ATSS's

Institute of Industrial and Computer Management and Research, Nigdi Pune

MCA Department

Academic Year: 2022-23

Practical Journal on

IT31L- Knowledge Representation and Artificial Intelligence: ML, DL (SEM-III)

Submitted By:

Roll no : 42
Name : Sakshi Santosh Pharande
Seat No.:
Date:

Course Outcome:

Student will be able to:

CO2: Develop ML, DL models using Python (Apply)

ATSS's

Institute of Industrial and Computer Management and Research, Nigdi Pune

MCA Department

INDEX

Students Name :____Roll No. _________

Sr. No	Program Title	Course Outcome	Page No.	Teacher's Sign with Date	Remarks
1.	Write a Program to implement the correlation matrix	CO2			
2.	Write a program to plot the correlation plot on dataset and visualize giving an overview of relationships among data on iris data.	CO2			
3.	Write a program to apply linear regression Model techniques to predict the data (use any of the dataset)	CO2			
4.	Write a program to apply logical regression Model techniques to predict the data on any dataset.	CO2			
5.	Write a program to use Clustering algorithms for unsupervised classification.	CO2			
6.	Write a program to use Association algorithms for supervised classification on any dataset.	CO2			
7.	Write a program for Developing and implementing Decision Tree model on the dataset.	CO2			

8.	Write a program for Bayesian classification on any dataset	CO2			
9.	Write a program for SVM classification on any dataset.	CO2			
10.	Write a program to Plot the cluster data using python visualizations.	CO2			
11.	Write a program for Creating & Visualizing Neural Network for the given data. (Use python).	CO2			
12.	Write a program for Recognize optical character using ANN.	CO2			
13.	Write a program to implement CNN	CO2			
14.	Write a program to implement RNN	CO2			
15.	Write a program to implement GAN	CO2			

Q.1 Write a Program to implement the correlation matrix

Program –

```
import numpy as np
x = np.array([3,5,11,21,28,35,56,61,72,88]) y =
np.array([11,15,20,33,48,51,71,89,91,100]) z =
np.array([104,100,89,81,76,66,69,43,17,11])
type(x) import matplotlib.pyplot as plt
plt.xlabel('Xvalues') plt.ylabel('Yvalues')
plt.scatter(x, y) plt.xlabel('Xvalues')
plt.ylabel('Zvalues') plt.scatter(x, z)
plt.xlabel('Xvalues') plt.ylabel('Yvalues')
plt.scatter(x, y) plt.scatter(x, z, color = 'r')
np.corrcoef(x, y) np.corrcoef(x, z)
np.corrcoef(z, y) import scipy.stats as st
st.pearsonr(x, y)[0] st.pearsonr(x, z)[0]
st.pearsonr(z, y)[0] import pandas as pd
x1.corr(y1) y1.corr(z1)
df =
   pd.DataFra
   me({ 'x': x,
   'y': y,
   'z': z })
df.corr()
df.corrwith(x1
)
st.spearmanr(x
, y)[0]
st.spearmanr(x
, z)[0] st.spearmanr(z,
y)[0]
df.corr(method='sper
man') st.kendalltau(x,
y)[0] st.kendalltau(x,
z)[0] st.kendalltau(z,
y)[0]
df.corr(method='kendall')
df.corrwith(x1,
method='kendall') cor =
df.corr(method='kendall')
cor.values
```

Output:

```
array([[1.        , 0.98757408],
       [0.98757408, 1.        ]])
```

```
np.corrcoef(x, z)
```

```
array([[ 1.        , -0.96149075],
       [-0.96149075,  1.        ]])
```

```
np.corrcoef(z, y)
```

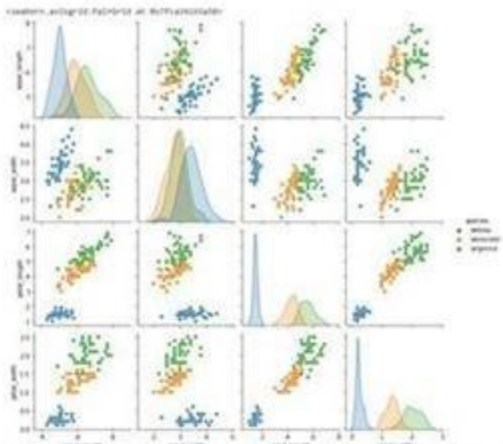
```
array([[ 1.        , -0.95002927],
       [-0.95002927,  1.        ]])
```

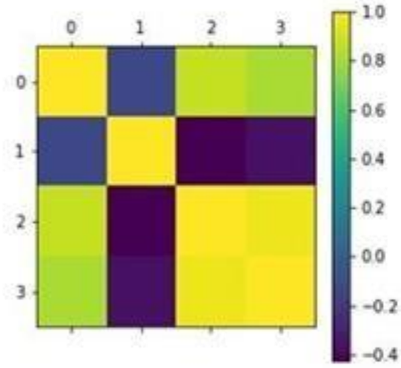
Q.2 Write a program to plot the correlation plot on dataset and visualize giving an overview of relationships among data on iris data.

Program – import
pandas as pd
x = ['slength','swidth','plength','pwidth','species']
df=pd.read_csv('https://archive.ics.uci.edu/ml/machine- learning-databases/iris/iris.data', names=x) df df = pd.read_csv('iris.csv') import seaborn as sns sns.get_dataset_names() iris = sns.load_dataset('iris') sns.pairplot(df, hue='Species')
import matplotlib.pyplot as plt
df.corr()
plt.figure(figsize=(16,9))
plt.matshow(df.corr())
plt.colorbar()
sns.heatmap(df.corr(), annot=True)

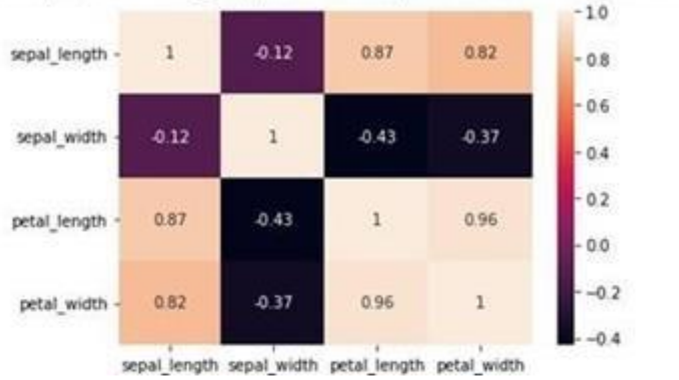
Output –

	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





<matplotlib.axes._subplots.AxesSubplot at 0x7fca2911afd0>



Q.3 Write a program to apply linear regression Model techniques to predict the data (use any of the dataset)

Program –

```
import pandas as pd  import
os os.getcwd()
df =
pd.read_csv('/content/sample_data/Salary_Data.csv')
df.shape  df.columns
x = df['YearsExperience'].values
y = df['Salary'].values  df.corr() import matplotlib.pyplot as plt
  plt.xlabel('Experience')  plt.ylabel('Salary')
plt.scatter(x, y)
from              sklearn.linear_model              import
LinearRegression   regressor  =  LinearRegression()
x    =    x.reshape(-1,1)       x    regressor.fit(x,    y)
regressor.predict([[5]])                    y_pred          =
regressor.predict(x)  import seaborn as sns
sns.regplot(x='YearsExperience', y='Salary', data=df)  result
= pd.DataFrame({
'Actual': y,
 'Predicted': y_pred
})
result
plt.xlabel('Experience')
plt.ylabel('Salary')  plt.grid()
plt.scatter(x, y, label = 'Actual')
plt.plot(x, y_pred, label = 'Predicted',
color='g')  plt.legend()  regressor.coef_
regressor.intercept_
5 * 9449.96232146 + 25792.200198668696

regressor.score(x, y)
from sklearn.metrics import r2_score, mean_absolute_error, mean_absolute_percentage_error
r2_score(y,  y_pred)    mean_absolute_error(y,  y_pred)    mean_absolute_percentage_error(y,
y_pred)  df = pd.read_csv('/content/sample_data/mtcars.csv')  df.shape
x = df[['disp','hp','wt']]
y = df['mpg']
regressor = LinearRegression()
```

regressor.fit(x, y)
regressor.intercept_
regressor.coef_
regressor.score(x, y) new
= [[221, 102, 3.81]]
regressor.predict(new) new = [[211, 134, 2.81]]
regressor.predict(new) x.corrwith(y)
y_pred = regressor.predict(x)
r2_score(y, y_pred)
plt.subplot(2,2,1)
plt.scatter(x['disp'], y) plt.subplot(2,2,2) plt.scatter(x['hp'], y) plt.subplot(2,2,3) plt.scatter(x['wt'], y)
sns.regplot(x='wt', y='mpg', data=pd.read_csv('/content/sample_data/mtcars.csv'))

