Duomenų analizės įvadas

Justas Mundeikis

2019 m. sausio 2 d.

Šio kurso turinys:

1. Įvadas: apie kursą, CLI, Git ir GitHub, R ir RStudio
2. R programavimo pagrindai
3. Tiriamoji duomenų analizė (angl.: exploratory data analysis)
4. Atkartojami tyrimai (angl.: reproducible research) su LaTex, RMarkdown, Jupyter

Kurso tikslas

In God we trust, all others bring data

- W. Edwards Deming

- Suteikti pagrindines reikalingas kompetencijas darbui su duomenimis
- Šis kursas labiau taikomasis, nėra aiškios takoskyros tarp paskaitų ir seminarų
- Kursas yra "lengvas", tačiau reikalauja daug praktinio darbo pastangų
- Dirbama tik su R, RStudio, Git ir Github, Jupyter?
- Neliesime Matlab, Python, Julia, Eviews
- Laikas **savistudijoms**
- Vietiniai PC vs. nuosavi notebook'ai

Kurso vertinimo strategija

- Savaitiniai namų darbai - 30%
  Savaitinių namų darbų tikslas parodyti studentams kokio pobūdžio uždavinius studentai turi gebėti savarankiškai spręsti. Namų darbų vertinimas: maksimumas iš leidžiamų 3 bandymų
- Neanonsuoti testai - 30%
  Neanonsuoti trumpi testai skirti užtikrinti, jog studentai nuolatos skirtų deramą dėmesį ir laiku įdėtų reikalingas pastangas studijoms
- Baigiamasis egzaminas - 40%
  Egzamine tikrinamos tiek teorijos tiek įgytos praktinės žinios
- $BP = ceiling(mean(SND) * 0.3 + mean(NT) * 0.3 + BE * 0.4)$

Sando pristatymas

- Sando pristatymas
- Klausimai?

Įvadas į duomenų analizę, Git ir GitHub, R ir RStudio

Šioje dalyje susipažinsime su

- CLI
- Git
- Github
- R
- RStudio

Command Line interface (CLI)

Kiekviena operacinė sistema turi CLI:

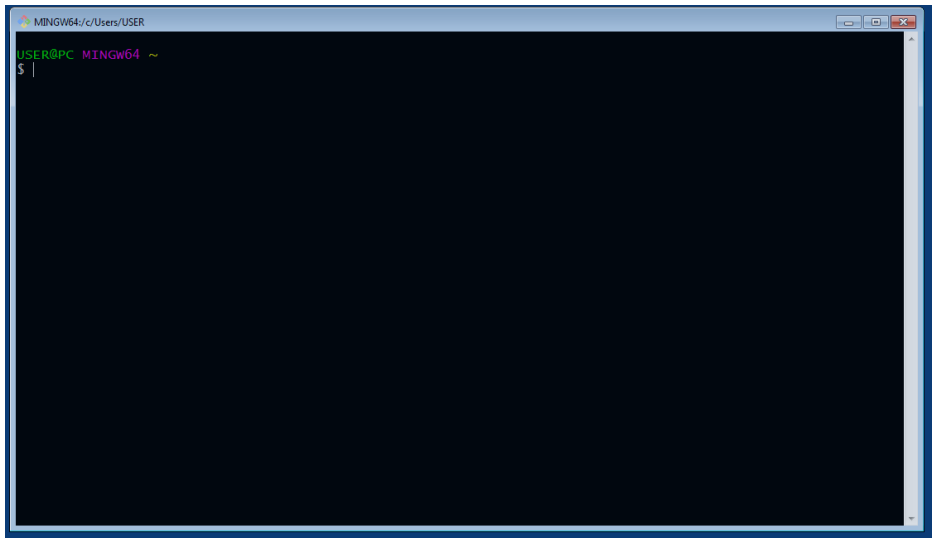
- Windows: Git Bash (), CMD
- Mac/ Linux: Terminal'as

Su CLI galima:

- Naviguoti tarp aplankų (folder'ių)
- Kurti, keisti, naikinti: failus, aplankus, programas
- Startuoti programas

