# Create a Large Dataset for Heart Disease with Appropriate Data Structures

**AIM:**

To create a large dataset for heart diseases with appropriate structures and fields like Patient name, Gender, Age, Admin date, Cholesterol, Fatal.

**STEPS:**

1. Import necessary packers:
   - Pandas - import pandas as pd
   - Numpy - import numpy as np
   - DateTime - from datetime import datetime

2. Create a dataframe using DataFrame method:

```
df=pd.DataFrame({"PatientName":pd.Categorical(["Akash","Aishwarya","Karthikeyan","Deepak","Manogna"]),"Gender":pd.Categorical(["Male","Female","Male","Male","Female"]),"Age":np.array([20,20,19,25,20]),"Admit_date":np.array([datetime(2020,7,1),datetime(2018,5,15),datetime(2023,12,12),datetime(2018,3,24),datetime(2020,5,17)]),"Cholesterol":np.array([180,200,150,300,280]),"Fatal":np.array([1,0,1,0,0])})
```

3. Display the created DataFrame:

```
df
```

| | PatientName | Gender | Age | Admit_date | Cholesterol | Fatal |
|---|---|---|---|---|---|---|
| 0 | Akash | Male | 20 | 2020-07-01 | 180 | 1 |
| 1 | Aishwarya | Female | 20 | 2018-05-15 | 200 | 0 |
| 2 | Karthikeyan | Male | 19 | 2023-12-12 | 150 | 1 |
| 3 | Deepak | Male | 25 | 2018-03-24 | 300 | 0 |
| 4 | Manogna | Female | 20 | 2020-05-17 | 280 | 0 |

4. Display first few records:

```
df.head(2)
```

| | PatientName | Gender | Age | Admit_date | Cholesterol | Fatal |
|---|---|---|---|---|---|---|
| 0 | Akash | Male | 20 | 2020-07-01 | 180 | 1 |
| 1 | Aishwarya | Female | 20 | 2018-05-15 | 200 | 0 |

5. Display last few records:

```
[9] df.tail(3)
```

| | PatientName | Gender | Age | Admit_date | Cholesterol | Fatal |
|---|---|---|---|---|---|---|
| 2 | Karthikeyan | Male | 19 | 2023-12-12 | 150 | 1 |
| 3 | Deepak | Male | 25 | 2018-03-24 | 300 | 0 |
| 4 | Manogna | Female | 20 | 2020-05-17 | 280 | 0 |

6. Display columns of the dataset:

```
df.columns
```

```
Index(['PatientName', 'Gender', 'Age', 'Admit_date', 'Cholesterol', 'Fatal'], dtype='object')
```

7. Display dataset values:

```
df.values
```

```
array([['Akash', 'Male', 20, Timestamp('2020-07-01 00:00:00'), 180, 1],
       ['Aishwarya', 'Female', 20, Timestamp('2018-05-15 00:00:00'), 200,
        0],
       ['Karthikeyan', 'Male', 19, Timestamp('2023-12-12 00:00:00'), 150,
        1],
       ['Deepak', 'Male', 25, Timestamp('2018-03-24 00:00:00'), 300, 0],
       ['Manogna', 'Female', 20, Timestamp('2020-05-17 00:00:00'), 280,
        0]], dtype=object)
```

8. Display statistical information:

```
[12] df.describe()
```

| | Age | Cholesterol | Fatal |
|---|---|---|---|
| count | 5.000000 | 5.000000 | 5.000000 |
| mean | 20.800000 | 222.000000 | 0.400000 |
| std | 2.387467 | 64.961527 | 0.547723 |
| min | 19.000000 | 150.000000 | 0.000000 |
| 25% | 20.000000 | 180.000000 | 0.000000 |
| 50% | 20.000000 | 200.000000 | 0.000000 |
| 75% | 20.000000 | 280.000000 | 1.000000 |
| max | 25.000000 | 300.000000 | 1.000000 |

9. Create a CSV file from the DataFrame:

df.to_csv('heart.csv')

10. Read the CSV file and display the information:

```
heart_df = pd.read_csv('heart.csv')
heart_df
```

| | Unnamed: 0 | PatientName | Gender | Age | Admit_date | Cholesterol | Fatal |
|---|---|---|---|---|---|---|---|
| 0 | 0 | Akash | Male | 20 | 2020-07-01 | 180 | 1 |
| 1 | 1 | Aishwarya | Female | 20 | 2018-05-15 | 200 | 0 |
| 2 | 2 | Karthikeyan | Male | 19 | 2023-12-12 | 150 | 1 |
| 3 | 3 | Deepak | Male | 25 | 2018-03-24 | 300 | 0 |
| 4 | 4 | Manogna | Female | 20 | 2020-05-17 | 280 | 0 |

11. Display the information of the DataFrame:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PatientName  5 non-null      category
 1   Gender       5 non-null      category
 2   Age          5 non-null      int64
 3   Admit_date   5 non-null      datetime64[ns]
 4   Cholesterol  5 non-null      int64
 5   Fatal        5 non-null      int64
dtypes: category(2), datetime64[ns](1), int64(3)
memory usage: 634.0 bytes
```

**RESULT:**

Program to create a large dataset for heart disease with appropriate attributes has been compiled and executed successfully.

# Data Visualization and Analysis

**AIM:**

To load the heart disease CSV file and visualize the data in different dimensions.

**STEPS:**

1. Import the necessary packages:
   - Pandas - import pandas as pd
   - Matplotlib - import matplotlib.pyplot as plt
   - Seaborn - import seaborn as sns

2. Read the CSV file and display the data:

   heart_df = pd.read_csv('heart.csv')

3. Display first few records from the data:

```
heart_df.head(3)
```

| | age | sex | cp | trestbps | chol | fbs | restecg | thalach | exang | oldpeak | slope | ca | thal | target |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 52 | 1 | 0 | 125 | 212 | 0 | 1 | 168 | 0 | 1.0 | 2 | 2 | 3 | 0 |
| 1 | 53 | 1 | 0 | 140 | 203 | 1 | 0 | 155 | 1 | 3.1 | 0 | 0 | 3 | 0 |
| 2 | 70 | 1 | 0 | 145 | 174 | 0 | 1 | 125 | 1 | 2.6 | 0 | 0 | 3 | 0 |

4. Check for null values:

```
heart_df.isna().sum()
```

```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

5. Display column with data type:

```
heart_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1025 entries, 0 to 1024
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       1025 non-null   int64
 1   sex       1025 non-null   int64
 2   cp        1025 non-null   int64
 3   trestbps  1025 non-null   int64
 4   chol      1025 non-null   int64
 5   fbs       1025 non-null   int64
 6   restecg   1025 non-null   int64
 7   thalach   1025 non-null   int64
 8   exang     1025 non-null   int64
 9   oldpeak   1025 non-null   float64
 10  slope     1025 non-null   int64
 11  ca        1025 non-null   int64
 12  thal      1025 non-null   int64
 13  target    1025 non-null   int64
dtypes: float64(1), int64(13)
memory usage: 112.2 KB
```

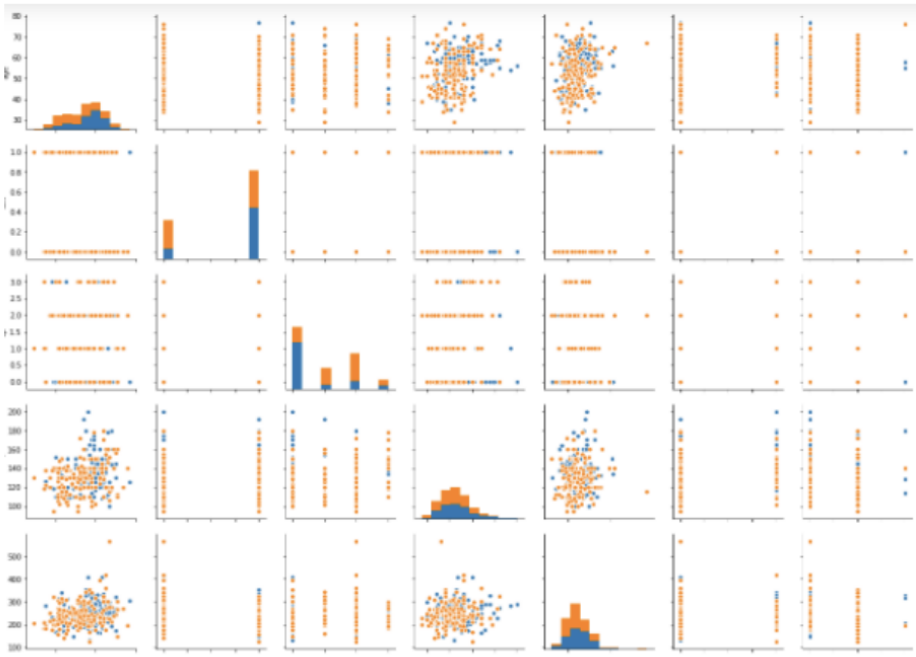
6. Correlation of the columns:

```
heart_df.corr()
```

| | age | sex | cp | trestbps | chol | fbs | restecg | thalach | exang | oldpeak | slope | ca | thal | target |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **age** | 1.000000 | -0.103240 | -0.071966 | 0.271121 | 0.219823 | 0.121243 | -0.132696 | -0.390227 | 0.088163 | 0.208137 | -0.169105 | 0.271551 | 0.072297 | -0.229324 |
| **sex** | -0.103240 | 1.000000 | -0.041119 | -0.078974 | -0.198258 | 0.027200 | -0.055117 | -0.049365 | 0.139157 | 0.084687 | -0.026666 | 0.111729 | 0.198424 | -0.279501 |
| **cp** | -0.071966 | -0.041119 | 1.000000 | 0.038177 | -0.081641 | 0.079294 | 0.043581 | 0.306839 | -0.401513 | -0.174733 | 0.131633 | -0.176206 | -0.163341 | 0.434854 |
| **trestbps** | 0.271121 | -0.078974 | 0.038177 | 1.000000 | 0.127977 | 0.181767 | -0.123794 | -0.039264 | 0.061197 | 0.187434 | -0.120445 | 0.104554 | 0.059276 | -0.138772 |
| **chol** | 0.219823 | -0.198258 | -0.081641 | 0.127977 | 1.000000 | 0.026917 | -0.147410 | -0.021772 | 0.067382 | 0.064880 | -0.014248 | 0.074259 | 0.100244 | -0.099966 |
| **fbs** | 0.121243 | 0.027200 | 0.079294 | 0.181767 | 0.026917 | 1.000000 | -0.104051 | -0.008866 | 0.049261 | 0.010859 | -0.061902 | 0.137156 | -0.042177 | -0.041164 |
| **restecg** | -0.132696 | -0.055117 | 0.043581 | -0.123794 | -0.147410 | -0.104051 | 1.000000 | 0.048411 | -0.065606 | -0.050114 | 0.086086 | -0.078072 | -0.020504 | 0.134468 |
| **thalach** | -0.390227 | -0.049365 | 0.306839 | -0.039264 | -0.021772 | -0.008866 | 0.048411 | 1.000000 | -0.380281 | -0.349796 | 0.395308 | -0.207888 | -0.098068 | 0.422895 |
| **exang** | 0.088163 | 0.139157 | -0.401513 | 0.061197 | 0.067382 | 0.049261 | -0.065606 | -0.380281 | 1.000000 | 0.310844 | -0.267335 | 0.107849 | 0.197201 | -0.438029 |
| **oldpeak** | 0.208137 | 0.084687 | -0.174733 | 0.187434 | 0.064880 | 0.010859 | -0.050114 | -0.349796 | 0.310844 | 1.000000 | -0.575189 | 0.221816 | 0.202672 | -0.438441 |
| **slope** | -0.169105 | -0.026666 | 0.131633 | -0.120445 | -0.014248 | -0.061902 | 0.086086 | 0.395308 | -0.267335 | -0.575189 | 1.000000 | -0.073440 | -0.094090 | 0.345512 |
| **ca** | 0.271551 | 0.111729 | -0.176206 | 0.104554 | 0.074259 | 0.137156 | -0.078072 | -0.207888 | 0.107849 | 0.221816 | -0.073440 | 1.000000 | 0.149014 | -0.382085 |
| **thal** | 0.072297 | 0.198424 | -0.163341 | 0.059276 | 0.100244 | -0.042177 | -0.020504 | -0.098068 | 0.197201 | 0.202672 | -0.094090 | 0.149014 | 1.000000 | -0.337838 |
| **target** | -0.229324 | -0.279501 | 0.434854 | -0.138772 | -0.099966 | -0.041164 | 0.134468 | 0.422895 | -0.438029 | -0.438441 | 0.345512 | -0.382085 | -0.337838 | 1.000000 |

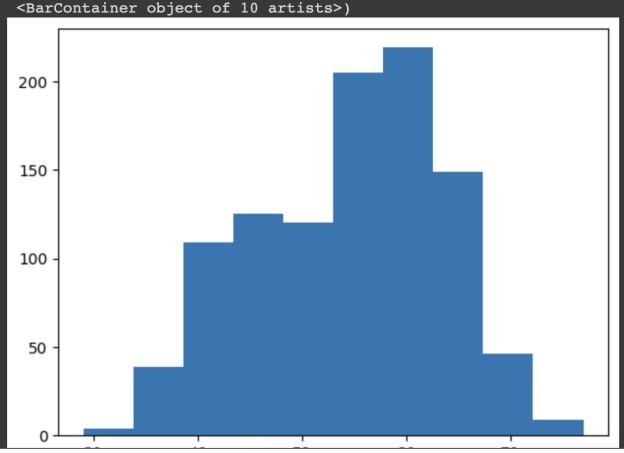
7. Display the linear relationship between the features of the column:

```
sns.pairplot(heart_df,hue='target')
```



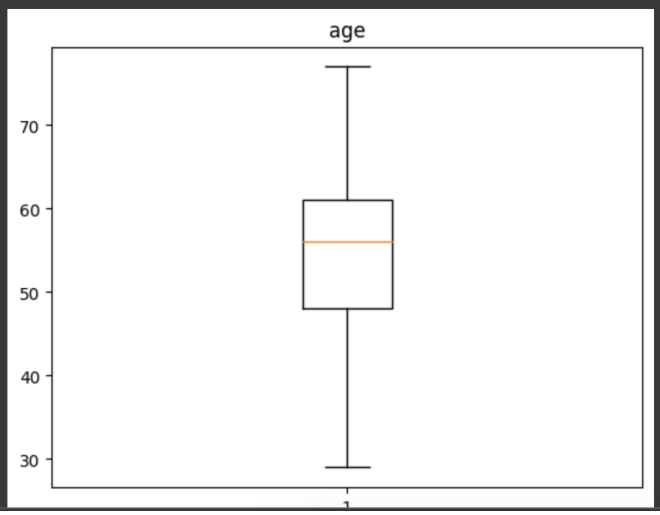
8. Shape of each feature and count of observations:

```
[14] heart_df.shape
     plt.hist(heart_df['age'])

     (array([  4.,  39., 109., 125., 120., 205., 219., 149.,  46.,   9.]),
      array([29. , 33.8, 38.6, 43.4, 48.2, 53. , 57.8, 62.6, 67.4, 72.2, 77. ]),
      <BarContainer object of 10 artists>)
```



