Analiza i vizualizacija dnevničkih zapisa u Pythonu

Vujasinović, Hrvoje

Undergraduate thesis / Završni rad

2020

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Organization and Informatics / Sveučilište u Zagrebu, Fakultet organizacije i informatike**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:211:554860

Rights / Prava: Attribution-NonCommercial 3.0 Unported/Imenovanje-Nekomercijalno 3.0

Download date / Datum preuzimanja: **2024-03-29**

Repository / Repozitorij:

Faculty of Organization and Informatics - Digital Repository

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Hrvoje Vujasinović

ANALIZA I VIZUALIZACIJA DNEVNIČKIH ZAPISA U PYTHONU

ZAVRŠNI RAD

Varaždin, 2020.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Hrvoje Vujasinović

Matični broj: 46252/17–R

Studij: Informacijski sustavi

ANALIZA I VIZUALIZACIJA DNEVNIČKIH ZAPISA U PYTHONU

ZAVRŠNI RAD

Mentor:

Dr. sc. Miran Zlatović

Varaždin, rujan 2020.

Hrvoje Vujasinović

Izjava o izvornosti

Izjavljujem da je moj završni/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/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Tema ovog završnog rada su dnevnički zapisi, njihova svrha i uloga na osobnim računalima i web poslužiteljima te analiza i vizualizacija podataka dobivenih iz dnevničkih zapisa. U radu ću najprije objasniti što su dnevnički zapisi i čemu služe, a zatim ću obraditi temu web poslužitelja. Nakon toga bit će opisana organizacija i struktura dnevničkih zapisa na osobnim računalima i to na Linux platformama za osobna računala, a za Windows platforme obradit ću Windows Event Viewer. Nakon toga obradit ću dnevničke zapise na web poslužiteljima i nekoliko formata dnevničkih zapisa koji se koriste na web poslužiteljima. U praktičnom dijelu rada izradit ću Python aplikaciju s grafičkim sučeljem pomoću modula Tkinter za parsiranje jednog od formata dnevničkih zapisa. Aplikacija će omogućavati parsiranje i pretraživanje podataka, ekstrakciju podskupa podataka i vizualizaciju ekstrahiranih podataka pomoću dostupnih Python biblioteka.

Ključne riječi: logs; event logs; server logs; Pyhon; web poslužitelji; parsiranje podataka; analiza podataka; vizualizacija podataka

Sadržaj

Sadržaj iii
1. Uvod 1
2. Dnevnički zapisi i njihova svrha 2
2.1. Dnevnički zapisi na osobnim računalima 3
2.1.1. Windows Event Viewer 3
2.1.2. Dnevnički zapisi na Linux platformama 6
2.2. Dnevnički zapisi na web poslužiteljima 7
2.2.1. Web poslužitelji 8
2.2.2. Common log format 9
2.2.3. Extended common log format 10
2.2.4. Microsoft IIS log format 12
3. Uvod u Pandas i Matplotlib 14
4. Python aplikacija za analizu i vizualizaciju 22
5. Zaključak 26
Popis literature 27
Popis slika 29
Prilozi 30

1. Uvod

Živimo u doba podataka i gotovo sve u našem životu zabilježeno je u digitalnom obliku. Podaci su zabilježene činjenice o stvarnom svijetu i njihovom obradom i analizom dolazimo do novih informacija koje nam pomažu u shvaćanju svijeta oko nas. Jedan od takvih izvora podataka su dnevnički zapisi koji su tema ovog završnog rada.

Osobna računala, web poslužitelji i informacijski sustavi generiraju dnevničke zapise (eng. *Logs*) koji dokumentiraju aktivnosti sustava. Takvi se zapisi sastoje od niza kronološki poredanih poruka koje sadrže informacije o aktivnostima i operacijama unutar operacijskog sustava, neke aplikacije, web poslužitelja ili nekog drugog uređaja. Analiza podataka dobivenih iz dnevničkih zapisa omogućuje praćenje ponašanja sustava i otkrivanje problema kao što su greške u radu ili napadi na sustav. Danas postoje i brojna programska rješenja za analizu dnevničkih zapisa koja omogućuju bolju ekstrakciju podataka iz dnevničkih zapisa i tako olakšavaju pronalaženje trendova i uzoraka u podacima pomoću kojih administratori sustava mogu donositi bolje poslovne odluke ili nadzirati sigurnost sustava. (E. Zhang, 2018)

U ovom radu bit će opisana organizacija dnevničkih zapisa na osobnim računalima i web poslužiteljima te izrađena vlastita Python aplikacija za analizu i vizualizaciju jednog formata dnevničkog zapisa.

2. Dnevnički zapisi i njihova svrha

Dnevnički zapisi su datoteke automatski generirane od strane računala koje sadrže popis događaja na tom računalu u obliku strukturiranih, kronološki poredanih poruka. Većina dnevničkih zapisa spremljeni su u formatu običnog teksta što omogućuje pregled u bilo kojem programu za uređivanje teksta i osigurava malu veličinu datoteke. (Christensson, 2010)

Postoji nekoliko standardiziranih formata dnevničkih zapisa, no uglavnom podaci, struktura, vrste i format poruka u dnevničkom zapisu nisu propisani pa tako ovise o odlukama i implementaciji razvojnih programera koji rade na razvoju određene aplikacije ili sustava. Iako se implementacije i formati poruka razlikuju sve su sastavljene od nekoliko zajedničkih komponenti. Svaka poruka ima datum i vrijeme kada je nastala, razinu zapisa, na primjer informativna poruka, poruka upozorenja, poruka greške ili neka druga definirana razina, i informacije o kontekstu koje nam pružaju dodatne informacije o stanju sustava ili aplikacije i okruženju u kojem je poruka nastala. (Lee, 2019)

Prema (Lee, 2019) dnevničke zapise možemo podijeliti na nekoliko osnovnih kategorija, a to su aplikacijski dnevnički zapisi (eng. *Application log*), sustavski dnevnički zapisi (eng. *System log*) i poslužiteljski dnevnički zapisi (eng. *Server log*). Aplikacijski dnevnički zapisi služe razvojnim programerima aplikacije kako bi lakše otkrili i otklonili probleme u radu aplikacije budući da se neki problemi ne mogu otkriti sve dok se aplikacija ne nađe u produkcijskoj okolini i bude dana na korištenje krajnjim korisnicima. Sustavski dnevnički zapisi sadrže informacije o radu sustava i sustavskim procesima dok poslužiteljski dnevnički zapisi bilježe informacije o korisnicima koji ga koriste.

Poslužiteljski dnevnički zapisi su od posebne važnosti za velike organizacije čije poslovanje ovisi o njihovoj mrežnoj infrastrukturi. Analiza i praćenje poslužiteljskih dnevničkih zapisa omogućuje im da povećaju pouzdanost svojih sustava ispravljanjem grešaka, sporih upita ili netočnih odgovora poslužitelja i tako poboljšaju iskustvo krajnjeg korisnika. Isto tako praćenjem dnevničkih zapisa osiguravaju sigurnost sustava i podataka od vanjskih napadača budući da ovi zapisi vode detalje o događajima vezanim za sigurnost kao što su uspješne i neuspješne prijave. Još jedan razlog za analizu poslužiteljskih dnevničkih zapisa je praćenje ponašanja korisnika na temelju kojega mogu optimizirati sustav da bolje odgovara njihovim potrebama ili donositi bolje poslovne odluke. Također organizacije mogu na vrijeme odgovoriti na povećanje opterećenja sustava zbog rasta broja korisnika uvođenjem dodatnih kapaciteta. (*What is a Log File?* | *Sumo Logic*, 2019)

Možemo reći da je glavna svrha svih dnevničkih zapisa dati korisnicima ili administratorima informacije o ponašanju sustava ili aplikacije kroz vrijeme kako bi

pravovremeno mogli reagirati na probleme, greške i sigurnosne prijetnje u radu sustava ili mogli donositi bolje poslovne odluke. U narednim poglavljima ovog rada detaljnije će biti opisana organizacija dnevničkih zapisa na osobnim računalima i poslužiteljima.

2.1. Dnevnički zapisi na osobnim računalima

Na modernim operacijskim sustavima za osobna računala ugrađeni su mehanizmi i protokoli za snimanje događaja i vođenje dnevničkih zapisa o događajima u sustavu. Takvi zapisi daju korisnicima uvid u rad sustava i prikupljaju informacije o radu sklopovlja i programa, problemima u sustavu i sigurnosnim događajima. U ovom poglavlju bit će obrađena organizacija sustavskih dnevničkih zapisa na Windows platformi i alat Windows Event Viewer te organizacija sustavskih dnevničkih zapisa na Linux platformi.

2.1.1. Windows Event Viewer

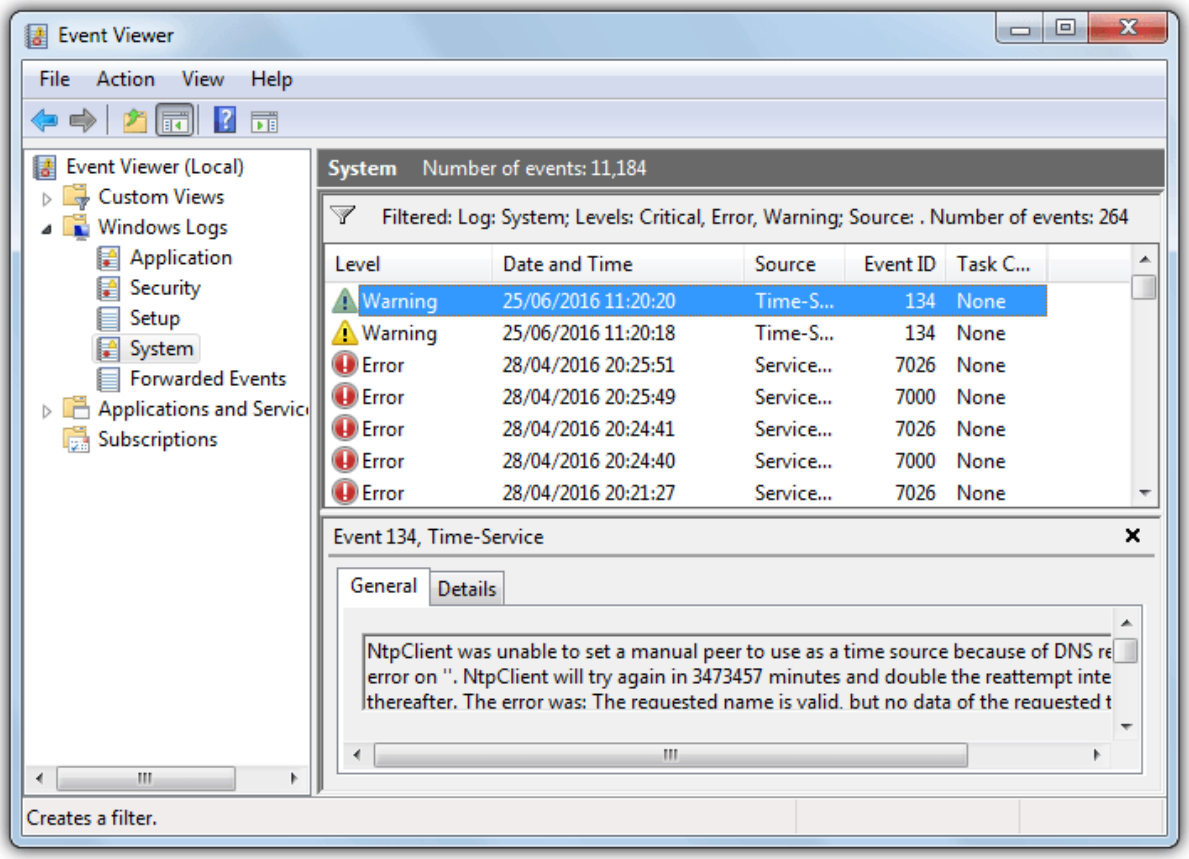
Na Windows operacijskim sustavima dnevnički zapisi spremaju se u XML formatu na lokaciju *C:\WINDOWS\system32\config*. Svaki zapis sastoji se od (*Event Log Monitoring Tool - A Tutorial*, bez dat.):

a) Datum - datum na koji je događaj nastao
b) Vrijeme - vrijeme u koje je događaj nastao
c) Korisnik - prijavljeni korisnik u trenutku nastanka događaja
d) Računalo - naziv računala na kojem se događaj dogodio
e) ID događaja - identifikator događaja koji specificira tip događaja
f) Izvor - aplikacija ili komponenta koja je uzrokovala događaj
g) Tip ili razina - tip događaja prema važnosti