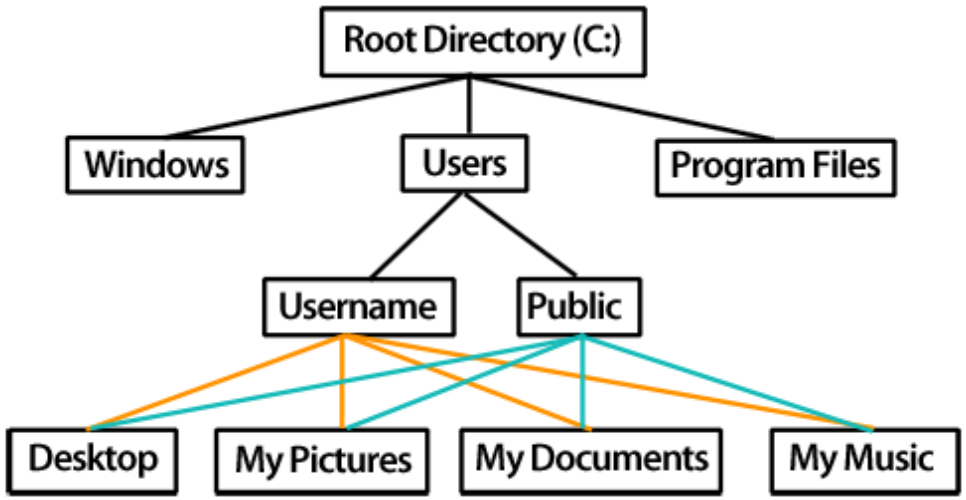
Intarpas GIT Bash instaliavimas

- Nors darbiniai kompiuteriai turi instaliuotą Git Bash, tiems kas neturi:
- https://git-scm.com/
- Windows 32/64, Linux žr. komandą
- Perimti teikiamus standartinius siūlymus, nebent antrame Setup lange pasirinkti, jog Git Bash rodytų ir "Addditional icons: On the desktop"
- startuojam Git Bash



Direktorijos

- "Directory" yra tiesiog kitas pavadinimas žodžiui aplankas
- Direktorijos kompiuteryje organizuotos kaip medžio šakos
- CLI padeda naviguoti tarp šių direktorijų
- "/" yra root directory Linux, C: yra root directory Windows
- root directory talpina visas kitas direktorijas



Direktorijos

- startavus matosi daug maž toks tekstas:

```
USER@PC MINGW64 ~
$
```

- $ ženklas reiškia: "gali rašyti komandą"
- tipinis įrašas: "command flag argument"
- komandos pvz: komanda liepianti atspausdinti kurioje direktorijoje esama: "pwd"

```
USER@PC MINGW64 ~
$ pwd
/c/Users/USER
```

Direktorijos

- komanda gali būti iššaukiama su tam tikra programa
- `git init`
- `python get-pip.py`
- flag: tam tikri nustatymai, galimi priklausomai nuo komandos ir visada su "-"
- argument - kiti nustatymai, pakeitimai ar panašūs dalykai
- `git commit -m "this is the initial commit"`
- jeigu flag yra žodis , tada naudojami du brūkšniai --
- `git reset --hard HASH`

CLI komandos

- "ls" nurodo visus failus ir folderius esančius direktorijoje
- "ls -a" nurodo visus matomus ir paslėptus failus ir folderius
- "ls -al" nurodo visų matomų ir paslėptų failų ir folderių detales
- "clear" išvalo CLI

CLI komandos

- "cd" reiškia "change directory"
- "cd" be argumentų sugrąžins į home directory
- "cd .." pakels viena direktorija aukščiau
- Uždavinys: su cd nueiti and "Desktop"

CLI komandos

- "mkdir" "make directory" sukuria folderį pvz: "Duomenų analizės įvadas"
- "rmdir" "remove directory", bet tik, jeigu folderis yra tuščias
- Nueiti į sukrtą direktoriją
- "touch" sukuria failą, pvz., "touch info.txt"
- Sukurti dvi direktorijas folder1 ir folder2
- "cp" kopijuoja failus (cp failas direktorija)
- "cp failas failas" padaro failo kopiją
- "cp info.txt folder1"
- "cp -r" folderis direktorija (-r recursive t.y. įtraukia viską, kas yra folderio viduje)
- "cp -r folder1 folder2"

CLI komandos

- "rm" "remove" su argumentu failo pavadinimu
- "rm -r" su direktorijos pavadinimu viskam kas direktorijoje
- "mw failas direktorija" perkelia failą (Cut+Paste)
- "mv" failassenas failasnaujas (pakeičia pavadinimą) (Rename)
- "echo" atspausdina tekstą (echo Labas; echo date)
- "nano failas" startuoja nano editorių
- "exit" uždaro CLI

