Machine Learning Project

April 24, 2021

[1]:
```python
#Importing useful libraries
import pandas as pd
#import numpy as np
import sklearn
from sklearn import linear_model
from sklearn.utils import shuffle
```

[2]:
```python
#Importing and reading data.
data = pd.read_csv("student-mat.csv", sep=";")
# Since our data is seperated by semicolons we need to do sep=";"
```

[3]:
```python
# Displaying the first 6 rows of the data.
data.head(n=6)
```

[3]:
```
  school sex  age address famsize Pstatus  Medu  Fedu      Mjob      Fjob  \
0     GP   F   18       U     GT3       A     4     4   at_home   teacher
1     GP   F   17       U     GT3       T     1     1   at_home     other
2     GP   F   15       U     LE3       T     1     1   at_home     other
3     GP   F   15       U     GT3       T     4     2    health  services
4     GP   F   16       U     GT3       T     3     3     other     other
5     GP   M   16       U     LE3       T     4     3  services     other

   … famrel freetime  goout  Dalc  Walc health absences  G1  G2  G3
0  …      4        3      4     1     1      3        6   5   6   6
1  …      5        3      3     1     1      3        4   5   5   6
2  …      4        3      2     2     3      3       10   7   8  10
3  …      3        2      2     1     1      5        2  15  14  15
4  …      4        3      2     1     2      5        4   6  10  10
5  …      5        4      2     1     2      5       10  15  15  15

[6 rows x 33 columns]
```

[4]:
```python
#Gives the number of columns and rows in our data
data.shape
```

[4]: (395, 33)

[5]:
```
data.values
```

[5]:
```
array([['GP', 'F', 18, …, 5, 6, 6],
       ['GP', 'F', 17, …, 5, 5, 6],
       ['GP', 'F', 15, …, 7, 8, 10],
       …,
       ['MS', 'M', 21, …, 10, 8, 7],
       ['MS', 'M', 18, …, 11, 12, 10],
       ['MS', 'M', 19, …, 8, 9, 9]], dtype=object)
```

[6]:
```
#Summary statistics of the data. Gives summary statistics of the data in␣
↪numeric form.
data.describe()
```

[6]:
```
              age        Medu        Fedu  traveltime   studytime    failures  \
count  395.000000  395.000000  395.000000  395.000000  395.000000  395.000000
mean    16.696203    2.749367    2.521519    1.448101    2.035443    0.334177
std      1.276043    1.094735    1.088201    0.697505    0.839240    0.743651
min     15.000000    0.000000    0.000000    1.000000    1.000000    0.000000
25%     16.000000    2.000000    2.000000    1.000000    1.000000    0.000000
50%     17.000000    3.000000    2.000000    1.000000    2.000000    0.000000
75%     18.000000    4.000000    3.000000    2.000000    2.000000    0.000000
max     22.000000    4.000000    4.000000    4.000000    4.000000    3.000000

           famrel    freetime       goout        Dalc        Walc      health  \
count  395.000000  395.000000  395.000000  395.000000  395.000000  395.000000
mean     3.944304    3.235443    3.108861    1.481013    2.291139    3.554430
std      0.896659    0.998862    1.113278    0.890741    1.287897    1.390303
min      1.000000    1.000000    1.000000    1.000000    1.000000    1.000000
25%      4.000000    3.000000    2.000000    1.000000    1.000000    3.000000
50%      4.000000    3.000000    3.000000    1.000000    2.000000    4.000000
75%      5.000000    4.000000    4.000000    2.000000    3.000000    5.000000
max      5.000000    5.000000    5.000000    5.000000    5.000000    5.000000

         absences          G1          G2          G3
count  395.000000  395.000000  395.000000  395.000000
mean     5.708861   10.908861   10.713924   10.415190
std      8.003096    3.319195    3.761505    4.581443
min      0.000000    3.000000    0.000000    0.000000
25%      0.000000    8.000000    9.000000    8.000000
50%      4.000000   11.000000   11.000000   11.000000
75%      8.000000   13.000000   13.000000   14.000000
max     75.000000   19.000000   19.000000   20.000000
```

[7]:
```
data.columns
```

[7]:
```
Index(['school', 'sex', 'age', 'address', 'famsize', 'Pstatus', 'Medu', 'Fedu',
       'Mjob', 'Fjob', 'reason', 'guardian', 'traveltime', 'studytime',
       'failures', 'schoolsup', 'famsup', 'paid', 'activities', 'nursery',
       'higher', 'internet', 'romantic', 'famrel', 'freetime', 'goout', 'Dalc',
       'Walc', 'health', 'absences', 'G1', 'G2', 'G3'],
      dtype='object')
```

[8]:
```
#Picking only a few columns of the data and saving the new data as data1

data1=data[["sex","age","G1","G2","G3"]]
data1.head()
```

[8]:

	sex	age	G1	G2	G3
0	F	18	5	6	6
1	F	17	5	5	6
2	F	15	7	8	10
3	F	15	15	14	15
4	F	16	6	10	10

[9]:
```
data.dtypes
```

[9]:
```
school         object
sex            object
age             int64
address        object
famsize        object
Pstatus        object
Medu            int64
Fedu            int64
Mjob           object
Fjob           object
reason         object
guardian       object
traveltime      int64
studytime       int64
failures        int64
schoolsup      object
famsup         object
paid           object
activities     object
nursery        object
higher         object
internet       object
romantic       object
famrel          int64
freetime        int64
goout           int64
```

```
Dalc          int64
Walc          int64
health        int64
absences      int64
G1            int64
G2            int64
G3            int64
dtype: object
```

[10]:
```
data.iloc[:,0:5]
```

[10]:
```
    school sex  age address famsize
0       GP   F   18       U     GT3
1       GP   F   17       U     GT3
2       GP   F   15       U     LE3
3       GP   F   15       U     GT3
4       GP   F   16       U     GT3
..     …  ..   …      …       …
390     MS   M   20       U     LE3
391     MS   M   17       U     LE3
392     MS   M   21       R     GT3
393     MS   M   18       R     LE3
394     MS   M   19       U     LE3

[395 rows x 5 columns]
```

[11]:
```
data.iloc[0:5,0:4]
```

[11]:
```
  school sex  age address
0     GP   F   18       U
1     GP   F   17       U
2     GP   F   15       U
3     GP   F   15       U
4     GP   F   16       U
```

[12]:
```
data.iloc[0:5,[6,14,21,30]]
```

[12]:
```
   Medu  failures internet  G1
0     4         0       no   5
1     1         0      yes   5
2     1         3      yes   7
3     4         0      yes  15
4     3         0       no   6
```

[13]:
```
data.loc[0:5,["G1","age","G3"]]
```

[13]:
```
    G1  age  G3
0    5   18   6
1    5   17   6
2    7   15  10
3   15   15  15
4    6   16  10
5   15   16  15
```

0.1 DECISION TREE FOR CLASSIFICATION

[14]:
```python
import numpy as np
import pandas as pd
import random
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn import tree

Age_male=[random.randint(5,80) for i in range(50000)]
Age_female=[random.randint(5,80) for i in range(50000)]
Gender_male=[1 for i in range(50000)]
Gender_female=[0 for i in range(50000)]
#Preference=[random.randint(1,3) for i in range(100000)]
Preference=[]
Age=Age_male+Age_female
Gender=Gender_male+Gender_female
for i in range(100000):
    if (Age[i]<=35 and Gender[i]==1):
        Preference.append('rhumba')
    elif (Age[i]<=35 and Gender[i]==0):
        Preference.append('bongo')
    else:
        Preference.append('reggae')


data=[[Age[i],Gender[i],Preference[i]] for i in range(len(Age))]
data=pd.DataFrame(data, columns=['Age','Gender','Preference'])
data.shape

print(data.head())
X=data.drop(columns='Preference')
y=data['Preference']

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2)

model=DecisionTreeClassifier()
```

```
model.fit(X_train,y_train)

predictions=model.predict(X_test)

score=accuracy_score(y_test,predictions)
print(' Accuracy score is '+ str(score))
```

```
   Age  Gender Preference
0   29       1     rhumba
1   26       1     rhumba
2   21       1     rhumba
3   40       1     reggae
4   76       1     reggae
 Accuracy score is 1.0
```

[15]:
```
model.predict([[5,1]])
```

[15]:
```
array(['rhumba'], dtype=object)
```

0.2 Logistic regression for Classication

[1]:
```
from sklearn.linear_model import LogisticRegression
logmodel=LogisticRegression()
#logmodel.fit(X,y)
#predict=logmodel.predict([[60,1]])
logmodel.fit(X_train,y_train)
predict=logmodel.predict(X_test)
score=accuracy_score(y_test,predict)

#tree.export_graphviz(model,out_file='music_recommender1.
↪dot',feature_names=['age','sex'],class_names=sorted(y.
↪unique()),label='all',rounded=True,filled=True)

score
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-b16f3e781325> in <module>
      3 #logmodel.fit(X,y)
      4 #predict=logmodel.predict([[60,1]])
----> 5 logmodel.fit(X_train,y_train)
      6 predict=logmodel.predict(X_test)
      7 score=accuracy_score(y_test,predict)

NameError: name 'X_train' is not defined
```

[17]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test,predict)
```

[17]:
```
array([[ 4020,     0,     0],
       [    0, 11832,     0],
       [    0,     0,  4148]])
```

[2]:
```
import pandas as pd
#Importing and reading data.
Iris= pd.read_csv("Iris.csv")
# Since our data is seperated by semicolons we need to do sep=";"
Iris=Iris.iloc[:,[1,2,3,4,5]]
Iris.head()
```

[2]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

[3]:
```
X=Iris.drop(columns='Species')
y=Iris['Species']

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2)

model=DecisionTreeClassifier()
model.fit(X_train,y_train)

predictions=model.predict(X_test)

A=[[y_test],[predictions]]
score=accuracy_score(y_test,predictions)
score
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-3-4d5c162d51e0> in <module>
      2 y=Iris['Species']
      3
----> 4 X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2)
      5
      6 model=DecisionTreeClassifier()

NameError: name 'train_test_split' is not defined
```

[4]:

```
logmodel=LogisticRegression()
#logmodel.fit(X,y)
#predict=logmodel.predict([[60,1]])
logmodel.fit(X_train,y_train)
predict=logmodel.predict(X_test)
score=accuracy_score(y_test,predict)

#tree.export_graphviz(model,out_file='music_recommender1.
↪dot',feature_names=['age','sex'],class_names=sorted(y.
↪unique()),label='all',rounded=True,filled=True)

score
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-4-81df3114425f> in <module>
----> 1 logmodel=LogisticRegression()
      2 #logmodel.fit(X,y)
      3 #predict=logmodel.predict([[60,1]])
      4 logmodel.fit(X_train,y_train)
      5 predict=logmodel.predict(X_test)

NameError: name 'LogisticRegression' is not defined
```

[5]:

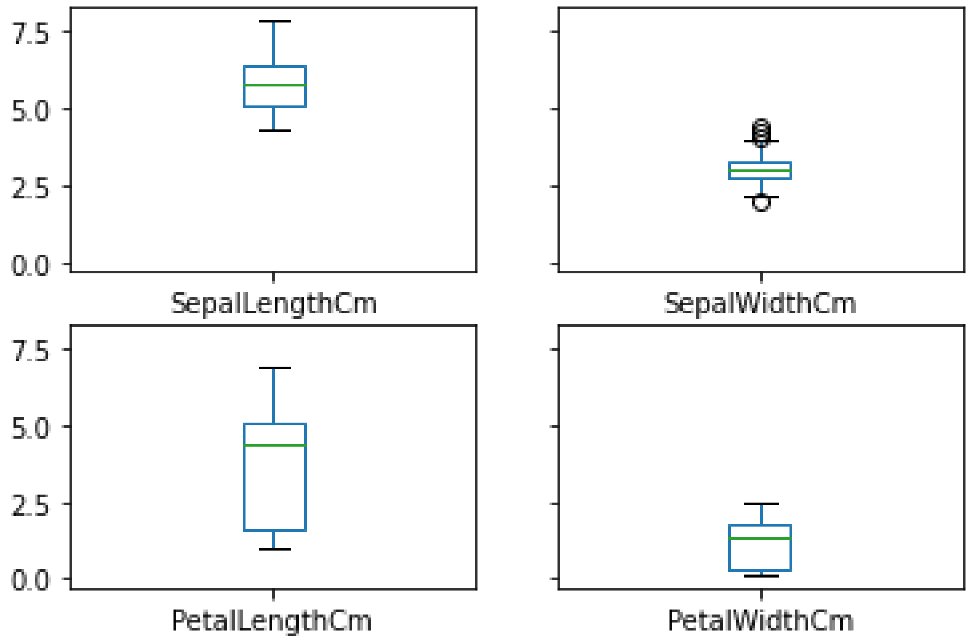
```
Iris.groupby('Species').size()
```

[5]:

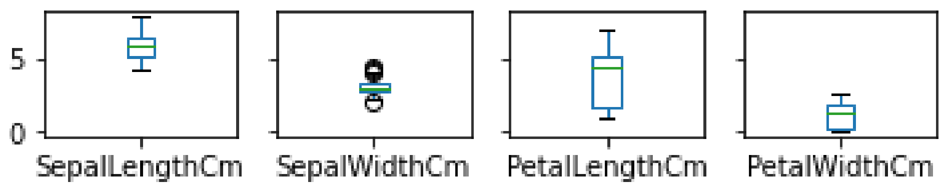
```
Species
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
dtype: int64
```

[6]:

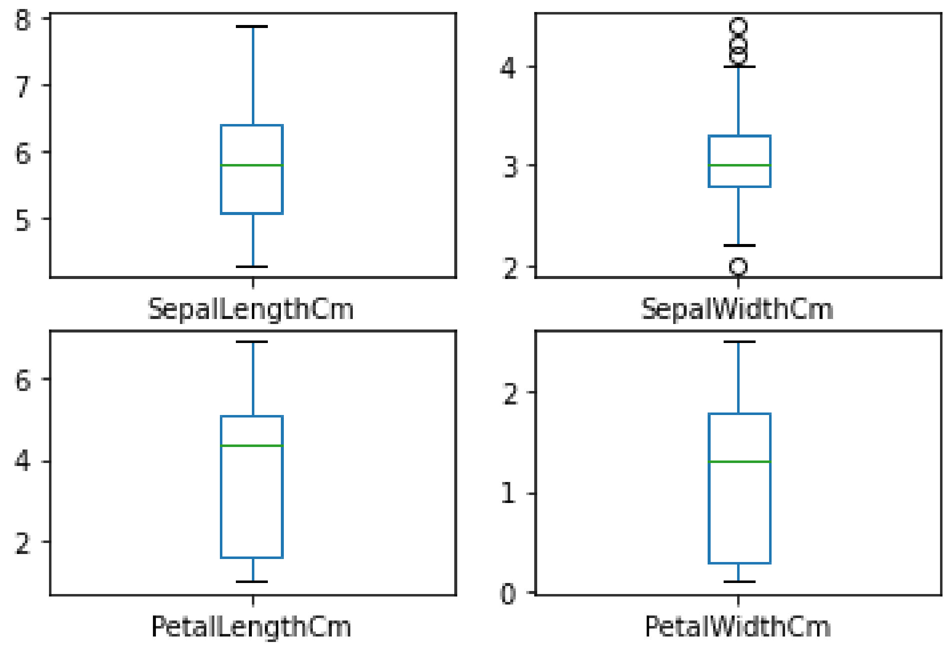
```
import matplotlib.pyplot as plt

Iris.plot(kind='box',subplots=True,layout=(2,2),sharex=True,sharey=True)
plt.show()
```



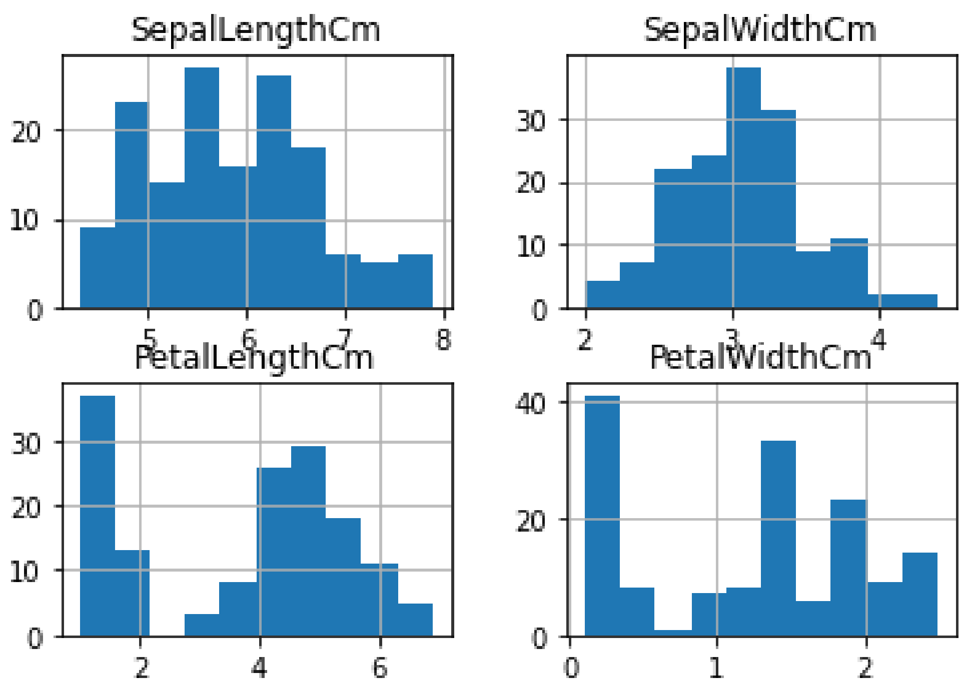
[7]:
```
Iris.plot(kind='box',subplots=True,layout=(4,4),sharex=True,sharey=True)
plt.show()
```



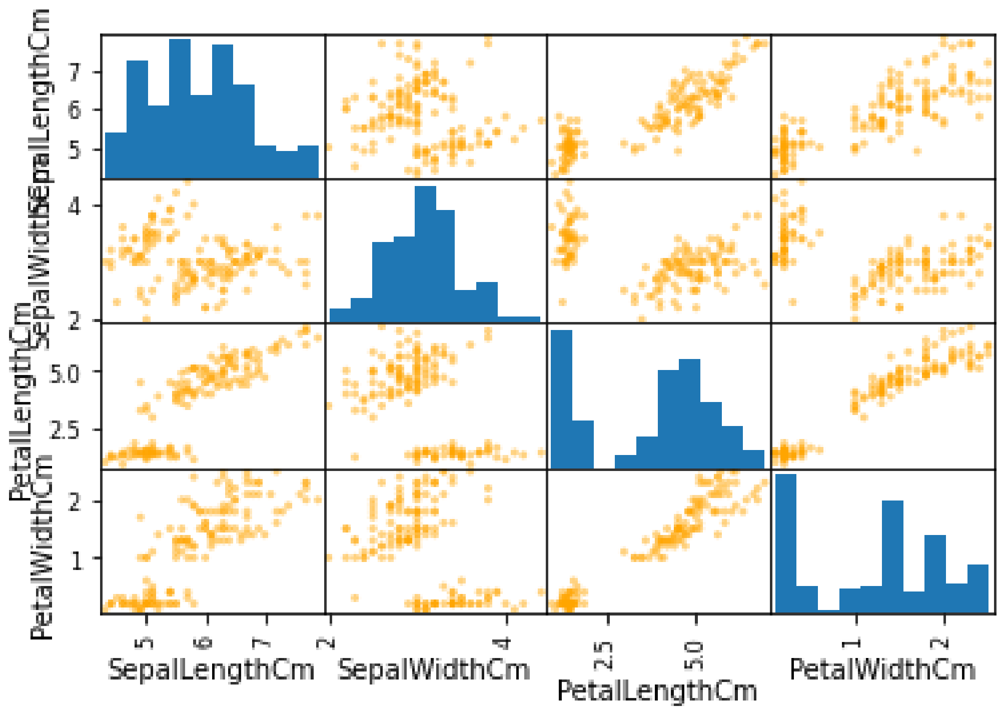
[8]:
```
Iris.plot(kind='box',subplots=True,layout=(2,2),sharex=False,sharey=False)
plt.show()
```



[9]:
```
Iris.hist()
plt.show()
```



[10]:
```
from pandas.plotting import scatter_matrix


scatter_matrix(Iris, color='orange')
plt.show()
```



[11]:
```
Iris.columns
```

[11]:
```
Index(['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm',
       'Species'],
      dtype='object')
```

[12]:
```
group_iris=Iris.groupby('Species')
```

[13]:
```
#for Species, Species_Iris in group_iris:
    #print(Species)
    #print(Species_Iris)
```

[14]:
```
group_iris.mean()
```

[14]:
```
                 SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
Species
```

