In [26]:
```
#Experiment 1: Data preprocessing: Handling missing values, handling categorical
import numpy as np
import matplotlib as plt
import pandas as pd
sheet = pd.read_csv(r'C:\Users\tejar\OneDrive\Desktop\ML lab\dataset.csv')
print(sheet)
```

```
   Country   Age   Salary Purchased
0   France  44.0  72000.0        No
1    Spain  27.0  48000.0       Yes
2  Germany  30.0  54000.0        No
3    Spain  38.0  61000.0        No
4  Germany  40.0      NaN       Yes
5   France  35.0  58000.0       Yes
6    Spain   NaN  52000.0        No
7   France  48.0  79000.0       Yes
8  Germany  50.0  83000.0        No
9   France  37.0  67000.0       Yes
```

In [8]:
```
#Experiment 1: Data preprocessing: Handling missing values, handling categorical
sheet.isnull()
```

Out[8]:

	Country	Age	Salary	Purchased
0	False	False	False	False
1	False	False	False	False
2	False	False	False	False
3	False	False	False	False
4	False	False	True	False
5	False	False	False	False
6	False	True	False	False
7	False	False	False	False
8	False	False	False	False
9	False	False	False	False

In [9]:
```
#Experiment 1: Data preprocessing: Handling missing values, handling categorical
sheet.fillna(0)
```

Out[9]:

	Country	Age	Salary	Purchased
0	France	44.0	72000.0	No
1	Spain	27.0	48000.0	Yes
2	Germany	30.0	54000.0	No
3	Spain	38.0	61000.0	No
4	Germany	40.0	0.0	Yes
5	France	35.0	58000.0	Yes
6	Spain	0.0	52000.0	No
7	France	48.0	79000.0	Yes
8	Germany	50.0	83000.0	No
9	France	37.0	67000.0	Yes

In [10]:
```
#Experiment 1: Data preprocessing: Handling missing values, handling categorical
sheet.fillna(method ='pad')
```

Out[10]:

	Country	Age	Salary	Purchased
0	France	44.0	72000.0	No
1	Spain	27.0	48000.0	Yes
2	Germany	30.0	54000.0	No
3	Spain	38.0	61000.0	No
4	Germany	40.0	61000.0	Yes
5	France	35.0	58000.0	Yes
6	Spain	35.0	52000.0	No
7	France	48.0	79000.0	Yes
8	Germany	50.0	83000.0	No
9	France	37.0	67000.0	Yes

In [11]:
```
#Experiment 1: Data preprocessing: Handling missing values, handling categorical
sheet.fillna(method ='bfill')
```

Out[11]:

	Country	Age	Salary	Purchased
0	France	44.0	72000.0	No
1	Spain	27.0	48000.0	Yes
2	Germany	30.0	54000.0	No
3	Spain	38.0	61000.0	No
4	Germany	40.0	58000.0	Yes
5	France	35.0	58000.0	Yes
6	Spain	48.0	52000.0	No
7	France	48.0	79000.0	Yes
8	Germany	50.0	83000.0	No
9	France	37.0	67000.0	Yes

In [19]:
```
#Experiment 1: Data preprocessing: Handling missing values, handling categorical
sheet["Age"].fillna(21, inplace = True)
print(sheet)
```

```
   Country   Age   Salary Purchased
0   France  44.0  72000.0        No
1    Spain  27.0  48000.0       Yes
2  Germany  30.0  54000.0        No
3    Spain  38.0  61000.0        No
4  Germany  40.0      NaN       Yes
5   France  35.0  58000.0       Yes
6    Spain  21.0  52000.0        No
7   France  48.0  79000.0       Yes
8  Germany  50.0  83000.0        No
9   France  37.0  67000.0       Yes
```

In [20]:
```
#Experiment 1: Data preprocessing: Handling missing values, handling categorical
sheet.replace(to_replace = np.nan, value = 21)
```

Out[20]:

	Country	Age	Salary	Purchased
0	France	44.0	72000.0	No
1	Spain	27.0	48000.0	Yes
2	Germany	30.0	54000.0	No
3	Spain	38.0	61000.0	No
4	Germany	40.0	21.0	Yes
5	France	35.0	58000.0	Yes
6	Spain	21.0	52000.0	No
7	France	48.0	79000.0	Yes
8	Germany	50.0	83000.0	No
9	France	37.0	67000.0	Yes

In [22]:
```
#Experiment 1: Data preprocessing: Handling missing values, handling categorical
y=sheet.dropna()
print(y)
```

```
   Country   Age   Salary Purchased
0   France  44.0  72000.0        No
1    Spain  27.0  48000.0       Yes
2  Germany  30.0  54000.0        No
3    Spain  38.0  61000.0        No
5   France  35.0  58000.0       Yes
7   France  48.0  79000.0       Yes
8  Germany  50.0  83000.0        No
9   France  37.0  67000.0       Yes
```

In [23]:
```
#Experiment 1: Data preprocessing: Handling missing values, handling categorical
sheet["Age"].fillna(np.mean(sheet["Age"]), inplace = True)
sheet["Salary"].fillna(np.mean(sheet["Salary"]), inplace = True)
print(sheet)
```

```
   Country        Age        Salary Purchased
0   France  44.000000  72000.000000        No
1    Spain  27.000000  48000.000000       Yes
2  Germany  30.000000  54000.000000        No
3    Spain  38.000000  61000.000000        No
4  Germany  40.000000  63777.777778       Yes
5   France  35.000000  58000.000000       Yes
6    Spain  38.777778  52000.000000        No
7   France  48.000000  79000.000000       Yes
8  Germany  50.000000  83000.000000        No
9   France  37.000000  67000.000000       Yes
```

In [25]:
```
#Experiment 1: Data preprocessing: Handling missing values, handling categorical
sheet["Age"].fillna(sheet["Age"].median(), inplace = True)
sheet["Salary"].fillna(sheet["Salary"].median(), inplace = True)
print(sheet)
```

```
   Country   Age   Salary Purchased
0   France  44.0  72000.0        No
1    Spain  27.0  48000.0       Yes
2  Germany  30.0  54000.0        No
3    Spain  38.0  61000.0        No
4  Germany  40.0  61000.0       Yes
5   France  35.0  58000.0       Yes
6    Spain  38.0  52000.0        No
7   France  48.0  79000.0       Yes
8  Germany  50.0  83000.0        No
9   France  37.0  67000.0       Yes
```

In [27]:
```
#Experiment 1: Data preprocessing: Handling missing values, handling categorical
sheet["Age"].fillna(sheet["Age"].mode(), inplace = True)
sheet["Salary"].fillna(sheet["Salary"].mode(), inplace = True)
print(sheet)
```

```
   Country   Age   Salary Purchased
0   France  44.0  72000.0        No
1    Spain  27.0  48000.0       Yes
2  Germany  30.0  54000.0        No
3    Spain  38.0  61000.0        No
4  Germany  40.0  61000.0       Yes
5   France  35.0  58000.0       Yes
6    Spain  44.0  52000.0        No
7   France  48.0  79000.0       Yes
8  Germany  50.0  83000.0        No
9   France  37.0  67000.0       Yes
```

