1. **Using the iris data set implement the KNN algorithm. Take different values for Test and training data set .Also use different values for k. Also find the accuracy level.**

   <u>CODE</u>

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
dataset = pd.read_csv("iris.csv")
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 4].values
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
from sklearn.metrics import classification_report, confusion_matrix
print(classification_report(y_test, y_pred))
from sklearn.metrics import accuracy_score
print ("Accuracy : ", accuracy_score(y_test, y_pred))
df = pd.DataFrame({'Real Values':y_test, 'Predicted Values':y_pred})
df
```

   <u>OUTPUT</u>

```
              precision    recall  f1-score   support

      Setosa       1.00      1.00      1.00         9
  Versicolor       1.00      0.80      0.89        10
   Virginica       0.85      1.00      0.92        11

    accuracy                           0.93        30
   macro avg       0.95      0.93      0.94        30
weighted avg       0.94      0.93      0.93        30

Accuracy :  0.9333333333333333
```

   Reference: https://stackabuse.com/k-nearest-neighbors-algorithm-in-python-and-scikit-learn/

2. **Download another data set suitable for the KNN and implement the KNN algorithm. Take different values for Test and training data set .Also use different values for k.**

   <u>CODE</u>

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
dataset = pd.read_csv("cancer.csv")
dataset.head()
dataset.info()
```

```
X = dataset.iloc[:, 2:35].values
print(X)
y = dataset.iloc[:, 1].values
print(y)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
from sklearn.metrics import classification_report, confusion_matrix
print(classification_report(y_test, y_pred))
from sklearn.metrics import accuracy_score
print ("Accuracy : ", accuracy_score(y_test, y_pred))
df = pd.DataFrame({'Real Values':y_test, 'Predicted Values':y_pred})
df
```

OUTPUT

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 568 entries, 0 to 567
Data columns (total 32 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   842302    568 non-null    int64
 1   M         568 non-null    object
 2   17.99     568 non-null    float64
 3   10.38     568 non-null    float64
 4   122.8     568 non-null    float64
 5   1001      568 non-null    float64
 6   0.1184    568 non-null    float64
 7   0.2776    568 non-null    float64
 8   0.3001    568 non-null    float64
 9   0.1471    568 non-null    float64
 10  0.2419    568 non-null    float64
 11  0.07871   568 non-null    float64
 12  1.095     568 non-null    float64
 13  0.9053    568 non-null    float64
```

```
 14  8.589     568 non-null    float64
 15  153.4     568 non-null    float64
 16  0.006399  568 non-null    float64
 17  0.04904   568 non-null    float64
 18  0.05373   568 non-null    float64
 19  0.01587   568 non-null    float64
 20  0.03003   568 non-null    float64
 21  0.006193  568 non-null    float64
 22  25.38     568 non-null    float64
 23  17.33     568 non-null    float64
 24  184.6     568 non-null    float64
 25  2019      568 non-null    float64
 26  0.1622    568 non-null    float64
 27  0.6656    568 non-null    float64
 28  0.7119    568 non-null    float64
 29  0.2654    568 non-null    float64
 30  0.4601    568 non-null    float64
 31  0.1189    568 non-null    float64
dtypes: float64(30), int64(1), object(1)
memory usage: 142.1+ KB
```

memory usage: 142.1+ KB

```
[[2.057e+01 1.777e+01 1.329e+02 ... 1.860e-01 2.750e-01 8.902e-02]
 [1.969e+01 2.125e+01 1.300e+02 ... 2.430e-01 3.613e-01 8.758e-02]
 [1.142e+01 2.038e+01 7.758e+01 ... 2.575e-01 6.638e-01 1.730e-01]
 ...
 [1.660e+01 2.808e+01 1.083e+02 ... 1.418e-01 2.218e-01 7.820e-02]
 [2.060e+01 2.933e+01 1.401e+02 ... 2.650e-01 4.087e-01 1.240e-01]
 [7.760e+00 2.454e+01 4.792e+01 ... 0.000e+00 2.871e-01 7.039e-02]]
['M' 'M' 'M' 'M' 'M' 'M' 'M' 'M' 'M' 'M' 'M' 'M' 'M' 'M' 'M' 'M' 'M' 'M'
 'B' 'B' 'B' 'M' 'M' 'M' 'M' 'M' 'M' 'M' 'M' 'M' 'M' 'M' 'M' 'M' 'M' 'M'
 'B' 'M' 'M' 'M' 'M' 'M' 'M' 'M' 'M' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'M' 'M'
 'B' 'M' 'M' 'B' 'B' 'B' 'B' 'M' 'B' 'M' 'M' 'B' 'B' 'B' 'B' 'M' 'B' 'M'
 'M' 'B' 'M' 'B' 'M' 'M' 'B' 'B' 'B' 'M' 'M' 'B' 'M' 'M' 'M' 'B' 'B' 'B'
 'M' 'B' 'B' 'M' 'M' 'B' 'B' 'B' 'M' 'M' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'M'
 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'M' 'M' 'B' 'M' 'M' 'B' 'B' 'B' 'M'
 'M' 'B' 'M' 'B' 'M' 'M' 'B' 'M' 'M' 'B' 'B' 'M' 'B' 'B' 'M' 'B' 'B' 'B'
 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'M' 'M'
 'B' 'M' 'B' 'B' 'M' 'M' 'B' 'B' 'M' 'M' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'M'
 'M' 'M' 'B' 'M' 'B' 'M' 'B' 'B' 'B' 'M' 'B' 'B' 'M' 'M' 'B' 'M' 'M' 'M'
 'M' 'B' 'M' 'M' 'M' 'B' 'M' 'B' 'M' 'B' 'B' 'M' 'B' 'M' 'M' 'M' 'M' 'B'
```

```
              precision    recall  f1-score   support

           B       0.96      0.93      0.94        71
           M       0.89      0.93      0.91        43

    accuracy                           0.93       114
   macro avg       0.92      0.93      0.93       114
weighted avg       0.93      0.93      0.93       114

Accuracy :  0.9298245614035088
```

Out[9]:

	Real Values	Predicted Values
0	B	B
1	B	B
2	B	B
3	B	B

0	B	B
1	B	B
2	B	B
3	B	B
4	B	B
...	...	...
109	B	B
110	B	B
111	M	M
112	M	M
113	B	B

114 rows × 2 columns

3. Using iris data set, implement naive bayes classification for different naive Bayes classification algorithms.((i) gaussian (ii) bernoulli etc)

- **Find out the accuracy level w.r.t to each algorithm**
- **Display the no:of mislabeled classification from test data set**
- **List out the class labels of the mismatching records**

i)CODE

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

dataset = pd.read_csv('iris.csv')

X = dataset.iloc[:,:4].values

