Analiza podataka u Pandas paketu: pregled mogućnosti i primjeri

Garić, Luka

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Luka Garić

ANALIZA PODATAKA U PANDAS PAKETU: PREGLED MOGUĆNOSTI I PRIMJERI

DIPLOMSKI RAD

Varaždin, 2020.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Luka Garić

Matični broj: 44116/15–R

Studij: Informacijsko i programsko inženjerstvo

ANALIZA PODATAKA U PANDAS PAKETU: PREGLED MOGUĆNOSTI I PRIMJERI

DIPLOMSKI RAD

Mentor:

Doc. dr. sc. Marcel Maretić

Varaždin, rujan 2020.

Luka Garić

Izjava o izvornosti

Izjavljujem da je moj diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Zahvaljujući bibliotekama NumPy i Pandas, Python je jedan od najefikasnijih i najjednostavnijih programskih jezika za dohvaćanje, obradu i analizu podataka. Upravo zbog toga postaje sve popularniji jezik za strojno učenje i umjetnu inteligenciju. Biblioteka NumPy pohranjivanjem podataka statičnog tipa žrtvuje fleksibilnost za drastično povećanje efikasnosti izvršavanja operacije te je brzo postala jedna od najkorištenijih Python biblioteka. Pandas nadograđuje NumPy s objektima i metodama za pohranu i obradu heterogenih tipova podataka te omogućuje jednostavnu i efikasnu obradu višedimenzionalnih podataka. NumPy i Pandas nadopunjuje biblioteka Matplotlib koja omogućuje brzu i jednostavnu vizualizaciju obrađenih podataka. U ovom diplomskom radu je kroz studiju slučaja na podacima vezanim za pandemiju koronavirusa prikazano korištenje ovih biblioteka. Rezultati studije slučaja prikazuju jačanje koronavirusa te zabrinjavajući porast broja zaraženih u Indiji kao jednoj od najmnogoljudnijih država svijeta.

Ključne riječi: Python; R; Jupyter Notebook; Pandas; Numpy; Matplotlib; analiza podataka; COVID-19; koronavirus

Sadržaj

1. Uvod

Programski jezik Python je jedan od najkorištenijih jezika za obradu i analizu podataka te postaje sve popularniji zbog područja u nastajanju poput strojnog učenja i umjetne inteligencije. Najpoznatije Python biblioteke za obradu podataka su NumPy i Pandas koje su postale neizostavne komponente prilikom obrade podataka u Python-u. Biblioteka NumPy spaja jednostavnu Python sintaksu i efikasnost programskog jezika C izvršavajući sve operacije u C-u. S obzirom da je biblioteka NumPy prvenstveno izrađena za pohranu heterogenih podataka i obradu numeričkih podataka, stvorena je biblioteka Pandas koja nadograđuje biblioteku NumPy s metoda i objektima za pohranu i obradu heterogenih podataka. Veliki broj tehnologija u nastajanju se fokusira upravo na pohranjivanje i obradu podataka. Stoga će znanje i iskustvo u osnovnim aspektima navedenih područja biti sve potrebnije kako novije tehnologije napreduju prema strojnom učenju i umjetnoj inteligenciji.

U ovom diplomskom radu ćemo ukratko opisati Python i R, dva najpoznatija programska jezika za obradu podataka te ćemo usporedbom prikazati prednosti i mane oba programska jezika. Nakon toga ćemo opisati što je to Jupyter Notebook, za što se koristi te kako se instalira s obzirom da je programski dio diplomskog rada odrađen koristeći isključivo Jupyter Notebook. Nakon toga slijedi detaljna razrada biblioteke NumPy gdje ćemo objasniti kako NumPy postiže veliku efikasnost, koji tipovi podataka postoje, kako kreirati i manipulirati NumPy objekte te koje su osnovne NumPy metode za obradu podataka. Prilikom objašnjava NumPy metoda ćemo koristiti isječke programskog kôda za lakše razumijevanje. U istom poglavlju ćemo objasniti i biblioteku Matplotlib, jednu od najpoznatijih Python biblioteka za vizualizaciju podataka. U ovom podpoglavlju ćemo objasniti osnovne vrste grafova, kreiranje i korištenje podgrafova, uređivanje naziva grafova, legendi i trake u boji te dodavanje anotacija i uređivanje natpisa na X i Y osi. Glavni fokus ovog diplomskog rada je na razradi i korištenju Pandas biblioteke za obradu i analizu heterogenih podataka. Prvo ćemo detaljno objasniti Pandas objekte, čemu služe te na koji način se mogu kreirati. Nakon toga ćemo iscrpno proći Pandas univerzalne funkcije, različite načine dohvaćanja vrijednosti iz objekata, rukovanje nepostojećim vrijednostima, spajanje Pandas objekata, rukovanje s vremenskim podacima te grupiranje podataka i korištenje naprednih Pandas metoda za efikasniju obradu podataka. Detaljnu razradu Pandas biblioteke ćemo potkrijepiti primjerima programskog kôda kako bi olakšali razumijevanje i ubrzali proces učenja na praktičnim primjerima.

Na kraju ćemo u sklopu studije slučaja izraditi realni primjer obrade, analize i vizualizacije podataka koristeći biblioteke NumPy i Pandas. Prikazat ćemo automatsko dohvaćanje podataka iz javnog *API*-a te ručno preuzimanje i učitavanje podataka. Nakon toga ćemo sve preuzete podatke obraditi i spojiti u jedan `DataFrame` objekt. Dobivene podatke ćemo vizualno prikazati koristeći biblioteku Matplotlib. Na temelju izrađenih grafova ćemo analizirati dobivene rezultate pokušavajući saznati nove informacije na temelju postojećih podataka. Studija slučaja će predstavljati ogledni primjerak preuzimanja, obrade, vizualizacije i analize podataka koji se može primjeniti na bilo koji skup podataka kako bi se iz postojećih podataka saznale nove informacije.

2. Programski jezici Python i R

2.1. Programski jezik Python

Nizozemski programer Guido van Rossum je bio ljubitelj skriptnog jezika ABC zbog njegove jednostavnosti i efikasnosti. No ABC je imao određene nedostatke poput neintuitivne terminologije, obaveznog korištenja velikih slova i ograničene mogućnosti na isključivo čitanje konzole i ispisivanje sadržaja u konzolu. Iako je ABC bio intuitivan, čitljiv i efikasan, upravo zbog navedenih nedostataka nikada nije u potpunosti zaživio. Vidjevši potencijala u takvom pristupu programiranja, Guido van Rossum počinje krajem 1989. godine razvijati programski jezik po uzoru na ABC, kojeg je nazvao Python. Glavni cilj razvijanja Python-a bio je kreiranje čitljivog, jednostavnog i efikasnog programskog jezika razumljivog i ljudima koji nisu prethodno upoznati s osnovama programiranja. [1]

TIOBE indeks mjeri popularnost programskih jezika na temelju broja dobivenih rezultata traženjem "<language> programming" na najpopularnijih 25 tražilica. [2] Prema TIOBE indeksu Python je 2000. godine bio 21. najpopoularniji jezik dok 2005. godine dolazi na visoko 6. mjesto. TIOBE indeks za 6. mjesec 2020. godine dodjeljuje Python-u ocjenu (engl. *rating*) od 9.09%, dakle broj dobivenih rezultata prilikom upisivanja "<language> Python" u najpopularnijih 25 tražilica čini 9.09% ukupnog broja dobivenih rezultata svih programskih jezika što ga stavlja na visoko treće mjesto. Prvo mjesto zauzima programski jezik C koji se primarno koristi za izradu aplikacija s potrebnom visokom razinom sigurnosti i kontrolom nad informacijama (SpaceX koristi programski jezik C za programiranje software-a). Drugo mjesto zauzima programski jezik Java koji ima razne primjene od izrade REST web servisa do izrade mobilnih aplikacija. S druge strane, Python se koristi primarno za obradu i analizu podataka. Dakle prva tri programska jezika spadaju u različite kategorije te si međusobno ne predstavljaju konkurenciju. Jedina konkurencija Python-u je programski jezik R koji se nalazi na 8. mjestu čineći 2.41% ukupnog broja dobivenih rezultata. [3] Također vrijedi napomenuti zanimljivo istraživanje YouTube kanala *Data Is Beautiful* gdje su za analizu korišteni podaci o količini programskog kôda na javnim GitHub repozitorijima u kombinaciji s raznim indeksima i anketama. Prema tom istraživanju Python zauzima prvo mjesto 2019. godine s 24.27% dok se R nalazi na 8. mjestu s 4.36%. [4]

Jul 2020	Jul 2019	Change	Programming Language	Ratings	Change
1	2	↑	C	16.45%	+2.24%
2	1	↓	Java	15.10%	+0.04%
3	3		Python	9.09%	-0.17%
4	4		C++	6.21%	-0.49%
5	5		C#	5.25%	+0.88%
6	6		Visual Basic	5.23%	+1.03%
7	7		JavaScript	2.48%	+0.18%
8	20	⇈	R	2.41%	+1.57%
9	8	↓	PHP	1.90%	-0.27%
10	13	↑	Swift	1.43%	+0.31%

Slika 1: TIOBE indeks za 6. mjesec 2020. godine [3]

Python je napisan korištenjem programskog jezika C koji koristi statičko definiranje varijabli te omogućuje veliku fleksibilnost manipuliranja memorijskim lokacijama kreiranih struktura i varijabli. Kako bi Python mogao podržavati dinamičku alokaciju varijabli svaki Python objekt je zapisan u C programskom jeziku kao struktura koja sadrži minimalno četiri podatka:

```
struct pythonObject{
    long ob_refcnt;
    PyTypeObject *ob_type;
    size_t ob_size;
    long ob_ival;
}
```

Prvi podatak **ob_refcnt** je broj reference koji Python-u olakšava jednostavnije označavanje i uklanjanje zauzeća memorijskih lokacija. Drugi podatak prije naziva varijable sadrži zvjezdicu što u programskom jeziku C znači da je ta varijabla pokazivač, odnosno da sadrži adresu memorijske lokacije na kojoj se nalazi traženi objekt. U ovom slučaju varijabla ***ob_type** je pokazivač koji pokazuje na adresu gdje se nalazi tip podatka koji je pohranjen. Podatak **ob_size** služi za specificiranje veličine pohranjenog podatka, dok se sama vrijednost Python objekta pohranjuje u varijablu **ob_ival**. [5] Korištenjem navedene strukture Python može dinamički promijeniti tip varijable ukoliko promijeni vrijednost pokazivača ***ob_type** da pokazuje na memorijsku lokaciju drugog tipa podatka. Ovakav pristup pohranjivanja varijabli pruža veliku fleksibilnost prilikom dodjeljivanja vrijednosti varijablama, no usporava izvršavanje kod kompleksnijih izračuna zbog obaveznog provjeravanja tipa podatka prije operacije nad varijablom.

Kako bi se poboljšala efikasnost repetitivnih operacija nad istim tipovima podataka kreirane su razne Python biblioteke koje omogućuju spremanje podataka u listu sa statičkim varijablama. Na taj način se smanjuje količina potrebne memorije za svaki podatak te se uvelike poboljšava efikasnost zbog izbjegavanja nepotrebnih provjera tipova podataka prije izvršavanja svake operacije.

2.2. Programski jezik R

Programski jezik R su 1992. godine kreirali profesori Robert Gentleman i Ross Ihaka iz želje za jednostavnim i efikasnim alatom za statističku obradu i prikaz podataka. Tadašnjim programskim jezicima za statističku obradu podataka, poput S i Scheme, su nedostajali jednostavnost i/ili efikasnost. Iz toga razloga su Robert Gentlemen i Ross Ihaka odlučili iz hobija kreirati njihovu verziju programskog jezika S koji bi imao jednostavnost pisanja i efikasnost izvršavanja. Svoju verziju programskog jezika su odlučili nazvati R zbog uske povezanosti s programskim jezikom S te zbog početnih slova u njihovim imenima. Godinu dana kasnije, fakultet na kojem su predavali (*University of Auckland*) odlučuje koristiti novokreirani programski jezik R na statističkim kolegijima preddiplomske razine zbog njegove jednostavne instalacije na tadašnjim Macintosh računalima. Ubrzo nakon toga, časopis S-news javno objavljuje članak o programskom jeziku R nakon čega postaje sve popularniji u statističkoj zajednici. [6]

Zajednica entuzijasta je narednih godina kreirala brojne pakete za R, zbog čega su Hornik i Leisch 1997. godine kreirali *Comprehensive R Archive Network* (kratica: *CRAN*) koja objedinjuje sve kreirane R pakete na jednom centralnom repozitoriju. Prva stabilna verzija programskog jezika R je izašla 2000. godine nakon čega, uz pomoć *CRAN*-a, povećava svoju popularnost te postaje jedan od najkorištenijih jezika za statističku obradu podataka. Upravo zbog toga što je besplatan i javno dostupan, zajednica programera konstantno dodaje nove pakete koji omogućuju statističarima da se fokusiraju na rješavanje statističkih problema umjesto na programiranje rješenja. Skupinu najpoznatijih paketa programskog jezika R pod nazivom *tidyverse* kreirao je programer Hadley Wicham. Neki od paketa *tidyverse*-a su *dplyr* za manipulaciju nad podacima, *ggplot2* za vizualizaciju podataka, *purrr* za dopunu funkcionalnog programiranja te *devtools* za jednostavni razvoj novih R paketa. [6]

Kako bi se programiranje u R-u pojednostavilo, R Team je 2011. godine kreirao *Integrated Development Environment* (kratica: *IDE*) pod nazivom RStudio. RStudio je trenutno najkorišteniji IDE za programski jezik R koji uvelike pojednostavljuje programiranje i vizualizaciju u R-u. [7] Nakon toga je u 2012. godini kreiran i RMarkdown koji omogućuje pokretanje R programskog kôda unutar markdown dokumenata. [8] Kako se zajednica oko programskog jezika R počela drastično povećavati pojavila se potreba za izradom interaktivnih web aplikacija u R-u zbog čega R Team 2012. godine kreira R paket za izradu web aplikacija pod nazivom *Shiny*. [9] Programski jezik R je razvijen od strane statističara te mu je primarna svrha statistička obrada i vizualizacija podataka. Upravo zbog toga je fokus bio na jednostavnosti programskog kôda kako bi statističarima bilo intuitivnije i jednostavnije izvršavati statističke analize. Zbog istih razloga ima veliki broj paketa za razne statističke analize i statističke testove koje izvršava jednostavno i efikasno.

2.3. Usporedba programskih jezika Python i R

Programski jezici Python i R su kreirani 1990-ih godina te su danas dva najkorištenija programska jezika za obradu i analizu podataka. Oba programska jezika su otvorenog kôda (engl. *Open Source*) te su se razvijala s velikim fokusom na čitljivosti programskog kôda. Tablica ispod prikazuje prednosti pojedinog programskog jezika po raznim parametrima gdje znak "+" kod samo jednog programskog jezika označava prednost dok isti znak kod oba programska jezika predstavlja podjednake mogućnosti prema navedenom parametru.

Tablica 1: Usporedba programskih jezika Python i R

Parametar usporedbe	Python	R
Statistička obrada podataka		+
Algoritmi umjetne inteligencije	+	
Kreiranje Markdown dokumenata	+ [Jupyter Notebook]	+ [RMarkdown]
IDE okruženja	+ [PyCharm, Sypder,..]	+ [RStudio]
Brzina izvršavanja	+	+
Web Scraping	+	
Fleksibilnost	+ [Generalna]	+ [Statistička]
Ekosistem	+	+
Vizualizacija podataka		+
Lakoća učenja	+ [Programeri]	+ [Statističari]

Objektivna usporedba Python i R programskih jezika je vrlo zahtjevna s obzirom na manjak objektivnih istraživanja. U ovom poglavlju će se usporediti Python i R na temelju nekoliko izvora prema 10 definiranih parametara zapisanih u tablici iznad. Prvi parametar je statistička obrada podataka u čemu programski jezik R ima veliku prednost s obzirom da je primarno zamišljen za statističku obradu podataka. S druge strane, Python predstavlja puno bolji izbor za izradu algoritama umjetne inteligencije s obzirom na generalnije mogućnost koje pruža kao programski jezik. Također, R paket *Keras* za izradu algoritama za duboko učenje (engl. *deep learning*) u pozadini koristi Python za izvršavanje. [10] Ukoliko se uspoređuju prema mogućnosti kreiranja *Markdown* dokumenata i dostupnih *IDE* alata, oba programska jezika imaju izvrsne opcije. Python sadrži *Jupyter Notebook* za kreiranje i uređivanje markdown dokumenata te razne *IDE* alate poput *PyCharm*, *Spyder* i *PyDev*. S druge strane, R sadrži *RMarkdown* i *RStudio IDE* za izvršavanje istih operacija.

Brzina izvršavanja je parametar koji je najteže procijeniti jer ovisi o vrsti operacije i broja izvršavanja. Prema istraživanju Dmitry-a Kisler-a Python je puno brži ako se uspoređuje vrijeme izvršavanja `for` petlji u oba programska jezika, no ako se umjesto `for` petlje u R-u koristi funkcija `lapply` tada R postaje brži kada je broj iteracija veći od 1000. [11] No vrijedi napomenuti kako se u navedenom istraživanju nije usporedila Numpy biblioteka (koja drastično ubrzava izvršavanje ponavljajuće operacije u Python-u) s `lapply` funkcijom. U slučaju izvršavanja algoritama strojnog učenja (engl. *Machine Learning*) izvršavanje programskog kôda je 5.8 puta brže u odnosu na isti programski kôd napisan u R-u. [12] Ako se usporedi brzina izvršavanja matematičkih operacija, prema članku A. Asaad-a, izvršavanje u Python-u je duplo brže u od-

nosu na R. Iz istog članka je vidljivo da su R biblioteke znatno brže kod izvršavanja određenih matematičkih kalkulacija. No nakon optimizacije programskog kôda Python odnosi pobjedu po pitanju brzine izvršavanja.

Ukoliko se uspoređuje mogućnost preuzimanja podataka s interneta (engl. *Web Scraping*), Python biblioteke poput *BeautifulSoup* i *requests* pružaju izradu bržih i jednostavnijih algoritama za obradu takvih operacija. [13] Sljedeći parametar usporedbe je fleksibilnost. S obzirom da je Python generički programski jezik, u tom području pruža više fleksibilnosti poput jednostavnijeg pokretanja web aplikacija te izrade i izvršavanja raznih skripti. S druge strane, programski jezik R pruža veću fleksibilnost kod statističkih obrada podataka s obzirom da ima veći izbor R paketa za izvršavanje raznih statističkih analiza. Nadalje, oba programska jezika pružaju izvrstan ekosistem s raznim programima i alatima. Sve Python biblioteke i ostali korisni programi se mogu jednostavno koristiti putem *Anaconda* repozitorija, dok se sve R biblioteke nalaze u *CRAN* centralnom repozitoriju. S obzirom na instalaciju biblioteka i paketa, R pruža puno jednostavniji proces instalacije paketa uz manju mogućnost potencijalnih grešaka. [14]

S obzirom da je vizualizacija podataka iznimno bitna prilikom statističkih obrada podataka, programski jezik R sadrži veliki broj paketa za brzu, jednostavnu i efikasnu vizualizaciju podataka poput *ggplot2, htmlwidgets* i *Leaflet*. Python također sadrži moćne biblioteke za vizualizaciju podataka poput Matplotlib i Seaborn, no tek u posljednje vrijeme sustiže R po pitanju jednostavnosti i efikasnosti vizualizacije podataka. [15] Zadnji parametar usporedbe je lakoća učenja programskog jezika. Prednost u ovoj kategoriji ovisi o znanjima i iskustvima osobe koja se odlučuje na učenje jednog od dva navedena programska jezika. Programski jezik R je rađen od strane statističara s ciljem efikasne statističke obrade podataka te je zbog toga programiranje u R-u je intuitivnije statističarima. S druge strane Python je kreiran od strane programera s ciljem izrade jednostavnog i efikasnog programskog jezika za generalno programiranje zbog čega je učenje Python-a intuitivnije programerima. Ukoliko osoba ne dolazi iz programerske ili statističke struke, programski jezik Python bi bio intuitivniji s obzirom na jednostavnost sintakse.

Na navedenim primjerima možemo vidjeti razlike u sintaksi gdje se primjećuje kako je Python više objektno orijentirani jezik (operacije se izvršavaju nad objektima) dok je R više orijentiran izvršavanjem funkcija (kojima se prosljeđuju parametri). Primjer sintakse programskog jezika Python gdje se učitavaju podaci iz *Comma Separated Values* (kratica: *CSV*) datoteke, nakon čega se ispisuje broj redaka i stupaca te prvi red iz učitanih podataka:

```
import pandas as pd
file = pd.read_csv("datoteka.csv")
file.shape
file.head(1)
```

Primjer sintakse programskog jezika R koji izvršava iste operacije:

```
library(readr)
file <- read_csv("datoteka.csv")
dim(file)
head(file, 1)
```

3. Jupyter Notebook

Jupyter Notebook je web aplikacija otvorenog koda (engl. *Open Source*) za kreiranje i uređivanje markdown dokumenata unutar kojih je moguće pisati i pokretati programski kôd. Jupyter Notebook dokument sastoji se od dvije vrste ćelija: *Code* i *Markdown*. *Code* ćelija omogućuje pisanje i pokretanje programskog kôda u nekoliko desetaka programskih jezika (najčešće u Python-u), dok *Markdown* ćelija služi za dodavanje i uređivanje tekstualnog dijela dokumenta. [16] Za formatiranje teksta unutar *Markdown* ćelije koristi se istoimeni Markdown jezik koji pruža puno opcija formatiranja poput dodavanja linkova, matematičkih jednadžbi, slika i uređivanja fonta. [17]

Instalacija Jupyter Notebook-a je moguća kroz Anaconda distribucijsku platformu putem GUI sučelja ili Anaconda konzole (`conda install -c conda-forge notebook`) te putem komandne linije operacijskog sustava korištenjem standardnog instalacijskog Python paketa pip (`pip install notebook`). Nakon što se Jupyter Notebook uspješno instalirao pokreće se putem Anaconda platforme ili komandne linije (`jupyter notebook`) nakon čega se automatski otvara u pregledniku na linku (`http://localhost:8888/`). Ukoliko je Jupyter Notebook pokrenut iz Anaconda Navigatora početni direktorij će za računala s operacijskim sustavom Windows automatski biti `C:/Users/Username` (ukoliko se na C disku nalazi operacijski sustav računala). S druge strane, ako se Jupyter Notebook pokreće iz komandne linije tada će početni direktorij biti direktorij u kojem se komandna linija nalazi u trenutku pozivanja naredbe.

Nakon uspješne instalacije i pokretanja može se kreirati novi Jupyter Notebook dokument putem New - Python gumba nakon čega se kreira datoteka s ekstenzijom *.ipynb* u kojoj je moguće pisati i pokretati Python programski kôd. Jupyter Notebook podržava dva načina rada: *Command* i *Edit*. Korisnik je u svakom trenutku u samo jednom načinu rada te ima aktivnu samo jednu ćeliju nad kojom može izvršavati akcije. Aktivna ćelija je označena obrubom koji s lijeve strane ima zelenu traku (u slučaju *Edit* načina rada) ili plavu traku (u slučaju *Command* načina rada). Većina kratica: na tipkovnici se razlikuje ovisno o trenutnom načinu rada što je vidljivo na popisu svih kratica: koji se prikazuje kombinacijom Ctrl + H u *Command* načinu rada.

Jupyter Notebook pohranjuje dokumente u JSON formatu s ekstenzijom *.ipynb* koji sadrže podatke o sadržaju ćelija, rezultat ispisa za svaku ćeliju te općenite meta podatke cijelog dokumenta poput naziva programskog jezika koji se koristi u dokumentu, verzije korištenog programskog jezika i *Multipurpose Internet Mail Extension* (kratica: *MIME*) tipa ekstenzije. Postoje dodatne Python biblioteke za interpretiranje *.ipynb* dokumenata poput paketa *Voila* koji se može instalirati naredbom *pip install voila*. *Voila* paket izvršava sve ćelije iz *.ipynb* datoteke te ih prikazuje u interaktivnoj HTML web stranici pokretanjem naredbe *voila nazivDatoteke.iypnb*. Istu naredbu je također moguće pokrenuti direktno iz otvorenog *.ipynb* dokumenta unutar Jupyter Notebook-a pritiskom na gumb *Voila*.

4. Python biblioteke

Python biblioteka je skupina metoda koje imaju određenu povezanost poput izvršavanja operacija nad istim objektom ili obavljanja iste zadaće poput vizualizacije podataka ili izvršavanje algoritama strojnog učenja. Veliki broj Python biblioteka omogućuje kreiranje iznimno kompleksnih programa te zahtjevnu obradu i analizu podataka s vrlo malo programskog kôda. Sve Python biblioteke se mogu instalirati putem distribucijske platforme *Anaconda* ili uz pomoć Python instalacijskoj paketa pip koji se može instalirati putem komandne linije s naredbom `python get-pip.py`. Najpoznatije Python biblioteke su NumPy za manipuliranje nad numeričkim podacima, Pandas za obradu tabličnih podataka, Matplotlib za grafički prikaz obrađenih podataka te biblioteka *scikit-learn* s raznim algoritmima za strojno učenje.

4.1. Numpy

Sve vrste podataka se mogu zapisati u obliku brojeva, odnosno imaju svoju numeričku reprezentaciju. Crno-bijela slika se može smatrati kao jednodimenzionalno polje brojeva koji predstavljaju svjetlinu pojedinog piksela, dok se slika u boji može zapisati u obliku dvodimenzionalnog polja brojeva gdje se za svaki piksel zapisuju svjetline *Red-Green-Blue* (kratica. RGB) spektra čime se dobiva numerička reprezentacija boje svakog piksela. Zvuk se također može reprezentirati u obliku jednodimenzionalnog polja brojeva koji označuju intenzivnost zvuka po vremenu, dok tekst ima različite vrste numeričkih reprezentacija poput frekvencije broja pojavljivanja riječi ili grupe riječi u određenom tekstu. S obzirom da računala na najnižoj razini izvršavaju operacije na bitovima, numerička obrada podataka omogućuje brže i efikasnije izvršavanje raznih operacija nad podacima. Iz tog razloga je pretvaranje podataka u njihovu numeričku reprezentaciju te izvršavanje operacija nad numeričkim podacima puno efikasnije rješenje. No s obzirom da je Python dinamički programski jezik, konstantna provjera tipa podatka prije izvršavanja svake operacije uvelike usporava izvršavanje programa čak i prilikom izvršavanja isključivo numeričkih izračuna.

Kako bi se riješio problem dinamičke alokacije varijabli, zajednica programera kreirala je Python biblioteku koja omogućuje kreiranje statičke liste numeričkih elemenata drastično povećavajući efikasnost izvršavanja operacija. Navedena Python biblioteka je prvi puta kreirana 1995. godine pod nazivom *Numeric* dok 2005. godine dobiva ime *Numeric Python* te postaje popularnija po skraćenici NumPy. [18] NumPy je trenutno najkorištenija Python biblioteka koja omogućuje kreiranje NumPy liste numeričkih elemenata. Kreirana lista omogućuje efikasno izvršavanje velikog broj operacija bez potrebe za provjerom tipa podatka prije svake operacije pri čemu se drastično smanjuje vrijeme potrebno za izvršavanje operacija nad velikim skupom numeričkih podataka. S obzirom da računala na najnižim razinama obavljaju matematičke operacije te da svi podaci imaju svoju numeričku reprezentaciju, prilikom izrade novih Python biblioteka postalo je neizbježno koristiti biblioteku NumPy. Konvertiranjem podataka u njihovu numeričku reprezentaciju te izvršavanjem operacija na numeričkoj razini s bibliotekom NumPy se drastično povećava efikasnost. Zbog toga se NumPy biblioteka koristi prilikom izrade brojnih drugih biblioteka te postaje najosnovnija Python biblioteka za obradu i analizu podataka. [19]

S obzirom na dinamičku alokaciju varijabli u Python-u, svaka varijabla sadrži prethodno navedeni ***ob_type** koji određuje tip pohranjenog podatka. Zbog toga Python liste mogu sadržavati varijable različitih tipova gdje svaka varijabla sadrži svoju referencu na tip podatka koji je pohranjen. NumPy lista se odriče heterogenosti kako bi se povećala efikasnost na način da sve informacije o podacima u listi spreme na centraliziranu lokaciju. Također, svi podaci u NumPy listi su pohranjeni na slijednim memorijskim lokacijama kako bi se dodatno povećala efikasnost. Vrijedi napomenuti da Python od verzije 3.3. sadrži ugrađeni *array* modul koji omogućuje kreiranje statičkih lista (npr. `staticka_lista = array.array('i', dynamic_array)`, gdje 'i' predstavlja `integer` tip podatka). Iako navedeni modul povećava efikasnost u odnosu na standardne Python liste, NumPy liste imaju puno veći broj operacija koje se mogu izvršavati nad kreiranom listom.

NumPy pruža 5 osnovnih tipova varijabli koji se mogu definirati atributom `dtype` prilikom kreiranja NumPy liste. Osnovni tipovi se definiraju uz obavezne sufikse koji određuju količinu memorijske lokacije u bajtovima (npr. `int8, int16, int32` predstavljajući `integer` s 8, 16 i 32 bajta memorijske lokacije respektivno). Također postoji dodatni sufiks "_" koji predstavlja osnovnu veličinu podatka. Standardni numerički tip podataka `int` sadrži dodatne sufikse `c` koji označava standardnu veličinu podatka u programskom jeziku C i `p` koji predstavlja `integer` tip podatka korišten za indeksiranje.

