tensorflow.keras.layers import Dense, Dropout, Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.layers import LeakyReLU
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),activation='linear',input_shape=(28,28,1),padding='same'))
model.add(LeakyReLU(alpha=0.1))
model.add(MaxPooling2D((2, 2),padding='same'))
model.add(Conv2D(64, (3, 3), activation='linear',padding='same'))
model.add(LeakyReLU(alpha=0.1))
model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
model.add(Conv2D(128, (3, 3), activation='linear',padding='same'))
model.add(LeakyReLU(alpha=0.1))
model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
model.add(Flatten())
model.add(Dense(128, activation='linear'))
model.add(LeakyReLU(alpha=0.1))
model.add(Dense(500, activation='softmax'))
model.compile(loss=keras.losses.categorical_crossentropy,
optimizer=keras.optimizers.Adam(),metrics=['accuracy'])
model.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_21 (Conv2D)          (None, 28, 28, 32)        320

 leaky_re_lu_28 (LeakyReLU)  (None, 28, 28, 32)        0

 max_pooling2d_21 (MaxPoolin  (None, 14, 14, 32)       0
 g2D)

 conv2d_22 (Conv2D)          (None, 14, 14, 64)        18496

 leaky_re_lu_29 (LeakyReLU)  (None, 14, 14, 64)        0

 max_pooling2d_22 (MaxPoolin  (None, 7, 7, 64)         0
 g2D)

 conv2d_23 (Conv2D)          (None, 7, 7, 128)         73856

 leaky_re_lu_30 (LeakyReLU)  (None, 7, 7, 128)         0

 max_pooling2d_23 (MaxPoolin  (None, 4, 4, 128)        0
 g2D)

 flatten_7 (Flatten)         (None, 2048)              0

 dense_14 (Dense)            (None, 128)               262272

 leaky_re_lu_31 (LeakyReLU)  (None, 128)               0

 dense_15 (Dense)            (None, 500)               64500

=================================================================
Total params: 419,444
Trainable params: 419,444
Non-trainable params: 0
```

13. Creating & Visualizing Neural Network for the given data. (Use python)

Code: -

14. Recognize optical character using ANN.

Code: -

```
from tensorflow.keras.datasets import mnist
(x_train,y_train),(x_test,y_test)=mnist.load_data()
x_train.shape
```

```
(60000, 28, 28)
```

```
X_train=x_train.reshape(60000,784)
X_test=x_test.reshape(10000,784)

from tensorflow.keras.utils import to_categorical
y_train=to_categorical(y_train,num_classes=10)
y_test=to_categorical(y_test,num_classes=10)
X_train=X_train/255
X_test=X_test/255
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
model=Sequential()
model.add(Dense(50,activation='relu',input_shape=(784,)))
model.add(Dense(50,activation='relu'))
model.add(Dense(10,activation='softmax'))
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 50)                39250

 dense_1 (Dense)             (None, 50)                2550

 dense_2 (Dense)             (None, 10)                510

=================================================================
Total params: 42,310
Trainable params: 42,310
Non-trainable params: 0
_________________________________________________________________
```

```
model.compile(loss='categorical_crossentropy',metrics=['accuracy'])
model.fit(X_train,y_train,batch_size=64,epochs=10,validation_data=(X_test,y_test))
```

