Essential R Programming

(All Essential Use of R for Data Analysis and Graphics)

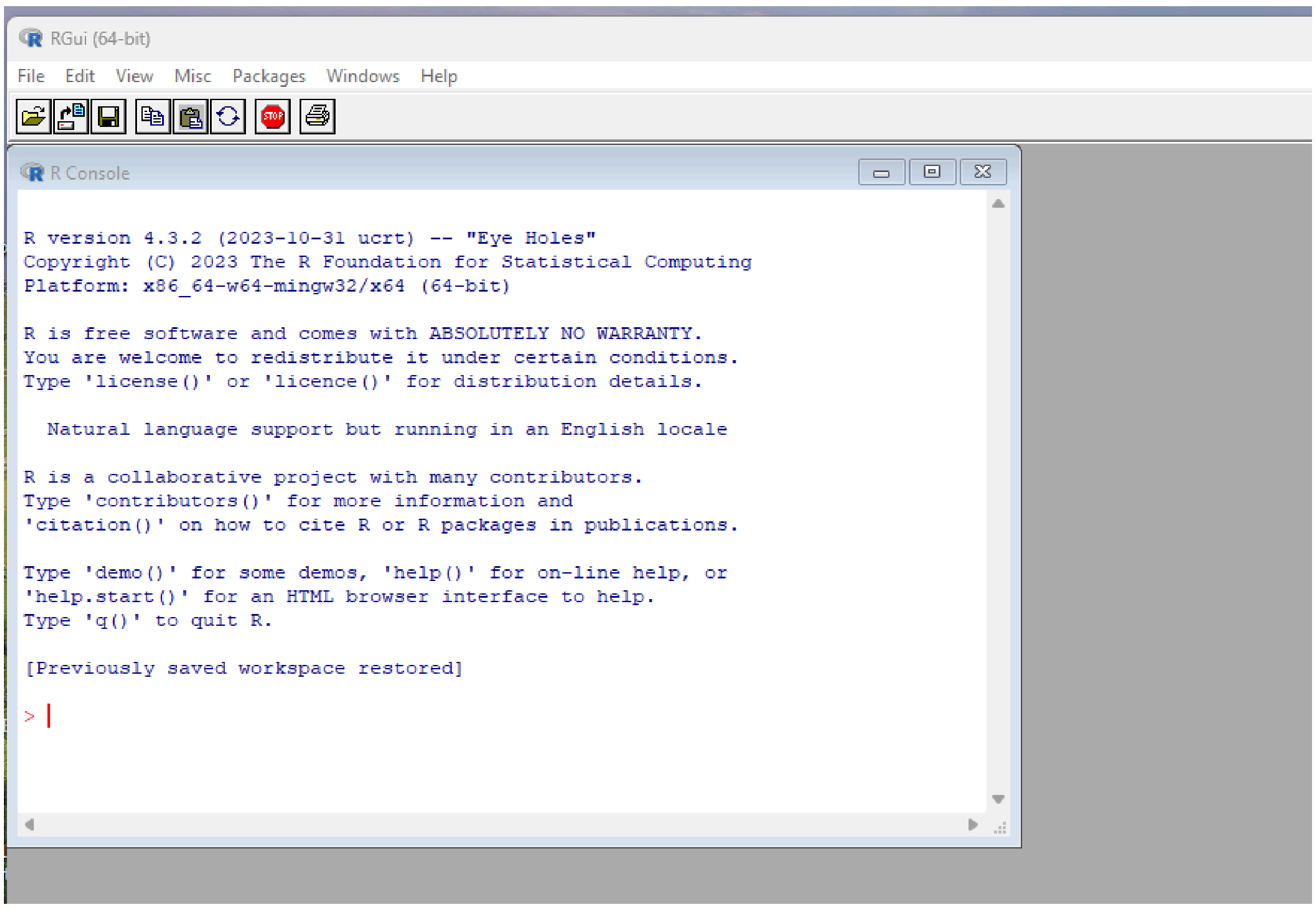
1. Starting Up

Before proceeding, ensure that R is installed on your system. If it's not yet installed, please follow the installation instructions relevant to your operating system.

- Ensure R is installed on your system.
- Follow installation instructions for your OS.
- For Windows users, download SetupR.exe from CRAN site.
- Run the installer by double-clicking the SetupR.exe icon.
- Follow on-screen instructions to complete installation.
- Download and install additional packages separately.
- Maintain separate working directories for projects.
- Refer to README file for more details.
- Windows users: Create separate desktop icons for each project directory.
- Right-click an existing R icon, select 'Copy'.
- Right-click on desktop, select 'Paste' to create a copy.
- Rename the copied icon as needed.
- Adjust properties to set 'Start in' directory.

1.1 Getting started under Windows

- Click on the R icon, or choose the icon corresponding to the project.
- For interactive use in Windows, there are multiple input methods for R commands.
- Users can type commands into the command window at the prompt.
- Figure 1 displays the command window upon starting R version 2.0.0.
- Users may select either method for inputting commands as per preference.



```
R version 4.3.2 (2023-10-31 ucrt) -- "Eye Holes"
Copyright (C) 2023 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Previously saved workspace restored]

> 
```

R-window Immediatly after starting up!.

