**GURU NANAK DEV ENGINEERING COLLEGE, LUDHIANA**

**DEPARTMENT OF IT**

Logo

Description automatically generated

**INTRODUCTION TO MACHINE LEARNING LABORATORY**

**LPCIT - 114**

**Submitted to: - Submitted by:-**

**Pf. Jaskiran Kaur Palakpreet Kaur**

**IT B2**

**U.R.N: - 2004836**

**C.R.N: - 2021144**

**INDEX**

|  |  |  |
| --- | --- | --- |
| **Sr. no.** | **Name of the practical** | **Signature** |
| 1. | Implement of Simple Linear Regression. |  |
| 2. | Implementation of Random Forest Regression |  |
| 3. | Implementation of Logistic Regression |  |
| 4. | Implementation of Decision Tree Classification |  |
| 5. | Implementation of K-Nearest Neighbors (K-NN) |  |
| 6. | Naive Bayes |  |
| 7. | K-Means Clustering |  |

# Practical no. 1

## **Aim:** To implement Simple Linear Regression

**Simple Linear Regression:**

Simple Linear Regression is a type of Regression algorithms that models the relationship between a dependent variable and a single independent variable. The relationship shown by a Simple Linear Regression model is linear or a sloped straight line, hence it is called Simple Linear Regression.

**Code:**

### Importing the libraries

import numpy as np

import matplotlib.pyplot as plt import pandas as pd

**Importing the dataset**

dataset = pd.read\_csv('Salary\_dataset.csv')

X = dataset.iloc[:, :-1].values

y = dataset.iloc[:, -1].values

### Splitting the dataset into the Training set and Test set

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 1/3, random\_state = 0)

### Training the Simple Linear Regression model on the Training set

from sklearn.linear\_model import LinearRegression

regressor = LinearRegression()

regressor.fit(X\_train, y\_train)

### Predicting the Test set results

y\_pred = regressor.predict(X\_test)

### Visualising the Training set results

plt.scatter(X\_train, y\_train, color = 'red')

plt.plot(X\_train, regressor.predict(X\_train), color = 'blue') plt.title('Salary vs Experience (Training set)')

plt.xlabel('Years of Experience') plt.ylabel('Salary')

plt.show()

 output 1.1

### Visualising the Test set results

plt.scatter(X\_test, y\_test, color = 'red')

plt.plot(X\_train, regressor.predict(X\_train), color = 'blue') plt.title('Salary vs Experience (Test set)')

plt.xlabel('Years of Experience') plt.ylabel('Salary')

plt.show()



# Practical no. 2

## **Aim:** Implementation of Random Forest Regression

### Random Forest Regression:

A random forest is simply a collection of decision trees whose results are aggregated into one final result. Their ability to limit overfitting without substantially increasing error due to bias is why they are such powerful models

### Code:

### Importing the libraries

import numpy as np

import matplotlib.pyplot as plt import pandas as pd

**Importing the dataset**

dataset = pd.read\_csv('position\_dataset.csv')

X = dataset.iloc[:, :-1].values

### y = dataset.iloc[:, -1].values

### Training the Random Forest Regression model on the whole dataset

from sklearn.ensemble import RandomForestRegressor

regressor = RandomForestRegressor(n\_estimators = 10, random\_state = 0)

regressor.fit(X, y)

### Predicting a new result

regressor.predict([[6.5]])

### Visualising the Random Forest Regression results (higher resolution)

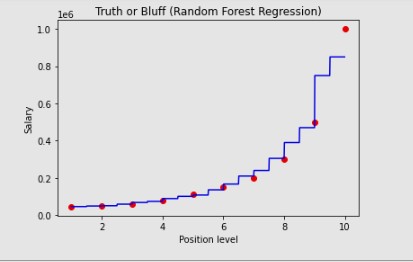
X\_grid = np.arange(min(X), max(X), 0.01)

X\_grid = X\_grid.reshape((len(X\_grid), 1))

plt.scatter(X, y, color = 'red')

plt.plot(X\_grid, regressor.predict(X\_grid), color = 'blue')

plt.title('Truth or Bluff (Random Forest Regression)')

plt.xlabel('Position level')

plt.ylabel('Salary')

# Practical No. 3

# Aim: Implementation of Logistic Regression

**Logistic Regression:**

This type of statistical model (also known as logit model) is often used for classification and predictive analytics. Logistic regression estimates the probability of an event occurring, such as voted or didn't vote, based on a given dataset of independent variables.