In [28]:
```
#Experiment 1: Data preprocessing: Handling missing values, handling categorical
from sklearn import preprocessing
# label_encoder object knows how to understand word labels.
label_encoder = preprocessing.LabelEncoder()
# Encode labels in column 'species'.
sheet['Country'].unique()
```

Out[28]:
```
array(['France', 'Spain', 'Germany'], dtype=object)
```

In [29]:
```
#Experiment 1: Data preprocessing: Handling missing values, handling categorical
sheet['Country']= label_encoder.fit_transform(sheet['Country'])
sheet['Country'].unique()
```

Out[29]:
```
array([0, 2, 1])
```

In [30]:
```
#Experiment 1: Data preprocessing: Handling missing values, handling categorical
print(sheet)
```

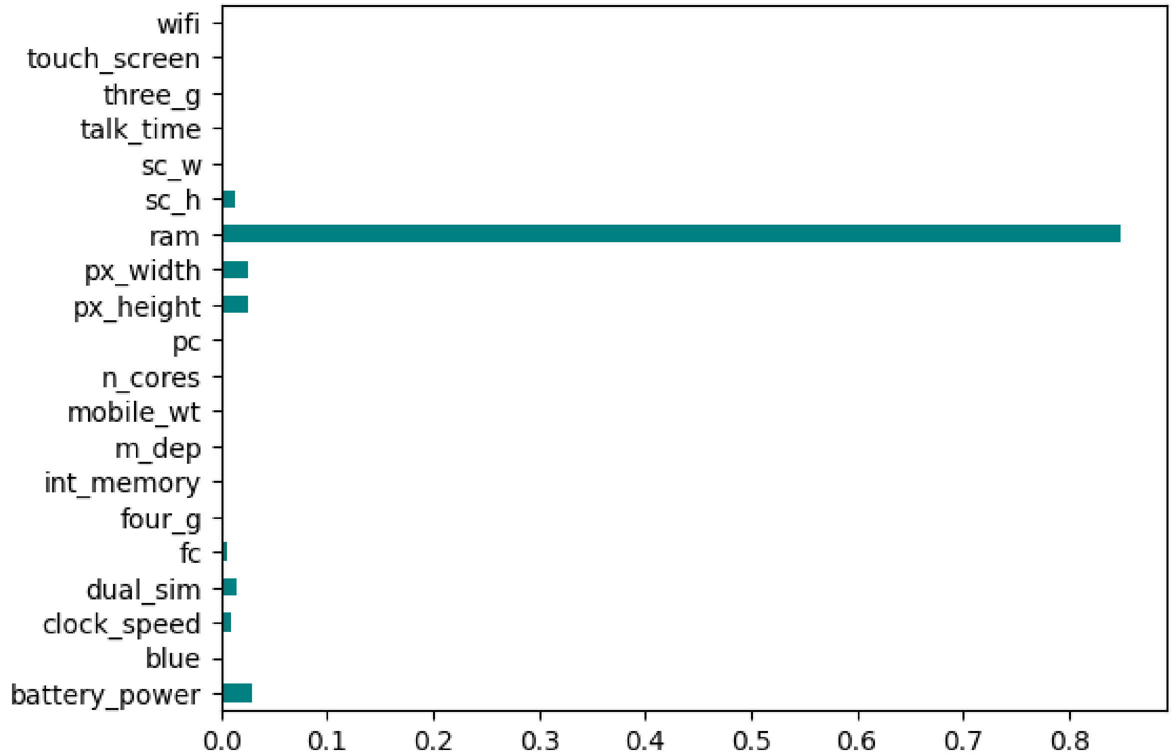
```
    Country   Age   Salary Purchased
0         0  44.0  72000.0        No
1         2  27.0  48000.0       Yes
2         1  30.0  54000.0        No
3         2  38.0  61000.0        No
4         1  40.0  61000.0       Yes
5         0  35.0  58000.0       Yes
6         2  44.0  52000.0        No
7         0  48.0  79000.0       Yes
8         1  50.0  83000.0        No
9         0  37.0  67000.0       Yes
```

In [31]:
```
#Experiment 1: Data preprocessing: Handling missing values, handling categorical
from sklearn import preprocessing
m=preprocessing.MinMaxScaler()
xa=m.fit_transform(y[["Age","Salary"]])
print(xa)
```

```
[[0.73913043 0.68571429]
 [0.         0.        ]
 [0.13043478 0.17142857]
 [0.47826087 0.37142857]
 [0.34782609 0.28571429]
 [0.91304348 0.88571429]
 [1.         1.        ]
 [0.43478261 0.54285714]]
```

In [8]:
```
#Experiment 1: Data preprocessing: Handling missing values, handling categorical
from sklearn.feature_selection import mutual_info_classif
import matplotlib.pyplot as plt
%matplotlib inline
data = pd.read_csv(r"C:\Users\tejar\OneDrive\Desktop\ML lab\train.csv")
z=data.iloc[:,0:20]
y=data.iloc[:,-1]
importances=mutual_info_classif(z,y)
print(importances)
feat_importances=pd.Series(importances,index=data.columns[0:len(data.columns)-1])
print(feat_importances)
feat_importances.plot(kind="barh",color="teal")
plt.show()
```

```
[0.02976132 0.         0.00898659 0.01524492 0.00530511 0.
 0.         0.00121833 0.         0.         0.0010891  0.02583614
 0.02582188 0.84891513 0.01300137 0.         0.         0.
 0.         0.        ]
battery_power    0.029761
blue             0.000000
clock_speed      0.008987
dual_sim         0.015245
fc               0.005305
four_g           0.000000
int_memory       0.000000
m_dep            0.001218
mobile_wt        0.000000
n_cores          0.000000
pc               0.001089
px_height        0.025836
px_width         0.025822
ram              0.848915
sc_h             0.013001
sc_w             0.000000
talk_time        0.000000
three_g          0.000000
touch_screen     0.000000
wifi             0.000000
dtype: float64
```