Output –

```
[14] # Train the algorithm with data
     regressor.fit(x, y)

     LinearRegression()
```

```
[15] # Prediction
     regressor.predict([[5]])

     array([73042.01180594])
```

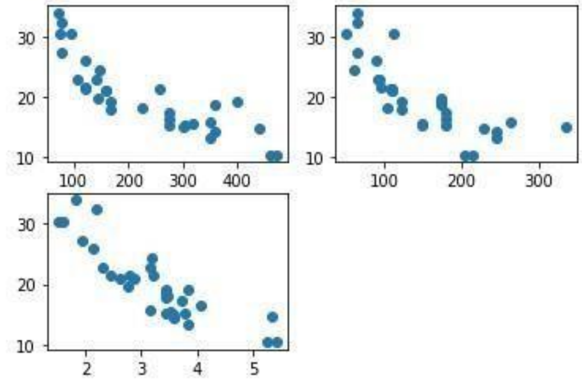
```
# prediction
new = [[221, 102, 3.81]]
regressor.predict(new)

/usr/local/lib/python3.7/dist-packages/sklearn/base.py:451: UserWarning: X does not have valid feature names, but LinearRegression was fitted with feature names
  "X does not have valid feature names, but"
array([19.23906496])
```

result

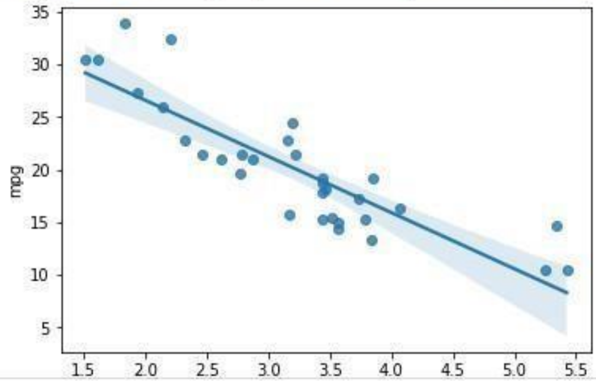
	Actual	Predicted
0	39343	36187.158752
1	46205	38077.151217
2	37731	39967.143681
3	43525	44692.124842
4	39891	46582.117306
5	56642	53197.090931
6	60150	54142.087163
7	54445	56032.079627
8	64445	56032.079627
9	57189	60757.060788
10	63218	62647.053252
11	55794	63592.049484
12	56957	63592.049484
13	57081	64537.045717
14	61111	68317.030645
15	67938	72097.015574
16	66029	73987.008038
17	83088	75877.000502
18	81363	81546.977895
19	93940	82491.974127
20	91738	90051.943985
21	98273	92886.932681
22	101302	100446.902538
23	113812	103281.891235
24	109431	108006.872395
25	105582	110841.861092
26	116969	115566.842252
27	112635	116511.838485
28	122391	123126.812110
29	121872	125016.804574

```
<matplotlib.collections.PathCollection at 0x7f97b54d3750>
```



```
sns.regplot(x='wt', y='mpg', data=pd.read_csv('/content/sample_data/mtcars.csv'))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f97b7568d50>
```



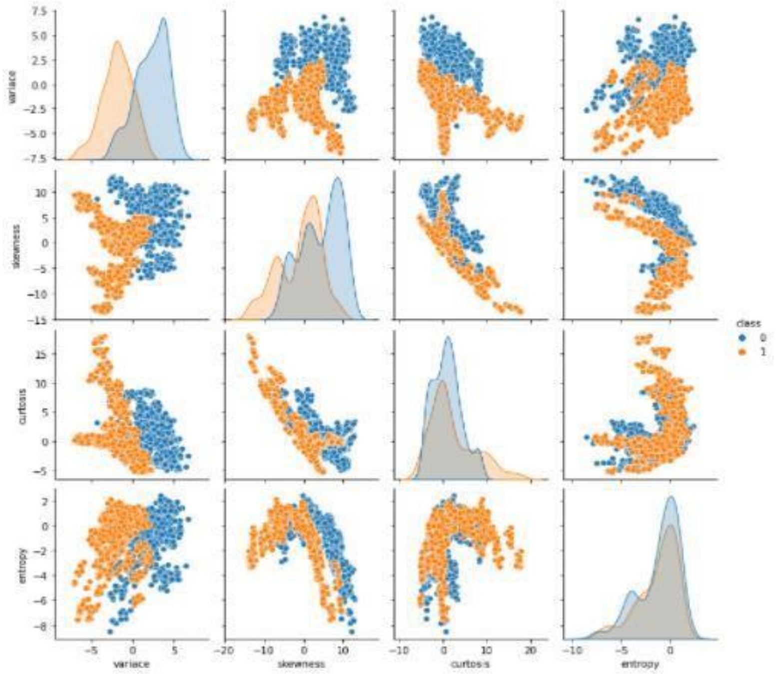
Q.4 Write a program to apply logical regression Model techniques to predict the data on any dataset.

Program –

```
import pandas as pd
df = pd.read_csv('banknotes.csv')
import seaborn as sns  sns.pairplot(df, hue='class')  x
= df.drop('class', axis = 1)   y = df['class']   x.shape
from sklearn.model_selection import
train_test_split   x_train, x_test, y_train, y_test =
train_test_split(
x, y, random_state=0,
test_size=0.25)  x_train.head()
x_train.shape
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression()  classifier.fit(x_train,
y_train)
x_test.shape
y_pred = classifier.predict(x_test)  set(y)
y.value_counts()   result
= pd.DataFrame({
'Actual': y_test,
'Predicted': y_pred
})
Result
from sklearn.metrics import plot_confusion_matrix, accuracy_score
plot_confusion_matrix(classifier, x_test, y_test);
y_test.value_counts()  accuracy_score(y_test, y_pred)   new1 =
[[0.7057,-5.4981,8.3368,-2.8715]]   new2 = [[-0.4665,2.3383,-
2.9812,-1.0431]]
classifier.predict(new1)
classifier.predict_proba(new1)
classifier.predict(new2)
classifier.predict_proba(new2)
```

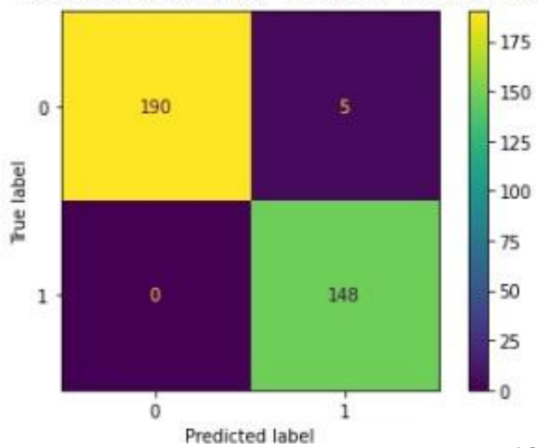
Output –

```
Out[6]: <seaborn.axisgrid.PairGrid at 0x214021750d0>
```



	Actual	Predicted
1023	1	1
642	0	0
1196	1	1
31	0	0
253	0	0
...	...	...
866	1	1
361	0	0
703	0	0
328	0	0
530	0	0

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils
  warnings.warn(msg, category=FutureWarning)
```



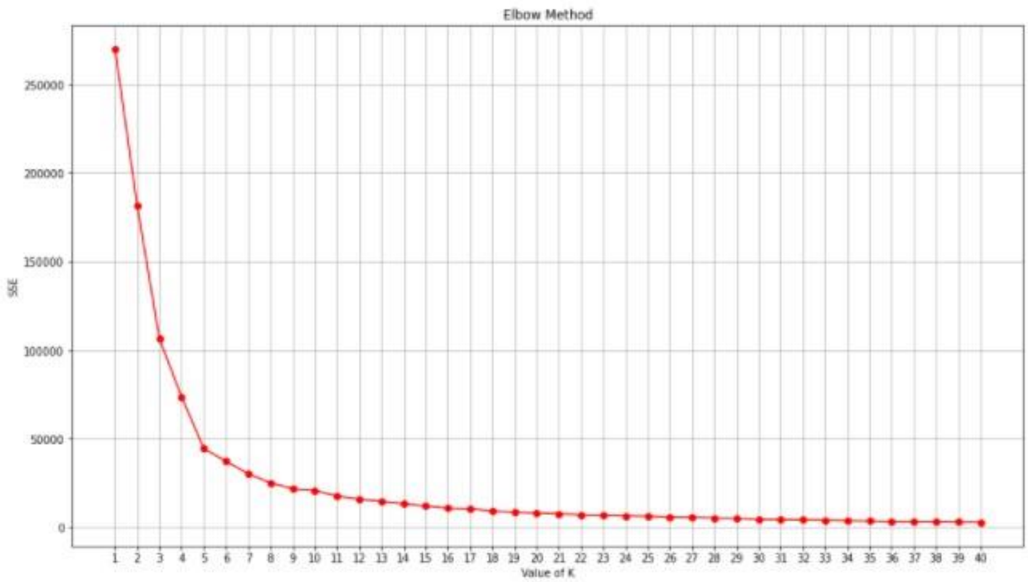
Q.5 Write a program to use Clustering algorithm for unsupervised classification.

Program –