```
Iris-setosa              5.006         3.418          1.464         0.244
Iris-versicolor          5.936         2.770          4.260         1.326
Iris-virginica           6.588         2.974          5.552         2.026
```

[15]:
```
group_iris.std()
```

[15]:
```
                 SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
Species
Iris-setosa           0.352490      0.381024       0.173511      0.107210
Iris-versicolor       0.516171      0.313798       0.469911      0.197753
Iris-virginica        0.635880      0.322497       0.551895      0.274650
```

[16]:
```
group_iris.count()
```

[16]:
```
                 SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
Species
Iris-setosa                 50            50             50            50
Iris-versicolor             50            50             50            50
Iris-virginica              50            50             50            50
```

[17]:
```
group_iris.max(),group_iris.min()
```

[17]:
```
(                 SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
 Species
 Iris-setosa                5.8           4.4            1.9           0.6
 Iris-versicolor            7.0           3.4            5.1           1.8
 Iris-virginica             7.9           3.8            6.9           2.5,
                  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
 Species
 Iris-setosa                4.3           2.3            1.0           0.1
 Iris-versicolor            4.9           2.0            3.0           1.0
 Iris-virginica             4.9           2.2            4.5           1.4)
```

[18]:
```
group_iris.get_group('Iris-setosa').head()
```

[18]:
```
   SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
0            5.1           3.5            1.4           0.2
1            4.9           3.0            1.4           0.2
2            4.7           3.2            1.3           0.2
3            4.6           3.1            1.5           0.2
4            5.0           3.6            1.4           0.2
```

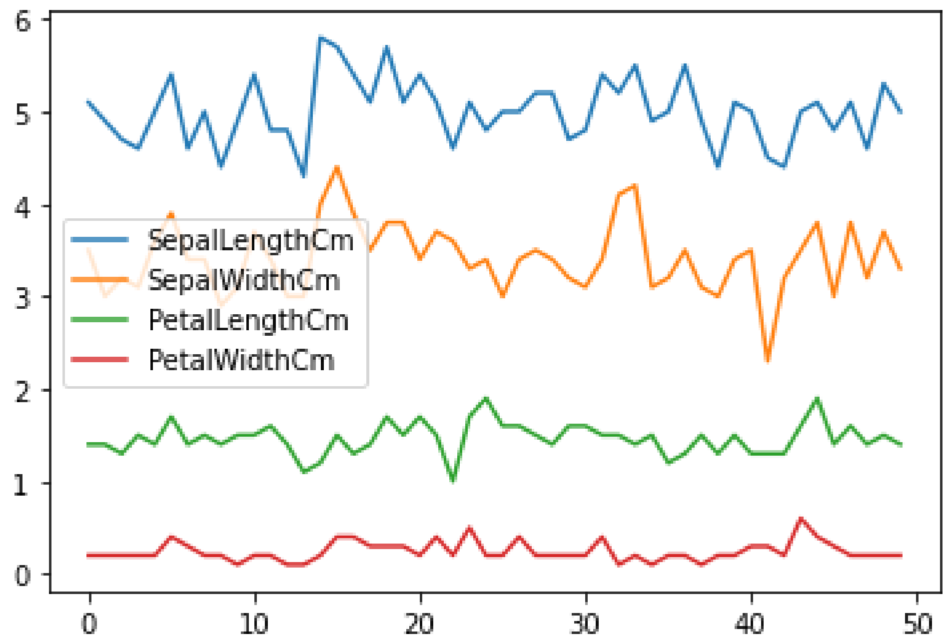
[19]:
```
group_iris.get_group('Iris-setosa').describe()
```

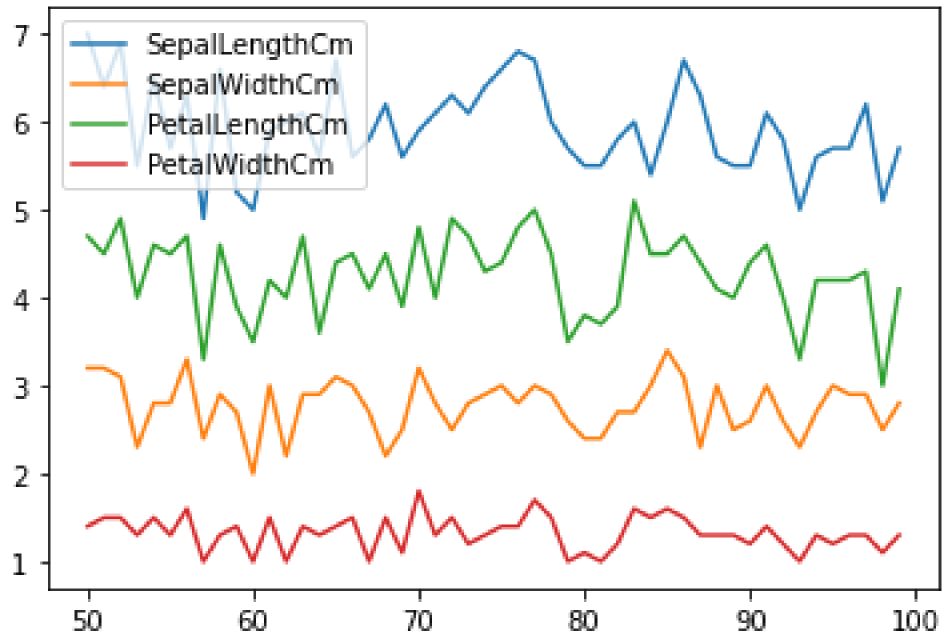
[19]:
```
       SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
count       50.00000     50.000000      50.000000      50.00000
mean         5.00600      3.418000       1.464000       0.24400
```

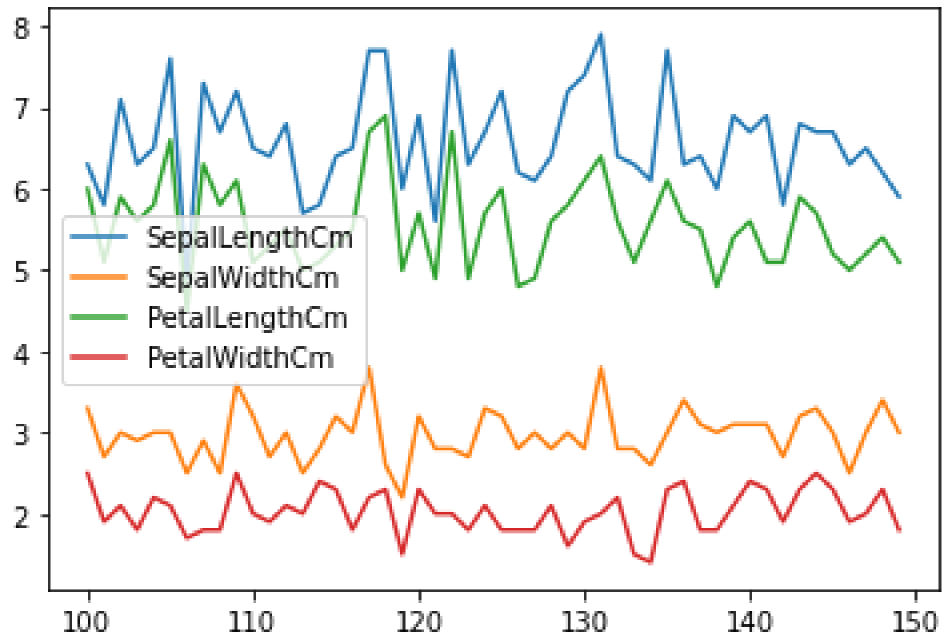
```
std          0.35249    0.381024    0.173511    0.10721
min          4.30000    2.300000    1.000000    0.10000
25%          4.80000    3.125000    1.400000    0.20000
50%          5.00000    3.400000    1.500000    0.20000
75%          5.20000    3.675000    1.575000    0.30000
max          5.80000    4.400000    1.900000    0.60000
```

[20]:
```
group_iris.plot()
```

[20]:
```
Species
Iris-setosa        AxesSubplot(0.125,0.125;0.775x0.755)
Iris-versicolor    AxesSubplot(0.125,0.125;0.775x0.755)
Iris-virginica     AxesSubplot(0.125,0.125;0.775x0.755)
dtype: object
```




SepalLengthCm
SepalWidthCm
PetalLengthCm
PetalWidthCm
1
2
3
4
5
6
7
50
60
70
80
90
100


SepalLengthCm
SepalWidthCm
PetalLengthCm
PetalWidthCm
2
3
4
5
6
7
8
100
110
120
130
140
150

0.3 Data ANALYSIS USING SEARBON LIBRARY

[21]:
```
#Importing and reading data.
data = pd.read_csv("Titanic.csv", sep=";")
# Since our data is seperated by semicolons we need to do sep=";"
```

[22]:
```
import numpy as np
import seaborn as sns
```

[23]:
```
Iris.head()
```

[23]:

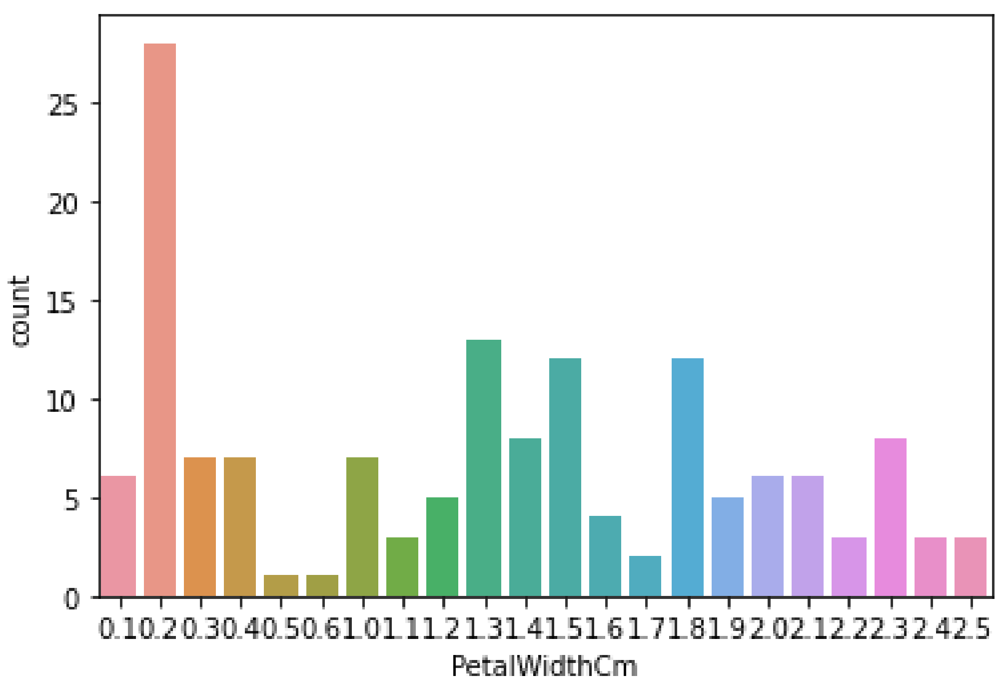
	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

[24]:
```
Iris.columns
```

[24]:
```
Index(['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm',
       'Species'],
      dtype='object')
```

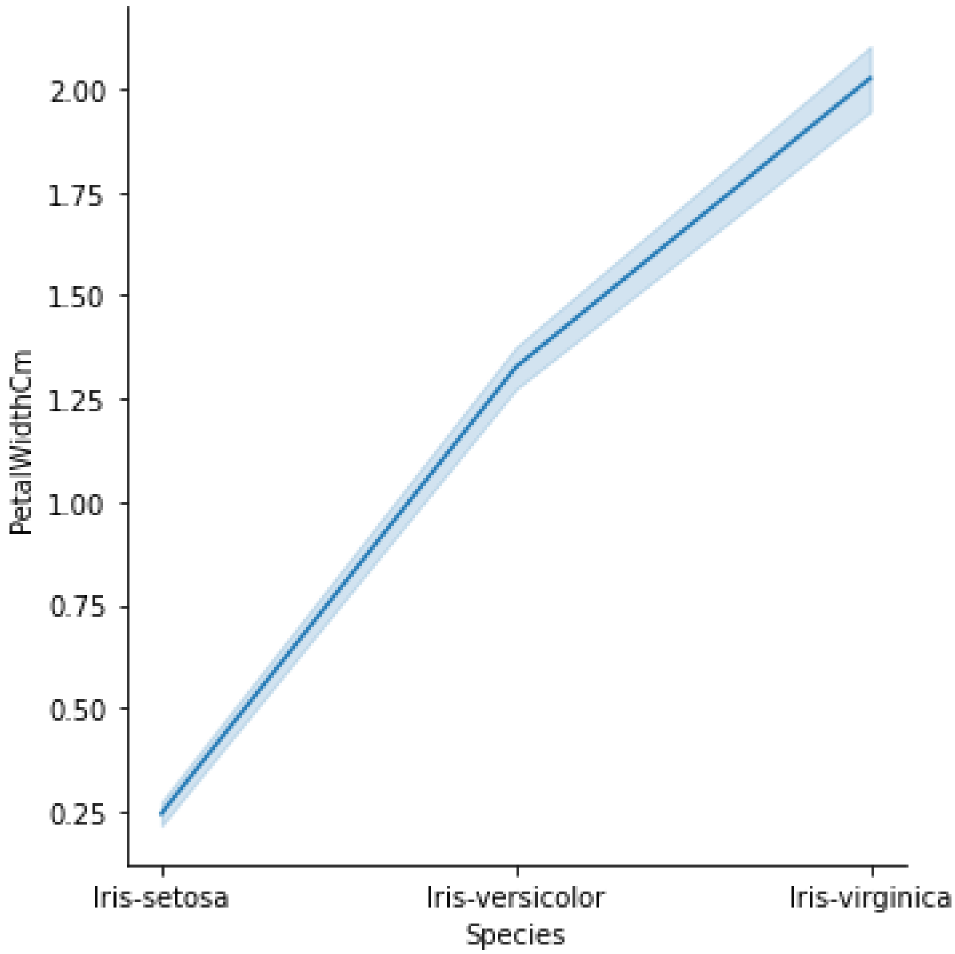
[25]:
```
sns.countplot(x="PetalWidthCm", data=Iris)
```

[25]:
```
<AxesSubplot:xlabel='PetalWidthCm', ylabel='count'>
```



[26]:
```
sns.relplot(x="Species", y="PetalWidthCm", data=Iris, kind="line")
```

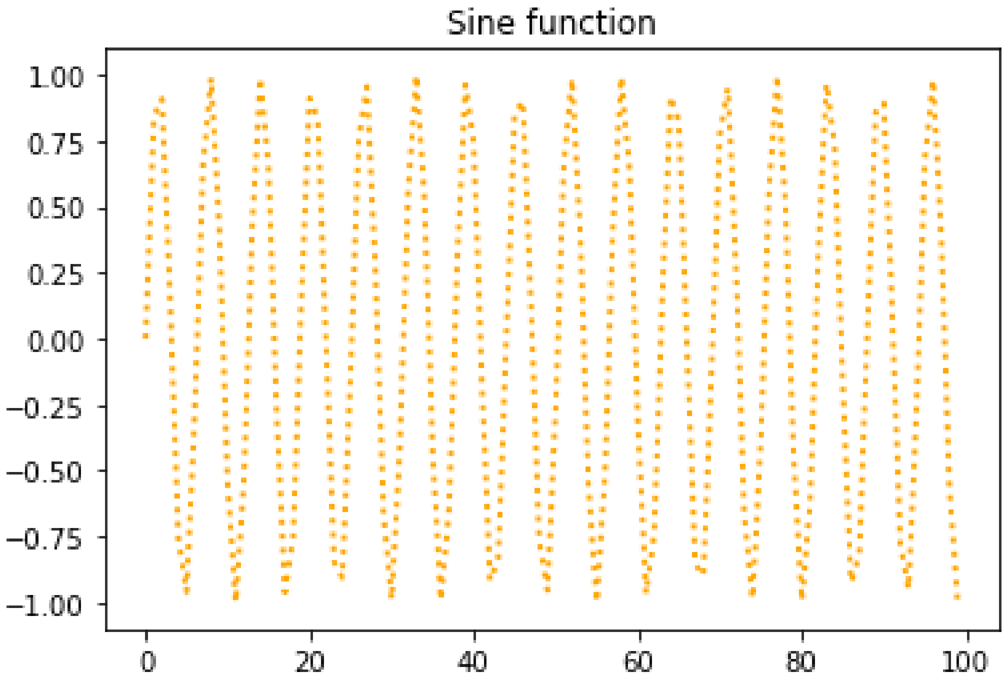
[26]: <seaborn.axisgrid.FacetGrid at 0x13c9912e0>



0.4 More on plotting

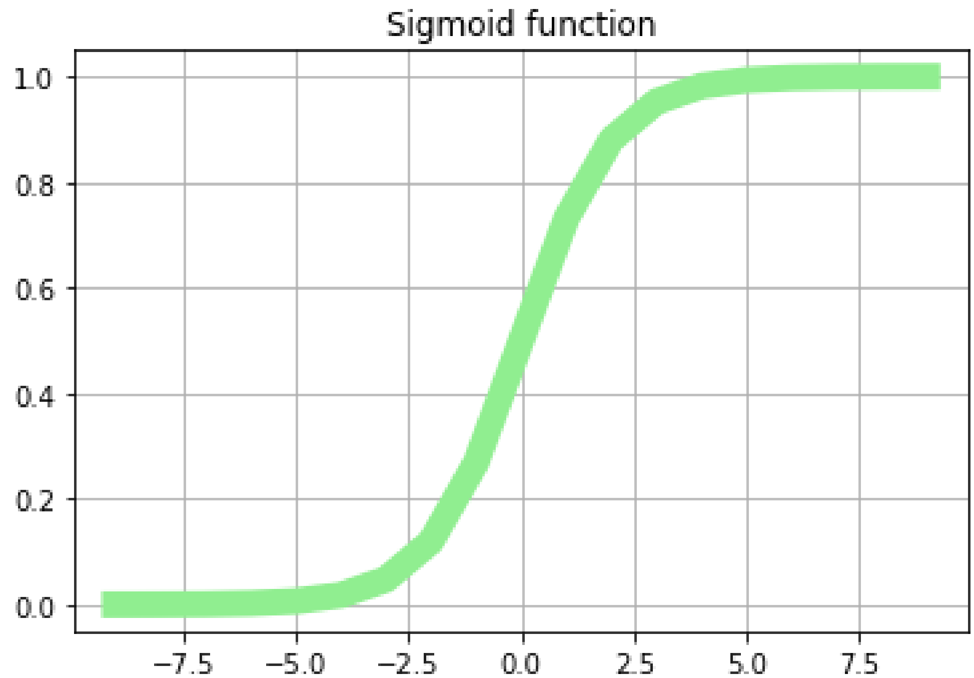
[27]:
```
import math

X=[i for i in range(100)]
y1=[math.sin(X[i]) for i in range(100)]
plt.plot(X,y1,color='orange',linewidth=2,linestyle=':')
plt.title('Sine function')
plt.show
```

[27]: <function matplotlib.pyplot.show(close=None, block=None)>



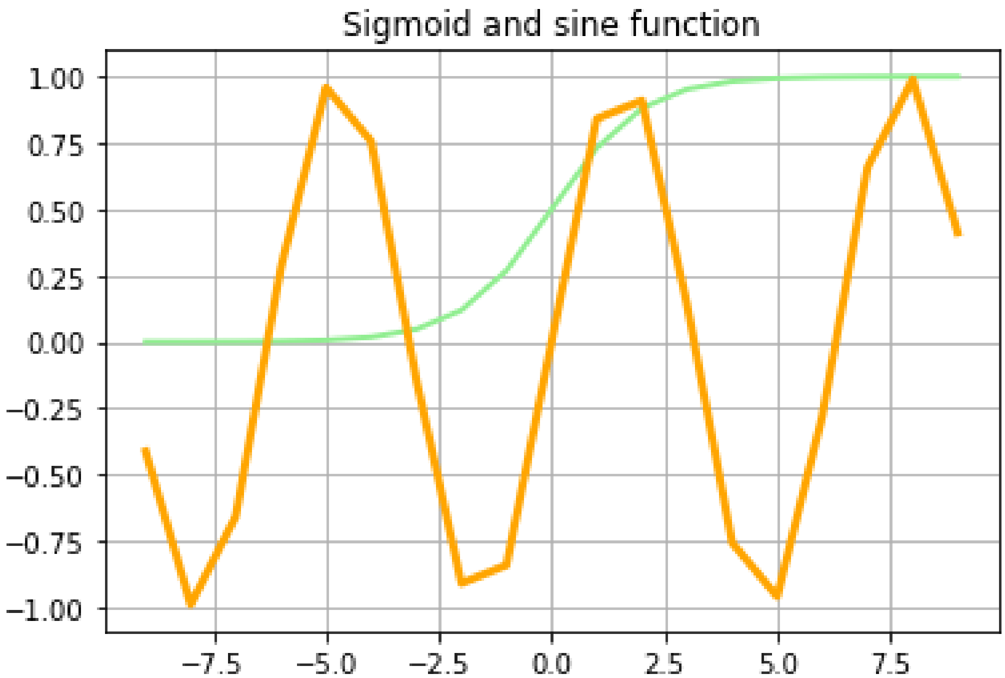
[28]:
```
X1=[-i for i in range(10)]
X2=[i for i in range(10)]
X3=sorted(X1)+X2

y2=[1/(1+math.exp(-x)) for x in X3]
plt.plot(X3,y2,color='lightgreen',linewidth=10)
plt.title('Sigmoid function')
plt.grid(True)
plt.show
```

[28]: <function matplotlib.pyplot.show(close=None, block=None)>



[29]:
```
X1=[-i for i in range(10)]
X2=[i for i in range(10)]
X3=sorted(X1)+X2
y1=[math.sin(x) for x in X3]
y2=[1/(1+math.exp(-x)) for x in X3]
plt.plot(X3,y2,color='lightgreen',linewidth=2)
plt.plot(X3,y1,color='orange',linewidth=3)
plt.title('Sigmoid and sine function')
plt.grid(True)
plt.show
```

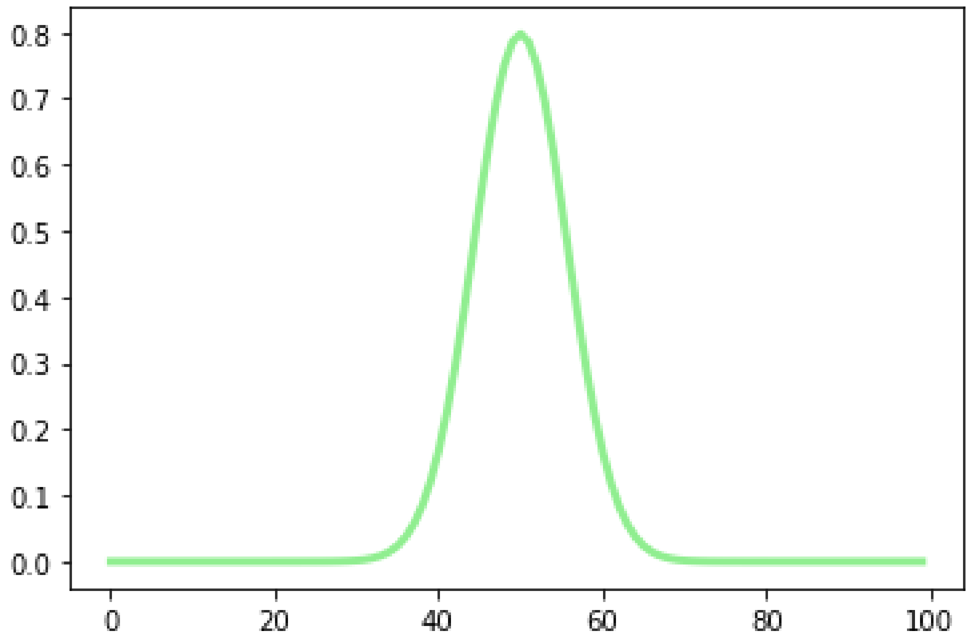
[29]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



0.5 Plotting Normal distribution

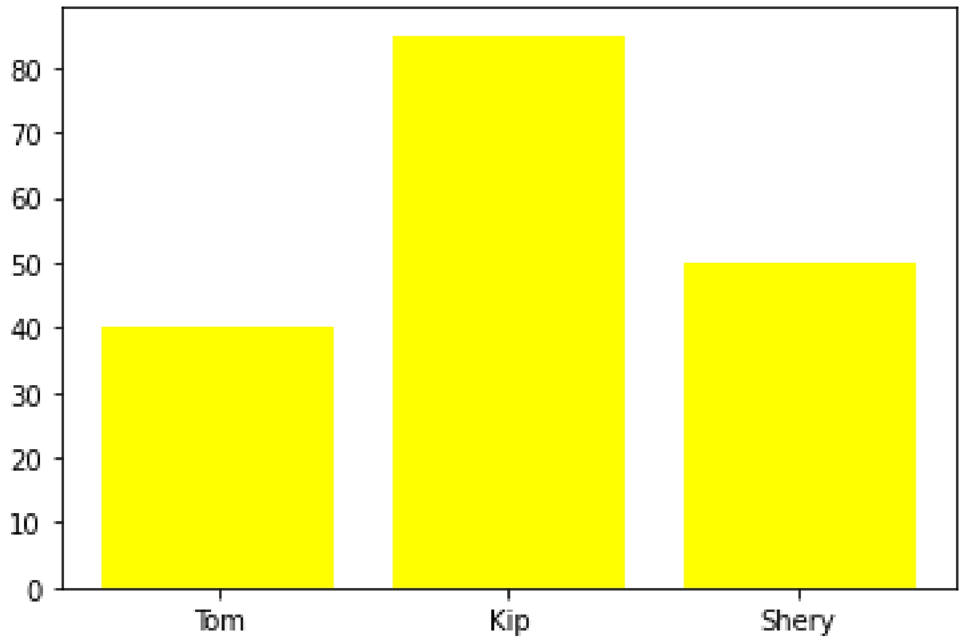
[30]:
```
d_p=[i for i in range(100)]
mu=50
delta=0.5

df=[(1/(delta*math.sqrt(2*math.pi)))*math.exp(-((d_p[i]-mu)/2*delta**2)**2) for␣
↪i in range(100)]

plt.plot(d_p,df,color='lightgreen',linewidth=3)
plt.show
```

[30]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



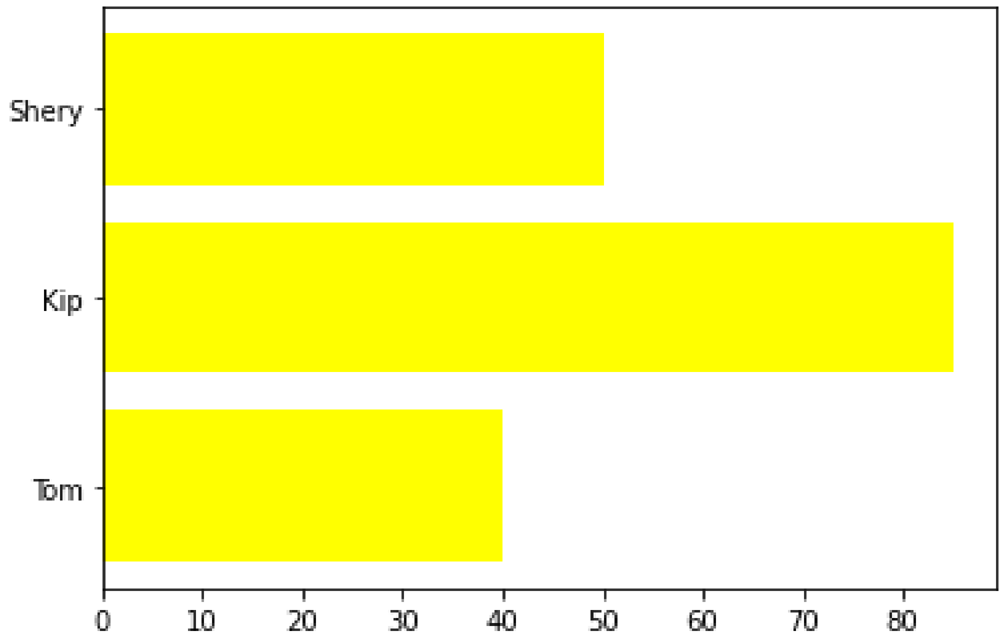
[31]:
```
#Bar plot
Students={"Tom":40,"Kip":85,"Shery":50}
Names=list(Students.keys())
Marks=list(Students.values())
plt.bar(Names,Marks,color='yellow')
plt.show
```

[31]: <function matplotlib.pyplot.show(close=None, block=None)>



[32]:
```
#Bar plot
Students={"Tom":40,"Kip":85,"Shery":50}
Names=list(Students.keys())
Marks=list(Students.values())
plt.barh(Names,Marks,color='yellow')
plt.show
```

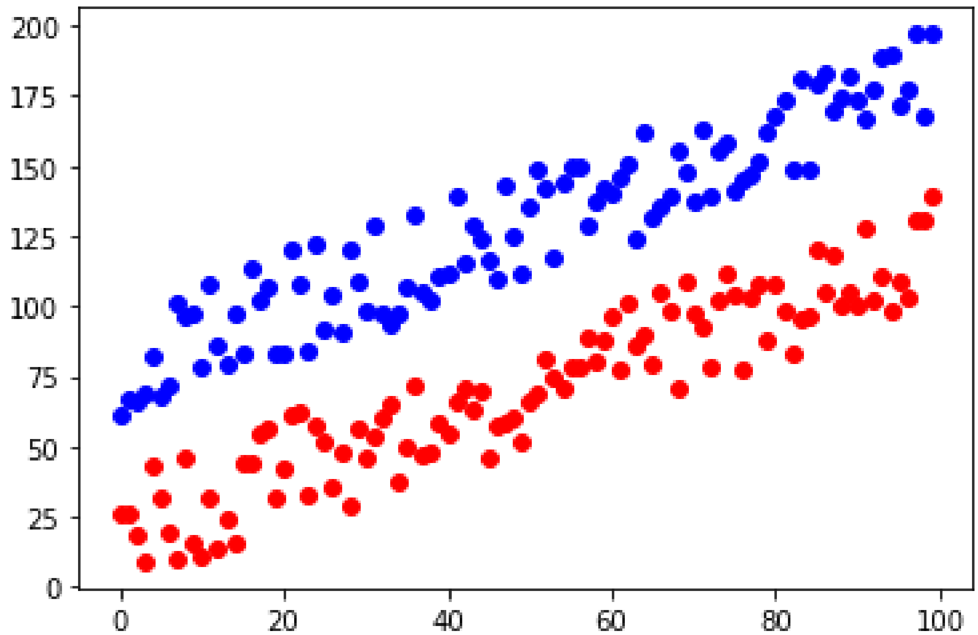
[32]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



0.6 Scatter plot

[33]:
```
import random
def rand():
    rand=random.randint(1,40)
    return rand

x=[i for i in range(100)]
y=[i+rand() for i in range(len(x))]
Z=[60+i+rand() for i in range(len(x))]

plt.scatter(x,y,color='red')
plt.scatter(x,Z,color='Blue')
plt.show
```

[33]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



0.7 Classifying Iris Data using Random Forest

[34]:
```
# first neural network with keras tutorial
import pandas as pd
import numpy as np
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import seaborn as sn
from numpy import loadtxt
from keras.models import Sequential
from keras.layers import Dense
```

[56]:
```
#Importing and reading data.
Iris= pd.read_csv("Iris.csv")
# Since our data is seperated by semicolons we need to do sep=";"
Iris=Iris.iloc[:,[1,2,3,4,5]]
Iris.head()
```

[56]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

[66]:
```
X=Iris.iloc[:,[1,2,3]]

y=Iris.iloc[:,4]
Xn=X+np.random.normal(0,10,X.shape)
Xn
```

[66]:
```
     SepalWidthCm  PetalLengthCm  PetalWidthCm
0        8.037819     -16.897404      0.570057
1       10.679024       7.298798     -3.438588
2       -4.856265      -9.883119     -1.110540
3       14.430799     -18.018041     -6.398917
4       -7.798025       9.249575     -5.343096
..            …              …             …
145      5.014799      10.607736    -15.880776
146      2.006759       7.390336     -8.103303
147     19.739857       6.815593     17.634047
148     -4.505230      -3.673001      4.542522
149    -13.786884       7.249656      2.772192