Na Windows platformi tip događaja ili razina zapisa može biti informacija (eng. *Information*), upozorenje (eng. *Warning*), greška (eng. *Error*), uspješna autorizacija (eng. *Success Audit*) ili neuspješna autorizacija (eng. *Failure Audit*). Informacija je događaj koji opisuje uspješnu operaciju neke radnje u aplikaciji, upravljačkom programu ili pokrenutom servisu. Upozorenje je događaj koji nije od prevelike važnosti, ali može ukazivati na neke buduće probleme koji bi mogli nastati. Greška je događaj koji ukazuje na problem koji je nastao u radu neke aplikacije ili komponente i pri tome uzrokovao gubitak podataka ili funkcionalnosti. Uspješna i neuspješna autorizacija su događaji koji opisuju uspješan ili neuspješan sigurnosni

događaj, na primjer kada se korisnik uspješno ili neuspješno prijavi. (*Event Log Monitoring Tool - A Tutorial*, bez dat.)

Event Viewer je sustavski alat koji dolazi u sklopu Windows operacijskog sustava i omogućuje pregled dnevničkih zapisa koje taj operacijski sustav generira. Alat omogućuje i dodatne opcije kao što su pretraživanje zapisa, filtriranje zapisa, izvoz zapisa u nekom drugom formatu pogodnom za daljnju analizu ili definiranje korisničkih vrsta događaja i dnevničkih zapisa.

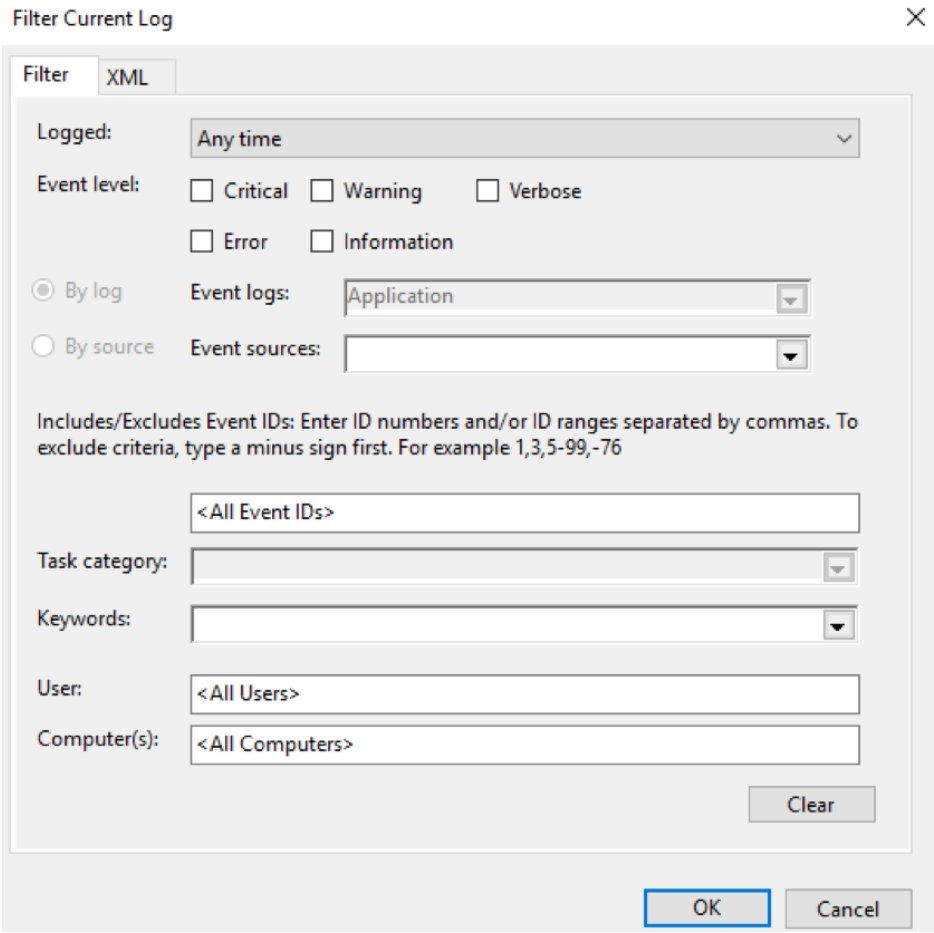


Slika 1: Izgled sučelja Windows Event Viewer-a (Izvor: raymond.cc, 2020)

S lijeve strane sučelja nalazi se okvir za navigaciju u kojem možemo izabrati dnevnički zapis koji želimo pregledati. Postoji pet zadanih kategorija Windows dnevničkih zapisa i to aplikacije (eng. *Application*), sigurnost (eng. *Security*), instalacija (eng. *Setup*), sustav (eng. *System*) i proslijeđeni događaji (eng. *Forwarded Events*). U dnevničkom zapisu aplikacije nalaze se informacije o događajima koje su generirale aplikacije instalirane na računalu. Dnevnički zapis sigurnost sadrži informacije o uspješnim i neuspješnim sigurnosnim događajima na računalu. Dnevnički zapis instalacija sadrži poruke o događajima vezanim uz instalaciju i nadogradnju operacijskog sustava. Dnevnički zapis sustav sadrži poruke koje je

generirao operacijski sustav dok dnevnički zapis proslijeđeni događaji sadrži poruke o događajima s drugih računala na mreži u slučaju da je računalo konfigurirano da prikuplja te podatke. (*Windows Logging Basics - The Ultimate Guide To Logging*, 2015)

Na desnoj strani sučelja nalazi se popis poruka u odabranom dnevničkom zapisu s detaljima o vremenu i okruženju u kojem je poruka nastala. Poruke su poredane kronološki od najstarijih na dnu prema najnovijim na vrhu. Odabirom neke od poruka možemo vidjeti opis događaja u donjem desnom okviru pod kraticom *General* i dodatne detalje pod kraticom *Details*. Alat omogućuje i napredno filtriranje dnevničkih zapisa prema različitim kriterijima kao što su vrijeme nastanka poruke, razina ili tip poruke, ključnim riječima, izvoru događaja, ID događaja, računalu ili korisniku.

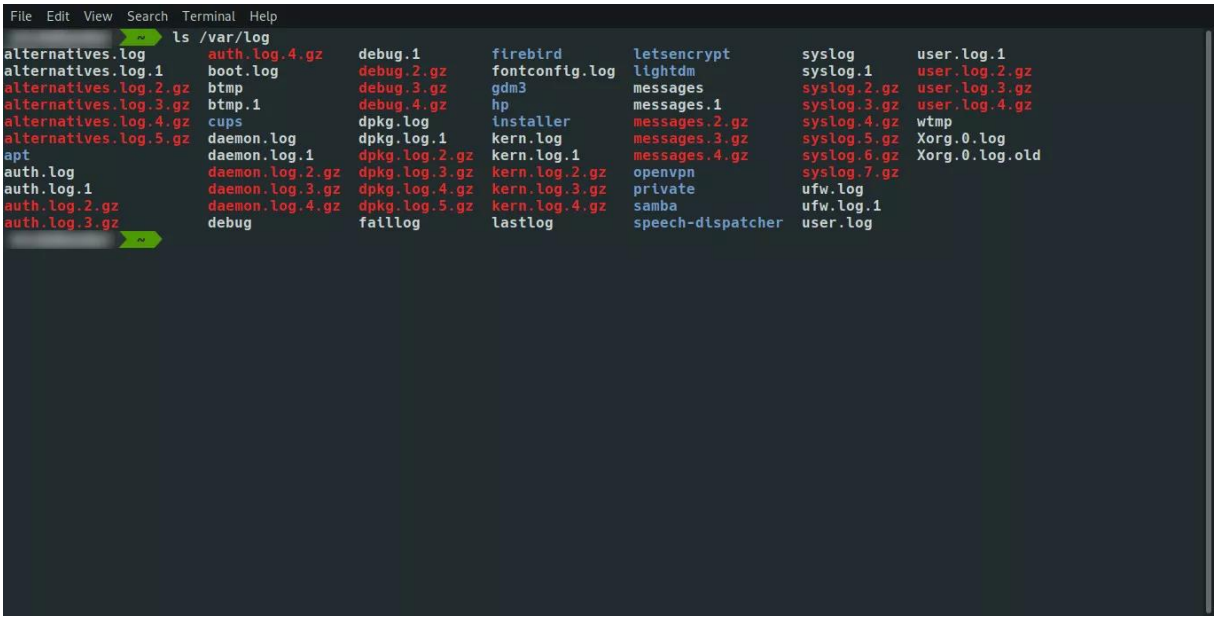


Slika 2: Opcije filtriranja u Event Viwer-u (Izvor: loggly.com, 2020)

2.1.2. Dnevnički zapisi na Linux platformama

Program koji obavlja prikupljanje i sastavljanje dnevničkih zapisa na Linux platformama je *Sysklogd* i on se sastoji od dva alata, *syslogd* za prikupljanje sustavskih poruka i *klogd* za prikupljanje poruka jezgre sustava. (*sysklogd(8): system logging utilities - Linux man page*, bez dat.)

Na Linux operacijskim sustavima dnevnički zapisi spremljeni su u tekstualnom obliku na lokaciji */var/log* ili u nekom od poddirektorija vezanim uz pojedinu aplikaciju. Neki dnevnički zapisi komprimiraju se zbog svoje veličine pa imaju *.gz* nastavak. Svaki zapis definiran je *Syslog* formatom i sastoji se od standardiziranog zaglavlja. Zaglavlje sadrži datum i vrijeme nastanka događaja, naziv aplikacije koja je generirala događaj, mjesto na računalu koje je izvor poruke i prioritet poruke. Pri tome se prioritet poruke sastoji od dva broja od kojih prvi opisuje tip procesa koji je generirao događaj i može biti u rasponu od 0 do 23 gdje 0 označava poruku jezgre sustava, a 23 poruku aplikacije pokrenute na sustavu. Drugi broj odnosi se na važnost poruke i može biti između 0 i 7 gdje 0 znači da je poruka hitna i sustav se ne može koristiti, a poruka važnosti 7 opisuje pogrešku. Format poruke može se modificirati pomoću datoteke za konfiguraciju. (*Linux Logging Basics - The Ultimate Guide To Logging*, 2015)



Slika 3: Primjer sadržaja */var/log* direktorija (Izvor: lifewire.com, 2020)

Prema (*Linux Logging Basics - The Ultimate Guide To Logging*, 2015) neki od važnijih dnevničkih zapisa na Linux platformama su:

a) */var/log/syslog* ili */var/log/messages*
b) */var/log/auth.log* ili */var/log/secure*

c) */var/log/kern.log*

d) */var/log/cron*

Točan naziv datoteke ovisi o verziji operacijskog sustava, na primjer verzije bazirane na *Debian*-u koriste */var/log/syslog* i */var/log/auth.log* dok verzije bazirane na *Red Hat*-u koriste */var/log/messages* i */var/log/secure.* Dnevnički zapis *syslog* ili *messages* sadrži opće poruke sustava i informacije vezane uz rad sustava. Dnevnički zapis *auth.log* ili *secure* sadrži informacije o događajima vezanim za sigurnost kao što su prijava korisnika i slično. *Kern.log* sadrži zapise o događajima, greškama i upozorenjima koje je generirala jezgra operacijskog sustava. *Cron* sadrži zapise o izvođenju korisnički zakazanih zadataka.

