DATA ANALYTICS LAB

(AcademicYear: 2024-25) .Tech III Year – II Semester (R22)

DATA ANALYTICS LAB

Experiment 7 **Date :**

Experiment 1 : Write a R program to Implement ARIMA on Time Series data

Aim: Write a R program program to Implement ARIMA on Time Series data

Description :

ARIMA (Autoregressive Integrated Moving Average) is a statistical model used for time series analysis and forecasting, predicting future values by combining past observations (AR), differencing to achieve stationarity (I), and past errors to refine predictions (MA).

ARIMA models explain a given time series based on its own past values (lags) and lagged forecast errors.

Components:

Autoregressive (AR): This part of the model uses past values of the time series to predict future values.

Integrated (I): This component addresses non-stationarity by differencing the time series data, making it stationary (i.e., having a constant mean and variance over time).

Moving Average (MA): This part incorporates past forecast errors to improve the accuracy of future predictions.

Notation:

A non-seasonal ARIMA model is often represented as ARIMA(p, d, q), where:

p is the order of the autoregressive (AR) part.

d is the order of integration (the number of times the data needs to be differenced).

q is the order of the moving average (MA) part.

To build an ARIMA model:

Data Preparation: Collect and prepare the time series data.

Stationarity Check: Ensure the data is stationary or make it stationary through differencing.

Model Identification: Determine the appropriate values for p, d, and q using techniques like autocorrelation function (ACF) and partial autocorrelation function (PACF) plots.

Parameter Estimation: Estimate the model parameters using techniques like maximum likelihood estimation.

Model Evaluation: Evaluate the model's performance using metrics like root mean squared error (RMSE) or mean absolute error (MAE).

Steps involved in ARIMA Model :

1. Load and Prepare the Time Series Data

For demonstration, we use the built-in AirPassengers dataset.

2. Check for Stationarity

ARIMA requires a stationary series, meaning that statistical properties like mean and variance should be constant over time.

If p-value > 0.05, the data is non-stationary, and we apply differencing.

If p-value ≤ 0.05, the data is stationary.

3. Apply Differencing (If Necessary)

If the time series is non-stationary, differencing is required.

4. Identify ARIMA Parameters (p, d, q)

Determine ARIMA parameters manually using ACF (AutoCorrelation Function) and PACF (Partial AutoCorrelation Function) plots.

Applications:

ARIMA models are widely used for various time series forecasting tasks, including:
Predicting stock prices.
Forecasting sales and demand.
Analyzing financial data.
Understanding and predicting trends in various datasets

Program:

```
# Install required packages if not already installed
if (!require(forecast)) install.packages("forecast", dependencies = TRUE)
if (!require(tseries)) install.packages("tseries", dependencies = TRUE)
install.packages("forecast")


# Load necessary libraries
library(forecast)
library(tseries)
```

```r
# Load a sample time series dataset (AirPassengers dataset)
data(AirPassengers)
ts_data <- ts(AirPassengers, start = c(1949, 1), frequency = 12)

# Open a new plot window
dev.new()

# Plot the original time series data
plot(ts_data, main = "AirPassengers Time Series", ylab = "Passengers", xlab = "Year", col =
"blue")

# Check stationarity using Augmented Dickey-Fuller (ADF) test
adf_test <- adf.test(ts_data)
print(adf_test)

# Open a new plot window for ACF
dev.new()
acf(ts_data, main = "ACF Plot")

# Open a new plot window for PACF
dev.new()
pacf(ts_data, main = "PACF Plot")

# If the series is non-stationary, apply first-order differencing
if (adf_test$p.value > 0.05) {
  ts_data_diff <- diff(ts_data, differences = 1)  # Keep the original ts_data unchanged
  print("Differencing applied to make the series stationary.")
} else {
  ts_data_diff <- ts_data
}

# Re-check stationarity after differencing
adf_test_diff <- adf.test(ts_data_diff, na.action = na.omit)
print(adf_test_diff)

# Determine the best ARIMA model automatically
best_model <- auto.arima(ts_data)  # Use original ts_data for ARIMA fitting

# Print model summary
summary(best_model)

# Forecast for the next 12 months
forecast_values <- forecast(best_model, h = 12)
```

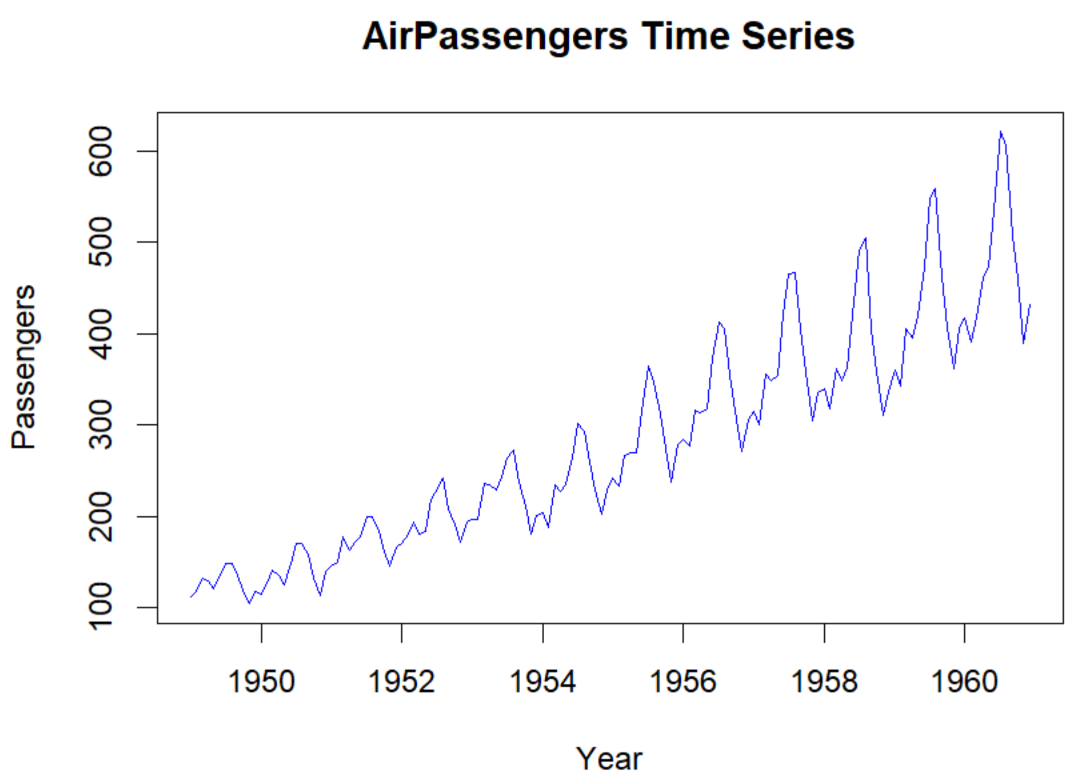
# Open a new plot window for forecast
dev.new()
plot(forecast_values, main = "ARIMA Forecast", col = "blue")