```
Epoch 1/10
938/938 [==============================] - 3s 3ms/step - loss: 0.0241 - accuracy: 0.9928 - val_loss: 0.1354 - val_accuracy: 0.9
725
Epoch 2/10
938/938 [==============================] - 3s 3ms/step - loss: 0.0203 - accuracy: 0.9940 - val_loss: 0.1442 - val_accuracy: 0.9
730
Epoch 3/10
938/938 [==============================] - 3s 3ms/step - loss: 0.0193 - accuracy: 0.9938 - val_loss: 0.1479 - val_accuracy: 0.9
709
Epoch 4/10
938/938 [==============================] - 3s 3ms/step - loss: 0.0187 - accuracy: 0.9941 - val_loss: 0.1434 - val_accuracy: 0.9
730
Epoch 5/10
938/938 [==============================] - 4s 4ms/step - loss: 0.0175 - accuracy: 0.9944 - val_loss: 0.1495 - val_accuracy: 0.9
740
Epoch 6/10
938/938 [==============================] - 4s 4ms/step - loss: 0.0160 - accuracy: 0.9952 - val_loss: 0.1624 - val_accuracy: 0.9
719
Epoch 7/10
938/938 [==============================] - 3s 3ms/step - loss: 0.0151 - accuracy: 0.9953 - val_loss: 0.1518 - val_accuracy: 0.9
715
Epoch 8/10
938/938 [==============================] - 3s 3ms/step - loss: 0.0132 - accuracy: 0.9959 - val_loss: 0.1654 - val_accuracy: 0.9
708
Epoch 9/10
938/938 [==============================] - 3s 4ms/step - loss: 0.0133 - accuracy: 0.9960 - val_loss: 0.1638 - val_accuracy: 0.9
722
Epoch 10/10
938/938 [==============================] - 4s 4ms/step - loss: 0.0129 - accuracy: 0.9960 - val_loss: 0.1710 - val_accuracy: 0.9
728

0]: <keras.callbacks.History at 0x276e4f8b4f0>
```

```
import numpy as np
X_train
```

```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

```
y_train[:5,:]
```

```
array([[0., 0., 0., 0., 0., 1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 1.]], dtype=float32)
```

```
img0 = np.array(X_train[0]).reshape(1,784)
model.predict(img0).argmax()
```

```
5
```

```
def recognise(img):
```

```
  img=np.array(img).reshape(1,784)
  return model.predict(img).argmax()
y_pre=model.predict(X_test).argmax(axis=1)
y_pre
```

```
array([7, 2, 1, ..., 4, 5, 6], dtype=int64)
```

```
len(y_pre)
```

```
10000
```

```
y_test.argmax(axis=1)
```

```
array([7, 2, 1, ..., 4, 5, 6], dtype=int64)
```

```
sum(y_pre==y_test.argmax(axis=1))
```

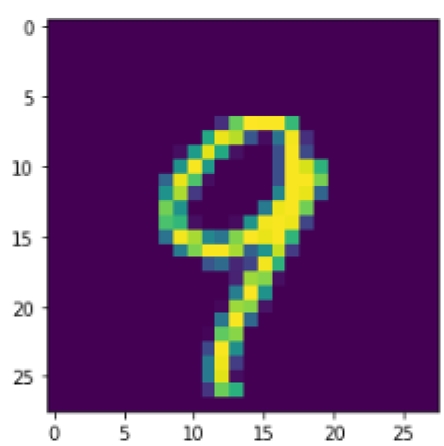
```
9711
```

```
9737/10000
```

```
0.9737
```

```
import matplotlib.pyplot as plt
plt.imshow(np.array(X_test[560]).reshape(28,28))
```

```
<matplotlib.image.AxesImage at 0x276eee2ea60>
```




```
recognise(X_test[560])
```

```
9
```

15. Write a program to implement CNN

Code: -

```
import numpy as np
import pandas as pd
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
os.listdir('/kaggle/input/dogs-vs-cats/')
filenames=os.listdir('../input/dogs-vs-cats/train/train')
len(filenames)
filenames[:5]
df=pd.DataFrame({'filename':filenames})
df.head()
df['class']=df['filename'].apply(lambda X:X[:3])
df.head()
from tensorflow.keras.preprocessing.image import ImageDataGenerator
data_gen=ImageDataGenerator(zoom_range=0.2,shear_range=0.2,horizontal_flip=True,rescale=1/255)
train_data=data_gen.flow_from_dataframe(df,'../input/dogs-vs-
cats/train/train',X='filename',y='class',target_size=(224,224))
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D,MaxPool2D,Flatten,Dense
model=Sequential()
model.add(Conv2D(16,(3,3),activation='relu',input_shape=(224,224,3)))
model.add(MaxPool2D())
model.add(Conv2D(32,(3,3),activation='relu'))
model.add(MaxPool2D())
model.add(Conv2D(64,(3,3),activation='relu'))
model.add(MaxPool2D())
model.add(Conv2D(64,(5,5),activation='relu'))
model.add(MaxPool2D())
model.add(Conv2D(128,(3,3),activation='relu'))
model.add(MaxPool2D())
model.add(Flatten())
model.add(Dense(2,activation='softmax'))
model.summary()
model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
model.fit_generator(train_data,epochs=5)
```

```
import cv2
def get_class(img_path):
    img=cv2.imread(img_path)
    img=cv2.resize(img,(224,224))
    img=img/255
    op=model.predict(img.reshape(1,224,224,3)).argmax()
    return 'cat' if op==0 else 'dog'
train_data.class_mode
get_class('../input/dogs-vs-cats/train/train/cat.10002.jpg')
```