```
Import pandas as pd
df = pd.read_csv('Mall_Customers.csv')
df.shape list(df.columns) x =
df.iloc[:,3:] x df.describe() import
seaborn as sns sns.kdeplot(df['Age'])
sns.kdeplot(df['Annual Income (k$)'])
sns.kdeplot(df['Spending Score (1-
100)']) sns.boxplot(df['Age'])
sns.boxplot(df['Annual Income (k$)'])
sns.boxplot(df['Spending Score (1-
100)'])
# Import the class
from sklearn.cluster import KMeans
# Create the object
km = KMeans(n_clusters=12, random_state=0)
# Train the algorithm
labels =
km.fit_predict(x) # Sum
of squared errors
km.inertia_ # elbow
method sse = [] for k in
range(1,41):
km = KMeans(n_clusters=k,
random_state=0) labels = km.fit_predict(x)
sse.append(km.inertia_) import matplotlib.pyplot as
plt plt.figure(figsize=(16,9)) plt.title('Elbow
Method') plt.xlabel('Value of K') plt.ylabel('SSE')
plt.grid()
plt.xticks(range(1,41))
plt.plot(range(1,41), sse, marker='o', color='r')
# Silhoutte method
from sklearn.metrics import silhouette_score
silh = [] for k in
range(2,16):
km = KMeans(n_clusters=k, random_state=0)
```

labels = km.fit_predict(x) score =
silhouette_score(x, labels)
silh.append(score) # plot the
silhoutte scores plt.title('Silhoutte
Analysis') plt.xlabel('Value of K')
plt.ylabel('Silhoutte Score')
plt.xticks(range(2,16))
plt.bar(range(2,16), silh,
color='g') # Create the object
km = KMeans(n_clusters=5, random_state=0)
# Train the algorithm labels
= km.fit_predict(x)
labels # Cluster labels km.labels_ #
SSE km.inertia_ # Centroids
km.cluster_centers_ # Extract the
clusters df[labels==2] # Boolean
filtering one = df[labels==1]
one.shape # Export the cluster
one.to_csv('one.csv') print('Cluster-
0:', len(df[labels==0]))
print('Cluster-1:',
len(df[labels==1])) print('Cluster-
2:', len(df[labels==2]))
print('Cluster-3:',
len(df[labels==3])) print('Cluster-
4:', len(df[labels==4]))
Prediction new =
[[45, 76]]
km.predict(new)[0
] # Prediction new
= [[25, 36]]
km.predict(new)[0
] # Prediction
new = [[85, 76]]
km.predict(new)[0
] # Prediction new
= [[45, 47]]
km.predict(new)[
0]

Output –




```
In [25]: # Train the algorithm
         labels = km.fit_predict(x)
```

```
In [26]: labels

Out[26]: array([4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3,
                4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 1,
                4, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 0, 2, 1, 2, 0, 2, 0, 2,
                1, 2, 0, 2, 0, 2, 0, 2, 0, 2, 1, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2,
                0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2,
                0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2,
                0, 2])
```

```
In [27]: # Cluster labels
         km.labels_

Out[27]: array([4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3,
                4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 1,
                4, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 0, 2, 1, 2, 0, 2, 0, 2,
                1, 2, 0, 2, 0, 2, 0, 2, 0, 2, 1, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2,
                0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2,
                0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2,
                0, 2])
```

In [33]:
```
# Export the cluster
one.to_csv('one.csv')
```

In [34]:
```
print('Cluster-0:', len(df[labels==0]))
print('Cluster-1:', len(df[labels==1]))
print('Cluster-2:', len(df[labels==2]))
print('Cluster-3:', len(df[labels==3]))
print('Cluster-4:', len(df[labels==4]))
```

```
Cluster-0: 35
Cluster-1: 81
Cluster-2: 39
Cluster-3: 22
Cluster-4: 23
```

In [30]:
```
# Extract the clusters
df[labels==2]  # Boolean filtering
```

Out[30]:

	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
123	124	Male	39	69	91
125	126	Female	31	70	77
127	128	Male	40	71	95
129	130	Male	38	71	75
131	132	Male	39	71	75
133	134	Female	31	72	71
135	136	Female	29	73	88
137	138	Male	32	73	73
139	140	Female	35	74	72
141	142	Male	32	75	93
143	144	Female	32	76	87
145	146	Male	28	77	97
147	148	Female	32	77	74
149	150	Male	34	78	90
151	152	Male	39	78	88
153	154	Female	38	78	76
155	156	Female	27	78	89
157	158	Female	30	78	78
159	160	Female	30	78	73
161	162	Female	29	79	83
163	164	Female	31	81	93
165	166	Female	36	85	75
167	168	Female	33	86	95
169	170	Male	32	87	63
171	172	Male	28	87	75
173	174	Male	36	87	92
175	176	Female	30	88	86
177	178	Male	27	88	69
179	180	Male	35	93	90
181	182	Female	32	97	86

Q.6 Write a program to use Association algorithms for supervised classification on any dataset.

Program –

```
dataset = [['Apple', 'Beer', 'Rice', 'Chicken'],
['Apple', 'Beer', 'Rice'],
['Apple', 'Beer'],
['Apple', 'Pear'],
['Milk', 'Beer', 'Rice', 'Chicken'],
['Milk', 'Beer', 'Rice'],
['Milk', 'Beer'],
['Apple', 'Pear']]
# Import the transaction encoder
from mlxtend.preprocessing import TransactionEncoder # Create the object
trans = TransactionEncoder() # Apply the operation df_t =
trans.fit_transform(dataset) trans.columns_ import pandas as pd
# Create a structured dataframe
df = pd.DataFrame(df_t, columns=trans.columns_)
# Support count
sum(df['Rice']) / len(df) #
Generate frequent itemsets
from mlxtend.frequent_patterns import apriori
freq_itemset = apriori(df, min_support=0.25, use_colnames=True) freq_itemset
# Generate strong association rules
from mlxtend.frequent_patterns import association_rules
rules = association_rules(freq_itemset, metric='confidence', min_threshold=0.5)
rules rules = rules[['antecedents','consequents','support','confidence']]
rules['antecedent_len'] = rules['antecedents'].apply(lambda x: len(x)) nrules =
rules[(rules['antecedent_len'] == 1) & (rules['support'] > 0.30)] nrules
# Prediction / Suggestion / Recommendation
nrules[nrules['antecedents'] == {'Apple'}]['consequents'][1] rules.sort_values(by='confidence',
ascending=False)
# Export the rules
rules.to_csv('rules.csv', index=False)
```

Output –

Out[31]:

	antecedents	consequents	support	confidence	antecedent_len
13	(Apple, Rice)	(Beer)	0.250	1.000000	2
17	(Chicken, Beer)	(Rice)	0.250	1.000000	2
2	(Pear)	(Apple)	0.250	1.000000	1
4	(Chicken)	(Beer)	0.250	1.000000	1
24	(Milk, Rice)	(Beer)	0.250	1.000000	2
6	(Milk)	(Beer)	0.375	1.000000	1
8	(Rice)	(Beer)	0.500	1.000000	1
9	(Chicken)	(Rice)	0.250	1.000000	1
20	(Chicken)	(Beer, Rice)	0.250	1.000000	1
18	(Chicken, Rice)	(Beer)	0.250	1.000000	2
25	(Milk)	(Beer, Rice)	0.250	0.666667	1
7	(Beer)	(Rice)	0.500	0.666667	1
22	(Beer, Milk)	(Rice)	0.250	0.666667	2
11	(Milk)	(Rice)	0.250	0.666667	1
14	(Apple, Beer)	(Rice)	0.250	0.666667	2
0	(Apple)	(Beer)	0.375	0.600000	1
16	(Rice)	(Apple, Beer)	0.250	0.500000	1
15	(Beer, Rice)	(Apple)	0.250	0.500000	2
1	(Beer)	(Apple)	0.375	0.500000	1
19	(Beer, Rice)	(Chicken)	0.250	0.500000	2
12	(Rice)	(Milk)	0.250	0.500000	1
21	(Rice)	(Chicken, Beer)	0.250	0.500000	1
10	(Rice)	(Chicken)	0.250	0.500000	1
23	(Beer, Rice)	(Milk)	0.250	0.500000	2
5	(Beer)	(Milk)	0.375	0.500000	1

Q.7 Write a program for Developing and implementing Decision Tree model on the dataset.

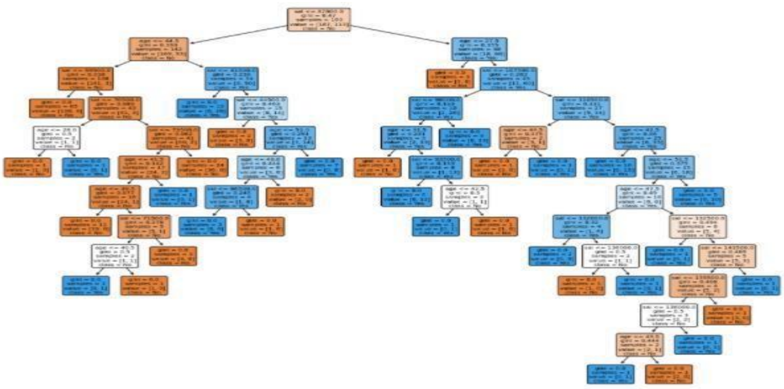
Program –