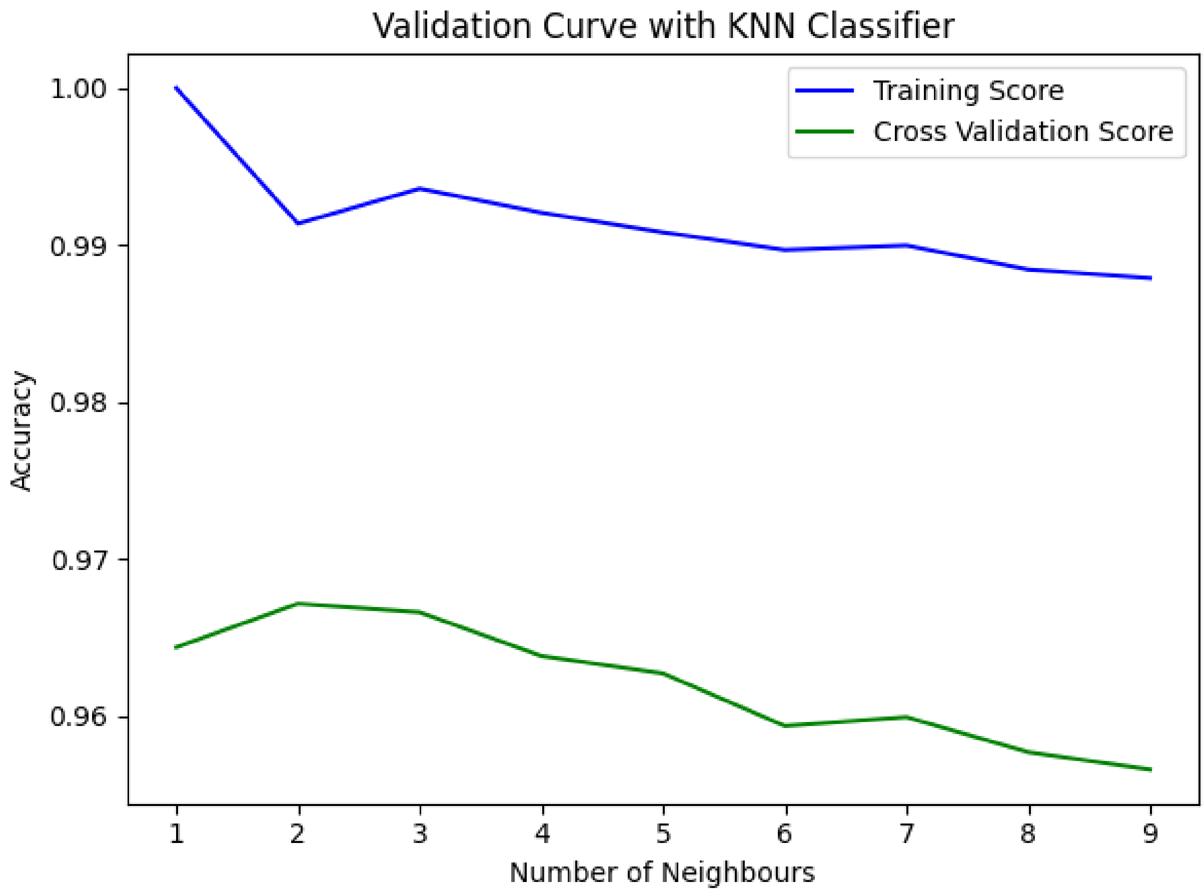


In [9]:
```
#Experiment 2: Model Evaluation and optimization: K-fold cross validation, learni
import numpy as np
from sklearn.model_selection import KFold
data=np.array([0.1,0.2,0.3,0.4,0.5,0.6])
kfold=KFold(3)
for train,test in kfold.split(data):
  print("train:%s,test:%s"%(data[train],data[test]))
```

```
train:[0.3 0.4 0.5 0.6],test:[0.1 0.2]
train:[0.1 0.2 0.5 0.6],test:[0.3 0.4]
train:[0.1 0.2 0.3 0.4],test:[0.5 0.6]
```

In [2]:

```
#Experiment 2: Model Evaluation and optimization: K-fold cross validation, learni
import matplotlib.pyplot as plt
import numpy as np
from sklearn.datasets import load_digits
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import validation_curve

# Loading dataset
dataset = load_digits()

# X contains the data and y contains the labels
X, y = dataset.data, dataset.target

# Setting the range for the parameter (from 1 to 10)
parameter_range = np.arange(1, 10, 1)

# Calculate accuracy on training and test set using the
# gamma parameter with 5-fold cross validation
train_score, test_score = validation_curve(KNeighborsClassifier(), X, y,param_nam

# Calculating mean and standard deviation of training score
mean_train_score = np.mean(train_score, axis = 1)
std_train_score = np.std(train_score, axis = 1)

# Calculating mean and standard deviation of testing score
mean_test_score = np.mean(test_score, axis = 1)
std_test_score = np.std(test_score, axis = 1)

# Plot mean accuracy scores for training and testing scores
plt.plot(parameter_range, mean_train_score,label = "Training Score", color = 'b')
plt.plot(parameter_range, mean_test_score,label = "Cross Validation Score", color

# Creating the plot
plt.title("Validation Curve with KNN Classifier")
plt.xlabel("Number of Neighbours")
plt.ylabel("Accuracy")
plt.tight_layout()
plt.legend(loc = 'best')
plt.show()
```


Validation Curve with KNN Classifier
Training Score
Cross Validation Score
Accuracy
1.00
0.99
0.98
0.97
0.96
1
2
3
4
5
6
7
8
9
Number of Neighbours

In [3]:
```
#Experiment 2: Model Evaluation and optimization: K-fold cross validation, learni
import pandas as pd
import numpy as np
from sklearn.model_selection import GridSearchCV
dataset=pd.read_csv(r"C:\Users\tejar\OneDrive\Desktop\ML lab\wineQualityReds.csv'
dataset.head()
X = dataset.iloc[:, 0:11].values
y = dataset.iloc[:, 11].values
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=10, random_st
from sklearn.preprocessing import StandardScaler
feature_scaler = StandardScaler()
X_train = feature_scaler.fit_transform(X_train)
X_test = feature_scaler.transform(X_test)
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier()

grid_param = {
    'n_estimators': [100, 300, 500, 800, 1000],
    'criterion': ['gini', 'entropy'],
    'bootstrap': [True, False]
}
gd_sr = GridSearchCV(estimator=classifier,
                     param_grid=grid_param,
                     scoring='accuracy',
                     cv=5,
                     n_jobs=-1)
gd_sr.fit(X_train, y_train)
best_parameters = gd_sr.best_params_
print(best_parameters)
best_result = gd_sr.best_score_
print(best_result)
```

```
{'bootstrap': True, 'criterion': 'entropy', 'n_estimators': 300}
0.6985516735114974
```

In [5]:
```
#Experiment 3: Compressing data via dimensionality reduction: PCA, LDA
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
iris=datasets.load_iris()
X=iris.data
Y=iris.target
tn=iris.target_names
lda=LinearDiscriminantAnalysis(n_components=1)
xr2=lda.fit(X,Y).transform(X)
print(xr2)
```

```
 [-6.22824009]
 [-5.22048773]
 [-6.80015   ]
 [-3.81515972]
 [-5.10748966]
 [-6.79671631]
 [-6.52449599]
 [-4.99550279]
 [-3.939853  ]
 [-5.2038309 ]
 [-6.65308685]
 [-5.10555946]
 [-5.50747997]
 [-6.79601924]
 [-6.84735943]
 [-5.64500346]
 [-5.1795646 ]
 [-4.9677409 ]
 [-5.88614539]
 [-4.68315426]]
```

In [6]:
```
#Experiment 3: Compressing data via dimensionality reduction: PCA, LDA
tn=iris.target_names
pca=PCA(n_components=2)
xr=pca.fit(X,Y).transform(X)
print(xr)
```