Tablica 2: NumPy tipovi podataka

Tip	Dozvoljeni sufiksi
`bool`	_
`int`	_, c, p, 8, 16, 32, 64
`uint`	8, 16, 32, 64
`float`	_, 16, 32, 64
`complex`	_, 32, 64

NumPy sadrži veliki broj funkcija za manipulaciju podataka NumPy liste. Osnovne NumPy funkcije se mogu podijeliti u sljedeće kategorije:

- **Atributi** - Dohvaćanje atributa NumPy liste poput dimenzije, veličine i vrste podataka
- **Indeksiranje** - Dohvaćanje i postavljanje vrijednosti pojedinih elemenata NumPy liste. Vrijedi napomenuti kako će NumPy ukloniti decimalne vrijednosti ukoliko ih definirani tip podataka liste ne podržava (poput `int`).
- **Dohvaćanje podniza** (engl. *slicing*) - Dohvaćanje i postavljanje vrijednosti podliste sintaksom *nazivListe[start:stop:povećanje]* (npr. `lista[::2]` dohvaća svaki drugi element unutar NumPy liste). Ukoliko je korak povećanja negativan tada se kreće od kraja liste prema početku te se vrijednosti start i stop zamjenjuju. Na isti način se mogu dohvatiti podaci u dvodimenzionalnoj listi odvajanjem vrijednosti sa zarezom (npr. `lista[:, ::2]` gdje prvi parametar označava da se dohvaćaju svi reci dok drugi parametar dohvaća svaki drugi stupac). Kombinacijom indeksiranja i dohvaćanja podniza je moguće dohvatiti pojedini redak ili stupac (npr. `lista[:, 0]` dohvaća prvi stupac).

- **Oblikovanje** (engl. *reshaping*) - Manipulacija dimenzije NumPy liste. Najfleksibilniji način oblikovanja NumPy liste je korištenje funkcije `.reshape()`. No moguće je koristiti funkciju samo ako odgovaraju veličine prije i nakon oblikovanja liste (npr. `lista.reshape(3, 3)` pretvara listu u dvodimenzionalno polje s 3 retka i 3 stupca samo ako lista prije pretvaranja sadrži 9 brojeva).
- **Spajanje** - Za spajanje nekoliko NumPy lista se koristi funkcija `np.concatenate()` koja prima dvije liste za spajanje te opcionalni parametar *axis* za određivanje načina spajanja (`axis=0` spaja liste vertikalno dok `axis=1` spaja liste horizontalno). Također je moguće koristiti funkcije `np.vstack()` i `np.hstack()` za vertikalno i horizontalno spajanje NumPy lista respektivno.
- **Razdvajanje** - Razdvajanje NumPy liste je moguće s funkcijom `np.split()` koja prima listu za razdvajanje te listu pozicija liste na kojima će se razdvojiti (npr. `x1, x2, x3 = np.split(x, [3, 5])` razdvaja listu *x* na tri dijela kod pozicija s indeksom 3 i 5). Za razdvajanje dvodimenzionalnih lista se koriste funkcije `np.vsplit()` i `np.hsplit()` za vertikalno i horizontalno razdvajanje respektivno (npr. `gornja, donja = np.vsplit(array, [2])` dijeli dvodimenzionalnu listu vertikalno po trećem retku).

Najveća prednost NumPy biblioteke je jednostavno i fleksibilno sučelje za izvršavanje optimiziranih operacija nad NumPy listama. Razlog velike efikasnosti NumPy operacija su tzv. *vektorizirane operacije*. Vektorizacija je proces delegiranja izvršavanja matematičkih operacija u optimiziraniji i efikasniji C programski kôd. Rezultat izvršavanja operacija u programskom jeziku C je iznimno efikasnije i brže izvršavanje operacija izbjegavajući konstantne provjere tipova podataka. Vektorizirane operacije su u Python-u implementirane uz pomoć univerzalnih funkcija (engl. *universal functions*, kratica: *ufuncs*). Vektorizirane funkcije drastično ubrzavaju izvršavanje repetitivnih operacija nad NumPy listama, iznimno su fleksibilne te se mogu koristiti i nad višedimenzionalnim listama. Mogu se podijeliti na dvije vrste: [5]

- **Unarne vektorizirane funkcije** - izvršavaju operacije nad samo jednim inputom
- **Binarne vektorizirane funkcije** - izvršavaju operacije nad dva inputa

Kod binarnih vektoriziranih funkcije je važno da oba inputa, odnosno obje NumPy liste budu istog oblika (npr. u slučaju 2D liste broj redaka i stupaca moraju odgovarati). Ukoliko oblici proslijeđene dvije liste ne odgovaraju prilikom pozivanja binarnih vektoriziranih funkcija slijede se unaprijed definirana pravila poznata pod nazivom *broadcasting*. *Broadcasting* predstavlja skup pravila za izjednačavanje oblika lista kako bi se nad njima mogle izvršavati matematičke operacije. Dvije osnovne vrste *broadcasting*-a su pretvaranje broja u 1D listu kopiranjem istog broja n puta (gdje n predstavlja broj elemenata druge liste) i pretvaranje 1D u 2D listu kopiranjem redaka n puta (gdje n predstavlja broj redaka druge 2D liste). Vrijedi napomenuti da se riječ "kopiranje" broja i redaka ne odnosi na fizičku alokaciju varijabli u memoriji nego se operacije izvršavaju nad istim skupom podataka (odnosno nad istim brojem ili retkom). Riječ "kopiranje" se koristi isključivo za jednostavnije razumijevanje načina rada *broadcasting*-a.

Prilikom izjednačavanja oblika listi *broadcasting* slijedi unaprijed definirana tri koraka:

1. **Izjednači dimenzije** ukoliko su liste različitih dimenzija - dodaju se jedinice na vodećoj (lijevoj) strani (npr. ako su inputi 1D i 2D liste tada se 1D lista s n elemenata koja ima oblik "n" pretvara u 2D listu nakon čega dobiva oblik "1, n" - odnosno ima jedan redak i n stupaca)
2. **Izjednači oblik** ukoliko su liste različitih oblika nakon izjednačavanja dimenzija - dupliciraju se vrijednosti na poziciji gdje je vrijednost oblika jednaka 1 (npr. ako su inputi 1D i 2D liste s oblicima "5" i "3, 5": nakon pretvaranja 1D u 2D listu vrijednosti dimenzija iznose "1, 5" i "3, 5". U tom slučaju se duplicira redak prve liste 3 puta tako da se izjednači oblik obje liste.
3. Ako oblici obje liste ne odgovaraju nakon prethodna dva koraka vrati grešku.

IPython sadrži tzv. čarobne naredbe (engl. *magic commands*) koje počinju sa znakom "%" ili "%%" označavajući da izvršavaju operaciju nad linijom programskog kôda ili nad cijelom ćelijom (odnosno na više programskih linija kôda). Čarobne naredbe su namijenjene za rješavanja svakodnevnih problema prilikom obrade podataka. Jedna od najkorištenijih čarobnih naredbi je naredba `%time` koja automatski mjeri vrijeme potrebno za izvršavanje linije u kojoj se naredba nalazi (ili `%%time` za mjerenje izvršavanja cijele ćelije). [5]

Korištenjem čarobne funkcije `%time` se vrlo lako mogu usporediti brzine izvršavanja Python petlji i vektoriziranih funkcija. Sljedeći primjer mjeri vrijeme potrebno za izračun recipročnih vrijednosti na skupu od 5 000 000 brojeva:

```
import numpy as np
np.random.seed(0)

def compute_reciprocals(values): # Python petlja
    output = np.empty(len(values))
    for i in range(len(values)):
        output[i] = 1.0 / values[i]
    return output

# NumPy lista s 5000000 nasumicnih brojeva
values = np.random.randint(1, 10, size=5000000)
# Racunanje reciprocnih brojeva s Python petljom
%time x = compute_reciprocals(values)
# Racunanje reciprocnih brojeva vektoriziranom funkcijom (ufuncs)
%time y = 1.0 / values
```

Prije korištenje NumPy funkcija potrebno je uvesti (engl. *import*) biblioteku pri čemu je općeprihvaćena kratica: za NumPy **np**. S obzirom da je biblioteku potrebno uvesti samo jednom, u daljnjim primjerima će se zbog jednostavnosti izostaviti navedena linija.

Navedeni primjer daje rezultat od 15.7 sekundi ukoliko se koristi Python petlja dok se za izvršavanje iste akcije korištenjem vektorizirane operacije vrijeme izvršavanja smanjuje na samo 16 milisekundi. U ovom primjeru su vektorizirane funkcije izvršile isti zadatak 980 puta brže od standardne Python petlje što predstavlja drastično poboljšanje efikasnosti. Upravo zbog velike efikasnosti vektoriziranih funkcija preporuča ih se koristiti umjesto Python petlji kad god je to moguće.

Tablica 3: Aritmetičke NumPy univerzalne funkcije (engl. *ufuncs*) [5]

Operator	Univerzalna funkcija	Opis
+	.add	Zbrajanje
-	.subtract	Oduzimanje
-	.negate	Negacija
*	.multiply	Množenje
/	.divide	Dijeljenje
//	.floor_divide	Dijeljenje bez ostatka
**	.power	Potenciranje
%	.mod	Ostatak pri dijeljenju

Tablica 3. prikazuje osnovne aritmetičke funkcije implementirane u NumPy paketu u obliku vektoriziranih funkcija. Navedene vektorizirane funkcije se mogu koristiti navođenjem samo operatora nakon čega NumPy mapira operator na odgovarajuću vektoriziranu funkciju (npr. `array + 2` postaje `np.add(array, 2)`). Također vrijedi napomenuti kako NumPy, uz navedene aritmetičke funkcije, podržava veliki broj drugih matematičkih funkcija poput logaritmiranja (npr. `np.ln(), np.log10()`), trigonometrijskih funkcija (npr. `np.sin(), np.cos() i np.tan()` te specijalnih funkcija implementiranih u `scipy.special` modulu (npr. `special .gamma() i special.beta()`). Sve navedene vektorizirane operacije sadrže opcionalni atribut *out* koji definira varijablu za spremanje rezultata operacije.

NumPy podržava dodatne dvije operacije za agregaciju podataka koje se mogu pozvati nad ostalim operacijama: `.reduce()` i `.accumulate()`. Funkcija `.reduce()` izvršava operaciju nad svim elementima liste i vraća agregirani rezultat (npr. `np.subtract.reduce(numpyLista)` oduzima sve elemente u listi). Funkcija `.accumulate()` također izvršava operaciju nad svim elementima liste vraćajući listu koja se sastoji od rezultata svakog koraka. Vrijedi napomenuti da najkorištenije agregatne funkcije imaju posebnu funkciju bez potrebe za pozivanje `.reduce()` ili `.accumulate()` funkcija. Neki od primjera su `np.add.reduce() - np.sum(), np.add.accumulate() - np.cumsum(), np.multiply.reduce() - np.prod()` te funkcija `np.multiply.accumulate()` koju je moguće zamijeniti sa `np.cumprod()`.

Uz navedene `np.sum()` i `np.prod()` funkcije za agregaciju podataka, NumPy sadrži i ostale agregirajuće funkcije prikazane u tablici 4. Sve agregirajuće funkcije osim `np.all()` i `np.any()` imaju verziju koja ignorira sve vrijednosti koje nisu numeričke (engl. *Not a Number*, kratica: *NaN*). Većina navedenih NumPy funkcija je također implementirana u sklopu Python-a (npr. *sum()* i `np.sum()`). Razlika između navedenih funkcija je u implementaciji. Python funkcije se prevode i izvršavaju u Python-u, dok se NumPy verzije funkcija prevode i izvršavaju u programskom jeziku C drastično poboljšavajući performanse.

Tablica 4: Agregirajuće NumPy funkcije [5]

Univerzalna funkcija	NaN sigurna verzija	Opis
.sum	.nansum	Izračunava sumu svih elemenata
.prod	.nanprod	Izračunava umnožak svih elemenata
.mean	.nanmean	Izračunava prosječnu vrijednost elemenata
.std	.nanstd	Izračunava standardnu devijaciju
.var	.nanvar	Izračunava varijancu
.min	.nanmin	Nalazi i vraća najmanju vrijednost
.max	.nanmax	Nalazi i vraća najveću vrijednost
.argmin	.nanargmin	Nalazi i vraća indeks minimalne vrijednosti
.argmax	.nanargmax	Nalazi i vraća indeks maksimalne vrijednosti
.median	.nanmedian	Izračunava medijan
.percentile	.nanpercentile	Izračunava statistički postotak
.any	-	Vrijedi li izraz na barem jednom elementu
.all	-	Vrijedi li izraz na svim elementima

NumPy također omogućuje filtriranje i indeksiranje podataka u NumPy listama koristeći vektorizirane funkcije usporedbe. Takav način filtriranja i indeksiranja podataka omogućuje drastično efikasniju obradu podataka. Kao i aritmetičke funkcije, funkcije usporedbe se također mogu koristiti navođenjem samo operatora usporedbe pri čemu se izraz mapira na odgovarajuću NumPy vektoriziranu funkciju (npr. `array < 2` postaje `np.greater(array, 2)`). Sve NumPy vektorizirane funkcije izvršavaju operacije nad svakim elementom NumPy liste vraćajući listu rezultata. Jedina iznimka su agregirajuće funkcije koje izvršavaju operacije nad svakim elementom, ali mogu vraćati skalar umjesto liste (npr. `np.sum()`). Tablica 5. prikazuje vektorizirane funkcije usporedbe i bitovne logičke operatore.

Tablica 5: Vektorizirane NumPy funkcije za usporedbu podataka

Operator	Univerzalna funkcija	Opis
==	.equal	Ispituje jednakost elemenata
!=	.not_equal	Ispituje nejednakost elemenata
<	.less	Ispituje je li element prije znaka < manji
<=	.less_equal	Ispituje je li element prije znaka < manji ili jednak
>	.greater	Ispituje je li element prije znaka < veći
>=	.greater_equal	Ispituje je li element prije znaka < veći ili jednak
&	.bitwise_and	Ispituje AND operaciju nad bitovima
|	.bitwise_or	Ispituje OR operaciju nad bitovima
^	.bitwise_xor	Ispituje XOR (isključivo ILI) operaciju nad bitovima
~	.bitwise_not	Ispituje NOT operaciju nad bitovima

Razlika između Python operatora **and**, **or** i NumPy operatora |, & je u interpretaciji objekata koji se uspoređuju. Python operatori **and** i **or** uspoređuju cijeli objekt, odnosno ispituju istinitost izraza na temelju cijelog objekta dok NumPy operatori | i & ispituju istinitost izraza na temelju bitova objekata. Zbog toga nije moguće usporediti dvije `bool` liste s Python operatorima dok je ista stvar moguća s NumPy operatorima.

S obzirom da je biblioteka NumPy fokusirana na pohranjivanje i obradu numeričkih podataka u sklopu biblioteke je implementirano nekoliko metoda za sortiranje podataka koristeći algoritme *quicksort, mergesort* i *heapsort*. Iako je moguće kreirati vlastiti algoritam sortiranja ili iskoristiti postojeće Python algoritme, standardni NumPy *quicksort* algoritam za sortiranje `np.sort()` je većinom puno efikasniji. Koristeći O notaciju složenosti algoritama, `np.sort()` ima rezultat od $O[NlogN]$ predstavljajući izvrstan odabir za sortiranje lista s velikom količinom podataka. Metoda `np.sort()` prima opcionalni parametar *axis* koji definira os po kojemu se vrijednosti sortiraju (`axis=0` sortira po stupcima, a `axis=1` sortira po recima). Također je moguće vratiti listu indeksa sortiranih vrijednosti s funkcijom `np.argsort()` koja se tada može proslijediti kao parametar prilikom indeksiranja kako bi se dobila sortirana lista.

```
x = np.array([2, 1, 4, 3, 5])
print(np.sort(x)) # Ispisuje sortiranu listu bez spremanja
x.sort() # Sortira listu i sprema ju u varijablu x
print("Indeksi uzlazno sortiranih elemenata: ", np.argsort(x))
# Indeksira elemente po indeksima sortiranih elemenata
print(x[np.argsort(x)]) # rezultat je uzlazno sortirana lista!
```

Iako su NumPy liste osmišljene za pohranjivanje homogenih podataka, NumPy biblioteka omogućuje pohranu heterogenih podataka u NumPy strukturirane liste. Strukturirane NumPy liste dozvoljavaju dodjeljivanje naziva pojedinoj skupini podataka kojima se tada može pristupiti putem naziva (npr. `lista['naziv']`). Ukoliko se strukturirane NumPy liste pretvore u tzv. *record arrays* tada je moguće pristupiti podacima putem atributa (npr. `lista.naziv`) no takav pristup zahtjeva spremanje dodatnih podataka te utječe na performanse. Na temelju navedenih NumPy strukturiranih lista je napravljena Python biblioteka Pandas koja sadrži puno više funkcionalnosti za manipuliranje heterogenih podataka. Prilikom kreiranja NumPy strukturiranih lista mogu se koristiti skraćeni kodovi tipova podataka koji su navedeni u tablici 6. Nakon određenih kodova se dodaje broj koji predstavlja količinu memorijske lokacije u byte-ovima koju će pojedini podatak zauzeti (npr. '`f8`' predstavlja `float` tip podatka s 8 byte-ova, odnosno 64 bitova - `np.float64`). Također postoji opcionalni prefiks kod pojedinih tipova podataka koji definira način zapisivanja numeričkih podataka gdje < predstavlja *little edian* dok > predstavlja *big edian* (npr. '`<i1`' predstavlja podatak tipa `np.int8` zapisan *little edian* konvencijom).

Tablica 6: NumPy tipovi podataka

Kod	Opis	Primjer
'b'	Bitovni podatak [bit]	.dtype('b')
'i'	Numeričke vrijednosti [integer]	.dtype('i4')
'u'	Pozitivne numeričke vrijednosti [unsigned int]	.dtype('u1')
'f'	Decimalne vrijednosti [float]	.dtype('f8')
'c'	Kompleksne vrijednosti [complex]	.dtype('c16')
'S', 'a'	Tekstualne vrijednosti [string]	.dtype('S5')
'U'	Unicode tekstualne vrijednosti	.dtype('U') == .str_
'V'	Nedefinirane vrijednosti [raw data]	.dtype('V') == .void

4.2. Matplotlib

Kako bi omogućio moćnu vizualizaciju u Python-u John Hunter je planirao, u suradnji s tvorcem *Interactive Python-a* (kratica. *IPython*) Fernandom Perez, omogućiti kreiranje interaktivnih grafova spajanjem programa *gnuplot* s komandnom linijom IPython-a. No Fernando Perez je u to vrijeme završavao doktorski studij te nije bio u mogućnosti sudjelovati u realizaciji ideje. Zbog toga je John Hunter 2003. godine počeo razvijati vlastitu Python biblioteku za vizualizaciju podataka koju je nazvao Matplotlib. U to vrijeme, velika prednost Matplotlib biblioteke bila je raznovrsnost pozadinskog dijela (engl. *backend*) koja omogućuje vizualizaciju podataka na svim operacijskim sustavima. Iz tog razloga postaje odabrana biblioteka za vizualizaciju podataka od strane *Space Telescope Science* instituta koji su financijski podržali biblioteku što je rezultiralo drastičnom proširivanju funkcionalnosti. [20]

Matplotlib je jedna od najpopularnijih Python biblioteka za vizualizaciju podataka s velikim brojem mogućnosti uređivanja izgleda vizualizacije. Najčešće korištene skraćenice prilikom uvođenja su ***plt*** za modul `matplotlib.pyplot` te ***mpl*** za cijelu Matplotlib biblioteku. Izgrađena je korištenjem NumPy lista te je dizajnirana da funkcionira sa širim SciPy ekosistemom. SciPy ekosistem je naziv za skupinu osnovnih Python biblioteka korištenih u znanstvene svrhe poput NumPy, Pandas i scikit-learn. [21] S obzirom na mogućnost korištenja Matplotlib-a na raznim platformama proces vizualizacije je drugačiji ovisno o aplikacijskom kontekstu. Postoje tri glavna aplikacijska konteksta iz kojih se pozivaju metode za vizualizaciju podataka:

1. **Python skripta** - za prikaz vizualizacije u Python skripti koristi se metoda `matplotlib .pyplot.show()` koja pretražuje sve trenutno aktivne figure te svaku figuru ispisuje u zasebni interaktivni prozor.

2. **IPython shell** - preporuča se postaviti način rada Matplotlib-a s čarobnom naredbom `%matplotlib` nakon čega se pozivom funkcije `.plot()` kreira interaktivni prozor s vizualizacijom. Daljnje izmjene u programskom kôdu se ažuriraju automatski no moguće je prisilno osvježiti sve vizualizacije pozivom funkcije `matplotlib.pyplot.draw()`.

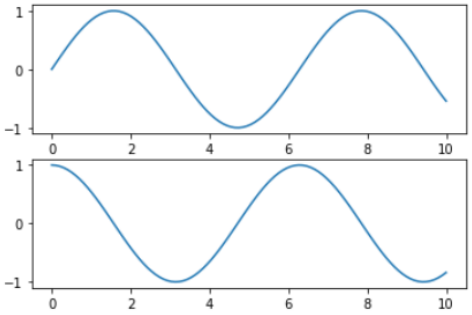
3. **IPython notebook** - S obzirom da IPython notebook koristi IPython shell također se preporuča postaviti način rada s naredbom `%matplotlib`. No kako IPython notebook kombinira HTML elemente, grafičke elemente i programski kôd unutar jednog dokumenta, standardni način rada postavljen s `%matplotlib` ispisuje interaktivne grafove unutar samog dokumenta umjesto kreiranja interaktivnog prozora. Uz standardni način rada postoji i čarobna naredba `%matplotlib inline` koja ispisuje statičke slike unutar dokumenta umjesto interaktivnih vizualizacija. Navedene čarobne naredbe načina rada Matplotlib-a je potrebno pozvati samo jednom po sesiji.

Vrijedi napomenuti kako je Matplotlib biblioteka bila iznimno napredna 2003. godine no u posljednje vrijeme dolaze naprednije biblioteke koje omogućuju brže i jednostavnije prikazivanje vizualno privlačnijih grafova. Unatoč tomu, zbog velikog broja mogućnosti Matplotlib-a te zbog neovisnosti o operacijskom sustavu se i dalje često koristi za vizualizaciju podataka i izgradnju naprednijih biblioteka za vizualizaciju (poput biblioteke Seaborn). [5]

S obzirom da je Matplotlib izrađen kao MATLAB alternativa za Python veliki dio sintakse je sličan MATLAB-u. Zbog toga je kreirano sučelje `matplotlib.pyplot` koje, po uzoru na MATLAB, sadrži sve alate potrebne za vizualizaciju podataka. Navedeno sučelje je vezano za sesiju te sadrži instance trenutne figure i osi figure. Sve pozvane metode unutar sučelja izvršavaju akcije nad navedenim instancama unutar sučelja. Također postoje metode `matplotlib.pyplot.gcf` i `matplotlib.pyplot.gca` koje dohvaćaju trenutnu figuru (engl. *get current figure*) i trenutne osi figure (engl. *get current axes*) respektivno. MATLAB stil sučelja je brz i praktičan za izradu jednostavnih vizualizacija, ali predstavlja probleme kod izrade kompleksnijih grafova. Iz tog razloga Matplotlib sadrži i objektno orijentirano sučelje koje pruža veću kontrolu nad figurama. Prilikom korištenja objektno orijentiranog sučelja metode za vizualizaciju se pozivaju direktno nad objektima `Figure` i `Axes`. `Figure` objekt je sadrži sve ostale objekte potrebne (uključujući i `Axes` objekt) za prikazivanje jedne ili više vizualizacija dok `Axes` objekt sadrži elemente potrebne za prikaz jedne vizualizacije.

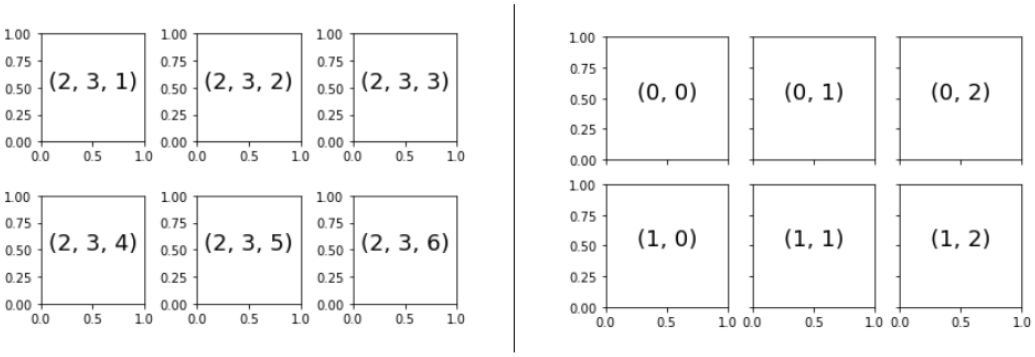
```
import matplotlib.pyplot as plt

# MATLAB sucelje
plt.figure() # Kreiranje figure
# Dohvacanje prvog podgrafa mreze grafova s 2 retka i jednim stupcem
plt.subplot(2, 1, 1)
plt.plot(x, np.sin(x)) # Crtanje sinus funkcije na dohvacenom grafu
plt.subplot(2, 1, 2) # Dohvacanje drugog grafa
plt.plot(x, np.cos(x)) # Crtanje kosinus funkcije na dohvacenom grafu

# Objektno orijentirano sucelje
# ax ce biti lista od dva objekta tipa Axes
fig, ax = plt.subplots(2) # Kreiranje figure i osi figure
ax[0].plot(x, np.sin(x)) # Crtanje sinusa na prvom grafu
ax[1].plot(x, np.cos(x)) # Crtanje sinusa na prvom grafu
```

U praksi se za `matplotlib.pyplot` koristi kratica: **plt** te se u daljnjim primjerima pretpostavlja da je biblioteka učitana s navedenom kraticom. Oba sučelja prikazuju isti graf sa sinus i kosinus funkcijama prikazan na slici 2.



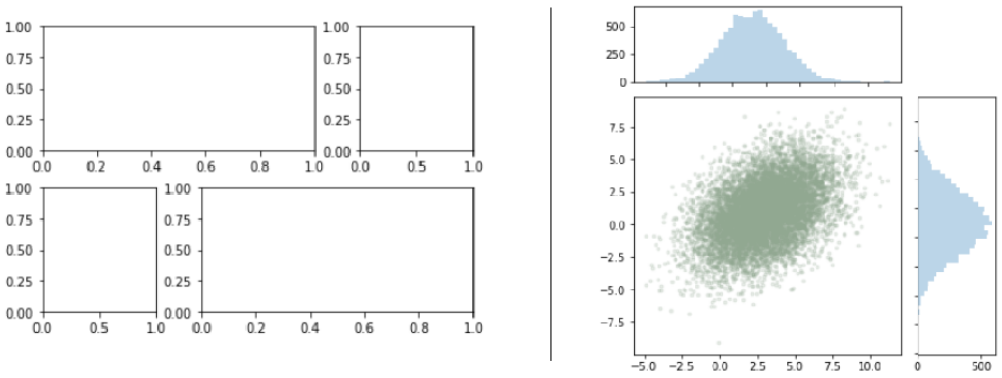
Slika 2: Rezultat vizualizacije sinus i kosinus funkcija

Jedna od bitnih značajki Matplotlib-a je kreiranje podgrafova unutar jedne figure kako bi se omogućila izrada bogatijih vizualizacija. Jedan od načina kreiranja podgrafova je metoda `plt.subplot()` koja koristi MATLAB stil sučelja te prima tri numerička argumenta: broj redaka, broj stupaca i indeks odabranog grafa (npr. `plt.subplot(2, 3, 5)`). Ovakav pristup kreiranja podgrafova automatski dodaje oznake na X i Y osi svakoga podgrafa što može rezultirati nepotrebnim ponavljanjem podataka na osima (ukoliko grafovi dijele osi). U tom slučaju se može koristiti metoda `plt.subplots()` koja se koristi u objektno orijentiranom sučelju te sadrži opcionalne argumente *sharex* i *sharey* (npr. `fig, ax = plt.subplots(2, 3, sharex='col'` kreira "mrežu" grafova kojima svi grafovi u istom stupcu dijele X os). S obzirom da se koristi u objektno orjentiranom sučelju, metoda `plt.subplots()` vraća objekt `Figure` i listu `Axes` objekata koji se tada koriste za vizualizaciju. No kako je dohvaćeni parametar lista `Axes` objekata podgrafovi se dohvaćaju standardnom notacijom dvodimenzionalne liste (npr. `ax[0, 1]` dohvaća podgraf na poziciji [0, 1]). Na slici 3. se mogu vidjeti primjeri dohvaćanja podgrafova funkcijom `plt.subplot()` na lijevoj strani te funkcijom `plt.subplots()` na desnoj strani.



Slika 3: Različiti načini indeksiranja podgrafova [5]

Za kreiranje kompliciranijih grafova postoji metoda `plt.GridSpec()` koja kreira praznu "mrežu" na kojoj se tada mogu kreirati podgrafovi. Metodi se prosljeđuje broj redaka i stupaca uz opcionalne parametre *wspace* i *hspace* za definiranje vertikalnog i horizontalnog razmaka između grafova. Nakon toga se kreirana mreža grafova može proslijediti funkciji `plt.subplot()` (npr. `plt.subplot(grid[0, :])` kreira graf u cijelom prvom retku). Primjer korištenja funkcije `plt.GridSpec()` je prikazan na slici 4. dok se programski kôd nalazi u prilogu.