```
y = dataset['variety'].values
dataset.head(5)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
y_pred
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
from sklearn.metrics import accuracy_score
print ("Accuracy : ", accuracy_score(y_test, y_pred))
cm
df = pd.DataFrame({'Real Values':y_test, 'Predicted Values':y_pred})
df
```

OUTPUT

```
Accuracy :  1.0
```

	Real Values	Predicted Values
0	Versicolor	Versicolor
1	Versicolor	Versicolor
2	Setosa	Setosa
3	Setosa	Setosa
4	Setosa	Setosa
5	Setosa	Setosa
6	Setosa	Setosa
7	Versicolor	Versicolor
8	Setosa	Setosa
9	Versicolor	Versicolor
10	Setosa	Setosa
11	Setosa	Setosa
12	Setosa	Setosa
13	Setosa	Setosa
14	Virginica	Virginica
15	Setosa	Setosa
16	Versicolor	Versicolor
17	Versicolor	Versicolor
18	Virginica	Virginica
19	Virginica	Virginica
20	Setosa	Setosa
21	Setosa	Setosa
22	Versicolor	Versicolor
23	Versicolor	Versicolor
24	Versicolor	Versicolor
25	Virginica	Virginica
26	Versicolor	Versicolor
27	Versicolor	Versicolor
28	Virginica	Virginica
29	Setosa	Setosa

CODE

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
dataset = pd.read_csv('iris.csv')
X = dataset.iloc[:,:4].values
y = dataset['variety'].values
dataset.head(5)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
from sklearn.naive_bayes import BernoulliNB
classifier = BernoulliNB()
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
y_pred
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
from sklearn.metrics import accuracy_score
print ("Accuracy : ", accuracy_score(y_test, y_pred))
cm
# df = pd.DataFrame({'Real Values':y_test, 'Predicted Values':y_pred})
# df
```

OUTPUT

```
Accuracy :  0.23333333333333334

array([[ 7,  0,  0],
       [13,  0,  0],
       [10,  0,  0]], dtype=int64)
```

---

ii)CODE

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

```
dataset = pd.read_csv('iris.csv')
X = dataset.iloc[:,:4].values
y = dataset['variety'].values
dataset.head(5)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
from sklearn.naive_bayes import BernoulliNB
classifier = BernoulliNB()
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
y_pred
#from sklearn.metrics import confusion_matrix
#cm = confusion_matrix(y_test, y_pred)
from sklearn.metrics import accuracy_score
print ("Accuracy : ", accuracy_score(y_test, y_pred))
#cm
df = pd.DataFrame({'Real Values':y_test, 'Predicted Values':y_pred})
df
```

OUTPUT

```
Accuracy :  0.26666666666666666
```

	Real Values	Predicted Values
0	Virginica	Versicolor
1	Virginica	Versicolor
2	Virginica	Versicolor
3	Versicolor	Versicolor
4	Setosa	Versicolor
5	Setosa	Versicolor
6	Setosa	Versicolor
7	Setosa	Versicolor
8	Setosa	Versicolor
9	Setosa	Versicolor
10	Versicolor	Versicolor
11	Setosa	Versicolor
12	Setosa	Versicolor
13	Setosa	Versicolor
14	Versicolor	Versicolor
15	Setosa	Versicolor
16	Versicolor	Versicolor
17	Virginica	Versicolor
18	Virginica	Versicolor
19	Versicolor	Versicolor
20	Virginica	Versicolor
21	Versicolor	Versicolor
22	Virginica	Versicolor
23	Virginica	Versicolor
24	Versicolor	Versicolor
25	Setosa	Versicolor
26	Versicolor	Versicolor
27	Virginica	Versicolor
28	Setosa	Versicolor
29	Setosa	Versicolor

References:
https://towardsdatascience.com/machine-learning-basics-naive-bayes-classification-964af6f2a965

https://scikit-learn.org/stable/modules/classes.html#module-sklearn.naive_bayes

4. Use car details CSV file and implement decision tree algorithm

- Find out the accuracy level.
- Display the no:of mislabeled classification from test data set
- List out the class labels of the mismatching records

CODE

```
import os

import numpy as np

import pandas as pd

import numpy as np, pandas as pd

import matplotlib.pyplot as plt

from sklearn import tree, metrics, model_selection


data = pd.read_csv('car.csv',names=['buying','maint','doors','persons','lug_boot','safety','class'])

data.head()
```

	buying	maint	doors	persons	lug_boot	safety	class
0	vhigh	vhigh	2	2	small	low	unacc
1	vhigh	vhigh	2	2	small	med	unacc
2	vhigh	vhigh	2	2	small	high	unacc
3	vhigh	vhigh	2	2	med	low	unacc
4	vhigh	vhigh	2	2	med	med	unacc

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1728 entries, 0 to 1727
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   buying    1728 non-null   object
 1   maint     1728 non-null   object
 2   doors     1728 non-null   object
 3   persons   1728 non-null   object
 4   lug_boot  1728 non-null   object
 5   safety    1728 non-null   object
 6   class     1728 non-null   object
dtypes: object(7)
memory usage: 94.6+ KB
```

```
data['class'],class_names = pd.factorize(data['class'])

print(class_names)
print(data['class'].unique())
```

```
Index(['unacc', 'acc', 'vgood', 'good'], dtype='object')
[0 1 2 3]
```

```
data['buying'],_ = pd.factorize(data['buying'])
data['maint'],_ = pd.factorize(data['maint'])
data['doors'],_ = pd.factorize(data['doors'])
data['persons'],_ = pd.factorize(data['persons'])
data['lug_boot'],_ = pd.factorize(data['lug_boot'])
data['safety'],_ = pd.factorize(data['safety'])
data.head()
```

	buying	maint	doors	persons	lug_boot	safety	class
0	0	0	0	0	0	0	0
1	0	0	0	0	0	1	0
2	0	0	0	0	0	2	0
3	0	0	0	0	1	0	0
4	0	0	0	0	1	1	0

data.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1728 entries, 0 to 1727
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   buying    1728 non-null   int64
 1   maint     1728 non-null   int64
 2   doors     1728 non-null   int64
 3   persons   1728 non-null   int64
 4   lug_boot  1728 non-null   int64
 5   safety    1728 non-null   int64
 6   class     1728 non-null   int64
dtypes: int64(7)
memory usage: 94.6 KB
```

X = data.iloc[:,:-1]

y = data.iloc[:,-1]

split data randomly into 70% training and 30% test

X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y, test_size=0.3, random_state=0)

train the decision tree

dtree = tree.DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=0)

dtree.fit(X_train, y_train)

```
DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=0)
```

use the model to make predictions with the test data

y_pred = dtree.predict(X_test)

how did our model perform?

accuracy = metrics.accuracy_score(y_test, y_pred)

print('Accuracy: {:.2f}'.format(accuracy))

```
Accuracy: 0.82
```

count_misclassified = (y_test != y_pred).sum()

print('Misclassified samples: {}'.format(count_misclassified))

```
Misclassified samples: 96
```