16. Write a program to implement RNN

Code: -

```
from tensorflow.keras.datasets import imdb
(X_train,y_train),(X_test,y_test)=imdb.load_data(num_words=20000)
X_train.shape,X_test.shape
```

```
((25000,), (25000,))
```

```
len(X_train[0]),len(X_train[1]),len(X_train[2]),len(X_train[3]),len(X_train[4])
```

```
(218, 189, 141, 550, 147)
```

```
y_train[:5]
```

```
array([1, 0, 0, 1, 0])
```

```
X_train[0]
```

```
Out[6]: [1,
 14,
 22,
 16,
 43,
 530,
 973,
 1622,
 1385,
 65,
 458,
 4468,
 66,
 3941,
 4,
 173,
 36,
 256,
 5,
```

```
import numpy as np
np.array(X_train[0])
```

```
array([   1,   14,   22,   16,   43,  530,  973, 1622, 1385,
         65,  458, 4468,   66, 3941,    4,  173,   36,  256,
          5,   25,  100,   43,  838,  112,   50,  670,    2,
          9,   35,  480,  284,    5,  150,    4,  172,  112,
        167,    2,  336,  385,   39,    4,  172, 4536, 1111,
         17,  546,   38,   13,  447,    4,  192,   50,   16,
          6,  147, 2025,   19,   14,   22,    4, 1920, 4613,
        469,    4,   22,   71,   87,   12,   16,   43,  530,
         38,   76,   15,   13, 1247,    4,   22,   17,  515,
         17,   12,   16,  626,   18, 19193,    5,   62,  386,
         12,    8,  316,    8,  106,    5,    4, 2223, 5244,
         16,  480,   66, 3785,   33,    4,  130,   12,   16,
         38,  619,    5,   25,  124,   51,   36,  135,   48,
         25, 1415,   33,    6,   22,   12,  215,   28,   77,
         52,    5,   14,  407,   16,   82, 10311,    8,    4,
        107,  117, 5952,   15,  256,    4,    2,    7, 3766,
          5,  723,   36,   71,   43,  530,  476,   26,  400,
        317,   46,    7,    4, 12118, 1029,   13,  104,   88,
          4,  381,   15,  297,   98,   32, 2071,   56,   26,
        141,    6,  194, 7486,   18,    4,  226,   22,   21,
        134,  476,   26,  480,    5,  144,   30, 5535,   18,
         51,   36,   28,  224,   92,   25,  104,    4,  226,
         65,   16,   38, 1334,   88,   12,   16,  283,    5,
         16, 4472,  113,  103,   32,   15,   16, 5345,   19,
        178,   32])
```

```
from tensorflow.keras.preprocessing.sequence import pad_sequences
X=pad_sequences(X_train,maxlen=200)
X_val=pad_sequences(X_test,maxlen=200)
len(X[0])
```

```
200
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM,Dense,Embedding
model=Sequential()
model.add(Embedding(20000,128,input_shape=(200,)))
model.add(LSTM(100,return_sequences=True))
model.add(LSTM(100))
model.add(Dense(1,activation='sigmoid'))
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
model.fit(X,y_train,validation_data=(X_val,y_test),epochs=5,batch_size=64)
```

```
Epoch 1/5
391/391 [==============================] - 284s 719ms/step - loss: 0.3937 - accuracy: 0.8202 - val_loss: 0.3396 - val_accuracy:
0.8546
Epoch 2/5
391/391 [==============================] - 291s 744ms/step - loss: 0.2017 - accuracy: 0.9252 - val_loss: 0.3343 - val_accuracy:
0.8648
Epoch 3/5
391/391 [==============================] - 285s 730ms/step - loss: 0.1272 - accuracy: 0.9552 - val_loss: 0.3831 - val_accuracy:
0.8570
Epoch 4/5
391/391 [==============================] - 284s 726ms/step - loss: 0.1051 - accuracy: 0.9620 - val_loss: 0.4718 - val_accuracy:
0.8502
Epoch 5/5
391/391 [==============================] - 286s 732ms/step - loss: 0.0872 - accuracy: 0.9707 - val_loss: 0.4961 - val_accuracy:
0.8302

<tensorflow.python.keras.callbacks.History at 0x7fea6b32d790>
```