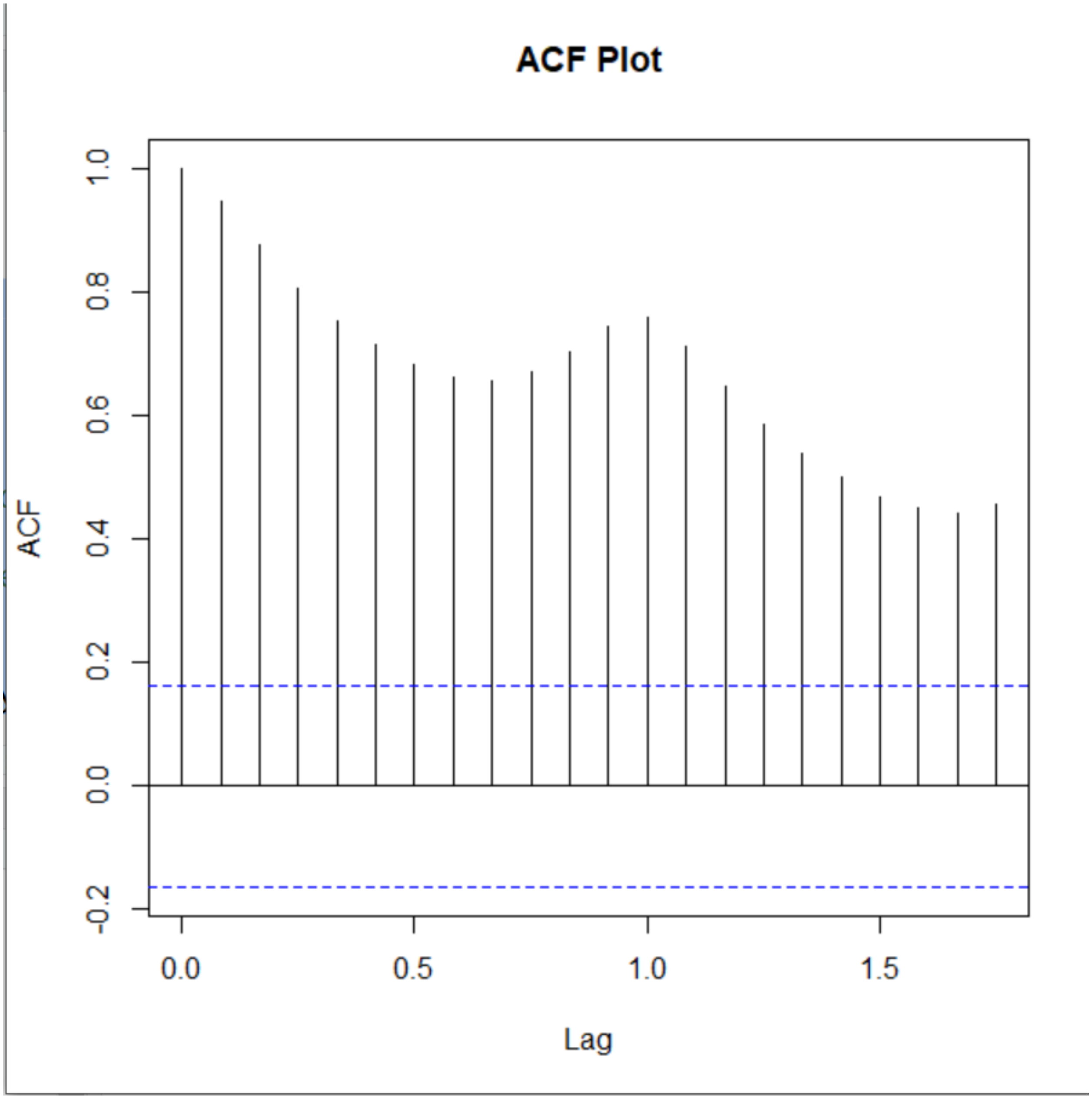
# Print forecasted values
print(forecast_values)

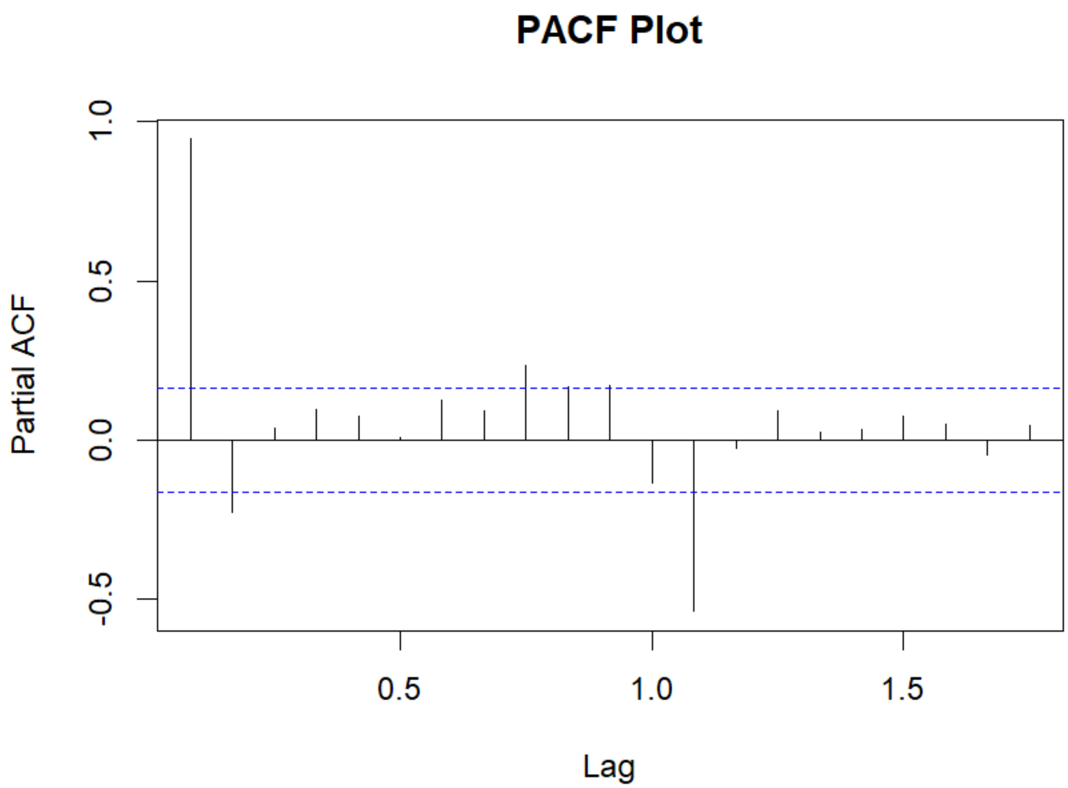
# Check residuals to validate the model
checkresiduals(best_model)

Output :

```
Augmented Dickey-Fuller Test

data:  ts_data
Dickey-Fuller = -7.3186, Lag order = 5, p-value =
0.01
alternative hypothesis: stationary
```




ACF Plot
ACF
1.0
0.8
0.6
0.4
0.2
0.0
-0.2
0.0
0.5
1.0
1.5
Lag


PACF Plot
Partial ACF
1.0
0.5
0.0
-0.5
0.5
1.0
1.5
Lag

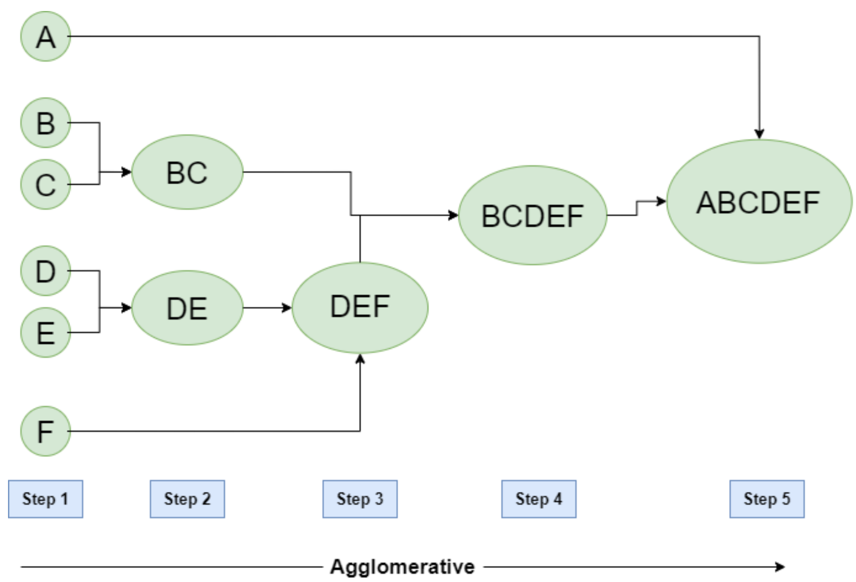
EXPERIMENT 8 Date :

Write R program for Object segmentation using hierarchical based methods

AIM: To implement hierarchical based methods

Description : Hierarchical clustering is a technique used to group similar data points together based on their similarity creating a hierarchy or tree-like structure

A dendrogram is like a family tree for clusters. It shows how individual data points or groups of data merge together.