The ">" symbol, known as the command line prompt, prompts you to begin typing your commands. For instance, if you type "2+2" and then press the Enter key, the result will be displayed on the screen.

```
> 2+2
[1] 4
>
```

In this case, the result is 4. The "[1]" notation indicates that the first requested element will be displayed, even though in this case there's only one element. The ">" symbol indicates that R is prepared for the next command.

Exiting R and Saving Workspace Objects

- To exit R, you can type `q()` at the command prompt.

  ```
  > q()
  ```

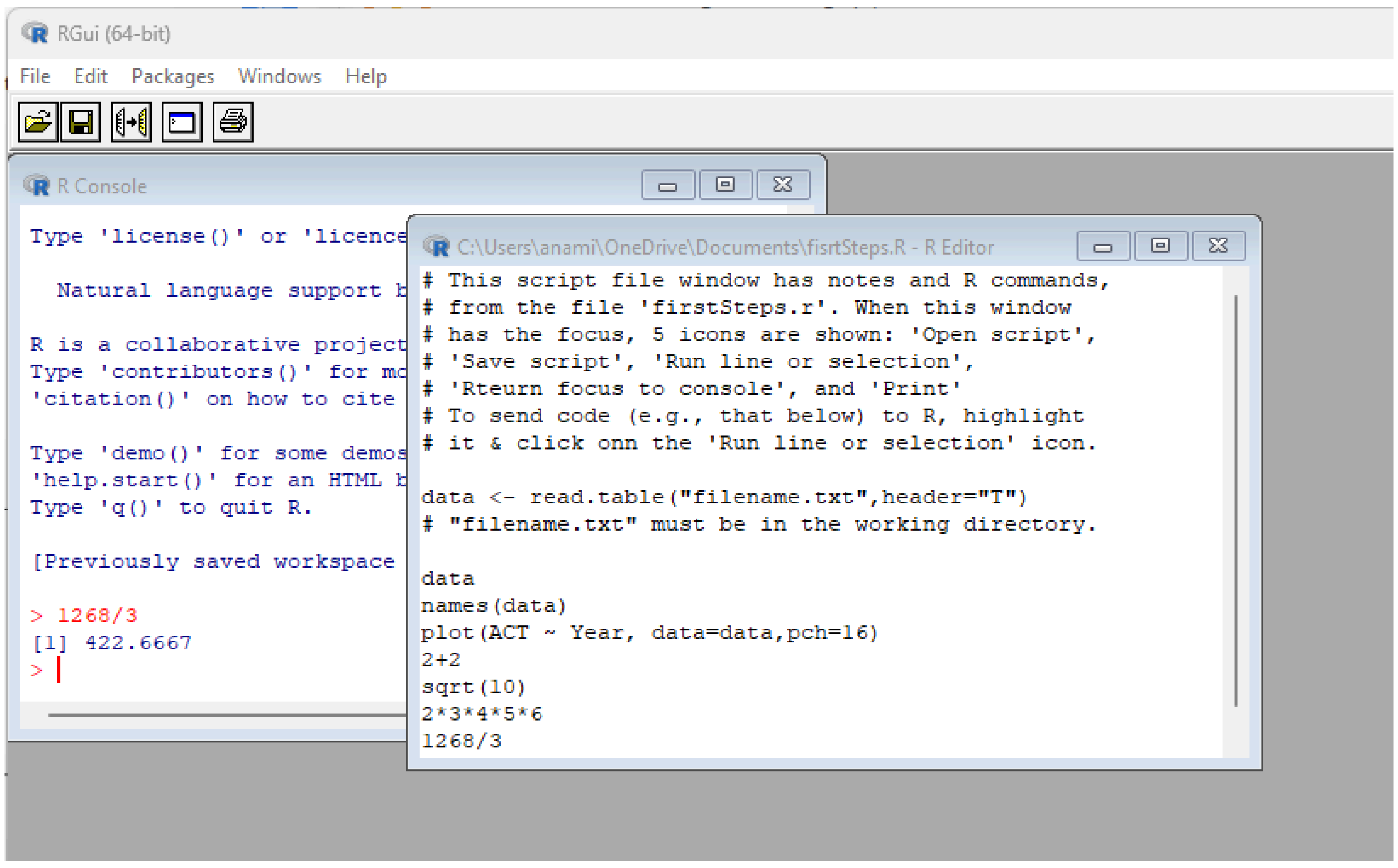
- Alternatively, you can click on the File menu and select Exit, or click on the "X" in the top right corner of the R window.

- When you exit, R will ask if you want to save the workspace image.
- Clicking "Yes" will save all the objects currently in the workspace, including those from the start of the session and any added later.

1.2 Use of an Editor Script Window

The screen snapshot in Figure2 shows a script file window. This allows input to R of statements from a file that has been set up in advance, or that have been typed or copied into the window. To get a script file window. Follow this steps:

- To access a script file window, navigate to the File menu.
- For a new blank window, select "New script".
- To load an existing file, choose "Open script..." and specify the file name.
- In the example provided (Figure 2), the file is named "firstSteps.R".
- Identify the commands intended for input to R within the script window.
- Utilize the "Run line or selection" icon, located in the middle of the script file editor toolbar in Figures 2 and 3, to send commands to R.