References:

https://www.24tutorials.com/machine-learning/case-study-decision-tree-model-for-car-quality/
https://notebook.community/bMzi/ML_in_Finance/0210_DecisionTrees
https://stackabuse.com/decision-trees-in-python-with-scikit-learn/

For Data Sets Refer:

https://www.kaggle.com (for data set)
http://archive.ics.uci.edu/ml/datasets.php

5.Implement Simple and multiple linear regression for the data sets ‘student_score.csv’ and ‘company_data .csv’ respectively

Ref: https://stackabuse.com/linear-regression-in-python-with-scikit-learn/

1) import numpy as np
   import pandas as pd
   import matplotlib.pyplot as plt
   #data set contains details of no.of hours spend by students for studt and their marks
   student = pd.read_csv('student_scores.csv')
   student.head()

	Hours	Scores
0	2.5	21
1	5.1	47
2	3.2	27
3	8.5	75
4	3.5	30

2) student.describe()

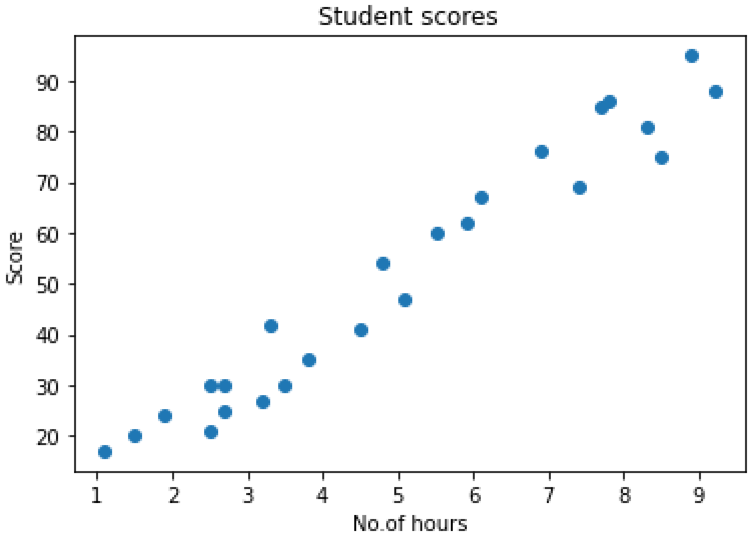
	Hours	Scores
count	25.000000	25.000000
mean	5.012000	51.480000
std	2.525094	25.286887
min	1.100000	17.000000
25%	2.700000	30.000000
50%	4.800000	47.000000
75%	7.400000	75.000000
max	9.200000	95.000000

3) student.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25 entries, 0 to 24
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Hours   25 non-null     float64
 1   Scores  25 non-null     int64
dtypes: float64(1), int64(1)
memory usage: 528.0 bytes
```

4) import matplotlib.pyplot as plt
   Xax=student.iloc[:,0]
   Yax=student.iloc[:,1]
   plt.scatter(Xax,Yax)
   plt.xlabel("No.of hours")
   plt.ylabel("Score")
   plt.title("Student scores")

plt.show()



5) #perform the simple linear regression model
#Equation : Y=w0+w1.x
#Here Y(marks)=w0+w1.x
#create x as hours and Y as marks

X = student.iloc[:, :-1]
y = student.iloc[:, 1]
print(X)

```
    Hours
0     2.5
1     5.1
2     3.2
3     8.5
4     3.5
5     1.5
6     9.2
7     5.5
8     8.3
9     2.7
10    7.7
11    5.9
12    4.5
13    3.3
14    1.1
15    8.9
16    2.5
17    1.9
18    6.1
19    7.4
20    2.7
21    4.8
22    3.8
23    6.9
24    7.8
```

6) print(y)

```
0     21
1     47
2     27
3     75
4     30
5     20
6     88
7     60
8     81
9     25
10    85
11    62
12    41
13    42
14    17
15    95
16    30
17    24
18    67
19    69
20    30
21    54
22    35
23    76
24    86
Name: Scores, dtype: int64
```

7) from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
print(X_train)

```
    Hours
2     3.2
22    3.8
1     5.1
6     9.2
19    7.4
7     5.5
21    4.8
16    2.5
9     2.7
17    1.9
8     8.3
13    3.3
20    2.7
3     8.5
18    6.1
15    8.9
11    5.9
0     2.5
5     1.5
24    7.8
```

8) from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)

```
LinearRegression()
```

9) print(regressor.intercept_)

```
3.4157789720979608
```

10) print(regressor.coef_)

```
print(regressor.coef_)
```

```
[9.544138]
```

11)
```
y_pred = regressor.predict(X_test)
for(i,j) in zip(y_test,y_pred):
    if i!=j:
        print("Actual value :",i,"Predicted value :",j)
print("Number of mislabeled points from test data set :", (y_test != y_pred).sum())
```

```
Actual value : 76 Predicted value : 69.2703311556657
Actual value : 41 Predicted value : 46.36439996138126
Actual value : 85 Predicted value : 76.90564155376049
Actual value : 17 Predicted value : 13.914330769478324
Actual value : 30 Predicted value : 36.82026196376275
Number of mislabeled points from test data set : 5
```

12) from sklearn import metrics

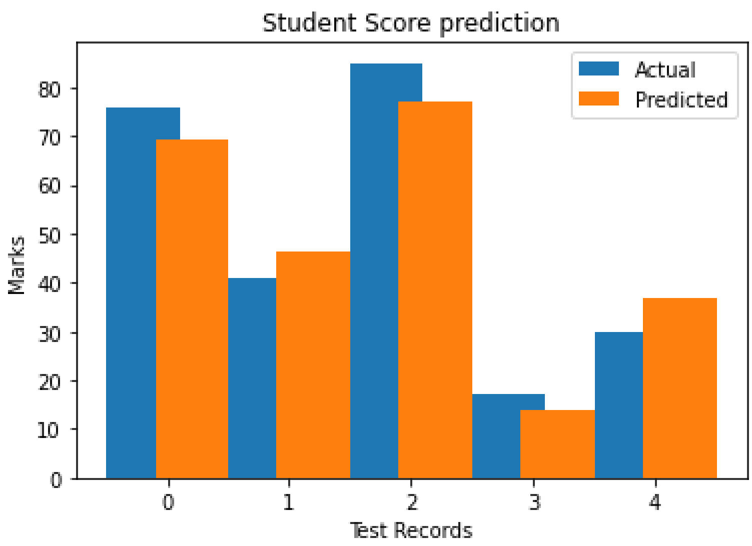
```
print("Mean Absolute error :", metrics.mean_absolute_error(y_test,y_pred))
print("Mean Squared error :", metrics.mean_squared_error(y_test,y_pred))
print("Root Mean Squared error :",
np.sqrt(metrics.mean_squared_error(y_test,y_pred)))
```

