# Pamatdarbību piemēri brīvpieejas programmā *R*

**Renārs Erts**,
LU Medicīnas fakultāte

**Irēna Kurcalte**,
kardioloģe, Rīgas Austrumu KUS

**Liliāna Civjāne**,
LU Medicīnas fakultāte

**Gustavs Latkovskis**,
kardiologs, LU Medicīnas fakultāte, profesors

## Īsumā

Statistiskā analīze mūsdienās ir kļuvusi par zinātniskās izpētes neatņemamu sastāvdaļu. Kvalitatīva datu apstrāde fundamentāli palielina iespēju veikt pētījumus starptautiskā līmenī un pētījuma rezultātus publicēt augsta līmeņa žurnālos. Datu analīzei ir pieejamas daudzas statistiskās analīzes programmas (piemēram, *IBM SPSS, STATA, SAS* u.c.), tomēr tās ir par maksu.

Mūsdienās arī aizvien vairāk attīstās pētniecība ar brīvpiekļuves datiem, piemēram, mājaslapā https://www.kaggle.com/ ir pieejami tūkstošiem datu failu, tai skaitā vairāki simti medicīnisku, līdz ar to, ikviens interesents var kļūt par datu pētnieku, izstrādājot jaunus veidus slimību diagnostikai utml. *Kaggle* ļauj jebkuram lietotājiem publicēt savus datu, veidot modeļus, sacensties ar citiem datu apstrādes speciālistiem labāku algoritmu izstrādē utml.

Pēdējā desmitgadē tiek daudz runāts par programmu *R*, kura spēcīgi ieņēmusi savu vietu bezmaksas statistikas programmas klāstā un nepārtraukti tiek attīstīta un pilnveidota.

*R* radīja Ross Ihaka (*Ross Ihaka*) un Roberts Gentlemans (*Robert Gentlem*) Auklendas Universitātē Jaunzēlandē, un pašlaik to turpina izstrādāt *R Development Core Team*. *R* ir nosaukta pēc pirmo divu *R* autoru vārdiem. Pats projekts tika izveidots 1992. gadā un pirmā programmas versija tika prezentēta 1995. gadā.

*R* ir brīvpieejas programma, kas nodrošina plašu statistisko analīzi (lineāro un nelineāro modelēšanu, klasiskos statistiskos testus, laikrindu analīzi, izdzīvotības analīzi utt.) un datu vizualizēšanas iespējas. Viena no papildus *R* priekšrocībām ir vienkāršība, ar kuru var izveidot sarežģītu, publikāciju kvalitātes cienīgu datu vizualizāciju, kuru izveidē lietotājam ir ļoti liela kontrole.

Ņemot vērā, ka *R* ir bezmaksas programma ar atvērtu kodu, tā izstrādi un testēšanu neveic atsevišķa kompānija ar personālu, bet paši šīs programmas lietotāji. Pāris desmitgadēs no maza izstrādātāju un entuziastu kodola šobrīd jau ir izveidojusies brīvprātīgo kopiena ar daudziem simtiem tūkstošu cilvēku, kuri dažādos veidos palīdz attīstīt *R*.

*R* piemīt reputācija, ka daudzas *lietas* var tikt izdarītas ar ļoti maza rakstāmā koda palīdzību. Un, ja ir sajūta, ka tūlīt ir jāapgūst *"Hello world"* sveiciens, ar ko parasti sākas gandrīz ikvienas programmēšanas valodas apguve, tad *R* programmā tā nenotiek. Atšķirībā no standarta programmēšanas valodām (*C, Pascal* utt), *R* programmā visas komandrindā ierakstītās komandas tiek momentā izpildītas, tāpēc rezultāts ir redzams uzreiz.

Plašs uzskats ir, ka *R* ir statistikas analīzes programma, kurā nepieciešamās izpildes komandas un funkcijas ir rakstāmas komandrindā, un tas savā ziņā ir taisnība, tomēr precīzāk programma *R* būtu definējama kā vide (*environment*), kurā tiek īstenotas statistikas un datu vizualizācijas metodes.

*R* programmas spēcīgās puses ir: ļoti daudz izmantojamo funkciju, jaunas statistiskās metodes tiek ātri ieviestas, liels *R* lietotāju kopienas atbalsts, daudzkārt atkārtojamu skritpu izmantošana, pieejami daudzi mācību materiāli utml. Savukārt *R* programmas vājās puses ir, piemēram, lietojamo komandu sintakses sarežģītība un standarta neesamība, daudzo izveidoto pakotņu stabilitāte un kvalitāte utml. Lielākoties *R* programmas ārkārtīgi daudzās funkcijas, ja vien tās neizmanto katru dienu, ir sarežģīti atcerēties no galvas, un lietotāja rokasgrāmatai vienmēr ir jābūt pa rokai.

Programmas *R* iespējas tiek nepārtraukti paplašinātas, izmantojot daudzu lietotāju radītas pakotnes (*packages*) – šobrīd jau ir izveidotas vairāk nekā 15000 pakotnes (*Rcmdr, knitr, ggplot, vioplot* u.c.), kas ļauj izmantot specializētas statistiskās tehnikas, datu vizualizācijas iespējas, importa/eksporta iespējas, atskaišu veidošanas rīkus u.c., un šie ir faktori, kāpēc *R* programmai ir tik daudz piekritēju.

*R* programmas izaugsmi samazina tas, ka trūkst lietotājam draudzīga grafiskā saskarne, kura būtu ar lielām datu analīzes, atlases, transformācijas un vizualizācijas iespējām. Var izmantot programmas *R* pakotni – *R commander* (*Rcmdr*), tomēr tā iespējas un lietošanas draudzīgums nav ne tuvu līdzvērtīgs *IBM SPSS* programmai. Protams "īstie speciālisti" visas darbības veic tikai komandrindā un uz citu datu apstrādes programmu lietotājiem mēdz skatīties "no augšas".

Neskatoties uz programmas *R* lietotāju apgalvojumiem par *R* eksponenciālo pielietojuma izaugsmi gan zinātnē, gan akadēmiskajā vidē, tomēr ir jāsaprot, ka dažādu statistikas apstrādes programmu dažādība cilvēces attīstībā tikai palīdz un nav vērts veidot svēto karu (*holy war*) starp dažādu statistikas programmu lietotājiem un pasniedzējiem.

Tas, kur *R* izmantošana ir pilnīgi nepieciešama, ir darbs ar lieliem datu apjomiem, kā arī sarežģītas struktūras un dažādos datu formātos esošos datiem, kā arī mašīnapmācībā un neironu tīklu risinājumu izstrādei.

Ir arī veikti pētījumi par *IBM SPSS* un *R* programmas izmantošanu statistikas studiju kursa apguvē, piemēram: *Jacob B. Rode, Megan M. Ringel. Statistical Software Output in the Classroom: A Comparison of R and SPSS. Teaching of Psychology*, 2019. (https://doi.org/10.1177/0098628319872605), 2019. gadā pētīja vai programmas *R* vai *IBM SPSS* apgūšana psiho-

loģijas studentiem rada lielāku sākotnējo trauksmi un vai nemiers mainās pēc studiju kursa apgūšanas. Katrs pētījumā iesaistītais pasniedzējs mācīja ievadstatistikas studiju kursu, pirmo grupu (N = 43) mācot tikai *R* programmu, bet otro grupu (N = 39) mācīja tikai *IBM SPSS* programmā. Abu grupu studenti tika aptaujāti, lai novērtētu viņu bažas un pārliecību par *R* vai *IBM SPSS* apgūšanu pirmajā un pēdējā studiju kursa dienā. Sākotnēji, uzsākot apmācību *R* programmā, studenti ziņoja par lielāku satraukumu un mazāku pārliecību, salīdzinot ar *IBM SPSS* apguves studiju kursa studentiem. Tomēr sākotnējā atšķirība studiju kursu beigās starp abām grupām mazinājās. Kopumā pētījuma autori secināja, ka, lai gan *R* programmas sākotnējā izmantošanai var šķist biedējošāka, tomēr studenti tam pielāgojas gandrīz tikpat labi kā *IBM SPSS* programmas lietotāji.

Papildus lasīšanai par programmu *R* ir pieejams žurnāls *R Journal* (https://journal.r-project.org/), kurā kopš 2009. gada tiek publicētas tiešsaistes brīvās piekļuves publikācijas. Žurnāls ir pilnīgi bezmaksas: tas neiekasē ne no autoriem maksu par publicēšanu, ne arī maksu par abonēšanu. Žurnāls publicē rakstus, kuros ir aprakstīti jauninājumi pašā *R* programmā, jaunas *R* programmatūras bibliotēkas un statistisko aprēķinu metodes, kas ieviestas programmā *R*.

Vai programmas *R* apgūšana ir vienkārša? Tas ir sarežģīti atbildāms jautājums. Daudzi pētnieki datu analīzi sāk apgūt, izmantojot *R*. Viss, kas apgūšanai ir nepieciešams ir laiks, vīzija, dati un skaidrs nodoms, kas ir jāizdara, un tad programmas apgūšana norisinās vienkārši, mēģinot un darot. Nav nepieciešams uztraukties par programmēšanas valodas zināšanām. Sintakse, komandas un darbības, ko izmanto *R*, diezgan ievērojami atšķiras no citām programmēšanas valodām un ir izstrādāta tieši datu analīzes programmēšanas vajadzībām.

Pārsvarā gandrīz jebkuri mācību materiāli par programmu *R* sākas ar informāciju par vektoriem, matricām, operatoriem, cikliem utt., kas neapšaubāmi ir svarīgi un noderīgi darbā ar lieliem un sarežģītiem datu apjomiem, tomēr šāda informācija ne visiem uzreiz ir nepieciešama un daudzi pasniedzēji un literatūras rakstītāji nespēj saprast, ka bez specifiskām zināšanām augstākajā matemātikā un programmēšanā, daudzas *R* programmas pamatlīmeņa darbības nav tūlīt saprotamas, kā arī sākumā var nobiedēt un atgrūst no šīs programmas apgūšanas.

Pēc šī raksta autoru domām, programmas *R* apguve ir jāuzsāk ar reālu, nelielu datu analīzi un vizualizāciju, un turpmākās advancētās iespējas var apgūt vēlāk – soli pa solim. Tāpēc šajā rakstā ir mēģināts vienkāršā veidā un vienkāršos vārdos parādīt ieskatu programmas R izmantošanu datu pamata ievadei, kodēšanai un pavisam vienkāršai aprakstošās statistikas rādītāju aprēķinu veikšanai. Turpmākos soļus, lietotājs, kurš gribēs iedziļināties programmas piedāvātajās, šķiet, bezgalīgajās iespējās, varēs apgūt pats.

Iespējams, ka Latvijas apstākļos statistikas metožu pamatu apguvi matemātiski mazāk orientētiem studentiem jēgpilnāk un laika ziņā ekonomiskāk, kā arī šobrīd visnotaļ interesantāk, ir veikt kādā no komerciāli pieejamām programmām. Medicīnas studentam pēc studiju beigšanas būtu nepieciešams prast apstrādāt datus gan kādā komerciāli pieejamā datu apstrādes programmā, gan arī turpināt apgūt programmas *R* iespējas datu analīzē un vizualizācijā.

## Programmas *R* instalācija

Programmas mājas lapa ir atrodama šeit: https://www.r-project.org, kur to var ielādēt gan *Microsoft Windows* (*x86* un *x64*), gan *Linux*, gan *macOS* platformām. Šajā rakstā tiek izmantota programma *R* ar versiju 3.6.2. (2019-12-12.). Pēc programmas instalācijas un palaišanas atveras logs (*skat. 1. attēlu*).

*RGUI* ir standarta grafiskais interfeiss, kas ir kopā ar *R*. Šeit atrodama komandrinda, ko sauc par konsoli, tā darbojas pēc *jautājuma – atbildes* principa.