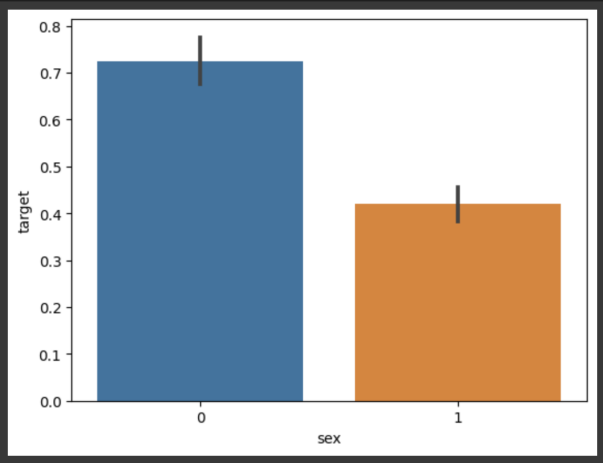

9. Draw Box-Plot to find outliers:

```
for x in heart_df.columns:
  plt.boxplot(heart_df[x])
  plt.title(x)
  plt.show()
```




10. Relationship between target and other variable using variable plots:

```
import seaborn as sns
sns.barplot(x='sex',y='target',data=heart_df)
plt.show()
```



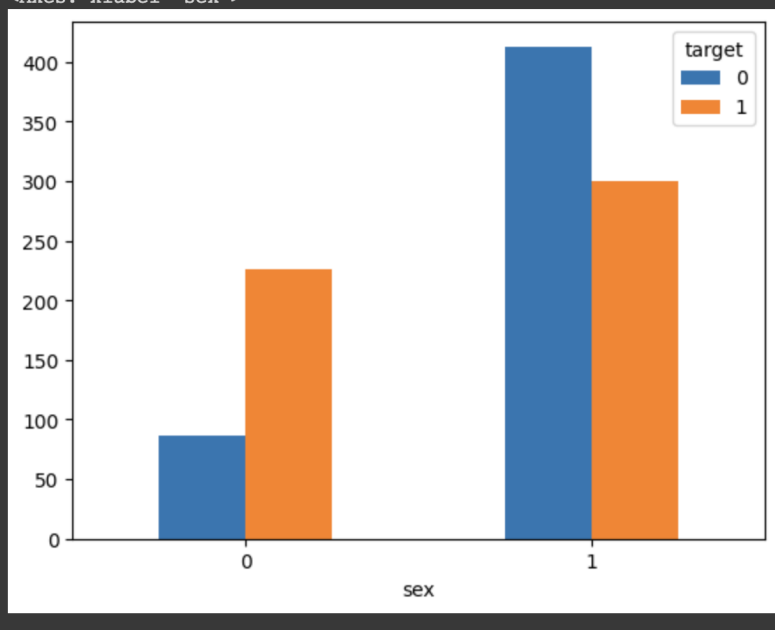

11. Print the count of male and female and check pain type:

```
[17] heart_df['sex'].value_counts()

     1    713
     0    312
     Name: sex, dtype: int64

[18] heart_df['cp'].value_counts()

     0    497
     2    284
     1    167
     3     77
     Name: cp, dtype: int64
```

12. Count of the target variable with respect to gender attribute:

```
my_crosstab = pd.crosstab(heart_df['sex'],heart_df['target'])
my_crosstab.plot(kind='bar', rot=0)
```

```
<Axes: xlabel='sex'>
```




**RESULT:**

Data Visualization and Analysis on the heart dataset has been compiled and executed successfully.

# Demonstration of Preprocessing on Sales Dataset

**AIM:**

To demonstrate the data preprocessing steps on a sales dataset.

**STEPS:**

1. Import necessary packages.

```
import pandas as pd
from sklearn.preprocessing import Imputer
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
```

2. Import sales dataset

```
import pandas as pd
x = pd.read_csv('/content/sample_data/salesnew.csv')
x = x[:-1]
x
```

| | S.No | Country | Age | Salary | Purchased |
|---|---|---|---|---|---|
| 0 | 1.0 | France | 44.0 | 72000.0 | No |
| 1 | 2.0 | Spain | 27.0 | 48000.0 | Yes |
| 2 | 3.0 | Germany | 30.0 | 54000.0 | No |
| 3 | 4.0 | Spain | 38.0 | 61000.0 | No |
| 4 | 5.0 | Germany | 40.0 | NaN | Yes |
| 5 | 6.0 | France | NaN | 58000.0 | Yes |
| 6 | 7.0 | Spain | NaN | 52000.0 | No |
| 7 | 8.0 | France | 48.0 | 79000.0 | Yes |
| 8 | 9.0 | Germany | 50.0 | 83000.0 | No |
| 9 | 10.0 | France | 37.0 | 67000.0 | Yes |

3. Handle missing values in age and salary columns

```
from sklearn.preprocessing import
Imputer
imputer =
Imputer(missing_values='NaN',
strategy='mean',axis = 0)
imputer =
imputer.fit(x.loc[:,"Age":"Salary"]
)
x.loc[:,"Age":"Salary"] =
imputer.transform(x.loc[:,"Age":"Sa
lary"])
x
```

| | S.No | Country | Age | Salary | Purchased |
|---|---|---|---|---|---|
| 0 | 1.0 | France | 44.00 | 72000.000000 | No |
| 1 | 2.0 | Spain | 27.00 | 48000.000000 | Yes |
| 2 | 3.0 | Germany | 30.00 | 54000.000000 | No |
| 3 | 4.0 | Spain | 38.00 | 61000.000000 | No |
| 4 | 5.0 | Germany | 40.00 | 63777.777778 | Yes |
| 5 | 6.0 | France | 39.25 | 58000.000000 | Yes |
| 6 | 7.0 | Spain | 39.25 | 52000.000000 | No |
| 7 | 8.0 | France | 48.00 | 79000.000000 | Yes |
| 8 | 9.0 | Germany | 50.00 | 83000.000000 | No |
| 9 | 10.0 | France | 37.00 | 67000.000000 | Yes |

4. Transform the data under category into numerical data

```
[ ] from sklearn import preprocessing
    le = preprocessing.LabelEncoder()
    x.loc[:,"Purchased"] = le.fit_transform(x.loc[:,"Purchased"])
    x.loc[:,"Country"] = le.fit_transform(x.loc[:,"Country"])
```

| | S.No | Country | Age | Salary | Purchased |
|---|---|---|---|---|---|
| 0 | 1.0 | 0 | 44.00 | 72000.000000 | 0 |
| 1 | 2.0 | 2 | 27.00 | 48000.000000 | 1 |
| 2 | 3.0 | 1 | 30.00 | 54000.000000 | 0 |
| 3 | 4.0 | 2 | 38.00 | 61000.000000 | 0 |
| 4 | 5.0 | 1 | 40.00 | 63777.777778 | 1 |
| 5 | 6.0 | 0 | 39.25 | 58000.000000 | 1 |
| 6 | 7.0 | 2 | 39.25 | 52000.000000 | 0 |
| 7 | 8.0 | 0 | 48.00 | 79000.000000 | 1 |
| 8 | 9.0 | 1 | 50.00 | 83000.000000 | 0 |
| 9 | 10.0 | 0 | 37.00 | 67000.000000 | 1 |

5. Split dataset into train/test and print training and testing

```
from sklearn.model_selection
import train_test_split
X = x.loc[:,"S.No":"Salary"]
Y = x.loc[:,"Purchased"]
x_train,x_test,y_train,y_test =
train_test_split(X,Y,test_size=
0.2,random_state=0)
print(x_train,"\n\n",x_test,"\n
\n",y_train,"\n\n",y_test)
```

```
   S.No  Country    Age        Salary
4   5.0        1  40.00  63777.777778
9  10.0        0  37.00  67000.000000
1   2.0        2  27.00  48000.000000
6   7.0        2  39.25  52000.000000
7   8.0        0  48.00  79000.000000
3   4.0        2  38.00  61000.000000
0   1.0        0  44.00  72000.000000
5   6.0        0  39.25  58000.000000

    S.No  Country   Age   Salary
2   3.0        1  30.0  54000.0
8   9.0        1  50.0  83000.0

 4    1
9    1
1    1
6    0
7    1
3    0
0    0
5    1
Name: Purchased, dtype: int32

 2    0
8    0
Name: Purchased, dtype: int32
```

6. Perform feature scaling on training and test dataset

```
[15] from sklearn.preprocessing import StandardScaler
     ss = StandardScaler()
     x_train = ss.fit_transform(x_train)
     x_test = ss.fit_transform(x_test)
```

```
[17] x_train

     array([[-0.13271219,  0.13483997,  0.16569781,  0.12381479],
            [ 1.63678363, -0.94387981, -0.36453519,  0.46175632],
            [-1.19440968,  1.21355975, -2.13197855, -1.53093341],
            [ 0.57508614,  1.21355975,  0.03313956, -1.11141978],
            [ 0.92898531, -0.94387981,  1.5796525 ,  1.7202972 ],
            [-0.48661135,  1.21355975, -0.18779086, -0.16751412],
            [-1.54830884, -0.94387981,  0.87267516,  0.98614835],
            [ 0.22118698, -0.94387981,  0.03313956, -0.48214934]])

[18] x_test

     array([[-1.,  0., -1., -1.],
            [ 1.,  0.,  1.,  1.]])
```

**RESULT:**

Demonstration of preprocessing of dataset done successfully.

# Implementation of Simple Linear Regression using Salary Dataset

**AIM:**

To implement simple linear regression using salary dataset.

**STEPS:**

1. Import necessary libraries and packages.
2. Read the salary dataset Salary_Data.csv.

```
import pandas as pd
df = pd.read_csv("/content/Salary_Data.csv")
df.head()
```

| | YearsExperience | Salary |
|---|---|---|
| 0 | 1.1 | 39343 |
| 1 | 1.3 | 46205 |
| 2 | 1.5 | 37731 |
| 3 | 2.0 | 43525 |
| 4 | 2.2 | 39891 |

3. Perform EDA (Exploratory Data Analysis) - check for null values, information about data, print statistical values using describe.

```
[5] df.corr()
```

| | YearsExperience | Salary |
|---|---|---|
| YearsExperience | 1.000000 | 0.978242 |
| Salary | 0.978242 | 1.000000 |

```
[3] df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30 entries, 0 to 29
Data columns (total 2 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   YearsExperience  30 non-null     float64
 1   Salary           30 non-null     int64
dtypes: float64(1), int64(1)
memory usage: 608.0 bytes
```

```
df.isna().sum()
```

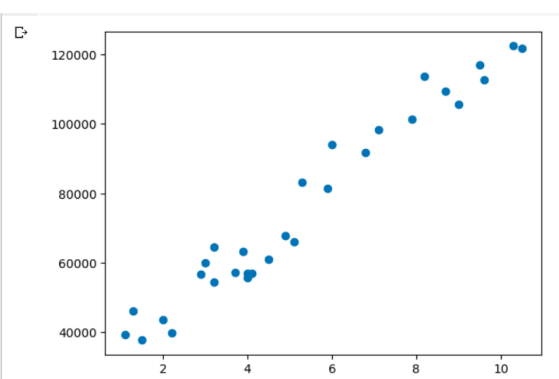
```
YearsExperience    0
Salary             0
dtype: int64
```

```
df.describe()
```

| | YearsExperience | Salary |
|---|---|---|
| count | 30.000000 | 30.000000 |
| mean | 5.313333 | 76003.000000 |
| std | 2.837888 | 27414.429785 |
| min | 1.100000 | 37731.000000 |
| 25% | 3.200000 | 56720.750000 |
| 50% | 4.700000 | 65237.000000 |
| 75% | 7.700000 | 100544.750000 |
| max | 10.500000 | 122391.000000 |

4. Prepare the data.
5. Visualise the data using scatter plot.

```
import matplotlib.pyplot as plt
import numpy as np
x = np.array(df["YearsExperience"]).reshape(-1,1)
y = np.array(df["Salary"]).reshape(-1,1)
plt.scatter(x,y)
plt.show()
```




6. Split the data.

7. Define the model.

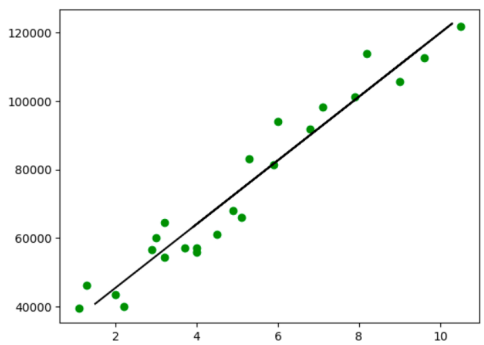
```
[8] from sklearn.model_selection import train_test_split as tts
    from sklearn.linear_model import LinearRegression
    xtrain,xtest,ytrain,ytest = tts(x,y,test_size=0.2,random_state=0)
    lr = LinearRegression().fit(xtrain,ytrain)
```

```
p = lr.predict(xtest)
p
```

```
array([[ 40748.96184072],
       [122699.62295594],
       [ 64961.65717022],
       [ 63099.14214487],
       [115249.56285456],
       [107799.50275317]])
```

8. Plot the visualization for training and testing data.
9. Print the predicted values and the actual values.

```
plt.scatter(xtrain, ytrain,color='g')
plt.plot(xtest, p,color='k')
plt.show()
```




10. Print the metrics Root Mean-Squared Error (RMSE), $R^2$ and MSE (Mean Squared Error), intercept and coefficient of regression.

```
[11] from sklearn.metrics import r2_score
     from sklearn.metrics import mean_squared_error
     from sklearn.metrics import mean_absolute_error
     from sklearn.metrics import median_absolute_error
     from math import sqrt
```

```
[14] print("""Mean Squared Error : {0} \n r2 Score: {1} \n Mean Absolute Error: {2}
      \n Median Absolute Error: {3} \n Root Mean Squared Error: {4}""".format(
         mean_squared_error(ytest,p),
         r2_score(ytest,p),
         mean_absolute_error(ytest,p),
         median_absolute_error(ytest,p),
         sqrt(mean_squared_error(ytest,p))
     ))
```

```
Mean Squared Error : 12823412.298126549
 r2 Score: 0.988169515729126
 Mean Absolute Error: 2446.1723690465055

 Median Absolute Error: 1675.4671961362037
 Root Mean Squared Error: 3580.979237321343
```

11. Predict the values for custom data.

```
print('Slope:' ,lr.coef_)
print('Intercept:', lr.intercept_)
```

```
Slope: [[9312.57512673]]
Intercept: [26780.09915063]
```

```
inp = int(input("YoE: "))
l = np.array(inp).reshape(-1,1)
lr.predict(l)
```

```
YoE: 5
array([[73342.97478427]])
```

**RESULT:**

Demonstration of preprocessing of dataset done successfully.

# Implementation of Multiple Linear Regression Using House Dataset

**AIM:**

To implement multiple linear regression on the house dataset.