```
import pandas as pd  #
Data import
df = pd.read_csv('Social_Network_Ads.csv')
df.shape  # input
x                                        =
df[['Age','EstimatedSalary']]   y
=    df['Purchased']         import
seaborn as sns
sns.jointplot(x='Age', y='EstimatedSalary', hue='Purchased', data=df)
sns.countplot(x=y)  y.value_counts()  # Cross-validation
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(
x,            y,            random_state=0,
test_size=0.25)             x_train.shape
x_test.shape  # Import the class
from sklearn.ensemble import RandomForestClassifier
# Create the object
classifier = RandomForestClassifier(random_state=0, n_estimators=10)
# Train the algorithm with data  classifier.fit(x_train,
y_train)
# Predictions
y_pred = classifier.predict(x_test)
# Combine the data  result
= pd.DataFrame({
'Actual': y_test,
'Predicted': y_pred
})
Result
from sklearn.metrics import plot_confusion_matrix, accuracy_score
plot_confusion_matrix(classifier, x_test, y_test);  accuracy_score(y_test,
y_pred)
# Single prediction  new1
= [[34, 123000]]  new2 =
[[25,                  48900]]
```

```
classifier.predict(new1)
classifier.predict(new2)
from sklearn.tree import
plot_tree import
matplotlib.pyplot as plt
classifier.estimators_[0]
plt.figure(figsize=(16,12)
)
plot_tree(classifier.estimators_[8], fontsize=7, feature_names=['age','sal'],
class_names=['No','Yes'], filled=True, rounded=True); classifier.feature_importances_
```

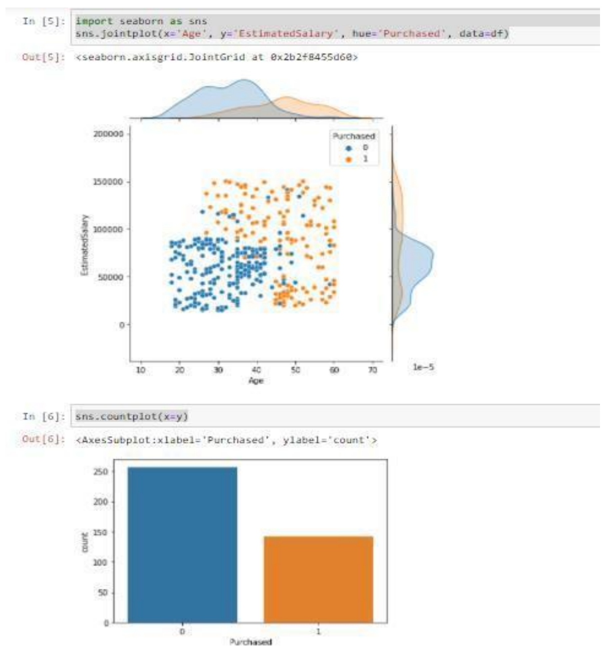
Output –

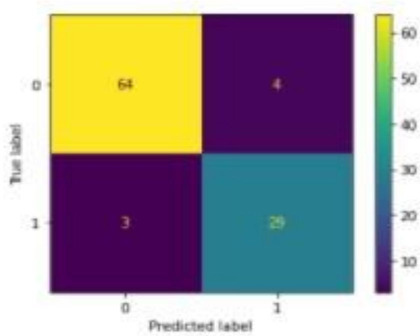


Out[16]:

	Actual	Predicted
132	0	0
309	0	0
341	0	0
196	0	0
246	0	0
...	...	...
146	1	0
135	0	0
390	1	1
264	1	1
364	1	1

100 rows × 2 columns





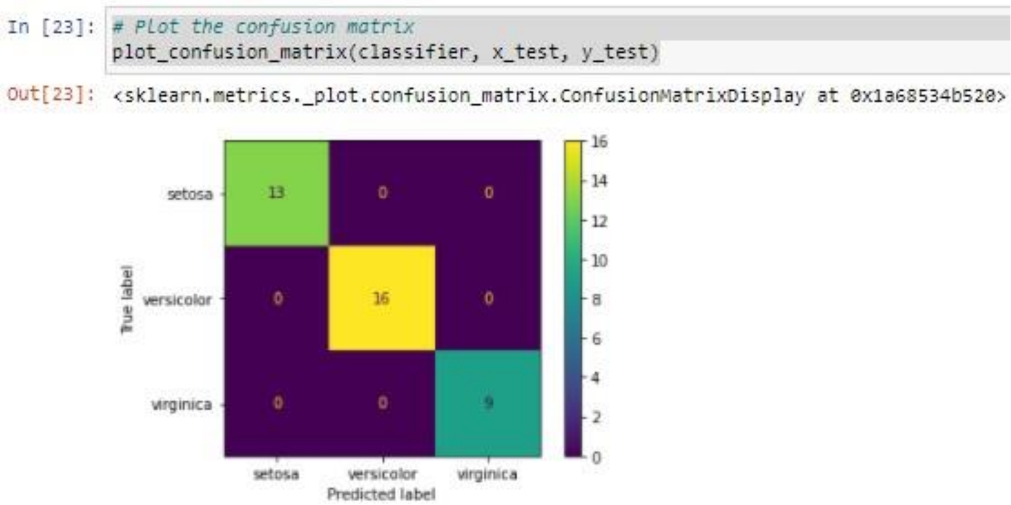
Q.8 Write a program for Bayesian classification on any dataset

Program –

```
# Import packages
import pandas as pd
import seaborn as sns
# Data import
df = pd.read_csv('iris (1).csv')
# The data shape df.shape
# The columns names
list(df.columns) # Let's
describe df.describe() #
Check the clusters
sns.pairplot(df,
hue='species')
# input data
x = df.drop('species', axis = 1)
# output data y =
df['species']
x.shape
sns.countplot(x = y)
y.value_counts()
# Cross validation -> hold out method from
sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split( x,
y, random_state=0, train_size=0.75) x_train.shape
x_test.shape # Import the class
from sklearn.naive_bayes import GaussianNB
# Create the object classifier =
GaussianNB() # Train the
algorithm with dataset
classifier.fit(x_train, y_train)
# Predictions
y_pred = classifier.predict(x_test)
# Import all functions
from sklearn.metrics import plot_confusion_matrix, accuracy_score from
sklearn.metrics import classification_report
# Plot the confusion matrix
```

```
plot_confusion_matrix(classifier, x_test, y_test)
# Accuracy
accuracy_score(y_test, y_pred)  #
Classification report
print(classification_report(y_test, y_pred))
# Print the probabilities
classifier.predict_proba(x_test
)  new1 = [[5.1,3.7,1.5,0.4]]
new2 = [[6.8,2.8,4.8,1.4]]
new3 = [[7.7,2.6,6.9,2.3]]  #
Predictions
classifier.predict(new1)[0]
classifier.predict(new2)[0]
classifier.predict(new3)[0]
```

Output –



Q.9 Write a program for SVM classification on any dataset.

Program –

```
import pandas as pd  #
Data import
df = pd.read_csv('banknotes.csv')
import seaborn as sns  sns.pairplot(df,
hue='class')
# Input data
x = df.drop('class', axis = 1)
# Output data
y = df['class']
x.shape
# Cross - validation -> hold out method  from
sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split( x,
y, random_state=0, test_size=0.25)  x_train.shape
x_test.shape  x_train  sns.countplot(x=y)
y.value_counts()  y_train.value_counts()
y_test.value_counts()  # Import the SVM class  from
sklearn.svm import SVC  # Create the object of SVC
classifier = SVC(random_state=0, kernel='sigmoid')
# Train the algorithm  classifier.fit(x_train,
y_train)
# Predictions
y_pred = classifier.predict(x_test)
from sklearn.metrics import plot_confusion_matrix, classification_report
from sklearn.metrics import accuracy_score
plot_confusion_matrix(classifier, x_test, y_test)
print(classification_report(y_test, y_pred))  accuracy_score(y_test,
y_pred)
new1 = [[3.73210,-3.884000,3.357700,-0.006049]]
classifier.predict(new1)
```

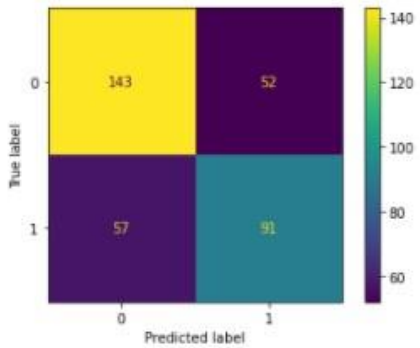
Output –