RGui (64-bit)
File Edit Packages Windows Help
R Console
C:\Users\anami\OneDrive\Documents\fisrtSteps.R - R Editor

```
Type 'license()' or 'licence

  Natural language support b

R is a collaborative project
Type 'contributors()' for mo
'citation()' on how to cite

Type 'demo()' for some demos
'help.start()' for an HTML b
Type 'q()' to quit R.

[Previously saved workspace

> 1268/3
[1] 422.6667
>
```

```
# This script file window has notes and R commands,
# from the file 'firstSteps.r'. When this window
# has the focus, 5 icons are shown: 'Open script',
# 'Save script', 'Run line or selection',
# 'Rteurn focus to console', and 'Print'
# To send code (e.g., that below) to R, highlight
# it & click onn the 'Run line or selection' icon.

data <- read.table("filename.txt",header="T")
# "filename.txt" must be in the working directory.

data
names(data)
plot(ACT ~ Year, data=data,pch=16)
2+2
sqrt(10)
2*3*4*5*6
1268/3
```

- On Unix systems, the common method of interaction is through the command line interface.
- Both Microsoft Windows and Linux (or Unix) offer an additional option: running R within the emacs editor.
- For Microsoft Windows users, there are appealing alternatives such as the R-WinEdt utility, which is compatible with the WinEdt editor, a shareware option, or the free tinn-R editor.

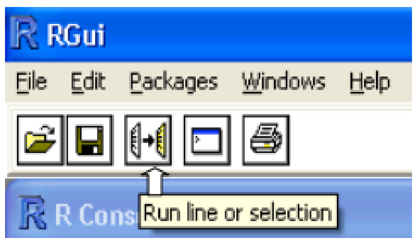


Fig. 3: This shows the five icons that appear when the focus is on a script file window. The icons are, starting from the left:
Open script, Save script, Run line or selection, Return focus to console, and Print. The text in a script file window can be edited, or new text added. Display file windows, which have a somewhat similar set of icons but do not allow editing, are another possibility.

1.3 A Short R Session

we will read into R a file that holds population figures for All countries, and total population till 2023 and indicators related to their population, then using this file to create a graph. The First 15 data in the file are:

X.	Country..or.dependency	Population..2023	Yearly.Change	Net.Change	Urban.Pop..	World.Share
1	India	1428627663	0.81%	11454490	36.3%	17.8%
2	China	1425671352	-0.02%	-215985	65%	17.7%
3	United States	339996563	0.5%	1706706	82.9%	4.2%
4	Indonesia	277534122	0.74%	2032783	59.1%	3.4%
5	Pakistan	240485658	1.98%	4660796	34.7%	3%
6	Nigeria	223804632	2.41%	5263420	53.9%	2.8%
7	Brazil	216422446	0.52%	1108948	88.4%	2.7%
8	Bangladesh	172954319	1.03%	1767947	40.9%	2.1%
9	Russia	144444359	-0.19%	-268955	74.7%	1.8%
10	Mexico	128455567	0.75%	951442	87.8%	1.6%
11	Ethiopia	126527060	2.55%	3147136	22.1%	1.6%
12	Japan	123294513	-0.53%	-657179	93.5%	1.5%
13	Philippines	117337368	1.54%	1778359	47.1%	1.5%
14	Egypt	112716598	1.56%	1726495	41.3%	1.4%
15	DR Congo	102262808	3.29%	3252596	45.5%	1.3%

In R, the <- is a left diamond bracket (<) followed by a minus sign (-). This sign is an assignment operator used to assign values to variables. When you see

```
data <- read.csv("C:/Users/anami/Downloads/World_Population.csv")
```

it means that the data read from the CSV file using the read.csv() function is assigned to the variable named "data". So, the data from the CSV file is stored in the "data" variable for further processing and analysis in R.

Now type in data at the command line prompt, displaying the object on the screen:

```
 > data
 |   X.  | Country..or.dependency | Population..2023 | Yearly.Change | Net.Change | Urban.Pop.. | World.Share |
|------|------------------------|------------------|---------------|------------|-------------|------------|
|    1 |                  India |       1428627663 |         0.81% |   11454490 |       36.3% |       17.8% |
|    2 |                  China |       1425671352 |        -0.02% |    -215985 |         65% |       17.7% |
|    3 |          United States |        339996563 |          0.5% |    1706706 |       82.9% |        4.2% |
|    4 |              Indonesia |        277534122 |         0.74% |    2032783 |       59.1% |        3.4% |
|    5 |               Pakistan |        240485658 |         1.98% |    4660796 |       34.7% |          3% |
...
```

We first of all remind ourselves of the column names:

```
> names(data)
 [1] "X."                     "Country..or.dependency."
 [3] "Population..2023."      "Yearly.Change"
 [5] "Net.Change"             "Density...P.Km.."
 [7] "Land.Area...Km.."       "Migrants..net."
 [9] "Fert..Rate"             "Med..Age"
[11] "Urban.Pop.."            "World.Share"
```

