PROGRAMMING OF DATA SCIENCE THEORY DA

ANKUR ROHILLA

21BDS0311

Domain 1: Time Series Analysis

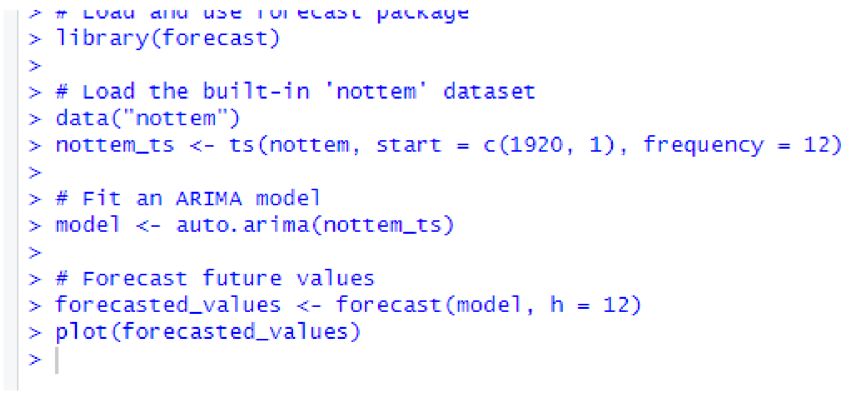
1. Popular Packages for Time Series Analysis

Time series analysis involves working with time-ordered data to identify trends, seasonality, and cyclical patterns. Several R packages help in analyzing and modeling time series data. Here are seven widely used ones:

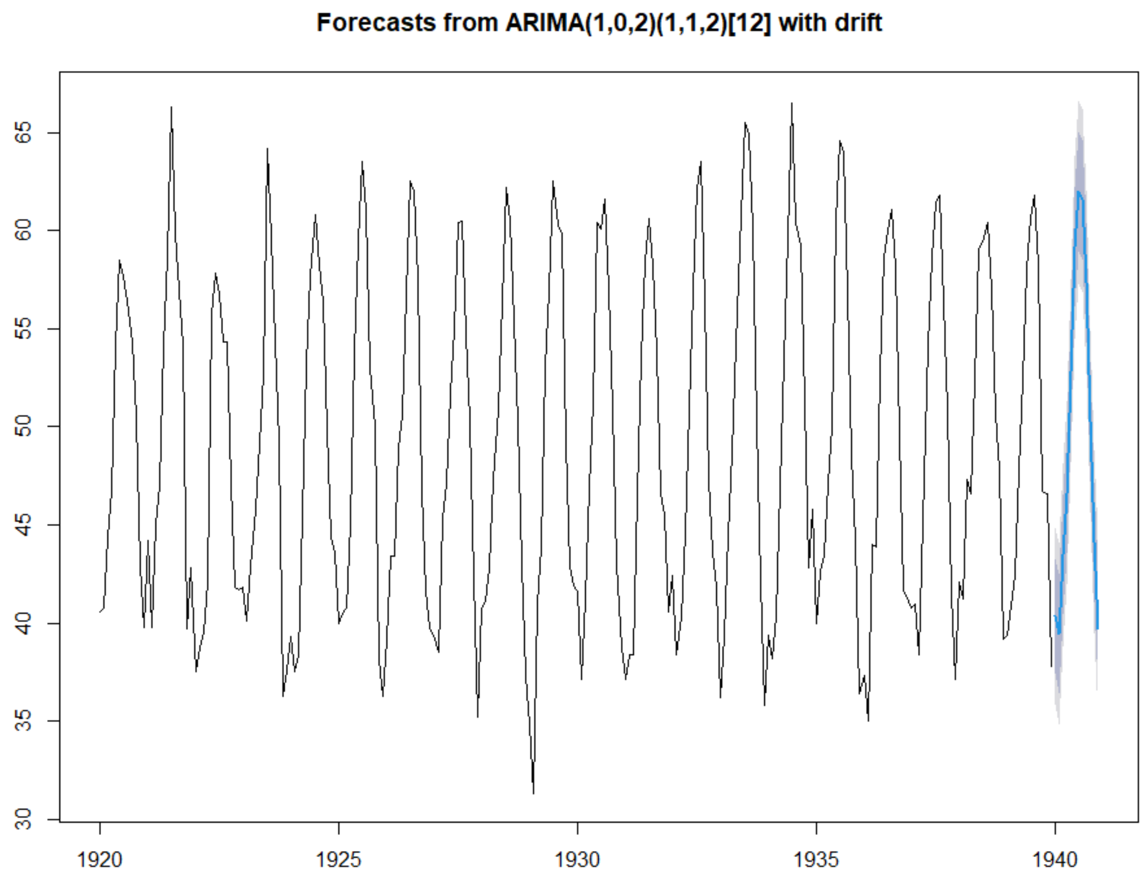
1. forecast

The forecast package is widely used for time series forecasting. It provides functions to fit models like ARIMA, ETS, and others. It helps in automatic model selection and producing accurate predictions.

R Code:



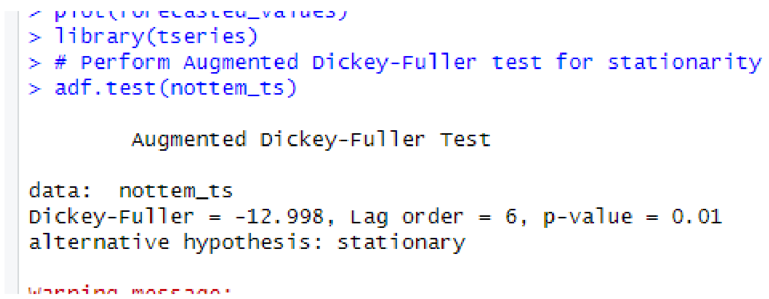
```
> # Load and use forecast package
> library(forecast)
>
> # Load the built-in 'nottem' dataset
> data("nottem")
> nottem_ts <- ts(nottem, start = c(1920, 1), frequency = 12)
>
> # Fit an ARIMA model
> model <- auto.arima(nottem_ts)
>
> # Forecast future values
> forecasted_values <- forecast(model, h = 12)
> plot(forecasted_values)
>
```



2. tseries

The tseries package provides tools for time series analysis, including unit root tests, spectral analysis, and GARCH models.

R Code:

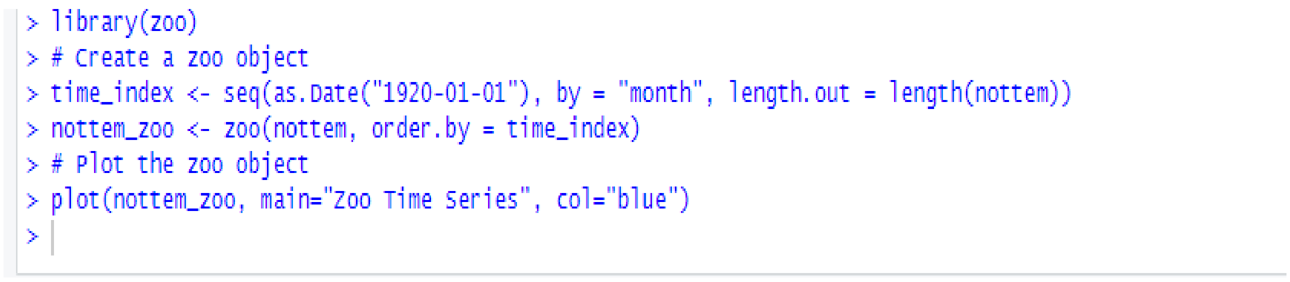


```
> library(tseries)
> # Perform Augmented Dickey-Fuller test for stationarity
> adf.test(nottem_ts)

        Augmented Dickey-Fuller Test

data:  nottem_ts
Dickey-Fuller = -12.998, Lag order = 6, p-value = 0.01
alternative hypothesis: stationary
```

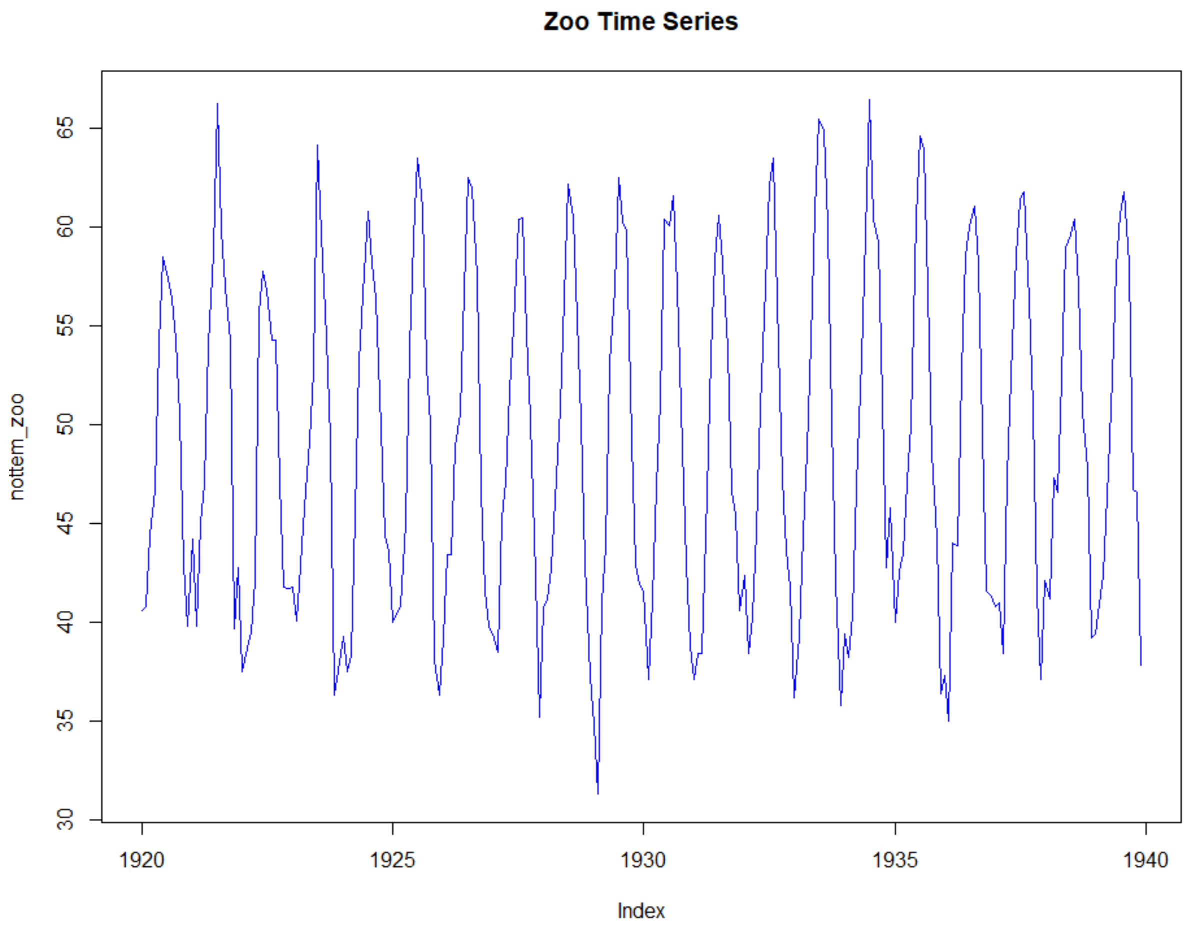
3. zoo

The zoo package provides an efficient way to handle irregular time series data. It allows indexing with dates and smooth handling of missing values.