Trumpas įvadas į Git

"Git is a version-control system for tracking changes in computer files and coordinating work on those files among multiple people. It is primarily used for source-code management in software development,but it can be used to keep track of changes in any set of files. As a distributed revision-control system, it is aimed at speed, data integrity, and support for distributed, non-linear workflows"

https://en.wikipedia.org/wiki/Git

Git

"Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency"

https://git-scm.com/

- Sukurta Linux kurėjo Linus Torvalds
- Populiariausias VCS
- Viskas išsaugoma lokaliai
- GIT naudojamas naudojant CLI
- http://git-scm.com/downloads

Git pagrdiniai nustatymai

- Kiekvienas išsaugojimas bus susietas su išsaugotu "user.name" ir "user.email"
- Tai reikia padaryti tik vieną kartą (dirbant su savo PC), arba pasikeisti kaskart prisėdus prie svetimo pc

```
$ git config --global user.name "Justas Mundeikis"
$ git config --global user.email mundeikis@gmx.de
$ git config --list
$ git config --global -l
$ git config --global core.pager cat
# core.pager is made to cat (cat=content, printed on CLI)
```

GitHub

"GitHub is a web-based hosting service for version control using Git. It is mostly used for computer code. It offers all of the distributed version control and source code management (SCM) functionality of Git as well as adding its own features. It provides access control and several collaboration features such as bug tracking, feature requests, task management, and wikis for every project"

https://en.wikipedia.org/wiki/GitHub

- "push" ir "pull" tarp lokalių ir internetinių repozitorijų
- suteikia homepage vartotojo repozitorijoms
- GitHub atlieka back-up funkciją lokalioms repozitorijoms
- leidžia bendradarbiauti, dalintis projektais, gerinti kitų kodą ir t.t.
- GitHub paskyros susikūrimas...

Git

- Su Git-Bash nueiname į "Desktop", kur sukuriame direktoriją "Duomenu analizes ivadas Sxxxxx" kur Sxxxx pažymi studento ID (šiuo kompiuteriu naudosis ir kiti studentai..)
- Šioje direktorijoje sukuriame dar vieną direktoriją "1 Ivadas"
- Inicializuojame git

```
$ git init
Initialized empty Git repository in C:/Users/USER/Desktop/↩
    Duomenu analizes ivadas Sxxx/1 Ivadas/.git/
```

- direktorijoje sukuriamas nematomas failas .git

Git kaip foto sesija

- GIT daro tarsi nuotraukas, pasirinktos direktorijos
- inicializavimas, tai tarsi foto kambario pasirinkimas, kuriame gali būti daug "veikėjų"
- "git add failas" yra tarsi "veikėjo" užvedimas ant foto scenos
- "git commit" yra pačios nuotraukos darymas
- tam kad nuotraukoje nebūtų tam tikrų asmenų galia naudoti sąrašą kuris slepiasi ".gitignore" faile

Git

- su komanda touch sukuriame failą readme.txt
- "notepad readme.txt" įrašome change1, išsaugome
- šis failas egzistuoja direktorijoje, tačiau nėra "sekamas"

```
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be ↩
      committed)

        readme.txt

nothing added to commit but untracked files present (use "↩
    git add" to track)
```

- Taigi failas readme.txt yra "untracked"

Git

- su komanda "git add readme.txt" įkeliama readme.txt į staging area

```
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   readme.txt
```

- Taigi failas reamde.txt yra staged bet dar ne "commited"
- Ką reiškia, jog failas yra staged?

Git

- notepad kitoje eilutėje įrašome "change2", išsaugome

```
$ git status
On branch master
No commits yet
Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   readme.txt

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in ↩
      working directory)

        modified:   readme.txt
```

- dabar matome, jog readme.txt yra "tracked" ir "untracked"
- Staging area esantis failas "readme.txt" yra išsaugotas tik su įrašu "change1", bet be su įrašu "change2"

Git

- norint išsaugoti naujausią versiją: "git add readme.txt"
- norint perduoti failą repozitorijai

```
$ git commit -m "sukurtas failas readme.txt"

[master (root-commit) 30cd9e9] sukurtas failas readme.txt
 1 file changed, 1 insertion(+)
 create mode 100644 readme.txt
```

- dabar patikrinus statusą

```
$ git status
On branch master
nothing to commit, working tree clean
```

Git

- readme.txt pirašome "change3", stagindam ir commitiname su -m "sukurtas change3"