17. Write a program to implement GAN

Code: -

```
import os
print(os.listdir("../input"))

from __future__ import print_function
import time
import torch
import torch.nn as nn
import torch.nn.parallel
import torch.optim as optim
import torch.utils.data
import torchvision.datasets as dset
import torchvision.transforms as transforms
import torchvision.utils as vutils
from torch.autograd import Variable
import matplotlib.pyplot as plt
import numpy as np
from torch import nn, optim
import torch.nn.functional as F
from torchvision import datasets, transforms
from torchvision.utils import save_image
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from tqdm import tqdm_notebook as tqdm

PATH = '../input/all-dogs/all-dogs/'
images = os.listdir(PATH)
print(f'There are {len(os.listdir(PATH))} pictures of dogs.')

fig, axes = plt.subplots(nrows=3, ncols=3, figsize=(12,10))

for indx, axis in enumerate(axes.flatten()):
    rnd_indx = np.random.randint(0, len(os.listdir(PATH)))
    img = plt.imread(PATH + images[rnd_indx])
    imgplot = axis.imshow(img)
    axis.set_title(images[rnd_indx])
    axis.set_axis_off()
```

```
plt.tight_layout(rect=[0, 0.03, 1, 0.95])

batch_size = 32
image_size = 64

random_transforms = [transforms.ColorJitter(), transforms.RandomRotation(degrees=20)]
transform = transforms.Compose([transforms.Resize(64),
                                transforms.CenterCrop(64),
                                transforms.RandomHorizontalFlip(p=0.5),
                                transforms.RandomApply(random_transforms, p=0.2),
                                transforms.ToTensor(),
                                transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

train_data = datasets.ImageFolder('../input/all-dogs/', transform=transform)
train_loader = torch.utils.data.DataLoader(train_data, shuffle=True,
                                           batch_size=batch_size)

imgs, label = next(iter(train_loader))
imgs = imgs.numpy().transpose(0, 2, 3, 1)

for i in range(5):
    plt.imshow(imgs[i])
    plt.show()

def weights_init(m):
    """
    Takes as input a neural network m that will initialize all its weights.
    """
    classname = m.__class__.__name__
    if classname.find('Conv') != -1:
        m.weight.data.normal_(0.0, 0.02)
    elif classname.find('BatchNorm') != -1:
        m.weight.data.normal_(1.0, 0.02)
        m.bias.data.fill_(0)

class G(nn.Module):
    def __init__(self):
        super(G, self).__init__()
        self.main = nn.Sequential(
                nn.ConvTranspose2d(100, 512, 4, stride=1, padding=0, bias=False),
```

```
                nn.BatchNorm2d(512),
                nn.ReLU(True),
                nn.ConvTranspose2d(512, 256, 4, stride=2, padding=1, bias=False),
                nn.BatchNorm2d(256),
                nn.ReLU(True),
                nn.ConvTranspose2d(256, 128, 4, stride=2, padding=1, bias=False),
                nn.BatchNorm2d(128),
                nn.ReLU(True),
                nn.ConvTranspose2d(128, 64, 4, stride=2, padding=1, bias=False),
                nn.BatchNorm2d(64),
                nn.ReLU(True),
                nn.ConvTranspose2d(64, 3, 4, stride=2, padding=1, bias=False),
                nn.Tanh()
                )

    def forward(self, input):
        output = self.main(input)
        return output

netG = G()
netG.apply(weights_init)

class D(nn.Module):
    def __init__(self):
        super(D, self).__init__()
        self.main = nn.Sequential(
                nn.Conv2d(3, 64, 4, stride=2, padding=1, bias=False),
                nn.LeakyReLU(negative_slope=0.2, inplace=True),
                nn.Conv2d(64, 128, 4, stride=2, padding=1, bias=False),
                nn.BatchNorm2d(128),
                nn.LeakyReLU(negative_slope=0.2, inplace=True),
                nn.Conv2d(128, 256, 4, stride=2, padding=1, bias=False),