```
In [13]: x_train
Out[13]:
```

	variace	skewness	curtosis	entropy
662	2.97360	8.794400	-3.635900	-1.375400
512	2.66480	10.754000	-3.399400	-4.168500
1193	-3.75730	-8.291600	10.303200	0.380590
682	3.73210	-3.884000	3.357700	-0.006049
1313	-1.50780	-7.319100	7.898100	1.228900
...	...	...	...	...
763	0.39012	-0.142790	-0.031994	0.350840
835	-0.94255	0.039307	-0.241920	0.315930
1216	0.60050	0.999450	-2.212600	0.097399
559	2.01650	-0.252460	5.170700	1.076300
684	-2.07590	10.822300	2.643900	-4.837000

1029 rows × 4 columns

```
In [25]: plot_confusion_matrix(classifier, x_test, y_test)
Out[25]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x23c9689fe50>
```



`

```
In [26]: print(classification_report(y_test, y_pred))
              precision    recall  f1-score   support

           0       0.71      0.73      0.72       195
           1       0.64      0.61      0.63       148

    accuracy                           0.68       343
   macro avg       0.68      0.67      0.67       343
weighted avg       0.68      0.68      0.68       343
```

Q .10 Write a program to Plot the cluster data using python visualizations.

Program –

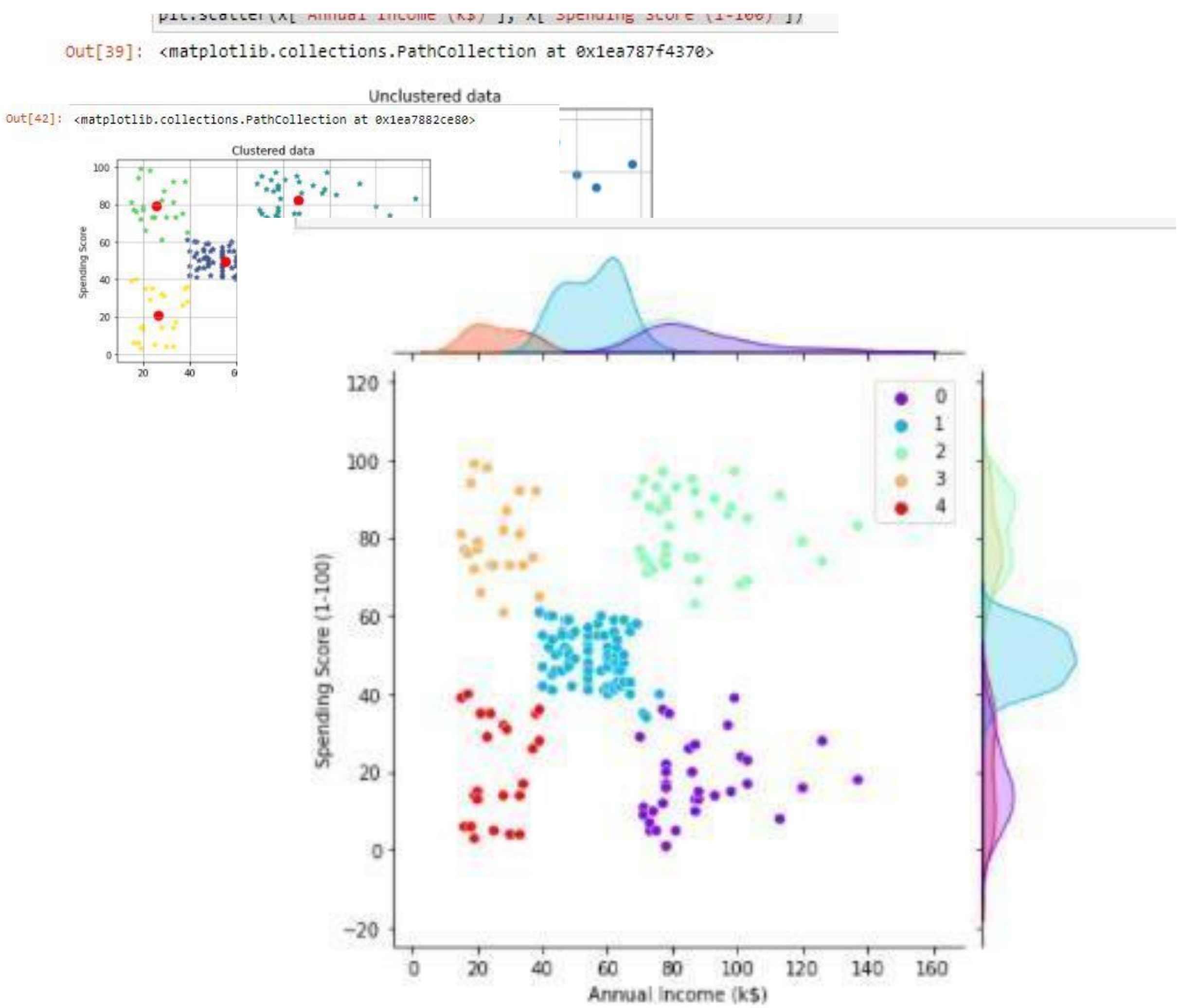
```
#  Import  packages
import pandas as pd
# Import the dataset
df = pd.read_csv('Mall_Customers.csv')
df.shape
list(df.columns
) # Input data  x
= df.iloc[:,3:]  x
# Summerize  df.describe()
# import seaborn package  import
seaborn as sns  sns.kdeplot(df['Age'])
sns.kdeplot(df['Annual Income (k$)'])
sns.kdeplot(df['Spending Score (1-
100)'])  sns.boxplot(df['Age'])
sns.boxplot(df['Annual Income (k$)'])  sns.boxplot(df['Spending
Score (1-100)'])
# Import the class
from sklearn.cluster import KMeans
# Create the object
km = KMeans(n_clusters=12, random_state=0)
#   Train   the   algorithm
labels                    =
km.fit_predict(x)  # Sum
of      squared      errors
km.inertia_      #   elbow
method  sse = []  for k in
range(1,41):
km = KMeans(n_clusters=k, random_state=0)
labels    =    km.fit_predict(x)
sse.append(km.inertia_)
import matplotlib.pyplot as plt
plt.figure(figsize=(16,9))
plt.title('Elbow         Method')
plt.xlabel('Value      of      K')
plt.ylabel('SSE')
```

```
plt.grid()
plt.xticks(range(1,41))
plt.plot(range(1,41), sse, marker='o', color='r')
# Silhoutte method
from sklearn.metrics import silhouette_score
silh = []   for  k  in
range(2,16):
km = KMeans(n_clusters=k, random_state=0)
labels = km.fit_predict(x)  score =
silhouette_score(x,            labels)
silh.append(score)     #   plot   the
silhoutte scores  plt.title('Silhoutte
Analysis')  plt.xlabel('Value of K')
plt.ylabel('Silhoutte           Score')
plt.xticks(range(2,16))
plt.bar(range(2,16),                silh,
color='g')  # Create the object
km = KMeans(n_clusters=5, random_state=0)
# Train the algorithm  labels
= km.fit_predict(x)
labels  # Cluster labels  km.labels_  #
SSE     km.inertia_     #   Centroids
km.cluster_centers_    # Extract  the
clusters    df[labels==2]  #  Boolean
filtering       one     =    df[labels==1]
one.shape     #   Export   the   cluster
one.to_csv('one.csv')   print('Cluster-
0:',                    len(df[labels==0]))
print('Cluster-1:',
len(df[labels==1]))     print('Cluster-
2:',                    len(df[labels==2]))
print('Cluster-3:',
len(df[labels==3]))     print('Cluster-
4:', len(df[labels==4]))
# Prediction  new =
[[45,             76]]
km.predict(new)[0
]  # Prediction  new
=      [[25,      36]]
km.predict(new)[0
```