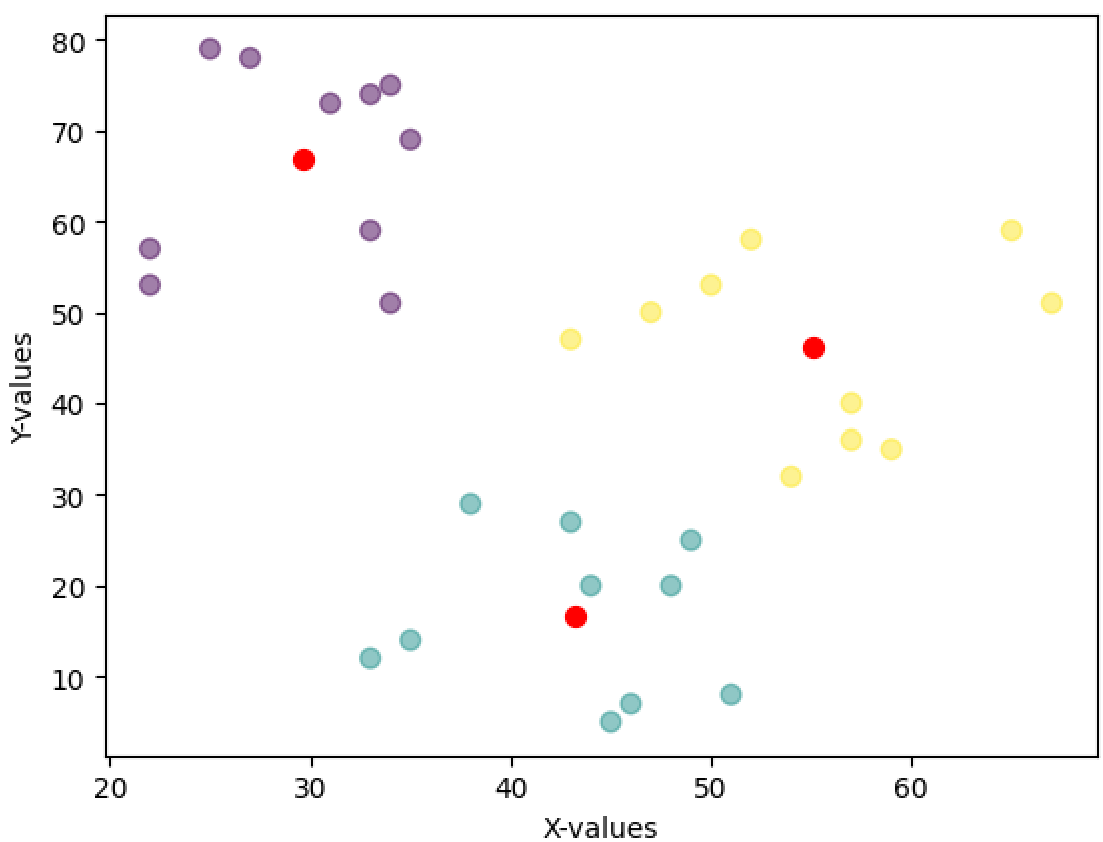
```
 [-2.61275523  0.01472994]
 [-2.78610927 -0.235112  ]
 [-3.22380374 -0.51139459]
 [-2.64475039  1.17876464]
 [-2.38603903  1.33806233]
 [-2.62352788  0.81067951]
 [-2.64829671  0.31184914]
 [-2.19982032  0.87283904]
 [-2.5879864   0.51356031]
 [-2.31025622  0.39134594]
 [-2.54370523  0.43299606]
 [-3.21593942  0.13346807]
 [-2.30273318  0.09870885]
 [-2.35575405 -0.03728186]
 [-2.50666891 -0.14601688]
 [-2.46882007  0.13095149]
 [-2.56231991  0.36771886]
 [-2.63953472  0.31203998]
 [-2.63198939 -0.19696122]
 [-2.58739848 -0.20431849]
```

In [23]:
```python
#Experiment 4: Ensemble Learning, Data Clustering & Classification
#Classification
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
X, y = make_classification(n_samples=1000, n_features=4,
                           n_informative=2, n_redundant=0,
                          random_state=0, shuffle=False)
print(len(X),len(y))
print(X,y)
clf = RandomForestClassifier(max_depth=2, random_state=0)
clf.fit(X, y)
print(clf.predict([[-1.66853167, -1.29901346,  0.2746472,  -0.60362044]]))
```

```
1000 1000
[[-1.66853167 -1.29901346  0.2746472  -0.60362044]
 [-2.9728827  -1.08878294  0.70885958  0.42281857]
 [-0.59614125 -1.37007001 -3.11685659  0.64445203]
 ...
 [ 0.91711204  1.10596645  0.86766522 -2.25625012]
 [ 0.10027664  1.45875846 -0.44360274 -0.67002328]
 [ 1.0415229  -0.01987143  0.15216419 -1.9405334 ]] [0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 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 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 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 1 1 1 1 1 1 1 1 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
 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 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 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 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 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 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 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 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 1 0 1 1 1 1
 1]
[0]
```

In [24]:
```
#Experiment 4: Ensemble Learning, Data Clustering & Classification
#Data Clustering


from pandas import DataFrame
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

Data = {'x': [25,34,22,27,33,33,31,22,35,34,67,54,57,43,50,57,59,52,65,47,49,48,3
        'y': [79,51,53,78,59,74,73,57,69,75,51,32,40,47,53,36,35,58,59,50,25,20,1
       }

df = DataFrame(Data,columns=['x','y'])

kmeans = KMeans(n_clusters=3).fit(df)
centroids = kmeans.cluster_centers_
print(centroids)
plt.xlabel("X-values")
plt.ylabel("Y-values")
plt.scatter(df['x'], df['y'], c= kmeans.labels_.astype(float), s=50, alpha=0.5)
plt.scatter(centroids[:, 0], centroids[:, 1], c='red', s=50)
plt.show()
```

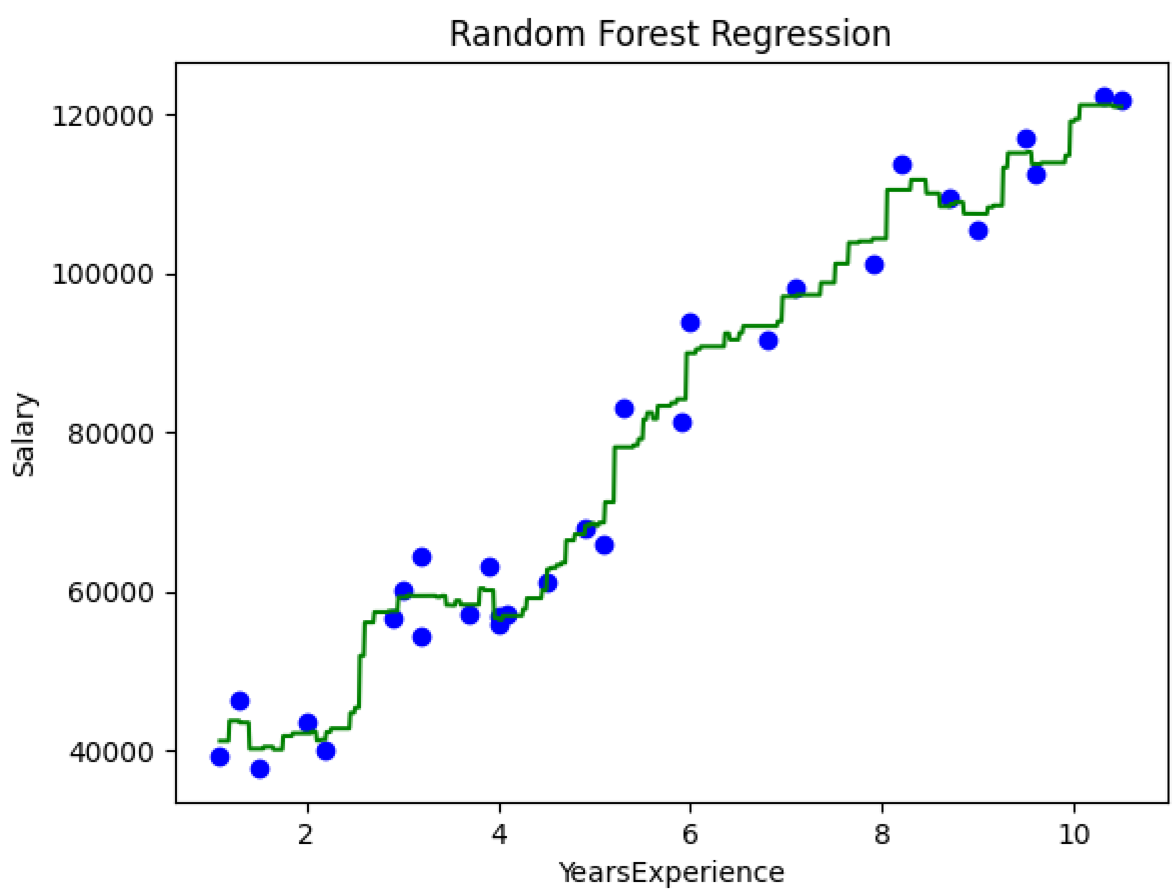
```
[[29.6 66.8]
 [43.2 16.7]
 [55.1 46.1]]
```