[150 rows x 3 columns]
```

[36]:
```
from sklearn.ensemble import RandomForestClassifier
np.random.seed(0)
```

[37]:
```
Iris['Is_train']=np.random.uniform(0,1,len(Iris))<=0.75
Iris.head()
```

[37]:
```
   SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm      Species  \
0            5.1           3.5            1.4           0.2  Iris-setosa
1            4.9           3.0            1.4           0.2  Iris-setosa
2            4.7           3.2            1.3           0.2  Iris-setosa
3            4.6           3.1            1.5           0.2  Iris-setosa
4            5.0           3.6            1.4           0.2  Iris-setosa

   Is_train
0      True
1      True
2      True
3      True
4      True
```

[38]:
```
train,test=Iris[Iris['Is_train']==True],Iris[Iris['Is_train']==False]
```

[39]:
```
len(train), len(test)
```

[39]: (118, 32)

[40]:
```
X_train=train.iloc[:,[2,3,4,5,6,7,8,9,10,11,12,13]]
y_train=train.iloc[:,15]
X_test=test.iloc[:,[2,3,4,5,6,7,8,9,10,11,12,13]]
y_test=test.iloc[:,15]

X_train.head()
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-40-6a7f62e08fc7> in <module>
----> 1 X_train=train.iloc[:,[2,3,4,5,6,7,8,9,10,11,12,13]]
      2 y_train=train.iloc[:,15]
      3 X_test=test.iloc[:,[2,3,4,5,6,7,8,9,10,11,12,13]]
      4 y_test=test.iloc[:,15]
      5

/opt/anaconda3/envs/Project/lib/python3.8/site-packages/pandas/core/indexing.py
 ↪in __getitem__(self, key)
    871                 # AttributeError for IntervalTree get_value
    872                 pass
--> 873             return self._getitem_tuple(key)
    874         else:
    875             # we by definition only have the 0th axis

/opt/anaconda3/envs/Project/lib/python3.8/site-packages/pandas/core/indexing.py
 ↪in _getitem_tuple(self, tup)
   1441     def _getitem_tuple(self, tup: Tuple):
   1442
-> 1443         self._has_valid_tuple(tup)
   1444         try:
   1445             return self._getitem_lowerdim(tup)

/opt/anaconda3/envs/Project/lib/python3.8/site-packages/pandas/core/indexing.py
 ↪in _has_valid_tuple(self, key)
    700                 raise IndexingError("Too many indexers")
    701             try:
--> 702                 self._validate_key(k, i)
    703             except ValueError as err:
    704                 raise ValueError(

/opt/anaconda3/envs/Project/lib/python3.8/site-packages/pandas/core/indexing.py
 ↪in _validate_key(self, key, axis)
   1365             # check that the key does not exceed the maximum size of the␣
 ↪index
   1366             if len(arr) and (arr.max() >= len_axis or arr.min() <␣
 ↪-len_axis):
-> 1367                 raise IndexError("positional indexers are out-of-bounds")
```

```
   1368         else:
   1369             raise ValueError(f"Can only index by location with a [{self
 ↪_valid_types}]")

IndexError: positional indexers are out-of-bounds
```

[63]:
```
y_train=pd.factorize(train['Species'])[0]
y_test=pd.factorize(test['Species'])[0]

y_train
```

[63]:
```
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, 0, 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, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2])
```

[64]:
```
## Creating a model
model=RandomForestClassifier(n_jobs=2,random_state=0)
model.fit(X_train,y_train)
```

[64]:
```
RandomForestClassifier(n_jobs=2, random_state=0)
```

[65]:
```
predictions=model.predict(X_test)
score=accuracy_score(y_test,predictions)
score
```

[65]:
```
0.9375
```

[66]:
```
pd.crosstab(test['Species'],predictions,rownames=['Actual␣
 ↪Species'],colnames=['predicted Species'])
```

[66]:

predicted Species	0	1	2
Actual Species			
Iris-setosa	13	0	0
Iris-versicolor	0	5	2
Iris-virginica	0	0	12

1 Deep Learning

[70]:
```
import tensorflow as tf

import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
```

[71]:
```
train_in=[[1,1,1],[1,0,1],[0,1,1],[0,0,1]]
train_out=[[1],[0],[0],[0]]
```

[72]:
```
# Defining weights of the model
w=tf.Variable(tf.random.normal([3,1], seed=12))
```

[73]:
```
# Place holders
x=tf.placeholder(tf.float32,[None,3])
x=tf.placeholder(tf.float32,[None,1])
```

[74]:
```
output=tf.nn.relu(tf.matmul(w,x))
```

[75]:
```
#Loss Function
loss=tf.reduce_sum(tf.square(output-y))
```

[76]:
```
# Optimizer
optimizer=tf.train.GradientDescentOptimizer(0.01)
train=optimizer.minimize(loss)
```

[79]:
```
#Initizlizing my variables
init=tf.global_variables_initializer()
sess = tf.compat.v1.Session()
sess.run(init)
```

1.1 A simple deep learning Example

1. Load Data.

The first step is to define the functions and classes we intend to use in this tutorial.

We will use the NumPy library to load our dataset and we will use two classes from the Keras library to define our model.

The imports required are listed below.

[80]:
```
# first neural network with keras tutorial
import random
from numpy import loadtxt
from keras.models import Sequential
from keras.layers import Dense
```

[81]:
```
Numb1=[random.randint(20,25) for i in range(1000)]
Numb2=[random.randint(45,50) for i in range(1000)]
Numb3=[random.randint(67,70) for i in range(1000)]
Numb4=[random.randint(0,1) for i in range(1000)]
```

[82]:
```
import numpy as np

Numb=[[Numb1[i],Numb2[i],Numb3[i],Numb4[i]]  for i in range(1000)]
Mydata=np.array(Numb)
```

We can now load our dataset.

In this Keras tutorial, we are going to use the Pima Indians onset of diabetes dataset. This is a standard machine learning dataset from the UCI Machine Learning repository. It describes patient medical record data for Pima Indians and whether they had an onset of diabetes within five years.

As such, it is a binary classification problem (onset of diabetes as 1 or not as 0). All of the input variables that describe each patient are numerical. This makes it easy to use directly with neural networks that expect numerical input and output values, and ideal for our first neural network in Keras.

[83]:
```
# load the dataset
dataset = loadtxt('pima-indians-diabetes.csv', delimiter=',')
#dataset =Mydata
# split into input (X) and output (y) variables
X = dataset[:,0:8]
y = dataset[:,8]
dataset
```

[83]:
```
array([[  6.   , 148.   ,  72.   , …,   0.627,  50.   ,   1.   ],
       [  1.   ,  85.   ,  66.   , …,   0.351,  31.   ,   0.   ],
       [  8.   , 183.   ,  64.   , …,   0.672,  32.   ,   1.   ],
       …,
       [  5.   , 121.   ,  72.   , …,   0.245,  30.   ,   0.   ],
       [  1.   , 126.   ,  60.   , …,   0.349,  47.   ,   1.   ],
       [  1.   ,  93.   ,  70.   , …,   0.315,  23.   ,   0.   ]])
```

2. Define Keras Model

Models in Keras are defined as a sequence of layers.

We create a Sequential model and add layers one at a time until we are happy with our network architecture.

The first thing to get right is to ensure the input layer has the right number of input features. This can be specified when creating the first layer with the input_dim argument and setting it to 8 for the 8 input variables.

How do we know the number of layers and their types?

This is a very hard question. There are heuristics that we can use and often the best network structure is found through a process of trial and error experimentation (I explain more about this here). Generally, you need a network large enough to capture the structure of the problem.

In this example, we will use a fully-connected network structure with three layers.

Fully connected layers are defined using the Dense class. We can specify the number of neurons or nodes in the layer as the first argument, and specify the activation function using the activation argument.

We will use the rectified linear unit activation function referred to as ReLU on the first two layers and the Sigmoid function in the output layer.

It used to be the case that Sigmoid and Tanh activation functions were preferred for all layers. These days, better performance is achieved using the ReLU activation function. We use a sigmoid on the output layer to ensure our network output is between 0 and 1 and easy to map to either a probability of class 1 or snap to a hard classification of either class with a default threshold of 0.5.

We can piece it all together by adding each layer:

The model expects rows of data with 8 variables (the input_dim=8 argument).

The first hidden layer has 12 nodes and uses the relu activation function.

The second hidden layer has 8 nodes and uses the relu activation function.

The output layer has one node and uses the sigmoid activation function.

[84]:
```
# define the keras model
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

3. Compile Keras Model.

Now that the model is defined, we can compile it.

Compiling the model uses the efficient numerical libraries under the covers (the so-called backend) such as Theano or TensorFlow. The backend automatically chooses the best way to represent the network for training and making predictions to run on your hardware, such as CPU or GPU or even distributed.

When compiling, we must specify some additional properties required when training the network. Remember training a network means finding the best set of weights to map inputs to outputs in our dataset.

We must specify the loss function to use to evaluate a set of weights, the optimizer is used to search through different weights for the network and any optional metrics we would like to collect and report during training.

In this case, we will use cross entropy as the loss argument. This loss is for a binary classification problems and is defined in Keras as "binary_crossentropy". You can learn more about choosing loss functions based on your problem here:

How to Choose Loss Functions When Training Deep Learning Neural Networks We will define the optimizer as the efficient stochastic gradient descent algorithm "adam". This is a popular version of gradient descent because it automatically tunes itself and gives good results in a wide range of problems. To learn more about the Adam version of stochastic gradient descent see the post:

Gentle Introduction to the Adam Optimization Algorithm for Deep Learning Finally, because it is a classification problem, we will collect and report the classification accuracy, defined via the metrics argument.

[85]:
```
# compile the keras model
model.compile(loss='binary_crossentropy', optimizer='Adam',␣
↪metrics=['accuracy'])
```

4. Fit Keras Model.

We have defined our model and compiled it ready for efficient computation.

Now it is time to execute the model on some data.

We can train or fit our model on our loaded data by calling the fit() function on the model.

Training occurs over epochs and each epoch is split into batches.

Epoch: One pass through all of the rows in the training dataset. Batch: One or more samples considered by the model within an epoch before weights are updated. One epoch is comprised of one or more batches, based on the chosen batch size and the model is fit for many epochs. For more on the difference between epochs and batches, see the post:

What is the Difference Between a Batch and an Epoch in a Neural Network? The training process will run for a fixed number of iterations through the dataset called epochs, that we must specify using the epochs argument. We must also set the number of dataset rows that are considered before the model weights are updated within each epoch, called the batch size and set using the batch_size argument.

For this problem, we will run for a small number of epochs (150) and use a relatively small batch size of 10.

These configurations can be chosen experimentally by trial and error. We want to train the model enough so that it learns a good (or good enough) mapping of rows of input data to the output classification. The model will always have some error, but the amount of error will level out after some point for a given model configuration. This is called model convergence.

[86]:
```
# fit the keras model on the dataset
model=model.fit(X, y, validation_split=0.33, epochs=150, batch_size=10)
```

```
Train on 514 samples, validate on 254 samples
Epoch 1/150
514/514 [==============================] - 0s 591us/sample - loss: 2.2004 - acc:
0.5486 - val_loss: 1.0577 - val_acc: 0.5433
Epoch 2/150
514/514 [==============================] - 0s 204us/sample - loss: 1.1585 - acc:
0.5759 - val_loss: 0.8943 - val_acc: 0.6693
Epoch 3/150
 10/514 […] - ETA: 0s - loss: 2.9211 - acc: 0.2000
```

```
/opt/anaconda3/envs/Project/lib/python3.8/site-
packages/tensorflow/python/keras/engine/training.py:2325: UserWarning:
`Model.state_updates` will be removed in a future version. This property should
not be used in TensorFlow 2.0, as `updates` are applied automatically.
  warnings.warn('`Model.state_updates` will be removed in a future version. '
```

```
514/514 [==============================] - 0s 195us/sample - loss: 1.0649 - acc:
0.5895 - val_loss: 0.8646 - val_acc: 0.7087
Epoch 4/150
514/514 [==============================] - 0s 156us/sample - loss: 1.0220 - acc:
0.6051 - val_loss: 0.8051 - val_acc: 0.6811
Epoch 5/150
514/514 [==============================] - 0s 170us/sample - loss: 0.9155 - acc:
0.6265 - val_loss: 0.7440 - val_acc: 0.6496
Epoch 6/150
514/514 [==============================] - 0s 163us/sample - loss: 0.8771 - acc:
0.6245 - val_loss: 0.7406 - val_acc: 0.6220
Epoch 7/150
514/514 [==============================] - 0s 183us/sample - loss: 0.8228 - acc:
0.6245 - val_loss: 0.7125 - val_acc: 0.6417
Epoch 8/150
514/514 [==============================] - 0s 169us/sample - loss: 0.7930 - acc:
0.6304 - val_loss: 0.7044 - val_acc: 0.6299
Epoch 9/150
514/514 [==============================] - 0s 188us/sample - loss: 0.7408 - acc:
0.6420 - val_loss: 0.6954 - val_acc: 0.6496
Epoch 10/150
514/514 [==============================] - 0s 204us/sample - loss: 0.7279 - acc:
0.6576 - val_loss: 0.6750 - val_acc: 0.6732
Epoch 11/150
514/514 [==============================] - 0s 172us/sample - loss: 0.7108 - acc:
0.6556 - val_loss: 0.8102 - val_acc: 0.5433
Epoch 12/150
514/514 [==============================] - 0s 208us/sample - loss: 0.6775 - acc:
0.6615 - val_loss: 0.6969 - val_acc: 0.6299
Epoch 13/150
514/514 [==============================] - 0s 181us/sample - loss: 0.6905 - acc:
0.6576 - val_loss: 0.7374 - val_acc: 0.7008
Epoch 14/150
514/514 [==============================] - 0s 201us/sample - loss: 0.6882 - acc:
0.6556 - val_loss: 0.7242 - val_acc: 0.6850
Epoch 15/150
514/514 [==============================] - 0s 184us/sample - loss: 0.6619 - acc:
0.6537 - val_loss: 0.6847 - val_acc: 0.7047
Epoch 16/150
514/514 [==============================] - 0s 207us/sample - loss: 0.6565 - acc:
0.6556 - val_loss: 0.6735 - val_acc: 0.6496
Epoch 17/150
514/514 [==============================] - 0s 232us/sample - loss: 0.6427 - acc:
0.6693 - val_loss: 0.7166 - val_acc: 0.6102
Epoch 18/150
514/514 [==============================] - 0s 182us/sample - loss: 0.6345 - acc:
0.6654 - val_loss: 0.7236 - val_acc: 0.6220
Epoch 19/150
```

```
514/514 [==============================] - 0s 186us/sample - loss: 0.6537 - acc:
0.6518 - val_loss: 0.6892 - val_acc: 0.6220
Epoch 20/150
514/514 [==============================] - 0s 205us/sample - loss: 0.6273 - acc:
0.6751 - val_loss: 0.6652 - val_acc: 0.6654
Epoch 21/150
514/514 [==============================] - 0s 172us/sample - loss: 0.6389 - acc:
0.6848 - val_loss: 0.6506 - val_acc: 0.6693
Epoch 22/150
514/514 [==============================] - 0s 184us/sample - loss: 0.6316 - acc:
0.6809 - val_loss: 0.6967 - val_acc: 0.6063
Epoch 23/150
514/514 [==============================] - 0s 176us/sample - loss: 0.6426 - acc:
0.6809 - val_loss: 0.6495 - val_acc: 0.6693
Epoch 24/150
514/514 [==============================] - 0s 224us/sample - loss: 0.6498 - acc:
0.6946 - val_loss: 0.6454 - val_acc: 0.6929
Epoch 25/150
514/514 [==============================] - 0s 166us/sample - loss: 0.6036 - acc:
0.6790 - val_loss: 0.6547 - val_acc: 0.6890
Epoch 26/150
514/514 [==============================] - 0s 175us/sample - loss: 0.6061 - acc:
0.6829 - val_loss: 0.6685 - val_acc: 0.6535
Epoch 27/150
514/514 [==============================] - 0s 173us/sample - loss: 0.6008 - acc:
0.7023 - val_loss: 0.7377 - val_acc: 0.6181
Epoch 28/150
514/514 [==============================] - 0s 152us/sample - loss: 0.6132 - acc:
0.6868 - val_loss: 0.6515 - val_acc: 0.6417
Epoch 29/150
514/514 [==============================] - 0s 158us/sample - loss: 0.6153 - acc:
0.6732 - val_loss: 0.6463 - val_acc: 0.6614
Epoch 30/150
514/514 [==============================] - 0s 152us/sample - loss: 0.6078 - acc:
0.6809 - val_loss: 0.6665 - val_acc: 0.6378
Epoch 31/150
514/514 [==============================] - 0s 182us/sample - loss: 0.5980 - acc:
0.6946 - val_loss: 0.6589 - val_acc: 0.6732
Epoch 32/150
514/514 [==============================] - 0s 192us/sample - loss: 0.5954 - acc:
0.6926 - val_loss: 0.6292 - val_acc: 0.7087
Epoch 33/150
514/514 [==============================] - 0s 175us/sample - loss: 0.5839 - acc:
0.7179 - val_loss: 0.6627 - val_acc: 0.6535
Epoch 34/150
514/514 [==============================] - 0s 181us/sample - loss: 0.5927 - acc:
0.7023 - val_loss: 0.6384 - val_acc: 0.6496
Epoch 35/150
```

```
514/514 [==============================] - 0s 198us/sample - loss: 0.6434 - acc:
0.6712 - val_loss: 0.7541 - val_acc: 0.5394
Epoch 36/150
514/514 [==============================] - 0s 165us/sample - loss: 0.6006 - acc:
0.6926 - val_loss: 0.6395 - val_acc: 0.6654
Epoch 37/150
514/514 [==============================] - 0s 203us/sample - loss: 0.5918 - acc:
0.7004 - val_loss: 0.6405 - val_acc: 0.6575
Epoch 38/150
514/514 [==============================] - 0s 171us/sample - loss: 0.5882 - acc:
0.6984 - val_loss: 0.7111 - val_acc: 0.5984
Epoch 39/150
514/514 [==============================] - 0s 197us/sample - loss: 0.5749 - acc:
0.7121 - val_loss: 0.6336 - val_acc: 0.6654
Epoch 40/150
514/514 [==============================] - 0s 177us/sample - loss: 0.5707 - acc:
0.7160 - val_loss: 0.6142 - val_acc: 0.6772
Epoch 41/150
514/514 [==============================] - 0s 192us/sample - loss: 0.5786 - acc:
0.7101 - val_loss: 0.6218 - val_acc: 0.6614
Epoch 42/150
514/514 [==============================] - 0s 209us/sample - loss: 0.5797 - acc:
0.7101 - val_loss: 0.6138 - val_acc: 0.7205
Epoch 43/150
514/514 [==============================] - 0s 240us/sample - loss: 0.5874 - acc:
0.7140 - val_loss: 0.6815 - val_acc: 0.6969
Epoch 44/150
514/514 [==============================] - 0s 191us/sample - loss: 0.5712 - acc:
0.7140 - val_loss: 0.6399 - val_acc: 0.6654
Epoch 45/150
514/514 [==============================] - 0s 178us/sample - loss: 0.5933 - acc:
0.7043 - val_loss: 0.6062 - val_acc: 0.6732
Epoch 46/150
514/514 [==============================] - 0s 188us/sample - loss: 0.5695 - acc:
0.7237 - val_loss: 0.6038 - val_acc: 0.6850
Epoch 47/150
514/514 [==============================] - 0s 212us/sample - loss: 0.5679 - acc:
0.7198 - val_loss: 0.5970 - val_acc: 0.6929
Epoch 48/150
514/514 [==============================] - 0s 180us/sample - loss: 0.5743 - acc:
0.6984 - val_loss: 0.6542 - val_acc: 0.6181
Epoch 49/150
514/514 [==============================] - 0s 189us/sample - loss: 0.5655 - acc:
0.7354 - val_loss: 0.6047 - val_acc: 0.7205
Epoch 50/150
514/514 [==============================] - 0s 178us/sample - loss: 0.5603 - acc:
0.7121 - val_loss: 0.5995 - val_acc: 0.7126
Epoch 51/150
```

```
514/514 [==============================] - 0s 199us/sample - loss: 0.5761 - acc:
0.7276 - val_loss: 0.6998 - val_acc: 0.6063
Epoch 52/150
514/514 [==============================] - 0s 229us/sample - loss: 0.5607 - acc:
0.7237 - val_loss: 0.6260 - val_acc: 0.6772
Epoch 53/150
514/514 [==============================] - 0s 189us/sample - loss: 0.5570 - acc:
0.7257 - val_loss: 0.5959 - val_acc: 0.7126
Epoch 54/150
514/514 [==============================] - 0s 195us/sample - loss: 0.5629 - acc:
0.6887 - val_loss: 0.6000 - val_acc: 0.6929
Epoch 55/150
514/514 [==============================] - 0s 178us/sample - loss: 0.5758 - acc:
0.7082 - val_loss: 0.6802 - val_acc: 0.6339
Epoch 56/150
514/514 [==============================] - 0s 201us/sample - loss: 0.5774 - acc:
0.7101 - val_loss: 0.6082 - val_acc: 0.6929
Epoch 57/150
514/514 [==============================] - 0s 185us/sample - loss: 0.5681 - acc:
0.7062 - val_loss: 0.5983 - val_acc: 0.6929
Epoch 58/150
514/514 [==============================] - 0s 200us/sample - loss: 0.5483 - acc:
0.7374 - val_loss: 0.5867 - val_acc: 0.7205
Epoch 59/150
514/514 [==============================] - 0s 190us/sample - loss: 0.5477 - acc:
0.7237 - val_loss: 0.6274 - val_acc: 0.6417
Epoch 60/150
514/514 [==============================] - 0s 186us/sample - loss: 0.5640 - acc:
0.7218 - val_loss: 0.5873 - val_acc: 0.7008
Epoch 61/150
514/514 [==============================] - 0s 199us/sample - loss: 0.5602 - acc:
0.7160 - val_loss: 0.6001 - val_acc: 0.7126
Epoch 62/150
514/514 [==============================] - 0s 212us/sample - loss: 0.5604 - acc:
0.7315 - val_loss: 0.5879 - val_acc: 0.7244
Epoch 63/150
514/514 [==============================] - 0s 182us/sample - loss: 0.5591 - acc:
0.7198 - val_loss: 0.5997 - val_acc: 0.7087
Epoch 64/150
514/514 [==============================] - 0s 189us/sample - loss: 0.5502 - acc:
0.7296 - val_loss: 0.6070 - val_acc: 0.6890
Epoch 65/150
514/514 [==============================] - 0s 180us/sample - loss: 0.5507 - acc:
0.7374 - val_loss: 0.6010 - val_acc: 0.7087
Epoch 66/150
514/514 [==============================] - 0s 193us/sample - loss: 0.5521 - acc:
0.7315 - val_loss: 0.5835 - val_acc: 0.7126
Epoch 67/150
```

```
514/514 [==============================] - 0s 185us/sample - loss: 0.5502 - acc:
0.7276 - val_loss: 0.5906 - val_acc: 0.7244
Epoch 68/150
514/514 [==============================] - 0s 226us/sample - loss: 0.5639 - acc:
0.7354 - val_loss: 0.5839 - val_acc: 0.7126
Epoch 69/150
514/514 [==============================] - 0s 169us/sample - loss: 0.5532 - acc:
0.7432 - val_loss: 0.6091 - val_acc: 0.7047
Epoch 70/150
514/514 [==============================] - 0s 185us/sample - loss: 0.5715 - acc:
0.7062 - val_loss: 0.5959 - val_acc: 0.7047
Epoch 71/150
514/514 [==============================] - 0s 172us/sample - loss: 0.5697 - acc:
0.7140 - val_loss: 0.6557 - val_acc: 0.6339
Epoch 72/150
514/514 [==============================] - 0s 184us/sample - loss: 0.5898 - acc:
0.6984 - val_loss: 0.6174 - val_acc: 0.6654
Epoch 73/150
514/514 [==============================] - 0s 183us/sample - loss: 0.5530 - acc:
0.7140 - val_loss: 0.5869 - val_acc: 0.7087
Epoch 74/150
514/514 [==============================] - 0s 191us/sample - loss: 0.5492 - acc:
0.7296 - val_loss: 0.5878 - val_acc: 0.7087
Epoch 75/150
514/514 [==============================] - 0s 189us/sample - loss: 0.5466 - acc:
0.7296 - val_loss: 0.5884 - val_acc: 0.7165
Epoch 76/150
514/514 [==============================] - 0s 185us/sample - loss: 0.5436 - acc:
0.7335 - val_loss: 0.5996 - val_acc: 0.6890
Epoch 77/150
514/514 [==============================] - 0s 178us/sample - loss: 0.5591 - acc:
0.7082 - val_loss: 0.5892 - val_acc: 0.7205
Epoch 78/150
514/514 [==============================] - 0s 186us/sample - loss: 0.5566 - acc:
0.7121 - val_loss: 0.6146 - val_acc: 0.6929
Epoch 79/150
514/514 [==============================] - 0s 189us/sample - loss: 0.5815 - acc:
0.7218 - val_loss: 0.5972 - val_acc: 0.6811
Epoch 80/150
514/514 [==============================] - 0s 185us/sample - loss: 0.5600 - acc:
0.7276 - val_loss: 0.5799 - val_acc: 0.7087
Epoch 81/150
514/514 [==============================] - 0s 195us/sample - loss: 0.5709 - acc:
0.7160 - val_loss: 0.5951 - val_acc: 0.7126
Epoch 82/150
514/514 [==============================] - 0s 170us/sample - loss: 0.5435 - acc:
0.7257 - val_loss: 0.5814 - val_acc: 0.7087
Epoch 83/150
```

```
514/514 [==============================] - 0s 193us/sample - loss: 0.5413 - acc:
0.7335 - val_loss: 0.5822 - val_acc: 0.7244
Epoch 84/150
514/514 [==============================] - 0s 179us/sample - loss: 0.5437 - acc:
0.7393 - val_loss: 0.5802 - val_acc: 0.7323
Epoch 85/150
514/514 [==============================] - 0s 195us/sample - loss: 0.5651 - acc:
0.7043 - val_loss: 0.6082 - val_acc: 0.7047
Epoch 86/150
514/514 [==============================] - 0s 174us/sample - loss: 0.5338 - acc:
0.7432 - val_loss: 0.5917 - val_acc: 0.6929
Epoch 87/150
514/514 [==============================] - 0s 183us/sample - loss: 0.5644 - acc:
0.6984 - val_loss: 0.6777 - val_acc: 0.6890
Epoch 88/150
514/514 [==============================] - 0s 157us/sample - loss: 0.5628 - acc:
0.7121 - val_loss: 0.6051 - val_acc: 0.6850
Epoch 89/150
514/514 [==============================] - 0s 191us/sample - loss: 0.5307 - acc:
0.7549 - val_loss: 0.5824 - val_acc: 0.7126
Epoch 90/150
514/514 [==============================] - 0s 179us/sample - loss: 0.5457 - acc:
0.7412 - val_loss: 0.5905 - val_acc: 0.7047
Epoch 91/150
514/514 [==============================] - 0s 195us/sample - loss: 0.5631 - acc:
0.7315 - val_loss: 0.5917 - val_acc: 0.7126
Epoch 92/150
514/514 [==============================] - 0s 189us/sample - loss: 0.6133 - acc:
0.7082 - val_loss: 0.5835 - val_acc: 0.7008
Epoch 93/150
514/514 [==============================] - 0s 196us/sample - loss: 0.5489 - acc:
0.7315 - val_loss: 0.5896 - val_acc: 0.7205
Epoch 94/150
514/514 [==============================] - 0s 145us/sample - loss: 0.5480 - acc:
0.7393 - val_loss: 0.5762 - val_acc: 0.7323
Epoch 95/150
514/514 [==============================] - 0s 171us/sample - loss: 0.5365 - acc:
0.7374 - val_loss: 0.5829 - val_acc: 0.7244
Epoch 96/150
514/514 [==============================] - 0s 156us/sample - loss: 0.5481 - acc:
0.7257 - val_loss: 0.5762 - val_acc: 0.7126
Epoch 97/150
514/514 [==============================] - 0s 178us/sample - loss: 0.5400 - acc:
0.7393 - val_loss: 0.5745 - val_acc: 0.7362
Epoch 98/150
514/514 [==============================] - 0s 185us/sample - loss: 0.5432 - acc:
0.7296 - val_loss: 0.6065 - val_acc: 0.7008
Epoch 99/150
```

```
514/514 [==============================] - 0s 181us/sample - loss: 0.5401 - acc:
0.7218 - val_loss: 0.5900 - val_acc: 0.7126
Epoch 100/150
514/514 [==============================] - 0s 151us/sample - loss: 0.5461 - acc:
0.7179 - val_loss: 0.5652 - val_acc: 0.7323
Epoch 101/150
514/514 [==============================] - 0s 152us/sample - loss: 0.5322 - acc:
0.7432 - val_loss: 0.5766 - val_acc: 0.7087
Epoch 102/150
514/514 [==============================] - 0s 142us/sample - loss: 0.5273 - acc:
0.7471 - val_loss: 0.5664 - val_acc: 0.7205
Epoch 103/150
514/514 [==============================] - 0s 127us/sample - loss: 0.5304 - acc:
0.7451 - val_loss: 0.5721 - val_acc: 0.7520
Epoch 104/150
514/514 [==============================] - 0s 128us/sample - loss: 0.5333 - acc:
0.7490 - val_loss: 0.6220 - val_acc: 0.6654
Epoch 105/150
514/514 [==============================] - 0s 122us/sample - loss: 0.5365 - acc:
0.7412 - val_loss: 0.6227 - val_acc: 0.7126
Epoch 106/150
514/514 [==============================] - 0s 125us/sample - loss: 0.5971 - acc:
0.7004 - val_loss: 0.6501 - val_acc: 0.6378
Epoch 107/150
514/514 [==============================] - 0s 132us/sample - loss: 0.5438 - acc:
0.7276 - val_loss: 0.5841 - val_acc: 0.7205
Epoch 108/150
514/514 [==============================] - 0s 132us/sample - loss: 0.5312 - acc:
0.7432 - val_loss: 0.5797 - val_acc: 0.7047
Epoch 109/150
514/514 [==============================] - 0s 129us/sample - loss: 0.5319 - acc:
0.7374 - val_loss: 0.5709 - val_acc: 0.7283
Epoch 110/150
514/514 [==============================] - 0s 126us/sample - loss: 0.5328 - acc:
0.7510 - val_loss: 0.6029 - val_acc: 0.7126
Epoch 111/150
514/514 [==============================] - 0s 126us/sample - loss: 0.5325 - acc:
0.7374 - val_loss: 0.5637 - val_acc: 0.7362
Epoch 112/150
514/514 [==============================] - 0s 122us/sample - loss: 0.5488 - acc:
0.7354 - val_loss: 0.5833 - val_acc: 0.7244
Epoch 113/150
514/514 [==============================] - 0s 124us/sample - loss: 0.5249 - acc:
0.7393 - val_loss: 0.5696 - val_acc: 0.7283
Epoch 114/150
514/514 [==============================] - 0s 127us/sample - loss: 0.5330 - acc:
0.7374 - val_loss: 0.5745 - val_acc: 0.7205
Epoch 115/150
```