```
] # Prediction new
= [[85, 76]]
km.predict(new)[0
] # Prediction new
= [[45, 47]]
km.predict(new)[0
] # Visualization of
clusters
plt.title('Uncluster
ed data')
plt.xlabel('Annual
Income')
plt.ylabel('Spendin
g Score')
plt.grid()
plt.scatter(x['Annual Income (k$)'], x['Spending Score (1-100)'])
# Save the centroids cent
= km.cluster_centers_
cent
# Visualization of clusters
plt.title('Clustered data')
plt.xlabel('Annual Income')
plt.ylabel('Spending
Score')
plt.grid()
plt.scatter(x['Annual Income (k$)'], x['Spending Score (1-100)'], c
= labels, marker='*')
plt.scatter(cent[:,0], cent[:,1], s=100, marker='o', color='r')
# Combined plot
plt.figure(figsize=(16,9))
plt.subplot(1,2,1)
plt.title('Unclustered data')
plt.xlabel('Annual Income')
plt.ylabel('Spending
Score')
plt.grid()
plt.scatter(x['Annual Income (k$)'], x['Spending Score (1-100)'])
plt.subplot(1,2,2) plt.title('Clustered data') plt.xlabel('Annual
Income') plt.ylabel('Spending Score')
plt.grid()
```

```
plt.scatter(x['Annual Income (k$)'], x['Spending Score (1-100)'],  c
= labels, marker='*')
plt.scatter(cent[:,0], cent[:,1], s=100, marker='o', color='r',
label       =       'Centroid')            plt.legend()
plt.savefig('Clusters.png')  import seaborn as
sns   # Visualization using joint plot   p =
sns.jointplot(x=x['Annual    Income    (k$)'],
y=x['Spending   Score   (1-100)'],     hue   =
labels,palette='rainbow',        )              #
sns.jointplot(x=cent[:,0],          y=cent[:,1])
p.savefig('seaborn_clusters.png')
```

Output –

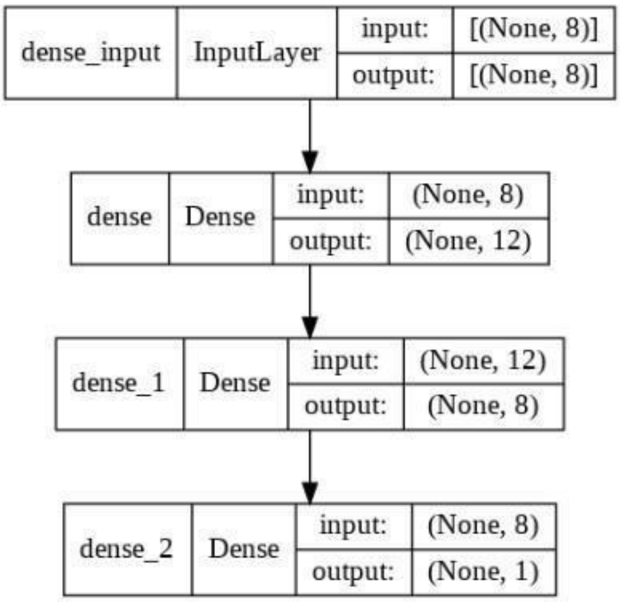


Q. 11 Write a program for Creating & Visualizing Neural Network for the given data. (Use python).

Program –

```
from keras.layers import Dense
from keras.models import Sequential
import numpy as np
# fix random seed for reproducibility  seed
= 7
np.random.seed(seed)  #
load dataset
dataset = np.loadtxt('pima-new.csv', delimiter=',')
dataset dataset.shape  # input data
X = dataset[:,:8]
# output data
Y = dataset[:,8]
X.shape
Y
# create the model  model
= Sequential()
model.add(Dense(12, input_dim=8, activation='relu')) # Input layer
model.add(Dense(8,     activation='relu'))     #     Hiddel     layer
model.add(Dense(1, activation='sigmoid')) # Output layer
# compile model
model.compile(loss='binary_crossentropy'
,  optimizer='adam',  metrics=['accuracy'])
# train the model
model.fit(X, Y, epochs=200, batch_size=10)
# Evaluate the model  scores =
model.evaluate(X, Y)
scores
new                                        =
[[7,475,82,69,120,22.2,0.645,57]]
model.predict(new)  # Visualize
from keras.utils.vis_utils import plot_mode
plot_model(model, show_shapes=True, show_layer_names=True,  to_file='neural_network.png')
```

Output –




```
In [18]: # train the model
         model.fit(X, Y, epochs=200, batch_size=10)

Epoch 1/200
77/77 [==============================] - 1s 2ms/step - loss: 5.7165 - accuracy: 0.6107
Epoch 2/200
77/77 [==============================] - 0s 2ms/step - loss: 1.4259 - accuracy: 0.5911
Epoch 3/200
77/77 [==============================] - 0s 2ms/step - loss: 1.1124 - accuracy: 0.6250
Epoch 4/200
77/77 [==============================] - 0s 2ms/step - loss: 0.9411 - accuracy: 0.6393
Epoch 5/200
77/77 [==============================] - 0s 2ms/step - loss: 0.8355 - accuracy: 0.6445
Epoch 6/200
77/77 [==============================] - 0s 2ms/step - loss: 0.7805 - accuracy: 0.6445
Epoch 7/200
77/77 [==============================] - 0s 2ms/step - loss: 0.7478 - accuracy: 0.6458
Epoch 8/200
77/77 [==============================] - 0s 2ms/step - loss: 0.7384 - accuracy: 0.6458
Epoch 9/200
77/77 [==============================] - 0s 2ms/step - loss: 0.6907 - accuracy: 0.6641
Epoch 10/200
```

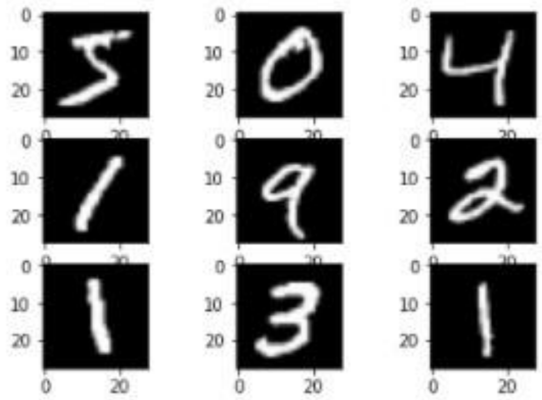
Q. 12 Write a for Recognize optical character using ANN.

Program –

```
from keras.datasets import mnist import matplotlib.pyplot as plt
(X_train, y_train), (X_test, y_test) = mnist.load_data() plt.subplot(3,3,1)
plt.imshow(X_train[0],   cmap=plt.get_cmap('gray'))   plt.subplot(3,3,2)
plt.imshow(X_train[1],   cmap=plt.get_cmap('gray'))   plt.subplot(3,3,3)
plt.imshow(X_train[2],   cmap=plt.get_cmap('gray'))   plt.subplot(3,3,4)
plt.imshow(X_train[3],   cmap=plt.get_cmap('gray'))   plt.subplot(3,3,5)
plt.imshow(X_train[4],   cmap=plt.get_cmap('gray'))   plt.subplot(3,3,6)
plt.imshow(X_train[5],   cmap=plt.get_cmap('gray'))   plt.subplot(3,3,7)
plt.imshow(X_train[6],   cmap=plt.get_cmap('gray'))   plt.subplot(3,3,8)
plt.imshow(X_train[7], cmap=plt.get_cmap('gray')) plt.subplot(3,3,9)
plt.imshow(X_train[8], cmap=plt.get_cmap('gray')) from keras.layers import Dense
from keras.models import Sequential import numpy as np  num_pixels =
X_train[0].shape[0] * X_train[0].shape[1] # Reshape X_train =
X_train.reshape(X_train.shape[0], num_pixels) X_test =
X_test.reshape(X_test.shape[0], num_pixels) import pandas as pd
pd.DataFrame(X_train).describe()
# normalize inputs from 0-255 to 0-1 X_train = X_train / 255
X_test = X_test / 255 set(y_train)  from
keras.utils import np_utils
y_train = np_utils.to_categorical(y_train) y_test = np_utils.to_categorical(y_test) y_train.shape
# Create the model  model
= Sequential()
model.add(Dense(784, input_dim= 784, activation='relu')) model.add(Dense(10,
activation='softmax'))  # compile model
model.compile(loss='categorical_crossentropy',
optimizer='adam', metrics=['accuracy'])
# Train the algorithm  model.fit(X_train, y_train, validation_data=(X_test, y_test),
epochs=10, batch_size=200)
scores = model.evaluate(X_train, y_train)
scores
```

Output –

```
Out[6]: <matplotlib.image.AxesImage at 0x152e823fc40>
```



```
In [24]: # Train the algorithm
         model.fit(X_train, y_train, validation_data=(X_test, y_test),
                 epochs=10, batch_size=200)