The object data is , in R parlance, a data frame. In R, the line

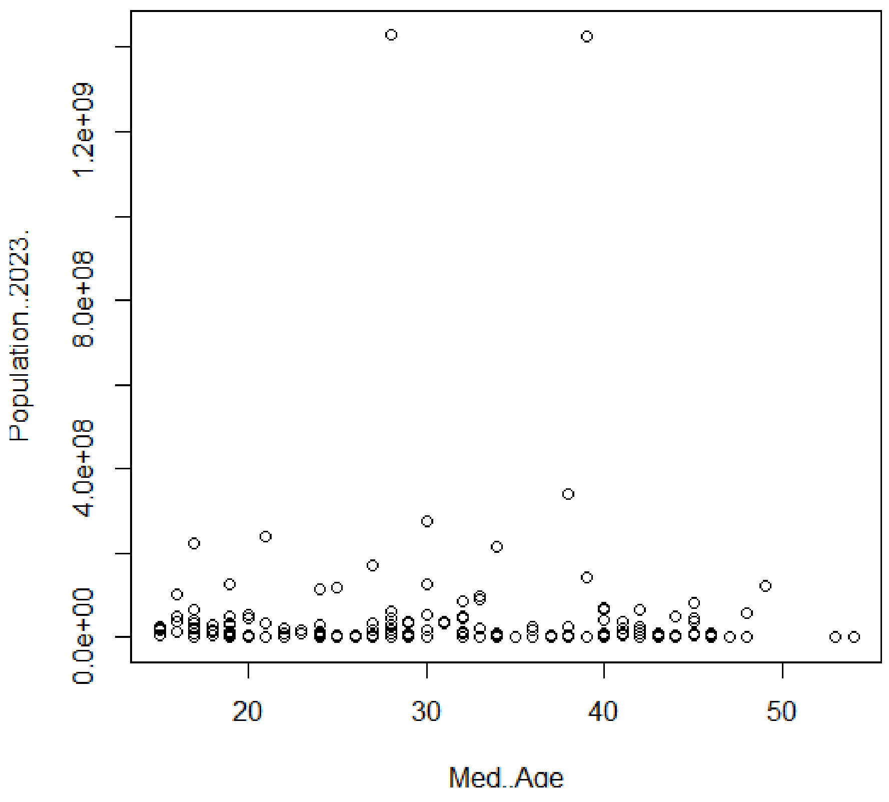
```
plot(Population..2023. ~ Med..Age, data=data)
```

creates a scatter plot where the population of 2023 (on the y-axis) is plotted against the median age (on the x-axis) using the data stored in the "data" dataframe.

- `Population..2023.` is the variable representing the population in the year 2023.
- `Med..Age` is the variable representing the median age.
- `data=data` specifies the dataframe from which the variables are taken. It indicates that the data for plotting is stored in the dataframe named "data".

So, this line of code generates a scatter plot to visualize the relationship between the population in 2023 and the median age.

This plot can be improved greatly. We can specify more informative axis labels, chnage size of the text and of the plotting symbols, and so on.



1.3.1 Entry of Data at the Command Line

A data frame is like a table with rows and columns containing information. In this case, we have two columns, and each column represents a numeric value. The data we have shows how far an elastic band moved when stretched over the end of a ruler by different amounts.

stretch	distance
46	148
54	182
48	173
50	166
44	109
42	141
52	166

The function data.frame() can be used to input these (or other) data directly at the command line. We will give the data frame the name elasticband:

```
elasticband <- data.frame(stretch=c(46,54,48,50,44,42,52),
 distance=c(148,182,173,166,109,141,166))
elasticband
```

A data.frame: 7 × 2

stretch	distance
<dbl>	<dbl>
46	148
54	182
48	173
50	166
44	109
42	141
52	166

1.3.2 Entry and/or editing of data in an editor window

To edit the data frame elasticband in a spreadsheet-like format, type

```
>elasticband <- edit(elasticband)
```

This thing is only supported in R Gui and in R studio, but not in jupyter ,google colab or any other notebook.

1.3.3 Options for read.table()

- **Parameters**: The `read.table()` function accepts various parameters in addition to the file name. One such parameter is `header`, which is set to `TRUE` if the file contains an initial row of header names. By default, `header` is set to `FALSE`.
- **Separator**: Users can specify the separator character(s) between columns using the `sep` parameter. By default, it is a space, but alternatives include `,` for comma-separated values and `\t` for tab-separated values.
- **Missing Values**: Users can specify the character(s) representing missing values using the `na.strings` parameter. The default missing value character is `NA`, but if the missing value character is a period ("."), it can be specified as `na.strings="."`.
- **Variants**: There are several variants of `read.table()` with different default parameter settings. One such variant is `read.csv()`, which is suitable for reading comma-delimited (csv) files generated from Excel spreadsheets.
- **Handling Errors**: If `read.table()` detects lines in the input file with different numbers of fields, data input will fail, and an error message will be displayed. In such cases, the `count.fields()` function can be used to report the number of fields identified on each line of the file.
- **Editor Window**: The data frame `elasticband` is shown in the editor window, indicating successful data import and framing.

1.3.4 Options for plot() and allied functions

The function plot() and related functions accept parameters that control the plotting symbol, and the size and colour of the plotting symbol.