Dnevnički zapisi na Linux platformama također podržavaju rotiranje kako bi se izbjegla prevelika veličina datoteka. Alat *log rotate* obavlja ovu funkciju tako da briše dnevničke zapise starije od postavljenog vremenskog intervala i stvara nove umjesto njih. Namještanje vremenskog intervala vrši se u */etc/logrotate.conf* datoteci te se za svaku aplikaciju može postaviti vlastiti interval rotacije dnevničkog zapisa, hoće li se zapis komprimirati, slanje zapisa na e-mail prije brisanja i brojne druge opcije. (Haas, 2020)

2.2. Dnevnički zapisi na web poslužiteljima

Na strani poslužitelja dnevničke zapise možemo podijeliti na nekoliko kategorija ovisno o vrsti informacija koje se u njih zapisuju, a to su dnevnički zapis pristupa (eng. *Access log*), dnevnički zapis agenata (eng. *Agent log*), dnevnički zapis grešaka (eng. *Error log*) i dnevnički zapis upućivanja (eng. *Referrer log*). I ovi dnevnički zapisi najčešće se u tekstualnom obliku i u njima se vode zapisi o aktivnosti poslužitelja. U dnevnički zapis pristupa spremaju se podaci o korisnicima od kojih su stigli zahtjevi za sadržajem i informacije o zahtjevima za sadržaj koje je web poslužitelj zaprimio kao što su metoda kojom je zahtjev poslan, statusni kod odgovora i vrijeme potrebno za odgovor na zahtjev. Dnevnički zapis agenata sadrži zapise o vrsti i verziji web preglednika s kojeg je korisnik uputio zahtjev prema web poslužitelju. U dnevnički zapis grešaka generiraju se poruke o pogreškama odgovora web poslužitelja, na primjer u slučaju da je korisnik zatražio sadržaj koji se više ne nalazi na poslužitelju ili korisnik nema po pristup traženom sadržaju. U dnevničkom zapisu upućivanja vodi se popis stranica na kojima se korisnik nalazio neposredno prije nego što je poslao zahtjev prema poslužitelju tj. linkovi s kojih je korisnik upućen prema poslužitelju koji mogu poslužiti kako bi analizirali izvor korisničkog prometa prema poslužitelju. (Six, 2020a)

Česta praksa kod rukovanja s dnevničkim zapisima na strani poslužitelja je prikupljanje svih dnevničkih zapisa na jedno računalo radi lakše analize. Većina poslužiteljskih sustava ima

mogućnost periodičkog slanja svojih dnevničkih zapisa na udaljenu lokaciju, a postoje i brojna druga programska rješenja za ovu namjenu. Problem koji se javlja kod centraliziranog prikupljanja dnevničkih zapisa je velika količina podataka koji se generiraju, već jedan korisnik može generirati tisuće zapisa pri jednom pristupu sadržaju poslužitelja, pa je najbolja praksa dnevničke zapise filtrirati i po potrebi pretvoriti u zajednički format pogodan za analizu prije slanja na centralno računalo ili poslužitelj za daljnju obradu. (A. Grimes, 2018)

U nastavku poglavlja bit će ukratko opisan princip rada web poslužitelja i nekoliko odabranih formata poslužiteljskih dnevničkih zapisa.

2.2.1. Web poslužitelji

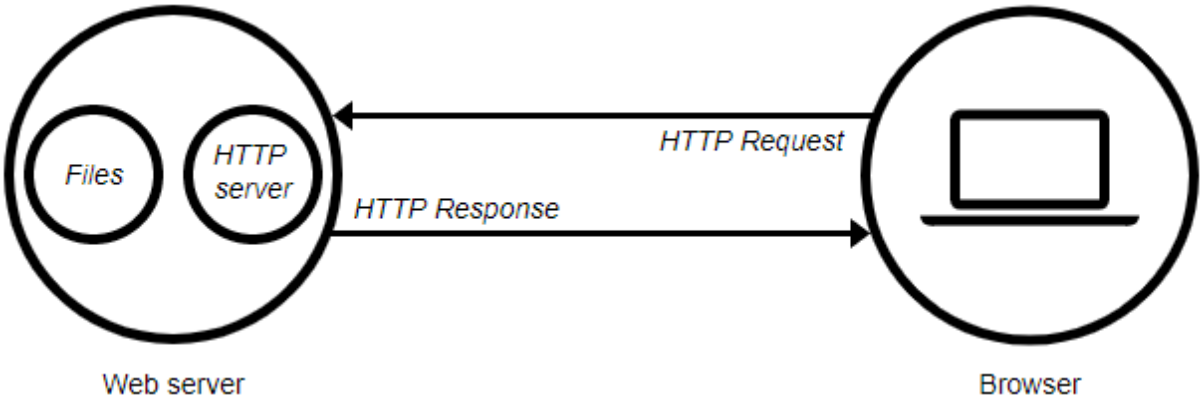
Web poslužitelji (eng. *Web server*) su računala s pripadnom programskom podrškom koja mogu zaprimati, obrađivati i odgovarati na zahtjeve klijenta pomoću HTTP ili drugih protokola. Klijent pomoću web preglednika (eng. *Browser*) šalje HTTP zahtjev koji sadrži informacije o klijentu, vrsti zahtjeva i traženom sadržaju prema web poslužitelju koji zatim prihvaća zahtjev i pronalazi traženi sadržaj te ga vraća klijentu u obliku HTTP odgovora. Ako traženi sadržaj nije pronađen ili je došlo do greške web poslužitelj vraća jedan od definiranih HTTP statusnih kodova. (*What is a web server? - Learn web development | MDN*, 2020)

```
GET / HTTP/1.1
Host: www.foi.unizg.hr
Connection: keep-alive
Cache-Control: max-age=0
Accept: text/html,application/xhtml+xml,application/
      xml;q=0.9,image/webp,*/*;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 6.2; WOW64)
      AppleWebKit/537.36 (KHTML, like Gecko)
      Chrome/33.0.1750.117 Safari/537.36
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-US,en;q=0.8
Cookie: ...|utmcmd=referral|utmcct=/search
If-Modified-Since: Mon, 03 Mar 2014 07:06:13 GMT
```

Slika 4: Primjer HTTP zahtjeva (Izvor: Kermek D., 2020)

Sadržaj koji se vraća klijentu može biti statička web stranica ili dinamički generirana web stranica. Statička web stranica ima predefiniranu strukturu i sadržaj, najčešće pomoću HTML-a i CSS-a, pa je ista za svakog klijenta kojem se šalje. Svaka stranica spremljena je na web poslužitelju kao zaseban dokument i ako ga je potrebno izmijeniti to može napraviti samo

administrator. S druge strane, dinamički generirane web stranice generiraju se u stvarnom vremenu uzimajući u obzir klijenta koji je poslao zahtjev. Na web poslužitelju spremljeni su samo predlošci i dizajn za web stranicu dok se sadržaj popunjava iz baze podataka ili nekog drugog izvora pomoću nekog od programskih jezika za programiranje na strani poslužitelja kao što su PHP, ASP i drugi. (Robben, 2019)



Slika 5: Princip rada web poslužitelja (Izvor: MDN, 2020)

2.2.2. Common log format

Common log format, poznat i pod nazivom *NCSA Common log format,* je standardizirani format dnevničkog zapisa koji se ne može uređivati ili prilagođavati. Ovaj format nije dostupan za FTP poslužitelje. Svaki zapis sastoji se od sljedećih elemenata (Six, 2020b):

a) IP adresa domaćina
b) identifikator korisnika - prema RFC 1413 protokolu (najčešće prazno)
c) korisničko ime
d) datum, vrijeme i vremenska zona
e) metoda i sadržaj koji se dohvaća
f) HTTP statusni kod
g) veličina dohvaćenog sadržaja - izražena u bajtovima

U ovom formatu uzima se lokalno vrijeme domaćina s vremenskom zonom, prazna polja označena su povlakom (-), a sama polja odvojena su razmakom. Primjer jednog ovakvog zapisa bio bi (Izvor: Izrada autora):

127.0.0.1 - lmodric [13/May/2020:16:36:46 +0200] "GET /projekt/css/style.css HTTP/1.1" 200 7701

U ovom primjeru IP adresa domaćina tj. računala s kojeg je zahtjev stigao je 127.0.0.1, identifikator korisnika je prazan, korisničko ime je lmodric, zahtjev je stigao 13. Svibnja 2020 godine u 16:36:46 sati po vremenskoj zoni GMT +2. Metoda kojom se sadržaj dohvaća je *GET*, sadržaj je datoteka pod nazivom *style.css* iz direktorija */projekt/css/* na poslužitelju. Na kraju zapisa vidimo da je poslužitelj odgovorio s HTTP statusnim kodom 200 za OK i veličinu tražene datoteke koja iznosi 7701 bajta.