R Code:



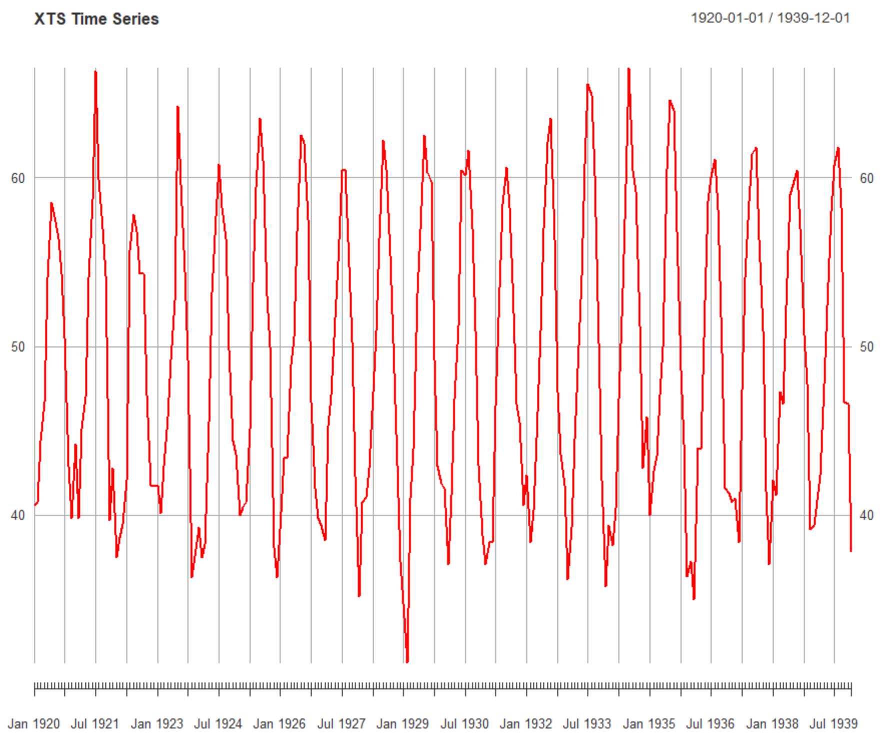
```
> library(zoo)
> # Create a zoo object
> time_index <- seq(as.Date("1920-01-01"), by = "month", length.out = length(nottem))
> nottem_zoo <- zoo(nottem, order.by = time_index)
> # Plot the zoo object
> plot(nottem_zoo, main="Zoo Time Series", col="blue")
> 
```



4. xts

The xts package extends zoo and is commonly used in financial time series analysis.

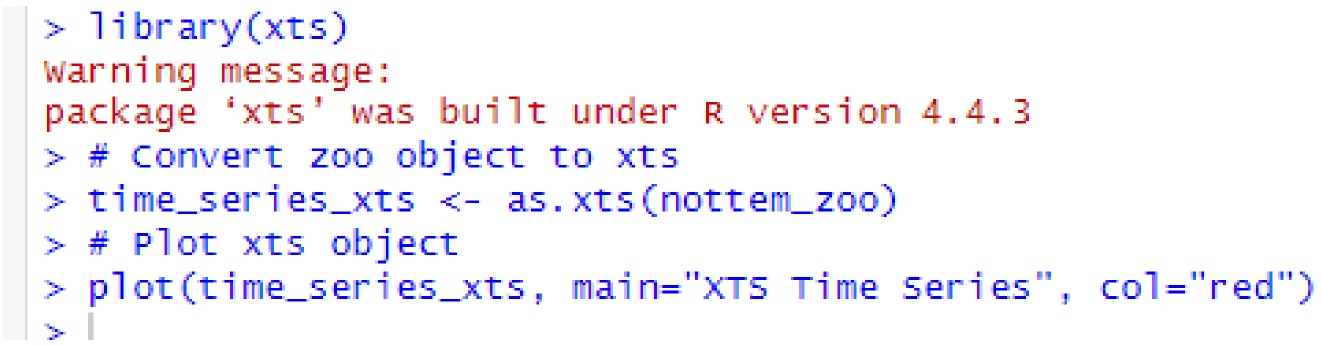
R Code:



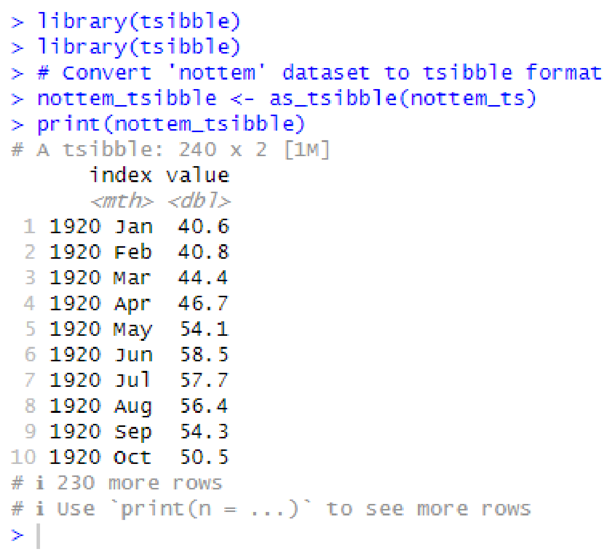
5. tsibble

The tsibble package is designed for tidy time series analysis, making it easy to handle indexed data.

R Code:



```
> library(xts)
Warning message:
package ‘xts’ was built under R version 4.4.3
> # Convert zoo object to xts
> time_series_xts <- as.xts(nottem_zoo)
> # Plot xts object
> plot(time_series_xts, main="XTS Time Series", col="red")
>
```

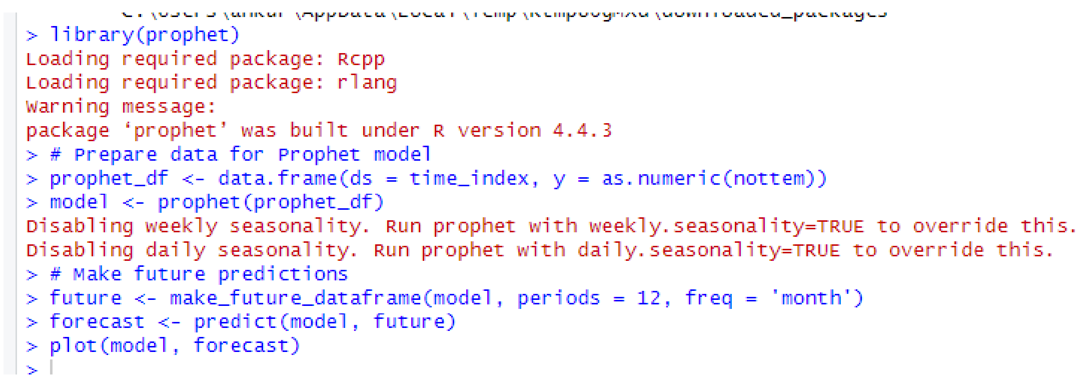


```
> library(tsibble)
> library(tsibble)
> # Convert 'nottem' dataset to tsibble format
> nottem_tsibble <- as_tsibble(nottem_ts)
> print(nottem_tsibble)
# A tsibble: 240 x 2 [1M]
      index value
      <mth> <dbl>
 1 1920 Jan  40.6
 2 1920 Feb  40.8
 3 1920 Mar  44.4
 4 1920 Apr  46.7
 5 1920 May  54.1
 6 1920 Jun  58.5
 7 1920 Jul  57.7
 8 1920 Aug  56.4
 9 1920 Sep  54.3
10 1920 Oct  50.5
# ℹ 230 more rows
# ℹ Use `print(n = ...)` to see more rows
> |
```

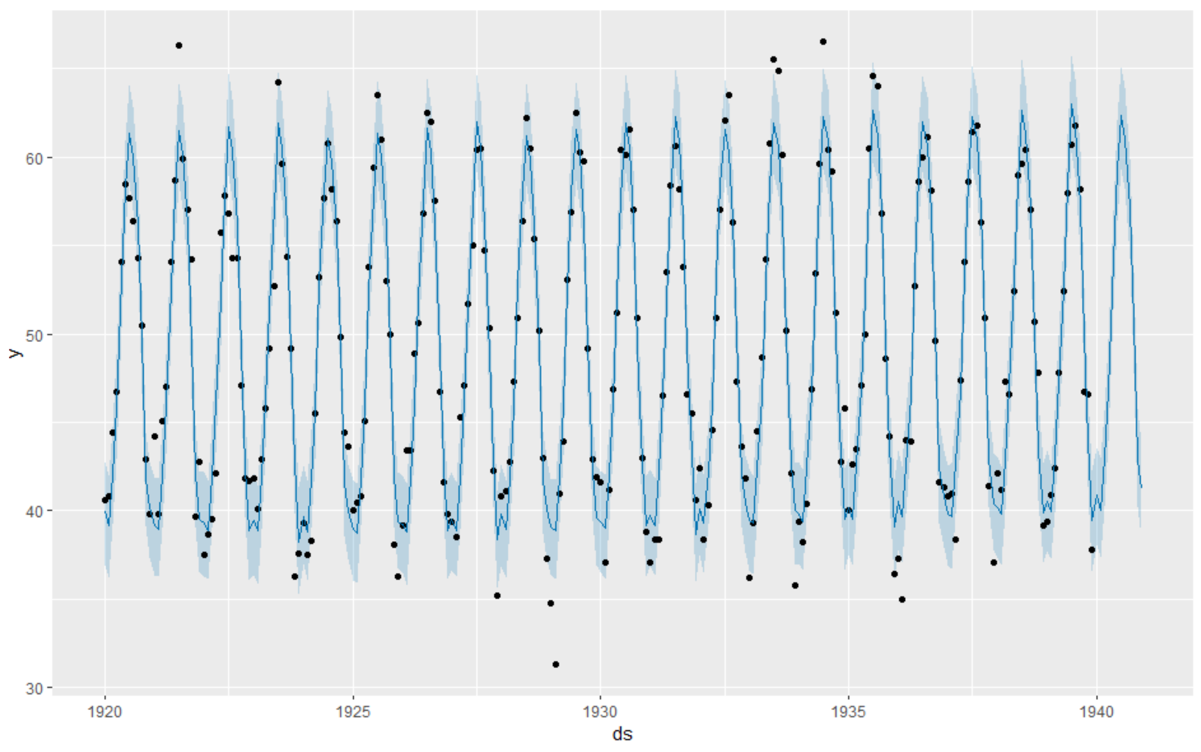
6. prophet

The prophet package is developed by Facebook and is useful for automatic time series forecasting with trend and seasonality components.

R Code:



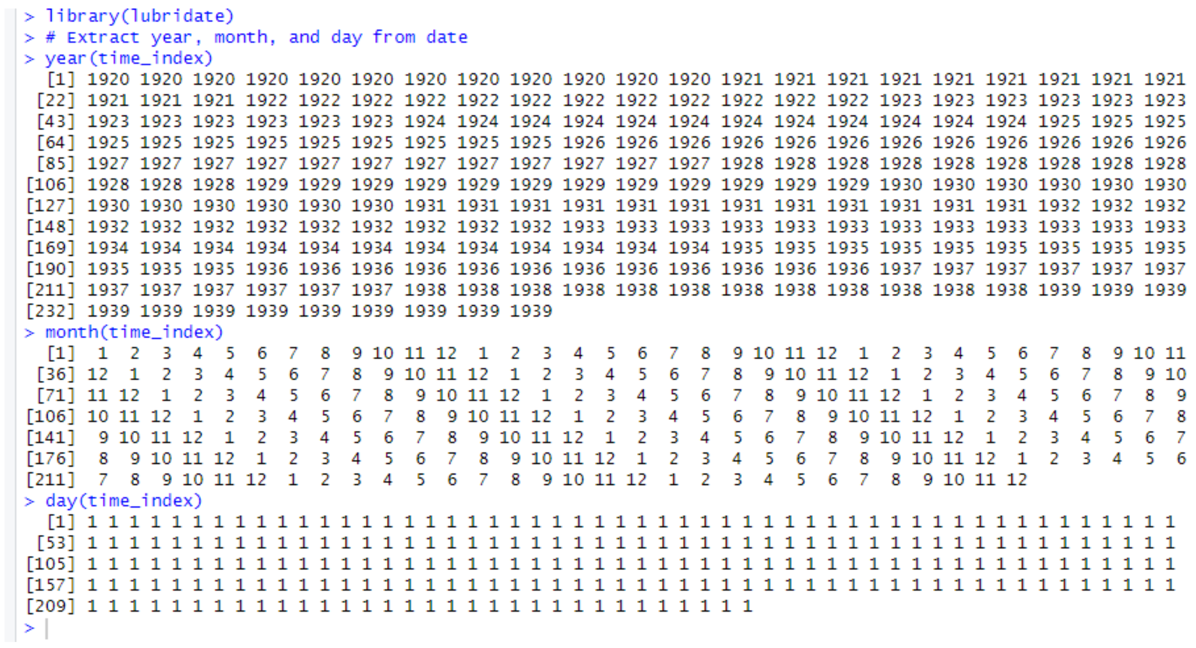
```
> library(prophet)
Loading required package: Rcpp
Loading required package: rlang
Warning message:
package ‘prophet’ was built under R version 4.4.3
> # Prepare data for Prophet model
> prophet_df <- data.frame(ds = time_index, y = as.numeric(nottem))
> model <- prophet(prophet_df)
Disabling weekly seasonality. Run prophet with weekly.seasonality=TRUE to override this.
Disabling daily seasonality. Run prophet with daily.seasonality=TRUE to override this.
> # Make future predictions
> future <- make_future_dataframe(model, periods = 12, freq = 'month')
> forecast <- predict(model, future)
> plot(model, forecast)
> |
```



7. lubridate

The lubridate package simplifies handling and manipulation of date and time variables.

