TIME TABLE

PRACTICE	MIN
Prerequisites	15
Loading packages	15
Looking at data	15
Factors vectors	15
Summary stats	15
Boxplots	10
Scatterplots	10
Bar charts	10
Customization	15
TOTAL	120

DONE PREREQUISITES

1. Check that R is installed on your machine. All of these are equivalent but lead to different interfaces:

   - open Emacs and open an R session with `M-x R`
   - open the CMD line terminal and enter the command `R`
   - open the CMD line terminal and enter the command `Rgui`

2. Check that you can execute R code blocks inside Emacs: execute the following code block named **??** by moving the cursor anywhere on the block - either on the metadata or on the line of code - and enter `C-c C-c`.

   The `result` of **??** is R `version` information. Because of the output size, it is automatically wrapped in an `example` block.

   ```
   version
   ```

3. Do you notice anything special about this last code block? Write it into a "quote" block below (create with `<q TAB`):

   > *The code block is not executed in a permanent R session.*

4. Alternatively to executing the block in the Org-mode buffer, you can move the cursor on the block and enter `C-c '`. This will open the source code in a new buffer where you can execute it or edit it. This time, the output will appear in the `*R*` buffer instead of the Org-mode file.

5. If R is not installed, you need to install it. If you cannot execute R code blocks, you are probably missing the correct Emacs init file `/.emacs`: download the file from here. You might also miss the ESS (Emacs Speaks Statistics) package. Try `M-x load-library ESS RET`.

6. If you're in an R session now, exit by entering `q()`. R is case-sensitive, so this must be lower-case. When asked if you want to save the workspace, say **no**[1].

7. To get to the footnote at the end of the last sentence, move the cursor on it and enter `C-c C-o` (same command as opening a link or an image). You return here with `C-c &`.

DONE LOADING PACKAGES

We work with the `whiteside` data frame from the `MASS` package. You have to load the package to access this data set.

1. Open an R session or shell buffer in Emacs with `M-x R`. You will be asked for the project directory in the Emacs minibuffer. Accept whatever choice is given to you.

2. The screen is now split and you see the R shell below. The first command (`setwd`) sets the working directory. At the `>` prompt, enter: `??whiteside`. The command `??` performs a fuzzy search through all packages available for your R session.

3. If the `MASS` package is not installed, `??` will not find anything. In this case, install it with the command `install.packages(MASS)` [2].

[1] If you say yes, R will save a copy of all your commands in that session in a file `.Rhistory`, and it will save all data in a file `.RData` to recreate your work space the way you left it.

[2] You can run this command in any case - installing `MASS` does not take very long and re-installing the package does no harm, it only takes time - unless your version of R is not in sync with the package. In this case, install the `remotes` package first to install a specific version of `MASS`.

4. Once the package is installed, try `??whiteside` again. Open the link `MASS::whiteside` in the web page that appears to get a short description of the dataset and read it[3].

5. To analyse the dataset, it needs to be loaded. To do this, load the `MASS` package with the `library` command and confirm with `search()`, which lists all packages that have been loaded.

   Run the code chunk below with `C-c C-c` [4] - if you cannot see the code but instead see a line ending with `...`, bring the cursor to that line and open the section with the `<TAB>` key.

```
library(MASS)
search()
```

6. You can also use the functions `any`, `grep` and `search` to check that `MASS` is loaded:

```
s <- search()
p <- "MASS"
g <- grep(p,s)
any(g)  # same as any(grep("MASS",search()))
```

```
[1] TRUE
```

7. In the next code block, try to get the same result but with a one-line command (without storing any objects in the process):

```
any(grep("MASS",search()))
```

```
[1] TRUE
```

[3] The format of this documentation is common for R, and it imitates the format of UNIX manual pages. After a *description* and a *usage* note, the *format* is described in terms of the variables. The *source* and *references* given. At the end, the *examples* section provides examples, which sometimes can be called interactively with the `example` function, e.g. `example(head)`.

[4] In class, I often use the `org-present` package to present Org-mode files and hide the metadata (e.g. for code blocks). If you like this, see here for a tutorial including the code to put in your `.emacs` file.

8. Interactions with the OS like loading a package are not remembered by the system unless they are tied to a named R session. In the code block below, replace `???` with the name of the R session that you started in (1), e.g. `*R*`. Then run the code block again with `C-c C-c`.

```
library(MASS)
search()
```

9. What happens if you just run the block with `C-c C-c` ?

10. Check your buffer list with `C-x C-b` (to return to the last or any other open buffer, use `C-x b` instead).

DONE LOOKING AT DATA

Before working with a data set, you need to take at least a superficial view at its entries (values).

