Praktická časť diplomovej práce

1 Sprievodca ekfmetriou

Sprievodca ekonometriou má za úlohu priblížiť Vám ekonometriu, a pomôcť Vám jej porozumieť. Sprievodcu píšem ako študent, ktorý sa ekonometriu začal učiť sám, a sám si prešiel zdĺhavým procesom bádania a usmerňovania. Sprievodca je zostrojený ako-tak súbežne s osnovou a zadaniami, ktoré obdržíte na hodine. Nebudeme sa konkrétne držať vypracovania zadaní, ale skôr princípmi, z ktorých zadania ťažia. Mnoho študentov tento predmet nezaujíma, a zadania vypracujú okopírovaním postupov starších spolužiakov, nuž, pochopiť ekonometriu a jej postupy nie je vôbec jednoduché, a dokážem pochopiť, keď si študenti hľadajú skratky. Na druhú stranu, ekonometria predstavuje skvelú vstupnú bránu do sveta analytiky. Človek je zavalený Machine Learningom, Data Sciencom a AI-čkom, nuž až po sfúknutí pozlátka zistí, že je to zmes matematiky, štatistiky a počítačovej vedy (CS – computer science). Ekonometria je teda skvelou výhovorkou, ako oprášiť matematiku, doučiť sa štatistiku, a naučiť sa troška programovania. R-ko sa môže zdať ako jazyk, ktorý žije v tieni Pythonu, avšak, akoby nejedna Dominika vedela povedať, netreba sa nechať voviesť do omylu. R-ko je najvhodnejší programovací jazyk pre štatistikov, Google ho zahrnul do najnovších kurzov Google Analytics. Mojou úlohou je pomôcť Vám prekonať problém, ktorý som na začiatku svojej cesty ekonometriou vôbec nepovažoval za problém, a to množstvo materiálov, ktoré zavalí študenta. Pomôžem Vám postupne zložiť skladačku konceptov a teórií, na ktorých ekonometria stojí. Náročnosť prezentovania konceptov bude prispôsobená. Nemá zmysel vysvetľovať odvodzovanie každého estimátora. Cieľom je poskytnúť všeobecný náhľad ekonometrie, a pomôcť Vám pochopiť, a teda príjemnejšie zvládnuť predmet Ekonometria.

Čo nás čaká:

- Základy programovania v R
- Intuitívny prehľad štatistických konceptov
- Ekonometrické techniky

2 Základy programovania v R

Sprievodca je interaktívny, teda začneme stiahnutím a inštaláciou R a RStudia. R má samo o sebe programovacie prostredie, avšak dnešným štandardom je používanie intregrovaného vývojového prostredia (IDE) v podobe RStudia.

2.1 Aritmetické operátory

Poďme teda rovno na vec. Začneme základnými funkciami. R môžeme používať ako kalkulačku, teda za pomoci klasických aritmetických operátorov môžeme sčítať, odčítať, násobiť, deliť či umocňovať:

```
5 + 5
```

```
## [1] 10
```

```
5 - 5
```

```
## [1] 0
```

```
5 * 5
```

```
## [1] 25
```

```
5 / 5
```

```
## [1] 1
```

```
5^2
```

```
## [1] 25
```

R-ko dokáže používať aj ďalšie aritmetické operátory:

```
# zobrazí zvyšok z delenia

5 %% 2
```

```
## [1] 1
```

My sa budeme zapodievať len tým, s čím sa na cvičeniach stretneme. Našou úlohou nie je naučiť sa dokonalo ovládať R, ale naučiť sa používať ho v dostatočnej miere, aby sme s ním zvládli to, čo budeme v najbližšej dobe potrebovať.

2.2 Balíky

Okrem základných operátorov budeme využívať aj funkcie:

```
mean(2, 4, 6)
```

```
## [1] 2
```

```
abs(-5)
```

```
## [1] 5
```

```
sqrt(8)
```

```
## [1] 2.828427
```

Tieto funkcie sa nachádzajú v balíkoch, ktoré si môžeme predstaviť ako také Addony. R figuruje balíkmi, ktoré sú predinštalované, a zahŕňajú najpoužívanejšie a najzákladnejšie funkcie. Vyššie použité funkcie sa nachádzajú v balíku base. To, v akom balíku sa funkcia nachádza zistíte po napísaní funkcie

to však len zapredpokladu, že už máte balík nainštalovaný. Ak narazíte na názov funkcie, ktorú chcete použiť, avšak nemáte nainštalovaný balík a chcete zistiť jeho názov, buď si funkciu zadajte do Google, alebo napíšte do konzoly:

```
# ??meno funkcie

??mean
```

My budeme často využívať predinštalované balíky **base** a **stats**, avšak za pochodu si budeme inštalovať aj ďalšie balíky, s ktorými sa na cvičeniach stretnete. Nový balík je potrebné prv nainštalovať a potom ho načítať do prostredia.

```
# stiahneme a našintalujeme pomocou R konzoly a funkcie install.packages()
# ! názov balíka je citlivý na veľkosť písma
# ! názov musí byť v úvodzovkách

install.packages("fBasics")

# po nainštalovaní máme balík stiahnutý v našom PC, a tento príkaz už viac nepoužívame
# ak však chceme funkcie z balíka použiť, musíme po zapnutí R-ka balík načítať príkazom

library(fBasics)

# ! tu už úvodzovky nie sú potrebné
```

> *Pri inštalovaní názov balíka zabalíme do úvodzoviek. Pri jeho načítaní pomocou library() už úvodzovky nepíšeme. Na pohovor si vezmeme oblek (úvodzovky), ale po prijatí už chodíme do práce bez obleku.*

2.3 Objekty

Často chceme vyrátané výsledky znova použiť, a preto by bolo vhodné si ich niekde uložiť. Na ukladanie a uskladnenie výsledkov slúžia objekty. Každý objekt má meno a obsah. Meno si môžeme zadať akékoľvek, musí však:

- začínať malým alebo veľkým písmenom a nie číslom
- obsahovať iba čísla, písmená alebo niektoré špeciálne znaky ako "." či "_".

Nezabúdajme, že R je case sensitive (rozlišuje veľké a malé písmená).

Povedzme že chceme vytvoriť objekt **a** a priradiť mu hodnotu $2 + 2$. Na priradenie obsahu je možné použiť " = ", avšak štandardom je používanie " < − ".

*Znak <- nie je nutné písať dvoma znakmi, používa sa na to skratka **"ľavý Alt" a "-"**. Na SK klávesnici nájdeme znak naľavo od pravého Shiftu, na EN klávesnici zvyčajne naľavo od Backspacu. Osobne programujem s EN klávesnicou, nech som použíteľný v akomkoľvek štáte bez potreby inštalovať SK klávesnicu.*

```
# Priradíme teda objektu "a" výsledok "2 + 2".

a <- 2 + 2

# Po napísaní názvu objektu do konzoly nám konzola ukáže už len výsledok.

a
```

```
## [1] 4
```

```
# Nové priradenie hodnoty starému objektu prepíše starú hodnotu.

a <- 5 + 5

a
```

```
## [1] 10
```

S daným objektom môžeme pracovať, akoby to bola číselná hodnota.

```
a^2
```

```
## [1] 100
```

2.4 Vektory

Pre priradenie viacerých hodnôt vytvoríme vektor. Vektor vytvoríme pomocou funkcie c() ako combine.

```
# Objektu "a" priradíme súbor číselných hodnôt a vytvoríme z neho vektor.

a <- c(5, 1, 4, 2, 3)

a
```

```
## [1] 5 1 4 2 3
```

S vektormi dokážeme rôzne pracovať. Sčítavať, násobiť, aplikovať na ne funkcie, všetko, čo nás napadne. Ups, čo nám napadne!

```
b <- c(5, 9, 6, 8, 7)
a + b
```

```
## [1] 10 10 10 10 10
```

```
min(a)
```

```
## [1] 1
```

```
sort(a)
```

```
## [1] 1 2 3 4 5
```

```
sum(a)
```