                nn.BatchNorm2d(256),
                nn.LeakyReLU(negative_slope=0.2, inplace=True),
                nn.Conv2d(256, 512, 4, stride=2, padding=1, bias=False),
                nn.BatchNorm2d(512),
                nn.LeakyReLU(negative_slope=0.2, inplace=True),
                nn.Conv2d(512, 1, 4, stride=1, padding=0, bias=False),
                nn.Sigmoid()
                )
```

```
    def forward(self, input):
        output = self.main(input)
        return output.view(-1)



netD = D()
netD.apply(weights_init)

class Generator(nn.Module):
    def __init__(self, nz=128, channels=3):
        super(Generator, self).__init__()

        self.nz = nz
        self.channels = channels

        def convlayer(n_input, n_output, k_size=4, stride=2, padding=0):
            block = [
                nn.ConvTranspose2d(n_input, n_output, kernel_size=k_size, stride=stride, padding=padding,
bias=False),
                nn.BatchNorm2d(n_output),
                nn.ReLU(inplace=True),
            ]
            return block

        self.model = nn.Sequential(
            *convlayer(self.nz, 1024, 4, 1, 0),
            *convlayer(1024, 512, 4, 2, 1),
            *convlayer(512, 256, 4, 2, 1),
            *convlayer(256, 128, 4, 2, 1),
            *convlayer(128, 64, 4, 2, 1),
            nn.ConvTranspose2d(64, self.channels, 3, 1, 1),
            nn.Tanh()
        )

    def forward(self, z):
        z = z.view(-1, self.nz, 1, 1)
        img = self.model(z)
        return img
```

```
class Discriminator(nn.Module):
    def __init__(self, channels=3):
        super(Discriminator, self).__init__()

        self.channels = channels

        def convlayer(n_input, n_output, k_size=4, stride=2, padding=0, bn=False):
            block = [nn.Conv2d(n_input, n_output, kernel_size=k_size, stride=stride, padding=padding,
bias=False)]
            if bn:
                block.append(nn.BatchNorm2d(n_output))
            block.append(nn.LeakyReLU(0.2, inplace=True))
            return block

        self.model = nn.Sequential(
            *convlayer(self.channels, 32, 4, 2, 1),
            *convlayer(32, 64, 4, 2, 1),
            *convlayer(64, 128, 4, 2, 1, bn=True),
            *convlayer(128, 256, 4, 2, 1, bn=True),
            nn.Conv2d(256, 1, 4, 1, 0, bias=False),
        )

    def forward(self, imgs):
        logits = self.model(imgs)
        out = torch.sigmoid(logits)

        return out.view(-1, 1)

!mkdir results
!ls

EPOCH = 0
LR = 0.001
criterion = nn.BCELoss()
optimizerD = optim.Adam(netD.parameters(), lr=LR, betas=(0.5, 0.999))
optimizerG = optim.Adam(netG.parameters(), lr=LR, betas=(0.5, 0.999))

for epoch in range(EPOCH):
    for i, data in enumerate(dataloader, 0):
```

```
        netD.zero_grad()

        real,_ = data
        input = Variable(real)
        target = Variable(torch.ones(input.size()[0]))
        output = netD(input)
        errD_real = criterion(output, target)

        noise = Variable(torch.randn(input.size()[0], 100, 1, 1))
        fake = netG(noise)
        target = Variable(torch.zeros(input.size()[0]))
        output = netD(fake.detach())
        errD_fake = criterion(output, target)

        errD = errD_real + errD_fake
        errD.backward()
        optimizerD.step()

        netG.zero_grad()
        target = Variable(torch.ones(input.size()[0]))
        output = netD(fake)
        errG = criterion(output, target)
        errG.backward()
        optimizerG.step()

        print('[%d/%d][%d/%d] Loss_D: %.4f; Loss_G: %.4f' % (epoch, EPOCH, i, len(dataloader), errD.item(),
errG.item()))
        if i % 100 == 0:
            vutils.save_image(real, '%s/real_samples.png' % "./results", normalize=True)
            fake = netG(noise)
            vutils.save_image(fake.data, '%s/fake_samples_epoch_%03d.png' % ("./results", epoch),
normalize=True)

batch_size = 32