1. Display the first six records of the `whiteside` data frame. Run the code block `??` below with `C-c C-c`.

```
head(x=whiteside)
```

2. Show only the first three records using `head`, by adding the argument `n=3` to the function call in `??` below, and run the block. Attributes are separated by commata: `f(x=..., n=...)`

```
head(x=whiteside,___)  # show first n=3 lines of x
```

3. Show the last three records using the function `tail` using the block `??`, and run the block.

```
-------------
```

4. What does the first line of the data frame show? What do the following lines show? How many lines are there?

   - **The first line shows** ...
   - **The following lines show** ...

5. What data does the data frame `whiteside` as a whole show?

   The `whiteside` data frame shows ...

SOLUTION

1. Show only the first three records using `head`, by adding the argument `n=3` to the function call in `??` below, and run the block.

   ```
   head(x=whiteside, n=3)
   ```

2. Show the *last* three records using the function `tail` using the block `??`, and run the block.

   ```
   tail(x=whiteside, n=3)
   ```

3. What does the first line of the data frame show? What do the following lines show?

   - **The first line shows** the names of the fields/variables recorded
   - **The following lines show** the first records of the data set

4. What data does the data frame `whiteside` contain?

   The `whiteside` data frame shows the weekly average heating gas consumption and the weekly average outside temperature for two successive winters, the first before, and the second after Whiteside installed insulation in his house.

DONE FACTOR VECTORS

To get a more detailed view at the data frame, we display its structure using the generic[5] `str` function.

1. Create a named R code block called `structure` by entering `<s TAB`. Add the header arguments[6]:

[5] To find out more about any R function, go to the console and look up the help, as in `help(str)` or (equivalently) `?str`. Generic functions work with (almost) any R object, and their output depends on the object type.

[6] This Org-mode code block header argument lets the computer know that you run R in a session buffer `*R*` and that you want to see the results (if any) right here.

```
R :session *R* :results output
```

Note: `*R*` should be the name of your R session buffer. If you don't have one yet, running the code block will create one, and you don't have to name the `:session` in the header.

[In class, we should have defined `<r` *as a template.]*

—– PUT YOUR CODE BELOW THIS LINE —–

2. In the codeblock `structure`, make a function call of `str` to the data frame `whiteside` to compactly display its structure, and run the code with `C-c C-c`. Make sure you understand the output.

3. The variable `Insul` is a *factor*, a vector used to represent *categorical* variables. You can extract its values (called *levels*) as shown in the code block `??` below using the operator `$`.

   First, store the values of the `Insul` vector in an object `x`.

   Next, print the structure of the vector.

   ```
   x <- whiteside$Insul  # store Insul in x
   str(x)                # show structure of x
   ```

4. `levels` defined for a `factor` vector represent its only possible values. Trying to insert a new value as in the code block `??` generates an error message: run the code block.

   ```
   x[2] <- "Unknown"
   ```

5. This is so because `x` is a `factor`. Show this by printing its object `class` and by printing the value of `is.factor` of `x`.

   ```
   class(x)
   is.factor(x)
   ```

   ```
   [1] "factor"
   [1] TRUE
   ```

6. We can use the function `as.character` to convert the `factor` into a `character` variable. Now, the redefinition works.

(a) Store `whiteside$Insul as.character` in `x`.

(b) Print the structure of `x` - it's now a `character` vector.

(c) Now replace `x[2]` by `"Unknown"`.

(d) Print the structure of `x` again to check the insertion.

```
x <- as.character(whiteside$Insul) # convert factor to character
str(x)
x[2] <- "Unknown" # replace the 2nd element of the vector
str(x)
```

DONE SUMMARY STATS

R is strong on statistics. The `summary` function returns simple statistical properties of each variable.

Create a named code block `??`. In it, call the function `summary` on the `whiteside` data frame. Open the explanatory notes below with `<TAB>`.

—– PUT YOUR CODE BELOW THIS LINE —–

The output contains the `mean` (average of the variable `x` over all records), and *Tukey's five-number summary*[7].