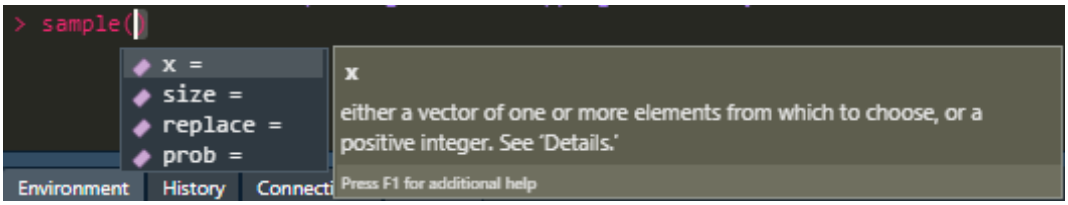
```
## [1] 15
```

Je vhodné poznať pár špeciálnych funkcií na tvorbu vektorov. A to:

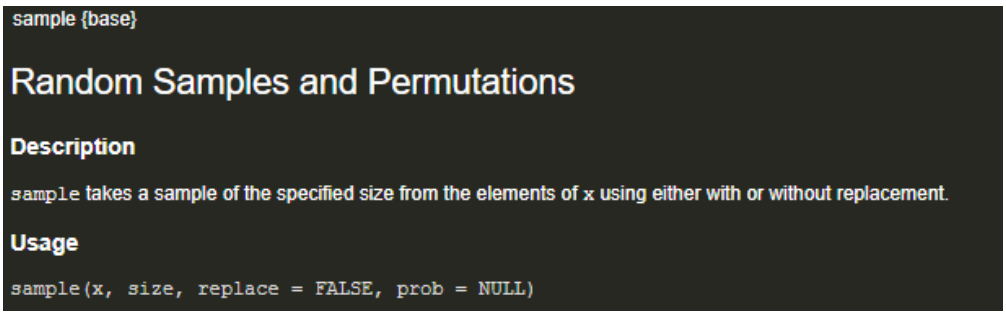
```
# sample(), seq() a rep()
```

Často si totižto potrebujete vytvoriť vektor hodnôt podľa vlastnej potreby. Hm, to som veľmi nič nové nepovedal. Ale... Čo vlastne tieto funkcie robia? A ako to zistíme? Ak pracujete s úplne novou funkciou, a máte už nainštalovaný balík, a taktiež ste ho už načítali do knižnice cez *library*(), je vhodné použiť ?*názovfunkcie* alebo *help*(*názovfunkcie*). Poskytne Vám to stručný popis, čo funkcia robí, všetky jej argumeny a aj pár príkladov. Ak už ste však s funkciou pracovali, ale nepamätáte si argumenty, rozpíšte funkciu, kliknite sa do zátvoriek funkcie *mean*(*tu.sa.kliknete*), a stlačíte *TAB*. Vyhodí Vám to argumenty funkcie. Skúsime si to na funkcii *sample*().

2.4.1 sample()



Pri jednej z vecí, ktorú Vám chcem ukázať, budeme potrebovať vektor výšky ľudí. Tak si ho teda vytvorme už teraz. Ešte predtým môžeme skúsiť napísať do konzoly ?*sample*, čím zistíme, že funkcia *sample*() nám náhodne vyberie hodnoty zo zadaného vektora.




```
# Ako prvé potrebujeme zadať "x", teda vektor hodnôt, z ktorých chceme vyberať.
# Pre nás to budú hodnoty medzi 160 až 190, ktoré navolíme ako 160:190.

# 160:190 jednoducho vypíše hodnoty zaradom od 160 do 190. A sample() z nich náhodne vyberie
# nami určený počet hodnôt.


# "size" je počet hodnôt, aký ma funkcia vybrať.

# "replace", či môže vybrať jednu hodnotu viackrát, alebo ju má vylúčiť.

# "prob" použijeme, ak chceme priradiť istým hodnotám inú váhu.

vyska <- sample(x = 160:190, size = 10, replace = TRUE)

print(vyska)
```

```
##  [1] 167 180 189 173 167 183 178 168 185 169
```

> *Funkcia by fungovala, aj keby sme to napísali ako* "sample(160:190, 10, TRUE)". *Je však vhodné písať aj argumenty. Hlavne pri funkciách, ktoré nie sú veľmi bežné. Ak po Vás niekto bude čítať kód, číta sa to lepšie.*

2.4.2 seq()

Funkcia *seq*() vygeneruje sekvenciu čísel. Ako argumenty zadávame buď *od*, *do* a *by*. Teda po akých inkrementoch bude daná sekvencia narastať.

```
seq(from = 2, to = 12, by = 0.5)
```

```
##  [1]  2.0  2.5  3.0  3.5  4.0  4.5  5.0  5.5  6.0  6.5  7.0  7.5  8.0  8.5  9.0
## [16]  9.5 10.0 10.5 11.0 11.5 12.0
```

> *V Rku používame na oddeľovanie desatinných miest bodku. Čiarkou oddeľujeme argumenty. Taktiež nezabúdajte používať medzery. Jedná sa o gramatiku programovania. Predsalen náš vyššie napísaný výraz vyzerá lepšie ako* seq(from=2,to=12,by=0.5).

```
# Zadáme "od", "do" a koľko častí má vektor mať.

seq(from = 4, to = 10, length = 4)
```

```
## [1]  4  6  8 10
```

```
seq(from = 4, to = 10, length = 8)
```

```
## [1]  4.000000  4.857143  5.714286  6.571429  7.428571  8.285714  9.142857
## [8] 10.000000
```

```
# Alternatívou je zadať "od", "po", a rozdeliť to po požadovaných kúskoch
# na požadovanú dĺžku.

seq(from = 4, by = 0.5, length = 25)
```

```
##  [1]  4.0  4.5  5.0  5.5  6.0  6.5  7.0  7.5  8.0  8.5  9.0  9.5 10.0 10.5 11.0
## [16] 11.5 12.0 12.5 13.0 13.5 14.0 14.5 15.0 15.5 16.0
```

2.4.3 rep()

Funkcie *rep*() jednoducho zopakuje zadané číslo, alebo vektor, x-krát.

```
rep(x = 1, times = 10)
```

```
##  [1] 1 1 1 1 1 1 1 1 1 1
```

```
rep(x = c(1, 2, 3), times = 3)
```

```
## [1] 1 2 3 1 2 3 1 2 3
```

2.5 Indexovanie

Indexovanie je v R-ku veľmi užitočný spôsob selektovania dát. Ide jednoducho o výber súboru dát, zo súboru dát. Indexujeme za použitia hranatých zátvoriek, ktoré bez medzery nalepíme k objektu, z ktorého chceme dáta vybrať. Najjednoduchšie sa to vysvetľuje ukážkou. A nezabúdajte, že R-ko začína od jednotky, nie od nuly. Aj keď nám, ekonómom neprogramátorom to asi ani nepríde divné.

```
# Vytvorme si obyčajný číselný vektor.

obycajny_vektor <- c(1:10)

obycajny_vektor
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
# Za použitia indexovania môžeme vybrať akúkoľvek hodnotu. Chceme vybrať prvú.

obycajny_vektor[1]
```

```
## [1] 1
```

```
# Alebo súbor hodnôt. Vyberieme prvú a poslednú hodnotu.

obycajny_vektor[c(1, 10)]
```

```
## [1]  1 10
```

Pravdepodobne by väčšina z vás napísala $obycajny\_vektor[1, 10]$, to by vám však vyhodilo chybu. Prečo je tomu tak plne pochopíte, keď si ukážeme matice. Aj keď to nie je nič zložité. Vektor si predstavme ako šípku, ktorá určuje smer. Je to teda, v našom prípade, reťazec čísel, jedna dlhá šnúra, ktorá nemá žiadne riadky ani stĺpce. R-ko však to, čo napíšeme do hranatých zátvoriek vníma ako:

$$[riadok, stĺpec]$$

$$[row, column]$$

> *Zo začiatku sa mi zvyklo pliesť, čo ide prvé. Zapamätal som si to ako RC autíčko. Také tie malé na ovládanie.*

Čiže prvý údaj predstavuje riadok, a druhý údaj stĺpec. Ak by sme napísali $[1, 10]$, R-ko by hľadalo v prvom riadku desiatu hodnotu. Do hranatých zátvoriek píšeme vlastne **súradnice**. V reťazci hodnôt máme ale iba reťazec hodnôt. (:D) Preto je potrebné použiť funkciu $c()$, aby R-ko vedelo, že má vyberať z reťazca hodnôt. Indexovanie je veeeľmi užitočné, a dá sa využiť veľmi kreatívne. Nám však stačí vedieť, čo to plus-mínus robí. Aby ste vedeli, čo sa deje, keď uvidíte hranaté zátvorky. O indexovaní si ešte povieme pri maticiach.

```
# Indexovaním môžeme aj odobrať hodnotu. Napr., ak chceme všetky okrem poslednej:

obycajny_vektor[-10]
```

```
## [1] 1 2 3 4 5 6 7 8 9
```

2.6 Iné typy vektorov

Vektory nemusia obsahovať len čísla. Môžu obsahovať napríklad aj **textové** alebo **logické** premenné. Existuje ešte aj štvrtý typ, **faktorové** vektory, ktorým sa ale nebudeme zaoberať.