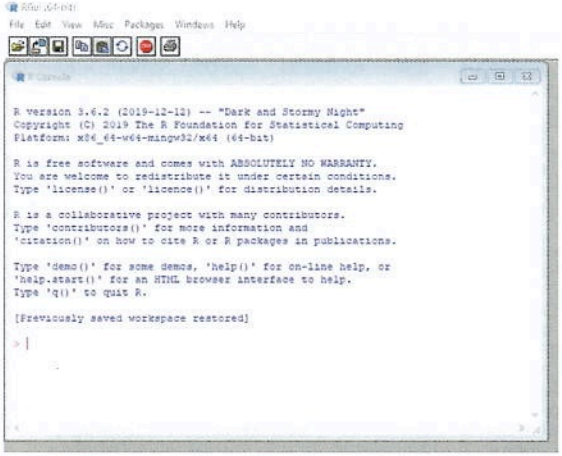
Daudzi lietotāji izmanto programmu *R* tikai ar komandrindu, kurā izpilda komandas, ar kuru palīdzību veic datu ievadi, analīzi un vizualizāciju. Tomēr komandrindas izmantošana vien ir diezgan garlaicīga. Par laimi, *R* ir pieejamas papildus vairākas grafiskas lietotāja saskarnes (*Graphical user interface* jeb *GUI*), kas nodrošina produktīvāku darbu. Tāpēc praktiskā ziņā vēlams datorā ielādēt un ieinstalēt, piemēram, daudzu lietotāju izmantoto *RStudio*.

## Programma *RStudio*

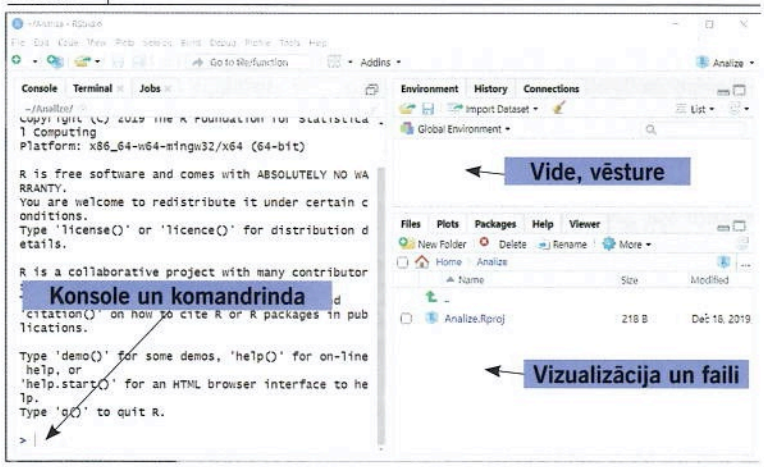
Lai ieinstalētu programmu *Rstudio*, nepieciešams apmeklēt mājas lapu: https://rstudio.com/ un ielādēt savam datoram atbilstošu programmas versiju. Jāpiezīmē, ka programma *RStudio* nestrādā, ja pirms tam nav instalēta programma *R*.

*RStudio* ir programma, kas darbojas uz *R* bāzes un ļauj daudz lielāku lietotāja darbošanās rīcības brīvību. Atšķirībā no *RGUI*,

1. attēls




2. attēls



šai videi ir papildus moduļi (piemēram, komandu vēsture utt.). *RStudio* piemīt ērtāks interfeiss, kas vienkāršo darbu ar *R*. Šeit ir pieejami dažādi rīki, kas paredzēti, piemēram, izpildāmās sintakses iekrāsošanai, automātiskai koda pabeigšanai, ērtai navigācijai pa skriptu, interaktīvai atkļūdošanai, kas ļauj ātri diagnosticēt un labot kļūdas, kā arī viss nepieciešamais projektu pārvaldībai – dažādu versiju uzskaitei utml.

Pēc programmas uzstādīšanas un palaišanas, atveras logs (*skat. 2. attēlu*).

## Piemērā izmantotie dati

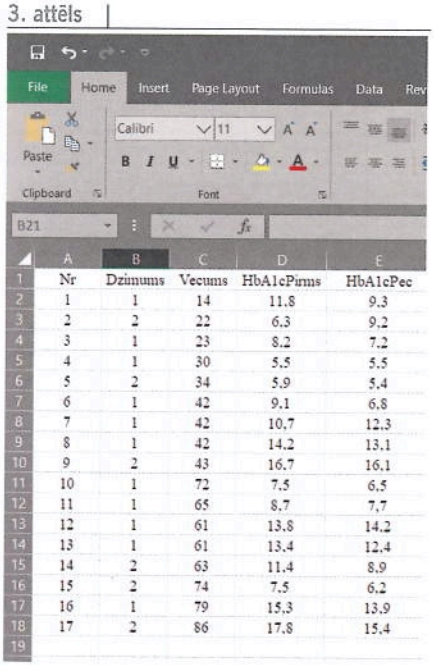
Šajā piemērā tiek izmantots *Microsoft Excel* (*xlsx*) fails, kurā ir kolonas ar pazīmēm (*skat. 3. attēlu*):

- numurs pēc kārtas (*Nr*);
- dzimums, 0 – Sieviete, 1 – Vīrietis (*Dzimums*);
- vecums, gados (*Vecums*);
- glikētais hemoglobīns pirms apmācības (*HbA1cPirms*);
- glikētais hemoglobīns pēc apmācības (*HbA1cPec*).

## Datu ievade

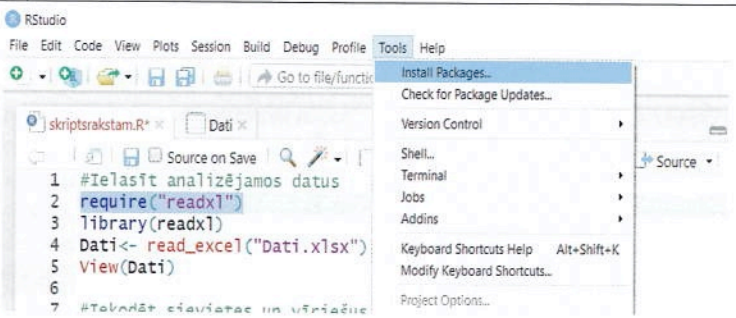
*R* komandas parasti sastāv no divām daļām: objektiem un funkcijām, tos atdala ar operatoru "<–", kas sastāv no divām rakstzīmēm "<" (*mazāk nekā*) un "–" (*mīnus*), kas atrodas strikti viens otram blakus un norāda uz objektu, kas saņem

3. attēls



| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | Nr | Dzimums | Vecums | HbA1cPirms | HbA1cPec |
| 2 | 1 | 1 | 14 | 11.8 | 9.3 |
| 3 | 2 | 2 | 22 | 6.3 | 9.2 |
| 4 | 3 | 1 | 23 | 8.2 | 7.2 |
| 5 | 4 | 1 | 30 | 5.5 | 5.5 |
| 6 | 5 | 2 | 34 | 5.9 | 5.4 |
| 7 | 6 | 1 | 42 | 9.1 | 6.8 |
| 8 | 7 | 1 | 42 | 10.7 | 12.3 |
| 9 | 8 | 1 | 42 | 14.2 | 13.1 |
| 10 | 9 | 2 | 43 | 16.7 | 16.1 |
| 11 | 10 | 1 | 72 | 7.5 | 6.5 |
| 12 | 11 | 1 | 65 | 8.7 | 7.7 |
| 13 | 12 | 1 | 61 | 13.8 | 14.2 |
| 14 | 13 | 1 | 61 | 13.4 | 12.4 |
| 15 | 14 | 2 | 63 | 11.4 | 8.9 |
| 16 | 15 | 2 | 74 | 7.5 | 6.2 |
| 17 | 16 | 1 | 79 | 15.3 | 13.9 |
| 18 | 17 | 2 | 86 | 17.8 | 15.4 |
| 19 | | | | | |

4. attēls




izteiksmes vērtību. To varētu saprast nozīmē "ir radīts no". Taustiņu kombinācija ātrai operatora "<–" uzrakstīšanai komandrindā ir uz datora klaviatūras nospiest Alt un mīnus zīmi. Lielākajā daļā darbību piešķiršanas zīmi "–" var izmantot kā alternatīvu. Šajā rakstā, uzskatāmības dēļ, ir izmantoti abi varianti.

Jāpiebilst, ka izdarot piešķiršanas operāciju, rezultāts bieži vien netiek parādīts. Lai rezultāts tiktu attēlots, izpildāmo komandu var ievietot iekavās. To var nodemonstrēt šajā mazā piemērā, pirmajā variantā, piešķirot mainīgajam *x* vērtību 5, tā vienkārši ievietojot atmiņā, otrā variantā ievietojot izpildāmo izteiksmi iekavās, pēc tās izpildes, rezultāts tiek attēlots uz ekrāna:

```
> x <- 5
>  (x  <- 5)
 [1] 5
> |
```

Programmā *R* nav speciāli iebūvētu funkciju *Microsoft Excel* failu importēšanai, tāpēc viens variants ir izmantot readxl bibliotēku.

Lai ievadītu datus programmā, komandrindā ir nepieciešams uzrakstīt šādas komandas:

```
require("readxl")
library(readxl)
Dati <- read_excel(file.
choose())
View(Dati)
```

Pirmajā rindā tiek norādīts, ka ir nepieciešama bibliotēka ar nosaukumu readxl, un ar komandu require() programmai tiks pieprasīta informācija par šādas bibliotēkas pieejamību un, ja tā nav uzinstalēta uz datora, tad tas tiks veikts automātiski. Vienkāršāk ir to izdarīt programmā *RStudio Tools – Install Packages* (*skat. 4. attēlu*).

Tālāk atvērsies logs (*skat. 5. attēlu*), kurā ieraksta nepieciešamās bibliotēkas nosaukumu

Otrajā rindā tiek definēts, ka datora atmiņā tiks ielādēta bibliotēka ar nosaukumu readxl, kura atbild par darbu ar *Microsoft Excel* failiem, jo bibliotēkas instalācija tiek veikta tikai vienreiz. Tomēr katru reizi, uzsākot darbu ar programmu *R*, nepieciešamās bibliotēkas ir jāielādē atmiņā turpmākajam darbam un to veic ar komandu library(). Šajā rakstā tiek izmantota tikai viena papildus bibliotēka readxl un pārējās darbībās ir mēģināts iztikt ar programmā *R* iebūvētajām standarta funkcijām. Reālā datu analīzē tiek izmantotas daudzas papildus bibliotēkas, piemēram, *tidyr, haven, ggplot2* u.c., kuras atbild par gan par *IBM SPSS* datu failu ielādi, gan datu transformācijām, gan advancētu grafiku veidošanai.

Trešajā rindā objektam ar nosaukumu *Dati* tiek piekārtota un izpildīta funkcija, kura atver logu, kurā ir jāizvēlas nepieciešamais fails, mūsu gadījumā ar nosaukumu *Dati.xlsx*.

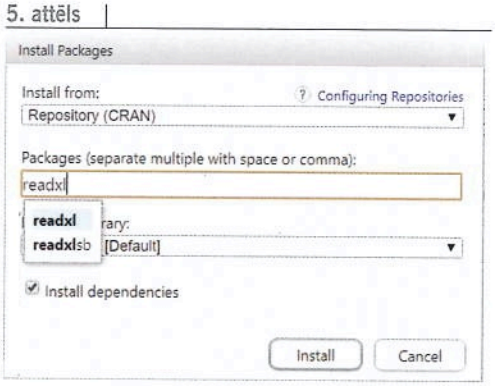
Ceturtajā rindā tiek izpildīta darbība – parādīt datus.

***Piezīme:*** ar bibliotēkas readxls palīdzību ir iespējams programmā *R* importēt gan *.xls*, gan *xlsx* datu failus. Papildus šīs bibliotēkas iespējas ir iespējams ierakstot *R* komandrindā komandu ?readxl, vai arī apmeklējot mājas lapu: https://readxl.tidyverse.org/

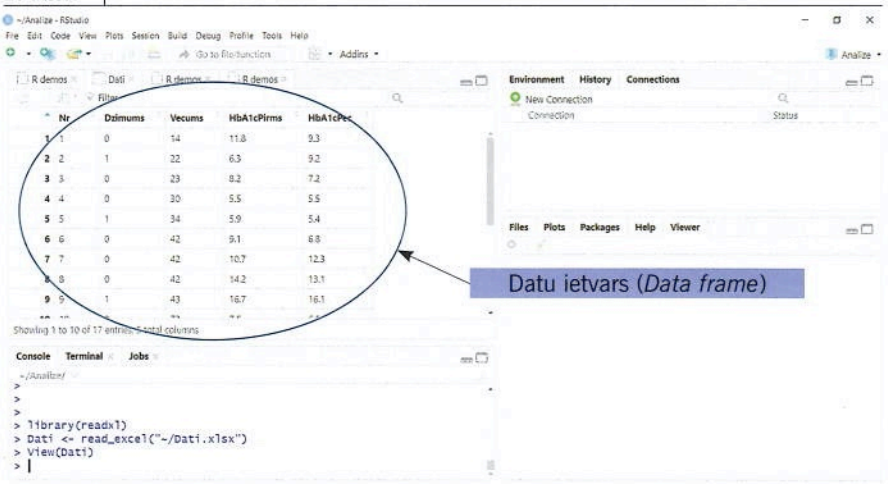
Pēc šo komandu izpildes programma *RStudio* izskatīsies kā parādīts 6. attēlā.