- *sample minimum*: smallest number in the dataset
- *lower quartile*: value for which 25% are smaller or equal
- *upper quartile*: value for which 75% are smaller or equal
- *sample median*: middle value of the data set
- *sample maximum*: larges value in the dataset

Below, create a `quote` block with `<q TAB`. In the block, write an observation of the `summary` data - at least one sentence for each variable that would help someone else reading this `summary` understand what he sees.

[7] For factors, if the number of levels is > 6, only the five most frequently occurring levels are listed, the others are lumped in one 'other' category. For L = 2 as here, all values are accounted for.

SOLUTION

```
summary(whiteside)
```

```
   Insul         Temp             Gas
Before:26   Min.   :-0.800   Min.   :1.300
After :30   1st Qu.: 3.050   1st Qu.:3.500
            Median : 4.900   Median :3.950
            Mean   : 4.875   Mean   :4.071
            3rd Qu.: 7.125   3rd Qu.:4.625
            Max.   :10.200   Max.   :7.200
```

Interpretation:

The `summary` data for the categorical (nominal) variable `Insul` report the number of observations (days) before and after the insulation was implemented.

For `Temp`, I notice that the temperature ranged between a little below freezing (0C) and cool (10C), with an average of about 4 degrees.

For `Gas`, the distribution also seems to be quite clustered around the average. The range of gas consumption per week is considerable (between 1.3 and 7.2 cubic feet).

Min/Max of `Temp` and `Gas` presumably are inversely correlated to one another.

The measurements of temperature and gas are accurate to the 3rd decimal.

DONE BOXPLOTS

We'll finish this practice run with a few glimpses into R's graphics capabilities.

Following up from the output of `summary`, a `boxplot` is a graphical representation of Tukey's five-number summary.

1. Run the code block `??` below to generate a `boxplot`[8]. Open the graphical result with `<F6>` and close it again with `<F7>`[9].

[8] Notice the changed header arguments: `:results output graphics file` to generate a graphics file, and `:file boxplot.png` as the file name.

[9] This key is bound to the Emacs Lisp function `org-display-inline-images`. The key

```
boxplot(Gas ~ Insul, data = whiteside)
```

In the boxplot, the "whiskers" at the top and the bottom represent the sample **minimum** and **maximum**. The "box" is bounded by the **upper quartile** at the top, and by the **lower quartile** at the bottom. The thick line in the middle is the **median** value. In the `After` level on the right hand side of the plot you see an open circle at the bottom: that's an **outlier**, which is "unusually small". The sample minimum therefore is the "smallest non-outlying value", and not the true minimum[10]

SOLUTION

(a) Plot `whiteside$Gas` splitting up the data according to factor levels.

```
boxplot(Gas ~ Insul, data = whiteside)
abline(h = mean(whiteside$Gas), col="blue", lwd=2, lty=2)
```

sequence `C-c C-x C-v` *toggles* the display of inline images (i.e. switches it on and off). `<F6>` only makes the images visible, `<F7>` only makes them disappear.

[10] Values that are at least 1.5 times the interquartile range (IQR, difference between upper and lower quartile) above/below of the upper/lower quartile are outliers.

.

2. Create a boxplot `boxplot2.png`, that shows the variable `Temp` instead of `Gas`. Only a small change is necessary to do this.
—– PUT YOUR CODE BELOW THIS LINE —–

3. When comparing with the output of `summary`, we're missing the average value, or `mean`. Modify your code blocks by adding these two lines below the `boxplot` command, and run each block again: the `abline` function simply draws a horizontal line at the average.

```
avg_Gas <- mean(whiteside$Gas)
abline(h = avg_Gas, col="blue", lwd=2)
```

```
avg_Temp <- mean(whiteside$Temp)
abline(h = avg_Temp, col="blue", lwd=2)
```