R Code:



```
> library(lubridate)
> # Extract year, month, and day from date
> year(time_index)
  [1] 1920 1920 1920 1920 1920 1920 1920 1920 1920 1920 1920 1920 1921 1921 1921 1921 1921 1921 1921 1921 1921
 [22] 1921 1921 1921 1922 1922 1922 1922 1922 1922 1922 1922 1922 1922 1922 1922 1923 1923 1923 1923 1923 1923
 [43] 1923 1923 1923 1923 1923 1923 1924 1924 1924 1924 1924 1924 1924 1924 1924 1924 1924 1924 1925 1925 1925
 [64] 1925 1925 1925 1925 1925 1925 1925 1925 1925 1926 1926 1926 1926 1926 1926 1926 1926 1926 1926 1926 1926
 [85] 1927 1927 1927 1927 1927 1927 1927 1927 1927 1927 1927 1927 1928 1928 1928 1928 1928 1928 1928 1928 1928
[106] 1928 1928 1928 1929 1929 1929 1929 1929 1929 1929 1929 1929 1929 1929 1929 1930 1930 1930 1930 1930 1930
[127] 1930 1930 1930 1930 1930 1930 1931 1931 1931 1931 1931 1931 1931 1931 1931 1931 1931 1931 1932 1932 1932
[148] 1932 1932 1932 1932 1932 1932 1932 1932 1932 1933 1933 1933 1933 1933 1933 1933 1933 1933 1933 1933 1933
[169] 1934 1934 1934 1934 1934 1934 1934 1934 1934 1934 1934 1934 1935 1935 1935 1935 1935 1935 1935 1935 1935
[190] 1935 1935 1935 1936 1936 1936 1936 1936 1936 1936 1936 1936 1936 1936 1936 1937 1937 1937 1937 1937 1937
[211] 1937 1937 1937 1937 1937 1937 1938 1938 1938 1938 1938 1938 1938 1938 1938 1938 1938 1938 1939 1939 1939
[232] 1939 1939 1939 1939 1939 1939 1939 1939 1939
> month(time_index)
  [1]  1  2  3  4  5  6  7  8  9 10 11 12  1  2  3  4  5  6  7  8  9 10 11 12  1  2  3  4  5  6  7  8  9 10 11
 [36] 12  1  2  3  4  5  6  7  8  9 10 11 12  1  2  3  4  5  6  7  8  9 10 11 12  1  2  3  4  5  6  7  8  9 10
 [71] 11 12  1  2  3  4  5  6  7  8  9 10 11 12  1  2  3  4  5  6  7  8  9 10 11 12  1  2  3  4  5  6  7  8  9
[106] 10 11 12  1  2  3  4  5  6  7  8  9 10 11 12  1  2  3  4  5  6  7  8  9 10 11 12  1  2  3  4  5  6  7  8
[141]  9 10 11 12  1  2  3  4  5  6  7  8  9 10 11 12  1  2  3  4  5  6  7  8  9 10 11 12  1  2  3  4  5  6  7
[176]  8  9 10 11 12  1  2  3  4  5  6  7  8  9 10 11 12  1  2  3  4  5  6  7  8  9 10 11 12  1  2  3  4  5  6
[211]  7  8  9 10 11 12  1  2  3  4  5  6  7  8  9 10 11 12  1  2  3  4  5  6  7  8  9 10 11 12
> day(time_index)
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [53] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[105] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[157] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[209] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
> 
```

2. Functions for Constructing and Plotting Time Series

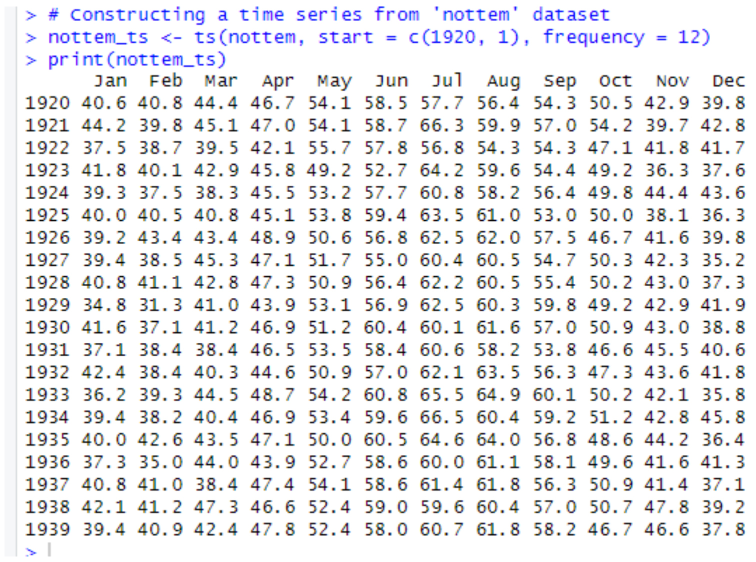
Time series construction and visualization are crucial steps in understanding trends and seasonality. Various R functions can be used for these tasks.

1. Constructing a Time Series

The ts() function is the base R function for creating time series objects. It requires:

- A numeric vector of values.
- The starting time (e.g., year and month for monthly data).
- The frequency (e.g., 12 for monthly data, 4 for quarterly data).

R Code:



```
> # Constructing a time series from 'nottem' dataset
> nottem_ts <- ts(nottem, start = c(1920, 1), frequency = 12)
> print(nottem_ts)
      Jan  Feb  Mar  Apr  May  Jun  Jul  Aug  Sep  Oct  Nov  Dec
1920 40.6 40.8 44.4 46.7 54.1 58.5 57.7 56.4 54.3 50.5 42.9 39.8
1921 44.2 39.8 45.1 47.0 54.1 58.7 66.3 59.9 57.0 54.2 39.7 42.8
1922 37.5 38.7 39.5 42.1 55.7 57.8 56.8 54.3 54.3 47.1 41.8 41.7
1923 41.8 40.1 42.9 45.8 49.2 52.7 64.2 59.6 54.4 49.2 36.3 37.6
1924 39.3 37.5 38.3 45.5 53.2 57.7 60.8 58.2 56.4 49.8 44.4 43.6
1925 40.0 40.5 40.8 45.1 53.8 59.4 63.5 61.0 53.0 50.0 38.1 36.3
1926 39.2 43.4 43.4 48.9 50.6 56.8 62.5 62.0 57.5 46.7 41.6 39.8
1927 39.4 38.5 45.3 47.1 51.7 55.0 60.4 60.5 54.7 50.3 42.3 35.2
1928 40.8 41.1 42.8 47.3 50.9 56.4 62.2 60.5 55.4 50.2 43.0 37.3
1929 34.8 31.3 41.0 43.9 53.1 56.9 62.5 60.3 59.8 49.2 42.9 41.9
1930 41.6 37.1 41.2 46.9 51.2 60.4 60.1 61.6 57.0 50.9 43.0 38.8
1931 37.1 38.4 38.4 46.5 53.5 58.4 60.6 58.2 53.8 46.6 45.5 40.6
1932 42.4 38.4 40.3 44.6 50.9 57.0 62.1 63.5 56.3 47.3 43.6 41.8
1933 36.2 39.3 44.5 48.7 54.2 60.8 65.5 64.9 60.1 50.2 42.1 35.8
1934 39.4 38.2 40.4 46.9 53.4 59.6 66.5 60.4 59.2 51.2 42.8 45.8
1935 40.0 42.6 43.5 47.1 50.0 60.5 64.6 64.0 56.8 48.6 44.2 36.4
1936 37.3 35.0 44.0 43.9 52.7 58.6 60.0 61.1 58.1 49.6 41.6 41.3
1937 40.8 41.0 38.4 47.4 54.1 58.6 61.4 61.8 56.3 50.9 41.4 37.1
1938 42.1 41.2 47.3 46.6 52.4 59.0 59.6 60.4 57.0 50.7 47.8 39.2
1939 39.4 40.9 42.4 47.8 52.4 58.0 60.7 61.8 58.2 46.7 46.6 37.8
> 
```

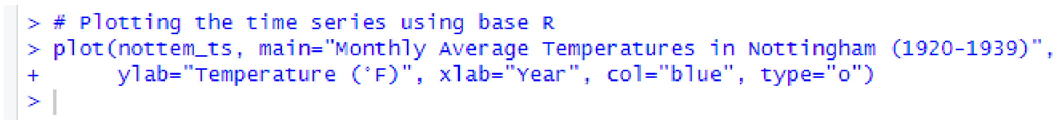
2. Plotting a Time Series

Visualizing time series data helps in identifying trends and seasonal patterns.

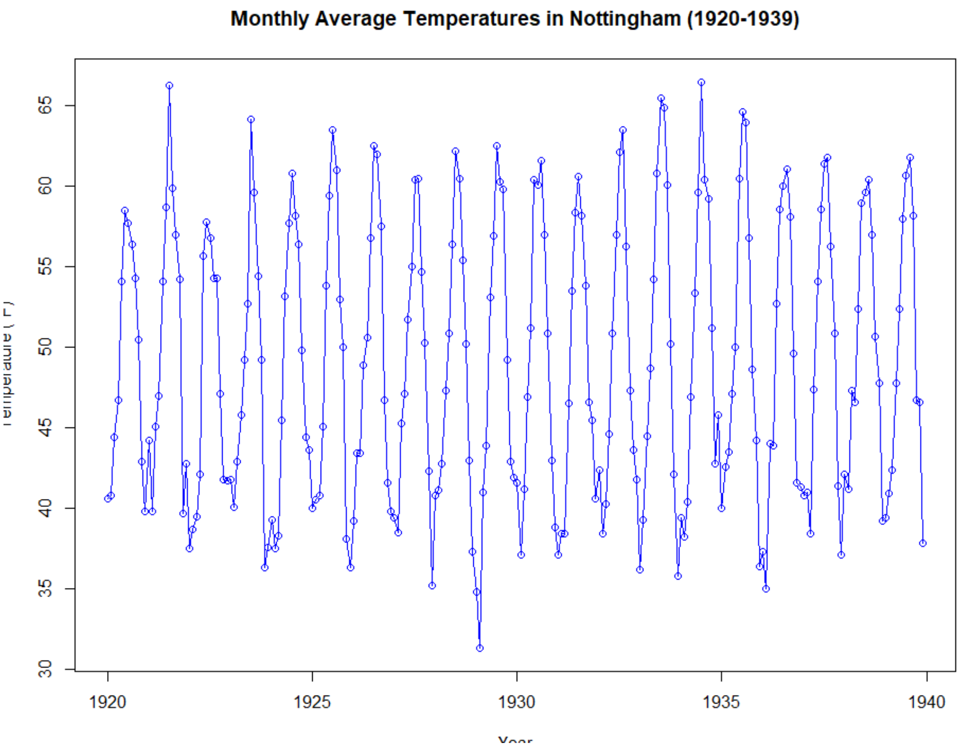
a) Using Base R plot() Function

The simplest way to visualize a time series is using the base R plot() function.

R Code:



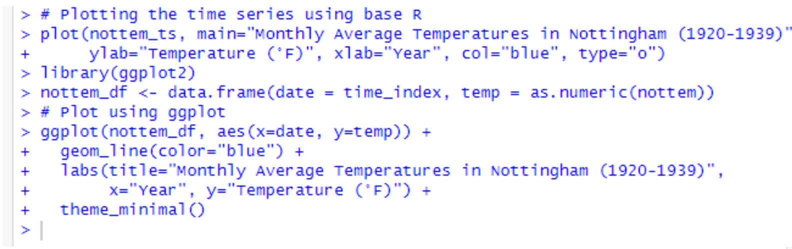
```
> # Plotting the time series using base R
> plot(nottem_ts, main="Monthly Average Temperatures in Nottingham (1920-1939)",
+      ylab="Temperature (°F)", xlab="Year", col="blue", type="o")
> 
```



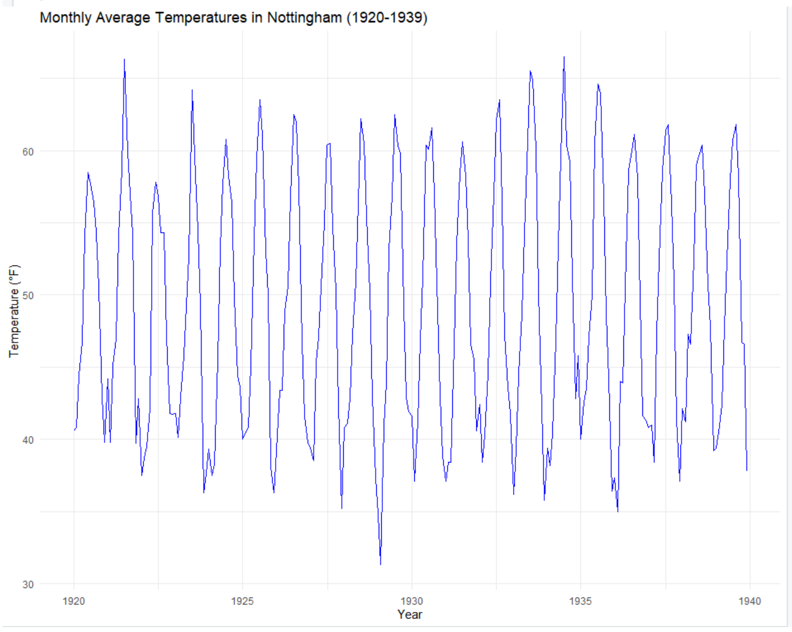
b) Using ggplot2 for Enhanced Visualization

The ggplot2 package allows customized time series plotting.

