Central University of South Bihar

DEPARTMENT OF STATISTICS

SCHOOL OF MATHEMATICS, STATISTICS AND COMPUTER SCIENCE

Master in Data Science and Applied Statistics

(MACHINE LEARNING-LAB-ASSIGNMENT)

SEMESTER – III
[SESSION: 2023-2025]

SUBMITTED TO	SUBMITTED BY
DR. SANDEEP KUMAR MAURYA	**SURESH KUMAR PRAJAPATI**
ASSISTANT PROFESSOR	En.No. CUSB2302222008

Lab 1 Cleaning house_price data

In [1]:
```
import  pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder,StandardScaler
```

In [2]:
```
df=pd.read_csv("House_Price.csv")
df.head()
```

Out[2]:

	price	crime_rate	resid_area	air_qual	room_num	age	dist1	dist2	dist3	dist4	teachers	poor_prop	airport	n_hos_beds	n_hot_rooms	waterbody	rainfall	bus_ter	parks
0	24.0	0.00632	32.31	0.538	6.575	65.2	4.35	3.81	4.18	4.01	24.7	4.98	YES	5.480	11.1920	River	23	YES	0.049347
1	21.6	0.02731	37.07	0.469	6.421	78.9	4.99	4.70	5.12	5.06	22.2	9.14	NO	7.332	12.1728	Lake	42	YES	0.046146
2	34.7	0.02729	37.07	0.469	7.185	61.1	5.03	4.86	5.01	4.97	22.2	4.03	NO	7.394	101.1200	NaN	38	YES	0.045764
3	33.4	0.03237	32.18	0.458	6.998	45.8	6.21	5.93	6.16	5.96	21.3	2.94	YES	9.268	11.2672	Lake	45	YES	0.047151
4	36.2	0.06905	32.18	0.458	7.147	54.2	6.16	5.86	6.37	5.86	21.3	5.33	NO	8.824	11.2896	Lake	55	YES	0.039474

In [3]:
```
df.shape
```

Out[3]:
```
(506, 19)
```

In [4]:
```
#To know about data describtion
df.describe()
```

Out[4]:

	price	crime_rate	resid_area	air_qual	room_num	age	dist1	dist2	dist3	dist4	teachers	poor_prop	n_hos_beds	n_hot_rooms	rainfall	parks
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	498.000000	506.000000	506.000000	506.000000
mean	22.528854	3.613524	41.136779	0.554695	6.284634	68.574901	3.971996	3.628775	3.960672	3.618972	21.544466	12.653063	7.899767	13.041605	39.181818	0.054454
std	9.182176	8.601545	6.860353	0.115878	0.702617	28.148861	2.108532	2.108580	2.119797	2.099203	2.164946	7.141062	1.476683	5.238957	12.513697	0.010632
min	5.000000	0.006320	30.460000	0.385000	3.561000	2.900000	1.130000	0.920000	1.150000	0.730000	18.000000	1.730000	5.268000	10.057600	3.000000	0.033292
25%	17.025000	0.082045	35.190000	0.449000	5.885500	45.025000	2.270000	1.940000	2.232500	1.940000	19.800000	6.950000	6.634500	11.189800	28.000000	0.046464
50%	21.200000	0.256510	39.690000	0.538000	6.208500	77.500000	3.385000	3.010000	3.375000	3.070000	20.950000	11.360000	7.999000	12.720000	39.000000	0.053507
75%	25.000000	3.677083	48.100000	0.624000	6.623500	94.075000	5.367500	4.992500	5.407500	4.985000	22.600000	16.955000	9.088000	14.170800	50.000000	0.061397
max	50.000000	88.976200	57.740000	0.871000	8.780000	100.000000	12.320000	11.930000	12.320000	11.940000	27.400000	37.970000	10.876000	101.120000	60.000000	0.086711

In [5]:
```
#To know how many data flot, object,int because it is convert in single variable
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 19 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   price        506 non-null    float64
 1   crime_rate   506 non-null    float64
 2   resid_area   506 non-null    float64
 3   air_qual     506 non-null    float64
 4   room_num     506 non-null    float64
 5   age          506 non-null    float64
 6   dist1        506 non-null    float64
 7   dist2        506 non-null    float64
 8   dist3        506 non-null    float64
 9   dist4        506 non-null    float64
 10  teachers     506 non-null    float64
 11  poor_prop    506 non-null    float64
 12  airport      506 non-null    object 
 13  n_hos_beds   498 non-null    float64
 14  n_hot_rooms  506 non-null    float64
 15  waterbody    351 non-null    object 
 16  rainfall     506 non-null    int64  
 17  bus_ter      506 non-null    object 
 18  parks        506 non-null    float64
dtypes: float64(15), int64(1), object(3)
memory usage: 75.2+ KB
```

In [6]:
```
# To remove the missing value and know the size of House_Price how many data is remove or not like row or columan.
df.isnull().sum()
df.shape
```

Out[6]:
```
(506, 19)
```

In [7]:
```
df.isnull().sum()
```

Out[7]:
```
price            0
crime_rate       0
resid_area       0
air_qual         0
room_num         0
age              0
dist1            0
dist2            0
dist3            0
dist4            0
teachers         0
poor_prop        0
airport          0
n_hos_beds       8
n_hot_rooms      0
waterbody      155
rainfall         0
bus_ter          0
parks            0
dtype: int64
```

In [8]:
```
# To know which type of data are present.
df['airport'].unique()
```

Out[8]:
```
array(['YES', 'NO'], dtype=object)
```

In [9]:
```
# To assign the value of yes as 1 & no as 0.

df['airport']=df['airport'].map({"YES":1,"NO":0})
df['airport']
```

Out[9]:
```
0      1
1      0
2      0
3      1
4      0
      ..
501    0
502    1
503    0
504    1
505    1
Name: airport, Length: 506, dtype: int64
```

In [10]:
```
# To know data type like int or float and yes or no etc.
df["airport"].unique()
```

Out[10]:
```
array([1, 0], dtype=int64)
```

In [11]:
```
df["waterbody"].unique()
```

Out[11]:
```
array(['River', 'Lake', nan, 'Lake and River'], dtype=object)
```

In [12]:
```
# To fill the missing value with NONE.
df["waterbody"].fillna("NONE",inplace=True)
```

In [13]:
```
#To assign the categorical data into numerical data.
df["waterbody"]=df["waterbody"].map({"NONE":0,"Lake":1,"River":2,"Lake and River":3})
df["waterbody"]
```

Out[13]:
```
0      2
1      1
2      0
3      1
4      1
      ..
501    3
502    3
503    0
504    0
505    0
Name: waterbody, Length: 506, dtype: int64
```

In [14]:
```
# Here result is assign
df["waterbody"].unique()
```

Out[14]:
```
array([2, 1, 0, 3], dtype=int64)
```

In [15]:
```
#  only bus_ter is as yes type of data
df["bus_ter"].unique()
```

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

In [16]:
```
#To remove  the bus_ter column which is not more effect the model
df.drop(columns=['bus_ter'], inplace=True)
```

In [17]:
```
# some missing value avaiable here so we remove some row which contains  missing value of data column.
df. dropna(subset=["n_hos_beds"],inplace=True)
```

In [18]:
```
# Here is no missing value available here.
df.isnull().sum()
```

Out[18]:
```
price          0
crime_rate     0
resid_area     0
air_qual       0
room_num       0
age            0
dist1          0
dist2          0
dist3          0
dist4          0
teachers       0
poor_prop      0
airport        0
n_hos_beds     0
n_hot_rooms    0
waterbody      0
rainfall       0
parks          0
dtype: int64
```

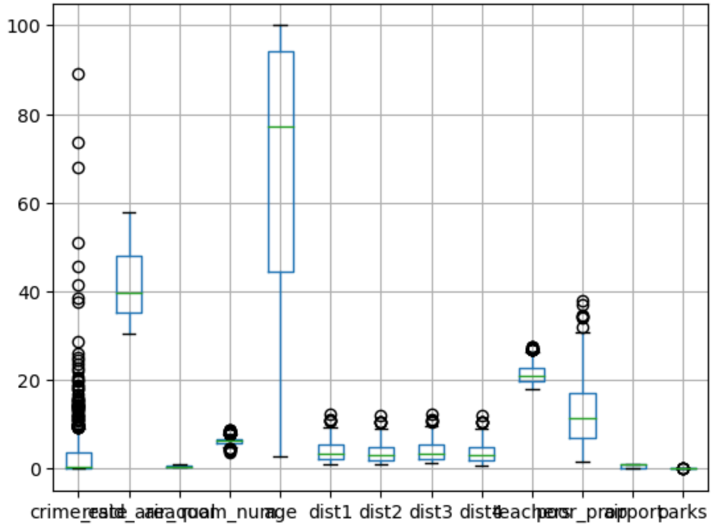
In [19]:
```
#To know how the correlated to each other,i.e.highly correlated or negative correlated.
round(df.corr(),2)
```

Out[19]:

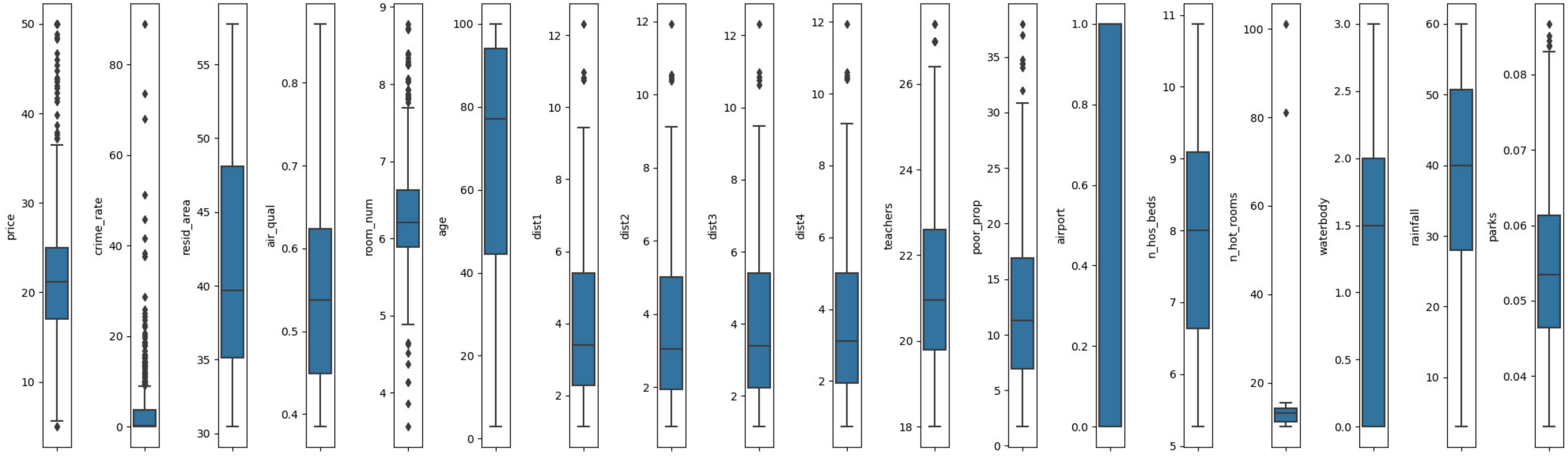
	price	crime_rate	resid_area	air_qual	room_num	age	dist1	dist2	dist3	dist4	teachers	poor_prop	airport	n_hos_beds	n_hot_rooms	waterbody	rainfall	parks
price	1.00	-0.38	-0.48	-0.43	0.70	-0.38	0.25	0.25	0.25	0.25	0.50	-0.74	0.19	0.11	0.02	0.05	-0.05	-0.39
crime_rate	-0.38	1.00	0.40	0.42	-0.22	0.35	-0.38	-0.38	-0.38	-0.38	-0.29	0.45	-0.09	0.02	0.01	-0.06	0.06	0.38
resid_area	-0.48	0.40	1.00	0.77	-0.39	0.65	-0.71	-0.71	-0.71	-0.71	-0.38	0.60	-0.11	0.01	-0.00	-0.05	0.05	0.71
air_qual	-0.43	0.42	0.77	1.00	-0.31	0.73	-0.77	-0.77	-0.77	-0.76	-0.19	0.59	-0.07	-0.05	-0.01	-0.04	0.10	0.91
room_num	0.70	-0.22	-0.39	-0.31	1.00	-0.24	0.21	0.21	0.20	0.21	0.35	-0.62	0.16	0.03	0.03	0.06	-0.06	-0.29
age	-0.38	0.35	0.65	0.73	-0.24	1.00	-0.75	-0.75	-0.75	-0.75	-0.26	0.60	0.00	-0.02	0.01	-0.09	0.08	0.67
dist1	0.25	-0.38	-0.71	-0.77	0.21	-0.75	1.00	1.00	1.00	0.99	0.23	-0.50	0.03	-0.03	-0.01	0.02	-0.04	-0.70
dist2	0.25	-0.38	-0.71	-0.77	0.21	-0.75	1.00	1.00	1.00	0.99	0.23	-0.50	0.02	-0.03	-0.01	0.02	-0.04	-0.71
dist3	0.25	-0.38	-0.71	-0.77	0.20	-0.75	1.00	1.00	1.00	0.99	0.23	-0.49	0.02	-0.03	-0.01	0.02	-0.04	-0.71
dist4	0.25	-0.38	-0.71	-0.76	0.21	-0.75	0.99	0.99	0.99	1.00	0.23	-0.50	0.02	-0.02	-0.00	0.03	-0.03	-0.70
teachers	0.50	-0.29	-0.38	-0.19	0.35	-0.26	0.23	0.23	0.23	0.23	1.00	-0.37	0.06	-0.01	-0.02	0.06	-0.04	-0.19
poor_prop	-0.74	0.45	0.60	0.59	-0.62	0.60	-0.50	-0.50	-0.49	-0.50	-0.37	1.00	-0.10	-0.07	0.00	-0.08	0.06	0.55
airport	0.19	-0.09	-0.11	-0.07	0.16	0.00	0.03	0.02	0.02	0.02	0.06	-0.10	1.00	-0.01	-0.07	-0.04	-0.01	-0.05
n_hos_beds	0.11	0.02	0.01	-0.05	0.03	-0.02	-0.03	-0.03	-0.03	-0.02	-0.01	-0.07	-0.01	1.00	-0.01	0.01	0.06	-0.07
n_hot_rooms	0.02	0.01	-0.00	-0.01	0.03	0.01	-0.01	-0.01	-0.01	-0.00	-0.02	0.00	-0.07	-0.01	1.00	-0.05	-0.01	0.01
waterbody	0.05	-0.06	-0.05	-0.04	0.06	-0.09	0.02	0.02	0.02	0.03	0.06	-0.08	-0.04	0.01	-0.05	1.00	0.07	-0.05
rainfall	-0.05	0.06	0.05	0.10	-0.06	0.08	-0.04	-0.04	-0.04	-0.03	-0.04	0.06	-0.01	0.06	-0.01	0.07	1.00	0.08
parks	-0.39	0.38	0.71	0.91	-0.29	0.67	-0.70	-0.71	-0.71	-0.70	-0.19	0.55	-0.05	-0.07	0.01	-0.05	0.08	1.00

In [20]:
```
# To draw the boxplot to know about the outlier
df.boxplot(column=["crime_rate","resid_area","air_qual","room_num","age","dist1","dist2","dist3","dist4","teachers","poor_prop","airport","parks"])
```

Out[20]:
```
<Axes: >
```



In [21]:
```
import seaborn as sns
import matplotlib.pyplot as plt

fig, axes = plt.subplots(nrows=1, ncols=df.shape[1], figsize=(20, 6))

# Create a separate box plot for each column
for i, column in enumerate(df.columns):
    sns.boxplot(data=df, y=column, ax=axes[i])
    axes[i].set_title(f'')

# Adjust layout
plt.tight_layout()
plt.show()
```



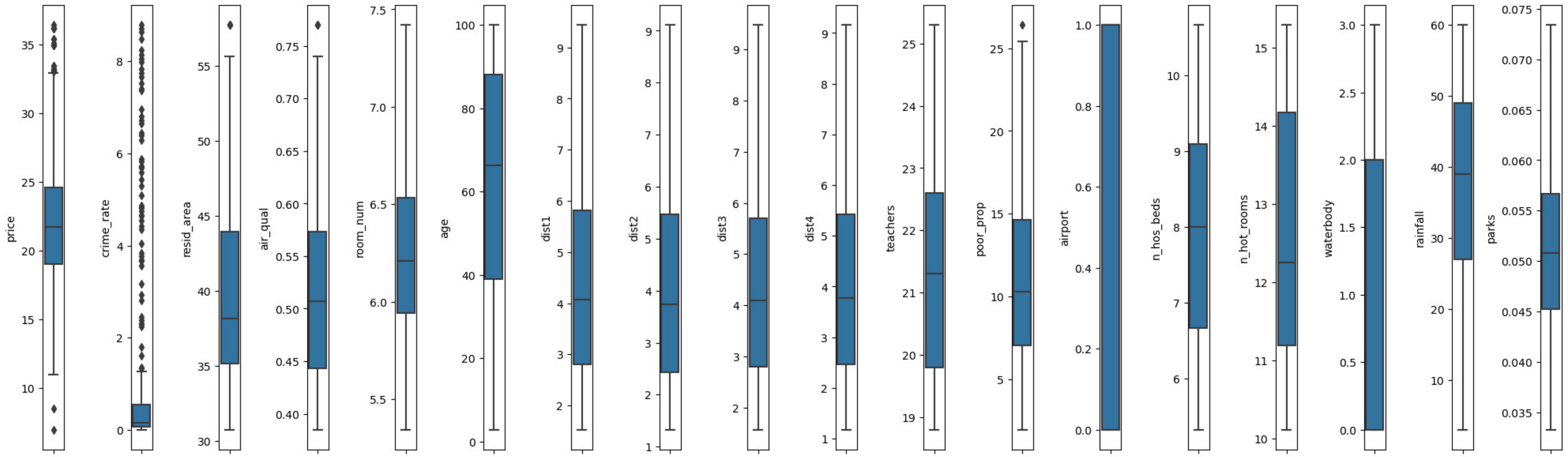
In [22]:
```python
def remove_outliers_iqr(df):
    # Create a copy of the DataFrame to avoid modifying the original
    cleaned_df = df.copy()

    # Iterate over each numeric column
    for column in cleaned_df.select_dtypes(include=[np.number]).columns:
        # Calculate Q1 (25th percentile) and Q3 (75th percentile)
        Q1 = cleaned_df[column].quantile(0.25)
        Q3 = cleaned_df[column].quantile(0.75)
        IQR = Q3 - Q1

        # Define bounds for outliers
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR

        # Filter out outliers
        cleaned_df = cleaned_df[(cleaned_df[column] >= lower_bound) & (cleaned_df[column] <= upper_bound)]

    return cleaned_df
```

In [23]:
```python
# Remove outliers from the dataset
cleaned_house_price_df = remove_outliers_iqr(df)
print(cleaned_house_price_df)
```

```
     price  crime_rate  resid_area  air_qual  room_num   age  dist1  dist2  \
0     24.0     0.00632       32.31     0.538     6.575  65.2   4.35   3.81   
1     21.6     0.02731       37.07     0.469     6.421  78.9   4.99   4.70   
3     33.4     0.03237       32.18     0.458     6.998  45.8   6.21   5.93   
4     36.2     0.06905       32.18     0.458     7.147  54.2   6.16   5.86   
5     28.7     0.02985       32.18     0.458     6.430  58.7   6.22   5.80   
..     ...         ...         ...       ...       ...   ...    ...    ...   
501   22.4     0.06263       41.93     0.573     6.593  69.1   2.64   2.45   
502   20.6     0.04527       41.93     0.573     6.120  76.7   2.44   2.11   
503   23.9     0.06076       41.93     0.573     6.976  91.0   2.34   2.06   
504   22.0     0.10959       41.93     0.573     6.794  89.3   2.54   2.31   
505   19.0     0.04741       41.93     0.573     6.030  80.8   2.72   2.24   

     dist3  dist4  teachers  poor_prop  airport  n_hos_beds  n_hot_rooms  \
0     4.18   4.01      24.7       4.98        1       5.480      11.1920   
1     5.12   5.06      22.2       9.14        0       7.332      12.1728   
3     6.16   5.96      21.3       2.94        1       9.268      11.2672   
4     6.37   5.86      21.3       5.33        0       8.824      11.2896   
5     6.23   5.99      21.3       5.21        1       7.174      14.2296   
..     ...    ...       ...        ...      ...         ...          ...   
501   2.76   2.06      19.0       9.67        0       9.348      12.1792   
502   2.46   2.14      19.0       9.08        1       6.612      13.1648   
503   2.29   1.98      19.0       5.64        0       5.478      12.1912   
504   2.40   2.31      19.0       6.48        1       7.940      15.1760   
505   2.64   2.42      19.0       7.88        1      10.280      10.1520   

     waterbody  rainfall     parks  
0            2        23  0.049347  
1            1        42  0.046146  
3            1        45  0.047151  
4            1        55  0.039474  
5            0        53  0.045910  
..         ...       ...       ...  
501          3        27  0.056006  
502          3        20  0.059903  
503          0        31  0.057572  
504          0        47  0.060694  
505          0        45  0.060336  

[346 rows x 18 columns]
```

In [24]:

```python
# After remove the plot show the result.
import seaborn as sns
import matplotlib.pyplot as plt

fig, axes = plt.subplots(nrows=1, ncols=cleaned_house_price_df.shape[1], figsize=(20, 6))

# Create a separate box plot for each column
for i, column in enumerate(cleaned_house_price_df.columns):
    sns.boxplot(data=cleaned_house_price_df, y=column, ax=axes[i])
    axes[i].set_title(f'')

# Adjust layout
plt.tight_layout()
plt.show()
```



In [48]:
```
Corln=cleaned_house_price_df.corr().round

Corln
```

Out[48]:
```
<bound method DataFrame.round of                price  crime_rate  resid_area  air_qual  room_num       age  \
price        1.000000   -0.423725   -0.523338 -0.541934  0.690281 -0.540837
crime_rate  -0.423725    1.000000    0.543851  0.686964 -0.087925  0.396657
resid_area  -0.523338    0.543851    1.000000  0.765275 -0.326610  0.568713
air_qual    -0.541934    0.686964    0.765275  1.000000 -0.254758  0.712397
room_num     0.690281   -0.087925   -0.326610 -0.254758  1.000000 -0.257448
age         -0.540837    0.396657    0.568713  0.712397 -0.257448  1.000000
dist1        0.340329   -0.426536   -0.668186 -0.764331  0.225044 -0.683794
dist2        0.337374   -0.424967   -0.671186 -0.767056  0.221140 -0.682782
dist3        0.337279   -0.429208   -0.672194 -0.768499  0.217026 -0.684020
dist4        0.341857   -0.422480   -0.673028 -0.763909  0.219525 -0.686512
teachers     0.509967   -0.335335   -0.391901 -0.467297  0.270747 -0.374904
poor_prop   -0.762687    0.445917    0.585140  0.596553 -0.600987  0.680436
airport      0.093032   -0.118009   -0.079627 -0.021079  0.075985  0.022036
n_hos_beds   0.032679   -0.015067    0.035257  0.019843 -0.040290  0.022124
n_hot_rooms -0.049872    0.006653   -0.053231  0.012567 -0.068215 -0.021908
waterbody    0.035904    0.054996    0.011126 -0.005745  0.027504 -0.083754
rainfall     0.010070    0.008199   -0.017529 -0.025012 -0.031480 -0.024975
parks       -0.484286    0.571601    0.665501  0.861808 -0.244505  0.639455

                dist1     dist2     dist3     dist4  teachers  poor_prop  \
price        0.340329  0.337374  0.337279  0.341857  0.509967  -0.762687
crime_rate  -0.426536 -0.424967 -0.429208 -0.422480 -0.335335   0.445917
resid_area  -0.668186 -0.671186 -0.672194 -0.673028 -0.391901   0.585140
air_qual    -0.764331 -0.767056 -0.768499 -0.763909 -0.467297   0.596553
room_num     0.225044  0.221140  0.217026  0.219525  0.270747  -0.600987
age         -0.683794 -0.682782 -0.684020 -0.686512 -0.374904   0.680436
dist1        1.000000  0.997382  0.997315  0.993108  0.365583  -0.488337
dist2        0.997382  1.000000  0.997681  0.992875  0.371386  -0.484781
dist3        0.997315  0.997681  1.000000  0.993017  0.367711  -0.485737
dist4        0.993108  0.992875  0.993017  1.000000  0.365464  -0.495200
teachers     0.365583  0.371386  0.367711  0.365464  1.000000  -0.420673
poor_prop   -0.488337 -0.484781 -0.485737 -0.495200 -0.420673   1.000000
airport      0.007330  0.001281  0.000223  0.000059 -0.017750  -0.014663
n_hos_beds  -0.058670 -0.061557 -0.053662 -0.049585  0.034591  -0.006270
n_hot_rooms  0.010282  0.013444  0.012897  0.019971 -0.045068  -0.013290
waterbody   -0.031865 -0.029910 -0.029776 -0.018612  0.035934  -0.032564
rainfall     0.050959  0.046014  0.045058  0.056861 -0.038146  -0.007176
parks       -0.676051 -0.679496 -0.684453 -0.681214 -0.379276   0.548048

              airport  n_hos_beds  n_hot_rooms  waterbody  rainfall     parks
price        0.093032    0.032679    -0.049872   0.035904  0.010070 -0.484286
crime_rate  -0.118009   -0.015067     0.006653   0.054996  0.008199  0.571601
resid_area  -0.079627    0.035257    -0.053231   0.011126 -0.017529  0.665501
air_qual    -0.021079    0.019843     0.012567  -0.005745 -0.025012  0.861808
room_num     0.075985   -0.040290    -0.068215   0.027504 -0.031480 -0.244505
age          0.022036    0.022124    -0.021908  -0.083754 -0.024975  0.639455
dist1        0.007330   -0.058670     0.010282  -0.031865  0.050959 -0.676051
dist2        0.001281   -0.061557     0.013444  -0.029910  0.046014 -0.679496
dist3        0.000223   -0.053662     0.012897  -0.029776  0.045058 -0.684453
dist4        0.000059   -0.049585     0.019971  -0.018612  0.056861 -0.681214
teachers    -0.017750    0.034591    -0.045068   0.035934 -0.038146 -0.379276
poor_prop   -0.014663   -0.006270    -0.013290  -0.032564 -0.007176  0.548048
airport      1.000000    0.008755    -0.011723  -0.103755 -0.022772  0.002570
n_hos_beds   0.008755    1.000000    -0.060742   0.050169  0.061829 -0.035785
n_hot_rooms -0.011723   -0.060742     1.000000   0.006061  0.068591  0.040626
waterbody   -0.103755    0.050169     0.006061   1.000000  0.079793 -0.003835
rainfall    -0.022772    0.061829     0.068591   0.079793  1.000000 -0.019033
parks        0.002570   -0.035785     0.040626  -0.003835 -0.019033  1.000000  >
```

Lab 2

basic building model simple linear regressin with the help of statomodel, skelearn,pca method.

In [26]:
```
# Example: Use cleaned data for modeling
from sklearn.model_selection import train_test_split

X = cleaned_house_price_df.drop('price', axis=1)  # Features
y = cleaned_house_price_df['price']  # Target variable

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [27]:
```
y=cleaned_house_price_df["price"]
print(y)
X=cleaned_house_price_df.drop(columns=["price"])
print(X)
```

```
0      24.0
1      21.6
3      33.4
4      36.2
5      28.7
       ... 
501    22.4
502    20.6
503    23.9
504    22.0
505    19.0
Name: price, Length: 346, dtype: float64
     crime_rate  resid_area  air_qual  room_num   age  dist1  dist2  dist3  \
0       0.00632       32.31     0.538     6.575  65.2   4.35   3.81   4.18   
1       0.02731       37.07     0.469     6.421  78.9   4.99   4.70   5.12   
3       0.03237       32.18     0.458     6.998  45.8   6.21   5.93   6.16   
4       0.06905       32.18     0.458     7.147  54.2   6.16   5.86   6.37   
5       0.02985       32.18     0.458     6.430  58.7   6.22   5.80   6.23   
..          ...         ...       ...       ...   ...    ...    ...    ...   
501     0.06263       41.93     0.573     6.593  69.1   2.64   2.45   2.76   
502     0.04527       41.93     0.573     6.120  76.7   2.44   2.11   2.46   
503     0.06076       41.93     0.573     6.976  91.0   2.34   2.06   2.29   
504     0.10959       41.93     0.573     6.794  89.3   2.54   2.31   2.40   
505     0.04741       41.93     0.573     6.030  80.8   2.72   2.24   2.64   

     dist4  teachers  poor_prop  airport  n_hos_beds  n_hot_rooms  waterbody  \
0     4.01      24.7       4.98        1       5.480      11.1920          2   
1     5.06      22.2       9.14        0       7.332      12.1728          1   
3     5.96      21.3       2.94        1       9.268      11.2672          1   
4     5.86      21.3       5.33        0       8.824      11.2896          1   
5     5.99      21.3       5.21        1       7.174      14.2296          0   
..     ...       ...        ...      ...         ...          ...        ...   
501   2.06      19.0       9.67        0       9.348      12.1792          3   
502   2.14      19.0       9.08        1       6.612      13.1648          3   
503   1.98      19.0       5.64        0       5.478      12.1912          0   
504   2.31      19.0       6.48        1       7.940      15.1760          0   
505   2.42      19.0       7.88        1      10.280      10.1520          0   

     rainfall     parks  
0          23  0.049347  
1          42  0.046146  
3          45  0.047151  
4          55  0.039474  
5          53  0.045910  
..        ...       ...  
501        27  0.056006  
502        20  0.059903  
503        31  0.057572  
504        47  0.060694  
505        45  0.060336  

[346 rows x 17 columns]
```

In [28]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=42)
X_train,X_test,y_train,y_test
```

Out[28]:

```
(     crime_rate  resid_area  air_qual  room_num   age  dist1  dist2  dist3  \
 124     0.09849       55.65     0.581     5.879  95.8   2.21   1.74   2.20   
 9       0.17004       37.87     0.524     6.004  85.9   6.67   6.55   6.85   
 188     0.12579       33.44     0.437     6.556  29.1   4.62   4.28   4.68   
 108     0.12802       38.56     0.520     6.474  97.1   2.72   2.18   2.54   
 459     6.80117       48.10     0.713     6.081  84.4   2.74   2.48   2.80   
 ..          ...         ...       ...       ...   ...    ...    ...    ...   
 238     0.08244       34.93     0.428     6.481  18.5   6.26   6.12   6.45   
 76      0.10153       42.83     0.437     6.279  74.5   4.33   3.72   4.26   
 114     0.14231       40.01     0.547     6.254  84.2   2.54   2.14   2.29   
 340     0.06151       35.19     0.515     5.968  58.5   5.06   4.76   4.97   
 109     0.26363       38.56     0.520     6.229  91.2   2.57   2.28   2.88   
 
      dist4  teachers  poor_prop  airport  n_hos_beds  n_hot_rooms  waterbody  \
 124   1.88      20.9      17.58        0       6.276      13.1504          2   
 9     6.29      24.8      17.10        1       9.478      14.1512          2   
 188   4.69      24.8       4.56        1       7.596      10.2384          2   
 108   2.30      19.1      12.27        0       8.596      10.1584          3   
 459   2.85      19.8      14.70        1       5.400      14.1600          1   
 ..     ...       ...        ...      ...         ...          ...        ...   
 238   5.93      23.4       6.36        1       7.374      12.1896          3   
 76    3.90      21.3      11.97        0       7.300      12.1600          0   
 114   2.05      22.2      10.45        0       6.670      12.1480          3   
 340   4.46      19.8       9.29        1       6.474      13.1496          0   
 109   2.45      19.1      15.55        1       9.788      14.1552          0   
 
      rainfall     parks  
 124        56  0.062119  
 9          45  0.050727  
 188        40  0.045949  
 108        48  0.050115  
 459        27  0.072636  
 ..        ...       ...  
 238        48  0.049068  
 76         22  0.045275  
 114        39  0.056980  
 340        20  0.053312  
 109        41  0.055474  
 
 [276 rows x 17 columns],
      crime_rate  resid_area  air_qual  room_num   age  dist1  dist2  dist3  \
 116     0.13158       40.01     0.547     6.176  72.5   2.93   2.71   3.00   
 193     0.02187       32.93     0.401     6.800   9.9   6.46   6.13   6.45   
 348     0.01501       32.01     0.435     6.635  29.7   8.62   8.18   8.54   
 274     0.05644       36.41     0.447     6.758  32.9   4.20   3.98   4.38   
 475     6.39312       48.10     0.584     6.162  97.4   2.50   2.17   2.50   
 ..          ...         ...       ...       ...   ...    ...    ...    ...   
 217     0.07013       43.89     0.550     6.642  85.1   3.52   3.19   3.69   
 78      0.05646       42.83     0.437     6.232  53.7   5.34   5.00   5.10   
 81      0.04462       34.86     0.426     6.619  70.4   5.71   5.11   5.71   
 18      0.80271       38.14     0.538     5.456  36.6   3.80   3.52   3.86   
 71      0.15876       40.81     0.413     5.961  17.5   5.44   5.20   5.42   
 
      dist4  teachers  poor_prop  airport  n_hos_beds  n_hot_rooms  waterbody  \
 116   2.28      22.2      12.04        0       6.524      13.1696          0   
 193   5.84      24.4       5.03        0       6.822      13.2488          2   
 348   8.03      23.0       5.99        1       6.290      13.1960          0   
 274   3.76      22.4       3.53        0      10.648      12.2592          2   
 475   1.65      19.8      24.10        1       9.066      11.1064          1   
 ..     ...       ...        ...      ...         ...          ...        ...   
 217   3.28      23.6       9.69        0       9.274      14.2296          0   
 78    4.62      21.3      12.34        1       8.824      15.1696          1   
 81    5.08      21.0       7.22        1       5.578      12.1912          1   
 18    4.00      19.0      11.69        1       8.504      12.1616          3   
 71    5.09      20.8       9.88        0      10.034      13.1736          3   
```

```
     rainfall     parks
116        23  0.052774
193        24  0.041103
348        45  0.048924
274        30  0.042515
475        27  0.063035
..        ...       ...
217        23  0.048611
78         53  0.037647
81         49  0.050033
18         41  0.054251
71         46  0.039985

[70 rows x 17 columns],
124    18.8
9      18.9
188    29.8
108    19.8
459    20.0
       ...
238    23.7
76     20.0
114    18.5
340    18.7
109    19.4
Name: price, Length: 276, dtype: float64,
116    21.2
193    31.1
348    24.5
274    32.4
475    13.3
       ...
217    28.7
78     21.2
81     23.9
18     20.2
71     21.7
Name: price, Length: 70, dtype: float64)
```