2.2.3.Extended common log format

Extended common log format vrlo je sličan *Common log format*-u, ali uključuje neka dodatna polja i fleksibilan je pa tako pruža više informacija. U ovom formatu zapisa imamo mogućnost dodavanja polja koja su nam važna i uklanjanje polja s informacijama koje nam nisu bitne. Na ovaj način moguće je utjecati i na veličinu datoteke dnevničkog zapisa. Polja su kao i kod *Common log format*-a odvojena razmakom, prazna polja označena su povlakom (-), ali vrijeme i datum zapisuju se prema *UTC* vremenu. *Extended common log format* datoteka započinje popisom smjernica (eng. *Directives*) označenih ljestvama (#) za taj dnevnički zapis i to mogu biti (M. Hallam-Baker & Behlendorf, bez dat.):

a) Verzija - u obliku cijeli broj.cijeli broj
b) Polja - popis polja koja su uključena u zapis
c) Program - naziv programa koji je generirao zapis
d) Početni datum - datum i vrijeme nastanka dnevničkog zapisa
e) Završni datum - datum i vrijeme završetka dnevničkog zapisa
f) Datum - datum i vrijeme nastanka zapisa
g) Komentar - komentar s dodatnim informacijama o dnevničkom zapisu

Od ovih smjernica samo su verzija i polja obavezne. Smjernica polja opisuje od kojih elemenata će se sastojati svaki zapis, a dostupna polja su redom:

a) datum - datum kada je zapis nastao
b) vrijeme - vrijeme u koje je zapis nastao
c) vrijeme obrade - vrijeme potrebno da se zahtjev obradi
d) bajtovi - veličina prenesenog sadržaja
e) priručna memorija - informacija je li se traženi sadržaj već nalazio u priručnoj memoriji
f) ip - ip adresa i port
g) dns - naziv DNS poslužitelja

h) status - HTTP statusni kod
i) komentar - komentar koji je vraćen uz statusni kod
j) metoda - metoda korištena za dohvaćanje sadržaja
k) uri - URI link
l) uri-stem - korijenski dio URI-a
m) uri-query - dio URI-a koji sadrži upit

U ovom formatu definirani su i prefiksi za polja. Za polja od a) do e) oni nisu obavezni dok su za ostala polja obavezni. Definirani prefiksi daju informaciju o tome između koga se zahtjev razmijenio i mogu biti:

a) c - prefiks koji označava klijenta
b) s - prefiks koji označava poslužitelja
c) r - prefiks koji označava udaljenog poslužitelja
d) cs - prefiks koji označava prijenos od klijenta do poslužitelja
e) sc - prefiks koji označava prijenos od poslužitelja do klijenta
f) sr - prefiks koji označava prijenos od poslužitelja do udaljenog poslužitelja
g) rs - prefiks koji označava prijenos od udaljenog poslužitelja do poslužitelja
h) x - prefiks koji identificira specifičnu aplikaciju

Primjer jednog dnevničkog zapisa ovog formata ima sljedeći oblik: (Izvor: w3.org, 2020):

#Version: 1.0

#Date: 12-Jan-1996 00:00:00

#Fields: time cs-method cs-uri

00:34:23 GET /foo/bar.html

12:21:16 GET /foo/bar.html

12:45:52 GET /foo/bar.html

12:57:34 GET /foo/bar.html

U ovom primjeru vidljivo je da je korištena verzija *Extended common log format*-a 1.0, zapis je dodan 12. Siječnja 1996 godine u 00:00:00 sati. Definirana polja u zapisu su vrijeme, metoda kojom je klijent poslao zahtjev i uri koji je klijent poslao prema poslužitelju, a smjer

slanja opisan je prefiksom cs. Na kraju zapisa vidimo 4 linije koje pokazuju da je klijent poslao 4 *GET* zahtjeva za sadržajem */foo/bar.html* u različitim vremenima.

2.2.4.Microsoft IIS log format

Ovaj format dnevničkog zapisa vrlo je sličan *Common log format*-u, također je fiksnog oblika i ne može se prilagođavati, ali pruža neka dodatna polja i samim time više informacija. Kod ovog formata vrijednosti polja u zapisu odvojena su zarezom (,), za polja kojima je vrijednost nepoznata koristi se povlaka (-), a u slučaju da polje sadrži znak koji se ne može printati (eng. *Nonprintable character*) koristi se plus (+) kako bi se zaštitio format zapisa. Ovo se može dogoditi u slučaju napada na sustav kod kojeg napadači šalju *CR* i *LF* znakove za kraj linije koji bi mogli uništiti strukturu zapisa i tako cijelu datoteku dnevničkog zapisa učiniti beskorisnom. Za vrijeme zapisa uzima se lokalno vrijeme računala, a vrijeme obrade mjeri se u milisekundama. (*IIS Logging* | *Microsoft Docs*, 2018)

Svaki zapis može se sastojati od sljedećih polja (*Brief introduction of log file formats*, bez dat.):

a) IP adresa klijenta
b) korisničko ime - korisničko ime ako je klijent autenticiran na poslužitelju
c) datum - datum na koji je zapis nastao
d) vrijeme - vrijeme u koje je zapis nastao
e) servis i instanca - servis i instanca koju je klijent koristio
f) naziv poslužitelja - naziv poslužitelja na kojem je generiran zapis
g) IP adresa poslužitelja
h) vrijeme obrade - vrijeme koje je bilo potrebno za obradu zahtjeva
i) bajtovi poslani od klijenta - količina bajtova koju je poslao klijent
j) bajtovi poslani od poslužitelja - količina bajtova koju je poslao poslužitelj
k) HTTP statusni kod
l) Windows statusni kod - Windows definirani kodovi, 0 označava uspjeh
m) metoda zahtjeva - metoda korištena za dohvaćanje sadržaja
n) ciljani sadržaj zahtjeva - naziv datoteke sadržaja koji se dohvaća

Prema (*Brief introduction of log file formats*, bez dat.) primjer jednog zapisa ovog formata ima sljedeći oblik:

192.168.114.201,-,03/20/98,7:55:20,W3SVC2,SALES1,192.168.114.201,
4502,163,3223,200,0,GET,DeptLogo.gif

Iz ovog zapisa možemo redom pročitati vrijednosti polja. IP adresa klijenta je 192.168.114.201, korisničko ime je prazno što je označeno povlakom, zahtjev je zaprimljen 20. ožujka 1998. godine u 7:55:20 sati, servis i instanca je W3SVC2, naziv poslužitelja je SALES1, IP adresa poslužitelja je 192.168.114.201, vrijeme obrade iznosilo je 4502 milisekunde, klijent je poslao 163 bajta podataka dok je poslužitelj odgovorio s 3223 bajta podataka. HTTP statusni kod odgovora je 200 za OK, Windows statusni kod je 0 što označava uspješno obrađen zahtjev, metoda kojom je klijent poslao zahtjev je *GET*, a traženi sadržaj je datoteka naziva *DeptLogo.gif.*

3. Uvod u Pandas i Matplotlib

Pandas i Matplotlib su dobro poznate Python biblioteke za strukturiranje, obradu i vizualizaciju podataka. U ovom poglavlju bit će na kratkom primjeru pokazan rad s ovim bibliotekama i neke osnovne funkcionalnosti. Za rad i demonstraciju koristit ću Jupyter Notebook alat.

Pandas se uglavnom koristi za analizu podataka i omogućuje čitanje i manipuliranje različitim formatima podataka npr. JSON, CSV ili Excel tablice. Dva glavna objekta kojima se manipulira u radu s Pandas bibliotekom su *Series* i *Dataframe*. *Series* je struktura slična listi i sadrži jednu listu podataka uz koju može biti i lista s nazivom koja se naziva index. Na ovu strukturu možemo gledati kao jedan stupac u tablici. *Dataframe* predstavlja podatke organizirane u tablicu gdje je svaki stupac jedan *Series* objekt koji sadrži jednu vrstu podataka. *Dataframe* možemo stvoriti učitavanjem podataka iz neke vanjske datoteke, *Series* objekta, Python rječnika ili obične liste.

Kako bi započeli rad s ovim bibliotekama moramo ih prvo uključiti u program naredbom import. Set podataka koji ću koristiti u primjeru su statistike nogometnih igrača za prvih 15 igrača preuzete sa stranice *whoscored.com* (*whoscored.com/Statistics*) i prebačene u .csv format. Nakon što smo uključili biblioteku kreiramo *Dataframe* objekt pomoću *read_csv* funkcije kojoj kao argument predajemo naziv datoteke pod uvjetom da se ona nalazi u istom direktoriju kao i naš program. Možemo pogledati strukturu učitane datoteke tako da pomoću naredbe *print()* prikažemo *Dataframe* objekt, no ovo nije praktično za jako velike datoteke pa stoga koristimo naredbu *head()* koja kao argument može primiti koliko redova želimo pogledati i vraća prvih n redova u objektu.

```
import pandas as pd
import matplotlib.pyplot as plt
dataFrame = pd.read_csv('playerData.csv')
```

```
dataFrame.columns
```

```
Index(['Igrač', 'Utakmice', 'Minuta', 'Golovi', 'Asistencije', 'Žuti kartoni',
       'Crveni kartoni', 'Udarci na gol', 'Točna dodavanja', 'Ocjena'],
      dtype='object')
```

```
dataFrame.head(5)
```

	Igrač	Utakmice	Minuta	Golovi	Asistencije	Žuti kartoni	Crveni kartoni	Udarci na gol	Točna dodavanja	Ocjena
0	L. Messi	32	2881	25	21	4	0	4.8	82.6	8.71
1	R. Lewandowski	31	2762	34	4	5	0	4.5	73.1	8.13
2	K. De Bruyne	32	2800	13	20	3	0	2.8	81.5	7.97
3	C. Ronaldo	33	2919	31	5	3	0	6.3	85.1	7.82
4	J. Iličić	21	1673	15	5	1	1	3.1	81.9	7.65

```
dataFrame.head(5)[["Igrač", "Utakmice", "Golovi"]]
```

	Igrač	Utakmice	Golovi
0	L. Messi	32	25
1	R. Lewandowski	31	34
2	K. De Bruyne	32	13
3	C. Ronaldo	33	31
4	J. Iličić	21	15

```
noviDataFrame = dataFrame.drop(columns=["Minuta", "Asistencije", "Žuti kartoni",
                                        "Crveni kartoni", "Udarci na gol"])
noviDataFrame.sort_values(by=["Utakmice", "Golovi"], ascending=False, inplace=True)
noviDataFrame.head(5)
```

	Igrač	Utakmice	Golovi	Točna dodavanja	Ocjena
7	C. Immobile	36	36	77.0	7.57
8	L. Alberto	36	6	83.5	7.55
9	A. Gomez	34	7	85.3	7.55
3	C. Ronaldo	33	31	85.1	7.82
6	T. Werner	33	28	75.6	7.60

Slika 6: Prvi primjer rada s *dataframe* objektom (Izvor: izrada autora)

U ovom primjeru učitana je datoteka „playerData.csv“ u varijablu *dataFrame* i prikazani su nazivi stupaca u objektu pomoću atributa *columns*. Zatim je pomoću *head()* funkcije prikazano samo prvih 5 zapisa, a u drugom primjeru samo prvih 5 zapisa s odabranim stupcima koji su predani kao lista elemenata. Nakon toga stvorena je nova varijabla *noviDataFrame* u koju je spremljena kopija *dataFrame* objekta bez odabranih stupaca koji su uklonjeni pomoću naredbe *drop().* Elementi u novom objektu su zatim sortirani prvo prema stupcu „Utakmice“ i zatim prema stupcu „Golovi“ od najvećeg prema najmanjem.