LR_G = 0.001
LR_D = 0.0005

beta1 = 0.5
epochs = 100
```

```
real_label = 0.9
fake_label = 0
nz = 128

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

netG = Generator(nz).to(device)
netD = Discriminator().to(device)

criterion = nn.BCELoss()

optimizerD = optim.Adam(netD.parameters(), lr=LR_D, betas=(beta1, 0.999))
optimizerG = optim.Adam(netG.parameters(), lr=LR_G, betas=(beta1, 0.999))

fixed_noise = torch.randn(25, nz, 1, 1, device=device)

G_losses = []
D_losses = []
epoch_time = []

def plot_loss (G_losses, D_losses, epoch):
    plt.figure(figsize=(10,5))
    plt.title("Generator and Discriminator Loss - EPOCH "+ str(epoch))
    plt.plot(G_losses,label="G")
    plt.plot(D_losses,label="D")
    plt.xlabel("iterations")
    plt.ylabel("Loss")
    plt.legend()
    plt.show()

def show_generated_img(n_images=5):
    sample = []
    for _ in range(n_images):
        noise = torch.randn(1, nz, 1, 1, device=device)
        gen_image = netG(noise).to("cpu").clone().detach().squeeze(0)
        gen_image = gen_image.numpy().transpose(1, 2, 0)
        sample.append(gen_image)

    figure, axes = plt.subplots(1, len(sample), figsize = (64,64))
    for index, axis in enumerate(axes):
```

```
        axis.axis('off')
        image_array = sample[index]
        axis.imshow(image_array)

    plt.show()
    plt.close()

for epoch in range(epochs):

    start = time.time()
    for ii, (real_images, train_labels) in tqdm(enumerate(train_loader), total=len(train_loader)):
        netD.zero_grad()
        real_images = real_images.to(device)
        batch_size = real_images.size(0)
        labels = torch.full((batch_size, 1), real_label, device=device)

        output = netD(real_images)
        errD_real = criterion(output, labels)
        errD_real.backward()
        D_x = output.mean().item()

        noise = torch.randn(batch_size, nz, 1, 1, device=device)
        fake = netG(noise)
        labels.fill_(fake_label)
        output = netD(fake.detach())
        errD_fake = criterion(output, labels)
        errD_fake.backward()
        D_G_z1 = output.mean().item()
        errD = errD_real + errD_fake
        optimizerD.step()

        netG.zero_grad()
        labels.fill_(real_label)
        output = netD(fake)
        errG = criterion(output, labels)
        errG.backward()
        D_G_z2 = output.mean().item()
        optimizerG.step()

        G_losses.append(errG.item())
```

```
        D_losses.append(errD.item())

        if (ii+1) % (len(train_loader)//2) == 0:
            print('[%d/%d][%d/%d] Loss_D: %.4f Loss_G: %.4f D(x): %.4f D(G(z)): %.4f / %.4f'
                  % (epoch + 1, epochs, ii+1, len(train_loader),
                     errD.item(), errG.item(), D_x, D_G_z1, D_G_z2))

    plot_loss (G_losses, D_losses, epoch)
    G_losses = []
    D_losses = []
    if epoch % 10 == 0:
        show_generated_img()

    epoch_time.append(time.time()- start)



print (">> average EPOCH duration = ", np.mean(epoch_time))

show_generated_img(7)

 if not os.path.exists('../output_images'):
    os.mkdir('../output_images')

im_batch_size = 50
n_images=10000

for i_batch in tqdm(range(0, n_images, im_batch_size)):
    gen_z = torch.randn(im_batch_size, nz, 1, 1, device=device)
    gen_images = netG(gen_z)
    images = gen_images.to("cpu").clone().detach()
    images = images.numpy().transpose(0, 2, 3, 1)
    for i_image in range(gen_images.size(0)):
        save_image(gen_images[i_image, :, :, :], os.path.join('../output_images',
f'image_{i_batch+i_image:05d}.png'))

fig = plt.figure(figsize=(25, 16))
for i, j in enumerate(images[:32]):
    ax = fig.add_subplot(4, 8, i + 1, xticks=[], yticks=[])
    plt.imshow(j)
```

```
import shutil
shutil.make_archive('images', 'zip', '../output_images')

torch.save(netG.state_dict(), 'generator.pth')
torch.save(netD.state_dict(), 'discriminator.pth')
```