**Code:**

### Importing the libraries

import numpy as np

import matplotlib.pyplot as plt import pandas as pd

### Importing the dataset

dataset = pd.read\_csv('Social\_Network\_Ads.csv')

X = dataset.iloc[:, :-1].values y = dataset.iloc[:, -1].values

### Splitting the dataset into the Training set and Test

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25, random\_state = 0) print(X\_train)

print(Y\_train)

### Feature Scaling

from sklearn.preprocessing import StandardScaler sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train) print(X\_train)

print(X\_test)

### Training the Logistic Regression model on the Training set

from sklearn.linear\_model import LogisticRegression classifier = LogisticRegression(random\_state = 0) classifier.fit(X\_train, y\_train)

### Predicting a new result

print(classifier.predict(sc.transform([[30,87000]])))

### Predicting the Test set results

y\_pred = classifier.predict(X\_test) print(np.concatenate((y\_pred.reshape(len(y\_pred),1), y\_test.reshape(len(y\_test),1)),1))

### Making the Confusion Matrix

from sklearn.metrics import confusion\_matrix, accuracy\_score cm = confusion\_matrix(y\_test, y\_pred)

print(cm) accuracy\_score(y\_test, y\_pred)

### Visualising the Training set results

from matplotlib.colors import ListedColormap

X\_set, y\_set = sc.inverse\_transform(X\_train), y\_train

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 10, stop = X\_set[:, 0].max() + 10, step = 0.25),

np.arange(start = X\_set[:, 1].min() - 1000, stop = X\_set[:, 1].max() + 1000, step = 0.25))

plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),

alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)

plt.title('Logistic Regression (Training set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

plt.show()

### 

### Visualising the Test set results

from matplotlib.colors import ListedColormap X\_set, y\_set = sc.inverse\_transform(X\_test), y\_test

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 10, stop = X\_set[:, 0].max() + 10, step = 0.25), np.arange(start = X\_set[:, 1].min() - 1000, stop = X\_set[:, 1].max() + 1000, step = 0.25))

plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape), alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

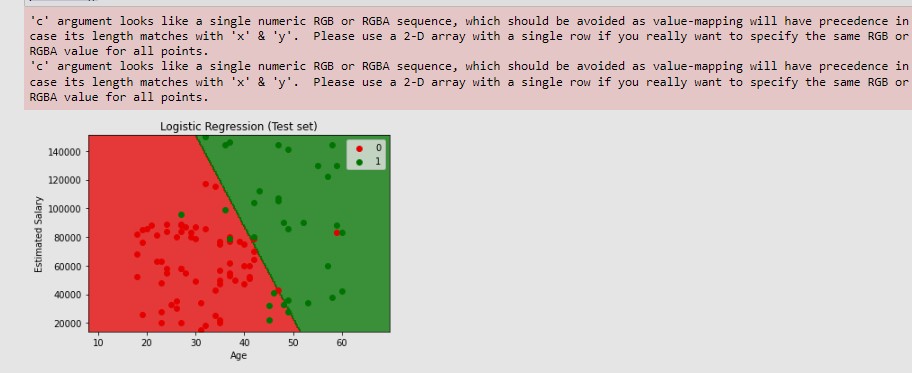
plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j) plt.title('Logistic Regression (Test set)')

plt.xlabel('Age') plt.ylabel('Estimated Salary') plt.legend()

plt.show()



# Practical No.4

## **Aim:** Implementation of Decision Tree Classification

**Decision Tree classification:**

Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.