Pēc datu ielasīšanas no faila tiek izveidots datu ietvars (angl., *Data frame*) ar

5. attēls



6. attēls




nosaukumu *Dati*. Tīri pēc definīcijas datu ievaru var uzskatīt par vektoru kopumu, kurā visi vektori jeb pētāmās pazīmes (kolonas) ir ar vienādu garumu.

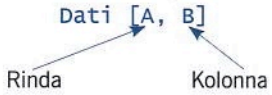
Datu ietvaru *RStudio* logā var apskatīt ar funkciju View(). Mūsu gadījumā izpildot View(Dati) tiks atvērta tabula ar datiem (skat. 7. attēlu).

***Piezīme:*** bieži vien darbojoties ar datu ietvaru – izpildot dažādas pārveides komandas, ietvars uzreiz netiek atjaunināts, tāpēc pēc atsevišķu funkciju izpildes ir nepieciešams atjaunināt datu ietvara skatu ar komandu View().

Visas augstāk rakstītās darbības var aizvietot manuāli *RStudio*, izvēloties no failu loga nepieciešamo failu un nospiežot *Import Dataset*, kā rezultātā tiks atvērts logs (*skat. 8. attēlu*).

## Pamatdarbības

Datu ievars sastāv no rindām un kolonām, uz kurām var veidot atsauces formā: *Dati[A, B]*, kur *A* norāda uz rindu, bet *B* uz kolonu.




Lai piekļūtu un apskatītu datu ietvara vektoru (kolonas jeb pazīmes vērtības), var izmantot vai nu [, [[vai $ operatoru).

Piekļūšana pazīmei ar [[vai $ ir līdzīga. Visbiežāk izmanto $ zīmi, piemēram, Dati$Vecums. Tomēr jāpiezīmē, ka piekļūšana datiem, izmantojot [, atgriezīs vērtības datu ietvara veidā, bet pārējie divi variantu atgriezīs datus vektora formā. To var novērot šajā piemērā:

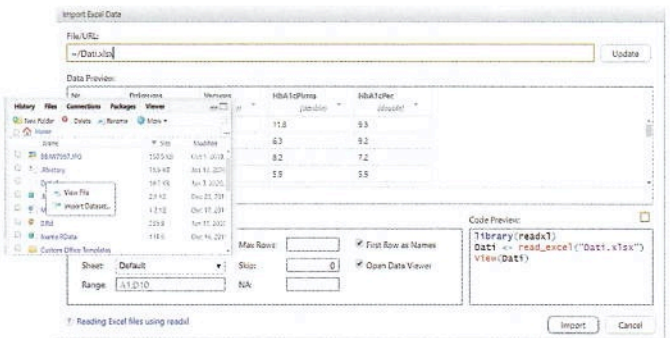
```
> Dati [[3]]
[1]14 22 23 30 34 42 42 42 43 72  65 61 61 63 74 79 86
> Dati[["vecums"]]
[1]14 22 23 30 34 42 42 42 43 72  65 61 61 63 74 79 86
> Dati$Vecums
[1]14 22 23 30 34 42 42 42 43 72 65 61 61 63 74 79 86
> Dati[3]
# A tibble: 17 x 1
   vecums
   <dbl>
 1  14
 2  22
 3  23
 4  30
 5  34
 6  42
 7  42
 8  42
 9  43
```

Jaunu datu pievienošanu var īstenot vairākos veidos, viens variants ir no komandrindas ar funkciju: Dati ["Jaunakolona"] <- scan (nlines=nrow(Dati))

7. attēls

| | Nr | Dzimums | Vecums | HbA1cPirms | HbA1cPec |
|---|---|---|---|---|---|
| 1 | 1 | 1 | 14 | 11.8 | 9.3 |
| 2 | 2 | 2 | 22 | 6.3 | 9.2 |
| 3 | 3 | 1 | 23 | 8.2 | 7.2 |
| 4 | 4 | 1 | 30 | 5.5 | 5.5 |
| 5 | 5 | 2 | 34 | 5.9 | 5.4 |
| 6 | 6 | 1 | 42 | 9.1 | 6.8 |
| 7 | 7 | 1 | 42 | 10.7 | 12.3 |
| 8 | 8 | 1 | 42 | 14.2 | 13.1 |

8. attēls



9. attēls

Data Editor

File Edit Help

| | Nr | Dzimums | Vecums | HbA1cPirms | HbA1cPec |
|---|---|---|---|---|---|
| 1 | 1 | 1 | 14 | 11.8 | 9.3 |
| 2 | 2 | 2 | 22 | 6.3 | 9.2 |
| 3 | 3 | 1 | 23 | 8.2 | 7.2 |
| 4 | 4 | 1 | 30 | 5.5 | 5.5 |
| 5 | 5 | 2 | 34 | 5.9 | 5.4 |
| 6 | 6 | 1 | 42 | 9.1 | 6.8 |
| 7 | 7 | 1 | 42 | 10.7 | 12.3 |
| 8 | 8 | 1 | 42 | 14.2 | 13.1 |
| 9 | 9 | 2 | 43 | 16.7 | 16.1 |
| 10 | 10 | 1 | 72 | 7.5 | 6.5 |
| 11 | 11 | 1 | 65 | 8.7 | 7.7 |
| 12 | 12 | 1 | 61 | 13.8 | 14.2 |
| 13 | 13 | 1 | 61 | 13.4 | 12.4 |
| 14 | 14 | 2 | 63 | 11.4 | 8.9 |
| 15 | 15 | 2 | 74 | 7.5 | 6.2 |
| 16 | 16 | 1 | 79 | 15.3 | 13.9 |
| 17 | 17 | 2 | 86 | 17.8 | 15.4 |
| 18 | | | | | |
| 19 | | | | | |

ievadīt skaitļus, cits variants ir izmantot programmā *R* iebūvēto pavisam vienkāršu datu redaktoru un izmantojot komandu edit() ir iespējams to atvērt,

```
> Dati <- edit(Dati)
>|
```

kurā datus var labot, dzēst un pievienot jaunus (*skat. 9. attēlu*).

***Piezīme:*** Ja ir vēlme izveidot jaunu un tukšu izklājlapu datu ievadei, var izmanto šo komandu JauniDati <- edit(data.frame()), kuras rezultātā tiks izveidots jauns datu ietvars ar nosaukumu *JauniDati*, kurā pašrocīgi var ierakstīt savus datus.

## Pamatdarbības ar datiem un datu kodēšana

Ja ir nepieciešamība aprēķināt starpību starp glikēto hemoglobīnu pirms apmācības un pēc, tad to var darīt šādi:

```
> Dati["HbA1cStarpiba"]<-Dati$HbA1cPirms-Dati$HbA1cPec
> |
```

Kā rezultātā tiks izveidota jauna kolona jeb pazīme, jeb vektors ar nosaukumu *HbA1cStarpiba* ar veikto aprēķinu rezultāta vērtībām (*skat. 10. attēlu*).

Ņemot vērā, ka pazīme *Dzimums* ir kategoriska un ir kodēta kā 1 – *sieviete* un 2 – *vīrietis*, tad varam šo pazīmi pārveidot kā faktoru (*factor*) ar funkcijas factor() palīdzību, kurā ar papildus nosacījumu ordered=FALSE tiek norādīts, ka pazīme *Dzimums* ir kategoriska jeb nomināla (viens līmenis kādā veidā nav labāks vai sliktāks kā cits):

```
> Dati$Dzimums=factor(Dati$Dzimums,labels=c("Sieviete", "Vīrietis"), ordered = FALSE)
> |
```

Pēc kā pazīme *Dzimums* no skaitļiem 1 un 2 tiks pārveidota par faktoru un *RStudio* datu logs izskatīsies kā redzams 11. att.

***Piezīme:*** funkcija factor() nav tik vienkārša kā izskatās. Ja mēģinātu pavisam vienkārši paskaidrot, tad tas ko tā dara ir – pārbauda visas analizējamās pazīmes jeb vektora vērtības, un atrod unikālās (ja vērtības ir burti vai vārdi, tad sakārto tos pēc alfabēta, ja cipari, tad augošā secībā), un tālāk ar vektora vērtībām labels=c() secīgi (!) piekārto t.s., etiķeti (angl., *label*) katrai analizējamā vektora unikālajai vērtībai.

Mūsu gadījumā funkcija factor() atrod, ka kolonā *Dzimums* ir divas vērtības, atbilstoši – 1 un 2, tālāk sekojoši funkcija no vektora labels=c("Sieviete", "Vīrietis") piekārto mazākai skaitļa vērtībai, mūsu gadījumā skaitlim 1 etiķeti *sieviete* (jo izveidotajā vektorā ar nosaukumu labels *sieviete* ir norādīta pirmā) un skaitlim 2 atbilstoši piekārto etiķeti *vīrietis*.

***Paskaidrojums:*** programmā *R* faktors ir datu veids, kurā tiek uzglabātas dažādas vērtības un kuru visbiežāk izmanto kategoriskās pazīmes definēšanai. Viena no visbiežāk izmantotām īpašībām faktoram ir statistiskajā modelēšanā, jo kategoriskā pazīme tai pašā regresijas analīzē tiek savādāk pielietota nekā kvantitatīvā. Līdz ar to, analizējamās pazīmes precīza nodefinēšana nodrošina arī korektu modelēšanas rezultāta aprēķinu.

Faktori atmiņā tiek uzglabāti kā vektors ar pozitīvu skaitļu vērtībām un atbilstošām simbolu virknēm. Faktora izveidošanai izmanto funkciju factor(), vienīgais nepieciešamais funkcijas arguments ir pazīme jeb vektors, kuru vēlamies pārveidot par faktoru. Par faktoru var pārveidot gan cipariskas vērtības, gan mainīgos lielumus simbolu virknēs. Lai mainītu faktora līmeņu secību, no to noklusējuma sakārtotajā secībā, funkcijā factors() tiek izmantots arguments levels(), kurā var ierakstīt visas iespējamās vērtības sev vēlamajā secībā.

Lai pārbaudītu, vai kāda pazīme tiešām ir definēta kā faktors, var izmantot funkciju is.factor():

```
> is.factor(Dati$Dzimums)
[1] TRUE
> |
```

Lai noteiktu faktoriālās pazīmes līmeņu skaitu, var izmantot komandu nlevels():

```
> nlevels(Dati$Dzimums)
[1] 2
> |
```

Lai pārbaudītu kā faktoriālā pazīme ir kodēta, var izmantot funkciju levels(), kura sniegs atbildi rindas veidā un kreisās puses vērtība atbildīs mazākajam skaitlim (šajā piemērā ir saprotams, ka ar 1 ir apzīmēta *sieviete* un ar 2 – *vīrietis*):

```
> levels(Dati$Dzimums)
[1] "Sieviete" "Vīrietis"
> |
```

***Piezīme:*** ja būtu nepieciešams mainīt (pārsaukt) faktoriālās pazīmes līmeņu nosaukumus, piemēram, vēlētos pārsaukt

10. attēls |

Filter

| | Nr | Dzimums | Vecums | HbA1cPirms | HbA1cPec | HbA1cStarpiba |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 14 | 11.8 | 9.3 | 2.5 |
| 2 | 2 | 2 | 22 | 6.3 | 9.2 | -2.9 |
| 3 | 3 | 1 | 23 | 8.2 | 7.2 | 1.0 |
| 4 | 4 | 1 | 30 | 5.5 | 5.5 | 0.0 |
| 5 | 5 | 2 | 34 | 5.9 | 5.4 | 0.5 |
| 6 | 6 | 1 | 42 | 9.1 | 6.8 | 2.3 |
| 7 | 7 | 1 | 42 | 10.7 | 12.3 | -1.6 |
| 8 | 8 | 1 | 42 | 14.2 | 13.1 | 1.1 |

11. attēls |

Filter

| | Nr | Dzimums | Vecums | HbA1cPirms | HbA1cPec | HbA1cStarpiba |
|---|---|---|---|---|---|---|
| 1 | 1 | Sieviete | 14 | 11.8 | 9.3 | 2.5 |
| 2 | 2 | Vīrietis | 22 | 6.3 | 9.2 | -2.9 |
| 3 | 3 | Sieviete | 23 | 8.2 | 7.2 | 1.0 |
| 4 | 4 | Sieviete | 30 | 5.5 | 5.5 | 0.0 |
| 5 | 5 | Vīrietis | 34 | 5.9 | 5.4 | 0.5 |
| 6 | 6 | Sieviete | 42 | 9.1 | 6.8 | 2.3 |
| 7 | 7 | Sieviete | 42 | 10.7 | 12.3 | -1.6 |
| 8 | 8 | Sieviete | 42 | 14.2 | 13.1 | 1.1 |
| 9 | 9 | Vīrietis | 43 | 16.7 | 16.1 | 0.6 |
| 10 | 10 | Sieviete | 72 | 7.5 | 6.5 | 1.0 |