R Code:



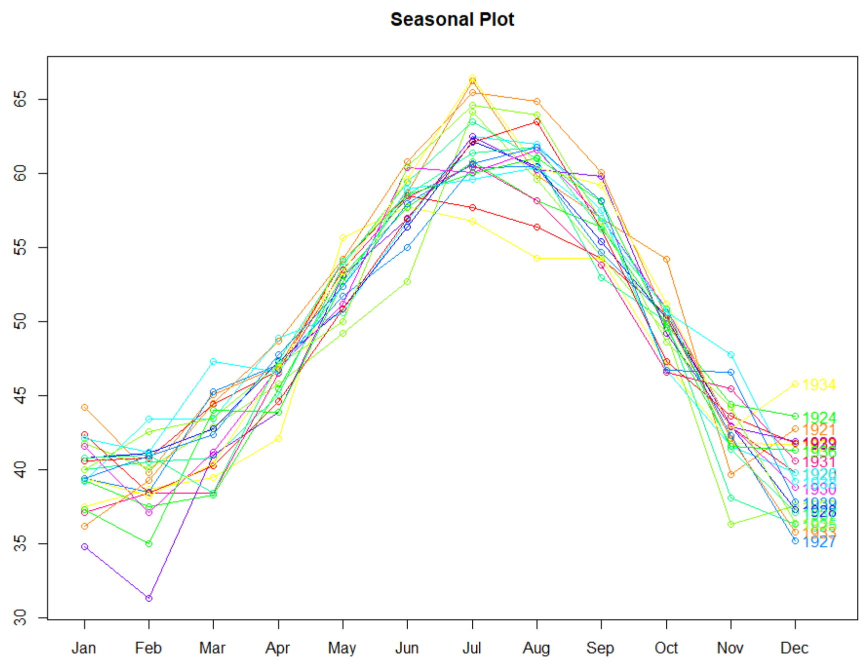
```
> # Plotting the time series using base R
> plot(nottem_ts, main="Monthly Average Temperatures in Nottingham (1920-1939)",
+      ylab="Temperature (°F)", xlab="Year", col="blue", type="o")
> library(ggplot2)
> nottem_df <- data.frame(date = time_index, temp = as.numeric(nottem))
> # Plot using ggplot
> ggplot(nottem_df, aes(x=date, y=temp)) +
+   geom_line(color="blue") +
+   labs(title="Monthly Average Temperatures in Nottingham (1920-1939)",
+        x="Year", y="Temperature (°F)") +
+   theme_minimal()
> 
```

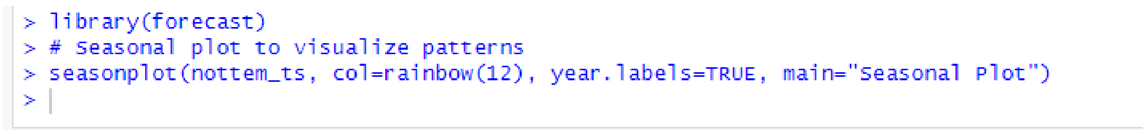


c) Using forecast Package for Seasonality Plot

The forecast package provides additional tools for decomposition and seasonality visualization.

R Code:





```
> library(forecast)
> # Seasonal plot to visualize patterns
> seasonplot(nottem_ts, col=rainbow(12), year.labels=TRUE, main="Seasonal Plot")
> 
```

This allows comparison of different months across years, helping to identify repeating seasonal patterns.

3)Functions for Decomposing Time Series

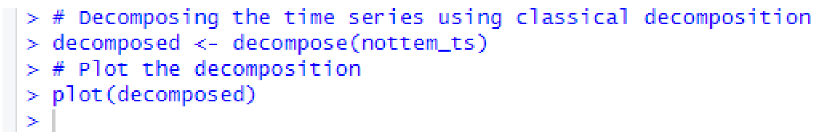
Time series decomposition involves breaking a time series into its components:

- **Trend**: The long-term progression of the series.
- **Seasonal**: Regular, repeating patterns over time.
- **Residual (Random Component)**: Irregular fluctuations that are not explained by trend or seasonality.

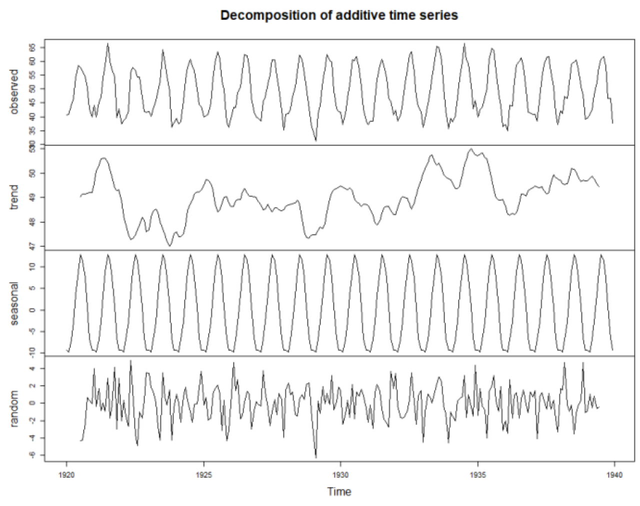
1. Decomposing Using decompose()

The base R function decompose() breaks down a time series into these components.

R Code:



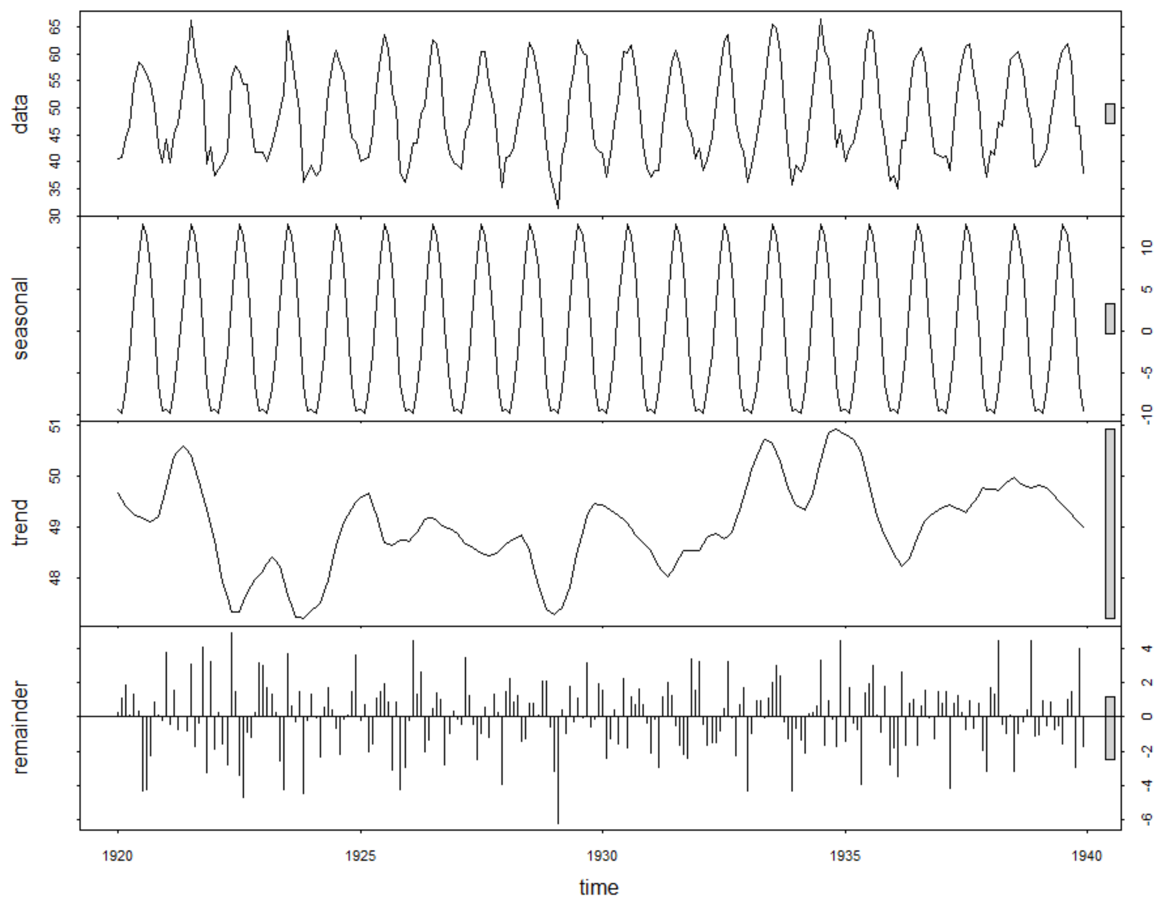
```
> # Decomposing the time series using classical decomposition
> decomposed <- decompose(nottem_ts)
> # Plot the decomposition
> plot(decomposed)
> 
```



2. Decomposing Using stl() for Seasonal and Trend Components

The stl() function provides a more flexible decomposition using LOESS (Local Regression).

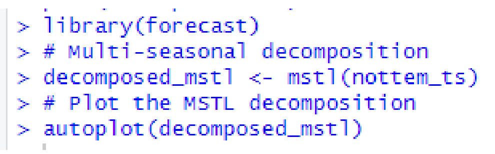
R Code:



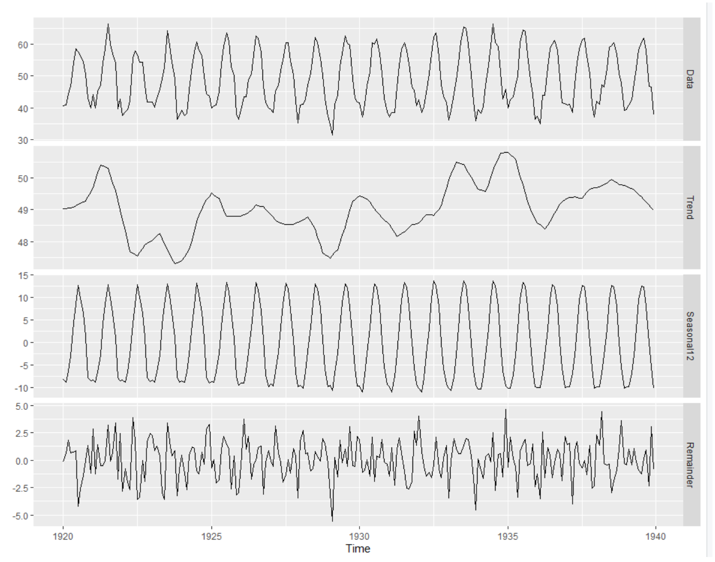
3. Decomposing Using forecast Package

The forecast package provides the mstl() function, which enhances STL decomposition for multiple seasonal patterns.