**STEPS:**

1. Importing necessary libraries:

- import pandas as pd
- import matplotlib.pyplot as plt
- import seaborn as sns
- from sklearn.model_selection import train_test_split
- from sklearn.linear_model import LinearRegression
- from sklearn.metrics import mean_squared_error,r2_score

2. Reading the dataset

df = pd.read_csv('ds/kc_house_data.csv')

3. Print the first 5 records.

df.head(5)

| | id | date | price | bedrooms | bathrooms | sqft_living | sqft_lot | floors | waterfront | view | ... | grade | sqft_above | sqft_basement |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 7129300520 | 20141013T000000 | 221900.0 | 3 | 1.00 | 1180 | 5650 | 1.0 | 0 | 0 | ... | 7 | 1180 | 0 |
| 1 | 6414100192 | 20141209T000000 | 538000.0 | 3 | 2.25 | 2570 | 7242 | 2.0 | 0 | 0 | ... | 7 | 2170 | 400 |
| 2 | 5631500400 | 20150225T000000 | 180000.0 | 2 | 1.00 | 770 | 10000 | 1.0 | 0 | 0 | ... | 6 | 770 | 0 |
| 3 | 2487200875 | 20141209T000000 | 604000.0 | 4 | 3.00 | 1960 | 5000 | 1.0 | 0 | 0 | ... | 7 | 1050 | 910 |
| 4 | 1954400510 | 20150218T000000 | 510000.0 | 3 | 2.00 | 1680 | 8080 | 1.0 | 0 | 0 | ... | 8 | 1680 | 0 |

5 rows × 21 columns

df.isna().sum()

```
df.isna().sum()
```

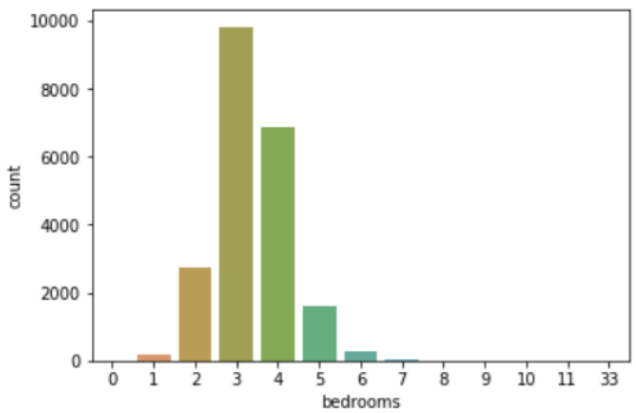
```
id               0
date             0
price            0
bedrooms         0
bathrooms        0
sqft_living      0
sqft_lot         0
floors           0
waterfront       0
view             0
condition        0
grade            0
sqft_above       0
sqft_basement    0
yr_built         0
yr_renovated     0
zipcode          0
lat              0
long             0
sqft_living15    0
sqft_lot15       0
dtype: int64
```

4. Handling missing values

5. Count plot on feature bedroom

sns.countplot(x='bedrooms',data=df)

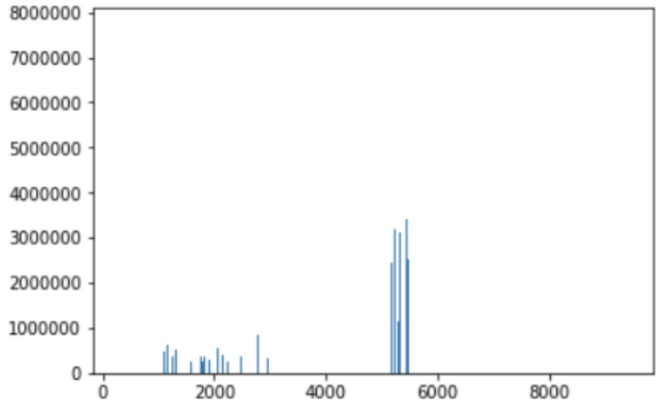
```
<matplotlib.axes._subplots.AxesSubplot at 0x1bbb9d2d748>
```




6. Bar plot between sqrt_above and price

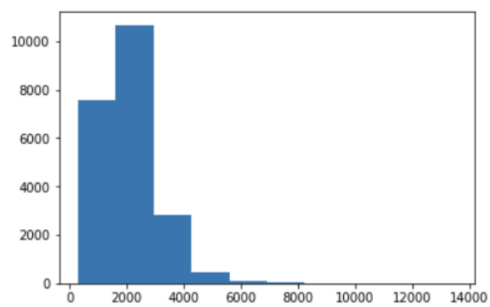
```
plt.bar(df['sqft_above'],df['price'])
```

```
<BarContainer object of 21613 artists>
```



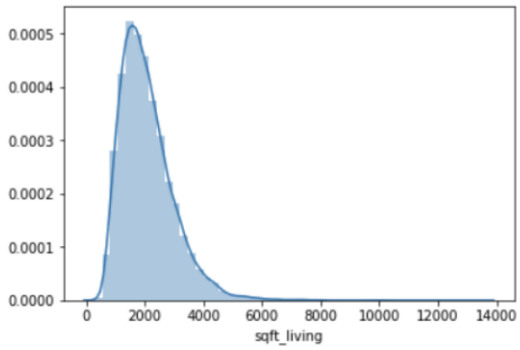

7. Histogram for sqrt_living

```
plt.hist(df['sqft_living'])
plt.show()
```




8. Distplot between sqft_living and density

```
sns.distplot(df['sqft_living'])
```




9. Mean, Median and Mode values

Mean

```
[ ] df.mean(numeric_only=True)
```

```
id                4.580302e+09
price             5.401822e+05
bedrooms          3.370842e+00
bathrooms         2.114757e+00
sqft_living       2.079900e+03
sqft_lot          1.510697e+04
floors            1.494309e+00
waterfront        7.541757e-03
view              2.343034e-01
condition         3.409430e+00
grade             7.656873e+00
sqft_above        1.788391e+03
sqft_basement     2.915090e+02
yr_built          1.971005e+03
yr_renovated      8.440226e+01
zipcode           9.807794e+04
lat               4.756005e+01
long             -1.222139e+02
sqft_living15     1.986552e+03
sqft_lot15        1.276846e+04
dtype: float64
```

## Median

```
df.median(numeric_only=True)
```

```
id               3.904930e+09
price            4.500000e+05
bedrooms         3.000000e+00
bathrooms        2.250000e+00
sqft_living      1.910000e+03
sqft_lot         7.618000e+03
floors           1.500000e+00
waterfront       0.000000e+00
view             0.000000e+00
condition        3.000000e+00
grade            7.000000e+00
sqft_above       1.560000e+03
sqft_basement    0.000000e+00
yr_built         1.975000e+03
yr_renovated     0.000000e+00
zipcode          9.806500e+04
lat              4.757180e+01
long            -1.222300e+02
sqft_living15    1.840000e+03
sqft_lot15       7.620000e+03
dtype: float64
```

## Mode

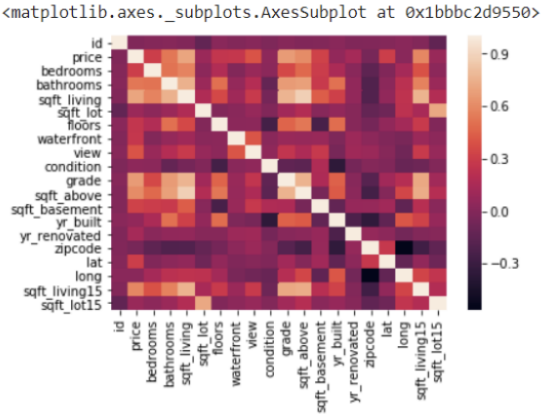
```
[ ] df.mode(numeric_only=True)
```

| | id | price | bedrooms | bathrooms | sqft_living | sqft_lot | floors | waterfront | view | condition |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 795000620.0 | 350000.0 | 3.0 | 2.5 | 1300.0 | 5000.0 | 1.0 | 0.0 | 0.0 | 3.0 |
| 1 | NaN | 450000.0 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| 2 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| 3 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |

10. Correlation among features

```
sns.heatmap(df.corr())
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1bbbc2d9550>
```



11. Splitting the dataset

```
independent_data = df[df.columns[3:]]
dependent_data = df['price']
x_train,x_test,y_train,y_test =
train_test_split(independent_data,dependent_data,test_size=0.3)
```

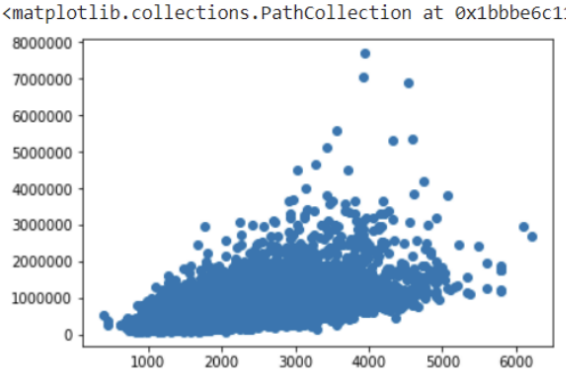
12. Perform linear regression

```
model1 = LinearRegression().fit(x_train[['grade']],y_train)
train_pred_model1 = model1.predict(x_train[['grade']])
test_pred_model1 = model1.predict(x_test[['grade']])
```

```
mean_squared_error(train_pred_model1,y_trai
76148352956.92657

[ ] r2_score(train_pred_model1,y_train)
-0.27038775707748863

[ ] r2_score(test_pred_model1,y_test)
-0.16151643391229697

[ ] print(model1.intercept_)
-1061522.9782836407

[ ] print(model1.coef_)
[209242.30797455]
```

13. Scatter plot between price and living space

```
plt.scatter(df[['sqft_living15']],df[['price']])
<matplotlib.collections.PathCollection at 0x1bbbe6c11d0>
```



14. Multiple linear regression on the data

```
model = LinearRegression()
model.fit(x_train,y_train)
```

```
mean_squared_error(model.predict(x_train),y_train)
```

```
40943775357.695114
```

```
r2_score(model.predict(x_train),y_train)
```

```
0.5696724671557423
```

```
r2_score(model.predict(x_test),y_test)
```

```
0.5771840747506204
```

```
print(model.coef_)
```

```
[-3.63678119e+04  3.89105359e+04  1.13079793e+02  7.94558296e-02
  1.02646944e+04  5.86355554e+05  5.24045335e+04  2.74921884e+04
  9.74455022e+04  7.25074745e+01  4.05723185e+01 -2.72554803e+03
  1.97352590e+01 -5.47490754e+02  6.00355056e+05 -1.97582469e+05
  1.59817467e+01 -3.86360526e-01]
```

```
print(model.intercept_)
```

```
5670844.520471991
```

**RESULT:**

Implementation of multiple linear regression using the house dataset has been compiled and executed successfully.

# IMPLEMENTATION OF LOGISTIC REGRESSION

**AIM:**

To implement logistic regression on heart dataset

**PROCEDURE:**

1. Import necessary packages

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

2. Read the csv file and display the data

```
dataset = pd.read_csv("heart (3).csv")
print(dataset)
```

```
      age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  \
0      52    1   0       125   212    0        1      168      0      1.0
1      53    1   0       140   203    1        0      155      1      3.1
2      70    1   0       145   174    0        1      125      1      2.6
3      61    1   0       148   203    0        1      161      0      0.0
4      62    0   0       138   294    1        1      106      0      1.9
...   ...  ...  ..       ...   ...  ...      ...      ...    ...      ...
1020   59    1   1       140   221    0        1      164      1      0.0
1021   60    1   0       125   258    0        0      141      1      2.8
1022   47    1   0       110   275    0        0      118      1      1.0
1023   50    0   0       110   254    0        0      159      0      0.0
1024   54    1   0       120   188    0        1      113      0      1.4

      slope  ca  thal  target
0         2   2     3       0
1         0   0     3       0
2         0   0     3       0
3         2   1     3       0
4         1   3     2       0
...     ...  ..   ...     ...
1020      2   0     2       1
1021      1   1     3       0
1022      1   1     2       0
1023      2   0     2       1
1024      1   1     3       0

[1025 rows x 14 columns]
```

3. Select the input and output columns

```
x = dataset.iloc[:, [0, 4]].values
y = dataset.iloc[:, -1].values
```

4. Split the dataset into train and test data

```
from sklearn.model_selection import train_test_split
xtrain, xtest, ytrain, ytest = train_test_split( x, y, test_size=0.2, random_state=0)
```

5. Train the model

```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression (random_state = 0)
classifier.fit(xtrain, ytrain)
y_pred = classifier.predict(xtest)
```

6. Analyze the results using evaluation metrics

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(ytest, y_pred)
print ("Confusion Matrix: \n", cm)
```

```
Confusion Matrix:
 [[61 37]
 [32 75]]
```

7. Predict the accuracy, sensitivity, specificity score

```
from sklearn.metrics import accuracy_score
print ("Accuracy: ", accuracy_score (ytest, y_pred))
```

```
Accuracy:  0.6634146341463415
```

```
sen = cm[1][1] / float(cm[1][1]+cm[1][0])
spe = cm[0][0] / float(cm[0][0]+cm[0][1])
acc = (cm[1][1] + cm[0][0]) / float(cm[0][0]+cm[0][1]+cm[1][0]+cm[1][1])
print(" Accuracy : ",acc,"\n","Sensitivity : ",sen,"\n","Specificity : ",spe)
```

```
Accuracy :  0.6634146341463415
Sensitivity :  0.7009345794392523
Specificity :  0.6224489795918368
```

8. Visualize the performance of the model

```
from matplotlib.colors import ListedColormap

X_set, y_set = xtest, ytest
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1,
                     stop = X_set[:, 0].max() + 1, step = 0.01),
                 np.arange(start = X_set[:, 1].min() - 1,
                     stop = X_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(
      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('Classifier (Test set)')
plt.xlabel('Age')
plt.ylabel('Cholestrol')
plt.legend()
plt.show()
```

```
<ipython-input-8-6c0fb6628f73>:19: UserWarning: *c* argument looks like a single numeric RGB
  plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
```

**Result:**

Thus the logistic regression has been implemented on the heart dataset.

# IMPLEMENTATION OF APRIORI ALGORITHM ON GROCERIES DATASET

## AIM

To write a R program to:

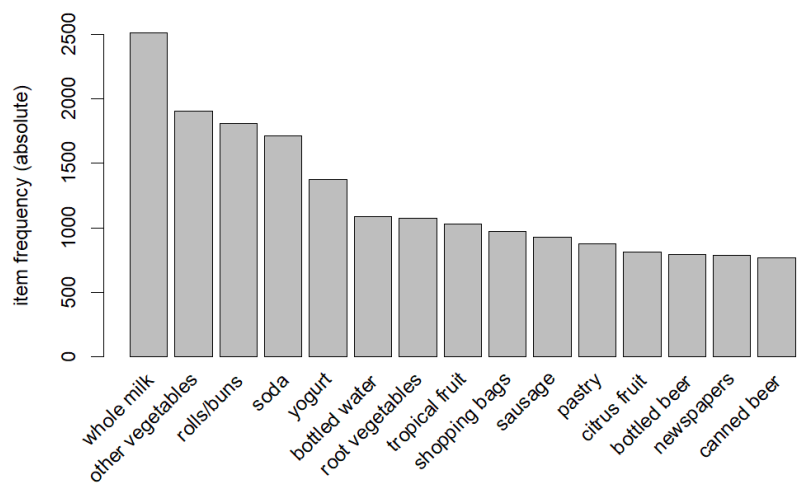
a) Apply Apriori algorithm on Groceries dataset.
b) Find the first 5 rules generated when the min support is 0.001 (0.1%) and min confidence is 0.9 (90%).

## STEPS

1. Import necessary libraries.
2. Plot a bar graph which represents the frequency of 15 most frequently occurring items.
3. Apply apriori algorithm on the groceries dataset along with a supply of 0.01 and confidence of 0.8.
4. Apply the apriori algorithm on the 'whole milk' itemset and sort the rules in descending order of confidence. Create datasets which default to both 'lhs' and 'rhs'.
5. Plot the data present in rules as an interactive graph.

## CODE