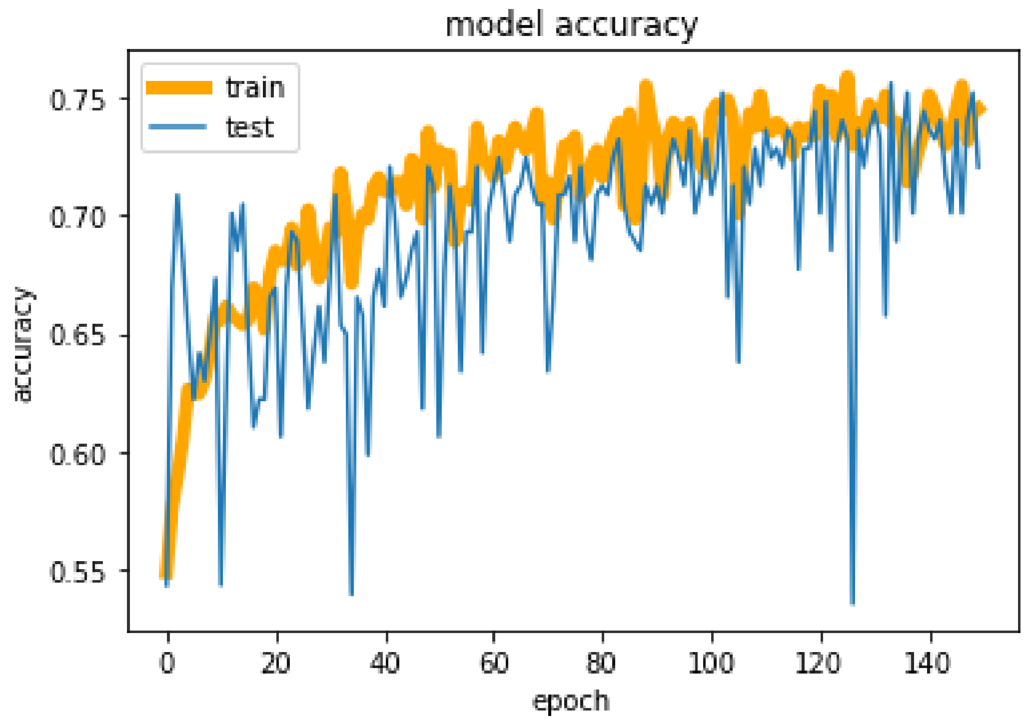
```
514/514 [==============================] - 0s 121us/sample - loss: 0.5638 - acc:
0.7335 - val_loss: 0.5626 - val_acc: 0.7362
Epoch 116/150
514/514 [==============================] - 0s 140us/sample - loss: 0.5359 - acc:
0.7257 - val_loss: 0.5698 - val_acc: 0.7323
Epoch 117/150
514/514 [==============================] - 0s 135us/sample - loss: 0.5328 - acc:
0.7374 - val_loss: 0.6395 - val_acc: 0.6772
Epoch 118/150
514/514 [==============================] - 0s 125us/sample - loss: 0.5347 - acc:
0.7335 - val_loss: 0.5602 - val_acc: 0.7283
Epoch 119/150
514/514 [==============================] - 0s 153us/sample - loss: 0.5383 - acc:
0.7374 - val_loss: 0.5699 - val_acc: 0.7283
Epoch 120/150
514/514 [==============================] - 0s 148us/sample - loss: 0.5467 - acc:
0.7335 - val_loss: 0.5663 - val_acc: 0.7441
Epoch 121/150
514/514 [==============================] - 0s 167us/sample - loss: 0.5233 - acc:
0.7529 - val_loss: 0.5869 - val_acc: 0.7008
Epoch 122/150
514/514 [==============================] - 0s 164us/sample - loss: 0.5360 - acc:
0.7471 - val_loss: 0.5542 - val_acc: 0.7480
Epoch 123/150
514/514 [==============================] - 0s 146us/sample - loss: 0.5484 - acc:
0.7510 - val_loss: 0.6051 - val_acc: 0.6850
Epoch 124/150
514/514 [==============================] - 0s 161us/sample - loss: 0.5383 - acc:
0.7335 - val_loss: 0.5649 - val_acc: 0.7283
Epoch 125/150
514/514 [==============================] - 0s 192us/sample - loss: 0.5255 - acc:
0.7451 - val_loss: 0.5572 - val_acc: 0.7402
Epoch 126/150
514/514 [==============================] - 0s 192us/sample - loss: 0.5225 - acc:
0.7588 - val_loss: 0.5616 - val_acc: 0.7323
Epoch 127/150
514/514 [==============================] - 0s 156us/sample - loss: 0.5435 - acc:
0.7296 - val_loss: 0.7726 - val_acc: 0.5354
Epoch 128/150
514/514 [==============================] - 0s 184us/sample - loss: 0.5492 - acc:
0.7393 - val_loss: 0.5662 - val_acc: 0.7362
Epoch 129/150
514/514 [==============================] - 0s 165us/sample - loss: 0.5358 - acc:
0.7354 - val_loss: 0.5907 - val_acc: 0.7205
Epoch 130/150
514/514 [==============================] - 0s 185us/sample - loss: 0.5321 - acc:
0.7471 - val_loss: 0.5547 - val_acc: 0.7362
Epoch 131/150
```

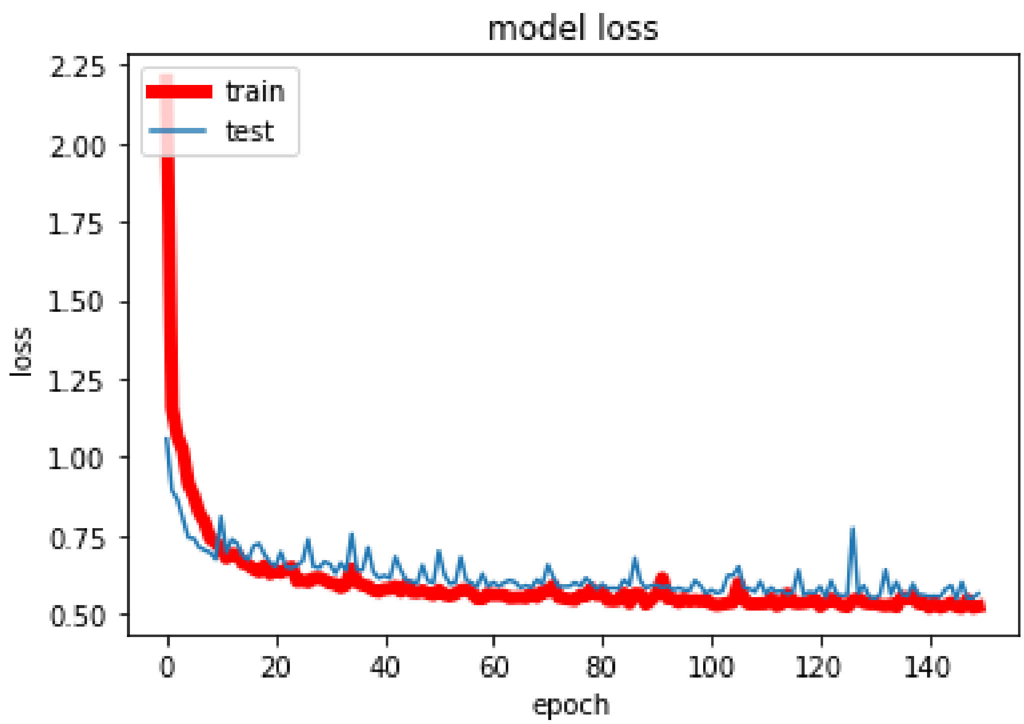
```
514/514 [==============================] - 0s 175us/sample - loss: 0.5328 - acc:
0.7412 - val_loss: 0.5501 - val_acc: 0.7441
Epoch 132/150
514/514 [==============================] - 0s 205us/sample - loss: 0.5281 - acc:
0.7412 - val_loss: 0.5610 - val_acc: 0.7323
Epoch 133/150
514/514 [==============================] - 0s 162us/sample - loss: 0.5248 - acc:
0.7510 - val_loss: 0.6409 - val_acc: 0.6575
Epoch 134/150
514/514 [==============================] - 0s 206us/sample - loss: 0.5295 - acc:
0.7393 - val_loss: 0.5625 - val_acc: 0.7559
Epoch 135/150
514/514 [==============================] - 0s 188us/sample - loss: 0.5221 - acc:
0.7393 - val_loss: 0.6012 - val_acc: 0.6890
Epoch 136/150
514/514 [==============================] - 0s 206us/sample - loss: 0.5535 - acc:
0.7354 - val_loss: 0.5623 - val_acc: 0.7323
Epoch 137/150
514/514 [==============================] - ETA: 0s - loss: 0.5428 - acc: 0.714 -
0s 179us/sample - loss: 0.5474 - acc: 0.7140 - val_loss: 0.5566 - val_acc:
0.7520
Epoch 138/150
514/514 [==============================] - 0s 162us/sample - loss: 0.5569 - acc:
0.7179 - val_loss: 0.5953 - val_acc: 0.7008
Epoch 139/150
514/514 [==============================] - 0s 186us/sample - loss: 0.5326 - acc:
0.7276 - val_loss: 0.5623 - val_acc: 0.7323
Epoch 140/150
514/514 [==============================] - 0s 135us/sample - loss: 0.5343 - acc:
0.7354 - val_loss: 0.5598 - val_acc: 0.7441
Epoch 141/150
514/514 [==============================] - 0s 146us/sample - loss: 0.5201 - acc:
0.7510 - val_loss: 0.5554 - val_acc: 0.7362
Epoch 142/150
514/514 [==============================] - 0s 138us/sample - loss: 0.5355 - acc:
0.7451 - val_loss: 0.5568 - val_acc: 0.7323
Epoch 143/150
514/514 [==============================] - 0s 149us/sample - loss: 0.5200 - acc:
0.7374 - val_loss: 0.5545 - val_acc: 0.7402
Epoch 144/150
514/514 [==============================] - 0s 150us/sample - loss: 0.5261 - acc:
0.7296 - val_loss: 0.5786 - val_acc: 0.7165
Epoch 145/150
514/514 [==============================] - 0s 160us/sample - loss: 0.5356 - acc:
0.7354 - val_loss: 0.5909 - val_acc: 0.7008
Epoch 146/150
514/514 [==============================] - 0s 153us/sample - loss: 0.5227 - acc:
0.7451 - val_loss: 0.5476 - val_acc: 0.7402
```

```
Epoch 147/150
514/514 [==============================] - 0s 142us/sample - loss: 0.5196 - acc:
0.7549 - val_loss: 0.6007 - val_acc: 0.7008
Epoch 148/150
514/514 [==============================] - 0s 140us/sample - loss: 0.5453 - acc:
0.7315 - val_loss: 0.5506 - val_acc: 0.7402
Epoch 149/150
514/514 [==============================] - 0s 136us/sample - loss: 0.5196 - acc:
0.7432 - val_loss: 0.5504 - val_acc: 0.7520
Epoch 150/150
514/514 [==============================] - 0s 131us/sample - loss: 0.5234 - acc:
0.7451 - val_loss: 0.5648 - val_acc: 0.7205
```

[87]:

```
import matplotlib.pyplot as plt


loss=list(model.history.values())[0]
accuracy=list(model.history.values())[1]
val_loss=list(model.history.values())[2]
val_accuracy=list(model.history.values())[3]

# summarize history for accuracy
plt.plot(accuracy,color='orange', linewidth=5)
plt.plot(val_accuracy)
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(loss, color='red', linewidth=5)
plt.plot(val_loss)
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```


model accuracy
train
test
0.75
0.70
0.65
0.60
0.55
accuracy
0
20
40
60
80
100
120
140
epoch


model loss
train
test
2.25
2.00
1.75
1.50
1.25
1.00
0.75
0.50
loss
0
20
40
60
80
100
120
140
epoch

1.2 Second example using Iris Data

First we need to convert the species column in binary form. Next we retrieve the data as a list and convert back as an array

[75]:
```
import pandas as pd
from numpy import loadtxt
from keras.models import Sequential
from keras.layers import Dense

#Importing and reading data.
Iris= pd.read_csv("Iris.csv")
# Since our data is seperated by semicolons we need to do sep=";"
Iris=Iris.iloc[:,[1,2,3,4,5]]

Iris_list=Iris.values.tolist()
for i in range(50):
    for j in range(50,100):
        Iris_list[i][4]=1
        Iris_list[j][4]=0

Iris_list1=[Iris_list[i] for i in range(40)]
Iris_list2=[Iris_list[i] for i in range(50,90)]
Iris_list3=Iris_list1+Iris_list2

Iris_list4=[Iris_list[i] for i in range(40,50)]
Iris_list5=[Iris_list[i] for i in range(90,100)]
Iris_list6=Iris_list4+Iris_list5
```

[76]:
```
train=np.array(Iris_list3)
train=train+np.random.normal(0,10,train.shape)
test=np.array(Iris_list6)
test=test+np.random.normal(0,10,test.shape)
# load the dataset
dataset =train
X_train= dataset[:,0:4]
y_train= dataset[:,4]
X_test=test[:,0:4]
y_test=test[:,4]
```

[77]:
```
# define the keras model
model = Sequential()
model.add(Dense(30, input_dim=4, activation='relu'))
model.add(Dense(4, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

[78]:
```
# compile the keras model
model.compile(loss='binary_crossentropy', optimizer='Adam',␣
↪metrics=['accuracy'])
```