18. Web scraping experiments (by using tools)

Code: -

```
import requests
from bs4 import BeautifulSoup
import csv

URL = "http://www.values.com/inspirational-quotes"
r = requests.get(URL)

soup = BeautifulSoup(r.content, 'html5lib')

quotes=[]

soup.find('div', attrs = {'id':'all_quotes'})
```

```
<div class="row" id="all_quotes">
            <div class="col-6 col-lg-3 text-center margin-30px-bottom sm-margin-30px-top">

        <a href="/inspirational-quotes/6377-at-211-degrees-water-is-hot-at-212-degrees"><img alt="At 211 degrees, water is ho
t. At 212 degrees, it boils. And with boiling water, comes steam. And with steam, you can power a train.  #&lt;Author:0x00007
f1889af8068&gt;" class="margin-10px-bottom shadow" height="310" src="https://assets.passiton.com/quotes/quote_artwork/6377/me
dium/20220204_friday_quote.jpg?1643401767" width="310"/></a>
        <h5 class="value_on_red"><a href="/inspirational-quotes/6377-at-211-degrees-water-is-hot-at-212-degrees">PERSISTENCE
</a></h5>

</div><div class="col-6 col-lg-3 text-center margin-30px-bottom sm-margin-30px-top">

        <a href="/inspirational-quotes/8301-the-key-of-persistence-opens-all-doors-closed"><img alt="The key of persistence o
pens all doors closed by resistance. #&lt;Author:0x00007f1889b1d318&gt;" class="margin-10px-bottom shadow" height="310" src
="https://assets.passiton.com/quotes/quote_artwork/8301/medium/20220203_thursday_quote.jpg?1643401731" width="310"/></a>
        <h5 class="value_on_red"><a href="/inspirational-quotes/8301-the-key-of-persistence-opens-all-doors-closed">PERSISTEN
CE</a></h5>

</div><div class="col-6 col-lg-3 text-center margin-30px-bottom sm-margin-30px-top">
```

```
for row in table.find_all_next('div', attrs = {'class': 'col-6 col-lg-3 text-center margin-30px-bottom sm-margin-30px-top'}):
    quote = {}
    quote['theme'] = row.h5.text
```

```
quote['url'] = row.a['href']
quote['img'] = row.img['src']
quote['lines'] = row.img['alt'].split(" #")[0]
quote['author'] = row.img['alt'].split(" #")[1]
quotes.append(quote)
```