```
# Import necessary libraries
library(datasets)
library(arules)
library(arulesViz)
data(Groceries)
itemFrequencyPlot(Groceries, topN = 15, type="absolute")
```




```
rules <- apriori(Groceries, parameter = list(supp = 0.001, conf =
0.8))
options(digits = 2)
inspect(rules[1:5])
```

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| [1] | {liquor,red/blush wine} | {bottled beer} | 0.0019 | 0.90 | 0.0021 | 11.2 | 19 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **[2]** | {curd,cereals} | {whole milk} | 0.0010 | 0.91 | 0.0011 | 3.6 | 10 |
| **[3]** | {yogurt,cereals} | {whole milk} | 0.0017 | 0.81 | 0.0021 | 3.2 | 17 |
| **[4]** | {butter,jam} | {whole milk} | 0.0010 | 0.83 | 0.0012 | 3.3 | 10 |
| **[5]** | {soups,bottled beer} | {whole milk} | 0.0011 | 0.92 | 0.0012 | 3.6 | 11 |

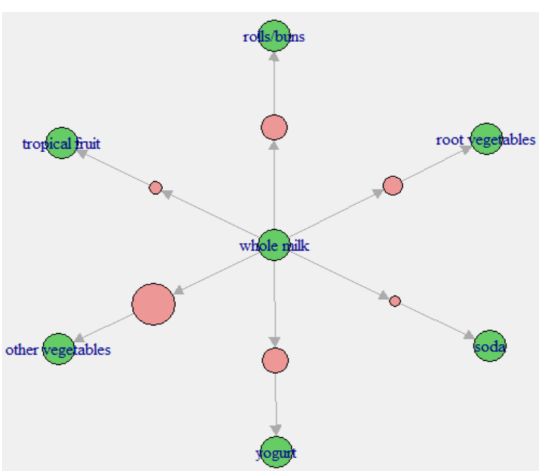
```
rules <- sort(rules, by = "confidence", decreasing = TRUE)
rules <- apriori(data = Groceries, parameter = list(supp = 0.001,
conf = 0.08), appearance = list(default = "lhs",rhs = "whole milk"),
control = list(verbose = F))
rules <- sort(rules, decreasing = TRUE, by = "confidence")
inspect(rules[1:5])
```

| | Lhs <chr> | rhs <chr> | Support <dbl> | Confidence <dbl> | Coverage <dbl> | Lift <dbl> | Count <dbl> |
|---|---|---|---|---|---|---|---|
| 1 | {rice, sugar} | {whole milk} | 0.0012 | 1 | 0.0012 | 3.9 | 12 |
| 2 | {canned fish, hygiene articles} | {whole milk} | 0.0011 | 1 | 0.0011 | 3.9 | 11 |
| 3 | {root vegetables, butter, rice } | {whole milk} | 0.0010 | 1 | 0.0010 | 3.9 | 10 |
| 4 | {root vegetables, whipped/sour cream } | {whole milk} | 0.0017 | 1 | 0.0017 | 3.9 | 17 |
| 5 | {butter, soft cheese, domestic eggs } | {whole milk} | 0.0010 | 1 | 0.0010 | 3.9 | 10 |

```
rules <- apriori(data = Groceries, parameter=list(supp = 0.001, conf
= 0.15, minlen = 2),appearance = list(default = "rhs",lhs = "whole
milk"),control = list(verbose=F))
rules <- sort(rules, decreasing = TRUE, by = "confidence")
inspect(rules[1:5])
```

| | lhs **<chr>** | rhs **<chr>** | Support **<dbl>** | Confidence **<dbl>** | Coverage **<dbl>** | Lift **<dbl>** | Count **<dbl>** |
|---|---|---|---|---|---|---|---|
| **[1]** | {whole milk} | {other vegetables} | 0.0075 | 0.29 | 0.26 | 1.5 | 736 |
| **[2]** | {whole milk} | {rolls/buns } | 0.0057 | 0.22 | 0.26 | 1.2 | 557 |
| **[3]** | {whole milk} | {yogurt } | 0.0056 | 0.22 | 0.26 | 1.6 | 551 |
| **[4]** | {whole milk} | {root vegetables } | 0.0049 | 0.19 | 0.26 | 1.8 | 481 |
| **[5]** | {whole milk} | {tropical fruit } | 0.0042 | 0.17 | 0.26 | 1.6 | 416 |

```
plot(rules, method="graph", shading=NA, engine="interactive")
```



**RESULT:** The R program to apply the Apriori algorithm on the groceries dataset was successfully implemented.

## Implementation of K-Means clustering on Iris Dataset using R Programming.

**Aim:**

To implement K-means clustering on Iris dataset using R programming.

**Procedure:**

1. Install ggplot2 an open-source data visualization package for the statistical programming language R

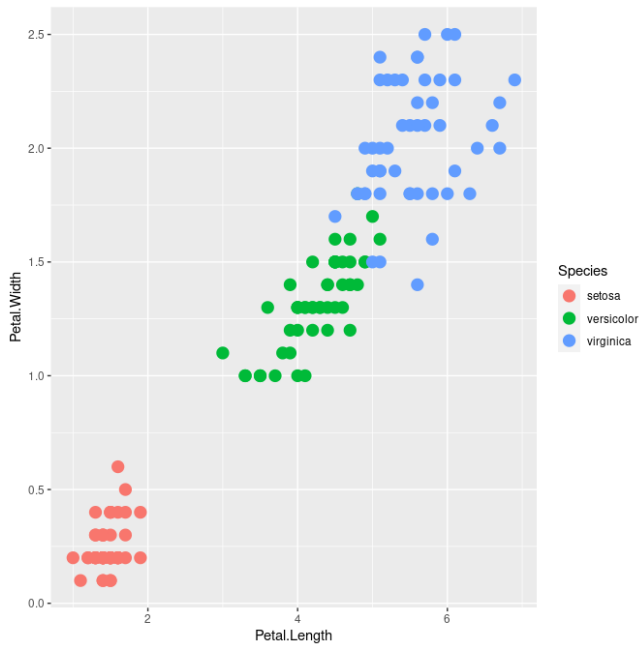
```
> library(ggplot2)
```

2. Import Iris dataset

```
> df <- iris
> head(iris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
```

3. Make scatterplot of the given dataset

```
> ggplot(df, aes(Petal.Length, Petal.Width)) + geom_point(aes(col=Species), size=4)
```




4. In the kmeans function, it is necessary to set the center, which is the number of groups we want to cluster to.

```
> set.seed(101)
> irisCluster <- kmeans(df[,1:4], center=3, nstart=20)
> irisCluster
```

```
K-means clustering with 3 clusters of sizes 38, 62, 50

Cluster means:
  Sepal.Length Sepal.Width Petal.Length Petal.Width
1     6.850000    3.073684     5.742105    2.071053
2     5.901613    2.748387     4.393548    1.433871
3     5.006000    3.428000     1.462000    0.246000

Clustering vector:
  [1] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 [38] 3 3 3 3 3 3 3 3 3 3 3 3 3 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 [75] 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 1 1 1 1 2 1 1 1 1
[112] 1 1 2 2 1 1 1 1 2 1 2 1 2 1 1 2 2 1 1 1 1 1 2 1 1 1 1 2 1 1 1 2 1 1 1 2 1
[149] 1 2

Within cluster sum of squares by cluster:
[1] 23.87947 39.82097 15.15100
 (between_SS / total_SS =  88.4 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
[6] "betweenss"    "size"         "iter"         "ifault"
```

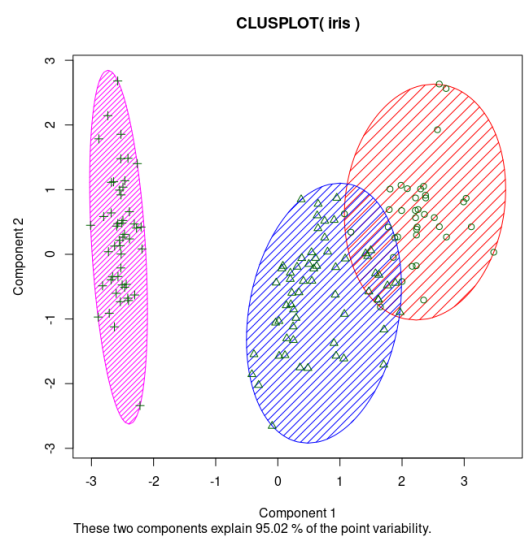
5. Compare the predicted clusters with the original data

```
> table(irisCluster$cluster, df$Species)

   setosa versicolor virginica
1       0          2        36
2       0         48        14
3      50          0         0
```

6. We plot out these clusters.

```
> library(cluster)
> clusplot(iris, irisCluster$cluster, color=T, shade=T, labels=0, lines=0)
```

7. View Graph




**Result:**

Implementation of K-means clustering on Iris dataset using R programming is done successfully

# IMPLEMENTATION OF K-MEANS CLUSTERING USING ELBOW METHOD

**Aim:**

To implement K-Means clustering using the Elbow method.

**Steps:**

1. Select the number of clusters for the dataset (K).
2. Select the K number of centroids randomly from the dataset.
3. Now we will use Euclidean distance or Manhattan distance as the metric to
4. Calculate the distance of the points from the nearest centroid and assign the points to that nearest cluster centroid, thus creating K clusters.
5. Now we find the new centroid of the clusters thus formed.
6. Again reassign the whole data point based on this new centroid, then repeat step 4.
7. We will continue this for a given number of iterations until the position of the centroid doesn't change, i.e. There is no more convergence.

Finding the optimal number of clusters is an important part of this algorithm. A commonly used method for finding the optimum K value is the Elbow Method.

**Code:**

1. Import Iris dataset

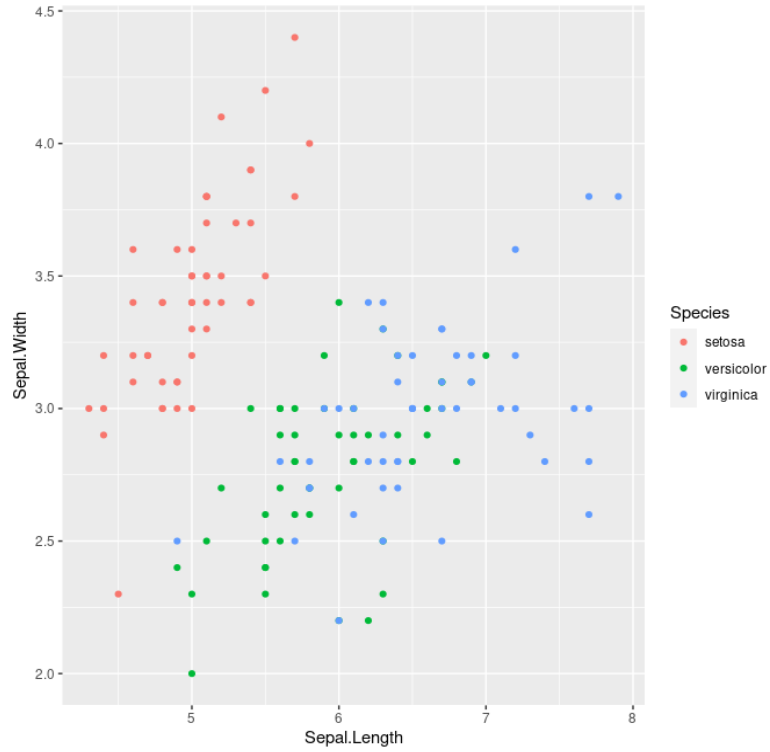
```
> df <- iris
> head(iris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
```

2. Scale the dataset

```
> irisScale = scale(iris[,-5])
> head(irisScale)
     Sepal.Length Sepal.Width Petal.Length Petal.Width
[1,]   -0.8976739  1.01560199    -1.335752   -1.311052
[2,]   -1.1392005 -0.13153881    -1.335752   -1.311052
[3,]   -1.3807271  0.32731751    -1.392399   -1.311052
[4,]   -1.5014904  0.09788935    -1.279104   -1.311052
[5,]   -1.0184372  1.24503015    -1.335752   -1.311052
[6,]   -0.5353840  1.93331463    -1.165809   -1.048667
```

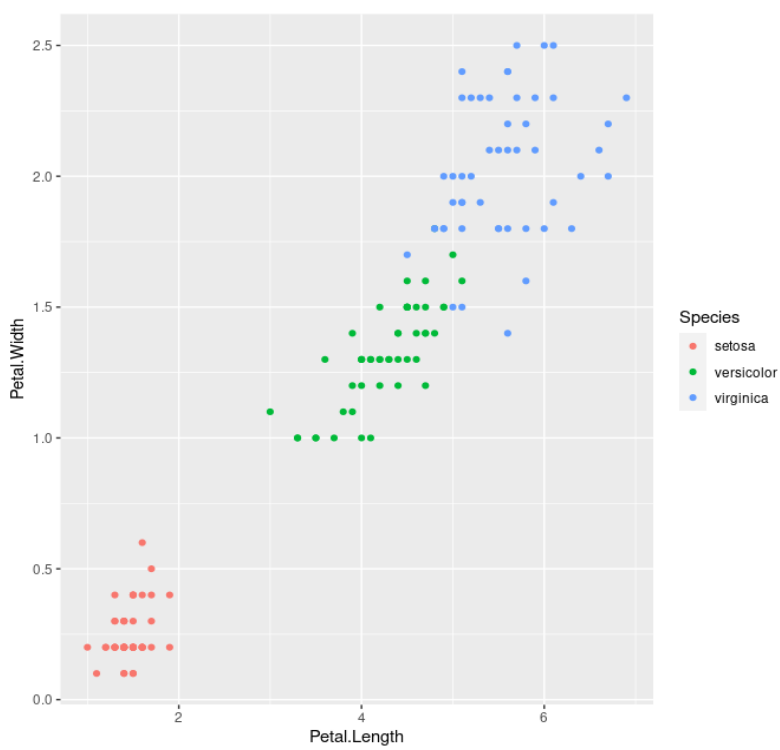
3. Plot the sepal data

```
> ggplot(iris,aes(x = Sepal.Length, y = Sepal.Width, col=
Species)) + geom_point()
```



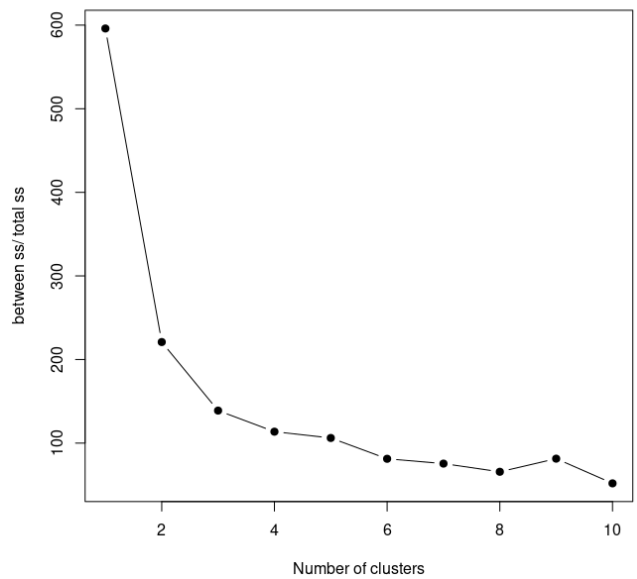

4. Plot the petal data