In [26]:
```
#Experiment 4: Ensemble Learning, Data Clustering & Classification
#Ensemble Learning

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.ensemble import RandomForestRegressor

data = pd.read_csv(r'C:\Users\tejar\OneDrive\Desktop\ML lab\Salary_Data.csv')
print(data)
# Fitting Random Forest Regression to the dataset
# create regressor object
regressor = RandomForestRegressor(n_estimators = 100, random_state = 0)

x= data.iloc [:, : -1] # ” : ” means it will select all rows,
y= data.iloc [:, -1 :] #“: -1 ” means that it will ignore last column

# fit the regressor with x and y data
"""
 y["Salary"] was a Series, now it's numpy array (it is a column-vector). If I app
then it becomes a row vector and no error message appears, through the predictior
 x is a DataFrame with feature names, while x.values is only values, without feat
"""

regressor.fit(x.values, y["Salary"].ravel())
#Y_pred = regressor.predict(np.array([6.5]).reshape(1, 1)) # test the output by c

# Visualising the Random Forest Regression results

# arrange for creating a range of values
# from min value of x to max
# value of x with a difference of 0.01
# between two consecutive values

X_grid = np.arange(min(x["YearsExperience"]), max(x["YearsExperience"]),0.01)

# reshape for reshaping the data into a len(X_grid)*1 array,
# i.e. to make a column out of the X_grid value
X_grid = X_grid.reshape((len(X_grid), 1))

# Scatter plot for original data
plt.scatter(x, y, color = 'blue')

# plot predicted data
plt.plot(X_grid, regressor.predict(X_grid),color = 'green')
plt.title('Random Forest Regression')
plt.xlabel('YearsExperience')
plt.ylabel('Salary')
plt.show()
```

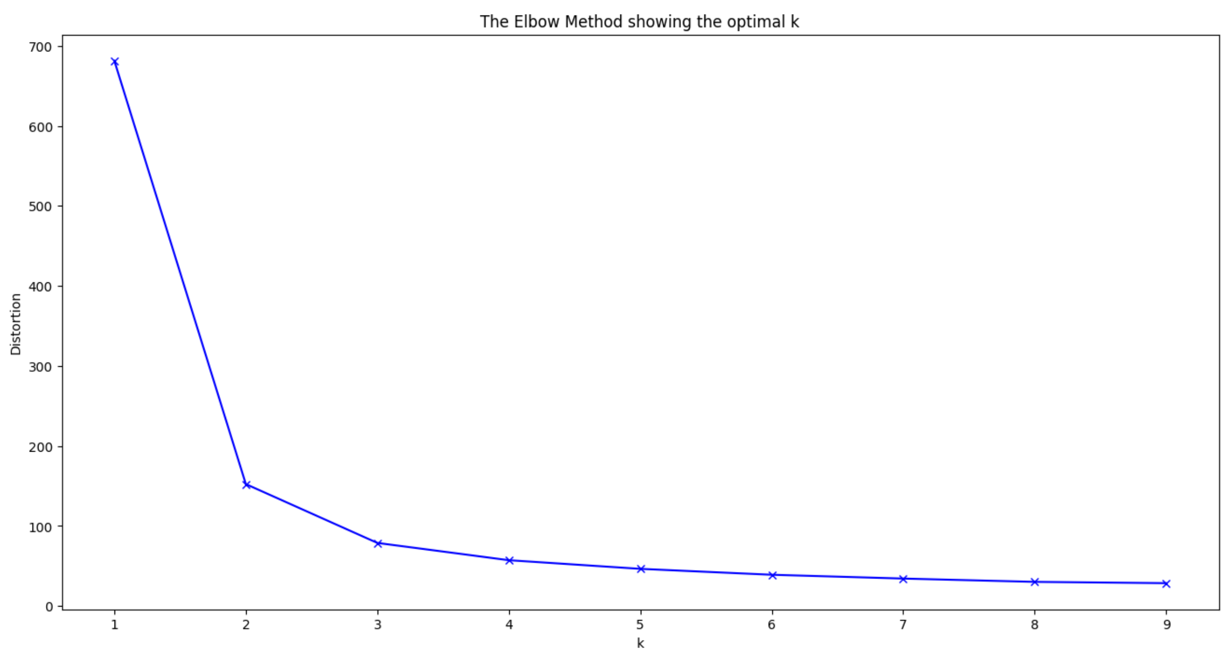
```
    YearsExperience    Salary
0               1.1   39343.0
1               1.3   46205.0
2               1.5   37731.0
3               2.0   43525.0
```

```
4               2.2   39891.0
5               2.9   56642.0
6               3.0   60150.0
7               3.2   54445.0
8               3.2   64445.0
9               3.7   57189.0
10              3.9   63218.0
11              4.0   55794.0
12              4.0   56957.0
13              4.1   57081.0
14              4.5   61111.0
15              4.9   67938.0
16              5.1   66029.0
17              5.3   83088.0
18              5.9   81363.0
19              6.0   93940.0
20              6.8   91738.0
21              7.1   98273.0
22              7.9  101302.0
23              8.2  113812.0
24              8.7  109431.0
25              9.0  105582.0
26              9.5  116969.0
27              9.6  112635.0
28             10.3  122391.0
29             10.5  121872.0
```