**Code:**

### Importing the libraries

import numpy as np

import matplotlib.pyplot as plt import pandas as pd

### Importing the dataset

dataset = pd.read\_csv('Social\_Network\_Ads.csv')

X = dataset.iloc[:, :-1].values

y = dataset.iloc[:, -1].values

**Splitting the dataset into the Training set and Test**

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25, random\_state = 0)

print(X\_train)

print(Y\_train)

print(X\_test)

print(y\_test)

### Feature Scaling

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25, random\_state = 0)

print(X\_train)

print(Y\_train)

print(X\_test)

print(y\_test)

**Training the Decision Tree Classification model on the Training set**

from sklearn.tree import DecisionTreeClassifier

classifier = DecisionTreeClassifier(criterion = 'entropy', random\_state = 0) classifier.fit(X\_train, y\_train)

### Predicting a new result

print(classifier.predict(sc.transform([[30,87000]])))

### Predicting the Test set results

y\_pred = classifier.predict(X\_test) print(np.concatenate((y\_pred.reshape(len(y\_pred),1), y\_test.reshape(len(y\_test),1)),1))

y\_pred = classifier.predict(X\_test) print(np.concatenate((y\_pred.reshape(len(y\_pred),1), y\_test.reshape(len(y\_test),1)),1))

### Making the Confusion Matrix

from sklearn.metrics import confusion\_matrix, accuracy\_score

cm = confusion\_matrix(y\_test, y\_pred)

print(cm)

accuracy\_score(y\_test, y\_pred)

**Visualising the Training set results**

from matplotlib.colors import ListedColormap

X\_set, y\_set = sc.inverse\_transform(X\_train), y\_train

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 10, stop = X\_set[:, 0].max() + 10, step = 0.25),

np.arange(start = X\_set[:, 1].min() - 1000, stop = X\_set[:, 1].max() + 1000, step = 0.25))

plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),

alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)

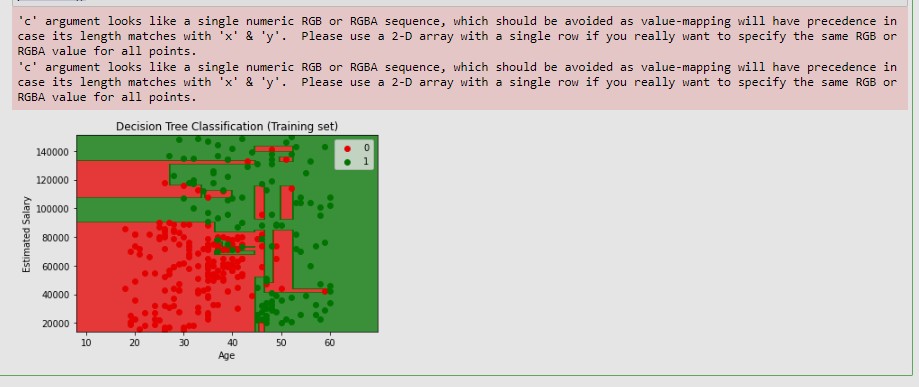
plt.title('Decision Tree Classification (Training set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

plt.show()



### Visualising the Test set results

from matplotlib.colors import ListedColormap

X\_set, y\_set = sc.inverse\_transform(X\_test), y\_test

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 10, stop = X\_set[:, 0].max() + 10, step = 0.25),

np.arange(start = X\_set[:, 1].min() - 1000, stop = X\_set[:, 1].max() + 1000, step = 0.25))

plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),

alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)

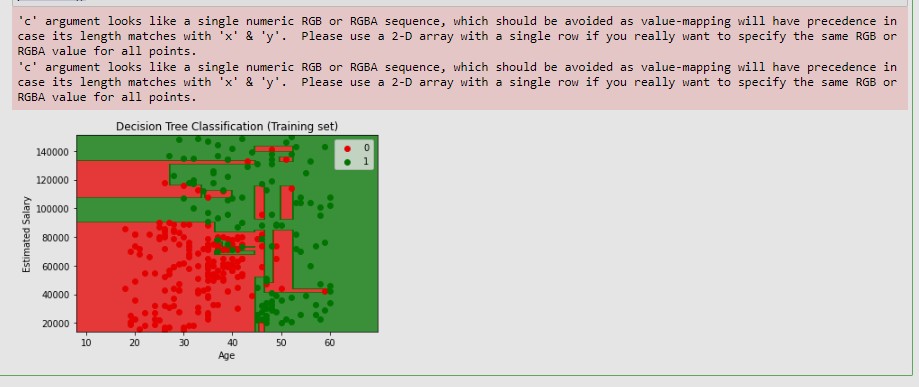
plt.title('Decision Tree Classification (Test set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

plt.show()