```
Mean Absolute error : 6.0188716892478995
Mean Squared error : 39.124239242163426
Root Mean Squared error : 6.254937189306014
```

13) import matplotlib.pyplot as plt

```
c=X_test['Hours'].count()
xax=np.arange(c)
print(xax)
X_axis = np.arange(len(xax))
plt.bar(X_axis-0.2, y_test, 0.6, label='Actual')
plt.bar(X_axis+0.2, y_pred, 0.6, label='Predicted')

plt.xlabel("Test Records")
plt.ylabel("Marks")
plt.title("Student Score prediction")
plt.legend()
plt.show()
```

```
[0 1 2 3 4]
```



Multiple Regression

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

	TV	Radio	Newspaper	Sales
0	230.1	37.8	69.2	22.1
1	44.5	39.3	45.1	10.4
2	17.2	45.9	69.3	12.0
3	151.5	41.3	58.5	16.5
4	180.8	10.8	58.4	17.9

```
advertising.describe()
```

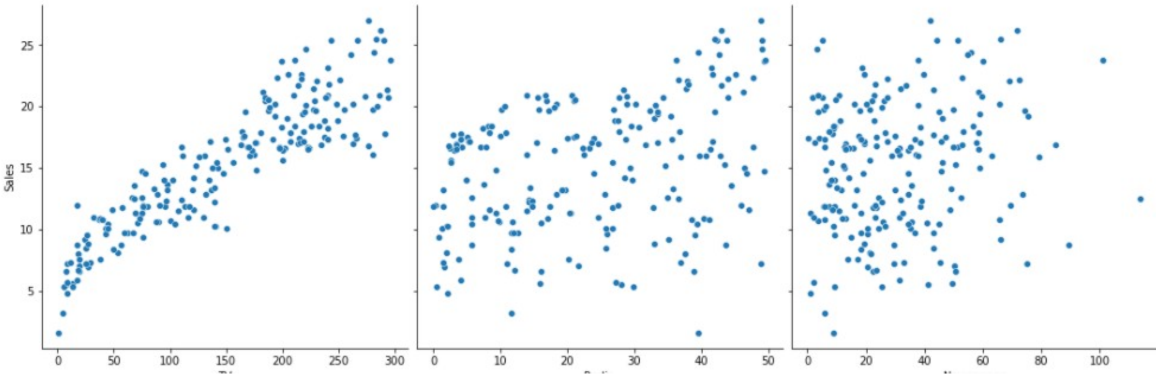
	TV	Radio	Newspaper	Sales
count	200.000000	200.000000	200.000000	200.000000
mean	147.042500	23.264000	30.554000	15.130500
std	85.854236	14.846809	21.778621	5.283892
min	0.700000	0.000000	0.300000	1.600000
25%	74.375000	9.975000	12.750000	11.000000
50%	149.750000	22.900000	25.750000	16.000000
75%	218.825000	36.525000	45.100000	19.050000
max	296.400000	49.600000	114.000000	27.000000

```
advertising.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   TV         200 non-null    float64
 1   Radio      200 non-null    float64
 2   Newspaper  200 non-null    float64
 3   Sales      200 non-null    float64
dtypes: float64(4)
memory usage: 6.4 KB
```

```
import matplotlib.pyplot as plt
import seaborn as sns
sns.pairplot(advertising, x_vars=['TV', 'Radio', 'Newspaper'],
        y_vars='Sales', height=5, aspect=1, kind='scatter')
```

plt.show()



#perform the multiple linear regression model
#Equation : Y=w0+w1.x1 + w2.x2 + w3.x3
#Here Y(sales)=w0+w1.x1(TV)+w2.x2(Radio)+w3.x3(Newspaper)
#create x and Y as sales

X = advertising.iloc[:, :-1]
print(X)

```
        TV  Radio  Newspaper
0    230.1   37.8       69.2
1     44.5   39.3       45.1
2     17.2   45.9       69.3
3    151.5   41.3       58.5
4    180.8   10.8       58.4
..     ...    ...        ...
195   38.2    3.7       13.8
196   94.2    4.9        8.1
197  177.0    9.3        6.4
198  283.6   42.0       66.2
199  232.1    8.6        8.7

[200 rows x 3 columns]
```

y = advertising.iloc[:, -1]
print(y)

```
0      22.1
1      10.4
2      12.0
3      16.5
4      17.9
       ...
195     7.6
196    14.0
197    14.8
198    25.5
199    18.4
Name: Sales, Length: 200, dtype: float64
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
print(X_train)
```

```
        TV  Radio  Newspaper
74   213.4   24.6       13.1
108   13.1    0.4       25.6
24    62.3   12.6       18.3
185  205.0   45.1       19.6
70   199.1   30.6       38.7
..     ...    ...        ...
132    8.4   27.2        2.1
72    26.8   33.0       19.3
91    28.6    1.5       33.0
147  243.2   49.0       44.3
78     5.4   29.9        9.4

[140 rows x 3 columns]
```

```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

```
LinearRegression()
```

```
print(regressor.intercept_)
```

```
4.834885358384083
```

```
print(regressor.coef_)
```

```
[ 0.05441868  0.11045214 -0.00292727]
```

```
y_pred = regressor.predict(X_test)
for(i,j) in zip(y_test,y_pred):
    if i!=j:
        print("Actual value :",i,"Predicted value :",j)
print("Number of mislabeled points from test data set :", (y_test != y_pred).sum())
```

```
Actual value : 18.0 Predicted value : 19.6231329850721
Actual value : 13.3 Predicted value : 13.308438358662285
Actual value : 11.6 Predicted value : 12.629676247192208
Actual value : 17.0 Predicted value : 19.47105688754111
Actual value : 12.3 Predicted value : 11.590783196229005
Actual value : 10.7 Predicted value : 11.348323111604875
Actual value : 11.5 Predicted value : 12.193874211599226
Actual value : 1.6 Predicted value : 9.221415842312133
Actual value : 14.2 Predicted value : 14.498654917672432
Actual value : 23.2 Predicted value : 22.4195357567519
Actual value : 12.6 Predicted value : 12.633285935460439
Actual value : 13.2 Predicted value : 13.506916188363006
Actual value : 18.9 Predicted value : 21.30047124135335
Actual value : 11.3 Predicted value : 9.614344565789548
Actual value : 17.3 Predicted value : 16.895894130341553
Actual value : 17.7 Predicted value : 19.461157758162322
Actual value : 20.2 Predicted value : 22.265150396440184
Actual value : 10.3 Predicted value : 12.653339706591243
Actual value : 16.6 Predicted value : 18.225248894921528
Actual value : 18.0 Predicted value : 17.77531629354258
Actual value : 18.4 Predicted value : 18.389881966359212
Actual value : 6.7 Predicted value : 7.120487442072934
Actual value : 22.1 Predicted value : 21.329147279261107
Actual value : 7.0 Predicted value : 8.107233580890917
Actual value : 17.2 Predicted value : 17.241601167641473
Actual value : 15.6 Predicted value : 15.932854944685918
Actual value : 9.2 Predicted value : 9.814012958114937
Actual value : 7.6 Predicted value : 8.07181587152485
Actual value : 16.6 Predicted value : 17.24408925929915
Actual value : 16.1 Predicted value : 21.53718441018264
```