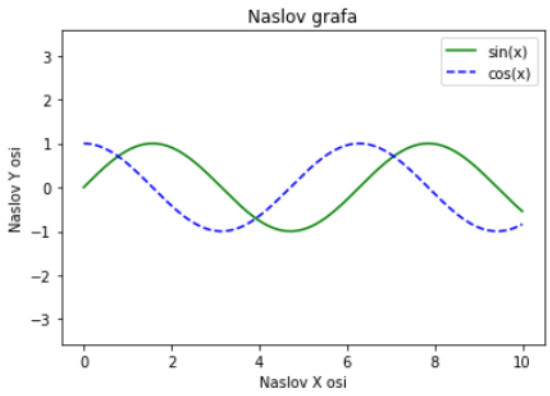


Slika 4: Primjer korištenja kompleksnijih podgrafova

Matplotlib omogućuje brzu i jednostavnu manipulaciju boje i stila prilikom crtanja linija na grafu korištenjem atributa *color* i *linestyle*. Atribut *color* prihvaća definiranje boje na razne načine poput navođenja boje imenom (npr. `'black', 'blue'`), korištenjem kodova definiranih boja [rgbcmyk], prosljeđivanjem vrijednosti skale sive boje (raspon od 0 do 1), prosljeđivanjem heksadecimalne vrijednosti boje (npr. '`#FFDD44`') ili numeričkih vrijednosti RGB skale (npr. `(1, 1, 0)` za žutu boju). Atribut *linestyle* također podržava četiri osnovna stila linija te koji se mogu definirati na dva načina: imenom ili kodovima. Imena podržanih stilova su *solid, dashed, dashdot* i *dotted* dok se isti stilovi mogu definirati s kodovima `'-'`, `'--'`, `'-.'` i '*:*' respektivno. Također vrijedi napomenuti kako se mogu kombinirati kodovi oba atributa unutar jednog argumenta (npr. '*–k*' predstavlja iscrtkanu plavu liniju dok '*-r*' predstavlja ravnu crvenu liniju).

Prilikom kreiranja grafa Matplotlib automatski definira limit x i y koordinata ovisno o liniji koja se prikazuje. Ove vrijednosti se mogu ručno izmijeniti s metodama *xlim* i *ylim* prosljeđivanjem numeričkih vrijednosti od/do koje predstavljaju limit određene osi. Navedim metodama se također može proslijediti tekstualna vrijednost koja predstavlja tip definiranja limita (npr. '*tight*' za smanjivanje duljine osi koliko je to moguće ili '*equal*' za održavanje omjera slike). Matplotlib također omogućuje postavljanje naslova grafa i naziva X i Y osi s metodama `.title()`, `.xlabel()` i *ylabel()* respektivno. Primjer programskog kôda za kreiranja grafa s osnovnim parametrima u Matplotlib-u je prikazan ispod te je rezultat izvršavanja kôda vidljiv na slici 5.

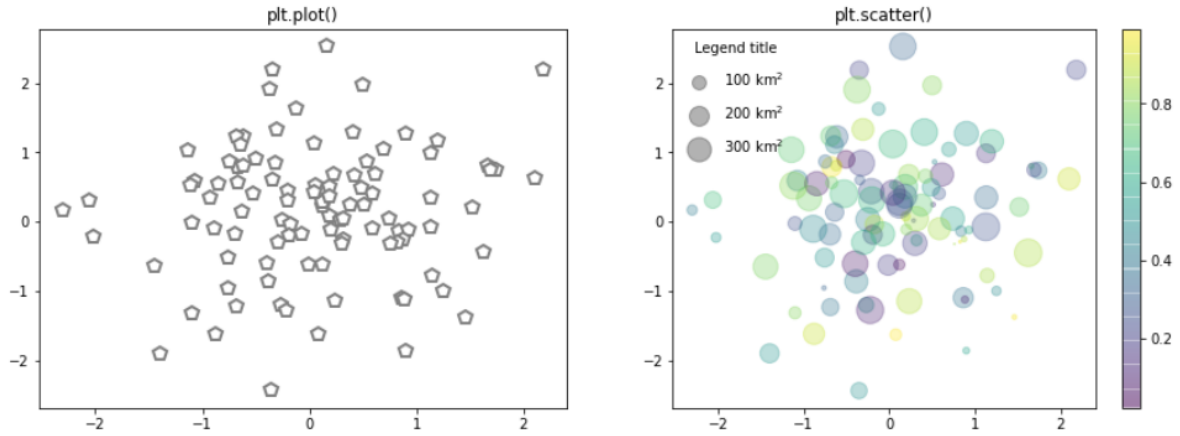
```
# Lista od 1000 jednako odvojenih brojeva između 0 i 10
x = np.linspace(0, 10, 1000)
plt.title("Naslov grafa") # Naslov
plt.xlabel("Naslov X osi") # X os
plt.ylabel("Naslov Y osi") # Y os
plt.axis('equal') # Odrzi omjer slike (aspect ratio)
# Nacrtaj sinus funkciju ravnom zelenom linijom s nazivom sin(x)
plt.plot(x, np.sin(x), '-g', label='sin(x)')
# Nacrtaj kosinus funkciju isprekidanom plavom linijom s nazivom cos(x)
plt.plot(x, np.cos(x), '--b', label='cos(x)')
plt.legend() # Nacrtaj legendu na grafu
```



Slika 5: Graf s osnovnim parametrima vizualizacije

Prilikom crtanja linija Matplotlib uzima podatke koji predstavljaju skupinu točaka te ih povezuje u obliku linije. Iste podatke je moguće prikazati i u obliku točaka gdje svaka točka predstavlja jedan podatak. Oblik podatka koji se prikazuje na grafu se također može odrediti s kodovima. Popis kodova za prikaz svih vrsta vizualizacija se može ispisati naredbom `help(plt.plot())`. Svi kodovi oblika vizualizacije se prosljeđuju kao tekstualni atribut bez naziva u `.plot()` funkciji (npr. `plt.plot(listaTocaka, 'o')` gdje 'o' predstavlja kod za prikaz podataka u obliku standardnih točaka). Navedeni kodovi za prikaz točaka se također mogu kombinirati s kodovima boja i linija (npr. `plt.plot(listaTocaka, '-ok')` ispisuje ravnu crnu liniju i iste podatke u obliku standardnih točaka dok `plt.plot(listaTocaka, '-xr')` ispisuje isprekidanu crvenu liniju te iste podatke u točkama oblika X).

Matplotlib sadrži posebnu funkciju `plt.scatter()` namjenjenu isključivo za prikaz grafova s točkama (engl. *scatter plots*). Navedena funkcija omogućuje dodjeljivanje dodatnih atributa svakoj točki zasebno (poput veličine i boje točaka). Na taj način se može prikazati više podataka te se omogućuje kreiranje vizualno privlačnijih grafova. No ukoliko se dodjeljuju dodatni atributi svakoj točki potrebno je kreirati i prikazati legende koje će dodijeljenim podacima dati određeni kontekst. Prikaz legende za raspon boja (engl. *color bar*) je omogućen jednostavnim pozivom metode `plt.colorbar()` koja je malo detaljnije obrađena kod prikazivanja histograma na slici 8. S druge strane, kreiranje točaka različitih veličina za prikaz u legendi je moguće dodavanjem novog, praznog elementa grafa kojemu će se dodijeliti atribut *label* (npr. `plt.plot([], [], s=100 label='Opis')` kreira objekt veličine 100 koji ne ispisuje na grafu zbog proslijeđenih praznih lista umjesto X i Y vrijednosti). Na taj način je kreiran element kojega će zbog *label* atributa "pokupiti" metoda `plt.legend()` i prikazati na grafu u legendi. Primjer kreiranja grafova s točkama i prikaz legende je vidljiv na slici 6 dok se programski kôd nalazi u prilogu.



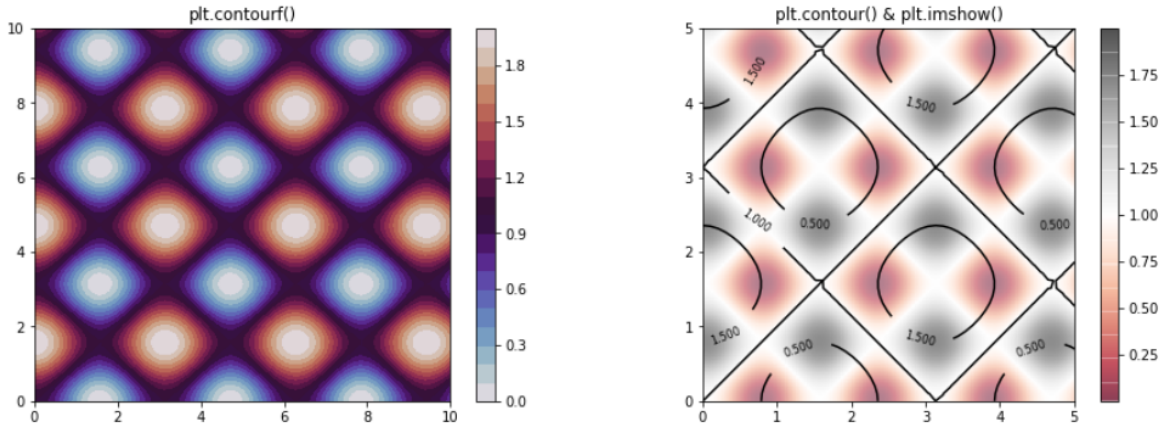
Slika 6: Razlika funkcija .plot() i .scatter()

Prilikom odlučivanja o metodi za prikaz grafova s točkama potrebno je uzeti u obzir broj točaka za prikaz i broj dimenzija podataka koji se žele prikazati. S obzirom na dodatne mogućnosti metode `plt.scatter()`, prilikom izrade vizualizacija se izvode dodatne operacije koje usporavaju rad i narušavaju performanse. Zbog toga je bolji izbor metoda `plt.plot()` ukoliko je podatke moguće prikazati u samo dvije dimenzije, no ako je potrebno prikazati više od dvije dimenzije (poput veličine i/ili boje) metoda `plt.scatter()` bi bila puno bolji izbor.

Kod prikaza podataka s tri dimenzije se mogu koristiti i tzv. *contour* grafovi. *Contour* graf je jednostavan način za prikaz 3D podataka u samo dvije dimenzije korištenjem kontura s naglašavanjem određenih područja različitim bojama. Matplotlib sadrži tri metode za prikazivanje *contour* grafova. Metoda `plt.contour()` prikazuje podatke u obliku linija gdje iscrtkane linije predstavljaju negativne vrijednosti dok ravne linije predstavljaju pozitivne vrijednosti. Ukoliko se žele popuniti praznine između linija može se koristiti metoda `plt.contourf()`. Također postoji i `plt.imgshow()` metoda koja prikazuje *contour* graf u obliku slike. S obzirom da metoda `plt.imgshow()` prikazuje samo sliku grafa, prilikom izrade nema potrebe za crtanjem konture čime se izbjegava kreiranje novog poligona za svaku konturu. Iz tog razloga se preporuča korištenje `plt.imgshow()` metode ukoliko je željeni rezultat samo prikaz grafa.

Prve dvije metode za izradu *contour* grafova se razlikuju samo u prazninama između nacrtanih linija dok metoda `plt.imgshow()` prikazuje graf na drugačiji način zbog čega zahtjeva drugačije atribute za prikaz istog grafa. Metode `plt.contour()` i `plt.contourf()` primaju parametre X, Y, Z koji predstavljaju X, Y i Z osi na grafu respektivno. Također je moguće povećati broj linija koje se kreiraju prosljeđivanjem numeričkog broja bez naziva atributa (predstavlja broj jednako odvojenih intervala unutar raspona podataka). S druge strane, metoda `plt.imgshow()` prima samo Z os zbog čega je potrebno ručno specificirati raspon X i Y osi s parametrom *extent* (npr. `extent=[0, 5, 0, 5]` označavajući sljedeće vrijednosti: X-min, X-max, Y-min, Y-max). Također je potrebno postaviti `origin='lower'` ukoliko se želi dobiti isti rezultat kao i s prve dvije funkcije. Razlog tomu su drugačiji standardi početnih položaja prilikom generiranja *contour* linija (kreće se od donjeg lijevog kuta) i generiranja slike (kreće se od gornjeg lijevog kuta).

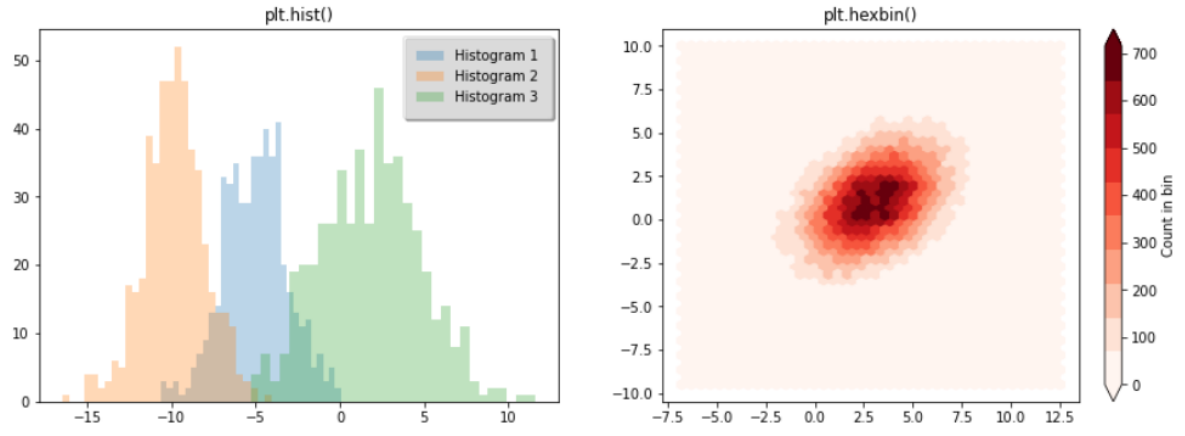
Metoda `plt.contour()` se može kombinirati s metodama `plt.imgshow()` ili `plt.contourf()` kako bi se na jednom grafu prikazale i konture i ispunjene praznine između kontura. Ukoliko se koriste dvije metode u kombinaciji potrebno je smanjiti vidljivost na metodi koja puni praznine između kontura s atributom *alpha* (npr. `alpha=0.5`). Također je moguće metodi `plt.contour()` proslijediti nazive kontura s raznim parametrima kako bi se završni graf dodatno obogatio. S obzirom da *contour* grafovi koriste boje za prikaz treće dimenzije preporuča se prikazati traku u boji s funkcijom `plt.colorbar()`.



Slika 7: Primjer *contour* grafova

Uz prethodno navedene grafove, Matplotlib sadrži jednostavne i intuitivne metode za izradu različitih vrsta histograma. Histogram je grafička reprezentacija distribucije podataka (najčešće) u obliku pravokutnika gdje se podaci grupiraju u određeni numerički raspon. Za prikaz jednostavnog 1D histograma u Matplotlib-u se koristi metoda `plt.hist()` dok se za prikaz 2D histograma mogu koristiti metode `plt.hist2d()` (histogram u obliku mreže kvadrata) i `plt.hexbin()` (histogram u obliku mreže heksagona). Navedenim metodama je potrebno proslijediti podatke za prikaz uz nekoliko opcionalnih atributa za podešavanje parametara izgleda histograma. Neki od najkorištenijih opcionalnih atributa su: *bins* za određivanje broja jednako distribuiranih raspona, *alpha* za postavljanje razine vidljivosti, *color* i *edgecolor* za određivanje boja histograma te *histtype* za određivanje vrste histograma (trenutno podržane vrste su *bar, barstacked, step, stepfilled*). Također je moguće kreirati više histograma na jednom grafu pozivanjem metode `plt.hist()` više puta. Vrijedi napomenuti kako je moguće dohvatiti broj vrijednosti u svakom rasponu bez potrebe crtanja grafa s metodom `np.histogram()` iz NumPy biblioteke (npr. `vrijednosti, raspon = np.histogram(podaci)` gdje će se u varijablu *vrijednosti* zapisati lista broja vrijednosti za svaki raspon dok će se broj raspona zapisati u varijablu *raspon*).

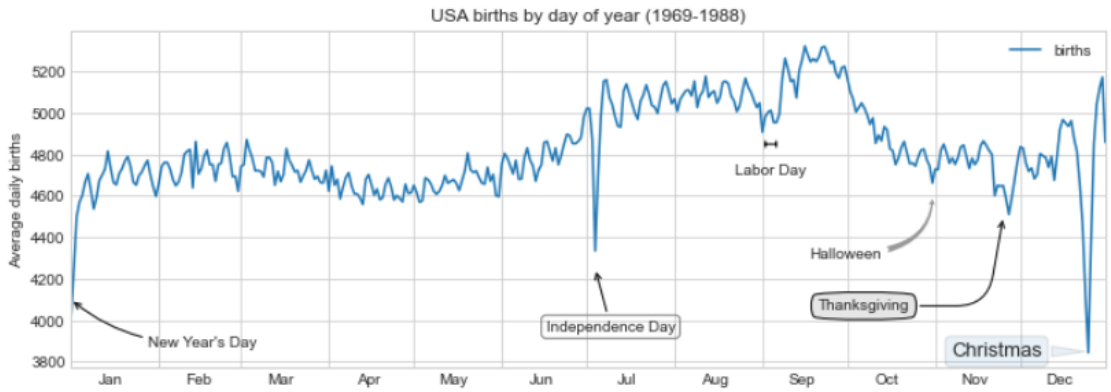
Prilikom izrade svih grafova u Matplotlib-u moguće je dodatno urediti legendu i traku u boji s opcionalnim atributima prilikom poziva metoda `plt.legend()` i `plt.colorbar()` respektivno. Neki od najkorištenijih atributa legende su *ncol* za definiranje broja stupaca, *loc* za definiranje lokacije legende (npr. `loc='upper left'`), *shadow* za prikaz sjene te *fancybox* za prikaz vizualno privlačnijeg okvira legende. S druge strane, traka u boji nudi nešto manji broj opcionalnih atributa za izmjenu izgleda. Neki od opcionalnih atributa su *label* za dodjeljivanje naziva trake u boji te atribut *extend* s kojim se navodi da li se prikazuju vrijednosti izvan granica (engl. *out-of-bounds*). Moguće vrijednosti *extend* atributa su `'min'`, `'max'`, `'both'` i `'neither'` te se na traci u boji ističu u obliku strelice ukoliko su uključeni. Također je moguće definirati fiksni broj boja koje će se koristiti na grafu no ta opcija se definira prilikom odabira boje za prikaz grafa (npr. kao atribut *cmap* u metodi `plt.hist()`). Broj boja se definira dohvaćanjem boja iz `plt.cmap` paketa (npr. `cmap = plt.cm.get_cmap('Reds', 4)` dohvaća četiri boje iz 'Reds' skupa boja). Primjeri 1D i 2D histograma su prikazani na slici 8. dok se programski kôd za prikaz slike nalazi u prilogu.



Slika 8: Primjer 1D i 2D histograma

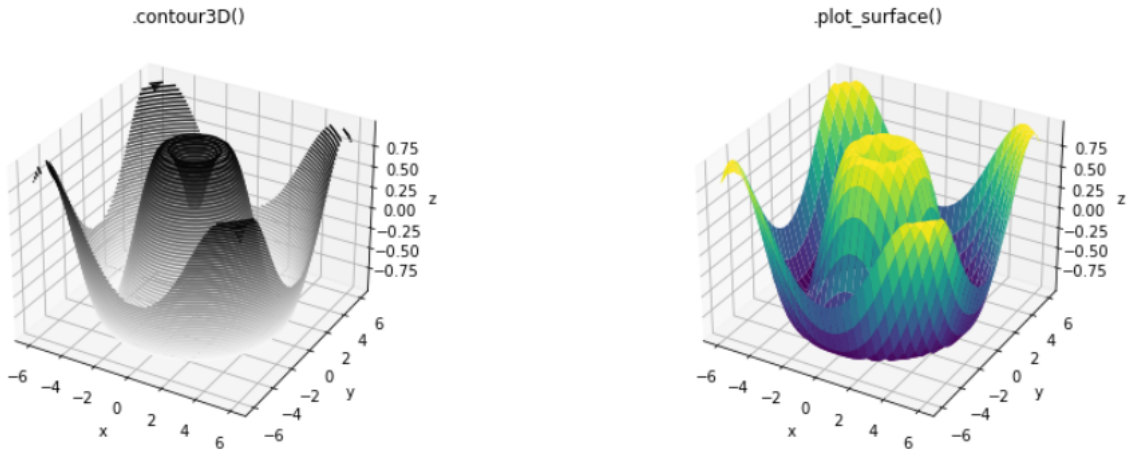
Prilikom izrade grafova često dolazi do potrebe za naglašavanjem pojedinih dijelova kako bi se dao kontekst određenim vrijednostima na prikazanom grafu. Najjednostavniji način za označavanje vrijednosti je pomoću Matplotlib funkcije `plt.annotate()` koja sadrži veliki broj opcionalnih parametara za izmjenu izgleda anotacija. Prilikom poziva `.annotate()` funkcije potrebno je proslijediti dva argumenta: tekst anotacije (bez specificiranja naziva atributa) te vrijednosti X i Y osi na koje se anotacija odnosi (npr. `plt.annotate('tekst', xy=(15, '15.5.2015')` postavlja tekst anotacije na poziciju grafa gdje vrijednost na X osi iznosi 15 te gdje je vrijednost na Y osi jednaka '15.5.2015'). Ukoliko se navedu samo obavezni atributi Matplotlib će ispisati jednostavni tekst anotacije na željenu poziciju. Metoda `.annotate()` sadrži brojne opcionalne atribute za uređivanja izgleda anotacije koji se mogu detaljno istražiti naredbom `plt.annotate?`. Neki od najkorištenijih opcionalnih atributa su *arrowprops* kojemu je moguće proslijediti Python *dictionary* s atributima izgleda strelice između teksta i označenog dijela grafa te atributi *xycoords* i *textcoords* za određivanje vrste koordinatnog sustava atributa *xy* i teksta anotacije respektivno. Ostali opcionalni atributi se prosljeđuju objektu `matplotlib.text.Text` gdje se može detaljnije urediti izgled teksta anotacije. [22]

No ponekad je potrebno ručno postaviti vrijednosti napisane na X i Y osima (engl. *ticks*) kako bi se detaljnije objasnili podaci na grafu. Matplotlib razlikuje glavne *tick*-ove (engl. *major ticks*) za prikaz važnijih podataka i sporedne *tick*-ove (engl. *minor ticks*) za prikaz podataka s manjom važnosti koji su većinom slabije istaknuti na grafu. Za uređivanje *tick*-ova u Matplotlib-u se koriste objekti *Locator* i *Formatter*. Objekt *Locator* se odnosi na linije iscrtane na grafu dok se *Formatter* odnosi na nazive koji se dodjeljuju iscrtanim linijama *Locator*-a. Navedeni objekti lokatora i oblikovača se mogu dohvatiti i postaviti na glavnim i sporednim *tick*-ovima s get/set funkcijama na svakoj osi (npr. ax.yaxis.get_major_locator() dohvaća objekt *Locator* Y osi za glavne tickove dok `ax.xaxis.get_minor_formatter()` dohvaća objekt *Formatter* X osi za sporedne *tick*-ove). Prilikom postavljanja lokatora i oblikovača koriste se posebni objekti koji određuju način definiranja pojedinih *tick*-ova (npr. `plt.NullLocator()` uklanja lokator čime se ne prikazuju linije na grafu, `plt.MaxNLocator(3)` postavlja najviše tri linije na grafu te `plt.FuncFormatter(funkcija)` čime se definiraju vrijednosti naziva pozivom proslijeđene funkcije). Popis, opis i primjeri svih vrsta lokatora i oblikovača se može naći u popisu literature pod brojem [23]. Primjer raznih vrsta anotacija uz ručno podešavanje *tick*-ova je vidljiv na slici 9. dok se kod za prikaz slike nalazi u prilogu.



Slika 9: Primjer grafa s anotacijama i prilagođenim *tick*-ovima [5]

Uz veliki broj metoda za kreiranje 2D grafova, Matplotlib sadrži brojne metode za izradu 3D grafova. Metode za izradu 3D grafova dolaze zajedno sa standardnom instalacijom biblioteke Matplotlib te se mogu uključiti u Python skriptu iz *mpl_toolkits* modula (`from mpl_toolkits import mptlot3d`). Nakon što je uvezen modul za izradu 3D grafova mogu se kreirati `Axes3D` objekti postavljanjem parametra *projection* na `'3d'` prilikom kreiranja `Axes` objekta (npr. `ax = plt.axes(projection='3d')`). Kreirani `Axes3D` objekti sadrže metode za izradu 3D vizualizacija poput `.plot3D, .scatter3D, .contour3D` i `.plot_surface()`. Prilikom poziva metoda za izradu 3D vizualizacija potrebno je proslijediti vrijednosti podataka za sve tri osi (X, Y i Z) u obliku liste (npr. `ax.contour3D(x, y, Z)`). Uz obavezne atribute svaka metoda za sadrži i drugačije opcionalne parametre za postavljanje određenih postavki grafova koje se mogu detaljnije istražiti naredbom `ax.nazivMetode?` (npr. `ax.plot_surface?` ispisuje popis i opis svih obaveznih i opcionalnih atributa metode `.plot_surface()`. Također vrijedi napomenut da je unaprijed definirani broj kontura prilično nizak te prilikom kreiranja 3D grafa s metodom `ax.contour3D` bez eksplicitnog navođenja broja kontura može doći do neočekivanih rezultata. Ukoliko je rezultat nezadovoljavajući može se ručno povećati broj kontura s numeričkim parametrom bez naziva (npr. `ax.contour3D(X, Y, Z, 50)`).



Slika 10: Primjer 3D vizualizacija [5]

Većina metoda za izradu 3D grafova funkcionira isključivo ako su liste podataka (X, Y i Z) istih veličina. Ukoliko podaci koji se trebaju prikazati ne sadrže jednake veličine u svim dimenzijama može se koristiti metoda `ax.scatter3D()` koja prikazuje svaki podatak u obliku točke 3D grafu ne tražeći poveznicu između točaka. No ukoliko se ipak želi prikazati površina na 3D grafu umjesto točaka može se koristiti metoda `ax.plot_trisurf()`. Navedena metoda pronalazi tri susjedne točke čineći površinu u obliku kvadrata nakon čega se spajaju sve kvadratne površine te se ispisuje površina od točaka.

Prilikom izrade 3D vizualizacija moguće je koristiti čarobnu naredbu `%matplotlib notebook` u Jupyter Notebook-u kako bi prikazali interaktivne 3D grafove umjesto slikovnih reprezentacija. Ukoliko se prikazuju interaktivni grafovi potrebno je pozvati funkciju `plt.show()` kako bi se izrađena vizualizacija prikazala. Isključivanjem interaktivnog grafa se uklanjaju sve opcije za interakciju s grafom dok slika samog grafa ostaje ispisana u obliku koji bi se inače ispisao s naredbom `%matplotlib inline`).

4.3. Pandas

S obzirom da je NumPy biblioteka izgrađena za bržu i efikasniju obradu podataka homogenog tipa pojavljuje se problem prilikom obrade heterogenih podataka. Iako NumPy sadrži strukturirane liste koje omogućuju pohranu i obradu heterogenih podataka, strukturirane liste nisu primarna svrha NumPy paketa te ne sadrže veliki broj opcija za rad s takvim podacima. Upravo zbog navedenog nedostatka funkcionalnosti obrade heterogenih podataka Wes McKinney 2008. godine kreira novu Python biblioteku koju naziva Pandas (skraćenica od **pan**el **da**ta). Pandas biblioteka je inicijalno kreirana za jednostavnu i efikasnu analizu financijskih podataka, no zbog jednostavnosti, efikasnosti i velikog broja metoda za obradu heterogenih podataka postaje jedna od glavnih Python biblioteka za analizu i obradu svih tipova podataka. Izrađena je na temelju NumPy strukturiranih lista dodavajući nove objekte i metode za obradu heterogenih podataka. [24]

Glavni objekti Pandas biblioteke nad kojima se izvršavaju sve operacije su objekt `Index` za pohranjivanje indeksa, objekt `Series` za pohranu i obradu jednodimenzionalnih podataka te objekt `DataFrame` za pohranu i obradu dvodimenzionalnih podataka. `Series` objekt predstavlja listu podataka koja sadrži kombinaciju ključ-vrijednost za svaki pohranjeni podatak u listi (kao i NumPy lista). Glavna razlika između `Series` objekta i NumPy liste je fleksibilnost pohranjivanja podataka. Indeks podaci, odnosno ključevi, moraju biti isključivo numeričke vrijednosti ako se radi o NumPy listi dok se u `Series` objektima mogu pohranjivati podaci bilo kojeg tipa. No vrijedi napomenuti da u `Series` objektima svi indeksi i sve vrijednosti moraju biti istog tipa podataka, ali nije nužno da indeksi i vrijednosti međusobno imaju isti tip podataka. Nakon kreiranja objekta je također moguće promijeniti tip podataka indeksa i/ili vrijednosti ukoliko je konverzija podataka moguća. Na ovaj način se omogućuje jednostavna i fleksibilna pohrana različitih tipova podataka izbjegavajući posljedice dinamičke alokacije varijabli.

DataFrame je poboljšana verzija NumPy strukturiranih lista koja se sastoji od nekoliko `Series` objekata čineći dvodimenzionalnu listu podataka (može interpretirati kao tablica). Svaki redak `DataFrame` objekta ima jedinstveni indeks te svi redovi moraju dijeliti podatke istog tipa u svakom stupcu. Svi stupci sadrže nazive prema kojima se tada može izvršavati veliki broj metoda za obradu heterogenih podataka. Neke od glavnih razlika NumPy strukturiranih lista i `DataFrame` objekta su mogućnost dodjeljivanja naziva stupcima i recima, grupiranje podataka po određenim kriterijima, spajanje nekoliko `DataFrame` objekata, rukovanje vremenskim podacima, efikasno dodavanje i brisanje pojedinih podataka, pretraživanje i izmjena podataka te efikasne operacije za rukovanje s nepostojećim ili netočnim podacima.