# Practical No. 5

## **Aim:** Implementation of K-Nearest Neighbors (K-NN)

1. **Nearest Neighbors:**

The k-nearest neighbors algorithm, also known as KNN or k-NN, is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point

**Code:**

**Importing the libraries**

import numpy as np

import matplotlib.pyplot as plt import pandas as pd

### Importing the dataset

dataset = pd.read\_csv('Social\_Network\_Ads.csv')

X = dataset.iloc[:, :-1].values y = dataset.iloc[:, -1].values

### Splitting the dataset into the Training set and Test

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25, random\_state = 0)

print(X\_train)

print(Y\_train)

print(X\_test)

print(y\_test)

### Feature Scaling

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

print(X\_train)

print(X\_test)

### Training the K-NN model on the Training set

from sklearn.neighbors import KNeighborsClassifier

classifier = KNeighborsClassifier(n\_neighbors = 5, metric = 'minkowski', p = 2) classifier.fit(X\_train, y\_train)

### Predicting a new result

print(classifier.predict(sc.transform([[30,87000]])))

### Predicting the Test set results

y\_pred = classifier.predict(X\_test) print(np.concatenate((y\_pred.reshape(len(y\_pred),1), y\_test.reshape(len(y\_test),1)),1))

### Making the Confusion Matrix

from sklearn.metrics import confusion\_matrix, accuracy\_score cm = confusion\_matrix(y\_test, y\_pred)

print(cm) accuracy\_score(y\_test,y\_prd)

### Visualising the Training set results

from matplotlib.colors import ListedColormap

X\_set, y\_set = sc.inverse\_transform(X\_train), y\_train

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 10, stop = X\_set[:, 0].max() + 10, step = 1),

np.arange(start = X\_set[:, 1].min() - 1000, stop = X\_set[:, 1].max() + 1000, step = 1))

plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),

alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)

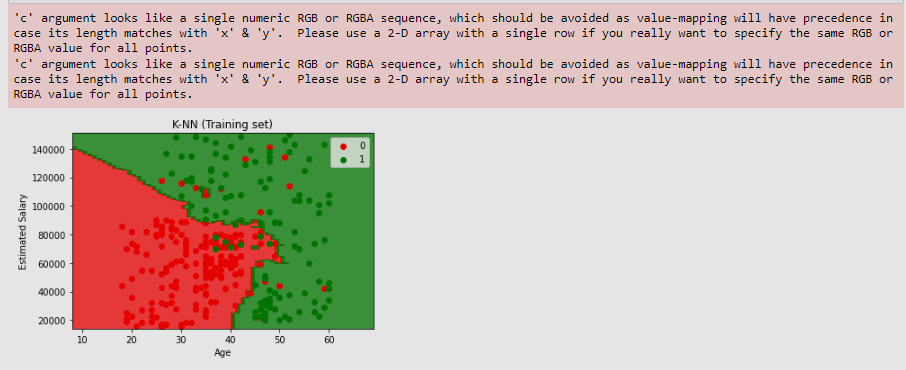
plt.title('K-NN (Training set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

plt.show()



### Visualizing the Test set results

from matplotlib.colors import ListedColormap X\_set, y\_set = sc.inverse\_transform(X\_test), y\_test

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 10, stop = X\_set[:, 0].max() + 10, step = 1), np.arange(start = X\_set[:, 1].min() - 1000, stop = X\_set[:, 1].max() + 1000, step = 1))

plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape), alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