[79]:
```
# fit the keras model on the dataset
model=model.fit(X_train, y_train,validation_split=0.33, epochs=100,␣
↪batch_size=10)
```

```
Epoch 1/100
6/6 [==============================] - 1s 46ms/step - loss: 5.5296 - accuracy:
0.0000e+00 - val_loss: -22.5639 - val_accuracy: 0.0000e+00
Epoch 2/100
6/6 [==============================] - 0s 12ms/step - loss: 9.4528 - accuracy:
0.0000e+00 - val_loss: -22.0321 - val_accuracy: 0.0000e+00
Epoch 3/100
6/6 [==============================] - 0s 12ms/step - loss: 0.4527 - accuracy:
0.0000e+00 - val_loss: -21.7011 - val_accuracy: 0.0000e+00
Epoch 4/100
6/6 [==============================] - 0s 12ms/step - loss: -3.2205 - accuracy:
0.0000e+00 - val_loss: -21.3742 - val_accuracy: 0.0000e+00
Epoch 5/100
6/6 [==============================] - 0s 9ms/step - loss: 1.2213 - accuracy:
0.0000e+00 - val_loss: -20.9858 - val_accuracy: 0.0000e+00
Epoch 6/100
6/6 [==============================] - 0s 14ms/step - loss: -5.4259 - accuracy:
0.0000e+00 - val_loss: -20.9449 - val_accuracy: 0.0000e+00
Epoch 7/100
6/6 [==============================] - 0s 12ms/step - loss: -1.5994 - accuracy:
0.0000e+00 - val_loss: -20.5780 - val_accuracy: 0.0000e+00
Epoch 8/100
6/6 [==============================] - 0s 12ms/step - loss: -12.2830 - accuracy:
0.0000e+00 - val_loss: -20.2285 - val_accuracy: 0.0000e+00
Epoch 9/100
6/6 [==============================] - 0s 14ms/step - loss: -8.7506 - accuracy:
0.0000e+00 - val_loss: -19.7237 - val_accuracy: 0.0000e+00
Epoch 10/100
6/6 [==============================] - 0s 13ms/step - loss: -12.8178 - accuracy:
0.0000e+00 - val_loss: -19.6739 - val_accuracy: 0.0000e+00
Epoch 11/100
6/6 [==============================] - 0s 14ms/step - loss: -11.2020 - accuracy:
0.0000e+00 - val_loss: -19.6014 - val_accuracy: 0.0000e+00
Epoch 12/100
6/6 [==============================] - 0s 12ms/step - loss: -13.9146 - accuracy:
0.0000e+00 - val_loss: -19.5185 - val_accuracy: 0.0000e+00
Epoch 13/100
6/6 [==============================] - 0s 15ms/step - loss: -8.5914 - accuracy:
0.0000e+00 - val_loss: -19.2496 - val_accuracy: 0.0000e+00
```

```
Epoch 14/100
6/6 [==============================] - 0s 14ms/step - loss: -26.4099 - accuracy:
0.0000e+00 - val_loss: -18.8971 - val_accuracy: 0.0000e+00
Epoch 15/100
6/6 [==============================] - 0s 13ms/step - loss: -3.4573 - accuracy:
0.0000e+00 - val_loss: -18.4768 - val_accuracy: 0.0000e+00
Epoch 16/100
6/6 [==============================] - 0s 11ms/step - loss: -26.8783 - accuracy:
0.0000e+00 - val_loss: -18.8509 - val_accuracy: 0.0000e+00
Epoch 17/100
6/6 [==============================] - 0s 14ms/step - loss: -14.8959 - accuracy:
0.0000e+00 - val_loss: -19.0514 - val_accuracy: 0.0000e+00
Epoch 18/100
6/6 [==============================] - 0s 13ms/step - loss: -28.4216 - accuracy:
0.0000e+00 - val_loss: -19.5341 - val_accuracy: 0.0000e+00
Epoch 19/100
6/6 [==============================] - 0s 13ms/step - loss: -21.0527 - accuracy:
0.0000e+00 - val_loss: -19.5619 - val_accuracy: 0.0000e+00
Epoch 20/100
6/6 [==============================] - 0s 12ms/step - loss: -14.9446 - accuracy:
0.0000e+00 - val_loss: -19.4513 - val_accuracy: 0.0000e+00
Epoch 21/100
6/6 [==============================] - 0s 14ms/step - loss: -16.2596 - accuracy:
0.0000e+00 - val_loss: -19.6134 - val_accuracy: 0.0000e+00
Epoch 22/100
6/6 [==============================] - ETA: 0s - loss: -50.0278 - accuracy:
0.0000e+ - 0s 11ms/step - loss: -28.0645 - accuracy: 0.0000e+00 - val_loss:
-19.6207 - val_accuracy: 0.0000e+00
Epoch 23/100
6/6 [==============================] - 0s 12ms/step - loss: -28.8804 - accuracy:
0.0000e+00 - val_loss: -19.5397 - val_accuracy: 0.0000e+00
Epoch 24/100
6/6 [==============================] - 0s 14ms/step - loss: -28.5418 - accuracy:
0.0000e+00 - val_loss: -19.5474 - val_accuracy: 0.0000e+00
Epoch 25/100
6/6 [==============================] - 0s 14ms/step - loss: -41.4781 - accuracy:
0.0000e+00 - val_loss: -19.6378 - val_accuracy: 0.0000e+00
Epoch 26/100
6/6 [==============================] - 0s 14ms/step - loss: -20.5467 - accuracy:
0.0000e+00 - val_loss: -19.2985 - val_accuracy: 0.0000e+00
Epoch 27/100
6/6 [==============================] - 0s 11ms/step - loss: -44.0416 - accuracy:
0.0000e+00 - val_loss: -19.3759 - val_accuracy: 0.0000e+00
Epoch 28/100
6/6 [==============================] - 0s 13ms/step - loss: -22.7624 - accuracy:
0.0000e+00 - val_loss: -19.3819 - val_accuracy: 0.0000e+00
Epoch 29/100
6/6 [==============================] - 0s 12ms/step - loss: -36.2763 - accuracy:
```

```
0.0000e+00 - val_loss: -19.8851 - val_accuracy: 0.0000e+00
Epoch 30/100
6/6 [==============================] - 0s 14ms/step - loss: -26.5608 - accuracy:
0.0000e+00 - val_loss: -20.3497 - val_accuracy: 0.0000e+00
Epoch 31/100
6/6 [==============================] - 0s 13ms/step - loss: -47.4665 - accuracy:
0.0000e+00 - val_loss: -20.5067 - val_accuracy: 0.0000e+00
Epoch 32/100
6/6 [==============================] - 0s 14ms/step - loss: -47.4722 - accuracy:
0.0000e+00 - val_loss: -20.6341 - val_accuracy: 0.0000e+00
Epoch 33/100
6/6 [==============================] - 0s 13ms/step - loss: -26.0575 - accuracy:
0.0000e+00 - val_loss: -20.7389 - val_accuracy: 0.0000e+00
Epoch 34/100
6/6 [==============================] - 0s 14ms/step - loss: -32.1847 - accuracy:
0.0000e+00 - val_loss: -21.2769 - val_accuracy: 0.0000e+00
Epoch 35/100
6/6 [==============================] - 0s 12ms/step - loss: -30.3718 - accuracy:
0.0000e+00 - val_loss: -21.5327 - val_accuracy: 0.0000e+00
Epoch 36/100
6/6 [==============================] - 0s 14ms/step - loss: -40.3625 - accuracy:
0.0000e+00 - val_loss: -21.8416 - val_accuracy: 0.0000e+00
Epoch 37/100
6/6 [==============================] - 0s 15ms/step - loss: -54.6598 - accuracy:
0.0000e+00 - val_loss: -22.3994 - val_accuracy: 0.0000e+00
Epoch 38/100
6/6 [==============================] - 0s 12ms/step - loss: -58.1028 - accuracy:
0.0000e+00 - val_loss: -23.2267 - val_accuracy: 0.0000e+00
Epoch 39/100
6/6 [==============================] - 0s 16ms/step - loss: -29.2339 - accuracy:
0.0000e+00 - val_loss: -23.9544 - val_accuracy: 0.0000e+00
Epoch 40/100
6/6 [==============================] - 0s 15ms/step - loss: -32.5209 - accuracy:
0.0000e+00 - val_loss: -24.5521 - val_accuracy: 0.0000e+00
Epoch 41/100
6/6 [==============================] - 0s 11ms/step - loss: -57.4719 - accuracy:
0.0000e+00 - val_loss: -26.1896 - val_accuracy: 0.0000e+00
Epoch 42/100
6/6 [==============================] - 0s 14ms/step - loss: -35.7421 - accuracy:
0.0000e+00 - val_loss: -27.1363 - val_accuracy: 0.0000e+00
Epoch 43/100
6/6 [==============================] - 0s 13ms/step - loss: -27.8223 - accuracy:
0.0000e+00 - val_loss: -27.8911 - val_accuracy: 0.0000e+00
Epoch 44/100
6/6 [==============================] - 0s 15ms/step - loss: -44.1945 - accuracy:
0.0000e+00 - val_loss: -28.7405 - val_accuracy: 0.0000e+00
Epoch 45/100
6/6 [==============================] - 0s 17ms/step - loss: -43.3079 - accuracy:
```

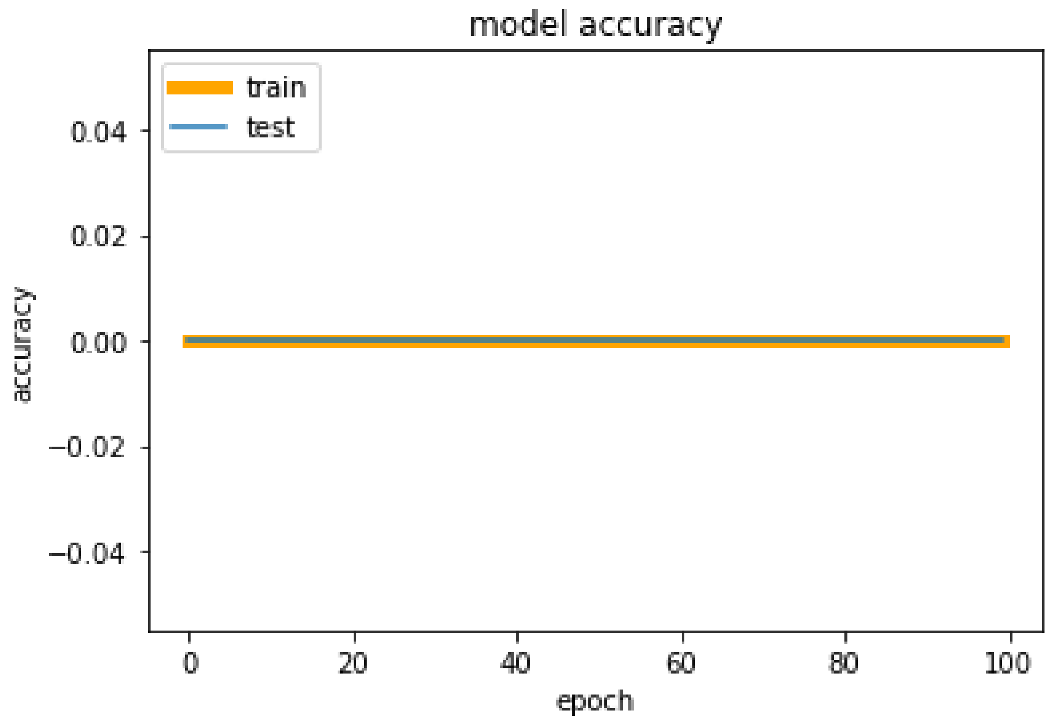
```
0.0000e+00 - val_loss: -29.5176 - val_accuracy: 0.0000e+00
Epoch 46/100
6/6 [==============================] - 0s 17ms/step - loss: -46.1292 - accuracy:
0.0000e+00 - val_loss: -30.3047 - val_accuracy: 0.0000e+00
Epoch 47/100
6/6 [==============================] - 0s 17ms/step - loss: -69.7764 - accuracy:
0.0000e+00 - val_loss: -30.9031 - val_accuracy: 0.0000e+00
Epoch 48/100
6/6 [==============================] - 0s 18ms/step - loss: -46.4587 - accuracy:
0.0000e+00 - val_loss: -31.9148 - val_accuracy: 0.0000e+00
Epoch 49/100
6/6 [==============================] - 0s 15ms/step - loss: -63.2033 - accuracy:
0.0000e+00 - val_loss: -33.2945 - val_accuracy: 0.0000e+00
Epoch 50/100
6/6 [==============================] - 0s 17ms/step - loss: -51.7353 - accuracy:
0.0000e+00 - val_loss: -34.0121 - val_accuracy: 0.0000e+00
Epoch 51/100
6/6 [==============================] - 0s 11ms/step - loss: -102.4959 -
accuracy: 0.0000e+00 - val_loss: -34.7947 - val_accuracy: 0.0000e+00
Epoch 52/100
6/6 [==============================] - 0s 12ms/step - loss: -84.1046 - accuracy:
0.0000e+00 - val_loss: -35.4863 - val_accuracy: 0.0000e+00
Epoch 53/100
6/6 [==============================] - 0s 13ms/step - loss: -44.6094 - accuracy:
0.0000e+00 - val_loss: -36.3535 - val_accuracy: 0.0000e+00
Epoch 54/100
6/6 [==============================] - 0s 12ms/step - loss: -48.4791 - accuracy:
0.0000e+00 - val_loss: -37.4296 - val_accuracy: 0.0000e+00
Epoch 55/100
6/6 [==============================] - 0s 12ms/step - loss: -116.0825 -
accuracy: 0.0000e+00 - val_loss: -39.0799 - val_accuracy: 0.0000e+00
Epoch 56/100
6/6 [==============================] - 0s 12ms/step - loss: -86.1726 - accuracy:
0.0000e+00 - val_loss: -40.5645 - val_accuracy: 0.0000e+00
Epoch 57/100
6/6 [==============================] - 0s 10ms/step - loss: -88.9899 - accuracy:
0.0000e+00 - val_loss: -41.8455 - val_accuracy: 0.0000e+00
Epoch 58/100
6/6 [==============================] - 0s 13ms/step - loss: -55.2879 - accuracy:
0.0000e+00 - val_loss: -42.3583 - val_accuracy: 0.0000e+00
Epoch 59/100
6/6 [==============================] - 0s 13ms/step - loss: -88.5719 - accuracy:
0.0000e+00 - val_loss: -43.3875 - val_accuracy: 0.0000e+00
Epoch 60/100
6/6 [==============================] - 0s 14ms/step - loss: -99.2751 - accuracy:
0.0000e+00 - val_loss: -45.3317 - val_accuracy: 0.0000e+00
Epoch 61/100
6/6 [==============================] - 0s 13ms/step - loss: -74.3642 - accuracy:
```

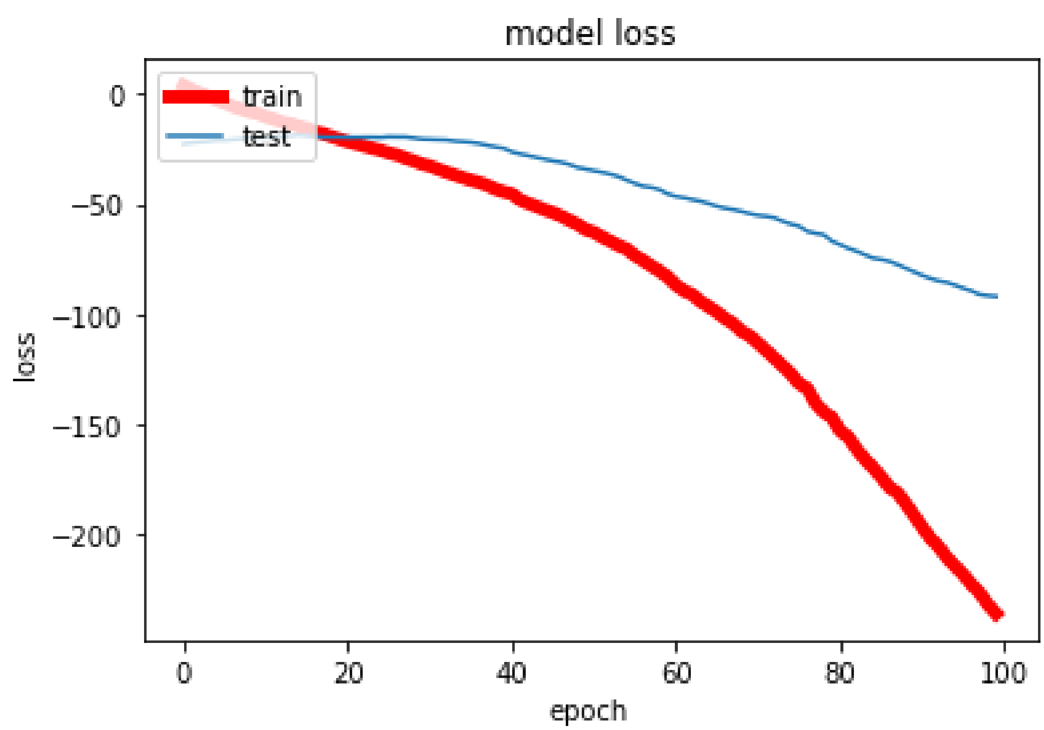
```
0.0000e+00 - val_loss: -46.4528 - val_accuracy: 0.0000e+00
Epoch 62/100
6/6 [==============================] - 0s 14ms/step - loss: -49.7480 - accuracy:
0.0000e+00 - val_loss: -47.0818 - val_accuracy: 0.0000e+00
Epoch 63/100
6/6 [==============================] - 0s 13ms/step - loss: -123.3102 -
accuracy: 0.0000e+00 - val_loss: -47.9164 - val_accuracy: 0.0000e+00
Epoch 64/100
6/6 [==============================] - 0s 13ms/step - loss: -40.4122 - accuracy:
0.0000e+00 - val_loss: -48.6501 - val_accuracy: 0.0000e+00
Epoch 65/100
6/6 [==============================] - 0s 14ms/step - loss: -77.2391 - accuracy:
0.0000e+00 - val_loss: -49.7246 - val_accuracy: 0.0000e+00
Epoch 66/100
6/6 [==============================] - 0s 12ms/step - loss: -105.3440 -
accuracy: 0.0000e+00 - val_loss: -50.9541 - val_accuracy: 0.0000e+00
Epoch 67/100
6/6 [==============================] - 0s 12ms/step - loss: -113.4174 -
accuracy: 0.0000e+00 - val_loss: -51.8312 - val_accuracy: 0.0000e+00
Epoch 68/100
6/6 [==============================] - 0s 12ms/step - loss: -160.3341 -
accuracy: 0.0000e+00 - val_loss: -52.4636 - val_accuracy: 0.0000e+00
Epoch 69/100
6/6 [==============================] - 0s 12ms/step - loss: -55.1092 - accuracy:
0.0000e+00 - val_loss: -53.3100 - val_accuracy: 0.0000e+00
Epoch 70/100
6/6 [==============================] - 0s 15ms/step - loss: -122.5891 -
accuracy: 0.0000e+00 - val_loss: -54.3123 - val_accuracy: 0.0000e+00
Epoch 71/100
6/6 [==============================] - 0s 15ms/step - loss: -108.1598 -
accuracy: 0.0000e+00 - val_loss: -55.0940 - val_accuracy: 0.0000e+00
Epoch 72/100
6/6 [==============================] - 0s 14ms/step - loss: -97.6528 - accuracy:
0.0000e+00 - val_loss: -55.5024 - val_accuracy: 0.0000e+00
Epoch 73/100
6/6 [==============================] - 0s 16ms/step - loss: -47.9994 - accuracy:
0.0000e+00 - val_loss: -56.1709 - val_accuracy: 0.0000e+00
Epoch 74/100
6/6 [==============================] - 0s 14ms/step - loss: -99.1516 - accuracy:
0.0000e+00 - val_loss: -57.7565 - val_accuracy: 0.0000e+00
Epoch 75/100
6/6 [==============================] - 0s 15ms/step - loss: -147.0015 -
accuracy: 0.0000e+00 - val_loss: -59.0263 - val_accuracy: 0.0000e+00
Epoch 76/100
6/6 [==============================] - 0s 12ms/step - loss: -71.9224 - accuracy:
0.0000e+00 - val_loss: -60.0429 - val_accuracy: 0.0000e+00
Epoch 77/100
6/6 [==============================] - 0s 14ms/step - loss: -238.0992 -
```

```
accuracy: 0.0000e+00 - val_loss: -62.3633 - val_accuracy: 0.0000e+00
Epoch 78/100
6/6 [==============================] - 0s 15ms/step - loss: -102.4890 -
accuracy: 0.0000e+00 - val_loss: -63.2379 - val_accuracy: 0.0000e+00
Epoch 79/100
6/6 [==============================] - 0s 12ms/step - loss: -49.4484 - accuracy:
0.0000e+00 - val_loss: -63.9212 - val_accuracy: 0.0000e+00
Epoch 80/100
6/6 [==============================] - 0s 13ms/step - loss: -152.5462 -
accuracy: 0.0000e+00 - val_loss: -66.8190 - val_accuracy: 0.0000e+00
Epoch 81/100
6/6 [==============================] - 0s 12ms/step - loss: -124.2960 -
accuracy: 0.0000e+00 - val_loss: -68.4202 - val_accuracy: 0.0000e+00
Epoch 82/100
6/6 [==============================] - 0s 13ms/step - loss: -246.0487 -
accuracy: 0.0000e+00 - val_loss: -70.0693 - val_accuracy: 0.0000e+00
Epoch 83/100
6/6 [==============================] - 0s 12ms/step - loss: -113.4953 -
accuracy: 0.0000e+00 - val_loss: -71.3820 - val_accuracy: 0.0000e+00
Epoch 84/100
6/6 [==============================] - 0s 14ms/step - loss: -149.1151 -
accuracy: 0.0000e+00 - val_loss: -73.0004 - val_accuracy: 0.0000e+00
Epoch 85/100
6/6 [==============================] - 0s 15ms/step - loss: -133.6067 -
accuracy: 0.0000e+00 - val_loss: -74.5233 - val_accuracy: 0.0000e+00
Epoch 86/100
6/6 [==============================] - 0s 15ms/step - loss: -146.8012 -
accuracy: 0.0000e+00 - val_loss: -75.1368 - val_accuracy: 0.0000e+00
Epoch 87/100
6/6 [==============================] - 0s 15ms/step - loss: -95.6296 - accuracy:
0.0000e+00 - val_loss: -76.0191 - val_accuracy: 0.0000e+00
Epoch 88/100
6/6 [==============================] - 0s 12ms/step - loss: -234.8985 -
accuracy: 0.0000e+00 - val_loss: -77.4795 - val_accuracy: 0.0000e+00
Epoch 89/100
6/6 [==============================] - 0s 12ms/step - loss: -248.1344 -
accuracy: 0.0000e+00 - val_loss: -79.1433 - val_accuracy: 0.0000e+00
Epoch 90/100
6/6 [==============================] - 0s 13ms/step - loss: -170.7168 -
accuracy: 0.0000e+00 - val_loss: -80.7605 - val_accuracy: 0.0000e+00
Epoch 91/100
6/6 [==============================] - 0s 11ms/step - loss: -99.0315 - accuracy:
0.0000e+00 - val_loss: -82.3805 - val_accuracy: 0.0000e+00
Epoch 92/100
6/6 [==============================] - 0s 13ms/step - loss: -106.0420 -
accuracy: 0.0000e+00 - val_loss: -83.7104 - val_accuracy: 0.0000e+00
Epoch 93/100
6/6 [==============================] - 0s 16ms/step - loss: -224.0792 -
```

```
accuracy: 0.0000e+00 - val_loss: -84.8463 - val_accuracy: 0.0000e+00
Epoch 94/100
6/6 [==============================] - 0s 18ms/step - loss: -208.3514 -
accuracy: 0.0000e+00 - val_loss: -85.5043 - val_accuracy: 0.0000e+00
Epoch 95/100
6/6 [==============================] - 0s 17ms/step - loss: -291.2307 -
accuracy: 0.0000e+00 - val_loss: -86.7501 - val_accuracy: 0.0000e+00
Epoch 96/100
6/6 [==============================] - 0s 12ms/step - loss: -338.4808 -
accuracy: 0.0000e+00 - val_loss: -88.1641 - val_accuracy: 0.0000e+00
Epoch 97/100
6/6 [==============================] - 0s 11ms/step - loss: -270.8459 -
accuracy: 0.0000e+00 - val_loss: -89.3843 - val_accuracy: 0.0000e+00
Epoch 98/100
6/6 [==============================] - 0s 13ms/step - loss: -344.4174 -
accuracy: 0.0000e+00 - val_loss: -91.0022 - val_accuracy: 0.0000e+00
Epoch 99/100
6/6 [==============================] - 0s 11ms/step - loss: -236.7900 -
accuracy: 0.0000e+00 - val_loss: -91.5900 - val_accuracy: 0.0000e+00
Epoch 100/100
6/6 [==============================] - 0s 13ms/step - loss: -190.3697 -
accuracy: 0.0000e+00 - val_loss: -91.9207 - val_accuracy: 0.0000e+00
```

[80]:
```python
loss=list(model.history.values())[0]
accuracy=list(model.history.values())[1]
val_loss=list(model.history.values())[2]
val_accuracy=list(model.history.values())[3]

# summarize history for accuracy
plt.plot(accuracy,color='orange', linewidth=5)
plt.plot(val_accuracy)
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(loss, color='red', linewidth=5)
plt.plot(val_loss)
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```


model accuracy
train
test
0.04
0.02
0.00
−0.02
−0.04
accuracy
0
20
40
60
80
100
epoch


model loss
train
test
0
−50
−100
−150
−200
loss
0
20
40
60
80
100
epoch

```
[95]: #predictions=model.predict(X_test)
      #pred=[round(sum(predictions[i])) for i in range(len(predictions))]
      #score=accuracy_score(y_test,pred)

      #tree.export_graphviz(model,out_file='music_recommender1.
       ↪dot',feature_names=['age','sex'],class_names=sorted(y.
       ↪unique()),label='all',rounded=True,filled=True)

      #score
```

1.3 Image Classification Using Deep Learning, MNIST DATASET

Import the required python libraries for Image Classification

```
[2]: #install required libraries
     import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
     import keras
     from keras.models import Sequential
     from keras.layers import Convolution2D
     from keras.layers import MaxPooling2D
     from keras.layers import Flatten
     from keras.layers import Dense
     from keras.wrappers.scikit_learn import KerasClassifier
     from keras.layers import Dropout
     #model evaluation packages
     from sklearn.metrics import f1_score, roc_auc_score, log_loss
     from sklearn.model_selection import cross_val_score, cross_validate
```

```
[3]: #read mnist fashion dataset
     mnist = keras.datasets.fashion_mnist
     (X_train, y_train), (X_test, y_test) = mnist.load_data()
     print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)
```

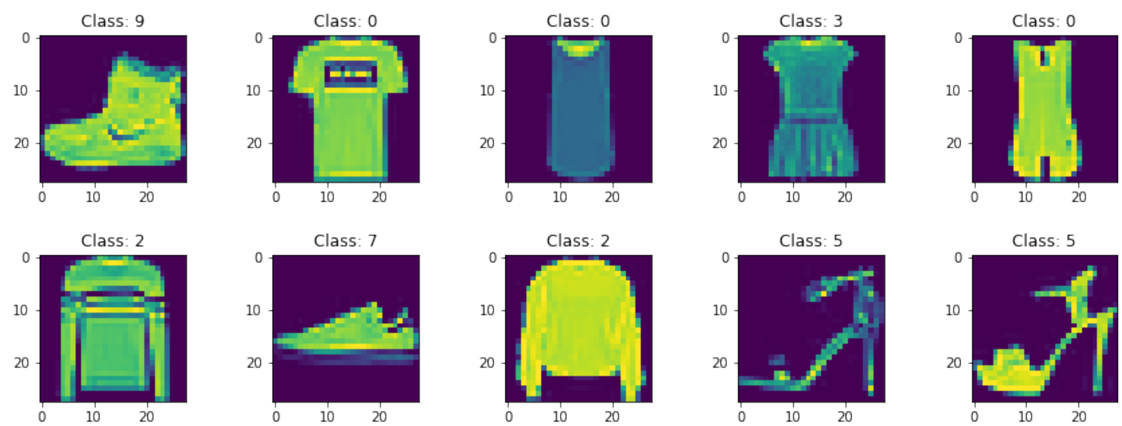
```
(60000, 28, 28) (60000,) (10000, 28, 28) (10000,)
```