Types of Hierarchical Clustering

1. Agglomerative Clustering
2. Divisive clustering

Workflow for Hierarchical Agglomerative clustering

1. Start with individual points
2. Calculate distances between clusters
3. Merge the closest (smallest distance) clusters
4. Update distance matrix
5. Repeat steps 3 and 4 until only one cluster left.
6. Create a dendrogram

Proogram :

```
# Finding distance matrix
distance_mat <- dist(mtcars, method = 'euclidean')
distance_mat
```

```
# Fitting Hierarchical clustering Model
# to training dataset
set.seed(240) # Setting seed
Hierar_cl <- hclust(distance_mat, method = "average")
Hierar_cl

# Plotting dendrogram
plot(Hierar_cl)

# Choosing no. of clusters
# Cutting tree by height
abline(h = 110, col = "green")

# Cutting tree by no. of clusters
fit <- cutree(Hierar_cl, k = 3 )
fit

table(fit)
rect.hclust(Hierar_cl, k = 3, border = "green")
```

OUTPUT:

Distance matrix:

> distance_mat

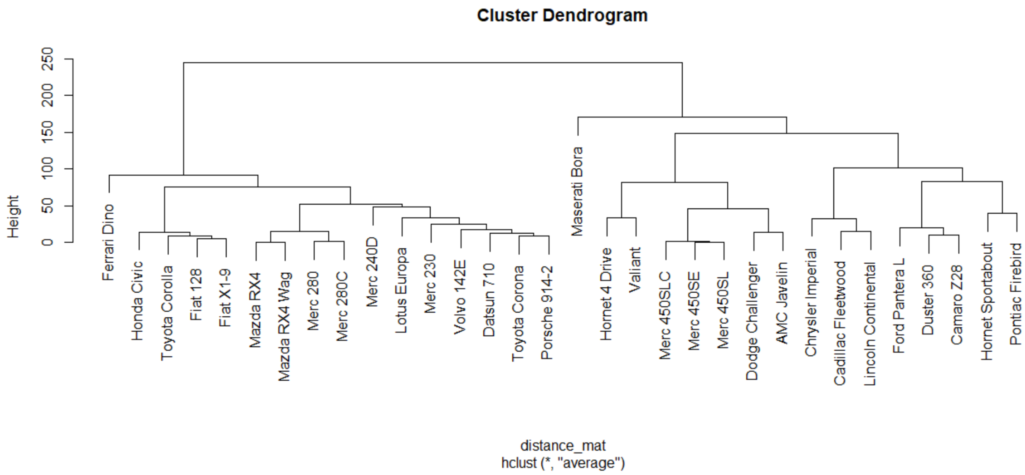
	Mazda RX4	Mazda RX4 Wag	Datsun 710	Hornet 4 Drive	Hornet Sportabout
Mazda RX4 Wag	0.6153251				
Datsun 710	54.9086059	54.8915169			
Hornet 4 Drive	98.1125212	98.0958939	150.9935191		
Hornet Sportabout	210.3374396	210.3358546	265.0831615	121.0297564	
Valiant	65.4717710	65.4392224	117.7547018	33.5508692	152.1241352
Duster 360	241.4076490	241.4088680	294.4790230	169.4299647	70.1767262
Merc 240D	50.1532711	50.1146059	49.6584796	121.2739722	241.5069657
Merc 230	25.4683117	25.3284509	33.1803843	118.2433145	233.4924012
Merc 280	15.3641921	15.2956865	66.9363534	91.4224033	199.3344960
Merc 280C	15.6724727	15.5837744	67.0261397	91.4612914	199.3406564
Merc 450SE	135.4307018	135.4254826	189.1954941	72.4964325	84.3888482
Merc 450SL	135.4014424	135.3960351	189.1631745	72.4313532	84.3683999
Merc 450SLC	135.4794674	135.4723157	189.2345426	72.5718466	84.4332423
Cadillac Fleetwood	326.3395903	326.3355070	381.0926242	234.4403876	116.2804201
Lincoln Continental	318.0469808	318.0429333	372.8012090	227.9726091	108.0624299
Chrysler Imperial	304.7203408	304.7169175	359.3014906	218.1548299	97.2049146
Fiat 128	93.2679950	93.2530993	40.9933763	184.9689734	302.0377212
Honda Civic	102.8307567	102.8238713	52.7704607	191.5518700	310.0324645
Toyota Corolla	100.6040368	100.5887588	47.6535017	192.6714187	309.5581776
Toyota Corona	42.3075233	42.2659224	12.9654743	138.5304725	252.3331988
Dodge Challenger	163.1150750	163.1134210	217.7795805	72.4403915	48.9838851
AMC Javelin	149.6047203	149.6014522	204.3188913	61.3601899	61.4274240
Camaro Z28	233.2228758	233.2248748	286.0049209	163.6632641	70.9665308
Pontiac Firebird	248.6780270	248.6762035	303.3583889	156.2240346	40.0052475
Fiat X1-9	92.5048389	92.4940020	39.8815148	184.4471198	301.5669483
Porsche 914-2	44.4033659	44.4073589	13.1357109	139.1579524	254.1452553
Lotus Europa	65.7328377	65.7362635	25.0948550	163.2367437	272.3582423
Ford Pantera L	245.4247064	245.4293785	297.2940489	180.1140339	89.5934049
Ferrari Dino	66.7661029	66.7764167	90.2415509	130.5523007	215.0673853
Maserati Bora	265.6454248	265.6491465	309.7718171	229.3419352	170.7094473
Volvo 142E	39.1894029	39.1626037	20.6939436	137.0363299	248.0063378

- The values are shown as per the distance matrix calculation with the method as euclidean.

- **Model Hierar_cl:**

```
> Hierar_cl

Call:
hclust(d = distance_mat, method = "average")

Cluster method   : average
Distance         : euclidean
Number of objects: 32
```

- In the model, the cluster method is average, distance is euclidean and no. of objects are 32.



- So, Tree is cut where k = 3 and each category represents its number of clusters.