R Code:



```
> library(forecast)
> # Multi-seasonal decomposition
> decomposed_mstl <- mstl(nottem_ts)
> # Plot the MSTL decomposition
> autoplot(decomposed_mstl)
```

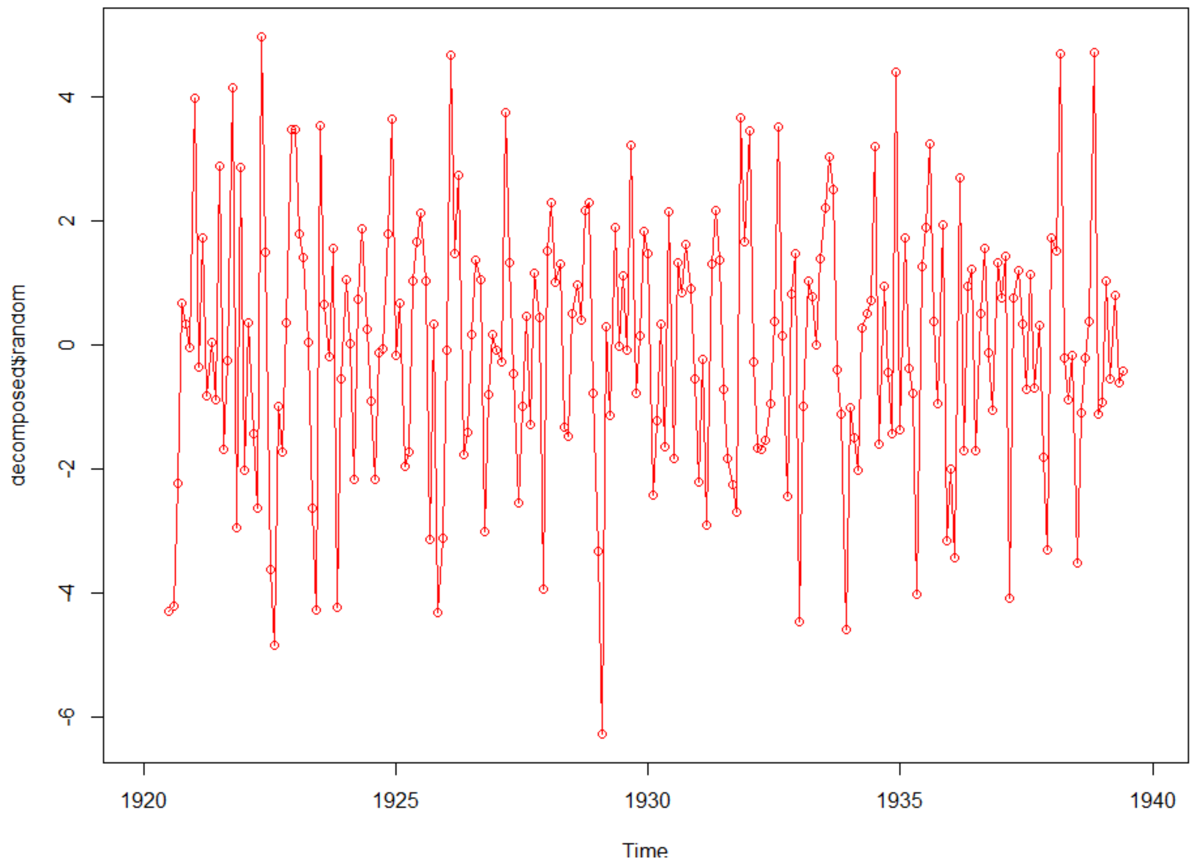


4. Checking Residuals After Decomposition

After decomposition, it's important to check if the residuals are random (white noise), ensuring that the trend and seasonal components were correctly extracted.

R Code:

Residuals After Decomposition



```
> # Plot residuals
> plot(decomposed$random, main="Residuals After Decomposition", col="red", type="o")
> # Perform Ljung-Box test for randomness
> Box.test(decomposed$random, lag=10, type="Ljung-Box")

        Box-Ljung test

data:  decomposed$random
X-squared = 47.667, df = 10, p-value = 7.138e-07
```

4)Base Functions for Forecasting Time Series Models

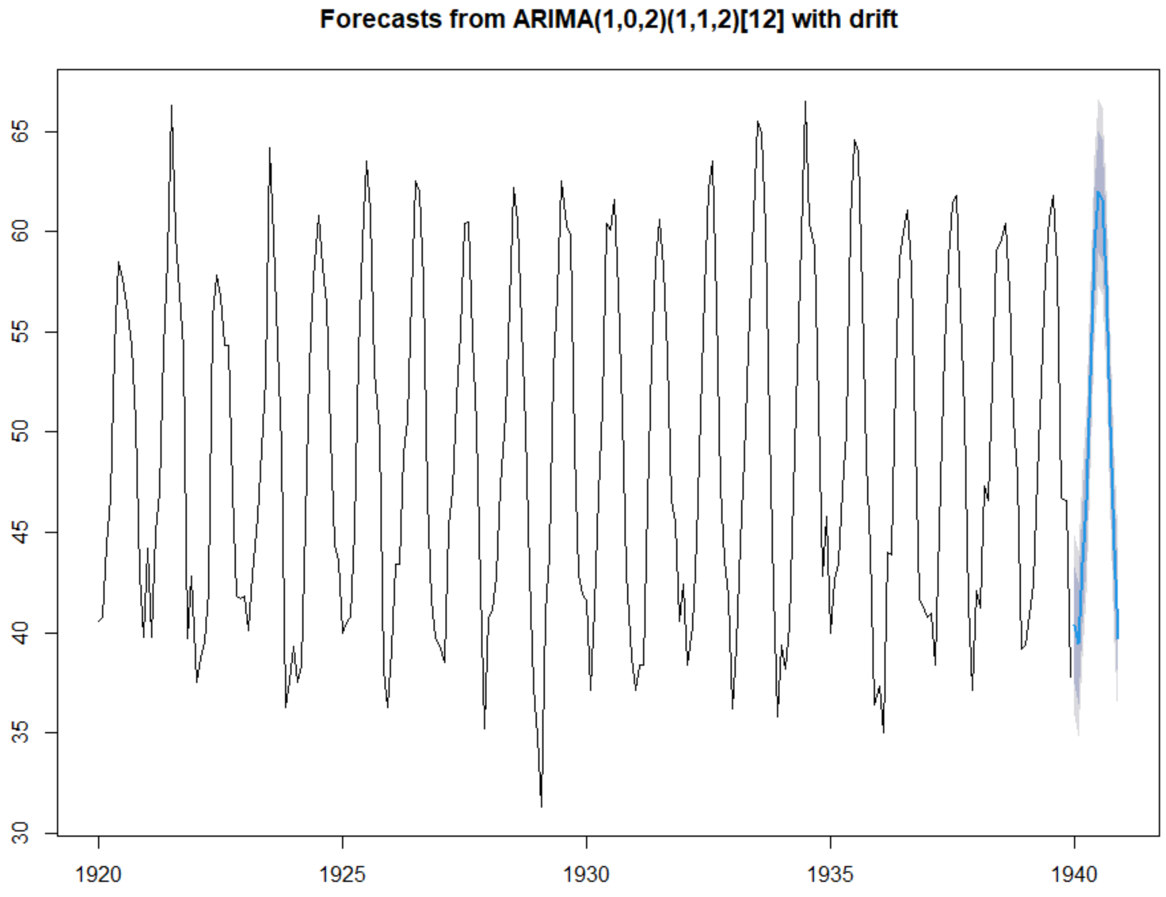
Time series forecasting involves predicting future values based on historical data. The three most common forecasting methods in R are:

1. ARIMA (AutoRegressive Integrated Moving Average)

ARIMA models are widely used for time series forecasting. They combine autoregression (AR), differencing (I), and moving averages (MA).

R Code:

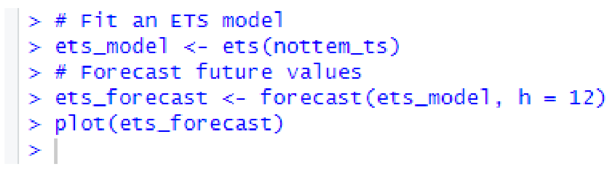
```
> # Fit an ARIMA model
> library(forecast)
> arima_model <- auto.arima(nottem_ts)
>
> # Forecast future values
> arima_forecast <- forecast(arima_model, h = 12)
> plot(arima_forecast)
```



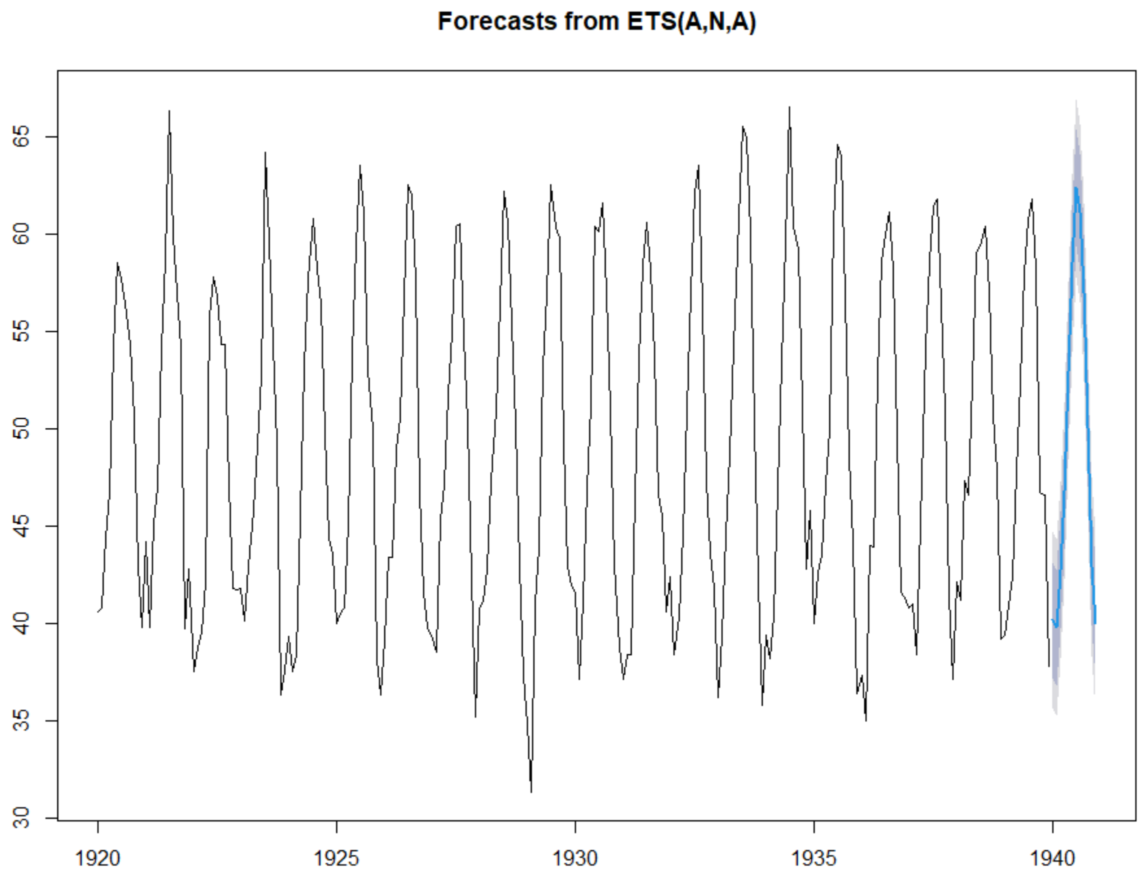
2. Exponential Smoothing (ETS Model)

The ETS model (Error, Trend, and Seasonality) is an alternative to ARIMA that automatically selects the best-fitting exponential smoothing model.

R Code:



```
> # Fit an ETS model
> ets_model <- ets(nottem_ts)
> # Forecast future values
> ets_forecast <- forecast(ets_model, h = 12)
> plot(ets_forecast)
> 
```

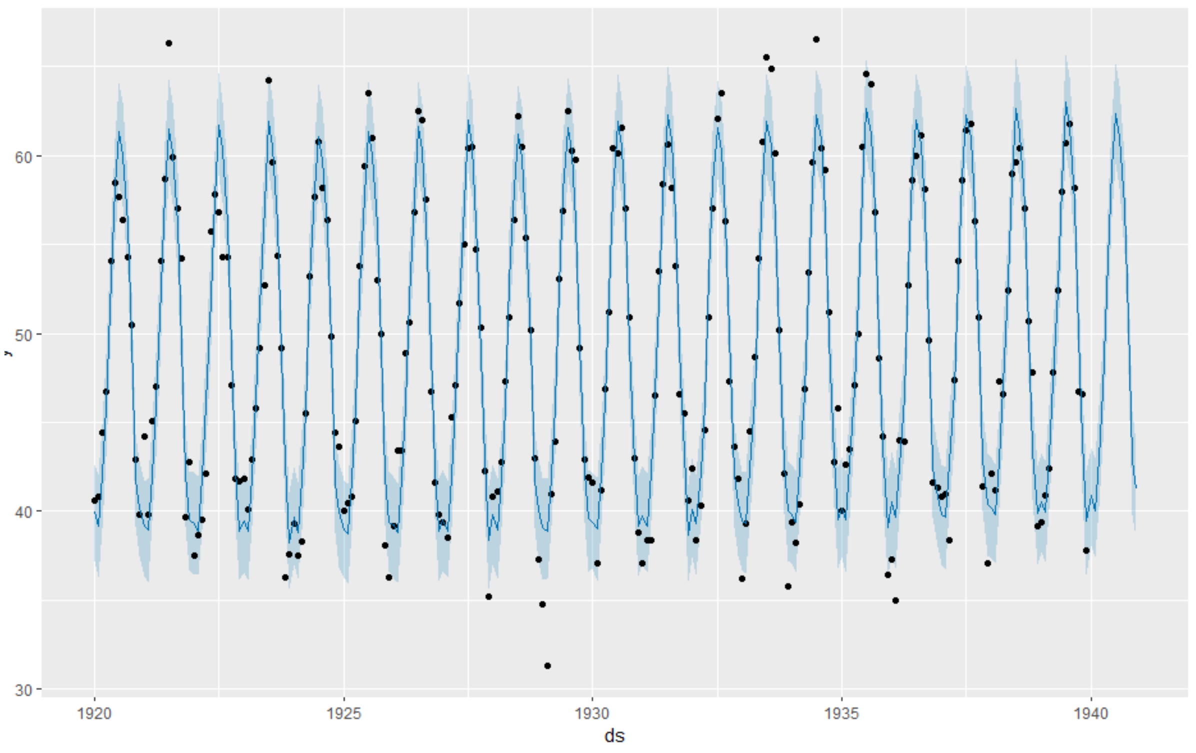


3. Prophet Model

Developed by Facebook, the Prophet model handles trend shifts, seasonality, and holidays.