1.4 Further Notational Details

As noted earlier, the command line prompt is

```
>
```

R commands (expressions) are typed following this prompt . There is also a continuation prompt, used when, following a carriage return, the command is still not complete. By default, the continuation prompt is

```
+
```

In these notes, we often continue commands over more than one line, but omit the + that will appear on the commands window if the command is typed in as we show it. For the names of R objects or commands, case is significant. Thus Austpop is different from austpop. For file names however, the Microsoft Windows conventions apply, and case does not distinguish file names. On Unix systems letters that have a different case are treated as different. Anything that follows a # on the command line is taken as comment and ignored by R.

Note: Recall that, in order to quit from the R session we had to type q(). This is because q is a function. Typing q on its own, without the parentheses, displays the text of the function on the screen. Try it!

1.5 On-line Help

To access help topics in R for Windows, you can type

```
>help()
```

in the command prompt. Alternatively, you can click on the "Help" menu item and use keywords to search for specific topics. For example, to get help on a particular R function like `plot()`, you can type

```
>help(plot)
```

Two useful search functions are

```
>help.search()
```

and

```
>apropos()
```

For instance,

```
>help.search("matrix")
```

lists all functions with help pages containing the term "matrix", while `apropos("matrix")` lists all function names that include "matrix".

The function `help.start()` opens a browser window providing access to comprehensive documentation on syntax, packages, and functions.

Experimenting with R functions often aids in understanding their precise actions.

1.6 The Loading or Attaching of Datasets

```
The recommended way to access datasets provided with these notes is to attach the file `usingR.RData`,
available from the author's web page.
Follow these steps:

1. Place the `usingR.RData` file in the working directory.
2. From within the R session, type:
```

```
> attach("usingR.RData")
```

Datasets mentioned in these notes, not included with R (e.g., from datasets or MASS packages), should then be available without further action.

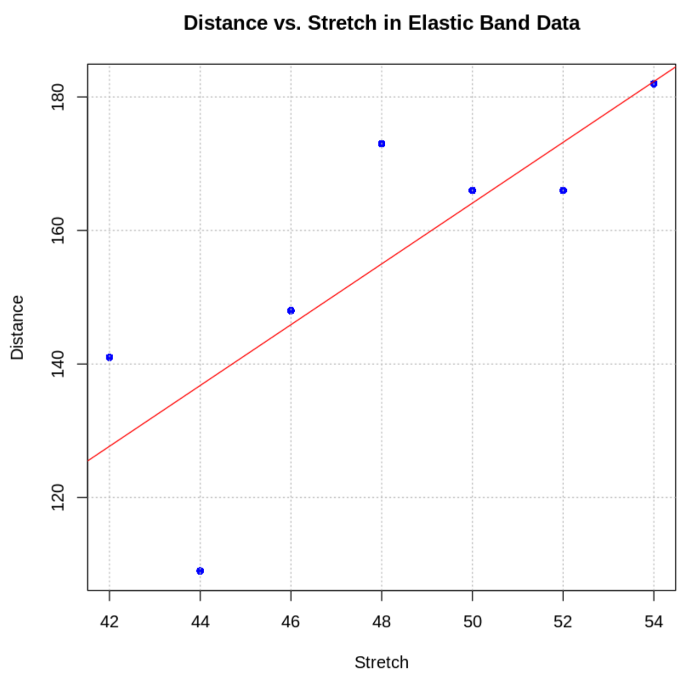
Users can also load (`load()`) or attach (`attach()`) specific files. These operations have a similar effect, but with `attach()`, datasets are loaded into memory only when required for use.

It's important to distinguish between attaching image files and data frames. The attachment of data frames will be discussed later in these notes.

1.7 Exercises

1. In the data frame elasticband from section 1.3.1, plot distance against stretch.

```
# Define custom colors
plot_color <- "blue"
grid_color <- "gray"

# Plot with customized parameters
plot(elasticband$stretch, elasticband$distance,
     xlab = "Stretch", ylab = "Distance",
     main = "Distance vs. Stretch in Elastic Band Data",
     col = plot_color, pch = 16)  # Set point color and shape
grid(col = grid_color)  # Add grid lines

# Add linear regression line
fit <- lm(distance ~ stretch, data = elasticband)  # Fit linear regression model
abline(fit, col = "red")  # Add regression line in red color
```



This code customizes a plot in R. Here's what each part does:

Define Custom Colors: Two custom colors are defined using hexadecimal codes, one for the plot points (plot_color) and one for the grid lines (grid_color).

Plot with Customized Parameters: The plot() function is used to create a scatter plot of the data stored in the elasticband data frame. Custom parameters are provided to label the axes (xlab, ylab), add a title (main), set the color and shape of the plot points (col, pch), and add grid lines using the grid() function.

Add Linear Regression Line: The lm() function is used to fit a linear regression model to the data, with distance as the dependent variable and stretch as the independent variable. The resulting model (fit) is then used to add a regression line to the plot using the abline() function, with the line color set to red.

2. The following ten observations, taken during the years 1970-79, are on October snow cover for Eurasia.

(Snow cover is in millions of square kilometers):

year	snow.cover
1970	6.5
1971	12.0
1972	14.9
1973	10.0
1974	10.7
1975	7.9
1976	21.9
1977	12.5
1978	14.5
1979	9.2

i. Enter the data into R. [Section 1.3.1 showed one way to do this. To save keystrokes, enter the successive years as 1970:1979]

ii. Plot snow.cover versus year.

iii Use the hist() command to plot a histogram of the snow cover values.