Osim sortiranja i brisanja postojećih stupaca možemo dodavati vlastite stupce ili filtrirati stupce po nekom uvjetu. U sljedećem primjeru dodan je stupac „Omjer golova“ kao omjer vrijednosti u stupcima „Golovi“ i „Utakmice“ zaokruženo na dva decimalna mjesta, a zatim su pomoću *loc* metode prikazani samo oni igrači za koje je vrijednost u stupcu „Omjer golova“ veća ili jednaka od 0.8 i vrijednost u stupcu „Golovi“ veća od 20. Možemo primijetiti da su

podaci u objektu ostali sortirani iz prijašnjeg primjera zbog argumenta *inplace=True* zbog kojeg je sortiran postojeći objekt. Pomoću metode *describe()* možemo vidjeti neke osnovne statističke informacije o odabranom *dataframe* objektu kao što su minimalna i maksimalna vrijednost, broj zapisa, prosjek i slično za svaki stupac.

```
noviDataFrame["Omjer golova"] = round(noviDataFrame["Golovi"]/noviDataFrame["Utakmice"], 2)
noviDataFrame
```

	Igrač	Utakmice	Golovi	Točna dodavanja	Ocjena	Omjer golova	Pozicija
7	C. Immobile	36	36	77.0	7.57	1.00	Napadač
8	L. Alberto	36	6	83.5	7.55	0.17	Krilo
9	A. Gomez	34	7	85.3	7.55	0.21	Krilo
3	C. Ronaldo	33	31	85.1	7.82	0.94	Napadač
6	T. Werner	33	28	75.6	7.60	0.85	Napadač
0	L. Messi	32	25	82.6	8.71	0.78	Napadač
2	K. De Bruyne	32	13	81.5	7.97	0.41	Vezni
1	R. Lewandowski	31	34	73.1	8.13	1.10	Napadač
13	R. Pereira	28	3	78.9	7.50	0.11	Branič
14	S. Gnabry	26	12	82.3	7.49	0.46	Krilo
5	J. Sancho	25	17	85.1	7.64	0.68	Krilo
10	A. Davies	24	3	86.5	7.52	0.12	Branič
4	J. Iličić	21	15	81.9	7.65	0.71	Napadač
12	M. Reus	18	11	82.5	7.50	0.61	Vezni
11	Z.Ibrahimović	16	10	70.9	7.51	0.62	Napadač

```
noviDataFrame.loc[(noviDataFrame["Omjer golova"] >= 0.8) & (noviDataFrame["Golovi"] > 20)]
```

	Igrač	Utakmice	Golovi	Točna dodavanja	Ocjena	Omjer golova
7	C. Immobile	36	36	77.0	7.57	1.00
3	C. Ronaldo	33	31	85.1	7.82	0.94
6	T. Werner	33	28	75.6	7.60	0.85
1	R. Lewandowski	31	34	73.1	8.13	1.10

```
noviDataFrame.describe()
```

	Utakmice	Golovi	Točna dodavanja	Ocjena	Omjer golova
count	15.000000	15.000000	15.000000	15.000000	15.000000
mean	28.333333	16.733333	80.786667	7.714000	0.584667
std	6.410557	11.253359	4.701651	0.333805	0.327237
min	16.000000	3.000000	70.900000	7.490000	0.110000
25%	24.500000	8.500000	77.950000	7.515000	0.310000
50%	31.000000	13.000000	82.300000	7.570000	0.620000
75%	33.000000	26.500000	84.300000	7.735000	0.815000
max	36.000000	36.000000	86.500000	8.710000	1.100000

Slika 7: Drugi primjer rada s *dataframe* objektom (Izvor: izrada autora)

Još jedna zanimljiva metoda za obradu podataka je *groupby()* koja nam omogućuje da podatke grupiramo prema jednom ili više stupaca pa onda na grupirane podatke primijenimo

neke druge funkcije npr. *mean(), sum(), max(), min()* i ostale. U sljedećem primjeru prvo je stvorena nova lista s popisom nogometnih pozicija koje odgovaraju igračima u objektu te je zatim ta lista dodana u *dataframe* kao novi stupac. Na novi objekt zatim je primijenjena metoda *groupby()* s parametrom „Pozicija“ te je na tako grupirane igrače primijenjena metoda *sum()* s parametrom „Golovi“. Rezultat ovih operacija je prikaz ukupnog broja golova prema pozicijama igrača.

```
noviStupac = ["Napadač",
              "Krilni napadač",
              "Krilni napadač",
              "Napadač",
              "Napadač",
              "Napadač",
              "Vezni",
              "Napadač",
              "Branič",
              "Krilni napadač",
              "Krilni napadač",
              "Branič",
              "Napadač",
              "Vezni",
              "Napadač"]
noviDataFrame["Pozicija"] = noviStupac
noviDataFrame.groupby("Pozicija").sum()["Golovi"]
```

```
Pozicija
Branič              6
Krilni napadač     42
Napadač           179
Vezni              24
Name: Golovi, dtype: int64
```

Slika 8: Treći primjer rada s *dataframe* objektom (Izvor: izrada autora)

Ako želimo izmijeniti neku vrijednost u cijelom objektu to možemo učiniti kombinacijom *loc()* metode i uvjeta po kojem mijenjamo vrijednost. U sljedećem primjeru sve pojave vrijednosti „Krilni napadač“ zamijenjene su vrijednošću „Krilo“. Na kraju kada smo zadovoljni s modificiranim podacima možemo cijeli *dataframe* objekt spremiti u nekom obliku po izboru. Ja sam u ovom slučaju koristio metodu *to_csv()* kako bi *noviDataFrame* objekt spremio u *.csv* obliku.

```
noviDataFrame.loc[noviDataFrame['Pozicija'] == 'Krilni napadač', 'Pozicija'] = "Krilo"
noviDataFrame.loc[noviDataFrame["Pozicija"] == "Krilo"]
```

	Igrač	Utakmice	Golovi	Točna dodavanja	Ocjena	Omjer golova	Pozicija
8	L. Alberto	36	6	83.5	7.55	0.17	Krilo
9	A. Gomez	34	7	85.3	7.55	0.21	Krilo
14	S. Gnabry	26	12	82.3	7.49	0.46	Krilo
5	J. Sancho	25	17	85.1	7.64	0.68	Krilo

```
noviDataFrame.to_csv("newPlayerData.csv", index=False)
```

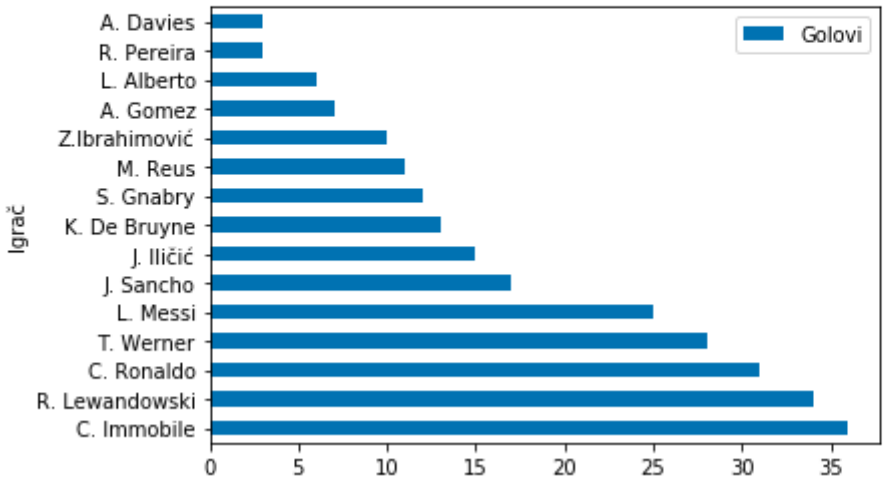
Slika 9: Četvrti primjer rada s *dataframe* objektom (Izvor: izrada autora)

U sljedećem dijelu ovog poglavlja ukratko ću opisati vizualizaciju podataka pomoću Matplotlib biblioteke. Koristit ću isti set podataka kao i ranije tj. isti *dataframe* objekt.

Kod Matplotlib biblioteke rad se svodi na manipulaciju objektima *Figure* i *Axes*. *Figure* je konačna slika i sadrži cijeli graf. Može se sastojati od ostalih elemenata kao što su podgrafovi, naslovi, podnaslovi, legende i drugi elementi. *Axes* definira podgraf koji je dio *Figure* objekta i može imati vlastito definirane elemente poput raspona x-osi, raspona y-osi, labela na osima, tip grafa, boje i ostalih elemenata.

U prvom primjeru prije crtanja grafa sortiran je *dataframe* i pozvana je funkcija *style.use* kojom je stil grafova postavljen na *„seaborn-colorblind“*, jedan od predefiniranih stilova u biblioteci. Jednostavnim pozivom funkcije *plot()* nad Pandas objektom i prosljeđivanjem parametara koji opisuju vrstu grafa i elemente na osima na brz način možemo dobiti vizualizaciju podataka. U ovom slučaju za vrstu grafa odabrao sam horizontalni stupčasti graf (*hbar)* te kao vrijednosti na x-osi broj golova igrača, a na y-osi ime igrača. No ovakav način rada ne dopušta nam uređivanje pojedinih detalja na grafu pa je stoga u sljedećim primjerima pokazan rad u kojem se direktno manipulira Matplotlib objektima.

```
noviDataFrame.sort_values(by="Golovi", ascending=False, inplace=True)
plt.style.use('seaborn-colorblind')
noviDataFrame.plot(kind="barh", y="Golovi", x="Igrač");
```



Slika 10: Primjer rada s *plot()* funkcijom (Izvor: Izrada autora)

U sljedećem primjeru na jedan *Figure* element dodana su dva *Axes* objekta koji prikazuju iste podatke, ali su na drugom podgrafu uređeni neki detalji poput raspona x-osi, boje linije i smjera x-osi. Na podgrafu za koji nisu definirani nikakvi detalji prikazan je stilom koji je odabran na početku, a raspon x-osi je određen automatski. Na *Figure* element također je dodan podnaslov i uređena je pozadinska boja. Svaki od dodanih grafova moguće je posebno uređivati prema želji te postoje brojne druge opcije koje je moguće prilagođavati.

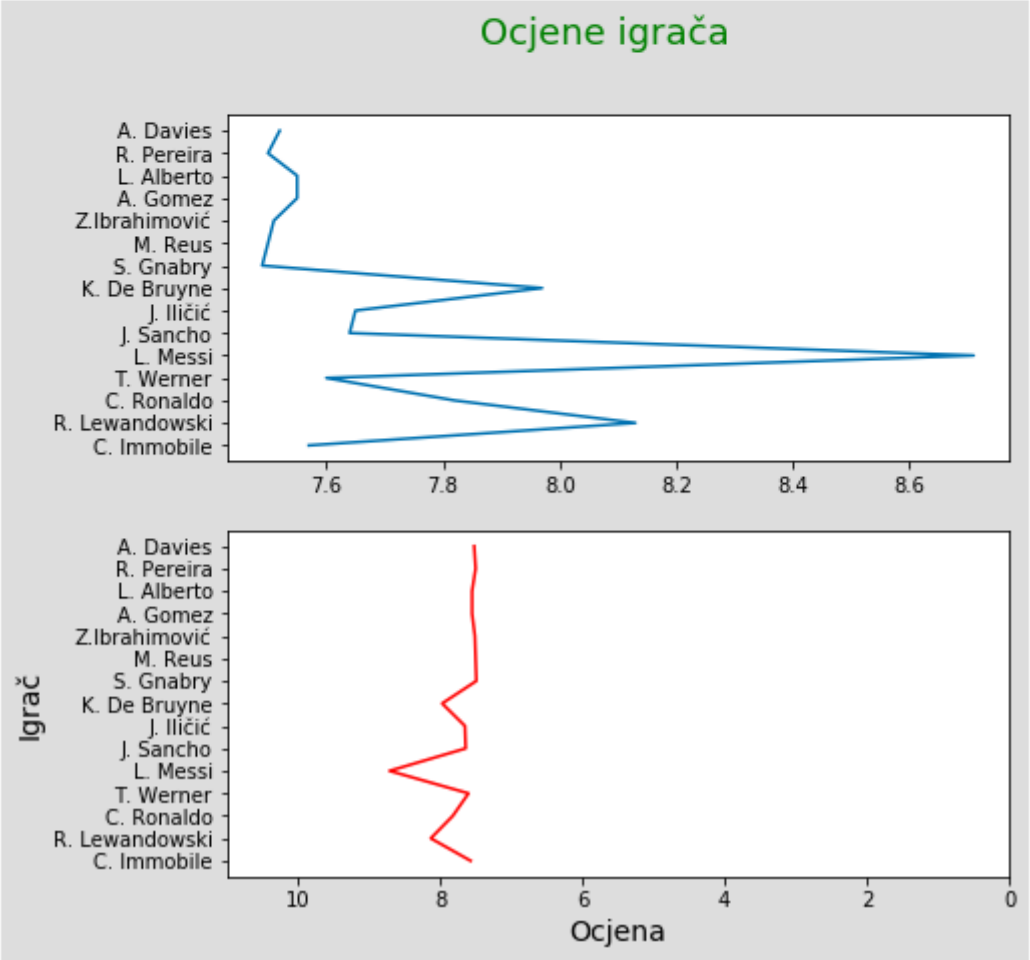
```
fig, ax = plt.subplots(2)

ax[0].plot(noviDataFrame["Ocjena"], noviDataFrame["Igrač"])
ax[1].plot(noviDataFrame["Ocjena"], noviDataFrame["Igrač"], "r")

fig.suptitle("Ocjene igrača", fontsize="18", color="g")
fig.set_facecolor("#dddddd")

fig.set_size_inches(7,7)

ax[1].set_xlim([0, 11])
ax[1].invert_xaxis()
ax[1].set_xlabel("Ocjena", fontsize="14")
ax[1].set_ylabel("Igrač", fontsize="14");
```



Slika 11: Primjer s dva *Axes* objekta na istom *Figure* objektu (Izvor: Izrada autora)

U zadnjem primjeru na istom grafu prikazana su dva seta podataka. Crvenom linijom prikazan je broj golova po igraču dok je zelenom linijom prikazan broj odigranih utakmica po igraču. Zatim je za svaki set podataka izračunat prosjek te su na graf dodane vertikalne linije koje ga prikazuju pomoću funkcije *axvline()* i pripadajuće labele za te linije koje su dodane

pomoću funkcije *text()*. Na kraju je za graf modificirana podjela x-osi i postavljene su labele za svaku od osi s definiranom veličinom i bojom slova.

```
fig, ax = plt.subplots()

ax.plot(noviDataFrame["Golovi"], noviDataFrame["Igrač"], "r")
ax.plot(noviDataFrame["Utakmice"], noviDataFrame["Igrač"], "g")

fig.suptitle("Usporedba odigranih utakmica i golova")

prosjekGolova = noviDataFrame["Golovi"].mean()
ax.axvline(x=prosjekGolova, color="r", linestyle=":", linewidth=3)

prosjekUtakmica = noviDataFrame["Utakmice"].mean()
ax.axvline(x=prosjekUtakmica, color="g", linestyle=":", linewidth=3)

ax.text(5, 14, "Prosjek golova")
ax.text(29, 14,"Prosjek utakmica")

ax.set_xticks([x for x in range(0,45,4)])
ax.set_xlabel("Broj golova i utakmica", fontsize="12", fontweight="bold", color="#0026ff")
ax.set_ylabel("Igrač", fontsize="12", fontweight="bold", color="#0026ff");
```

Slika 12: Složeniji primjer uređivanja grafa (Izvor: Izrada autora)

Biblioteka Matplotlib nudi brojne druge opcije prilagođavanja *Figure* i *Axes* objekata i kombiniranja različitih vrsti i detalja grafova. Dodatne informacije mogu se pronaći u dokumentaciji koja je opsežna i detaljno opisuje svaku opciju.

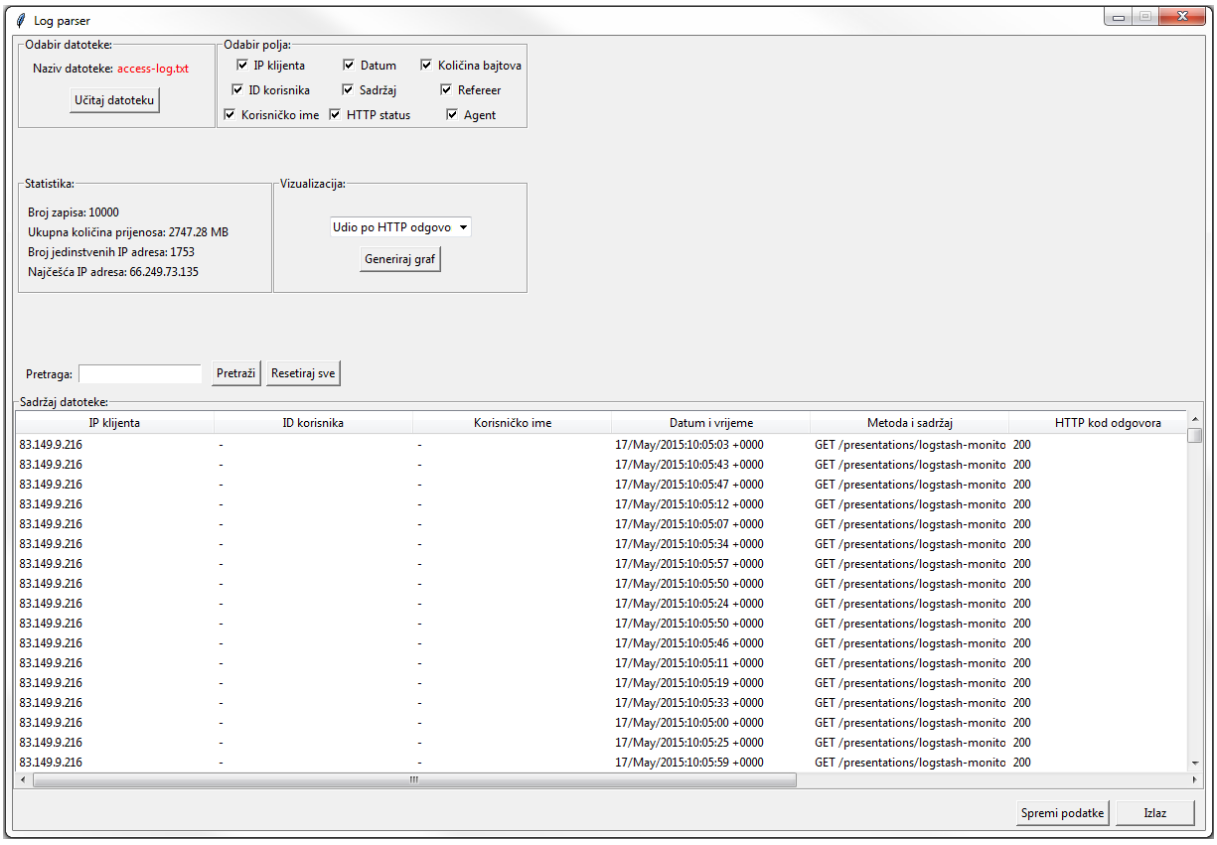
4. Python aplikacija za analizu i vizualizaciju

U ovom poglavlju bit će opisana izrađena aplikacija koja omogućuje analizu i vizualizaciju dnevničkih zapisa. Kao primjer dnevničkog zapisa koji će se analizirati odabran je *Apache server access log* (*examples/apache_logs*, 2017) koji sadrži 10000 zapisa. Zapisi su u sljedećem formatu:

a) IP adresa klijenta
b) ID korisnika
c) Korisničko ime
d) Datum i vrijeme
e) Metoda i sadržaj
f) HTTP kod odgovora
g) Veličina u bajtovima
h) Referrer
i) Korisnički agent

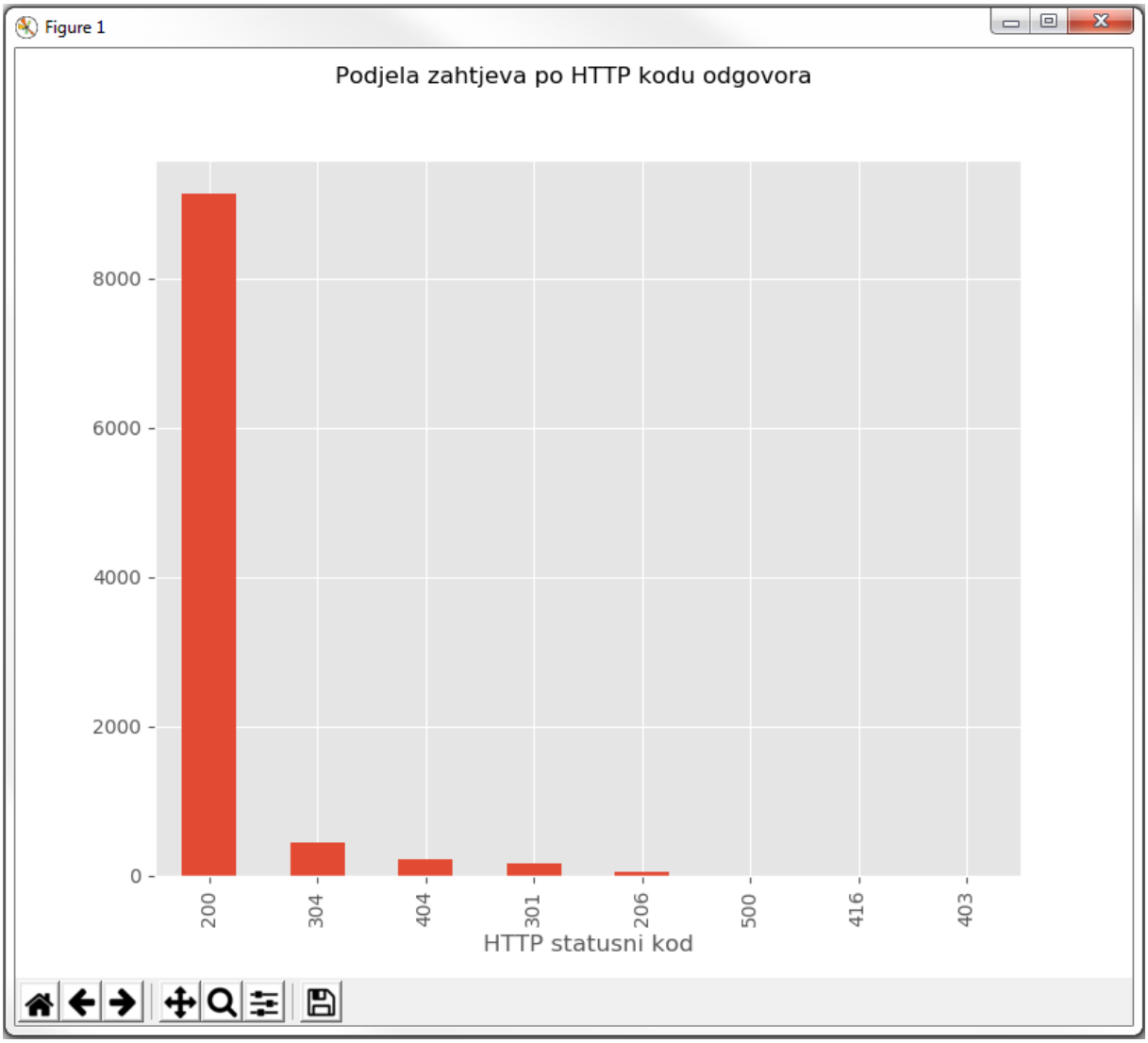
Aplikacija je izrađena pomoću Python programskog jezika i dostupnih Python biblioteka. Korištene su biblioteke *os, re, tkinter, pandas* i *matplotlib.* Biblioteka *os* omogućuje interakciju s funkcionalnostima operacijskog sustava, *re* je biblioteka za rad s regularnim izrazima, a *tkinter* je Python biblioteka koja omogućuje izradu grafičkih sučelja. Biblioteke *pandas* i *matplotlib* objašnjene su u poglavlju 3. Za rad s aplikacijom potrebno je najprije instalirati sve korištene biblioteke koje nisu uključene u Python.