```
> ggplot(iris,aes(x = Petal.Length, y = Petal.Width, col=
Species)) + geom_point()
```



5. Finding the optimum number of clusters using Elbow method

```
> K.max = 10  #maximum 10 cluster assume
> wss = rep(NA, K.max = 10) #Repeat NA for 10 cluster i.e create vector with NA of size 10/1
> nClust = list() #Initialize empty list for number of cluster
>
> for(i in 1:K.max){
+   irisclasses= kmeans(irisScale,i)    #object stored in iris class
+   wss[i] = irisclasses$tot.withinss   #in irisclass,total wss distance allocated
+   nClust[[i]] = irisclasses$size      #size allocated to componant of list you have created
+ }
> plot(1:K.max, wss,
+      type="b",pch = 19,   #b means both(line,point), pch: specify symbol used along with plot
+      xlab= "Number of clusters",
+      ylab="between ss/ total ss")
```

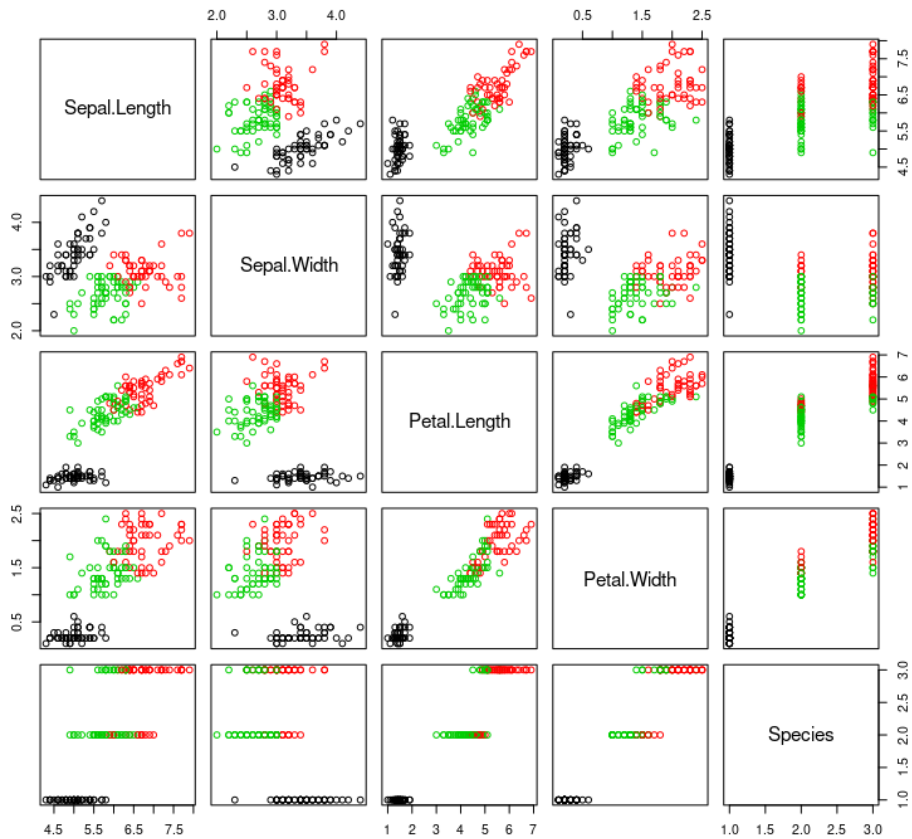



6. Fit Kmeans model for the data

```
> fitK = kmeans(irisScale, 3)
> str(fitK)
List of 9
 $ cluster     : int [1:150] 1 1 1 1 1 1 1 1 1 1 ...
 $ centers     : num [1:3, 1:4] -1.0112 1.1322 -0.0501 0.8504 0.0881 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:3] "1" "2" "3"
  .. ..$ : chr [1:4] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width"
 $ totss       : num 596
 $ withinss    : num [1:3] 47.4 47.5 44.1
 $ tot.withinss: num 139
 $ betweenss   : num 457
 $ size        : int [1:3] 50 47 53
 $ iter        : int 2
 $ ifault      : int 0
 -_attr(*, "class")= chr "kmeans"
```

7. Plot graph after fitting

```
> plot(iris,col = fitK$cluster)
```




8. Predict the fits

```
> table(Predicted=fitK$cluster,Actual =iris$Species)
         Actual
Predicted setosa versicolor virginica
        1     50          0         0
        2      0         11        36
        3      0         39        14
```

9. Print model metrics

```
> #Total number of correctly classified instances are:
> 50 + 39 + 36
[1] 125
> #Total number of incorrectly classified instances are:
> 11 + 14
[1] 25
> #Accuracy
> 125/(125+25)    #i.e our model has achieved 83% accuracy
[1] 0.8333333
```

**Result:**

K-Means clustering using Elbow method is implemented successfully

# IMPLEMENTATION OF LOGISTIC REGRESSION ON mtcars DATASET

## AIM

To implement the logistic regression algorithm on the *mtcars* dataset using R programming language.

## STEPS

1. Import the mtcars dataset.
2. This dataset describes different models of a car with their various engine specifications.
3. The transmission mode is described by the column 'am' which is a binary value of 0 or 1.
4. Let us create a logistic regression model between the columns 'am' and 3 other columns – hp, wt and cyl.
5. We use the glm() function in the R language for logistic regression.
6. Set glm(formula = am ~ cyl + hp + wt, data = input, family = binomial) to a variable.
7. Display the result of the method.

## CODE

```
data(mtcars)
input <- mtcars[,c("am", "cyl", "hp", "wt")]
print(head(input))
```

### OUTPUT:

```
                   am cyl  hp    wt
Mazda RX4           1   6 110 2.620
Mazda RX4 Wag       1   6 110 2.875
Datsun 710          1   4  93 2.320
Hornet 4 Drive      0   6 110 3.215
Hornet Sportabout   0   8 175 3.440
Valiant             0   6 105 3.460
```

```
am.data = glm(formula = am ~ cyl + hp + wt, data = input, family =
binomial)
print(summary(am.data))
```

### OUTPUT:

```
Call:
glm(formula = am ~ cyl + hp + wt, family = binomial, data = input)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-2.17272  -0.14907  -0.01464   0.14116   1.27641

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) 19.70288    8.11637   2.428   0.0152 *
```

```
cyl          0.48760    1.07162   0.455   0.6491
hp           0.03259    0.01886   1.728   0.0840 .
wt          -9.14947    4.15332  -2.203   0.0276 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 43.2297  on 31  degrees of freedom
Residual deviance:  9.8415  on 28  degrees of freedom
AIC: 17.841

Number of Fisher Scoring iterations: 8
```

## CONCLUSION

In the summary as the p-value in the last column is more than 0.05 for the variables "cyl" and "hp", hence, we consider them to be insignificant in contributing to the value of the variable "am". Thus, only weight (wt) impacts the "am" value in this regression model.

## RESULT

The logistic regression algorithm was implemented on the *mtcars* dataset.

# IMPLEMENTATION OF DECISION TREE ALGORITHM USING CTREE

## AIM:

To implement a Decision tree algorithm using CTREE in R.

## PROCEDURE:

1. Import necessary libraries and load the dataset

| Install: | Import: |
|---|---|
| install.packages('datasets')<br>install.packages('caTools')<br>install.packages('party')<br>install.packages('dplyr')<br>install.packages('magrittr') | library(datasets)<br>library(caTools)<br>library(party)<br>library(dplyr)<br>library(magrittr) |

2. Load and Print the first few data from the dataset

```
data("readingSkills")
head(readingSkills)
```

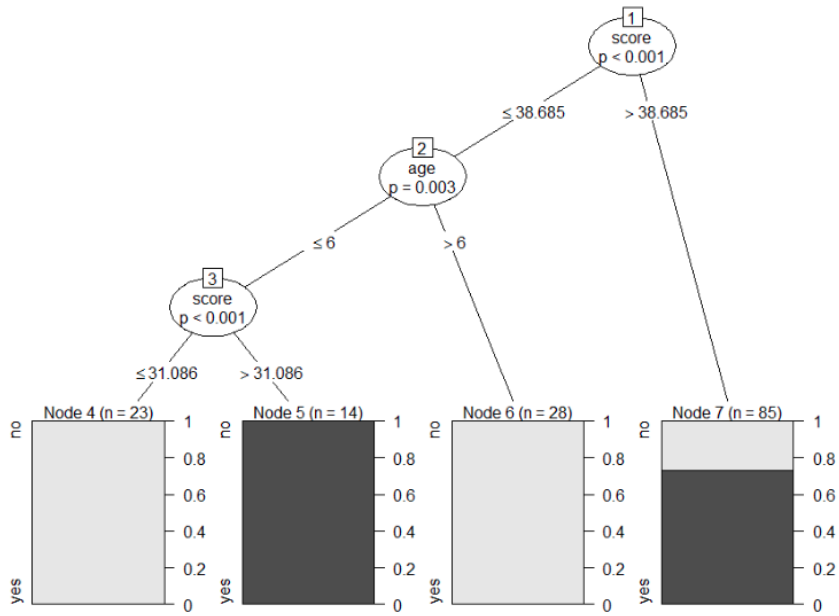
```
> data("readingSkills")
> head(readingSkills)
  nativeSpeaker age shoeSize    score
1           yes   5 24.83189 32.29385
2           yes   6 25.95238 36.63105
3            no  11 30.42170 49.60593
4           yes   7 28.66450 40.28456
5           yes  11 31.88207 55.46085
6           yes  10 30.07843 52.83124
```

3. Split the dataset

```
sample_data = sample.split(readingSkills, SplitRatio = 0.8)
train_data <- subset(readingSkills, sample_data == TRUE)
test_data <- subset(readingSkills, sample_data == FALSE)
```

4. Visualize the model using Ctree

```
model<- ctree(nativeSpeaker ~ ., train_data)
plot(model)
```




5. Predict the value

```
> predict_model<-predict(ctree_,test_data)
> predict_model<-predict(ctree_,test_data)
> m_at <- table(test_data$nativeSpeaker,predict_model)
> m_at
     predict_model
      no yes
  no  13  13
  yes  0  24
```

6. Print the accuracy of the model found

```
> ac_Test <- sum(diag(table_mat)) / sum(table_mat)
> print(paste('Accuracy for test is found to be', ac_Test))
[1] "Accuracy for test is found to be 0.74"
>
```

Here the accuracy-test from the confusion matrix is calculated and is found to be 0.74. Hence this model is found to predict with an accuracy of 74 %.

**RESULT:** Implementation of decision tree using ctree done successfully.

# SENTIMENTAL ANALYSIS ON SPAM TEXT FILE

## AIM:

To perform text analysis and sentiment analysis on the spam text file.

**PROCEDURE:**

1. **Installing and loading R packages**

```
install.packages("tm")  # for text mining
install.packages("SnowballC") # for text stemming
install.packages("wordcloud") # word-cloud generator
install.packages("RColorBrewer") # color palettes
install.packages("syuzhet") # for sentiment analysis
install.packages("ggplot2") # for plotting graphs
# Load
library("tm")
library("SnowballC")
library("wordcloud")
library("RColorBrewer")
library("syuzhet")
library("ggplot2")
```

2. **Reading file data into R**

```
text <- readLines(file.choose())
```

3. **Load the data as a corpus**

```
TextDoc <- Corpus(VectorSource(text))
```

4. **Clean the corpus in the following steps:-**

- Replace "/", "@" and "|" with space
- Convert the text to lower case
- Remove numbers
- Remove english common stopwords
- Remove our own stop words by specifying our custom stopwords as a character vector

```
#Replacing "/", "@" and "|" with space
toSpace <- content_transformer(function (x , pattern ) gsub(pattern, " ", x))
TextDoc <- tm_map(TextDoc, toSpace, "/")
# Convert the text to lower case
TextDoc <- tm_map(TextDoc, content_transformer(tolower))
# Remove numbers
TextDoc <- tm_map(TextDoc, removeNumbers)
# Remove english common stopwords
```

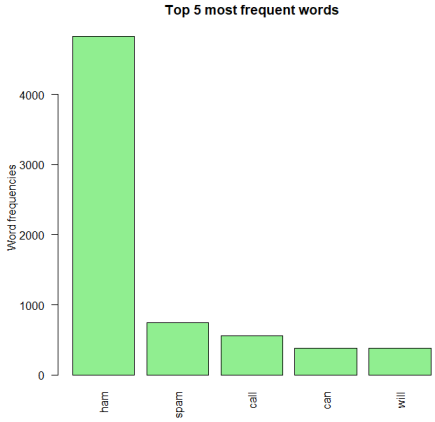
```
TextDoc <- tm_map(TextDoc, removeWords, stopwords("english"))
# Remove your own stop word
# specify your custom stopwords as a character vector
TextDoc <- tm_map(TextDoc, removeWords, c("s", "company", "team"))
```

5. Create a term document matrix to view the frequency of each word in the spam file and display the top 5 most frequent words.