EXPERIMENT 9 Date :

Write R program for Perform Visualization techniques (types of maps - Bar, Colum, Line, Scatter, 3D Cubes etc)

AIM: To implement data Visualization techniques (tBar, Colum, Line, Scatter, 3D Cubes etc)

Consider the following airquality data set for visualization in R:

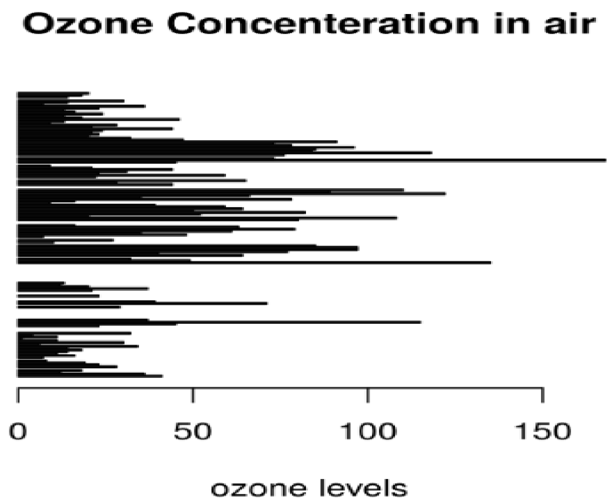
Ozone	Solar R.	Wind	Temp	Month	Day
41	190	7.4	67	5	1
36	118	8.0	72	5	2
12	149	12.6	74	5	3
18	313	11.5	62	5	4
NA	NA	14.3	56	5	5
28	NA	14.9	66	5	6

a) AIM: To implement Bar Graph using R

PROGRAM:

```
# Horizontal Bar Plot for
# Ozone concentration in air
barplot(airquality$Ozone,
        main = 'Ozone Concenteration in air',
        xlab = 'ozone levels', horiz = TRUE)
```

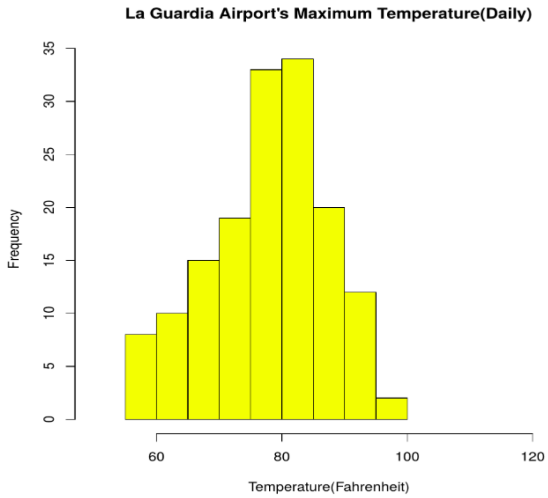
OUTPUT:



b) AIM: To implement Histogram

```
# Histogram for Maximum Daily Temperature
data(airquality)

hist(airquality$Temp, main ="La Guardia Airport's\
Maximum Temperature(Daily)",
        xlab ="Temperature(Fahrenheit)",
        xlim = c(50, 125), col ="yellow",
        freq = TRUE)
```

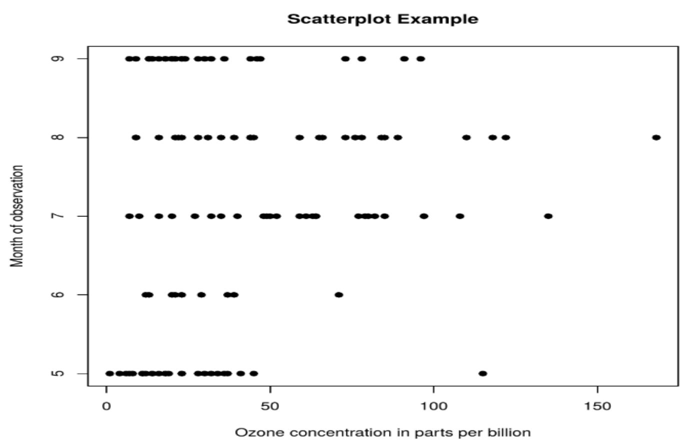
OUTPUT:



c) AIM: To implement scatter graph

Program:

```
# Scatter plot for Ozone Concentration per month
data(airquality)
plot(airquality$Ozone, airquality$Month,
main ="Scatterplot Example",
xlab ="Ozone Concentration in parts per billionn",
ylab =" Month of observation ", pch = 19)
```



d) AIM: To implement line graph

PROGRAM:

```
# Create the data for the chart.
v <- c(17, 25, 38, 13, 41)
t <- c(22, 19, 36, 19, 23)
m <- c(25, 14, 16, 34, 29)
# Plot the bar chart.
plot(v, type = "o", col = "red",
```

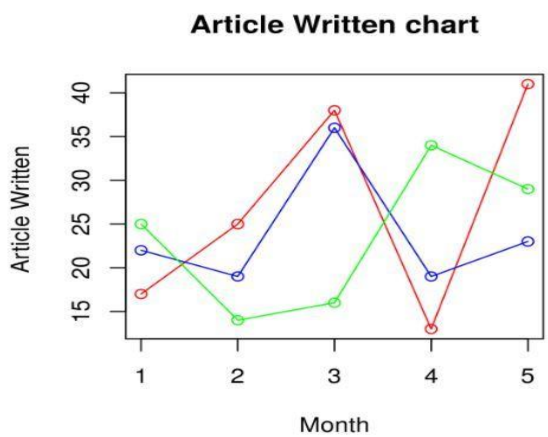
xlab = "Month", ylab = "Article Written ",

main = "Article Written chart")

lines(t, type = "o", col = "blue")

lines(m, type = "o", col = "green")

OUTPUT:

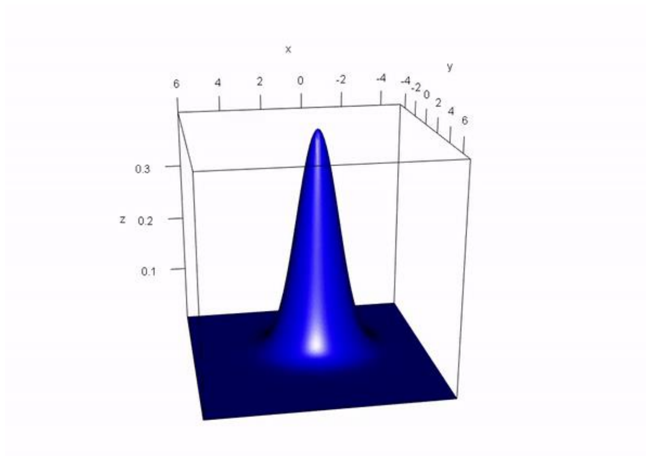


e) AIM: To implement 3D plots graph

PROGRAM:

```
# import and load rgl package
install.packages("rgl")
library(rgl)
# Generate some sample data
x <- seq(-5, 6, by = 0.1)
y <- seq(-5, 7, by = 0.1)
z <- outer(x, y, function(x, y) dnorm(sqrt(x^2 + y^2)))
# Create a 3D surface plot
persp3d(x, y, z, col = "blue")

# add animation
```

```
play3d(spin3d(axis = c(0, 0, 1)), duration = 10)
```



Experiment 10 Date :

Write a R program to perform Descriptive analytics on healthcare data

AIM: To implement Descriptive Analytics on healthcare data

Description : Descriptive analytics is the process of summarizing and interpreting historical data to understand what has happened in the past.

Goals of Descriptive Analytics in Healthcare :

- Understand patient demographics (e.g., age, gender distribution)
- Analyze clinical metrics like BMI(Body mass index), blood pressure, cholesterol
- Identify prevalence of conditions like diabetes, hypertension
- Spot trends over time (e.g., increasing obesity rates)
- Evaluate resource use (e.g., hospital admissions, medication use)

Program :

```
# Install required packages if not already installed
if (!require("summarytools")) install.packages("summarytools", dependencies = TRUE)
# Load the libraries
library(rgl)
library(dplyr)
library(summarytools)

# Try to load the data from CSV, if not found, create a sample dataset
file_path <- "health_data.csv"

if (!file.exists(file_path)) {
  message("File not found. Creating sample dataset...")
  set.seed(123)
  health_data <- data.frame(
    Age = sample(20:80, 100, replace = TRUE),
    Gender = factor(sample(c("Male", "Female"), 100, replace = TRUE)),
    BMI = round(runif(100, 18, 35), 1),
    BloodPressure = sample(90:180, 100, replace = TRUE),
    Cholesterol = sample(150:300, 100, replace = TRUE),
    Diabetes = factor(sample(c("Yes", "No"), 100, replace = TRUE))
  )
  write.csv(health_data, file_path, row.names = FALSE)
} else {
  health_data <- read.csv(file_path, stringsAsFactors = TRUE)
}
```

