1 Part I

1.1

R knows six different vector types, namely: logical, integer, real, complex, character (string) and raw. To give some examples for every type:

```
> # define logical object
> log <- TRUE
> is.logical(log)
```

```
[1] TRUE
```

```
> # define integer object
> int <- 1:5
> is.integer(int)
```

```
[1] TRUE
```

```
> # define real (numeric double) object
> real <- 2.5
> is.double(real)
```

```
[1] TRUE
```

```
> # define complex object
> comp <- 1+2i
> is.complex(comp)
```

```
[1] TRUE
```

```
> # define character (string) object
> char <- "a"
> is.character(char)
```

```
[1] TRUE
```

```
> # define raw object
> rawd <- as.raw(22) # corresponds to 16
> is.raw(rawd)
```

```
[1] TRUE
```

1.2

Difference between generic and numeric vector:

- An *atomic* vector contains only one single “atomic” data type in all entries. An example would be a vector which contains only integers.
- A *generic* vector (like a `list`) can contain different types of data. An example would be a vector which contains characters and numbers.

1.3

To explain: *A data frame is a list, but not evey list is a data frame.*

- A `list` is an object containing collections of objects. The types of the entries inside of the list can be different. It is for example allowed that a `list` contains a vector of real values (doubles) and a vector of characters. The length of the containing vectors can be **different**.
- A `data frame` is an object containing colletions of objects. The types of the entries inside of the list can be different. The length of the vectors have to be **the same**. The `data frame` has a matrix-like structure.

`list` and `data frame` are very similar, but the `data frame` has one more restriction (same length of all vectors). That's why a `data frame` is always a list, but a `list` is not always a `data frame`.

2 Part II

For random number generation R uses pseudo-random numbers. Starting from an initial state, called *seed state*, it will produce a deterministic sequence, which is used as random numbers. If we choose the same seed in every turn, we get the same results. To make the results of random numbers comparable, we first set the seed in a sepecific state, using `set.seed`.

```
> # set seed state to specific state
> set.seed(1)
```

After setting the seed, we define a vector with (pseudo-) random values. In this case we create $1 \cdot 10^8$ random values following normal distribution. Using the function `rnorm`, we create a distribution with mean 5 and standard deviation of 10 and saving them in a vector called `largeVector`.

```
> # define vector with normal random values
> largeVector <- rnorm(1e6, mean=5, sd=10)
```

The function `cumsum` calculates the cumulative sum of the values of the vector. It takes all elements one by one and calculates for this element the sum of all elements before, including the current element. These values will be the new elements of the new vector. Consider following example:

$$\begin{pmatrix} 1 \\ 4 \\ 3 \end{pmatrix} \xrightarrow{\texttt{cumsum}} \begin{pmatrix} 1 \\ 5 \\ 8 \end{pmatrix}$$

In the the first line of the follwing snippet, it first calculates the `cumsum` of the whole vector `largeVector`. Afterwards it just takes the first 100 elements and saves them in vector `a`. In the second line, it first takes the 100 first elements of `largeVector` and calculates the `cumsum` afterwards, which is saved in vector `b`. The second line should be much faster (see below), even if the results is the same (see also below).

```
> # get cumulative sum of the first 100 elements of largeVector
> a <- cumsum(largeVector)[1:100]
> b <- cumsum(largeVector[1:100])
```

As mentioned before, the results of vectors `a` and `b` should be the same. To check if all elements of the two vectors are exactly equal, we can use the function `identical`, where the result is `TRUE`.

```
> # check if both methods lead to exactly same result
> identical(a,b)
```

```
[1] TRUE
```

In the next step, we can compare the speed of the two ways to calculate the vectors `a` and `b`. To check the elapsed time while calculating we can use the function `system.time`, which gives us the CPU caluclation time.

```
> # get CPU calulation time of first method
> system.time(cumsum(largeVector)[1:100])
```

```
   user  system elapsed
  0.008   0.000   0.008
```

```
> # get CPU calulation time of second method
> system.time(cumsum(largeVector[1:100]))
```

```
   user  system elapsed
  0.000   0.000   0.001
```

The *user* CPU time and the *system* CPU time is a technical distinction in time running the R code and time used in operating system kernel on behalf of the R code. The interesing time is the *elapsed* time, which is the sum of the *user* time and the *system* time. We can see that the first operation of taking the `cumsum` of the whole `largeVector` with 100 million elements (and reducing the vector to 100 elements afterwards) takes a lot more CPU calucaltion time than taking the `cumsum` of the first 100 elements directly. The second method is much more efficient than the first method, because in the end we are only interested in the `cumsum` of the first 100 elements of the vector.