```
[8]: #reshape data from 3-D to 2-D array
     X_train = X_train.reshape(60000, 784)
     X_test = X_test.reshape(10000, 784)
     #feature scaling
     from sklearn.preprocessing import MinMaxScaler
     minmax = MinMaxScaler()
     #fit and transform training dataset
     X_train = minmax.fit_transform(X_train)
     X_train=X_train/255
```

```
#transform testing dataset
X_test = minmax.transform(X_test)
print('Number of unique classes: ', len(np.unique(y_train)))
print('Classes: ', np.unique(y_train))
```

```
Number of unique classes:  10
Classes:  [0 1 2 3 4 5 6 7 8 9]
```

[9]:
```
fig, axes = plt.subplots(nrows=2, ncols=5,figsize=(15,5))
ax = axes.ravel()
for i in range(10):
    ax[i].imshow(X_train[i].reshape(28,28))
    ax[i].title.set_text('Class: ' + str(y_train[i]))
plt.subplots_adjust(hspace=0.5)
plt.show()
```



[100]:
```
#initializing CNN model
classifier_e25 = Sequential()
#add 1st hidden layer
classifier_e25.add(Dense(input_dim = X_train.shape[1], units = 256,
 ↪kernel_initializer='uniform', activation='relu'))
#add output layer
classifier_e25.add(Dense(units = 10, kernel_initializer='uniform',
 ↪activation='softmax'))
#compile the neural network
classifier_e25.compile(optimizer='adam',
 ↪loss='sparse_categorical_crossentropy', metrics=['accuracy'])
#model summary
classifier_e25.summary()
```

```
WARNING:tensorflow:From /opt/anaconda3/envs/Project/lib/python3.8/site-
packages/tensorflow/python/keras/initializers/initializers_v1.py:58: calling
```

```
RandomUniform.__init__ (from tensorflow.python.ops.init_ops) with dtype is
deprecated and will be removed in a future version.
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to the
constructor
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_6 (Dense)              (None, 256)               200960
_________________________________________________________________
dense_7 (Dense)              (None, 10)                2570
=================================================================
Total params: 203,530
Trainable params: 203,530
Non-trainable params: 0
_________________________________________________________________
```

[101]:
```
#fit training dataset into the model
model= classifier_e25.fit(X_train, y_train, validation_split=0.33, epochs=25,␣
↪batch_size=10)
```

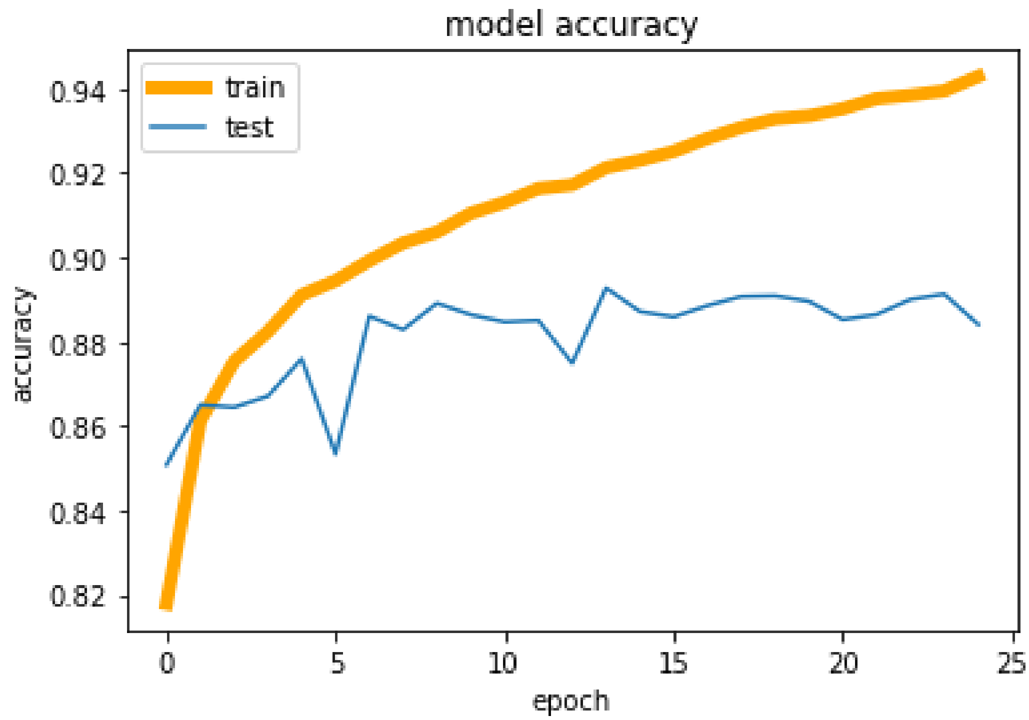
```
Train on 40199 samples, validate on 19801 samples
Epoch 1/25
40130/40199 [============================>.] - ETA: 0s - loss: 0.5094 - acc:
0.8178

/opt/anaconda3/envs/Project/lib/python3.8/site-
packages/tensorflow/python/keras/engine/training.py:2325: UserWarning:
`Model.state_updates` will be removed in a future version. This property should
not be used in TensorFlow 2.0, as `updates` are applied automatically.
  warnings.warn('`Model.state_updates` will be removed in a future version. '

40199/40199 [==============================] - 12s 297us/sample - loss: 0.5091 -
acc: 0.8179 - val_loss: 0.4233 - val_acc: 0.8508
Epoch 2/25
40199/40199 [==============================] - 11s 283us/sample - loss: 0.3782 -
acc: 0.8612 - val_loss: 0.3847 - val_acc: 0.8650
Epoch 3/25
40199/40199 [==============================] - 11s 283us/sample - loss: 0.3391 -
acc: 0.8753 - val_loss: 0.3732 - val_acc: 0.8645
Epoch 4/25
40199/40199 [==============================] - 13s 333us/sample - loss: 0.3178 -
acc: 0.8824 - val_loss: 0.3771 - val_acc: 0.8671
Epoch 5/25
40199/40199 [==============================] - 14s 353us/sample - loss: 0.2944 -
acc: 0.8911 - val_loss: 0.3385 - val_acc: 0.8760
Epoch 6/25
40199/40199 [==============================] - 15s 362us/sample - loss: 0.2818 -
```

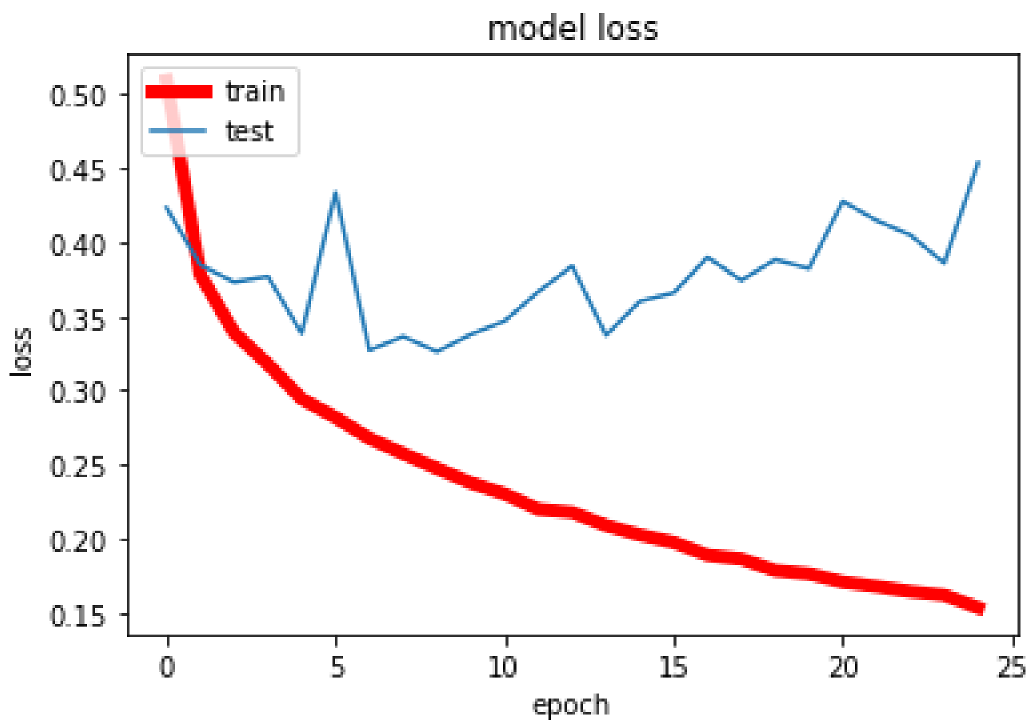
```
acc: 0.8945 - val_loss: 0.4338 - val_acc: 0.8533
Epoch 7/25
40199/40199 [==============================] - 14s 342us/sample - loss: 0.2677 -
acc: 0.8992 - val_loss: 0.3272 - val_acc: 0.8861
Epoch 8/25
40199/40199 [==============================] - 12s 294us/sample - loss: 0.2574 -
acc: 0.9034 - val_loss: 0.3365 - val_acc: 0.8829
Epoch 9/25
40199/40199 [==============================] - 11s 271us/sample - loss: 0.2474 -
acc: 0.9060 - val_loss: 0.3263 - val_acc: 0.8891s - loss: 0.2472 -
Epoch 10/25
40199/40199 [==============================] - 11s 277us/sample - loss: 0.2377 -
acc: 0.9105 - val_loss: 0.3379 - val_acc: 0.8864
Epoch 11/25
40199/40199 [==============================] - 12s 307us/sample - loss: 0.2302 -
acc: 0.9130 - val_loss: 0.3470 - val_acc: 0.8847: 0.
Epoch 12/25
40199/40199 [==============================] - 12s 292us/sample - loss: 0.2198 -
acc: 0.9163 - val_loss: 0.3668 - val_acc: 0.8851
Epoch 13/25
40199/40199 [==============================] - 10s 258us/sample - loss: 0.2178 -
acc: 0.9172 - val_loss: 0.3840 - val_acc: 0.8749
Epoch 14/25
40199/40199 [==============================] - 11s 283us/sample - loss: 0.2089 -
acc: 0.9213 - val_loss: 0.3373 - val_acc: 0.8928
Epoch 15/25
40199/40199 [==============================] - 13s 330us/sample - loss: 0.2026 -
acc: 0.9230 - val_loss: 0.3603 - val_acc: 0.8872
Epoch 16/25
40199/40199 [==============================] - 14s 353us/sample - loss: 0.1975 -
acc: 0.9251 - val_loss: 0.3660 - val_acc: 0.8859
Epoch 17/25
40199/40199 [==============================] - 15s 376us/sample - loss: 0.1889 -
acc: 0.9282 - val_loss: 0.3899 - val_acc: 0.8886
Epoch 18/25
40199/40199 [==============================] - 14s 357us/sample - loss: 0.1863 -
acc: 0.9308 - val_loss: 0.3744 - val_acc: 0.8908
Epoch 19/25
40199/40199 [==============================] - 17s 421us/sample - loss: 0.1785 -
acc: 0.9329 - val_loss: 0.3882 - val_acc: 0.8909
Epoch 20/25
40199/40199 [==============================] - 12s 306us/sample - loss: 0.1763 -
acc: 0.9336 - val_loss: 0.3823 - val_acc: 0.8896
Epoch 21/25
40199/40199 [==============================] - 14s 345us/sample - loss: 0.1708 -
acc: 0.9353 - val_loss: 0.4276 - val_acc: 0.8853
Epoch 22/25
40199/40199 [==============================] - 18s 445us/sample - loss: 0.1678 -
```

```
acc: 0.9377 - val_loss: 0.4146 - val_acc: 0.8865
Epoch 23/25
40199/40199 [==============================] - 15s 362us/sample - loss: 0.1643 -
acc: 0.9385 - val_loss: 0.4048 - val_acc: 0.8901
Epoch 24/25
40199/40199 [==============================] - 20s 496us/sample - loss: 0.1619 -
acc: 0.9395 - val_loss: 0.3859 - val_acc: 0.8913
Epoch 25/25
40199/40199 [==============================] - 22s 544us/sample - loss: 0.1534 -
acc: 0.9429 - val_loss: 0.4537 - val_acc: 0.8841
```

[102]:

```
loss=list(model.history.values())[0]
accuracy=list(model.history.values())[1]
val_loss=list(model.history.values())[2]
val_accuracy=list(model.history.values())[3]

# summarize history for accuracy
plt.plot(accuracy,color='orange', linewidth=5)
plt.plot(val_accuracy)
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(loss, color='red', linewidth=5)
plt.plot(val_loss)
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```


model accuracy
train
test
accuracy
0.82
0.84
0.86
0.88
0.90
0.92
0.94
0
5
10
15
20
25
epoch


model loss
train
test
loss
0.15
0.20
0.25
0.30
0.35
0.40
0.45
0.50
0
5
10
15
20
25
epoch

[141]:
```
#evaluate the model for testing dataset
test_loss_e25 = classifier_e25.evaluate(X_test, y_test, verbose=0)
#calculate evaluation parameters
f1_e25 = f1_score(y_test, classifier_e25.predict_classes(X_test),␣
↪average='micro')
roc_e25 = roc_auc_score(y_test, classifier_e25.predict_proba(X_test),␣
↪multi_class='ovo')
#create evaluation dataframe
stats_e25 = pd.DataFrame({'Test accuracy' :  round(test_loss_e25[1]*100,3),
                          'F1 score'      : round(f1_e25,3),
                          'ROC AUC score' : round(roc_e25,3),
                          'Total Loss'    : round(test_loss_e25[0],3)}, index=[0])
#print evaluation dataframe
display(stats_e25)
```

```
WARNING:tensorflow:From <ipython-input-141-513845c4a642>:4:
Sequential.predict_classes (from tensorflow.python.keras.engine.sequential) is
deprecated and will be removed after 2021-01-01.
Instructions for updating:
Please use instead:* `np.argmax(model.predict(x), axis=-1)`,   if your model
does multi-class classification   (e.g. if it uses a `softmax` last-layer
activation).* `(model.predict(x) > 0.5).astype("int32")`,   if your model does
binary classification   (e.g. if it uses a `sigmoid` last-layer activation).
WARNING:tensorflow:From <ipython-input-141-513845c4a642>:5:
Sequential.predict_proba (from tensorflow.python.keras.engine.sequential) is
deprecated and will be removed after 2021-01-01.
Instructions for updating:
Please use `model.predict()` instead.
```

	Test accuracy	F1 score	ROC AUC score	Total Loss
0	86.94	0.869	0.989	0.543

1.4 Character Classification Using Deep Learning, MNIST DATASET

[142]:
```
#install required libraries
import pandas as pd
import numpy as np
#data visualization packages
import matplotlib.pyplot as plt
#keras packages
import keras
from keras.models import Sequential
from keras.layers import Convolution2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier
```

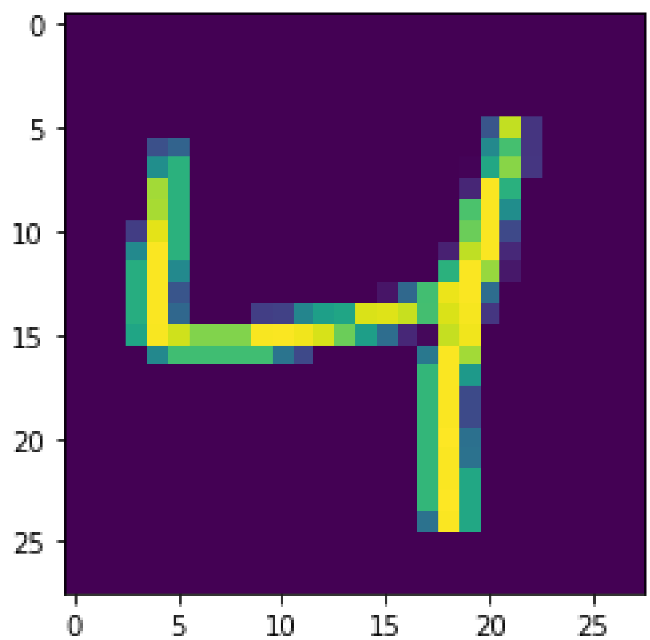
```
from keras.layers import Dropout
#model evaluation packages
from sklearn.metrics import f1_score, roc_auc_score, log_loss
from sklearn.model_selection import cross_val_score, cross_validate

#read mnist fashion dataset
mnist = keras.datasets.mnist
(X_train, y_train), (X_test, y_test) = mnist.load_data()
print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)
y_train
```

```
(60000, 28, 28) (60000,) (10000, 28, 28) (10000,)
```

[142]:
```
array([5, 0, 4, …, 5, 6, 8], dtype=uint8)
```

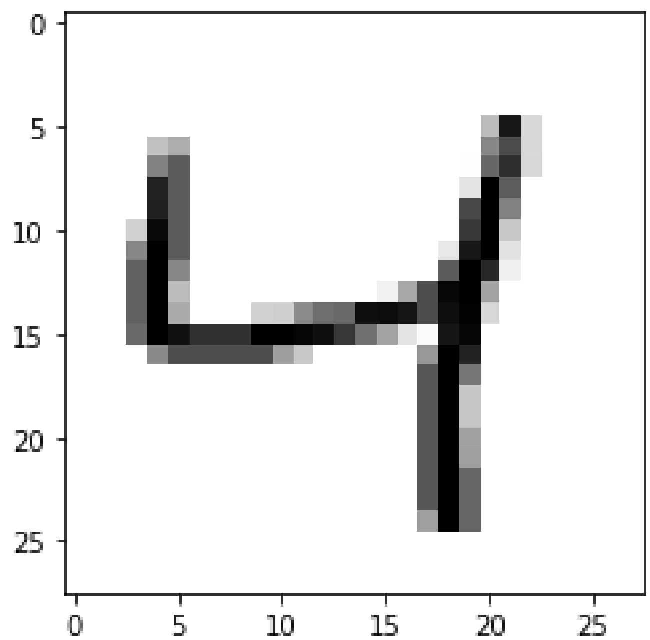
[143]:
```
plt.imshow(X_train[2])
plt.show
```

[143]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



[144]:
```
plt.imshow(X_train[2],cmap=plt.cm.binary)
plt.show
```

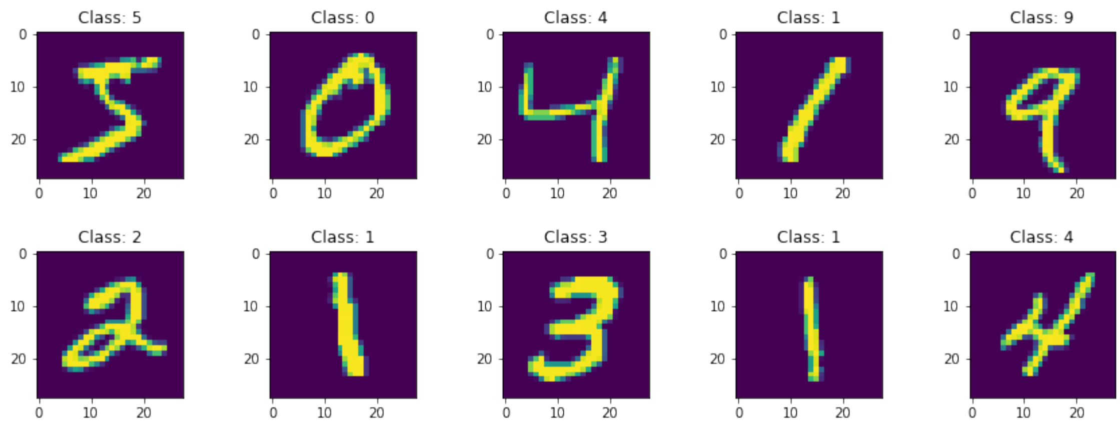
[144]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



[145]:
```
#reshape data from 3-D to 2-D array
X_train = X_train.reshape(60000, 784)
X_test = X_test.reshape(10000, 784)
#feature scaling
from sklearn.preprocessing import MinMaxScaler
minmax = MinMaxScaler()
#fit and transform training dataset
X_train = minmax.fit_transform(X_train)
#transform testing dataset
X_test = minmax.transform(X_test)
print('Number of unique classes: ', len(np.unique(y_train)))
print('Classes: ', np.unique(y_train))
```

```
Number of unique classes:  10
Classes:  [0 1 2 3 4 5 6 7 8 9]
```

[146]:
```
fig, axes = plt.subplots(nrows=2, ncols=5,figsize=(15,5))
ax = axes.ravel()
for i in range(10):
    ax[i].imshow(X_train[i].reshape(28,28))
    ax[i].title.set_text('Class: ' + str(y_train[i]))
plt.subplots_adjust(hspace=0.5)
plt.show()
```



[147]:
```
#initializing CNN model
classifier_e25 = Sequential()
#add 1st hidden layer
classifier_e25.add(Dense(input_dim = X_train.shape[1], units = 256,␣
↪kernel_initializer='uniform', activation='relu'))
#add output layer
classifier_e25.add(Dense(units = 10, kernel_initializer='uniform',␣
↪activation='softmax'))
#compile the neural network
classifier_e25.compile(optimizer='adam',␣
↪loss='sparse_categorical_crossentropy', metrics=['accuracy'])
#model summary
classifier_e25.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_16 (Dense)             (None, 256)               200960
_________________________________________________________________
dense_17 (Dense)             (None, 10)                2570
=================================================================
Total params: 203,530
Trainable params: 203,530
Non-trainable params: 0
_________________________________________________________________
```

[148]:
```
model= classifier_e25.fit(X_train, y_train, validation_split=0.33, epochs=25,␣
↪batch_size=10)
```

```
Train on 40199 samples, validate on 19801 samples
Epoch 1/25
```

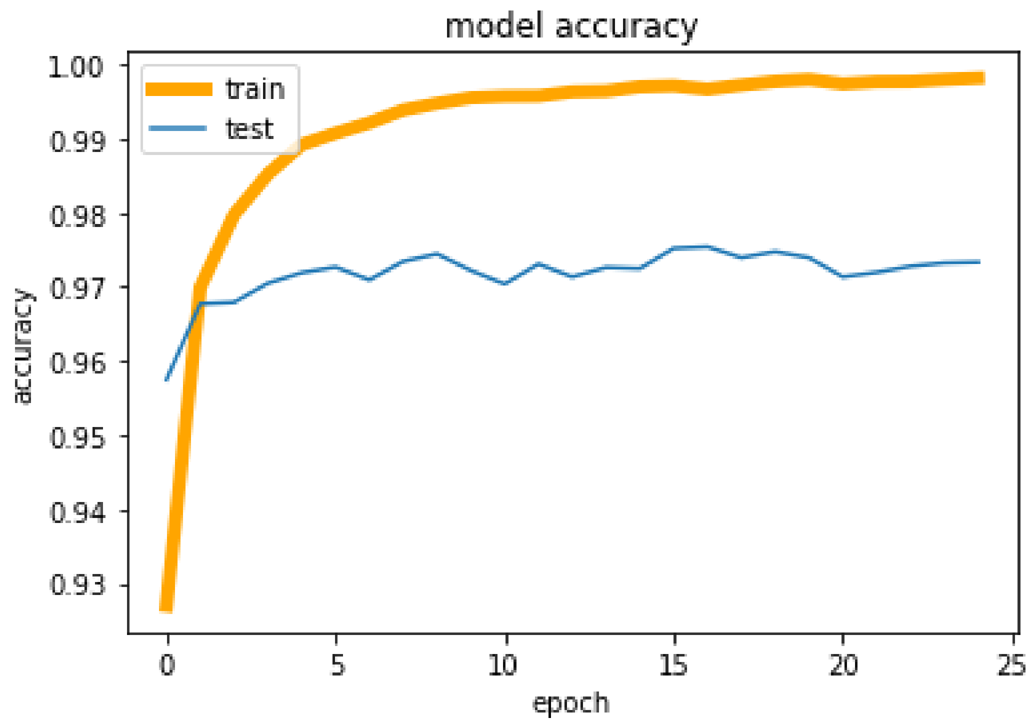
```
40199/40199 [==============================] - 9s 215us/sample - loss: 0.2471 -
acc: 0.9270 - val_loss: 0.1409 - val_acc: 0.9575
Epoch 2/25
40199/40199 [==============================] - 8s 205us/sample - loss: 0.0984 -
acc: 0.9699 - val_loss: 0.1034 - val_acc: 0.9677
Epoch 3/25
40199/40199 [==============================] - 9s 224us/sample - loss: 0.0644 -
acc: 0.9798 - val_loss: 0.1108 - val_acc: 0.9679
Epoch 4/25
40199/40199 [==============================] - 9s 217us/sample - loss: 0.0467 -
acc: 0.9852 - val_loss: 0.1052 - val_acc: 0.9705
Epoch 5/25
40199/40199 [==============================] - 9s 215us/sample - loss: 0.0334 -
acc: 0.9892 - val_loss: 0.1133 - val_acc: 0.9719
Epoch 6/25
40199/40199 [==============================] - 9s 213us/sample - loss: 0.0267 -
acc: 0.9907 - val_loss: 0.1185 - val_acc: 0.9727
Epoch 7/25
40199/40199 [==============================] - 9s 212us/sample - loss: 0.0222 -
acc: 0.9921 - val_loss: 0.1331 - val_acc: 0.9710
Epoch 8/25
40199/40199 [==============================] - 9s 216us/sample - loss: 0.0190 -
acc: 0.9939 - val_loss: 0.1286 - val_acc: 0.9734
Epoch 9/25
40199/40199 [==============================] - 9s 212us/sample - loss: 0.0166 -
acc: 0.9948 - val_loss: 0.1236 - val_acc: 0.9744
Epoch 10/25
40199/40199 [==============================] - 9s 216us/sample - loss: 0.0134 -
acc: 0.9955 - val_loss: 0.1513 - val_acc: 0.9722
Epoch 11/25
40199/40199 [==============================] - 9s 213us/sample - loss: 0.0137 -
acc: 0.9957 - val_loss: 0.1512 - val_acc: 0.9704
Epoch 12/25
40199/40199 [==============================] - 8s 208us/sample - loss: 0.0131 -
acc: 0.9958 - val_loss: 0.1580 - val_acc: 0.9731
Epoch 13/25
40199/40199 [==============================] - 8s 211us/sample - loss: 0.0108 -
acc: 0.9963 - val_loss: 0.1690 - val_acc: 0.9714
Epoch 14/25
40199/40199 [==============================] - 9s 217us/sample - loss: 0.0115 -
acc: 0.9964 - val_loss: 0.1705 - val_acc: 0.9726
Epoch 15/25
40199/40199 [==============================] - 9s 212us/sample - loss: 0.0093 -
acc: 0.9970 - val_loss: 0.1764 - val_acc: 0.9725
Epoch 16/25
40199/40199 [==============================] - 8s 211us/sample - loss: 0.0089 -
acc: 0.9971 - val_loss: 0.1716 - val_acc: 0.9752
Epoch 17/25
```