In [1]:
```
#Experiment-5 Write a program to evaluate clustering model
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.cluster import KMeans
from sklearn import datasets
iris = datasets.load_iris()
df=pd.DataFrame(iris['data'])

print(df.head())
iris['target']
distortions = []
K = range(1,10)
for k in K:
    kmeanModel = KMeans(n_clusters=k)
    kmeanModel.fit(df)
    distortions.append(kmeanModel.inertia_)
plt.figure(figsize=(16,8))
plt.plot(K, distortions, 'bx-')
plt.xlabel('k')
plt.ylabel('Distortion')
plt.title('The Elbow Method showing the optimal k')
plt.show()
```

```
     0    1    2    3
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
```

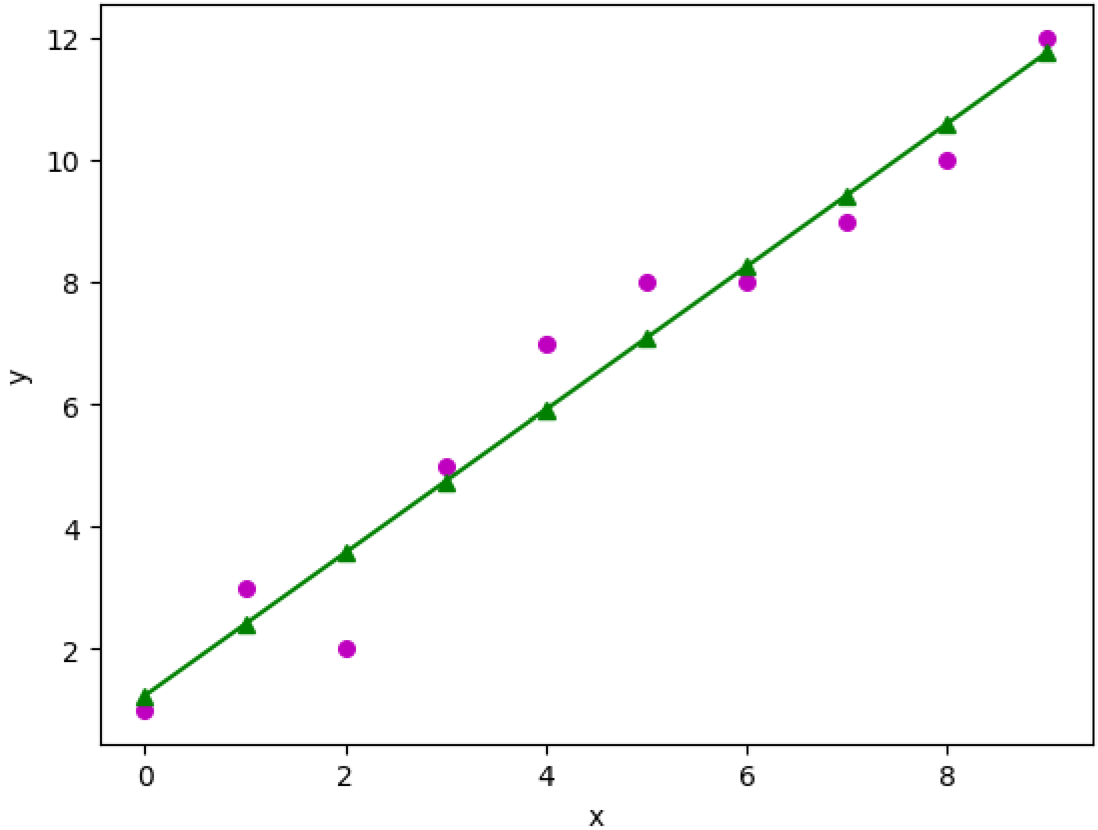


In [10]:
```
#Experiment 6: Vector addition
from numpy import array
a = array([1, 2, 3])
print(a)
b = array([1, 2, 3])
print(b)
c = a + b
print(c)
a = array([1, 2, 3])
print(a)
b = array([0.5, 0.5, 0.5])
print(b)
c = a - b
print(c)
a = array([1, 2, 3])
print(a)
b = array([1, 2, 3])
print(b)
c = a * b
print(c)
a = array([1, 2, 3])
print(a)
b = array([1, 2, 3])
print(b)
c = a / b
print(c)
a = array([1, 2, 3])
print(a)
b = array([1, 2, 3])
print(b)
c = a.dot(b)
print(c)
d = a @ b
print(d)
s=2
e=s*a
print(e)
```

```
[1 2 3]
[1 2 3]
[2 4 6]
[1 2 3]
[0.5 0.5 0.5]
[0.5 1.5 2.5]
[1 2 3]
[1 2 3]
[1 4 9]
[1 2 3]
[1 2 3]
[1. 1. 1.]
[1 2 3]
[1 2 3]
14
14
[2 4 6]
```

In [16]:
```
#Experiment 7: Regression model.
import numpy as np
import matplotlib.pyplot as plt
def estimate_coef(x, y):
  # number of observations/points
  n = np.size(x)
  # mean of x and y vector
  mx = np.mean(x)
  my = np.mean(y)
  # calculating cross-deviation and deviation about x
  SSxy = np.sum(y*x) - n*my*mx
  SSxx = np.sum(x*x) - n*mx*mx
  # calculating regression coefficients
  b1 = SSxy / SSxx
  b0 = my - b1*mx
  return (b0, b1)
def plot_regression_line(x, y, b):
  # plotting the actual points as scatter plot
  plt.scatter(x, y, color = "m",
  marker = "o", s = 30)
  print("x-values : ",x)
  print("y-values : ",y)
  # predicted response vector
  y_pred = b[0] + b[1]*x
  #y_pred represents the predicted response value
  #b[0] and b[1] are regression coefficients and represent
  #y-intercept and slope of regression line respectively.
  # plotting the regression line
  plt.plot(x, y_pred, marker='^', color = "g")
  # putting labels
  plt.xlabel('x')
  plt.ylabel('y')
  # function to show plot
  plt.show()
  print("x-values : ",x)
  print("y-values : ",y_pred)
x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])
# estimating coefficients
#here estimate_coef(x, y) is a user defined function returns regression
#coefficients
b = estimate_coef(x, y)
print("Estimated coefficients b_0 : ",b[0], " and b_1 : ",b[1])
# plotting regression line
plot_regression_line(x, y, b)
```

```
Estimated coefficients b_0 :  1.2363636363636363  and b_1 :  1.1696969696969697
x-values :  [0 1 2 3 4 5 6 7 8 9]
y-values :  [ 1  3  2  5  7  8  8  9 10 12]
```




```
x-values :  [0 1 2 3 4 5 6 7 8 9]
y-values :  [ 1.23636364  2.40606061  3.57575758  4.74545455  5.91515152  7.084
84848
  8.25454545  9.42424242 10.59393939 11.76363636]
```

In [21]:

```
#Experiment 8: Write a program to reduce variance of a linear regression model us
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error

df = pd.read_csv(r"C:\Users\tejar\OneDrive\Desktop\ML lab\realestate.csv")
X = df.drop(columns = ['Y house price of unit area','X1 transaction date', 'X2 ho
Y = df['Y house price of unit area']
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.2,random_st
print("Lasso")
lasso = Lasso()
lasso.fit(x_train, y_train)
y_pred_lasso = lasso.predict(x_test)
mse = mean_squared_error(y_test, y_pred_lasso)
print(lasso.score(x_train, y_train))
print(lasso.score(x_test,y_test))
print("Ridge")
ridge = Ridge()
ridge.fit(x_train, y_train)
y_pred_ridge = ridge.predict(x_test)
mse = mean_squared_error(y_test, y_pred_ridge)
print(ridge.score(x_train, y_train))
print(ridge.score(x_test,y_test))
```

```
Lasso
0.4731920099128133
0.5769026422529159
Ridge
0.4758712465042749
0.5834098092708352
```