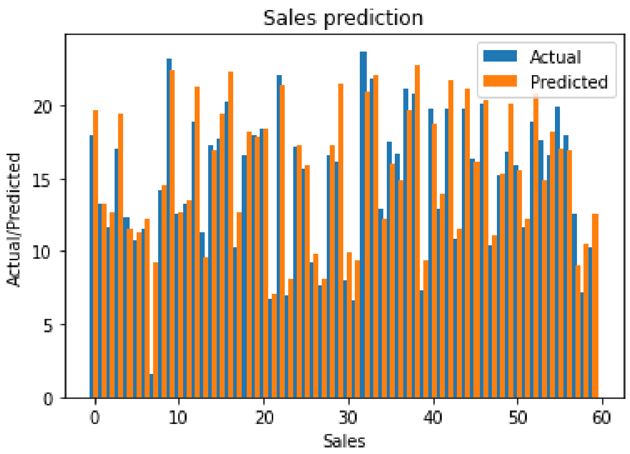
```
Actual value : 21.8 Predicted value : 22.11714873643335
Actual value : 12.9 Predicted value : 12.24172273842917
Actual value : 17.5 Predicted value : 15.956455247767064
Actual value : 16.7 Predicted value : 14.855431394128658
Actual value : 21.2 Predicted value : 19.70279428720466
Actual value : 20.8 Predicted value : 22.746586722943107
Actual value : 7.3 Predicted value : 9.414955243763849
Actual value : 19.8 Predicted value : 18.797882366422925
Actual value : 12.9 Predicted value : 13.932861260132384
Actual value : 19.8 Predicted value : 21.68847856864288
Actual value : 10.8 Predicted value : 11.507027883270196
Actual value : 19.8 Predicted value : 21.135922336380666
Actual value : 16.4 Predicted value : 16.1314240806 7001
Actual value : 20.1 Predicted value : 20.39027162911635
Actual value : 10.4 Predicted value : 11.055958359288084
Actual value : 15.2 Predicted value : 15.319170605692749
Actual value : 16.8 Predicted value : 20.07719756905186
Actual value : 15.9 Predicted value : 15.519157553678733
Actual value : 11.6 Predicted value : 12.2047855314722
Actual value : 18.9 Predicted value : 20.759113647419312
Actual value : 17.6 Predicted value : 14.899620097103586
Actual value : 16.6 Predicted value : 18.119983975440608
Actual value : 19.9 Predicted value : 17.01161015049727
Actual value : 17.9 Predicted value : 16.944628168884364
Actual value : 12.6 Predicted value : 9.001742559147871
Actual value : 7.2 Predicted value : 10.489892255018589
Actual value : 10.3 Predicted value : 12.580375265443143
Number of mislabeled points from test data set : 60
```

```
from sklearn import metrics
print("Mean Absolute error :", metrics.mean_absolute_error(y_test,y_pred))
print("Mean Squared error :", metrics.mean_squared_error(y_test,y_pred))
print("Root Mean Squared error :", np.sqrt(metrics.mean_squared_error(y_test,y_pred)))
```

```
Mean Absolute error : 1.4439815071936666
Mean Squared error : 3.9021227595995556
Root Mean Squared error : 1.9753791432531518
```

```
import matplotlib.pyplot as plt
c=X_test['TV'].count()
xax=np.arange(c)
print(xax)
X_axis = np.arange(len(xax))
plt.bar(X_axis-0.2, y_test, 0.6, label='Actual')
plt.bar(X_axis+0.2, y_pred, 0.6, label='Predicted')

plt.xlabel("Sales")
plt.ylabel("Actual/Predicted")
plt.title("Sales prediction")
plt.legend()
plt.show()
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
 48 49 50 51 52 53 54 55 56 57 58 59]
```



<u>Neural Networks</u>

1.Create a neural network for the given ‘houseprice.csv’ to predict the whether price of the house is above or below median value or not.

import tensorflow as tf

import keras

import pandas

import sklearn

import matplotlib

import pandas as pd

df=pd.read_csv('housepricedata.csv')

df

	LotArea	OverallQual	OverallCond	TotalBsmtSF	FullBath	HalfBath	BedroomAbvGr	TotRmsAbvGrd	Fireplaces	GarageArea	AboveMedianPrice
0	8450	7	5	856	2	1	3	8	0	548	1
1	9600	6	8	1262	2	0	3	6	1	460	1
2	11250	7	5	920	2	1	3	6	1	608	1
3	9550	7	5	756	1	0	3	7	1	642	0
4	14260	8	5	1145	2	1	4	9	1	836	1
...	...	...	...	...	...	...	...	...	...	...	...
1455	7917	6	5	953	2	1	3	7	1	460	1
1456	13175	6	6	1542	2	0	3	7	2	500	1
1457	9042	7	9	1152	2	0	4	9	2	252	1
1458	9717	5	6	1078	1	0	2	5	0	240	0
1459	9937	5	6	1256	1	1	3	6	0	276	0

1460 rows × 11 columns

dataset = df.values

dataset

```
array([[ 8450,     7,     5, ...,     0,   548,     1],
       [ 9600,     6,     8, ...,     1,   460,     1],
       [11250,     7,     5, ...,     1,   608,     1],
       ...,
       [ 9042,     7,     9, ...,     2,   252,     1],
       [ 9717,     5,     6, ...,     0,   240,     0],
       [ 9937,     5,     6, ...,     0,   276,     0]], dtype=int64)
```

X = dataset[:,0:10]

Y = dataset[:,10]

from sklearn import preprocessing

min_max_scaler = preprocessing.MinMaxScaler()

X_scale = min_max_scaler.fit_transform(X)

X_scale

```
array([[0.0334198 , 0.66666667, 0.5       , ..., 0.5       , 0.        ,
        0.3864598 ],
       [0.03879502, 0.55555556, 0.875     , ..., 0.33333333, 0.33333333,
        0.32440056],
       [0.04650728, 0.66666667, 0.5       , ..., 0.33333333, 0.33333333,
        0.42877292],
       ...,
       [0.03618687, 0.66666667, 1.        , ..., 0.58333333, 0.66666667,
        0.17771509],
       [0.03934189, 0.44444444, 0.625     , ..., 0.25      , 0.        ,
        0.16925247],
       [0.04037019, 0.44444444, 0.625     , ..., 0.33333333, 0.        ,
        0.19464034]])
```