"Sieviete" uz "Siev." un "Vīrietis" uz "Vīr.", tad to varētu darīt šādi:

```
> levels(Dati$Dzimums)<-c("Siev.", "Vīr .")
> View(Dati)
> |
```

***Piezīme:*** nomainīt nosaukumus varētu arī, iespējams, vieglāk atceramā veidā:

```
> levels(Dati$Dzimums)[1]<-"siev."
> levels(Dati$Dzimums)[2]<-"Vīr."
> |
```

Citreiz datu vizualizācijai vai loģistiskās regresijas aprēķiniem ir nepieciešamība precīzi definēt references faktoru jeb pamatlīmeni, balstoties uz kuru, tiks veikti turpmākie aprēķini. Un šim nolūkam nākas faktoriālās pazīmes kodējumu pārveidot citā secībā. Mūsu gadījumā, ja vēlētos pārveidot pazīmes kodējumu, lai, piemēram, *vīrietis* būtu ar vērtību 1 un *sieviete* ar vērtību 2, bet, protams, lai pati datu struktūra nemainītos, to var veikt šādi:

```
> levels(Dati$Dzimums)
[1] "Sieviete" "Vīrietis"
> Dati$Dzimums<-factor(Dati$Dzimums, levels<-c ("Vīrietis", "Sieviete"), ordered=FALSE)
> levels(Dati$Dzimums)
[1]"vīrietis" "sieviete"
> |
```

***Piezīme:*** tātad šajā gadījumā atkal tiek izsaukta funkcija factor(), kura skanē pazīmi *Dzimums*, atrod vērtības 1 un 2, un ar parametra levels=c("Vīrietis", "Sieviete") palīdzību samaina vietām vērtības, nemainot pašu datu struktūru (pie kam šis variants nenozīmē vienkāršu faktora etiķetes pārsaukšanu). Jāpiebilst, ka šajā gadījumā īpaši svarīgi ir nekļūdīties faktoriālās pazīmes kodējuma nosaukumā, piemēram, šī komanda nedarbosies:

```
> (Dati$Dzimums <- factor Dati$Dzimums , levels<-c"Vīr.", "Siev."), ordered=FALSE))
[1] <NA> <NA> <NA> <NA> <NA> <NA> <NA>
[8]<NA> <NA> <NA> <NA> <NA> <NA> <NA>
[15] <NA> <NA> <NA>
Levels: vīr. siev.
> |
```

***Piezīme:*** to pašu būtu iespējams veikt arī ar funkciju relevel(), kurā ar papildus parametru ref= tiek norādīts, kurš faktors tiek ņemts kā pamatlīmenis:

```
> Dati$Dzimums <-relevel (Dati$Dzimums, ref= "Vīrietis")
> levels(Dati$Dzimums)
[1]"Vīrietis" "Sieviete"
> |
```

***Piezīme:*** funkcija relevel() darbojas tikai ar nominālām pazīmēm un, saprotamu iemeslu dēļ, darbā ar rangu datiem dod kļūdas paziņojumu.

Programmā *R* nav tik vienkārši uzreiz iegūt informāciju kā ir kodēta katra faktoriālā pazīme. *IBM SPSS* programmā šis jautājums ir daudz vienkāršāk risināms. Viens variants ir izveidot atsevišķu funkciju, ar kuras palīdzību var apskatīt datu kodējumu. Šajā gadījumā var izveidot funkciju ar nosaukumu, piemēram, DatuKodejumalegusana(), kura var sniegt atbildi:

```
> DatuKodejumaIegusana <- function(x){+ unique(data.frame (Pazīme = as.character(x), Kodējums = as.numeric(x)))}
> DatuKodejumaIegusana(Dati$Dzimums)
   Pazīme Kodējums
1 Sieviete 1
2 Vīrietis 2
> |
```

Lai faktoriālo pazīmi pārvērstu atpakaļ cipariskā veidā, var izmantot funkciju unclass() (ar šo darbību ir jābūt īpaši uzmanīgam(!)):

```
> Dati$Dzimums <- unclass Dati$Dzimums)
> Dati$Vecgrupa <- unclass(Dati$Vecgrupa)
> view (Dati)
> |
```

Kā rezultātā iegūs (*skat. 12. attēlu*).

Lai labāk izprastu, kādā veidā *R* ņems vērā kategoriskos mainīgos, var izmantot funkciju contrast(), kura parādīs, kā mainīgos programma *R* interpretēs regresijas modeļos:

```
> contrasts(Dati$Dzimums)
          Vīrietis
Sieviete        0
Vīrietis        1
> |
```

Lai pārbaudītu, cik un vai pašā kolonā jeb pazīmē *vecums* ir iztrūkstoša (*missing*) vērtība, izmantojam funkciju is.na(), kura sniegs FALSE/TRUE par katru pazīmes vērtību, kā arī funkcija sum() sasummēs iztrūkstošo vērtību skaitu un sniegs atbildi:

```
> is.na(Dati$Vecums)
 [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[11] FALSE FALSE FALSE FALSE FALSE FALSE FALSE
> sum(is.na Dati$Vecums))
[1] 0
> |
```

Lai pārbaudītu, vai visā datu ietvarā ir kaut viena iztrūkstoša vērtība, var izmantot funkciju any() šādā veidā:

```
> any(is.na(Dati))
[1] FALSE
> |
```

***Piezīme:*** ar funkcija any() pārbauda, vai kaut vienā vektora elementā nosacījums izpildās. Līdzīga šai funkcijai ir all(), kura pārbauda, vai visos vektora elementos izpildās apgalvojums. Šīs funkcijas iet cauri katram vektora elementam, un lietotājam nav jāveido for() vai while() cikli, lai veiktu datu pazīmju pārbaudi. Šīs funkcijas sniedz atbildi FALSE/TRUE veidā par pārbaudāmo apgalvojumu. Funkcijas vienlaikus veic arī vairākus salīdzinājumus, piemēram:

```
> all(Dati$Vecums>0)
[1] TRUE
> all(Dati$Dzimums== "Sieviete")
[1] FALSE
> all(Dati$HbAlcStarpiba<0)
[1] FALSE
> all(Dati$HbAlcStarpiba<0 | Dati$HbAlcStarpiba>0)
[1] FALSE
> all Dati$HbAlcStarpiba<=0 | Dati$HbAlcStarpiba>0)
```

12. attēls |

| | Nr | Dzimums | Vecums | HbA1cPirms | HbA1cPec | HbA1cStarpiba | Vecgrupa |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 14 | 11.8 | 9.3 | 2.5 | 1 |
| 2 | 2 | 2 | 22 | 6.3 | 9.2 | -2.9 | 1 |
| 3 | 3 | 1 | 23 | 8.2 | 7.2 | 1.0 | 1 |
| 4 | 4 | 1 | 30 | 5.5 | 5.5 | 0.0 | 1 |
| 5 | 5 | 2 | 34 | 5.9 | 5.4 | 0.5 | 1 |
| 6 | 6 | 1 | 42 | 9.1 | 6.8 | 2.3 | 2 |
| 7 | 7 | 1 | 42 | 10.7 | 12.3 | -1.6 | 2 |

```
[1] TRUE
> 1
```

Vēl programmā *R* ir speciāls operators %in%, ar kura palīdzību var pārbaudīt, vai kāda vērtība pieder pazīmei jeb vektoram, piemēram:

```
> "Siev." %in% Dati$Dzimums
[1] FALSE
> "sieviete" %in% Dati$Dzimums
[1] TRUE
> 42 %in% Dati$Vecums
[1] TRUE
> 420 %in% Dati$Vecums
[1] FALSE
> c("Sieviete","Vīrieti")%in% Dati$Dzimums
[1] TRUE FALSE
> |
```

Savukārt, lai novērtētu, cik pavisam respondenti ir ar pilnībā aizpildītiem datiem (nevienā rindā nav NA), izmantojam funkciju complete.cases(), kura sniegs atbildi par katra respondenta visu pazīmju aizpildītām vērtībām. Tad kopējo TRUE skaitu iegūstam ar funkciju sum():

```
> complete .cases(Dati $Dzimums)
[1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
[13] TRUE TRUE TRUE TRUE TRUE
> sum(complete.cases(Dati))
[1] 17
> |
```

Ja vēlamies pazīmi *vecums* automatizēti iedalīt papildus trīs faktoros – mazāk par 40 (neieskaitot) gadi, no 40 līdz 60 gadiem, vairāk par 60 (ieskaitot) gadiem, tad vispirms ar komandu izveidojam jaunu kolonu esošajos datos ar nosaukumu *vecgrupa*:

```
> Dati["Vecgrupa"]=NA
> |
```

***Piezīme:*** Iai pievienotu jaunu kolonu var izmantot arī funkciju cbind():

```
> Dati<-cbind(Dati, Vecgrupa=NA)
> View (Dati)
```

kā rezultātā datu logs izskatīsies (*skat. 13. attēlu*). (jau ar pievienotu jaunu kolonu *vecgrupa*, kurā ir tukšas vērtības NA (*Not Available*)).

Un izpildot sekojošas funkcijas tiks automātiski kolonā *vecgrupa* tiks ielikti skaitļi 1, 2, 3, atbilstoši respondenta vecumam:

```
> Dati$Vecgrupa[Dati$ Vecums<40]=1
> Dati$Vecgrupa[Dati$ Vecums>=40 & Dati $Vecums<60]=2
> Dati $Vecgrupa[Dati$ Vecums>=60]=3
> |
```

Kā rezultātā kopējais datu logs izskatīsies šādi (jau ar pazīmes *vecgrupa* iedalījumu ar atbilstošiem skaitļiem 1, 2, 3) (*skat. 14. attēlu*).

Un turpmākajā solī ar funkciju factor() ir iespējams izveidot pazīmi *vecgrupa* kā faktoru ar paša lietotāja definētiem nosaukumiem (angl. *labels*). Ja funkcijā ar papildus parametru tiks norādīts ordered=TRUE, kas nozīmētu, ka *vecgrupa* būs rangu pazīme (respektīvi, viena vecuma grupa ir lielāka nekā iepriekšējā):

```
> Dati$Vecgrupa=factor(Dati$ Vecgrupa,labels=c("< 40 gadi",
+ "no 40 līdz 59 gadi", "vairāk nekā 60 gadi"), ordered=TRUE)
> |
```

13. attēls |

| | Nr | Dzimums | Vecums | HbA1cPirms | HbA1cPec | HbA1cStarpiba | Vecgrupa |
|---|---|---|---|---|---|---|---|
| 1 | 1 | Sieviete | 14 | 11.8 | 9.3 | 2.5 | NA |
| 2 | 2 | Vīrietis | 22 | 6.3 | 9.2 | -2.9 | NA |
| 3 | 3 | Sieviete | 23 | 8.2 | 7.2 | 1.0 | NA |
| 4 | 4 | Sieviete | 30 | 5.5 | 5.5 | 0.0 | NA |
| 5 | 5 | Vīrietis | 34 | 5.9 | 5.4 | 0.5 | NA |
| 6 | 6 | Sieviete | 42 | 9.1 | 6.8 | 2.3 | NA |
| 7 | 7 | Sieviete | 42 | 10.7 | 12.3 | -1.6 | NA |
| 8 | 8 | Sieviete | 42 | 14.2 | 13.1 | 1.1 | NA |

Showing 1 to 10 of 17 entries, 7 total columns

14. attēls |

| | Nr | Dzimums | Vecums | HbA1cPirms | HbA1cPec | HbA1cStarpiba | Vecgrupa |
|---|---|---|---|---|---|---|---|
| 1 | 1 | Sieviete | 14 | 11.8 | 9.3 | 2.5 | 1 |
| 2 | 2 | Vīrietis | 22 | 6.3 | 9.2 | -2.9 | 1 |
| 3 | 3 | Sieviete | 23 | 8.2 | 7.2 | 1.0 | 1 |
| 4 | 4 | Sieviete | 30 | 5.5 | 5.5 | 0.0 | 1 |
| 5 | 5 | Vīrietis | 34 | 5.9 | 5.4 | 0.5 | 1 |
| 6 | 6 | Sieviete | 42 | 9.1 | 6.8 | 2.3 | 2 |
| 7 | 7 | Sieviete | 42 | 10.7 | 12.3 | -1.6 | 2 |
| 8 | 8 | Sieviete | 42 | 14.2 | 13.1 | 1.1 | 2 |
| 9 | 9 | Vīrietis | 43 | 16.7 | 16.1 | 0.6 | 2 |