Git

- norint žinoti kas kada kaip keitė failą: "git log"
- pateiktas log'as kiekvienam atrodys kitaip, bet esmė ta pati:

```
$ git log
commit 1077a8e2e27424339d56a2dfd20b4aec56b0d3fb (HEAD -> ↩
    master)
Author: Justas Mundeikis <mundeikis@gmx.de>
Date:   Tue Jan 1 07:19:02 2019 -0800

    sukurtas failas readme.txt

commit 30cd9e9503f37bc748de35ff66d2fae8d01e3d72
Author: Justas Mundeikis <mundeikis@gmx.de>
Date:   Tue Jan 1 07:16:54 2019 -0800

    sukurtas change3
```

Git

- su touch sukuriame kitą failą "basic.R"
- o readme.txt įrašome "change4"
- "git status" parodo, jog pkaeistas readme.txt failas, ir untracked basic.R failas
- su komanda "git add ." staginame visus failus
- galimi kiti variantai : git stage -A, git stage -u (naudojamas tik update'inti esamus failus)
- tada git commit -m "sukurtas change4 ir sukurtas faials basic.R"

Git

- kartais yra tam tikri failai, kurių nenorime sekti (pvz duomenys, nereikalingi LaTex failai ir t.t.)
- touch data.csv
- git status parodo failą kaip untracked
- todėl sukuriame faila .gitignoe ir jame įrašome failų pavadinimus, arba galūnes kurių nenorime trackinti

```
git touch .gitignore
notepad .gitignore
```

- atsidariusme editoriuje įrašome

```
*.csv
```

- * reiškia bet kokį pavadinimą, po kurio seka taškas ir csv, alternatyviai galima specifikuoti konkretų failą "data.csv"
- "git status" neberodo failo "data.csv" bet rodo ".gitignore", todėl staginame ir commitiname pakeitimus

Git branch'inimas

- Bazinis scenarijus:
  - A kuria projektą, B nori prisidėti, tačiau ir A dirba tuo pat metu...
  - B atsiskelia atšaką (brach'ina) A projektą, padaro savo pakeitimus ir pateikia A juos sujungti
  - A peržiūri pakeitimus, priima/atmeta

Git branch'inimas

- "git branch NewBranch" sukuria naują atšaką pavadinimu NewBranch
- "git checkout NewBranch" išmeta iš "master" į "NewBranch"
- atitinkmai "git choeckout master" visada sugrąžina atgal į master atšaką

```
USER@PC MINGW64 ~/Desktop/Duomenu analizes ivadas Sxxx/1 ↩
    Ivadas (master)
$ git branch NewBranch

USER@PC MINGW64 ~/Desktop/Duomenu analizes ivadas Sxxx/1 ↩
    Ivadas (master)
$ git checkout NewBranch
Switched to branch 'NewBranch'

USER@PC MINGW64 ~/Desktop/Duomenu analizes ivadas Sxxx/1 ↩
    Ivadas (NewBranch)
```

- dabar visi pakeitimai vyks tik šioje atšakoje ir nepaveiks master šakos

Darbas Git atšakoje

- sukuriame naują faila "touch advanced.R"
- readme.txt įrašome papildomą eilutę "change5"
- staginti ir commitinti pakeitimus sugrįžtame į master atšaką "git chekcout master"
- Rezultatas: trūksta advanced.R failo, readme.txt turi tik 4 įrašus!
- norint sujungti naują atšaką į master "git merge NewBranch"

Merge problemos

- master branch readme.txt sukuriame eilute "change6", addinam ir commitiname (jeigu nesukurti nauji failai: git add -a -m "...")
- nueiname į atšaką NewBranch, ir ten esančiame faile sukuriame change7, addinam ir commitiniame
- grįžtame į master branch "git chekckout master"
- ir bandome sujungti "git merge NewBranch"

```
$ git merge NewBranch
Auto-merging readme.txt
CONFLICT (content): Merge conflict in readme.txt
Automatic merge failed; fix conflicts and then commit the ↩
    result.
```

- git negalėjo automatiškai sutvarkyti failų, todėl atsidarome readme.txt failą ir tvarkome patys

Konfliktinio failo tvarkymas

- Atsidarius readme.txt matome

```
change1
change2
change3
change4
change5
<<<<<<< HEAD
change6
=======
change7
>>>>>>> NewBranch
```