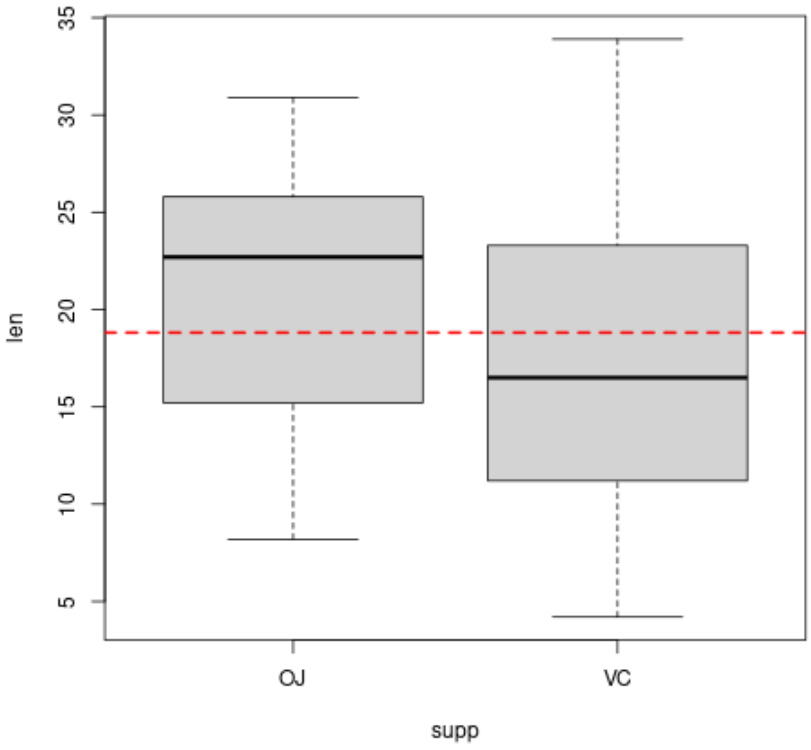
4. Can you transfer this to recreating the boxplot for the `ToothGrowth` data set, showing the distributions of the length (`len`) of the teeth as a function of the Vitamin C supply type (`supp`)?

(b) Plot `whiteside$Temp` splitting up the data according to factor levels.

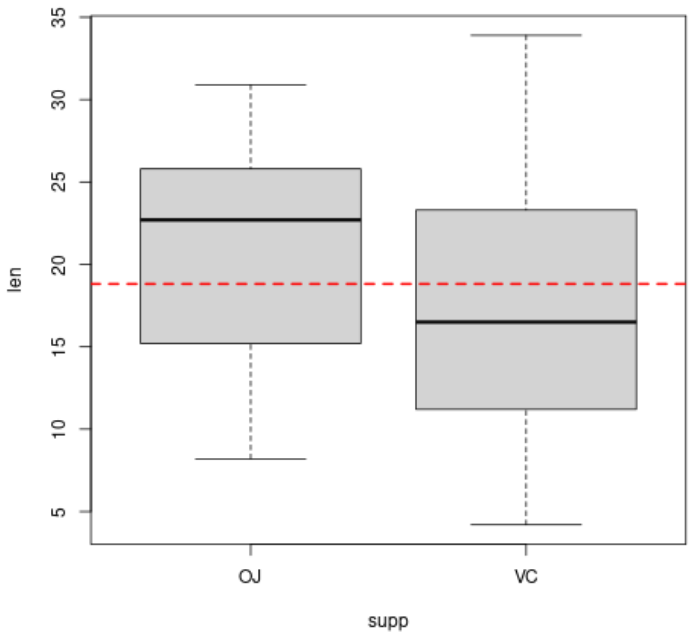
```
boxplot(Temp ~ Insul, data = whiteside)
abline(h = mean(whiteside$Temp), col="red", lwd=2, lty=2)
```

(c) Can you transfer this to recreating the boxplot for the `ToothGrowth` data set, showing the distributions of the length (`len`) of the teeth as a function of the Vitamin C supply type (`supp`)?

```
boxplot(len ~ supp, data=ToothGrowth)
abline(h=mean(ToothGrowth$len),col="red",lty=2,lwd=2)
```



Add the average length as a thick dashed red line to the plot.



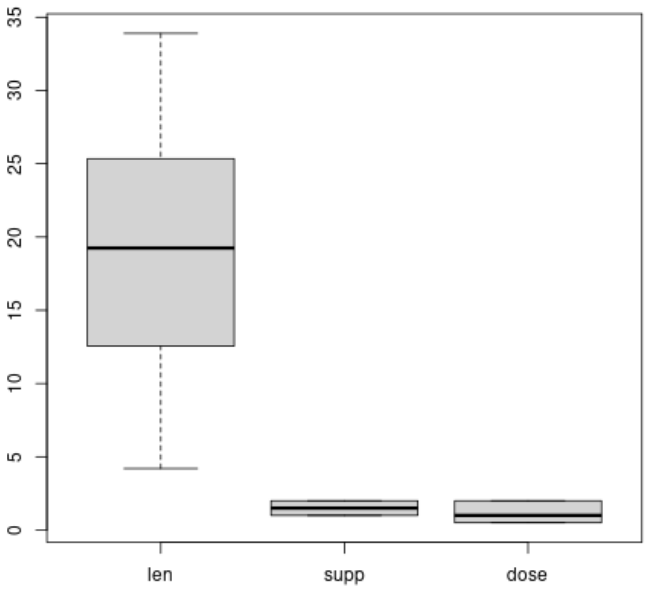
Add the average length as a thick dashed red line to the plot.