---

from sklearn.model_selection import train_test_split

X_train, X_val_and_test, Y_train, Y_val_and_test = train_test_split(X_scale, Y, test_size=0.3)

X_val, X_test, Y_val, Y_test = train_test_split(X_val_and_test, Y_val_and_test, test_size=0.5)

```
print(X_train.shape, X_val.shape, X_test.shape, Y_train.shape, Y_val.shape,
Y_test.shape)
```

```
(1022, 10) (219, 10) (219, 10) (1022,) (219,) (219,)
```

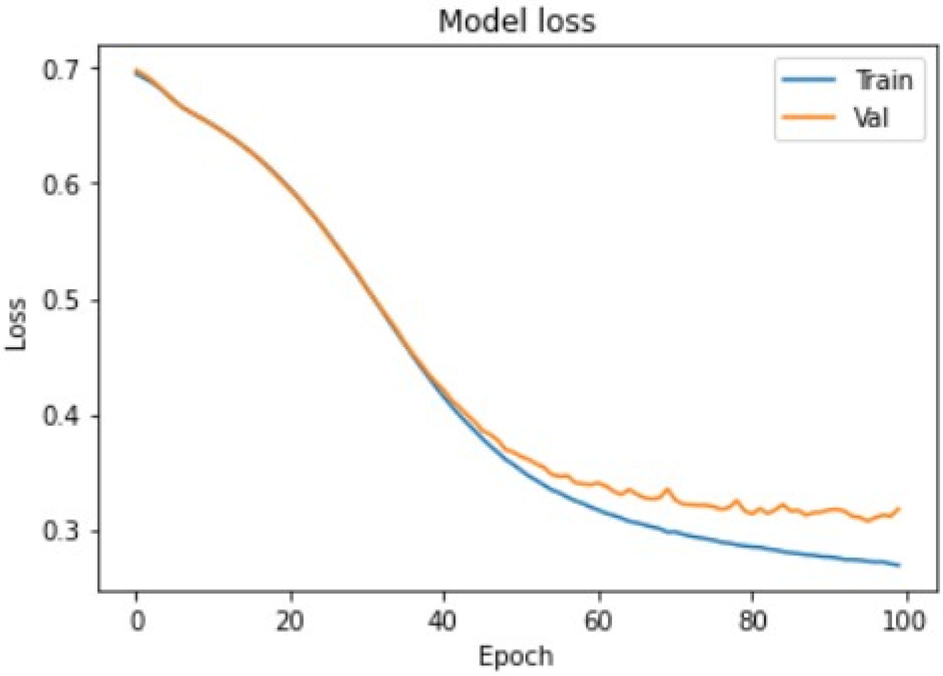
```
from keras.models import Sequential
from keras.layers import Dense

model = Sequential([
    Dense(32, activation='relu', input_shape=(10,)),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid'),
])
model.compile(optimizer='sgd',
              loss='binary_crossentropy',
              metrics=['accuracy'])
hist = model.fit(X_train, Y_train,
          batch_size=32, epochs=100,
          validation_data=(X_val, Y_val))
```

```
767
Epoch 95/100
32/32 [==============================] - 0s 4ms/step - loss: 0.2773 - accuracy: 0.8914 - val_loss: 0.3128 - val_accuracy: 0.8
767
Epoch 96/100
32/32 [==============================] - 0s 4ms/step - loss: 0.2773 - accuracy: 0.8894 - val_loss: 0.3155 - val_accuracy: 0.8
767
Epoch 97/100
32/32 [==============================] - 0s 4ms/step - loss: 0.2760 - accuracy: 0.8894 - val_loss: 0.3114 - val_accuracy: 0.8
813
Epoch 98/100
32/32 [==============================] - 0s 4ms/step - loss: 0.2755 - accuracy: 0.8914 - val_loss: 0.3099 - val_accuracy: 0.8
813
Epoch 99/100
32/32 [==============================] - 0s 4ms/step - loss: 0.2749 - accuracy: 0.8904 - val_loss: 0.3078 - val_accuracy: 0.8
904
Epoch 100/100
32/32 [==============================] - 0s 4ms/step - loss: 0.2743 - accuracy: 0.8914 - val_loss: 0.3140 - val_accuracy: 0.8
767
```

```
model.evaluate(X_test, Y_test)[1]
```

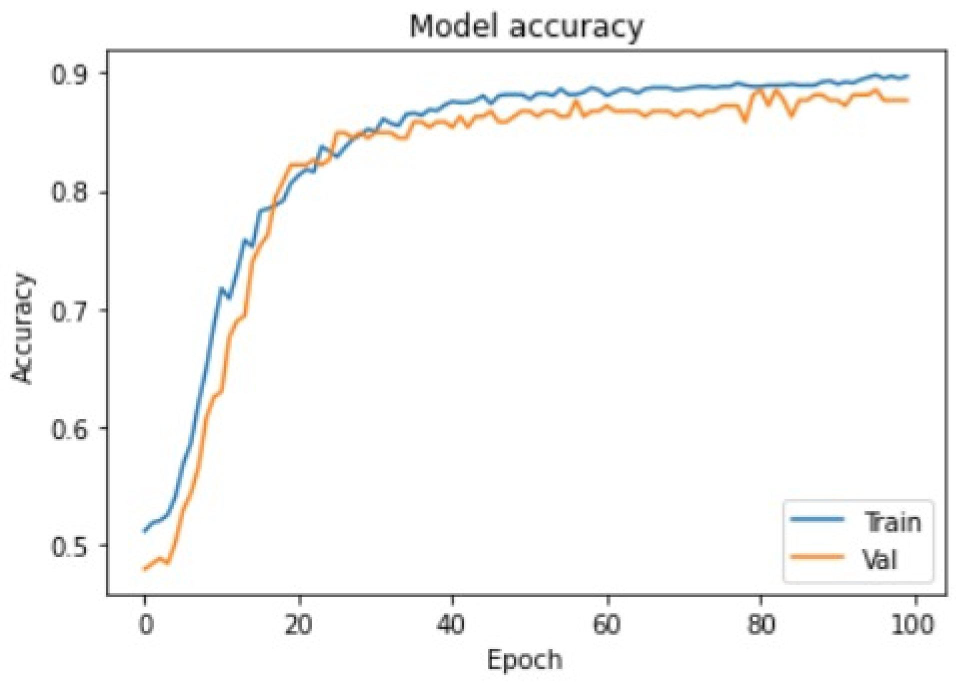
```
7/7 [==============================] - 0s 3ms/step - loss: 0.2653 - accuracy: 0.8584
0.8584474921226501
```

```
import matplotlib.pyplot as plt

plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper right')
plt.show()
```




```
plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
```

```
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='lower right')
plt.show()
```




```
model_2 = Sequential([
    Dense(1000, activation='relu', input_shape=(10,)),
    Dense(1000, activation='relu'),
    Dense(1000, activation='relu'),
    Dense(1000, activation='relu'),
    Dense(1, activation='sigmoid'),
])
model_2.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
hist_2 = model_2.fit(X_train, Y_train,
          batch_size=32, epochs=100,
          validation_data=(X_val, Y_val))
```

```
8721
Epoch 95/100
32/32 [==============================] - 1s 38ms/step - loss: 0.2040 - accuracy: 0.9129 - val_loss: 0.4571 - val_accuracy: 0.
8904
Epoch 96/100
32/32 [==============================] - 1s 42ms/step - loss: 0.1727 - accuracy: 0.9315 - val_loss: 0.4910 - val_accuracy: 0.
8950
Epoch 97/100
32/32 [==============================] - 1s 45ms/step - loss: 0.1240 - accuracy: 0.9472 - val_loss: 0.6341 - val_accuracy: 0.
8813
Epoch 98/100
32/32 [==============================] - 1s 37ms/step - loss: 0.1060 - accuracy: 0.9569 - val_loss: 0.6325 - val_accuracy: 0.
8767
Epoch 99/100
32/32 [==============================] - 1s 39ms/step - loss: 0.1429 - accuracy: 0.9442 - val_loss: 0.7589 - val_accuracy: 0.
8676
Epoch 100/100
32/32 [==============================] - 1s 39ms/step - loss: 0.1572 - accuracy: 0.9315 - val_loss: 0.7380 - val_accuracy: 0.
8676
```