R Code:

```
> library(prophet)
> # Prepare data for Prophet
> time_index <- seq(as.Date("1920-01-01"), by = "month", length.out = length(nottem))
> prophet_df <- data.frame(ds = time_index, y = as.numeric(nottem))
> prophet_model <- prophet(prophet_df)
Disabling weekly seasonality. Run prophet with weekly.seasonality=TRUE to override this.
Disabling daily seasonality. Run prophet with daily.seasonality=TRUE to override this.
> # Make future predictions
> future <- make_future_dataframe(prophet_model, periods = 12, freq = 'month')
> forecast <- predict(prophet_model, future)
> plot(prophet_model, forecast)
```



5). Identifying Correlation and Variance in Time Series

Understanding correlation and variance in time series data helps analyze relationships and fluctuations in data.

1. Variance Calculation

Variance measures the dispersion of the data points.

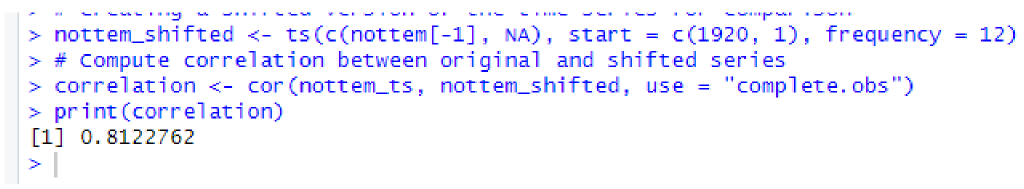
R Code:

```
> # Compute variance of temperature data
> var_temp <- var(nottem_ts)
> print(var_temp)
[1] 73.48474
>
```

2. Correlation Between Two Time Series

Correlation quantifies the relationship between two time series.

R Code:



```
> # Creating a shifted version of the time series for comparison
> nottem_shifted <- ts(c(nottem[-1], NA), start = c(1920, 1), frequency = 12)
> # Compute correlation between original and shifted series
> correlation <- cor(nottem_ts, nottem_shifted, use = "complete.obs")
> print(correlation)
[1] 0.8122762
> |
```

A correlation **close to 1 or -1** indicates a strong relationship, while a value near **0** suggests weak or no correlation.

Domain 2: Social Network Analysis and Mining

1. Popular Packages for Social Network Analysis

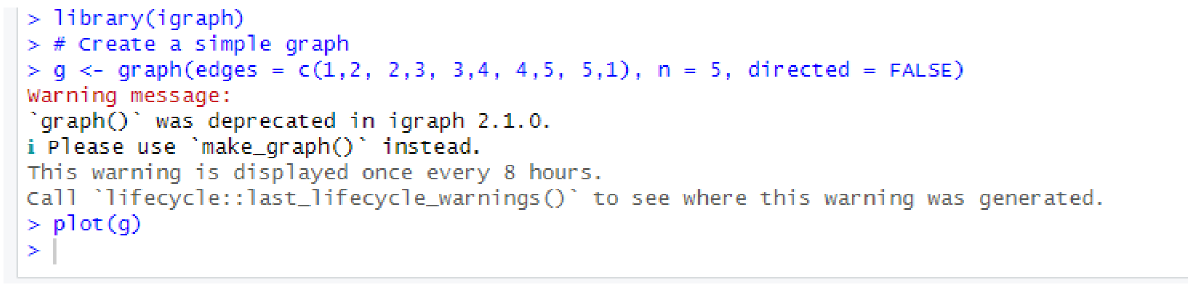
Social Network Analysis (SNA) is a methodological approach used to study relationships between entities such as individuals, organizations, or systems. It helps in understanding connectivity, influence, community formation, and information flow. In R, multiple packages provide functionalities for network creation, visualization, and statistical analysis.

Below are seven of the most commonly used R packages for SNA:

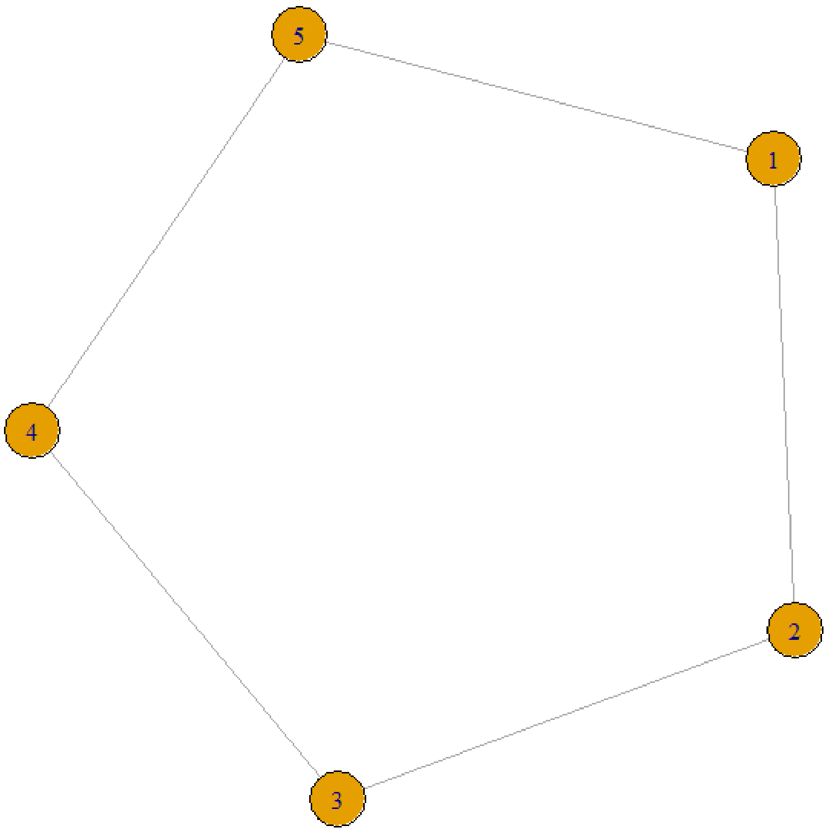
1. igraph

The igraph package is one of the most powerful and widely used packages for network analysis. It allows users to create, manipulate, visualize, and analyze complex networks efficiently. It includes functions for network clustering, shortest paths, and centrality measures.

R Code:



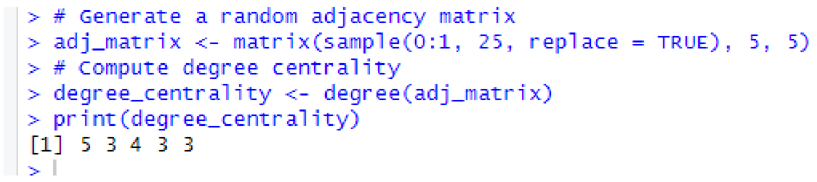
```
> library(igraph)
> # Create a simple graph
> g <- graph(edges = c(1,2, 2,3, 3,4, 4,5, 5,1), n = 5, directed = FALSE)
Warning message:
`graph()` was deprecated in igraph 2.1.0.
ℹ Please use `make_graph()` instead.
This warning is displayed once every 8 hours.
Call `lifecycle::last_lifecycle_warnings()` to see where this warning was generated.
> plot(g)
> 
```



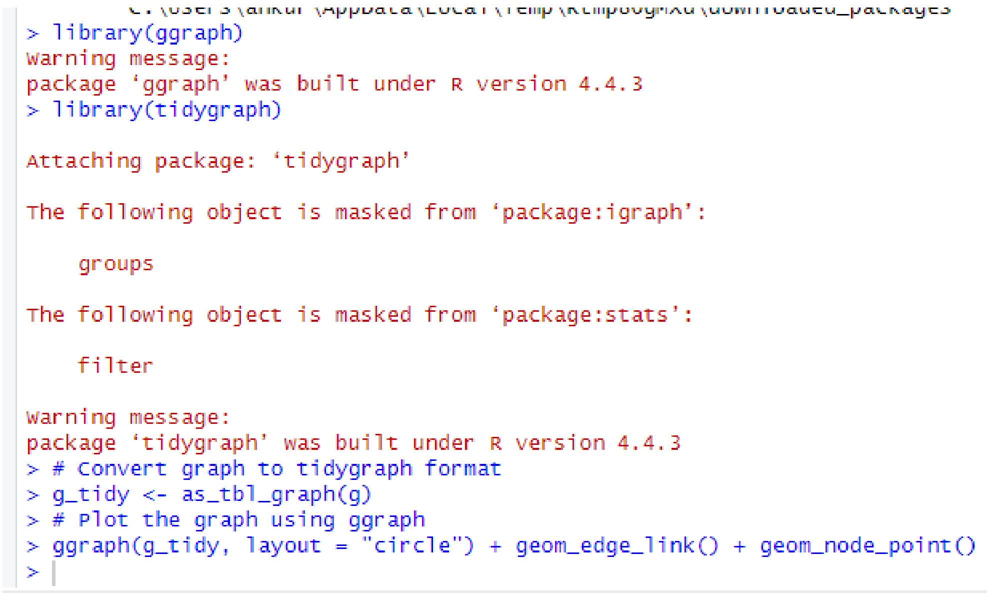
2. ggraph

The ggraph package is an extension of ggplot2 designed for network visualization. It provides various layout algorithms, edge bundling techniques, and customizable aesthetics to produce high-quality network plots.

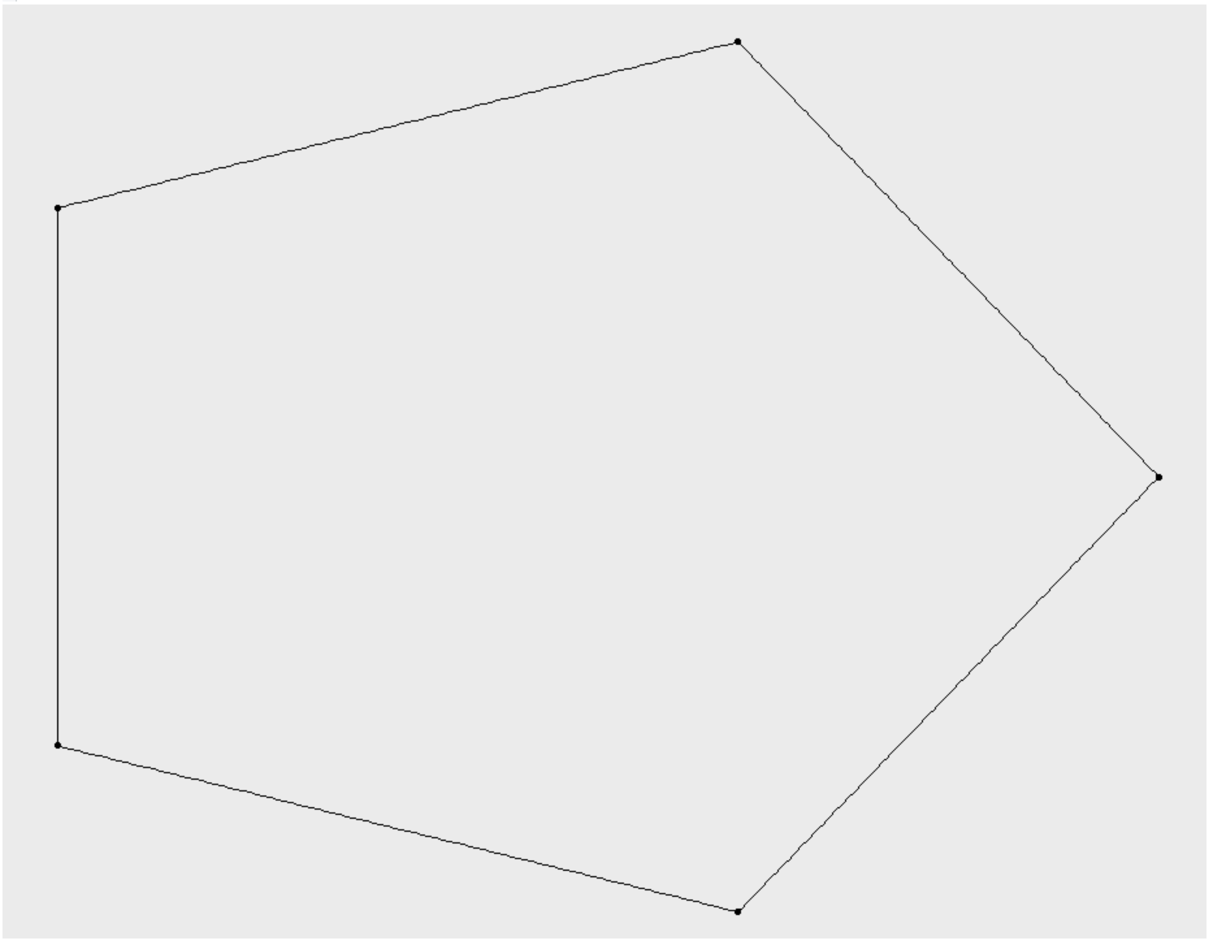
R Code:



```
> # Generate a random adjacency matrix
> adj_matrix <- matrix(sample(0:1, 25, replace = TRUE), 5, 5)
> # Compute degree centrality
> degree_centrality <- degree(adj_matrix)
> print(degree_centrality)
[1] 5 3 4 3 3
> 
```



```
        C:\Users\ankur\AppData\Local\Temp\RtmpoUgMxd\downloaded_packages
> library(ggraph)
Warning message:
package ‘ggraph’ was built under R version 4.4.3
> library(tidygraph)

Attaching package: ‘tidygraph’

The following object is masked from ‘package:igraph’:

    groups

The following object is masked from ‘package:stats’:

    filter

Warning message:
package ‘tidygraph’ was built under R version 4.4.3
> # Convert graph to tidygraph format
> g_tidy <- as_tbl_graph(g)
> # Plot the graph using ggraph
> ggraph(g_tidy, layout = "circle") + geom_edge_link() + geom_node_point()
> 
```