S obzirom da `Series` i `DataFrame` objekti pohranjuju podatke u obliku ključ-vrijednost, `Index` objekt je kreiran isključivo za pohranjivanje ključeva liste. Ključna razlika između `Index` objekta i NumPy liste je teža promjena podataka kod `Index` objekata kako bi bilo sigurnije i jednostavnije dijeliti ključeve s drugim objektima/listama. Series, `DataFrame` i `Index` objekti, zajedno sa svim navedenim mogućnostima Pandas paketa, će biti detaljnije objašnjeni i prikazani na primjerima programskog kôda za lakše razumijevanje u poglavlju 5.

5. Biblioteka Pandas

Kao što je navedeno u poglavlju 4.3. Pandas je Python biblioteka za jednostavnu i efikasnu obradu i analizu heterogenih podataka. Kreirana je 2008. godine od strane programera Wes McKinney-a zbog nedostatka funkcionalnosti obrade heterogenih podataka NumPy biblioteke. U ovom poglavlju će se detaljno obraditi Pandas objekti, funkcionalnosti koje Pandas pruža za obradu heterogenih podataka te neke naprednije metode Pandas biblioteke.

5.1. Pandas objekti

Glavni objekti Pandas biblioteke su `Series`, `DataFrame` i `Index` čiji su osnovni aspekti opisani u poglavlju 4.3. Navedeni Pandas objekti su napravljeni koristeći NumPy strukturirane liste zbog čega se mogu smatrati kao naprednijim verzijama NumPy strukturiranih lista. U ovom poglavlju će se detaljnije objasniti Pandas objekti, sličnosti i razlike između objekata, mogućnosti kreiranja pojedinih objekta te načini dohvaćanja podataka.

Series objekt je jednodimenzionalna lista indeksiranih podataka koja se može protumačiti kao specijalizirani Python rječnik (engl. *dictionary*) ili kao generalizirana NumPy lista. Za razliku od Python rječnika, svi ključevi i sve vrijednosti pohranjene u `Series` objektu moraju biti istog tipa podataka, iako nije nužno da međusobno (ključ i vrijednost) budu istog tipa. Na ovaj način se postiže efikasnost izvršavanja NumPy liste i fleksibilnost pohranjivanja heterogenih podataka Pandas rječnika. No s obzirom da je `Series` objekt izrađen korištenjem NumPy liste može se promatrati i kao generalizirana NumPy lista koja može pohranjivati sve tipove podataka kao ključeve umjesto isključivo numeričkih vrijednosti. Upravo zbog navedenih poveznica između `Series` objekta, NumPy liste i Python rječnika, kreiranje `Series` objekta je vrlo intuitivno, odnosno kreira se jednostavnim prosljeđivanjem liste podataka ili Python rječnika. Prilikom kreiranja `Series` objekta je također moguće navesti opcionalni atribut `index` kojemu se prosljeđuje lista ključeva/indeksa novog `Series` objekta. S obzirom da rječnici već imaju definirane ključeve, kreiranje `Series` objekta prosljeđivanjem rječnika i korištenjem atributa *index* će popuniti jedino vrijednosti koje postoje u listi indeksa dok se ostale vrijednosti ignoriraju. Primjeri kreiranja `Series` objekta su prikazani u programskom kôdu ispod.

```
import pandas as pd

print(pd.Series([2, 4, 6])) # Kreiranje proslijedivanjem liste
# Kreiranje s brojem i 'index' parametrom
# Isti broj će se zapisati na svim indeksima
print(pd.Series(5, index=[100, 200, 300]))
print(pd.Series({2:'a', 1:'b', 3:'c'})) # Rjecnik
# Rjecnik i 'index' parametar
# Popune se jedino vrijednosti ciji kljuc postoji u index listi
print(pd.Series({2:'a', 1:'b', 3:'c'}, index=[3, 2]))
```

Najčešće korištena kratica: za Pandas biblioteku je **pd** te se u daljnjim primjerima podrazumijeva korištenje Pandas biblioteke putem navedene kratice. Izvršavanje osnovnih operacija nad `Series` objektom poput dohvaćanja podataka, dodavanja novih zapisa ili izmjene postojećih vrijednosti je prilično intuitivno te se može izvršiti na različite načine. Jedan od načina dohvaćanja podataka je putem operacije dohvaćanja podniza koja promatra `Series` objekt kao listu podataka (npr. `series['Petar':'Ivan']` dohvaća sve vrijednosti počevši od vrijednosti čiji ključ je 'Petar' i završavajući na vrijednosti čiji ključ je 'Ivan'). Dohvaćanje svih vrijednosti i/ili ključeva je moguće s metodama `.keys()`, `.values` i `.items()` koje promatraju `Series` objekt kao Python rječnik. Također moguće provjeriti postoji li vrijednost u objektu standardnim Python izrazom `'vrijednost' in objekt`. Novi zapis u `Series` objekt se dodaje dodjeljivanjem vrijednosti novom ključu objekta (npr. `series['kljuc'] = vrijednost`).

```
series = pd.Series({'Zagreb': 10000, 'Slavonski Brod': 35000,
                'Osijek': 31000, 'Varaždin': 42000})

print(series['Zagreb':'Osijek']) # Dohvacanje podniza
print('Zagreb' in series) # Postoji li vrijednost
print(series.keys()) # Ispis kljuceva
print(series.values) # Ispis vrijednosti

series["Dubrovnik"] = 20000 # Dodavanje nove vrijednosti
print(list(series.items())) # Ispis kljuceva i vrijednosti
```

Prilikom dohvaćanja podniza potrebno je obratiti pozornost na koji način se određuje raspon indeksa. Ovisno o načinu raspona indeksa postoji dva načina dohvaćanja podniza:

- **Eksplicitni** - eksplicitno određivanje raspona prema nazivu indeksa. Eksplicitno indeksiranje uključuje zadnji indeks (npr. `series['Zagreb':'Osijek']` uključuje i zapis s ključem 'Osijek').
- **Implicitni** - implicitno određivanje raspona indeksa brojevima počevši od 0. Implicitno indeksiranje ne uključuje zadnji indeks (npr. `series[0:2]` prikazuje prvu i drugu vrijednost objekta - treća vrijednost objekta s indeksom 2 se ne prikazuje).

No ovakav pristup indeksiranja postaje neintuitivan kada `Series` objekt sadrži eksplicitne brojeve kao indeks vrijednosti. Ako se koristi indeksiranje proslijeđeni broj se promatra kao eksplicitna vrijednost indeksa, odnosno ako se dohvaća samo jedan zapis (npr. `series[1]` dohvaća vrijednost čiji je ključ jednak 1). No ako se koristi operacija dohvaćanja podniza tada se proslijeđeni parametri gledaju kao implicitne vrijednosti indeksa (npr. `series[1-3]` dohvaća zapise na drugoj i trećoj poziciji u `Series` objektu). Zbog prethodno opisanih komplikacija Pandas biblioteka uključuje metode `.loc[]` za eksplicitno i `.iloc[]` za implicitno indeksiranje. Navedene metode omogućuju kontrolu nad odabirom načina indeksiranja pružajući veću fleksibilnost prilikom dohvaćanja podataka.

```
data2 = pd.Series(['a', 'b', 'c', 'd', 'e'], index=[1, 2, 3, 4, 5])

# .loc[] - Eksplicitno indeksiranje
print(data2.loc[1]) # Vrijednost ciji je indeks 1
# Vrijednosti izmedu indeksa 1 i 3 (ukljucujuci i zadnji indeks)
print(data2.loc[1:3])

# .iloc[] - Implicitno indeksiranje
print(); print(data2.iloc[1]) # Drugi zapis u listi
# Sve vrijednosti od druge do cetvrte pozicije (bez zadnje)
print(data2.iloc[1:3])
```

Kako bi se omogućilo pohranjivanje dvodimenzionalnih podataka kreiran je objekt DataFrame. `DataFrame` objekt se može promatrati kao tablica u kojoj su svaki redak i svaki stupac jedan `Series` objekt. Svaki redak u `DataFrame` objektu sadrži indeks zbog čega se također može promatrati kao rječnik čiji ključ predstavlja naziv indeksa retka dok je vrijednost cijeli `Series` objekt. Važno napomenuti da svi reci `DataFrame` objekta dijele stupce koji također imaju fiksno definirani tip podataka te im se može dodijeliti naziv. Dakle svaki redak i svaki stupac `DataFrame` objekta se može promatrati kao zasebni `Series` objekt. `DataFrame` objekt je moguće kreirati na sljedeće načine: [19]

- **Učitavanjem podataka iz datoteke** - Najkorištenija Pandas metoda za učitavanje podataka iz datoteka je `pd.read_csv()` koja učitava podatke iz CSV datoteke. Također postoje i metode `pd.read_excel()` za učitavanje podataka iz excel datoteka (potrebno instalirati *xlrd* modul) te `pd.read_sql()` za učitavanje podataka iz baze podataka (potrebno instalirati *sqlalchemy* biblioteku za pristup bazi podataka).
- **Učitavanjem podataka iz drugih objekata** - Pandas omogućuje kreiranje `DataFrame` objekta prosljeđivanjem `Series` objekta, Python rječnika, Python liste, NumPy strukturirane liste i zapisa u strukturiranim oblicima poput JSON-a.

```
df = pd.read_csv("datasets/datoteka.csv") # CSV
df = pd.read_excel("datasets/datoteka.xlsx") # Excel

# Rjecnik Series objekata
print(pd.DataFrame({'prviStupac': series1,'drugiStupac': series2}))
# Strukturirana NumPy lista
print(pd.DataFrame(np.zeros(3, dtype=[('prvi', 'i8'), ('drugi', 'f8')])))
# DataFrame s jednim stupcem iz NumPy liste
print(pd.DataFrame(numpyLista, columns=['prvi']))
# Python rjecnik
print(pd.DataFrame([{'a': i, 'b': 2 * i} for i in range(3)]))
```

DataFrame objekti se također mogu interpretirati kao specijalizirani Python rječnik i/ili generalizirana NumPy lista. U odnosu na Python rječnik, `DataFrame` mapira naziv stupca (ključ) na `Series` objekt (vrijednosti u stupcu) dok Python rječnik mapira bilo koju kombinaciju ključ-vrijednost. No ako se `DataFrame` promatra kao lista `Series` objekata koji dijele retke i stupce, tada pruža više mogućnosti u odnosu na NumPy dvodimenzionalnu listu s obzirom da se svakom retku i/ili stupcu može dodijeliti naziv. Nazivima redaka i stupaca `DataFrame` objekta se može pristupiti metodama *dataframe.index* i *dataframe.columns* respektivno.

Operacija indeksiranja nad `DataFrame` objektima se izvršava nad stupcima prosljeđivanjem naziva stupca kojemu se želi pristupiti. Stupcima `DataFrame` objekta je moguće pristupiti pozivom stupca kao atributa (npr. `dataframe.stupac` ili indeksiranjem (npr. `dataframe["stupac"]`). Pristupanje stupcima kao atributima `DataFrame` objekta nije moguće ukoliko naziv stupca nije tekstualni podatak ili ako sadrži razmake. Također postoji mogućnost da će se naziv stupca poklopiti s nazivom `DataFrame` metode nakon čega može doći do nenamjernog poziva metode umjesto stupca. Uz to, kreiranje novog stupca nije moguće s ovim pristupom (npr. umjesto `df.noviStupac = 1` je potrebno koristiti `df["noviStupac"] = 1`). Zbog navedenih razloga se preporuča pristupati vrijednostima stupaca indeksiranjem.

Za razliku od indeksiranja, operacija dohvaćanje podniza se odnosi na retke `DataFrame` objekta te slijedi sva pravila kao i dohvaćanje podniza nad `Series` objektima (npr. `dataframe['a':'c']` dohvaća sve retke počevši od retka s indeksom 'a' i završavajući s retkom čiji je indeks 'c'). S obzirom da su nazivi indeksa eksplicitno navedeni dohvaćeni podaci obuhvaćaju i zadnji zapis s indeksnom 'c'. Također vrijedi napomenuti da `DataFrame` objekti sadrže metode `.loc[]` i `.iloc[]` za dohvaćanje podataka prosljeđivanjem eksplicitnih i implicitnih vrijednosti indeksa respektivno. Dohvaćanje podataka na temelju uvjeta (engl. *boolean masking*) se također odnosi na retke `DataFrame` objekta (npr. `dataframe[dataframe.iznos > 4]` dohvaća sve retke čija je vrijednost u stupcu 'iznos' veća od 4).

```
post_broj = pd.Series({'Zagreb': 10000, 'Varaždin': 42000,
                'Osijek': 31000, 'Slavonski Brod': 35000})
stanovnistvo = pd.Series({'Zagreb': 806341, 'Varaždin': 46946,
                'Osijek': 107784, 'Slavonski Brod': 59141})
data = pd.DataFrame({'postal':post_broj, 'pop_total':stanovnistvo})

print(data.values[0]) # Lista - dohvaca prvi redak
print(data["area"]) # DataFrame - dohvaca prvi stupac

# Prikazi retke do 4 pozicije (iskljucujuci 4 poziciju)
# Prikazi stupce do 3 pozicije (iskljucujuci 3 poziciju)
print(data.iloc[:3, :2]) # Implicitno indeksiranje
# Prikazi retke do retka s kljucem 'Osijek' (ukljucujuci i 'Osijek')
# Prikazi stupce do stupca 'postal' (ukljucujuci i 'postal')
print(data.loc[:'Osijek', :'postal']) # Eksplicitno indeksiranje
```

5.2. Obrada podataka

Biblioteka Pandas nasljeđuje efikasnost izvršavanja operacija iz NumPy modula korištenjem unarnih vektoriziranih funkcija poput negacije i binarnih vektoriziranih funkcija (dodavanje, oduzimanje, množenje i dijeljenje). Unarne i binarne vektorizirane funkcije su detaljnije objašnjene u poglavlju 4.1. No Pandas nadograđuje navedene NumPy vektorizirane funkcije s obzirom da ih izvršava nad dvodimenzionalnim podacima koji se mogu referencirati putem retka/indeksa i/ili stupca. Prilikom izvršavanja unarnih vektoriziranih funkcija indeksi ostaju isti dok se pozivom binarnih vektoriziranih funkcija indeksi resetiraju i sortiraju. Resetiranje i sortiranje indeksa uzima uniju indeksa oba proslijeđena objekta te dodjeljuje NaN vrijednosti ako se susretne s nepostojećim vrijednostima. Dakle ako se izvršava funkcija zbrajanja nad dva `DataFrame` objekta s nazivima stupaca ('a', 'b', 'c') i ('a', 'b') tada će se zbrojiti stupci 'a' i 'b' iz oba objekta dok će stupac 'c' sadržavati NaN vrijednost. Sve aritmetičke operacije s NaN vrijednosti vraćaju NaN kao rezultat. Ukoliko se žele izbjeći operacije s NaN vrijednostima mogu se koristiti Pandas metode umjesto operatora koje imaju parametar *fill_value* za popunjavanje NaN vrijednosti u objektu (npr. umjesto `dataframe1 + dataframe2` koristiti `dataframe1.add(dataframe2, fill_value=0)`. Tablica 7. sadrži popis Pandas univerzalnih funkcija i Python operatora koji se mogu koristiti umjesto punog naziva funkcije.

Tablica 7: Aritmetičke Pandas univerzalne funkcije (engl. *ufuncs*)

Operator	Univerzalna funkcija	Opis
+	.add	Zbrajanje
-	.subtract	Oduzimanje
*	.multiply	Množenje
/	.divide	Dijeljenje
//	.floordiv	Dijeljenje bez ostatka
**	.pow	Potenciranje
%	.mod	Ostatak pri dijeljenju

Sve Pandas univerzalne funkcije izvršavaju operacije nad svakim podatkom u `DataFrame` objektu zasebno. Zbog velike efikasnosti NumPy-a, osim unarnih i binarnih vektoriziranih funkcija, Pandas nasljeđuje i ostale funkcionalnosti NumPy biblioteke poput *broadcasting*-a, indeksiranja i operacije dohvaćanja podniza detaljnije opisanih u poglavlju 4.1. Većina preuzetih funkcionalnosti su nadograđene u Pandas paketu kako bi mogle jednostavno i efikasno izvršavati operacije nad heterogenim dvodimenzionalnim podacima kojima s dodatnim nazivima stupaca i redaka. Programski kôd ispod prikazuje izvršavanje raznih operacija nad `DataFrame` objektom uz objašnjenje u obliku komentara.

```
# Kreiraj dva DataFrame-a s nasumicnim brojevima
rand = np.random.RandomState(40)
A = pd.DataFrame(rand.randint(10, size=(3, 4)), columns=list('ABCD'))
B = pd.DataFrame(rand.randint(10, size=(3, 3)), columns=list('ABC'))
```

```
print(A - B) # Bez popunjavanja NaN vrijednosti
# Popuni NaN vrijednosti s najvecom vrijednosti iz liste B
print(); print(A.subtract(B, fill_value=B.stack().max()))

print(); print(A) # 3x4 DataFrame
# Oduzimanje DataFrame od Series - broadcasting
# Svaki redak se oduzima sa Series objektom
print(); print(A - A.iloc[0])
print(); print(A.subtract(A['B'], axis=0)) # Oduzimanje stupaca

poluredak = A.iloc[0, ::2] # Svaka druga vrijednost u prvom retku
# Oduzima samo vrijednosti koje imaju isti naziv stupca
# Ostali stupci koji ne postoje u poluretku ce imati NaN vrijednost
print(); print(A - poluredak)
```

No većina podataka koji se trebaju obraditi nisu konzistentni, sadrže prazna polja te NaN ili None vrijednosti. S obzirom da svaka aritmetička operacija s NaN vrijednosti rezultira novom NaN vrijednosti, prilikom obrađivanja podataka treba posebno obratiti pozornosti na vrijednosti koje nedostaju u podacima. S obzirom da je None vrijednost Python objekt, ako lista sadrži None vrijednost tip liste će automatski biti tipa `object` zbog čega će se sve operacije izvršavati na Python razini. S druge strane, NaN vrijednost je podržana u programskom jeziku C zbog čega je moguće efikasno izvršavati operacije s NaN vrijednostima. Iz tog razloga se preporuča koristiti NaN vrijednosti kad god je to moguće. Podaci koji nedostaju se mogu označiti na dva načina:

- **Pomoću liste `bool` vrijednosti** (engl. *Boolean Masks*) - u ovom slučaju se kreira lista istog oblika i dimenzije kao i podaci nad kojima je potrebno označiti nepostojeće vrijednosti. Pomoćna lista sadrži vrijednost 0/`False` ako isti indeks u listi podataka sadrži podatak, odnosno vrijednost 1/`True` ako isti indeks u listi podataka ne sadrži vrijednost. Osim pohranjivanja pozicija nepostojećih vrijednosti, *boolean masking* se može koristiti i za filtriranje podataka `DataFrame` objekta. Prosljeđivanjem `bool` liste (npr. `dataframe [bool_lista]`) se ispisuju samo oni zapisi čiji indeks u `bool` listi sadrži vrijednost 1 (odnosno `True`).
- **Pomoću zamjenske vrijednosti** - sve nepostojeće vrijednosti se zamjenjuju s unaprijed definiranom vrijednosti koja se nakon toga tretira kao nepostojeća. Podatak koji u ovom slučaju može predstavljati nepostojeću vrijednost ovisi o dogovoru te može biti nasumični broj (npr. -99999: sve nepostojeće vrijednosti se zamjenjuju s vrijednosti -99999) ili globalno prihvaćena konvencija zapisa nepostojećih vrijednosti poput *Not a Number* (kratica: *NaN*). NaN vrijednost je posebna vrijednost za decimalne podatke te je univerzalno prihvaćena i korištena u većini programskih jezika. Ostali tipovi podataka poput : `integer`-a, : `string`-a i : `bool`-a ne sadrže unificiranu vrijednost za pohranu nepostojećeg podatka.

Biblioteka Pandas većinom koristi **zamjenske vrijednosti** None i NaN za pohranu i obradu nepostojećih vrijednosti dok se lista `bool` vrijednosti koristi za filtriranje podataka. S obzirom da je None vrijednost jedan od Python objekata koji drastično smanjuju efikasnost izvršavanja operacija, Pandas preferira NaN za pohranu nepostojećih vrijednosti te pretvara None u NaN kad god je to moguće. None objekt također rezultira greškom ukoliko se nad njim izvode aritmetičke operacije dok iste operacije nad NaN objektom vraćaju NaN vrijednost kao rezultat. Iz tog razloga se preporuča korištenje NaN vrijednosti iako se mora paziti prilikom rukovanja podacima s NaN vrijednostima kako se ne bi nepotrebno izgubili bitni podaci. Pandas sadrži veliki broj metoda za rukovanje s nepostojećim vrijednostima:

- **.isnull()** i **.notnull()** - kreira `bool` listu `bool` gdje su vrijednosti 1/`True` za `.isnull()` i 0/`False` za `.notnull()` na pozicijama gdje se nalaze nepostojeće vrijednosti.
- **.dropna()** - filtrira podatke uklanjajući sve redove koji sadrže nepostojeće vrijednosti. Sadrži dodatne parametre s kojima se može odrediti način uklanjanja podataka poput *axis* koji određuje uklanjaju li se stupci ili reci (npr. `axis='columns'`). Uz *axis* moguće je koristiti i parametar *how* s kojim se određuje u kojim slučajevima se uklanja redak/stupac (npr. `how='all'` uklanja stupac/redak samo ako su sve vrijednosti nepostojeće ili `how='any'` koji uklanja stupac/redak ako je barem jedna vrijednost nepostojeća). S parametrom *tresh* se može odrediti minimalni broj vrijednosti koje stupac/redak moraju sadržavati inače se uklanjaju.
- **.fillna()** - metoda prima minimalno jedan parametar čiju vrijednost puni na svim nepostojećim vrijednostima Pandas objekta. Također je moguće odrediti metodu punjenja nepostojećih vrijednosti s parametrom *method* (npr. *Forward Fill* - `.fillna(method='ffill')` popunjava nepostojeće vrijednosti s vrijednosti koja joj prethodi). Uz atribut *method* je moguće proslijediti i atribut *axis* kojim se određuje način popunjavanja vrijednosti (npr. `.fillna(method='bfill', axis=0)` puni nepostojeće vrijednosti sa sljedećom vrijednosti u istom stupcu; `axis=1` preuzima sljedeću vrijednost u istom retku).

```
x = pd.Series([np.nan, 2], dtype=float); print(x)
# Puni nepostojece vrijednosti sa sljedecom vrijednosti
# x[0] je NaN - preuzima vrijednost od x[1]
print(); print(x.fillna(method='bfill'))

# 3x3 DataFrame
df = pd.DataFrame(np.arange(1, 10).reshape(3, 3))
df.iloc[1, 2] = np.nan # Postavi NaN vrijednost
print(); print(df)
# Ukloni retke koji sadrze barem jednu nepostojecu vrijednost
print(); print(df.dropna(axis='rows', how='any'))
```

Prilikom dodavanja novih ili ažuriranja postojećih vrijednosti potrebno je obratiti pozornost na tip podatka koji se dodaje. Ukoliko se tipovi podataka nove vrijednosti i ostalih vrijednosti u Pandas objektu razlikuju, Pandas će automatski promijeniti tip podataka objekta kako bi mogao pohraniti novu vrijednost. Pretvorba podataka također ovisi o tome dodaje li se potpuno nova vrijednost ili se izmjenjuje postojeća. Ako se u `bool` listi postojeća vrijednost postavi na None tada Pandas pretvara None objekt u `False` ne mijenjajući tip podataka liste. No ukoliko se na istoj listi doda novi podatak čija se vrijednost postavi na None tada Pandas pretvara tip podataka objekta iz `bool` u Python objekt. Ako se vrijednost novog podatka postavi na NaN (umjesto None) tada se tip podataka objekta pretvara iz `bool` u float koji, za razliku od Python objekta, podržava brze i efikasne univerzalne funkcije.

Također postoji metoda `.astype()` koja omogućuje ručnu promjenu tipa podatka koji se pohranjuje u Pandas objektu (npr. `series.astype("float64")` pretvara tip podataka `Series` objekta u float64). Prije ručnog pretvaranja tipa podataka u tip podatka `bool` potrebno je imati na umu da Python pretvara vrijednosti None i 0 u `False` dok će sve ostale vrijednosti poprimiti vrijednost `True` (uključujući i NaN). Iz navedenih razloga je potrebno obratiti pozornost na krajnji tip podataka ako se podaci dodaju ili izmjenjuju te na vrijednosti podataka kod ručne pretvorbe tipa podataka. Tijekom dodavanja i ažuriranja podataka te ručne pretvorbe tipa podataka se zbog efikasnosti preporuča izbjegavati Python objekt tip podataka.

```
lista = pd.Series([10, 0, 1]) # Lista brojcanih vrijednosti
print(lista)
lista[0] = True # Pretvara brojcanu listu u Object listu
print(); print(lista)

# Rucno pretvori Object listu u bool listu
lista = lista.astype("bool")
print(); print(lista)
lista[0] = None # None vrijednost se pretvara u False
print(); print(lista)

lista[1] = np.nan # Pretvara bool listu u float64 listu
print(); print(lista)
lista[4] = None # Pretvara float64 listu u Object listu
print(); print(lista)

# Rucno pretvori Object listu u bool listu
# Pretvara None i 0 u False, a NaN i 1 u True
print(); print(lista.astype("bool"))
```

Najčešće i najbitnije operacije većine studija slučaja su prikupljanje i obrada podataka iz različitih izvora te spajanje nekoliko skupina podataka u jedan objekt, odnosno spajanje više `DataFrame` objekata u jedan. Pandas omogućuje spajanje objekata na dva načina:

- **Spajanje dodavanjem vrijednosti** - spaja dva Pandas objekta tako što na kraju prvog objekta doda sve vrijednosti drugog objekta. Ukoliko se spajaju `DataFrame` objekti s različitim stupcima Pandas popunjava stupce bez podataka s NaN vrijednostima. Dostupne metode za spajanje dodavanjem vrijednosti su `.append()` i `.concat()`.
- **Spajanje prema ključnim riječima** - spajanje dva Pandas objekta prema pravilima relacijske algebre tražeći zajednički stupac oba objekta čiji se podaci koriste prilikom spajanja. Metoda vraća grešku ako ne postoji stupac s istim imenom u oba Pandas objekta te dohvaća samo one retke koji imaju isti zapis u zajedničkom stupcu (obavlja unutarnje spajanje nad dva `DataFrame` objekta). Dostupne metode za spajanje prema ključnim riječima su `.join()` i `.merge()`.

Metoda `.append()` se koristi za jednostavno spajanje dva Pandas objekta bez dodatnih parametara te se poziva kao metoda Pandas objekta (npr. `df1.append(df2)` dodaje vrijednosti objekta df2 na kraj objekta df1). S druge strane, metoda `.concat()` sadrži veliki broj parametara za bolju kontrolu nad spajanjem te se poziva nad Pandas bibliotekom, prosljeđujući objekte kao parametre (npr. `pd.concat(df1, df2)`). Neki od najbitnijih parametara su *axis* koji određuje prema kojoj osi će se objekti spojiti (`axis=0` za spajanje redaka i `axis=1` za spajanje stupaca), parametar *join* koji određuje način spajanja objekata (osnovna vrijednost je `join='outer'` koja prikazuje uniju stupaca dok `join='inner'` prikazuje presjek stupaca). Također postoje parametri *verify_integrity* koji vraća grešku ako objekti sadrže iste indekse te *ignore_index* koji resetira indeks vrijednosti.

```
# Biblioteka za rad s vremenskim podacima
from datetime import datetime
np.random.seed(datetime.now().microsecond)

df1 = pd.DataFrame(np.random.randint(10, size = (5, 5)),
                   columns=list('ABCDE'))
df2 = pd.DataFrame(np.random.randint(10, size = (5, 5)),
                   columns=list('ABCDE'))

# .append() - ne resetira indeks
print(df1.append(df2))

# ignore_index=True - resetiraj indeks vrijednosti
print(); print(pd.concat([df1, df2], ignore_index=True))
```

S druge strane, spajanje objekata prema ključnim riječima se omogućuje spajanje dva `DataFrame` objekta prema pravilima relacijske algebre. Pozivom metoda `.join()` i `.merge()` bez prosljeđivanja dodatnih parametara se objekti spajaju unutarnjim spajanjem. Metoda `.join()` automatski uzima indeks vrijednosti kao stupac prema kojem spaja objekte te se poziva kao metoda `DataFrame` objekta (npr. `df1.join(df2)` spaja objekte df1 i df2 prema indeks vrijednostima). S druge strane, metoda `.merge()` omogućuje specificiranje stupaca prema kojima će se objekti spojiti. Ako se ne navedu nazivi stupaca tada metoda traži stupac s istim nazivom u oba objekta te prema tom stupcu spaja objekte. Koristeći navedene metode potrebno je imati na umu tri vrste relacijskih spajanja:

- **Jedan prema jedan** (engl. *One-to-One*) - stupac prema kojemu se objekti spajaju ne sadrži ponavljajuće vrijednosti u niti jednom objektu.
- **Jedan prema više** (engl. *One-to-Many*) - stupac prema kojemu se objekti spajaju sadrži ponavljajuće vrijednosti u samo jednom objektu. Dohvaćaju se sve ponavljajuće vrijednosti.
- **Više prema više** (engl. *Many-to-Many*) - stupac prema kojemu se objekti spajaju sadrži ponavljajuće vrijednosti u oba objekta. Dohvaćaju se sve kombinacije ponavljajućih vrijednosti iz oba objekta (kartezijev produkt vrijednosti???).