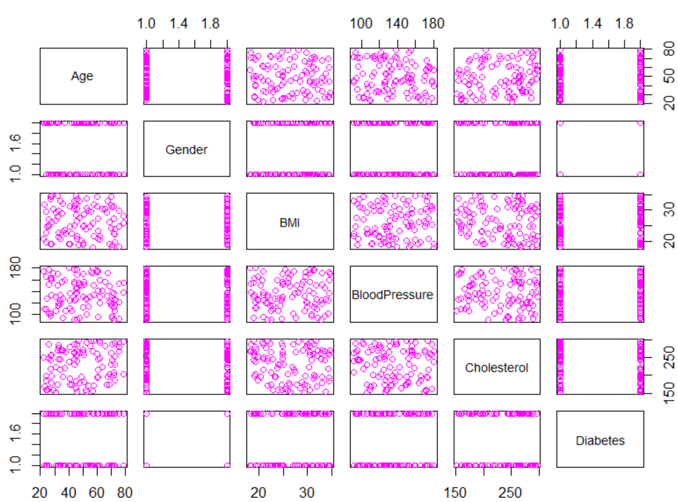
```
# View structure
str(health_data)

# Summary statistics for numeric variables
numeric_vars <- select(health_data, where(is.numeric))
summary(numeric_vars)

# Frequency tables for categorical variables
cat_vars <- select(health_data, where(is.factor))
lapply(cat_vars, table)

# Cross-tabulation: Gender vs Diabetes
print(table(health_data$Gender, health_data$Diabetes))

# Descriptive report using summarytools
print(dfSummary(health_data), method = "browser")

dev.new()
# 1. Plot Healthcare Attributes
plot(health_data, col ="magenta",)

dev.new()
# Plots
# 2. Age Distribution
hist(health_data$Age, main =" Age Distribution ",
    xlab ="Age(in Yeaars)",
    xlim = c(0, 125), col = "green",
    freq = TRUE)
```

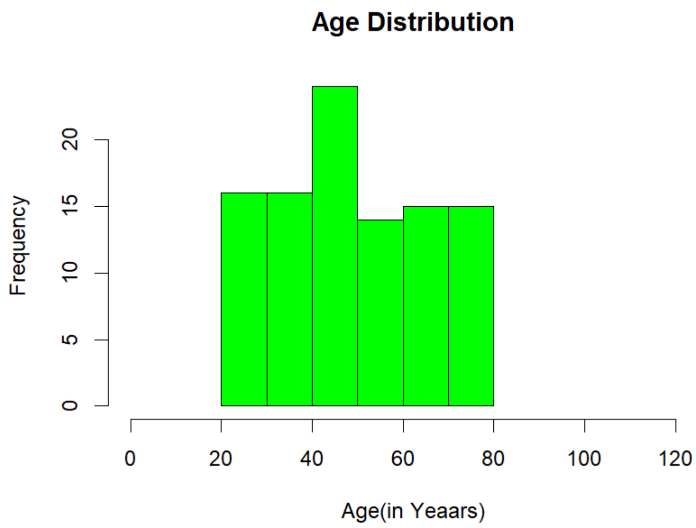
Output :

```
'data.frame':    100 obs. of  6 variables:

 $ Age          : int  50 34 70 33 22 61 69 73 62 56 ...
 $ Gender       : Factor w/ 2 levels "Female","Male": 1 1 2 2 2
1 1 1 1 2 ...
 $ BMI          : num  24.8 33 24.2 22.9 20.9 20.9 26.2 22.3 21.
7 29.5 ...
 $ BloodPressure: int  170 118 115 116 174 96 149 115 130 173 ..
.
 $ Cholesterol  : int  264 270 231 245 251 245 175 290 297 297 .
..
 $ Diabetes     : Factor w/ 2 levels "No","Yes": 1 2 1 2 2 1 2 1
 1 1 ...
```

```
> summary(numeric_vars)
      Age             BMI         BloodPressure
 Min.   :22.00   Min.   :18.10   Min.   : 91.0
 1st Qu.:34.00   1st Qu.:21.80   1st Qu.:115.8
 Median :47.50   Median :26.05   Median :133.5
 Mean   :49.19   Mean   :26.26   Mean   :135.8
 3rd Qu.:62.00   3rd Qu.:30.57   3rd Qu.:156.0
 Max.   :79.00   Max.   :34.70   Max.   :180.0
  Cholesterol
 Min.   :152.0
 1st Qu.:193.5
 Median :243.0
 Mean   :230.8
 3rd Qu.:268.0
 Max.   :297.0

$Gender
Female   Male
    55     45

$Diabetes
 No Yes
 52  48

> # Cross-tabulation: Gender vs Diabetes
         No Yes
  Female 32  23
  Male   20  25
```

Graph of Healthcare Attributes




Age Distribution
Frequency
0
5
10
15
20
0
20
40
60
80
100
120
Age(in Yeaars)

EXPERIMENT 11 Date :

Write a R program to Perform Predictive analytics on Product Sales data methods

AIM: To Perform Predictive analytics on Product Sales data

Description : Predictive analytics uses historical data and statistical modeling to forecast future outcomes, determining the likelihood of specific events or trends

Steps in Predictive analytics :

1. Focus on the Future: Predictive analytics is not about understanding past events, but about anticipating future trends and outcomes.

2. Data-Driven: It relies heavily on data, both current and historical, to identify patterns and relationships that can be used to make predictions.

3. Statistical Modeling: Techniques like regression analysis, time series analysis, and machine learning algorithms are used to build models that can predict future outcomes.

4. Decision Support: The predictions generated by predictive analytics can be used to make informed decisions, such as optimizing business processes, identifying risks, or forecasting demand.

Example:

Sales Forecasting: Predicting future sales based on historical sales data, market trends, and promotional activities.