3. sna

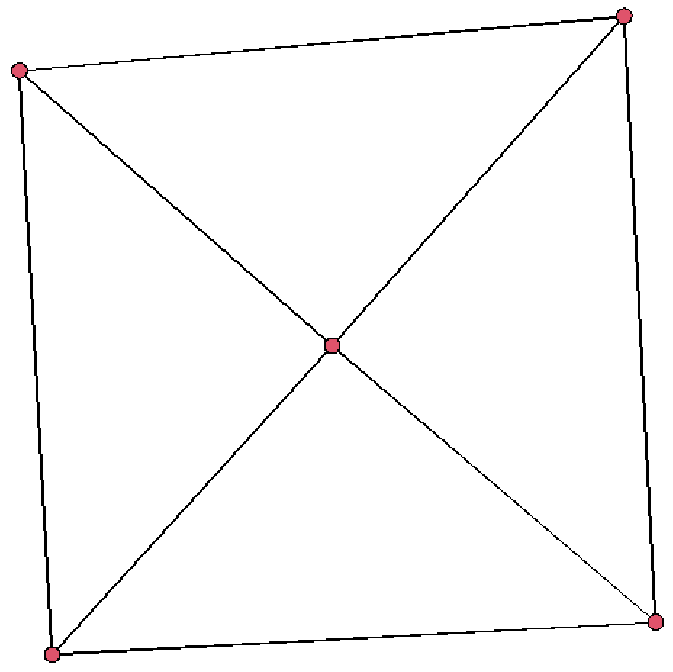
The sna package (Social Network Analysis) includes a variety of functions for analyzing network structures, including centrality measures, clustering algorithms, and structural equivalence tests. It is particularly useful for understanding the importance of nodes and edges within a network.

R Code:

4. network

The network package allows for the creation, storage, and visualization of network objects in R. Unlike igraph, which represents networks as graphs, network uses adjacency matrices and edge lists.

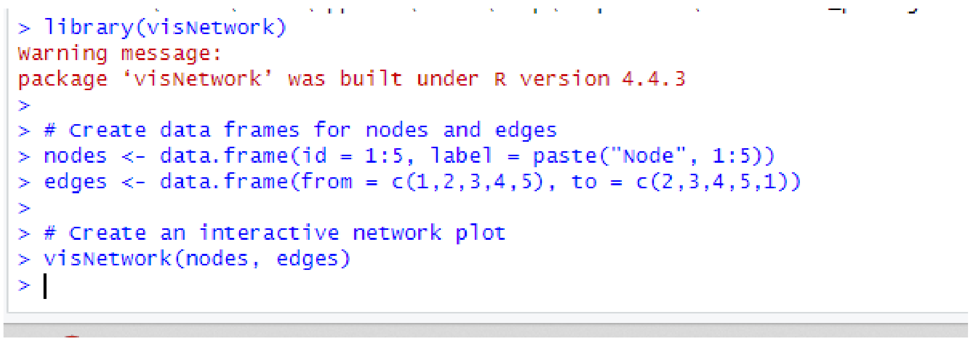
R Code:



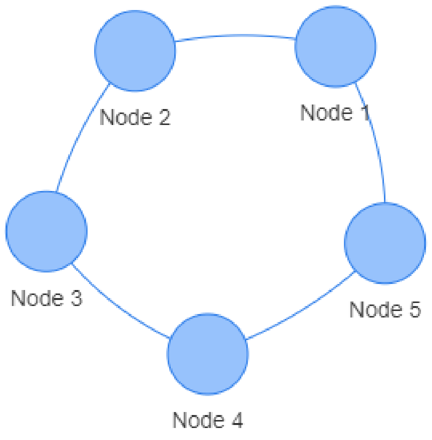
5. visNetwork

The visNetwork package is used for interactive network visualization. It provides dynamic plots where users can zoom, drag, and interact with nodes and edges. It is useful for web-based applications.

R Code:



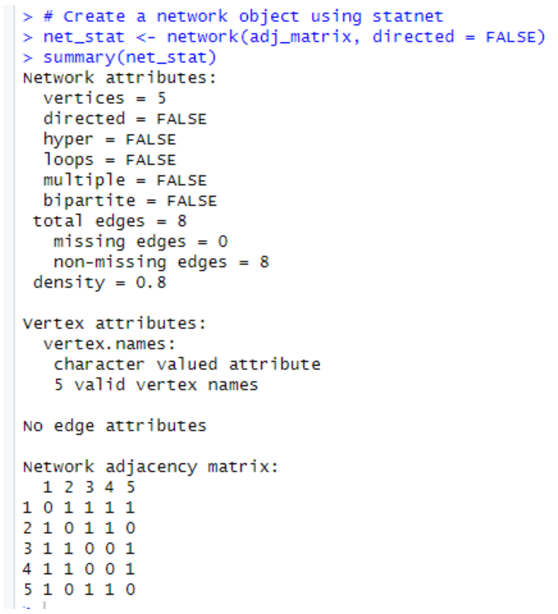
```
> library(visNetwork)
Warning message:
package ‘visNetwork’ was built under R version 4.4.3
>
> # Create data frames for nodes and edges
> nodes <- data.frame(id = 1:5, label = paste("Node", 1:5))
> edges <- data.frame(from = c(1,2,3,4,5), to = c(2,3,4,5,1))
>
> # Create an interactive network plot
> visNetwork(nodes, edges)
> |
```



6. statnet

The statnet package provides a suite of functions for statistical modeling of networks, particularly useful in social sciences. It includes methods for network regression, exponential random graph models (ERGMs), and dynamic network analysis.

R Code:

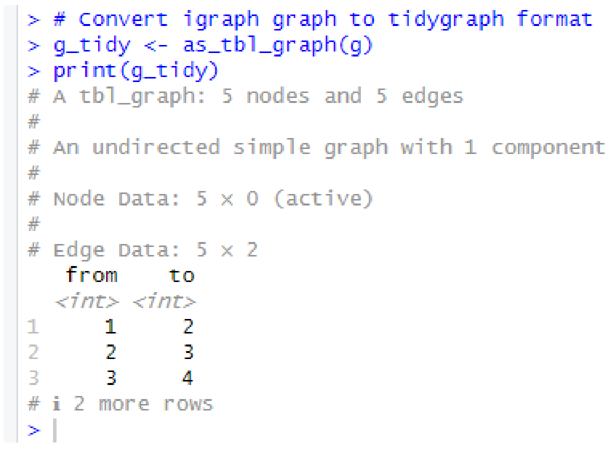


```
> # Create a network object using statnet
> net_stat <- network(adj_matrix, directed = FALSE)
> summary(net_stat)
Network attributes:
  vertices = 5
  directed = FALSE
  hyper = FALSE
  loops = FALSE
  multiple = FALSE
  bipartite = FALSE
 total edges = 8
   missing edges = 0
   non-missing edges = 8
 density = 0.8

Vertex attributes:
  vertex.names:
   character valued attribute
   5 valid vertex names

No edge attributes

Network adjacency matrix:
  1 2 3 4 5
1 0 1 1 1 1
2 1 0 1 1 0
3 1 1 0 0 1
4 1 1 0 0 1
5 1 0 1 1 0
```

7. tidygraph

The tidygraph package integrates network analysis with the tidyverse framework, making it easier to manipulate graph data using dplyr and tidyr. It is particularly useful for users familiar with ggplot2 and dplyr.

R Code:



```
> # Convert igraph graph to tidygraph format
> g_tidy <- as_tbl_graph(g)
> print(g_tidy)
# A tbl_graph: 5 nodes and 5 edges
#
# An undirected simple graph with 1 component
#
# Node Data: 5 × 0 (active)
#
# Edge Data: 5 × 2
   from    to
  <int> <int>
1     1     2
2     2     3
3     3     4
# ℹ 2 more rows
> 
```

2)Functions for Network Graph Construction and Plotting

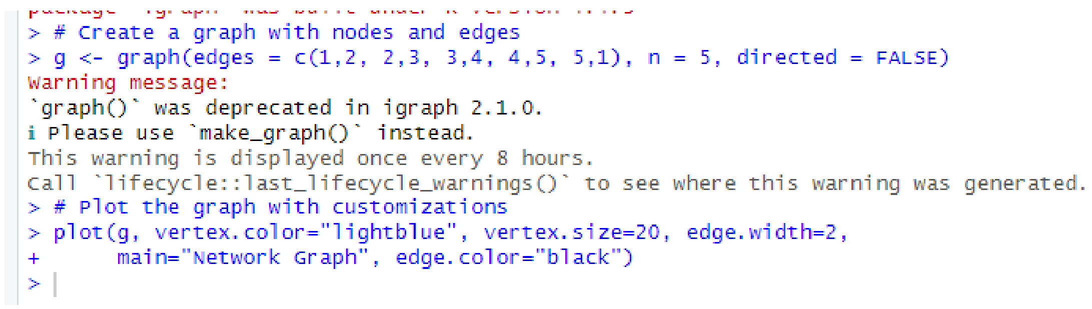
Network graphs are used to represent relationships between nodes (entities) and edges (connections). Constructing and plotting a network graph involves defining nodes and edges, setting the graph layout, and customizing aesthetics.

Detailed Explanation:

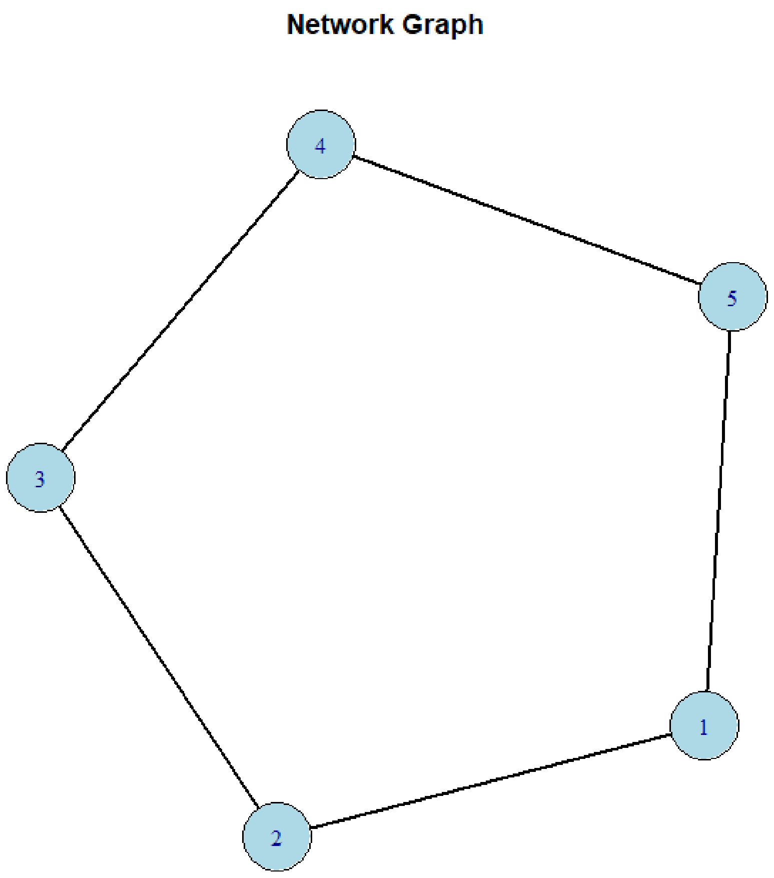
A network graph consists of:

- **Nodes (Vertices)**: Represent entities such as individuals, computers, or cities.
- **Edges (Links)**: Represent connections between nodes.
- **Directed vs. Undirected Graphs**: In a directed graph, edges have direction, while in an undirected graph, edges have no direction.
- **Weighted Graphs**: Each edge has a weight representing the strength of the relationship.