- <<<<< *HEAD* yra tai kas yra aktyvioje atšakoje
- >>>> *NewBranch* yra kas ateina iš sujungiamos atšakos
- atskirta ======

Konfliktinio failo tvarkymas

- Sutvarkome failą, taip kaip norime

```
change1
change2
change3
change4
change5
change6
change7
```

- saviname, ir tada "git commit -a -m "sujungtas failas iš NewBranch bei pašalintas konfliktas"
- Yra papildomų įrankių, kurie padeda atlikti merge'inimo darbus, nes dažniausiai konfliktų visada bus

Konfliktinio failo tvarkymas

- master atšakoje sukuriame failą "touch markdnwon.md",
- readme.txt papildome įrašu "error entry"
- jeigu failo nestaginame ir necommitiname (nes pvz reikia trumpam kažką pakeisti kitoje atšakoje), ir periname į kitą atšaką, failas lieka o tai gali sukurti ateityje daug problemų
- todėl "git add ."
- "git stach" nukelia ne commitintą failą į stached area. Failo neberodo darbalaukyje
- dabar galime darbuotis kitose atšakose ir vėl grįžus: "git stach apply" ir markdown.md failas vėl atsiranda baigus ji taisyti, galima commitinti.
- tai ir padarome

Git atsatatymas 1

- Na štai po vidurnakčio pasidarbavus, padarėme klaidą:
- readme.txt papildome įrašu "error entry"
- `git commit -a -m "padarme klaida`
- Tarkime kažką sugadinome, bet žinome, jog versija prieš tai veikė gerai
- su "git log" susirandame "blogo" commit hashą pvz 123456
- `git log --oneline` pateikia logą trumpąją versiją
- komanda `git revert HASH` atstato pasirinktą versiją, versijos su "error entry" nebėra!
- po įvedimo "git revert hash" atsiranda langas, message langas (atitinka -m "...."), nes revert'inimas yra naujas commit

Git atsatatymas 2

- Tarkime jus labai daug darbavotės, turite n commit padarę ir surpatote, kad pirmas commit buvo geras, o po to viskas ne.
- "git reset –hard hash" komanda padaro hard-reset, t.y. resetina į pasirinktą commitą, tačiau ištrina viską, kas buvo daryta po to!
- revert'inti `git reset --hard HASH` neįmanoma, priešingai nei paties revert.

Git remote

- norint žinoti, ar lokali repozitorija yra susieta su nuotoline repozitorija (pvz Github, Bitbucket ar pan)
- `git remote`

- Nueiname kiekvienas į savo github ir ten susikuriame repo:
  - pavadinimas: test-repo
  - Description paliekam tučią
  - inicializuojame su
- Git Bash lange pakylame viena direktorija aukščiau, lauk iš "1 Įvadas" folderio su `cd ..`
- GitHub nusikopijuojame sukurtos repo HTPPS adresą
- Git Bash įrašome `git clone HTTPS`
- Dabartinėje direktorijoje atsirado test-repo folderis, keliaujame į jį `cd "test-repo"`
- `git remote`
- `git remote -v` parodo HTTPS

Git repo klonavimas

- Klonavimas reiškia, jog mes sukuriame remote repo kloną savo kompiuteryje, su visa Git istorija.
- Tarkime tai ne mūsų remote repo ir po klonavimo me skurį laiką nieko nedarėme. Galbūt tuo metu originalo autorius kažką pakeite, tada galima
- `git fetch origin`
- Tai persiunčia pakeitimus, bet nemerge'ina
- `git remote pull origin`
- atitinka fetch+merge

Git repo klonavimas

- Sukurkime test-repo direktorijoje naują failą
- `touch info.txt`
- `notepad info.txt` prirašome ko nors, išsaugome
- `git add info.txt`
- `git commit -m "pridetas info.txt failas"`
- dabar galime push'inti lokalius pakeitimus į github:
- `git push origin`
- pareikalavus įrašome github username ir userpassword
- GitHub atnaujiname (F5) ir voila, failas info.txt yra remote repozitorijoje

Lokalios repo sukėlimas į remote repo

- Nueiname kiekvienas į savo GitHub ir ten susikuriame repo:
  - pavadinimas: test-repo2
  - Description paliekam tučią
  - NE inicializuojame su README!!!!
- keliaujame į savo "1 Įvadas" direktoriją
- `git remote add origin HTPPS"`
- `git push origin master`