In [29]:
```python
import statsmodels.api as sm
# Add a constant to the model (intercept)

X_train_sm = sm.add_constant(X_train)
X_train_sm
X_test_sm = sm.add_constant(X_test)
X_test_sm
# Fit the model using statsmodels
model = sm.OLS(y_train, X_train_sm).fit()


print(model.summary())
```

```
                            OLS Regression Results                            
==============================================================================
Dep. Variable:                  price   R-squared:                       0.746
Model:                            OLS   Adj. R-squared:                  0.729
Method:                 Least Squares   F-statistic:                     44.51
Date:                Fri, 22 Nov 2024   Prob (F-statistic):           1.33e-66
Time:                        18:53:44   Log-Likelihood:                -668.56
No. Observations:                 276   AIC:                             1373.
Df Residuals:                     258   BIC:                             1438.
Df Model:                          17                                         
Covariance Type:            nonrobust                                         
===============================================================================
                  coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------
const          -8.7599      5.645     -1.552      0.122     -19.875       2.356
crime_rate     -0.2813      0.125     -2.256      0.025      -0.527      -0.036
resid_area     -0.0345      0.045     -0.771      0.441      -0.123       0.054
air_qual      -11.1398      5.676     -1.962      0.051     -22.318       0.038
room_num        5.4250      0.544      9.966      0.000       4.353       6.497
age            -0.0415      0.011     -3.617      0.000      -0.064      -0.019
dist1           0.4472      1.425      0.314      0.754      -2.360       3.254
dist2          -0.3723      1.524     -0.244      0.807      -3.372       2.628
dist3          -1.1344      1.521     -0.746      0.457      -4.130       1.862
dist4           0.2074      0.845      0.245      0.806      -1.457       1.872
teachers        0.5629      0.107      5.248      0.000       0.352       0.774
poor_prop      -0.2545      0.063     -4.016      0.000      -0.379      -0.130
airport         0.6052      0.355      1.704      0.090      -0.094       1.305
n_hos_beds      0.1131      0.120      0.945      0.346      -0.123       0.349
n_hot_rooms     0.0291      0.104      0.279      0.781      -0.177       0.235
waterbody      -0.0649      0.166     -0.391      0.696      -0.392       0.262
rainfall        0.0048      0.014      0.352      0.725      -0.022       0.032
parks          -3.4894     41.961     -0.083      0.934     -86.120      79.141
==============================================================================
Omnibus:                       21.707   Durbin-Watson:                   1.993
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               27.010
Skew:                           0.600   Prob(JB):                     1.36e-06
Kurtosis:                       3.954   Cond. No.                     2.26e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.26e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Here

$$R^2$$

= 0.746 it means it explains 74.6% varaiance explain,Adj-R-square =0.729 it represent the model performence.

F-statistics 44.51 (1.33e-66) represent low p-value it means atleast one predictor variable has significane.

Coefficients:-

here,room_num increment in 1 unit then house price increament approx 5.43,also poor_prop represent if 1 unit increment then house price decrement in approx in -0.2545

Statistical significance

if p-value is <0.05 then it is significance otherwise it is not significance here,all predictor variable which are less than the p-value then significane other one is not significance.

Model Diagnostics

Omnibus test represent the residuals normally distributes or not ,low p-value indicate it is not normally distributed.

Durbin-Watson Statistics if it is approx 2 then there is no autocorrelation if it is higher then create a problem.

Condition number indicate if it is greater (2.26e+0) then it means multicollinearity or numerical problem.

In [30]:
```
print(model.params)
```

```
const          -8.759872
crime_rate     -0.281278
resid_area     -0.034502
air_qual      -11.139810
room_num        5.425007
age            -0.041492
dist1           0.447244
dist2          -0.372286
dist3          -1.134372
dist4           0.207417
teachers        0.562927
poor_prop      -0.254503
airport         0.605244
n_hos_beds      0.113145
n_hot_rooms     0.029100
waterbody      -0.064891
rainfall        0.004803
parks          -3.489377
dtype: float64
```

In [31]:
```
print(model.conf_int())
```

```
                     0          1
const       -19.875462   2.355717
crime_rate   -0.526837  -0.035718
resid_area   -0.122617   0.053614
air_qual    -22.317861   0.038240
room_num      4.353050   6.496965
age          -0.064080  -0.018904
dist1        -2.359632   3.254119
dist2        -3.372490   2.627918
dist3        -4.130252   1.861507
dist4        -1.456998   1.871832
teachers      0.351708   0.774145
poor_prop    -0.379308  -0.129698
airport      -0.094279   1.304766
n_hos_beds   -0.122645   0.348935
n_hot_rooms  -0.176625   0.234826
waterbody    -0.391510   0.261729
rainfall     -0.022036   0.031643
parks       -86.119565  79.140812
```

Simple linear regression with help of skelearn

In [32]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
```

In [33]:
```
y=cleaned_house_price_df["price"]
print(y)
```

```
X=cleaned_house_price_df.drop(columns=["price"])
print(X)
```

```
0      24.0
1      21.6
3      33.4
4      36.2
5      28.7
       ...
501    22.4
502    20.6
503    23.9
504    22.0
505    19.0
Name: price, Length: 346, dtype: float64
     crime_rate  resid_area  air_qual  room_num   age  dist1  dist2  dist3  \
0       0.00632       32.31     0.538     6.575  65.2   4.35   3.81   4.18
1       0.02731       37.07     0.469     6.421  78.9   4.99   4.70   5.12
3       0.03237       32.18     0.458     6.998  45.8   6.21   5.93   6.16
4       0.06905       32.18     0.458     7.147  54.2   6.16   5.86   6.37
5       0.02985       32.18     0.458     6.430  58.7   6.22   5.80   6.23
..          ...         ...       ...       ...   ...    ...    ...    ...
501     0.06263       41.93     0.573     6.593  69.1   2.64   2.45   2.76
502     0.04527       41.93     0.573     6.120  76.7   2.44   2.11   2.46
503     0.06076       41.93     0.573     6.976  91.0   2.34   2.06   2.29
504     0.10959       41.93     0.573     6.794  89.3   2.54   2.31   2.40
505     0.04741       41.93     0.573     6.030  80.8   2.72   2.24   2.64

     dist4  teachers  poor_prop  airport  n_hos_beds  n_hot_rooms  waterbody  \
0     4.01      24.7       4.98        1       5.480      11.1920          2
1     5.06      22.2       9.14        0       7.332      12.1728          1
3     5.96      21.3       2.94        1       9.268      11.2672          1
4     5.86      21.3       5.33        0       8.824      11.2896          1
5     5.99      21.3       5.21        1       7.174      14.2296          0
..     ...       ...        ...      ...         ...          ...        ...
501   2.06      19.0       9.67        0       9.348      12.1792          3
502   2.14      19.0       9.08        1       6.612      13.1648          3
503   1.98      19.0       5.64        0       5.478      12.1912          0
504   2.31      19.0       6.48        1       7.940      15.1760          0
505   2.42      19.0       7.88        1      10.280      10.1520          0

     rainfall     parks
0          23  0.049347
1          42  0.046146
3          45  0.047151
4          55  0.039474
5          53  0.045910
..        ...       ...
501        27  0.056006
502        20  0.059903
503        31  0.057572
504        47  0.060694
505        45  0.060336

[346 rows x 17 columns]
```

In [34]:
```
# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [35]:
```
# Create a linear regression model
model = LinearRegression()

# Fit the model to the training data
model.fit(X_train, y_train)
```

Out[35]:

```
▾ LinearRegression
LinearRegression()
```

In [36]:

```
# Perform cross-validation and calculate scores
cv_scores = cross_val_score(model, X, y, cv=5)  # 5-fold cross-validation

print("Cross-Validation Scores:", cv_scores)
print("Mean CV Score:", np.mean(cv_scores))
```

```
Cross-Validation Scores: [0.76154645 0.53154105 0.39993753 0.66360877 0.30909851]
Mean CV Score: 0.533146461120538
```

here, mean CV indicate is coloser to 1 then better performence.

In [37]:

```
# Make predictions on the test set
y_pred = model.predict(X_test)
```

In [38]:

```
# Calculate Mean Squared Error and R-squared value
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error:", mse)
print("R-squared:", r2)
```

```
Mean Squared Error: 5.345971075283202
R-squared: 0.8200129581910771
```

here,

$$R^2$$

=0.8200129581910771 model is better in performence.

Simple linear regression with help of pca method

In [39]:

```
y=cleaned_house_price_df["price"]
print(y)
X=cleaned_house_price_df.drop(columns=["price"])
print(X)
```

```
0      24.0
1      21.6
3      33.4
4      36.2
5      28.7
       ...
501    22.4
502    20.6
503    23.9
504    22.0
505    19.0
Name: price, Length: 346, dtype: float64
     crime_rate  resid_area  air_qual  room_num   age  dist1  dist2  dist3  \
0       0.00632       32.31     0.538     6.575  65.2   4.35   3.81   4.18
1       0.02731       37.07     0.469     6.421  78.9   4.99   4.70   5.12
3       0.03237       32.18     0.458     6.998  45.8   6.21   5.93   6.16
4       0.06905       32.18     0.458     7.147  54.2   6.16   5.86   6.37
5       0.02985       32.18     0.458     6.430  58.7   6.22   5.80   6.23
..          ...         ...       ...       ...   ...    ...    ...    ...
501     0.06263       41.93     0.573     6.593  69.1   2.64   2.45   2.76
502     0.04527       41.93     0.573     6.120  76.7   2.44   2.11   2.46
503     0.06076       41.93     0.573     6.976  91.0   2.34   2.06   2.29
504     0.10959       41.93     0.573     6.794  89.3   2.54   2.31   2.40
505     0.04741       41.93     0.573     6.030  80.8   2.72   2.24   2.64

     dist4  teachers  poor_prop  airport  n_hos_beds  n_hot_rooms  waterbody  \
0     4.01      24.7       4.98        1       5.480      11.1920          2
1     5.06      22.2       9.14        0       7.332      12.1728          1
3     5.96      21.3       2.94        1       9.268      11.2672          1
4     5.86      21.3       5.33        0       8.824      11.2896          1
5     5.99      21.3       5.21        1       7.174      14.2296          0
..     ...       ...        ...      ...         ...          ...        ...
501   2.06      19.0       9.67        0       9.348      12.1792          3
502   2.14      19.0       9.08        1       6.612      13.1648          3
503   1.98      19.0       5.64        0       5.478      12.1912          0
504   2.31      19.0       6.48        1       7.940      15.1760          0
505   2.42      19.0       7.88        1      10.280      10.1520          0

     rainfall     parks
0          23  0.049347
1          42  0.046146
3          45  0.047151
4          55  0.039474
5          53  0.045910
..        ...       ...
501        27  0.056006
502        20  0.059903
503        31  0.057572
504        47  0.060694
505        45  0.060336

[346 rows x 17 columns]
```

In [40]:
```
# Standardizing the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

In [41]:
```
from sklearn.decomposition import PCA
# Apply PCA to reduce dimensions ( 2 components)
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Display explained variance ratio for each principal component
print("Explained Variance Ratio:", pca.explained_variance_ratio_)
```

```
Explained Variance Ratio: [0.43970253 0.08296809]
```

In [42]:
```
# split the data
X_train,X_test,y_train,y_test=train_test_split(X_pca,y,test_size=0.2,random_state=42)
```

In [43]:
```
# Create a linear regression model
model = LinearRegression()

# Fit the model to the training data
model.fit(X_train, y_train)
```

Out[43]:
```
▾ LinearRegression
LinearRegression()
```

In [44]:
```
# To make prediction
y_pred=model.predict(X_test)
```

In [45]:
```
from sklearn.metrics import mean_squared_error,r2_score
# Evaluation the model performence
mse=mean_squared_error(y_test,y_pred)
r2=r2_score(y_test,y_pred)
print("Mean Square Error:",mse)
print("R- square:",r2)
```

```
Mean Square Error: 5.973267371744891
R- square: 0.7988932770054
```

In [46]:
```
plt.scatter(y_test, y_pred)
plt.xlabel('Actual house_price')
plt.ylabel('Predicted house_price')
plt.title('Actual house_price vs Predicted house_price')
plt.plot([y.min(), y.max()], [y.min(), y.max()], color='red')  # Diagonal line for reference
plt.show()
```

In [ ]:

Lab 1 Data cleaning of House_Price_data

Lab 1

Date(30-09-2024)

```
#install.packages("tidyverse") # this is contain a lot of package like dplyr,forcats,ggplot2,lubricate,
purr,tidyr,tibble,stringer,readr
library(tidyverse)
```

```
## Warning: package 'tidyverse' was built under R version 4.3.3
```

```
## Warning: package 'ggplot2' was built under R version 4.3.3
```

```
## Warning: package 'tidyr' was built under R version 4.3.3
```

```
## Warning: package 'readr' was built under R version 4.3.3
```

```
## Warning: package 'dplyr' was built under R version 4.3.3
```

```
## Warning: package 'stringr' was built under R version 4.3.3
```

```
## Warning: package 'lubridate' was built under R version 4.3.3
```

```
## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
## ✓ dplyr     1.1.4     ✓ readr     2.1.5
## ✓ forcats   1.0.0     ✓ stringr   1.5.1
## ✓ ggplot2   3.5.1     ✓ tibble    3.2.1
## ✓ lubridate 1.9.3     ✓ tidyr     1.3.1
## ✓ purrr     1.0.2
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## ✕ dplyr::filter() masks stats::filter()
## ✕ dplyr::lag()    masks stats::lag()
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become error
s
```

```
sessionInfo()     # This is use for personal know which type of package intalls or not
```

```
## R version 4.3.2 (2023-10-31 ucrt)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 11 x64 (build 22631)
##
## Matrix products: default
##
##
## locale:
## [1] LC_COLLATE=English_United States.utf8
## [2] LC_CTYPE=English_United States.utf8
## [3] LC_MONETARY=English_United States.utf8
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United States.utf8
##
## time zone: Asia/Calcutta
## tzcode source: internal
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] lubridate_1.9.3 forcats_1.0.0   stringr_1.5.1   dplyr_1.1.4
##  [5] purrr_1.0.2     readr_2.1.5     tidyr_1.3.1     tibble_3.2.1
##  [9] ggplot2_3.5.1   tidyverse_2.0.0
##
## loaded via a namespace (and not attached):
##  [1] gtable_0.3.5      jsonlite_1.8.8    compiler_4.3.2    tidyselect_1.2.1
##  [5] jquerylib_0.1.4   scales_1.3.0      yaml_2.3.7        fastmap_1.2.0
```

```
##  [9] R6_2.5.1          generics_0.1.3    knitr_1.46        munsell_0.5.1    
## [13] bslib_0.8.0       pillar_1.9.0      tzdb_0.4.0        rlang_1.1.1      
## [17] utf8_1.2.4        stringi_1.8.4     cachem_1.1.0      xfun_0.44        
## [21] sass_0.4.9        timechange_0.3.0  cli_3.6.1         withr_3.0.0      
## [25] magrittr_2.0.3    digest_0.6.33     grid_4.3.2        rstudioapi_0.16.0
## [29] hms_1.1.3         lifecycle_1.0.4   vctrs_0.6.5       evaluate_0.23    
## [33] glue_1.7.0        fansi_1.0.6       colorspace_2.1-0  rmarkdown_2.27   
## [37] tools_4.3.2       pkgconfig_2.0.3   htmltools_0.5.8
```

```
# To load the libraries
library(dplyr)   # For use in data manipulation
#(filter,arrange,summarize,and transform in a very readable ,modern type of data frame)
library(tidyr)    # For use in data cleaning
```

To load the house_price_data

```
setwd("D:\\sandip sir 3rd sem lab")
data=read.csv("House_Price.csv")
head(data)
```

```
##   price crime_rate resid_area air_qual room_num  age dist1 dist2 dist3 dist4
## 1  24.0    0.00632      32.31    0.538    6.575 65.2  4.35  3.81  4.18  4.01
## 2  21.6    0.02731      37.07    0.469    6.421 78.9  4.99  4.70  5.12  5.06
## 3  34.7    0.02729      37.07    0.469    7.185 61.1  5.03  4.86  5.01  4.97
## 4  33.4    0.03237      32.18    0.458    6.998 45.8  6.21  5.93  6.16  5.96
## 5  36.2    0.06905      32.18    0.458    7.147 54.2  6.16  5.86  6.37  5.86
## 6  28.7    0.02985      32.18    0.458    6.430 58.7  6.22  5.80  6.23  5.99
##   teachers poor_prop airport n_hos_beds n_hot_rooms waterbody rainfall bus_ter
## 1     24.7      4.98     YES      5.480     11.1920     River       23     YES
## 2     22.2      9.14      NO      7.332     12.1728      Lake       42     YES
## 3     22.2      4.03      NO      7.394    101.1200      None       38     YES
## 4     21.3      2.94     YES      9.268     11.2672      Lake       45     YES
## 5     21.3      5.33      NO      8.824     11.2896      Lake       55     YES
## 6     21.3      5.21     YES      7.174     14.2296      None       53     YES
##        parks
## 1 0.04934731
## 2 0.04614563
## 3 0.04576397
## 4 0.04715060
## 5 0.03947400
## 6 0.04590965
```

To Inspect the data

```
# Check structure of the data
str(data)
```

```
## 'data.frame':    506 obs. of  19 variables:
##  $ price      : num  24 21.6 34.7 33.4 36.2 28.7 22.9 22.1 16.5 18.9 ...
##  $ crime_rate : num  0.00632 0.02731 0.02729 0.03237 0.06905 ...
##  $ resid_area : num  32.3 37.1 37.1 32.2 32.2 ...
##  $ air_qual   : num  0.538 0.469 0.469 0.458 0.458 0.458 0.524 0.524 0.524 0.524 ...
##  $ room_num   : num  6.58 6.42 7.18 7 7.15 ...
##  $ age        : num  65.2 78.9 61.1 45.8 54.2 58.7 66.6 96.1 100 85.9 ...
##  $ dist1      : num  4.35 4.99 5.03 6.21 6.16 6.22 5.87 6.04 6.18 6.67 ...
##  $ dist2      : num  3.81 4.7 4.86 5.93 5.86 5.8 5.47 5.85 5.85 6.55 ...
##  $ dist3      : num  4.18 5.12 5.01 6.16 6.37 6.23 5.7 6.25 6.3 6.85 ...
##  $ dist4      : num  4.01 5.06 4.97 5.96 5.86 5.99 5.2 5.66 6 6.29 ...
##  $ teachers   : num  24.7 22.2 22.2 21.3 21.3 21.3 24.8 24.8 24.8 24.8 ...
##  $ poor_prop  : num  4.98 9.14 4.03 2.94 5.33 ...
##  $ airport    : chr  "YES" "NO" "NO" "YES" ...
##  $ n_hos_beds : num  5.48 7.33 7.39 9.27 8.82 ...
##  $ n_hot_rooms: num  11.2 12.2 101.1 11.3 11.3 ...
##  $ waterbody  : chr  "River" "Lake" "None" "Lake" ...
##  $ rainfall   : int  23 42 38 45 55 53 41 56 55 45 ...
##  $ bus_ter    : chr  "YES" "YES" "YES" "YES" ...
##  $ parks      : num  0.0493 0.0461 0.0458 0.0472 0.0395 ...
```

```
# Summarize the data
summary(data)
```

```
##      price         crime_rate         resid_area       air_qual     
##  Min.   : 5.00   Min.   : 0.00632   Min.   :30.46   Min.   :0.3850  
##  1st Qu.:17.02   1st Qu.: 0.08205   1st Qu.:35.19   1st Qu.:0.4490  
##  Median :21.20   Median : 0.25651   Median :39.69   Median :0.5380  
##  Mean   :22.53   Mean   : 3.61352   Mean   :41.14   Mean   :0.5547  
##  3rd Qu.:25.00   3rd Qu.: 3.67708   3rd Qu.:48.10   3rd Qu.:0.6240  
##  Max.   :50.00   Max.   :88.97620   Max.   :57.74   Max.   :0.8710  
##                                                                     
##     room_num          age             dist1            dist2       
##  Min.   :3.561   Min.   :  2.90   Min.   : 1.130   Min.   : 0.920  
##  1st Qu.:5.886   1st Qu.: 45.02   1st Qu.: 2.270   1st Qu.: 1.940  
##  Median :6.208   Median : 77.50   Median : 3.385   Median : 3.010  
##  Mean   :6.285   Mean   : 68.57   Mean   : 3.972   Mean   : 3.629  
##  3rd Qu.:6.623   3rd Qu.: 94.08   3rd Qu.: 5.367   3rd Qu.: 4.992  
##  Max.   :8.780   Max.   :100.00   Max.   :12.320   Max.   :11.930  
##                                                                     
##      dist3            dist4            teachers       poor_prop    
##  Min.   : 1.150   Min.   : 0.730   Min.   :18.00   Min.   : 1.73  
##  1st Qu.: 2.232   1st Qu.: 1.940   1st Qu.:19.80   1st Qu.: 6.95  
##  Median : 3.375   Median : 3.070   Median :20.95   Median :11.36  
##  Mean   : 3.961   Mean   : 3.619   Mean   :21.54   Mean   :12.65  
##  3rd Qu.: 5.407   3rd Qu.: 4.985   3rd Qu.:22.60   3rd Qu.:16.95  
##  Max.   :12.320   Max.   :11.940   Max.   :27.40   Max.   :37.97  
##                                                                     
##    airport            n_hos_beds       n_hot_rooms      waterbody        
##  Length:506         Min.   : 5.268   Min.   : 10.06   Length:506        
##  Class :character   1st Qu.: 6.635   1st Qu.: 11.19   Class :character  
##  Mode  :character   Median : 7.999   Median : 12.72   Mode  :character  
```

```
##                     Mean   : 7.900   Mean   : 13.04  
##                     3rd Qu.: 9.088   3rd Qu.: 14.17  
##                     Max.   :10.876   Max.   :101.12  
##                     NA's   :8                        
##     rainfall       bus_ter               parks        
##  Min.   : 3.00   Length:506         Min.   :0.03329  
##  1st Qu.:28.00   Class :character   1st Qu.:0.04646  
##  Median :39.00   Mode  :character   Median :0.05351  
##  Mean   :39.18                      Mean   :0.05445  
##  3rd Qu.:50.00                      3rd Qu.:0.06140  
##  Max.   :60.00                      Max.   :0.08671  
## 
```

To check the missing value

```
# Check missing values in each column
colSums(is.na(data))
```

```
##       price  crime_rate  resid_area    air_qual    room_num         age 
##           0           0           0           0           0           0 
##       dist1       dist2       dist3       dist4    teachers   poor_prop 
##           0           0           0           0           0           0 
##     airport  n_hos_beds n_hot_rooms   waterbody    rainfall     bus_ter 
##           0           8           0           0           0           0 
##       parks 
##           0 
```

```r
# Get positions of missing values in the dataset
which(is.na(data), arr.ind = TRUE)
```

```
##      row col
## [1,]  51  14
## [2,] 113  14
## [3,] 216  14
## [4,] 261  14
## [5,] 360  14
## [6,] 404  14
## [7,] 417  14
## [8,] 497  14
```

To handle the missing value

```r
# Remove rows with any missing values
data_clean <- na.omit(data)
head(data_clean)
```

```
##   price crime_rate resid_area air_qual room_num  age dist1 dist2 dist3 dist4
## 1  24.0    0.00632      32.31    0.538    6.575 65.2  4.35  3.81  4.18  4.01
## 2  21.6    0.02731      37.07    0.469    6.421 78.9  4.99  4.70  5.12  5.06
## 3  34.7    0.02729      37.07    0.469    7.185 61.1  5.03  4.86  5.01  4.97
## 4  33.4    0.03237      32.18    0.458    6.998 45.8  6.21  5.93  6.16  5.96
## 5  36.2    0.06905      32.18    0.458    7.147 54.2  6.16  5.86  6.37  5.86
## 6  28.7    0.02985      32.18    0.458    6.430 58.7  6.22  5.80  6.23  5.99
##   teachers poor_prop airport n_hos_beds n_hot_rooms waterbody rainfall bus_ter
## 1     24.7      4.98     YES      5.480     11.1920     River       23     YES
## 2     22.2      9.14      NO      7.332     12.1728      Lake       42     YES
## 3     22.2      4.03      NO      7.394    101.1200      None       38     YES
## 4     21.3      2.94     YES      9.268     11.2672      Lake       45     YES
## 5     21.3      5.33      NO      8.824     11.2896      Lake       55     YES
## 6     21.3      5.21     YES      7.174     14.2296      None       53     YES
##        parks
## 1 0.04934731
## 2 0.04614563
## 3 0.04576397
## 4 0.04715060
## 5 0.03947400
## 6 0.04590965
```

for checking the missing value which column have missing value

```
# Check missing values in each column
colSums(is.na(data_clean))
```

```
##       price  crime_rate  resid_area    air_qual    room_num         age
##           0           0           0           0           0           0
##       dist1       dist2       dist3       dist4    teachers   poor_prop
##           0           0           0           0           0           0
##     airport  n_hos_beds n_hot_rooms   waterbody    rainfall     bus_ter
##           0           0           0           0           0           0
##       parks
##           0
```

```
# Get positions of missing values in the dataset
which(is.na(data_clean), arr.ind = TRUE)
```

```
##      row col
```

OR if you want to Impute the missing values (i.e. fill in with mean or median)

```
# Impute missing values in 'n_hos_beds' column with the mean
data$n_hos_beds[is.na(data$n_hos_beds)] <- mean(data$n_hos_beds, na.rm = TRUE)
```

```r
str(data) # for checking the value of observation
```

```
## 'data.frame':    506 obs. of  19 variables:
##  $ price      : num  24 21.6 34.7 33.4 36.2 28.7 22.9 22.1 16.5 18.9 ...
##  $ crime_rate : num  0.00632 0.02731 0.02729 0.03237 0.06905 ...
##  $ resid_area : num  32.3 37.1 37.1 32.2 32.2 ...
##  $ air_qual   : num  0.538 0.469 0.469 0.458 0.458 0.458 0.524 0.524 0.524 0.524 ...
##  $ room_num   : num  6.58 6.42 7.18 7 7.15 ...
##  $ age        : num  65.2 78.9 61.1 45.8 54.2 58.7 66.6 96.1 100 85.9 ...
##  $ dist1      : num  4.35 4.99 5.03 6.21 6.16 6.22 5.87 6.04 6.18 6.67 ...
##  $ dist2      : num  3.81 4.7 4.86 5.93 5.86 5.8 5.47 5.85 5.85 6.55 ...
##  $ dist3      : num  4.18 5.12 5.01 6.16 6.37 6.23 5.7 6.25 6.3 6.85 ...
##  $ dist4      : num  4.01 5.06 4.97 5.96 5.86 5.99 5.2 5.66 6 6.29 ...
##  $ teachers   : num  24.7 22.2 22.2 21.3 21.3 21.3 24.8 24.8 24.8 24.8 ...
##  $ poor_prop  : num  4.98 9.14 4.03 2.94 5.33 ...
##  $ airport    : chr  "YES" "NO" "NO" "YES" ...
##  $ n_hos_beds : num  5.48 7.33 7.39 9.27 8.82 ...
##  $ n_hot_rooms: num  11.2 12.2 101.1 11.3 11.3 ...
##  $ waterbody  : chr  "River" "Lake" "None" "Lake" ...
##  $ rainfall   : int  23 42 38 45 55 53 41 56 55 45 ...
##  $ bus_ter    : chr  "YES" "YES" "YES" "YES" ...
##  $ parks      : num  0.0493 0.0461 0.0458 0.0472 0.0395 ...
```

```r
colSums(is.na(data))  # for checking the missing value
```

```
##       price  crime_rate  resid_area    air_qual    room_num         age 
##           0           0           0           0           0           0 
##       dist1       dist2       dist3       dist4    teachers   poor_prop 
##           0           0           0           0           0           0 
##     airport  n_hos_beds n_hot_rooms   waterbody    rainfall     bus_ter 
##           0           0           0           0           0           0 
##       parks 
##           0
```

```
which(is.na(data),arr.ind=TRUE) # for checking the position in the dataset
```

```
##      row col
```

it is use for checking duplicated row

```
# To check the Duplicate rows of given data
duplicates<-duplicated(data)  # in the output false i.e. there is no duplicate in the row
```

Dublicate column if happen then remove

```
# Specific columns ke basis par duplicates ko remove karna

remove_duplicate <- data %>%distinct(price, room_num, .keep_all = TRUE)
```

Another method to remove for Duplicate value

```
# Remove duplicate rows by using distinct (tidyverse) package
data1 <- data %>%distinct(.,price,crime_rate,resid_area,air_qual,room_num,age,dist1,dist2,dist3,dist4,teachers,poor_prop,airport,n_hos_beds,n_hot_rooms,waterbody,rainfall,  bus_ter,parks  )

head(data1)
```

```
##   price crime_rate resid_area air_qual room_num  age dist1 dist2 dist3 dist4
## 1  24.0    0.00632      32.31    0.538    6.575 65.2  4.35  3.81  4.18  4.01
## 2  21.6    0.02731      37.07    0.469    6.421 78.9  4.99  4.70  5.12  5.06
## 3  34.7    0.02729      37.07    0.469    7.185 61.1  5.03  4.86  5.01  4.97
## 4  33.4    0.03237      32.18    0.458    6.998 45.8  6.21  5.93  6.16  5.96
## 5  36.2    0.06905      32.18    0.458    7.147 54.2  6.16  5.86  6.37  5.86
## 6  28.7    0.02985      32.18    0.458    6.430 58.7  6.22  5.80  6.23  5.99
##   teachers poor_prop airport n_hos_beds n_hot_rooms waterbody rainfall bus_ter
## 1     24.7      4.98     YES      5.480     11.1920     River       23     YES
## 2     22.2      9.14      NO      7.332     12.1728      Lake       42     YES
## 3     22.2      4.03      NO      7.394    101.1200      None       38     YES
## 4     21.3      2.94     YES      9.268     11.2672      Lake       45     YES
## 5     21.3      5.33      NO      8.824     11.2896      Lake       55     YES
## 6     21.3      5.21     YES      7.174     14.2296      None       53     YES
##        parks
## 1 0.04934731
## 2 0.04614563
## 3 0.04576397
## 4 0.04715060
## 5 0.03947400
## 6 0.04590965
```

To change the categorical value into numerical into level encoding method apply here.

```
p=data1$airport
head(p)
```

```
## [1] "YES" "NO"  "NO"  "YES" "NO"  "YES"
```

```
q=data1$waterbody
head(q)
```

```
## [1] "River" "Lake"  "None"  "Lake"  "Lake"  "None"
```

```
r=data1$bus_ter
head(r)
```

```
## [1] "YES" "YES" "YES" "YES" "YES" "YES"
```

Assign them a numerical value

```
data_airport_numeric<- as.numeric(factor(p))
head(data_airport_numeric)
```

```
## [1] 2 1 1 2 1 2
```

```
#similarly
data_waterbody_numeric <- as.numeric(factor(q))
head(data_waterbody_numeric)
```

```
## [1] 4 1 3 1 1 3
```

To add column to the data distribution

```
df=data1 %>% mutate(waterbody=data_waterbody_numeric,airport=data_airport_numeric)
head(df)
```

```
##   price crime_rate resid_area air_qual room_num  age dist1 dist2 dist3 dist4
## 1  24.0    0.00632      32.31    0.538    6.575 65.2  4.35  3.81  4.18  4.01
## 2  21.6    0.02731      37.07    0.469    6.421 78.9  4.99  4.70  5.12  5.06
## 3  34.7    0.02729      37.07    0.469    7.185 61.1  5.03  4.86  5.01  4.97
## 4  33.4    0.03237      32.18    0.458    6.998 45.8  6.21  5.93  6.16  5.96
## 5  36.2    0.06905      32.18    0.458    7.147 54.2  6.16  5.86  6.37  5.86
## 6  28.7    0.02985      32.18    0.458    6.430 58.7  6.22  5.80  6.23  5.99
##   teachers poor_prop airport n_hos_beds n_hot_rooms waterbody rainfall bus_ter
## 1     24.7      4.98       2      5.480     11.1920         4       23     YES
## 2     22.2      9.14       1      7.332     12.1728         1       42     YES
## 3     22.2      4.03       1      7.394    101.1200         3       38     YES
## 4     21.3      2.94       2      9.268     11.2672         1       45     YES
## 5     21.3      5.33       1      8.824     11.2896         1       55     YES
## 6     21.3      5.21       2      7.174     14.2296         3       53     YES
##        parks
## 1 0.04934731
## 2 0.04614563
## 3 0.04576397
## 4 0.04715060
## 5 0.03947400
## 6 0.04590965
```