(d) Is `boxplot` a "generic" R function?

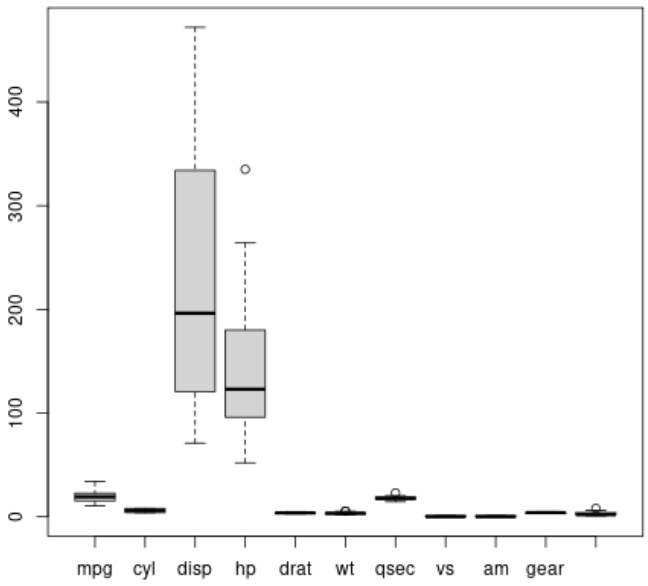
```
methods(boxplot)
```

```
[1] boxplot.default  boxplot.formula* boxplot.matrix
see '?methods' for accessing help and source code
```

```
boxplot(ToothGrowth)
```




```
boxplot(mtcars)
```



Looking at the last example, which boxplots make sense for `mtcars`?

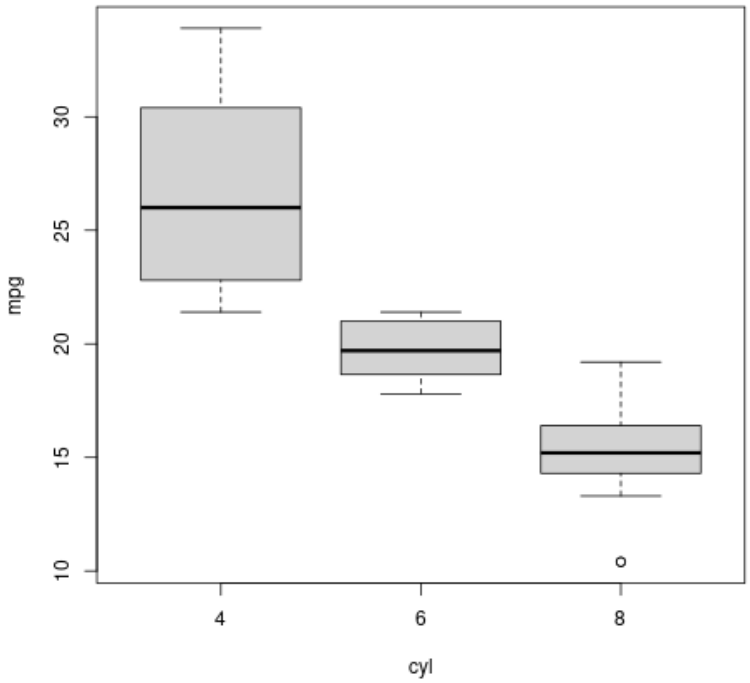
```
str(mtcars)
```

```
'data.frame': 32 obs. of  11 variables:
 $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
 $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
 $ disp: num  160 160 108 258 360 ...
 $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
 $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
 $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
 $ qsec: num  16.5 17 18.6 19.4 17 ...
 $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
 $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
 $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
 $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

Answer: All categorical variables are suitable as independent variables, and all truly numeric variables as dependent variables.

Example: Miles-per-gallon as a function of the number of cylinders

```
boxplot(mpg ~ cyl, data=mtcars)
```



DONE SCATTERPLOTS

The `plot` function is another versatile, generic function in R. Applied to a data frame, it produces a matrix of *scatterplots*, showing how each variable relates to the others.

(a) Run the code block named **??** below. Open the notes to see the explanation of this *scatterplot* matrix with `<TAB>`.