2.6.1 Vektor textových premenných

```
# Pre vytvorenie vektora obsahujúceho textové reťazce, musí byť obsah ohraničený
# úvodzovkami "".

vector_characters <- c("c", "musíme", "používať", "stále", "ak", "chceme",
                       "viac", "hodnôt")

vector_characters
```

```
## [1] "c"        "musíme"   "používat" "stále"    "ak"       "chceme"   "viac"
## [8] "hodnôt"
```

Vektor tvorený textovým reťazcom nájde svoje uplatnenie napríklad pri pomenovaní hodnôt.

```
nazvy <- c("prvy", "druhy", "treti")
cisla <- c(1, 2, 3)

# Použijeme na to funkciu names()

names(cisla) <- nazvy

# Ak sa teraz pozrieme na vektor "cisla", uvidíme, že sme číslam priradili názvy.
# Na vypísanie výsledku môžeme použíť aj funkciu print().

print(cisla)
```

```
##  prvy druhy treti
##     1     2     3
```

Vektor **"cisla"** sme vlozili do funkcie **names()**. Aplikovali sme funkciu na vektor, ktorému sme chceli priradiť názvy. **Priradiť**, teda symbol priradenia **<-**, potom už len vektor s názvami, ktoré chceme priradiť. Pre lepšiu ilustráciu si to napíšeme nanovo, bez zadefinovaného vektora "nazvy".

```
names(cisla) <- c("adin", "dos", "tres")

print(cisla)
```

```
## adin  dos tres
##    1    2    3
```

2.6.2 Vektor logických premenných

```
# Logické operátory, inak známe ako Booleovské operátory, nám ako výsledok
# poskytnú výstup v podobe TRUE alebo FALSE.
# ! Pre overenie rovnosti použijeme "==".

vektor <-  5

vektor == 5
```

```
## [1] TRUE
```

```
vektor == 6
```

```
## [1] FALSE
```

Logický operátor	Popis
<	menšie než
<=	menšie alebo rovné
>	väčšie než
>=	väčšie alebo rovné
==	rovná sa
!=	nerovná sa
!x	nie je x
x	y
x & y	x a y
isTRUE(x)	test či je x pravdivé

Logické operátory sa zídu pri indexovaní, alebo pri zisťovaní počtu vyhovujúcich hodnôt.

```
# Vektor výšky ľudí, ktorý sme si skôr vytvorili.

vyska <- sample(x = 160:190, size = 10, replace = TRUE)

# Použitie logického operátora na zistenie, kto má viac ako 170cm.

viac_ako_170 <- vyska > 170

# Výsledky však nebudú číselnými hodnotami, ale hodnotami booleovského typu.

print(viac_ako_170)
```

```
##  [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE
```

```r
# To nám však nebráni zužiťkovať to pomocou funkcie "sum()" a zistiť počet
# vyhovujúcich hodnôt. Zráta to všetky TRUE hodnoty.

sum(viac_ako_170)
```

```
## [1] 9
```

```r
# Čo dokážeme pomocou indexovania pretvoriť na číselné hodnoty.

vyska_v_cm <- vyska[vyska > 170]

print(vyska_v_cm)
```

```
## [1] 174 174 180 185 175 175 181 185 178
```

2.7 Matice

Matíc sa netreba ľakať. Osobne som mal (vraj mal) v maticiach isté medzery, a z mojich skúsenosti nie som jediný študent ekonómie s týmto nedostatkom, nedostatkom vedomostí. Možno sa momentálne venujú maticiam na predmete Matematika viac, nuž, aby som prešiel k veci, pre zvládnutie základov ekonometrie nepotrebujete absolútne vedomosti matíc. Ono, matica je len akási množina čísel usporiadaná do riadkov a stĺpcov (rc, spomínate?), plus sa na ňu vzťahujú nejaké vlastnosti. Nejaké dosť podstatné vlastnosti. Ako som ale vravel, netreba sa ľakať. My použijeme matice, okrem iného, na počítanie Beta estimátorov v regresií by hand, teda ručný výpočet nejakých hodnôt. Ak ste už zo štatistiky zabudli, čo je regresia, tiež nevadí. Čo je regresia a prečo používame matice si vysvetlíme neskôr. Teraz sa ich naučíme zostrojiť, a vysvetlíme si pár **pojmov** a **vlastností** týkajúcich sa matíc, s ktorými sa na hodinách stretnete.

2.7.1 Spôsoby vytvárania matice

V aplikovanej ekonometrií sa matice väčšinou vytvárajú z existujúcich datasetov. Vo všeobecnosti však máme tri možné spôsoby vytvárania matíc v R. A to pomocou:

1. funkcie *matrix*(),
2. funkcie *rbind*(),
3. funkcie *cbind*().

```r
# Pri funkcií matrix() zadáme vektor, a argumenty v podobe počtu riadkov, stĺpcov,
# a či má byť vektor usporiadaný po riadkoch alebo nie po riadkoch.

vektor <- c(1, 2, 3, 4, 5, 6, 7, 8, 9)

# Vytvoríme si štvorcovú maticu 3x3

matica1 <- matrix(vektor, nrow = 3, ncol = 3, byrow = TRUE)

matica1
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
## [3,]    7    8    9
```

```r
# Zmeníme usporiadanie na FALSE, takže bude matica usporiadaná po stĺpcoch.

matica2 <- matrix(vektor, nrow = 3, ncol = 3, byrow = FALSE)

matica2
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

Ďalšie dve funkcie fungujú na princípe zlepenia riadkov alebo stĺpcov dohromady. Sú to intuitívne funkcie. Keďže bind znamená v preklade spájať. Teda row bind, spájanie riadkov, a column bind ako spájanie stĺpcov.

```r
# Potrebujeme si vytvoriť vektory, ktoré budeme chcieť zlepiť.
# Vektor c(1:3) bude taký istý ako c(1, 2, 3).

vektor1 <- c(1:3)
vektor2 <- c(4, 5, 6)
vektor3 <- c(7, 8, 9)

# Zviazanie po riadkoch.

matica_riadky <- rbind(vektor1, vektor2, vektor3)

matica_riadky
```

```
##         [,1] [,2] [,3]
## vektor1    1    2    3
## vektor2    4    5    6
## vektor3    7    8    9
```

```r
matica_stlpce <- cbind(vektor1, vektor2, vektor3)

matica_stlpce
```

```
##      vektor1 vektor2 vektor3
## [1,]       1       4       7
## [2,]       2       5       8
## [3,]       3       6       9
```

2.7.2 Indexovanie

Indexovanie matíc je veľmi intuitívne. Do hranatých zátvoriek zadáme ako prvú hodnotu riadok, z ktorého chceme extrahovať hodnotu, a ako druhú súradnicu zadáme stĺpec. Ak chceme vybrať celý riadok, zadáme len prvú hodnotu, a druhú necháme prázdnu. Pri výbere celého stĺpca to funguje presne naopak.