- Yra du metodai kaip sukurti GitHub repozitoriją
  1. Tiesiog sukurti naują repozitoriją
  2. "Fork" ("šakutinti") kito GitHub vartotojo jau egzistuojančią repozitoriją

Markdown sintaksė

- GitHub sukuriant repo, ją galima inicijuoti su readme.md
- .md reiškia, jog tai yra markdown formatas
- Markdown is a lightweight markup language with plain text formatting syntax. Its design allows it to be converted to many output formats, but the original tool by the same name only supports HTML. Markdown is often used to format readme files, for writing messages in online discussion forums, and to create rich text using a plain text editor. (https://en.wikipedia.org/wiki/Markdown)
- Labai trumpa pagalba dėl formatavimo: https://commonmark.org/help/
- Vėliau mes susipažinsime su RMarkdown

R paketai

- dauguma R paketų saugomi CRAN (Comprehensive R Archive Network), iš kur atsisiunčiamas ir pats R
- basinė R versija turi tik keletą naudingų paketų
- available.packages() funkcija, kuri surenką visą informaciją apie ezistuojančius R paketus @CRAN

```
a <- available.packages()
length(a)
```

- Šiuo metu : 228140 paketai
- taip pat galima ir iš github
- install.packages("ggplot")
- install.packages(c("ggplot", "dplyr"))
- iš R
- library(ggplot) čia nebereikia kabučių!
- search() parodo visus įjungtus paketus

R ir RStudio instaliavimas

- R reikia instaliuoti iš CRAN
- https://cran.r-project.org/
- Paleidžiame R
- Tam kad būtų lengviau dirbti su R, turėti aibę papildomų funkcijų, instaliuojame RStudio
- https://www.rstudio.com/products/rstudio/download/
- Startuojame RStudio

- Aprašomoji
- Tiriamoji
- Inferencinė
- Progozuojamoji
- Pražastiniai ryšiai
- Mechanistinė

Aprašyti duomenis

- Įprastai pirma statistinė analizė, kuri atliekama
- Nedaromos jokios išvados ar prognozės
- https://osp.stat.gov.lt/pagrindiniai-salies-rodikliai
- https://osp.stat.gov.lt/statistika-vizualiai

Tiriamoji analizė

- Naudojama aptikti sąsajoms tarp duomenų
- Padeda rasti kelią kuriuo galima judėti tyrime pirmyn
- EDA nenaudojama generalizuojant ar prognozuojant
- "Correlation does not imply causation"

Tikslas: naudojant maža dalį duomenų daryti išvadas apie bendrą populiaciją

- Inferencinė analizė - statistikos pagrindas
- Taikant inferencinę analizę siekiama nustatyti dominantį kiekį bei su prognoze susijusią paklaidą

Naudojant turimą informaciją apie tam tikrus opbjektus prognozuoti reikmšes kitiems objektams

- Jeigu X prognozuoja Y nereiškia, kad X iššaukia Y
- Prognozavimo taiklumas priklauso nuo teisingo matuojamų kintamųjų pasirinkimo
- Kuo daugiau duomenų ir kuo paprastesnis modelis!
- Fivethirthyeigh.blogs.nytimes.com
- AMAZON pirkiniai
- Reklama

Kas nutika vienam kitntamajam, kai pakeičiamas kitas kintamasis

- Reikalingos randomizuotos studijos
- Yra būdų kaip tai apeiti (ekonometrika magistre / PhD)
- Dažniausiai gaunami vidutiniai efektai
- Aksinis standartas
- MMA ....

Mechanistinė analizė: kaip ir kokie būtent pokyčiai vieno kintamojo keičia daro įtaką kitiems kintamiesiems (fizikos/inžinerijos sritis)

Duomenys

> *"Data are values of qualitative or quantitative variables, belonging to a set of items"*

wiki/data Kokybiniai: Šalis, lytis, vardas Kiekybiniai: Aukštis, svoris, spaudimas, kiekis

Kaip atrodo duomenys?

PAV, html, csv

- Svarbiausias aspektas - klausimas
- Antras pagal svarbumą - duomenys
- Dažnai duomenys apribos arba išlaisvins Jus, bet tik duomenys be klausimo, neišgelbės

Big Data

- Travers and Milgram 1696 Sociometry
- Leskovec and Horvitz WWW'08
- Dont use hadoo your data isnt that big
- "The data may not contain the asnwer. The combination of some data and an aching desire for an answer does not ensure that a reasonable answer can be extracted from a given body of data.." - John Tukey