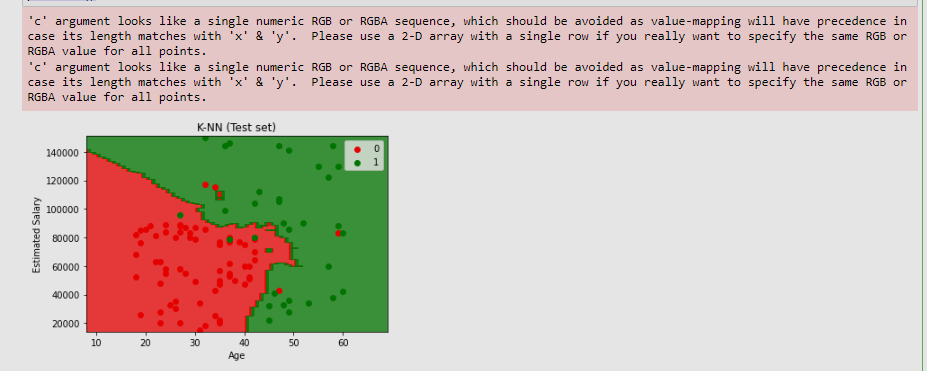
plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j) plt.title('K-NN (Test set)')

plt.xlabel('Age') plt.ylabel('Estimated Salary') plt.legend()

plt.show()



# Practical No. 6

## **Aim:** Implementation of Naive Bayes

**Naive Bayes:**

The Naïve Bayes classifier is a supervised machine learning algorithm, which is used for classification tasks, like text classification. It is also part of a family of generative learning algorithms, meaning that it seeks to model the distribution of inputs of a given class or category

**Code:**

### Importing the libraries

import numpy as np

import matplotlib.pyplot as plt import pandas as pd

### Importing the dataset

dataset = pd.read\_csv('Social\_Network\_Ads.csv')

X = dataset.iloc[:, :-1].values

y = dataset.iloc[:, -1].values

### Splitting the dataset into the Training set and Test

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25, random\_state = 0)

print(X\_train)

print(Y\_train)

print(X\_test)

print(y\_test)

### Feature Scaling

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

print(X\_train)

print(X\_test)

### Training the K-NN model on the Training set

from sklearn.neighbors import KNeighborsClassifier

classifier = KNeighborsClassifier(n\_neighbors = 5, metric = 'minkowski', p = 2) classifier.fit(X\_train, y\_train)

### Predicting a new result

print(classifier.predict(sc.transform([[30,87000]])))

### Predicting the Test set results

y\_pred = classifier.predict(X\_test) print(np.concatenate((y\_pred.reshape(len(y\_pred),1), y\_test.reshape(len(y\_test),1)),1))

### Making the Confusion Matrix

from sklearn.metrics import confusion\_matrix, accuracy\_score cm = confusion\_matrix(y\_test, y\_pred)

print(cm) accuracy\_score(y\_test, y\_pred)

### Visualizing the Training set results

from matplotlib.colors import ListedColormap

X\_set, y\_set = sc.inverse\_transform(X\_train), y\_train

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 10, stop = X\_set[:, 0].max() + 10, step = 1),

np.arange(start = X\_set[:, 1].min() - 1000, stop = X\_set[:, 1].max() + 1000, step = 1))

plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),

alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)

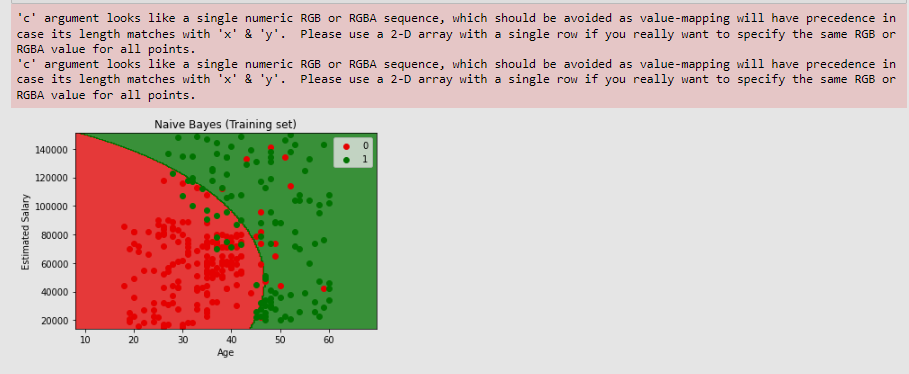
plt.title('K-NN (Training set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

plt.show()