Obje metode za spajanje objekata pravilima relacijske algebre imaju atribut *how* za specificiranje načina spajanja (npr. `how='outer'` za vanjsko spajanje; dozvoljene vrijednosti su *inner, outer, left* i *right*). Uz atribut *how* moguće je koristiti i atribut *on* koji se kod metode `.join()` se odnosi na razinu ili naziv indeksa prema kojemu se objekti spajaju (u slučaju `MultiIndex` objekta objašnjenog u poglavlju 4.3.). No isti atribut se kod metode `pd.merge()` odnosi na naziv stupca prema kojemu se objekti spajaju. Ukoliko je potrebno spojiti objekte prema stupcima različitih imena tada se mogu koristiti atributi *right_on* i *left_on* koji specificiraju naziv stupaca za spajanje lijevog i desnog objekta respektivno. Također postoje atributi *left_index* i *right_index* koji se postave na `True` ako se prilikom spajanja lijevog ili desnog objekt spaja s indeks vrijednostima. Ukoliko objekti sadrže iste nazive stupaca mogu se koristiti atributi *lsuffix* i *rsuffix* pozivom metode `.join()` ili atribut *suffixes* pozivom metode `pd.merge()` (npr. `suffixes=["_left", "_right"]`).

```
from datetime import datetime
np.random.seed(datetime.now().microsecond)
df1 = pd.DataFrame(np.random.randint(10, size = (5, 5)),
                   columns=list('ABCDE'))
df2 = pd.DataFrame(np.random.randint(10, size = (5, 5)),
                   columns=list('ABCDE'))

# Spoji prema indeks vrijednostima
print(); print(df1.join(df2, lsuffix='_X', rsuffix='_Y'))
# Spoji prema stupcu 'A' lijevog i indeks vrijednosti desnog objekta
print(); print(pd.merge(df1, df2, left_on="A", right_index=True))
```

Uz navedene operacije s nepostojećim vrijednostima i metode za spajanje objekata Pandas sadrži i razne agregatne funkcije koje mogu pružiti dodatne informacije o podacima. Agregatne funkcije smanjuju dimenziju podataka što znači da će rezultat agregatne funkcije nad 2D DataFrame-om biti 1D `Series` objekt, odnosno skalar ako se agregatna funkcija poziva nad 1D `Series` objektom. Agregatne funkcije nad `DataFrame` objektom izvršavaju operacije nad svakim stupcem vraćajući jedan zapis u rezultatu. Atributom *axis* je moguće promijeniti os na kojoj se izvršavaju operacije (npr. `df.max(axis=1)` ispisuje maksimalnu vrijednost svakog retka dok `df.sum()`, odnosno `df.sum(axis=0)`, zbraja i ispisuje vrijednosti svakog stupca). Popis svih agregatnih funkcija se nalazi u tablici 8. Uz sve navedene agregatne funkcije postoji i metoda `.describe()` koja izračunava nekoliko osnovnih agregatnih funkcija poput `.min(), .max(), .std(), .mean()` i `.count()` nad podacima te ih prikazuje u obliku tablice (npr. `df.dropna().describe()` izračunava osnovne agregatne funkcije ignorirajući redove s NaN vrijednostima).

Tablica 8: Agregatne Pandas funkcije

Funkcija	Opis
.count()	Ukupni broj zapisa
.prod()	Umnožak svih zapisa
.sum()	Zbroj svih zapisa
.std(); .var()	Standardna devijacija i varijacija
.first(); .last()	Prvi i zadnji zapis
.mean(); .median()	Aritmetička sredina i centralna vrijednost
.min(); .max()	Minimalni i maksimalni zapis

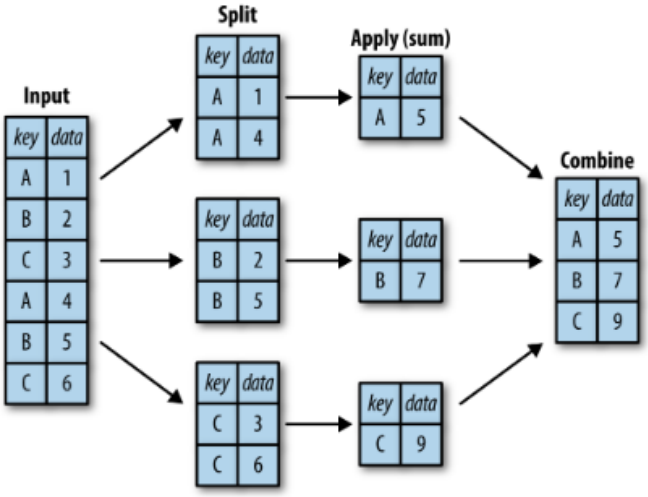
Agregatne funkcije se mogu koristiti zajedno s *GroupBy* operacijom. *GroupBy* je jedna od najkorisnijih operacija Pandas biblioteke koja omogućuje podjelu podataka u određene grupe nad kojima se izvršava određena operacija nakon čega se dobiveni rezultati spajaju u jedan objekt. S obzirom da se u sklopu jedne *GroupBy* operacije odvija veći broj akcija, za lakše razumijevanje se može podijeliti na sljedeće korake:

Grupiraj podatke - podaci se dijele u grupe prema specificiranim ključevima koji se prosljeđuju kao parametar funkcije `.groupby()` (npr. `df.groupby('column1')` grupira cijeli objekt prema podacima stupca s nazivom 'column1' dok `df.groupby('column1')['column2']` grupira samo vrijednosti stupca 'column2' prema podacima stupca s nazivom 'column1'). Podatke je moguće grupirati i prosljeđivanjem liste vrijednosti koja određuje grupu svakog retka objekta (npr. `df.groupby([1, 1, 2])` grupira prva dva retka u grupu s nazivom 1, a treći redak u grupu s nazivom 2). Također se može proslijediti i Python funkcija koja prima indeks vrijednost retka i vraća naziv željene grupu proslijeđenog retka (npr. `df.groupby(str.lower)` stavlja svaki redak u grupu s vrijednosti `str.lower(indeks_vrijednost)`). Ako je potrebno ručno odrediti koje indeks vrijednosti se svrstavaju u koju grupu može se proslijediti Python rječnik (npr. `df.groupby('A': 'samoglasnik', 'B': 'suglasnik')` svrstava retke s indeksom 'A' u grupu naziva 'samoglasnik, a retke s indeksom 'B' u grupu s nazivom 'suglasnik').

Izvrši operaciju - izvršava određenu funkciju nad svakom grupom odvojeno te se poziva nakon grupiranja (npr. `df.groupby(column).sum()` prvo grupira vrijednosti prema stupcu s nazivom 'column' nakon čega izračunava zbroj elemenata nad svakom grupom zasebno). Pandas podržava veliki broj operacija koje se mogu izvršiti u sklopu *GroupBy* operacije poput agregatnih funkcija, funkcija filtriranja i transformiranja podataka te korisničkih funkcija.

- **Agregatne funkcije** - mogu se pozvati direktno nakon `.groupby()` operacije ili putem funkcije `.aggregate()` (npr. `df.groupby(column).aggregate([min, max])` izvršava `.min()` i `.max()` funkcije nad grupama). Naziv funkcije se može proslijediti u obliku stringa ili naziva funkcije - `'min'`, `min` i `np.min` su ispravno proslijeđeni argumenti za izvršavanje funkcije `.min()`.
- **Funkcije filtriranja** - na temelju određenog uvjeta se filtriraju podaci objekta te se vraćaju samo oni reci koji ispunjavaju traženi uvjet. Funkcije filtriranja se pozivaju metodom `.filter()` (npr. `df.groupby(column).filter(lambda x : x['data2'].std() > 4)`) izvršava anonimnu funkciju koja vraća 1/`True` za one retke čija je standardna devijacija vrijednosti stupca 'data2' veća od 4 time filtrirajući samo takve retke).
- **Funkcije transformiranja** - izvršavaju određenu operaciju nad svakim retkom unutar grupe pozivom funkcije `.transform()`. Funkcije transformiranja vraćaju isti objekt s istim oblikom i dimenzijom mijenjajući jedino vrijednosti objekta (npr. `df.groupby(column).transform(lambda x : x['data2'] + 4)`) uvećava vrijednosti stupca 'data2').
- **Korisničke funkcije** - moguće je izvršiti vlastitu funkciju nad svakom grupom elemenata pozivom funkcije `.apply()` i prosljeđivanjem naziva vlastite funkcije kao parametar. Jedini uvjet je da kreirana funkcija prima `DataFrame` objekt te da vraća DataFrame, `Series` ili skalar kao rezultat.

Spoji rezultate - Nakon grupiranja podataka i izvršavanja zadanih operacija se spajaju svi dobiveni rezultati u jedan objekt koji se tada vraća kao finalni rezultat *GroupBy* operacije.



Slika 11: Postupak *GroupBy* operacije [5]

Primjer programskog kôda za izvršavanje *GroupBy* operacija korištenjem sva četiri načina izvršavanja operacija je prikazan ispod. Prvotno se kreira `DataFrame` objekt s tri stupca nad kojim će se izvršavati *GroupBy* operacija. Prve dvije *GroupBy* operacije koriste metodu `.aggregate()`. Prvoj metodi `.aggregate()` je proslijeđena lista agregatnih funkcija za izvršavanje nad svim grupama dok se drugoj `.aggregate()` metodi prosljeđuje Python rječnik koji spaja stupac s operacijom koju je potrebno izvršiti nad tim stupcem. Treća *GroupBy* operacija filtrira retke `DataFrame` objekta ovisno o uvjetu koji je proslijeđen kao parametar u funkciju `.filter()`. Zadnje dvije *GroupBy* funkcije `.transform()` i `.apply()` se razlikuju u načinu izvršavanja proslijeđene operacije. Funkcija `.transform()` izvršava zadanu operaciju nad svakim retkom unutar grupe održavajući oblik i dimenziju (npr. oduzimanje retka od aritmetičke sredine grupe). S druge strane, metodi `.apply()` se može proslijediti bilo koja funkcija koja prima `DataFrame` objekt kao parametar. Proslijeđena funkcija također mora vraćati `DataFrame` ili `Series` objekt ili skalar. Sama funkcija može sadržavati bilo kakve izračune što uvelike povećava fleksibilnost *GroupBy* operacije.

```
rand = np.random.RandomState(0)
df = pd.DataFrame({'key': ['A', 'B', 'C', 'A', 'B', 'C'],
                   'data1': range(6), 'data2': rand.randint(0, 10, 6)},
                   columns = ['key', 'data1', 'data2'])
print(df)

# [Aggregate] Izvrsi agregatne funkcije nad svim stupcima
print(); print(df.groupby('key').aggregate(['min', np.median, max]))
# [Aggregate] Izvrsi odredene funkcije za svaki stupac zasebno
print(); print(df.groupby('key')
                   .aggregate({'data1': 'min','data2': 'max'}))

# [Filter] Filtriraj vrijednosti prema drugom stupcu
print(); print(df.groupby('key')
                   .filter(lambda x : x['data2'].std() > 4))

# [Transform] Oduzmi vrijednosti grupe od aritmeticke sredine grupe
print(); print(df.groupby('key').transform(lambda x: x - x.mean()))

# Korisnicka funkcija
def add_data3(x):
    x['data3'] = x['data1'] / x['data2']
    return x # Vraca cijeli DataFrame objekt

# [Apply] Izvrsi korisnicku funkciju nad svakom grupom podataka
print(df.groupby('key').apply(add_data3))
```

Pandas biblioteka je izrađena za pohranu i obradu heterogenih podataka zbog čega mora uzeti u obzir i obradu tekstualnih podataka. S obzirom da NumPy ne sadrži operacije za obradu tekstualnih podataka potrebno je koristiti sporije petlje koje su podložne greškama u slučaju nepostojećih vrijednosti. Pandas rješava taj problem uvođenjem vektoriziranih operacija nad tekstualnim podacima koje delegiraju izvršavanje operacija u optimiziraniji programski jezik C drastično poboljšavajući performanse. Također, vektorizirane funkcije izvršavaju operacije nad svakim elementom objekta čime se pojednostavljuje sintaksa. Pandas sadrži veliki broj vektoriziranih funkcija nad tekstualnim podacima poput `.swapcase()`, `.strip()`, `.lower()`, `.upper()`, `.find()`, `.capitalize()`, `.len()`, `.startswith()`, `.endswith()`, `.isdigit()`, `.isspace()`, `.istitle()`, `.isdecimal()`, `.split()` i `.partition()`. Popis i opis svih vektoriziranih funkcija nad tekstualnim podacima Pandas biblioteke se može ispisati naredbom `help(pd.Series.str)`. Sve vektorizirane funkcije nad tekstualnim podacima u biblioteci Pandas se izvršavaju pozivanjem atributa `.str` (npr. `series.str.lower()` pretvaran velika u mala slova u `Series` objektu).

Pandas također podržava različite metode za rad s *Regular Expression* izrazima (kratica: *RegEx*). Neke od metoda su `.match()` za provjeru *RegEx* izraza nad svakim elementnom objekta, `.findall()` za vraćanje svih elemenata koji ispunjavaju *RegEx* izraz te `.extract()` za provjeru *RegEx* izraza i vraćanje definiranih grupa u *RegEx* izrazu kao tekstualnu vrijednosti. Određene vektorizirane funkcije poput `.replace`, `.count()`, `.split()`, `.rsplit()` i `.contains()` također podržavaju *RegEx* izraze (npr. `names.str.count('([A-Za-z])+')` provjerava koliko ima grupiranih kombinacija slova koji sadrže isključivo mala i velika slova u svakom elementu objekta). [25]

No ponekad jedan tekstualni element sadrži više informacija u obliku kodova gdje svaki kod predstavlja jednu vrijednost. Na taj način jedan zapis može sadržavati veći broj informacija što može biti korisno kada postoji grupiranje objekata gdje jedan objekt može pripadati više grupa odjednom. U tom slučaju znak 'A' može predstavljati da je osoba državljanin Republike Hrvatske, 'B' da osoba voli pecati te 'C' i 'D' da osoba ima vozačku dozvolu i višu stručnu spremu respektivno. U ovakvom slučaju je moguće pohraniti sve informacije u jednom stupcu s graničnikom (npr. 'A;C' označava da je osoba iz RH te da posjeduje vozačku dozvolu). Takve informacije je moguće pretvoriti u binarnu matricu s metodom `.get_dummies()` nad kojom se tada mogu efikasno izvršavati razne operacije. Metoda se poziva nad tekstualnim podacima te joj se prosljeđuje graničnik koji odvaja kodove (npr. `series['info'].str.get_dummies(";")` kreira binarnu matricu odvajajući podatke stupca 'info' prema graničniku ";").

```
# RegEx izrazi
names = pd.Series(['Petar', 'Ivan Ivic', 'Marko Maric'])
print(names.str.match('([A-Za-z])+'))# Ispunjava li element RegEx izraz
print(names.str.startswith('P')) # Vraca True ako element pocinje s 'P'

# Metoda get_dummies()
names_info = pd.DataFrame({'name':names,'info':['B;A', 'D;A','D']})
names_info["info"].str.get_dummies(";")
```

Kod obrade heterogenih podataka često se dolazi u kontakt i s vremenskim podacima. Zbog toga Python ima paket `datetime` namijenjen isključivo za pohranu i obradu takvih podataka. No zbog dinamičnog pohranjivanja podataka i neefikasnog izvršavanja operacija nad Python objektima NumPy sadrži vlastite objekte za rad s vremenskim podacima poput `datetime64` i `timedelta64` koji omogućuju obradu podataka putem efikasnih vektoriziranih funkcija opisanih u poglavlju 4.1. Objekti `datetime64` i `timedelta64` su izrađeni na temelju osnovne vremenske jedinice koja se može promijeniti ovisno o željenoj preciznosti vremena. S obzirom da su oba objekta pohranjena u 64-bita postoji ograničeni vremenski raspon koji se može zapisati u pojedini objekt. Povećavanjem osnovne vremenske jedinice se povećava vremenski raspon koji se može pohraniti ali se smanjuje vremenska preciznost (npr. `np.datetime64('2020-05-04', 'ns')` kreira `datetime64` objekt koji može spremati vrijeme u nanosekundama u rasponu od ± 292 godina). Dostupni kodovi uz opis osnovne vremenske jedinice i raspona vremena koji se može pohraniti su opisani u tablici 9.

Tablica 9: Odnos vremenskog raspona i preciznosti [5]

Kod	Osnovna vremenska jedinica	Raspon vremena u godinama
D	Dan	± 2.5e16
h	Sat	± 1.0e15
m	Minuta	± 1.7e13
s	Sekunda	± 2.9e12
ms	Milisekunda	± 2.9e9
us	Mikrosekunda	± 2.9e6
ns	Nanosekunda	± 292

Biblioteka Pandas nadograđuje NumPy objekte `datetime64` i `timedelta64` s dodatnim metodama za pretvorbu raznih podataka u vremenske objekte te za izvođenje raznih operacija nad kreiranim vremenskim objektima. Ovisno o vrsti pohranjenog vremenskog podatka Pandas sadrži tri objekta:

- **Timestamp** - odnosi se na fiksnu točku u vremenu (npr. 16.8.2020. 17:00). Povezani objekt `DatetimeIndex` pohranjuje indeks vrijednosti tipa Timestamp. `Timestamp` objekt je izrađen koristeći NumPy `datetime64` objekt te predstavlja efikasnu zamjenu za Python `datetime` objekt.
- **Period** - odnosi se na trajanje između dvije točke u vremenu te se većinom koristi kod specijalnih slučajeva gdje postoje intervali vremena koji se ne preklapaju i konstantno ponavljaju (npr. 24 sata u danu). Trajanje između dvije točke se računa na temelju dva *datetime64* objekta iz NumPy biblioteke. Pandas također sadrži povezani objekt `PeriodIndex` koji pohranjuje indeks vrijednosti tipa Period.
- **Timedelta** - odnosi se na točno određenu vremensku duljinu (npr. 45 sekundi). Timedelta objekt je implementiran koristeći `timedelta64` objekt iz NumPy biblioteke te predstavlja efikasnu zamjenu za Python `timedelta`. Također postoji `TimedeltaIndex` koji pohranjuje indeks vrijednosti tipa Timedelta.

Pandas omogućuje veliku fleksibilnost prilikom kreiranja objekata `Timestamp, Period` i `Timedelta`. Jedan od načina kreiranja vremenskih podataka je putem metode `pd.to_datetime()` koja vraća `Timestamp` ako je proslijeđena jedna vrijednost ili `DatetimeIndex` objekt ako je proslijeđena lista vrijednosti. Metodi `pd.to_datetime()` je moguće proslijediti datum u raznim tekstualnim oblicima nakon čega metoda razdvaja dan, mjesec i godinu iz teksta te kreira jedan `Timestamp` objekt (npr. `pd.to_datetime(['2020-Aug-6', '08-07-2020'])` kreira `DatetimeIndex` objekt s dva zapisa koji su proslijeđeni u različitim tekstualnim oblicima). Uz to je moguće pomaknuti vremenske podatke za određeni broj definirane vremenske jedinice metodom `.shift()` (npr. `datum.shift(5, 'm')` pomiče vrijeme za 5 minuta unaprijed).

Pandas također omogućuje kreiranje liste vremenskih zapisa s određenim fiksnim razmakom koristeći funkciju `pd.date_range()`. Funkciji se prosljeđuju tri parametra: datum od kojeg počinje kreiranje liste zapisa te parametri *period* i *freq* koji predstavljaju broj kreiranih zapisa i fiksni razmak između svakog zapisa (npr. `pd.date_range('2015-07-03', periods=8, freq='1H')` kreira 8 vremenskih zapisa u razmaku od 1 sati). Veliki broj kodova frekvencija (npr. H:sati, T:minute, S:sekunde, L:milisekunde, B:radni dan, W-WED:tjedni početak u srijedu, BQS-APR:početak poslovne godine u ožujku...) i mogućnost spajanja kodova (npr. 2H2T:dva sata i dvije minute) omogućuju veliku fleksibilnost prilikom određivanja fiksnog razmaka između kreiranih datuma. [26]

```
# Prosljedivanjem liste datuma se kreira DatetimeIndex objekt
# Sadrzi fiksnu tocku u vremenu (npr.~07-07-2020 00:00)
dates = pd.to_datetime([datetime(2020, 7, 3),
    '4th of July, 2020','2020-Jul-6', '07-07-2020', '20200708'])
print(); print(dates)

# Pomakni vremenske podatke tjedan dana unaprijed
print(); print(dates.shift(7, freq='D'))

# Pretvaranje DatetimeIndex u PeriodIndex
# 'D' predstavlja duljinu jednog dana; sadrzi cijeli dan u zapisu
print(); print(dates.to_period('D'))
# Oduzimanje datuma vraca objekt TimedeltaIndex (vremenska duljina)
print(); print(dates - dates[0])

# Kreiraj listu od 8 vremenskih zapisa
# Razmak izmedu zapisa: 2 sata 30 minuta i 30 sekundi
print(); print(pd.date_range('2015-07-03', periods=8, freq='2H30T30S'))

# Kreiraj listu od 9 vremenskih zapisa
# Razmak izmedu zapisa: 1 poslovni kvartal pocevsi od cetvrtog mjeseca
print(); print(pd.date_range('2020-08', periods=9, freq="BQS-APR"))
```

5.3. Napredne funkcionalnosti

Većina podataka se može reprezentirati u obliku tablice gdje nazivi redaka i stupaca pružaju kontekst pohranjenim podacima. Na taj način jedan redak može sadržavati podatke o državama dok stupac može sadržavati vrstu podataka koja se sprema za svaku državu poput ukupne populacije. No prilikom obrade stvarnih podataka ponekad se jedan zapis može vezati na više od dvije informacije (npr. podatak koji se odnosi na ukupnu populaciju stanovništva određene države u određenoj godini). U takvim slučajevima se može koristiti `MultiIndex` objekt koji omogućuje pohranjivanje indeks vrijednosti stupaca i/ili redaka u dvije ili više razina (npr. 1. razina - naziv države; 2. razina - godine po državi). `MultiIndex` objekt pohranjuje informacije o indeks vrijednostima u atributima na sljedeći način:

- **levels** - dvodimenzionalna lista koja pohranjuje sve vrijednosti po razinama (npr. ako `multiIndex.levels` ispisuje `[['Hrvatska', 'Austrija', 'Makedonija'], [2019, 2020]]` tada sadrži nazive država na prvoj razini i godine na drugoj razini indeksa).
- **codes** - dvodimenzionalna lista koja pohranjuje međusobne odnose vrijednosti prema razinama, odnosno označava na kojim pozicijama se nalaze multi indeksu 2D listi *levels* (npr. ako `multiIndex.codes` ispisuje `[[0, 0, 1, 1, 2, 2], [0, 1, 0, 1, 0, 1]]` tada se prvi multi index nalazi na poziciji (0,0), drugi na poziciji (0,1), itd).
- **names** - sadrži nazive razina multi indeksa u obliku liste (npr. `multiIndex.names` ispisuje `["Drzava", "Godina"]` označavajući da se na prvoj razini pohranjuju podaci o državama dok se na drugoj razini spremaju podaci o godinama).

Slika 12. prikazuje `DataFrame` objekt s multi indeksom na recima i stupcima. Prilikom dohvaćanja podataka s multi indeksom prvo se indeksiraju vrijednosti stupaca od najviše razine prema nižima te nakon toga vrijednosti redaka (npr. u slučaju `DataFrame` objekta sa slike 12. `df["Petar"]["Turist"]["Italija"][2019]` dohvaća broj dana koje je Petar proveo kao turist u Italiji 2019. godine). Ako je potrebno indeksirati podatke prema indeks vrijednostima redaka tada je potrebno transponirati `DataFrame` objekt, odnosno zamijeniti retke i stupce (npr. `df.T["Italija"]` dohvaća sve podatke za Italiju). Također je moguće dohvatiti podatke u obliku liste s atributom `.values` (npr. `df.T["Italija"][2019].values` dohvaća sve podatke Italije za 2019. godinu).

Ime		Petar		Marin		Ivana	
Putovanje		Turist	Posao	Turist	Posao	Turist	Posao
Drzava	Godina						
Italija	2019	35	28	19	47	33	40
	2020	17	5	43	3	40	2
Austrija	2019	36	30	46	30	31	41
	2020	28	21	41	29	39	21

Slika 12: DataFrame s `MultiIndex` stupcima i recima

Osim dohvaćanja podatak putem indeks vrijednosti moguće je koristiti i metode `.loc[]` i `.iloc[]` koje su detaljno opisane u poglavlju 4.1. S obzirom da funkcija `.iloc[]` izvodi operacije isključivo s numeričkim indeksima osnovne 2D liste podataka, vrlo lako je moguće dohvatiti samo određene podatke zanemarujući `MultiIndex` vrijednosti (npr. `df.iloc[:, 0]` dohvaća i ispisuje prvi redak zajedno sa svim `MultiIndex` vrijednostima). S druge strane, metoda `.loc[]` koristi MultiIndeks vrijednosti za indeksiranje zbog čega je potrebno pisati nazive indeks vrijednosti umjesto numeričkih vrijednosti (npr. `df.loc[("Italija", 2019), "Petar"]` dohvaća broj dana koje je Petar proveo 2019 godine u Italiji). Vrijedi napomenuti da metoda `.loc[]` ne podržava operaciju dohvaćanja podniza (npr. `df.loc[(:, 2019), (:, 'Posao')]` rezultira `SyntaxError` greškom) nego se mora koristiti `IndexSlice` objekt. Prije dohvaćanja podniza nad `MultiIndex`-om potrebno je sortirati indeks vrijednosti (ako nisu sortirane). Pokušaj dohvaćanja podniza nad `MultiIndex`-om rezultira greškom `UnsortedIndexError`. Sortiranje indeksa je moguće s metodama `.sort_index()` i `.sortlevel()`.

No ponekad je potrebno prebaciti određenu razinu indeksa sa stupaca na retke ili obrnuto. Metoda `.stack()` uzima indeks vrijednosti stupaca najniže razine te ih prebacuje u indeks vrijednosti redaka najniže razine. U slučaju slike 12. izvršavanje naredbe `df.stack()` će pomaknuti vrijednosti *Turist* i *Posao* na najnižu razinu indeks vrijednosti redaka nakon čega će `MultiIndex` redaka sadržavati tri razine predstavljajući državu, godinu i razlog putovanja respektivno. Funkcija `.unstack()` obavlja suprotnu operaciju, odnosno prebacuje indeks vrijednosti redaka najniže razine u indeks vrijednosti stupaca najniže razine. S obzirom da metode `.stack()` i `.unstack()` obavljaju suprotne operacije moguće je s jednom operacijom poništiti drugu. Nadalje, korištenjem atributa *level* se može eksplicitno definirati koju razinu je potrebno prebaciti (npr. `df.stack(level=0)` prebacuje najvišu razinu indeksa vrijednosti stupca na najnižu razinu indeksa vrijednosti stupca). Također je moguće pretvoriti indeks vrijednosti redaka u podatke stupaca (kreirajući novi stupac) s metodom `.reset_index()`. Pozivanjem metode `.reset_index()` bez parametara se svi indeks podaci redaka prebacuju u podatke stupaca no moguće je prebaciti samo određenu razinu indeks vrijednosti retka s atributom *level* (npr. u slučaju DataFrame-a sa slike 12. `df.reset_index(level=1)` pretvara indeks vrijednosti godine u stupac). Programski kôd ispod prikazuje načine indeksiranja `MultiIndex`-a `DataFrame` objekta kreiranog za ispis slike 12.

```
# Indeksiranje putem naziva indeks vrijednosti
print(df["Petar"]["Turist"]["Italija"][2019])
print(df["Petar", "Turist"]["Italija"])

# Indeksiranje metodama .loc[] i .iloc[]
print(df.loc[("Italija", 2019), "Petar"])
print(df.iloc[:, 0])

# Indeksiranje operacijom dohvacanja podniza
#df.loc[(:, 2019), (:, 'Posao')] # SyntaxError!
idx = pd.IndexSlice
health_data.loc[idx[:, 2019], idx[:, 'Posao']]
```

Multi indeksiranje omogućuje lakše pohranjivanje višedimenzionalnih podataka u 2D `DataFrame` objekt, no ponekad je potrebno nad 2D podacima izvršiti grupiranje po više stupaca te nad dobivenim podacima izvršiti određene operacije. Izvršavanje *GroupBy* operacije nad jednim stupcem je detaljnije opisana u poglavlju 4.2. no stvari se dodatno zakompliciraju kada je potrebno grupirati po više stupaca. Moguće je proslijediti listu stupaca u `.groupby()` metodu te odrediti stupac nad kojim se operacije izvršavaju i operaciju koja se izvršava (npr. `df.groupby(['Spol', 'Klasa'])['Godina'].aggregate('mean').unstack()` izračunava prosječni broj godina po spolu i putnoj klasi). No ovakav način grupiranja se nepotrebno komplicira i rezultira nečitkim programskim kôdom. Zbog toga Pandas sadrži metodu `.pivot_table()` koja predstavlja jednostavniji način grupiranja objekta prema dvije ili više vrijednosti (npr. `df.pivot_table('Godina', index='Spol', columns='Putna_klasa')` također izračunava prosječni broj godina po spolu i putnoj klasi). Metoda prima sljedeće parametre za detaljnije specificiranje načina grupiranja:

- **aggfunc** - opcionalni atribut kojemu se mogu proslijediti metoda, lista metoda ili Python rječnik. Ako se *aggfunc* ne navede tada se automatski postavlja na *mean* funkciju (zbog toga je izračun aritmetičke sredina osnovna operacija `.pivot_table()` metode). Ukoliko se proslijedi metoda ili lista metoda tada se proslijeđene metode izvršavaju nad stupcima objekta koji su navedeni u atributu *values*. Ako se proslijedi Python rječnik tada se ključ odnosi na stupac, a vrijednost na operaciju koja se izvršava nad stupcem. Na taj način se može detaljnije odrediti nad kojim stupcem se izvršava koja operacija.
- **values** - opcionalni atribut koji određuje stupac nad kojim će se operacija izvršiti. Naziv atributa *values* se ne mora eksplicitno navoditi (npr. `df.pivot_table('Godina')` ili `df.pivot_table(values='Godina')`). Ako se atribut ne navede operacije se izvršavaju nad svim stupcima osim stupaca prema kojima se podaci grupiraju.
- **index** i **columns** - definiraju stupce prema kojima se podaci grupiraju. Grupe će se ispisati u retku završne tablice i stupcu završne tablice respektivno (npr. rezultat operacije `df.pivot_table('Godina', index='Spol', columns='Putna_klasa')` će biti `DataFrame` objekt koji će sadržavati grupiranje po spolu u recima i grupiranje po putnoj klasi u stupcima).
- **dropna** i **fill_value** - atributi s kojima se određuje obrada nepostojećih vrijednosti. Atribut *dropna* uklanja sve stupce koji ne sadrže podatke dok atribut *fill_value* zamjenjuje sve nepostojeće vrijednosti sa željenim znakom (npr. `dropna=True` ili `fill_value=0`).
- **margins** i **margins_name** - ako je atribut *margins* postavljen na `True` tada se izračunava zbroj vrijednosti stupaca nad kojima se izvršavaju operacije. Atribut *margins_name* dodatno specificira nad kojim stupcem je potrebno izračunati zbroj. Ako *margins* nije postavljen ili je postavljen na `False` tada se neće izračunavati zbroj vrijednosti neovisno o tome je li postavljen atribut *margins_name*.

Slika 13. prikazuje podatke `DataFrame` objekta korištenog u primjerima metode `.pivot_table()` dok se programski kôd za prikaz slike nalazi u prilogu.

	Godina	Putna_klasa	Visina	IQ	Spol
0	15	3	113.06	100	Z
1	1	3	30.00	111	Z
2	9	1	62.84	120	M
3	17	2	44.12	84	Z
4	35	2	193.43	118	Z
...	...	...	...	...	...
196	33	3	196.57	118	M
197	32	1	200.00	118	M
198	4	3	38.81	115	Z
199	31	3	141.97	113	Z
200	18	2	174.54	119	M

Slika 13: Prikaz DataFrame-a korištenog za metodu `.pivot_table()`

Programski kôd ispod prikazuje razliku između metoda `.groupby()` i `.pivot_table()` uz različite primjere korištenja metode `.pivot_table()`. Osim osnovnih operacija, metoda `.pivot_table()` omogućuje grupiranje prema više stupaca prosljeđivanjem liste s dvije vrijednosti u atribute *index* i *column*. Prva vrijednost liste *index* atributa mora biti stupac prema kojemu se grupira dok drugi atribut mora biti lista koja sadrži naziv grupe za svaki redak. Vrijednosti liste *column* atributa su obrnuti (lista prema kojoj se grupira i naziv stupca za grupiranje).

```
# Prosjecni broj godina po spolu i klasi [groupby vs pivot_table]
print(df.groupby(['Spol', 'Putna_klasa'])['Godina'].aggregate('mean')
    .unstack())
print(df.pivot_table('Godina', index='Spol', columns='Putna_klasa'))

# Prosjecni broj godina, min visina i max IQ po spolu i putnoj klasi
print(df.pivot_table(index='Spol', columns='Putna_klasa',
    aggfunc={'Godina':np.mean, 'Visina':'min', 'IQ': max},
    margins = True))

# Prosjecna visina po spolu, putnoj klasi i godinama
# Godine su podijeljene na 4 kvartila (Quantile Cut)
godina = pd.qcut(df['Godina'], 4)
print(df.pivot_table('Visina', ['Spol', godina], 'Putna_klasa'))

# Prosjecna visina po spolu, putnoj klasi, godinama i IQ-u
# IQ je podijeljen na dvije grupe (80-100] & (100-120]
iq = pd.cut(df['IQ'], [80, 100, 120])
godina = pd.qcut(df['Godina'], 4)
print(df.pivot_table('Visina', ['Spol', godina], [iq, 'Putna_klasa']))
```

Nadalje, iako su NumPy i Pandas operacije puno efikasnije od ugrađenih Python operacija i dalje je potrebno obratiti pozornost na performanse prilikom izvršavanja operacija. Pozivom Pandas i NumPy metoda se prije izvršavanja svake operacije privremeno kreiraju objekti u memoriji što može dovesti do nepotrebnog alociranja memorije i narušavanja performansi. Navedeni nedostatak se posebno ističe prilikom izvršavanja nekoliko operacija odjednom (npr. `(numpyList < 5) & (numpyList > 1)` izvršava tri operacije te alocira memoriju na svakom koraku čime se nepotrebno alocira memorija tri puta). Kako bi se izbjeglo nepotrebno alociranje memorije moguće je koristiti Pandas metode **.eval()** i **.query()**. Navedene metode koriste Numexpr biblioteku koja delegira izvršavanje operacija u programski kod C izbjegavajući nepotrebnu alokaciju memorije. Na manjem skupu podataka su NumPy i Pandas metode efikasnije no prilikom obrade većeg broj podataka s više operacija metode `.eval()` i `.query()` pružaju puno bolje performanse.

```
# Kreiraj cetiri 10000x1000 DataFrame-a s nasumicnim brojevima
rng = np.random.RandomState(42)
df1, df2, df3, df4 = (pd.DataFrame(rng.rand(10000, 1000))
    for i in range(4))

# NumPy vektorizirana funkcija zbrajanja DataFrame objekata ~ 77ms
%timeit df1 + df2 + df3 + df4
# Pandas eval() funkcija zbrajanja DataFrame objekata ~ 33ms
%timeit pd.eval('df1 + df2 + df3 + df4')
```

Metodame `.eval()` i `.query()` primaju obvezni parametar *expr* u obliku tekstualnog podatka koji se može proslijediti bez eksplicitnog navođenja naziva parametra. Atribut *expr* sadrži izraz koji je potrebno izvršiti nad podacima te se prosljeđuje u programski jezik C u obliku tekstualnog podatka gdje se interpretiraju vrijednosti svakog znaka i izvršavaju operacije. Obje metode podržavaju veliki broj operatora poput aritmetičkih, bitovnih, and/or operatora i operatora usporedbe čime pružaju veliku fleksibilnost prilikom kreiranja izraza. Također podržavaju vezanje izraza (npr. `df1 > df2 < df3` je jednako kao `(df1 > df2) & (df2 < df3)`), korištenje podataka iz Pandas objekata te izvršavanje funkcija indeksiranja (npr. `pd.eval('df1.T[1] / df2.iloc[2]')` dijeli vrijednosti stupca s nazivom 1 transponiranog df1 objekta s vrijednostima trećeg retka df2 objekta).

Metoda `.eval()` se može pozvati nad Pandas bibliotekom (npr. `pd.eval((df.A - df.B) * (df.C / 2))`) ili direktno nad `DataFrame` objektom (npr. `dataframe.eval('(A - B) * (C / 2)')`). Ukoliko se pozivaju direktno nad objektom tada se omogućuje izmjena i dodavanje novih podataka te se podaci stupaca mogu dohvatiti direktno s nazivom stupaca (npr. `df.eval('D = (A - B) * C')` kreira novi stupac s nazivom "D" na temelju izračuna podataka nad stupcima "A", "B" i "C"). Također je moguće pristupiti lokalnim varijablama sa znakom @ (npr. `lokalna_varijabla=2; df.eval('A + @lokalna_varijabla')` zbraja vrijednosti stupca "A" s lokalnom Python varijablom).

Metoda `.eval()` uvijek vraća rezultat operacije proslijeđenog izraza nad podacima. Ako se metodi proslijedi uvjetni izraz tada je rezultat `bool` lista koja vraća `True` ako je redak prošao uvjetni izraz. No ako je potrebno ispisati podatke redaka koji su prošli uvjetni izraz tada se cijeli izraz mora proslijediti kao parametar metodi indeksiranja (npr. `pd.eval('df[(df.A < 0.5) & (df.B < 0.5)]')`). Zbog toga je kreirana metoda `.query()` koja izvršava uvjetni izraz te filtrira `DataFrame` na temelju dobivene `bool` liste vraćajući podatke redaka koji su prošli uvjetni izraz (bez potrebe za pozivanjem metode indeksiranja). Metodu `.query()` je moguće pozvati samo nad `DataFrame` objektom (npr. `dataframe.query('(A < 0.5) & (B < 0.5)')`).

```
# DataFrame s 1000 redaka i stupcima A, B i C
df = pd.DataFrame(rng.rand(1000, 3), columns=['A', 'B', 'C'])
varijabla = 1
# Aritmeticke operacije i koristenje lokalne varijable
# Kreiranje novog stupca funkcijom df.eval()
print(df.eval('D = -A * C / (B + @varijabla)'))

# Filtriranje podataka metodom pd.eval()
print(pd.eval('df[(df.A < 0.5) & (df.B < 0.5)]'))
# Filtriranje podataka metodom df.query()
print(df.query('(A < 0.5) & (B < 0.5)'))

# Ulancane operacije
print(pd.eval('df1 < df2 <= df3 != df4'))
# Boolean izrazi and/or i bitovni operatori & i |
print(pd.eval('(df1 < 0.5) and (df2 < 0.5) | (df3 < 0.2)'))
# DataFrame funkcije transponiranja i indeksiranja
print(pd.eval('(df1.T[1] < 0.5) or (df3.iloc[1] < 0.5)'))
```

Prilikom odlučivanja koje metode za obradu podataka koristiti, potrebno je uzeti u obzir sljedeće kriterije:

- **Vrijeme izvršavanja operacije** - metode `.eval()` i `.query()` drastično povećavaju efikasnost i skraćuju vrijeme izvršavanja operacija nad velikim skupovima podataka, no ako se operacije izvršavaju nad manjim skupom podataka tada mogu biti sporije od NumPy i Pandas univerzalnih funkcija.
- **Količinu memorije koja se alocira** - NumPy i Pandas univerzalne funkcije izvršavaju operacije u koracima, privremeno alocirajući memoriju prije izvođenja svakog koraka. Korištenjem metoda `.eval()` i `.query()` se delegira izvođenje operacija u programski jezik C izbjegavajući nepotrebno alociranje memorije.

Dakle, ako se izvršava veliki broj operacija nad većim skupom podataka tada metode `.eval()` i `.query()` pružaju efikasnije rješenje. S druge strane, ako se izvršava manji broj operacija nad manjem skupu podataka tada navedene metode mogu biti sporije od NumPy i Pandas univerzalnih funkcija.

6. Studija slučaja

Python biblioteke poput biblioteka NumPy i Pandas omogućuju brzu, jednostavnu i efikasnu obradu i analizu podataka dok biblioteka Matplotlib pruža veliki broj metoda za prikaz podataka. Korištenjem navedenih Python biblioteka ćemo??? prikazati obradu, analizu i prikaz koronavirus (COVID-19) podataka. S obzirom da se koronavirus podaci svakodnevno ažuriraju potrebno je kreirati Python skriptu koja će preuzeti najnovije podatke putem aplikacijskog programskog sučelja (engl. *Application Programming Interface*, kratica: *API*) te ih automatski obraditi i prikazati na grafu.

Za preuzimanje koronavirus podataka korišten je javno dostupan COVID-19 API koji koristi podatke prikupljene od strane sveučilišta *Johns Hopkins CSSE*. [27] Navedeni API pruža pristupne točke (engl. *endpoint*) za preuzimanje unaprijed filtriranih podataka prema državi, datumu i statusu koronavirus slučaja. No s obzirom da se filtriranje obavlja interno, vraćajući samo filtrirane podatke, nije moguće kreirati kompleksnije filtriranje. Iz tog razloga će se koristiti pristupna točka */all* koja vraća sve dostupne koronavirus podatke. Zabilježeni koronavirus statusi u dobivenim podacima su: potvrđeni (engl. *confirmed*), aktivni (engl. *active*), oporavljeni (engl. *recovered*) i preminuli (engl. *deaths*). Preuzeti podaci se spremaju u datoteku uz pomoć *DictWriter* objekta iz *csv* biblioteke. Prilikom ispisivanja dobivenih podataka (spremljenih u varijablu *json_data*) su vidljivi nazivi stupaca. S obzirom da `DictWriter` zapisuje samo vrijednosti podataka iz dobivenog JSON-a, potrebno je ručno dodati nazive stupaca prosljeđujući ih kao parametar prilikom kreiranja `DictWriter` objekta. Nakon toga se pozivaju metode `.writeheader()` i `.writerows()` koje ispisuju nazive stupaca i proslijeđene podatke respektivno.

```
url = "https://api.covid19api.com/all"
fileName = "../data/covid_all_data.csv"

print("Downloading Covid data...")
response = requests.get(url)
if(response.status_code == 200):
    print("All Covid-19 data successfully downloaded!\n")
    json_data = response.json()
    with open(fileName, "w", newline="") as file:
        title = "Country,CountryCode,Province,Lat,Lon,Confirmed," +
        "Deaths,Recovered,Active,Date,CityCode,City".split(",")
        cw = csv.DictWriter(file, title, delimiter=',',
            quotechar='"', quoting=csv.QUOTE_MINIMAL)
        cw.writeheader()
        cw.writerows(json_data)
        print("All Covid-19 data successfully saved!\n")
else:
    print("Error while fetching data!\n")
```

Podaci se dohvaćaju u *JavaScript Object Notation* (kratica: *JSON*) obliku te sadrže broj koronavirus slučaja za svaki status po državama. No veće države poput Sjedinjenih Američkih Država, Australije i Kine objavljuju podatke po gradovima zbog čega jedan zapis može sadržavati koronavirus podatke određenog grada ili određene države. Uz broj koronavirus slučaja po statusu, dostupni su podaci o nazivu grada i županije (ako se radi o podacima određenog grada), ISO dvoslovna oznaka države te geografska širina i dužina.

	Country	CountryCode	Province	Lat	Lon	Confirmed	Deaths	Recovered	Active	Date	CityCode	City
0	Afghanistan	AF	NaN	33.94	67.71	0	0	0	0	2020-01-22T00:00:00Z	NaN	NaN
1	Afghanistan	AF	NaN	33.94	67.71	0	0	0	0	2020-01-23T00:00:00Z	NaN	NaN
2	Afghanistan	AF	NaN	33.94	67.71	0	0	0	0	2020-01-24T00:00:00Z	NaN	NaN
3	Afghanistan	AF	NaN	33.94	67.71	0	0	0	0	2020-01-25T00:00:00Z	NaN	NaN
4	Afghanistan	AF	NaN	33.94	67.71	0	0	0	0	2020-01-26T00:00:00Z	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...
771897	Zimbabwe	ZW	NaN	-19.02	29.15	5378	141	4105	1132	2020-08-18T00:00:00Z	NaN	NaN
771898	Zimbabwe	ZW	NaN	-19.02	29.15	5643	150	4442	1051	2020-08-19T00:00:00Z	NaN	NaN
771899	Zimbabwe	ZW	NaN	-19.02	29.15	5745	151	4525	1069	2020-08-20T00:00:00Z	NaN	NaN
771900	Zimbabwe	ZW	NaN	-19.02	29.15	5815	152	4587	1076	2020-08-21T00:00:00Z	NaN	NaN
771901	Zimbabwe	ZW	NaN	-19.02	29.15	5893	153	4629	1111	2020-08-22T00:00:00Z	NaN	NaN

771902 rows × 12 columns

Slika 14: Podaci o Koronavirus slučajevima

Prvi uvid u dobivene podatke prikazane na slici 14. prikazuje da postoji veliki broj NaN vrijednosti u stupcima "Province", "CityCode" i "City". Navedeni stupci sadrže NaN vrijednosti zbog toga što prikazane države (Afganistan i Zimbabwe) prijavljuju broj koronavirus slučajeva na nacionalnoj razini. Potrebno je pronaći koje države prijavljuju podatke na razini pokrajina te postoje li za te države ukupni podaci. Ako ne postoje tada je potrebno zbrojiti sve koronavirus slučajeve prije uklanjanja redaka koji sadrže zapise pojedinih gradova ili pokrajina. Prilikom učitavanja koronavirus podataka potrebno postaviti atribut *encoding* na "latin-1" kako bi se svi znakovi ispravno učitali s obzirom da neke države u nazivu sadrže posebne znakove. Države koje objavljuju koronavirus podatke na razini pokrajina su: Australija, Kanada, Kina, Danska, Francuska, Nizozemska, Ujedinjeno Kraljevstvo i Sjedinjene Američke Države. Ispisom ukupnog broja zapisa za države s najviše zapisa je vidljivo da Sjedinjene Američke Države imaju 713 226 zapisa od ukupno 771 902. Dakle otprilike 92.4% svih zapisa se odnosi na Sjedinjene Američke Države. Druga država je Kina sa 7062 zapisa, odnosno 0.009%, a treća Nizozemska s 3506 zapisa, odnosno samo 0.005%. Zbog velikog nesrazmjera u podacima je potrebno ostaviti isključivo zapise na državnoj razini čime se broj zapisa smanjuje za otprilike 92.5%.

```
# Ucitavanje koronavirus podataka iz CSV datoteke
out = pd.read_csv("../data/covid_all_data.csv", encoding="latin-1")
# Drzave koje objavljuju podatke po pokrajinama
out[out["Province"].notna()]["Country"].unique()

# Ispisi ukupan broj zapisa za 9 drzava s najvise zapisa
out.groupby("Country").count().sort_values("Confirmed",
    ascending=False).iloc[:9, :]["Confirmed"]
```

Prije uklanjanja zapisa na razini gradova i pokrajina potrebno je provjeriti postoje li države koje nemaju podatke na državnoj razini. Najjednostavniji način za izvršavanje takve operacije je kreiranje liste koja sadrži samo države s prijavljenim podacima na nacionalnoj razini i liste koja sadrži samo države bez prijavljenih podataka na nacionalnoj razini te primijeniti operaciju simetrične razlike nad kreiranim listama. Biblioteka NumPy sadrži metodu **`.setdiff1d()`** kojoj se prosljeđuju dvije liste na temelju kojih se primjenjuje operacija simetrične razlike (npr. `np.setdiff1d(ar1, ar2)` pronalazi i ispisuje isključivo vrijednosti koje postoje u ar1, a ne postoje u ar2). Metoda također sadrži opcionalni atribut *assume_unique* koji se može postaviti na `True` ako obje liste ne sadrže ponavljajuće vrijednosti kako bi uklonila dodatna provjera povećavajući performanse. Rezultat primjene simetrične razlike prikazuje da Australija i Kina ne sadrže podatke na državnoj razini te ih je potrebno ručno zbrojiti.

```
# Simetricna razlika liste sa i bez podataka nacionalne razine
s_ukupnim_podacima = out[out["Province"].notna()]["Country"].unique()
bez_ukupnih_podataka = out[out["Province"].isna()]["Country"].unique()
np.setdiff1d(s_ukupnim_podacima,bez_ukupnih_podataka,assume_unique=True)
```

Prije zbrajanja podataka prvo je potrebno filtrirati samo državu čiji podaci se trebaju zbrojiti. Rezultat filtriranja se tada može grupirati prema nazivu države, ISO dvoslovnoj oznaci države i datumu. Nad grupiranim podacima se zbrajaju sve vrijednosti pozivanjem funkcije `.sum()`. Rezultat filtriranja, grupiranja i zbrajanja je `DataFrame` objekt koji sadrži isključivo podatke filtrirane države na nacionalnoj razini. Nakon toga je potrebno pozvati metodu `.reset_index()` kako bi se `MultiIndex` vrijednosti prebacili u podatke stupaca. Kada su izračunati podaci nacionalne razine mogu se ukloniti svi zapisi pojedinih pokrajina i gradova. S obzirom da svi zapisi pojedinih gradova sadrže i naziv pokrajine, uklanjanjem zapisa s podacima u stupcu "Province" se obuhvaćaju sve vrijednosti koje se trebaju ukloniti. Nakon toga se metodom `.append()` dodaju izračunate vrijednosti nacionalne razine koji su nedostajali. S obzirom da izračunati podaci sadrže stare indeks vrijednosti potrebno je postaviti atribut *ignore_index* na `True` kako bi se dodanim zapisima automatski dodijelili nove indeks vrijednosti.

```
# Zbrajanje podataka [Kina - CN]
china = countries.query("CountryCode == 'CN'")
    .groupby(['Country', 'CountryCode', 'Date']).sum()
china.reset_index(inplace = True)
# Zbrajanje podataka [Australija - AU]
australia = countries.query("CountryCode == 'AU'")
    .groupby(['Country', 'CountryCode', 'Date']).sum()
australia.reset_index(inplace = True)

# Uklanjanje zapisa pojedinih pokrajina i gradova
countries = countries.query("Province == ''")
# Dodavanje podataka Kine i Australije
countries = countries.append(china, ignore_index=True)
countries = countries.append(australia, ignore_index=True)
```

Nakon što su svi podaci filtrirani potrebno je postaviti indeks na naziv države kako bi se izbjeglo nepotrebno spremanje indeks vrijednosti u CSV datoteku. Također se mogu ukloniti stupci koji su, uklanjanjem zapisa prema pokrajinama, postali neupotrebljivi. S obzirom da su nazivi država postavljeni kao indeks vrijednosti, stupac "Country" je nepotreban te se može ukloniti kako bi se izbjeglo duplo spremanje naziva države. No prije spremanja u CSV datoteku potrebno je formatirati stupac "Date" s obzirom da je svaki zapis u formatu *2020-01-22T00:00:00Z* s fiksnim vremenom *00:00:00Z*. Korištenjem vektorizirane funkcije tekstualnih podataka `.split()` moguće je podijeliti zapise prema graničniku "T" te uzeti prvu vrijednost, odnosno samo datum.

```
# Postavljanje indeksa i uklanjanje nepotrebnih stupaca
countries.index = countries["Country"]
not_needed = ["Province", "CityCode", "City", "Country", "Lat", "Lon"]
countries.drop(columns=not_needed, inplace = True)

# Uklanjanje vremena iz timestampa - svugdje postavljeno na ponoc
countries["Date"] = countries["Date"].str.split("T", expand = True)[0]

# Spremanje podataka u CSV datoteku
countries.to_csv("../data/covid_countries_data.csv", encoding="latin-1")

# Prikaz svakog 5000. retka
countries[::5000]
```

Slika 15. prikazuje svaki 5000. redak filtriranih koronavirus podataka. Nakon sređivanja preuzetih podataka preostali su stupci koji sadrže broj koronavirus slučajeva prema statusu, ISO dvoslovna oznaka države, naziv države i datum zabilježenih podataka.

Country	CountryCode	Confirmed	Deaths	Recovered	Active	Date
Afghanistan	AF	0	0	0	0	2020-01-22
Brunei Darussalam	BN	135	1	92	42	2020-04-09
Dominica	DM	18	0	18	0	2020-06-26
Holy See (Vatican City State)	VA	0	0	0	0	2020-02-11
Liberia	LR	141	16	45	80	2020-04-29
Nepal	NP	17344	39	11249	6056	2020-07-16
Saint Vincent and Grenadines	VC	0	0	0	0	2020-03-02
Tajikistan	TJ	1936	41	641	1254	2020-05-19

Slika 15: Filtriranih podaci o koronavirus slučajevima

Pozivom metode `.describe()` se prikazuje osnovna statistika koronavirus podataka prikazana na slici 16. Prvi redak se odnosi na zbroj redaka koji sadrže zapis (odnosno `.notna()` `.count()`) na kojemu je vidljivo da u među dobivenim podacima nema nepostojećih vrijednosti (zbroj redaka sa zapisima svakog stupca jednak ukupnom broju redaka; `DataFrame` objekt sadrži ukupno 43 937). Također se može vidjeti prosječni broj slučajeva po državama prema pojedinom koronavirus statusu. Prosječan broj oboljelih po državi je 42 178 dok je prosječan broj trenutno aktivnih slučajeva 17 051. Također se može vidjeti odnos prosječnog broja preminulih i oporavljenih (1764 preminulih i 23 602 oporavljenih), dakle prosječna stopa smrtnosti iznosi 6,95%. Uz to je moguće vidjeti najveći broj zabilježenih slučajeva prema koronavirus statusu (država s najvećim broj slučajeva ima ukupno 6 486 227 slučaja dok država s najvećim brojem oporavljenih osoba bilježi ukupno 3 657 701 osoba oporavljenih od koronavirusa).

U retku koji prikazuje minimalan broj slučajeva prema statusu se može uočiti anomalija u stupcu "Recovered" s obzirom da negativan broj oporavljenih nije moguć. Naredbom `countries.query("Recovered < 0")` se ispisuju svi zapisi koji imaju negativan broj oporavljenih gdje se vidi da postoji samo jedan zapis s anomalijom u kojemu je, prema dobivenim podacima, Nizozemska prijavila -1 oporavljenih na dan 11.8.2020. Detaljnijim uvidom u prijavljene podatke Nizozemske je vidljivo da ne prijavljuju podatke oporavljenih slučajeva. Naredba `countries.query("CountryCode == 'NL' and Recovered > 0")` ispisuje sve zapise u kojima je Nizozemska prijavila oporavljene slučajeve. Jedino datumi 13.7. i 14.7. imaju prijavljene oporavljene slučajeve kada je prijavljeno ukupno 99 oporavljenih osoba od koronavirusa. Ostalim danima nisu prijavljeni oporavljeni slučajevi, odnosno prijavljeno je 0 oporavljenih osoba.

	Confirmed	Deaths	Recovered	Active
count	4.393700e+04	43937.000000	4.393700e+04	4.393700e+04
mean	4.217789e+04	1764.385734	2.360201e+04	1.705081e+04
std	2.790050e+05	9989.443182	1.494222e+05	1.549371e+05
min	0.000000e+00	0.000000	-1.000000e+00	0.000000e+00
25%	8.000000e+00	0.000000	0.000000e+00	1.000000e+00
50%	6.230000e+02	10.000000	1.400000e+02	1.810000e+02
75%	7.568000e+03	153.000000	3.025000e+03	2.351000e+03
max	6.486227e+06	191766.000000	3.657701e+06	6.396489e+06

Slika 16: Statistika koronavirus podataka

Na slici 16. se također mogu vidjeti kvantili koji pružaju detaljniji uvid u distribuciju podataka. Četvrtina država nema oporavljenih niti preminulih slučajeva te imaju najviše jedan aktivan slučaj i 8 oboljelih slučajeva. Medijan pokazuje da pola država ima najviše 10 preminulih i 140 oporavljenih slučajeva s najviše 623 oboljelih i 181 aktivnih slučajeva. Na temelju statističkih podataka se može zaključiti da četvrtina država ima veliki broj zaraženih dok ostale tri četvrtine država ima najviše 7 568 oboljelih.

Spajanje podataka iz više izvora omogućuje otkrivanje novih informacija na temelju postojećih podataka. Zbog toga će se preuzeti i obraditi dodatni podaci o državama kako bi se mogli usporediti s koronavirus podacima. Podaci o državama preuzeti su s Kaggle web stranice te sadrže veliki broj informacija o državama poput gustoće naseljenosti, iznosa BDP-a, regije i postotka pismenosti stanovništva. [28]

Preuzeti podaci sadrže razmak na kraju naziva svake države (npr. 'Croatia ') koje je potrebno ukloniti funkcijom `.strip()`. S obzirom da se podaci o državama trebaju spojiti s koronavirus podacima potrebno je odrediti stupac prema kojemu će se podaci spajati. Kako preuzeti podaci o državama ne sadrže ISO dvoslovne oznake, podatke je potrebno spojiti prema nazivima država. Zbog toga se prije spajanja mora osigurati da sve države imaju isti naziv u oba `DataFrame` objekta. Primjenjivanjem operacije simetrične razlike metodom `.setdiff1d()` nad popisom naziva država iz oba `DataFrame` objekta se ispisuju sve države koje imaju drugačije nazive. Kako bi se podaci o državama ispravno spojili s koronavirus podacima potrebno je preimenovati sve države s drugačijim nazivima za koje postoje zapisi o koronavirus podacima. Od ukupno 186 država za koje postoje koronavirus podaci, 26 država imaju drugačiji naziv u oba `DataFrame` objekta. S obzirom da će se u studiji slučaja spajati više `DataFrame` objekata s koronavirus podacima države će se preimenovati u ostalim `DataFrame` objektima prema nazivima iz koronavirus podataka. U programskom kôdu ispod se nalaze četiri primjera preimenovanja naziva država.

```
# Ucitavanje podataka iz CSV datoteka
covid = pd.read_csv("../data/covid_countries_data.csv",
    encoding="latin-1")
countries = pd.read_csv("../data/countries_of_the_world.csv")

# Uklanjanje razmaka u nazivima drzava
countries["Country"] = countries["Country"].str.strip()

# Simetricna razlika
# Ispis drzava koje postoje samo u koronavirus podacima
print(np.setdiff1d(covid["Country"].unique(),
    countries["Country"].unique(), assume_unique=True))

# Simetricna razlika
# Ispis drzava koje postoje samo u dodatnim podacima o drzavama
print(); print(np.setdiff1d(countries["Country"].unique(),
    covid["Country"].unique(), assume_unique=True))

# Preimenovanje naziva drzava
countries.loc[countries.Country == "Gambia, The", "Country"] = "Gambia"
countries.loc[countries.Country == "Laos", "Country"] = "Lao PDR"
countries.loc[countries.Country == "Vietnam", "Country"] = "Viet Nam"
countries.loc[countries.Country == "Burma", "Country"] = "Myanmar"
```

Nakon što su preimenovane sve države s različitim nazivima, ponovnim primjenjivanjem simetrične razlike se ispisuje pet država. Ispisane države imaju koronavirus podatke, ali ne sadrže dodatne podatke o državama u drugom `DataFrame` objektu stoga ih je potrebno ručno dodati. Države kojima nedostaju podaci su: South Sudan, Palestinska Autonomna Područja, Vatican City, Crna Gora i Kosovo. Programski kôd za dodavanje podataka o državama je prikazan ispod na primjeru dodavanja podataka o državi South Sudan.

```
# Kreiranje Python rjecnika sa svim podacima o drzavi
south_sudan =  {'Country':'South Sudan','Region':'SUB-SAHARAN AFRICA',
                'Population':11193725,'Area (sq. mi.)':235890,
                'Pop. Density (per sq. mi.)':'47',
                'Coastline (coast/area ratio)':'0',
                'Net migration':'-174,2',
                'Infant mortality (per 1000 births)':'58.6',
                'GDP ($ per capita)':1119.65,'Literacy (%)':'27',
                'Climate':'2',
                'Arable (%)':'86', 'Crops (%)':'4', 'Other (%)':'10',
                'Birthrate':'34,70', 'Deathrate':'10,30',
                'Agriculture':'72,10', 'Industry':'7,40',
                'Service':'20,50'}

# Dodavanje novog zapisa u DataFrame s metodom .append()
countries = countries.append(south_sudan, ignore_index=True)
```

Naredbom `countries.isna().sum()` se ispisuje ukupan broj nepostojećih vrijednosti po svakom stupcu. Ispisane vrijednosti pokazuju da podaci o pismenosti stanovništva, klimi i ekonomskim sektorima sadrže 18, 23 i 15 nepostojećih vrijednosti respektivno dok ostali stupci sadrže ispod 10 nepostojećih vrijednosti. Nakon toga je potrebno ispisati sve države koje ne sadrže podatke u stupcima s nepostojećim vrijednostima te ih ručno dodati.

```
# Ispis drzava koji ne sadrze podatke o pismenosti stanovnistva
print(countries[countries["Literacy (%)"].isna()]["Country"].unique())

# Dodavanje nepostojecih vrijednosti
countries.loc[countries.Country == "Serbia", "Birthrate"] = "9,2"
countries.loc[countries.Country == "Serbia", "Deathrate"] = "14,8"
countries.loc[countries.Country == "Monaco", "Agriculture"] = "0"
countries.loc[countries.Country == "Monaco", "Industry"] = "14"
countries.loc[countries.Country == "Monaco", "Service"] = "86"
```

Nakon obrade podataka o državama postavlja se stupac "Country" kao indeks `DataFrame` objekta te se uklanjaju svi nepotrebni stupci. Vrijedi napomenuti da metoda `.to_csv()` sadrži atribut *index* koji određuje hoće li se indeks vrijednosti spremiti u CSV datoteku. Postavljanjem atributa *index* na `False` se indeks vrijednosti ne zapisuju u CSV datoteku. Uz uklanjanje nepotrebnih stupaca se površina država pretvara iz milja u kilometre s obzirom da dodatni podaci o populaciji sadrže gustoću naseljenosti po kvadratnom kilometru.

```
# Postavljanje indeks vrijednosti i pretvaranje milja u kilometre
countries.index = countries["Country"]
countries["Area (sq. km.)"] = countries["Area (sq. mi.)"] * 2.59

# Uklanjanje nepotrebnih stupaca
countries.drop(columns=["Pop. Density (per sq. mi.)", "Population",
                        "Area (sq. mi.)", "Country",  "Climate",
                        "Phones (per 1000)"], inplace = True)

countries.to_csv("../data/countries_data.csv")
```

Prilikom spajanja dodatnih podataka o državama s koronavirus podacima će se ukloniti sve države koje ne sadrže koronavirus podatke. Zbog toga je prilikom popunjavanja nepostojećih vrijednosti potrebno ručno dodati vrijednosti samo onim državama koje sadrže koronavirus podatke. Na slici 17. su vidljive NaN vrijednosti za državu American Samoa koje nije potrebno ručno dodati s obzirom da ne postoje koronavirus podaci za navedenu državu.

Country	Region	Coastline (coast/area ratio)	Net migration	Infant mortality (per 1000 births)	GDP ($ per capita)	Literacy (%)	Arable (%)	Crops (%)	Other (%)	Birthrate	Deathrate	Agriculture	Industry	Service
Afghanistan	ASIA (EX. NEAR EAST)	0,00	23,06	163,07	700.00	36,0	12,13	0,22	87,65	46,6	20,34	0,38	0,24	0,38
Albania	EASTERN EUROPE	1,26	-4,93	21,52	4500.00	86,5	21,09	4,42	74,49	15,11	5,22	0,232	0,188	0,579
Algeria	NORTHERN AFRICA	0,04	-0,39	31	6000.00	70,0	3,22	0,25	96,53	17,14	4,61	0,101	0,6	0,298
American Samoa	OCEANIA	58,29	-20,71	9,27	8000.00	97,0	10	15	75	22,46	3,27	NaN	NaN	NaN
Andorra	WESTERN EUROPE	0,00	6,6	4,05	19000.00	100,0	2,22	0	97,78	8,71	6,25	11,9	33,6	54,5
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
South Sudan	SUB-SAHARAN AFRICA	0	-174,2	58.6	1119.65	27	86	4	10	34,70	10,30	72,10	7,40	20,50
Palestinian Territory	NEAR EAST	9.76	-10.56	15.8	3198.87	96,3	17,60	19,09	63,31	30,9	3,5	0,07	0,28	0,65
Holy See (Vatican City State)	WESTERN EUROPE	0	0	0	21198.00	100	0	0	100	0	0	0	0	100
Montenegro	EASTERN EUROPE	3.50	-480	2.3	8844.24	98,8	12,9	1,2	85,9	11,70	10,80	7,9	17,1	75
Republic of Kosovo	EASTERN EUROPE	0	-3.72	12	4302.28	96,8	27,4	1,9	70,70	17,09	7,0	11,9	17,70	70,40

232 rows × 15 columns

Slika 17: Dodatnih podaci o državama

No podaci o državama sadrže samo ukupnu populaciju po državi za 2017. godinu. Za ažurnije podatke o populaciji se mogu koristiti službeni zapisi od *United Nations* koji bilježe aktualne podatke o populaciji stanovništva koji buduća predviđanja populacije po državama prema raznim kriterijima. Za studiju slučaja potrebni su samo aktualni podaci o stanovništvu preuzeti iz podgrupe *Total Population*. [29] Podaci su ažurirani 2019. godine dok su podaci za 2020. godinu zapisani u obliku predviđanja. S obzirom da su podaci ažurirani u proteklih godinu dana, koristit će se podaci predviđeni prema *Momentum* tipu predviđanja, odnosno predviđanje populacije prema trenutnoj stopi rasta ili pada populacije. Nakon što su filtrirani samo potrebni podaci za 2020. godinu se uklanjaju svi nepotrebni stupci. Ispisom podataka se može primijetiti kako je ukupna populacija zapisana u tisućama stanovnika. Kako bi se uskladili s koronavirus podacima potrebno ih je pomnožiti s 1000 te zaokružiti na cijeli broj izbjegavajući decimalni broj populacije. Dobiveni podaci o gustoći naseljenosti se mogu zaokružiti na dvije decimale zbog jednostavnosti prikaza.

```
# Momentum - predvidanje populacije prema trenutnoj stopi rasta/pada
data = data.query('(Variant == "Momentum") & (Time == "2020")')

# Skaliranje i zaokruzivanje podataka
data["PopTotal"] = np.round(data["PopTotal"] * 1000)
data["PopDensity"] = np.round(data["PopDensity"], 2)
```

Podaci o stanovništvu također ne sadrže ISO dvoslovne oznake država zbog čega je potrebno primijeniti operaciju simetrične razlike nad nazivima država oba skupa podataka. Proces mijenjanja naziva država i dopunjavanja nepostojećih vrijednosti je opisan prilikom opisivanja obrade podataka o državama. Kada su svi podaci filtrirani i dopunjeni spremaju se u CSV datoteku funkcijom `.to_csv()`.

Location	PopMale	PopFemale	PopTotal	PopDensity
Afghanistan	1.997626e+07	1.895208e+07	3.892834e+07	59.63
Africa	6.698783e+08	6.707198e+08	1.340598e+09	45.22
Albania	1.464714e+06	1.413086e+06	2.877800e+06	105.03
Algeria	2.215381e+07	2.169724e+07	4.385104e+07	18.41
American Samoa	NaN	NaN	5.519700e+04	275.98
...	...	...	...	...
World	3.929974e+09	3.864825e+09	7.794799e+09	59.92
Yemen	1.502498e+07	1.480098e+07	2.982597e+07	56.49
Zambia	9.103006e+06	9.280950e+06	1.838396e+07	24.73
Zimbabwe	7.092010e+06	7.770917e+06	1.486293e+07	38.42
Republic of Kosovo	9.326550e+05	8.782060e+05	1.810861e+06	166.33

286 rows × 4 columns

Slika 18: Dodatni podaci o stanovništvu

Kada su svi podaci obrađeni i spremljeni potrebno ih je vizualizirati. Biblioteka GeoPandas omogućuje jednostavnu pohranu i obradu geoprostornih podataka te je kompatibilna s bibliotekom Matplotlib. Geoprostorni podaci država su preuzeti s *Natural Earth* stranice u *shapefile* formatu. [30] *Shapefile* je datotečni format za jednostavnu pohranu podataka o geografskim položajima i oblicima. Stupac *geometry* sadrži zemljopisne značajke pohranjene kao skup točaka, linija i/ili mnogokuta. Preuzeta *shapefile* datoteka sadrži ukupno 95 stupaca s raznim podacima o državama, no za potrebe studije slučaja su ključni naziv države, odnosno stupac *ADMIN*, ISO dvoslovna oznaka te stupac *geometry*. Uz to je potrebno preimenovati nazive ukupno 25 država primjenom operacije simetrične razlike i preimenovanjem država kako bi se uskladili s koronavirus podacima. Također je potrebno dodati ISO dvoslovne oznake za Francusku i Norvešku za koje nisu postojali zapisi. Nakon toga se obrađeni GeoDataFrame objekt sprema u *shapefile* datoteku funkcijom `.to_file()`.

```
# Ucitavanje podataka i preimenovanje stupaca
shapefile = '../data/geometry_raw/ne_50m_admin_0_countries.shp'
gdf = gpd.read_file(shapefile)[['ADMIN', 'ISO_A2', 'geometry']]
gdf.columns = ['Country', 'ISO_A2', 'geometry']

# Dodavanje nepostojecih ISO kodova
gdf.loc[gdf.Country == "France", "ISO_A2"] = "FR"
gdf.loc[gdf.Country == "Norway", "ISO_A2"] = "NO"

# Preimenovanje drzava i spremanje u CSV datoteku
gdf.loc[gdf.Country == "Vietnam", "Country"] = "Viet Nam"
gdf.loc[gdf.Country == "The Bahamas", "Country"] = "Bahamas"
gdf.to_file("../data/geometry_clean/geometry.shp")
```

	Country	ISO_A2	geometry
0	Zimbabwe	ZW	POLYGON ((31.28789 -22.40205, 31.19727 -22.344...
1	Zambia	ZM	POLYGON ((30.39609 -15.64307, 30.25068 -15.643...
2	Yemen	YE	MULTIPOLYGON (((53.08564 16.64839, 52.58145 16...
3	Viet Nam	VN	MULTIPOLYGON (((104.06396 10.39082, 104.08301 ...
4	Venezuela (Bolivarian Republic)	VE	MULTIPOLYGON (((-60.82119 9.13838, -60.94141 9...
...	...	...	...
236	Albania	AL	POLYGON ((19.34238 41.86909, 19.34551 41.91885...
237	Afghanistan	AF	POLYGON ((66.52227 37.34849, 66.82773 37.37129...
238	Siachen Glacier	-99	POLYGON ((77.04863 35.10991, 77.00449 35.19634...
239	Antarctica	AQ	MULTIPOLYGON (((-45.71777 -60.52090, -45.49971...
240	Sint Maarten	SX	POLYGON ((-63.12305 18.06895, -63.01118 18.068...

241 rows × 3 columns

Slika 19: Geografskih podaci država

Kreirane su skripte za obradu i spremanje koronavirus podataka, podataka o državama te najnovijih podataka o populaciji stanovništva. Ukoliko se u međuvremenu dobiju noviji podaci s istom ili sličnom strukturom moguće je obraditi nove podatke uz vrlo malo izmjena u kreiranim skriptama. Na taj način je moguće uvijek imati aktualne podatke bez potrebe za konstantnim ažuriranjem skripti za obradu podataka. Također, svi podaci se nakon obrade spremaju u posebnu datoteku kako bi se izbjeglo nepotrebno pozivanje skripte svaki puta kada su potrebni obrađeni podaci. No prije vizualizacije podataka je potrebno sve obrađene `DataFrame` objekte spojiti u jedan `DataFrame` objekt kako bi se pojednostavio prikaz podataka. S obzirom da će se u ovoj studiji slučaja prikazivati geografski podaci putem stupca *geometry* preporuča se koristiti GeoDataFrame objekt koji omogućuje jednostavnu vizualizaciju geografskih podataka.

Prilikom spajanja `DataFrame` objekata se koristi `.merge()` metoda objašnjena u poglavlju 5.2. Prilikom spajanja `DataFrame` objekata je potrebno obratiti pažnju na nazive stupaca prema kojima se spaja te ih po potrebi ručno navesti (ako se stupci prema kojima se spajaju objekti drugačije nazivaju). Nakon spajanja obrađenih podataka se brišu svi nepotrebni stupci te se gotovi podaci spremaju u datoteku.

```
# Spajanje koronavirus podataka s ostalim podacima
temp = pd.merge(covid, pop, left_on="Country",
    right_on="Location", how="left")
final = pd.merge(temp, countries, how="left")

# Uklanjanje nepotrebnih stupaca i spremanje u CSV datoteku
final.drop(columns=["Location"], inplace = True)
final.to_csv("../data/final.csv", index = False)

# Prikazi svaki 5000ti redak
final[::5000]
```

Country	CountryCode	Confirmed	Deaths	Recovered	Active	Date	PopMale	PopFemale	PopTotal	PopDensity	...	Literacy (%)	Arable (%)	Crops (%)	Other (%)
Afghanistan	AF	0	0	0	0	2020-01-22	19976265.0	18952076.0	38928341.0	59.63	...	36,0	12,13	0,22	87,65
Brunei Darussalam	BN	135	1	92	42	2020-04-09	226987.0	210496.0	437483.0	83.01	...	93,9	0,57	0,76	98,67
Dominica	DM	18	0	18	0	2020-06-26	37399.0	35991.0	73390.0	97.85	...	94,0	6,67	20	73,33
Holy See (Vatican City State)	VA	0	0	0	0	2020-02-11	536.0	265.0	801.0	1838.64	...	100	0	0	100
Liberia	LR	141	16	45	80	2020-04-29	2542539.0	2515138.0	5057677.0	52.51	...	57,5	3,95	2,28	93,77
Nepal	NP	17344	39	11249	6056	2020-07-16	13348435.0	15788373.0	29136808.0	203.26	...	45,2	21,68	0,64	77,68
Saint Vincent and Grenadines	VC	0	0	0	0	2020-03-02	56218.0	54729.0	110947.0	284.48	...	96,0	17,95	17,95	64,1
Tajikistan	TJ	1936	41	641	1254	2020-05-19	4805738.0	4731904.0	9537642.0	68.14	...	99,4	6,61	0,92	92,47

Slika 20: Prikaz podataka nakon obrade

Najefikasniji način spremanja geografskih podataka je u obliku *shapefile* datoteke dok se ostali podaci spremaju u CSV formatu. Spremajući sve podatke u jednu CSV datoteku se zauzima ukupno 650 MB memorijskog prostora zbog neefikasne pohrane geografskih podataka. Također, za spremanje svih podataka u jednu *shapefile* datoteku je potrebno ukupno 290 MB memorijskog prostora. Kako bi se izbjeglo neefikasno spremanje podataka potrebno je spremiti obrađene podatke u CSV datoteku te geograske podatke u *shapefile* datoteku. Na taj način geografski podaci zauzimaju ukupno 1.5 MB memorijskog prostora dok ostali obrađeni podaci zauzimaju 8.2 MB što drastično smanjuje količinu zauzetog memorijskog prostora. No zbog toga je prije vizualizacije podataka potrebno učitati obje datoteke te ih spojiti kako bi se podaci ispravno prikazali. Iako se moraju provoditi dodatne operacije, učitavanje i spajanje dva `DataFrame` objekta s ukupno 10 MB je puno brže od učitavanja jedne datoteke od 290 MB.

Kreirana funkcija `.load_shapefile()` učitava oba `DataFrame` objekta te ih spaja koristeći `.merge()` metodu. Nakon spajanja se uklanjaju nepotrebni stupci te se `DataFrame` objekt pretvara u GeoDataFrame zbog jednostavnosti prikaza geografskih podataka. Kreirani GeoDataFrame objekt se vraća kao rezultat metode kako bi se mogao koristiti za vizualizaciju podataka.

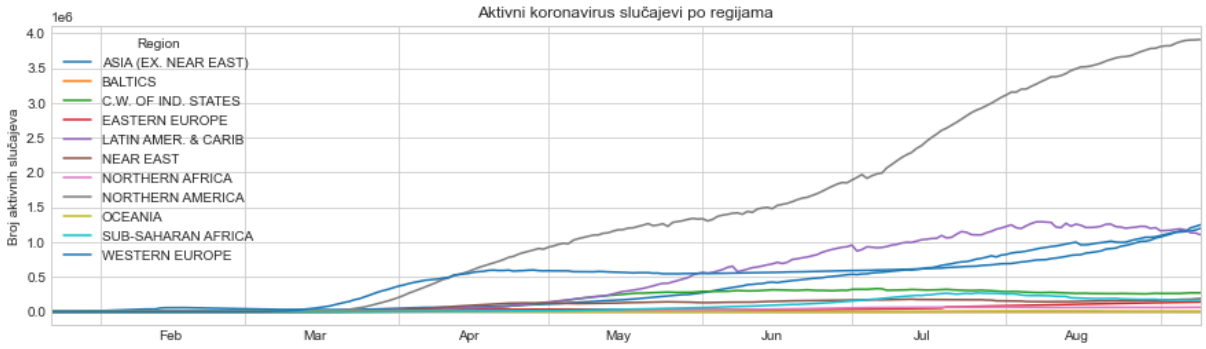
```
def load_shapefile():
    # Ucitavanje podataka iz datoteka
    covid = pd.read_csv("../data/final.csv", index_col=[0])
    gdf = gpd.read_file('../data/geometry_clean/geometry.shp')

    # Spajanje finalnih podataka s geografskim podacima
    merge = pd.merge(covid, gdf, left_on=covid.index,
        right_on="Country", how="left")

    # Uklanjanje nepotrebnih stupaca
    merge.index = merge["Country"]
    merge.drop(columns=["CountryCode", "Country"], inplace = True)

    # Pretvaranje DataFrame objekta u GeoDataFrame
    merge_gdf = gpd.GeoDataFrame(merge)
    return merge_gdf

merge_gdf = load_shapefile()
```

Redak kreiranog GeoDataFrame objekta se odnosi na jedan dan zabilježenih koronavirus slučajeva po državi te sadrži ukupno 26 podatka (stupca) za svaki zapis. Nakon obrade podataka se dobiveni podaci mogu koristiti u različite svrhe poput vizualizacije putem Python biblioteka, daljnje obrade u obliku strojnog učenja te javne objave podataka putem *API*-a. Kao rezultat studije slučaja će se dobiveni podaci vizualizirati koristeći metodu `FuncAnimation()` biblioteke Matplotlib detaljnije opisana u poglavlju 6.1.

6.1. Rezultati

Prikaz obrađenih podataka je relativno jednostavan s obzirom da nema potrebe za dodatnim manipulacijama nad podacima. Nakon što se podaci učitaju iz datoteke potrebno je odrediti što se točno želi prikazati te koji je najjednostavniji način za dobivanje željenih rezultata. Kako bi se prikazao ukupan broj aktivnih slučajeva po regijama podaci se prvo grupiraju prema regiji i datumu. Nakon toga se metodom `.pivot_table()` kreira `DataFrame` objekt isključivo s podacima za prikaz nad kojim se tada može pozvati metoda `.plot()` za jednostavnu vizualizaciju podataka.

Slika 21. prikazuje ukupan broj aktivnih koronavirus slučajeva po regijama dok se programski kôd za prikaz slike nalazi u prilogu. Na grafu je vidljivo da Sjeverna Amerika ima najveći broj aktivnih koronavirus slučajeva od travnja 2020. godine. Sjeverna Amerika trenutno broji blizu 4 000 000 aktivnih koronavirus slučajeva dok Azija, Zapadna Europa te Latinska Amerika imaju oko 1 250 000 trenutno aktivnih slučajeva. Ostale regije su u svakom trenutku imale manje od 400 000 aktivnih koronavirus slučajeva. Na grafu je također vidljivo da se broj aktivnih slučajeva u Latinskoj Americi zadnjih mjesec dana smanjuje dok se u istom vremenskom periodu broj aktivnih slučajeva povećava u Aziji i Zapadnoj Europi. Također, Zapadna Europa je u travnju 2020. godine imala drastično povećanje aktivnih slučajeva nakon čega je uveden veliki broj koronavirus mjera. Provedene mjere su uspješno smanjile dotadašnje povećanje broja aktivnih slučajeva koji je stagnirao do sredine 8. mjeseca kada se ponovno počinje povećavati. S druge strane, provedene koronavirus mjere u Sjevernoj Americi nisu uspjele zaustaviti porast aktivnih slučajeva koji se konstantno povećava od početka pandemije.

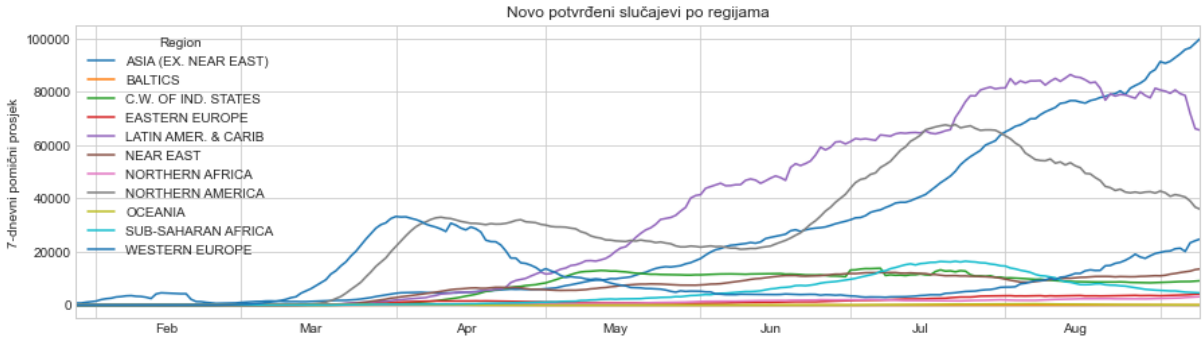


Slika 21: Aktivni koronavirus slučajevi po regijama

Prikaz aktivnih koronavirus slučajeva po danima vizualizira aktualno stanje prema regijama. Iz takvog grafa je moguće vidjeti kako pandemija napreduje te koje regije imaju najveće opterećenje zdravstvenog sustava. Uz to je moguće procijeniti koje regije su najuspješnije kod zaustavljanja porasta broja aktivnih slučajeva te koje regije su pretrpjele najveći broj zaraženih osoba. No kako bi se mogao detaljnije procijeniti utjecaj koronavirus mjera te potencijalno vizualizirati valove infekcije potrebno je prikazati broj novo zaraženih osoba.

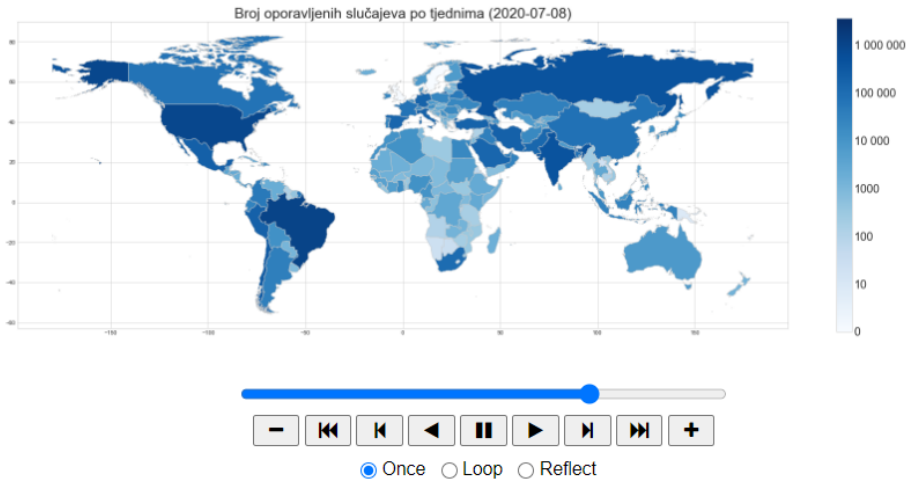
S obzirom da trenutni podaci prikazuju samo ukupan broj zaraženih osoba, prije vizualizacije je potrebno izračunati dnevni broj novo oboljelih osoba. Podatke je potrebno grupirati po regiji te putem metode `.apply()` izračunati razliku trenutnog ukupnog broja oboljelih od jučerašnjeg ukupnog broja oboljelih NumPy metodom `.diff()`. No navedena metoda ne izračunava dnevni broj novo zaraženih osoba za prvi zabilježeni dan po regiji. Podatke za prvi dan je moguće popuniti metodom `.fillna()` kako bi se izbjegle NaN vrijednosti na mjesta gdje metoda `.diff()` nije izračunala dnevni broj novo zaraženih osoba. Zbog velikih razlika u broju novo zaraženih po danima koristit će se 7-dnevni pomični prosjek kako bi se izravnale krivulje grafa. Pomični prosjek se također računa za svaku regiju zasebno zbog čega je potrebno prethodno grupirati podatke prema regiji te izračunati prosjek uz pomoć metode `.apply()`. Pomični prosjek se izračunava metodom `.rolling()` kojoj se može proslijediti parametar *window* za određivanje broja podataka koji se uzimaju za izračun prosjeka. Nakon toga se metodom `.pivot_table()` grupiraju samo podaci bitni za prikaz te se prikazuju metodom `.plot()`.

Slika 22. prikazuje broj novo potvrđenih koronavirus slučajeva po regijama. U odnosu na sliku 21. se može uočiti kako je pandemija pogađala regije u nekoliko valova. Sredinom ožujka 2020. godine se dogodio prvi veliki skok u broju zaraženih u Zapadnoj Europi te Sjevernoj Americi nakon čega se broj novo zaraženih počinje smanjivati u obje regije sredinom travnja. Zapadna Europa je uspjela drastično smanjiti broj novo zaraženih do sredine srpnja 2020. godine kada se pojavljuje novi val zaraze ponovno povećavajući broj zaraženih osoba. S druge strane, Sjeverna Amerika je smanjila broj novo zaraženih s 35 000 na 20 000 u razdoblju od travnja do lipnja kada se broj novo zaraženih počinje drastično povećavati. Sredinom srpnja se broj novo zaraženih u Sjevernoj Americi počeo smanjivati označavajući uspješnu borbu protiv koronavirusa. No kako se broj aktivnih slučajeva povećava dok se broj novo zaraženih osoba smanjuje može se zaključiti da se povećalo vrijeme potrebno za oporavak od koronavirusa što može označavati mutaciju i ojačavanje virusa. Na grafu je također vidljiv drastičan porast novo zaraženih u Latinskoj Americi od početka svibnja do kraja srpnja te veliki pad broja novo zaraženih osoba početkom rujna označavajući pozitivan smjer kretanja prema pobjedi koronavirusa. No s druge strane, drastičan i konstantan porast broja novo zaraženih osoba u Aziji je vrlo zabrinjavajući. Najveći porast broja novo zaraženih osoba Azijske regije je u Indiji s ukupnom populacijom od 1.38 milijardi ljudi. Velika gustoća naseljenosti i velik broj ljudi su opasni uvjeti za širenje koronavirusa te je nužno hitno poduzeti sve potrebne mjere kako bi se spriječilo daljnje širenje virusa u jednoj od najmnogoljudnijih država svijeta.



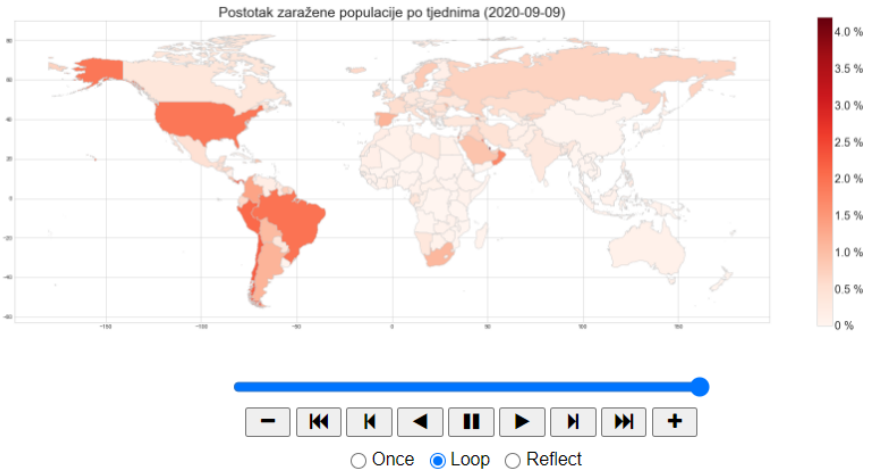
Slika 22: Novo potvrđeni slučajevi po regijama

Koronavirus podatke je moguće prikazati po državama na grafu s geografskim oblicima korištenjem GeoDataFrame objekta s geometrijskim podacima svih zemalja pohranjenih u stupcu *geometry*. Pozivanjem metode `.plot()` nad takvim GeoDataFrame objektom se automatski prikazuje graf s geografskim oblicima. Također je moguće kreirati interaktivne grafove korištenjem metode `FuncAnimation()` koja kreira veći broj grafova s istim oblicima i različitim podacima. Za prikaz interaktivnog grafa je kreirana `create_animation` metoda kojoj se prosljeđuje GeoDataFrame objekt uz ostale parametre za prilagodbu grafa. Metoda kreira prilagođenu traku u boji, figuru i osi te kreira interaktivni graf s podacima na tjednoj bazi. Slika 23. prikazuje stanje broja oporavljenih slučajeva 8.7.2020. godine. Na slici se može vidjeti anomalija u podacima za Ujedinjeno Kraljevstvo koje je na navedeni datum prijavilo 0 oporavljenih slučajeva.



Slika 23: Interaktivni prikaz oporavljenih slučajeva po tjednima

Slika 24. prikazuje postotak populacije zaražene koronavirusom po državama 9.9.2020. godine. Na slici je vidljivo da je velik postotak populacije Sjevernih Američkih Država, Čilea i Brazila zaraženo koronavirusom. Najveći postotak zaražene populacije koronavirusom ima Qatar (ukupno 4.2% populacije) koji je slabije vidljiv na grafu zbog male površine države.



Slika 24: Interaktivni prikaz postotka zaražene populacije po tjednima

7. Zaključak

Koristeći biblioteke NumPy i Pandas, programski jezik Python je jedan od najefikasnijih i najjednostavnijih programskih jezika za dohvaćanje, obradu i analizu podataka. Upravo iz tog razloga se sve više koristi u nastajućim područjima poput strojnog učenja i umjetne inteligencije te postaje jedan od najpopularnijih programskih jezika. Python također sadrži razne okvire za izradu web stranica poput okvira *Django* i *Flask* koji, uz Python biblioteke, omogućuju kreiranje iznimno fleksibilnih programa, skripti i web aplikacija obuhvaćajući sva područja od dohvaćanja i obrađivanja podataka do kreiranja web aplikacija i vizualizacije rezultata.

Biblioteka NumPy sprema podatke u liste podataka statičnog tipa dok se podaci spremljeni u Python listu pohranjuju u dinamičnom obliku. Pohranjivanjem podataka statičnog tipa se žrtvuje fleksibilnost za drastično povećanje efikasnosti izvršavanja operacije. Iz tog razloga je korištenje biblioteke NumPy postalo ključno za efikasnu obradu velikih količina podataka u Python-u. No biblioteka NumPy se fokusira isključivo na pohranu homogenih podataka s metodama za obradu numeričkih tipova podataka. Biblioteka Pandas nadograđuje biblioteku NumPy s objektima i metodama za pohranu i obradu heterogenih tipova podataka. Objekt `DataFrame` se koristi za jednostavno pohranjivanje i prikazivanje dvodimenzionalnih podataka u obliku tablice. Nad kreiranim `DataFrame` objektom se tada mogu izvršavati ugrađene Pandas metode i postojeće NumPy metode. S obzirom da je Pandas najpoznatija Python biblioteka za obradu podataka, razne biblioteke za vizualizaciju podataka podržavaju Pandas objekte te je moguće vrlo jednostavno vizualizirati podatke pohranjene u `DataFrame` objekt. NumPy i Pandas biblioteke spajaju efikasnost programskog jezika C s jednostavnom Python sintaksom pružajući intuitivno i jednostavno programiranje s izvrsnim performansama. Zbog efikasnosti, jednostavnosti te velikog broja metoda za manipulaciju raznih tipova i oblika podataka, biblioteke NumPy i Pandas su najvažnije Python biblioteke za obradu podataka. Detaljna dokumentacija i ugrađene Python naredbe za prikaz dokumentacije poput naredbe `help()` te znaka "?" uvelike olakšavaju upoznavanje s bibliotekama te pružaju brz i jednostavan način istraživanja svih mogućnosti Python biblioteka. Uz to, biblioteka Pandas sadrži razne metode za obradu nepostojećih, tekstualnih i višedimenzionalnih podataka te olakšava vizualizaciju ispisujući podatke u obliku uređene tablice.

Studija slučaja odrađena u ovom diplomskom radu prikazuje realni primjer obrade, analize i vizualizacije koronavirus podataka. Istraživanjem se vizualno prikazalo kako se broj novo zaraženih osoba povećavao u valovima te koja je uspješnost pojedinih regija u smanjivanju broja novo zaraženih osoba. Također se može zaključiti da se vremenski period oporavka od koronavirusa povećao u Sjevernoj Americi. Povećanje vremena potrebnog za oporavak i drastično povećanje broja novo zaraženih osoba u Indiji predstavljaju glavne točke na koje se treba fokusirati u daljnjoj borbi protiv koronavirusa. Provedena studija slučaja predstavlja jednostavan primjer korištenja Pandas paketa za obradu podataka. Za daljnje istraživanje je potrebno iskoristiti metode NumPy i Pandas biblioteka detaljno opisane u ovom radu kako bi se dobili noviji i kvalitetniji podaci otvarajući vrata za nova otkrića. Dobiveni podaci se tada mogu koristiti za daljnju obradu putem strojnog učenja, pružanje podataka javnosti putem javnog *API*-a te za napredniju vizualizaciju podataka na web ili desktop aplikacijama.

Popis literature

[1] G. van Rossum, „Foreword for 'Programming Python' (1st ed.)", 1996. adresa: https://www.python.org/doc/essays/foreword/, [Pristupano 12.07.2020.]

[2] TIOBE, „TIOBE Programming Community Index Definition", 2020. adresa: https://www.tiobe.com/tiobe-index/programming-languages-definition/, [Pristupano 25.07.2020.]

[3] ——, „TIOBE Index for July 2020", 2020. adresa: https://www.tiobe.com/tiobe-index/, [Pristupano 25.07.2020.]

[4] D. I. Beautiful, „Most Popular Programming Languages 1965 - 2019", 2019. adresa: https://www.youtube.com/watch?v=0g847HVwRSI, [Pristupano 12.07.2020.]

[5] J. VanderPlas, *Python Data Science Handbook. Essential Tools for Working with Data*. O'Reilly Media, 2016, ISBN: 9781491912058.

[6] N. Thieme, „R generation", 2018. adresa: https://rss.onlinelibrary.wiley.com/doi/full/10.1111/j.1740-9713.2018.01169.x, [Pristupano 21.07.2020.]

[7] RStudio, „RStudio Connect 1.8.4", 2020. adresa: https://rstudio.com/, [Pristupano 22.07.2020.]

[8] ——, „RMarkdown", 2020. adresa: https://rmarkdown.rstudio.com/, [Pristupano 22.07.2020.]

[9] ——, „Shiny", 2020. adresa: https://shiny.rstudio.com/, [Pristupano 22.07.2020.]

[10] R. Cotton, „Python vs. R for Data Science: What's the Difference?", 2020. adresa: https://www.datacamp.com/community/blog/when-to-use-python-or-r, [Pristupano 25.07.2020.]

[11] D. Kisler, „Loops in R and Python: Who is faster?", 2020. adresa: https://datascienceplus.com/loops-in-r-and-python-who-is-faster/, [Pristupano 25.07.2020.]

[12] J. Korstanje, „Is Python faster than R?", 2020. adresa: https://towardsdatascience.com/is-python-faster-than-r-db06c5be5ce8, [Pristupano 25.07.2020.]

[13] I. Dataquest Labs, „Python vs R: Head to Head Data Analysis", 2020. adresa: https://www.dataquest.io/blog/python-vs-r/, [Pristupano 25.07.2020.]

[14] Datacamp, „When Should I Use Python vs. R?", 2020. adresa: https://s3.amazonaws.com/assets.datacamp.com/email/other/Python+vs+R.pdf, [Pristupano 25.07.2020.]

[15] D.-D. Science, „Python vs R for Data Science: And the winner is..”, 2018. adresa: https://medium.com/@data_driven/python-vs-r-for-data-science-and-the-winner-is-3ebb1a968197, [Pristupano 22.07.2020.]

[16] P. Jupyter, „Jupyter Notebook”, 2020. adresa: https://jupyter.org/, [Pristupano 12.07.2020.]

[17] M. Cone, „Markdown Guide - Basic Syntax”, 2020. adresa: https://www.markdownguide.org/basic-syntax/, [Pristupano 12.07.2020.]

[18] T. E. Oliphant, „Python for Scientific Computing”, 2007. adresa: https://web.archive.org/web/20131014035918/http://www.vision.ime.usp.br/~thsant/pool/oliphant-python_scientific.pdf, [Pristupano 25.07.2020.]

[19] D. W. A.J. Henley, *Learn Data Analysis with Python: Lessons in Coding*. Apress, 2017, ISBN: 9781484234853.

[20] J. VanderPlas, „Matplotlib and the Future of Visualization in Python”, 2013. adresa: https://jakevdp.github.io/blog/2013/03/23/matplotlib-and-the-future-of-visualization-in-python/, [Pristupano 30.07.2020.]

[21] SciPy, „Scientific computing tools for Python”, 2020. adresa: https://scipy.org/about.html, [Pristupano 30.07.2020.]

[22] T. M. development team, „pylab_examples example code: annotation_demo2.py”, 2017. adresa: https://matplotlib.org/examples/pylab_examples/annotation_demo2.html, [Pristupano 04.08.2020.]

[23] J. VanderPlas, „Customizing Ticks”, 2017. adresa: https://jakevdp.github.io/PythonDataScienceHan 04.10-customizing-ticks.html, [Pristupano 04.08.2020.]

[24] W. McKinney, „pandas: a Foundational Python Library for Data Analysis and Statistics”, 2011. adresa: https://www.dlr.de/sc/Portaldata/15/Resources/dokumente/pyhpc2011/submissions/pyhpc2011_submission_9.pdf, [Pristupano 28.07.2020.]

[25] kanoki, „How to use Regex in Pandas”, 2019. adresa: https://kanoki.org/2019/11/12/how-to-use-regex-in-pandas/, [Pristupano 16.08.2020.]

[26] Pandas, „Time series / date functionality”, 2020. adresa: https://pandas.pydata.org/pandas-docs/stable/user_guide/timeseries.html#offset-aliases, [Pristupano 16.08.2020.]

[27] K. Redelinghuys, „COVID-19 API”, 2020. adresa: https://covid19api.com/, [Pristupano 22.08.2020.]

[28] F. Lasso, „Countries of the World”, 2018. adresa: https://www.kaggle.com/fernandol/countries-of-the-world, [Pristupano 12.05.2020.]

[29] U. Nations, „World Population Prospects 2019”, 2019. adresa: https://population.un.org/wpp/Download/Standard/CSV/, [Pristupano 17.05.2020.]

[30] N. Earth, „1:50m Cultural Vectors”, 2020. adresa: https://www.naturalearthdata.com/downloads/50m-cultural-vectors/, [Pristupano 21.06.2020.]

Popis slika

Popis tablica

Prilozi

Programski kôd za prikaz slike 3.

```
# 1. Celija [Lijevi graf]
import matplotlib.pyplot as plt
import numpy as np

# Create figure
fig = plt.figure() # Kreiraj figuru
# Podesi razmak izmedu podgrafova
fig.subplots_adjust(hspace=0.5, wspace=0.5)

for i in range(1, 7):
    plt.subplot(2, 3, i) # Nacrtaj podgraf
    # Dodaj kombinaciju brojeva za dohvacanje podgrafa
    plt.text(0.5, 0.5, str((2,3,i)), fontsize=18, ha='center')

# 2. Celija [Desni graf]
# Kreiraj figuru i listu Axes objekata koji dijele X i Y osi
# Automatski se kreiraju grafovi!
fig, ax = plt.subplots(2, 3, sharex='col', sharey='row')

# Dodaj index vrijednosti na svakom grafu
for i in range(2):
    for j in range(3):
        ax[i, j].text(0.5, 0.5, str((i, j)),fontsize=18, ha='center')
```

Programski kôd za prikaz slike 4.

```
import matplotlib.pyplot as plt
import numpy as np

# 1. Celija [Lijevi graf]
# Kreiraj praznu "mrezu" grafova s 2 retka i 3 stupca
grid = plt.GridSpec(2, 3, wspace=0.4, hspace=0.3)

# Kreiraj podgraf na poziciji (0, 2)
plt.subplot(grid[0, 2])
# Kreiraj podgraf u prvom retku na svim stupcima do stupca s indeksom 2
plt.subplot(grid[0, :2])
# Kreiraj podgraf na poziciji (1, 0)
plt.subplot(grid[1, 0])
# Kreiraj podgraf u drugom retku na svim stupcima osim prvog
plt.subplot(grid[1, 1:])

# 2. Celija [Desni graf]
# Kreiraj podatke normalnom raspodjelom
mean = [3, 1]
cov = [[5, 2], [2, 5]]
x, y = np.random.multivariate_normal(mean, cov, 10000).T

# Kreiraj figuru i praznu "mrezu"
fig = plt.figure(figsize=(6, 6))
grid = plt.GridSpec(4, 4, hspace=0.2, wspace=0.2)

# Kreiraj podgrafove
main_ax = fig.add_subplot(grid[1:, :-1])
y_hist = fig.add_subplot(grid[0, :-1], xticklabels=[])
x_hist = fig.add_subplot(grid[1:, -1], yticklabels=[])

# Kreiraj podgraf s tockama na glavnoj osi
main_ax.plot(x, y, 'o', color="#91a891", markersize=3, alpha=0.2)

# Kreiraj dva histograma na sporednim osima
x_hist.hist(x, 50, orientation='horizontal', color="#bbd5e8")
x_hist.invert_yaxis()
y_hist.hist(y, 50, orientation='vertical', color='#bbd5e8')
y_hist.invert_xaxis()
```

Programski kôd za prikaz slike 6.

```
import numpy as np
import matplotlib.pyplot as plt
from datetime import datetime
# Trenutna sekunda za seed - drugaciji graf prilikom svakog izvrsavanja
rng = np.random.RandomState(datetime.now().second)
plt.figure(figsize=(15, 5)) # Velicina figure
# Postavljanje nasumicnih brojeva, boja i velicina tocaka
x = rng.randn(100)
y = rng.randn(100)
colors = rng.rand(100)
sizes = 400 * rng.rand(100)

# Ispis grafa funkcijom .plot() s dodatnim atributima
plt.subplot(1, 2, 1)
plt.title('plt.plot()')
plt.plot(x, y, 'p', markersize=10, markerfacecolor='white',
         markeredgecolor='grey', markeredgewidth=2)

# Ispis grafa funkcijom .scatter()
# Alpha atribut predstavlja razinu prozirnosti (trasparency/opacity)
plt.subplot(1, 2, 2)
plt.title('plt.scatter()')
plt.scatter(x, y, c=colors, s=sizes, alpha=0.3, cmap='viridis')
plt.colorbar() # Prikaz trake u boji

# Kreiranje 'nepostojecih' tocaka s label elementima
# Zbog praznih lista se nece ispisati u grafu
for area in [100, 200, 300]: # Zeljene velicine tocaka
    plt.scatter([],[],c='k',alpha=0.3,s=area,label=str(area)+' km$^2$')

# Funkcija .legend() ce ispisati sve elemente s 'label' atributom
plt.legend(scatterpoints=1,frameon=False,labelspacing=1,title='Legenda')
```

Programski kôd za prikaz slike 7.

```
import numpy as np
import matplotlib.pyplot as plt

# Kreiraj 2D podatke
x = np.linspace(0, 10, 100)
y = np.linspace(0, 10, 100)
plt.figure(figsize=(15, 5)) # Velicina figure

# Kreiraj 2D podatke od 1D liste na sljedeci nacin:
# Smatraj x listu kao redak te ga kopiraj y.length puta i spremi u X
# Smatraj y listu kao stupac te ga kopiraj x.length puta i spremi u Y
X, Y = np.meshgrid(x, y)
# Izracunaj trecu dimenziju Z
Z = np.sin(Y) ** 2 + np.cos(X) ** 2

# Kreiraj contour graf ispunjen bojom s 2D listama X, Y, Z
plt.subplot(1, 2, 1)
plt.title('plt.contourf()')
plt.contourf(X, Y, Z, 20, cmap='twilight')
plt.colorbar()

# Kreiraj contour graf linija i polutransparentnu sliku contour grafa
plt.subplot(1, 2, 2)
plt.title('plt.contour() & plt.imshow()')
# Kreiraj contour graf samo s linijama od podataka iz 2D lista X, Y, Z
# Pozitivne vrijednosti su ravne linije, a negativne iscrtkane
contours = plt.contour(X, Y, Z, 3, colors='black')
plt.clabel(contours, inline=True, fontsize=8)
# Kreiraj polutransparentnu sliku contour grafa
plt.imshow(Z, extent=[0, 5, 0, 5], origin='lower',
          cmap='RdGy', alpha=0.5)
plt.colorbar()
```

Programski kôd za prikaz slike 8.

```
import numpy as np
import matplotlib.pyplot as plt

# Kreiraj podatke
x1 = np.random.normal(-5, 2, 500)
x2 = np.random.normal(-10, 2, 500)
x3 = np.random.normal(2, 3, 500)
plt.figure(figsize=(15, 5)) # Velicina figure

# Kreiraj tri histograma s prilagodenom legendom
plt.subplot(1, 2, 1)
plt.title('plt.hist()')
# Rjecnik atributa koji su isti za sve histograme
kwargs = dict(histtype='stepfilled', alpha=0.3, bins=30)
# Kreiraj histograme
plt.hist(x1, **kwargs, label='Histogram 1')
plt.hist(x2, **kwargs, label='Histogram 2')
plt.hist(x3, **kwargs, label='Histogram 3')
plt.legend(loc='upper right', ncol=1, fancybox=True,
           framealpha=0.6, shadow=True, borderpad=1)

# Kreiraj podatke za 2D histogram Gauss-ovom distribucijom
mean = [3, 1]
cov = [[5, 2], [2, 5]]
x, y = np.random.multivariate_normal(mean, cov, 100000).T

# Kreiraj 2D histogram s mrezom heksagona i prilagodenom trakom u boji
plt.subplot(1, 2, 2)
plt.title('plt.hexbin()')
# Kreiraj 2D histogram s mrezom heksagona
plt.hexbin(x, y, gridsize=40, cmap=plt.cm.get_cmap('Reds', 10))
# Kreiraj 2D histogram s mrezom kvadrata
#plt.hist2d(x, y, bins=40, cmap=plt.cm.get_cmap('Blues', 10))
cb = plt.colorbar(label='Count in bin', extend='both')
```

Programski kôd za prikaz slike 9.

```
import matplotlib as mpl
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')
%matplotlib inline
import numpy as np
import pandas as pd
from datetime import datetime

# https://raw.githubusercontent.com/jakevdp/data-CDCbirths/master/births.csv
births = pd.read_csv('data/births.csv')

# Izracunaj kvantile i sigmu
quartiles = np.percentile(births['births'], [25, 50, 75])
mu, sig = quartiles[1], 0.74 * (quartiles[2] - quartiles[0])
# Pocisti podatke - obradeno u Pandas poglavlju
births = births.query('(births > @mu - 5 * @sig) & ' +
    '(births < @mu + 5 * @sig)')
births['day'] = births['day'].astype(int)
births.index = pd.to_datetime(10000 * births.year + 100 * births.month
    + births.day, format='%Y%m%d')

# Grupiraj podatke po mjesecu i danu
births_by_date = births.pivot_table('births',
    [births.index.month, births.index.day])
births_by_date.index = [datetime(2020, month, day)
    for (month, day) in births_by_date.index]
fig, axes = plt.subplots(figsize=(12, 4)) # Kreiraj figuru
# Postavi osi na prethodno kreirane Axes objekte i ispisi graf
births_by_date.plot(ax = axes);


# Dodavanje anotacija
axes.annotate("New Year's Day", xy=('2020-1-1', 4100), xycoords='data',
    xytext=(50, -30), textcoords='offset points',
    arrowprops=dict(arrowstyle="->",connectionstyle="arc3,rad=-0.2"))
axes.annotate("Independence Day",xy=('2020-7-4',4250), xycoords='data',
    bbox=dict(boxstyle="round", fc="none", ec="gray"),xytext=(10, -40),
    textcoords='offset points', ha='center',
    arrowprops=dict(arrowstyle="->"))
axes.annotate('Labor Day', xy=('2020-9-4', 4850), xycoords='data',
    ha='center', xytext=(0, -20), textcoords='offset points')
```

```
axes.annotate('', xy=('2020-9-1', 4850), xytext=('2020-9-7', 4850),
    xycoords='data', textcoords='data',
    arrowprops={'arrowstyle': '|-|,widthA=0.2,widthB=0.2', })
axes.annotate('Halloween', xy=('2020-10-31', 4600),  xycoords='data',
    xytext=(-80, -40), textcoords='offset points',
    arrowprops=dict(arrowstyle="fancy",fc="0.6", ec="none",
    connectionstyle="angle3,angleA=0,angleB=-90"))
axes.annotate('Thanksgiving', xy=('2020-11-25', 4500),
    xycoords='data',xytext=(-120, -60), textcoords='offset points',
    bbox=dict(boxstyle="round4,pad=.5", fc="0.9"),
    arrowprops=dict(arrowstyle="->",
    connectionstyle="angle,angleA=0,angleB=80,rad=20"))
axes.annotate('Christmas', xy=('2020-12-25', 3850),  xycoords='data',
    xytext=(-30, 0), textcoords='offset points',size=13, ha='right',
    va="center",bbox=dict(boxstyle="round", alpha=0.1),
    arrowprops=dict(arrowstyle="wedge,tail_width=0.5", alpha=0.1))

# Postavi naslov grafa i naziv Y osi
axes.set(title='USA births by day of year (1969-1988)',
         ylabel='Average daily births')

# Formatiranje tick-ova
# Postavi sporedne lokatore iskljucivo na sredinu svakog mjeseca
axes.xaxis.set_minor_locator(mpl.dates.MonthLocator(bymonthday=15))
# Ukloni glavne tick-ove
axes.xaxis.set_major_formatter(plt.NullFormatter())
# Za oznacavanje sporednih tickova koristi naziv mjeseca
axes.xaxis.set_minor_formatter(mpl.dates.DateFormatter('%h'))
```

Programski kôd za prikaz slike 10.

```
# %matplotlib notebook za interaktivne grafove
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d

# Kreiranje podataka za prikaz
x = np.linspace(-6, 6, 30)
y = np.linspace(-6, 6, 30)

fig = plt.figure(figsize=(15, 5)) # Kreiraj figuru
# Kreiraj jedan 3D podgraf (objekt Axes3D)
ax = fig.add_subplot(1, 2, 1, projection='3d')

# Pretvori 1D listu podataka u 2D
X, Y = np.meshgrid(x, y)
Z = np.sin(np.sqrt(X ** 2 + Y ** 2))

# Nacrtaj contour podgraf
ax.contour3D(x, y, Z, 50, cmap='binary')
ax.set_title('.contour3D()')
ax.set(xlabel='x', ylabel='y', zlabel='z')

# Nacrtaj povrsinu funkcijom .plot_surface
ax = fig.add_subplot(1, 2, 2, projection='3d')
ax.set_title('.plot_surface()')
ax.plot_surface(X, Y, Z, cmap='viridis')
ax.set(xlabel='x', ylabel='y', zlabel='z')

# Samo u slucaju interaktivnih grafova (%matplotlib notebook)
#plt.show()
```

Programski kôd za prikaz slike 12.

```
import pandas as pd
import numpy as np
import random

# 2D MultiIndex redaka s nazivima
index = pd.MultiIndex.from_product([["Italija", "Austrija"],
            [2019, 2020]], names=['Drzava', 'Godina'])

# 2D MultiIndex stupaca s nazivima
columns = pd.MultiIndex.from_product([['Petar', 'Marin', 'Ivana'],
            ['Turist', 'Posao']], names=['Ime', 'Putovanje'])

# Kreiraj nasumicne brojeve
data = np.random.randint(0, 50, size=(4, 6))

# Kreiraj DataFrame s 2D indeksom na recima i stupcima (4D podaci)
df = pd.DataFrame(data, index=index, columns=columns)

# Prikazi podatke DataFrame-a
df
```

Programski kôd za prikaz slike 13.

```
import pandas as pd
import numpy as np
import random

# Kreiraj DataFrame s nasumicnim godinama izmedu 1 i 50
# randint() ne ukljucuje zadnji broj -> [1-51)
df = pd.DataFrame(np.random.randint(1, 51, size=(201, 1)),
    columns=["Godina"])

# Dodaj stupac Putna_klasa s nasumicnim klasama izmedu 1 i 3
df["Putna_klasa"] = np.random.randint(1, 4, size=(201, 1))

# Radnom visina - Godina * (random 0-1) * 10
df["Visina"] = df["Godina"] * np.random.rand(201) * 10
# Ako je visina veca od 200 - postavi na 200
df["Visina"] = np.round(np.where(df["Visina"] > 200, 200,
    df["Visina"]), 2)
# Ako je visina manja od 30 - postavi 30
df["Visina"] = np.round(np.where(df["Visina"] < 30, 30,
    df["Visina"]), 2)

# Random IQ izmedu 80 i 120
df["IQ"] = np.random.randint(80, 121, size=(201, 1))

# Random spol izmedu 0 i 1 (1 - M, 0 - Z)
df["Spol"] = np.random.randint(0, 2, size=(201, 1))
df["Spol"] = np.where(df["Spol"] == 1, "M", "Z")

# Prikazi podatke DataFrame-a
df
```

Prikaz svih biblioteka korištenih za izradu studije slučaja.

```
# Numpy
import numpy as np
# Metoda za testiranje negativne beskonacnosti
from numpy import isneginf

# Pandas
import pandas as pd

# GeoPandas za pohranu geografskih podataka
import geopandas as gpd

# Matplotlib
import matplotlib as mpl
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from mpl_toolkits.axes_grid1 import make_axes_locatable

# Zahtjevi za API
import requests

# CSV za pohranu kompleksnijih podataka dobivenih iz API-a
import csv

# Prikaz animacijskih grafova u web pregledniku
from IPython.display import HTML
```

Programski kôd za prikaz slike 21.

```
plt.style.use('seaborn-whitegrid')

# Ucitaj obradene podatke
data = pd.read_csv("../data/final.csv", encoding="latin-1")

# Grupiraj podatke prema regiji
region = data.groupby(['Region', 'Date']).sum()

# Izracunaj gustocu naseljenosti regije
region["PopDensity"] = region["PopTotal"] / region["Area (sq. km.)"]
region.reset_index(inplace = True)

# Grupiraj podatke koji se prikazuju
table = region.pivot_table(index='Date', columns='Region',
    values='Active')

# Kreiraj figure i pretvori indeks u datetime objekt
fig, axes = plt.subplots(figsize=(15, 4))
table.plot(ax = axes);
table.index = pd.to_datetime(table.index)
axes.set(title='Aktivni koronavirus slučajevi po regijama',
    ylabel='Broj aktivnih slučajeva')

# Formatiraj tickove
axes.xaxis.set_minor_locator(mpl.dates.MonthLocator(bymonthday=15))
axes.xaxis.set_major_formatter(plt.NullFormatter())
axes.xaxis.set_minor_formatter(mpl.dates.DateFormatter('%h'))
```

Programski kôd za prikaz slike 22.

```
plt.style.use('seaborn-whitegrid')

# Ucitaj obradene podatke
data = pd.read_csv("../data/final.csv", encoding="latin-1")

# Grupiraj podatke prema regiji
region = data.groupby(['Region', 'Date']).sum()

# Izracunaj gustocu naseljenosti regije
region["PopDensity"] = region["PopTotal"] / region["Area (sq. km.)"]
region.reset_index(inplace = True)

# Izracunaj broj novozarazenih po danima
region["DailyConfirmed"] = region.groupby("Region")
    .apply(lambda x: x["Confirmed"].diff()).values
region["DailyConfirmed"].fillna(region["Confirmed"], inplace = True)

# Izracunaj 7-dnevni pomicni prosjek novozarazenih
region["weekAvgConfirmed"] = region.groupby("Region")
    .apply(lambda x : x.iloc[:,12].rolling(window=7).mean()).values

# Grupiraj podatke koji se prikazuju
avg = region.pivot_table(index='Date', columns='Region',
    values='weekAvgConfirmed')

# Kreiraj figure i pretvori indeks u datetime objekt
fig, axes = plt.subplots(figsize=(15, 4))
avg.plot(ax = axes);
avg.index = pd.to_datetime(avg.index)
axes.set(title='Novo potvrđeni slučajevi po regijama',
    ylabel='7-dnevni pomični prosjek')

# Formatiraj tickove
axes.xaxis.set_minor_locator(mpl.dates.MonthLocator(bymonthday=15))
axes.xaxis.set_major_formatter(plt.NullFormatter())
axes.xaxis.set_minor_formatter(mpl.dates.DateFormatter('%h'))
```

Funkcija za kreiranje animacije na temelju proslijeđenih parametara.

```
def create_animation(merge_gdf, vmax, cmap, title,
        column, yticklabels=None):
    # Inicijalizacija figure i osi
    fig, ax = plt.subplots(1, figsize=(30, 10))
    ax.axis('off')

    # Kreiraj listu sa svakim sedmim danom za prikaz
    dates = merge_gdf["Date"].unique()
    week_days = dates[::7]

    # Kreiranje fiksne trake u boji
    def init():
        sm = plt.cm.ScalarMappable(cmap=cmap,
            norm=plt.Normalize(vmin=0, vmax=vmax))
        cbar = fig.colorbar(sm)
        if yticklabels is not None:
            cbar.ax.set_yticklabels(yticklabels)
        cbar.ax.tick_params(labelsize=20)
        return fig

    # Prikaz jednog grafa
    def plot_graph(date):
        ax.clear()
        gdf = merge_gdf[merge_gdf["Date"] == date]
        graph_title = title + ' ({0})'.format(date)
        ax.set_title(graph_title,
            fontdict={'fontsize': '25', 'fontweight' : '3'})
        gdf.plot(column=column, cmap=cmap, linewidth=0.8,
            ax=ax, edgecolor='0.8', vmin=0, vmax=vmax)

    # Matplotlib funkcija za kreiranje Timespan grafa
    animation = FuncAnimation(fig, plot_graph, week_days,
        blit=False, interval=200, init_func=init)s

    plt.close()
    return animation
```