For checking the name function which type of column exist here

```
print(names(df))
```

```
##  [1] "price"       "crime_rate"  "resid_area"  "air_qual"    "room_num"   
##  [6] "age"         "dist1"       "dist2"       "dist3"       "dist4"      
## [11] "teachers"    "poor_prop"   "airport"     "n_hos_beds"  "n_hot_rooms"
## [16] "waterbody"   "rainfall"    "bus_ter"     "parks"
```

To remove the bus_ter from the data1

```
home_data <- df[, !(names(df) %in% c("bus_ter"))]
head(home_data)
```

```
##   price crime_rate resid_area air_qual room_num  age dist1 dist2 dist3 dist4
## 1  24.0    0.00632      32.31    0.538    6.575 65.2  4.35  3.81  4.18  4.01
## 2  21.6    0.02731      37.07    0.469    6.421 78.9  4.99  4.70  5.12  5.06
## 3  34.7    0.02729      37.07    0.469    7.185 61.1  5.03  4.86  5.01  4.97
## 4  33.4    0.03237      32.18    0.458    6.998 45.8  6.21  5.93  6.16  5.96
## 5  36.2    0.06905      32.18    0.458    7.147 54.2  6.16  5.86  6.37  5.86
## 6  28.7    0.02985      32.18    0.458    6.430 58.7  6.22  5.80  6.23  5.99
##   teachers poor_prop airport n_hos_beds n_hot_rooms waterbody rainfall
## 1     24.7      4.98       2      5.480     11.1920         4       23
## 2     22.2      9.14       1      7.332     12.1728         1       42
## 3     22.2      4.03       1      7.394    101.1200         3       38
## 4     21.3      2.94       2      9.268     11.2672         1       45
## 5     21.3      5.33       1      8.824     11.2896         1       55
## 6     21.3      5.21       2      7.174     14.2296         3       53
##        parks
## 1 0.04934731
## 2 0.04614563
## 3 0.04576397
## 4 0.04715060
## 5 0.03947400
## 6 0.04590965
```

For scaling the data like standardizes each column in the dataframe by subtracting the mean

and dividing by the standard deviation

```
scaled_df <- scale(home_data)
head(scaled_df)
```

```
##           price crime_rate resid_area   air_qual  room_num        age     dist1
## [1,]  0.1602176 -0.4193669 -1.2866362 -0.1440749 0.4132629 -0.1198948 0.1792735
## [2,] -0.1011583 -0.4169267 -0.5927944 -0.7395304 0.1940824  0.3668034 0.4828022
## [3,]  1.3255187 -0.4169290 -0.5927944 -0.7395304 1.2814456 -0.2655490 0.5017727
## [4,]  1.1839401 -0.4163384 -1.3055857 -0.8344581 1.0152978 -0.8090878 1.0614038
## [5,]  1.4888787 -0.4120741 -1.3055857 -0.8344581 1.2273620 -0.5106743 1.0376906
## [6,]  0.6720789 -0.4166314 -1.3055857 -0.8344581 0.2068916 -0.3508100 1.0661464
##           dist2     dist3     dist4   teachers  poor_prop    airport n_hos_beds
## [1,] 0.08594659 0.1034665 0.1862743  1.4575580 -1.0744990  0.9011172 -1.6517865
## [2,] 0.50803153 0.5469051 0.6864642  0.3027945 -0.4919525 -1.1075405 -0.3875704
## [3,] 0.58391197 0.4950134 0.6435908  0.3027945 -1.2075324 -1.1075405 -0.3452478
## [4,] 1.09136241 1.0375180 1.1151983 -0.1129203 -1.3601708  0.9011172  0.9339861
## [5,] 1.05816472 1.1365841 1.0675612 -0.1129203 -1.0254866 -1.1075405  0.6309018
## [6,] 1.02970955 1.0705400 1.1294895 -0.1129203 -1.0422909  0.9011172 -0.4954247
##      n_hot_rooms waterbody    rainfall      parks
## [1,]  -0.3530483  1.041470 -1.29312847 -0.4802878
## [2,]  -0.1658355 -1.647223  0.22520777 -0.7814096
## [3,]  16.8122016  0.145239 -0.09444197 -0.8173059
## [4,]  -0.3386943 -1.647223  0.46494507 -0.6868912
## [5,]  -0.3344186 -1.647223  1.26406940 -1.4088855
## [6,]   0.2267618  0.145239  1.10424453 -0.8036044
```

For use the model prediction

```
library(ggplot2)
# To set seed for reproducibility and split the data
set.seed(123)
train_indices <-sample(1:nrow(scaled_df),0.8*nrow(scaled_df))
head(train_indices)
```

```
## [1] 415 463 179  14 195 426
```

```
train_data <- scaled_df[train_indices,]
head(train_data)
```

```
##           price crime_rate resid_area   air_qual    room_num        age
## [1,] -1.6911954  4.8982568  1.0149946  1.1935426 -2.51293952  1.1163897
## [2,] -0.3298623  0.3535872  1.0149946  1.3661384  0.04606437  0.5124576
## [3,]  0.8027668 -0.4123798 -1.0330050 -0.3857090  0.81888923  0.2069391
## [4,] -0.2318463 -0.3468869 -0.4368257 -0.1440749 -0.47769171 -0.2406812
## [5,]  0.7156415 -0.4184287 -1.1962619 -1.3263561  0.45453718 -1.7682741
## [6,] -1.5496168  1.4237880  1.0149946  1.0727255 -0.55312398  0.9529728
##           dist1      dist2      dist3      dist4   teachers  poor_prop
## [1,] -1.0585544 -1.0095772 -0.9721079 -1.0141813 -0.8057784  3.4066275
## [2,] -0.4420118 -0.5685222 -0.5475391 -0.4520632 -0.8057784  0.1872182
## [3,] -0.3708722 -0.4167613 -0.3541244 -0.5330463  0.8570810 -0.8028307
## [4,]  0.3547510  0.4558637  0.4572739  0.4625697 -1.1753027 -0.6151835
## [5,]  1.2036828  1.1387877  1.1790410  1.0770886  1.3189864 -1.1585201
## [6,] -0.8641064 -0.9099842 -0.9390859 -0.8617425 -0.8057784  1.6435843
##         airport n_hos_beds n_hot_rooms waterbody    rainfall       parks
## [1,]  0.9011172 -1.2695182  -0.1881300  1.041470  0.06538290  1.39279225
## [2,] -1.1075405 -1.3719115   0.4035909  1.041470 -1.37304090  1.53103745
## [3,] -1.1075405 -1.3664505  -0.3440389  0.145239 -0.89356630  0.02475372
## [4,]  0.9011172 -1.3596243   0.2140875  0.145239 -0.01452953 -0.09311120
## [5,] -1.1075405  0.9435428   0.2273726  1.041470  1.58371914 -0.52864109
## [6,]  0.9011172 -1.5930810   0.3864883 -1.647223  0.94441967  1.20872493
```

```
test_data <- scaled_df[-train_indices,]
head(test_data)
```

```
##            price crime_rate resid_area   air_qual   room_num        age
## [1,]  0.16021761 -0.4193669 -1.2866362 -0.1440749  0.4132629 -0.1198948
## [2,] -0.47144095 -0.3459336 -0.4368257 -0.1440749 -0.2684739  0.5657458
## [3,]  0.06220162 -0.2975737 -0.4368257 -0.1440749 -0.4976172 -1.3952572
## [4,] -0.25362766 -0.3267801 -0.4368257 -0.1440749 -1.1793541 -1.1359217
## [5,] -0.84172356 -0.3089856 -0.4368257 -0.1440749 -0.3382132  0.7185050
## [6,] -0.27540899 -0.4087735 -0.7545936 -0.4806367 -0.6314027 -0.2548913
##            dist1       dist2      dist3      dist4  teachers  poor_prop
## [1,]  0.17927351  0.08594659  0.1034665  0.1862743  1.457558 -1.0744990
## [2,]  0.29783940  0.27090516  0.2449895  0.4578060 -1.175303 -0.3351131
## [3,]  0.32629521  0.36101318  0.2638592  0.3863503 -1.175303 -0.8504426
## [4,] -0.08157146 -0.05158670 -0.0474913  0.1815106 -1.175303 -0.1348628
## [5,]  0.25041304  0.36575571  0.3015987  0.3291857 -1.175303  0.6479340
## [6,] -0.27601953 -0.16540736 -0.1607097 -0.1900590 -0.343873 -0.1740726
##         airport n_hos_beds n_hot_rooms  waterbody    rainfall       parks
## [1,]  0.9011172 -1.6517865 -0.35304830  1.0414701 -1.29312847 -0.48028776
## [2,] -1.1075405 -0.6387753  0.01985038  0.1452390  0.78459480  0.51052700
## [3,] -1.1075405 -1.6640737 -0.54530032  0.1452390  0.54485750  0.02300766
## [4,]  0.9011172  0.4124628 -0.16797328 -0.7509921  0.14529533 -0.01910366
## [5,]  0.9011172  0.5435262  0.01465850 -1.6472231 -1.53286577 -0.53681378
## [6,] -1.1075405 -0.2728898  0.40435442  0.1452390 -0.01452953 -0.85559881
```

Checking what type of data like dataframe or not ,or array

```
str(train_data)
```

```
##  num [1:404, 1:18] -1.691 -0.33 0.803 -0.232 0.716 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr [1:18] "price" "crime_rate" "resid_area" "air_qual" ...
```

```
str(test_data)
```

```
##  num [1:102, 1:18] 0.1602 -0.4714 0.0622 -0.2536 -0.8417 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr [1:18] "price" "crime_rate" "resid_area" "air_qual" ...
```

Converting the data into as dataframe

```
# Convert the matrix back to a data frame
train_data <- as.data.frame(train_data)
test_data <- as.data.frame(test_data)
# Check the structure again to confirm it's a data frame
str(train_data)
```

```
## 'data.frame':    404 obs. of  18 variables:
##  $ price      : num  -1.691 -0.33 0.803 -0.232 0.716 ...
##  $ crime_rate : num  4.898 0.354 -0.412 -0.347 -0.418 ...
##  $ resid_area : num  1.015 1.015 -1.033 -0.437 -1.196 ...
##  $ air_qual   : num  1.194 1.366 -0.386 -0.144 -1.326 ...
##  $ room_num   : num  -2.5129 0.0461 0.8189 -0.4777 0.4545 ...
##  $ age        : num  1.116 0.512 0.207 -0.241 -1.768 ...
##  $ dist1      : num  -1.059 -0.442 -0.371 0.355 1.204 ...
##  $ dist2      : num  -1.01 -0.569 -0.417 0.456 1.139 ...
##  $ dist3      : num  -0.972 -0.548 -0.354 0.457 1.179 ...
##  $ dist4      : num  -1.014 -0.452 -0.533 0.463 1.077 ...
##  $ teachers   : num  -0.806 -0.806 0.857 -1.175 1.319 ...
##  $ poor_prop  : num  3.407 0.187 -0.803 -0.615 -1.159 ...
##  $ airport    : num  0.901 -1.108 -1.108 0.901 -1.108 ...
##  $ n_hos_beds : num  -1.27 -1.372 -1.366 -1.36 0.944 ...
##  $ n_hot_rooms: num  -0.188 0.404 -0.344 0.214 0.227 ...
##  $ waterbody  : num  1.041 1.041 0.145 0.145 1.041 ...
##  $ rainfall   : num  0.0654 -1.373 -0.8936 -0.0145 1.5837 ...
##  $ parks      : num  1.3928 1.531 0.0248 -0.0931 -0.5286 ...
```

```
str(test_data)
```

```
## 'data.frame':    102 obs. of  18 variables:
##  $ price      : num  0.1602 -0.4714 0.0622 -0.2536 -0.8417 ...
##  $ crime_rate : num  -0.419 -0.346 -0.298 -0.327 -0.309 ...
##  $ resid_area : num  -1.287 -0.437 -0.437 -0.437 -0.437 ...
##  $ air_qual   : num  -0.144 -0.144 -0.144 -0.144 -0.144 ...
##  $ room_num   : num  0.413 -0.268 -0.498 -1.179 -0.338 ...
##  $ age        : num  -0.12 0.566 -1.395 -1.136 0.719 ...
##  $ dist1      : num  0.1793 0.2978 0.3263 -0.0816 0.2504 ...
##  $ dist2      : num  0.0859 0.2709 0.361 -0.0516 0.3658 ...
##  $ dist3      : num  0.1035 0.245 0.2639 -0.0475 0.3016 ...
##  $ dist4      : num  0.186 0.458 0.386 0.182 0.329 ...
##  $ teachers   : num  1.46 -1.18 -1.18 -1.18 -1.18 ...
##  $ poor_prop  : num  -1.074 -0.335 -0.85 -0.135 0.648 ...
##  $ airport    : num  0.901 -1.108 -1.108 0.901 0.901 ...
##  $ n_hos_beds : num  -1.652 -0.639 -1.664 0.412 0.544 ...
##  $ n_hot_rooms: num  -0.353 0.0199 -0.5453 -0.168 0.0147 ...
##  $ waterbody  : num  1.041 0.145 0.145 -0.751 -1.647 ...
##  $ rainfall   : num  -1.293 0.785 0.545 0.145 -1.533 ...
##  $ parks      : num  -0.4803 0.5105 0.023 -0.0191 -0.5368 ...
```

To fit the linear regression model

```
model <-lm(price~.,data=train_data)
model
```

```
##
## Call:
## lm(formula = price ~ ., data = train_data)
##
## Coefficients:
## (Intercept)   crime_rate   resid_area     air_qual     room_num          age
##  -0.0004156   -0.0574209   -0.0317813   -0.1912701    0.2888081   -0.0091162
##       dist1        dist2        dist3        dist4     teachers    poor_prop
##  -0.2577738    0.7187468   -0.8052426    0.0604881    0.2302357   -0.4627006
##     airport   n_hos_beds  n_hot_rooms    waterbody     rainfall        parks
##   0.0609833    0.0479151    0.0171679    0.0135825   -0.0029603    0.0243372
```

To view the summary of the data

*** indicates a p-value less than 0.001 (highly significant).

** indicates a p-value between 0.001 and 0.01 (significant).

* indicates a p-value between 0.01 and 0.05 (marginally significant).

. indicates a p-value between 0.05 and 0.1 (borderline significance).

A blank space indicates a p-value greater than 0.1 (not significant).

Residual Standard Error (RSE) the standard deviation of the residuals(prediction errors) in the model.A lower RSE indicates a better fit of the model to the data.here(0.5355) model deviate from actual value.

d.f. (380) means (n-k)

6 observation deleted due to contained missing values for one or more predictor value.

Multiple R-squared is a measure of how well the independent variables in the model explain the variability of the dependent variable.here(72.64%) house prices can explained by the independent variables in the model.Higher values indicate a better fit.

Adjusted R-squared adjusts the R-squared value based on the number of predictors in the model and the number of observations. It accounts for the potential overfitting that can occur when adding more variables.(0.141 suggests that after adjusting the number of predictors approximately 71.41% of the variance is explained.)

The F-statistic tests the overall significance of the regression model. to determine if the independent variables have a statistically significant relationship with the dependent variable.(an F-statistic of 59.33 indicates that the model is significantly better at predicting the dependent variable than a model with no predictors)

p-value associated with the F-statistic is extremely small(2.2e-16)it means strong evidence against the null hypothesis (all coefficient are equal to zero.)

```
summary(model)
```

```
##
## Call:
## lm(formula = price ~ ., data = train_data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.9747 -0.3073 -0.0927  0.1764  2.9565
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.0004156  0.0267374  -0.016   0.9876
## crime_rate  -0.0574209  0.0296704  -1.935   0.0537 .
## resid_area  -0.0317813  0.0481535  -0.660   0.5096
## air_qual    -0.1912701  0.0806007  -2.373   0.0181 *
## room_num     0.2888081  0.0368416   7.839 4.46e-14 ***
## age         -0.0091162  0.0468615  -0.195   0.8459
## dist1       -0.2577738  0.4700626  -0.548   0.5837
## dist2        0.7187468  0.5140350   1.398   0.1628
## dist3       -0.8052426  0.5033840  -1.600   0.1105
## dist4        0.0604881  0.2692505   0.225   0.8224
## teachers     0.2302357  0.0313235   7.350 1.19e-12 ***
## poor_prop   -0.4627006  0.0446437 -10.364  < 2e-16 ***
## airport      0.0609833  0.0274982   2.218   0.0272 *
## n_hos_beds   0.0479151  0.0268804   1.783   0.0754 .
## n_hot_rooms  0.0171679  0.0243920   0.704   0.4820
## waterbody    0.0135825  0.0271057   0.501   0.6166
## rainfall    -0.0029603  0.0271148  -0.109   0.9131
## parks        0.0243372  0.0669352   0.364   0.7164
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.535 on 386 degrees of freedom
## Multiple R-squared:  0.7257, Adjusted R-squared:  0.7136
## F-statistic: 60.07 on 17 and 386 DF,  p-value: < 2.2e-16
```

To make the prediction

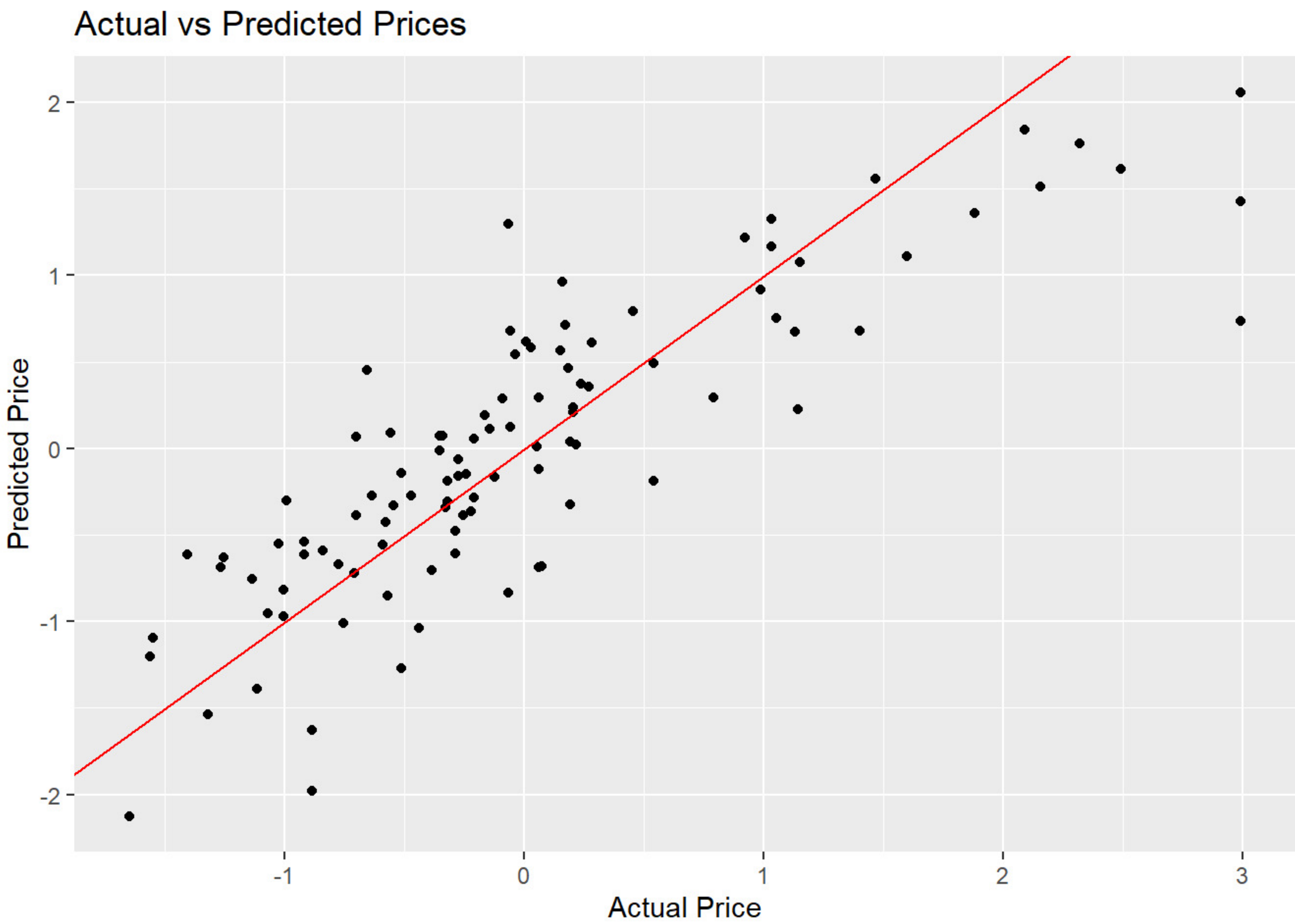
```
# Make predictions on the test set
predictions <- predict(model, newdata = test_data)
test_data$predicted_price <- predictions
```

for checking

```
# Calculate RMSE
rmse <- sqrt(mean((test_data$predicted_price - test_data$price)^2))
print(paste("RMSE:", rmse))
```

```
## [1] "RMSE: 0.544063271456163"
```

```r
# Visualize actual vs predicted prices
ggplot(test_data, aes(x = price, y = predicted_price)) +
  geom_point() +
  geom_abline(slope = 1, intercept = 0, color = "red") +
  labs(title = "Actual vs Predicted Prices", x = "Actual Price", y = "Predicted Price")
```


Actual vs Predicted Prices
Predicted Price
Actual Price
2
1
0
-1
-2
-1
0
1
2
3

Lab 2

To load library

```r
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 4.3.3
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

To load House_price

```r
library(readr)
```

```
## Warning: package 'readr' was built under R version 4.3.3
```

```
data<-read_csv("D:\\sandip sir 3rd sem lab\\House_Price.csv",show_col_types=FALSE)
data
```

```
## # A tibble: 506 × 19
##    price crime_rate resid_area air_qual room_num   age dist1 dist2 dist3 dist4
##    <dbl>      <dbl>      <dbl>    <dbl>    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1  24      0.00632       32.3    0.538     6.58  65.2  4.35  3.81  4.18  4.01
##  2  21.6    0.0273        37.1    0.469     6.42  78.9  4.99  4.7   5.12  5.06
##  3  34.7    0.0273        37.1    0.469     7.18  61.1  5.03  4.86  5.01  4.97
##  4  33.4    0.0324        32.2    0.458     7.00  45.8  6.21  5.93  6.16  5.96
##  5  36.2    0.0690        32.2    0.458     7.15  54.2  6.16  5.86  6.37  5.86
##  6  28.7    0.0298        32.2    0.458     6.43  58.7  6.22  5.8   6.23  5.99
##  7  22.9    0.0883        37.9    0.524     6.01  66.6  5.87  5.47  5.7   5.2
##  8  22.1    0.145         37.9    0.524     6.17  96.1  6.04  5.85  6.25  5.66
##  9  16.5    0.211         37.9    0.524     5.63 100    6.18  5.85  6.3   6
## 10  18.9    0.170         37.9    0.524     6.00  85.9  6.67  6.55  6.85  6.29
## # ℹ 496 more rows
## # ℹ 9 more variables: teachers <dbl>, poor_prop <dbl>, airport <chr>,
## #   n_hos_beds <dbl>, n_hot_rooms <dbl>, waterbody <chr>, rainfall <dbl>,
## #   bus_ter <chr>, parks <dbl>
```

To explore the data

```
# Checking the structure of the dataset
str(data)
```

```
## spc_tbl_ [506 × 19] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ price      : num [1:506] 24 21.6 34.7 33.4 36.2 28.7 22.9 22.1 16.5 18.9 ...
##  $ crime_rate : num [1:506] 0.00632 0.02731 0.02729 0.03237 0.06905 ...
##  $ resid_area : num [1:506] 32.3 37.1 37.1 32.2 32.2 ...
##  $ air_qual   : num [1:506] 0.538 0.469 0.469 0.458 0.458 0.458 0.524 0.524 0.524 0.524 ...
##  $ room_num   : num [1:506] 6.58 6.42 7.18 7 7.15 ...
##  $ age        : num [1:506] 65.2 78.9 61.1 45.8 54.2 58.7 66.6 96.1 100 85.9 ...
##  $ dist1      : num [1:506] 4.35 4.99 5.03 6.21 6.16 6.22 5.87 6.04 6.18 6.67 ...
##  $ dist2      : num [1:506] 3.81 4.7 4.86 5.93 5.86 5.8 5.47 5.85 5.85 6.55 ...
##  $ dist3      : num [1:506] 4.18 5.12 5.01 6.16 6.37 6.23 5.7 6.25 6.3 6.85 ...
##  $ dist4      : num [1:506] 4.01 5.06 4.97 5.96 5.86 5.99 5.2 5.66 6 6.29 ...
##  $ teachers   : num [1:506] 24.7 22.2 22.2 21.3 21.3 21.3 24.8 24.8 24.8 24.8 ...
##  $ poor_prop  : num [1:506] 4.98 9.14 4.03 2.94 5.33 ...
##  $ airport    : chr [1:506] "YES" "NO" "NO" "YES" ...
##  $ n_hos_beds : num [1:506] 5.48 7.33 7.39 9.27 8.82 ...
##  $ n_hot_rooms: num [1:506] 11.2 12.2 101.1 11.3 11.3 ...
##  $ waterbody  : chr [1:506] "River" "Lake" "None" "Lake" ...
##  $ rainfall   : num [1:506] 23 42 38 45 55 53 41 56 55 45 ...
##  $ bus_ter    : chr [1:506] "YES" "YES" "YES" "YES" ...
##  $ parks      : num [1:506] 0.0493 0.0461 0.0458 0.0472 0.0395 ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   price = col_double(),
##   ..   crime_rate = col_double(),
##   ..   resid_area = col_double(),
##   ..   air_qual = col_double(),
##   ..   room_num = col_double(),
##   ..   age = col_double(),
```

```
##   ..   dist1 = col_double(),
##   ..   dist2 = col_double(),
##   ..   dist3 = col_double(),
##   ..   dist4 = col_double(),
##   ..   teachers = col_double(),
##   ..   poor_prop = col_double(),
##   ..   airport = col_character(),
##   ..   n_hos_beds = col_double(),
##   ..   n_hot_rooms = col_double(),
##   ..   waterbody = col_character(),
##   ..   rainfall = col_double(),
##   ..   bus_ter = col_character(),
##   ..   parks = col_double()
##   .. )
##  - attr(*, "problems")=<externalptr>
```

```
# Summary statistics
summary(data)
```

```
##      price         crime_rate         resid_area       air_qual     
##  Min.   : 5.00   Min.   : 0.00632   Min.   :30.46   Min.   :0.3850  
##  1st Qu.:17.02   1st Qu.: 0.08205   1st Qu.:35.19   1st Qu.:0.4490  
##  Median :21.20   Median : 0.25651   Median :39.69   Median :0.5380  
##  Mean   :22.53   Mean   : 3.61352   Mean   :41.14   Mean   :0.5547  
##  3rd Qu.:25.00   3rd Qu.: 3.67708   3rd Qu.:48.10   3rd Qu.:0.6240  
##  Max.   :50.00   Max.   :88.97620   Max.   :57.74   Max.   :0.8710  
##                                                                     
##     room_num          age             dist1            dist2       
##  Min.   :3.561   Min.   :  2.90   Min.   : 1.130   Min.   : 0.920  
##  1st Qu.:5.886   1st Qu.: 45.02   1st Qu.: 2.270   1st Qu.: 1.940  
##  Median :6.208   Median : 77.50   Median : 3.385   Median : 3.010  
##  Mean   :6.285   Mean   : 68.57   Mean   : 3.972   Mean   : 3.629  
##  3rd Qu.:6.623   3rd Qu.: 94.08   3rd Qu.: 5.367   3rd Qu.: 4.992  
##  Max.   :8.780   Max.   :100.00   Max.   :12.320   Max.   :11.930  
##                                                                     
##      dist3            dist4           teachers       poor_prop    
##  Min.   : 1.150   Min.   : 0.730   Min.   :18.00   Min.   : 1.73  
##  1st Qu.: 2.232   1st Qu.: 1.940   1st Qu.:19.80   1st Qu.: 6.95  
##  Median : 3.375   Median : 3.070   Median :20.95   Median :11.36  
##  Mean   : 3.961   Mean   : 3.619   Mean   :21.54   Mean   :12.65  
##  3rd Qu.: 5.407   3rd Qu.: 4.985   3rd Qu.:22.60   3rd Qu.:16.95  
##  Max.   :12.320   Max.   :11.940   Max.   :27.40   Max.   :37.97  
##                                                                     
##    airport            n_hos_beds       n_hot_rooms      waterbody        
##  Length:506         Min.   : 5.268   Min.   : 10.06   Length:506        
##  Class :character   1st Qu.: 6.635   1st Qu.: 11.19   Class :character  
##  Mode  :character   Median : 7.999   Median : 12.72   Mode  :character  
```

```
##                    Mean   : 7.900   Mean   : 13.04
##                    3rd Qu.: 9.088   3rd Qu.: 14.17
##                    Max.   :10.876   Max.   :101.12
##                    NA's   :8
##     rainfall       bus_ter              parks
##  Min.   : 3.00   Length:506         Min.   :0.03329
##  1st Qu.:28.00   Class :character   1st Qu.:0.04646
##  Median :39.00   Mode  :character   Median :0.05351
##  Mean   :39.18                      Mean   :0.05445
##  3rd Qu.:50.00                      3rd Qu.:0.06140
##  Max.   :60.00                      Max.   :0.08671
##
```

```
# Check for NA values
sum(is.na(data))     # for count the total sum of missing value
```

```
## [1] 8
```

```
colSums(is.na(data)) # for count for each column
```

```
##       price  crime_rate  resid_area    air_qual    room_num         age
##           0           0           0           0           0           0
##       dist1       dist2       dist3       dist4    teachers   poor_prop
##           0           0           0           0           0           0
##     airport  n_hos_beds n_hot_rooms   waterbody    rainfall     bus_ter
##           0           8           0           0           0           0
##       parks
##           0
```