Epoch 1/10
300/300 [==============================] - 8s 25ms/step - loss: 0.2751 - accuracy: 0.9225 - val_lo
9574
Epoch 2/10
300/300 [==============================] - 7s 24ms/step - loss: 0.1116 - accuracy: 0.9673 - val_lo
9712
Epoch 3/10
300/300 [==============================] - 7s 23ms/step - loss: 0.0707 - accuracy: 0.9793 - val_lo
9760
Epoch 4/10
300/300 [==============================] - 7s 22ms/step - loss: 0.0507 - accuracy: 0.9854 - val_lo
9783
Epoch 5/10
300/300 [==============================] - 7s 22ms/step - loss: 0.0357 - accuracy: 0.9903 - val_lo
9779
Epoch 6/10
300/300 [==============================] - 7s 22ms/step - loss: 0.0267 - accuracy: 0.9924 - val_lo
9805
Epoch 7/10
300/300 [==============================] - 7s 22ms/step - loss: 0.0199 - accuracy: 0.9950 - val_lo
9823
Epoch 8/10
300/300 [==============================] - 7s 22ms/step - loss: 0.0156 - accuracy: 0.9960 - val_lo
9816
Epoch 9/10
```

```
In [15]: import pandas as pd
         pd.DataFrame(X_train).describe()
```

Out[15]:

	0	1	2	3	4	5	6	7	8	9	...	774	775	776	
count	60000.0	60000.0	60000.0	60000.0	60000.0	60000.0	60000.0	60000.0	60000.0	60000.0	...	60000.000000	60000.000000	60000.000000	60000
mean	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.200433	0.088867	0.045633	0
std	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	6.042472	3.956189	2.839845	1
min	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.000000	0.000000	0
25%	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.000000	0.000000	0
50%	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.000000	0.000000	0
75%	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.000000	0.000000	0
max	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	254.000000	254.000000	253.000000	253

8 rows × 784 columns

Q.13 Write a program to implement CNN

Program –

```
from keras.models import Sequential from keras.layers import Dense
from keras.layers import Conv2D from keras.layers import MaxPool2D from keras.layers
import Flatten
# Create the object of model classifier = Sequential()
# Add first convolution layer
# Parameters - filters, kernel size, input shape, activation classifier.add(Conv2D(32,(3,3),
input_shape = (64, 64, 3),  activation = 'relu'))
# Add first max pooling layer
classifier.add(MaxPool2D(pool_size = (2,2)))
# Add second convolution layer
classifier.add(Conv2D(32, (3,3), activation = 'relu'))
# Add max pooling layer classifier.add(MaxPool2D(pool_size = (2,2)))
# Convert the 2D data to 1D format classifier.add(Flatten())
# Add the output layer classifier.add(Dense(units=1, activation='sigmoid'))
# Compile the model classifier.compile(optimizer='adam',  loss='binary_crossentropy',
metrics=['accuracy'])
#  Image  augmentation    from
keras.preprocessing.image
import ImageDataGenerator train_datagen = ImageDataGenerator(rescale=1/255,
shear_range=0.2, zoom_range=0.2, horizontal_flip=True, vertical_flip=True)
test_datagen = ImageDataGenerator(rescale = 1./255)
# Import the train images
train                                                  =
train_datagen.flow_from_directory('/content/sample_data',
target_size=(64, 64), batch_size=32, class_mode='binary')  test =
test_datagen.flow_from_directory('/content/sample_data',
target_size=(64, 64), batch_size=32, class_mode='binary')
# Train the algorithm
classifier.fit(train, epochs=10, validation_data=test, validation_steps=10)
train.class_indices  # Prediction  import numpy as np
from keras.preprocessing.image import load_img from keras.preprocessing.image import
img_to_array
test_image = load_img('/content/sample_data/sample1.jpg', target_size=(64, 64)) test_image =
img_to_array(test_image)
test_image = np.expand_dims(test_image, axis = 0)  #test_image.shape
```

result = classifier.predict(test_image) if result[0][0] == 1:
print('Orange') else: print('Apple')

Output –

```
test_image = load_img('dataset/dog.3923.jpg', target_size=(64, 64))
test_image = img_to_array(test_image)
test_image = np.expand_dims(test_image, axis = 0)
#test_image.shape

result = classifier.predict(test_image)
if result[0][0] == 1:
    print('Orange')
else:
    print('Apple')
```

```
Orange
```

Q.14 Write a program to implement RNN

Program –

```
import matplotlib.pyplot as plt import pandas as pd  import
numpy as np # Data import
df = pd.read_csv('/content/sample_data/Google_Stock_Price_Train.csv') # first 5 entries
df.head() df.describe() df.info()
training_set = df.iloc[:,[1,2]].values # Visualize the trend plt.plot(training_set)
# Feature scaling  from sklearn.preprocessing import MinMaxScaler scaler
= MinMaxScaler()  training_set_scaled = scaler.fit_transform(training_set)
# The scaled data   training_set_scaled # plot the scaled data
plt.plot(training_set_scaled) X_train = []  y_train = []
for i in range(60, 1258): X_train.append(training_set_scaled[i-60:i, 0])
y_train.append(training_set_scaled[i, 0])
X_train, y_train = np.array(X_train), np.array(y_train)
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1)) # Import the classes
from keras.models import Sequential from keras.layers import Dense
from keras.layers import LSTM from keras.layers import Dropout # Create the model  regressor
= Sequential() # add LSTM layer
regressor.add(LSTM(units = 50, return_sequences = True, input_shape = (X_train.shape[1], 1)))
regressor.add(LSTM(units = 50, return_sequences = True)) regressor.add(Dropout(0.2))
regressor.add(LSTM(units = 50, return_sequences = True)) regressor.add(Dropout(0.2))
regressor.add(LSTM(units = 50))
regressor.add(Dropout(0.2)) # Output layer regressor.add(Dense(1))
# Compile the model
regressor.compile(optimizer='adam', loss='mean_squared_error') # Train the algorithm
regressor.fit(X_train, y_train, epochs=100, batch_size = 32)
testing_set =
pd.read_csv('/content/sample_data/Google_Stock_Price_Test.csv')
testing_set.shape  testing_set
real_stock_price = testing_set.iloc[:,[1,2]].values real_stock_price
dataset_total = pd.concat((df['Open'],  testing_set['Open']), axis =
0)  dataset_total
inputs = dataset_total[len(dataset_total) -
len(testing_set) - 60:].values
inputs.shape
inputs = inputs.reshape(-1,2) inputs.shape
# Perform the scaling  inputs =
scaler.transform(inputs) inputs
```

Output –

```
6] # Train the algorithm
   regressor.fit(X_train, y_train, epochs=100, batch_size = 32)
```

```
Epoch 1/100
38/38 [==============================] - 28s 229ms/step - loss: 0.0432
Epoch 2/100
38/38 [==============================] - 5s 138ms/step - loss: 0.0064
Epoch 3/100
38/38 [==============================] - 5s 141ms/step - loss: 0.0051
Epoch 4/100
38/38 [==============================] - 5s 137ms/step - loss: 0.0046
Epoch 5/100
38/38 [==============================] - 5s 141ms/step - loss: 0.0049
Epoch 6/100
38/38 [==============================] - 5s 137ms/step - loss: 0.0052
Epoch 7/100
38/38 [==============================] - 5s 141ms/step - loss: 0.0050
Epoch 8/100
38/38 [==============================] - 5s 139ms/step - loss: 0.0045
Epoch 9/100
38/38 [==============================] - 5s 137ms/step - loss: 0.0052
Epoch 10/100
38/38 [==============================] - 5s 138ms/step - loss: 0.0048
Epoch 11/100
38/38 [==============================] - 5s 140ms/step - loss: 0.0037
Epoch 12/100
38/38 [==============================] - 5s 137ms/step - loss: 0.0042
Epoch 13/100
38/38 [==============================] - 5s 139ms/step - loss: 0.0038
Epoch 14/100
38/38 [==============================] - 5s 141ms/step - loss: 0.0038
Epoch 15/100
38/38 [==============================] - 5s 141ms/step - loss: 0.0037
Epoch 16/100
38/38 [==============================] - 5s 136ms/step - loss: 0.0037
Epoch 17/100
38/38 [==============================] - 5s 138ms/step - loss: 0.0037
Epoch 18/100
38/38 [==============================] - 6s 156ms/step - loss: 0.0034
Epoch 19/100
38/38 [==============================] - 5s 138ms/step - loss: 0.0039
Epoch 20/100
38/38 [==============================] - 5s 138ms/step - loss: 0.0036
Epoch 21/100
38/38 [==============================] - 5s 140ms/step - loss: 0.0032
```

	Date	Open	High	Low	Close	Volume
0	1/3/2017	778.81	789.63	775.80	786.14	1,657,300
1	1/4/2017	788.36	791.34	783.16	786.90	1,073,000
2	1/5/2017	786.08	794.48	785.02	794.02	1,335,200
3	1/6/2017	795.26	807.90	792.20	806.15	1,640,200
4	1/9/2017	806.40	809.97	802.83	806.65	1,272,400
5	1/10/2017	807.86	809.13	803.51	804.79	1,176,800
6	1/11/2017	805.00	808.15	801.37	807.91	1,065,900
7	1/12/2017	807.14	807.39	799.17	806.36	1,353,100
8	1/13/2017	807.48	811.22	806.69	807.88	1,099,200
9	1/17/2017	807.08	807.14	800.37	804.61	1,362,100
10	1/18/2017	805.81	806.21	800.99	806.07	1,294,400
11	1/19/2017	805.12	809.48	801.80	802.17	919,300
12	1/20/2017	806.91	806.91	801.69	805.02	1,670,000
13	1/23/2017	807.25	820.87	803.74	819.31	1,963,600
14	1/24/2017	822.30	825.90	817.82	823.87	1,474,000
15	1/25/2017	829.62	835.77	825.06	835.67	1,494,500
16	1/26/2017	837.81	838.00	827.01	832.15	2,973,900
17	1/27/2017	834.71	841.95	820.44	823.31	2,965,800
18	1/30/2017	814.66	815.84	799.80	802.32	3,246,600
19	1/31/2017	796.86	801.25	790.52	796.79	2,160,600