```r
# Budeme pracovať s vyššie vytvorenou maticou "matica_riadky".

matica_riadky[1, 3] # jeden prvok, prvý riadok, tretí stĺpec
```

```
## vektor1
##       3
```

```r
matica_riadky[1, ] # celý prvý riadok
```

```
## [1] 1 2 3
```

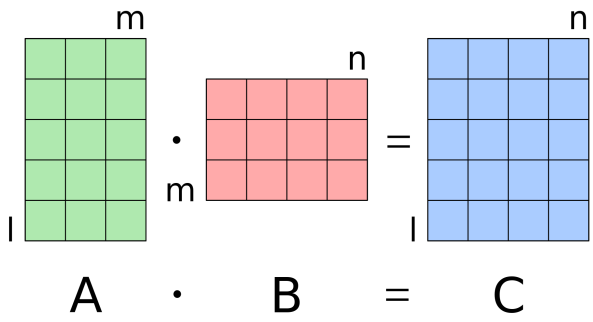
```r
matica_riadky[ , 1] # celý prvý stĺpec
```

```
## vektor1 vektor2 vektor3
##       1       4       7
```

2.7.3 Násobenie, sčítanie a odčítanie matíc

Matice majú pĺno pravidiel. My si prejdeme len tie, na ktoré na cvičeniach narazíme.

- Sčítavať a odčítavať môžeme iba matice, ktoré majú rovnaký počet riadkov a stĺpcov. Ako pri klasickom odčítaní neplatí, že: $A - B = B - A$.
- Pri násobení matíc musí platiť, že počet stĺpcov matice A musí byť zhodný s počtom riadkov matice B.
  - Výsledkom je potom matica, ktorá ma počet **r**iadkov ako prvá matica a počet stĺp**c**ov ako druhá matica, **rc** ;).



Pri násobení matice skalárom, aka. jedným číslom, použijeme ako operátor klasickú hviezdičku. Každý prvok matice bude prenásobený určeným číslom.

```r
matica_riadky * 5
```

```
##         [,1] [,2] [,3]
## vektor1    5   10   15
## vektor2   20   25   30
## vektor3   35   40   45
```

Pri násobení dvoch matíc sa používa trocha netradičný operátor $\% * \%$.

```
matica_riadky %*% matica_stlpce
```

```
##         vektor1 vektor2 vektor3
## vektor1      14      32      50
## vektor2      32      77     122
## vektor3      50     122     194
```

Sčítanie matíc.

```
matica_riadky + matica_stlpce
```

```
##         [,1] [,2] [,3]
## vektor1    2    6   10
## vektor2    6   10   14
## vektor3   10   14   18
```

2.7.4 Transpozícia matice

Transponovaním matice dôjde k vzájomnej výmene riadkov a stĺpcov matice. Takúto maticu označujeme ako A^T. Ak bola prvotná matica (m, n), po transpozícií vznikne matica s rozmermi (n, m). V R-ku matice transponujeme pomocou funkcie t() ako transpose.

```
matica_riadky
```

```
##         [,1] [,2] [,3]
## vektor1    1    2    3
## vektor2    4    5    6
## vektor3    7    8    9
```

```
t(matica_riadky)
```

```
##      vektor1 vektor2 vektor3
## [1,]       1       4       7
## [2,]       2       5       8
## [3,]       3       6       9
```

2.7.5 Hodnosť (rank) matice

V zadaniach od vás bude požadované vyrátať hodnosť matice, čo sa zvykne označovať aj ako rank matice. Hodnosť matice je maximálny počet lineárne nezávislých riadkov, alebo stĺpcov, v matici. Dva vektory $\overrightarrow{a}$, $\overrightarrow{b}$ nazývame lineárne závislé vektory práve vtedy, ak existuje reálne číslo k také, že platí:

$$\overrightarrow{b} = k\overrightarrow{a}$$

Ak táto rovnosť neplatí, vektory sú lineárne nezávislé. Keď sme spomínali maximálny počet buď riadkov alebo stĺpcov, mysleli sme tým, že ak nepracujeme so štvorcovou maticou, maximálna hodnosť môže byť najviac rovná tomu, čoho je menej. Ak máme maticu 3x4, jej maximálna hodnosť môže byť 3. Lebo riadkov máme menej. Ak by sme mali maticu 4x2, maximálna hodnosť môže byť 2. Hodnosť matice sa v R-ku vyráta pomocou funkcie *qr*().

```
# Predpokladajme, že pracujeme so štvorcovou maticou.
# Ak by ste mali overiť, či sú vektory lineárne nezávislé,
# všetky ich spojíme do matice, a vyrátame rank matice. Ak bude rank
# rovný počtu vektorov, vektory sú navzájom  lineárne nezávislé.

vektor1 <- c(2, 3, 1, 9)
vektor2 <- c(1, 0, 3, 4)
vektor3 <- c(2, 9, 0, 3)
vektor4 <- c(4, 7, 2, 4)

matica <- rbind(vektor1, vektor2, vektor3, vektor4)


# Operátor "$" dokáže extrahovať konkrétnu hodnotu, ktorú chceme extrahovať. Skúste použiť
# príkaz "qr(matica)", ktorý nám vyráta ešte pár ďalších vecí. Nás však zaujíma iba rank.

qr(matica)$rank # vyrátame hodnosť matice
```

```
## [1] 4
```

```
# Vidíme, že rank je rovný počtu riadkov/stĺpcov, teda naše vektory sú lineárne nezávislé.
```

2.7.6 Determinant matice

Determinant matice si môžeme predstaviť ako hodnotu, ktorá je priradená matici podľa toho, ako vyzerá. Matice môžu mať determinant nulový alebo nenulový. Mohli by sme to rátať ručne, necháme to však R-ko vyrátať za nás. Nás budú zaujímať aké vlastnosti sa s determinantom spájajú.

2.7.7 Inverzná matica

Inverzná matica je matica A^{-1}, ktorá nám po vynásobení pôvodnou maticou A, dá jednotkovú maticu. Funguje to aj naopak, teda platí vzťah:

$$A * A^{-1} = A^{-1} * A = E$$

Inverznú maticu vytvoríme pomocou funkcie *solve*()

2.7.8 Singulárna / Regulárna matica

Ak je matica regulárna, znamená to, že má inverziu. Ak je singulárna, nemá inverziu, teda A^{-1} neexistuje.

- Matica je singulárna ak:
  - má determinant rovný nule,
  - sa hodnost nerovná poctu riadkov (ak má napr. 3x3 matica hodnost 2)

- Matica je regulárna ak:
  - má nenulový determinant,
  - sa hodnost rovná poctu riadkov/stlpcov.

3 Práca s datasetmi a ich analýza

Vrhneme sa na:

- načítanie datasetov do R-ka a objektov,
- manipuláciu datasetov,
- vysvetlenie lineárneho regresného modelu,
- prácu s regresnými modelmi.

3.1 Načítanie dát

Načítať dáta je možné rôznymi spôsobmi. V okne *Environment* môžete kliknúť na *Import dataset* a vybrať typ súboru, aký chcete importovať. Sofistikovanejšie je však načítavanie údajov pomocou funkcií. Tých je tiež niekoľko, plus, existujú rôzne balíky, ktoré sú vyvinuté na zlepšenie práce s dátami a ich vizáže. Vám však bude stačíť jediná funkcia a to *read.csv*2. Predtým než funkciu použijeme, si však treba nastoliť isté štandardy. Ideálne je, aby ste mali vytvorenú zložku, v ktorej budete mať všetky datasety (excelovské súbory) a pekne oddelené zložky k cvičeniam. Nazývame to "working directory" AKA pracovný adresár. Na zistenie, kde je Váš momentálny pracovný adresár použijeme funkciu *getwd*() (get working directorty). Potrebujeme to vedieť preto, lebo do funkcie *read.csv*2 potrebujeme zadať argument umiestnenia súboru. A je ľahšie zadať:

```
read.csv2("udaje_o_pocte_kaciatok")