In [17]:

```
#Experiment 9: Write a program to implement logistic regression for binary classi
import numpy
#X represents the size of a tumor in centimeters.
X = numpy.array([3.78, 2.44, 2.09, 0.14, 1.72, 1.65, 4.92, 4.37, 4.96, 4.52, 3.69

#Note: X has to be reshaped into a column from a row for the LogisticRegression()
#y represents whether or not the tumor is cancerous (0 for "No", 1 for "Yes").
y = numpy.array([0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1])
from sklearn import linear_model
logr = linear_model.LogisticRegression()
logr.fit(X,y)
#predict if tumor is cancerous where the size is 3.46mm:
predicted = logr.predict(numpy.array([3.46]).reshape(-1,1))
print(predicted)
```

```
[0]
```

In [6]:
```
#Experiment 10: Perceptron for digits.
from sklearn.datasets import load_digits
from sklearn.linear_model import Perceptron
X, y = load_digits(return_X_y=True)
clf = Perceptron(tol=1e-3,eta0=0.1, random_state=0)
clf.fit(X, y)
clf.score(X, y)
```

Out[6]:
```
0.9393433500278241
```

In [13]:
```
#Experiment 11: Feed-Forward Network for wheat seeds dataset.
import pandas as pd
data= pd.read_csv(r'C:\Users\tejar\OneDrive\Desktop\ML lab\seeds.csv')
print(data.shape)
data.head()
data.describe()
x=data.iloc[:,:7]
y=data.iloc[:,7]
print(x.shape)
print(y.shape)
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.5,random_state=0)
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
import keras
from keras.models import Sequential
from keras.layers import Dense
model=Sequential()
model.add(Dense(units=6,kernel_initializer='uniform',activation='relu',input_dim=
model.add(Dense(units=6,kernel_initializer='uniform',activation='relu'))
model.add(Dense(units=1,kernel_initializer='uniform',activation='sigmoid'))
model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
model.fit(x_train,y_train,batch_size=10,epochs=20)
```

```
(199, 8)
(199, 7)
(199,)
(99, 7)
(99,)
(100, 7)
(100,)
Epoch 1/20
10/10 [==============================] - 1s 3ms/step - loss: 0.6855 - accuracy:
0.3030
Epoch 2/20
10/10 [==============================] - 0s 4ms/step - loss: 0.6643 - accuracy:
0.3434
Epoch 3/20
10/10 [==============================] - 0s 9ms/step - loss: 0.6418 - accuracy:
0.3434
Epoch 4/20
10/10 [==============================] - 0s 4ms/step - loss: 0.6111 - accuracy:
0.3434
Epoch 5/20
10/10 [==============================] - 0s 4ms/step - loss: 0.5723 - accuracy:
0.3434
Epoch 6/20
10/10 [==============================] - 0s 4ms/step - loss: 0.5166 - accuracy:
0.3434
Epoch 7/20
10/10 [==============================] - 0s 4ms/step - loss: 0.4413 - accuracy:
0.3434
Epoch 8/20
10/10 [==============================] - 0s 4ms/step - loss: 0.3411 - accuracy:
0.3434
```

```
Epoch 9/20
10/10 [==============================] - 0s 4ms/step - loss: 0.2130 - accuracy:
0.3434
Epoch 10/20
10/10 [==============================] - 0s 4ms/step - loss: 0.0511 - accuracy:
0.3434
Epoch 11/20
10/10 [==============================] - 0s 4ms/step - loss: -0.1506 - accurac
y: 0.3434
Epoch 12/20
10/10 [==============================] - 0s 4ms/step - loss: -0.3895 - accurac
y: 0.3434
Epoch 13/20
10/10 [==============================] - 0s 11ms/step - loss: -0.6706 - accurac
y: 0.3434
Epoch 14/20
10/10 [==============================] - 0s 4ms/step - loss: -0.9901 - accurac
y: 0.3434
Epoch 15/20
10/10 [==============================] - 0s 4ms/step - loss: -1.3511 - accurac
y: 0.3434
Epoch 16/20
10/10 [==============================] - 0s 4ms/step - loss: -1.7336 - accurac
y: 0.3434
Epoch 17/20
10/10 [==============================] - 0s 5ms/step - loss: -2.1810 - accurac
y: 0.3434
Epoch 18/20
10/10 [==============================] - 0s 4ms/step - loss: -2.6435 - accurac
y: 0.3434
Epoch 19/20
10/10 [==============================] - 0s 5ms/step - loss: -3.1583 - accurac
y: 0.3434
Epoch 20/20
10/10 [==============================] - 0s 4ms/step - loss: -3.7509 - accurac
y: 0.3434
```