3 Part III

We consider dataset from "Munchner Mietspiegel 2003" which contains 13 variables about 2053 flats in Munich. In the dataset the logical variables have following encoding: 'yes' is 1 and 'no' is 0. The variables are:

- **nm**: rent in EUR
- **nmqm**: rent per m^2 in EUR
- **wfl**: living space in m^2
- **rooms**: number of rooms
- **bj**: year of construction
- **bez**: district
- **wohngut**: good residential area (yes/no)
- **wohnbest**: good residential area (yes/no)
- **ww0**: water heating (yes/no)
- **zh0**: central heating (yes/no)
- **badkach0**: tiles in bathroom (yes/no)
- **badextra**: optional extras in bathroom (yes/no)
- **kueche**: luxury kitchen (yes/no)

3.1 Data import and descriptive statistics

First we read the data into our environment using `load` function. We will have a look to the raw data using `head` and we will get some first descriptive statistic information of the interval scaled variables using `summary` function.

```
> load('miete.Rdata')
> head(miete)
```

```
      nm  nmqm wfl rooms   bj bez wohngut wohnbest ww0 zh0 badkach0 badextra
1 741.39 10.90  68     2 1918   2       1        0   0   0        0        0
2 715.82 11.01  65     2 1995   2       1        0   0   0        0        0
3 528.25  8.38  63     3 1918   2       1        0   0   0        0        0
4 553.99  8.52  65     3 1983  16       0        0   0   0        0        1
5 698.21  6.98 100     4 1995  16       1        0   0   0        0        1
6 935.65 11.55  81     4 1980  16       0        0   0   0        0        0
  kueche
1      0
2      0
3      0
4      0
5      1
6      0
```

```
> summary(miete$nm)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  77.31  390.00  534.30  570.10  700.50 1790.00 
```

```
> summary(miete$nmqm)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  1.470   6.800   8.470   8.394  10.090  20.090 
```

```
> summary(miete$wfl)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
   17.0    53.0    67.0    69.6    83.0   185.0 
```

```
> summary(miete$rooms)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  1.000   2.000   3.000   2.598   3.000   6.000 
```

```
> summary(miete$bj)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
   1918    1948    1960    1958    1973    2001 
```

We get the `min`, the `max`, the first quantile, the third quantile, the `mean` and the `median`. If we also want to get some extra information like the standard deviation and maybe skew and kurtosis, we can use the `psych` library and the containing function `describe`. In this case we include all variables.

```
> library(psych)
> describe(miete)
```

```
         vars    n    mean     sd  median trimmed    mad     min     max
nm          1 2053  570.09 245.43  534.30  547.36 223.78   77.31 1789.55
nmqm        2 2053    8.39   2.47    8.47    8.42   2.43    1.47   20.09
wfl         3 2053   69.60  25.16   67.00   67.98  22.24   17.00  185.00
rooms       4 2053    2.60   0.98    3.00    2.58   1.48    1.00    6.00
bj          5 2053 1957.98  24.88 1960.00 1958.27  17.79 1918.00 2001.00
bez*        6 2053   11.27   7.04   10.00   10.87   8.90    1.00   25.00
wohngut     7 2053    0.39   0.49    0.00    0.36   0.00    0.00    1.00
wohnbest    8 2053    0.02   0.15    0.00    0.00   0.00    0.00    1.00
ww0         9 2053    0.04   0.18    0.00    0.00   0.00    0.00    1.00
zh0        10 2053    0.09   0.28    0.00    0.00   0.00    0.00    1.00
badkach0   11 2053    0.19   0.39    0.00    0.11   0.00    0.00    1.00
badextra   12 2053    0.09   0.29    0.00    0.00   0.00    0.00    1.00
kueche     13 2053    0.07   0.26    0.00    0.00   0.00    0.00    1.00
           range  skew kurtosis   se
nm       1712.24  1.05     1.80 5.42
nmqm       18.62  0.03     0.23 0.05
```

```
wfl       168.00  0.85     1.57 0.56
rooms       5.00  0.40     0.26 0.02
bj         83.00 -0.41    -0.85 0.55
bez*       24.00  0.35    -1.02 0.16
wohngut     1.00  0.45    -1.80 0.01
wohnbest    1.00  6.53    40.60 0.00
ww0         1.00  5.05    23.52 0.00
zh0         1.00  2.97     6.82 0.01
badkach0    1.00  1.62     0.63 0.01
badextra    1.00  2.80     5.84 0.01
kueche      1.00  3.28     8.75 0.01
```

With the above results we can also do a quick validation. The `min` and `max` of the logical (yes/no) variables should be 0 and 1 respectively, which is the case. To get the amount of missing values we can calulcate the `sum` of `is.na()`.

```
> sum(is.na(miete))
```

```
[1] 0
```

There are no missing values in the whole dataset.