# než

read.csv2("C:/Desktop/MilanRozok/ekonometria/test/udaje_o_pocte_kaciatok.csv")
```

Určenie nového adresára je možné urobiť pomocou *setwd*() (set working directory), kde ako argument zadáme cestu do nového adresára, avšak, jednoduchšie je kliknúť vľavo hore na:

Session -> Set Working Directory -> Choose directory,

a vybrať si adresár manuálne. Všetky datasety potom môžeme načítať funkciou *read.csv*2() už len pomocou uvedenia názvu v úvodzovách, a nemusíme uvádzať celú cestu umiestnenia súboru.

3.1.1 read.csv2()

Súbory typu .CSV znamenajú doslovne *Comma – separated values*, teda hodnoty oddelené čiarkami. Keď sa bavíme o formáte .CSV predstavte si súbor, kde každá hodnota má svoj riadok, a každá premenná má svoj stĺpec. Ako hodnoty sa chápe oddelenie stlpcov, teda stĺpce sú väčšinou oddelené čiarkami. To je však taký teoretický prístup, v praxi môžu byť tieto hodnoty oddelené aj inými spôsobmi. Hlavné však je pozerať na koncovku súboru, resp. súbor (napr. z Excelu), uložiť ako .CSV súbor. Funkcie *read.csv*() a *read.csv*2() robia to isté, jediné v čom sa líšia je ich defaultné nastavenie. *read.csv*() ráta, že sa na oddelenie desatinných miest používa bodka (čo je také americké), a *read.csv*2(), používa na oddelenie desatinných miest čiarku (čo je také európske). To je dôvod, prečo primárne používame *read.csv*2().

3.1.2 Práca so vstavanými datasetmi

R-ko obsahuje vstavané datasety, ktoré si môžeme všetky vypísať pomocou:

```
data()
```

V tomto sprievodcovi budeme pracovať s dátami, ktoré si môžete načítať zo vstavaného balíka *datasets*. Je to z dôvodu, že je to proste jednoduché. Na hodinách budete pracovať s pravými ekonomickými datasetmi, avšak pre ukážku, ako čo funguje, a ako s čím súvisí nám postačia základné datasety, ktoré sú ľahké na pochopenie. Ak si budete chcieť overiť nejakú funkciu či teóriu, budete si môcť za pochodu načítať dataset, s ktorým ste oboznámený, a otestovať, čo potrebujete.

```
# Pre načítanie datasetu do objektu použijeme trocha nezvyčajný prístup, kde:
# datasets predstavuje názov balíku, a "mtcars" dataset, ktorý chceme sprístupniť.
# Pomocou "::" sprístupníme konkrétny objekt z balíka.

data <- datasets::mtcars

# Ak by sme datasetu nechceli priradiť vlastný názov, ale ponechať originálny:

data("mtcars")

# Funkcia bude fungovať aj bez úvodzoviek.
```

4 Jednoduchá lineárna regresia

Hlavným nástrojom ekonometrie je regresia. Cieľom regresie je zistiť ako určená/é nezávislé premenné, ovplyvňujú jednu závislú premennú. Ak Y je závislá a X nezávislá, tak regresujeme, robíme regresiu, Y na X. Čo však tá magická skrinka vlastne robí?

```
# Predpokladajme, že máme načítaný dataset "mtcars".
# Pomocou funkcie "head()" si načítame prvých 6 riadkov.
# Alternatíva je "tail()" (ako chvost), pre vypísanie posledných 6 riadkov.

head(data)
```

```
##                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

```
# Vidíme, že je to súbor aút, s rôznymi parametrami.
# My sa budeme snažiť vysvetliť vplyv "hp" (horsepower, konské sily),
# na "mph" (miles per galon, čiže koľko kilometrov má auto dojazd).
```

Pozrime sa na tento model:

$$y_i = \beta_0 + \beta x_i + \epsilon_i.$$

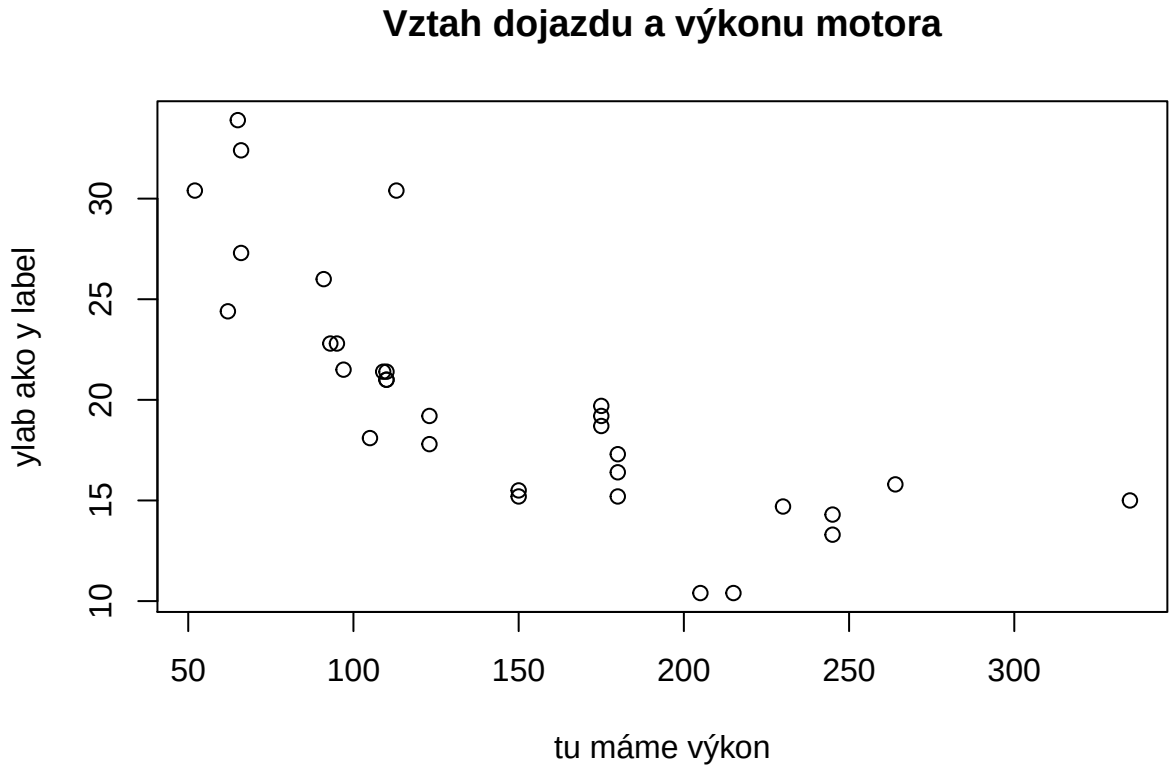
Jedná sa o základný model lineárnej regresie. Ak si bety predstavíme ako obyčajné hodnoty, model v tomto jednoduchom znení by Vám z matematiky mohol byť povedomý. Výsledkom modelu je priamka y, kde β_0 je obyčajná hodnota v ktorej je os Y pretnutá, β_1 určuje sklon priamky. Keďže priamka nebude prechádzať každou nameranou hodnotou, ϵ v modeli znázorňuje tento priestor medzi meraniami a priamkou. Označujeme ho ako náhodnú zložku. Môže byť spôsobená mnohými spôsobmi. Okrem iného napríklad: chybným meraním, náhodou, alebo premennými, ktoré sme do modelu nezahrnuli. Jedno auto so 150 koňskými silami môže na plnú nádrž prejsť 500km a druhé len 350. Môže to byť spôsobené váhou, avšak v modeli máme len výkon auta. Táto zložka sa v priemere rovná nule, a z modelu nám odpadne (viac o tom neskôr). Tento model si môžeme v našom príklade prepísať ako:

$$mpg_i = \beta_0 + \beta_1 hp_i + \epsilon_i.$$

Priamka, ktorá by vznikla výsledkom tohto modelu sa nazýva populačná regresná priamka. Na vyhodnotenie takéhoto modelu by sme však potrebovali dáta z celej určenej populácie, čo je veľmi často prakticky nemožné. Vo všeobecnosti sú populačné parametre β_0 a β_1 neznáme. Avšak dokážeme ich odhadnúť pomocou estimátorov. Preto pracujeme so vzorkami, ktoré dokážeme reálne odpozorovať a zozbierať. Estimátory zo vzorky dokážu poskytnúť dostatočne dôveryhodné odhady koeficientu v populácií. Poďme si to teda ukázať na našej vzorke.