```
cleaned_data<- na.omit(data)
cleaned_data
```

```
## # A tibble: 498 × 19
##    price crime_rate resid_area air_qual room_num   age dist1 dist2 dist3 dist4
##    <dbl>      <dbl>      <dbl>    <dbl>    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1  24      0.00632       32.3    0.538     6.58  65.2  4.35  3.81  4.18  4.01
##  2  21.6    0.0273        37.1    0.469     6.42  78.9  4.99  4.7   5.12  5.06
##  3  34.7    0.0273        37.1    0.469     7.18  61.1  5.03  4.86  5.01  4.97
##  4  33.4    0.0324        32.2    0.458     7.00  45.8  6.21  5.93  6.16  5.96
##  5  36.2    0.0690        32.2    0.458     7.15  54.2  6.16  5.86  6.37  5.86
##  6  28.7    0.0298        32.2    0.458     6.43  58.7  6.22  5.8   6.23  5.99
##  7  22.9    0.0883        37.9    0.524     6.01  66.6  5.87  5.47  5.7   5.2
##  8  22.1    0.145         37.9    0.524     6.17  96.1  6.04  5.85  6.25  5.66
##  9  16.5    0.211         37.9    0.524     5.63 100    6.18  5.85  6.3   6
## 10  18.9    0.170         37.9    0.524     6.00  85.9  6.67  6.55  6.85  6.29
## # ℹ 488 more rows
## # ℹ 9 more variables: teachers <dbl>, poor_prop <dbl>, airport <chr>,
## #   n_hos_beds <dbl>, n_hot_rooms <dbl>, waterbody <chr>, rainfall <dbl>,
## #   bus_ter <chr>, parks <dbl>
```

Use the dplyr Package

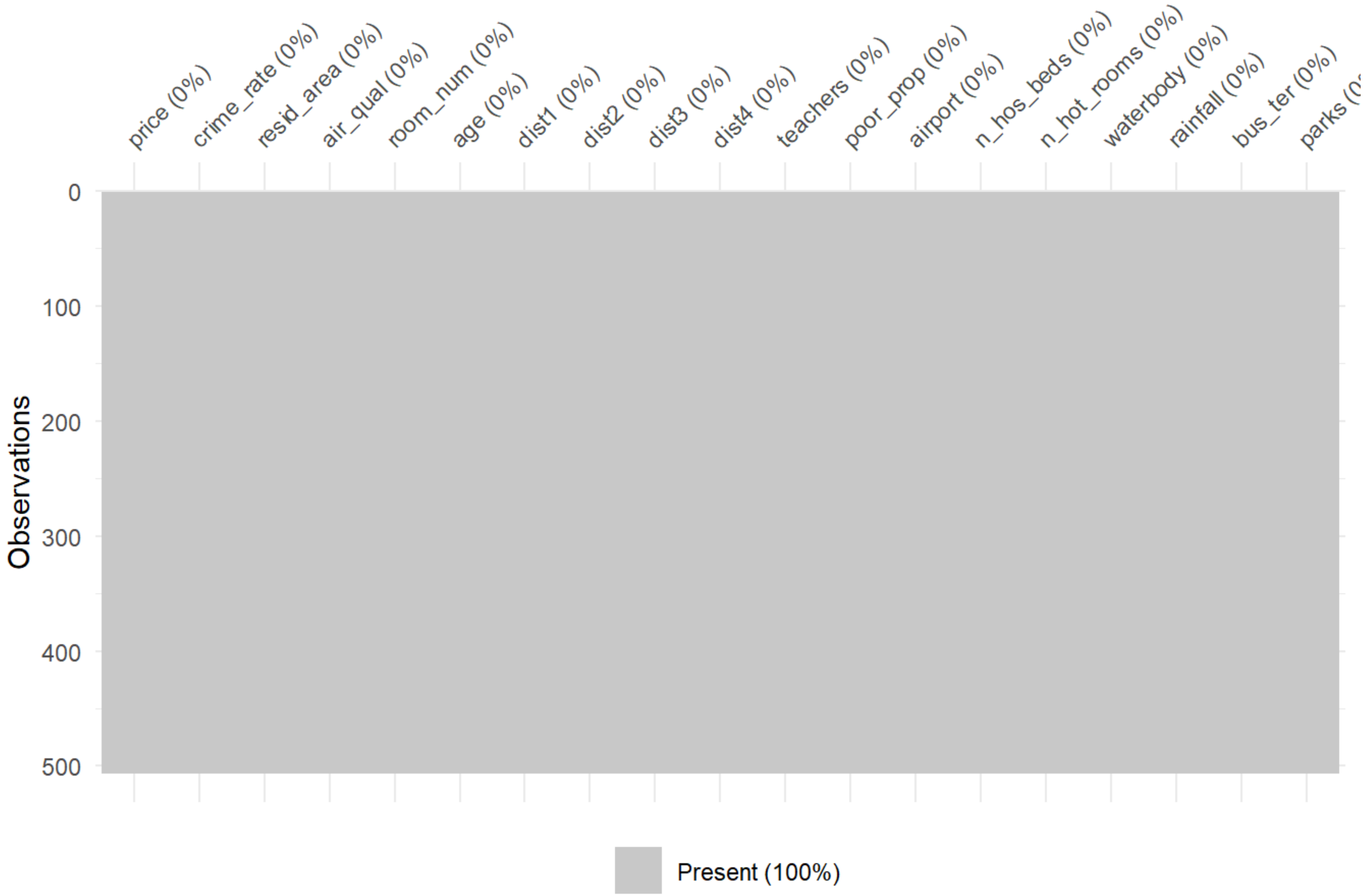
```
library(dplyr)
data <- data %>%
    mutate( n_hos_beds = ifelse(is.na( n_hos_beds), mean( n_hos_beds, na.rm = TRUE),  n_hos_beds))
```

Visualizing Missing data

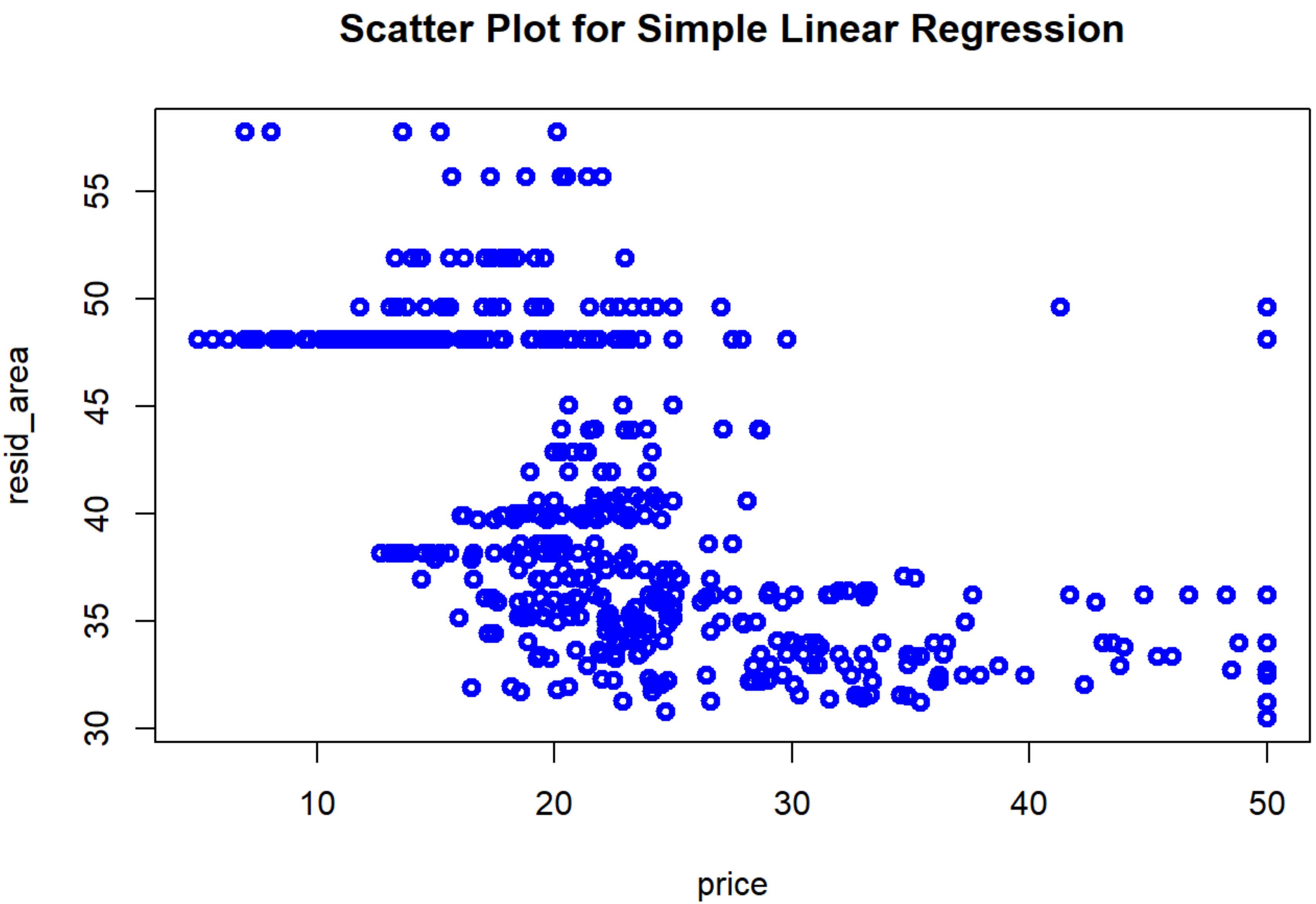
```
#install.packages("visdat")
library(visdat)
```

```
## Warning: package 'visdat' was built under R version 4.3.3
```

```
vis_miss(data)  # Visualize missing values
```


price (0%)
crime_rate (0%)
resid_area (0%)
air_qual (0%)
room_num (0%)
age (0%)
dist1 (0%)
dist2 (0%)
dist3 (0%)
dist4 (0%)
teachers (0%)
poor_prop (0%)
airport (0%)
n_hos_beds (0%)
n_hot_rooms (0%)
waterbody (0%)
rainfall (0%)
bus_ter (0%)
Observations
0
100
200
300
400
500
Present (100%)

```
p=data$price
r=data$resid_area
### To extract mpg and wt variables from mtcars dataset.
plot(p,r, main="Scatter Plot for Simple Linear Regression", xlab="price", ylab=
"resid_area", col="blue", lwd=3) ### To make scatter plot.
```


Scatter Plot for Simple Linear Regression
resid_area
price
30
35
40
45
50
55
10
20
30
40
50

Select the Feature and target variable

```
# For example, using 'Avg.Area.Income' as a feature and 'Price' as the target variable
X <- data$resid_area  # Features
y <-data$Price             # Target variable
```

```
## Warning: Unknown or uninitialised column: `Price`.
```

Split the data

```
# Set seed for reproducibility
set.seed(42)

# Create a training set (80%) and test set (20%)
train_indices <- sample(1:nrow(data), size = 0.8 * nrow(data))
train_data <- data[train_indices, ]
test_data <- data[-train_indices, ]
```

```
# Check column names in your dataset
colnames(data)
```

```
##  [1] "price"       "crime_rate"  "resid_area"  "air_qual"    "room_num"
##  [6] "age"         "dist1"       "dist2"       "dist3"       "dist4"
## [11] "teachers"    "poor_prop"   "airport"     "n_hos_beds"  "n_hot_rooms"
## [16] "waterbody"   "rainfall"    "bus_ter"     "parks"
```

```
P=data$price
```

```
# Select only the columns 'price' and 'location'
df_selected <- data %>% select(price, resid_area)
head(df_selected)
```

```
## # A tibble: 6 × 2
##   price resid_area
##   <dbl>      <dbl>
## 1  24         32.3
## 2  21.6       37.1
## 3  34.7       37.1
## 4  33.4       32.2
## 5  36.2       32.2
## 6  28.7       32.2
```

Create a Linear Regression Model

```
# Fit a linear regression model
model <- lm(price ~resid_area , data = train_data)

# Display model summary
summary(model)
```

```
##
## Call:
## lm(formula = price ~ resid_area, data = train_data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -13.254  -4.864  -1.441   3.164  32.650
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  47.6235     2.4147   19.72   <2e-16 ***
## resid_area   -0.6106     0.0575  -10.62   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.888 on 402 degrees of freedom
## Multiple R-squared:  0.2191, Adjusted R-squared:  0.2171
## F-statistic: 112.8 on 1 and 402 DF,  p-value: < 2.2e-16
```

Make Predictions

```
# Make predictions on the test set
predictions <- predict(model, newdata = test_data)
```

Evaluating the model performence

```
# Calculate Mean Squared Error (MSE)
mse <- mean((test_data$Price - predictions)^2)
```

```
## Warning: Unknown or uninitialised column: `Price`.
```

```
# Calculate R-squared value
rsq <- summary(model)$r.squared

cat("Mean Squared Error:", mse, "\n")
```

```
## Mean Squared Error: NaN
```

```
cat("R-squared:", rsq, "\n")
```

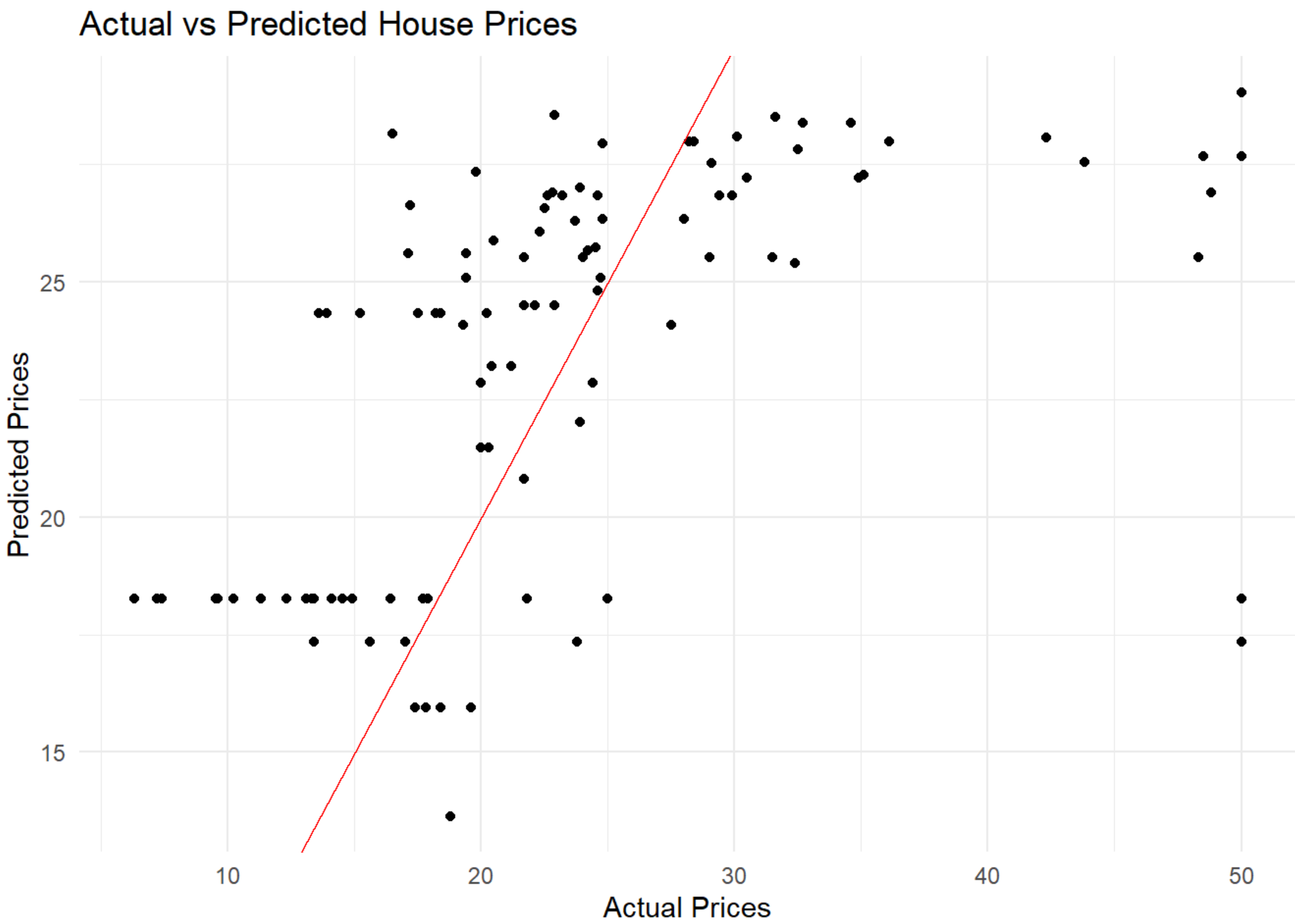
```
## R-squared: 0.2190632
```

To visualize Results

```
#install.packages("ggplot2")
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 4.3.3
```

```
#  a scatter plot of actual vs predicted prices
ggplot(data = test_data, aes(x = price, y = predictions)) +
  geom_point() +
  geom_abline(slope = 1, intercept = 0, color = "red") + # Line of perfect prediction
  labs(title = "Actual vs Predicted House Prices",
       x = "Actual Prices",
       y = "Predicted Prices") +
  theme_minimal()
```



Actual vs Predicted House Prices
Predicted Prices
25
20
15
10
20
30
40
50
Actual Prices

here, mse and r squre is very low , linear regression does not fit well, consider using other modeling techniques such as polynomial regression, decision trees, or ensemble methods.

Naive Bayes Classifier for Titanic Dataset

Lab3

2024-11-24

To install the package

```
#install.packages("e1071")   # Install e1071 package
library(e1071) # For Naive Bayes
```

```
## Warning: package 'e1071' was built under R version 4.3.3
```

```
library(dplyr) # For data manipulation
```

```
## Warning: package 'dplyr' was built under R version 4.3.3
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

To load the data of Titanic

```
# Load your Titanic dataset
titanic_data <- read.csv("D:\\sandip sir 3rd sem lab\\titanic_dataset.csv")

# View the first few rows of the dataset
head(titanic_data)
```

```
##   survived pclass    sex age sibsp parch    fare embarked class   who
## 1        0      3   male  22     1     0  7.2500        S Third   man
## 2        1      1 female  38     1     0 71.2833        C First woman
## 3        1      3 female  26     0     0  7.9250        S Third woman
## 4        1      1 female  35     1     0 53.1000        S First woman
## 5        0      3   male  35     0     0  8.0500        S Third   man
## 6        0      3   male  NA     0     0  8.4583        Q Third   man
##   adult_male deck embark_town alive alone
## 1       True      Southampton    no False
## 2      False    C   Cherbourg   yes False
## 3      False      Southampton   yes  True
## 4      False    C Southampton   yes False
## 5       True      Southampton    no  True
## 6       True       Queenstown    no  True
```

Now data preprocessing before the training the data

Here first deal with convert categorical variable to factors

```
str(titanic_data)
```

```
## 'data.frame':    891 obs. of  15 variables:
##  $ survived   : int  0 1 1 1 0 0 0 0 1 1 ...
##  $ pclass     : int  3 1 3 1 3 3 1 3 3 2 ...
##  $ sex        : chr  "male" "female" "female" "female" ...
##  $ age        : num  22 38 26 35 35 NA 54 2 27 14 ...
##  $ sibsp      : int  1 1 0 1 0 0 0 3 0 1 ...
##  $ parch      : int  0 0 0 0 0 0 0 1 2 0 ...
##  $ fare       : num  7.25 71.28 7.92 53.1 8.05 ...
##  $ embarked   : chr  "S" "C" "S" "S" ...
##  $ class      : chr  "Third" "First" "Third" "First" ...
##  $ who        : chr  "man" "woman" "woman" "woman" ...
##  $ adult_male : chr  "True" "False" "False" "False" ...
##  $ deck       : chr  "" "C" "" "C" ...
##  $ embark_town: chr  "Southampton" "Cherbourg" "Southampton" "Southampton" ...
##  $ alive      : chr  "no" "yes" "yes" "yes" ...
##  $ alone      : chr  "False" "False" "True" "False" ...
```

```
colnames(titanic_data)
```

```
##  [1] "survived"    "pclass"      "sex"         "age"         "sibsp"
##  [6] "parch"       "fare"        "embarked"    "class"       "who"
## [11] "adult_male"  "deck"        "embark_town" "alive"       "alone"
```

```
# Open a spreadsheet-like view of the data
#View(titanic_data)
```

```
# Convert 'Survived', 'Sex', and 'Pclass' etc to factors
titanic_data$Survived <- as.factor(titanic_data$survived)
titanic_data$Sex <- as.factor(titanic_data$sex)
titanic_data$embarked <- as.factor(titanic_data$embarked)
titanic_data$class <- as.factor(titanic_data$class)
titanic_data$who <- as.factor(titanic_data$who)
titanic_data$embark_town <- as.factor(titanic_data$embark_town)
titanic_data$alive <- as.factor(titanic_data$alive)
titanic_data$alone <- as.factor(titanic_data$alone)
```

```
str(titanic_data)
```

```
## 'data.frame':    891 obs. of  17 variables:
##  $ survived   : int  0 1 1 1 0 0 0 0 1 1 ...
##  $ pclass     : int  3 1 3 1 3 3 1 3 3 2 ...
##  $ sex        : chr  "male" "female" "female" "female" ...
##  $ age        : num  22 38 26 35 35 NA 54 2 27 14 ...
##  $ sibsp      : int  1 1 0 1 0 0 0 3 0 1 ...
##  $ parch      : int  0 0 0 0 0 0 0 1 2 0 ...
##  $ fare       : num  7.25 71.28 7.92 53.1 8.05 ...
##  $ embarked   : Factor w/ 4 levels "","C","Q","S": 4 2 4 4 4 3 4 4 4 2 ...
##  $ class      : Factor w/ 3 levels "First","Second",..: 3 1 3 1 3 3 1 3 3 2 ...
##  $ who        : Factor w/ 3 levels "child","man",..: 2 3 3 3 2 2 2 1 3 1 ...
##  $ adult_male : chr  "True" "False" "False" "False" ...
##  $ deck       : chr  "" "C" "" "C" ...
##  $ embark_town: Factor w/ 4 levels "","Cherbourg",..: 4 2 4 4 4 3 4 4 4 2 ...
##  $ alive      : Factor w/ 2 levels "no","yes": 1 2 2 2 1 1 1 1 2 2 ...
##  $ alone      : Factor w/ 2 levels "False","True": 1 1 2 1 2 2 2 1 1 1 ...
##  $ Survived   : Factor w/ 2 levels "0","1": 1 2 2 2 1 1 1 1 2 2 ...
##  $ Sex        : Factor w/ 2 levels "female","male": 2 1 1 1 2 2 2 2 1 1 ...
```

```
# View levels of some factors
levels(titanic_data$sex)
```

```
## NULL
```

```
levels(titanic_data$pclass)
```

```
## NULL
```

To Handle the missing value

```
# Check for NA values
sum(is.na(titanic_data))     # for count the total sum of missing value
```

```
## [1] 177
```

```
colSums(is.na(titanic_data)) # for count for each column
```

```
##    survived      pclass         sex         age       sibsp       parch 
##           0           0           0         177           0           0 
##        fare    embarked       class         who  adult_male        deck 
##           0           0           0           0           0           0 
## embark_town       alive       alone    Survived         Sex 
##           0           0           0           0           0
```

here is the missing value in the only age i.e.177

Then fill the missing value by its means value

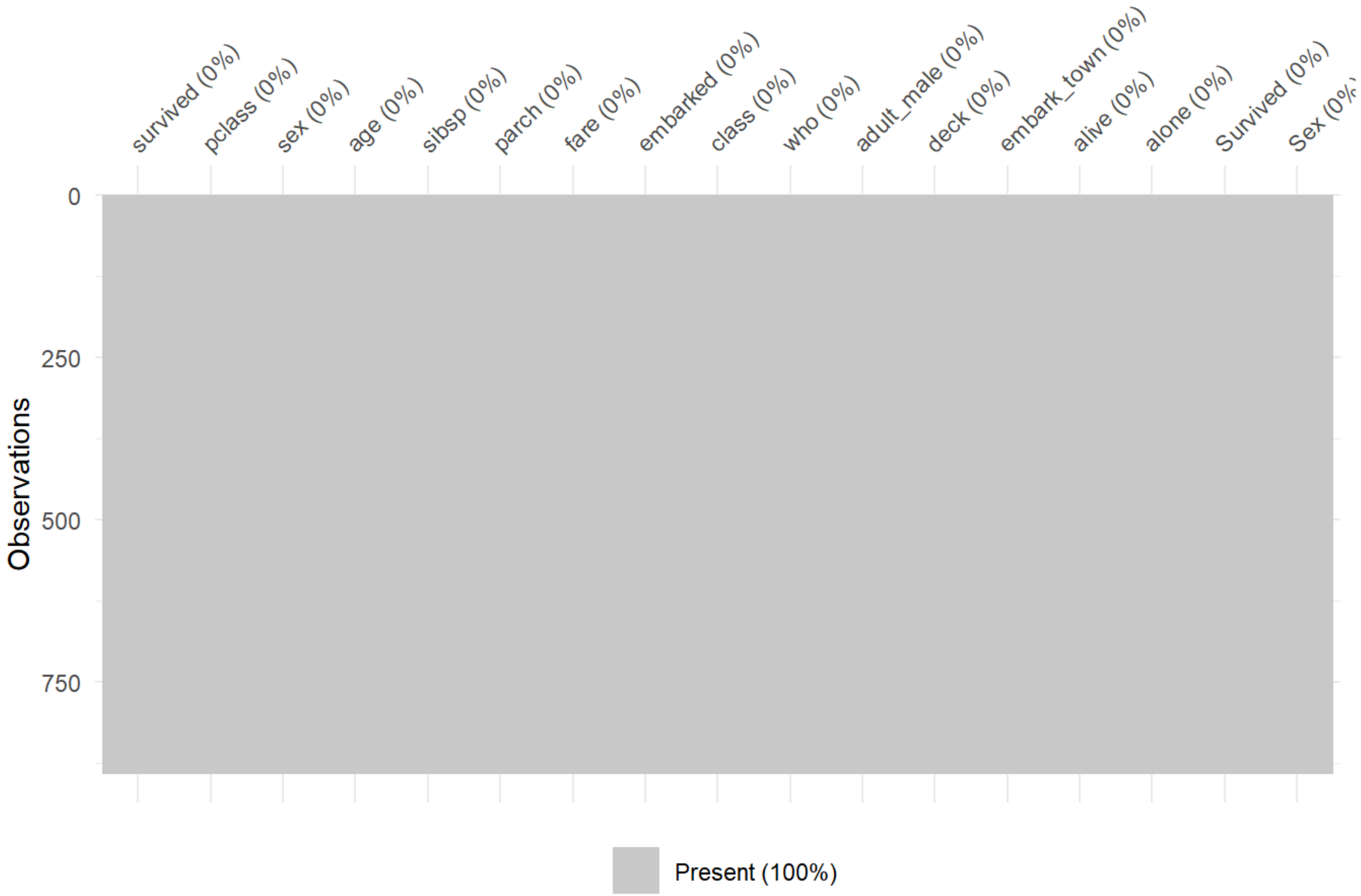
```
library(dplyr)
titanic_data <- titanic_data %>%
    mutate( age = ifelse(is.na( age), mean( age, na.rm = TRUE),  age))
```

Visualizing Missing data

```
#install.packages("visdat")
library(visdat)
```

```
## Warning: package 'visdat' was built under R version 4.3.3
```

```
vis_miss(titanic_data)  # Visualize missing values
```




```
sum(is.na(titanic_data))
```

```
## [1] 0
```

there is no any missing value present in the dataset

Now select the feature that is relevent for modeling

```
# Select relevant features for modeling
titanic_model_data <- titanic_data %>%
    select(survived, pclass,sex,age, fare,sibsp,embarked)

# Adjust features as necessary
# View cleaned data
head(titanic_model_data)
```

```
##   survived pclass    sex      age    fare sibsp embarked
## 1        0      3   male 22.00000  7.2500     1        S
## 2        1      1 female 38.00000 71.2833     1        C
## 3        1      3 female 26.00000  7.9250     0        S
## 4        1      1 female 35.00000 53.1000     1        S
## 5        0      3   male 35.00000  8.0500     0        S
## 6        0      3   male 29.69912  8.4583     0        Q
```

Split the Data into Traing and Testing sets

```
set.seed(123)  # For reproducibility
train_indices <- sample(1:nrow(titanic_model_data), size = 0.8 * nrow(titanic_model_data))
train_data <- titanic_model_data[train_indices, ]
test_data <- titanic_model_data[-train_indices, ]
```

Train the Naive Bayes Model

```
# Train the Naïve Bayes model
naive_bayes_model <- naiveBayes(survived ~ ., data = train_data)

# Print model summary
print(naive_bayes_model)
```

```
##
## Naive Bayes Classifier for Discrete Predictors
##
## Call:
## naiveBayes.default(x = X, y = Y, laplace = laplace)
##
## A-priori probabilities:
## Y
##        0        1
## 0.616573 0.383427
##
## Conditional probabilities:
##    pclass
## Y       [,1]      [,2]
##   0 2.517084 0.7488528
##   1 1.956044 0.8733651
##
##    sex
## Y      female      male
##   0 0.1457859 0.8542141
##   1 0.6849817 0.3150183
##
##    age
## Y       [,1]     [,2]
##   0 30.49592 12.24763
##   1 28.86927 13.49467
##
##    fare
```

```
## Y       [,1]     [,2]
##   0 22.79986 31.88659
##   1 48.47419 69.36519
##
##    sibsp
## Y        [,1]      [,2]
##   0 0.5535308 1.2354099
##   1 0.4798535 0.7228904
##
##    embarked
## Y                      C          Q          S
##   0 0.00000000 0.13667426 0.08656036 0.77676538
##   1 0.00000000 0.29304029 0.09890110 0.60805861
```

Make Prediction

```
# Make predictions on test data
predictions <- predict(naive_bayes_model, newdata = test_data)

# View predictions
head(predictions)
```

```
## [1] 0 0 0 1 0 0
## Levels: 0 1
```

Model Evaluation

```
# Create a confusion matrix
confusion_matrix <- table(test_data$survived, predictions)

# Print confusion matrix
print(confusion_matrix)
```

```
##    predictions
##      0  1
##   0 98 12
##   1 28 41
```

```
# Calculate accuracy
accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
cat("Accuracy:", accuracy * 100, "%\n")
```

```
## Accuracy: 77.65363 %
```

Accuracy is 77.65363% reliable performance.

Lab 3

Naive bayes classifier

Simplicity: Easy to understand and implement, making it accessible for beginners.

Speed: Fast training and prediction times, suitable for large datasets.

High-Dimensional Data: Performs well in high-dimensional spaces, like text classification.

Robustness: Handles irrelevant features effectively, often yielding good results in practice.

Probabilistic Output: Provides probabilities of class membership, allowing for better interpretation of results.

Generative Model: Learns the distribution of features within each class, useful for generating new data points.

In [2]:

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
```

In [3]:

```
# Load the dataset
df = pd.read_csv("D:\\sandip sir 3rd sem lab\\titanic_dataset.csv")
print(df)
```

```
     survived  pclass     sex   age  sibsp  parch     fare embarked   class  \
0           0       3    male  22.0      1      0   7.2500        S   Third
1           1       1  female  38.0      1      0  71.2833        C   First
2           1       3  female  26.0      0      0   7.9250        S   Third
3           1       1  female  35.0      1      0  53.1000        S   First
4           0       3    male  35.0      0      0   8.0500        S   Third
..        ...     ...     ...   ...    ...    ...      ...      ...     ...
886         0       2    male  27.0      0      0  13.0000        S  Second
887         1       1  female  19.0      0      0  30.0000        S   First
888         0       3  female   NaN      1      2  23.4500        S   Third
889         1       1    male  26.0      0      0  30.0000        C   First
890         0       3    male  32.0      0      0   7.7500        Q   Third

       who  adult_male deck  embark_town alive  alone
0      man        True  NaN  Southampton    no  False
1    woman       False    C    Cherbourg   yes  False
2    woman       False  NaN  Southampton   yes   True
3    woman       False    C  Southampton   yes  False
4      man        True  NaN  Southampton    no   True
..     ...         ...  ...          ...   ...    ...
886    man        True  NaN  Southampton    no   True
887  woman       False    B  Southampton   yes   True
888  woman       False  NaN  Southampton    no  False
889    man        True    C    Cherbourg   yes   True
890    man        True  NaN   Queenstown    no   True

[891 rows x 15 columns]
```

In [6]:

```
# Encode categorical variables
#fit_transform() combines two steps:
#it fits the encoder to the unique values in the 'sex' column and transforms those values into numerical label
#For example, 'male' might be encoded as 1 and 'female' as 0.
label_encoder = LabelEncoder()
df['sex'] = label_encoder.fit_transform(df['sex'])
print(df['sex'])
# This encodes the unique values in the 'embarked' column ( 'C', 'Q', 'S') into numerical labels.
df['embarked'] = label_encoder.fit_transform(df['embarked'])
print(df['embarked'])
```

```
0      1
1      0
2      0
3      0
4      1
      ..
886    1
887    0
888    0
889    1
890    1
Name: sex, Length: 891, dtype: int64
0      2
1      0
2      2
3      2
4      2
      ..
886    2
887    2
888    2
889    0
890    1
Name: embarked, Length: 891, dtype: int32
```

In [7]:
```
#for cheacking the missing value
df.isnull().sum()
```

Out[7]:
```
survived         0
pclass           0
sex              0
age            177
sibsp            0
parch            0
fare             0
embarked         0
class            0
who              0
adult_male       0
deck           688
embark_town      2
alive            0
alone            0
dtype: int64
```

In [12]:
```
# Fill missing values
print(df['age'])
x=df['age'].fillna(df['age'].mean(), inplace=True)
x
print(df['deck'])
y=df['deck'].fillna('Unknown', inplace=True)
y
```

```
0      22.000000
1      38.000000
2      26.000000
3      35.000000
4      35.000000
         ...
886    27.000000
887    19.000000
888    29.699118
889    26.000000
890    32.000000
Name: age, Length: 891, dtype: float64
0      Unknown
1            C
2      Unknown
3            C
4      Unknown
        ...
886    Unknown
887          B
888    Unknown
889          C
890    Unknown
Name: deck, Length: 891, dtype: object
```

In [28]:
```
print(df)
print(df.columns)
print(df['deck'])
```

```
     survived  pclass  sex        age  sibsp  parch     fare  embarked  \
0           0       3    1  22.000000      1      0   7.2500         2
1           1       1    0  38.000000      1      0  71.2833         0
2           1       3    0  26.000000      0      0   7.9250         2
3           1       1    0  35.000000      1      0  53.1000         2
4           0       3    1  35.000000      0      0   8.0500         2
..        ...     ...  ...        ...    ...    ...      ...       ...
886         0       2    1  27.000000      0      0  13.0000         2
887         1       1    0  19.000000      0      0  30.0000         2
888         0       3    0  29.699118      1      2  23.4500         2
889         1       1    1  26.000000      0      0  30.0000         0
890         0       3    1  32.000000      0      0   7.7500         1

        deck  embark_town
0    Unknown  Southampton
1          C    Cherbourg
2    Unknown  Southampton
3          C  Southampton
4    Unknown  Southampton
..       ...          ...
886  Unknown  Southampton
887        B  Southampton
888  Unknown  Southampton
889        C    Cherbourg
890  Unknown   Queenstown

[891 rows x 10 columns]
Index(['survived', 'pclass', 'sex', 'age', 'sibsp', 'parch', 'fare',
       'embarked', 'deck', 'embark_town'],
      dtype='object')
0      Unknown
1            C
2      Unknown
3            C
4      Unknown
        ...
886    Unknown
887          B
888    Unknown
889          C
890    Unknown
Name: deck, Length: 891, dtype: object
```

In [30]:

```python
# Select features and target variable
X = df[['pclass', 'sex', 'age', 'fare', 'embarked']]
print(X)
y = df['survived']
print(y)
```

```
     pclass  sex        age     fare  embarked
0         3    1  22.000000   7.2500         2
1         1    0  38.000000  71.2833         0
2         3    0  26.000000   7.9250         2
3         1    0  35.000000  53.1000         2
4         3    1  35.000000   8.0500         2
..      ...  ...        ...      ...       ...
886       2    1  27.000000  13.0000         2
887       1    0  19.000000  30.0000         2
888       3    0  29.699118  23.4500         2
889       1    1  26.000000  30.0000         0
890       3    1  32.000000   7.7500         1

[891 rows x 5 columns]
0      0
1      1
2      1
3      1
4      0
      ..
886    0
887    1
888    0
889    1
890    0
Name: survived, Length: 891, dtype: int64
```

In [35]:

```python
# Split the dataset into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
X_train, X_test, y_train, y_test
```

Out[35]:
```
(     pclass  sex        age      fare  embarked
 331       1    1  45.500000   28.5000         2
 733       2    1  23.000000   13.0000         2
 382       3    1  32.000000    7.9250         2
 704       3    1  26.000000    7.8542         2
 813       3    0   6.000000   31.2750         2
 ..      ...  ...        ...       ...       ...
 106       3    0  21.000000    7.6500         2
 270       1    1  29.699118   31.0000         2
 860       3    1  41.000000   14.1083         2
 435       1    0  14.000000  120.0000         2
 102       1    1  21.000000   77.2875         2