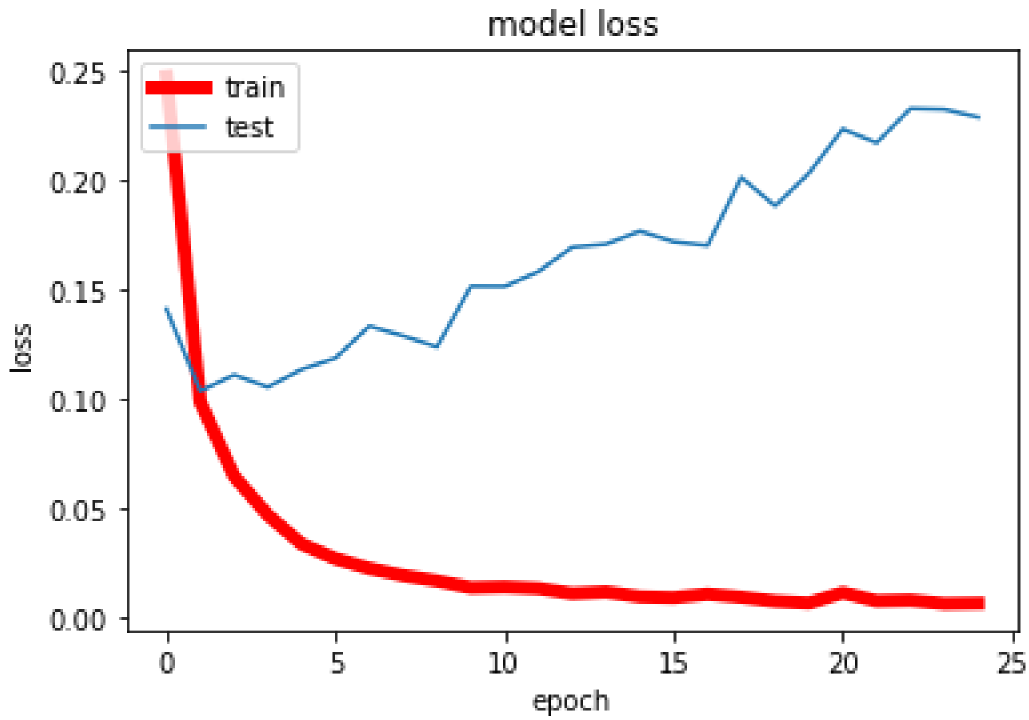
```
40199/40199 [==============================] - 9s 213us/sample - loss: 0.0105 -
acc: 0.9967 - val_loss: 0.1698 - val_acc: 0.9754
Epoch 18/25
40199/40199 [==============================] - 9s 213us/sample - loss: 0.0091 -
acc: 0.9972 - val_loss: 0.2010 - val_acc: 0.9739
Epoch 19/25
40199/40199 [==============================] - 9s 212us/sample - loss: 0.0072 -
acc: 0.9977 - val_loss: 0.1879 - val_acc: 0.9747
Epoch 20/25
40199/40199 [==============================] - 9s 213us/sample - loss: 0.0064 -
acc: 0.9980 - val_loss: 0.2029 - val_acc: 0.9739
Epoch 21/25
40199/40199 [==============================] - 9s 213us/sample - loss: 0.0111 -
acc: 0.9974 - val_loss: 0.2231 - val_acc: 0.9714
Epoch 22/25
40199/40199 [==============================] - 8s 210us/sample - loss: 0.0074 -
acc: 0.9977 - val_loss: 0.2168 - val_acc: 0.9720
Epoch 23/25
40199/40199 [==============================] - 9s 214us/sample - loss: 0.0077 -
acc: 0.9977 - val_loss: 0.2325 - val_acc: 0.9728
Epoch 24/25
40199/40199 [==============================] - 9s 218us/sample - loss: 0.0061 -
acc: 0.9980 - val_loss: 0.2321 - val_acc: 0.9732
Epoch 25/25
40199/40199 [==============================] - 9s 212us/sample - loss: 0.0063 -
acc: 0.9982 - val_loss: 0.2285 - val_acc: 0.9733
```

[149]:

```
#model= classifier_e25.fit(X_train, y_train, validation_split=0.33, epochs=10,␣
↪batch_size=10)

loss=list(model.history.values())[0]
accuracy=list(model.history.values())[1]
val_loss=list(model.history.values())[2]
val_accuracy=list(model.history.values())[3]

# summarize history for accuracy
plt.plot(accuracy,color='orange', linewidth=5)
plt.plot(val_accuracy)
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(loss, color='red', linewidth=5)
plt.plot(val_loss)
plt.title('model loss')
```

```
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```





1.5 Air Quality Prediction Using Machine Learning

[150]:
```
import pandas as pd
import numpy  as np
```

[151]:
```
import warnings
warnings.filterwarnings("ignore")
```

[152]:
```
airdata=pd.read_csv('station_day.csv')
airdata
```

[152]:
```
       StationId        Date  PM2.5    PM10     NO     NO2    NOx    NH3  \
0          AP001  2017-11-24  71.36  115.75   1.75   20.65  12.40  12.19
1          AP001  2017-11-25  81.40  124.50   1.44   20.50  12.08  10.72
2          AP001  2017-11-26  78.32  129.06   1.26   26.00  14.85  10.28
3          AP001  2017-11-27  88.76  135.32   6.60   30.85  21.77  12.91
4          AP001  2017-11-28  64.18  104.09   2.56   28.07  17.01  11.42
…          …           …   …     …      …    …     …     …
108030     WB013  2020-06-27   8.65   16.46    NaN     NaN    NaN    NaN
108031     WB013  2020-06-28  11.80   18.47    NaN     NaN    NaN    NaN
108032     WB013  2020-06-29  18.60   32.26  13.65  200.87 214.20  11.40
108033     WB013  2020-06-30  16.07   39.30   7.56   29.13  36.69  29.26
```

```
108034     WB013  2020-07-01  10.50   36.50   7.78   22.50   30.25  27.23

           CO    SO2      O3  Benzene  Toluene  Xylene    AQI    AQI_Bucket
0        0.10  10.76  109.26     0.17     5.92    0.10    NaN           NaN
1        0.12  15.24  127.09     0.20     6.50    0.06  184.0      Moderate
2        0.14  26.96  117.44     0.22     7.95    0.08  197.0      Moderate
3        0.11  33.59  111.81     0.29     7.63    0.12  198.0      Moderate
4        0.09  19.00  138.18     0.17     5.02    0.07  188.0      Moderate
…          …     …       …        …        …   …    …            …
108030   0.69   4.36   30.59     1.32     7.26     NaN   50.0          Good
108031   0.68   3.49   38.95     1.42     7.92     NaN   65.0  Satisfactory
108032   0.78   5.12   38.17     3.52     8.64     NaN   63.0  Satisfactory
108033   0.69   5.88   29.64     1.86     8.40     NaN   57.0  Satisfactory
108034   0.58   2.80   13.10     1.31     7.39     NaN   59.0  Satisfactory

[108035 rows x 16 columns]
```

[153]:
```
airdata.shape
```

[153]:
```
(108035, 16)
```

Notice from the description of the data below, Each column has alot of missing data

[154]:
```
airdata.describe()
```

[154]:
```
              PM2.5          PM10            NO           NO2           NOx  \
count  86410.000000  65329.000000  90929.000000  91488.000000  92535.000000
mean      80.272571    157.968427     23.123424     35.240760     41.195055
std       76.526403    123.418672     34.491019     29.510827     45.145976
min        0.020000      0.010000      0.010000      0.010000      0.000000
25%       31.880000     70.150000      4.840000     15.090000     13.970000
50%       55.950000    122.090000     10.290000     27.210000     26.660000
75%       99.920000    208.670000     24.980000     46.930000     50.500000
max     1000.000000   1000.000000    470.000000    448.050000    467.630000

                NH3            CO           SO2            O3       Benzene  \
count  59930.000000  95037.000000  82831.000000  82467.000000  76580.000000
mean      28.732875      1.605749     12.257634     38.134836      3.358029
std       24.897797      4.369578     12.984723     39.128004     11.156234
min        0.010000      0.000000      0.010000      0.010000      0.000000
25%       11.900000      0.530000      5.040000     18.895000      0.160000
50%       23.590000      0.910000      8.950000     30.840000      1.210000
75%       38.137500      1.450000     14.920000     47.140000      3.610000
max      418.900000    175.810000    195.650000    963.000000    455.030000

            Toluene        Xylene           AQI
count  69333.000000  22898.000000  87025.000000
```

```
mean     15.345394    2.423446   179.749290
std      29.348587    6.472409   131.324339
min       0.000000    0.000000     8.000000
25%       0.690000    0.000000    86.000000
50%       4.330000    0.400000   132.000000
75%      17.510000    2.110000   254.000000
max     454.850000  170.370000  2049.000000
```

Now have created a new data set, by deleting rows with missing values.

[155]:
```
data = airdata.dropna(axis = 0, how ='any')
data.shape
```

[155]:
```
(10314, 16)
```

[156]:
```
data.columns
```

[156]:
```
Index(['StationId', 'Date', 'PM2.5', 'PM10', 'NO', 'NO2', 'NOx', 'NH3', 'CO',
       'SO2', 'O3', 'Benzene', 'Toluene', 'Xylene', 'AQI', 'AQI_Bucket'],
      dtype='object')
```

[157]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10314 entries, 1 to 106147
Data columns (total 16 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   StationId   10314 non-null  object
 1   Date        10314 non-null  object
 2   PM2.5       10314 non-null  float64
 3   PM10        10314 non-null  float64
 4   NO          10314 non-null  float64
 5   NO2         10314 non-null  float64
 6   NOx         10314 non-null  float64
 7   NH3         10314 non-null  float64
 8   CO          10314 non-null  float64
 9   SO2         10314 non-null  float64
 10  O3          10314 non-null  float64
 11  Benzene     10314 non-null  float64
 12  Toluene     10314 non-null  float64
 13  Xylene      10314 non-null  float64
 14  AQI         10314 non-null  float64
 15  AQI_Bucket  10314 non-null  object
dtypes: float64(13), object(3)
memory usage: 1.3+ MB
```

[158]:
```
data.AQI_Bucket.unique()
```

[158]:
```
array(['Moderate', 'Poor', 'Very Poor', 'Satisfactory', 'Good', 'Severe'],
      dtype=object)
```

[159]:
```
print(data.NO2.min())
print(data.NO2.max())
```

```
0.01
254.78
```

[160]:
```
pd.Categorical(data['AQI_Bucket']).describe()
```

[160]:

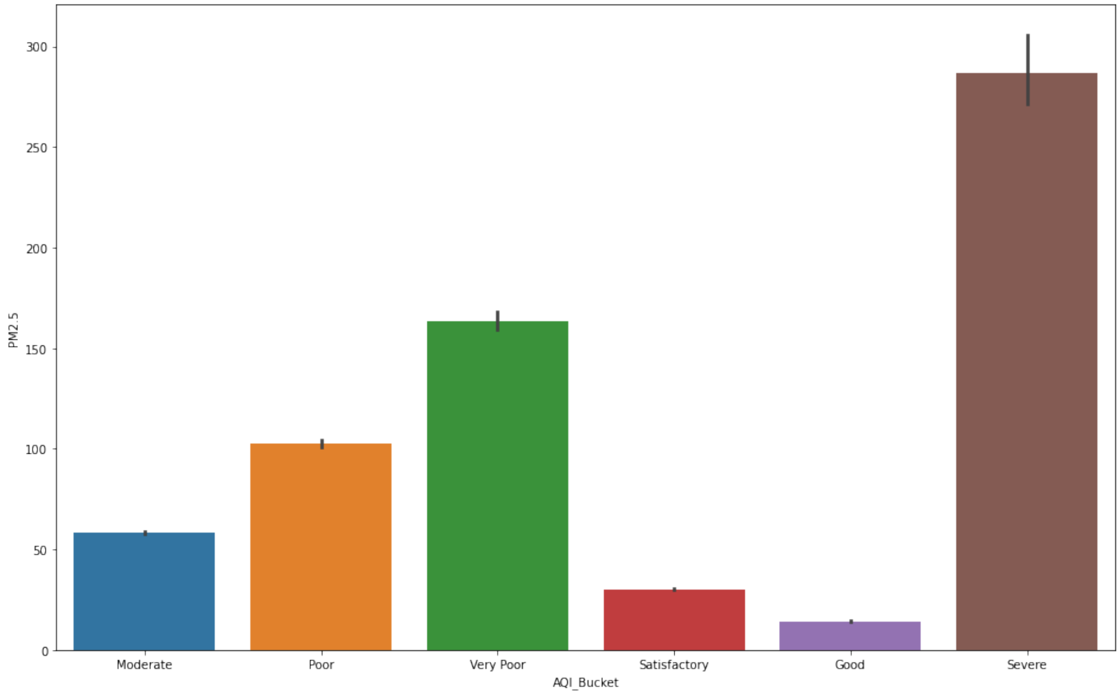
	counts	freqs
categories		
Good	1073	0.104033
Moderate	4402	0.426799
Poor	610	0.059143
Satisfactory	3730	0.361644
Severe	100	0.009696
Very Poor	399	0.038685

[161]:
```
data.columns
```

[161]:
```
Index(['StationId', 'Date', 'PM2.5', 'PM10', 'NO', 'NO2', 'NOx', 'NH3', 'CO',
       'SO2', 'O3', 'Benzene', 'Toluene', 'Xylene', 'AQI', 'AQI_Bucket'],
      dtype='object')
```

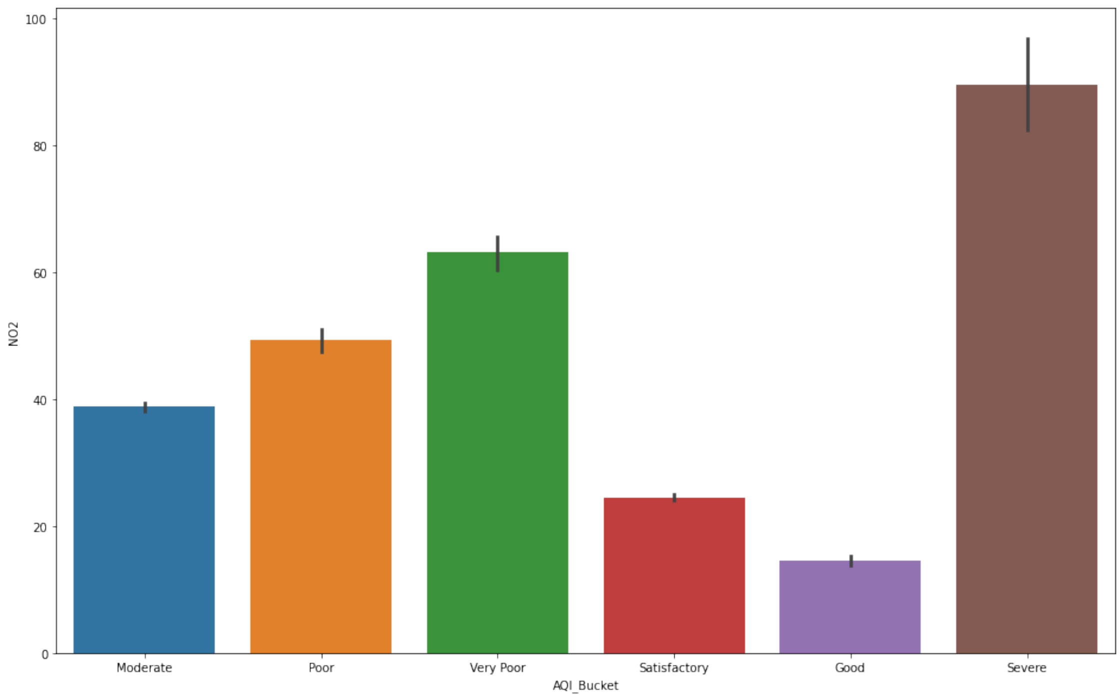
[162]:
```
import matplotlib.pyplot as plt
import seaborn as s

fig,ax=plt.subplots(figsize=(16,10))
ax=s.barplot(x='AQI_Bucket',y='PM2.5', data=data)
plt.show
```

[162]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



[163]:
```
import matplotlib.pyplot as plt
import seaborn as s

fig,ax=plt.subplots(figsize=(16,10))
ax=s.barplot(x='AQI_Bucket',y='NO2', data=data)
plt.show
```

[163]: <function matplotlib.pyplot.show(close=None, block=None)>



[1]:
```
#Heatmap plot Diagram
#fig,ax=plt.subplots(figsize=(16,10))
#s.heatmap(data.corr(),ax=ax, annot=True)
```

1.6 CHECKING AIR QUALITY INDEX USING DECISION TREE

[165]:
```
#Importing useful libraries
import pandas as pd
#import numpy as np
import sklearn
from sklearn import linear_model
from sklearn.utils import shuffle
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn import tree

airdata=pd.read_csv('station_day.csv')


New_data=data.iloc[:,[2,3,4,5,6,7,8,9,10,11,12,13,14,15]]

New_data.head()
```

[165]:
```
    PM2.5    PM10    NO    NO2    NOx    NH3    CO    SO2      O3  Benzene  \
1  81.40  124.50  1.44  20.50  12.08  10.72  0.12  15.24  127.09     0.20
2  78.32  129.06  1.26  26.00  14.85  10.28  0.14  26.96  117.44     0.22
3  88.76  135.32  6.60  30.85  21.77  12.91  0.11  33.59  111.81     0.29
4  64.18  104.09  2.56  28.07  17.01  11.42  0.09  19.00  138.18     0.17
5  72.47  114.84  5.23  23.20  16.59  12.25  0.16  10.55  109.74     0.21

   Toluene  Xylene    AQI AQI_Bucket
1     6.50    0.06  184.0   Moderate
2     7.95    0.08  197.0   Moderate
3     7.63    0.12  198.0   Moderate
4     5.02    0.07  188.0   Moderate
5     4.71    0.08  173.0   Moderate
```

[166]:
```
mydata=New_data
X=mydata.drop(columns='AQI_Bucket')
y=mydata['AQI_Bucket']

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2)

model=DecisionTreeClassifier()
model.fit(X_train,y_train)

predictions=model.predict(X_test)

A=[[y_test],[predictions]]
score=accuracy_score(y_test,predictions)

#tree.export_graphviz(model,out_file='music_recommender.
 ↪dot',feature_names=['age','sex'],class_names=sorted(y.
 ↪unique()),label='all',rounded=True,filled=True)
score
```

[166]: 1.0

1.7 DEEP LEARNING ALGORITHM FOR PREDICTING THE AIR QUALITY

[167]:
```
#Importing useful libraries
import pandas as pd
#import numpy as np
import sklearn
from sklearn import linear_model
from sklearn.utils import shuffle
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
```

```python
from sklearn.metrics import accuracy_score
from sklearn import tree

airdata=pd.read_csv('station_day.csv')


New_data=data.iloc[:,[2,3,4,5,6,7,8,9,10,11,12,13,14,15]]

dataset=New_data

from sklearn.impute import SimpleImputer

imputer = SimpleImputer(missing_values= np.nan, strategy= 'mean')

imputer = imputer.fit(dataset.iloc[:, 0:13])
X= imputer.transform(dataset.iloc[:, 0:13])

y=list(dataset.iloc[:,13])

from sklearn.preprocessing import LabelEncoder
encoder=LabelEncoder
#y1=encoder.fit_transform(y)
Y=pd.get_dummies(y)
for i in range(len(y)) :
    if(y[i]=='Very Poor'):
        y[i]=1
    elif (y[i]=='Poor'):
        y[i]=2
    elif (y[i]=='Moderate'):
        y[i]=3
    elif (y[i]=='Satisfactory'):
        y[i]=4
    elif (y[i]=='Good'):
        y[i]=5
    else:
        y[i]=6





#initializing CNN model
classifier_e25 = Sequential()
#add 1st hidden layer
classifier_e25.add(Dense(input_dim = 13, units = 13,␣
 ↪kernel_initializer='uniform', activation='relu'))
#add output layer
```

```python
classifier_e25.add(Dense(units = 13, kernel_initializer='uniform',␣
↪activation='softmax'))
#compile the neural network
classifier_e25.compile(optimizer='adam',␣
↪loss='sparse_categorical_crossentropy', metrics=['accuracy'])
#model summary
classifier_e25.summary()
```

```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_18 (Dense)             (None, 13)                182
_________________________________________________________________
dense_19 (Dense)             (None, 13)                182
=================================================================
Total params: 364
Trainable params: 364
Non-trainable params: 0
_________________________________________________________________
```