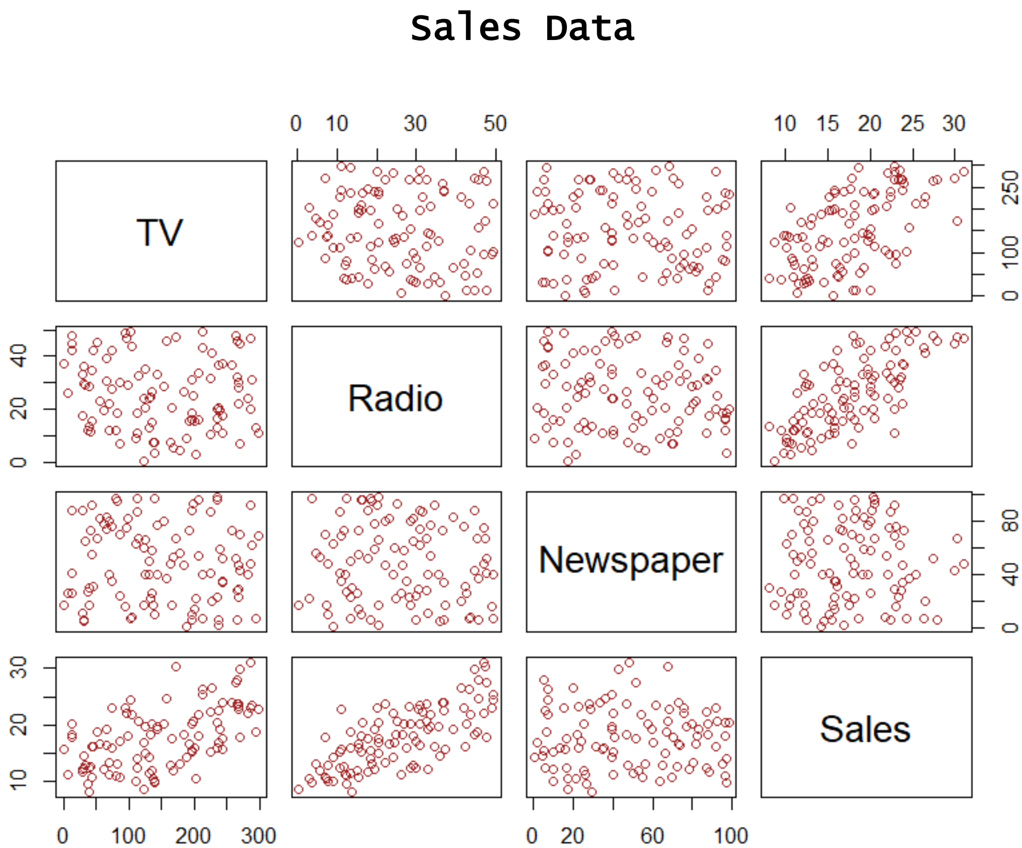
Proogram :

```
# Load the data
sales_data <- read.csv("product_sales.csv")
head(sales_data )


head(sales_data )

dev.new()
plot(sales_data, col="brown")


# Convert 'Season' to a factor
sales_data$Season <- as.factor(sales_data$Season)

# Split data into training (80%) and testing (20%)
set.seed(123)
sample_index <- sample(1:nrow(sales_data), 0.8 * nrow(sales_data))
train_data <- sales_data[sample_index, ]
test_data <- sales_data[-sample_index, ]
```

```
# Build a linear regression model
model <- lm(Sales ~ Price + Advertising + Season, data = train_data)

# Predict on test data
predicted_sales <- predict(model, newdata = test_data)

# Calculate RMSE
rmse <- sqrt(mean((test_data$Sales - predicted_sales)^2))
cat("Linear Regression RMSE:", rmse, "\n")
summary(model)


dev.new()

# -------------------------
# Plot: Actual vs Predicted
# -------------------------
plot(test_data$Sales, predicted_sales,
     main = "Actual vs Predicted Sales",
     xlab = "Actual Sales",
     ylab = "Predicted Sales",
     col = "blue",
     pch = 19)
abline(a = 0, b = 1, col = "red", lwd = 2)  # Reference line
```

OUTPUT:

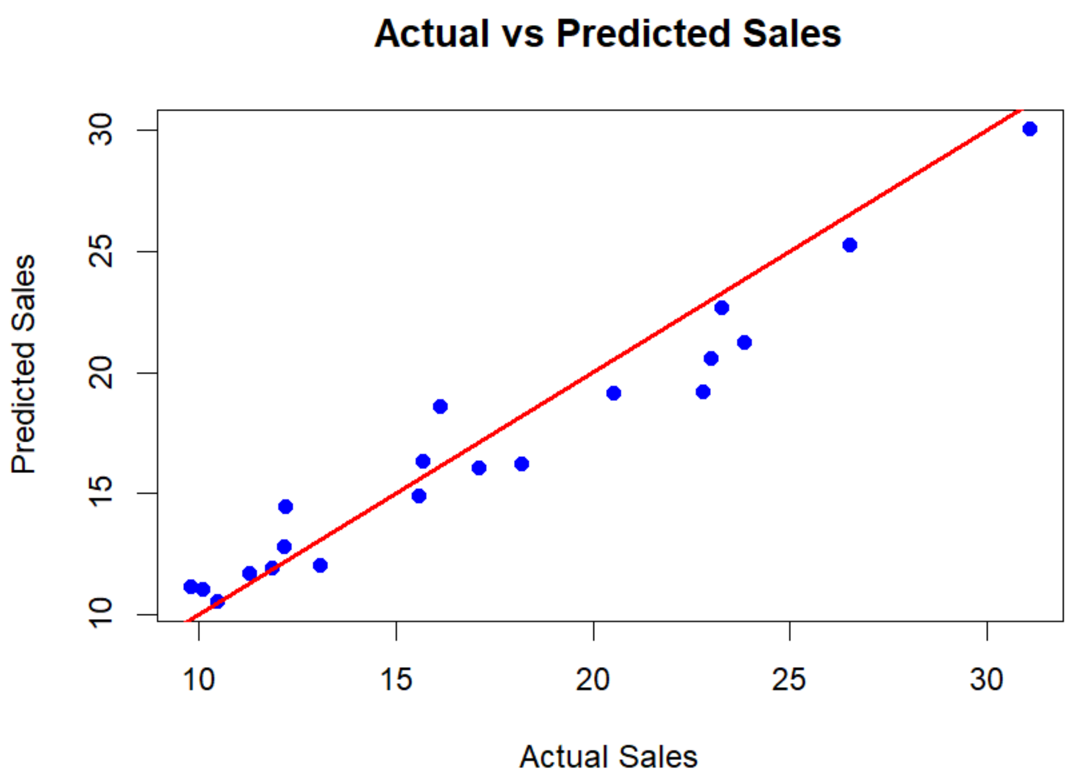
```
     TV   Radio   Newspaper   Sales
1  86.3  30.0      23.9    17.11161
2 236.5  16.6      96.2    20.51756
3 122.7  24.4      60.1    16.92830
4 264.9  47.7      51.5    27.42344
5 282.1  24.1      40.3    22.14082
6  13.7  44.5      88.0    18.23741
```




```
Linear Regression RMSE: 1.609265
```

```
summary(model)
```

```
Residuals:
    Min      1Q  Median      3Q     Max 
-2.4771 -0.9458 -0.1164  0.8441  4.2106 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) 2.316178   0.562091   4.121 9.54e-05 ***
TV          0.039634   0.002340  16.940  < 2e-16 ***
Radio       0.328001   0.012074  27.166  < 2e-16 ***
Newspaper   0.021045   0.005086   4.138 8.98e-05 ***
---

Residual standard error: 1.384 on 76 degrees of freedom
Multiple R-squared:  0.9189,	Adjusted R-squared:  0
.9157 
F-statistic:   287 on 3 and 76 DF,  p-value: < 2.2e-16
```


Actual vs Predicted Sales
Predicted Sales
Actual Sales
10
15
20
25
30
10
15
20
25
30

EXPERIMENT 12 Date :

Write a R program to Apply Predictive analytics for Weather forecasting.

AIM: To Apply Predictive analytics for Weather forecasting.

Description : Predictive analytics uses historical data and statistical modeling to forecast future outcomes, determining the likelihood of specific events or trends

Weather forecasting : It involves a sequence of steps

Step 1 : Data Collection : Data collects from Ground stations like Temperature, humidity, wind speed, pressure, rainfall, etc.

Step 2 : Data Preprocessing & Quality Control : Raw data is often noisy or incomplete, so: Missing values are estimated or removed.
Outliers are detected and handled.

Step 3 : Feature Engineering : To improve model performance:
Create derived variables (e.g., wind chill, heat index).
Convert date/time into cyclical features.
Convert categorical data (e.g., weather types) into numeric encodings.