```
TextDoc_dtm <- TermDocumentMatrix(TextDoc)
dtm_m <- as.matrix(TextDoc_dtm)
```

```
> dtm_m <- as.matrix(TextDoc_dtm)
> dtm_v <- sort(rowSums(dtm_m),decreasing=TRUE)
> dtm_d <- data.frame(word = names(dtm_v),freq=dtm_v)
> head(dtm_d, 5)
     word freq
ham   ham 4826
spam spam  748
call call  554
can   can  382
will will  378
```

6. Create a bar plot to represent the frequency of the 5 most frequently occurring words.
barplot(dtm_d[1:5,]$freq, las = 2, names.arg = dtm_d[1:5,]$word, col ="lightgreen", main ="Top 5 most frequent words", ylab = "Word frequencies")




7. **Generate word cloud**
set.seed(1234)
wordcloud(words = dtm_d$word, freq = dtm_d$freq, min.freq = 5, max.words=100, random.order=FALSE, rot.per=0.40, colors=brewer.pal(8, "Dark2"))

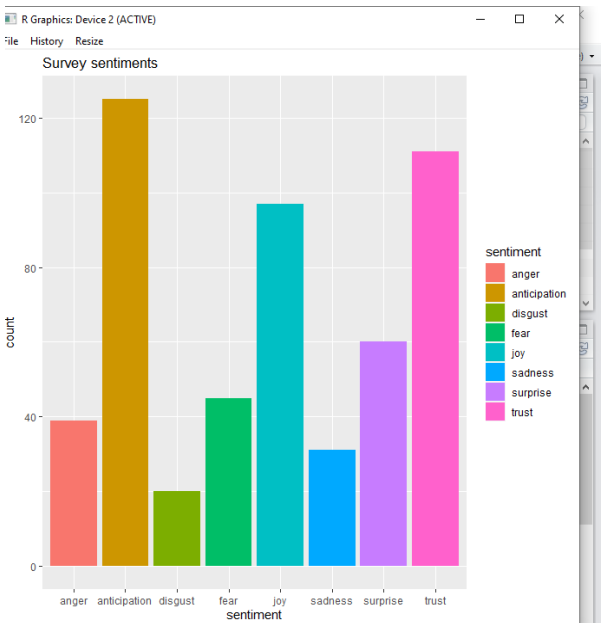
8. **Perform a sentiment analysis using the get_nrc_sentiment() method.**

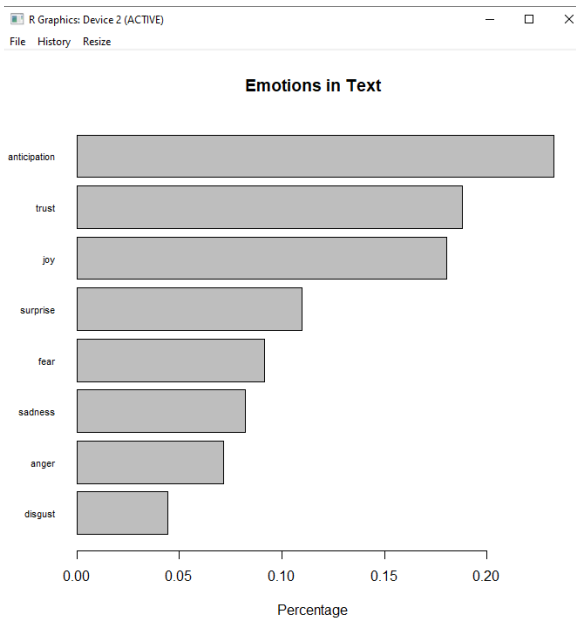
```
# regular sentiment score using get_sentiment() function and method of your choice
# please note that different methods may have different scales
syuzhet_vector <- get_sentiment(text, method="syuzhet")
# see the first row of the vector
head(syuzhet_vector)
# see summary statistics of the vector
summary(syuzhet_vector)
d<-get_nrc_sentiment(text)
# head(d,10) - to see top 10 lines of the get_nrc_sentiment dataframe
head (d,10)
#transpose
td<-data.frame(t(d))
#The function rowSums computes column sums across rows for each level of a grouping variable.
td_new <- data.frame(rowSums(td[2:253]))
#Transformation and cleaning
names(td_new)[1] <- "count"
td_new <- cbind("sentiment" = rownames(td_new), td_new)
rownames(td_new) <- NULL
td_new2<-td_new[1:8,]
```

```
Console ~/
> head(syuzhet_vector)
[1] 0.00 0.55 0.60 1.00 0.00 0.00
> summary(syuzhet_vector)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-4.5000  0.0000  0.0000  0.4044  0.8000  9.0500
> d<-get_nrc_sentiment(text)
> head (d,10)
   anger anticipation disgust fear joy sadness surprise trust negative positive
1      0            0       0    0   0       0        0     0        0        0
2      2            0       0    1   0       1        0     0        2        0
3      0            0       0    0   0       0        0     0        0        1
4      0            0       0    0   0       0        0     0        0        1
5      0            0       0    0   0       0        0     0        1        0
6      0            0       0    0   0       0        0     0        0        0
7      0            1       0    0   2       0        0     2        0        3
8      1            1       1    1   1       1        1     2        1        3
9      0            0       0    0   0       0        0     0        1        0
10     0            3       0    0   2       0        2     1        0        3
```

**9.** Generate a bar plot with the sentiment on the x-axis and the frequency on the y-axis.

```
#Plot One - count of words associated with each sentiment
quickplot(sentiment, data=td_new2, weight=count, geom="bar", fill=sentiment,
ylab="count")+ggtitle("Survey sentiments")

barplot(
  sort(colSums(prop.table(d[, 1:8]))),
  horiz = TRUE,
  cex.names = 0.7,
  las = 1,
  main = "Emotions in Text", xlab="Percentage"
)
```




**RESULT:**

Text analysis and sentiment analysis were successfully performed on the spam text file.

# SENTIMENTAL ANALYSIS USING NAIVE BAYES

**AIM:**

To perform sentiment analysis using naive bayes.

**PROCEDURE:**

**Loading libraries**

```
library(RTextTools)
library(e1071)
library(caret)
```

**Creating sample data**

```
sentPositive = c(
  "I like it", "like it a lot", "It's really good",
  "recommend!", "Enjoyed!", "like it","like)
sentNegative = c(
  "Not good at all!", "rude",
  "It is rude", "I don't like this type",
  "poor", "Boring", "Not good!", )
df = data.frame(sentiment = "positive", text = sentPositive)
df = rbind(df, data.frame(sentiment = "negative", text = sentNegative))
index = sample(1:nrow(df), size = .9 * nrow(df))
```

**#split the df data into the train and test parts.**

```
train = df[index, ]
test = df[-index, ]
```

```
> df = data.frame(sentiment = "positive", text = sentPositive)
> df = rbind(df, data.frame(sentiment = "negative", text = sentNegative))
>
> index = sample(1:nrow(df), size = .9 * nrow(df))
> train = df[index, ]
> test = df[-index, ]
>
> head(train)
   sentiment            text
21  positive          it's ok
9   positive      outstanding
36  negative             poor
10  positive             good
7   positive It's really good
13  positive      really good
>
> head(test)
   sentiment                          text
12  positive                 like it a lot
16  positive                      liked!
20  positive                    Very good
31  positive                      awesome
42  negative                 not satisfied
47  negative feel tired after watching this
```

**Preparing document matrix**

**# create matrix data from the text of a train and test data with a crete_matrix function of the RTextTool package. The RTextTool is a package for text classification. A create_matrix() creates a document-term matrix.**

```
mTrain = create_matrix(train[,2], language="english",
                       removeStopwords=FALSE, removeNumbers=TRUE,
                       stemWords=FALSE)
matTrain = as.matrix(mTrain)
mTest = create_matrix(test[,2], language="english",
                       removeStopwords=FALSE, removeNumbers=TRUE,
```

stemWords=FALSE)
matTest = as.matrix(mTest)
print(matTest)

```
           Terms
Docs        bit enjoyed good not poor show strange very
  Enjoyed!    0       1    0   0    0    0      0    0
  Very good   0       0    1   0    0    0      0    1
  it's ok     0       0    0   0    0    0      0    0
  Not good!   0       0    1   1    0    0      0    0
  poor show   0       0    0   0    1    1      0    0
  a bit strange 1     0    0   0    0    0      1    0
```

**Defining the model**

labelTrain = as.factor(train[,1])
labelTest = as.factor(test[,1])
model = naiveBayes(matTrain, labelTrain)
pred = predict(model, matTrain)

**#Predict as positive or negative and evaluate the fitted model**

confusionMatrix(labelTrain, pred)
pred = predict(model, matTest);
data.frame(test,pred)

```
> data.frame(test,pred)
   sentiment                          text     pred
12  positive                 like it a lot positive
16  positive                        liked! negative
20  positive                     Very good negative
31  positive                       awesome negative
42  negative                 not satisfied negative
47  negative feel tired after watching this negative
```

confusionMatrix(labelTest, pred)

**Sensitivity = True positives / True positives + False negatives**
**Specificity = True negatives / True negatives + False positives**
**Accuracy = true positive + true negative / TP + TN + FP +FN**

```
Confusion Matrix and Statistics

          Reference
Prediction negative positive
  negative       20        0
  positive       18        9

               Accuracy : 0.617
                 95% CI : (0.4638, 0.7549)
    No Information Rate : 0.8085
    P-Value [Acc > NIR] : 0.9994

                  Kappa : 0.2985

 Mcnemar's Test P-Value : 6.151e-05

            Sensitivity : 0.5263
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 0.3333
             Prevalence : 0.8085
         Detection Rate : 0.4255
   Detection Prevalence : 0.4255
      Balanced Accuracy : 0.7632

       'Positive' Class : negative
```

**RESULT:**

Thus sentiment analysis using naive Bayes was successfully performed.

**Integration of Hadoop and R programming Language – Case study**

**AIM**

To study different ways of integration of Hadoop and R.

**What is R and Hadoop?**

- R is an open-source programming language.
- It is best suitable for statistical and graphical analysis.
- As we are in need of strong data analytics and visualization features, then we need to combine R with Hadoop.
- Hadoop is an open-source tool that is founded by the **ASF – Apache Software Foundation**.
- It provides an efficient framework for running jobs.

**THE PURPOSE BEHIND R AND HADOOP INTEGRATION**

R is one of the most preferred programming languages for statistical computing and data analysis.
But without additional packages, it lacks a bit in terms of memory management and handling large data.

On the other hand, Hadoop is a powerful tool to process and analyze large amounts of data with its distributed file system HDFS and the map-reduce processing approach.
By integrating these two technologies, R's statistical computing power can be combined with efficient distributed computing. This means that we can:

- Use Hadoop to execute the R codes.
- Use R to access the data stored in Hadoop.

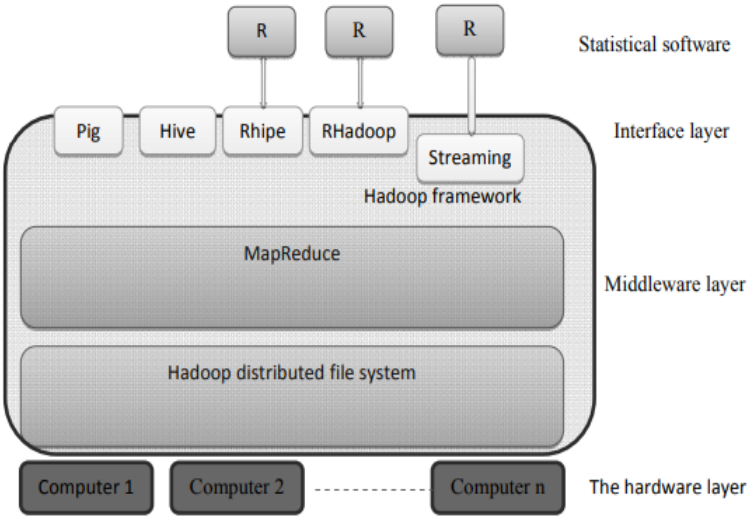



**Figure 1 Integration between R and Hadoop**

**R AND HADOOP INTEGRATION METHODS**

There are different types of methods for integrating R programming with Hadoop:

1. **RHADOOP – Install R on Workstations and Connect to Data in Hadoop**

The most commonly used open-source analytics solution to integrate R programming language with Hadoop is RHadoop. RHadoop developed by Revolution Analytics lets users directly ingest data from HBase database subsystems and HDFS file systems. RHadoop package is the 'go-to' solution for using R on Hadoop because of its simplicity and cost advantage.

The R Hadoop method is a collection of 3 packages.

- **Rhdfs**: This is an R package for providing all Hadoop HDFS access to R. All distributed files can be managed with R functions.
- **Rmr**: This is an R package for providing Hadoop MapReduce interfaces to R. With the help of this package, the Mapper and Reducer can easily be developed.
- **Rhbase**: This is an R package for handling data at HBase distributed database through R.

A script that utilizes the RHadoop functionality is as follows.

```
library(rmr)
map<-function(k,v){...}
reduce<-function(k,vv){...}
mapreduce(input = "data.txt", output ="output",
          textinputformat = rawtextinputformat,
          map = map, reduce = reduce)
```

**2. Hadoop Streaming**

- It's R database management capability with integration with HBase.
- Hadoop streaming is the R Script available as part of the R package on CRAN. Also, this intends to make R more accessible to Hadoop streaming applications.
- It involves writing MapReduce codes in R language, which makes it extremely user-friendly.
- Java is the native language for MapReduce but according to today's need, it doesn't suit high-speed data analysis. Thus, in today's time, we need faster mapping and reducing steps with Hadoop.
- Hadoop streaming has gained huge demand as we can write the codes in Python, Perl or even Ruby

  A command line with map and reduce tasks implemented as R scripts would look like the following.

```
$ ${HADOOP_HOME}/bin/Hadoop jar
$ {HADOOP_HOME}/contrib/streaming/*.jar\
-inputformat org.apache.hadoop.mapred.TextInputFormat \
-input input_data.txt \
-output output \
-mapper /home/tst/src/map.R \
-reducer /home/tst/src/reduce.R \
-file /home/ts/src/map.R \
-file /home/tst/src/reduce.R
```

**3. RHIPE**

- RHIPE stands for **R and Hadoop Integrated Programming Environment**.
- Divide and recombine developed this integrated programming environment for carrying out an efficient analysis of a large amount of data.

- It involves working with R and Hadoop integrated programming environment. Also, one can use Python, Java or Perl to read data sets in RHIPE.
- There are various functions in RHIPE that lets you interact with HDFS. Hence, this way you can read, save the complete data that is created using RHIPE MapReduce.
- The general structure of the **R script** that uses **RHIPE** is shown below as follows.