Out[13]:
```
<keras.callbacks.History at 0x221c80792a0>
```

In [5]:
```python
#Experiment 12: Write a program to implement a neural network for regression.
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Dense
from sklearn import metrics
dataframe_tr = pd.read_csv (r'C:\Users\tejar\OneDrive\Desktop\ML lab\Wine.csv')
dataframe_ts = pd.read_csv (r'C:\Users\tejar\OneDrive\Desktop\ML lab\Wine.csv')
dataset_tr = dataframe_tr.values
dataset_ts = dataframe_ts.values
total_cols=len(dataframe_tr.columns)-1
X_train = dataset_tr[:,0:total_cols]
Y_train = dataset_tr[:,total_cols]
X_test = dataset_ts[:,0:total_cols]
Y_test = dataset_ts[:,total_cols]
# Construction of DNN Model
model = Sequential()
model.add(Dense(10, input_dim=total_cols, activation='relu')) # Hidden 1
model.add(Dense(10, activation='relu')) # Hidden 2
model.add(Dense(10, activation='relu')) # Hidden 3
model.add(Dense(10, activation='relu')) # Hidden 4
model.add(Dense(1)) # Output
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(X_train, Y_train, epochs=50,validation_split=0.2)

Y_train_pred_DNN = model.predict(X_train)
Y_test_pred_DNN = model.predict(X_test)
print(np.sqrt(metrics.mean_squared_error(Y_train, Y_train_pred_DNN)))
print(np.sqrt(metrics.mean_squared_error(Y_test, Y_test_pred_DNN)))
```

```
Epoch 1/50
5/5 [==============================] - 6s 127ms/step - loss: 207.3957 - val_l
oss: 31.2352
Epoch 2/50
5/5 [==============================] - 0s 22ms/step - loss: 9.7916 - val_los
s: 3.5749
Epoch 3/50
5/5 [==============================] - 0s 20ms/step - loss: 35.2298 - val_los
s: 8.3404
Epoch 4/50
5/5 [==============================] - 0s 17ms/step - loss: 30.8133 - val_los
s: 0.8117
Epoch 5/50
5/5 [==============================] - 0s 19ms/step - loss: 4.6093 - val_los
s: 7.4241
Epoch 6/50
5/5 [==============================] - 0s 18ms/step - loss: 5.0932 - val_los
s: 15.9111
Epoch 7/50
5/5 [==============================] - 0s 18ms/step - loss: 7.8177 - val_los
s: 10.4926
Epoch 8/50
5/5 [==============================] - 0s 18ms/step - loss: 2.8576 - val_los
s: 3.4890
Epoch 9/50
```

```
5/5 [==============================] - 0s 19ms/step - loss: 2.0009 - val_los
s: 1.5343
Epoch 10/50
5/5 [==============================] - 0s 22ms/step - loss: 2.6266 - val_los
s: 2.2140
Epoch 11/50
5/5 [==============================] - 0s 20ms/step - loss: 1.5715 - val_los
s: 4.4941
Epoch 12/50
5/5 [==============================] - 0s 24ms/step - loss: 1.3710 - val_los
s: 5.4007
Epoch 13/50
5/5 [==============================] - 0s 18ms/step - loss: 1.4545 - val_los
s: 4.3310
Epoch 14/50
5/5 [==============================] - 0s 18ms/step - loss: 1.1948 - val_los
s: 3.3965
Epoch 15/50
5/5 [==============================] - 0s 18ms/step - loss: 1.1838 - val_los
s: 3.0431
Epoch 16/50
5/5 [==============================] - 0s 20ms/step - loss: 1.1627 - val_los
s: 3.3046
Epoch 17/50
5/5 [==============================] - 0s 18ms/step - loss: 1.1230 - val_los
s: 3.8737
Epoch 18/50
5/5 [==============================] - 0s 18ms/step - loss: 1.1291 - val_los
s: 3.8324
Epoch 19/50
5/5 [==============================] - 0s 18ms/step - loss: 1.0974 - val_los
s: 3.3555
Epoch 20/50
5/5 [==============================] - 0s 17ms/step - loss: 1.0641 - val_los
s: 3.1363
Epoch 21/50
5/5 [==============================] - 0s 18ms/step - loss: 1.0652 - val_los
s: 3.0094
Epoch 22/50
5/5 [==============================] - 0s 18ms/step - loss: 1.0473 - val_los
s: 3.2607
Epoch 23/50
5/5 [==============================] - 0s 18ms/step - loss: 1.0177 - val_los
s: 3.2044
Epoch 24/50
5/5 [==============================] - 0s 18ms/step - loss: 0.9968 - val_los
s: 3.0189
Epoch 25/50
5/5 [==============================] - 0s 18ms/step - loss: 0.9790 - val_los
s: 3.0725
Epoch 26/50
5/5 [==============================] - 0s 18ms/step - loss: 0.9704 - val_los
s: 2.9164
Epoch 27/50
5/5 [==============================] - 0s 19ms/step - loss: 0.9496 - val_los
s: 3.1650
Epoch 28/50
```

```
5/5 [==============================] - 0s 18ms/step - loss: 0.9287 - val_los
s: 2.8773
Epoch 29/50
5/5 [==============================] - 0s 17ms/step - loss: 0.9061 - val_los
s: 2.7994
Epoch 30/50
5/5 [==============================] - 0s 17ms/step - loss: 0.8858 - val_los
s: 2.8795
Epoch 31/50
5/5 [==============================] - 0s 18ms/step - loss: 0.8697 - val_los
s: 2.8688
Epoch 32/50
5/5 [==============================] - 0s 18ms/step - loss: 0.8582 - val_los
s: 2.8388
Epoch 33/50
5/5 [==============================] - 0s 19ms/step - loss: 0.8578 - val_los
s: 2.6255
Epoch 34/50
5/5 [==============================] - 0s 18ms/step - loss: 0.8373 - val_los
s: 2.8457
Epoch 35/50
5/5 [==============================] - 0s 19ms/step - loss: 0.8272 - val_los
s: 2.7568
Epoch 36/50
5/5 [==============================] - 0s 19ms/step - loss: 0.8159 - val_los
s: 2.6768
Epoch 37/50
5/5 [==============================] - 0s 18ms/step - loss: 0.8085 - val_los
s: 2.4079
Epoch 38/50
5/5 [==============================] - 0s 16ms/step - loss: 0.7940 - val_los
s: 2.6053
Epoch 39/50
5/5 [==============================] - 0s 19ms/step - loss: 0.7836 - val_los
s: 2.6953
Epoch 40/50
5/5 [==============================] - 0s 18ms/step - loss: 0.7815 - val_los
s: 2.5413
Epoch 41/50
5/5 [==============================] - 0s 18ms/step - loss: 0.7705 - val_los
s: 2.1828
Epoch 42/50
5/5 [==============================] - 0s 18ms/step - loss: 0.7574 - val_los
s: 2.2038
Epoch 43/50
5/5 [==============================] - 0s 17ms/step - loss: 0.7389 - val_los
s: 2.3739
Epoch 44/50
5/5 [==============================] - 0s 18ms/step - loss: 0.7255 - val_los
s: 2.0141
Epoch 45/50
5/5 [==============================] - 0s 17ms/step - loss: 0.7092 - val_los
s: 1.9563
Epoch 46/50
5/5 [==============================] - 0s 18ms/step - loss: 0.7012 - val_los
s: 1.9492
Epoch 47/50
```

```
5/5 [==============================] - 0s 17ms/step - loss: 0.6879 - val_los
s: 2.1869
Epoch 48/50
5/5 [==============================] - 0s 19ms/step - loss: 0.7013 - val_los
s: 2.1596
Epoch 49/50
5/5 [==============================] - 0s 19ms/step - loss: 0.6889 - val_los
s: 1.7832
Epoch 50/50
5/5 [==============================] - 0s 17ms/step - loss: 0.6805 - val_los
s: 1.9534
6/6 [==============================] - 0s 5ms/step
6/6 [==============================] - 0s 3ms/step
0.9663675332890533
0.9663675332890533
```

In [3]:

```
#Experiment 13: Write a program to save and load a trained machine learning model
import joblib
import numpy as np
from sklearn import datasets
from sklearn.neighbors import KNeighborsClassifier
iris_X, iris_y = datasets.load_iris(return_X_y=True)
np.random.seed(0)
indices = np.random.permutation(len(iris_X))
iris_X_train = iris_X[indices[:-10]]
iris_y_train = iris_y[indices[:-10]]
iris_X_test = iris_X[indices[-10:]]
iris_y_test = iris_y[indices[-10:]]
knn = KNeighborsClassifier()
knn.fit(iris_X_train, iris_y_train)
joblib.dump(knn, 'my_trained_model.pkl', compress=9)
!pip install joblib
knn.predict(iris_X_test)
new_observation = [[ 6.3, 2.5, 5.1, 1.9]]
knn.predict(new_observation)
```

```
Requirement already satisfied: joblib in c:\users\tejar\appdata\local\programs
\python\python310\lib\site-packages (1.2.0)

WARNING: You are using pip version 22.0.4; however, version 22.3 is available.
You should consider upgrading via the 'C:\Users\tejar\AppData\Local\Programs\Py
thon\Python310\python.exe -m pip install --upgrade pip' command.
```

Out[3]:

```
array([2])
```

In [ ]: