

Uvod v obdelavo velepodatkov v Pythonu

Krajše izobraževanje (NOO)

Potek izobraževanja

■ Ponedeljek, 12. 05. 2025, 10:00-14:00

- Uvod in predstavitev predavatelja, dostop do učnih materialov
- Velepodatki, knjižnica Pandas in osnovne funkcionalnosti
- **10:00-12:00 – teoretični del v G3-Gauss**
- **12:00-14:00 – praktični del v G3-Gauss**

■ Torek, 13. 05. 2025, 10:00-14:00




- Pandas in velepodatki, prečiščevanje podatkov
- **10:00-12:00 – teoretični del v G3-Gauss**
- **12:00-14:00 – praktični del v G3-diplomska soba FERi, 1. nad.**

■ Sreda, 14. 05. 2025, 10:00-14:00

- Analiza in delo z velepodatki
- **10:00-12:00 – teoretični in praktični del v G3-Shannon**
- **12:00-14:00 – preverjanje znanja in anketa v G2-P1 Alfa**

Predstavitev predavatelja

- **Kontakt:**

-  mladen.borovic@um.si
-  +386 2 220 7460
-  [LinkedIn](#)

- **Raziskovalna področja:**

- Aplikacije umetne inteligence
- Priporočilni sistemi in iskalniki
- Obdelava naravnega jezika
- Detekcija podobnih vsebin
- Visokozmogljivo računalništvo (HPC)



dr. Mladen Borovič

Univerza v Mariboru

Fakulteta za elektrotehniko, računalništvo in informatiko

Inštitut za računalništvo

Laboratorij za heterogene računalniške sisteme

Motivacija in cilji izobraževanja

- Predmetniki študijskih programov **ne zajamejo** vseh aktualnih tematik
- Dodatna izobraževanja kot **alternativna oblika podajanja znanja**
- Krajši format, hiter in učinkovit pristop k pridobivanju novih znanj

- Cilji tega izobraževanja 🙌
 - Približati udeležencem tematike na področju **podatkovne znanosti in velepodatkov**
 - Naučiti udeležence **dobrih praks** obdelave velepodatkov
 - Demonstrirati pridobljena znanja **na praktičnih primerih**

- Na koncu izobraževanja se izvaja **preverjanje znanja**
 - Pogoji za pridobitev mikrodokazila (1 ECTS)
- **Anketa**
 - Povratna informacija o vaši izkušnji udeležbe na krajšem izobraževanju



Učni materiali in gradivo

■ GitHub - <https://github.com/lhrs-workshops/noo-uovp>


■ Prosojnice in zvezki Jupyter

- Prosojnice vsebujejo teoretični del in so barvno označene v zgornjem desnem kotu

 **Prvi dan** (12. 05. 2025)

 **Drugi dan** (13. 05. 2025)

 **Tretji dan** (14. 05. 2025)

- Zvezki Jupyter vsebujejo praktične primere z razlago
- Za izvedbo zvezkov Jupyter lahko uporabite platformo Google Colab  [Open in Colab](#)

Velepodatki

■ Kaj so velepodatki (ang. big data)?

*"Big data refers to data sets that are so **large, complex, or rapidly generated** that traditional data management and **processing tools struggle to capture, store, manage, and analyze them** within a **reasonable time frame**."*

■ Ko definiramo velepodatke, govorimo o **5V**:

- **Volume**: velikost podatkov
- **Velocity**: hitrost, s katero nastajajo novi podatki
- **Variety**: različni formati in viri podatkov
- **Veracity**: kvaliteta, pravilnost in zaupanje v podatke
- **Value**: skrita vrednost v podatkih in njihova uporabnost



The 5 Vs of Big Data

VELOCITY

- Batch
- Near time
- Real time
- Streams

VARIETY

- Structured
- Unstructured
- Semistructured
- All the above

VOLUME

- Terabytes
- Records
- Transactions
- Tables, files

VERACITY

- Trustworthiness
- Authenticity
- Origin, reputation
- Accountability

VALUE

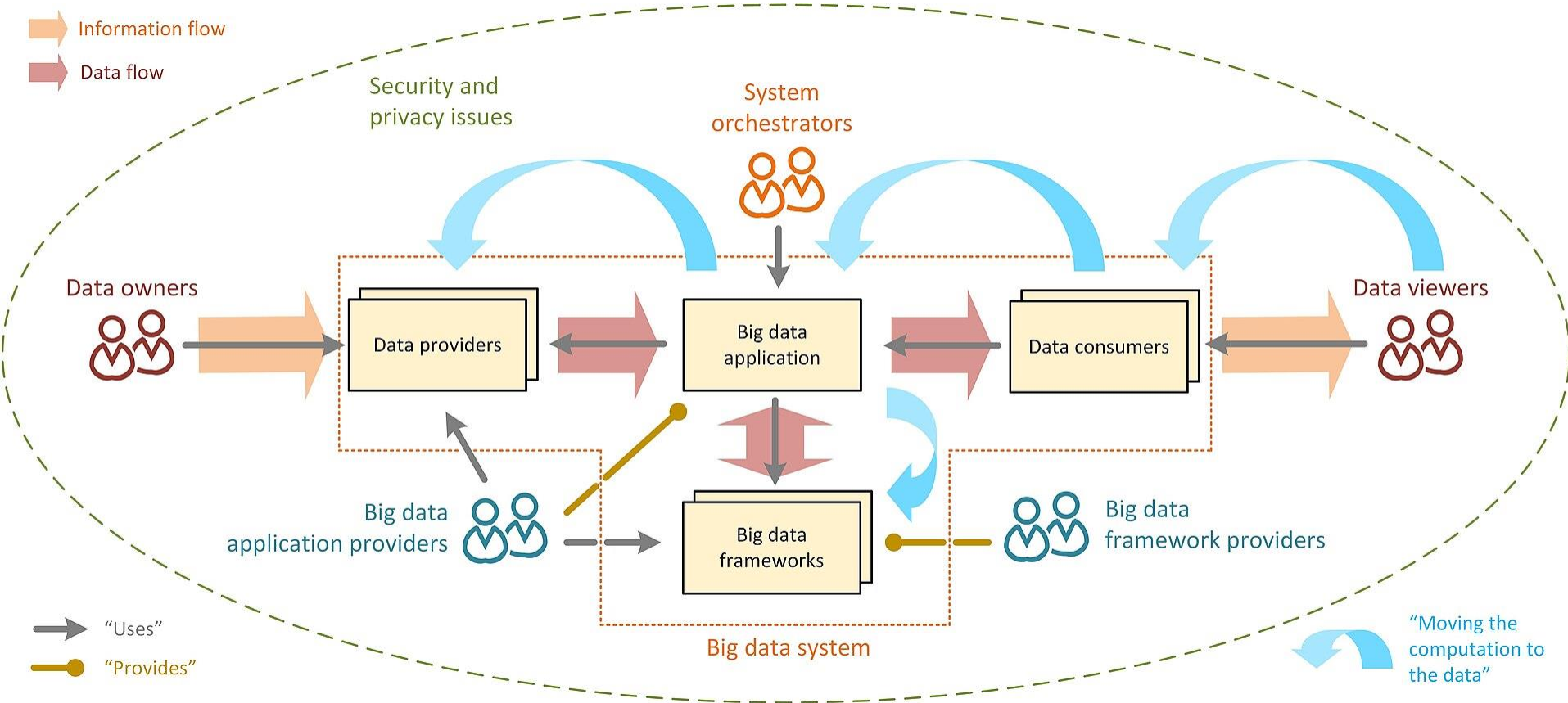
- Statistical
- Events
- Correlations
- Hypothetical

Velepodatki

- Kaj so velepodatki in kaj niso velepodatki?
- Velepodatki so:
 - Vsebine na družbenih omrežjih (slike, objave, komentarji, prijatelji, ...)
 - Podatki s senzorjev IoT
 - Satelitske slike in oddaljeno zaznavanje
 - Dnevnik ("logi") porazdeljenih spletnih mikrostoritev
- Velepodatki niso:
 - Rezultat ankete v obliki datoteke CSV
 - Excelova preglednica za upravljanje osebnih financ
 - Lokalna baza podatkov v skladišču
 - Dnevnik podatkov s senzorja temperature
- Pravilo "čez palec": ko podatkov ne moremo več obdelovati v delovnem pomnilniku (RAM-u) enega računalnika, potem govorimo o velepodatkih
- Ustrezna orodja za določeno velikost podatkov

Zakaj potrebujemo velepodatke?

- Včasih so podatki v oblikah, ki zavzamejo **ogromno sistemskih virov**
 - Slike v visoki ločljivosti, sekvence DNK, besedilo, ...
- Velepodatki so lahko ključnega pomena za ponudnike storitev
- **Analiza velepodatkov** v sklopu raziskav lahko vodi do novih spoznanj
 - Razvoj algoritmov v družbenih omrežjih, priporočilni sistemi, razvoj novih zdravil v medicini, ...
- Delo z velepodatki prinaša nove **izzive**
 - **Shranjevanje** – kako učinkovito, varno in trajno shraniti veliko količino podatkov?
 - **Obdelava** – kako učinkovito dostopati do velike količine podatkov in jih obdelati v sprejemljivem času?
 - **Računski viri** – skalabilni računalniški sistemi, ki lahko podprejo delo z velepodatki



Orodja za delo z velepodatki

- Tradicionalno se za delo s podatki uporabljajo **podatkovne baze**
 - **Relacijske** podatkovne baze (RDBMS – **R**elational **d**atabase **m**anagement **s**ystems)
 - MySQL, PostgreSQL, Oracle Database, Microsoft SQL Server, MariaDB, IBM Db2, SQLite
- Relacijske podatkovne baze imajo več pomanjkljivosti
 - Stara tehnologija
 - Relacijski model in SQL sta lahko težavna (zahteva se striktna strukturira podatkov)
 - Največ težav se pojavi pri skaliranju
- **Moderni pristopi** – NoSQL, porazdeljene podatkovne baze, "in-memory"
 - Apache Spark, Dask, Pandas, ...
 - Strukturirani in nestrukturirani podatki
 - Preprostost, "single-machine"
 - Podatki so shranjeni kar v RAM-u
- **Knjižnica Pandas**
 - Fokus tega izobraževanja
 - Popularno orodje za obdelavo podatkov (in tudi velepodatkov)
 - Tesna povezanost z modernimi aplikacijami podatkovne znanosti in umetne inteligence



Knjižnica Pandas

- Danes **najbolj uporabljena knjižnica** za delo s podatkovnimi zbirkami v **Pythonu**
- **Odprtokodna** knjižnica, ki se redno posodablja
- Podatkovna znanost, strojno učenje, učenje (globokih) nevronske mreže
- Dokumentacija in uporabne povezave
 - API reference (<https://pandas.pydata.org/docs/reference/index.html>)
 - User guide (https://pandas.pydata.org/docs/user_guide/index.html)
 - Pandas Cheat Sheet (https://pandas.pydata.org/Pandas_Cheat_Sheet.pdf)
 - Izvedba v brskalniku (<https://pandas.pydata.org/try.html>)
- Namestitev knjižnice v Pythonu
 - **PIP** => pip install pandas
 - **conda** => conda create -c conda-forge -n pandas_environment python pandas

Podatkovne strukture Pandas

- Najbolj **osnovna** podatkovna struktura v Pandas je stolpec (**Series**)
- Vsak stolpec vsebuje indeks, ki se uporabi pri izvedbi operacij nad podatki
- Indeks je privzeto številčnega tipa in se začne z 0, lahko pa ga tudi spremenimo

Series 1		Series 2		Series 3		Series 4	
INDEX	DATA	INDEX	DATA	INDEX	DATA	INDEX	DATA
0	A	A	1	0	[1, 2]	Jan-18	11
1	B	B	2	1	A	Feb-18	23
2	C	C	3	2	1	Mar-18	43
3	D	D	4	3	(4, 5)	Apr-18	21
4	E	E	5	4	{"a": 1}	May-18	17
5	F	F	6	5	6	Jun-18	6

- Za Series 1 => `pd.Series(data=['A', 'B', 'C', 'D', 'E', 'F'])`
- Za Series 2 => `pd.Series(data=[1, 2, 3, 4, 5, 6], index=['A', 'B', 'C', 'D', 'E', 'F'])`

Podatkovne strukture Pandas

Series 1		Series 2		Series 3		Series 4	
INDEX	DATA	INDEX	DATA	INDEX	DATA	INDEX	DATA
0	A	A	1	0	[1, 2]	Jan-18	11
1	B	B	2	1	A	Feb-18	23
2	C	C	3	2	1	Mar-18	43
3	D	D	4	3	(4, 5)	Apr-18	21
4	E	E	5	4	{"a": 1}	May-18	17
5	F	F	6	5	6	Jun-18	6

- Series lahko naredimo tudi s Pythonovo podatkovno strukturo [Dictionary](#)
 - S tem definiramo tudi lasten indeks!
 - list, NumPy array, ndarray
- Za Series 2 => `pd.Series(data={'A': 1, 'B': 2, 'C': 3, 'D': 4, 'E': 5, 'F': 6})`

Podatkovne strukture Pandas

- Najpogosteje uporabljena podatkovna struktura v Pandas je tabela (**DataFrame**)
- Tabelo sestavimo iz več stolpcev (**Series**)
- Tabela ima prav tako indeks, ki pa je **skupen vsem stolpcem**
- Posamezna vrednost v indeksu definira **vrstico**

Series 1			Series 2			Series 3			Dataframe			
INDEX	DATA		INDEX	DATA		INDEX	DATA		INDEX	SERIES 1	SERIES 2	SERIES 3
0	A		0	1		0	[1, 2]		0	A	1	[1, 2]
1	B		1	2		1	A		1	B	2	A
2	C	&	2	3	&	2	1	=	2	C	3	1
3	D		3	4		3	(4, 5)		3	D	4	(4, 5)
4	E		4	5		4	{"a": 1}		4	E	5	{"a": 1}
5	F		5	6		5	6		5	F	6	6

Podatkovne strukture Pandas

Series 1			Series 2			Series 3			Dataframe			
INDEX	DATA		INDEX	DATA		INDEX	DATA		INDEX	SERIES 1	SERIES 2	SERIES 3
0	A		0	1		0	[1, 2]		0	A	1	[1, 2]
1	B		1	2		1	A		1	B	2	A
2	C	&	2	3	&	2	1	=	2	C	3	1
3	D		3	4		3	(4, 5)		3	D	4	(4, 5)
4	E		4	5		4	{"a": 1}		4	E	5	{"a": 1}
5	F		5	6		5	6		5	F	6	6

■ Podobnost s tabelami v podatkovnih bazah

- **Bodite pozorni na tip podatkov!** Podatkovne baze imajo strogo definirano strukturo podatkov, v Pandas pa je to bolj ohlapno definirano

■ DataFrame lahko naredimo z združevanjem več Series ali pa tudi s:

- sezname (list), ndarray, Dictionary, seznam tuple

Podatkovne strukture Pandas

- Še ena podatkovna struktura Pandas je **Panel**, ki predstavlja več tabel (DataFrame) v zaporedju
- Analogija je:
 - 1D – Series (vektor)
 - 2D – DataFrame (matrika)
 - 3D – Panel (tenzor)
- Panel je uporabna podatkovna struktura za prikaz sprememb tabelaričnih podatkov skozi čas (kar je lahko zelo specifično)
- Omejenost uporabe na 3 dimenzije
 - V primeru večdimenzionalnih podatkov je bolj smiselno uporabiti tenzorje (ang. tensor)
 - To je še posebej uporabno pri učenju globokih nevronske mreže
- Panel se danes uporablja redkeje, saj obstaja učinkovitejša podatkovna struktura – **MultiIndex**, ki uporablja DataFrame in napredno indeksiranje

Izbiranje podatkov in filtriranje

- Podatke v podatkovnih strukturah Pandas lahko **izbiramo**, **filtriramo** ali drugače **režemo** (ang. slicing)

- Te operacije so podobne ukazom v **SQL**

```
SELECT * FROM [tabela] WHERE [pogoji]
```

- Več različnih načinov izbiranja in filtriranja podatkov v Series in DataFrame

- Imejmo DataFrame z imeni in starostmi

```
df = pd.DataFrame({ 'Ime': ["Alenka", "Bojan", "Ciril"], 'Starost': [25, 26, 23] })
```

- Operator **[]**

```
df['Ime'] => vrne Series (stolpec) z vsemi imeni
```

```
df[['Ime']] => vrne DataFrame z vsemi imeni
```

```
df[['Ime', 'Starost']] => vrne DataFrame z vsemi imeni in starostmi
```

```
df[0:1] => vrne DataFrame z eno vrstico, od indeksa 0 naprej
```

```
df[1:] => vrne DataFrame z vsemi vrsticami od indeksa 1 naprej
```

Izbiranje podatkov in filtriranje

■ Metodi `.loc[]` in `.iloc[]`

- `.iloc[]` za celoštevilčne indekse, `.loc[]` za oznake

■ `.iloc[]`

`df.iloc[0]` => vrne Series, ki predstavlja prvo vrstico

`df.iloc[0:2]` => vrne DataFrame s prvima dvema vrsticama od indeksa 0 naprej

`df.iloc[2, 1]` => vrne podatek iz 3. vrstice in 2. stolpca

■ `.loc[]`

`df.loc[:, 'Ime']` => vrne Series z vsemi imeni (stolpec Ime)

`df.loc[:, 'Ime':'Starost']` => vrne DataFrame z vsemi imeni in starostmi

`df.loc[[0, 2], ['Ime']]` => vrne DataFrame z imeni v 1. in 3. vrstici

■ `.loc[]` lahko **kombiniramo** tudi s celoštevilčnimi indeksi

- `df.index` in `df.columns`

`df.loc[df.index[0], 'Ime']` => vrne vrednost stolpca Ime v 1. vrstici

`df.loc[2, df.columns[1]]` => vrne vrednost 1. stolpca (v našem primeru Ime) v 3. vrstici

Izbiranje podatkov in filtriranje

■ Logično indeksiranje

- V selektor podamo logični izraz, ki izvede filtriranje

`df[df['Starost'] > 20]` => vrne DataFrame, ki vsebuje vse vrstice, kjer je starost > 20

`df[df['Ime'] == "Alenka"]` => vrne DataFrame, ki vsebuje vse vrstice, kjer je ime Alenka

■ Metoda `.query()`

- Sintaksa je zelo podobna SQL

`df.query("`Starost` > 20")` => vrne DataFrame, ki vsebuje vse vrstice, kjer je starost > 20

`df.query("`Ime` == 'Alenka'")` => vrne DataFrame, ki vsebuje vse vrstice, kjer je ime Alenka

`df.query("`Ime` == 'Alenka' & `Starost` > 20")` => vrne DataFrame, ki vsebuje vse vrstice, kjer je ime Alenka in starost > 20

- **Bodite pozorni na posebne narekovaje `` ob imenih stolpcev!**
 - Ti narekovaji niso obvezni, vendar lahko neuporaba vodi v nepričakovano delovanje!
- Metoda `.query()` spada med naprednejše metode knjižnice Pandas

Nalaganje in shranjevanje

- V Pandas lahko podatke naložimo iz **različnih formatov**
- Daleč najbolj pogost format je **CSV** (ang. comma-separated values)
- CSV in JSON - zelo pogosta formata za izmenjavo podatkov v podatkovni znanosti
- Enostavno nalaganje podatkov
 - `df = pd.read_csv('podatki.csv')` => naloži datoteko 'podatki.csv' v DataFrame
 - `df = pd.read_json('podatki.json')` => naloži datoteko 'podatki.json' v DataFrame
- Dodatni parametri
 - `df = pd.read_csv('podatki.csv', sep='\t', usecols=['Ime', 'Starost'])`
 - Naloži datoteko 'podatki.csv', pri tem upošteva **tabulator kot separator**
 - Torej, gre za datoteko TSV (ang. tabulator-separated values)
 - Dodatno **naloži samo stolpca Ime in Starost**

Nalaganje in shranjevanje

■ Shranjevanje podatkov

`df.to_csv('podatki.csv')` => shrani DataFrame v datoteko 'podatki.csv'

- Pri tem se shrani tudi indeks, kar ni vedno zaželeno!

`df.to_csv('podatki.csv', index=False)` => shrani DataFrame v datoteko 'podatki.csv' brez indeksa

- Manjša datoteka CSV; smiselno, ker `.read_csv()` privzeto ustvari indeks

■ Podobno velja za ostale formate

- Splača se preveriti posebnosti
- Shranjevanje v JSON ne shrani indeksa

Uporabne funkcije in atributi

- `.head()` vrne prvih N vrstic v DataFrame-u
`df.head()` => prikaže prvih 5 vrstic
`df.head(10)` => prikaže prvih 10 vrstic
- `.tail()` vrne zadnjih N vrstic v DataFrame-u
`df.tail()` => prikaže zadnjih 5 vrstic
`df.tail(10)` => prikaže zadnjih 10 vrstic
- `.shape` vrne dimenzije DataFrame-a
`df.shape` => vrne (10, 5) za DataFrame z 10 vrsticami in 5 stolpci
- `.info()` vrne informacije o podatkovnih tipih, porabi pomnilnika, vrednosti
`df.info()`
- `.describe()` vrne statistiko številčnih vrednosti v DataFrame-u
`df.describe()`

Pandas in velepodatki

- Velepodatki obsegajo na milijone pa tudi milijarde ali več zapisov
 - Časovna zahtevnost nalaganja, branja, filtriranja, obdelovanja in shranjevanja
- Tradicionalne metode za delo s podatki niso dovolj optimalne za velepodatke
- Zakaj Pandas in ne podatkovne baze?
 - Splošna razširjenost, popularnost, uporabne funkcionalnosti, enostavnost uporabe
 - Povezava z drugimi knjižnicami za podatkovno znanost (scikit-learn, Tensorflow, PyTorch, ...)
 - Kar v Pandas naredimo v ~10 vrsticah, z drugimi tehnologijami naredimo v ~100+ vrsticah kode
- Primarni namen podatkovnih baz je trajna hramba podatkov in hitro branje

Pandas in velepodatki

- Pri obdelavi in analizah si želimo **hitro branje** in **učinkovito manipulacijo**
 - **SQL** lahko postane izredno kompleksen
 - **Dodatno indeksiranje** in druge posebnosti podatkovnih baz nam lahko delajo težave
- Kljub prednostim Pandas lahko naletimo na težave **(če nismo previdni)**
 - Prekoračitev delovnega pomnilnika
 - Počasna izvedba operacij za obdelavo podatkov
- **Z dobrimi praksami združimo učinkovitost Pandas in optimalnost operacij**

Nalaganje podatkov v kosih

- Namesto branja vseh podatkov, jih preberemo v **manjših kosih**
- Ti kosi so bolj **obvladljivi** in ne zahtevajo toliko sistemskih virov
- Nalaganje podatkov **v kosih** (ang. chunking) je podprto v metodah Pandas
iterator = pd.read_csv('podatki.csv', chunksize=10000) => prebere prvih 10.000 zapisov
 - V tem primeru .read_csv() vrne iterator
 - Za nadaljnjo obdelavo potrebujemo torej zanko for, ki uporablja iterator

Selektivno nalaganje

- Če poznamo strukturo velepodatkovne zbirke, potem lahko izberemo tiste podatke, ki jih dejansko potrebujemo za nadaljnje delo (in ne vseh podatkov)
- Primer: velepodatkovna zbirka z 200 stolpci, zanima nas samo 5 stolpcev
 - Ne naložimo vseh 200 stolpcev, temveč samo tistih 5, ki nas zanimajo
- Selektivno nalaganje dosežemo z uporabo parametra `usecols`
`df = pd.read_csv('podatki.csv', usecols=['Ime', 'Starost'])` => naložimo samo imena in starosti

Selektivno nalaganje

- Kaj pa, če ne poznamo strukture velepodatkovne zbirke?
- 💡 Trik #1: uporabimo Pythonov modul `csv` in izluščimo stolpce

```
import csv
```

```
cols = csv.DictReader(open('podatki.csv', 'r')).fieldnames => vrne seznam stolpcev v velepodatkovni zbirki
```

- 💡 Trik #2: iznajdljiva uporaba parametra `nrows` v `.read_csv()`

```
cols = pd.read_csv('podatki.csv', nrows=0).columns.tolist() => vrne seznam stolpcev v velepodatkovni zbirki
```



Optimizacija podatkovnih tipov

- **Dejstvo:** veliko podatkovnih znanstvenikov in analitikov **nima formalne izobrazbe iz področja računalništva**
 - Povečini se ti kadri sproti **priučijo** znanj računalništva, ki so vezana na njihovo delo
 - **Ponavadi se osnove izpustijo**, kar je lahko ključnega pomena za **optimalnost kode**
- S **pametno izbiro** podatkovnih tipov zmanjšamo porabo sistemskih sredstev
 - Pri delu z **velepodatki** je to še toliko bolj pomembno
- Podatki v neoptimalno izbranih podatkovnih tipih zasedajo več pomnilnika
 - Delovni (RAM) in trajni (trdi disk)
 - Priložnost za **optimizacijo** (pohitritev in zmanjšanje porabe pomnilnika)
- Potrebno je dobro poznavanje podatkovnih tipov in njihova velikost

Optimizacija podatkovnih tipov

- Imejmo datoteko 'podatki.csv', ki med drugim vsebuje stolpec **Starost**
 - Če datoteko CSV naložimo **brez izbire** podatkovnih tipov, pride do **avtomatske detekcije**
 - Pandas dodeli **podatkovni tip int64**, ki je največji celoštevilčni podatkovni tip v Pythonu
 - **Zakaj? Pandas ne ve nič o podatkih in predvideva maksimalno porabo pomnilnika**
- Starost je podatek **celoštevilčnega** tipa, ki je vedno pozitiven in omejen
 - Interval **[0, ~120)** => noben ni star manj kot 0 let, malo jih je starih več kot 120 let
 - Podatkovni tip int64 je definiran na intervalu **(-9223372036854775808, 9223372036854775807)**
 - **Dodatno ta tip ni striktno omejen (Python 3), temveč ga omejuje velikost pomnilnika (torej lahko hrani še večje vrednosti!)**
 - Vsako celo število shranjeno s tipom int64 tako zaseda 64 bitov pomnilnika (8 B)
 - **Imejmo milijardo zapisov => stolpec Starost bo zasedal 8 GB!**
 - Jasno je, da int64 ni optimalni tip za podatek o starosti
- Izbira optimalnega tipa
 - Smiselno je tip **uint8** (8-bitno celo število brez predznaka) => interval **[0, 255]**
 - Vsako celo število shranjeno s tipom uint8 tako zaseda 8 bitov pomnilnika (1 B)
 - Imejmo milijardo zapisov => **stolpec Starost bo zasedal 1 GB (prihranili smo 7 GB!)**

Optimizacija podatkovnih tipov

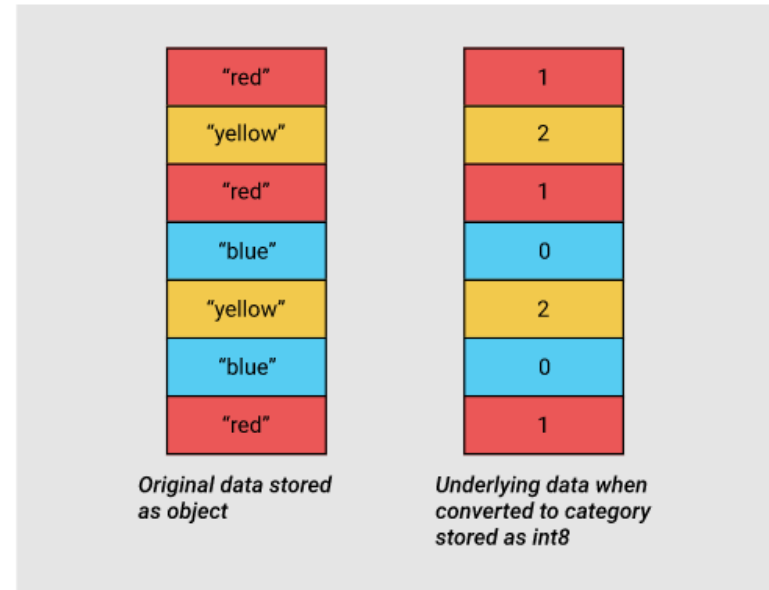
- Podobno logiko lahko uporabimo tudi za ostale podatkovne tipe
 - float64 (8 B) => float (4 B)
 - int8 (1 B) z vrednostmi 0 in 1 => bool (1 b)
 - char, object => **category**

- Podatkovne tipe lahko spremenimo **po nalaganju** ali pa **ob nalaganju** podatkov

- Parameter **dtype** => podamo Dictionary s preslikavo podatkovnih tipov
`df = pd.read_csv('podatki.csv', dtype={'Starost': uint8})`

- **Avtomatska pretvorba numeričnih** podatkovnih tipov po nalaganju

- Pandas lahko avtomatsko poskusi določiti **najbolj ustrezen numerični** podatkovni tip
`df['Starost'] = pd.to_numeric(df['Starost'], downcast='unsigned')`



Drugi tipi izboljšav

- Odvisno od podatkov je smiselno uporabiti tudi druge tipe izboljšav
- Za podatke **v redki obliki** (ang. sparse data) => primerne podatkovne strukture
 - `SparseDataFrame`, `SparseArray`, `SparseDtype`,
- Obstajajo tudi drugi podatkovni formati
 - CSV in JSON sta najbolj pogosto uporabljena, vendar nista optimalna
 - `Parquet`, `Feather`, HDF5, ... (https://pandas.pydata.org/docs/user_guide/io.html)
- **Kompromis** med hitrostjo branja/shranjevanja, velikostjo in prenosljivostjo

Predobdelava podatkov

- Velepodatkovne zbirke je načeloma **vedno treba obdelati** pred analiziranjem
 - Avtorji podatkovnih zbirk ne pripravijo podatkov v ustreznih oblikah (**nekateri tudi namerno!**)
 - Manjkajoči podatki, napake in anomalije, duplikati
- **Predobdelava** je ključen korak pri obdelavi velepodatkov
- Vedno predpostavimo, da se v podatkih pojavljajo napake
 - Pregled podatkov, **EDA – ang. Exploratory Data Analysis**
 - Pandas ima nabor uporabnih funkcij za predobdelavo podatkov
- Odvisno od nadaljnjega dela s podatki se odločimo o **strategiji predobdelave**
 - Ali lahko ignoriramo napake v podatkih?
 - Ali interpoliramo manjkajoče podatke?
 - Ali lahko zmanjšamo obseg podatkov tako, da uporabimo le "čiste" podatke?

Manjkajoči podatki

- `.isna()` vrne DataFrame, kjer True pomeni manjkajoč podatek
`df.isna()`
- `.notna()` inverzna funkcija `.isna()`
`df.notna()`
- `.dropna()` odstrani vse manjkajoče podatke iz DataFrame-a
`df.dropna()`
- `.fillna()` zamenja manjkajoče podatke s podano vrednostjo
`df.fillna(0)` => zamenja manjkajoče podatke s številom 0
`df.fillna(value={"A": 0, "B": 1})` => zamenja manjkajoče podatke v stolpcu A z 0 in v stolpcu B z 1

Uporabne funkcije

- **.apply()** izvede podano funkcijo nad elementi v DataFrame-u
 - df.apply(np.sqrt) => izvede funkcijo np.sqrt() nad vsemi elementi
 - df.apply(np.sum, axis=0) => izvede funkcijo np.sum() nad stolpci
 - df.apply(np.sum, axis=1) => izvede funkcijo np.sum() nad vrsticami
 - df.apply(lambda x: '-', axis=0) => postavi vse vrednosti stolpcev na '-'
- **Vektorske operacije** vs. **zanke** v Pythonu
 - Če uporabljamo Pandas se načeloma poslužujemo **vektorskih operacij**
 - Zanke v Pythonu so tudi možnost, vendar bistveno **upočasni delovanje**
 - Vektorske operacije so izredno optimizirane
 - **Namig:** vzporedno s Pandas lahko uporabljamo tudi funkcije knjižnice NumPy
- **.replace()** zamenja vrednost v DataFrame-u
 - df.replace('-', pd.NA) => zamenja vse vrednosti '-' s konstanto pd.NA

Podvajanje in deduplikacija

- `.duplicated()` vrne Series s True, kjer obstaja podvajanje
`df.duplicated()`
- `.drop_duplicates()` odstrani duplikate iz DataFrame-a
`df.drop_duplicates()`
- `.reset_index()` ponovno zgradi indeks v DataFrame-u
 - To je uporabno kadar smo z več zaporednimi operacijami **spremenili obliko** DataFrame-a
 - Začetni DataFrame smo samo **filtrirali** => **indeks je ostal enak**, kar lahko vodi v neoptimalnost
 - S **ponovno gradnjo indeksa** dobimo nov indeks, dostop do podatkov v DataFrame-u je **optimalen**`df.reset_index()`

Analiza in delo z velepodatki

- Opremljeni z znanjem o predobdelavi podatkov se lotimo dela z velepodatki
 - [Predobdelava](#) > [Analiza](#) > [Rezultati](#)
- Analiza velepodatkov z agregatnimi funkcijami
 - [.groupby\(\)](#), [.sum\(\)](#), [.min\(\)](#), [.max\(\)](#), [.mean\(\)](#), [.value_counts\(\)](#), [.unique\(\)](#)
- Vizualizacija z Matplotlib
 - [.plot\(\)](#) [funkcija v Pandas]
- Praktični primeri
 - 5. zvezek Jupyter (Analiza in delo z velepodatki)

Analiza in delo z velepodatki

- Sintetična velepodatkovna zbirka UK Population
- Naložite velepodatkovno zbirko in preverite porabo delovnega pomnilnika
- Predobdelava podatkov
 - Lastne funkcije ali obstoječe funkcije, ki izvajajo vektorske operacije?
- Optimizacija podatkovnih tipov
 - Katere izboljšave so smiselne?
 - Za koliko lahko izboljšamo porabo delovnega pomnilnika?

Analiza in delo z velepodatki

- Katera okrožja s prebivalci v Veliki Britaniji se nahajajo severno od okrožja York?
 - Rezultat naj bo seznam okrožij
- Razmislite katere korake bo potrebno izvesti, da dobimo rezultat



Analiza in delo z velepodatki

- V katerih okrožjih v Veliki Britaniji živijo ljudje z imenom Robert?
 - Rezultat naj bo slovar, ki je padajoče urejen po številu oseb z imenom Robert v vsakem okrožju
 - Ključ v slovarju naj bo okrožje, vrednost pa pripadajoče število oseb z imenom Robert
- Razmislite katere korake bo potrebno izvesti, da dobimo rezultat?
 - Namig: agregatne funkcije so zelo učinkovite za reševanje takšnih problemov 😊



Analiza in delo z velepodatki

- Zanima nas 10 največjih okrožij v Veliki Britaniji po številu prebivalcev.
 - Rezultat naj bo v obliki stolpčnega grafikona, z okrožji urejenimi v padajočem vrstnem redu po številu prebivalcev
- Razmislite katere korake bo potrebno izvesti, da dobimo rezultat?
 - Namig: agregatne funkcije so zelo učinkovite za reševanje takšnih problemov 😊
 - Namig: vizualizacijo lahko naredimo s funkcijo `.plot()` v Pandas

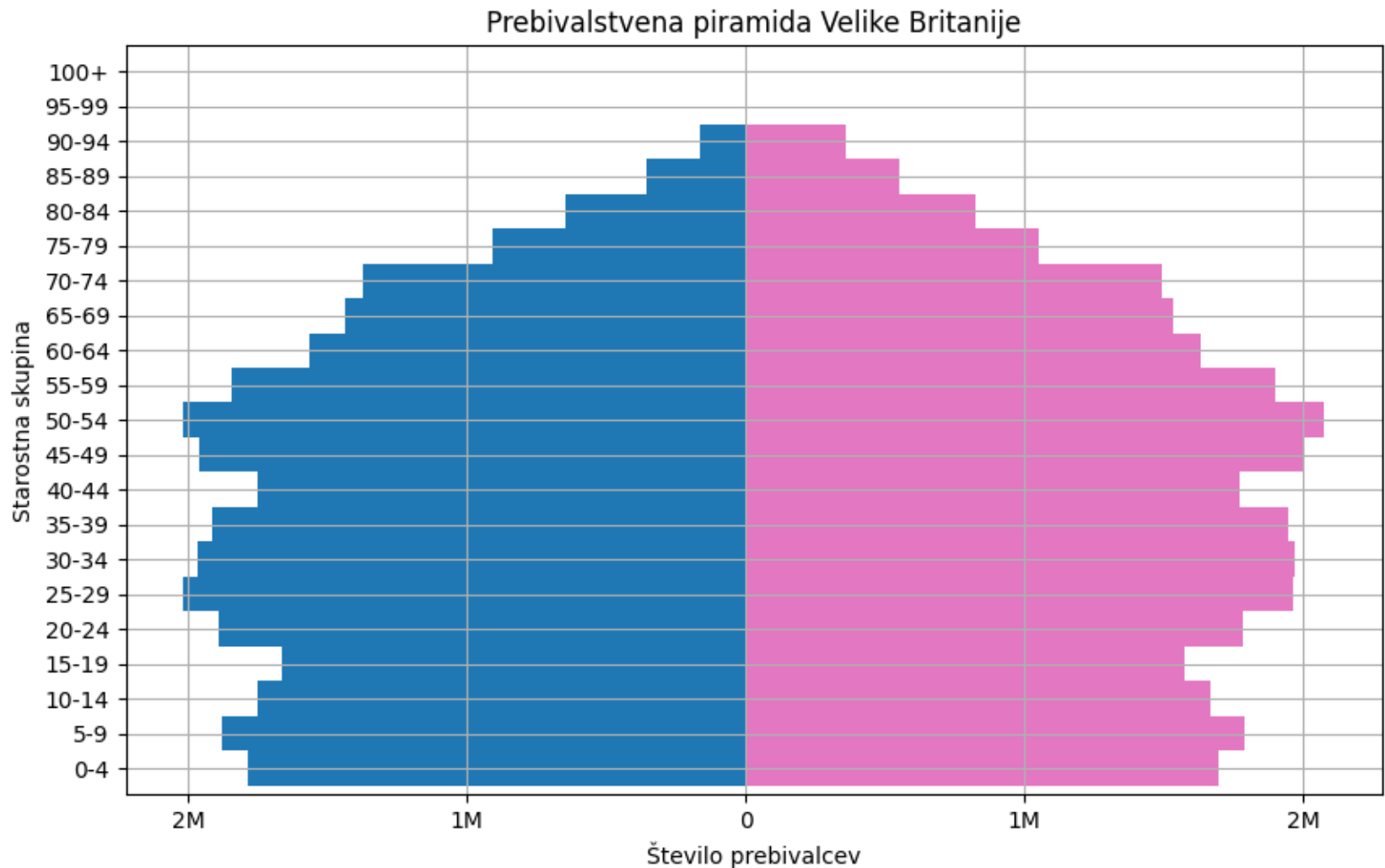


Analiza in delo z velepodatki

- Podatke o prebivalstvu želimo prikazati v obliki prebivalstvene piramide
 - Starostne skupine naj obsegajo obdobje 5 let (npr. 0-4, 5-9, 10-14, ...)
 - Na levi strani naj bodo z modro prikazani podatki o moških, na desni pa z roza o ženskah
- Razmislite katere korake bo potrebno izvesti, da dobimo rezultat?
 - Namig: DataFrame s podatki bo potrebno razdeliti na starostne skupine (nov stolpec?)
 - Namig: agregatne funkcije so zelo učinkovite za reševanje takšnih problemov 😊
 - Namig: vizualizacijo lahko naredimo s funkcijo `.plot()` v Pandas
 - Namig: levo stran prebivalstvene piramide lahko dobimo tako, da vrednosti pomnožimo z -1



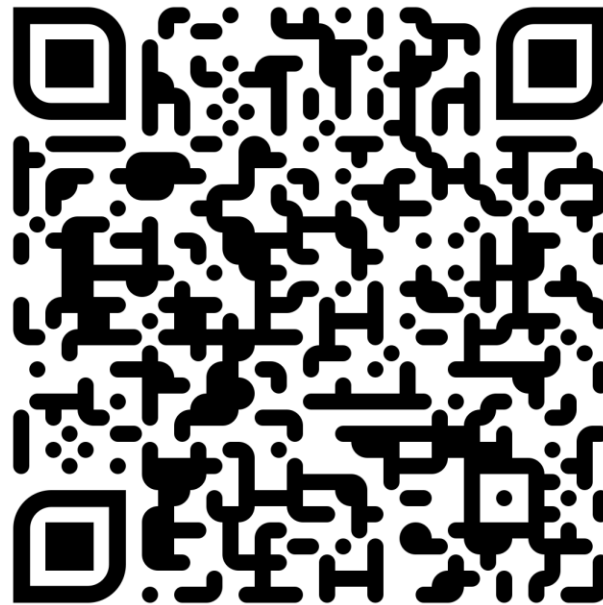
Analiza in delo z velepodatki



Zaključek

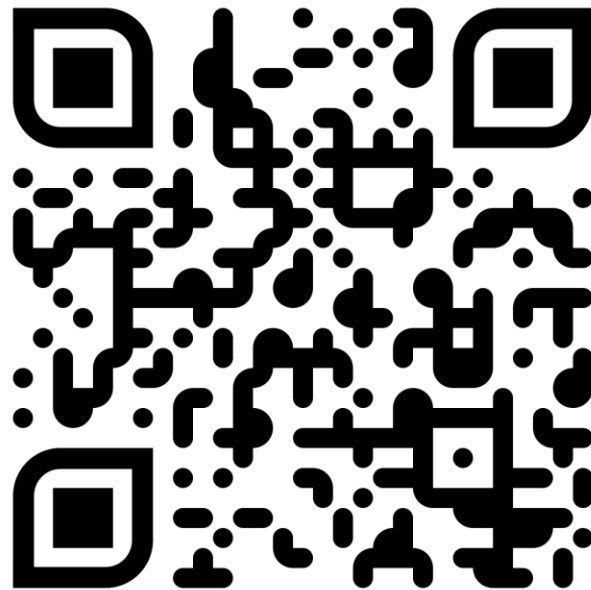
- V sklopu izobraževanja ste spoznali podrobnosti pri **delu z velepodatki**
 - Izzivi pri nalaganju, predobelavi in analizi velepodatkov
- Knjižnica **Pandas** je samo eno izmed modernih orodij za delo z velepodatki
 - Polars, cuDF (NVIDIA RAPIDS), Dask, FireDucks, Apache Spark
- Velepodatki nas spodbujajo k **optimalnim pristopom**
 - Poznavanje podatkovnih tipov
 - Inženirska iznajdljivost
- **Samostojno predelajte še ostale praktične primere v zvezkih Jupyter**

Preverjanje znanja



<https://classroom.github.com/classrooms/188869980-uovp-noo-2025>

Anketa



<https://forms.gle/CTWw79jEdwkb8FNaA>

Vabljeni še na druga izobraževanja

- **Uvod v obdelavo velepodatkov v Pythonu (12.-15. 05. 2025)**



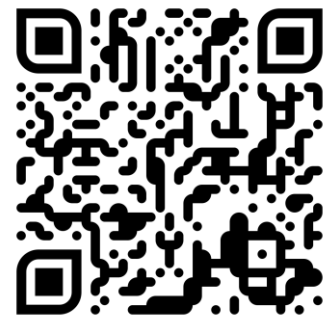
- **Napredna obdelava velepodatkov v Pythonu (02.-04. 06. 2025)**

- <https://krajsa-izobrazevanja.feri.um.si/NOVP>



- **Uvod v ogrodje NVIDIA RAPIDS (16.-18. 06. 2025)**

- <https://krajsa-izobrazevanja.feri.um.si/UONR>



Hvala za udeležbo!