```
library(Rhipe)
rhint(TRUE, TRUE);
map<-expression({lapply(map.values, function(mapper)...)})
reduce<-expression(
pre = {...},
reduce = {...},
post = {...}, }
x <- rhmr(
map = map,reduce = reduce,
ifolder = inputPath,
ofolder= outputPath,
inout=c('text','text'),
jobname= 'a job name'))
rhex(z)
```

**4. ORCH**

- It is called Oracle R Connector. It can be used to particularly work with Big Data in Oracle appliance and also, on a non-Oracle framework like Hadoop.
- ORCH helps in accessing the Hadoop cluster via R and also to write the mapping and reducing functions. Also, one can manipulate the data residing in the Hadoop Distributed File System.

**5. IBM's BigR**

- IBM's BigR provides end-to-end integration between IBM's Hadoop package – BigInsights and R.
- BigR enables users to focus on the R program to analyze the data stored in the HDFS instead of MapReduce jobs.
- The combination of the BigInsights and the BigR technologies provides parallel execution of R code across the Hadoop cluster.

**Challenges and Limitations**

- The integration with R from the client side part is high for Rhipe and Rhadoop and is missing for R and Streaming.
- Rhipe and RHadoop allows users to define and call their own map and reduce functions within R while Streaming uses a command line approach where the Map and reduce functions are passed as arguments.
- Regarding the licensing scheme, all three approaches require GPL-2 and GPL-3 for R and Apache 2.0 for Hadoop, Streaming, Rhipe and RHadoop.
- Hadoop with R seems to be the most used approach for simple Map-Reduce jobs.
- For more complex jobs the solution should be Rhipe or RHadoop.

**RESULT**

Thus, different ways of integration of Hadoop and R were studied.

**Implementation of an application that stores big data in Hbase using Hadoop**

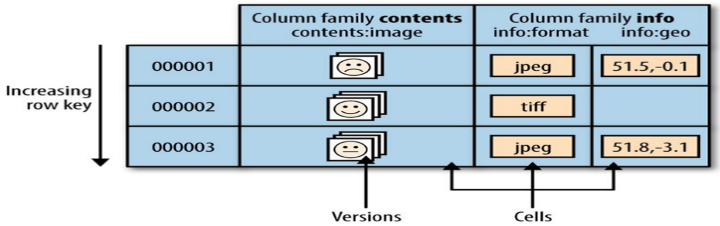
**Aim:**

To create a table and store the big data in Hbase using Hadoop.

**Procedure:**

**HBase Data Model**

- HBase stores data as a sparse, multidimensional, sorted map.
- The map is indexed by its key (the row key), and values are stored in cells (consisting of a column key and column value). The row key and column keys are strings and the column value is an uninterrupted byte array (which could represent any primitive or complex data type).
- HBase is multidimensional, as each cell is versioned with a time stamp.
- Row columns are grouped into column families. At table design time, more column families is defined. Column families will be used as physical storage groups for columns.
- A table's column families must be specified up front as part of the table schema definition, but new column family members can be added on demand.
- Physically, all column family members are stored together on the file system. HBase is a column-family-oriented store.



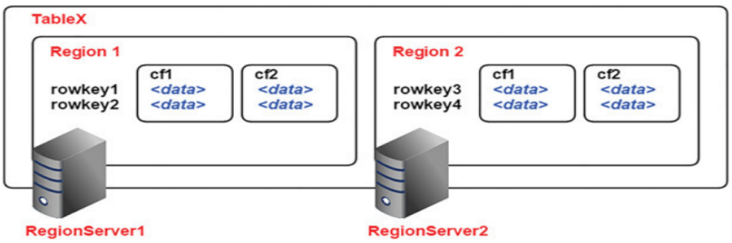

info:format and info:geo are both members of the info column family

contents:image belongs to the contents family.

A new column info:camera can be offered by a client as part of an update

**Regions**

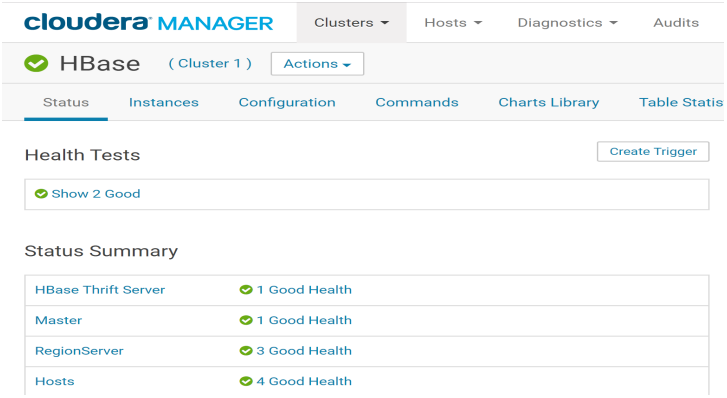
- Tables are automatically partitioned horizontally by HBase into regions. Each region comprises a subset of a table's rows, usually a range of sorted row keys.
- Initially, a table comprises a single region, but as the region grows it eventually crosses a configurable size threshold, at which point it splits at a row boundary into two new regions of approximately equal size. Until this first split happens, all loading will be against the single server hosting the original region.
- Regions are the units that get distributed over an HBase cluster.



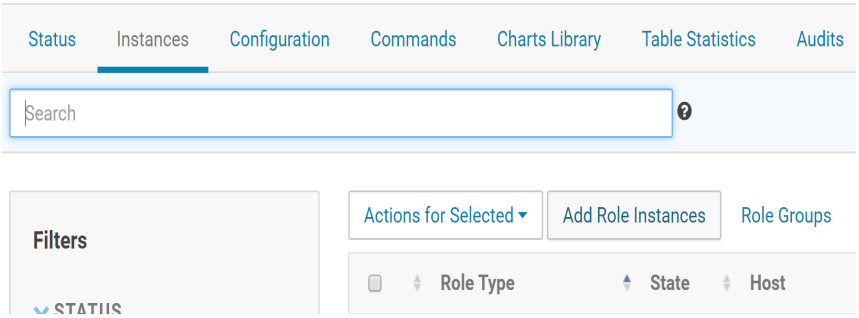

**HBase browser in HUE**

**Add HBase Thrift Server role**

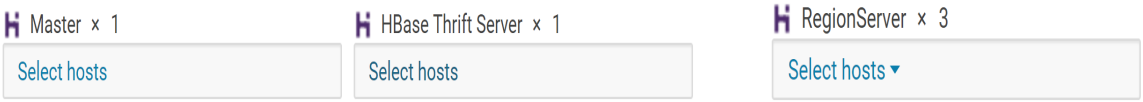
The Thrift Server role is not added by default when you install HBase, but it is required before you can use certain other features such as the Hue HBase browser.
Follow the below steps to add the Thrift Server role:
In Cloudera Manager, go to the HBase service:

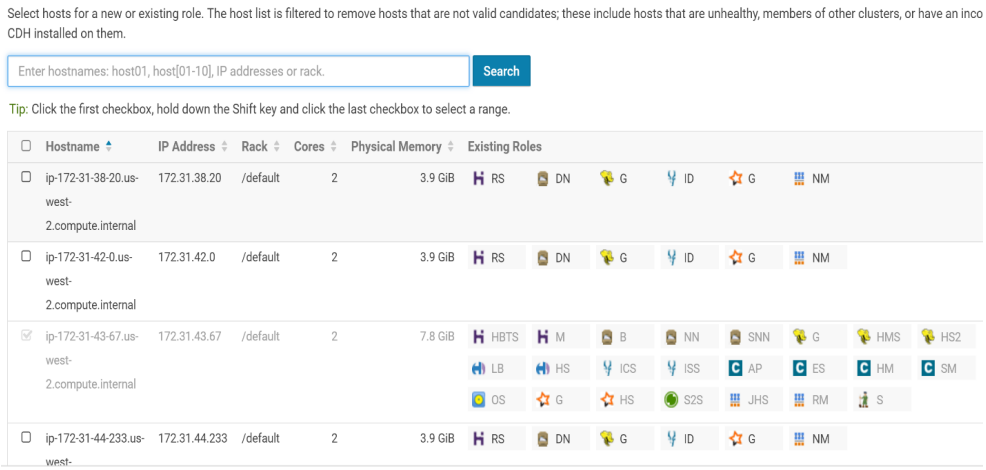



Click the Instances tab:

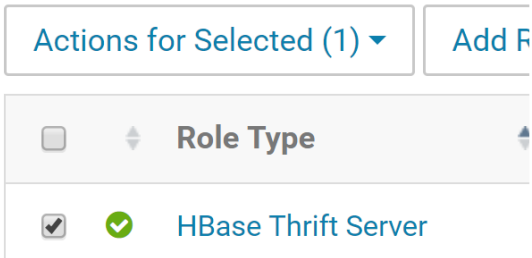



Click the Add Role Instances button.
Select the host(s) where you want to add the Thrift Server role (you only need one for Hue) and click Continue. The Thrift Server role should appear in the instances list for the HBase server.





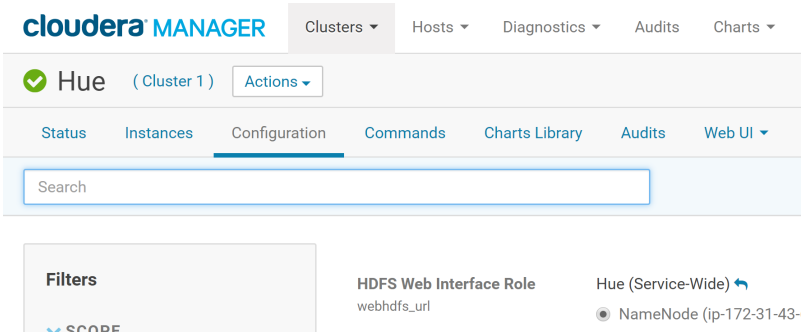

Select the Thrift Server role instance.




Select Actions for Selected > Start.

**Configure Hue to point to the Thrift Server**

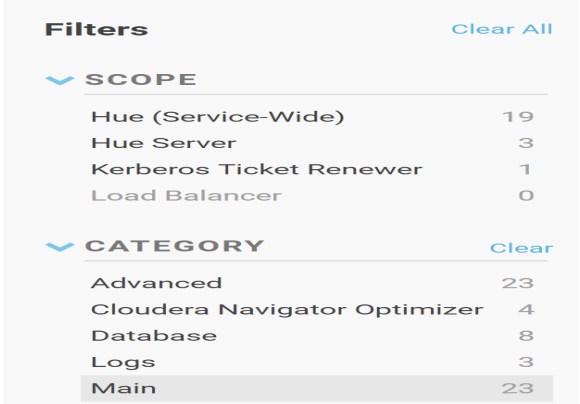
In Cloudera Manager, select the Hue service.



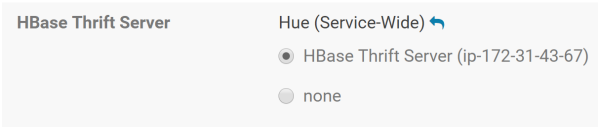

Click the Configuration tab.

Select Scope > All.

Select Category > Main.

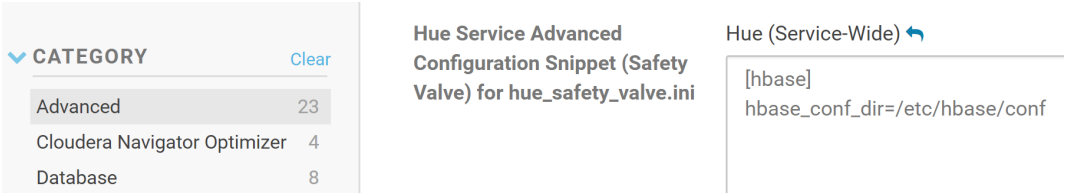



For the HBase Service property, make sure it is set to the HBase service for which you enabled the Thrift Server role (if you have more than one HBase service instance).
In the HBase Thrift Server property, click the edit field and select the Thrift Server role for Hue to use.



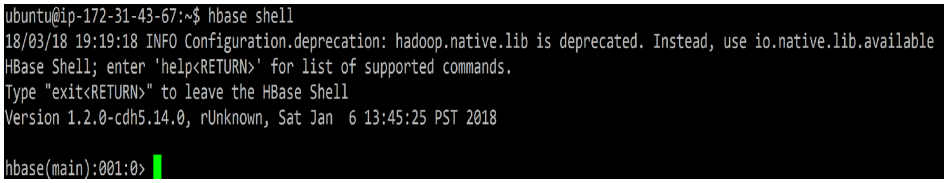

Select Category > Advanced.Locate the Hue Service Advanced Configuration Snippet (Safety Valve) for hue_safety_valve.ini property and add the following property:

```
[hbase]
hbase_conf_dir=/etc/hbase/conf
```




Click Save Changes to commit the changes.
**Demo HBase**Connect to your cluster using SSH and run the HBase shell:

```
hbase shell
```




Create a new HBase table with two column families (cf1 and cf2) using the following statement:

**create 'my-hbase-table', **
**{NAME => 'cf1' , VERSIONS => 3}, **
**{NAME => 'cf2'}**

VERSIONS => 4 means we want to keep 4 versions maximum of this column. The subsequent put statements insert data into a cell as defined by the row key (**rowkey1** or **rowkey2**, in this case) and a column specified in the format **<column_family>:<column_name>**.

Unlike a traditional database, the columns are not defined at table design time and are not typed.

**put 'my-hbase-table', 'rowkey1', 'cf1:fname', 'John'**
**put 'my-hbase-table', 'rowkey1', 'cf1:lname', 'Doe'**
**put 'my-hbase-table', 'rowkey2', 'cf1:fname', 'Jeffrey'**
**put 'my-hbase-table', 'rowkey2', 'cf1:lname', 'Aven'**
**put 'my-hbase-table', 'rowkey2', 'cf1:city', 'Hayward'**
**put 'my-hbase-table', 'rowkey2', 'cf2:photo', '<image>'**
**Enter the scan command to show the content of the table:**
**scan 'my-hbase-table'**

```
hbase(main):025:0> scan 'my-hbase-table'
ROW                                    COLUMN+CELL
 rowkey1                               column=cf1:fname, timestamp=1521401521350, value=John
 rowkey1                               column=cf1:lname, timestamp=1521401485900, value=Doe
 rowkey2                               column=cf1:city, timestamp=1521401485984, value=Hayward
 rowkey2                               column=cf1:fname, timestamp=1521401485945, value=Jeffrey
 rowkey2                               column=cf1:lname, timestamp=1521401485964, value=Aven
 rowkey2                               column=cf2:photo, timestamp=1521401494878, value=<image>
2 row(s) in 0.0970 seconds
```

The data inserted you see in the console can be conceptually viewed like this:

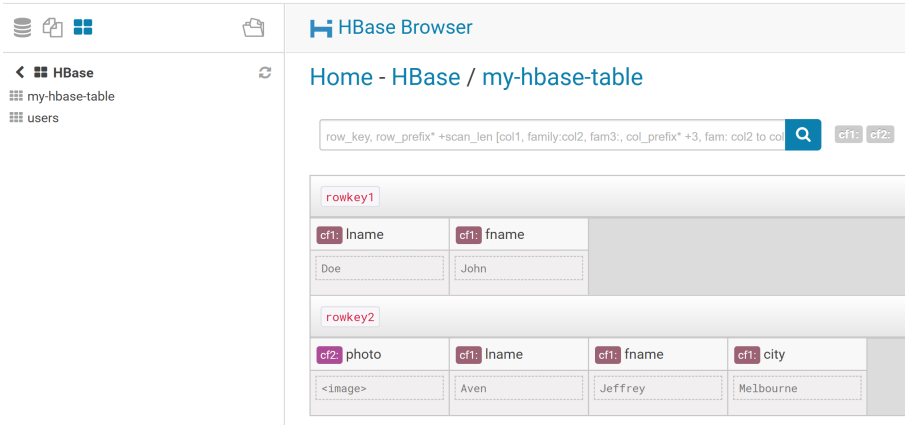
| Row Key | Column Family "cf1" | Column Family "cf2" |
|---|---|---|
| rowkey1 | fname: John, lname: Doe | |
| rowkey2 | fname: Jeffrey, lname: Aven, city: Hayward | photo: *<image>* |

HBase supports sparsity. Not every column needs to exist in each row in a table and nulls are not stored. Although HBase data is stored on HDFS, an immutable file system, HBase allows in-place updates to cells within HBase tables. It does this by creating a new version of the cell with a new time stamp if the column key already exists, and then a background compaction process collapses multiple files into a smaller number of larger files.

**Update a cellput 'my-hbase-table', 'rowkey2', 'cf1:city', 'Melbourne'**
**Get the latest version of a cell:get 'my-hbase-table', 'rowkey2', {COLUMNS => ['cf1:city']}**
**Get multiple versions of a cell:get 'my-hbase-table', 'rowkey2', {COLUMNS => ['cf1:city'], VERSIONS => 2}**

```
hbase(main):026:0> put 'my-hbase-table', 'rowkey2', 'cf1:city', 'Melbourne'
0 row(s) in 0.0120 seconds

hbase(main):027:0> get 'my-hbase-table', 'rowkey2', {COLUMNS => ['cf1:city']}
COLUMN                                 CELL
 cf1:city                              timestamp=1521401673336, value=Melbourne
1 row(s) in 0.0250 seconds

hbase(main):028:0> get 'my-hbase-table', 'rowkey2', {COLUMNS => ['cf1:city'], VERSIONS => 2}
COLUMN                                 CELL
 cf1:city                              timestamp=1521401673336, value=Melbourne
 cf1:city                              timestamp=1521401485984, value=Hayward
2 row(s) in 0.0080 seconds
```

Log in to HUE. The HBase browser should now be available in the main menu.
We can now browse across HBase tables using in HUE:




To drop the table in HBase, type-in the following command in the HBase shell (NOTE: you need to disable a table prior to dropping the table in HBase):

**disable 'my-hbase-table'drop 'my-hbase-table'Load a file in HDFS into HBase using Pig**

Suppose we have a file in HDFS we want to load into a HBase table. We can easily do that using Pig. This can be done in two steps:

1. Create the table in HBase.
2. Run the Pig script to load the file into the newly created table.

Let's consider we want to move the file **u.user** located in **/user/ubuntu/ml-100k/** to HBase. This file has 5 columns and is pipe (|) delimited.

In the Terminal, enter the HBase shell:

**hbase shell**

**Then create a new table users with one column family userinfo using the following command:**

**create 'users', 'userinfo'**