Showing 1 to 10 of 17 entries, 7 total columns

15. attēls |

| | Nr | Dzimums | Vecums | HbA1cPirms | HbA1cPec | HbA1cStarpiba | Vecgrupa |
|---|---|---|---|---|---|---|---|
| 1 | 1 | Sieviete | 14 | 11.8 | 9.3 | 2.5 | < 40 gadi |
| 2 | 2 | Vīrietis | 22 | 6.3 | 9.2 | -2.9 | < 40 gadi |
| 3 | 3 | Sieviete | 23 | 8.2 | 7.2 | 1.0 | < 40 gadi |
| 4 | 4 | Sieviete | 30 | 5.5 | 5.5 | 0.0 | < 40 gadi |
| 5 | 5 | Vīrietis | 34 | 5.9 | 5.4 | 0.5 | < 40 gadi |
| 6 | 6 | Sieviete | 42 | 9.1 | 6.8 | 2.3 | no 40 līdz 59 gadi |
| 7 | 7 | Sieviete | 42 | 10.7 | 12.3 | -1.6 | no 40 līdz 59 gadi |
| 8 | 8 | Sieviete | 42 | 14.2 | 13.1 | 1.1 | no 40 līdz 59 gadi |
| 9 | 9 | Vīrietis | 43 | 16.7 | 16.1 | 0.6 | no 40 līdz 59 gadi |
| 10 | 10 | Sieviete | 72 | 7.5 | 6.5 | 1.0 | vairāk nekā 60 gadi |

Showing 1 to 12 of 17 entries, 7 total columns

***Piezīme:*** papildus pastāv funkcija cut(), ar kuras palīdzību būtu iespējams sadalīt pazīmi vienāda intervāla daļās, piemēram, Dati$JaunsVecumaledalijums <- cut(Dati$Vecums, breaks=3, ordered_result = TRUE) automātiski pazīmi iedalītu šādās grupās: 14–38 gadi, 38–62 gadi un 62–86 gadi.

***Piezīme:*** funkcijai factor() līdzīga funkcija ir ordered(), kurā nav jānorāda papildus nosacījums, ka izveidotai pazīmei ir rangi, jo šī funkcija jau pēc noklusējuma to dara:

```
> Dati$Vecgrupa=ordered
Dati$Vecgrupa,labels= c "< 40
gadi", + "no 40 līdz 59 gadi",
"vairāk nekā 60 gadi"))
```

***Piezīme:*** liela nozīme programmā *R* ir, vai pazīmei ir definēti rangi. Tas sniedz papildus priekšrocības, piemērot, datu apstrādes rezultātus attēlojot sakārtotā secībā vai arī regresijas modeļu analīzē.

Rezultātā datu logs izskatīsies kā redzams 15. attēlā.

***Piezīme:*** pazīmes *vecgrupa* izveidi varēja arī ietilpināt vienā funkcijā cut(), kurā var norādīt visas nepieciešamās darbības, gan pazīmes *vecums* robežas, gan nosaukumus, gan rangu secību:

```
> Dati["Vecgrupa"]<- cut
(Dati$Vecums, breaks = c
0,40,60, Inf), labels = c
("< 40 gadi'', "no 40 līdz 59
gadi", "vairāk nekā 60 gadi "),
ordered_result = TRUE)
> View (Dati)
> |
```

Ar komandu is.ordered() ir iespējams noteikt, kura faktoriālā pazīme ir rangu veida, bet kura – nav:

```
> is.ordered(Dati$Dzimums)
[1] FALSE
> is.ordered(Dati$Vecgrupa)
[1] TRUE
> I
```

Lietojot iepriekš izveidoto funkciju DatuKodejumalegusana(), iegūstam šo:

```
> DatuKodejumaIegusana(Dati$
Vecgrupa)
    Pazīme              Kodējums
1  < 40 gadi                1
6  no 40 līdz 59 gadi       2
10 vai rāk nekā 60 gadi     3
> |
```

## Datu apskate un piekļuve

Ņemot vērā, ka ar komandu View() lielu datu bāzu gadījumā attēlošanas logā netiktu parādītas visas vērtības, tad datu kopskata vērtēšanai programmā *R* ir vairākas funkcijas, piemēram, head() un tail().

Funkcija head() var būt noderīga, jo sniegs informāciju tikai par pirmajām 6 rindām (*skat. 16. attēlu*).

Līdzīgi funkcija tail() pēc noklusējuma sniegs informāciju tikai par pēdējām 6 rindām (*skat. 17. attēlu*).

Lai apskatītu tieši konkrētu skaitu pēdējo datu rindu, piemēram, vēlamoties apskatīt nevis tikai sešas, bet 8 pēdējās datu rindas, jāizmanto funkciju tail(), kurā norāda vēlamo skaitu

Lai apskatītu datus konkrētā rindā (piemēram, 3.), var izmantot komandu, kurā kvadrātveida iekavās norāda pētāmās rindas vērtību, klāt pievienojot komata zīmi:

```
> Dati[3,]
# A tibble: 1 X 7
Nr  Dzimums  Vecums  HbA1cPec
HbA1cStarpiba Vecgrupa
<dbl> <fct> <dbl> <dbl> <dbl>
<dbl> <ord>
1  3 Sieviete 23 8.2  7.2  1.00
<40 gadi
> |
```

***Piezīme:*** rakstot norādi uz pazīmi vai funkcijām, ir nepieciešams ievērot lielo un mazo burtu nosacījumu, jo, piemēram, izsaukums dati[3,] ģenerēs kļūdas paziņojumu:

```
> dati [3,]
Error: object 'dati' not found
> |
```

Lai iegūtu konkrētas pazīmes noteiktu vērtību konkrētā rindā, piemēram, lai iegūtu piektā pēc kārtas respondenta vecumu (pazīme *vecums* atrodas 3. kolonā), var izmantot visus šos pieraksta variantus:

```
> Dati[5,"Vecums"]
# A tibble: 1 x 1
Vecums
<dbl>
1  34
```

16. attēls |

```
> head(Dati)
# A tibble: 6 x 7
     Nr Dzimums  Vecums HbA1cPirms HbA1cPec HbA1cStarpiba Vecgrup
  <dbl> <fct>     <dbl>      <dbl>     <dbl>          <dbl> <ord>
1     1 Sieviete     14       11.8       9.3            2.5 < 40 ga
2     2 Vīrietis     22        6.3       9.2           -2.90 < 40 ga
3     3 Sieviete     23        8.2       7.2            1.00 < 40 ga
4     4 Sieviete     30        5.5       5.5            0    < 40 ga
5     5 Vīrietis     34        5.9       5.4            0.5  < 40 ga
6     6 Sieviete     42        9.1       6.8            2.3  no 40 1
> |
```

17. attēls |

```
> tail(Dati)
# A tibble: 6 x 7
     Nr Dzimums  Vecums HbA1cPirms HbA1cPec HbA1cStarpiba Vecgru
  <dbl> <fct>     <dbl>      <dbl>     <dbl>          <dbl> <ord>
1    12 Sieviete     61       13.8      14.2         -0.400 vairāk
2    13 Sieviete     61       13.4      12.4          1     vairāk
3    14 Vīrietis     63       11.4       8.9          2.5   vairāk
4    15 Vīrietis     74        7.5       6.2          1.30  vairāk
5    16 Sieviete     79       15.3      13.9          1.4   vairāk
6    17 Vīrietis     86       17.8      15.4          2.4   vairāk
> |
```

18. attēls |

```
> tail(Dati, n=8)
# A tibble: 8 x 7
     Nr Dzimums  Vecums HbA1cPirms HbA1cPec HbA1cStarpiba Vecg
  <dbl> <fct>     <dbl>      <dbl>     <dbl>          <dbl> <ord
1    10 Sieviete     72        7.5       6.5          1     vair
2    11 Sieviete     65        8.7       7.7          1.00  vair
3    12 Sieviete     61       13.8      14.2         -0.400 vair
4    13 Sieviete     61       13.4      12.4          1     vair
5    14 Vīrietis     63       11.4       8.9          2.5   vair
6    15 Vīrietis     74        7.5       6.2          1.30  vair
7    16 Sieviete     79       15.3      13.9          1.4   vair
8    17 Vīrietis     86       17.8      15.4          2.4   vair
> |
```

19. attēls

```
> Dati[9:14,]
# A tibble: 6 x 7
     Nr Dzimums  Vecums HbA1cPirms HbA1cPec HbA1cStarpiba Vecgr
  <dbl> <fct>     <dbl>      <dbl>    <dbl>         <dbl> <ord>
1     9 Vīrietis     43       16.7     16.1         0.600 no 40
2    10 Sieviete     72        7.5      6.5         1     vairā
3    11 Sieviete     65        8.7      7.7         1.00  vairā
4    12 Sieviete     61       13.8     14.2        -0.400 vairā
5    13 Sieviete     61       13.4     12.4         1     vairā
6    14 Vīrietis     63       11.4      8.9         2.5   vairā
> |
```

20. attēls

```
> Dati[,c("HbA1cPirms", "HbA1cPec")]
# A tibble: 17 x 2
   HbA1cPirms HbA1cPec
        <dbl>    <dbl>
 1       11.8      9.3
 2        6.3      9.2
 3        8.2      7.2
 4        5.5      5.5
 5        5.9      5.4
 6        9.1      6.8
 7       10.7     12.3
 8       14.2     13.1
 9       16.7     16.1
```

21. attēls

```
> Dati[c(4:10),c(1,3,4)]
# A tibble: 7 x 3
     Nr Vecums HbA1cPirms
  <dbl>  <dbl>      <dbl>
1     4     30        5.5
2     5     34        5.9
3     6     42        9.1
4     7     42       10.7
5     8     42       14.2
6     9     43       16.7
7    10     72        7.5
> |
```

22. attēls

```
> Dati[Dati$Vecums>60,]
# A tibble: 8 x 7
     Nr Dzimums  Vecums HbA1cPirms HbA1cPec HbA1cStarpiba Vecgrupa
  <dbl> <fct>     <dbl>      <dbl>    <dbl>         <dbl> <ord>
1    10 Sieviete     72        7.5      6.5         1     vairāk …
2    11 Sieviete     65        8.7      7.7         1.00  vairāk …
3    12 Sieviete     61       13.8     14.2        -0.400 vairāk …
4    13 Sieviete     61       13.4     12.4         1     vairāk …
5    14 Vīrietis     63       11.4      8.9         2.5   vairāk …
6    15 Vīrietis     74        7.5      6.2         1.30  vairāk …
7    16 Sieviete     79       15.3     13.9         1.4   vairāk …
8    17 Vīrietis     86       17.8     15.4         2.4   vairāk …
> |
```

```
> Dati [[3]][5]
[1] 34
>  Dati [["vecums"]][5]
[1] 34
> |
```

Datu analīzē noderīga ir arī funkcija split(), kura iedala datus, piemēram:

```
> Sievvecums<-split
(Dati$Vecums, Dati$Dzimums==
"Sieviete")
> SievVecums
$'FALSE'
[1] 22 34 43 63 74 86
$'TRUE'
 [1] 14 23 30 42 42 42 72 65
61 61 79
> Sievvecums[[2]]
 [1] 14 23 30 42 42 42 72 65
61 61 79
> |
```

Lai apskatītu noteiktās rindās esošus datus, piemēram, no 9. līdz 14. rindai, tad dara kā attēlots 19. attēlā.

Lai apskatītu vai atlasītu turpmākai analīzei divas atsevišķas pazīmes, tad ar funkciju c() izveidojam vektoru no abām pazīmēm un ievietojam to datu ietvara atsaucē uz kolonu (*skat. 20. attēlu*).

Lai atlasītu konkrētas rindas un kolonas, piemēram, vēlamies apskatīt datus no 4. līdz 10. rindai un 1., 3., 4. kolonu, tad ar funkciju c() izveido divus atbilstošus vektorus un ievieto tos atsaucē uz datu ietvaru (*skat. 21. attēlu*).

Ja ir nepieciešams atlasīt visās kolonās esošus datus, bet ar nosacījumu (piemēram, respondentu vecums virs 60 gadiem), tad atsaucē uz datiem kvadrātveida iekavās norāda vēlamo un pievieno komata zīmi (*skat. 22. attēlu*).