Skúsme si hodiť do grafu výkon a dojazd, nech sa pozrieme na vzťah medzi týmito premennými.

```
# Plotneme si dáta, ktoré dáme do regresie, čiže "mpg" a "hp".

plot(x = data$hp, y = data$mpg, ylab = "ylab ako y label", xlab = "tu máme výkon",
     main = "Vzťah dojazdu a výkonu motora")
```



Vidíme istú negatívnu závislosť. Chceli by sme si to však potvrdiť číslami. Bolo by fajn napasovať medzi tieto pozorovania takú priamku, ktorá bude ku každému pozorovaniu čo najbližšie. Keďže nepoznáme parametre populácie, budeme pracovať s ich estimátormi, odhadcami. Estimátory označíme striežkou ako $\hat{\beta}_0$ a $\hat{\beta}_1$. Model bude vyzerať následovne:

$$\hat{mpg}_i = \hat{\beta}_0 + \hat{\beta}_1 hp_i.$$

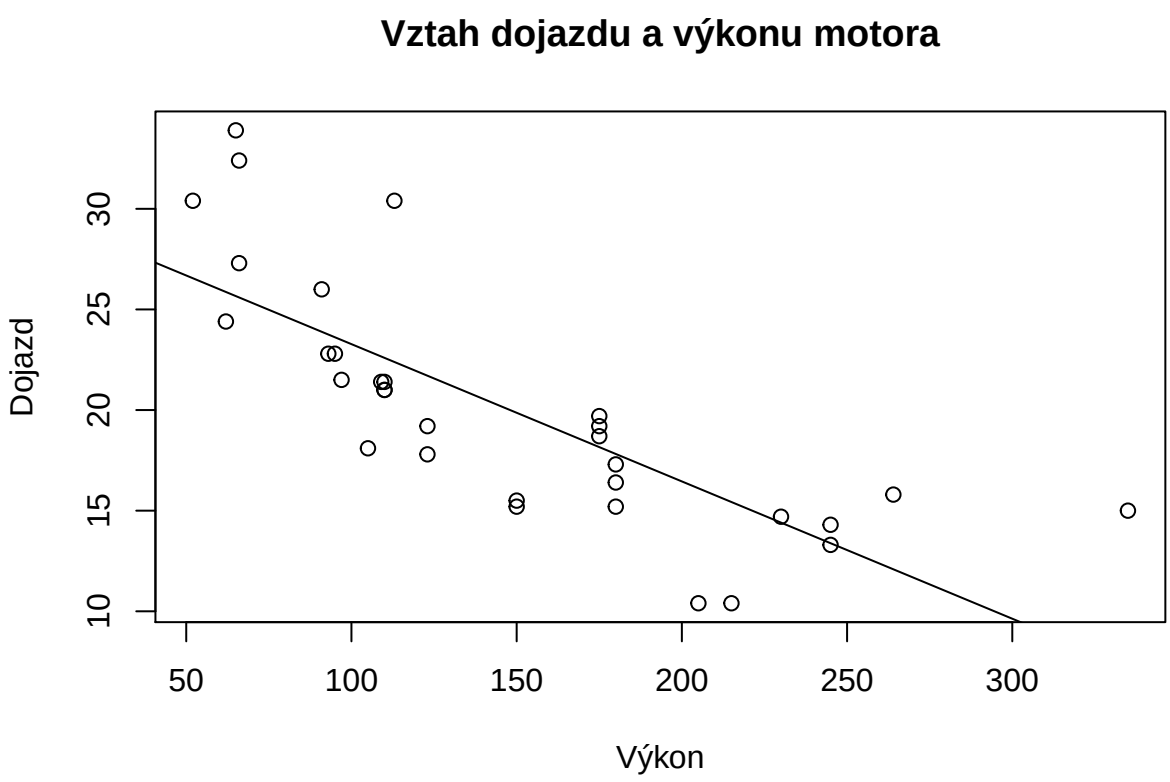
Skúsme si takýto model zostrojiť v R-ku, a priamku dopasovať do grafu.

```
# Chceme zistiť dopad "hp" na "mpg".
# Použijeme funkciu "lm()", ako linear model.
# Na konci musíme ako argument uviesť dáta, ktoré použijeme.

model <- lm(mpg ~ hp, data = data)

# Alternatívne môžeme model napísať pomocou extrakcie takto.
# Preferujem tento postup.

model <- lm(data$mpg ~ data$hp)

# Dostaneme dva koeficienty. Jeden pre intercept Beta0 a druhý pre Beta1 ako "hp".

model
```

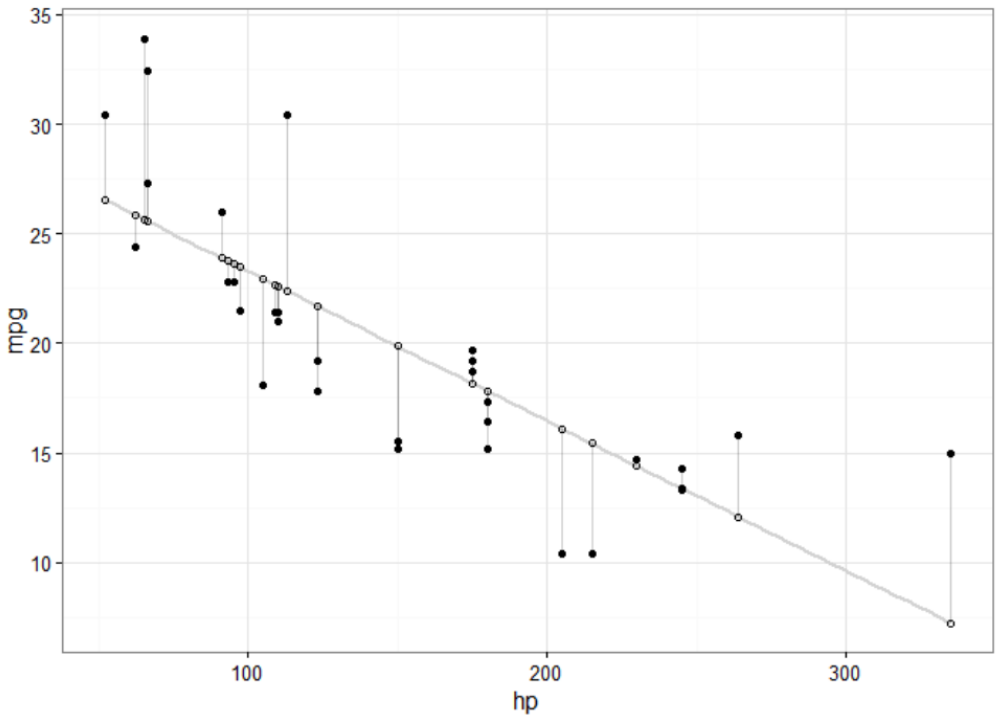
```
##
## Call:
## lm(formula = data$mpg ~ data$hp)
##
## Coefficients:
## (Intercept)      data$hp
##    30.09886     -0.06823
```

```
# Intercept sa väčšinou neinterpretuje, slúži len pre určenie počiatočnej hodnoty.
# Ak by sme koeficient interpretovali, mohli by sme povedať, že auto s nula koňmi
# prejde 30 míľ na galón. Koeficient B1 by sme interpretovali ako:
# "každá extra konská sila, zníži dojazd o -0.068 míle".
```

```
# Znova si plotneme dáta.

plot(x = data$hp, y = data$mpg, ylab = "Dojazd", xlab = "Výkon",
     main = "Vzťah dojazdu a výkonu motora")

# A pomocou funkcie "abline()" napasujeme do grafu priamku modelu.
# Abline ako priamka z bodu A do bodu B.

abline(model)
```



Z grafu pozorujeme, že aj tu priamka neprechádza všetkými pozorovaniami, túto kolmú vzdialenosť medzi priamkou a každým pozorovaním označíme ako $\hat{u}$. V tomto modeli to neoznačuje odhad náhodnej zložky ϵ, ale výpočtovú nepresnosť modelu. Môžeme to brať ako takého súrodenca k náhodnej zložke. Obe chyby predstavujú podobnú vec, vzdialenosť medzi priamkou a pozorovaním. Priamku populácie je však neznáma, teda aj táto vzdialenosť ϵ je neznáma. Na druhú stranu, zvyšky $\hat{u}$ sú vyrátané z dát, a dokážeme ich presne zmerať. Ako však vyrátame bety? Iste ste už začuli o OLS, teda Ordinary Least Squares alebo Metódy najmenších štvorcov. Bety so striežkou nazývame ako OLS estimátory. Metóda najmenších štvorcov sa to volá preto, lebo vezmeme zvyšky $\hat{u}$ z modelu, umocníme ich na druhú mocninu (urobíme z nich štvorce), a minimalizujeme ich. Osobne si nemyslím, že v tomto štádiu vašej výučby má veľký význam sústrediť sa na odvodenie týchto estimátorov. Osobne mi to moc nedalo, preto sa skôr zameriame na výsledky týchto odvodzovaní, nech nadobudneme trocha intuície. Vysvetlíme si graficky, čo touto metódou chceme docieliť.