```
plot(whiteside)
```

The diagonal elements of the output identify the x-axis in all plots of that column, and the y-axis in all the other plots of that row. E.g. the matrix element `[3,2]` (3rd row, 2nd column) below the diagonal element `Temp` plots `y = Gas` against `x = Temp`, while the element `[2,3]` (2nd row, 3rd column) plots `y = Temp` against `x = Gas`.

In the four plots involving the `factor` variable `Insul`, the two `levels` of `Insul`, `Before` and `After` are represented by 1 and 2. You can e.g. see at one glance from `[3,1]` or `[1,3]` that the `Gas` values are smaller when `Insul = 2`, i.e. less heating gas was consumed after insulation was installed than before.

(b) Create another code block named 4b that uses `plot` to plot only the `Temp` variable of the `whiteside` data set. Can you explain the graph? *Tip:* Use `sort` to sort the values and plot again.

—– PUT YOUR CODE BELOW THIS LINE —–

```
plot1.png
```

The left set of data points represents the 26 values with `Insul=Before`, the right set of data points represents the 30 values with `Insul=After`.

These points represent average weekly winter temperatures recorded before and after the wall insulation in Whiteside's house. The observations are ordered from coldest to warmest within each heating season.

SOLUTIONS

(a) Run the code block named **??** below. Open the notes to see the explanation of this *scatterplot* matrix.

```
plot(whiteside)
```

(b) Create another code block 4b that uses `plot` to plot only the `Temp` variable of the `whiteside` data set.

```
plot(whiteside$Temp)
```

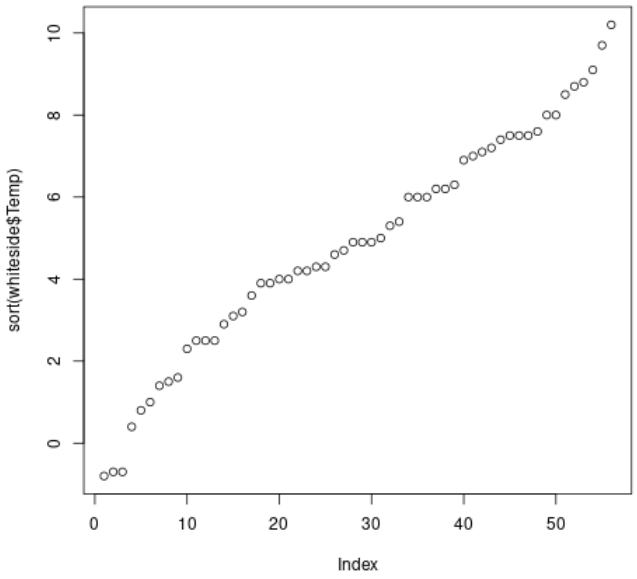
- Let's sort the plot - sorting is done with `sort`:

  ```
  sort(c(4,5,1,2,4))
  ```

  ```
  [1] 1 2 4 4 5
  ```

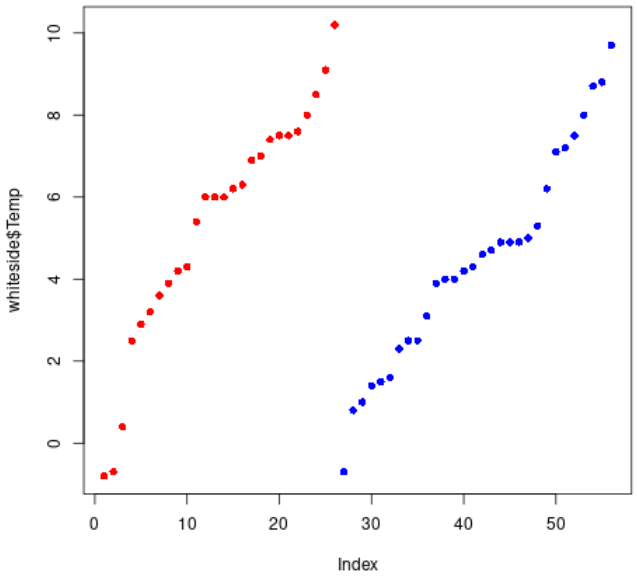
- In the plot:

  ```
  plot(sort(whiteside$Temp))
  ```



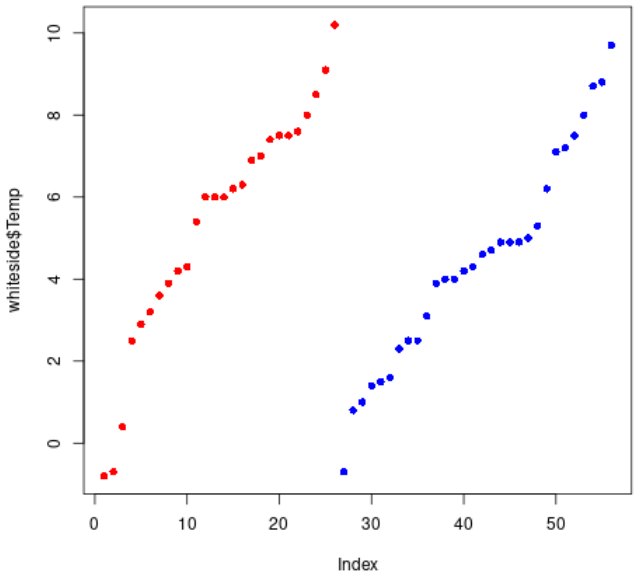
- To distinguish between the before/after values, use `color` as a third dimension:

```
colors <- ifelse(whiteside$Insul == "Before",
      "red",  # for Before
      "blue") # for After
plot(whiteside$Temp,
  col=colors,
  pch=16)
```



- There's a way to do this without the `ifelse` function, with the `unclass` function, which converts the `factor` into its numeric levels (1 for "Before", and 2 for "After"):

```
plot(whiteside$Temp,
  col=c("red","blue")[unclass(whiteside$Insul)],
  pch=16)
```



DONE BARCHARTS

When applying `plot` to a categorical variable, you get a *barchart*.

(a) Use `plot` to plot the `Insul` variable of the `whiteside` dataset only. Put the code in the code block `??` below and run it.

(b) Open and close the inline image that is generated for inspection

(c) Open and close the explanation in the notes.

The chart shows the number of measurements before and after the wall insulation of Whiteside's house, made over two consecutive heating periods.

SOLUTIONS

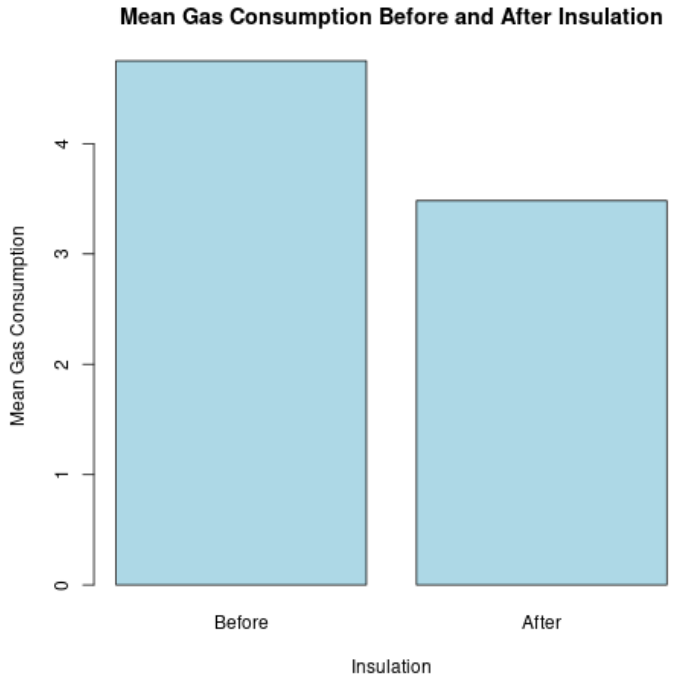
- This solution contains a few refinements such as a label for the y-axis, and a title for the graph.

```
plot(whiteside$Insul,
     main = "Measurements before and after insulation\
of house walls from the whiteside dataset.",
ylab = "Number of measurements")
```

- How about the function `barplot`, which also exists?

```
mean_consumption <- tapply(whiteside$Gas, whiteside$Insul, mean) # groups Gas

barplot(mean_consumption,
  main = "Mean Gas Consumption Before and After Insulation",
  xlab = "Insulation",
  ylab = "Mean Gas Consumption",
  col = "lightblue")
```



DONE CUSTOMIZATION

Three extensions to the scatterplots shown: changing plotting symbols, the inclusion of a legend, and linear regression reference lines.

(a) Run `??` to create a scatterplot of `Gas` vs. `Temp` from `whiteside`, with distinct point shapes (`pch`) for the `Before` and `After` data subsets.

- Open the code block with `<TAB>` to look at it
- Run the code block with `C-c C-c`
- Open / close the inline image with `<F6>` / `<F7>`
- Open the image in a separate window by putting the cursor on the link and typing `C-c C-o` (or `M-x org-open-at-point`).