 [712 rows x 5 columns],
      pclass  sex        age     fare  embarked
 709       3    1  29.699118  15.2458         0
 439       2    1  31.000000  10.5000         2
 840       3    1  20.000000   7.9250         2
 720       2    0   6.000000  33.0000         2
 39        3    0  14.000000  11.2417         0
 ..      ...  ...        ...      ...       ...
 433       3    1  17.000000   7.1250         2
 773       3    1  29.699118   7.2250         0
 25        3    0  38.000000  31.3875         2
 84        2    0  17.000000  10.5000         2
 10        3    0   4.000000  16.7000         2

 [179 rows x 5 columns],
 331    0
 733    0
 382    0
 704    0
 813    0
       ..
 106    1
 270    0
 860    0
 435    1
 102    0
 Name: survived, Length: 712, dtype: int64,
 709    1
 439    0
 840    0
 720    1
 39     1
       ..
 433    0
 773    0
 25     1
 84     1
 10     1
 Name: survived, Length: 179, dtype: int64)
```

In [36]:
```python
# Create and train the Naive Bayes classifier
#
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
model.fit(X_train, y_train)
```

Out[36]:
```
▾ GaussianNB
GaussianNB()
```

In [43]:
```python
# Make predictions
y_pred = model.predict(X_test)
y_pred
```

Out[43]:
```
array([0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1,
       0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1,
       1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0,
       0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1,
       0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0,
       1, 1, 1], dtype=int64)
```

In [40]:
```python
# Evaluate the model
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, y_pred)
```

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

```
0.7932960893854749
Accuracy: 0.79
```

In [45]:
```python
#Did Women Have a Higher Chance of Survival Compared to Men?
import pandas as pd
# Group by 'sex' and calculate the survival rate
survival_rate_by_sex = df.groupby('sex')['survived'].mean()

# Print the survival rates
print(survival_rate_by_sex)
# female has high survival rate as compared man
```

```
sex
0    0.742038
1    0.188908
Name: survived, dtype: float64
```

here,0(female): 74.20% of females in the dataset survived.

1(male): The survival rate is 0.18% indicating about 18.9% of males in the dataset survived.

In [51]:
```python
# Did First-Class Passengers Have a Better Survival Rate Than Those in Lower Classes?
# Group by 'pclass' and calculate the survival rate
survival_rate_by_class = df.groupby('pclass')['survived'].mean()

# Print the survival rates
print(survival_rate_by_class)
```

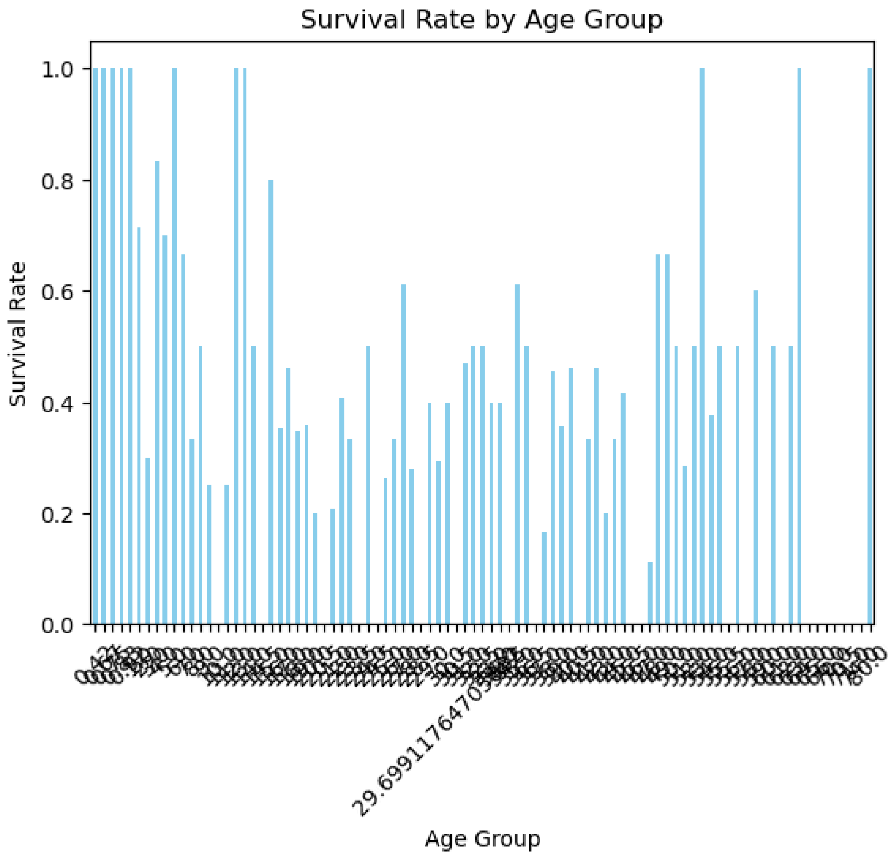
```
pclass
1    0.629630
2    0.472826
3    0.242363
Name: survived, dtype: float64
```

survival rate of first class is higher than survival rate of other two

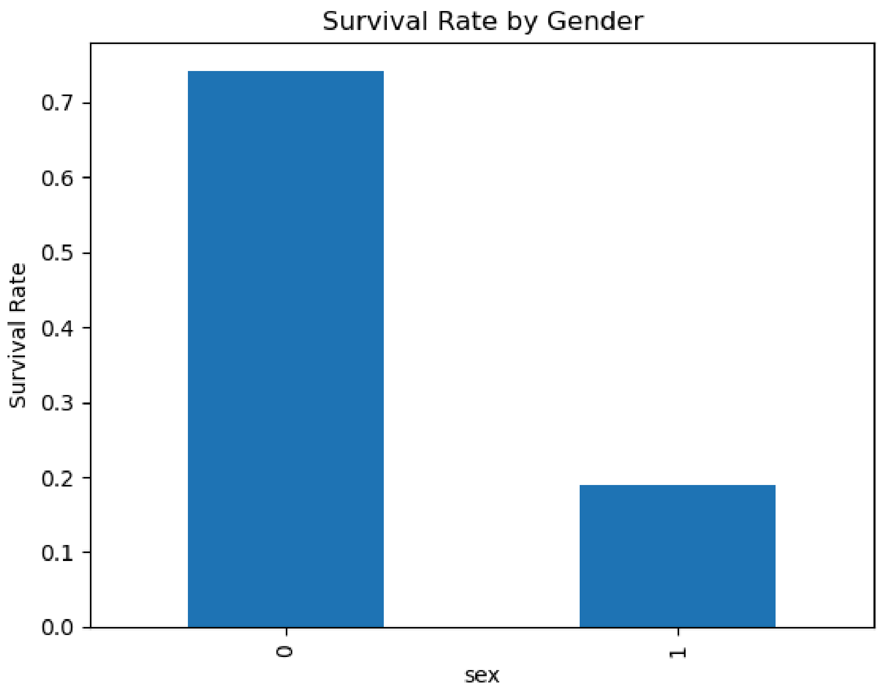
In [55]:
```python
# Group by 'age' and calculate the survival rate
survival_rate_by_age_group = df.groupby('age')['survived'].mean()

# Print the survival rates
print(survival_rate_by_age_group)
```

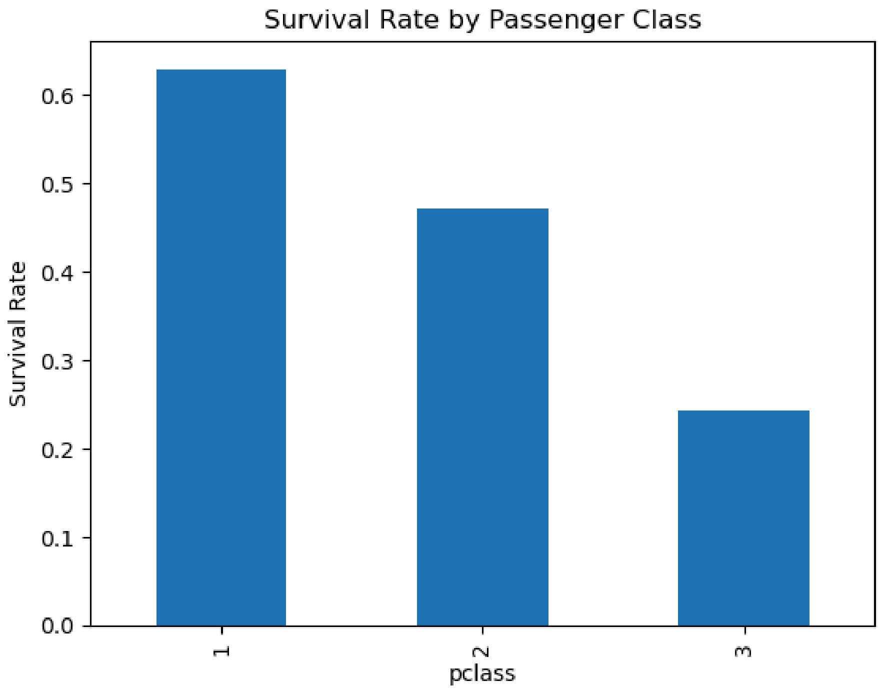
```
age
0.42     1.0
0.67     1.0
0.75     1.0
0.83     1.0
0.92     1.0
        ...
70.00    0.0
70.50    0.0
71.00    0.0
74.00    0.0
80.00    1.0
Name: survived, Length: 89, dtype: float64
```

In [56]:
```python
import matplotlib.pyplot as plt

# Plotting survival rates by age group
survival_rate_by_age_group.plot(kind='bar', title='Survival Rate by Age Group', color='skyblue')
plt.ylabel('Survival Rate')
plt.xlabel('Age Group')
plt.xticks(rotation=45)
plt.show()
```



In [57]:
```python
import matplotlib.pyplot as plt

# Plotting survival rates by sex
survival_rate_by_sex.plot(kind='bar', title='Survival Rate by Gender')
plt.ylabel('Survival Rate')
plt.show()

# Plotting survival rates by class
survival_rate_by_class.plot(kind='bar', title='Survival Rate by Passenger Class')
plt.ylabel('Survival Rate')
plt.show()
```




Survival Rate by Passenger Class
Survival Rate
0.0
0.1
0.2
0.3
0.4
0.5
0.6
1
2
3
pclass

Decision Tree

Decision trees are a type of supervised learning algorithm used for both classification and regression tasks.

Interpretability:

visually represent decisions and their possible consequences.

Non-linearity:

Unlike linear models, decision trees can capture non-linear relationships between features without requiring transformation.

Handling Mixed Data Types:

it can handle both categorical and numerical data, making them versatile for various datasets.

No Need for Feature Scaling:

Decision trees do not require normalization or standardization of features.

Applications:

Commonly used in fields like finance for credit scoring, healthcare for disease diagnosis, and marketing for customer segmentation.

Logistic Regression

Simplicity and Efficiency:

it a good choice for large datasets.

Interpretability:

Probabilistic Output:

useful for decision-making processes, such as threshold setting in classification tasks.

Robustness to Overfitting:

With appropriate regularization techniques, logistic regression can be less prone to overfitting compared to more complex models.

In [1]:
```
import numpy as np     # Linear algebra purpose
import pandas as pd # Data processing,CSV file
from sklearn.metrics import confusion_matrix,roc_curve,classification_report,
accuracy_score,auc,roc_auc_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier,export_graphviz
from sklearn.naive_bayes import GaussianNB
```

In [2]:
```
iris=load_iris()  #from  loading the iris data
```

In [3]:
```
print(iris.DESCR) # Details description
```

```
.. _iris_dataset:

Iris plants dataset
--------------------

**Data Set Characteristics:**

    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica

    :Summary Statistics:

    ============== ==== ==== ======= ===== ====================
                    Min  Max   Mean    SD   Class Correlation
    ============== ==== ==== ======= ===== ====================
    sepal length:   4.3  7.9   5.84   0.83    0.7826
    sepal width:    2.0  4.4   3.05   0.43   -0.4194
    petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
    petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)
    ============== ==== ==== ======= ===== ====================

    :Missing Attribute Values: None
    :Class Distribution: 33.3% for each of 3 classes.
    :Creator: R.A. Fisher
    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
    :Date: July, 1988

The famous Iris database, first used by Sir R.A. Fisher. The dataset is taken
from Fisher's paper. Note that it's the same as in R, but not as in the UCI
Machine Learning Repository, which has two wrong data points.

This is perhaps the best known database to be found in the
pattern recognition literature.  Fisher's paper is a classic in the field and
is referenced frequently to this day.  (See Duda & Hart, for example.)  The
data set contains 3 classes of 50 instances each, where each class refers to a
type of iris plant.  One class is linearly separable from the other 2; the
latter are NOT linearly separable from each other.

.. topic:: References

   - Fisher, R.A. "The use of multiple measurements in taxonomic problems"
     Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to
     Mathematical Statistics" (John Wiley, NY, 1950).
   - Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis.
     (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.
   - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System
     Structure and Classification Rule for Recognition in Partially Exposed
     Environments".  IEEE Transactions on Pattern Analysis and Machine
     Intelligence, Vol. PAMI-2, No. 1, 67-71.
   - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IEEE Transactions
     on Information Theory, May 1972, 431-433.
   - See also: 1988 MLC Proceedings, 54-64.  Cheeseman et al"s AUTOCLASS II
     conceptual clustering system finds 3 classes in the data.
   - Many, many more ...
```

In [4]:
```
X=iris.data       # To select the feature variable of the data
X.shape
```

Out[4]:
```
(150, 4)
```

In [5]:
```
y=iris.target   # To selcect the target variable of the data i.e. classification (0,1)
y.shape
```

Out[5]:
```
(150,)
```

In [6]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y, train_size=0.80,random_state=42)
```

In [7]:
```
print(len(X_train),len(X_test),len(y_train),len(y_test)) # To find the length
```

```
120 30 120 30
```

In [8]:
```
clf1=LogisticRegression()     #

# classification regression
clf2=DecisionTreeClassifier()
```

In [9]:
```
clf1.fit(X_train,y_train)
clf2.fit(X_train,y_train)
```

Out[9]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

In [10]:
```
y_pred1 = clf1.predict(X_test)
y_pred2 = clf2.predict(X_test)
```

In [11]:
```
print('Accuracy of Logistic Regression',accuracy_score(y_test,y_pred1))
print('Accuracy of Decision Trees',accuracy_score(y_test,y_pred2))
```

```
Accuracy of Logistic Regression 1.0
Accuracy of Decision Trees 1.0
```

In [12]:
```
pd.DataFrame(confusion_matrix(y_test,y_pred1),columns=list(range(0,3)))
```

Out[12]:

	0	1	2
0	10	0	0
1	0	9	0
2	0	0	11

In [13]:
```
pd.DataFrame(confusion_matrix(y_test,y_pred2),columns=list(range(0,3)))
```

Out[13]:

	0	1	2
0	10	0	0
1	0	9	0
2	0	0	11

Confusion just indicate the evaluate the performence of model to find the precision,recall,F1-score.

In [14]:
```
print('Logistic Regression',classification_report(y_test,y_pred1))
print('Decision Trees',classification_report(y_test,y_pred2))
```

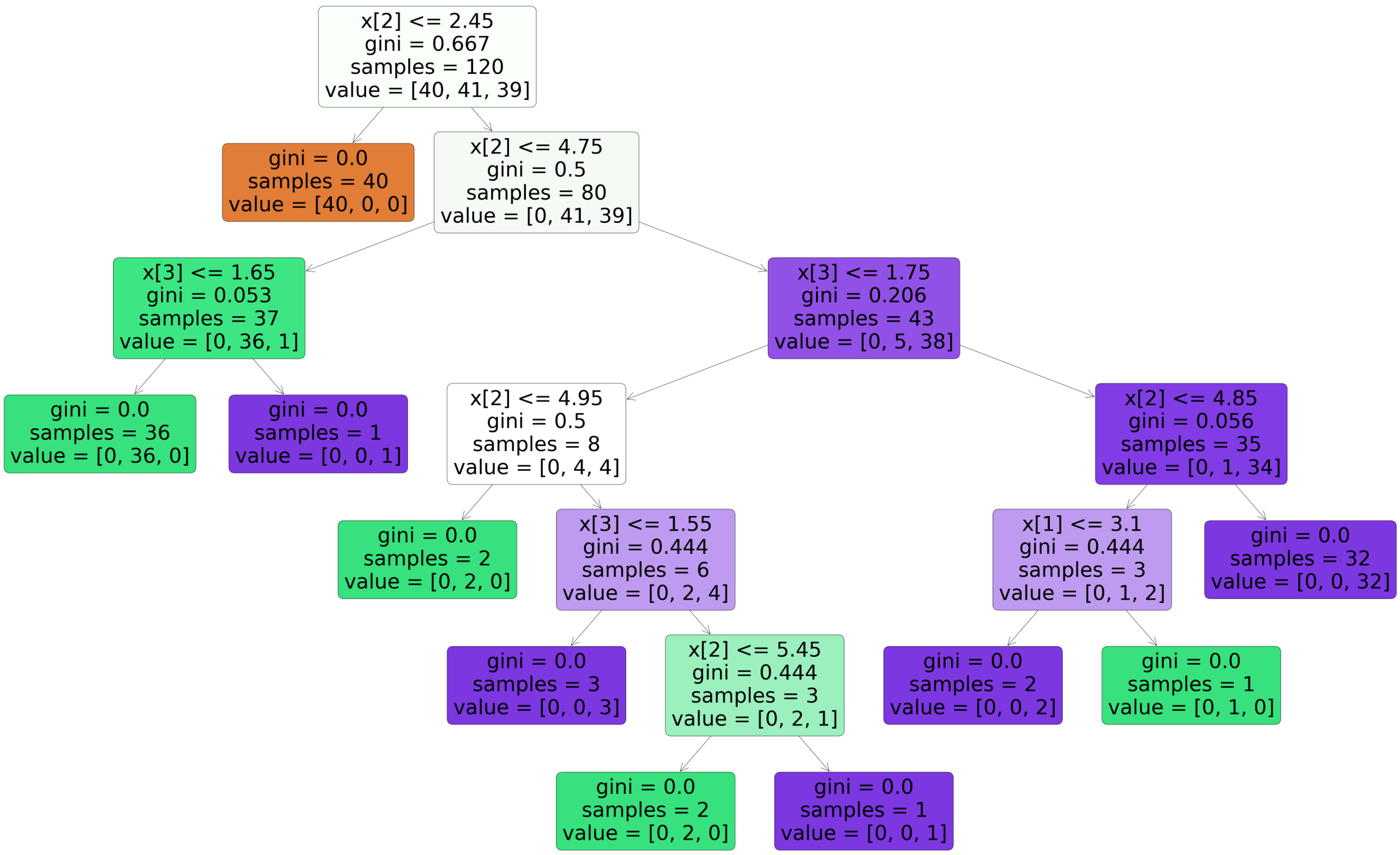
```
Logistic Regression               precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30

Decision Trees               precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

In [15]:
```
# Logistic Regression for probabilities
y_pred1 = clf1.predict_proba(X_test)

# Decision Trees for probabilities
y_pred2 = clf2.predict_proba(X_test)

# ROC AUC score calculation here we use multi_class for more than two classification of target value,where ovr=one to rest
print('Logistic Regression', roc_auc_score(y_test, y_pred1, multi_class='ovr'))
print('Decision Trees', roc_auc_score(y_test, y_pred2, multi_class='ovr'))
```

```
Logistic Regression 1.0
Decision Trees 1.0
```

In [16]:
```
from sklearn.tree import plot_tree
```

In [17]:
```
import matplotlib.pyplot as plt
from matplotlib.pylab import rcParams
rcParams['figure.figsize']= 80,50
plot_tree(clf2,filled=True,rounded=True)
plt.show()
```



$$G = 1 - \sum_{i=1}^{k} p_i^2$$

G: Represents the Gini impurity.

k: The number of classes in the dataset.

$$p_i$$

: The probability of an element belonging to class i.

if gini impurity indicate the how possibility of misclassification of data. The range of gini impurity is 0 to 1 then,0 indicate no misclassification,whereas 1 indicate the misclassification that for use to splitting.

Decision Tree for Iris data

Lab 5

2024-11-24

Syntax rpart(formula,data,method)

Parameters:

formula: indicates the formula based on which model has to be fitted

data: indicates the dataframe method: indicates the method to create decision tree. “anova” is used for regression and “class” is used as method for classification.

```
library(rpart)
```

```
## Warning: package 'rpart' was built under R version 4.3.3
```

```
library(rpart.plot)
```

```
## Warning: package 'rpart.plot' was built under R version 4.3.3
```

To load the iris data

```
data(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
```

```
str(iris)
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

Split the data into Training and Testing sets for model

```
library(caTools)
```

```
## Warning: package 'caTools' was built under R version 4.3.3
```

```
set.seed(42)  # For reproducibility
sample_split <- sample.split(iris$Species, SplitRatio = 0.8)
train_set <- subset(iris, sample_split == TRUE)
test_set <- subset(iris, sample_split == FALSE)
```

To build the decision tree model

```
# Build the regression tree model
model <- rpart(Species ~ .,  data = train_set,  method = "class")  # 'anova' is used for regression but
here use class for classification
model
```

```
## n= 120
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 120 80 setosa (0.33333333 0.33333333 0.33333333)
##   2) Petal.Length< 2.6 40  0 setosa (1.00000000 0.00000000 0.00000000) *
##   3) Petal.Length>=2.6 80 40 versicolor (0.00000000 0.50000000 0.50000000)
##     6) Petal.Width< 1.65 39  1 versicolor (0.00000000 0.97435897 0.02564103) *
##     7) Petal.Width>=1.65 41  2 virginica (0.00000000 0.04878049 0.95121951) *
```

Root Node (Node 1):

n = 120: The total number of samples in this node is 120. loss = 80: There are 80 misclassifications at this node. yval = setosa: The predicted class for this node is "setosa". yprob = (0.33333333, 0.33333333, 0.33333333): The probabilities for each class (setosa, versicolor, virginica) are approximately equal, suggesting uncertainty about which class is dominant at this root level.

Node 2: (Split based on Petal.Length)

Condition: Petal.Length < 2.6 n = 40: There are 40 samples in this node. loss = 0: No misclassifications; all samples are correctly classified. yval = setosa: All samples in this node are classified as "setosa". yprob = (1.00000000, 0.00000000, 0.00000000): This indicates that all samples in this node belong to the "setosa" class with certainty.

###Node 3: (Split based on Petal.Length)

Condition: Petal.Length >= 2.6 n = 80: There are 80 samples in this node. loss = 40: There are 40 misclassifications at this node. yval = versicolor: The predicted class is “versicolor”. yprob = (0.00000000, 0.50000000, 0.50000000): The probabilities indicate uncertainty; half of the samples are classified as “versicolor” and half as “virginica”.

Node 6: (Further split based on Petal.Width)

Condition: Petal.Width < 1.65 n = 39: There are 39 samples in this node. loss = 1: One misclassification occurs here. yval = versicolor: The predicted class is “versicolor”. yprob = (0.00000000, 0.97435897, 0.02564103): This suggests that nearly all samples are classified as “versicolor”, with a very small probability for “virginica”.

Node 7:

Condition: Petal.Width >= 1.65 n = 41: There are 41 samples in this node. loss = 2: Two misclassifications occur here. yval = virginica: The predicted class is “virginica”. yprob = (0.00000000, 0.04878049, 0.95121951): This indicates a strong prediction towards “virginica”, with a very low probability for “versicolor”.

```r
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.3.3
```

```
## Loading required package: ggplot2
```

```
## Warning: package 'ggplot2' was built under R version 4.3.3
```

```
## Loading required package: lattice
```

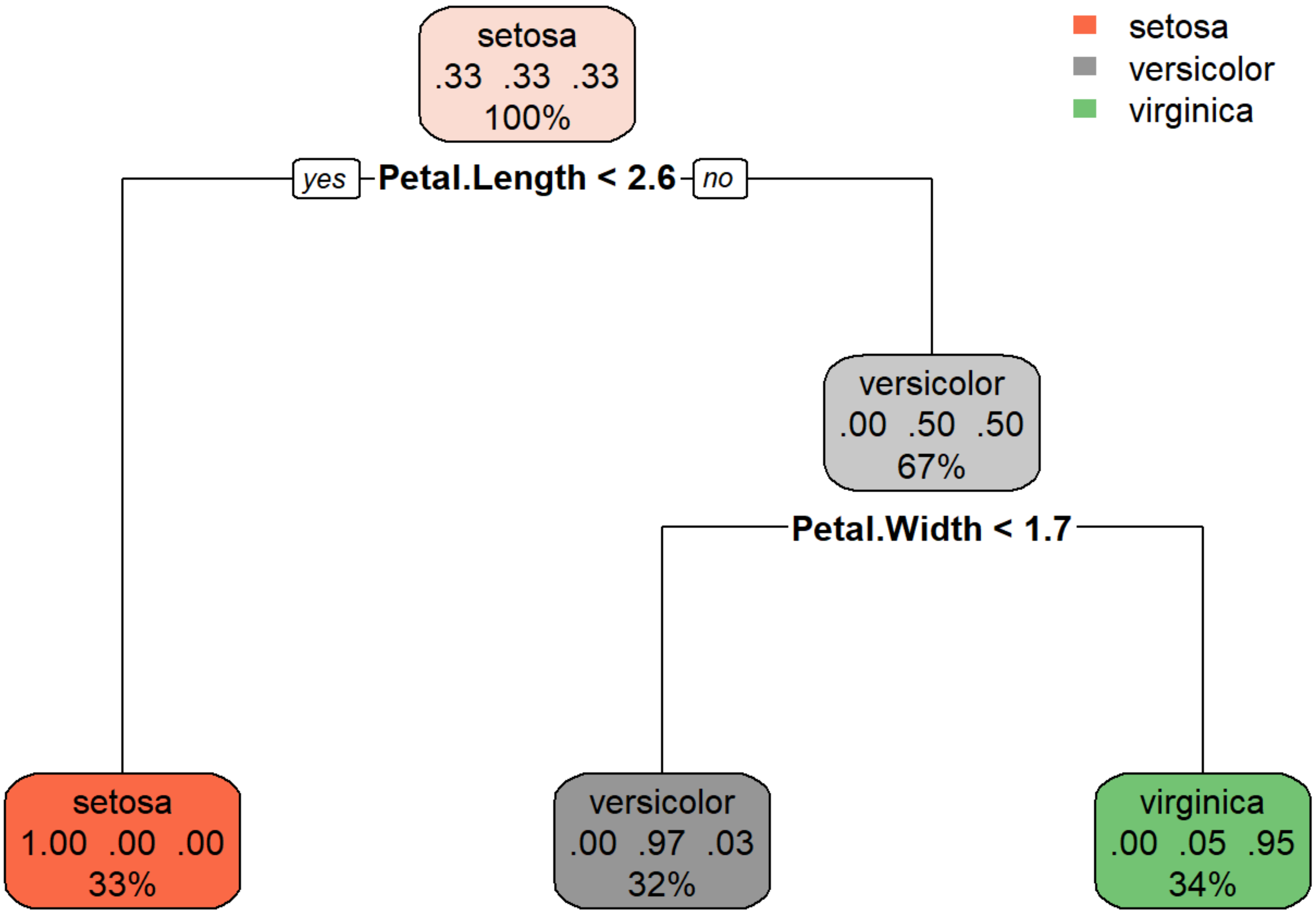
```
## Warning: package 'lattice' was built under R version 4.3.3
```

```
# Predict on the original data
predictions <- predict(model, newdata=test_set,type="class")
head(predictions)  # Show first few predictions
```

```
##      1      2     13     16     17     23
## setosa setosa setosa setosa setosa setosa
## Levels: setosa versicolor virginica
```

Plot the model prediction

```
rpart.plot(model)
```



Evaluate

the model performence

```
confusion_mat <- confusionMatrix(data = predictions, reference = test_set$Species)
print(confusion_mat)
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction   setosa versicolor virginica
##   setosa         10          0         0
##   versicolor      0         10         3
##   virginica       0          0         7
##
## Overall Statistics
##
##                Accuracy : 0.9
##                  95% CI : (0.7347, 0.9789)
##     No Information Rate : 0.3333
##     P-Value [Acc > NIR] : 1.665e-10
##
##                   Kappa : 0.85
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: setosa Class: versicolor Class: virginica
## Sensitivity                 1.0000            1.0000           0.7000
## Specificity                 1.0000            0.8500           1.0000
## Pos Pred Value              1.0000            0.7692           1.0000
## Neg Pred Value              1.0000            1.0000           0.8696
## Prevalence                  0.3333            0.3333           0.3333
## Detection Rate              0.3333            0.3333           0.2333
```

```
## Detection Prevalence        0.3333          0.4333          0.2333
## Balanced Accuracy           1.0000          0.9250          0.8500
```

```
#confusion_matrix(test_set$Species,predictions)
```

Conclusion

for setosa and versicolor species, with high accuracy and strong class-specific metrics like sensitivity and specificity. However, there is room for improvement in identifying virginica species accurately, as indicated by lower sensitivity and detection rates for this class.