plt.plot(hist_2.history['loss'])

plt.plot(hist_2.history['val_loss'])

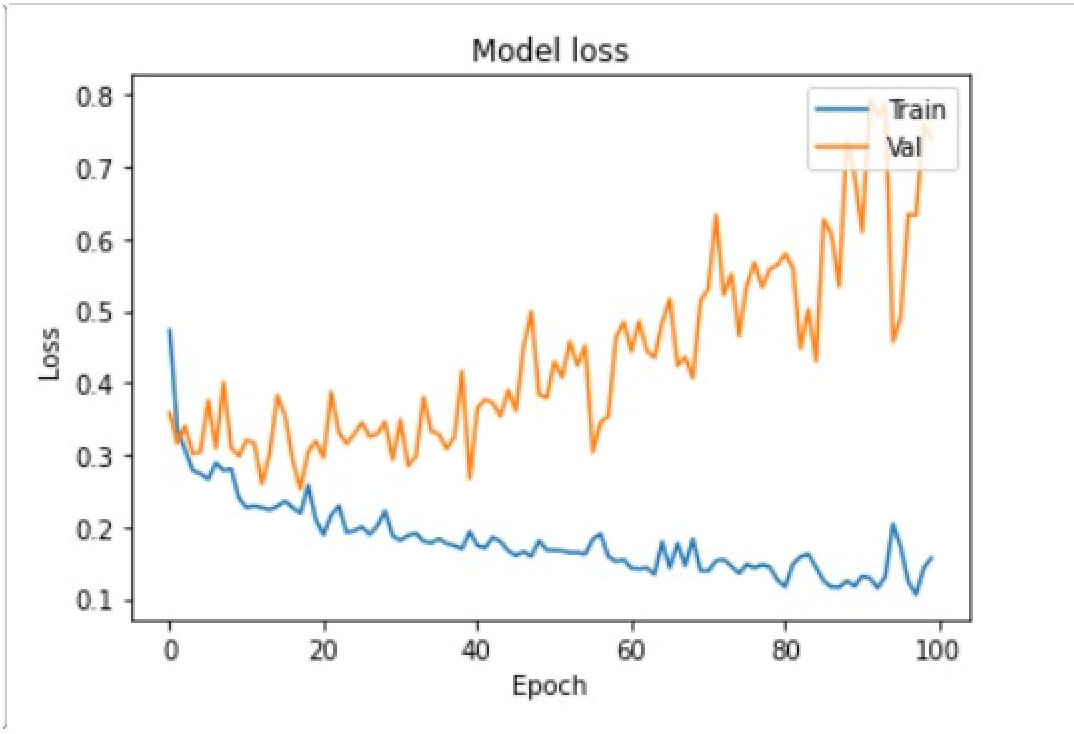
plt.title('Model loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

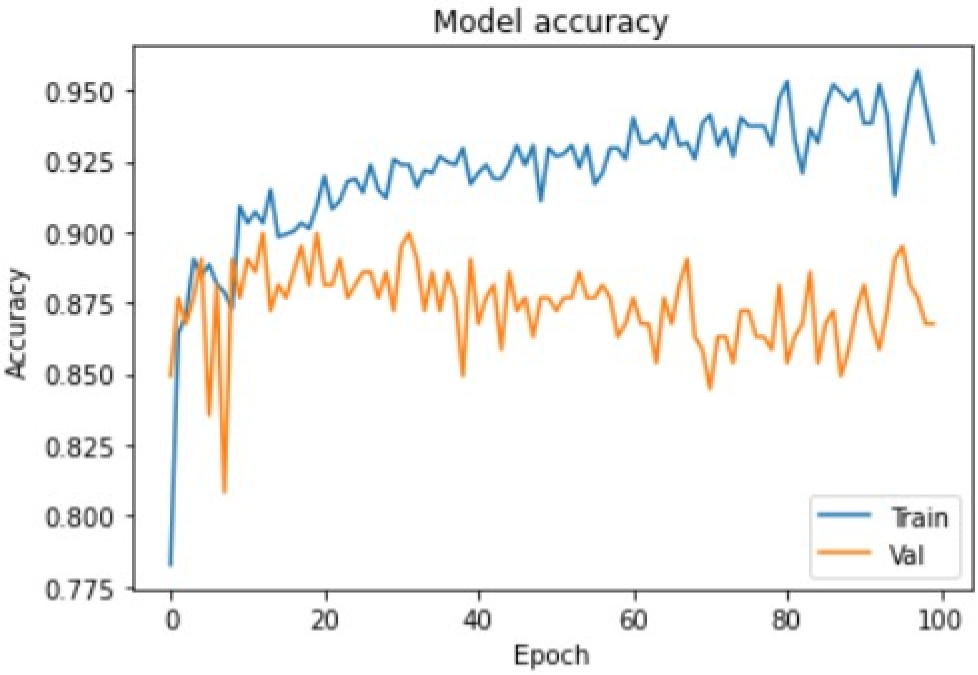
plt.legend(['Train', 'Val'], loc='upper right')

plt.show()



plt.plot(hist_2.history['accuracy'])

```
plt.plot(hist_2.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='lower right')
plt.show()
```

