Duomenų analizės įvadas

1. Dalis

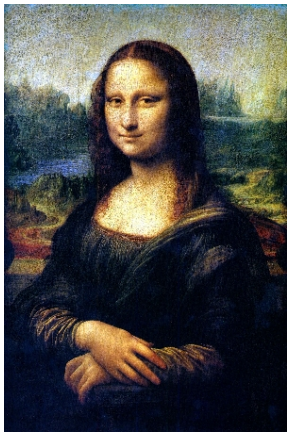
Justas Mundeikis

2019 m. vasario 11 d.

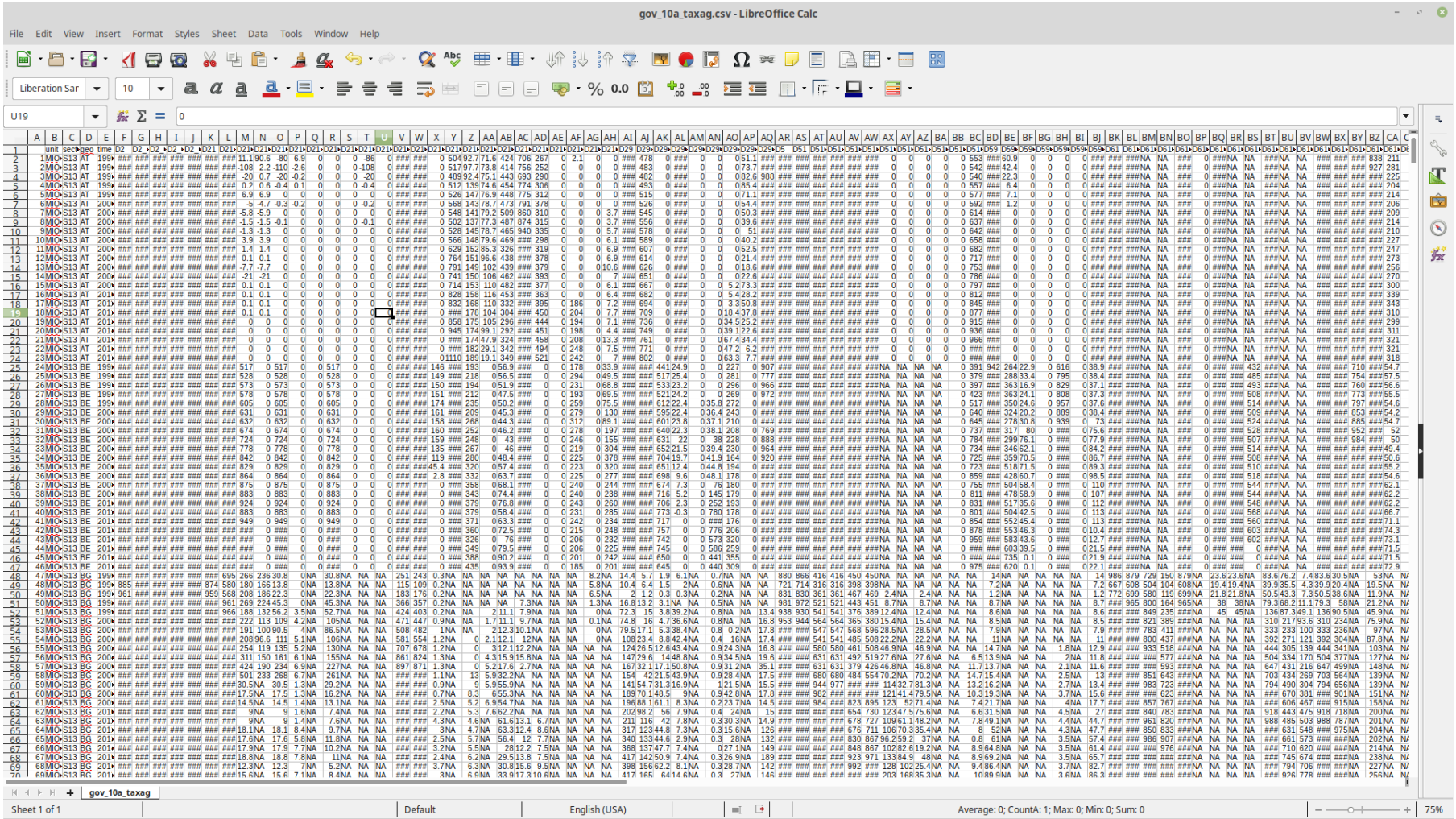
1. Dalies turinys

1. Įvadas į duomenų analizę
   - Duomenų analizės menas
   - Duomenų analizės epiciklai
   - Duomenų analizės klausimai
   - Duomenys, matavimo skalės
2. Command line interface
   - Intro
   - Direktorijos
   - CLI komandos
3. Git ir GitHub
   - Intro
   - Darbas su Git
   - Git branch
   - Markdown sintaksė
4. R
   - Įvadas i R

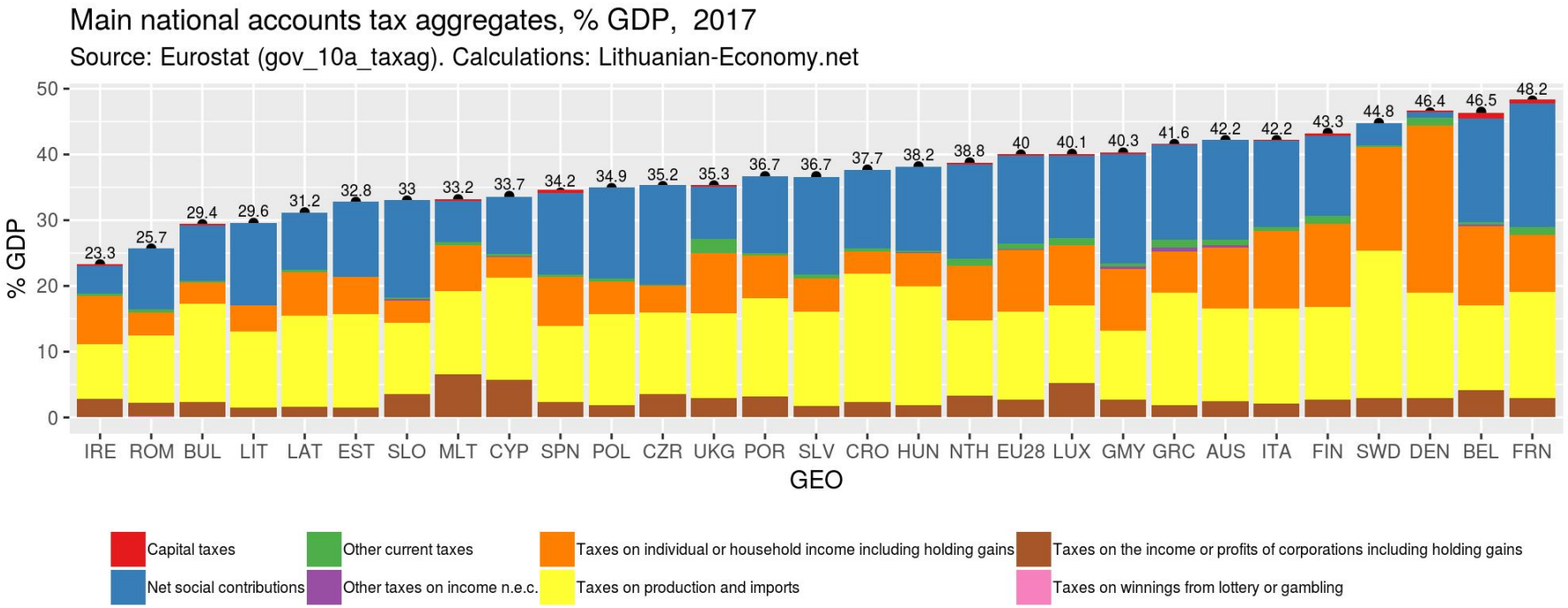
Duomenų analizės menas




Duomenų analizės menas



Duomenų analizės menas



Duomenų analizės menas

- *"Science is knowledge which we understand so well that we can teach it to a computer; and if we don't fully understand something, it is an art to deal with it."*
  Donald Knuth (1974) (Knuth: Computer Programming as an Art)
- Neegzistuoja jokio formalaus aprašymo, kaip reikia atlikti "duomenų analizę"
- Nors yra žinomi tam tikri įrankiai, statistiniai, ekonometriniai metodai, kuriais galima naudotis...
- kiekvieno "tyrėjo" (ekonomisto, duomenų analitiko, studento...) asmeninių pasirinkimų aibė nulemia atliekamos analizės kokybę bei naudą

Mokslinio tyrimo žingsniai

- Labai daug skaityti
- Išvystyti klausimą / hipotezę
- Nuspręsti kokia metodika bus taikoma
- Parengti duomenų surinkimo procesą (tyrimo protokolas)
- Surinkti duomenis
- Atlikti tiriamąją statistiką
- Atlikti aprašomąją statistiką
- Modeliuoti, atlikti prognozes
- Interpretuoti rezultatus
- Aprašyti tyrimo eigą bei rezultatus

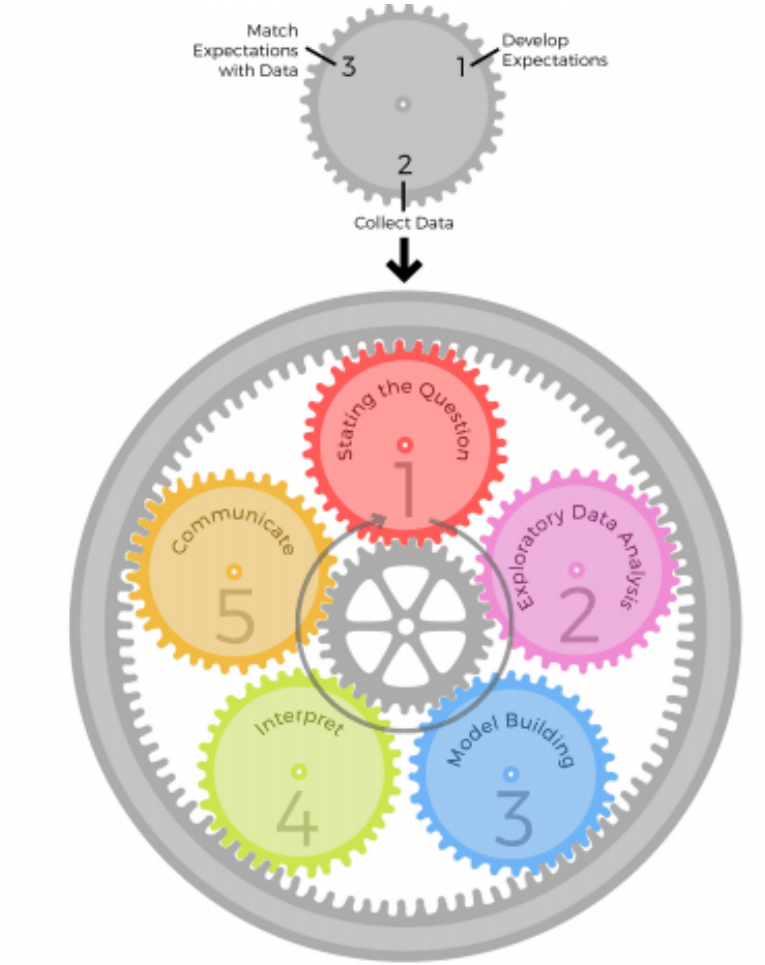
Duomenų analizės žingsnių epiciklai

1 lentelė: The Art of Data Science, Roger D. Peng & Elizabeth Matsui

Epycicles of analysis	**Set expectations**	**Collect information**	**Revise expectations**
Question	Question is of interest to audience	Literature search / Experts	Sharpen question
EDA	Data are appropriate for question	Make exploratory plots of data	Refine question or collect more data
Formal modeling	Primary model answers question	Fit secondary models, sensitivity analysis	Revise formal model to include more predictors
Interpretation	Interpretation of analyses provides a specific & meaningful answer to the question	Interpret totality of analyses with focus on effect sizes & uncertainty	Revise EDA and / or models to provide specific & interpretable answer
Communication	Process & results of analysis are understood, complete & meaningful to audiance	Seek feedback	Revise analyses or approach to presentation

Duomenų analizės žingsnių epiciklai

1 pav.: The Art of Data Science, Roger D. Peng & Elizabeth Matsui



6 Klausimų tipai

Remiantis R.Peng ir J.Leek (Science 2015) egzistuoja 6 klausimų tipai:

- Aprašomieji
- Tiriamieji
- Inferenciniai
- Progozuojamieji
- Pražastinių ryšių
- Mechanistiniai

6 Klausimų tipai

Aprašomieji klausimai:

- Kuriais siekiama gauti duomenų aprašymą, arba charakteristikų santraukas
- Nedaromos jokios išvados ar prognozės, nes patys rezultatai yra išvados per se
- Pvz., Moterų ir vyrų dalis tyrimo imtyje, vidutinis tiriamųjų amžius, vidutinė metinė infliacija, medianinės pajamos ir t.t.
- LSD šalies rodikliai
- LSD statistika vizualiai

6 Klausimų tipai

Tiriamieji klausimai

- Klausimai, kuriais siekiama nustatyti sąsajas bei trendus
- Padeda rasti kelią kuriuo galima judėti tyrime pirmyn, pvz., generuoti hipotezes
- Dažniausiai tokių klausimų atsakymui braižomi grafikai, padedantys surpasti duomenis
- Tačiau "Correlation does not imply causation" (žr sekanti skaidrė!!!)

6 Klausimų tipai

2 pav.: Šokolado vartojimas ir Nobelio prizai (Franz H. Messerli, M.D., 2012)

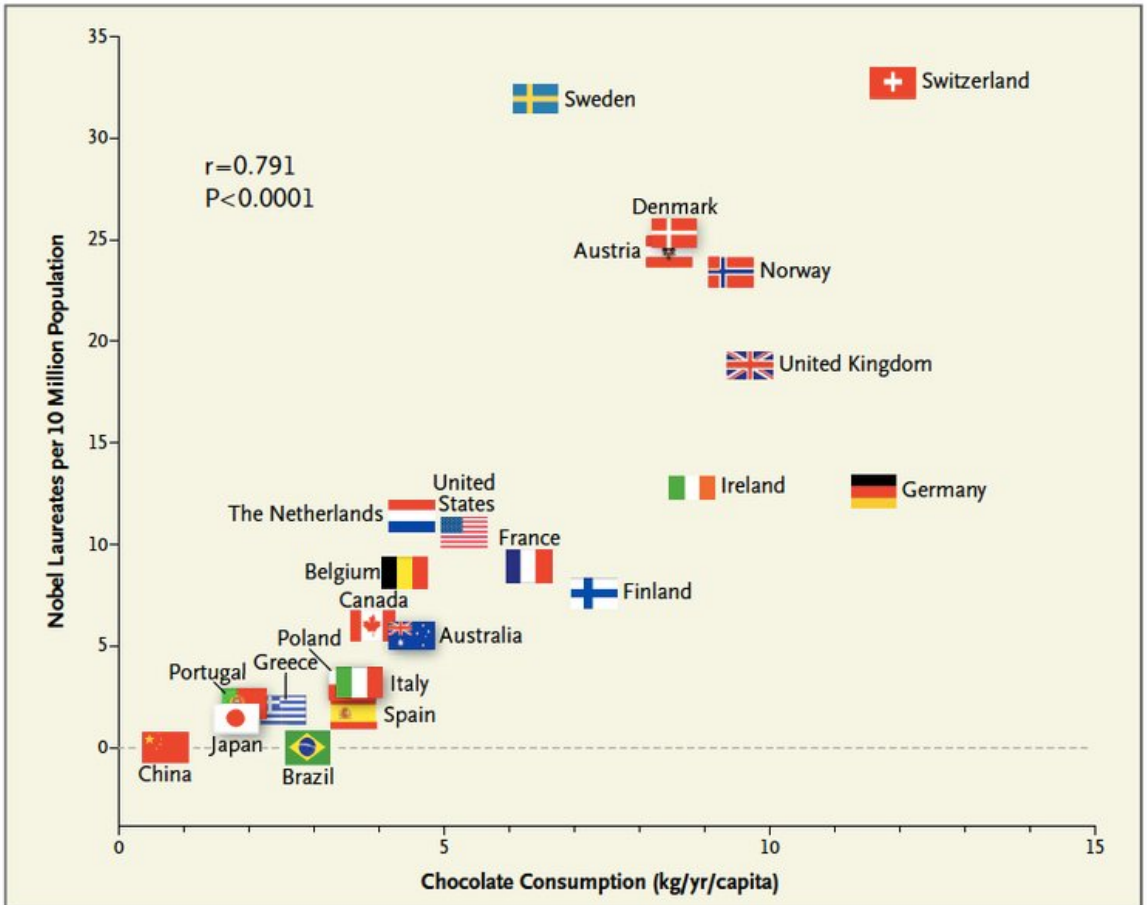


Figure 1. Correlation between Countries' Annual Per Capita Chocolate Consumption and the Number of Nobel Laureates per 10 Million Population.

6 Klausimų tipai

Inferenciniai klausimai:

- Klausimai, kuriais siekiama atsakyti klausimus apie bendrą populiaciją, tiriant tik imtį
- Pvz., Ekonomikos kurso 1 gr. baigiamasis pažymys 8. Ar visas 1 kursas gavo 8?
- Taikant inferencinę analizę siekiama nustatyti dominantį kiekį bei su prognoze susijusią paklaidą

6 Klausimų tipai

Prognozuojamieji klausimai

- Klausimai, kuriais siekiama "atspėti" ateitį
- Naudojant turimą informaciją apie tam tikrus objektus prognozuoti reikšmes kitiems objektams
- Svarbu: Jeigu X prognozuoja Y nereiškia, kad X iššaukia Y
- Prognozavimo taiklumas priklauso nuo teisingo matuojamų kintamųjų pasirinkimo
- Kuo daugiau duomenų ir kuo paprastesnis modelis!
- https://fivethirtyeight.com/
- AMAZON IBM E570

6 Klausimų tipai

Priežastinių ryšių klausimai:

- Klausimai, norint sužinoti, ar pakeitus vieną faktorių, kinta kitas faktorius
- How does a lack of sleep impact memory, problem solvingand critical thinking skills amongstcollege students?
- Reikalingos randomizuotos studijos
- Ekspertimentų galimybė ekonomikos šakoje ribota
- Yra būdų kaip tai apeiti (ekonometrika magistre / PhD)
- Dažniausiai gaunami vidutiniai efektai
- Siekiama atsakyti "ar" bet ne "kaip"

6 Klausimų tipai

Mechanistiniai klausimai

- Klausimai, kuriais siekiama nustatyti "kaip"
- Kaip ir kokie būtent pokyčiai vieno kintamojo keičia daro įtaką kitiems kintamiesiems (fizikos/inžinerijos sritis)

6 Klausimų tipai

- Svarbu suprasti, jog pvz., iškėlus prognozuojamąjį klausimą, tyrimo eigoje bus atsakyti ir į aprašomuosius, tiriamuosius, inferencinius klausimus

Koks yra geras klausimas?

Geras klausimas pasižymi šiomis savybėmis:

1. Klausimas turi būti įdomus tikslinei auditorijai
2. Klausimas dar neturi būti atsakytas
3. Klausimas turi būti logiškas / prasmingas (pagrįstas teorija)
4. Klausimas turi būti atsakomas (netinka: "Kokia yra gyvenimo prasmė?/ Ar egzistuoja dievas?"), kitaip tariant, turi egzistuoti duomenys ir metodikos, kurių pagalba būtų galima atsakyti į klausimą
5. Klausimas turi būti labai konkretus
   - Blogas klausimas: ar sveika mityba skatina ilgesnį gyvenimą
   - Geras klausimas: ar 250gr daržovių kasdien suaugusiam asmeniui padidina tikėtiną gyvenimo trukmę 10 metų?
   - Blogas klausimas: kas ekonomikos nuosmukio laikotarpiu nukenčia labiausiai
   - Geras klausimas: Kurioms iš soc grupių: bedarbiai, pensininkai, daugiavaikės šeimos per ekonominę 2008-2009 krizę labiausiai padidėjo rizika patirti santykinį skurdą

Duomenys, matavimo skalės

- Duomenys yra faktai arba skaičiai, kurie yra renkami, analizuojami bei apibendrinami pristatymo ar interpretavimo tikslais
- Duomenys surinkti tam tikro tyrimo metu vadinami duomenų set'u arba duomenų masyvu
- **Elementai** - subjektai, apie kurios renkami duomenys
- **Kintamasis** - elemento charakteristika
- Tyrimo metu surinkti **matavimai** apie visus dominančius elementus ir jų kintamuosius ir yra duomenys / duomenų set'as
- Duomenų set'as vieno elemento vadinamas **obzervacija**

Duomenys, matavimo skalės

Kintamųjų tipas apibrėžia informacijos kiekį slypinti duomenyse, bet kartu ir apriboja galimus taikyti statistinius metodus jų analizei.

- Kategoriniai kintamieji:
  - Nominalūs kintamieji: Lytis, Spalva
    Galima tik suskaičiuoti vienetus
  - Ranginiai kintamieji: Dydžiai S,M,L; Kredito reitingai F - AAA
    Juos galima prasmingai suranguoti!
- Kiekybiniai kintamieji:
  - Intervaliniai kintamieji: pažymiai (neturi 0)
    +,-, yra prasmingi, bet daugyba, dalyba nėra prasmingi
  - Santykiniai kintamieji: svoris, ūgis, atstumas (turi 0)
    +,-, daugyba, dalyba yra prasmingi

Duomenys, matavimo skalės

- 'Tarpsekciniai' duomenys (angl.: cross-sectional data): vienu ar panašiu metu užfiksuoti skirtingų elementų matavimai: pvz Europos šalių 2018m. BVP €
- Laiko eilučių duomenys (angl.: Time series data): Matavimai surinkti per du ar daugiau laikotarpių vienam elementui
- 'Tarpsekcinės' laiko eilutės (n elementų, t laikotarpį, taigi $n \times t$ matavimų)

Big Data

- Lietuvoje dauguma įmonių nelabai supranta ką reiškia "big-data"
- Big-data be AI perteklinis duomenų kaupimas
- Problema su AI - niekas nesupranta AI
- Tačiau su laiku AI keis ir ekonomikos mokslą:
- Video: AEA AFA Joint Luncheon - The Impact of Machine Learning on Econometrics and Economics
- John Tukey: "The data may not contain the answer. 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"

Apibendrinant:

- Svarbiausias duomenų analizės / tyrimo aspektas - klausimas!
- Antras pagal svarbumą - duomenys
- Dažnai duomenys apribos arba išlaisvins Jus, bet tik duomenys be klausimo, neišgelbės :D

Command Line interface (CLI)

Kiekviena operacinė sistema turi CLI:

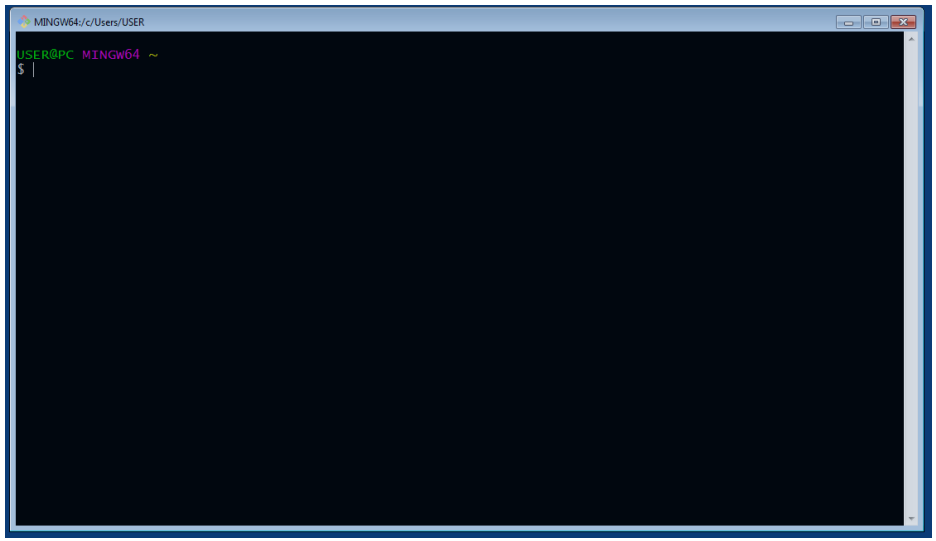
- Windows: Git Bash (), CMD
- Mac/ Linux: Terminal'as

Su CLI galima:

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

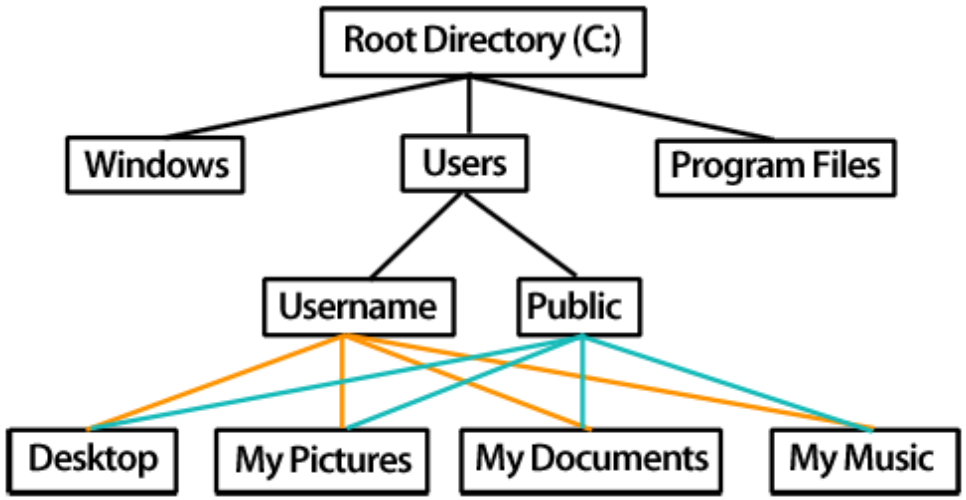
Intarpas GIT Bash instaliavimas

- 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 susideda iš: "command flag argument"
- komandos pvz: komanda liepianti atspausdinti kurioje direktorijoje esama: "pwd"

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

CLI komandos

- 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

VCS - Version Control System

- Kai darbuojatės ir rašote dokumentus, darote juose pakeitimus, dažnai turite 10-20 dokumentų pvz., `bakalaurinis.doc`, `bakalaurinis2.doc`, `bakalaurinis2019_01_02.doc`, `bakalaurinis2019_01_02(1).doc` ir t.t.
- dar panašiai tiek pat .xls failų, kiek mažiau skirtingų .ppt failų
- Galų gale tai veda link chaoso!
- Todėl labai svarbu, ypač dirbant tu tekstiniais dokumentais (ne Word'iniais), pvz., R skriptais, LaTex failais, turėti vieną failą, bet būti išsisaugojus kartu visas jo ankstesnes formas, galėti atstatyti.
- Bendradarbiaujant su kitais, taip pat svarbu galėti dalintis turimu dokumentu, kodu, leisti kitiems jį keisti, ir galų galų integruoti pakeitimus į motininį failą

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
- Populiariausia VCS
- Viskas išsaugoma lokaliai
- GIT naudojamas naudojant CLI, nors Windows yra ir GUI
- Dowload GIT

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
- Bendradarbiaujant tai padeda atpažinti kas padarė kokius pakeitimus

```
$ git config --global user.name "Justas Mundeikis"
$ git config --global user.email justas.mundeikis@evaf.vu.lt
$ 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",
- Mūsų tikslas sukurti aplanką "Sxxx" (stud.nr), o jame aplanką "test"
- Arba galima kurti pirmąjį aplanką, pereiti į jį ir tada kurti anrąjį, arba
- `mkdir -p s175/test`
- `cd s175/test`
- Inicializuojame git

```
USER@PC MINGW64 ~/Desktop/s175/test
$ git init
Initialized empty Git repository in C:/Users/USER/Desktop/↩
    s175/test/.git/

USER@PC MINGW64 ~/Desktop/s175/test (master)
```

- direktorijoje sukuriamas nematomas failas .git

Git kaip foto sesija

- GIT daro "pasirinktos direktorijos nuotraukas"
- 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 galima naudoti sąrašą kuris slepiasi ".gitignore" faile

Git

- su komanda touch sukuriame failą readme.txt
- `touch readme.txt`
- atidarysime failą su Notepad
- `notepad readme.txt`
- įrašome "change1" ir išsaugome failą "ctrl+s", uždarome failą

Git

- šis failas egzistuoja *repo*, 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" kitaip tariant ne ant "scenos"

Git

- su `git add readme.txt` įkeliame readme.txt į staging area
- patikriname statusą:

```
$ 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 readme.txt yra *staged* bet dar ne *commited*
- Ką reiškia, jog failas yra staged?

Git

- `notepad readme.txt`
- 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 ne su įrašu "change2"

Git

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

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

[master (root-commit) 30cd9e9] sukuriamas 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

- `git commit -m "...."`
- -m reiškia "message"
- turime apie 72 ženklus, bet galima ir daugiau
- geriausia naudoti esamąjį laiką

3 pav.: https://m.xkcd.com/1296/

Git

Toliau keičiame naudojamą failą:

- readme.txt prirašome "change3"
- `git add readme.txt`
- `git commit -m "pridedamas change 3"`
- Taigi jau padarėme 2 *commit* (= 2 nuotraukas)

Git

- Norint žinoti kas, kada ir 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

    pridedamas change3

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

    sukuriamas failas readme.txt
```

- *hash* (Secure Hash - SHA 'shah'): 1077a8e2e27424339d56a2dfd20b4aec56b0d3fb

Git

- Dabar sukursime naują failą *basic.R* ir *readme.txt* pridėsime change4
- `touch basic.R`
- `notepad readme.txt`
- readme.txt įrašome "change4", išsaugome, uždarome
- `git status` parodo, jog pakeistas *readme.txt* failas, ir untracked *basic.R* failas
- su komanda `git add .` staginame visus failus
- galimi kiti variantai : `git add -A` , `git add -u` (pastarasis naudojamas tik update'inti esamus failus)
- tada `git commit -m "pridedamas change4 ir sukuriamas failas basic.R`

Git

- Kartais yra tam tikri failai, kurių nenorime sekti (pvz duomenys, nereikalingi LaTex failai ir t.t.)
- Sukuriame duomenų failą
- `touch data.csv`
- `git status` parodo failą kaip *untracked*
- todėl sukuriame failą `touch .gitignore`, kuriame galima surašyti failų pavadinimus, arba failų tipą/galūnes, kurių nenorime trackinti
- `notepad .gitignore`
- *Notepad* 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 (branch'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*
- `git checkout master` visada sugrąžina atgal į *master* šaką

```
USER@PC MINGW64 ~/Desktop/s175/test (master)
$ git branch NewBranch

USER@PC MINGW64 ~/Desktop/s175/test (master)
$ git checkout NewBranch
Switched to branch 'NewBranch'

USER@PC MINGW64 ~/Desktop/s175/test (NewBranch)
```

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

Darbas Git atšakoje

Tarkime dabar asmuo B darbuojasi NewBranch atšakoje tobulindamas projektą

- `touch advanced.R`
- `notepad readme.txt` įrašome papildomą eilutę "change5"
- staginam ir commitinam pakeitimus:
  - `git add .`
  - `git commit -m "sukuriamas advanced.R ir pridedmas change5"`
- sugrįžtame į master atšaką su `git checkout master`
- Rezultatas: trūksta *advanced.R* failo, *readme.txt* turi tik 4 įrašus!
- Norint sujungti naują atšaką į *master* `git merge NewBranch`

Merge problemos

1. *master* atšakoje *readme.txt* sukuriame eilutę "change6", stage'inam ir commit'iname
2. nueiname į atšaką *NewBranch* ir ten esančiame *readme.txt* sukuriame "change7", stage'inam ir commit'iname
3. grįžtame į *master* atšaką `git chekckout master`
4. 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.
```

- gGchanget 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
```

- išsaugome pakeitimus ir tada
- `git commit -a -m "sujungtas failas is NewBranch bei pasalintas konfliktas`
- `git commit -a -m "..."` galima naudoti, jeigu nėra sukurtų naujų failų
- Yra papildomų įrankių, kurie padeda atlikti merge'inimo darbus, nes dažniausiai konfliktų visada bus

Git stash

- master atšakoje sukuriame failą *markdown.md*
- jeigu failo 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 .`, tada
- `git stash`, kas nukelia ne commitintą failą į *stashed area*. Failo neberodo darbalaukyje
- dabar galime darbuotis kitose atšakose ir vėl grįžus: `git stash apply` ir *markdown.md* failas vėl atsiranda baigus ji taisyti, galima commitinti.

Git atsatatymas 1

- Na štai po vidurnakčio pasidarbavus, padarėme klaidą:
- *readme.txt* papildome įrašu "error entry"
- `git commit -a -m "padareme 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 atstatymas 2

- Tarkime jus labai daug darbavotės, turite n commit padarę ir surpatote, kad pirmas commit buvo geras, o po to viskas ne.
- `git revert HASH` palieka buvusias versijas, bet šiuo atveju, jos nebereikalingos
- `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.
- Taigi geriau su hard-reset nesižaist :D

Git remote

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

Git repo klonavimas

- Nueiname kiekvienas į savo github ir ten susikuriame repo:
  - pavadinimas: test2
  - Description paliekam tuščia
  - inicializuojame su Readme.md
- Git Bash lange pakylame viena direktorija aukščiau, lauk iš "test" 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 "test2"`
- `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 mes kurį laiką nieko nedarėme. Galbūt tuo metu originalo autorius kažką pakeitė, 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
  - Description paliekam tučią
  - NE inicializuojame su README!!!!
- keliaujame į savo "test" 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

Įvadas į R

- R istorija
- R instaliavimas

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

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

Literatūra

- The Art of Data Science, R.Peng, E.Matsui 1-3 skyriai