```
ubuntu@ip-172-31-43-67:~$ hbase shell
18/03/18 21:31:08 INFO Configuration.deprecation: hadoop.native.lib is deprecated.
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 1.2.0-cdh5.14.0, rUnknown, Sat Jan  6 13:45:25 PST 2018

hbase(main):001:0> create 'users', 'userinfo'
0 row(s) in 1.4790 seconds

=> Hbase::Table - users
hbase(main):002:0> list
TABLE
my-hbase-table
users
2 row(s) in 0.0130 seconds

=> ["my-hbase-table", "users"]
hbase(main):003:0>
```

**Exit the HBase shellexit**

**and run the Pig shell/Grunt:**

**pig**In the Pig shell, run the following command to load the file:

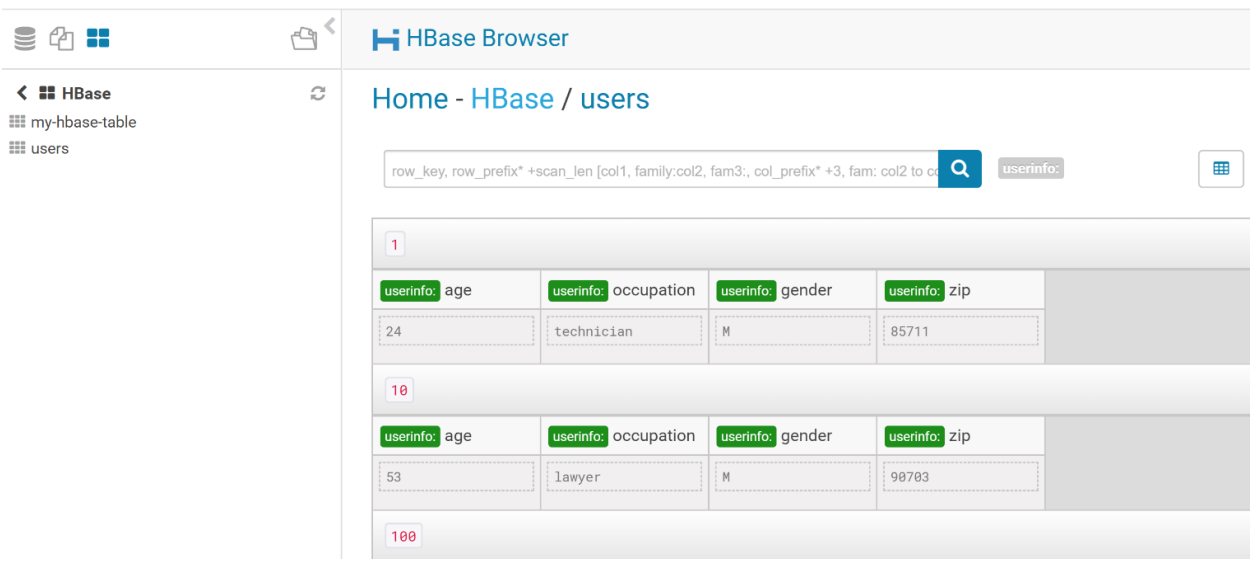
**users = LOAD '/user/ubuntu/ml-100k/u.user'**

**USING PigStorage('|') AS (userID:int, age:int, gender:chararray, occupation:chararray, zip:int);**

**Then we can store the relation in the HBase table. By default, the first column of therelation will be the row key of the HBase table.STORE users INTO 'hbase://users' USING org.apache.pig.backend.hadoop.hbase.HBaseStorage ( 'userinfo:age,userinfo:gender,userinfo:occupation,userinfo:zip');**

```
Success!

Job Stats (time in seconds):
JobId   Maps    Reduces MaxMapTime      MinMapTIme      AvgMapTime      MedianMapTime   MaxReduceTime   MinReduceTime
ime     MedianReducetime        Alias   Feature Outputs
job_1521397123245_0001  1       0       4       4       4       4       n/a     n/a     n/a     n/a     users   MAP_ONLY
ase://users,

Input(s):
Successfully read 943 records (23028 bytes) from: "/user/ubuntu/ml-100k/u.user"

Output(s):
Successfully stored 943 records in: "hbase://users"

Counters:
Total records written : 943
Total bytes written : 0
Spillable Memory Manager spill count : 0
Total bags proactively spilled: 0
Total records proactively spilled: 0
```

We can now browse the table in HUE using the HBase browser:




**Result:**

Thus the HBASE table is created and implemented successfully in Hadoop.

# IMPLEMENTATION OF SUPPORT VECTOR MACHINE (SVM) ON SALARY DATASET

**AIM:**

To implement support vector machine (SVM) on salary dataset.

**PROCEDURE:**

1. importing libraries,import numpy as nm,import matplotlib.pyplot as mtp
   import pandas as pd
2. Importing datasets
   data_set= pd.read_csv('User_Data.csv')
   #Extracting Independent and dependent Variable
   x= data_set.iloc[:, [2,3]].values
   y= data_set.iloc[:, 4].values
3. Splitting the dataset into training 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= 0.25, random_state=0)
   #feature Scalingfrom sklearn.preprocessing import StandardScaler
   st_x= StandardScaler()
   x_train= st_x.fit_transform(x_train)
   x_test= st_x.transform(x_test)
4. Support vector classifier
   from sklearn.svm import SVC
   classifier = SVC(kernel='linear', random_state=0)
   classifier.fit(x_train, y_train)

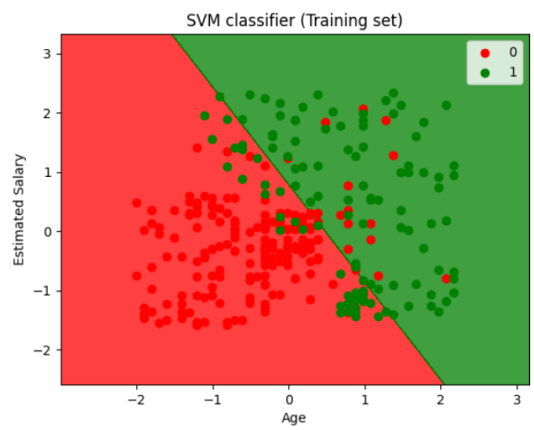
   ```
   SVC
   SVC(kernel='linear', random_state=0)
   ```

5. Predicting the test set result
   y_pred= classifier.predict(x_test)
6. Creating the Confusion matrix
   from sklearn.metrics import confusion_matrix
   cm= confusion_matrix(y_test, y_pred)
   Confusion matrix values

| | 0 | 1 |
|---|---|---|
| 0 | 66 | 2 |
| 1 | 8 | 24 |

7. Training set of SVM classifier
   from matplotlib.colors import ListedColormap

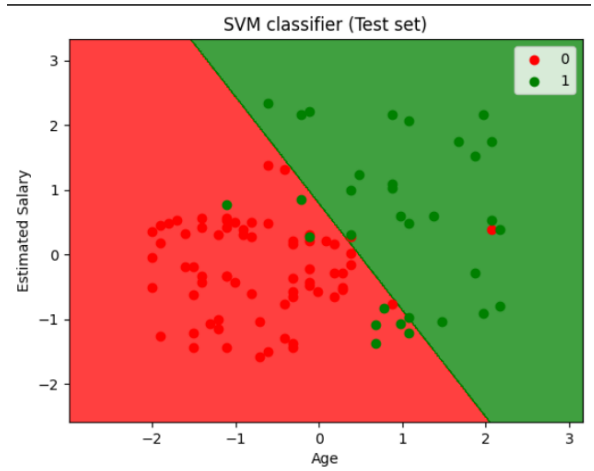
```
x_set, y_set = x_train, y_train
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() +
1, step  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(),
x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('red', 'green')))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('red', 'green'))(i), label = j)
mtp.title('SVM classifier (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```




8. Test set of SVM classifier

```
#Visulaizing the test set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() +
1, step  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step =
0.01))mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(),
x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('red','green' )))
mtp.xlim(x1.min(), x1.max())mtp.ylim(x2.min(), x2.max())
```

```
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('red', 'green'))(i), label = j)
mtp.title('SVM classifier (Test set)')mtp.xlabel('Age')mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```




**RESULT:**

Thus support vector machine (SVM) on salary dataset was implemented successfully.

# IMPLEMENTATION OF DECISION TREE ALGORITHM USING rpart

**AIM:**

To implement decision tree using RPART

**STEPS:**

**#Installing libraries**

**install.packages('rpart')** - Rpart is a powerful machine learning library in R that is used for building classification and regression trees. This library implements recursive partitioning and is very easy to use.

**install.packages('caret')** - caret enables you to train different types of algorithms using a simple train function. This layer of abstraction provides a common interface to train models in R

**install.packages('rpart.plot') -** Plot 'rpart' models. Extends plot.rpart() and text.rpart() in the 'rpart' package.

**install.packages('rattle')** - `Rattle` is a popular GUI for data mining using `R`. It presents statistical and visual summaries of data, transforms data so that it can be readily modelled

**#Loading libraries**

library(rpart,quietly = TRUE)
library(caret,quietly = TRUE)
library(rpart.plot,quietly = TRUE)
library(rattle)

**# Read mushroom dataset**

A dataset containing a total of 8124 observations of different mushrooms was used to build a classification tree using the rpart package. The goal is to predict whether a given mushroom is edible or poisonous to humans. Each observation has 23 features including the mushroom class.

**mushrooms <- read.csv ("/mushrooms.csv")**
**str(mushrooms)**
**# number of rows with missing values**
**nrow(mushrooms) - sum(complete.cases(mushrooms))**
**# deleting redundant variable `veil.type`**
**mushrooms$veil.type <- NULL**

In order to have an undertanding for the importance of the 21 different features in identifying the right category (*edible* or *poisonous*) we are going to:

1. Create a table for each feature versus class type (*edible*,*poisonus*)
2. Report the number of columns where there is a zero in the table created in step 1
3. Reorder the list created in step 2 by the number of zeroes reported
4. Plot the sorted list from step 3

**table(mushrooms$class,mushrooms$odor)**

```
  a   c   f   l   m   n   p   s   y
## e 400   0   0 400   0 3408   0   0   0
```

```
##   p    0  192 2160    0   36  120  256  576  576
```

We can instantly spot that mushrooms with *odor* equals "c","f","m","p","s" and "y" are poisonous. On the other hand, all mushrooms with *almond* odor (400) are edible in this dataset.

```
number.perfect.splits <- apply(X=mushrooms[-1], MARGIN = 2, FUN = function(col){
t <- table(mushrooms$class,col)
```

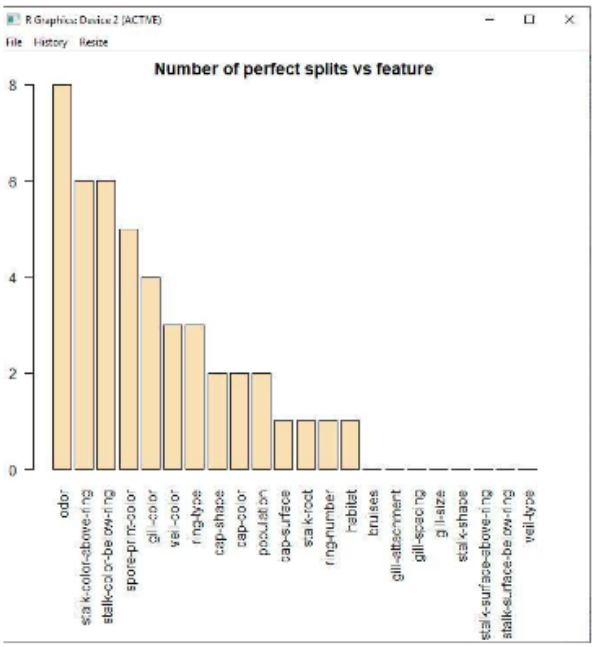
we will use how many perfect splits as a proxy for how good is a variable in splitting the data into classes edible and poisonous

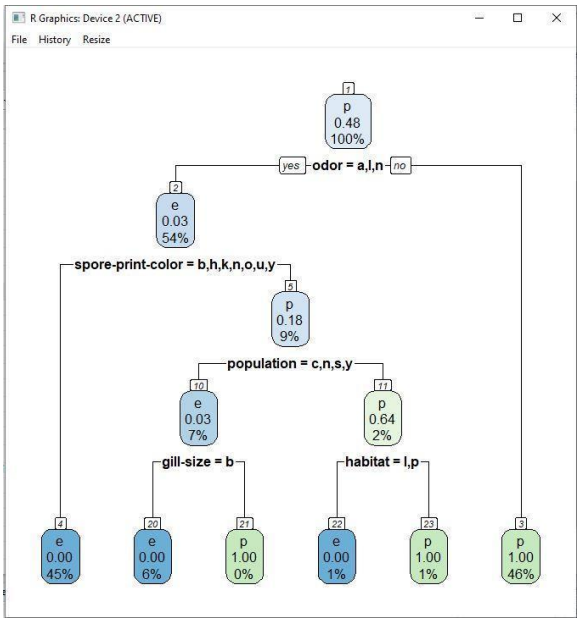
```
sum(t == 0)
})
# Descending order of perfect splits
order <- order(number.perfect.splits,decreasing = TRUE)
number.perfect.splits <- number.perfect.splits[order]
# Plot graph
par(mar=c(10,2,2,2))
barplot(number.perfect.splits,
main="Number of perfect splits vs feature",
xlab="",ylab="Feature",las=2,col="wheat")
#data splicing
set.seed(12345)
train <- sample(1:nrow(mushrooms),size = ceiling(0.80*nrow(mushrooms)),replace = FALSE)
# training set
mushrooms_train <- mushrooms[train,]
# test set
mushrooms_test <- mushrooms[-train,]
```

**# penalty matrix - peanalty matrix: will set a penalty which is 10 times bigger for classifying a poissoness mushroom as edible than classifying an edible mushroom as poisonness**

```
penalty.matrix <- matrix(c(0,1,10,0), byrow=TRUE, nrow=2)
# building the classification tree with rpart
tree <- rpart(class~.,
data=mushrooms_train,
parms = list(loss = penalty.matrix),
method = "class")
#choosing the best complexity parameter "cp" to prune the tree
cp.optim <- tree$cptable[which.min(tree$cptable[,"xerror"]),"CP"]
# tree prunning using the best complexity parameter.
tree <- prune(tree, cp=cp.optim)
# Visualize the decision tree with rpart.plot
rpart.plot(tree, nn=TRUE)
#Testing the model
pred <- predict(object=tree,mushrooms_test[-1],type="class")
#Calculating accuracy
t <- table(mushrooms_test$class,pred)
> confusionMatrix(t)
Confusion Matrix and Statistics
```

```
##    pred
##       e   p
##   e 839   0
##   p   0 785
##  Accuracy : 1
```




**RESULT:**

Implementation of decision tree using RPART done successfully.