Ja ir nepieciešams atlasīt datus ar diviem vienlaicīgi izpildāmiem nosacījumiem, tad atsaucē uz datiem kvadrātveida iekavās norāda abus nosacījumus ar & operatoru (*skat. 23. attēlu*).

Līdzīgi var izmantot funkciju subset(), kura ir vienkāršākais no visiem pazīmju atlases veidiem. Piemēram, lai no datiem atlasītu *sievietes* (šajā gadījumā aiz nosacījuma *dzimums* ar divām vienādības zīmēm norādām meklēto) ar vecumu virs 30 (ieskaitot), var darīt kā redzams 24. attēlā.

***Piezīme:*** ja ir nepieciešams no funkcijas subset() rezultāta turpmākai analīzei un transformācijām atlasīt tikai vienu vai vai vairākas kolonas, tad papildus var pievienot nosacījumu select, ar kuru vektora veidā

23. attēls

```
> Dati[Dati$Vecums>60 & Dati
# A tibble: 4 x 7
     Nr Dzimums  Vecums HbA1
  <dbl> <fct>     <dbl>
1    12 Sieviete     61
2    13 Sieviete     61
3    16 Sieviete     79
4    17 Vīrietis     86
> |
```

24. attēls

```
> subset(Dati, Dzimums=="Sieviete" & Vecums>=30)
# A tibble: 9 x 7
     Nr Dzimums  Vecums HbA1cPirms HbA1cPec HbA1cStarpiba Vecgrup
  <dbl> <fct>     <dbl>      <dbl>    <dbl>         <dbl> <ord>
1     4 Sieviete     30        5.5      5.5         0     < 40 ga
2     6 Sieviete     42        9.1      6.8         2.3   no 40 l
3     7 Sieviete     42       10.7     12.3        -1.6   no 40 l
4     8 Sieviete     42       14.2     13.1         1.10  no 40 l
5    10 Sieviete     72        7.5      6.5         1     vairāk
6    11 Sieviete     65        8.7      7.7         1.00  vairāk
7    12 Sieviete     61       13.8     14.2        -0.400 vairāk
8    13 Sieviete     61       13.4     12.4         1     vairāk
9    16 Sieviete     79       15.3     13.9         1.4   vairāk
```

norāda vēlamo kolonu numuru/s (vai nosaukumu/s):

```
> subset(Dati, Dzimums== "Sieviete" & vecums>=30, select=c(Vecums, HbAlcPirms))
# A tibble: 9 X 2
vecums HbAlcPirms
<dbl>       <dbl>
1 30          5.5
2 42          9.1
3 42          10.7
4 42          14.2
5 72          7.5
6 65          8.7
7 61          13.8
8 61          13.4
9 79          15.3
> |
```

Ja, savukārt, ir nepieciešamība uzzināt, kurās datu ietvara rindās atrodas precīzi definēts nosacījums, piemēram, vēlamies atrast sievietes vecumā virs 30 gadiem (ieskaitot), tad izmanto funkciju which():

```
> which(Dati$Dzimums== "Sieviete" & Dati$Vecums>=30)
[1] 4 6 7 8 10 11 12 13 16
> |
```

Līdzīgi, lai atrastu, kurā/s rindā/s ir, piemēram, maksimālā/s *HbA1cStarpiba* izmaiņas, lieto:

```
> which(Dati$HbAlcStarpiba== max(Dati$HbAlcStarpiba))
[1] 1 14
> |
```

Lai nejaušā veidā atlasītu, piemēram, piecus respondentus no kopējiem datiem, tad var izmantot funkciju sample() (*skat. 25. attēlu*).

Lai nomainītu datu kolonas nosaukumu, jāizmanto funkcija names(). Piemēram, lai nomainītu pazīmes *vecgrupa*, kura ir septītā kolona pēc kārtas uz nosaukumu *vecuma grupa*, būtu jādara šādi:

```
> names(Dati)[7]= "Vecuma grupa"
> |
```

***Piezīme:*** pazīmes nosaukuma izveide, kurā atrodas atstarpe, var mazliet sarežģīt darbošanos ar šī vektora nosaukumu, jo turpmāk, atsaucoties uz šo pazīmi būs jāizmanto papildus simbols – vai nu "Dati$"Vecuma grupa", vai `:

```
> Dati$'Vecuma grupa'
[1] < 40 gadi       < 40 g
[3] < 40 gadi       < 40 g
[5] < 40 gadi       no 40
[7] no 40 līdz 59 gadi no 40
```

Datu ietvarā ir iespējams arī mainīt kolonu atrašanās vietu, piemēram, ja vēlētos pārvietot datu kolonu *vecgrupa* blakus pazīmei *vecums*, būtu nepieciešams izveidot vektoru c(), kurā tiek ierakstīts vēlamo kolonu izvietojums:

```
> Dati <- Dati[c(1,2,3,7,4,5,6)]
> |
```

Kā rezultātā iegūtu (*skat. 26. attēlu*).

***Piezīme:*** ja vēlētos pašu pēdējo kolonu (mūsu gadījumā 7.) pārvietot sākumā, tad varētu izmantot šo pieraksta veidu: Dati<-Dati[,c(7,1:6)] (Ar divpunkta zīmi (:) tiek apzīmēts skaitļu rindas: 1,2,3,4,5,6 saīsināts pieraksts.

***Piezīme:*** lai izdzēstu datu kolonu, piemēram, *vecgrupa,* var izmantot šādu funkciju: Dati$Vecgrupa=NULL (programmā *R* ir īpašs objekts ar nosaukumu NULL. To lieto, kad ir nepieciešams norādīt vai precizēt, ka objekta nav. To nevajadzētu sajaukt ar vektoru vai sarakstu ar nulles garumu).

***Piezīme:*** lai izdzēstu konkrētu rindu datos, piemēram, 5. rindu, var izmantot komandu Dati=Dati[-c(5),]

25. attēls |

```
> Dati[sample(1:nrow(Dati), 5, replace=FALSE),]
# A tibble: 5 x 7
     Nr Dzimums  Vecums HbA1cPirms HbA1cPec HbA1cStarpiba Vecgr
  <db1> <fct>     <db1>      <db1>     <db1>         <db1> <ord>
1    16 Sieviete     79       15.3      13.9           1.4  vairā
2    17 Vīrietis     86       17.8      15.4           2.4  vairā
3    15 Vīrietis     74        7.5       6.2          1.30  vairā
4     2 Vīrietis     22        6.3       9.2         -2.90  < 40
5    10 Sieviete     72        7.5       6.5           1    vairā
> |
```

26. attēls |

fails | skriptsrakstam.R | Dati

Filter

| | Nr | Dzimums | Vecums | Vecgrupa | HbA1cPirms | HbA1cPec | HbA1cStarpiba |
|---|---|---|---|---|---|---|---|
| 1 | 1 | Sieviete | 14 | < 40 gadi | 11.8 | 9.3 | 2.5 |
| 2 | 2 | Vīrietis | 22 | < 40 gadi | 6.3 | 9.2 | -2.9 |
| 3 | 3 | Sieviete | 23 | < 40 gadi | 8.2 | 7.2 | 1.0 |
| 4 | 4 | Sieviete | 30 | < 40 gadi | 5.5 | 5.5 | 0.0 |
| 5 | 5 | Vīrietis | 34 | < 40 gadi | 5.9 | 5.4 | 0.5 |
| 6 | 6 | Sieviete | 42 | no 40 līdz 59 gadi | 9.1 | 6.8 | 2.3 |
| 7 | 7 | Sieviete | 42 | no 40 līdz 59 gadi | 10.7 | 12.3 | -1.6 |
| 8 | 8 | Sieviete | 42 | no 40 līdz 59 gadi | 14.2 | 13.1 | 1.1 |

## Informācijas iegūšana par datubāzes objektiem

Lai apskatītu visus definētos objekta atribūtus, var izmantot funkciju attributes():

```
> attributes(Dati$Vecgrupa)
$levels
[1]"< 40 gadi" "no 40 līdz 59 gadi"
[3] "vairāk nekā 60 gadi"
$class
[1]"ordered" "factor"
> |
```

Lai noteiktu objekta veidu, var izmantot funkciju class():

```
> class(Dati)
[1] "tbl_df " "tbl" "data.frame"
> class(Dati$Dzimums)
[1] "factor"
> class(Dati$Vecums)
[1] "numeric"
> |
```

Lai noteiktu objekta dimensijas, izmantojam funkciju dim(), kā rezultātā iegūstam rindu un kolonu skaitu:

```
> dim(Dati)
[1] 17  7
> |
```

Citreiz ir nepieciešams atsevišķi iegūt datu ietvara rindu un kolonu skaitu, tad izmanto nrow() un ncol() funkcijas:

```
> nrow (Dati)
[1] 17
> ncol(Dati)
```

```
[1]  7
> |
```

Lai noskaidrotu visu datu ietvarā esošo pazīmju nosaukumus, izmantojam funkciju names() un norādām objekta nosaukumu:

```
> names (Dati)
[1] "Nr" "Dzimums" "Vecums" "HbAlcPirms"
[5]"HbAlcPec" "HbAlcStarpiba" "Vecgrupa"
> 1
```

Komanda str() sniedz informāciju par pašu datu struktūru un pirmajiem datiem, piemēram, str(Dati) sniegs informāciju par objektu ar nosaukumu *Dati* (*skat. 27. attēlu*).

Komanda summary() noder vispārējai datu kopsavilkuma apskatei, piemēram, summary(Dati) sniegs pilnīgi visu datu kopsavilkumu atsevišķi katrai kvantitatīvajai un kvalitatīvajai pazīmei.

Kvantitatīvajai pazīmei tiks aprēķināta minimālā vērtība (*Min.*), pirmā kvartīle (*1st Qu.*), mediāna (*Median*), vidējā vērtība (*Mean*), trešā kvartīle (*3rd Qu.*) un maksimālā vērtība (*Max.*).

Kvalitatīvajai pazīmē tiks aprēķināts skaits (*skat. 28. attēlu*).

Ja analizējamo pazīmju ir ļoti daudz, tad piemēram, funkcija summary (Dati$Dzimums) sniegs vienas kategoriskās pazīmes *dzimums* kopējo *sieviešu* un *vīriešu* skaitu:

```
> summary(Dati$Dzimums)
Sieviete Vīrietis
      11         6
> |
```

***Piezīme:*** tādu pašu rezultātu var sniegt speciāla funkcija xtabs(), kura domāta kontingences tabulu izveidei, bet vienkāršākā gadījumā sniedz atbildi arī par vienas kategoriskās pazīmes biežumu:

```
> xtabs(~Dzimums, data=Dati)
Dzimums
Sieviete Vīrietis
      11         6
> |
```

Ja komanda summary() pielietota atsevišķai kvantitatīvai pazīmei, piemēram, summary(Dati$Vecums), tā dos iespēju apskatīt pazīmes *vecums* vidējo vērtību, minimālo un maksimālo vērtību, pirmo kvartīli, mediānu un trešo kvartīli:

```
> summary(Dati$Vecums)
Min. 1st Qu. Median Mean 3rd Qu. Max.
14.00 34.00 43.00 50.18 65.00 86.00
> |
```

Lai iegūtu pazīmes *vecums* aprakstošās statistikas vērtības sievietēm un vīriešiem, var pielietot funkciju by(), kura rada labi formatētu rezultāta attēlojumu:

```
> by Dati[,"Vecums"], Dati$Dzimums, summary)
Dati$Dzimums: Sieviete
Vecums
Min.        :14.00
1st Qu.     :36.oo
Median      :42.00
Mean        :48.27
3rd QU.     :63.00
Max.        :79.00
--------------------------
Dati$Dzimums: vīrietis
Vecums
Min.        :22.00
1st QU .    :36.25
Median :    53.00
Mean        :53.67
3rd Qu.     :71.25
Max.        :86.00
> |
```

## Datu analīze

Lai aprēķinātu pazīmes *dzimums* procentuālo attiecību, var izmantot šādu aprēķinu, un rezultāts būs:

```
> 100* table(Dati$Dzimums)/ sum(table(Dati$Dzimums)))
Sieviete Vīrietis
64.70588 35.29412
> |
```

Lai aprēķinātu *vecgrupa* procentuālo attiecību un iegūtu noapaļotu rezultātu ar 2 zīmēm aiz komata, izmantojam funkciju round(), kurā tiek norādīts vēlamais decimāldaļu skaits aiz komata, kā rezultātā iegūsim:

```
> round(100*table(Dati$ Vecgrupa/sum(table Dati$ Vecgrupa)),2)

< 40 gadi  no 40 līdz 59 gadi  vairāk nekā 60 gadi
23.53        47.06        29.41
> |
```