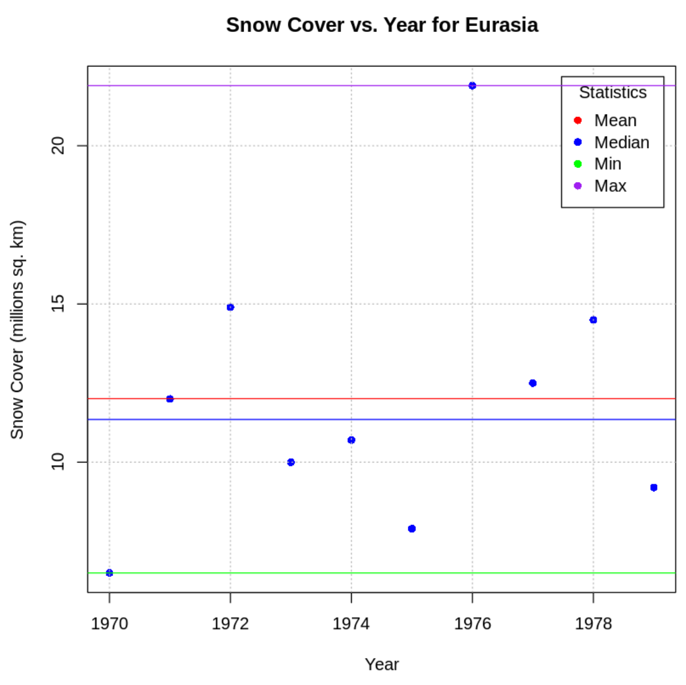
iv. Repeat ii and iii after taking logarithms of snow cover.

```
#(i) Define the data
years <- 1970:1979
snow_cover <- c(6.5, 12.0, 14.9, 10.0, 10.7, 7.9, 21.9, 12.5, 14.5, 9.2)

# Combine years and snow_cover into a single data vector
data <- data.frame(year = years, snow_cover = snow_cover)
data
```

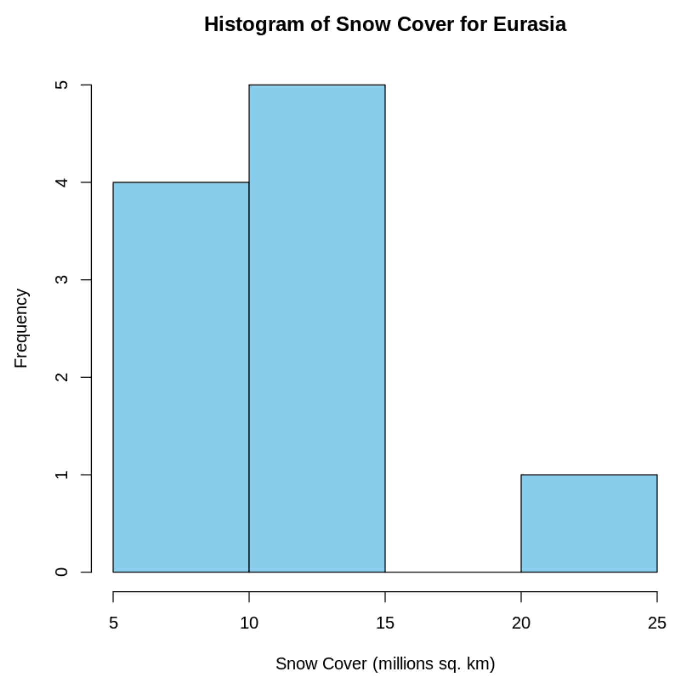
A data.frame: 10 × 2

year	snow_cover
<int>	<dbl>
1970	6.5
1971	12.0
1972	14.9
1973	10.0
1974	10.7
1975	7.9
1976	21.9
1977	12.5
1978	14.5
1979	9.2