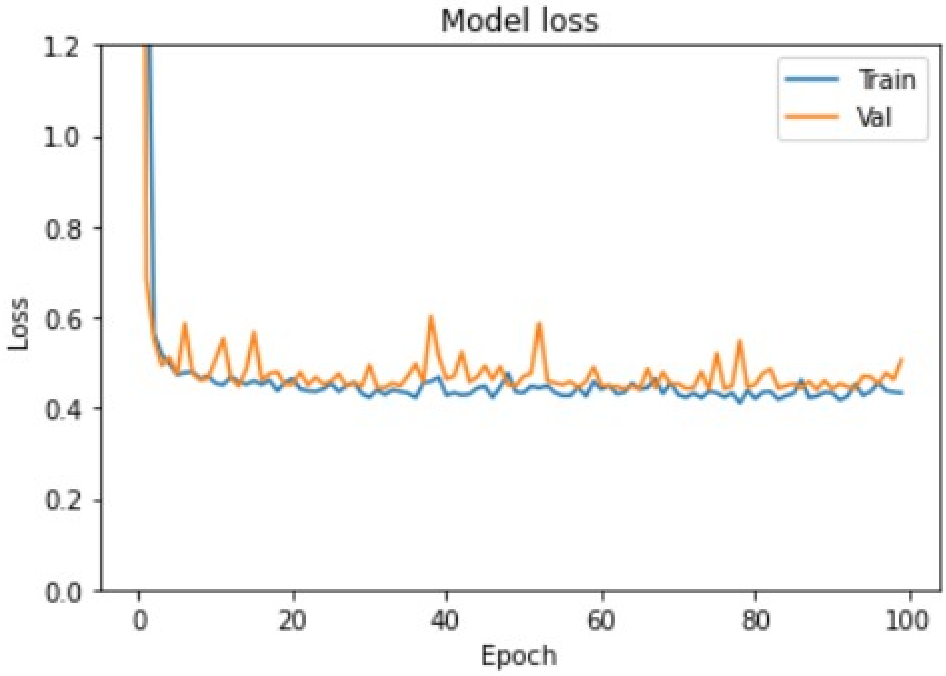


```
from keras.layers import Dropout
from keras import regularizers

model_3 = Sequential([
        Dense(1000, activation='relu', kernel_regularizer=regularizers.l2(0.01), input_shape=(10,)),
    Dropout(0.3),
    Dense(1000, activation='relu', kernel_regularizer=regularizers.l2(0.01)),
    Dropout(0.3),
    Dense(1000, activation='relu', kernel_regularizer=regularizers.l2(0.01)),
    Dropout(0.3),
```

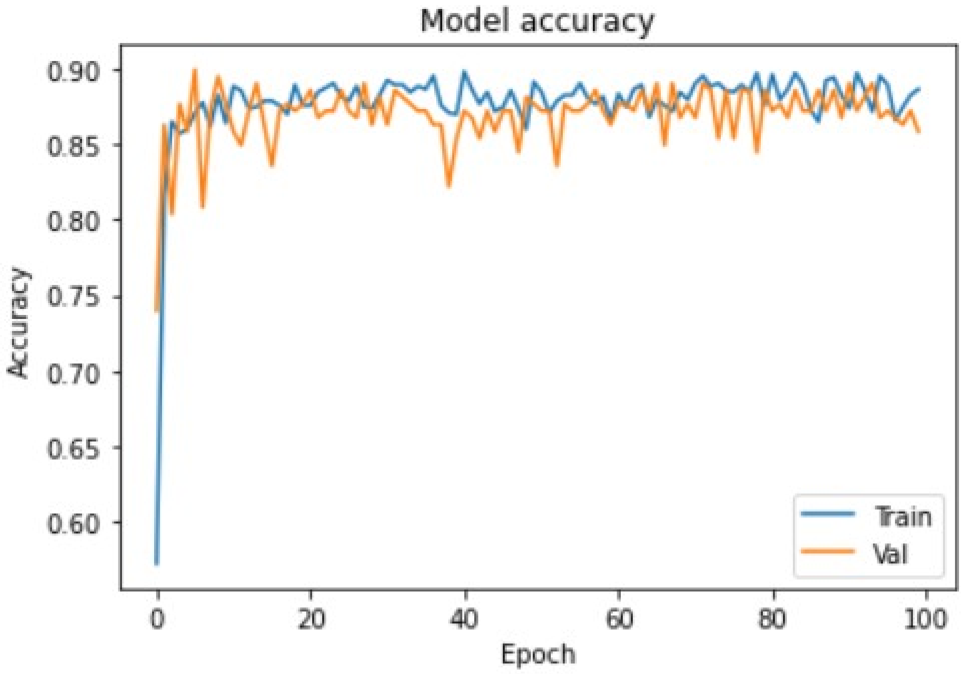
```
    Dense(1000, activation='relu', kernel_regularizer=regularizers.l2(0.01)),
    Dropout(0.3),
    Dense(1, activation='sigmoid', kernel_regularizer=regularizers.l2(0.01)),
])

model_3.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

hist_3 = model_3.fit(X_train, Y_train,
          batch_size=32, epochs=100,
          validation_data=(X_val, Y_val))
```

```
8904
Epoch 95/100
32/32 [==============================] - 2s 64ms/step - loss: 0.4279 - accuracy: 0.8953 - val_loss: 0.4703 - val_accuracy: 0.
8676
Epoch 96/100
32/32 [==============================] - 2s 60ms/step - loss: 0.4363 - accuracy: 0.8894 - val_loss: 0.4686 - val_accuracy: 0.
8721
Epoch 97/100
32/32 [==============================] - 2s 54ms/step - loss: 0.4557 - accuracy: 0.8659 - val_loss: 0.4532 - val_accuracy: 0.
8676
Epoch 98/100
32/32 [==============================] - 2s 56ms/step - loss: 0.4385 - accuracy: 0.8748 - val_loss: 0.4772 - val_accuracy: 0.
8630
Epoch 99/100
32/32 [==============================] - 2s 60ms/step - loss: 0.4351 - accuracy: 0.8826 - val_loss: 0.4628 - val_accuracy: 0.
8721
Epoch 100/100
32/32 [==============================] - 2s 59ms/step - loss: 0.4332 - accuracy: 0.8865 - val_loss: 0.5059 - val_accuracy: 0.
8584
```

```
plt.plot(hist_3.history['loss'])
plt.plot(hist_3.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper right')
plt.ylim(top=1.2, bottom=0)
plt.show()
```




```
plt.plot(hist_3.history['accuracy'])
plt.plot(hist_3.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='lower right')
plt.show()
```


Model accuracy
Accuracy
Epoch
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0
20
40
60
80
100
Train
Val