R Code:



```
> # Create a graph with nodes and edges
> g <- graph(edges = c(1,2, 2,3, 3,4, 4,5, 5,1), n = 5, directed = FALSE)
Warning message:
`graph()` was deprecated in igraph 2.1.0.
i Please use `make_graph()` instead.
This warning is displayed once every 8 hours.
Call `lifecycle::last_lifecycle_warnings()` to see where this warning was generated.
> # Plot the graph with customizations
> plot(g, vertex.color="lightblue", vertex.size=20, edge.width=2,
+      main="Network Graph", edge.color="black")
> 
```



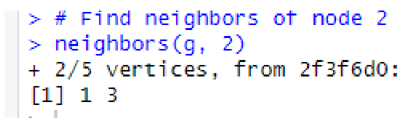
3. Neighbor Nodes

Neighbor nodes are directly connected to a given node. Identifying neighbors is useful for analyzing influence and connectivity in networks.

Detailed Explanation:

- **Degree of a Node**: Number of edges connected to a node.
- **In-Degree and Out-Degree (Directed Graphs)**: In-degree represents incoming connections, while out-degree represents outgoing connections.
- **Finding Neighbors**: Used to identify direct connections in social networks or communication networks.

R Code:



```
> # Find neighbors of node 2
> neighbors(g, 2)
+ 2/5 vertices, from 2f3f6d0:
[1] 1 3
```

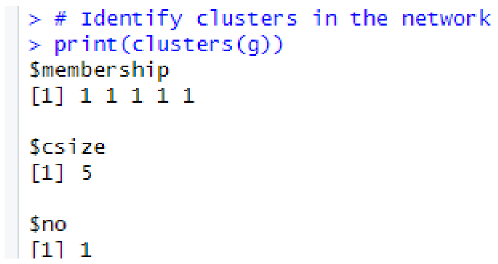
4. Clusters

Clusters refer to groups of tightly connected nodes. Clustering helps in detecting communities within a network.

Detailed Explanation:

- **Connected Components**: Identify subgraphs where all nodes are connected.
- **Cluster Coefficient**: Measures how well neighbors of a node are connected to each other.
- **Applications**: Useful in social networks (friend groups) and biological networks (gene interactions).

R Code:



```
> # Identify clusters in the network
> print(clusters(g))
$membership
[1] 1 1 1 1 1

$csize
[1] 5

$no
[1] 1
```

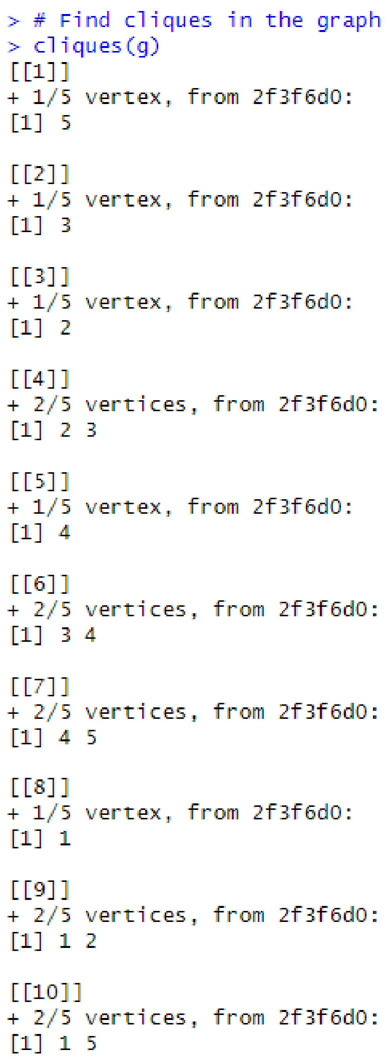
5. Cliques

A clique is a subset of nodes where each node is directly connected to every other node in the subset. Cliques are useful in studying tightly-knit groups.

Detailed Explanation:

- **Maximum Clique**: Largest clique in a network.
- **Applications**: Used in social media analysis, fraud detection, and research collaborations.

R Code:



```
> # Find cliques in the graph
> cliques(g)
[[1]]
+ 1/5 vertex, from 2f3f6d0:
[1] 5

[[2]]
+ 1/5 vertex, from 2f3f6d0:
[1] 3

[[3]]
+ 1/5 vertex, from 2f3f6d0:
[1] 2

[[4]]
+ 2/5 vertices, from 2f3f6d0:
[1] 2 3

[[5]]
+ 1/5 vertex, from 2f3f6d0:
[1] 4

[[6]]
+ 2/5 vertices, from 2f3f6d0:
[1] 3 4

[[7]]
+ 2/5 vertices, from 2f3f6d0:
[1] 4 5

[[8]]
+ 1/5 vertex, from 2f3f6d0:
[1] 1

[[9]]
+ 2/5 vertices, from 2f3f6d0:
[1] 1 2

[[10]]
+ 2/5 vertices, from 2f3f6d0:
[1] 1 5
```

6. Community Detection

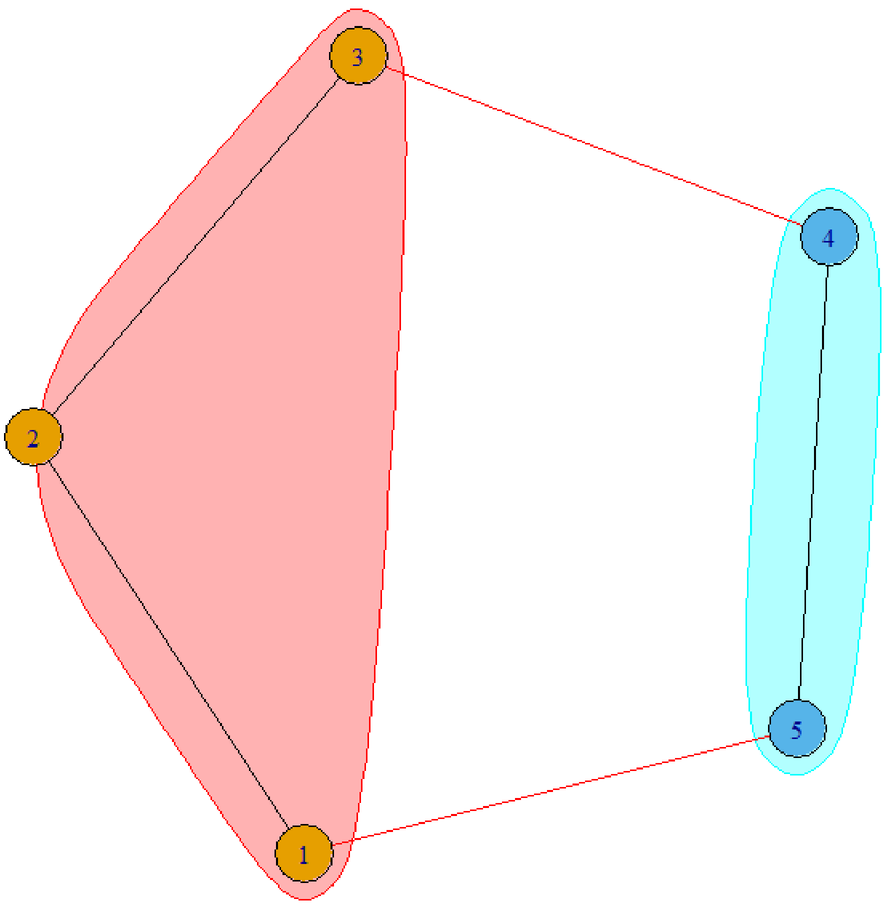
Community detection algorithms identify densely connected subgroups within a network. This is useful for studying relationships in large graphs.

Detailed Explanation:

- **Modularity**: Measures how well a network is divided into communities.
- **Algorithms**: Fast Greedy, Louvain, and Walktrap methods.
- **Applications**: Detecting political groups, online communities, and recommendation systems.

R Code:

```
> # Detect communities using fast greedy method
> community <- cluster_fast_greedy(g)
> plot(community, g)
> 
```



7. Blocks (Partitioning of Networks)

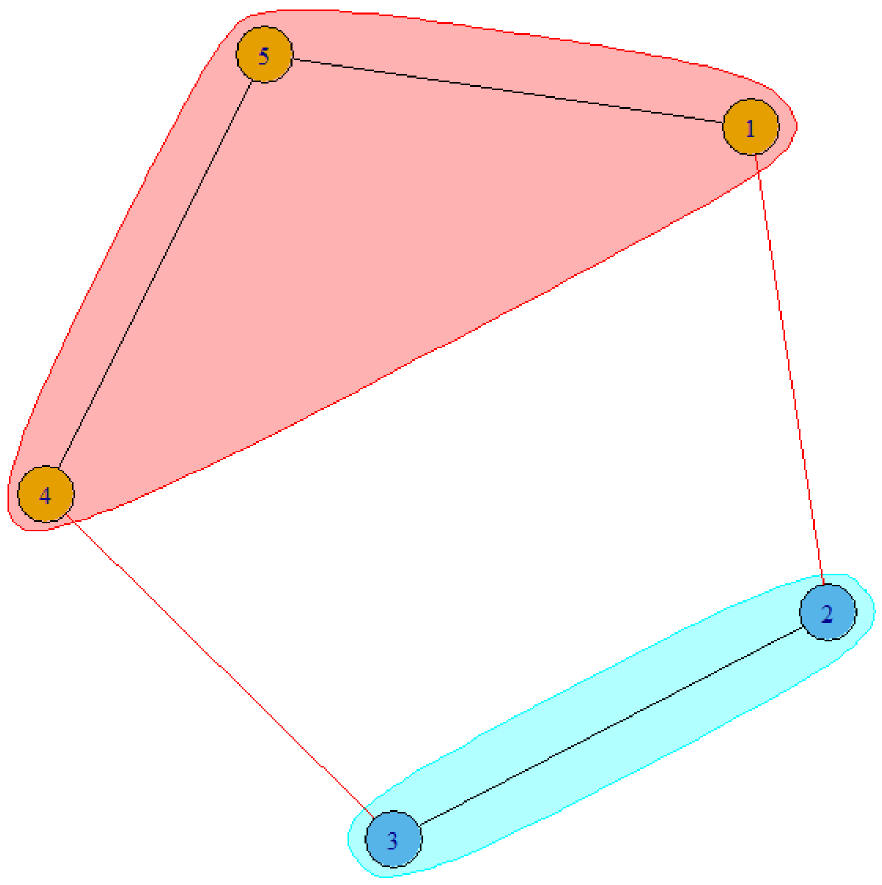
Block modeling partitions a network into distinct groups based on connectivity patterns.

Detailed Explanation:

- **Block Structure**: Groups of nodes with similar connectivity patterns.
- **Applications**: Used in sociology and criminology for identifying hidden structures in networks.

R Code:

```
> # Assign block memberships
> blockmodel <- cluster_edge_betweenness(g)
> plot(blockmodel, g)
```



8. Edges

Edges represent connections between nodes. Understanding edge properties helps analyze relationship strength and interaction patterns.

Detailed Explanation:

- **Weighted Edges**: Represent stronger or weaker relationships.
- **Directed vs. Undirected Edges**: Show one-way or mutual connections.
- **Applications**: Used in transportation, social networks, and supply chain networks.

R Code:

```
> # Get edge list
> E(g)
+ 5/5 edges from 2f3f6d0:
[1] 1--2 2--3 3--4 4--5 1--5
```

8. Subgraphs

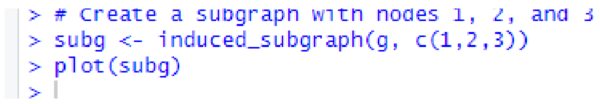
A subgraph is a smaller section of a network extracted from a larger graph.

Detailed Explanation:

- **Induced Subgraph**: Created using a subset of nodes from the original graph.

- **Applications**: Used to analyze sub-communities in large networks.

R Code:



```
> # Create a subgraph with nodes 1, 2, and 3
> subg <- induced_subgraph(g, c(1,2,3))
> plot(subg)
> 
```