```
array([[0.9299055 , 0.93086447],
       [0.92750577, 0.9439371 ],
       [0.93876032, 0.9337778 ],
       [0.93483518, 0.93112593],
       [0.94636878, 0.96556296],
       [0.97510976, 0.9595122 ],
       [0.97808617, 1.        ],
       [0.98076494, 0.97071731],
       [0.98450406, 0.96038994],
       [0.9371419 , 0.9281379 ],
       [0.90804747, 0.87670644],
       [0.92153434, 0.93784899],
       [0.93165414, 0.95235961],
       [0.88812412, 0.88593198],
       [0.87032145, 0.88518498],
       [0.90743359, 0.91538275],
       [0.89941588, 0.91773582],
       [0.9089404 , 0.90210469],
       [0.89456061, 0.91568155],
       [0.9132934 , 0.88936822],
       [0.86589404, 0.88987245],
       [0.90335962, 0.89601658],
       [0.91777662, 0.93149943],
       [0.94114145, 0.95745793],
       [0.96413424, 0.9638822 ],
       [0.96971501, 0.95058547],
       [0.96294367, 0.96108092],
       [0.95475854, 0.95185538],
       [0.95163331, 0.95708443],
       [0.93796041, 0.93663511],
       [0.92955205, 0.94711188],
       [0.94307612, 0.95999776],
       [0.98087655, 0.98352849],
       [0.97827219, 0.98218388],
       [0.98288563, 0.98207182],
       [0.979779  , 0.97841149],
       [0.98182528, 0.9823893 ],
       [1.01045465, 1.02416569],
       [1.03930724, 1.03367135],
       [0.99624228, 0.96298579]])
```

Q.15 Write a program to implement GAN

Program –

```
from future import print_function, division from keras.datasets import mnist  from
keras.layers import Input, Dense, Reshape, Flatten, Dropout
from keras.layers import BatchNormalization, Activation, ZeroPadding2D from
keras.layers.advanced_activations import LeakyReLU
from keras.layers.convolutional import UpSampling2D, Conv2D from keras.models import
Sequential, Model
from tensorflow.keras.optimizers import Adam import matplotlib.pyplot as plt
import sys  import numpy as np class GAN():  def init (self): self.img_rows =
28  self.img_cols = 28  self.channels = 1
self.img_shape = (self.img_rows, self.img_cols, self.channels) self.latent_dim = 100  optimizer
= Adam(0.0002, 0.5)
# Build and compile the discriminator self.discriminator = self.build_discriminator()
self.discriminator.compile(loss='binary_crossentropy',
optimizer=optimizer, metrics=['accuracy'])
# Build the generator
self.generator = self.build_generator()
# The generator takes noise as input and generates imgs z = Input(shape=(self.latent_dim,))
img = self.generator(z)
# For the combined model we will only train the generator self.discriminator.trainable = False
# The discriminator takes generated images as input and determines validity validity =
self.discriminator(img)
# The combined model (stacked generator and discriminator) # Trains the generator to fool the
discriminator self.combined = Model(z, validity)
self.combined.compile(loss='binary_crossentropy',
optimizer=optimizer)
def build_generator(self): model = Sequential()
model.add(Dense(256, input_dim=self.latent_dim))
model.add(LeakyReLU(alpha=0.2))  model.add(BatchNormalization(momentum=0.8))
model.add(Dense(512)) model.add(LeakyReLU(alpha=0.2))
model.add(BatchNormalization(momentum=0.8)) model.add(Dense(1024))
model.add(LeakyReLU(alpha=0.2))  model.add(BatchNormalization(momentum=0.8))
model.add(Dense(np.prod(self.img_shape), activation='tanh'))
model.add(Reshape(self.img_shape))  model.summary()
noise = Input(shape=(self.latent_dim,))
img = model(noise)
```

```
return Model(noise, img) def build_discriminator(self):  model
= Sequential()
model.add(Flatten(input_shape=self.img_shape))                    model.add(Dense(512))
model.add(LeakyReLU(alpha=0.2)) model.add(Dense(256))
model.add(LeakyReLU(alpha=0.2))          model.add(Dense(1,          activation='sigmoid'))
model.summary()
img = Input(shape=self.img_shape) validity = model(img)
return  Model(img,  validity)   def  train(self,  epochs,
batch_size=128, sample_interval=50):
# Load the dataset
(X_train, _), (_, _) = mnist.load_data()
# Rescale -1 to 1
X_train = X_train / 127.5 - 1.
X_train = np.expand_dims(X_train, axis=3)
# Adversarial ground truths valid = np.ones((batch_size, 1)) fake = np.zeros((batch_size, 1))
for epoch in range(epochs): #
# Train Discriminator #
# Select a random batch of images
idx = np.random.randint(0, X_train.shape[0], batch_size) imgs = X_train[idx]
noise = np.random.normal(0, 1, (batch_size, self.latent_dim)) # Generate a batch of new images
gen_imgs = self.generator.predict(noise)
# Train the discriminator
d_loss_real = self.discriminator.train_on_batch(imgs, valid) d_loss_fake =
self.discriminator.train_on_batch(gen_imgs, fake) d_loss = 0.5 * np.add(d_loss_real,
d_loss_fake)
# # Train Generator #
noise = np.random.normal(0, 1, (batch_size, self.latent_dim))
# Train the generator (to have the discriminator label samples as valid) g_loss =
self.combined.train_on_batch(noise, valid)
# Plot the progress
print ("%d [D loss: %f, acc.: %.2f%%] [G loss: %f]" % (epoch, d_loss[0], 100*d_loss[1  ],
g_loss))
# If at save interval => save generated image samples   if
epoch % sample_interval == 0: self.sample_images(epoch)
def sample_images(self, epoch): r, c = 5, 5
noise = np.random.normal(0, 1, (r * c, self.latent_dim)) gen_imgs =
self.generator.predict(noise)  # Rescale images 0 - 1  gen_imgs =
0.5 * gen_imgs + 0.5  fig, axs = plt.subplots(r, c) cnt = 0  for i in
range(r): for j in range(c):
```

axs[i,j].imshow(gen_imgs[cnt, :,:,0], cmap='gray') axs[i,j].axis('off')
cnt += 1 fig.savefig("/content/sample_data/d.jpg" % epoch)
plt.close()
gan = GAN()
gan.train(epochs=200, batch_size=32, sample_interval=200)

Output –

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_1 (Flatten)          (None, 784)               0
_________________________________________________________________
dense_7 (Dense)              (None, 512)               401920
_________________________________________________________________
leaky_re_lu_5 (LeakyReLU)    (None, 512)               0
_________________________________________________________________
dense_8 (Dense)              (None, 256)               131328
_________________________________________________________________
leaky_re_lu_6 (LeakyReLU)    (None, 256)               0
_________________________________________________________________
dense_9 (Dense)              (None, 1)                 257
=================================================================
Total params: 533,505
Trainable params: 533,505
Non-trainable params: 0
_________________________________________________________________
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_10 (Dense)             (None, 256)               25856
_________________________________________________________________
leaky_re_lu_7 (LeakyReLU)    (None, 256)               0
_________________________________________________________________
batch_normalization_3 (Batch (None, 256)               1024
_________________________________________________________________
dense_11 (Dense)             (None, 512)               131584
_________________________________________________________________
leaky_re_lu_8 (LeakyReLU)    (None, 512)               0
_________________________________________________________________
batch_normalization_4 (Batch (None, 512)               2048
_________________________________________________________________
dense_12 (Dense)             (None, 1024)              525312
_________________________________________________________________
leaky_re_lu_9 (LeakyReLU)    (None, 1024)              0
_________________________________________________________________
batch_normalization_5 (Batch (None, 1024)              4096
_________________________________________________________________
dense_13 (Dense)             (None, 784)               803600
_________________________________________________________________
reshape_1 (Reshape)          (None, 28, 28, 1)         0
=================================================================
Total params: 1,493,520
Trainable params: 1,489,936
Non-trainable params: 3,584
```