Step 4 : . Model Building : Use Statistical or Machine Learning Models:
Use historical data to learn patterns.
Common methods: Linear regression for temperature
SVM / Decision Trees for classification (e.g., rain prediction)
Time series models like ARIMA, LSTM

Step 5 : . Model Evaluation :
Models are tested using Training/testing split or cross-validation
Metrics like: RMSE / MAE for temperature
Accuracy, Precision, Recall for rain or storm predictions

Step 6 : . Forecasting : Forecasts are generated for:
Short-term (1–3 days): Highly accurate
Medium-term (4–7 days): Good reliability
Long-term (>7 days): Increasing uncertainty

Forecasts may include: Temperature, Rainfall likelihood, Wind speed and direction, Storm alerts

Proogram :

```
# Load libraries
library(lubridate)
library(e1071)

# Load data
weather_data <- read.csv("weather_data.csv", stringsAsFactors = FALSE)

# Initial plot
dev.new()
plot(weather_data, main = "Weather Dataset", col = "green")

# Parse date and extract features
weather_data$date <- ymd(weather_data$date)
weather_data$day_of_year <- yday(weather_data$date)
weather_data$month <- month(weather_data$date)
weather_data$weekday <- wday(weather_data$date)

# Create rain label if applicable
if (!"rain_label" %in% names(weather_data) && "rain" %in% names(weather_data))
{
  weather_data$rain_label <- as.factor(ifelse(weather_data$rain > 0, "Yes", "No"))
}

# Split into training and testing sets
set.seed(123)
sample_size <- floor(0.8 * nrow(weather_data))
train_indices <- sample(seq_len(nrow(weather_data)), size = sample_size)
train_data <- weather_data[train_indices, ]
test_data <- weather_data[-train_indices, ]

# Train temperature model
model_temp <- lm(temperature ~ humidity + pressure + day_of_year + month +
weekday, data = train_data)
train_data$predicted_temp <- predict(model_temp, newdata = train_data)

# Train SVM model for rain prediction if applicable
rain_model_exists <- FALSE
if ("rain_label" %in% names(weather_data)) {
  model_rain_svm <- svm(rain_label ~ humidity + predicted_temp + pressure +
day_of_year + month + weekday,
                        data = train_data, type = "C-classification", kernel = "radial")
  rain_model_exists <- TRUE
}

# Evaluate temperature model
```

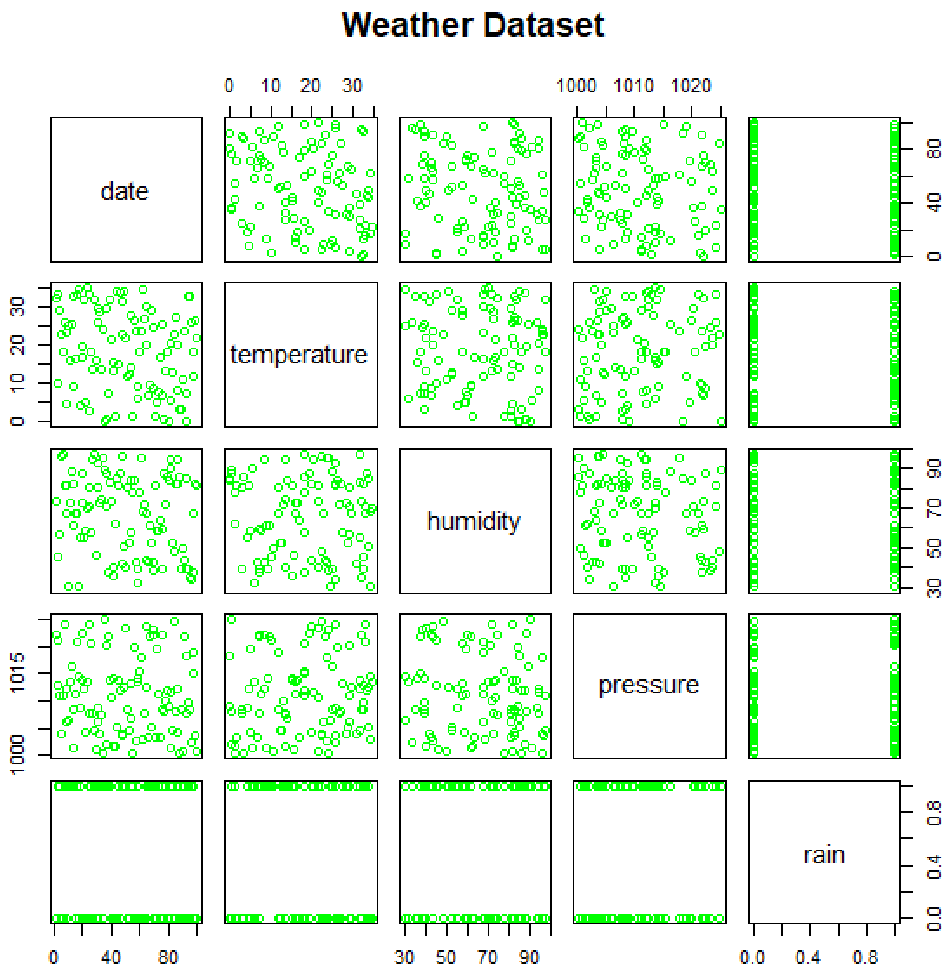
```
predictions <- predict(model_temp, newdata = test_data)
rmse <- sqrt(mean((test_data$temperature - predictions)^2, na.rm = TRUE))
cat("Root Mean Squared Error (RMSE):", round(rmse, 2), "\n")

# Summary
summary(model_temp)
if (rain_model_exists) print(summary(model_rain_svm))

# Plot actual vs predicted temperature
dev.new()
plot(test_data$temperature, predictions,
     col = "blue", pch = 16,
     main = "Actual vs Predicted Temperature",
     xlab = "Actual Temperature", ylab = "Predicted Temperature")
abline(0, 1, col = "red", lwd = 2)

# --- Future Forecasting ---

# Generate next 7 days
future_dates <- seq(max(weather_data$date) + 1, by = "day", length.out = 7)

# Create future data frame
future_data <- data.frame(
  date = future_dates,
  day_of_year = yday(future_dates),
  month = month(future_dates),
  weekday = wday(future_dates),
  humidity = mean(train_data$humidity, na.rm = TRUE),
  pressure = mean(train_data$pressure, na.rm = TRUE)
)

# Ensure factor compatibility
if (is.factor(train_data$weekday)) {
  future_data$weekday <- factor(future_data$weekday, levels =
levels(train_data$weekday))
}

# Predict temperature
future_data$predicted_temp <- predict(model_temp, newdata = future_data)
future_data$predicted_temperature <- round(future_data$predicted_temp, 2)

# Predict rain if model exists
if (rain_model_exists) {
  raw_preds <- predict(model_rain_svm, newdata = future_data)
  raw_preds <- as.character(raw_preds)
```

```
    future_data$rain_prediction <- ifelse(raw_preds == "Yes", "RAIN=YES",
"RAIN=NO")
  } else {
    future_data$rain_prediction <- rep("RAIN=NA", nrow(future_data))
  }

  # Display forecast
  cat("\nNext 7 Days Forecast:\n")
  print(future_data[, c("date", "predicted_temperature", "rain_prediction")])

  # Plot forecast
  dev.new()
  plot(future_data$date, future_data$predicted_temperature, type = "o",
       col = "red", lwd = 4, pch = 5,
       main = "7-Day Forecast: Temperature & Rain",
       xlab = "Date", ylab = "Temperature (°C)",
       ylim = range(future_data$predicted_temperature, na.rm = TRUE) + c(-1, 2))
  grid()

  # Add temperature labels
  text(future_data$date, future_data$predicted_temperature + 0.4,
       labels = future_data$predicted_temperature,
       col = "red", cex = 1)

  # Add rain prediction labels with color mapping
  rain_colors <- ifelse(future_data$rain_prediction == "RAIN=YES", "blue",
                        ifelse(future_data$rain_prediction == "RAIN=NO", "magenta",
"yellow"))

  text(future_data$date, future_data$predicted_temperature + 1,
       labels = future_data$rain_prediction,
       col = rain_colors, font = 2, cex = 0.9)

  # Add legend
  legend("topright", legend = c("Temperature (°C)", "RAIN=YES", "RAIN=NO",
"RAIN=NA"),
         col = c("red", "blue", "magenta", "yellow"),
         pch = 16, bty = "n")

  # --- Bonus: Plot raw temperature over time with rain indicators ---
  dev.new()
  plot(weather_data$date, weather_data$temperature, type = "o",
       col = "magenta", lwd = 2, pch = 16,
       main = "Temperature Over Time",
       xlab = "Date", ylab = "Temperature (°C)")
  points(weather_data$date[weather_data$rain > 0],
```

```
        weather_data$temperature[weather_data$rain > 0],
        col = "blue", pch = 17, cex = 1.2)
legend("topright",
        legend = c("Temperature", "Rainy Days"),
        col = c("magenta", "blue"),
        pch = c(16, 17),
        bty = "n")
```