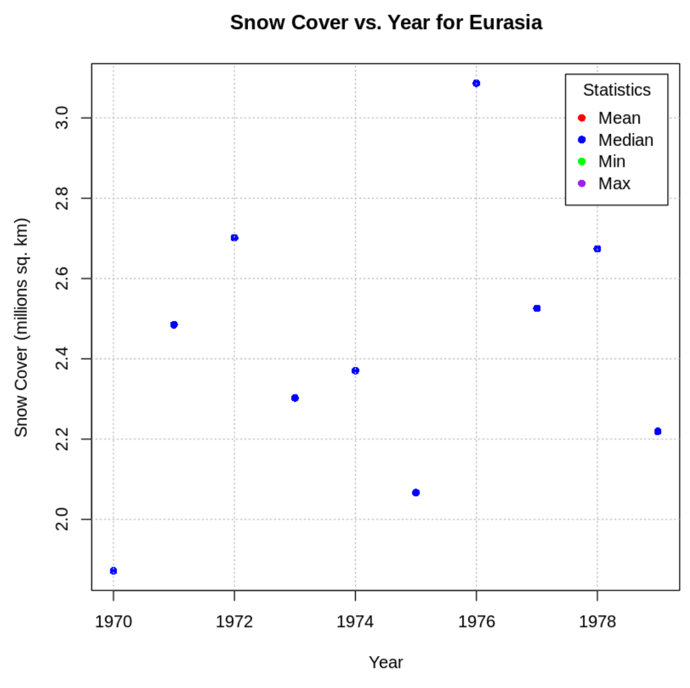
```
# Define custom colors
plot_color <- "blue"
grid_color <- "gray"

#(ii) Plot with customized parameters
plot(years, snow_cover,
     xlab = "Year", ylab = "Snow Cover (millions sq. km)",
     main = "Snow Cover vs. Year for Eurasia",
     col = plot_color, pch = 16)  # Set point color and shape
grid(col = grid_color)  # Add grid lines

# Add statistical information
legend("topright", inset = 0.02, legend = c("Mean", "Median", "Min", "Max"),
       col = c("red", "blue", "green", "purple"), pch = c(16, 16, 16, 16),
       title = "Statistics")
abline(h = mean(snow_cover), col = "red")  # Add mean line
abline(h = median(snow_cover), col = "blue")  # Add median line
abline(h = min(snow_cover), col = "green")  # Add min line
abline(h = max(snow_cover), col = "purple")  # Add max line
```



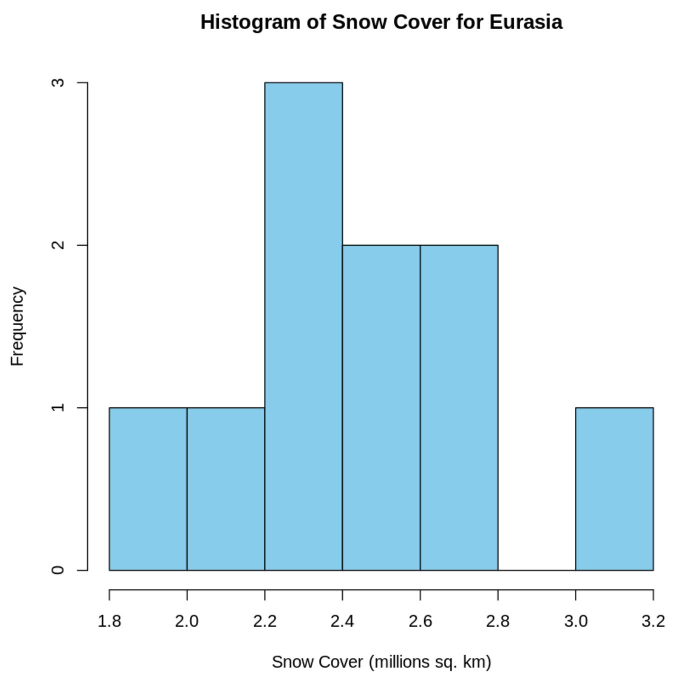
```r
# Define custom colors
hist_color <- "skyblue"

#(iii) Plot histogram with customized parameters
hist(snow_cover,
     xlab = "Snow Cover (millions sq. km)", ylab = "Frequency",
     main = "Histogram of Snow Cover for Eurasia",
     col = hist_color)  # Set histogram color
```



```r
#(iv) Repeating ii and iii after taking logarithms of snow cover.
# Define custom colors
plot_color <- "blue"
grid_color <- "gray"

#(ii) Plot with customized parameters
plot(years, log(snow_cover),
     xlab = "Year", ylab = "Snow Cover (millions sq. km)",
     main = "Snow Cover vs. Year for Eurasia",
     col = plot_color, pch = 16)  # Set point color and shape
grid(col = grid_color)  # Add grid lines

# Add statistical information
legend("topright", inset = 0.02, legend = c("Mean", "Median", "Min", "Max"),
       col = c("red", "blue", "green", "purple"), pch = c(16, 16, 16, 16),
       title = "Statistics")
abline(h = mean(snow_cover), col = "red")  # Add mean line
abline(h = median(snow_cover), col = "blue")  # Add median line
abline(h = min(snow_cover), col = "green")  # Add min line
abline(h = max(snow_cover), col = "purple")  # Add max line
```



```r
# Define custom colors
hist_color <- "skyblue"

#(iii) Plot histogram with customized parameters
hist(log(snow_cover),
     xlab = "Snow Cover (millions sq. km)", ylab = "Frequency",
     main = "Histogram of Snow Cover for Eurasia",
     col = hist_color)  # Set histogram color
```



3. Input the following data, on damage that had occurred in space shuttle launches prior to the disastrous launch of Jan 28 1986. These are the data, for 6 launches out of 24, that were included in the pre-launch charts that were used in deciding whether to proceed with the launch. (Data for the 23 launches where information is available is in the data set orings, from the DAAG package.)

Temperature	Erosion incidents	Blowby incidents	Total incidents
53	3	2	5
57	1	0	1
63	1	0	1
70	1	0	1
70	1	0	1
75	0	2	1

Enter these data into a data frame, with (for example) column names temperature, erosion, blowby and total. (Refer back to Section 1.3.1).