### Visualising the Test set results

from matplotlib.colors import ListedColormap

X\_set, y\_set = sc.inverse\_transform(X\_test), y\_test

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 10, stop = X\_set[:, 0].max() + 10, step = 0.25),

np.arange(start = X\_set[:, 1].min() - 1000, stop = X\_set[:, 1].max() + 1000, step = 0.25))

plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),

alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)

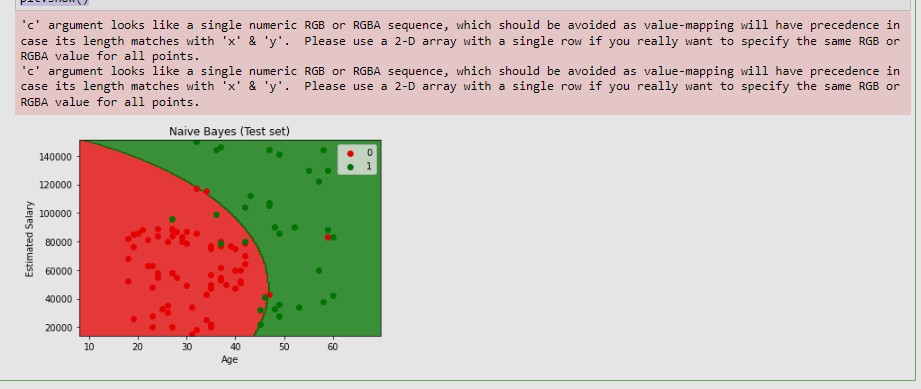
plt.title('Naive Bayes (Test set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

plt.show()



# Practical No. 7

## **Aim**: Implementation of K-Means Clustering

**K-Means Clustering:**

K-Means Clustering is an unsupervised learning algorithm that is used to solve the clustering problems in machine learning or data science. In this topic, we will learn what is K-means clustering algorithm, how the algorithm works, along with the Python implementation of k-means clustering**.**

**Code:**

**Importing the libraries**

import numpy as np

import matplotlib.pyplot as plt import pandas as pd

### Importing the dataset

dataset = pd.read\_csv('Mall\_Customers.csv')

X = dataset.iloc[:, [3, 4]].values

### Using the elbow method to find the optimal number of clusters

from sklearn.cluster import KMeans

wcss = []

for i in range(1, 11):

kmeans = KMeans(n\_clusters = i, init = 'k-means++', random\_state = 42)

kmeans.fit(X)

wcss.append(kmeans.inertia\_)

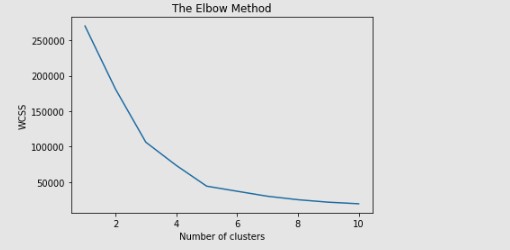
plt.plot(range(1, 11), wcss)

plt.title('The Elbow Method')

plt.xlabel('Number of clusters')

plt.ylabel('WCSS')

plt.show()



### Training the K-Means model on the dataset

kmeans = KMeans(n\_clusters = 5, init = 'k-means++', random\_state = 42) y\_kmeans = kmeans.fit\_predict(X)

### Visualising the clusters

plt.scatter(X[y\_kmeans == 0, 0], X[y\_kmeans == 0, 1], s = 100, c = 'red', label = 'Cluster 1')

plt.scatter(X[y\_kmeans == 1, 0], X[y\_kmeans == 1, 1], s = 100, c = 'blue', label = 'Cluster 2')

plt.scatter(X[y\_kmeans == 2, 0], X[y\_kmeans == 2, 1], s = 100, c = 'green', label = 'Cluster 3')

plt.scatter(X[y\_kmeans == 3, 0], X[y\_kmeans == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')

plt.scatter(X[y\_kmeans == 4, 0], X[y\_kmeans == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5') plt.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:, 1], s = 300, c = 'yellow', label = 'Centroids') plt.title('Clusters of customers')

plt.xlabel('Annual Income (k$)') plt.ylabel('Spending Score (1-100)') plt.legend()

plt.show()