OUTPUT:

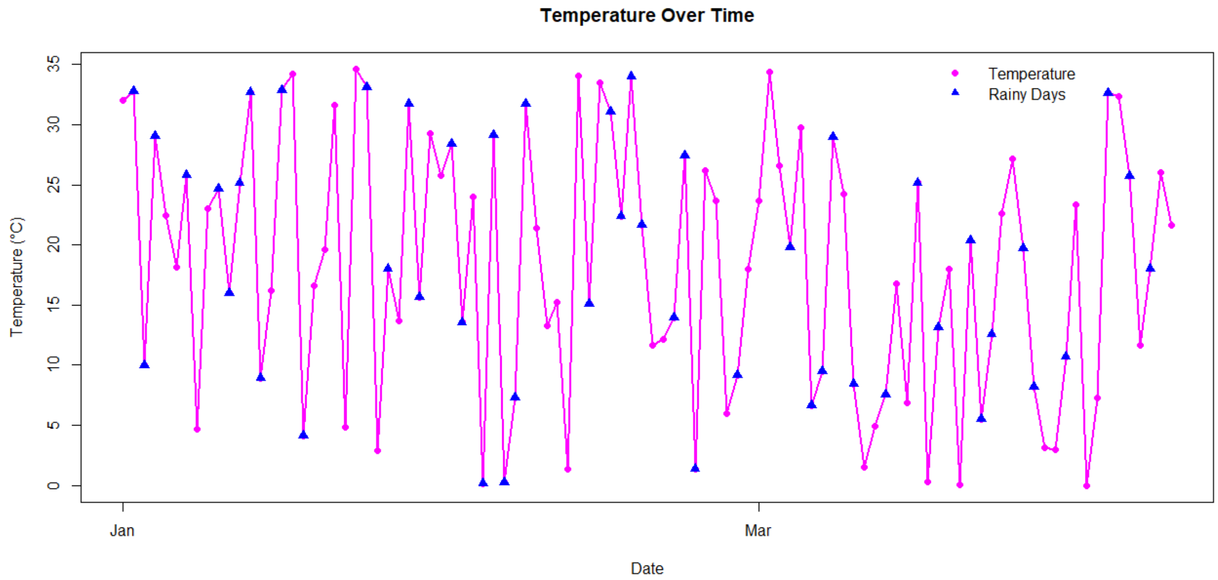
```
> head(weather_data )

        date temperature humidity pressure rain
1 2024-01-01    32.01821 73.83717 1022.128    0
2 2024-01-02    32.79764 45.20104 1012.928    1
3 2024-01-03    10.01488 45.15971 1021.298    1
4 2024-01-04    29.06567 57.22615 1011.070    1
5 2024-01-05    22.46109 95.97190 1003.947    0
6 2024-01-06    18.16836 97.38256 1011.058    0
```

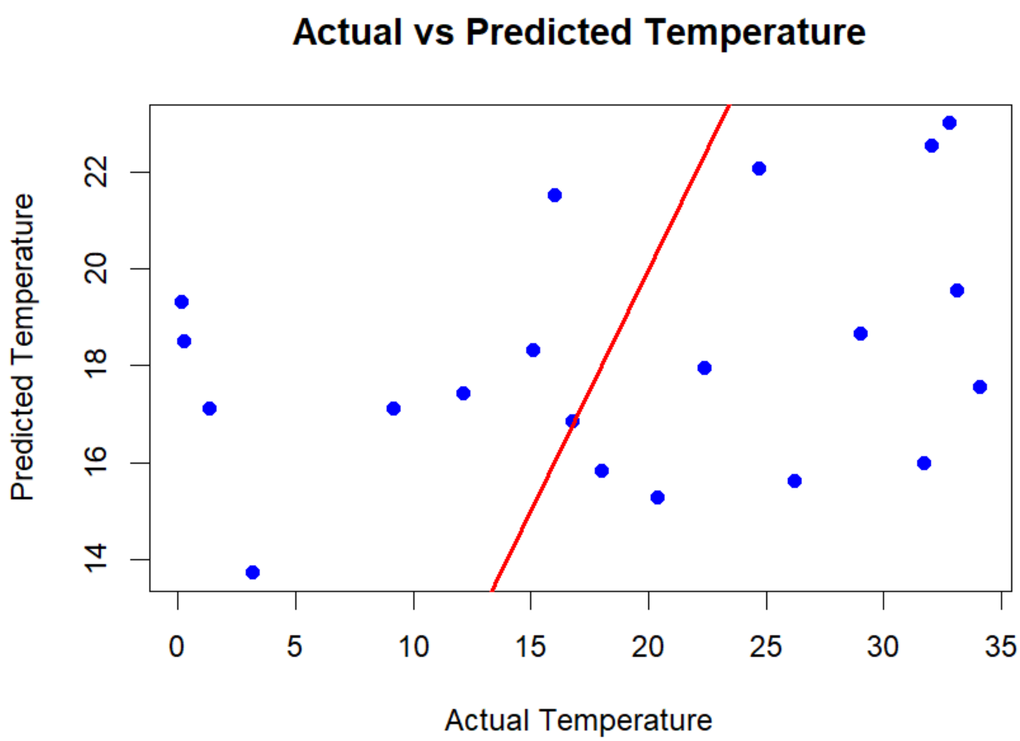



```
Root Mean Squared Error (RMSE): 10.86

> #Summary of Models
> summary(model_temp)

Call:
lm(formula = temperature ~ humidity + pressure + day_of
_year +
    month + weekday, data = train_data)

Residuals:
     Min       1Q   Median       3Q      Max
-16.4739  -9.5256   0.7116   8.9975  16.6716
```

```
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -35.73927  174.87503  -0.204    0.839
humidity     -0.02731    0.06225  -0.439    0.662
pressure      0.05533    0.17191   0.322    0.748
day_of_year  -0.17027    0.13703  -1.243    0.218
month         3.19049    3.97729   0.802    0.425
weekday       0.36879    0.58653   0.629    0.531

Residual standard error: 10.41 on 74 degrees of freedom
Multiple R-squared:  0.05175,   Adjusted R-squared:  -
0.01232
F-statistic: 0.8077 on 5 and 74 DF,  p-value: 0.5479
```

```
> summary(model_rain_svm)

Call:
svm(formula = rain_label ~ humidity + predicted_temp +
    pressure + day_of_year + month + weekday,
    data = train_data, type = "C-classification",
    kernel = "radial")

Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1

Number of Support Vectors:  72

 ( 38 34 )
Number of Classes:  2

Levels:
 No Yes
```


Temperature Over Time
Temperature
Rainy Days
Temperature (°C)
0
5
10
15
20
25
30
35
Jan
Mar
Date


Actual vs Predicted Temperature
Predicted Temperature
14
16
18
20
22
0
5
10
15
20
25
30
35
Actual Temperature

```
Next 7 Days Forecast:

        date predicted_temperature rain_prediction
1 2024-04-10                 15.47         RAIN=NO
2 2024-04-11                 15.67         RAIN=NO
3 2024-04-12                 15.87         RAIN=NO
4 2024-04-13                 16.07         RAIN=NO
5 2024-04-14                 13.68         RAIN=NO
6 2024-04-15                 13.88         RAIN=NO
7 2024-04-16                 14.08         RAIN=NO
```