3.2 Identify relevant regressors and fit regression model

To idenfity relevant regressors we can apply `lm()`, which calculates a linear model, to all variables. The first argument of the function is the formular. In our case we want to do a regression of the rent in EUR (`miete$nm`) on all other variables (we can use the `.` to include all variables). In the second argument we set our dataset.

```
> regrel <- lm(miete$nm ~ ., data = miete)
```

We can omit all variables which have no significant slope. To get the slope we can have a look to the `summary` of the result of the linear regression.

```
> summary(regrel)
```

```
Call:
lm(formula = miete$nm ~ ., data = miete)

Residuals:
    Min      1Q  Median      3Q     Max
-511.18  -19.25    7.27   27.61  328.92

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -903.79835  141.15504  -6.403 1.89e-10 ***
nmqm          65.33018    0.71106  91.878  < 2e-16 ***
```

```
wfl            8.29763    0.11435  72.563  < 2e-16 ***
rooms          2.52042    2.82157   0.893  0.37182
bj             0.16281    0.07238   2.249  0.02459 *
bez2          16.40503   11.36715   1.443  0.14912
bez3          19.32578   11.70555   1.651  0.09890 .
bez4          13.71814   11.57481   1.185  0.23609
bez5          13.97651   11.53417   1.212  0.22575
bez6          19.08077   13.22698   1.443  0.14930
bez7          15.46245   13.22880   1.169  0.24260
bez8          25.69115   13.43470   1.912  0.05598 .
bez9          24.60636   11.30829   2.176  0.02967 *
bez10         16.47009   13.67074   1.205  0.22843
bez11         27.77357   13.30970   2.087  0.03704 *
bez12         19.89847   12.55420   1.585  0.11312
bez13         24.86605   12.36682   2.011  0.04449 *
bez14         26.41505   13.58424   1.945  0.05197 .
bez15         21.80571   14.57315   1.496  0.13473
bez16         24.95110   12.21704   2.042  0.04125 *
bez17         22.44807   13.25100   1.694  0.09041 .
bez18         14.62115   12.74367   1.147  0.25138
bez19         25.02291   12.25527   2.042  0.04130 *
bez20         15.82728   13.92549   1.137  0.25585
bez21         30.21527   13.55152   2.230  0.02588 *
bez22         27.76029   17.20209   1.614  0.10673
bez23         20.26190   20.57118   0.985  0.32476
bez24         29.69055   16.27294   1.825  0.06822 .
bez25         26.62587   12.09422   2.202  0.02781 *
wohngut       -3.53028    3.73040  -0.946  0.34408
wohnbest      27.20104   10.44385   2.605  0.00927 **
ww0          -45.99383    9.42126  -4.882 1.13e-06 ***
zh0           11.53773    6.44474   1.790  0.07356 .
badkach0       4.52359    3.84478   1.177  0.23951
badextra       7.25462    5.31838   1.364  0.17270
kueche        27.28826    5.84529   4.668 3.24e-06 ***
---
Signif. codes:  0 âĂŸ***âĂŹ 0.001 âĂŸ**âĂŹ 0.01 âĂŸ*âĂŹ 0.05 âĂŸ.âĂŹ 0.1 âĂŸ âĂŹ 1

Residual standard error: 65.64 on 2017 degrees of freedom
Multiple R-squared:  0.9297,	Adjusted R-squared:  0.9285
F-statistic:   762 on 35 and 2017 DF,  p-value: < 2.2e-16
```

We would suggest to include all variables which are significant on a 99% level (* or **). With the relevant variables we can fit the regression.

```
> summary(lm(miete$nm ~ miete$nmqm + miete$wfl + miete$wohnbest +
+              miete$ww0 + miete$kueche, data = miete))
```

```
Call:
lm(formula = miete$nm ~ miete$nmqm + miete$wfl + miete$wohnbest +
    miete$ww0 + miete$kueche, data = miete)

Residuals:
    Min      1Q  Median      3Q     Max
-511.22  -18.90   10.02   26.26  326.83

Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)    -554.0541     7.7059 -71.900  < 2e-16 ***
miete$nmqm       64.7417     0.6463 100.166  < 2e-16 ***
miete$wfl         8.3237     0.0600 138.723  < 2e-16 ***
miete$wohnbest   31.9400    10.0179   3.188  0.00145 **
miete$ww0       -44.2141     8.2233  -5.377 8.45e-08 ***
miete$kueche     31.1426     5.7326   5.433 6.22e-08 ***
---
Signif. codes:  0 âĂŸ***âĂŹ 0.001 âĂŸ**âĂŹ 0.01 âĂŸ*âĂŹ 0.05 âĂŸ.âĂŹ 0.1 âĂŸ âĂŹ 1

Residual standard error: 65.75 on 2047 degrees of freedom
Multiple R-squared:  0.9284,        Adjusted R-squared:  0.9282
F-statistic:  5308 on 5 and 2047 DF,  p-value: < 2.2e-16
```