`

Logistics regression for iris data

Lab 7

2024-11-24

```
rm(list=ls())
```

Some important library that use in

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.3.3
```

```
## Loading required package: ggplot2
```

```
## Warning: package 'ggplot2' was built under R version 4.3.3
```

```
## Loading required package: lattice
```

```
## Warning: package 'lattice' was built under R version 4.3.3
```

To load the dataset

```
data(iris)
```

To prepare the data

```
set.seed(123)   # use for reproducibility
index<-createDataPartition(iris$Species,p=0.8,list=FALSE) # for split the data 80% & 20%
train_d<-iris[index,] # for train
test_d<-iris[-index,]  # for test
```

For train the logistic model

```
library(nnet)
model<-multinom(Species~., data=train_d)
```

```
## # weights:  18 (10 variable)
## initial  value 131.833475
## iter  10 value 14.537939
## iter  20 value 4.489823
## iter  30 value 3.639148
## iter  40 value 2.956851
## iter  50 value 2.806269
## iter  60 value 2.496827
## iter  70 value 1.897817
## iter  80 value 1.726458
## iter  90 value 1.669028
## iter 100 value 1.630859
## final  value 1.630859
## stopped after 100 iterations
```

Make the predictions

```
# to predict the test data
preds<-predict(model,newdata=test_d)
```

To create the confusionMatrix

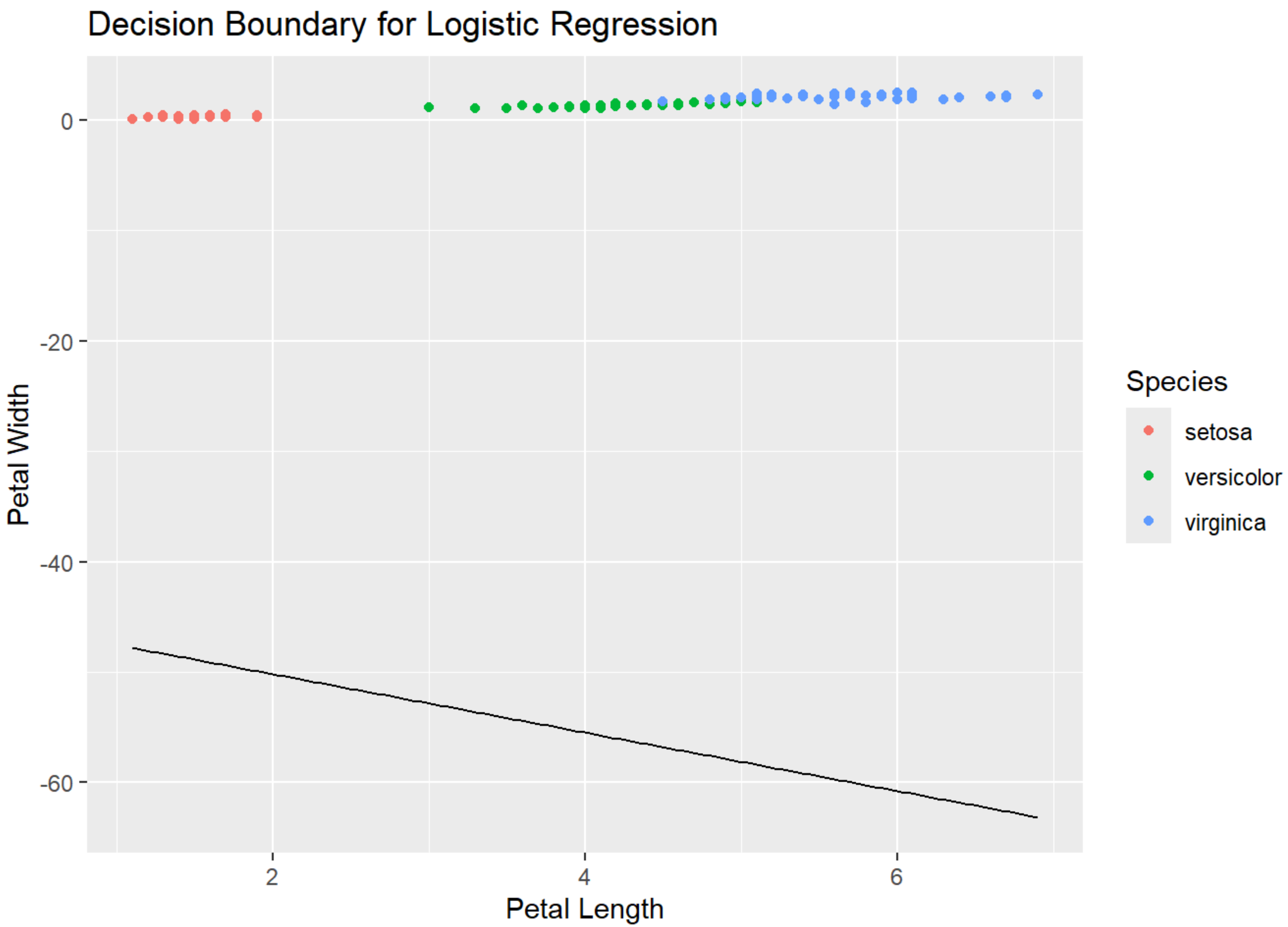
```
confusion_mat <- confusionMatrix(preds, test_d$Species)  # Generate confusion matrix
print(confusion_mat)  # Print confusion matrix and statistics
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction   setosa versicolor virginica
##   setosa         10          0         0
##   versicolor      0         10         2
##   virginica       0          0         8
##
## Overall Statistics
##
##                Accuracy : 0.9333
##                  95% CI : (0.7793, 0.9918)
##     No Information Rate : 0.3333
##     P-Value [Acc > NIR] : 8.747e-12
##
##                   Kappa : 0.9
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: setosa Class: versicolor Class: virginica
## Sensitivity                 1.0000            1.0000           0.8000
## Specificity                 1.0000            0.9000           1.0000
## Pos Pred Value              1.0000            0.8333           1.0000
## Neg Pred Value              1.0000            1.0000           0.9091
## Prevalence                  0.3333            0.3333           0.3333
## Detection Rate              0.3333            0.3333           0.2667
```

```
## Detection Prevalence         0.3333           0.4000           0.2667
## Balanced Accuracy            1.0000           0.9500           0.9000
```

To plot the curve

```
library(ggplot2)
# Plotting decision boundaries
ggplot(train_d, aes(x = Petal.Length, y = Petal.Width, color = Species)) +
  geom_point() +
  stat_function(fun = function(x) { -coef(model)[1] - coef(model)[2] * x / coef(model)[3] }, color = "b
lack") + labs(title = "Decision Boundary for Logistic Regression",
       x = "Petal Length",
       y = "Petal Width")
```



Decision Boundary for Logistic Regression
Petal Width
0
-20
-40
-60
2
4
6
Petal Length
Species
setosa
versicolor
virginica

KNN

Simplicity:

Easy to understand and implement, making it beginner-friendly.

No Training Phase:

Functions as a "lazy learner," storing the training data and making predictions on-the-fly.

Adaptability:

Easily accommodates new data without retraining.

Versatility:

Suitable for both classification and regression tasks.

Non-parametric:

Makes no assumptions about data distribution, allowing for broad applicability.

Effective for Non-linear Data:

Captures complex relationships and non-linear decision boundaries.

Customizable Distance Metrics:

Users can choose different distance measures based on specific needs.

In [1]:
```
import numpy as np
import pandas as pd
```

In [2]:
```
df=pd.read_csv("D:\\sandip sir 3rd sem lab\\breast_cancer_dataset.csv")
df.head()
```

Out[2]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	worst area	worst smoothness	worst compactness	worst concavity	worst concave points	worst symmetry	worst fractal dimension	target
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	17.33	184.60	2019.0	0.1622	0.6656	0.7119	0.2654	0.4601	0.11890	0
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	23.41	158.80	1956.0	0.1238	0.1866	0.2416	0.1860	0.2750	0.08902	0
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	25.53	152.50	1709.0	0.1444	0.4245	0.4504	0.2430	0.3613	0.08758	0
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	26.50	98.87	567.7	0.2098	0.8663	0.6869	0.2575	0.6638	0.17300	0
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	16.67	152.20	1575.0	0.1374	0.2050	0.4000	0.1625	0.2364	0.07678	0

5 rows × 31 columns

In [3]:
```
df.shape
```

Out[3]:
```
(569, 31)
```

In [4]:
```
df.isnull().sum()
```

Out[4]:
```
mean radius                0
mean texture               0
mean perimeter             0
mean area                  0
mean smoothness            0
mean compactness           0
mean concavity             0
mean concave points        0
mean symmetry              0
mean fractal dimension     0
radius error               0
texture error              0
perimeter error            0
area error                 0
smoothness error           0
compactness error          0
concavity error            0
concave points error       0
symmetry error             0
fractal dimension error    0
worst radius               0
worst texture              0
worst perimeter            0
worst area                 0
worst smoothness           0
worst compactness          0
worst concavity            0
worst concave points       0
worst symmetry             0
worst fractal dimension    0
target                     0
dtype: int64
```

In [5]:
```
X=df.iloc[:,0:30]
X.head()
```

Out[5]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	worst perimeter	worst area	worst smoothness	worst compactness	worst concavity	worst concave points	worst symmetry	worst fractal dimension
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	25.38	17.33	184.60	2019.0	0.1622	0.6656	0.7119	0.2654	0.4601	0.11890
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	24.99	23.41	158.80	1956.0	0.1238	0.1866	0.2416	0.1860	0.2750	0.08902
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	23.57	25.53	152.50	1709.0	0.1444	0.4245	0.4504	0.2430	0.3613	0.08758
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	14.91	26.50	98.87	567.7	0.2098	0.8663	0.6869	0.2575	0.6638	0.17300
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	22.54	16.67	152.20	1575.0	0.1374	0.2050	0.4000	0.1625	0.2364	0.07678

5 rows × 30 columns

In [6]:
```
y=df.iloc[:,-1]
y.head()
```

Out[6]:
```
0    0
1    0
2    0
3    0
4    0
Name: target, dtype: int64
```

In [7]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(df.iloc[:,0:30],df.iloc[:,-1],test_size=0.2,random_state=2)
```

In [8]:
```
print(X_train.shape,X_test.shape,y_train.shape,y_test.shape)
```

```
(455, 30) (114, 30) (455,) (114,)
```

In [9]:
```
from sklearn.preprocessing import StandardScaler
scaler= StandardScaler()
X_train=scaler.fit_transform(X_train)
X_test=scaler.transform(X_test)
```

In [10]:
```
X_train
```

Out[10]:
```
array([[-0.01330339,  1.7757658 , -0.01491962, ..., -0.13236958,
        -1.08014517, -0.03527943],
       [-0.8448276 , -0.6284278 , -0.87702746, ..., -1.11552632,
        -0.85773964, -0.72098905],
       [ 1.44755936,  0.71180168,  1.47428816, ...,  0.87583964,
         0.4967602 ,  0.46321706],
       ...,
       [-0.46608541, -1.49375484, -0.53234924, ..., -1.32388956,
        -1.02997851, -0.75145272],
       [-0.50025764, -1.62161319, -0.527814  , ..., -0.0987626 ,
         0.35796577, -0.43906159],
       [ 0.96060511,  1.21181916,  1.00427242, ...,  0.8956983 ,
        -1.23064515,  0.50697397]])
```

In [11]:
```
X_train.shape
```

Out[11]:
```
(455, 30)
```

In [12]:
```
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=3)
```

In [13]:
```
knn.fit(X_train,y_train)
```

Out[13]:
```
▾ KNeighborsClassifier
KNeighborsClassifier(n_neighbors=3)
```

In [14]:
```
from sklearn.metrics import accuracy_score ,classification_report
y_pred=knn.predict(X_test)
print(accuracy_score(y_pred,y_test))
print(classification_report(y_pred,y_test))
```

```
0.9912280701754386
              precision    recall  f1-score   support

           0       0.98      1.00      0.99        44
           1       1.00      0.99      0.99        70

    accuracy                           0.99       114
   macro avg       0.99      0.99      0.99       114
weighted avg       0.99      0.99      0.99       114
```

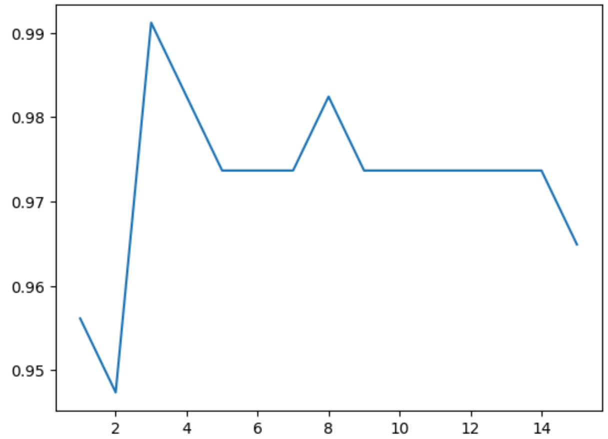
Precision is 0.98 it means 98% of instances predicted as class '0' were correct. for class 1 Precision is 1.00 indicating perfect precision all instances predicted as class '1' were correct.

Recall also known as sensitivity or True Positive Rate measures the ratio of true positive predictions to the total actual positives.here 1 indicates 0 correctly indentified. 0.99 indicating that 99% of actual instances of class '1' were correctly indentified. F1 scoring performing the 0.99 suggesting that the model performs well in balancing precision and recall.

these metrics classification model performs exceptionally well across both classes with high precision, recall, and F1-scores close to perfect (1.00). The accuracy rate further confirms that it effectively distinguishes between classes with minimal errors. This analysis suggests that model is robust and reliable for making predictions in this context.

In [15]:
```
# Initialize scores list
scores = []

for i in range (1,16):
    knn=KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train,y_train)
    y_pred=knn.predict(X_test)
    scores.append(accuracy_score(y_test,y_pred))
```

In [16]:
```
import matplotlib.pyplot as plt
plt.plot(range(1,16),scores)
plt.show()
```



Random forest

1. High accuracy Random Forest generally provides high accuracy in predictions due to its ensemble nature.

   it reduces the risk of overfitting that is common with single decision trees.

2. Robustness to Noise and Outliers predictions are based on the majority vote from multiple trees, the influence of any single noisy data point is minimized.

3. Handles High-Dimensional Data it is especially beneficial in fields like genomics or image processing where high-dimensional data is common. It automatically performs feature selection and can indicate the importance of different features in making predictions.

4. Non-Parametric Nature Random Forest does not assume any specific distribution for the data.

5. Easy to Use and Interpret it provides insights into feature importance, helping users understand which variables significantly impact predictions.

6. Parallel Processing Capability This property enhances computational efficiency, especially when dealing with large datasets.

7. Versatile Applications Random Forest is applicable across various domains, including finance (for credit scoring and fraud detection), healthcare (for disease diagnosis), and e-commerce (for customer behavior prediction). Its versatility makes it a go-to choice for many machine learning tasks.

8. Good Performance with Minimal Data Preprocessing Random Forest can handle missing values and does not require extensive preprocessing steps like normalization or one-hot encoding for categorical variables, simplifying the data preparation process

In [17]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

In [18]:
```
df=pd.read_csv("D:\\sandip sir 3rd sem lab\\breast_cancer_dataset.csv")
df.head()
```

Out[18]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	worst area	worst smoothness	worst compactness	worst concavity	worst concave points	worst symmetry	worst fractal dimension	target
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	17.33	184.60	2019.0	0.1622	0.6656	0.7119	0.2654	0.4601	0.11890	0
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	23.41	158.80	1956.0	0.1238	0.1866	0.2416	0.1860	0.2750	0.08902	0
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	25.53	152.50	1709.0	0.1444	0.4245	0.4504	0.2430	0.3613	0.08758	0
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	26.50	98.87	567.7	0.2098	0.8663	0.6869	0.2575	0.6638	0.17300	0
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	16.67	152.20	1575.0	0.1374	0.2050	0.4000	0.1625	0.2364	0.07678	0

5 rows × 31 columns

In [19]:
```
X=df.iloc[:,0:30]
X
```

Out[19]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	worst perimeter	worst area	worst smoothness	worst compactness	worst concavity	worst concave points	worst symmetry	worst fractal dimension
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419	0.07871	...	25.380	17.33	184.60	2019.0	0.16220	0.66560	0.7119	0.2654	0.4601	0.11890
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812	0.05667	...	24.990	23.41	158.80	1956.0	0.12380	0.18660	0.2416	0.1860	0.2750	0.08902
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069	0.05999	...	23.570	25.53	152.50	1709.0	0.14440	0.42450	0.4504	0.2430	0.3613	0.08758
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597	0.09744	...	14.910	26.50	98.87	567.7	0.20980	0.86630	0.6869	0.2575	0.6638	0.17300
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809	0.05883	...	22.540	16.67	152.20	1575.0	0.13740	0.20500	0.4000	0.1625	0.2364	0.07678
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	0.05623	...	25.450	26.40	166.10	2027.0	0.14100	0.21130	0.4107	0.2216	0.2060	0.07115
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	0.05533	...	23.690	38.25	155.00	1731.0	0.11660	0.19220	0.3215	0.1628	0.2572	0.06637
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	0.05648	...	18.980	34.12	126.70	1124.0	0.11390	0.30940	0.3403	0.1418	0.2218	0.07820
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	0.07016	...	25.740	39.42	184.60	1821.0	0.16500	0.86810	0.9387	0.2650	0.4087	0.12400
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	0.05884	...	9.456	30.37	59.16	268.6	0.08996	0.06444	0.0000	0.0000	0.2871	0.07039

569 rows × 30 columns

In [20]:

```
y=df.iloc[:,-1]
y
```

Out[20]:

```
0      0
1      0
2      0
3      0
4      0
      ..
564    0
565    0
566    0
567    0
568    1
Name: target, Length: 569, dtype: int64
```

In [21]:

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [22]:

```
# Initialize the Random Forest Classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
```

In [23]:

```
# Train the model
rf_classifier.fit(X_train, y_train)
```

Out[23]:

```
▾        RandomForestClassifier
RandomForestClassifier(random_state=42)
```

In [24]:

```
# Make predictions on the test set
y_pred = rf_classifier.predict(X_test)
```

In [25]:

```
# Evaluate the model's performance
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("\nClassification Report:")
print(classification_report(y_test, y_pred))

print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}")
```

```
Confusion Matrix:
[[40  3]
 [ 1 70]]

Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.93      0.95        43
           1       0.96      0.99      0.97        71

    accuracy                           0.96       114
   macro avg       0.97      0.96      0.96       114
weighted avg       0.97      0.96      0.96       114

Accuracy: 0.9649
```

its accuracy is 0.9649 its a better performence.

Random forest for KNN

Lab 6

2024-11-24

Load the library

```r
library(class)
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.3.3
```

```
## Loading required package: ggplot2
```

```
## Warning: package 'ggplot2' was built under R version 4.3.3
```

```
## Loading required package: lattice
```

```
## Warning: package 'lattice' was built under R version 4.3.3
```

```r
library(ggplot2)
```

To load the data breast cancer

```
breast_can_d<-read.csv("D:\\sandip sir 3rd sem lab\\breast_cancer_dataset.csv")
head(breast_can_d)
```

```
##   mean.radius mean.texture mean.perimeter mean.area mean.smoothness
## 1       17.99        10.38         122.80    1001.0         0.11840
## 2       20.57        17.77         132.90    1326.0         0.08474
## 3       19.69        21.25         130.00    1203.0         0.10960
## 4       11.42        20.38          77.58     386.1         0.14250
## 5       20.29        14.34         135.10    1297.0         0.10030
## 6       12.45        15.70          82.57     477.1         0.12780
##   mean.compactness mean.concavity mean.concave.points mean.symmetry
## 1          0.27760         0.3001             0.14710        0.2419
## 2          0.07864         0.0869             0.07017        0.1812
## 3          0.15990         0.1974             0.12790        0.2069
## 4          0.28390         0.2414             0.10520        0.2597
## 5          0.13280         0.1980             0.10430        0.1809
## 6          0.17000         0.1578             0.08089        0.2087
##   mean.fractal.dimension radius.error texture.error perimeter.error area.error
## 1                0.07871       1.0950        0.9053           8.589     153.40
## 2                0.05667       0.5435        0.7339           3.398      74.08
## 3                0.05999       0.7456        0.7869           4.585      94.03
## 4                0.09744       0.4956        1.1560           3.445      27.23
## 5                0.05883       0.7572        0.7813           5.438      94.44
## 6                0.07613       0.3345        0.8902           2.217      27.19
##   smoothness.error compactness.error concavity.error concave.points.error
## 1         0.006399           0.04904         0.05373              0.01587
## 2         0.005225           0.01308         0.01860              0.01340
## 3         0.006150           0.04006         0.03832              0.02058
## 4         0.009110           0.07458         0.05661              0.01867
## 5         0.011490           0.02461         0.05688              0.01885
## 6         0.007510           0.03345         0.03672              0.01137
```

```
##   symmetry.error fractal.dimension.error worst.radius worst.texture
## 1        0.03003                0.006193        25.38         17.33
## 2        0.01389                0.003532        24.99         23.41
## 3        0.02250                0.004571        23.57         25.53
## 4        0.05963                0.009208        14.91         26.50
## 5        0.01756                0.005115        22.54         16.67
## 6        0.02165                0.005082        15.47         23.75
##   worst.perimeter worst.area worst.smoothness worst.compactness worst.concavity
## 1          184.60     2019.0           0.1622            0.6656          0.7119
## 2          158.80     1956.0           0.1238            0.1866          0.2416
## 3          152.50     1709.0           0.1444            0.4245          0.4504
## 4           98.87      567.7           0.2098            0.8663          0.6869
## 5          152.20     1575.0           0.1374            0.2050          0.4000
## 6          103.40      741.6           0.1791            0.5249          0.5355
##   worst.concave.points worst.symmetry worst.fractal.dimension target
## 1               0.2654         0.4601                 0.11890      0
## 2               0.1860         0.2750                 0.08902      0
## 3               0.2430         0.3613                 0.08758      0
## 4               0.2575         0.6638                 0.17300      0
## 5               0.1625         0.2364                 0.07678      0
## 6               0.1741         0.3985                 0.12440      0
```

To see the data

```
str(breast_can_d)
```

```
## 'data.frame':	569 obs. of  31 variables:
##  $ mean.radius            : num  18 20.6 19.7 11.4 20.3 ...
##  $ mean.texture           : num  10.4 17.8 21.2 20.4 14.3 ...
##  $ mean.perimeter         : num  122.8 132.9 130 77.6 135.1 ...
##  $ mean.area              : num  1001 1326 1203 386 1297 ...
##  $ mean.smoothness        : num  0.1184 0.0847 0.1096 0.1425 0.1003 ...
##  $ mean.compactness       : num  0.2776 0.0786 0.1599 0.2839 0.1328 ...
##  $ mean.concavity         : num  0.3001 0.0869 0.1974 0.2414 0.198 ...
##  $ mean.concave.points    : num  0.1471 0.0702 0.1279 0.1052 0.1043 ...
##  $ mean.symmetry          : num  0.242 0.181 0.207 0.26 0.181 ...
##  $ mean.fractal.dimension : num  0.0787 0.0567 0.06 0.0974 0.0588 ...
##  $ radius.error           : num  1.095 0.543 0.746 0.496 0.757 ...
##  $ texture.error          : num  0.905 0.734 0.787 1.156 0.781 ...
##  $ perimeter.error        : num  8.59 3.4 4.58 3.44 5.44 ...
##  $ area.error             : num  153.4 74.1 94 27.2 94.4 ...
##  $ smoothness.error       : num  0.0064 0.00522 0.00615 0.00911 0.01149 ...
##  $ compactness.error      : num  0.049 0.0131 0.0401 0.0746 0.0246 ...
##  $ concavity.error        : num  0.0537 0.0186 0.0383 0.0566 0.0569 ...
##  $ concave.points.error   : num  0.0159 0.0134 0.0206 0.0187 0.0188 ...
##  $ symmetry.error         : num  0.03 0.0139 0.0225 0.0596 0.0176 ...
##  $ fractal.dimension.error: num  0.00619 0.00353 0.00457 0.00921 0.00511 ...
##  $ worst.radius           : num  25.4 25 23.6 14.9 22.5 ...
##  $ worst.texture          : num  17.3 23.4 25.5 26.5 16.7 ...
##  $ worst.perimeter        : num  184.6 158.8 152.5 98.9 152.2 ...
##  $ worst.area             : num  2019 1956 1709 568 1575 ...
##  $ worst.smoothness       : num  0.162 0.124 0.144 0.21 0.137 ...
##  $ worst.compactness      : num  0.666 0.187 0.424 0.866 0.205 ...
##  $ worst.concavity        : num  0.712 0.242 0.45 0.687 0.4 ...
```

```
##  $ worst.concave.points   : num  0.265 0.186 0.243 0.258 0.163 ...
##  $ worst.symmetry         : num  0.46 0.275 0.361 0.664 0.236 ...
##  $ worst.fractal.dimension: num  0.1189 0.089 0.0876 0.173 0.0768 ...
##  $ target                 : int  0 0 0 0 0 0 0 0 0 0 ...
```

To prepare the data

```
sum(is.na(breast_can_d))
```

```
## [1] 0
```

Split the Data into Training and Testing

```
set.seed(123)  # For reproducibility
index <- createDataPartition(breast_can_d$target, p = 0.7, list = FALSE)
train_data <- breast_can_d[index, ]
test_data <- breast_can_d[-index, ]

# Separate features and labels
train_labels <- train_data$target
test_labels <- test_data$target

train_data$target <- NULL  # Remove label column from training data
test_data$target <- NULL    # Remove label column from testing data
```

Train the KNN Model

```
k <- 5  # Number of neighbors
predictions <- knn(train = train_data, test = test_data, cl = train_labels, k = k)
```

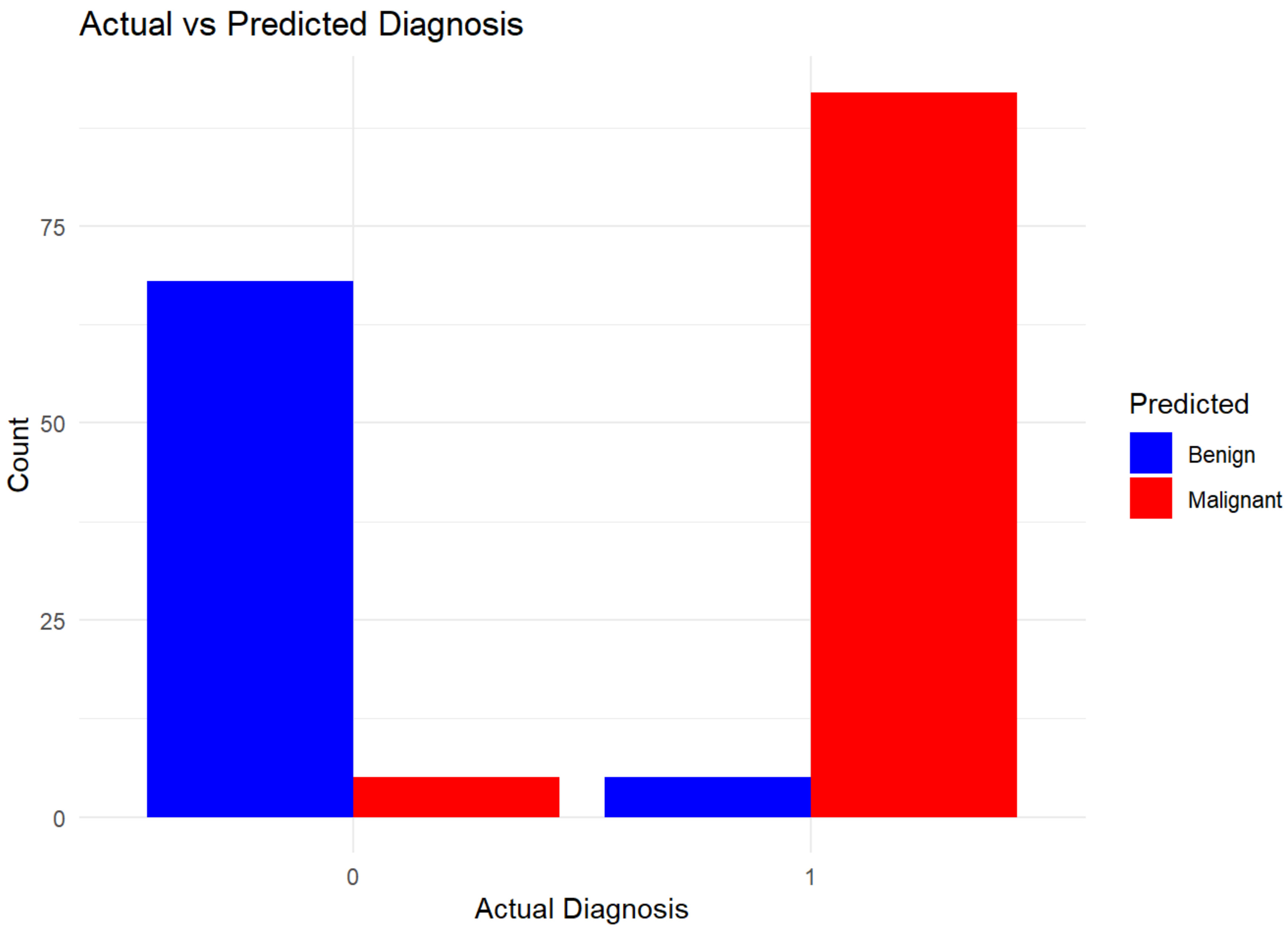
Evaluate the model performence

```
confusion_mat <- confusionMatrix(predictions, as.factor(test_labels))
print(confusion_mat)  # Print confusion matrix and statistics
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 68  5
##          1  5 92
##
##                Accuracy : 0.9412
##                  95% CI : (0.8945, 0.9714)
##     No Information Rate : 0.5706
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.88
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9315
##             Specificity : 0.9485
##          Pos Pred Value : 0.9315
##          Neg Pred Value : 0.9485
##              Prevalence : 0.4294
##          Detection Rate : 0.4000
##    Detection Prevalence : 0.4294
##       Balanced Accuracy : 0.9400
##
##        'Positive' Class : 0
##
```

Visualize Results

```
# Plotting actual vs predicted values for visual assessment
results_df <- data.frame(Actual = as.factor(test_labels), Predicted = predictions)

ggplot(results_df, aes(x = Actual, fill = Predicted)) +
  geom_bar(position = "dodge") +
  labs(title = "Actual vs Predicted Diagnosis",
       x = "Actual Diagnosis",
       y = "Count") +
  scale_fill_manual(values = c("blue", "red"), labels = c("Benign", "Malignant")) +
  theme_minimal()
```


Actual vs Predicted Diagnosis
Count
75
50
25
0
0
1
Actual Diagnosis
Predicted
Benign
Malignant

Random forest method for breast_cancer_dataset

Lab 7

2024-11-24

```
# Install randomForest package if not installed
#install.packages("randomForest")


# Load the library
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 4.3.3
```

```
## randomForest 4.7-1.2
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.3.3
```

```
## Loading required package: ggplot2
```

```
## Warning: package 'ggplot2' was built under R version 4.3.3
```

```
##
## Attaching package: 'ggplot2'
```

```
## The following object is masked from 'package:randomForest':
##
##     margin
```

```
## Loading required package: lattice
```

```
## Warning: package 'lattice' was built under R version 4.3.3
```

```
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 4.3.3
```

```
##
## Attaching package: 'dplyr'
```

```
## The following object is masked from 'package:randomForest':
##
##     combine
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
#install.packages("mlbench")
library(mlbench)
```

```
## Warning: package 'mlbench' was built under R version 4.3.3
```

To load Breast_cancer_dataset

```
setwd("D:\\sandip sir 3rd sem lab")
data=read.csv("breast_cancer_dataset.csv")
head(data)
```

```
##   mean.radius mean.texture mean.perimeter mean.area mean.smoothness
## 1       17.99        10.38         122.80    1001.0         0.11840
## 2       20.57        17.77         132.90    1326.0         0.08474
## 3       19.69        21.25         130.00    1203.0         0.10960
## 4       11.42        20.38          77.58     386.1         0.14250
## 5       20.29        14.34         135.10    1297.0         0.10030
## 6       12.45        15.70          82.57     477.1         0.12780
##   mean.compactness mean.concavity mean.concave.points mean.symmetry
## 1          0.27760         0.3001             0.14710        0.2419
## 2          0.07864         0.0869             0.07017        0.1812
## 3          0.15990         0.1974             0.12790        0.2069
## 4          0.28390         0.2414             0.10520        0.2597
## 5          0.13280         0.1980             0.10430        0.1809
## 6          0.17000         0.1578             0.08089        0.2087
##   mean.fractal.dimension radius.error texture.error perimeter.error area.error
## 1                0.07871       1.0950        0.9053           8.589     153.40
## 2                0.05667       0.5435        0.7339           3.398      74.08
## 3                0.05999       0.7456        0.7869           4.585      94.03
## 4                0.09744       0.4956        1.1560           3.445      27.23
## 5                0.05883       0.7572        0.7813           5.438      94.44
## 6                0.07613       0.3345        0.8902           2.217      27.19
##   smoothness.error compactness.error concavity.error concave.points.error
## 1         0.006399           0.04904         0.05373              0.01587
## 2         0.005225           0.01308         0.01860              0.01340
## 3         0.006150           0.04006         0.03832              0.02058
## 4         0.009110           0.07458         0.05661              0.01867
## 5         0.011490           0.02461         0.05688              0.01885
## 6         0.007510           0.03345         0.03672              0.01137
```

```
##   symmetry.error fractal.dimension.error worst.radius worst.texture
## 1        0.03003                0.006193        25.38         17.33
## 2        0.01389                0.003532        24.99         23.41
## 3        0.02250                0.004571        23.57         25.53
## 4        0.05963                0.009208        14.91         26.50
## 5        0.01756                0.005115        22.54         16.67
## 6        0.02165                0.005082        15.47         23.75
##   worst.perimeter worst.area worst.smoothness worst.compactness worst.concavity
## 1          184.60     2019.0           0.1622            0.6656          0.7119
## 2          158.80     1956.0           0.1238            0.1866          0.2416
## 3          152.50     1709.0           0.1444            0.4245          0.4504
## 4           98.87      567.7           0.2098            0.8663          0.6869
## 5          152.20     1575.0           0.1374            0.2050          0.4000
## 6          103.40      741.6           0.1791            0.5249          0.5355
##   worst.concave.points worst.symmetry worst.fractal.dimension target
## 1               0.2654         0.4601                 0.11890      0
## 2               0.1860         0.2750                 0.08902      0
## 3               0.2430         0.3613                 0.08758      0
## 4               0.2575         0.6638                 0.17300      0
## 5               0.1625         0.2364                 0.07678      0
## 6               0.1741         0.3985                 0.12440      0
```