Tieto vzdialeností predstavujú naše "residuals" $\hat{u}_i$, naše zvyšky z modelu, naše nepresnosti. Každý data point má svoj zvyšok. Logicky chceme, aby priamka čo najtesnejšie vystihovala dáta, chceme tým pádom čo najviac zmenšiť zvyšky. Poďme sa k tým zvyškom dopracovať.

Pozorovanie sa rovná bodu na priamke $\hat{\beta}_0 + \hat{\beta}x_i$ plus zvyšok $\hat{u}_i$:

$$y_i = \hat{\beta}_0 + \hat{\beta}x_i + \hat{u}_i.$$

Poprehadzujme si to, nech máme na jednej strane zvyšok $\hat{u}_i$:

$$\hat{u}_i = y_i - \hat{\beta}_0 - \hat{\beta}x_i.$$

Väčšinou narazíme na takýto zápis zvyškov, a to rozdiel medzi pozorovanou hodnotou a odhadnutou hodnotou na priamke:

$$\hat{u}_i = y_i - \hat{y}_i.$$

a keďže

$$\hat{y}_i = \hat{\beta}_0 + \hat{\beta}x_i,$$

Tak sme tam, kde sme boli na začiatku, keď sme si dali zvyšok $\hat{u}_i$ na jednu stranu. OLS metódoa urobí to, že minimalizuje súčet našich umocnených zvyškov $\hat{u}_i$:

$$min \sum (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i)^2.$$

Tým, že minimalizujeme zvyšky dostaneme čo najlepšiu priamku, ktorá bude sedieť čo najtesnejšie s dátami. Táto minimalizácia nám poskytne vzorec na výpočet OLS estimátorov. Estimátorov parametrov populácie. Bavíme sa o odhadcoch priesečníka, β_0, a sklonu, β_1. A keď sa bavíme o odhadcoch, píšeme ich ako $\hat{\beta}_0$ a $\hat{\beta}_1$. Nech si pamätáte.

Na hodinách sa stretnete so vzorcom pre výpočet biet v maticovej forme. A to:

$$Est(\beta) = \hat{\beta} = (X^T X)^{-1} X^T y.$$

T-čka znamenajú transpose matice a -1 vyrátanie inverzie.

```
# Inverznu maticu vyrátame pomocou "solve()" a transpozíciu pomocou "t()".
# V R-ku by sme to vyrátali ako:

solve(t(X) %*% X) %*% t(X) %*% y)
```

Tento vzorec by som nazval funkčným, avšak určite nie intuitívnym. Pracujete s ním preto, lebo:

- takáto forma je ľahšie spracovateľná pre výpočtovú techniku,
- R-ko pri práci s datasetom ho aj tak pretvorí na maticu, predtým než podá výsledky,
- tento vzorec funguje ako pre jednoduchú lineárnu regresiu (jedno Y a jedno X), tak aj pre viacnásobnú regresiu (jedno Y a viacero X).

Ono, R-ko to urobí všetko za Vás, samo vyráta všetky koeficienty, nuž nechceli by ste vedieť, čo tá $\hat{\beta}$ vlastne robí? Pri jednoduchej lineárnej regresií dokážeme OLS beta estimátory zapísať aj takto:

$$\hat{\beta}_0 = \overline{y} - (\hat{\beta}_1 \overline{x})$$

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sum_{i=1}^{n}(x_i - \overline{x})^2}$$

Čiara nad písmenom znamená priemer, teda ybar(takto to čítame) je priemer všetkých hodnôt "y" v datasete.

Neopúšťajte ma. Ak Vám tento vzorec nie je povedomý, nič sa nedeje. Pozrieme sa na to. Rozdeľme si to na vrch a spodok. Začneme menovateľom, lebo je kratší, hm.

Súčet od i = 1 po n znamená, že vykonáme operáciu pre každé x v datasete a výsledky operácií sčítame.

Summation znak sa nazýva grécky sigma.

Od každého x odčítame priemernú hodnotu $\overline{x}$ a umocníme to. Získame tým variabilitu okolo priemeru, inak povedané, ako veľmi sú data v datasete rozptýlené. Dostaneme rozptyl. Je dôležité podotknúť, že rozptyl kvôli umocneniu nikdy nebude záporný. Dôležité je to preto, lebo vzťah v čitateli určuje, či je β pozitívna alebo negatívna. Takže menovateľ neovplyvní kladnosť či zápornosť čitateľa.

Vzťah v čitateli vyzerá celkom podobne k tomu v menovateli, hm? A nie je to náhoda. Vzťah v čitateli je obyčajná kovariancia, inak povedané, spoločný rozptyl. Predstavuje závislosť medzi dvoma veličinami. Tento vzťah neumocňujeme, pretože chceme vidieť, či bude závislosť pozitívna, alebo negatívna.

Kovariancia či korelácia? Čo je čo? Korelácia je štandardizovaná kovariancia. Obe vysvetľujú to isté, líšia sa len v rozsahu. Kovarianciu predelíme násobkom smerodajných odchýlok x a y, a získame koreláciu. Štandardizujeme to preto, aby sme sa vedeli orientovať, aký veľký je v skutočnosti vzťah medzi premennými. Ak sa bavíme o kovariancii hmotnosti v kilogramoch a dĺžky lietadla v metroch, výsledné hodnoty budú veľmi vysoké. Ak budú kladné, budeme vedieť, že je medzi nimi lineárna závislosť, ale nevieme aká veľká. Ak by sme porovnávali kovarianciu kurzu EUR a USD, výsledné hodnoty budú síce menšie, ale stále o nič viac vhodné na interpretovanie. Ak však tieto hodnoty predelíme násobkom smerodajných odchýlok, teda ich štandardizujeme, výsledné hodnoty budú porovnateľné pre akékoľvek premenné. Keďže ťažké a dlhé lietadlo bude mať úmerne veľké smerodajné odchýlky, kdežto výmenný kurz bude mať smerodajnú odchýlku prislúchajúcu hodnotám kurzu. Hodnoty v korelácií budú spadať medzi hranice -1 a 1. Kde záporná hodnota predstavuje negatívnu lineárnu závislosť, a naopak.

Pri odhade však nechceme hodnoty štandardizovať, ale vecne odhadnúť použiteľné hodnoty koeficientov. $\hat{\beta}_1$ si jednoducho zapíšeme ako:

$$\hat{\beta}_1 = \frac{Cov(x, y)}{Var(x)}.$$

Otestujme si, či to naozaj funguje.

```
# Pripomeňme si hodnoty nášho modelu.

model
```

```
##
## Call:
## lm(formula = data$mpg ~ data$hp)
##
## Coefficients:
## (Intercept)      data$hp
##    30.09886     -0.06823
```

```
# Na poradí pri kovariancii nezáleží.

kovar <- cov(data$mpg, data$hp)

rozptyl <- var(data$hp)

beta1 <- kovar/rozptyl

print(beta1)
```

```
## [1] -0.06822828
```

```
# Oba parametre majú hodnotu -0.068 a rovnajú sa.
# Skúsme B0.

ybar <- mean(data$mpg)
xbar <- mean(data$hp)

beta0 <- ybar - (beta1 * xbar)

print(beta0)
```

```
## [1] 30.09886
```

```
# Sedí. :)
```

Oba parametre majú hodnotu -0.068 a rovnajú sa.

Nevravím, že sme objavili Ameriku, ale aspoň viete, že $\hat{\beta}_0$ je nejaký priemer y a od toho odčítame $\hat{\beta}_1$ vynásobenú priemerom x. A neskrýva sa za tým žiaden ťažký imaginárny vzorec. Taktiež, že $\hat{\beta}_1$ je spoločný rozptyl závislej a nezávislej premennej, vydelený rozptylom nezávislej premennej.

Ako však vieme, či estimátorom $\hat{\beta}_0$ a $\hat{\beta}_1$ môžeme veriť?