Tādu pašu rezultātu var iegūt izmantojot vienkāršāku funkciju prop.table():

```
> round(100*prop.table(table (Dati$Vecgrupa)),2)
< 40 gadi  no 40 līdz 59 gadi  vairāk nekā 60 gadi
29.41        23.53        47.06
> |
```

Ja ir nepieciešams bieži aprēķināt skaitu

27. attēls |

```
> str(Dati)
Classes 'tbl_df', 'tbl' and 'data.frame':       17 obs. of  7 variables:
 $ Nr           : num  1 2 3 4 5 6 7 8 9 10 ...
 $ Dzimums      : Factor w/ 2 levels "Sieviete","Vīrietis": 1 2 1 1 2 1 1
 1 2 1 ...
 $ Vecums       : num  14 22 23 30 34 42 42 42 43 72 ...
 $ HbAlcPirms   : num  11.8 6.3 8.2 5.5 5.9 9.1 10.7 14.2 16.7 7.5 ...
 $ HbAlcPec     : num  9.3 9.2 7.2 5.5 5.4 6.8 12.3 13.1 16.1 6.5 ...
 $ HbAlcStarpiba: num  2.5 -2.9 1 0 0.5 ...
 $ Vecgrupa     : Ord.factor w/ 3 levels "< 40 gadi"<"no 40 līdz 59 gadi"
<..: 1 1 1 1 1 2 2 2 2 3 ...
> |
```

28. attēls |

```
> summary(Dati)
       Nr          Dzimums      Vecums        HbAlcPirms      HbAlcPec
 Min.   : 1   Sieviete:11   Min.   :14.00   Min.   : 5.50   Min.   : 5.40
 1st Qu.: 5   Vīrietis: 6   1st Qu.:34.00   1st Qu.: 7.50   1st Qu.: 6.80
 Median : 9                 Median :43.00   Median :10.70   Median : 9.20
 Mean   : 9                 Mean   :50.18   Mean   :10.81   Mean   :10.01
 3rd Qu.:13                 3rd Qu.:65.00   3rd Qu.:13.80   3rd Qu.:13.10
 Max.   :17                 Max.   :86.00   Max.   :17.80   Max.   :16.10
 HbAlcStarpiba                    Vecgrupa
 Min.   :-2.9000   < 40 gadi          :5
 1st Qu.: 0.5000   no 40 līdz 59 gadi :4
 Median : 1.0000   vairāk nekā 60 gadi:8
 Mean   : 0.8059
 3rd Qu.: 1.4000
 Max.   : 2.5000
> |
```

un procentuālo attiecību, tad ir iespējams izveidot funkciju, kuru jebkurā brīdī var izsaukt ar nepieciešamo apēķināmo kategorisko pazīmi:

```
> procenti <- function(x){
+ tabula <- table(x)
+ rezult<-cbind(tabula,round
prop.table tabula *100,2))
+ colnames(rezult)<-
c "Skaits",'"Procenti %")
+ return (rezult)}
> procenti(Dati$Vecgrupa)
              Skaits Procenti %
< 40 gadi          5 29.41
no 40 līdz 59 gadi 4 23.53
vairāk nekā 60 gadi 8 47.06
> |
```

Lai atsevišķi aprēķinātu pazīmes *vecums* vidējo vērtību, izmantojam:

```
> mean(Dati$Vecums)
[1] 50.17647
> |
```

***Piezīme:*** ja analizējamās pazīmēs ir kādas iztrūkstošas vērtības (tiek apzīmētas ar NA (*Not Available*)), tad daudzu aprēķinu funkciju rezultāts arī būs NA. Lai to novērstu, funkciju aprēķinu rindā papildus var ierakstīt nosacījumu na.rm = TRUE, kas angliski nozīmē – *"Not available, remove"*, un, ja piešķiramā vērtība ir TRUE, tas nozīmē, ka datu kolonā iztrūkstošās vērtības tiks ignorētas. Piemēram, mean(Dati$Vecums, na.rm=TRUE). Šo var uzskatāmi parādīt šajā piemērā:

```
> mean(v., na.rm=TRUE)
[1] 3
>  vērtības<-c(1,2,3,4 ,5,NA)
> mean(vērtības)
[1] NA
> mean(vērtības, na.rm=TRUE)
[1] 3
> |
```

Lai iegūtu noapaļotu kvantitatīvās pazīmes vērtību izmanto komandu round():

```
> round(mean(Dati$Vecums), 2)
[1]  50.18
> |
```

***Piezīme:*** ar funkciju paste() ir iespējams izpildāmās funkcijas aprēķināto vērtību ievietot teikumā:

```
> paste ("Vidējais
respondentu vecums ir", round
mean(Dati$Vecums),
  2), "gadi")
  [1]"Vidējais respondentu
vecums ir 50.18 gadi"
  > 1
```

Lai aprēķinātu pazīmes vidējo vērtību analizējamo datu tikai pirmajām 8 vērtībām, izmantojam funkciju mean(), kurā kvadrātveida iekavās norādam nepieciešamo vērtību intervālu, kuram tiks veiks aprēķins:

```
> mean(Dati$Vecums [1:8])
[1] 31.125
> |
```

Kāmēr programmā *R* funkcijas mean() un median() ir standartā, tad, lai aprēķinātu modālo jeb visbiežāk sastopamo vērtību, vispirms pašam ir jāizveido atsevišķa funkcija, piemēram, ar nosaukumu mode(), kurā kā aprēķināmo parametru var norādīt pazīmi:

```
> mode<-function(x){
+ er=unique(x)
+ er[which .max(tabulate(match
(x, er)))]
+ }
> |
```

Izveidotās funkcijas mode() rezultātā iegūsim, ka modālais vecums analizējamos datos ir 42 gadi:

```
> mode(Dati $Vecums)
[1] 42
> |
```

Funkcija sd() sniedz iespēju aprēķināt kvantitatīvās pzīmes standartnovirzes vērtību:

```
> sd(Dati$Vecums)
[1] 21.70609
> |
```

Funkcija range() sniegs atbildi par pazīmes minimālo un maksimālo vērtību:

```
> range(Dati$Vecums)
[1] 14 86
> |
```

Funkcija quantile() sniedz iespēju aprēķināt kvartīles:

```
> quantile(Dati$Vecums)
0%  25%  50%  75%  100%
14   34   43   65   86
> |
```

Ja ir nepieciešams aprēķināt konkrētu procentīli, tad izmantojam funkciju quantile(), kurā kā papildus parametru norādām meklējamo vērtību (no 0 līdz 1). Piemēram, pazīmes *vecums* 35. procentīle tiek aprēķināta:

```
> quantile(Dati$Vecums, probs
= 0.35)
  35%
  42
  > |
```

Lai aprēķinātu pazīmes deciles, izmantojam funkciju quantile(), kurā norāda atsauci uz pētāmo pazīmi un ar funkciju seq() jeb *sequency* definē deciles no 0 līdz 10

```
  > quantile(Dati$Vecums, seq
(from=0, to=1, by=0.1))
  0%  10%  20%  30%  40%  50%
60%  70%  80%  90%  100%
  14.0  22.6  30.8  40.4  42.0
43.0  61.0  63.4 70.6 76.0 86.0
  > |
```

Funkcija IQR() aprēķina starpkvartīļu robežu:

```
  > IQR(Dati$Vecums)
  [1] 31
  > |
```

Ar funkciju mad() var aprēķināt mediānas absolūto novirzi:

```
  > mad(Dati$Vecums)
  [1]  29.652
  > |
```

Daudzos aprēķinos noderīga funkcija ir apply(), zemāk attēlotā piemērā katram respondentam tiek aprēķināta vidējā vērtība no 4. un 5. kolonas (kas atbilst *HbA1cPirms* un *HbA1cPec*). Šajā funkcijā ar papildus argumentu skaitli 1 jeb MARGIN=1 tiek norādīts, ka aprēķini jāveic katrai datu rindai.

```
  > apply Dati[,4:5]'MARGIN=1,
FUN=mean)
  [1] 10.55 7.75 7.70 5.50 5.65
7.95 11.50 13.65
  [9] 16.40 7.00 8.20 14.00
12.90 10.15 6.85 14.60
  [17] 16.60
  > |
```

Savukārt funkcijā apply() ar papildus nosacījumu MARGIN=2 tiek norādīts, ka aprēķināmā funkcija jāveic ierakstītajai kolonai:

```
  > apply(Dati[,4:5], MARGIN=2,
FUN=mean)
  HbAlcPirms  HbAlcPec
  10.81176    10.00588
  > |
```

***Piezīme:*** tādu pašu rezultātu var iegūt, atsaucoties uz datu kolonām ar vektoru, kuram priekšā ir mīnus zīme un norādot, kuras kolonas ir jāizslēdz no aprēķiniem:

```
  > apply Dati[,-c 1,2,3,6,7)],
MARGIN=2, FUN=mean)
  HbAlcPirms  HbAlcPec
  10.81176    10.00588
  > |
```

Lai aprēķinātu vidējo vecumu atsevišķi *sievietēm* un *vīriešiem*, var izmantot arī funkciju tapply(), kurā kā argumentus norāda pētāmo un faktoriālo pazīmi, kā arī

aprēķināmās funkcijas nosaukumu:

```
> tapply Dati$Vecums, Dati$Dzimums, FUN=mean)
  Sieviete Vīrietis
  48.27273 53.66667
> |
```

***Piezīme:*** funkcija tapply() tiek izmantota, lai pielietotu aprēķināmo funkciju uz vektora apakšgrupām.

Līdzīgu vidējās vērtības aprēķinu var iegūt arī, izmantojot funkciju with(), kas var būt noderīga, lai vienkāršotu izsaukumus uz izpildāmām pazīmēm, īpaši, ja izteiksmē to ir daudz:

```
> with(Dati, tapply(Vecums, Dzimums, FUN=mean))
  Sieviete Vīrietis
  48.27273 53.66667
> |
```

***Piezīme:*** līdzīgi, lai aprēķinātu sieviešu un vīriešu vidējo vecumu, var izmantot datu atlases komandas, vispirms izveidojot jaunas pazīmes ar nosaukumu Sievietes un Viriesi, un pielietojot atbilstošu aprēķinu funkciju:

```
> Sievietes=Dati[Dati$Dzimums=="Sieviete",]
> mean Sievietes$Vecums)
[1] 48.27273
> Viriesi=Dati[Dati$Dzimums=="Vīrietis ",]
> mean Viriesi$Vecums)
[1] 53.66667
> |
```

***Piezīme:*** tādu pašu rezultātu var iegūt izmantojot apkopojuma funkciju agregate(), kura sadala datus apakšgrupā/s un katrai no tām aprēķina norādīto funkciju:

```
> aggregate(Vecums~Dzimums, Dati, FUN=mean)
     Dzimums  Vecums
1  Sieviete 48.27273
2  Vīrietis 53.66667
> |
```

***Piezīme:*** principā funkcijai ir paredzēti šādi argumenti: aggregate(formula, dati, funkcija, ...). Tā ir ļoti nozīmīga funkcija datu analīzei un ir daudz sarežģītāka un domāta darbam ar vairāku pazīmju šķēlumiem un ar vienu aprēķināmo funkciju. Rezultātā tiek iegūts jauns datu ietvars. Savukārt funkcija tapply() ir domāta darbam ar vienu datu vektoru, kura rezultātā iegūst datu matricu vai masīvu.