To explore the dataset

```
str(data)
```

```
## 'data.frame':    569 obs. of  31 variables:
##  $ mean.radius            : num  18 20.6 19.7 11.4 20.3 ...
##  $ mean.texture           : num  10.4 17.8 21.2 20.4 14.3 ...
##  $ mean.perimeter         : num  122.8 132.9 130 77.6 135.1 ...
##  $ mean.area              : num  1001 1326 1203 386 1297 ...
##  $ mean.smoothness        : num  0.1184 0.0847 0.1096 0.1425 0.1003 ...
##  $ mean.compactness       : num  0.2776 0.0786 0.1599 0.2839 0.1328 ...
##  $ mean.concavity         : num  0.3001 0.0869 0.1974 0.2414 0.198 ...
##  $ mean.concave.points    : num  0.1471 0.0702 0.1279 0.1052 0.1043 ...
##  $ mean.symmetry          : num  0.242 0.181 0.207 0.26 0.181 ...
##  $ mean.fractal.dimension : num  0.0787 0.0567 0.06 0.0974 0.0588 ...
##  $ radius.error           : num  1.095 0.543 0.746 0.496 0.757 ...
##  $ texture.error          : num  0.905 0.734 0.787 1.156 0.781 ...
##  $ perimeter.error        : num  8.59 3.4 4.58 3.44 5.44 ...
##  $ area.error             : num  153.4 74.1 94 27.2 94.4 ...
##  $ smoothness.error       : num  0.0064 0.00522 0.00615 0.00911 0.01149 ...
##  $ compactness.error      : num  0.049 0.0131 0.0401 0.0746 0.0246 ...
##  $ concavity.error        : num  0.0537 0.0186 0.0383 0.0566 0.0569 ...
##  $ concave.points.error   : num  0.0159 0.0134 0.0206 0.0187 0.0188 ...
##  $ symmetry.error         : num  0.03 0.0139 0.0225 0.0596 0.0176 ...
##  $ fractal.dimension.error: num  0.00619 0.00353 0.00457 0.00921 0.00511 ...
##  $ worst.radius           : num  25.4 25 23.6 14.9 22.5 ...
##  $ worst.texture          : num  17.3 23.4 25.5 26.5 16.7 ...
##  $ worst.perimeter        : num  184.6 158.8 152.5 98.9 152.2 ...
##  $ worst.area             : num  2019 1956 1709 568 1575 ...
##  $ worst.smoothness       : num  0.162 0.124 0.144 0.21 0.137 ...
##  $ worst.compactness      : num  0.666 0.187 0.424 0.866 0.205 ...
##  $ worst.concavity        : num  0.712 0.242 0.45 0.687 0.4 ...
```

```
##  $ worst.concave.points   : num  0.265 0.186 0.243 0.258 0.163 ...
##  $ worst.symmetry         : num  0.46 0.275 0.361 0.664 0.236 ...
##  $ worst.fractal.dimension: num  0.1189 0.089 0.0876 0.173 0.0768 ...
##  $ target                 : int  0 0 0 0 0 0 0 0 0 0 ...
```

```
summary(data)
```

```
##   mean.radius      mean.texture   mean.perimeter     mean.area     
##  Min.   : 6.981   Min.   : 9.71   Min.   : 43.79   Min.   : 143.5  
##  1st Qu.:11.700   1st Qu.:16.17   1st Qu.: 75.17   1st Qu.: 420.3  
##  Median :13.370   Median :18.84   Median : 86.24   Median : 551.1  
##  Mean   :14.127   Mean   :19.29   Mean   : 91.97   Mean   : 654.9  
##  3rd Qu.:15.780   3rd Qu.:21.80   3rd Qu.:104.10   3rd Qu.: 782.7  
##  Max.   :28.110   Max.   :39.28   Max.   :188.50   Max.   :2501.0  
##  mean.smoothness   mean.compactness  mean.concavity    mean.concave.points
##  Min.   :0.05263   Min.   :0.01938   Min.   :0.00000   Min.   :0.00000    
##  1st Qu.:0.08637   1st Qu.:0.06492   1st Qu.:0.02956   1st Qu.:0.02031    
##  Median :0.09587   Median :0.09263   Median :0.06154   Median :0.03350    
##  Mean   :0.09636   Mean   :0.10434   Mean   :0.08880   Mean   :0.04892    
##  3rd Qu.:0.10530   3rd Qu.:0.13040   3rd Qu.:0.13070   3rd Qu.:0.07400    
##  Max.   :0.16340   Max.   :0.34540   Max.   :0.42680   Max.   :0.20120    
##  mean.symmetry    mean.fractal.dimension  radius.error    texture.error   
##  Min.   :0.1060   Min.   :0.04996        Min.   :0.1115   Min.   :0.3602  
##  1st Qu.:0.1619   1st Qu.:0.05770        1st Qu.:0.2324   1st Qu.:0.8339  
##  Median :0.1792   Median :0.06154        Median :0.3242   Median :1.1080  
##  Mean   :0.1812   Mean   :0.06280        Mean   :0.4052   Mean   :1.2169  
##  3rd Qu.:0.1957   3rd Qu.:0.06612        3rd Qu.:0.4789   3rd Qu.:1.4740  
##  Max.   :0.3040   Max.   :0.09744        Max.   :2.8730   Max.   :4.8850  
##  perimeter.error    area.error       smoothness.error   compactness.error
##  Min.   : 0.757   Min.   :  6.802   Min.   :0.001713   Min.   :0.002252  
##  1st Qu.: 1.606   1st Qu.: 17.850   1st Qu.:0.005169   1st Qu.:0.013080  
##  Median : 2.287   Median : 24.530   Median :0.006380   Median :0.020450  
##  Mean   : 2.866   Mean   : 40.337   Mean   :0.007041   Mean   :0.025478  
##  3rd Qu.: 3.357   3rd Qu.: 45.190   3rd Qu.:0.008146   3rd Qu.:0.032450  
##  Max.   :21.980   Max.   :542.200   Max.   :0.031130   Max.   :0.135400  
```

```
##  concavity.error   concave.points.error symmetry.error
##  Min.   :0.00000   Min.   :0.000000     Min.   :0.007882
##  1st Qu.:0.01509   1st Qu.:0.007638     1st Qu.:0.015160
##  Median :0.02589   Median :0.010930     Median :0.018730
##  Mean   :0.03189   Mean   :0.011796     Mean   :0.020542
##  3rd Qu.:0.04205   3rd Qu.:0.014710     3rd Qu.:0.023480
##  Max.   :0.39600   Max.   :0.052790     Max.   :0.078950
##  fractal.dimension.error  worst.radius   worst.texture   worst.perimeter
##  Min.   :0.0008948       Min.   : 7.93   Min.   :12.02   Min.   : 50.41
##  1st Qu.:0.0022480       1st Qu.:13.01   1st Qu.:21.08   1st Qu.: 84.11
##  Median :0.0031870       Median :14.97   Median :25.41   Median : 97.66
##  Mean   :0.0037949       Mean   :16.27   Mean   :25.68   Mean   :107.26
##  3rd Qu.:0.0045580       3rd Qu.:18.79   3rd Qu.:29.72   3rd Qu.:125.40
##  Max.   :0.0298400       Max.   :36.04   Max.   :49.54   Max.   :251.20
##    worst.area     worst.smoothness  worst.compactness worst.concavity
##  Min.   : 185.2   Min.   :0.07117   Min.   :0.02729   Min.   :0.0000
##  1st Qu.: 515.3   1st Qu.:0.11660   1st Qu.:0.14720   1st Qu.:0.1145
##  Median : 686.5   Median :0.13130   Median :0.21190   Median :0.2267
##  Mean   : 880.6   Mean   :0.13237   Mean   :0.25427   Mean   :0.2722
##  3rd Qu.:1084.0   3rd Qu.:0.14600   3rd Qu.:0.33910   3rd Qu.:0.3829
##  Max.   :4254.0   Max.   :0.22260   Max.   :1.05800   Max.   :1.2520
##  worst.concave.points worst.symmetry   worst.fractal.dimension     target
##  Min.   :0.00000      Min.   :0.1565   Min.   :0.05504         Min.   :0.0000
##  1st Qu.:0.06493      1st Qu.:0.2504   1st Qu.:0.07146         1st Qu.:0.0000
##  Median :0.09993      Median :0.2822   Median :0.08004         Median :1.0000
##  Mean   :0.11461      Mean   :0.2901   Mean   :0.08395         Mean   :0.6274
##  3rd Qu.:0.16140      3rd Qu.:0.3179   3rd Qu.:0.09208         3rd Qu.:1.0000
##  Max.   :0.29100      Max.   :0.6638   Max.   :0.20750         Max.   :1.0000
```

data cleaning(Preprocessing)

```
# To remove the missing value
data<- na.omit(data)
head(data)
```

```
##   mean.radius mean.texture mean.perimeter mean.area mean.smoothness
## 1       17.99        10.38         122.80    1001.0         0.11840
## 2       20.57        17.77         132.90    1326.0         0.08474
## 3       19.69        21.25         130.00    1203.0         0.10960
## 4       11.42        20.38          77.58     386.1         0.14250
## 5       20.29        14.34         135.10    1297.0         0.10030
## 6       12.45        15.70          82.57     477.1         0.12780
##   mean.compactness mean.concavity mean.concave.points mean.symmetry
## 1          0.27760         0.3001             0.14710        0.2419
## 2          0.07864         0.0869             0.07017        0.1812
## 3          0.15990         0.1974             0.12790        0.2069
## 4          0.28390         0.2414             0.10520        0.2597
## 5          0.13280         0.1980             0.10430        0.1809
## 6          0.17000         0.1578             0.08089        0.2087
##   mean.fractal.dimension radius.error texture.error perimeter.error area.error
## 1                0.07871       1.0950        0.9053           8.589     153.40
## 2                0.05667       0.5435        0.7339           3.398      74.08
## 3                0.05999       0.7456        0.7869           4.585      94.03
## 4                0.09744       0.4956        1.1560           3.445      27.23
## 5                0.05883       0.7572        0.7813           5.438      94.44
## 6                0.07613       0.3345        0.8902           2.217      27.19
##   smoothness.error compactness.error concavity.error concave.points.error
## 1         0.006399           0.04904         0.05373              0.01587
## 2         0.005225           0.01308         0.01860              0.01340
## 3         0.006150           0.04006         0.03832              0.02058
## 4         0.009110           0.07458         0.05661              0.01867
## 5         0.011490           0.02461         0.05688              0.01885
## 6         0.007510           0.03345         0.03672              0.01137
```

```
##   symmetry.error fractal.dimension.error worst.radius worst.texture
## 1        0.03003                0.006193        25.38         17.33
## 2        0.01389                0.003532        24.99         23.41
## 3        0.02250                0.004571        23.57         25.53
## 4        0.05963                0.009208        14.91         26.50
## 5        0.01756                0.005115        22.54         16.67
## 6        0.02165                0.005082        15.47         23.75
##   worst.perimeter worst.area worst.smoothness worst.compactness worst.concavity
## 1          184.60     2019.0           0.1622            0.6656          0.7119
## 2          158.80     1956.0           0.1238            0.1866          0.2416
## 3          152.50     1709.0           0.1444            0.4245          0.4504
## 4           98.87      567.7           0.2098            0.8663          0.6869
## 5          152.20     1575.0           0.1374            0.2050          0.4000
## 6          103.40      741.6           0.1791            0.5249          0.5355
##   worst.concave.points worst.symmetry worst.fractal.dimension target
## 1               0.2654         0.4601                 0.11890      0
## 2               0.1860         0.2750                 0.08902      0
## 3               0.2430         0.3613                 0.08758      0
## 4               0.2575         0.6638                 0.17300      0
## 5               0.1625         0.2364                 0.07678      0
## 6               0.1741         0.3985                 0.12440      0
```

```
# Convert the target variable (Class) into a factor
data$target <- factor(data$target)
```

To split train and test data

```r
nrows <- NROW(data)
set.seed(218)                             ## fix random value
index <- sample(1:nrows, 0.8 * nrows)   ## shuffle and divide
index
```

```
##   [1] 212 379 560 295 443 561 275 329 254 425 491 344  28 559 202 151  73 478
##  [19] 179 169 268 530 447 320 159 297 249  79 276 190   9 567 181 434 405 353
##  [37] 446 398 377 498 366  75 333 376  34 110 529 144 213 475 305 301 420 431
##  [55] 308 189 486 206   3 518 357  39  55 412  41 331 182 228 473 540 307 462
##  [73]  89 423 237 378 547  98  92 544 348  23 271  70 542 168  99 161 372 225
##  [91] 173 164  47 501 430 497 509 495 516  90 354 204  82 121 332 538 500 244
## [109] 455  71 284 342 464 485 176 519  74 536 138 505  63  43  76 152 564 340
## [127] 477 334 494 184 150  62 269 454 553 345   1  88  52 330 532 335  93 450
## [145] 248 193 517 552 241 309 141 390 306  46 521 312 408 381 103 178 550 192
## [163] 267   4 490 310 124 153 147 511 315 382 371  31 537 162 374  66 545 261
## [181] 549 105 558 428  14 406 458 136 401 171 427 115  30 397 510  56 456 107
## [199] 351 201   2 280 101 499 417 515 148 170 238 472  64 539 435 245 488 563
## [217] 277 343 476 250 304 546 527 252 369 566 364 294 514 111  78 113 389 278
## [235] 449 165  17 392 479 292 255  21  58 256 469 380  65 504  15 123 119 258
## [253]  87  13 543 291 388 467 214 461 493 356 350  60 116 264 106 508 300 440
## [271] 429 336 355 409 199 128 223 492 419 452 270  25 289 211  96  18 260 337
## [289] 104 125 127 524  50 339  11 556 502 391 439 441 303 318 489 186 234 525
## [307] 142 385 239 158 240 129 506  57 554 174 548 180 282 109 411  42 232 242
## [325]  84 444 145 487 215 208 221 433 395 274 118  83 185 328 200 448 143  91
## [343] 259 533 551  33  81 207 218 557 387 140  61  53 418 421 361 422 117  80
## [361] 522 368 513   8 283 251 236 108 465 198  44  72  12 338 120 363 191 175
## [379] 474  35 358 183 407 194 534  94 235 482  68 166 568 317 365 177 265 216
## [397] 438  29 131   7 187 290 393  45 314  59 243 272 210  69  67 149 209 266
## [415]  51  49 373  77 470 349 160 451 386 460  38 360 528 188  19 132 196 483
## [433] 122 102 230 424 503   5  26 222  37 311 370  20 432 130  32  40 410  16
## [451] 146 403 233 302 172
```

```
#train <- data                                    ## 569 test data (100%)
train <- data[index,]                             ## 455 test data (80%)
head(train)
```

```
##     mean.radius mean.texture mean.perimeter mean.area mean.smoothness
## 212       11.84        18.94          75.51     428.0         0.08871
## 379       13.66        15.15          88.27     580.6         0.08268
## 560       11.51        23.93          74.52     403.5         0.09261
## 295       12.72        13.78          81.78     492.1         0.09667
## 443       13.78        15.79          88.37     585.9         0.08817
## 561       14.05        27.15          91.38     600.4         0.09929
##     mean.compactness mean.concavity mean.concave.points mean.symmetry
## 212          0.06900        0.02669            0.013930        0.1533
## 379          0.07548        0.04249            0.024710        0.1792
## 560          0.10210        0.11120            0.041050        0.1388
## 295          0.08393        0.01288            0.019240        0.1638
## 443          0.06718        0.01055            0.009937        0.1405
## 561          0.11260        0.04462            0.043040        0.1537
##     mean.fractal.dimension radius.error texture.error perimeter.error
## 212                0.06057       0.2222        0.8652           1.444
## 379                0.05897       0.1402        0.5417           1.101
## 560                0.06570       0.2388        2.9040           1.936
## 295                0.06100       0.1807        0.6931           1.340
## 443                0.05848       0.3563        0.4833           2.235
## 561                0.06171       0.3645        1.4920           2.888
##     area.error smoothness.error compactness.error concavity.error
## 212      17.12         0.005517           0.01727        0.020450
## 379      11.35         0.005212           0.02984        0.024430
## 560      16.97         0.008200           0.02982        0.057380
## 295      13.38         0.006064           0.01180        0.006564
## 443      29.34         0.006432           0.01156        0.007741
## 561      29.84         0.007256           0.02678        0.020710
```

```
##     concave.points.error symmetry.error fractal.dimension.error worst.radius
## 212             0.006747        0.01616                0.002922        13.30
## 379             0.008356        0.01818                0.004868        14.54
## 560             0.012670        0.01488                0.004738        12.48
## 295             0.007978        0.01374                0.001392        13.50
## 443             0.005657        0.01227                0.002564        15.27
## 561             0.016260        0.02080                0.005304        15.30
##     worst.texture worst.perimeter worst.area worst.smoothness worst.compactness
## 212         24.99           85.22      546.3           0.1280            0.1880
## 379         19.64           97.96      657.0           0.1275            0.3104
## 560         37.16           82.28      474.2           0.1298            0.2517
## 295         17.48           88.54      553.7           0.1298            0.1472
## 443         17.50           97.90      706.6           0.1072            0.1071
## 561         33.17          100.20      706.7           0.1241            0.2264
##     worst.concavity worst.concave.points worst.symmetry worst.fractal.dimension
## 212         0.14710              0.06913         0.2535                 0.07993
## 379         0.25690              0.10540         0.3387                 0.09638
## 560         0.36300              0.09653         0.2112                 0.08732
## 295         0.05233              0.06343         0.2369                 0.06922
## 443         0.03517              0.03312         0.1859                 0.06810
## 561         0.13260              0.10480         0.2250                 0.08321
##     target
## 212      1
## 379      1
## 560      1
## 295      1
## 443      1
## 561      1
```

```
test <- data[-index,]                    ## 114 test data (20%)
head(test)
```

```
##    mean.radius mean.texture mean.perimeter mean.area mean.smoothness
## 6       12.450        15.70          82.57     477.1         0.12780
## 10      12.460        24.04          83.97     475.9         0.11860
## 22       9.504        12.44          60.34     273.9         0.10240
## 24      21.160        23.04         137.20    1404.0         0.09428
## 27      14.580        21.53          97.41     644.8         0.10540
## 36      16.740        21.59         110.10     869.5         0.09610
##    mean.compactness mean.concavity mean.concave.points mean.symmetry
## 6           0.17000        0.15780             0.08089        0.2087
## 10          0.23960        0.22730             0.08543        0.2030
## 22          0.06492        0.02956             0.02076        0.1815
## 24          0.10220        0.10970             0.08632        0.1769
## 27          0.18680        0.14250             0.08783        0.2252
## 36          0.13360        0.13480             0.06018        0.1896
##    mean.fractal.dimension radius.error texture.error perimeter.error area.error
## 6                 0.07613       0.3345        0.8902           2.217      27.19
## 10                0.08243       0.2976        1.5990           2.039      23.94
## 22                0.06905       0.2773        0.9768           1.909      15.70
## 24                0.05278       0.6917        1.1270           4.303      93.99
## 27                0.06924       0.2545        0.9832           2.110      21.05
## 36                0.05656       0.4615        0.9197           3.008      45.19
##    smoothness.error compactness.error concavity.error concave.points.error
## 6          0.007510           0.03345         0.03672              0.01137
## 10         0.007149           0.07217         0.07743              0.01432
## 22         0.009606           0.01432         0.01985              0.01421
## 24         0.004728           0.01259         0.01715              0.01038
## 27         0.004452           0.03055         0.02681              0.01352
## 36         0.005776           0.02499         0.03695              0.01195
```

```
##    symmetry.error fractal.dimension.error worst.radius worst.texture
## 6         0.02165                0.005082        15.47         23.75
## 10        0.01789                0.010080        15.09         40.68
## 22        0.02027                0.002968        10.23         15.66
## 24        0.01083                0.001987        29.17         35.59
## 27        0.01454                0.003711        17.62         33.21
## 36        0.02789                0.002665        20.01         29.02
##    worst.perimeter worst.area worst.smoothness worst.compactness
## 6           103.40      741.6           0.1791            0.5249
## 10           97.65      711.4           0.1853            1.0580
## 22           65.13      314.9           0.1324            0.1148
## 24          188.00     2615.0           0.1401            0.2600
## 27          122.40      896.9           0.1525            0.6643
## 36          133.50     1229.0           0.1563            0.3835
##    worst.concavity worst.concave.points worst.symmetry worst.fractal.dimension
## 6          0.53550              0.17410         0.3985                 0.12440
## 10         1.10500              0.22100         0.4366                 0.20750
## 22         0.08867              0.06227         0.2450                 0.07773
## 24         0.31550              0.20090         0.2822                 0.07526
## 27         0.55390              0.27010         0.4264                 0.12750
## 36         0.54090              0.18130         0.4863                 0.08633
##    target
## 6       0
## 10      0
## 22      1
## 24      0
## 27      0
## 36      0
```

```
prop.table(table(train$target))
```

```
##
##        0        1
## 0.378022 0.621978
```

```
test$diagnosis
```

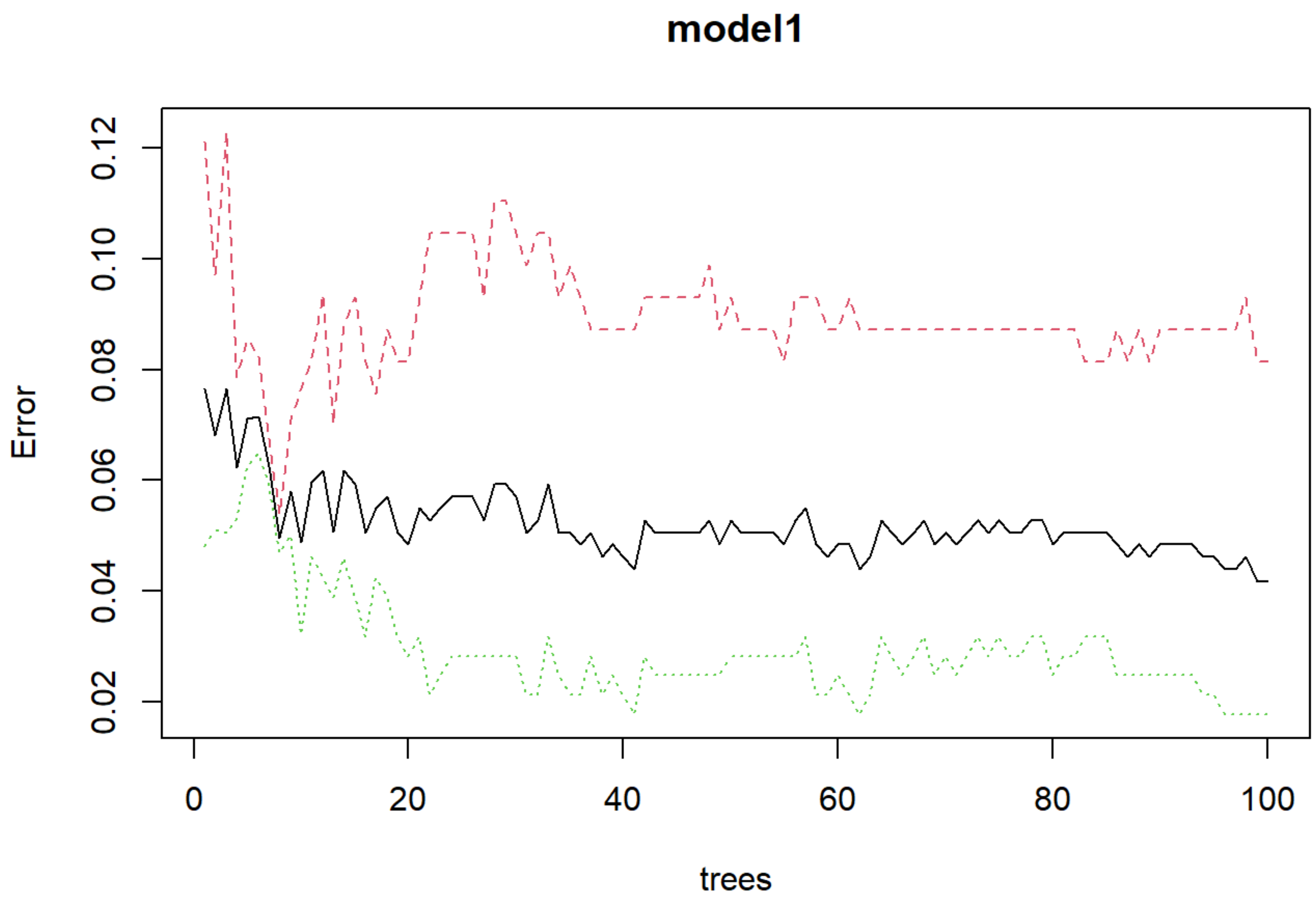
```
## NULL
```

Random Forest Model train

```
# To training the Random Forest model
set.seed(218)

# target variable is predicted using all other variables in the dataset.

# data that contains both contains predictor variables and response variable.

# ntree: no. of tree to grow in the forest,if increase it then accuracy is better through it take more computational time.

# mtry: number of predictor variables to consider at each split.usually use square root of the no. of predictors in classification and in regression is set to one-third of the number of predictors.

# importance: compute variable importance measures,if set is true

# proximity: measures how similar observation are based on the classification tree,this is useful for clustering.

model1<-randomForest(target ~.,data=train,ntree=100,mtry=4,importance=TRUE)
# To view the model summary
print(model1)
```

```
##
## Call:
##  randomForest(formula = target ~ ., data = train, ntree = 100,      mtry = 4, importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 100
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 4.18%
## Confusion matrix:
##     0   1 class.error
## 0 158  14  0.08139535
## 1   5 278  0.01766784
```

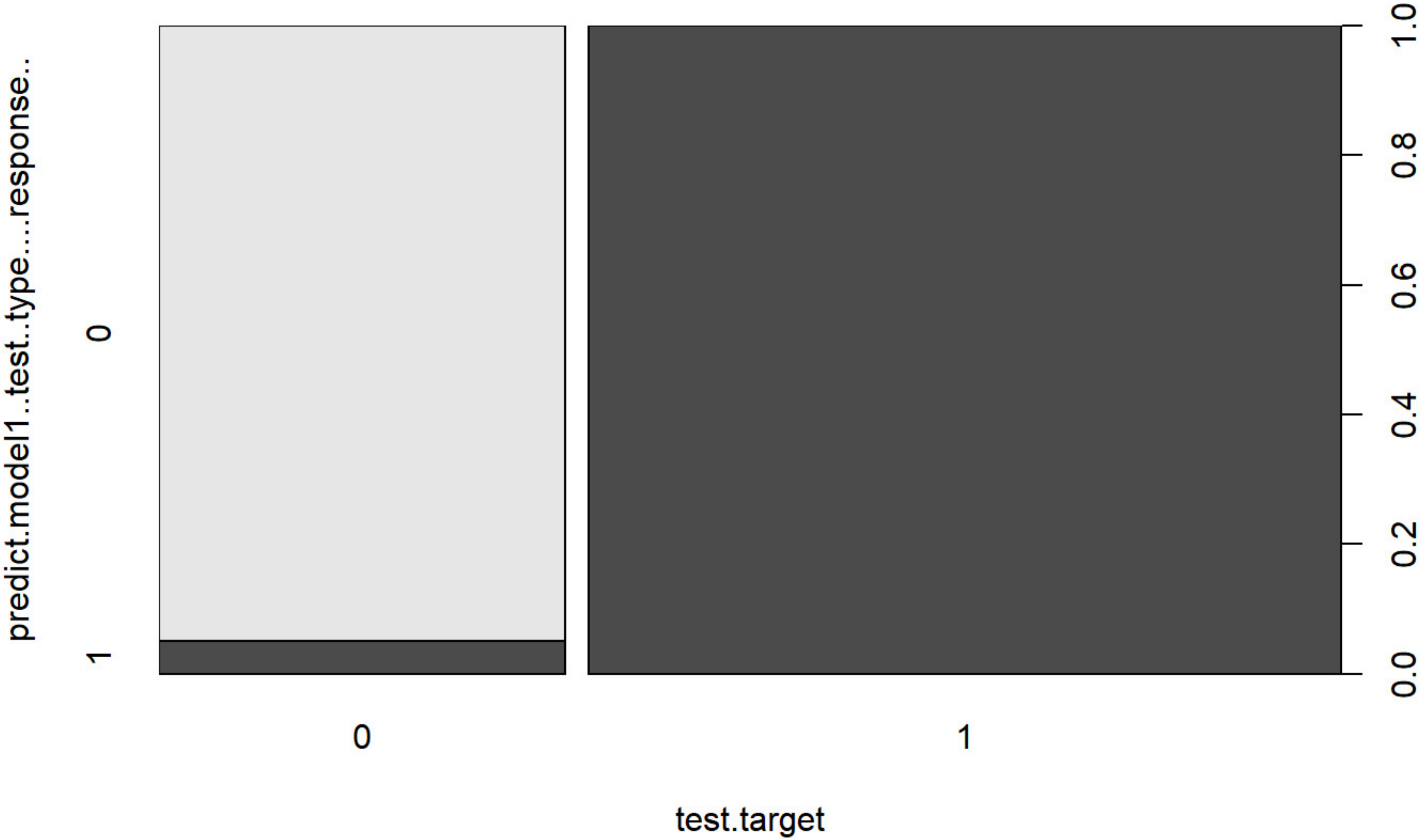
```
plot(model1)
```


model1
Error
0.02
0.04
0.06
0.08
0.10
0.12
trees
0
20
40
60
80
100

```
result<- data.frame(test$target,predict(model1,test,type="response"))
head(result)
```

```
##    test.target predict.model1..test..type....response..
## 6            0                                         0
## 10           0                                         0
## 22           1                                         1
## 24           0                                         0
## 27           0                                         0
## 36           0                                         0
```

```
plot(result)
```



predict.model1..test..type.....response..
0
1
1.0
0.8
0.6
0.4
0.2
0.0
0
1
test.target

```r
# Make predictions on test data
predictions <- predict(model1, newdata = test)

# View predictions
print(predictions)
```

```
##   6  10  22  24  27  36  48  54  85  86  95  97 100 112 114 126 133 134 135 137
##   0   0   1   0   0   0   0   0   1   0   0   1   0   1   1   1   0   1   0   1
## 139 154 155 156 157 163 167 195 197 203 205 217 219 220 224 226 227 229 231 246
##   0   1   1   1   0   0   1   0   0   0   1   1   0   0   0   1   1   1   0   1
## 247 253 257 262 263 273 279 281 285 286 287 288 293 296 298 299 313 316 319 321
##   1   0   0   0   0   0   1   0   1   1   1   1   1   1   1   1   1   1   1   1
## 322 323 324 325 326 327 341 346 347 352 359 362 367 375 383 384 394 396 399 400
##   0   1   0   1   1   1   1   1   1   0   1   1   0   1   1   1   0   1   1   1
## 402 404 413 414 415 416 426 436 437 442 445 453 457 459 463 466 468 471 480 481
##   1   1   1   1   1   1   1   0   1   0   0   1   1   1   1   1   1   1   0   1
## 484 496 507 512 520 523 526 531 535 541 555 562 565 569
##   1   1   1   1   1   1   1   1   1   1   1   1   0   1
## Levels: 0 1
```

```r
# Create confusion matrix
confusion_matrix <- table(test$target, predictions)

# Print confusion matrix
print(confusion_matrix)
```

```
##    predictions
##      0  1
##   0 38  2
##   1  0 74
```

```
# Calculate accuracy
accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
cat("Accuracy:", accuracy)
```

```
## Accuracy: 0.9824561
```

```
# Convert the target variable (Class) into a factor
data$target <- factor(data$target)
```

```
# Random Forest model  creation
model2 <- randomForest(target~ ., data = train, ntree = 100, mtry = 3)

# Model summary print
print(model2)
```

```
##
## Call:
##  randomForest(formula = target ~ ., data = train, ntree = 100,      mtry = 3)
##                Type of random forest: classification
##                      Number of trees: 100
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 4.62%
## Confusion matrix:
##     0   1 class.error
## 0 160  12  0.06976744
## 1   9 274  0.03180212
```

```
# Check for empty classes
table(data$target)
```

```
##
##   0   1
## 212 357
```

```
train$target <-as.factor(train$target)
# Fit Random Forest Model again after cleaning data
model3 <- randomForest(target ~ ., data = train, ntree = 500, mtry = 3)

# Print model summary
print(model3)
```

```
##
## Call:
##  randomForest(formula = target ~ ., data = train, ntree = 500,      mtry = 3)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 4.62%
## Confusion matrix:
##     0   1 class.error
## 0 160  12  0.06976744
## 1   9 274  0.03180212
```

```
str(test)
```

```
## 'data.frame':    114 obs. of  31 variables:
##  $ mean.radius            : num  12.4 12.5 9.5 21.2 14.6 ...
##  $ mean.texture           : num  15.7 24 12.4 23 21.5 ...
##  $ mean.perimeter         : num  82.6 84 60.3 137.2 97.4 ...
##  $ mean.area              : num  477 476 274 1404 645 ...
##  $ mean.smoothness        : num  0.1278 0.1186 0.1024 0.0943 0.1054 ...
##  $ mean.compactness       : num  0.17 0.2396 0.0649 0.1022 0.1868 ...
##  $ mean.concavity         : num  0.1578 0.2273 0.0296 0.1097 0.1425 ...
##  $ mean.concave.points    : num  0.0809 0.0854 0.0208 0.0863 0.0878 ...
##  $ mean.symmetry          : num  0.209 0.203 0.181 0.177 0.225 ...
##  $ mean.fractal.dimension : num  0.0761 0.0824 0.069 0.0528 0.0692 ...
##  $ radius.error           : num  0.335 0.298 0.277 0.692 0.254 ...
##  $ texture.error          : num  0.89 1.599 0.977 1.127 0.983 ...
##  $ perimeter.error        : num  2.22 2.04 1.91 4.3 2.11 ...
##  $ area.error             : num  27.2 23.9 15.7 94 21.1 ...
##  $ smoothness.error       : num  0.00751 0.00715 0.00961 0.00473 0.00445 ...
##  $ compactness.error      : num  0.0335 0.0722 0.0143 0.0126 0.0306 ...
##  $ concavity.error        : num  0.0367 0.0774 0.0198 0.0171 0.0268 ...
##  $ concave.points.error   : num  0.0114 0.0143 0.0142 0.0104 0.0135 ...
##  $ symmetry.error         : num  0.0216 0.0179 0.0203 0.0108 0.0145 ...
##  $ fractal.dimension.error: num  0.00508 0.01008 0.00297 0.00199 0.00371 ...
##  $ worst.radius           : num  15.5 15.1 10.2 29.2 17.6 ...
##  $ worst.texture          : num  23.8 40.7 15.7 35.6 33.2 ...
##  $ worst.perimeter        : num  103.4 97.7 65.1 188 122.4 ...
##  $ worst.area             : num  742 711 315 2615 897 ...
##  $ worst.smoothness       : num  0.179 0.185 0.132 0.14 0.152 ...
##  $ worst.compactness      : num  0.525 1.058 0.115 0.26 0.664 ...
##  $ worst.concavity        : num  0.5355 1.105 0.0887 0.3155 0.5539 ...
```

```
##  $ worst.concave.points   : num  0.1741 0.221 0.0623 0.2009 0.2701 ...
##  $ worst.symmetry         : num  0.399 0.437 0.245 0.282 0.426 ...
##  $ worst.fractal.dimension: num  0.1244 0.2075 0.0777 0.0753 0.1275 ...
##  $ target                 : Factor w/ 2 levels "0","1": 1 1 2 1 1 1 1 1 2 1 ...
```

```
colnames(test)
```

```
##  [1] "mean.radius"             "mean.texture"
##  [3] "mean.perimeter"          "mean.area"
##  [5] "mean.smoothness"         "mean.compactness"
##  [7] "mean.concavity"          "mean.concave.points"
##  [9] "mean.symmetry"           "mean.fractal.dimension"
## [11] "radius.error"            "texture.error"
## [13] "perimeter.error"         "area.error"
## [15] "smoothness.error"        "compactness.error"
## [17] "concavity.error"         "concave.points.error"
## [19] "symmetry.error"          "fractal.dimension.error"
## [21] "worst.radius"            "worst.texture"
## [23] "worst.perimeter"         "worst.area"
## [25] "worst.smoothness"        "worst.compactness"
## [27] "worst.concavity"         "worst.concave.points"
## [29] "worst.symmetry"          "worst.fractal.dimension"
## [31] "target"
```

```
train$target <-as.factor(train$target)

# Train the random forest model

learn_rf <- randomForest(target ~ ., data = train, ntree = 500, proximity = TRUE, importance = TRUE)

pre_rf   <- predict(learn_rf, test)

actual_value <-as.factor(test$target)
cm_rf    <- confusionMatrix(pre_rf, actual_value)
cm_rf
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 38  0
##          1  2 74
##
##                Accuracy : 0.9825
##                  95% CI : (0.9381, 0.9979)
##     No Information Rate : 0.6491
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.961
##
##  Mcnemar's Test P-Value : 0.4795
##
##             Sensitivity : 0.9500
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.9737
##              Prevalence : 0.3509
##          Detection Rate : 0.3333
##    Detection Prevalence : 0.3333
##       Balanced Accuracy : 0.9750
##
##        'Positive' Class : 0
##
```

Support Vector Machines (SVMs)

it is a powerful supervised machine learning algorithm widely used for classification and regression tasks.