[168]:
```python
model=classifier_e25.fit(X, y, validation_split=0.33, epochs=100, batch_size=10)
```

```
Train on 6910 samples, validate on 3404 samples
Epoch 1/100
6910/6910 [==============================] - 1s 163us/sample - loss: 1.1402 -
acc: 0.5401 - val_loss: 1.0592 - val_acc: 0.5076
Epoch 2/100
6910/6910 [==============================] - 1s 165us/sample - loss: 0.8598 -
acc: 0.6253 - val_loss: 1.0988 - val_acc: 0.5308
Epoch 3/100
6910/6910 [==============================] - 1s 133us/sample - loss: 0.7840 -
acc: 0.6677 - val_loss: 0.9924 - val_acc: 0.6266
Epoch 4/100
6910/6910 [==============================] - 1s 132us/sample - loss: 0.7170 -
acc: 0.7133 - val_loss: 0.9615 - val_acc: 0.7045
Epoch 5/100
6910/6910 [==============================] - 1s 133us/sample - loss: 0.6532 -
acc: 0.7499 - val_loss: 0.8814 - val_acc: 0.7115
Epoch 6/100
6910/6910 [==============================] - 1s 133us/sample - loss: 0.5945 -
acc: 0.7832 - val_loss: 0.8670 - val_acc: 0.7206
Epoch 7/100
6910/6910 [==============================] - 1s 144us/sample - loss: 0.5349 -
acc: 0.8127 - val_loss: 0.7805 - val_acc: 0.7938
Epoch 8/100
6910/6910 [==============================] - 1s 135us/sample - loss: 0.4835 -
```

```
acc: 0.8285 - val_loss: 0.7018 - val_acc: 0.8237
Epoch 9/100
6910/6910 [==============================] - 1s 134us/sample - loss: 0.4411 -
acc: 0.8492 - val_loss: 0.7508 - val_acc: 0.8314
Epoch 10/100
6910/6910 [==============================] - 1s 132us/sample - loss: 0.4031 -
acc: 0.8630 - val_loss: 0.6568 - val_acc: 0.8273
Epoch 11/100
6910/6910 [==============================] - 1s 132us/sample - loss: 0.3774 -
acc: 0.8650 - val_loss: 0.6794 - val_acc: 0.8543
Epoch 12/100
6910/6910 [==============================] - 1s 134us/sample - loss: 0.3492 -
acc: 0.8771 - val_loss: 0.5850 - val_acc: 0.8422
Epoch 13/100
6910/6910 [==============================] - 1s 147us/sample - loss: 0.3288 -
acc: 0.8839 - val_loss: 0.5227 - val_acc: 0.8831
Epoch 14/100
6910/6910 [==============================] - 1s 134us/sample - loss: 0.3115 -
acc: 0.8983 - val_loss: 0.5619 - val_acc: 0.8546
Epoch 15/100
6910/6910 [==============================] - 1s 131us/sample - loss: 0.2983 -
acc: 0.8952 - val_loss: 0.5105 - val_acc: 0.8687
Epoch 16/100
6910/6910 [==============================] - 1s 131us/sample - loss: 0.2829 -
acc: 0.8991 - val_loss: 0.5356 - val_acc: 0.8505
Epoch 17/100
6910/6910 [==============================] - 1s 140us/sample - loss: 0.2671 -
acc: 0.9091 - val_loss: 0.4559 - val_acc: 0.8916
Epoch 18/100
6910/6910 [==============================] - 1s 152us/sample - loss: 0.2587 -
acc: 0.9109 - val_loss: 0.4935 - val_acc: 0.8904
Epoch 19/100
6910/6910 [==============================] - 1s 135us/sample - loss: 0.2528 -
acc: 0.9116 - val_loss: 0.4668 - val_acc: 0.9130
Epoch 20/100
6910/6910 [==============================] - 1s 138us/sample - loss: 0.2435 -
acc: 0.9107 - val_loss: 0.3956 - val_acc: 0.8848
Epoch 21/100
6910/6910 [==============================] - 1s 131us/sample - loss: 0.2335 -
acc: 0.9208 - val_loss: 0.3779 - val_acc: 0.9227
Epoch 22/100
6910/6910 [==============================] - 1s 132us/sample - loss: 0.2276 -
acc: 0.9185 - val_loss: 0.4221 - val_acc: 0.8751
Epoch 23/100
6910/6910 [==============================] - 1s 141us/sample - loss: 0.2251 -
acc: 0.9182 - val_loss: 0.3602 - val_acc: 0.9036
Epoch 24/100
6910/6910 [==============================] - 1s 138us/sample - loss: 0.2163 -
```

```
acc: 0.9205 - val_loss: 0.3236 - val_acc: 0.9145
Epoch 25/100
6910/6910 [==============================] - 1s 133us/sample - loss: 0.2109 -
acc: 0.9256 - val_loss: 0.4417 - val_acc: 0.9145
Epoch 26/100
6910/6910 [==============================] - 1s 131us/sample - loss: 0.2079 -
acc: 0.9229 - val_loss: 0.3264 - val_acc: 0.9119
Epoch 27/100
6910/6910 [==============================] - 1s 150us/sample - loss: 0.2012 -
acc: 0.9245 - val_loss: 0.3711 - val_acc: 0.9142
Epoch 28/100
6910/6910 [==============================] - 1s 148us/sample - loss: 0.2000 -
acc: 0.9278 - val_loss: 0.3139 - val_acc: 0.9160
Epoch 29/100
6910/6910 [==============================] - 1s 165us/sample - loss: 0.1925 -
acc: 0.9275 - val_loss: 0.3230 - val_acc: 0.9239
Epoch 30/100
6910/6910 [==============================] - 1s 143us/sample - loss: 0.1898 -
acc: 0.9317 - val_loss: 0.3615 - val_acc: 0.9069
Epoch 31/100
6910/6910 [==============================] - 1s 143us/sample - loss: 0.1830 -
acc: 0.9333 - val_loss: 0.3222 - val_acc: 0.9051
Epoch 32/100
6910/6910 [==============================] - 1s 152us/sample - loss: 0.1867 -
acc: 0.9330 - val_loss: 0.2783 - val_acc: 0.9345
Epoch 33/100
6910/6910 [==============================] - 1s 161us/sample - loss: 0.1751 -
acc: 0.9407 - val_loss: 0.2893 - val_acc: 0.9266
Epoch 34/100
6910/6910 [==============================] - 1s 150us/sample - loss: 0.1812 -
acc: 0.9313 - val_loss: 0.3349 - val_acc: 0.9260
Epoch 35/100
6910/6910 [==============================] - 1s 153us/sample - loss: 0.1754 -
acc: 0.9399 - val_loss: 0.2721 - val_acc: 0.9166
Epoch 36/100
6910/6910 [==============================] - 1s 162us/sample - loss: 0.1750 -
acc: 0.9311 - val_loss: 0.3339 - val_acc: 0.9104
Epoch 37/100
6910/6910 [==============================] - 1s 141us/sample - loss: 0.1790 -
acc: 0.9305 - val_loss: 0.2514 - val_acc: 0.9365
Epoch 38/100
6910/6910 [==============================] - 1s 179us/sample - loss: 0.1656 -
acc: 0.9370 - val_loss: 0.2571 - val_acc: 0.9239
Epoch 39/100
6910/6910 [==============================] - 1s 144us/sample - loss: 0.1665 -
acc: 0.9382 - val_loss: 0.2390 - val_acc: 0.9183
Epoch 40/100
6910/6910 [==============================] - 1s 141us/sample - loss: 0.1559 -
```

```
acc: 0.9436 - val_loss: 0.3249 - val_acc: 0.8722
Epoch 41/100
6910/6910 [==============================] - 1s 139us/sample - loss: 0.1653 -
acc: 0.9370 - val_loss: 0.2584 - val_acc: 0.9239
Epoch 42/100
6910/6910 [==============================] - 1s 140us/sample - loss: 0.1551 -
acc: 0.9444 - val_loss: 0.2866 - val_acc: 0.9213
Epoch 43/100
6910/6910 [==============================] - 1s 145us/sample - loss: 0.1591 -
acc: 0.9365 - val_loss: 0.2327 - val_acc: 0.9321
Epoch 44/100
6910/6910 [==============================] - 1s 138us/sample - loss: 0.1489 -
acc: 0.9452 - val_loss: 0.2839 - val_acc: 0.8890
Epoch 45/100
6910/6910 [==============================] - 1s 133us/sample - loss: 0.1604 -
acc: 0.9381 - val_loss: 0.2196 - val_acc: 0.9462
Epoch 46/100
6910/6910 [==============================] - 1s 133us/sample - loss: 0.1556 -
acc: 0.9394 - val_loss: 0.3635 - val_acc: 0.8634
Epoch 47/100
6910/6910 [==============================] - 1s 150us/sample - loss: 0.1545 -
acc: 0.9407 - val_loss: 0.2097 - val_acc: 0.9222
Epoch 48/100
6910/6910 [==============================] - 1s 140us/sample - loss: 0.1453 -
acc: 0.9447 - val_loss: 0.1882 - val_acc: 0.9410
Epoch 49/100
6910/6910 [==============================] - 1s 135us/sample - loss: 0.1441 -
acc: 0.9420 - val_loss: 0.2195 - val_acc: 0.9233
Epoch 50/100
6910/6910 [==============================] - 1s 132us/sample - loss: 0.1446 -
acc: 0.9447 - val_loss: 0.2084 - val_acc: 0.9251
Epoch 51/100
6910/6910 [==============================] - 1s 143us/sample - loss: 0.1484 -
acc: 0.9453 - val_loss: 0.1801 - val_acc: 0.9357
Epoch 52/100
6910/6910 [==============================] - 1s 134us/sample - loss: 0.1508 -
acc: 0.9379 - val_loss: 0.2961 - val_acc: 0.8913
Epoch 53/100
6910/6910 [==============================] - 1s 132us/sample - loss: 0.1420 -
acc: 0.9440 - val_loss: 0.1894 - val_acc: 0.9407
Epoch 54/100
6910/6910 [==============================] - 1s 144us/sample - loss: 0.1398 -
acc: 0.9454 - val_loss: 0.2530 - val_acc: 0.9210
Epoch 55/100
6910/6910 [==============================] - 1s 131us/sample - loss: 0.1386 -
acc: 0.9454 - val_loss: 0.1604 - val_acc: 0.9501
Epoch 56/100
6910/6910 [==============================] - 1s 132us/sample - loss: 0.1344 -
```

```
acc: 0.9485 - val_loss: 0.1722 - val_acc: 0.9548
Epoch 57/100
6910/6910 [==============================] - 1s 131us/sample - loss: 0.1361 -
acc: 0.9467 - val_loss: 0.1811 - val_acc: 0.9468
Epoch 58/100
6910/6910 [==============================] - 1s 138us/sample - loss: 0.1363 -
acc: 0.9463 - val_loss: 0.2011 - val_acc: 0.9339
Epoch 59/100
6910/6910 [==============================] - 1s 150us/sample - loss: 0.1351 -
acc: 0.9446 - val_loss: 0.2302 - val_acc: 0.9069
Epoch 60/100
6910/6910 [==============================] - 1s 131us/sample - loss: 0.1351 -
acc: 0.9456 - val_loss: 0.1722 - val_acc: 0.9477
Epoch 61/100
6910/6910 [==============================] - 1s 132us/sample - loss: 0.1272 -
acc: 0.9546 - val_loss: 0.2002 - val_acc: 0.9201
Epoch 62/100
6910/6910 [==============================] - 1s 131us/sample - loss: 0.1266 -
acc: 0.9520 - val_loss: 0.2394 - val_acc: 0.9075
Epoch 63/100
6910/6910 [==============================] - 1s 130us/sample - loss: 0.1419 -
acc: 0.9437 - val_loss: 0.2204 - val_acc: 0.9127
Epoch 64/100
6910/6910 [==============================] - 1s 137us/sample - loss: 0.1239 -
acc: 0.9520 - val_loss: 0.1556 - val_acc: 0.9551
Epoch 65/100
6910/6910 [==============================] - 1s 144us/sample - loss: 0.1292 -
acc: 0.9473 - val_loss: 0.1835 - val_acc: 0.9483
Epoch 66/100
6910/6910 [==============================] - 1s 137us/sample - loss: 0.1243 -
acc: 0.9541 - val_loss: 0.1778 - val_acc: 0.9512
Epoch 67/100
6910/6910 [==============================] - 1s 145us/sample - loss: 0.1274 -
acc: 0.9515 - val_loss: 0.1699 - val_acc: 0.9539
Epoch 68/100
6910/6910 [==============================] - 1s 127us/sample - loss: 0.1332 -
acc: 0.9423 - val_loss: 0.1752 - val_acc: 0.9468
Epoch 69/100
6910/6910 [==============================] - 1s 127us/sample - loss: 0.1346 -
acc: 0.9441 - val_loss: 0.2335 - val_acc: 0.9342
Epoch 70/100
6910/6910 [==============================] - 1s 142us/sample - loss: 0.1266 -
acc: 0.9472 - val_loss: 0.1762 - val_acc: 0.9583
Epoch 71/100
6910/6910 [==============================] - 1s 128us/sample - loss: 0.1240 -
acc: 0.9485 - val_loss: 0.1607 - val_acc: 0.9439
Epoch 72/100
6910/6910 [==============================] - 1s 127us/sample - loss: 0.1243 -
```

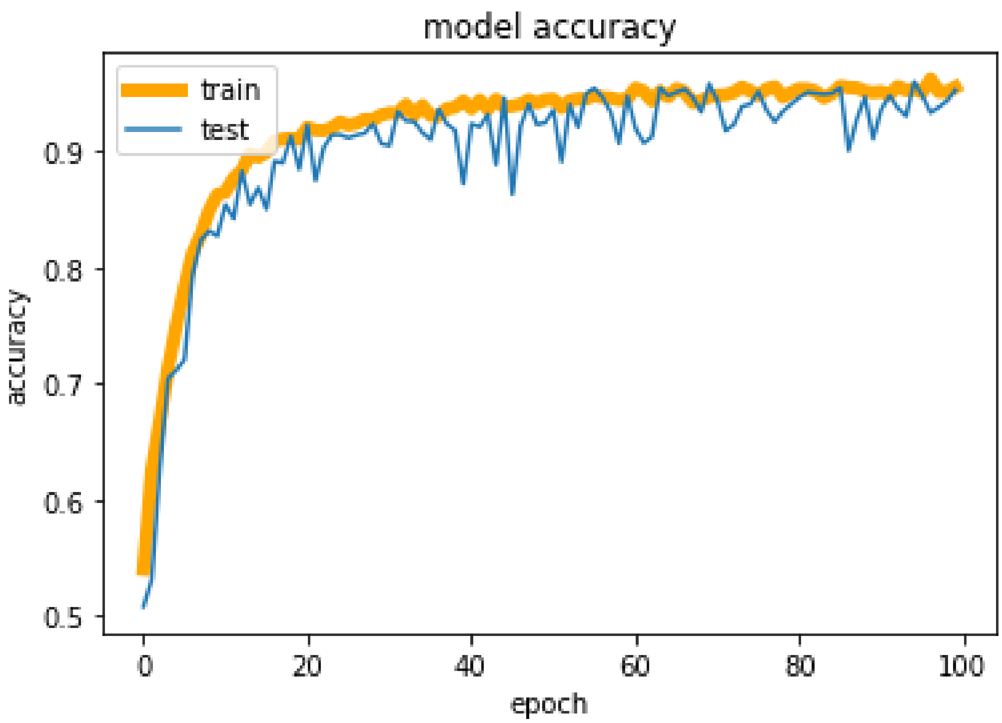
```
acc: 0.9489 - val_loss: 0.2596 - val_acc: 0.9180
Epoch 73/100
6910/6910 [==============================] - 1s 127us/sample - loss: 0.1250 -
acc: 0.9511 - val_loss: 0.2231 - val_acc: 0.9233
Epoch 74/100
6910/6910 [==============================] - 1s 127us/sample - loss: 0.1179 -
acc: 0.9554 - val_loss: 0.2556 - val_acc: 0.9389
Epoch 75/100
6910/6910 [==============================] - 1s 132us/sample - loss: 0.1226 -
acc: 0.9522 - val_loss: 0.2199 - val_acc: 0.9410
Epoch 76/100
6910/6910 [==============================] - 1s 137us/sample - loss: 0.1255 -
acc: 0.9485 - val_loss: 0.1591 - val_acc: 0.9518
Epoch 77/100
6910/6910 [==============================] - 1s 128us/sample - loss: 0.1176 -
acc: 0.9538 - val_loss: 0.1869 - val_acc: 0.9357
Epoch 78/100
6910/6910 [==============================] - 1s 130us/sample - loss: 0.1152 -
acc: 0.9559 - val_loss: 0.2140 - val_acc: 0.9257
Epoch 79/100
6910/6910 [==============================] - 1s 139us/sample - loss: 0.1273 -
acc: 0.9470 - val_loss: 0.2387 - val_acc: 0.9348
Epoch 80/100
6910/6910 [==============================] - 1s 132us/sample - loss: 0.1219 -
acc: 0.9509 - val_loss: 0.2004 - val_acc: 0.9410
Epoch 81/100
6910/6910 [==============================] - 1s 145us/sample - loss: 0.1164 -
acc: 0.9541 - val_loss: 0.1610 - val_acc: 0.9468
Epoch 82/100
6910/6910 [==============================] - 1s 134us/sample - loss: 0.1171 -
acc: 0.9538 - val_loss: 0.1542 - val_acc: 0.9512
Epoch 83/100
6910/6910 [==============================] - 1s 133us/sample - loss: 0.1231 -
acc: 0.9520 - val_loss: 0.1934 - val_acc: 0.9504
Epoch 84/100
6910/6910 [==============================] - 1s 131us/sample - loss: 0.1251 -
acc: 0.9462 - val_loss: 0.1944 - val_acc: 0.9495
Epoch 85/100
6910/6910 [==============================] - 1s 136us/sample - loss: 0.1173 -
acc: 0.9501 - val_loss: 0.1775 - val_acc: 0.9504
Epoch 86/100
6910/6910 [==============================] - 1s 152us/sample - loss: 0.1128 -
acc: 0.9564 - val_loss: 0.1516 - val_acc: 0.9551
Epoch 87/100
6910/6910 [==============================] - 1s 148us/sample - loss: 0.1105 -
acc: 0.9553 - val_loss: 0.2503 - val_acc: 0.9007
Epoch 88/100
6910/6910 [==============================] - 1s 140us/sample - loss: 0.1124 -
```

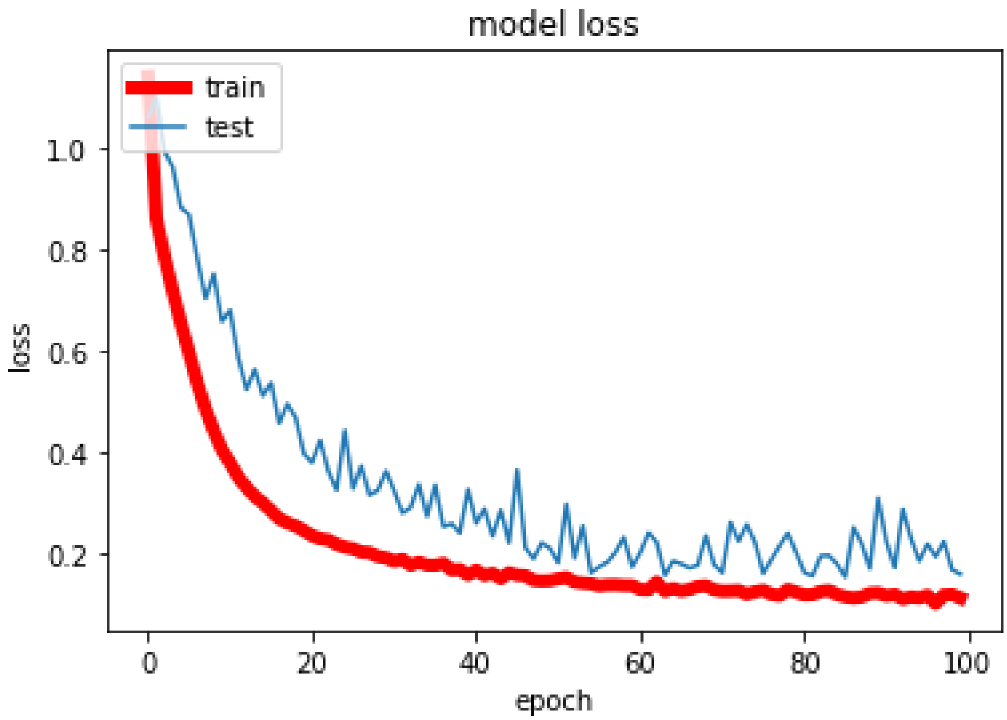
```
acc: 0.9548 - val_loss: 0.2180 - val_acc: 0.9277
Epoch 89/100
6910/6910 [==============================] - 1s 137us/sample - loss: 0.1196 -
acc: 0.9520 - val_loss: 0.1682 - val_acc: 0.9474
Epoch 90/100
6910/6910 [==============================] - 1s 166us/sample - loss: 0.1205 -
acc: 0.9509 - val_loss: 0.3081 - val_acc: 0.9110
Epoch 91/100
6910/6910 [==============================] - 1s 200us/sample - loss: 0.1149 -
acc: 0.9517 - val_loss: 0.2230 - val_acc: 0.9365
Epoch 92/100
6910/6910 [==============================] - 2s 235us/sample - loss: 0.1171 -
acc: 0.9492 - val_loss: 0.1706 - val_acc: 0.9483
Epoch 93/100
6910/6910 [==============================] - 1s 200us/sample - loss: 0.1084 -
acc: 0.9562 - val_loss: 0.2857 - val_acc: 0.9374
Epoch 94/100
6910/6910 [==============================] - 1s 153us/sample - loss: 0.1124 -
acc: 0.9528 - val_loss: 0.2277 - val_acc: 0.9307
Epoch 95/100
6910/6910 [==============================] - 1s 174us/sample - loss: 0.1106 -
acc: 0.9537 - val_loss: 0.1838 - val_acc: 0.9600
Epoch 96/100
6910/6910 [==============================] - 1s 176us/sample - loss: 0.1158 -
acc: 0.9512 - val_loss: 0.2171 - val_acc: 0.9480
Epoch 97/100
6910/6910 [==============================] - 1s 138us/sample - loss: 0.0992 -
acc: 0.9625 - val_loss: 0.1924 - val_acc: 0.9336
Epoch 98/100
6910/6910 [==============================] - 1s 157us/sample - loss: 0.1165 -
acc: 0.9515 - val_loss: 0.2219 - val_acc: 0.9383
Epoch 99/100
6910/6910 [==============================] - 2s 233us/sample - loss: 0.1174 -
acc: 0.9515 - val_loss: 0.1665 - val_acc: 0.9439
Epoch 100/100
6910/6910 [==============================] - 2s 229us/sample - loss: 0.1105 -
acc: 0.9554 - val_loss: 0.1579 - val_acc: 0.9530
```

[169]:

```
loss=list(model.history.values())[0]
accuracy=list(model.history.values())[1]
val_loss=list(model.history.values())[2]
val_accuracy=list(model.history.values())[3]

# summarize history for accuracy
plt.plot(accuracy,color='orange', linewidth=5)
plt.plot(val_accuracy)
plt.title('model accuracy')
```

```
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(loss, color='red', linewidth=5)
plt.plot(val_loss)
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```





1.8 Matrix Decomposition

[170]:
```
# LU decomposition
from numpy import array
from scipy.linalg import lu
# define a square matrix
A = array([
[1, 2, 3],
[4, 5, 6],
[7, 8, 9]])
print(A)
# factorize
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

[171]:
```
P, L, U = lu(A)
print(P)
print(L)
print(U)
# reconstruct
```

```
B = P.dot(L).dot(U)
B
```

```
[[0. 1. 0.]
 [0. 0. 1.]
 [1. 0. 0.]]
[[1.         0.         0.        ]
 [0.14285714 1.         0.        ]
 [0.57142857 0.5        1.        ]]
[[7.         8.         9.        ]
 [0.         0.85714286 1.71428571]
 [0.         0.         0.        ]]
```

[171]:
```
array([[1., 2., 3.],
       [4., 5., 6.],
       [7., 8., 9.]])
```

[172]:
```
# QR decomposition
from numpy import array
from numpy.linalg import qr
# define rectangular matrix
A = array([
[1, 2],
[3, 4],
[5, 6]])
print(A)
# factorize
Q, R = qr(A, 'complete')
print(Q)
print(R)
# reconstruct
B = Q.dot(R)
print(B)
```

```
[[1 2]
 [3 4]
 [5 6]]
[[-0.16903085  0.89708523  0.40824829]
 [-0.50709255  0.27602622 -0.81649658]
 [-0.84515425 -0.34503278  0.40824829]]
[[-5.91607978 -7.43735744]
 [ 0.          0.82807867]
 [ 0.          0.        ]]
[[1. 2.]
 [3. 4.]
 [5. 6.]]
```

[173]:
```python
# Cholesky decomposition
from numpy import array
from numpy.linalg import cholesky
# define symmetrical matrix
A = array([
[2, 1, 1],
[1, 2, 1],
[1, 1, 2]])
print(A)
# factorize
L = cholesky(A)
print(L)
# reconstruct
B = L.dot(L.T)
print(B)
```

```
[[2 1 1]
 [1 2 1]
 [1 1 2]]
[[1.41421356 0.         0.        ]
 [0.70710678 1.22474487 0.        ]
 [0.70710678 0.40824829 1.15470054]]
[[2. 1. 1.]
 [1. 2. 1.]
 [1. 1. 2.]]
```

[174]:
```python
# eigendecomposition
from numpy import array
from numpy.linalg import eig
# define matrix
A = array([
[1, 2, 3],
[4, 5, 6],
[7, 8, 9]])
print(A)
# factorize
values, vectors = eig(A)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

[175]:
```python
print(values)
print(vectors)
```

```
[ 1.61168440e+01 -1.11684397e+00 -1.30367773e-15]
[[-0.23197069 -0.78583024  0.40824829]
 [-0.52532209 -0.08675134 -0.81649658]
```

```
 [-0.8186735   0.61232756  0.40824829]]
```

[176]:
```python
vectors.dot(values)
```

[176]:
```
array([ -2.86098561,  -8.3696465 , -13.87830739])
```

[177]:
```python
# singular-value decomposition
from numpy import array
from scipy.linalg import svd
# define a matrix
A = array([
[1, 2],
[3, 4],
[5, 6]])
print(A)
# factorize
U, s, V = svd(A)
print(U)
print(s)
print(V)
```

```
[[1 2]
 [3 4]
 [5 6]]
[[-0.2298477   0.88346102  0.40824829]
 [-0.52474482  0.24078249 -0.81649658]
 [-0.81964194 -0.40189603  0.40824829]]
[9.52551809 0.51430058]
[[-0.61962948 -0.78489445]
 [-0.78489445  0.61962948]]
```

1.9 Calculate Principal Component Analysis

[178]:
```python
# principal component analysis
from numpy import array
from numpy import mean
from numpy import cov
from numpy.linalg import eig
# define matrix
A = array([
[1, 2],
[3, 4],
[5, 6]])
print(A)
# column means
M = mean(A.T, axis=1)
# center columns by subtracting column means
```

```
C = A - M
# calculate covariance matrix of centered matrix
V = cov(C.T)
# factorize covariance matrix
values, vectors = eig(V)
print(vectors)
print(values)
# project data
P = vectors.T.dot(C.T)
print(P.T)
```

```
[[1 2]
 [3 4]
 [5 6]]
[[ 0.70710678 -0.70710678]
 [ 0.70710678  0.70710678]]
[8. 0.]
[[-2.82842712  0.        ]
 [ 0.          0.        ]
 [ 2.82842712  0.        ]]
```

1.10 Principal Component Analysis in scikit-learn

[179]:
```
# principal component analysis with scikit-learn
from numpy import array
from sklearn.decomposition import PCA
# define matrix
A = array([
[1, 2],
[3, 4],
[5, 6]])
print(A)
# create the transform
pca = PCA(2)
# fit transform
pca.fit(A)
# access values and vectors
print(pca.components_)
print(pca.explained_variance_)
# transform data
B = pca.transform(A)
print(B)
```

```
[[1 2]
 [3 4]
 [5 6]]
```

```
[[ 0.70710678  0.70710678]
 [-0.70710678  0.70710678]]
[8. 0.]
[[-2.82842712e+00 -2.22044605e-16]
 [ 0.00000000e+00  0.00000000e+00]
 [ 2.82842712e+00  2.22044605e-16]]
```