5 Trocha štatistiky

Predtým, než si povieme o podmienkach lineárnej regresie Vás oboznámim s pár záležitosťami, s ktorými sa stretnete, a napriek tomu, že sú pomerne jednoduché by Vám zabrali dosť googlenia.

Prejdeme si:

- očakávanú hodnotu,
- zákon veľkých čísel,
- náhodný výber,
- centrálna limitná veta,
- normálne rozdelenie.

Možno ste si všimli, že pri interpretáciach koeficientov sa často opakuje niečo v zmyysle:“V priemere nám pri zvýšeni bla bla bla narastie o bla bla bla.” To **v priemere** je veľmi podstatné. Väčšinou pracujeme so vzorkami, ktoré boli zozbierané z populácie, ktorá je pre nás neznáma. Ideálne boli vybrané náhodne, teda pozorovania vo vzorke sú náhodnými veličinami, a štatistiky ktoré z nich vyrátame sú potom tiež náhodné veličiny. Prečo je to podstatné sa dozvieme v nasledujúcich konceptoch.

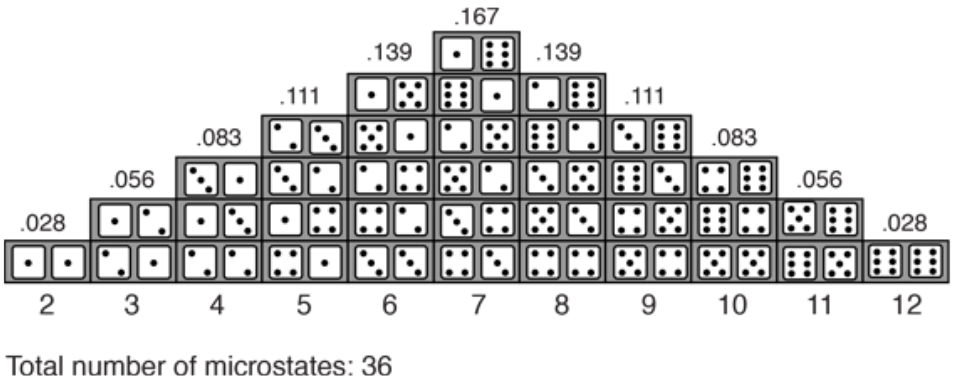
> _Keď sa bavíme o štatistikách, máme na mysli akúkoľvek hodnotu, ktorú je možné vyrátať a opisuje niečo. Priemer je štatistika, rozptyl je štatistika."

5.1 Očakávaná hodnota

Spomínam si, ako som otvoril videa Bena Lamberta, a tam na mňa vyskočilo hneď nejaké tlačené É-čko a rôzne kvačky. Veľké tlačené E znamená expected value, teda očakávaná hodnota. Je to tak, ako to znie. Očakávaná hodnota náhodnej premennej je jednoducho povedané priemer, ktorý by sme vyrátali za dlhú dobu a pri niekoľkonásobnom opakovanom výbere vzorky. Pre diskrétnu náhodnú premennú vyrátame túto hodnotu ako vážený súčet, kde váha je určená pravdepodobnosťou výskytu. Vzorcom:

$$E(y) = y_1p_1 + y_2p_2 + ... + y_kp_k = \sum_{i=1}^{k} y_ip_i$$

A teraz príklad zo života na pochopenie. Prečo myslíte, sa hovorí “Lucky Seven”, teda že sedmička je šťastné číslo? Je to preto, lebo keď sčítate všetky kombinácie čísel na dvoch hracích kockách, najviac hodnôt vyjde pre 7, teda aj pravdepodobnosť, že padne toto číslo je väčšia, ako pri ostatných súčtoch. V priemere potom padne najviac ľudom sedmička, ľudia si to všímajú a stavujú na sedmičky, alebo také niečo, nie som moc na gambling.



5.2 Zákon veľkých čísel

S tým úzko súvisí aj zákon veľkých čísel. Zákon vraví, že ak rovnaký experiment opakujeme nezávisle od seba nespočetne veľa krát, priemer výsledkov bude blízko k **očakávanej hodnote**. Výsledok sa bude približovať k očakávanej hodnote, ako sa bude počet pokusov zvyšovať.

Príklad: Keď si s niekym budete hádzať mincu, možno padne 10-krát za sebou orol, ale keby sme mincu hodili miliónkrát, výsledky by boli približne 50/50. Inak povedané, šanca že padne hlava (alebo orol) je 50%, ak minca nie je cinknutá. Očakávaná hodnota pri hode mincou je 0.5. Šanca že padne hlava je 1 a možné výsledky sú 2, teda pravedpodobnosť, že padne hlava je 0.5, čiže 50%.

5.3 Náhodný výber štatistiky

V angličtine random sampling. To ing značí nejakú činnosť. Náhodný výber znie skôr ako jeden výber, avšak pri random sampling ide o niečo iné. Väčšina ekonometrických procedúr pracuje s priemermi vzoriek. Čiže tento náhodný výber, sa bude týkať priemeru. Povedzme, že chceme odhadnúť priemernú výšku v populácií. Väčšinou predpokladáme, že pozorovania sú zozbierané náhodne z veľkej, nepoznanej populácie. Vyrátanie priemeru z takejto vzorky má za následok to, že tento priemer je *náhodnou premennou. Táto náhodná premenná má potom rozdelenie pravdepodobnosti, nazývané výberové rozdelenie.* Výberové rozdelenie závisí od rozdelenia populácie, z ktorej sme vzorku zobrali. Predpokladajme, že máme normálne distribuovanú populáciu, a vyberieme z nej veľa veľa vzoriek, vyrátame priemer týchto vzoriek, a urobíme z týchto priemerov histogram. Rozdelenie tohto histogramu bude kopírovať rozdelenie, z ktorého sme tieto vzorky zobrali, teda normálne rozdelenie. Náhodný výber by mal eliminovať odchýlku, keďže každý z populácie má rovnakú šancu byť vybraný. Získame teda rozdelenie bez odchýlky, ktoré kopíruje rozdelenie populácie. Výberové rozdelenie môže byť blízko normálneho rozdelenia, aj keď populácia z ktorej sme brali vzorky nemá normálne rozdelenie. A to vďaka Centrálnej limitnej vete.

5.4 Centrálna limitná veta

Kdežto Zákon veľkých čísel sa zameriaval skôr na odhad danej štatistiky, Centrálna limitná veta súvisí s rozdelením vzorky. Podstatou je, že ak vezmeme dostatočne veľké množstvo priemerov vzoriek, súbor týchto priemerov bude mať normálne rozdelenie, bez ohľadu na rozdelenie populácie. Takáto vzorka by mala mať aspoň 30 pozorovaní. Nie je však potrebné zbierať veľa veľa vzoriek, keďže na vzorku použijeme estimátor, napríklad na odhad priemeru, a samotný výsledok bude náhodná veličina (ako sme už spomenuli pri náhodnom výbere), ktorá sama pochádza z náhodného výberu. Čiže na splnenie predpokladu, že výsledné rozdelenie budeme môcť odhadnúť normálnym rozdelením, závisí už len od veľkosti vzorky. Čím vzdialenejšie od normálneho rozdelenia je rozdelenie populácie, tým väčšia vzorka bude potrebná, aby toto pravidlo platilo.

6 Podmienky lineárnej regresie

My chceme, aby naše estimátory boli BLUE! A tým nemyslíme modré, ale Best Linear Unbiased Estimators! Najlepší Lineárni Nevychýlení Odhadcovia!

Unbiased znamena že v priemere Beta trafí cieľ, teda priemer.

Vy sa učíte rátať toto pomocou solve x transpose, bude to fungovať aj na SLR, avšak ukažeme si beta estimatory... bla bla.. vyratame cov a var.. rovno si naš model plotnime nech si ukažeme chyby. OLS sa snaži napasovať priamku tak, aby boli chyby najmenšie. čiže residuals vs fitted

$\sum_{n=1}^{n}$

$\sum_{n=1}^{n}$

$\hat{\beta}_0$

$$\hat{y} = a + bx$$

where

$$\hat{\beta_0} = \bar{y} - \hat{\beta}_1 \bar{x}$$

#nainštalovat len jeden balik a potom ked prejdeme vektory tak c(viac balikov)

6.1 Import údajov

Table Header

Fukncia	Second Header
Content Cell	Content Cell
Content Cell	Content Cell