Aplikacija je podijeljena u dvije Python datoteke. Datoteka *Parser.py* sadrži klasu Parser koja je zadužena za parsiranje i obradu podataka dok je u datoteci *App.py* definirano korisničko sučelje i funkcije koje pomoću klase Parser obrađuju korisničke unose i prikazuju podatke.



Slika 13: Izgled sučelja aplikacije s učitanim dnevničkim zapisom (Izvor: Izrada autora)

Na sučelju aplikacije potrebno je najprije odabrati datoteku koju želimo analizirati pomoću gumba „Učitaj datoteku“. Ako je datoteka uspješno učitana pojavljuje se poruka uspjeha i naziv datoteke ispisan je nakon teksta „Naziv datoteke:“, u protivnom aplikacija javlja pogrešku. Aplikacija nakon učitavanja podataka prikazuje statistiku o datoteci i sav sadržaj u tablici u donjem dijelu sučelja u obliku tablice. Statistika prikazuje ukupan broj pročitanih zapisa, ukupnu količinu podataka koja je prenesena u megabajtima, broj jedinstvenih IP adresa u dnevničkom zapisu i najčešću IP adresu. Korisnik može uključiti i isključiti polja koja želi prikazati pomoću selektora u grupi „Odabir polja:“. U grupi „Vizualizacija“ moguće je prikazati nekoliko vrsta grafova pomoću gumba „Generiraj graf“. Prvi graf prikazuje količinu pojedinog HTTP statusnog koda odgovora u obliku stupčastog dijagrama. Drugi graf prikazuje količinu pojedine korištene HTTP metode zahtjeva u obliku horizontalnog stupčastog dijagrama. Treći graf prikazuje količinu zahtjeva po IP adresi za prvih pet IP adresa po broju zahtjeva u obliku horizontalnog stupčastog dijagrama. Posljednji graf prikazuje udio zahtjeva grupirano po danu u obliku tortnog dijagrama. Iznad tablice koja sadrži podatke iz dnevničkog zapisa nalazi se polje za unos i gumb za pretraživanje podataka te gumb za resetiranje prikaza pomoću kojeg se sučelje i prikazani podaci vraćaju u početni oblik kakav su imali nakon učitavanja datoteke. U donjem desnom kutu nalazi se gumb koji omogućuje spremanje trenutno prikazanih podataka u obliku *.csv* datoteke i gumb za izlaz iz aplikacije.

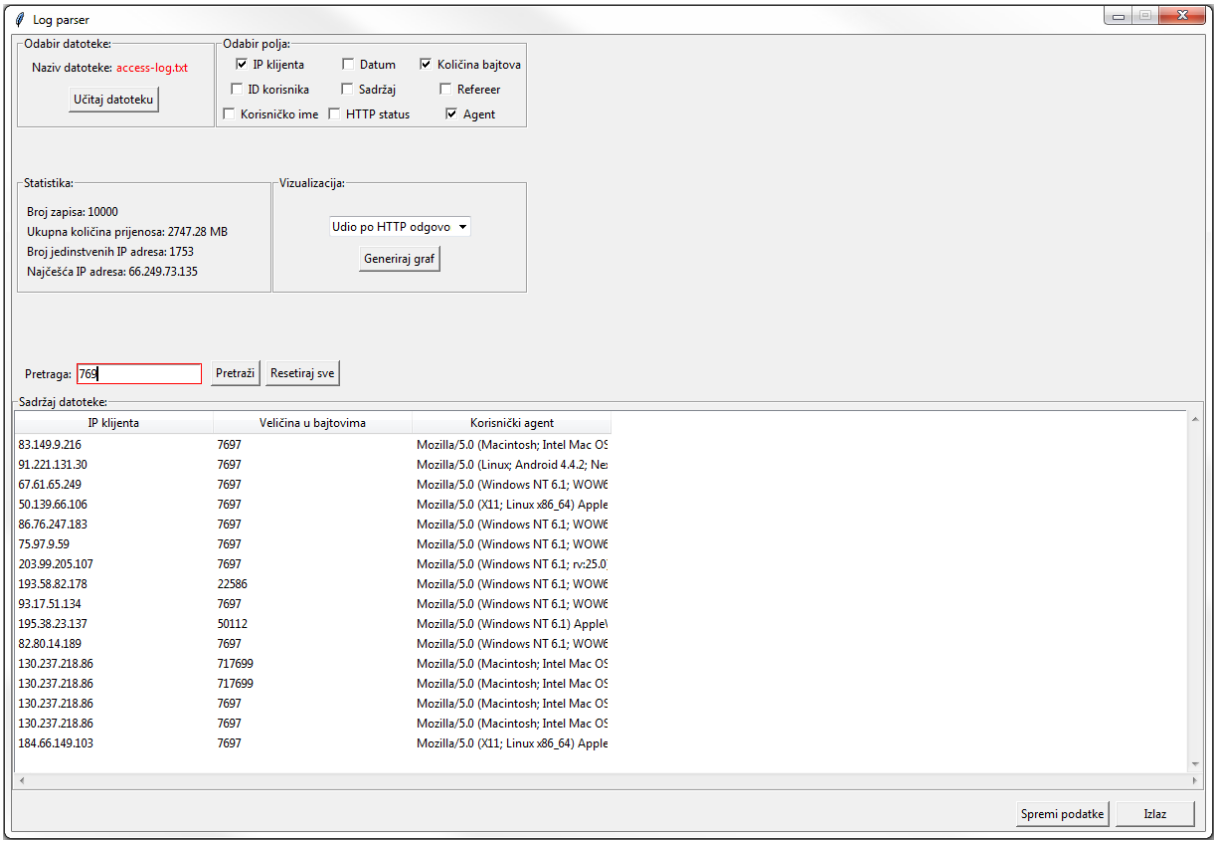


Slika 14: Graf s prikazom količine zahtjeva po HTTP statusnom kodu (Izvor: Izrada autora)

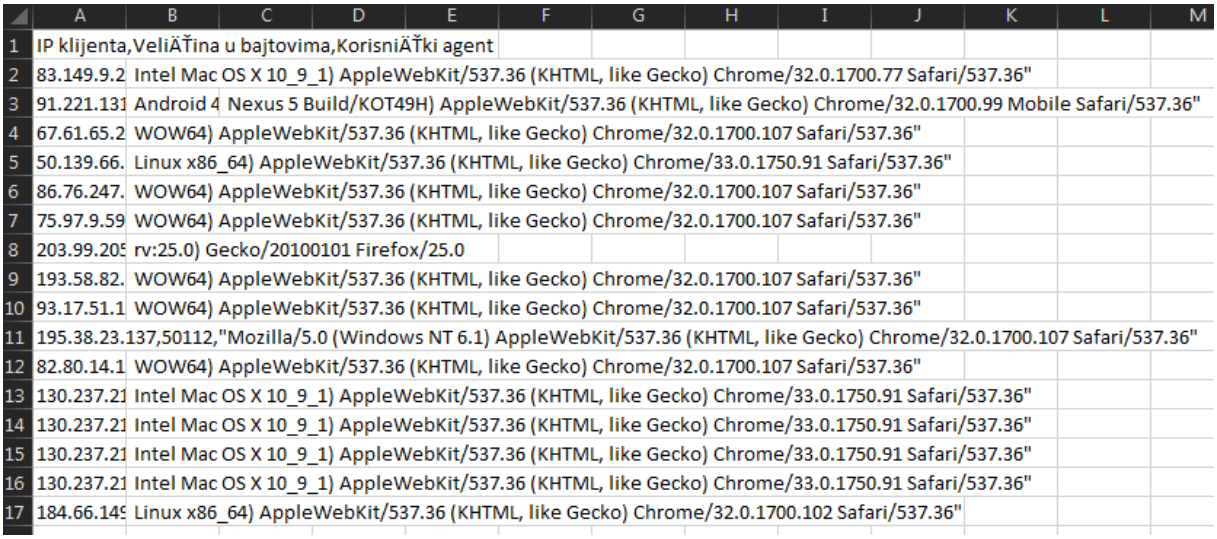
Klasa Parser radi na način da se na najprije pozove funkcija koja otvara odabranu tekstualnu datoteku te je čita redak po redak. Svaki redak se pomoću regularnih izraza dijeli na polja koja se dodaju u listu. Lista s poljima se tada dodaje u drugu listu koja na kraju sadrži sve liste s poljima za pojedini zapis. Kada se pročita čitava datoteka kreira se Pandas *dataframe* objekt iz liste koja sadrži sve zapise podijeljene po poljima. Sve ostale funkcije najprije rade kopiju ovog objekta te onda manipuliraju i izmjenjuju tu kopiju kako bi se u prvom objektu sačuvali originalni podaci iz datoteke. Klasa sadrži funkcije za pretraživanje i filtriranje podataka, generiranje statistike za dnevnički zapis, generiranje grafova i spremanje odabranih podataka. Funkcije manipuliraju podacima pomoću Pandas metoda dok se grafovi generiraju pomoću matplotlib biblioteke.

Na primjeru izrađene aplikacije pokazano je samo nekoliko oblika informacija koje je moguće dobiti analizom dnevničkih zapisa. Na tržištu su danas dostupni brojni profesionalni

alati koji omogućuju opsežniju analizu i vizualizaciju podataka iz različitih formata dnevničkih zapisa.



Slika 15: Prikaz funkcionalnosti pretraživanja i filtriranja polja (Izvor: Izrada autora)



Slika 16: Prikaz funkcionalnosti spremanja odabranih podataka u *.csv* obliku (Izvor: Izrada autora)

5. Zaključak

Tema ovog završnog rada su dnevnički zapisi te njihova analiza i vizualizacija u pomoću programskog jezika Python. U prvom dijelu rada pojašnjen je pojam dnevničkih zapisa te njihova svrha. Opisano je nekoliko osnovnih kategorija u koje možemo svrstati dnevničke zapise te način na koji se oni koriste. Poseban naglasak stavljen je na poslužiteljske dnevničke zapise koji u današnjem svijetu imaju veliku ulogu u nadziranju sustava organizacija čije poslovanje ovisi o njihovoj mrežnoj infrastrukturi.

U nastavku poglavlja obrađena je organizacija dnevničkih zapisa na osobnim računalima i to na Windows platformi s alatom Windows Event Viewer, te na Linux platformi. Objašnjena je struktura i vrste dnevničkih zapisa za pojedinu platformu. Nakon osobnih računala obrađena je i organizacija dnevničkih zapisa na poslužiteljima. Prije toga ukratko je opisan i sam pojam poslužitelja i način na koji rade. Na kraju potpoglavlja o poslužiteljskim dnevničkim zapisima obrađena je struktura i format zapisa za nekoliko odabranih formata poslužiteljskih dnevničkih zapisa. Formati koji su obrađeni su *Common log format, Extended common log format* i *Microsoft IIS log format.*

Drugi dio rada odnosi se na praktični dio u kojem je opisana izrađena aplikacija za analizu i vizualizaciju dnevničkih zapisa pomoću programskog jezika Python. U prvom poglavlju ovog dijela rada opisane su dvije najvažnije biblioteke, Pandas i Matplotlib, koje su korištene za obradu i vizualizaciju podataka. Kroz nekoliko jednostavnih primjera prikazan je način rada s ovim bibliotekama te neke osnovne funkcionalnosti. Drugi dio praktičnog dijela rada odnosi se na izrađenu aplikaciju. Opisane su sve funkcionalnosti koje su ugrađene u aplikaciju te je na primjeru *Apache server access* dnevničkog zapisa prikazan način na koji aplikacija obrađuje i vizualizira podatke.