1.High-Dimensional Data Handling: SVMs excel in high-dimensional spaces, making them particularly effective for applications like image classification and text categorization, where the number of features often exceeds the number of samples.

2.Robustness to Overfitting: SVMs reduce the risk of overfitting, especially in scenarios where the dataset is small or has noise. This is achieved by focusing on support vectors.

3.Versatile Kernel Functions: SVMs can handle both linear and non-linear data through the use of kernel functions.

4.Memory Efficiency: define the decision boundary. This is particularly beneficial when working with large datasets.

5.Effective for Unbalanced Datasets: SVMs can be tailored to handle unbalanced datasets by assigning different weights to classes, improving classification performance for minority classes.

6.Good Generalization Performance: SVMs have strong generalization capabilities, meaning they can effectively classify new, unseen data based on patterns learned from the training set.

7.Application Versatility: SVMs are applicable in various fields such as bioinformatics (gene classification), image recognition, handwriting detection, and more, due to their flexibility and effectiveness across different types of data.

In [16]:
```
from sklearn import datasets
from sklearn.datasets import load_iris
iris=datasets.load_iris()
```

In [17]:
```
print(iris.DESCR) # Details of iris data
```

```
.. _iris_dataset:

Iris plants dataset
--------------------

**Data Set Characteristics:**

    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica

    :Summary Statistics:

    ============== ==== ==== ======= ===== ====================
                    Min  Max   Mean    SD   Class Correlation
    ============== ==== ==== ======= ===== ====================
    sepal length:   4.3  7.9   5.84   0.83    0.7826
    sepal width:    2.0  4.4   3.05   0.43   -0.4194
    petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
    petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)
    ============== ==== ==== ======= ===== ====================

    :Missing Attribute Values: None
    :Class Distribution: 33.3% for each of 3 classes.
    :Creator: R.A. Fisher
    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
    :Date: July, 1988

The famous Iris database, first used by Sir R.A. Fisher. The dataset is taken
from Fisher's paper. Note that it's the same as in R, but not as in the UCI
Machine Learning Repository, which has two wrong data points.

This is perhaps the best known database to be found in the
pattern recognition literature.  Fisher's paper is a classic in the field and
is referenced frequently to this day.  (See Duda & Hart, for example.)  The
data set contains 3 classes of 50 instances each, where each class refers to a
type of iris plant.  One class is linearly separable from the other 2; the
latter are NOT linearly separable from each other.

.. topic:: References

   - Fisher, R.A. "The use of multiple measurements in taxonomic problems"
     Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to
     Mathematical Statistics" (John Wiley, NY, 1950).
   - Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis.
     (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.
   - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System
     Structure and Classification Rule for Recognition in Partially Exposed
     Environments".  IEEE Transactions on Pattern Analysis and Machine
     Intelligence, Vol. PAMI-2, No. 1, 67-71.
   - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IEEE Transactions
     on Information Theory, May 1972, 431-433.
   - See also: 1988 MLC Proceedings, 54-64.  Cheeseman et al"s AUTOCLASS II
     conceptual clustering system finds 3 classes in the data.
   - Many, many more ...
```

In [6]:
```
X = iris.data  # features
y = iris.target  # target labels
```

In [8]:
```
from sklearn.model_selection import train_test_split
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

In [9]:
```
from sklearn.svm import SVC
# Create an SVM model with a specified kernel (e.g., 'linear', 'rbf', 'poly')
model = SVC(kernel='linear')  # change the kernel as needed
```

In [10]:
```
# Train the model
model.fit(X_train, y_train)
```

Out[10]:
```
SVC
SVC(kernel='linear')
```

In [11]:
```
# Make predictions on the test set
y_pred = model.predict(X_test)
```

In [13]:
```
from sklearn.metrics import accuracy_score
#  the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy of SVM model: {accuracy:.2f}')
```

```
Accuracy of SVM model: 1.00
```

Highly effective model on the specific test set

iris data for svm

Lab 8

2024-11-24

```
# Install necessary packages if you don't have them already
#install.packages("e1071")   # For SVM model
#install.packages("caret")   # For data partition and model evaluation

# Load libraries
library(e1071)
```

```
## Warning: package 'e1071' was built under R version 4.3.3
```

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.3.3
```

```
## Loading required package: ggplot2
```

```
## Warning: package 'ggplot2' was built under R version 4.3.3
```

```
## Loading required package: lattice
```

```
## Warning: package 'lattice' was built under R version 4.3.3
```

```
rm(list=ls()) # clear all previous data
```

```
library(ggplot2)
```

```
# Load the Iris dataset
data(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
```

```
# Check dataset structure
str(iris)
```

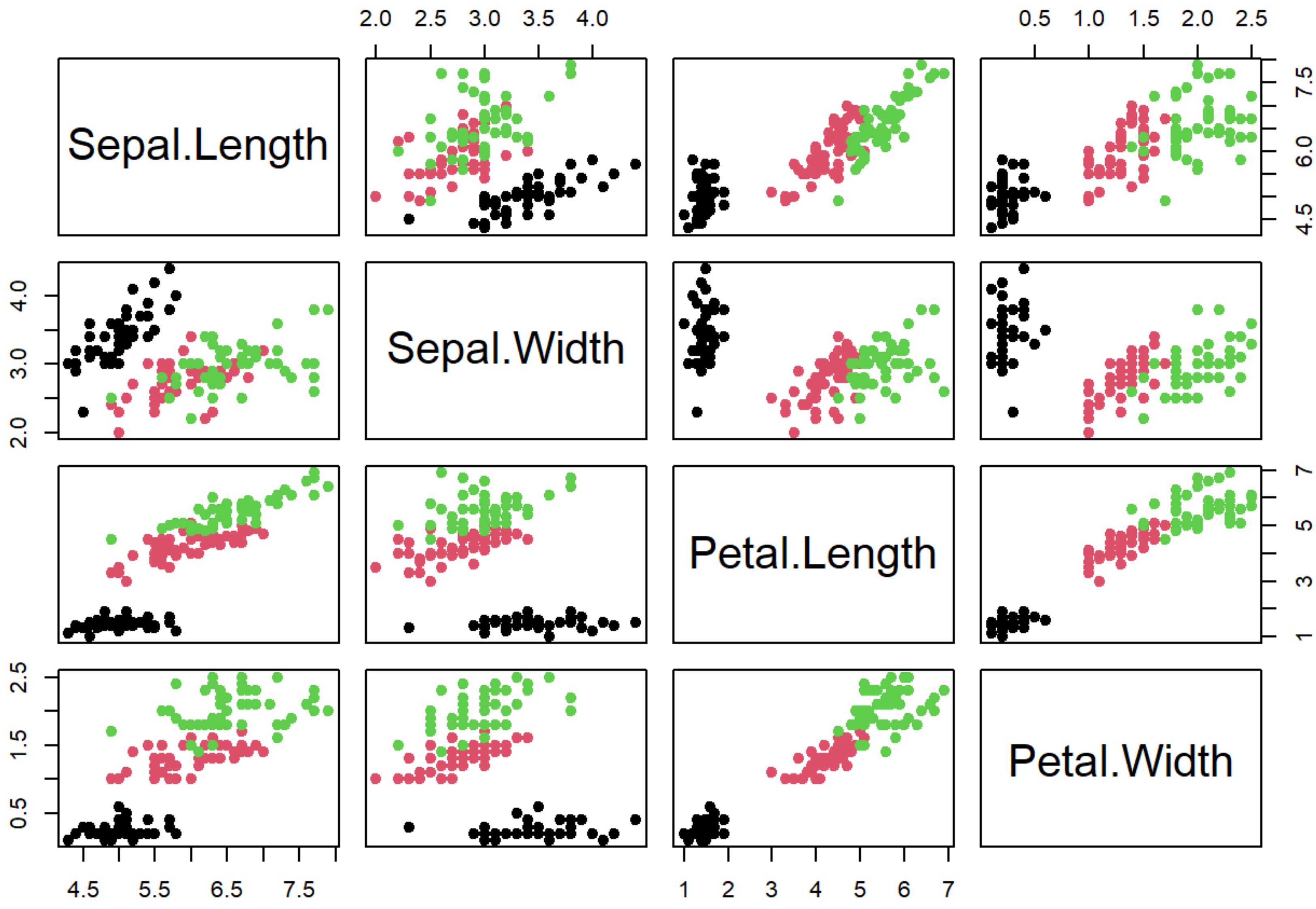
```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```
# Set a seed for reproducibility
set.seed(123)
```

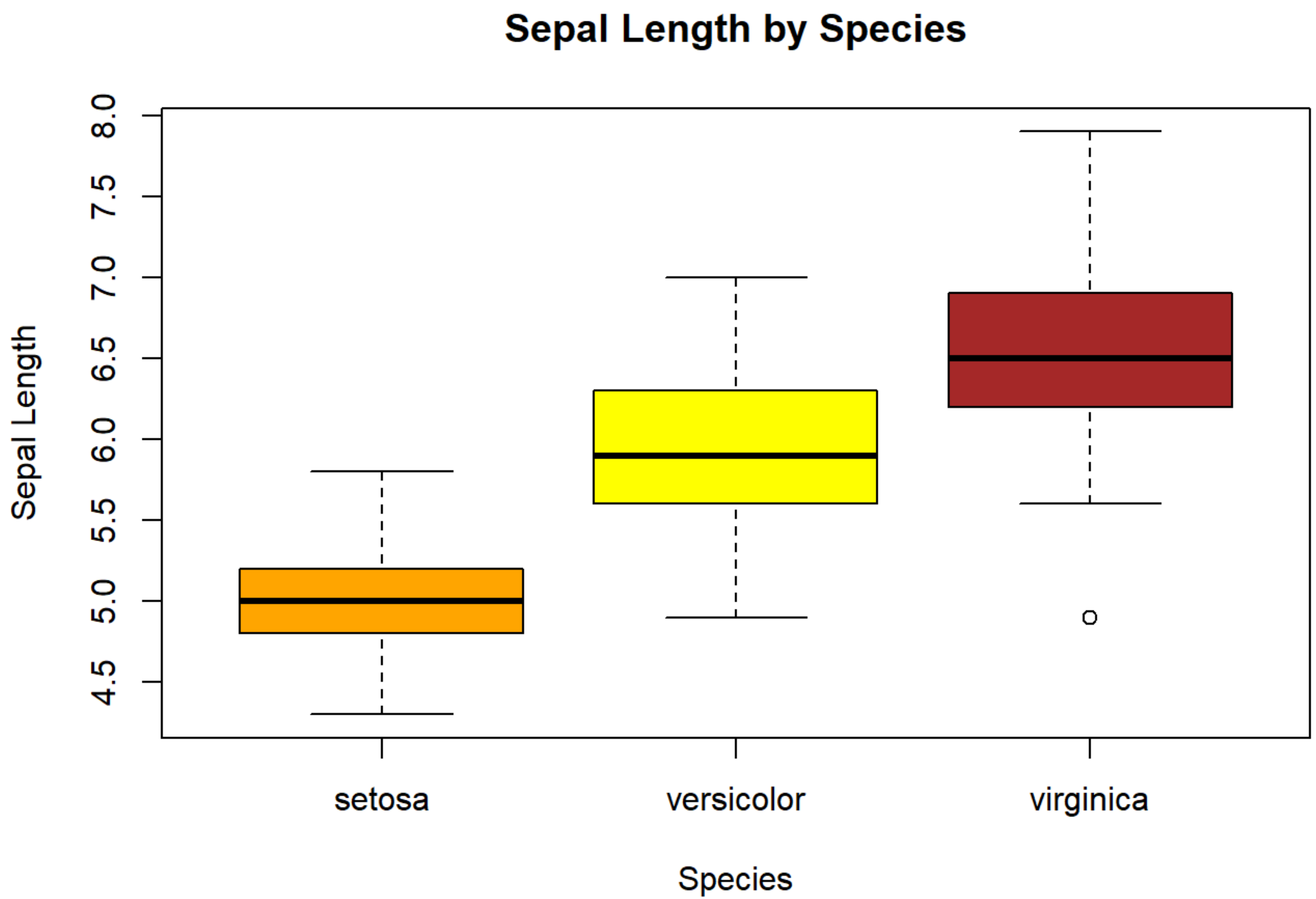
```
summary(iris)
```

```
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width   
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100  
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300  
##  Median :5.800   Median :3.000   Median :4.350   Median :1.300  
##  Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199  
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800  
##  Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500  
##        Species  
##  setosa    :50  
##  versicolor:50  
##  virginica :50  
##                 
##                 
## 
```

```
#Visualize the dataset to understand the distribution of classes and features.


# Pair plot to explore relationships
pairs(iris[1:4], col = iris$Species, pch = 19)
```


Sepal.Length
Sepal.Width
Petal.Length
Petal.Width
2.0 2.5 3.0 3.5 4.0
0.5 1.0 1.5 2.0 2.5
4.5 6.0 7.5
2.0 3.0 4.0
1 3 5 7
0.5 1.5 2.5
4.5 5.5 6.5 7.5
1 2 3 4 5 6 7

```
# Boxplot for feature distribution by species
boxplot(iris$Sepal.Length ~ iris$Species, main = "Sepal Length by Species",
xlab = "Species", ylab = "Sepal Length", col = c("orange", "yellow", "brown"))
```


Sepal Length by Species
Sepal Length
8.0
7.5
7.0
6.5
6.0
5.5
5.0
4.5
setosa
versicolor
virginica
Species

```r
# Set seed for reproducibility
set.seed(123)
# Split the data into training (80%) and testing (20%) sets
index <- createDataPartition(iris$Species, p = 0.8, list = FALSE)
train_data <- iris[index, ]
test_data <- iris[-index, ]
```

```r
# Check dimensions
dim(train_data)
```

```
## [1] 120   5
```

```r
dim(test_data)
```

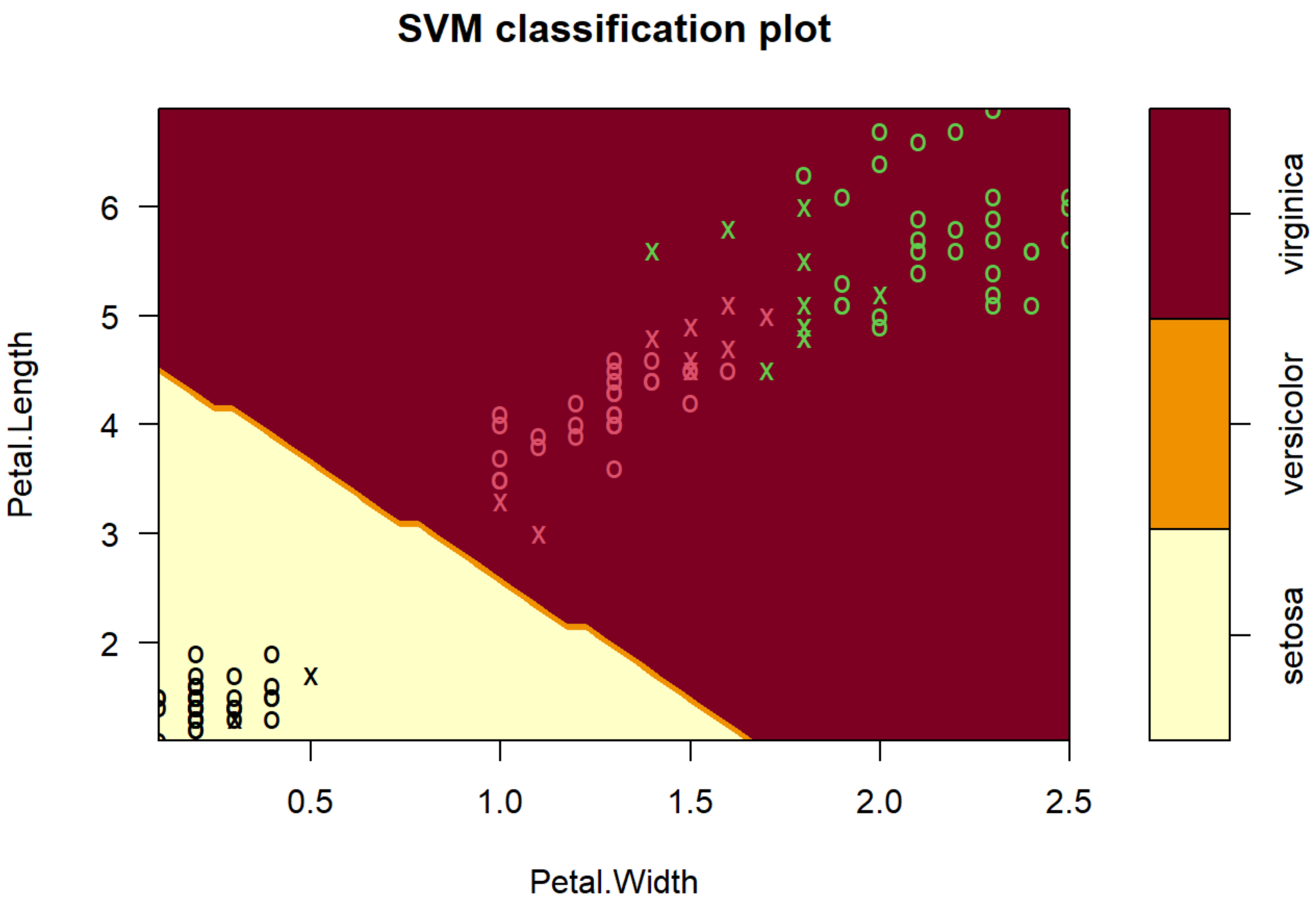
```
## [1] 30  5
```

Train the SVM model using the e1071 package.

```
# Train the SVM model
# Species is a target variable,svm(e1071 package),kernel for linear separable,cost=0.5 Sets the paramet
ers for balancing the training error and model complexity (soft margin),scale for improve the model per
formence,cross for Indicates cross-validation.
svm_model <- svm(Species ~ ., data = train_data, kernel =    "linear",Cost=1,scale=TRUE,Cross=N)
print(svm_model)
```

```
##
## Call:
## svm(formula = Species ~ ., data = train_data, kernel = "linear",
##     Cost = 1, Cross = N, scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  25
```

```
plot(svm_model, train_data, Petal.Length ~ Petal.Width)
```



small

value of support vector may indicate a simpler decision boundary. ### larger value of support vector may indicate a complex boundary, potentially due to noise or overlapping classes in the data.

```
best3=tune(svm,Species ~ ., data = train_data, kernel = "linear",range=list(Cost=0.01,0.1,0.5,1,10,10
0),scale=TRUE)
print(best3)
```

```
##
## Error estimation of 'svm' using 10-fold cross validation: 0.01666667
```

```
best4=best3$best.model
print(best4)
```

```
##
## Call:
## best.tune(METHOD = svm, train.x = Species ~ ., data = train_data,
##     ranges = list(Cost = 0.01, 0.1, 0.5, 1, 10, 100), kernel = "linear",
##     scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  25
```

```
summary(svm_model)
```

```
##
## Call:
## svm(formula = Species ~ ., data = train_data, kernel = "linear",
##     Cost = 1, Cross = N, scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  25
##
##  ( 2 13 10 )
##
##
## Number of Classes:  3
##
## Levels:
##  setosa versicolor virginica
```

```
svm_model$index
```

```
##  [1]  19  34  42  44  46  47  54  56  58  62  63  64  69  70  79  86  92  98 100
## [20] 101 103 107 110 118 120
```

Use the test data to evaluate the model's performance.

```
predictions=predict(svm_model,test_data)
print(predictions)
```

```
##          1          2          6         16         23         34         35 
##     setosa     setosa     setosa     setosa     setosa     setosa     setosa 
##         38         44         47         51         60         64         73 
##     setosa     setosa     setosa versicolor versicolor versicolor versicolor 
##         74         87         91         94         95         97        104 
## versicolor versicolor versicolor versicolor versicolor versicolor  virginica 
##        109        111        113        116        120        127        134 
##  virginica  virginica  virginica  virginica  virginica  virginica versicolor 
##        138        147 
##  virginica  virginica 
## Levels: setosa versicolor virginica
```

```
# Create a confusion matrix
conf_matrix <- confusionMatrix(predictions, test_data$Species)

# View the confusion matrix and model performance
print(conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction   setosa versicolor virginica
##   setosa         10          0         0
##   versicolor      0         10         1
##   virginica       0          0         9
##
## Overall Statistics
##
##                Accuracy : 0.9667
##                  95% CI : (0.8278, 0.9992)
##     No Information Rate : 0.3333
##     P-Value [Acc > NIR] : 2.963e-13
##
##                   Kappa : 0.95
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: setosa Class: versicolor Class: virginica
## Sensitivity                 1.0000            1.0000           0.9000
## Specificity                 1.0000            0.9500           1.0000
## Pos Pred Value              1.0000            0.9091           1.0000
## Neg Pred Value              1.0000            1.0000           0.9524
## Prevalence                  0.3333            0.3333           0.3333
## Detection Rate              0.3333            0.3333           0.3000
```

```
## Detection Prevalence        0.3333          0.3667          0.3000
## Balanced Accuracy           1.0000          0.9750          0.9500
```

Mcnemar’s Test P-Value : NA: This test is used to compare two classifiers; if it’s NA, it may indicate that there was no need for comparison or insufficient data.

kappa A value close to 1 indicates almost perfect agreement.

```
# Tune SVM model to find the best cost and gamma parameters
tuned_model <- tune(svm, Species ~ ., data = train_data,
                    ranges = list(cost = 10^(-1:2), gamma = c(0.5, 1, 2)))

print(tuned_model)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##     1     1
##
## - best performance: 0.03333333
```

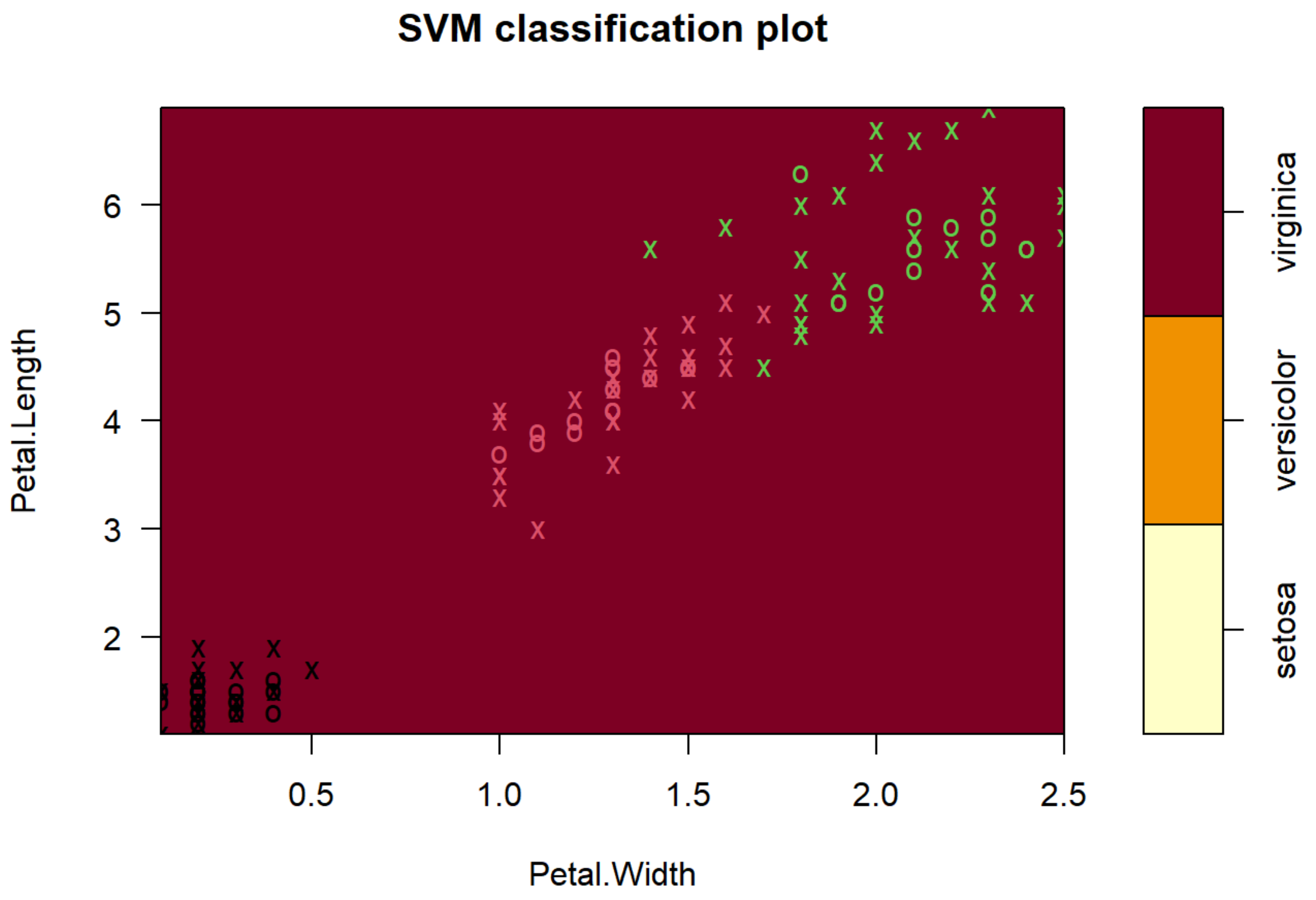
```
# View the best parameters
print(tuned_model$best.parameters)
```

```
##   cost gamma
## 6    1     1
```

```
# You can also experiment with other kernels like "radial", "polynomial", etc.
 svm_model3 <- svm(Species ~ ., data = train_data, kernel = "radial",Cost=0.4,scale=TRUE,gamma=2)
svm_model3
```

```
##
## Call:
## svm(formula = Species ~ ., data = train_data, kernel = "radial",
##     Cost = 0.4, gamma = 2, scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  70
```

```
plot(svm_model3, train_data, Petal.Length ~ Petal.Width)
```




```
summary( svm_model3)
```

```
##
## Call:
## svm(formula = Species ~ ., data = train_data, kernel = "radial",
##     Cost = 0.4, gamma = 2, scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  70
##
##  ( 16 27 27 )
##
##
## Number of Classes:  3
##
## Levels:
##  setosa versicolor virginica
```

```
svm_model3$index
```

```
##  [1]   3   4   6  11  12  15  17  19  20  21  27  28  30  34  35  36  42  43  44
## [20]  46  47  49  50  51  52  53  55  56  58  59  60  62  63  64  65  69  70  71
## [39]  72  74  75  77  79  81  85  86  88  89  90  91  92  93  94  96  97  98  99
## [58] 100 101 103 104 105 106 107 108 110 113 116 119 120
```

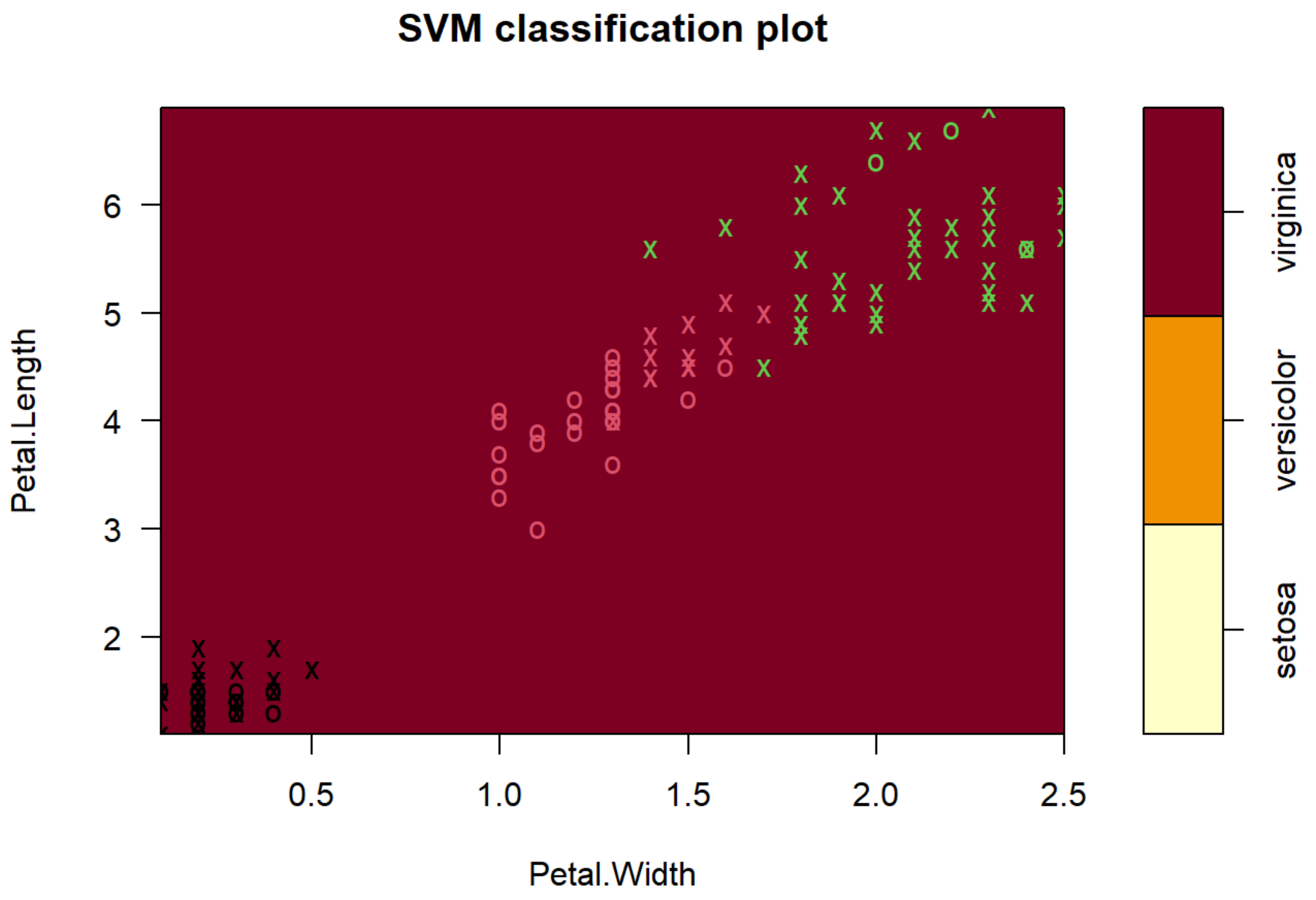
```
bestl=tune(svm,Species ~ ., data = train_data, kernel = "radial",range=list(Cost=0.01,0.1,0.5,1,10,10
0),scale=TRUE,gamma=c(0.5,1,2,10))
print(bestl)
```

```
##
## Error estimation of 'svm' using 10-fold cross validation: 0.04166667
```

```
 svm_model2 <- svm(Species ~ ., data = train_data, kernel = "polynomial",Cost=0.5,scale=TRUE,Cross=N,de
gree=2)
svm_model2
```

```
##
## Call:
## svm(formula = Species ~ ., data = train_data, kernel = "polynomial",
##     Cost = 0.5, Cross = N, degree = 2, scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  1
##      degree:  2
##      coef.0:  0
##
## Number of Support Vectors:  78
```

```
# Plotting SVM model with two features
plot(svm_model2, train_data, Petal.Length ~ Petal.Width)
```


SVM classification plot
Petal.Length
Petal.Width
0.5
1.0
1.5
2.0
2.5
2
3
4
5
6
virginica
versicolor
setosa

```
best1=tune(svm,Species ~ ., data = train_data, kernel = "polynomial",ranges=list(Cost=0.01,0.1,0.5,1,1
0,100),degree=c(1,2,3,4))
bestl=best1$best.model
best1
```

```
##
## Error estimation of 'svm' using 10-fold cross validation: 0.03333333
```

```
summary(svm_model2)
```

```
##
## Call:
## svm(formula = Species ~ ., data = train_data, kernel = "polynomial",
##     Cost = 0.5, Cross = N, degree = 2, scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  1
##      degree:  2
##      coef.0:  0
##
## Number of Support Vectors:  78
##
##  ( 25 16 37 )
##
##
## Number of Classes:  3
##
## Levels:
##  setosa versicolor virginica
```

```
svm_model2$index
```

```
##  [1]   1   2   6   7   9  10  11  15  17  19  20  21  22  25  26  27  29  31  32
## [20]  34  35  36  37  38  40  41  42  43  44  46  53  54  56  58  61  62  63  64
## [39]  69  70  75  81  82  83  84  85  86  87  88  89  90  91  92  94  95  96  97
## [58]  98  99 100 101 102 103 104 106 107 108 110 111 112 113 114 115 116 117 118
## [77] 119 120
```

```
# Make predictions on the test data
prediction1 <- predict(svm_model, newdata = test_data)
```

```
# Create a confusion matrix
conf_matrix <- confusionMatrix(prediction1, test_data$Species)

# View the confusion matrix and model performance
print(conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction   setosa versicolor virginica
##   setosa         10          0         0
##   versicolor      0         10         1
##   virginica       0          0         9
##
## Overall Statistics
##
##                Accuracy : 0.9667
##                  95% CI : (0.8278, 0.9992)
##     No Information Rate : 0.3333
##     P-Value [Acc > NIR] : 2.963e-13
##
##                   Kappa : 0.95
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: setosa Class: versicolor Class: virginica
## Sensitivity                 1.0000            1.0000           0.9000
## Specificity                 1.0000            0.9500           1.0000
## Pos Pred Value              1.0000            0.9091           1.0000
## Neg Pred Value              1.0000            1.0000           0.9524
## Prevalence                  0.3333            0.3333           0.3333
## Detection Rate              0.3333            0.3333           0.3000
```

```
## Detection Prevalence          0.3333          0.3667          0.3000
## Balanced Accuracy             1.0000          0.9750          0.9500
```