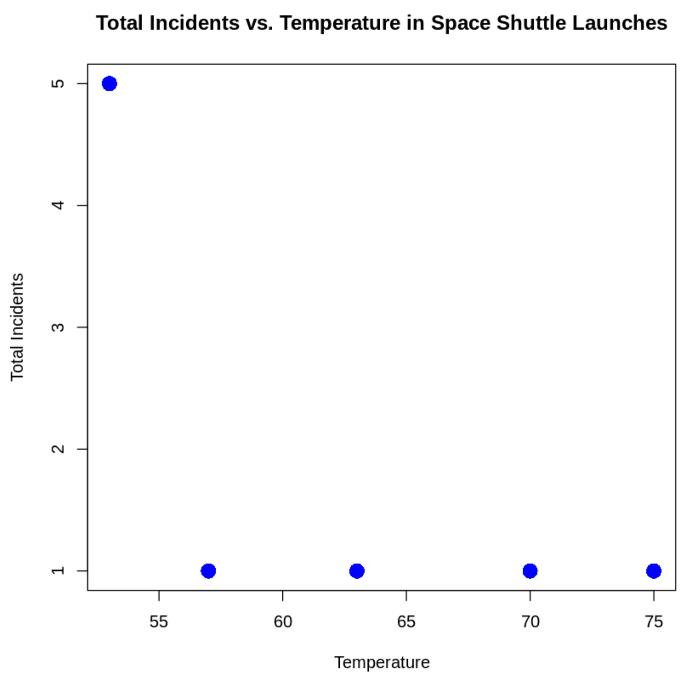
Plot total incidents against temperature and also Create histogram plots of all columns in one place.

```
# Create a data frame with the provided data
shuttle_data <- data.frame(
  Temperature = c(53, 57, 63, 70, 70, 75),
  Erosion_incidents = c(3, 1, 1, 1, 1, 0),
  Blowby_incidents = c(2, 0, 0, 0, 0, 2),
  Total_incidents = c(5, 1, 1, 1, 1, 1)
)

shuttle_data
```

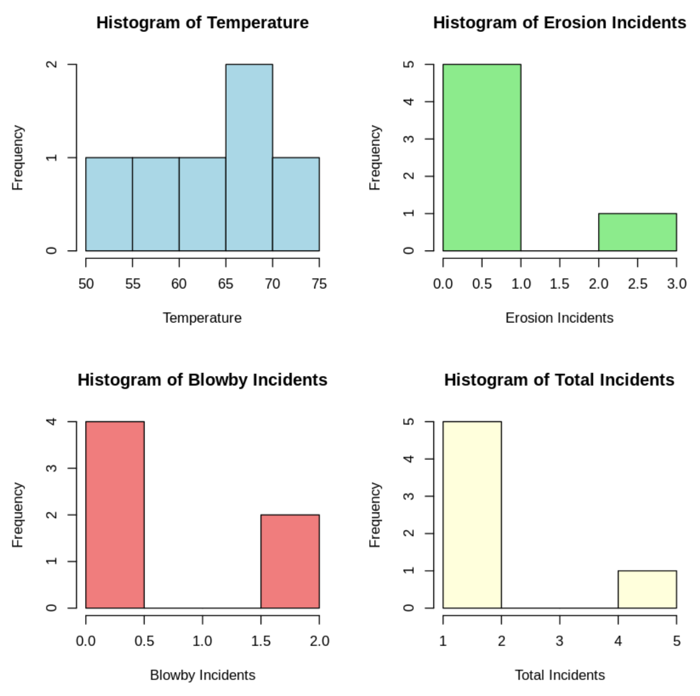
A data.frame: 6 × 4

Temperature	Erosion_incidents	Blowby_incidents	Total_incidents
<dbl>	<dbl>	<dbl>	<dbl>
53	3	2	5
57	1	0	1
63	1	0	1
70	1	0	1
70	1	0	1
75	0	2	1

```
# Plot total incidents against temperature with larger circle size
plot(shuttle_data$Temperature, shuttle_data$Total_incidents,
     xlab = "Temperature", ylab = "Total Incidents",
     main = "Total Incidents vs. Temperature in Space Shuttle Launches",
     col = "blue", pch = 16, cex = 2)
```



```
# Create histogram plots of all columns in one
par(mfrow = c(2, 2))  # Set up a 2x2 layout for multiple plots

# Histogram of Temperature
hist(shuttle_data$Temperature,
     xlab = "Temperature", ylab = "Frequency",
     main = "Histogram of Temperature",
     col = "lightblue")

# Histogram of Erosion incidents
hist(shuttle_data$Erosion_incidents,
     xlab = "Erosion Incidents", ylab = "Frequency",
     main = "Histogram of Erosion Incidents",
     col = "lightgreen")

# Histogram of Blowby incidents
hist(shuttle_data$Blowby_incidents,
     xlab = "Blowby Incidents", ylab = "Frequency",
     main = "Histogram of Blowby Incidents",
     col = "lightcoral")

# Histogram of Total incidents
hist(shuttle_data$Total_incidents,
     xlab = "Total Incidents", ylab = "Frequency",
     main = "Histogram of Total Incidents",
     col = "lightyellow")
```


Histogram of Temperature
Frequency
Temperature
Histogram of Erosion Incidents
Erosion Incidents
Histogram of Blowby Incidents
Blowby Incidents
Histogram of Total Incidents
Total Incidents