Savukārt, lai aprēķinātu pazīmes *HbAlcPirms* vidējo vērtību šķēlumā ar *dzimums* un *vecgrupa*, tad funkcijā aggregate() tas ir jānorāda šādi:

```
> aggregate(HbAlcPirms~Dzimums+Vecgrupa, Dati, FUN=mean)
  Dzimums Vecgrupa HbAlcPirms
1 Sieviete < 40 gadi 8.50000
2 Vīrietis  < 40 gadi 6.10000
3 Sieviete  no 40 līdz 59 gadi 11.33333
4 Vīrietis  no 40 līdz 59 gadi 16.70000
5 Sieviete  vairāk nekā 60 gadi 11.74000
6 Vīrietis  vairāk nekā 60 gadi 12.23333
> |
```

Ja nepieciešams funkcijā aggregate() norādīt vairākas analizējamās pazīmes atsevišķā šķēlumā (šajā gadījumā *dzimums*/*vecgrupa*), tad to var darīt šādi:

```
> aggregate(cbind Pirms=HbAlcPirms, Pēc=HbAlcPec) ~ Dzimums + vecgrupa, Dati, FUN=mean)
  Dzimums vecgrupa Pirms Pēc
1 Sieviete  < 40 gadi 8.50000 7.333333
2 Vīrietis  < 40 gadi 6.10000  7.300000
3 Sieviete  no 40 līdz 59 gadi 11.33333 10.733333
4 Vīrietis  no 40 līdz 59 gadi 16.70000 16.100000
5 Sieviete vai rāk nekā 60 gadi 11.74000 10.940000
6 Vīrietis vairāk nekā 60 gadi 12.23333 10.166667
>
```

Lai aprēķinātu *sieviešu* un *vīriešu* vecuma standartnovirzi, izmantojam:

```
> with(Dati, tapply(Vecums, Dzimums, FUN=sd))
  Sieviete Vīrietis
  20.91933 24.69548
> |
```

Līdzīgi, lai aprēķinātu *sieviešu* un *vīriešu* vecuma mediānu, var izmantot šo komandu:

```
> tapply Dati$Vecums, Dati$Dzimums, FUN=median)
  Sieviete Vīrietis
  42          53
> |
```

Aprēķināt *sieviešu* un *vīriešu* vecuma kvartīles var šādi:

```
> tapply(Dati$Vecums, Dati$Dzimums, FUN=quantile)
$Sieviete
0% 25% 50% 75% 100%
14  36  42  63  79

$Vīrietis
0%    25%    50%    75%   100%
22.00 36.25 53.00 71.25 86.00
> |
```

Lai aprēķinātu vidējo vērtību, piemēram, pazīmei *vecums*, ar šķēlumu *dzimums* un *vecgrupa*, tad izmantojam funkciju tapply(), kurā vēlamo datu iedalījumu norāda ar funkciju list() , kuras nozīme ir veidot sarakstu:

```
> tapply(Dati$Vecums, list Dati$Dzimums, Dati$Vecgrupa), FUN=mean)
  < 40 gadi no 40 līdz 59 gadi
  Sieviete    22.33333       42
  Vīrietis    28.00000       43
  vairāk nekā 60 gadi
  Sieviete    67.60000
  Vīrietis    74.33333
  > |
```

Iepriekšējā piemērā labi noderētu funkcija with(), lai samazinātu izpildāmās komandas garumu un būtu vieglāk lasāma izpildāmā komanda:

```
> with Dati, tapply(Vecums, list(Dzimums, Vecgrupa), FUN=mean))
  < 40 gadi no 40 līdz 59 gadi
  Sieviete    22.33333       42
  Vīrietis    28.00000       43
  vairāk nekā 60 gadi
  Sieviete    67.60000
  Vīrietis    74.33333
  > |
```

Ja ir nepieciešams aprēķināt standartkļūdu, tad var izmantot kādu no *R* programmas pakotnēm, vai arī pašam izveidot funkciju un jebkurā brīdī to pielietot ar nepieciešamo argumentu. Piemēram, šajā gadījumā tiek izveidota funkcija ar nosaukumu stdKluda(), kura aprēķina izteiksmi:

```
> stdKluda <- function(x){
+ return (sd(x)/sqrt(length x)))}
> stdKluda(Dati$Vecums)
[1] 5.2645
> |
```

Izveidoto funkciju, kas aprēķina standartkļūdu, var izmantot turpmākos aprēķinos, piemēram, aprēķinot *sieviešu* un *vīriešu* vidējā vecuma standartkļūdu:

```
> tapply(Dati$Vecums, Dati$Dzimums, FUN=stdKluda)
  Sieviete    Vīrietis
  6.307414    10.081887
> |
```

Kopumā ir iespējams izveidot funkciju, kas vienlaidus vienā teikumā automātiski sniegs nepieciešamo informāciju atskaites veidā un varēs izmantot to atkārtoti pēc nepieciešamības jebkurā brīdī, piemēram, izveidojam funkciju ar nosaukumu VecumaApraksts():

```
VecumaApraksts <- function(x)
{kopa <- (paste("Pētījumā bija
iekļauti",
    length(x), "respondenti,",
    "to minimālais vecums
    bija", min(x), "gadi,
    maksimālais", max (x),
    "gadi, vecuma amplitūda",
    max(x)-min(x), "gadi.
    Respondentu vidējais vecums
    bija",
round(mean (x),2), "gadi."))
return <- (print(kopa))}
```

Izpildot šo funkciju, rezultātā iegūs:

```
> vecumaApraksts(Dati
$Vecums)
[1] "Pētījumā bija ieklauti
17 respondenti, to minimālais
vecums bija 14 gadi,
maksimālais 86 gadi, vecuma
amplitūda 72 gadi. Respondentu
vidējais vecums bija 50.18
gadi."
> |
```

Līdzīgi var veidot sev piemērotas datu analīzes funkcijas, kurās kā argumentu var norādīt dažādas pazīmes un vērtības. Šajā pavisam vienkāršajā piemērā, balstoties uz klasisko nulles hipotēzes nozīmīguma pārbaudi (angl., *NHST*) ir izveidota funkcija AtkarigoIzlasuTtests(), kurā kā argumentus norāda divas pētāmās pazīmes un nozīmīguma līmeni. Rezultātā iegūst informāciju par datu normālsadalījuma esamību un atkarīgo izlašu t–testa rezultātu:

```
> AtkarigoizlasuTtests <-
function(x,y,alpha) {
  + shap_px <- shapiro.test
(x[0:5000])$p.value
  + shap_py <- shapiro.test
(y[0:5000])$p.value
  + p <- t.test(x,y, paired =
TRUE)$p.value
  + if(all(c(shap_px ,shap_py)
>=alpha))
  + {txtsh <- "atbilst"}
  + else
  + {txtsh <- "neatbilst"}
  + if(p>=alpha)
  + {txtt <- "nav"}
  + else
  + {txtt <- "i r"}
  + kopa <- paste("Dati",
txtsh, "normālsadalījumam.",
"Dati", txtt,
  + "statistiski ticami
atšķirīgi, jo",
  + round (p,2), "<", alpha)
  + return((kopa))
  + }
  > |
```

Izsaucot mūsu izveidoto funkciju AtkarigoIzlasuTtests() ar nepieciešamajiem argumentiem, iegūs gatavu rezultātu:

```
> AtkarigoizlasuTtests
Dati$HbAlcPirms, Dati$HbAlcPec,
alpha = 0.05)
[1] "Dati atbilst
normālsadalījumam. Dati ir
statistiski ticami atšķirīgi,
jo 0.03 < 0.05"
> |
```

Šādas un līdzīgas funkcijas datu analīzei var izveidot katrs pats un izmantot jebkuru savu datu analīzei.

## Datu apstrāde *Microsoft Word* vidē

Izmantojot papildus *R* bibliotēkas (gandrīz visu iespējamo bibliotēku sarakstu var apskatīt mājaslapā https://cran.r-project.org/web/packages/available_packages_by_name.html), ir iespējams jau automatizēt kādu daļu no veicamās analīzes, izveidojot funkcijas, kas *Microsoft Word/ Powerpoint* dokumentā automātiski ģenerēs aprēķinātos rezultātus, formatēs atskaites/ publikācijas tekstu un vizualizēs datus ar nepieciešamajām diagrammām.

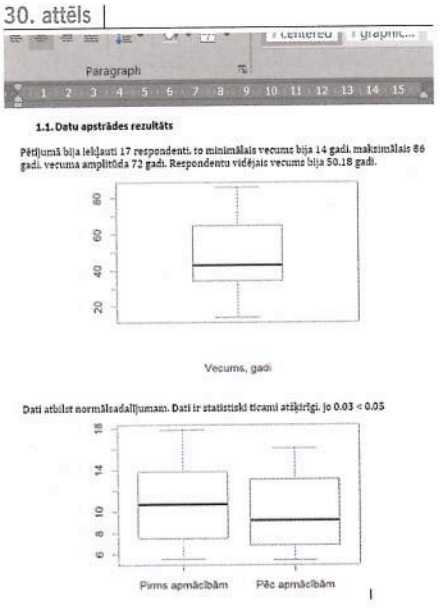
Piemēram, šādas komandas (*skat. 29. attēlu*).

Automātiski izveidos *Microsoft Word* veida dokumentu ar nosaukumu *Apstrades_Dokuments.docx*, kurā būs gan datu apstrādes rezultāts, gan attēli (*skat. 30. attēlu*).

29. attēls |

```
library(officer)
library(magrittr)

picHbAlc <- tempfile(fileext = ".png")
png(filename = picHbAlc, width = 5, height = 4, units = 'in', res = 150)
with(Dati,boxplot(HbAlcPirms, HbAlcPec, names=c("Pirms apmācībām","Pēc apmācībām")))
dev.off()

picVecums <- tempfile(fileext = ".png")
png(filename = picVecums, width = 5, height = 4, units = 'in', res = 150)
with(Dati, (boxplot(Vecums,xlab="Vecums, gadi")))
dev.off()

trezult <- AtkarigoIzlasuTtests(Dati$HbAlcPirms, Dati$HbAlcPec, alpha = 0.05)
vecapr <- VecumaApraksts(Dati$Vecums)

dokuments <- read_docx()
#styles_info(dokuments)

dokuments <- dokuments %>%

  body_add_par("Datu apstrādes rezultāts", style = "heading 2") %>%
  body_add_par("", style = "Normal") %>%
  body_add_par(vecapr, style = "Normal") %>%
  body_add_img(src = picVecums, width = 5, height = 4, style = "centered") %>%
body_add_par(trezult, style = "Normal") %>%
body_add_img(src = picHbAlc, width = 5, height = 4, style = "centered")

print(dokuments, target = "Apstrades_Dokuments.docx")
```

30. attēls |



## *R* programmas skripti

Lai gan programmā *R* ir iespējams izpildīt komandas no konsoles vienu pēc otras, tomēr *R* viena no spēcīgām pusēm ir tā saukto skriptu izmantošana, jeb – vienā atsevišķā failā lietotājs var ierakstīt pilnīgi visas izpildāmās darbības un izsaucamās funkcijas un šo skriptu atkārtot jebkurā brīdī neierobežotu skaitu reižu.

Kā redzams attēlā, visas iepriekšrakstītās komandas datu ievadīšanai, kodēšanai un analīzei ir ierakstītas vienuviet. Un, ja jaunā datu apstrādē ir nepieciešams veikt darbības, kuras ir diezgan līdzīgas tām, kuras jau ir bijušas, tad samērā ātri ir iespējams izveidot jaunu skriptu, rediģējot esošos. Tāpat var izveidot skriptu, kurā var sarakstīt dažādas funkcijas datu analīzei un vizualizēšanai.

Skripta rindā aiz simbola # var ierakstīt komentārus brīvā veidā, kuri netiek izpildīti. Un pēc pogas *Run* nospiešanas iezīmētajās rindās esošās komandas tiks izpildītas. Tas ir īpaši noderīgi, ja analizējamie dati tiek laiku pa laikam papildināti un tad nav nepieciešamība rakstīt komandas katru reizi, kad vien ir nepieciešams veikt analīzi (*skat. 31. attēlu*).

***Piezīme:*** skripti nav tas pats, kas *R* programmas bibliotēkas.

31. attēls

skriptsrakstam.R* | Dati

Source on Save | Run | Source

```
1  #Ielasīt analizējamos datus
2  require("readxl")
3  library(readxl)
4  #Dati<- read_excel("Dati.xlsx")
5  Dati <- read_excel(file.choose())
6  View(Dati)
7
8  #Iekodēt sievietes un vīriešus
9  Dati$Dzimums=factor(Dati$Dzimums,labels=c("Sieviete","Vīrietis"),
10                      ordered = FALSE)
11
12 #Aprēķināt HbA1c starpību
13 Dati["HbA1cStarpiba"] <- Dati$HbA1cPirms-Dati$HbA1cPec
14
15 Dati["Vecgrupa"]=NA #Izveidot jaunu kolonu
16
17 #Iedalīt Vecumu trīs grupās
18 Dati$Vecgrupa[Dati$Vecums<40]=1
19 Dati$Vecgrupa[Dati$Vecums>=40 & Dati$Vecums<60]=2
```

## Nobeigums

Šajā rakstā par programmas *R* izmantošanu, vienkāršā veidā parādītas pamatdarbības, kuras praktiski jebkurš lietotājs var izmantot savu datu pamata apstrādei. Un, cerams, ka šis nelielais ievads spēs ieinteresēt lasītājus uzsākt patstāvīgu programmas *R* turpmāko apguvi.