```
plot(x = whiteside$Temp,
y = whiteside$Gas,
pch = c(6,16)[whiteside$Insul])
```

The last line is worth analysing:

- The factor `whiteside$Insul` has two levels
- The `pch` parameter is a vector of two elements, triangles (`6`) and solid circles (16), which are applied to the levels.
- Try to see this with a self-created example:
  (a) Create a factor with two levels
  (b) Index it using a vector

```
fac <- factor(c("male","female"))
fac
c(1,2)[fac] # index vector according to factor levels
c(2,1)[fac]
```

```
[1] male   female
Levels: female male
[1] 2 1
[1] 1 2
```

- In ??, a `legend` is added to the last scatterplot. The legend is laid on top of the plot using a vector of string values.

```
plot(x = whiteside$Temp,
y = whiteside$Gas,
pch = c(6,16)[whiteside$Insul])
legend(x = "topright",
  legend=c("Insul = Before", "Insul = After"),
  pch = c(6,16))
```

- In ??, reference lines are added to the last scatterplot. The lines are drawn with different line types (`lty`). Two linear regression models (`lm`) are defined that fit the observed data[11], and the `abline` function is used to draw the lines..

```
plot(x = whiteside$Temp,
y = whiteside$Gas,
pch = c(6,16)[whiteside$Insul])
legend(x = "topright",
  legend=c("Insul = Before", "Insul = After"),
  pch = c(6,16))
model_1 <- lm(Gas~Temp,
 data=whiteside,
 subset=which(Insul == "Before"))
model_2 <- lm(Gas~Temp,
 data=whiteside,
 subset=which(Insul == "After"))
abline(model_1, lty=2)
abline(model_2)
```

DONE TEST QUESTIONS

You now should be able to answer these test questions. You can find short answers in the footnote[12]:

[11] One could also fit a single linear regression model to the data set using the independent variables `Temp` and `Insul` as so-called *predictors*, to predict the values of the measured/observed dependent variable `Gas`.

[12] Answers: 1) Installed: R, Emacs + ESS; code block in an Org-mode file; init commands in the `~/.emacs` file. 2) `search()`. 3) `tail`. 4) `str`. 5) Only the values defined

(a) What do you need to run R code blocks inside the GNU Emacs editor?

    The R program, the ESS package.

(b) Which command lists all packages loaded in your current R session?

    ```
    search()
    ```

(c) Which command lists the last six entries of a data frame `data`?

    ```
    data |> tail(3)
    ```

(d) Which command compactly displays the structure of any R object?

    ```
    str
    ```

(e) Which values are allowed for factor variables?

    Only the factor's `levels`

(f) What is the output of the `summary` function?

    Statistical summary: minimum, maximum, median (50%), mean, 3rd (75%) and 2nd (25%) quartile.

(g) What is a generic function in R?

    A function that accepts multiple data structures and still returns a meaningful result. Check with `methods` if a function is generic.

(h) What is a boxplot?

    A plot of Tukey's five-point summary that is used to compare numeric distributions of different categorical variables.

---

by the factor levels are allowed. 6) The arithmetic mean and Tukey's five-point summary (lower/upper quartile, min/max, median). 7) A function that accepts different R objects (like a data frame) and returns different results for each. 8) A graph displaying Tukey's five-point summary for an R object, e.g. a data frame. 9) A matrix of scatterplots that shows how each variable of a dataset relates to the others. 10) Changing plotting symbols, including a legend, and drawing reference lines.

(i) What is a matrix of scatterplots?

A pair-plot - all variables are plotted against one another. Only half of the pair-plot is unique, the other diagonal is its mirror

(j) Which scatterplot customizations have you seen here?

- Change point character (`pch`)
- Add legend to plot (`legend`)
- Add reference lines (`abline`)
- Change line width and line type (`lwd`, `lty`)
- Change axis labels and title (`xlab`, `ylab`, `main`, `title`)

References

- Pearson (2018), EDA Using R, CRC Press, Chapter 1.3 (pp. 11-21).