Na kraju možemo zaključiti da su dnevnički zapisi vrlo vrijedan izvor podataka i informacija, kako za administratore velikih sustava tako i za korisnike osobnih računala, jer pružaju uvid u pozadinu rada sustava ili aplikacija te omogućuju da analiziramo njihovo ponašanje kroz neki odabrani vremenski interval. Pomoću programskog jezika Python i pripadnih dostupnih biblioteka možemo na brz i jednostavan način obraditi i vizualizirati podatke sadržane u dnevničkim zapisima ili se pak možemo odlučiti za neki od brojnih dostupnih alata za obradu dnevničkih zapisa .

Popis literature

A. Grimes, R. (2018). *Why you need centralized logging and event log management | CSO Online*. https://www.csoonline.com/article/3280123/why-you-need-centralized-logging-and-event-log-management.html

Brief introduction of log file formats. (bez dat.). Preuzeto 29. kolovoz 2020., od https://www.loganalyzer.net/log-analysis/log-file-format.html#W3CExtendedLogFileFormat

Christensson, P. (2010, travanj 14). *Log File Definition*. https://techterms.com/definition/logfile

E. Zhang. (2018). *What is Log Analysis? Use Cases, Best Practices, and More | Digital Guardian*. https://digitalguardian.com/blog/what-log-analysis-use-cases-best-practices-and-more

Event Log Monitoring Tool - A Tutorial. (bez dat.). Preuzeto 24. kolovoz 2020., od https://www.manageengine.com/network-monitoring/Eventlog_Tutorial_Part_I.html

examples/apache_logs. (2017, studeni 6). https://github.com/elastic/examples/blob/master/Common Data Formats/apache_logs/apache_logs

Haas, J. (2020, veljača 14). *What Are Linux Log Files and How Can You Read Them?* https://www.lifewire.com/introduction-to-linux-log-files-2192233

IIS Logging | Microsoft Docs. (2018, svibanj 31). https://docs.microsoft.com/en-us/windows/win32/http/iis-logging

Lee, D. (2019). *Application Logs: What They Are and How to Use Them - XpoLog*. https://www.xplg.com/application-logs-what-how/

Linux Logging Basics - The Ultimate Guide To Logging. (2015, rujan 15). https://www.loggly.com/ultimate-guide/linux-logging-basics/

M. Hallam-Baker, P., & Behlendorf, B. (bez dat.). *Extended Log File Format*. Preuzeto 28. kolovoz 2020., od https://www.w3.org/TR/WD-logfile.html

Robben. (2019, studeni). *How do servers work? A Detailed Guide into Web Servers*. https://host4geeks.com/blog/how-do-servers-work/?

Six, B. (2020a). *Server Log Files in a Nutshell | Graylog*. https://www.graylog.org/post/server-log-files-in-a-nutshell

Six, B. (2020b, siječanj 15). *Log Formats – a (Mostly) Complete Guide | Graylog*. https://www.graylog.org/post/log-formats-a-complete-guide

sysklogd(8): system logging utilities - Linux man page. (bez dat.). Preuzeto 26. kolovoz 2020., od https://linux.die.net/man/8/sysklogd

What is a Log File? | Sumo Logic. (2019). https://www.sumologic.com/glossary/log-file/

What is a web server? - Learn web development | MDN. (2020). https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_is_a_web_server

Windows Logging Basics - The Ultimate Guide To Logging. (2015, rujan 15). https://www.loggly.com/ultimate-guide/windows-logging-basics/

Popis slika

Slika 1: Izgled sučelja Windows Event Viewer-a 4
Slika 2: Opcije filtriranja u Event Viwer-u 5
Slika 3: Primjer sadržaja */var/log* direktorija 6
Slika 4: Primjer HTTP zahtjeva 8
Slika 5: Princip rada web poslužitelja 9
Slika 6: Prvi primjer rada s *dataframe* objektom 15
Slika 7: Drugi primjer rada s *dataframe* objektom 16
Slika 8: Treći primjer rada s *dataframe* objektom 17
Slika 9: Četvrti primjer rada s *dataframe* objektom 18
Slika 10: Primjer rada s *plot()* funkcijom 19
Slika 11: Primjer s dva *Axes* objekta na istom *Figure* objektu 20
Slika 12: Složeniji primjer uređivanja grafa 21
Slika 13: Izgled sučelja aplikacije s učitanim dnevničkim zapisom 23
Slika 14: Graf s prikazom količine zahtjeva po HTTP statusnom kodu 24
Slika 15: Prikaz funkcionalnosti pretraživanja i filtriranja polja 25
Slika 16: Prikaz funkcionalnosti spremanja odabranih podataka u *.csv* obliku 25

Prilozi

Izvorni kod datoteke App.py

```
from tkinter import Tk, ttk, Button, Label, Entry, messagebox, Checkbutton, LabelFrame, Scrollbar, filedialog, END, IntVar
from Parser import Parser
import os


def LoadFile():
    path = os.getcwd()
    filename = filedialog.askopenfilename(initialdir=path, title="Odaberite datoteku", filetypes=[("Text files","*.txt")])


    if len(filename) > 0:
        try:
            parser.ClearData()
            parser.SetFilename(filename)
            splitFilename = filename.split("/")
            loadedFileLabel.config(text = splitFilename[len(splitFilename)-1])
            FillDataTable(parser.DataFrame)
            DisplayStatistics()
            messagebox.showinfo("Uspjeh!", "Datoteka uspješno učitana!")
        except:
            messagebox.showwarning("Greška", "Dogodila se pogreška! Pokušajte ponovno.")


def ClearDataTable():
    dataTable.delete(*dataTable.get_children())


def FillDataTable(dataFrame):
    ClearDataTable()
    dataTable["column"] = list(dataFrame.columns)
    dataTable["show"] = "headings"
    for column in dataTable["columns"]:
        dataTable.heading(column, text=column)


    dataFrameRows = dataFrame.to_numpy().tolist()
    for row in dataFrameRows:
        dataTable.insert("", "end", values=row)
```

```
def SearchAndFilter():
    term = searchInput.get()
    checkboxList = [varClientIP.get(),
                    varClientID.get(),
                    varUsername.get(),
                    varDate.get(),
                    varContent.get(),
                    varHTTP.get(),
                    varBytes.get(),
                    varRef.get(),
                    varAgent.get()]

    if len(parser.dataFrameList) > 0:
        parser.SearchAndFilter(term, checkboxList)
        FillDataTable(parser.ModifiedDataFrame)

def DisplayStatistics():
    statisticsDict = parser.GetStatistics()

    recordsText = "Broj zapisa: " + statisticsDict["records"]
    megabytesText = "Ukupna količina prijenosa: " +
statisticsDict["mbytes"] + " MB"
    uniqueText = "Broj jedinstvenih IP adresa: " +
statisticsDict["uniqueIPs"]
    topText = "Najčešća IP adresa: " + statisticsDict["topIP"]

    amountOfRecordsLabel.config(text=recordsText)
    megabytesLabel.config(text=megabytesText)
    uniqueIPsLabel.config(text=uniqueText)
    topIPLabel.config(text=topText)

def DisplayGraph():
    graphValues = {
                    "Udio po HTTP odgovoru": 0,
                    "Udio po HTTP metodi": 1,
                    "Top 5 po broju zahtjeva": 2,
                    "Količina zahtjeva po danu": 3
                }

    if len(parser.dataFrameList) > 0:
```

```
        parser.DisplayGraph(graphValues[graphsCombobox.get()])
        window.quit()

def SaveData():
    if len(parser.ModifiedDataFrame) > 0:
        path = filedialog.asksaveasfilename(defaultextension=".csv")
        parser.SaveDataFrameAsCSV(parser.ModifiedDataFrame, path)
        messagebox.showinfo("Uspjeh!", "Datoteka uspješno kreirana!")

def ResetUI():
    FillDataTable(parser.DataFrame)
    ResetCheckboxes()
    searchInput.delete(0, END)

def ResetCheckboxes():
    clientIPCheckbox.select()
    clientIDCheckbox.select()
    usernameCheckbox.select()
    dateCheckbox.select()
    contentCheckbox.select()
    httpCheckbox.select()
    bytesCheckbox.select()
    refCheckbox.select()
    agentCheckbox.select()

#-----------------------------------------------parser--------------------
---------------------------
parser = Parser()

#-----------------------------------------------window--------------------
---------------------------
window = Tk()
window.geometry("1200x800+250+50")
window.resizable(0, 0)
window.title("Log parser")

#-----------------------------------------------fileselect----------------
---------------------------
fileFrame = LabelFrame(window, text="Odabir datoteke:", height=94,
width=200, borderwidth=2, relief="groove")
fileFrame.place(x=5, y=0)
```

```
filenameLabel = Label(fileFrame, text="Naziv datoteke:")
filenameLabel.place(x=10, y=5)

loadedFileLabel = Label(fileFrame, text="/", foreground="red")
loadedFileLabel.place(x=95, y=5)

filenameButton = Button(fileFrame, text="Učitaj datoteku",
command=LoadFile)
filenameButton.place(x=50, y=35)

#-----------------------------------------------checkboxes------------------
----------------------------

varClientIP = IntVar()
varClientID = IntVar()
varUsername = IntVar()
varDate = IntVar()
varContent = IntVar()
varHTTP = IntVar()
varBytes = IntVar()
varRef = IntVar()
varAgent = IntVar()

checkboxesFrame = LabelFrame(window, text="Odabir polja:", height=120,
width=600, borderwidth=2, relief="groove")
checkboxesFrame.place(x=205, y=0)

clientIPCheckbox = Checkbutton(checkboxesFrame, text="IP klijenta",
variable=varClientIP, command=SearchAndFilter)
clientIPCheckbox.grid(row=0, column=0)
clientIPCheckbox.select()

clientIDCheckbox = Checkbutton(checkboxesFrame, text="ID korisnika",
variable=varClientID, command=SearchAndFilter)
clientIDCheckbox.grid(row=1, column=0)
clientIDCheckbox.select()

usernameCheckbox = Checkbutton(checkboxesFrame, text="Korisničko ime",
variable=varUsername, command=SearchAndFilter)
usernameCheckbox.grid(row=2, column=0)
usernameCheckbox.select()

dateCheckbox = Checkbutton(checkboxesFrame, text="Datum", variable=varDate,
command=SearchAndFilter)
```

```
dateCheckbox.grid(row=0, column=1)
dateCheckbox.select()


contentCheckbox = Checkbutton(checkboxesFrame, text="Sadržaj",
variable=varContent, command=SearchAndFilter)
contentCheckbox.grid(row=1, column=1)
contentCheckbox.select()


httpCheckbox = Checkbutton(checkboxesFrame, text="HTTP status",
variable=varHTTP, command=SearchAndFilter)
httpCheckbox.grid(row=2, column=1)
httpCheckbox.select()


bytesCheckbox = Checkbutton(checkboxesFrame, text="Količina bajtova",
variable=varBytes, command=SearchAndFilter)
bytesCheckbox.grid(row=0, column=2)
bytesCheckbox.select()


refCheckbox = Checkbutton(checkboxesFrame, text="Refereer",
variable=varRef, command=SearchAndFilter)
refCheckbox.grid(row=1, column=2)
refCheckbox.select()


agentCheckbox = Checkbutton(checkboxesFrame, text="Agent",
variable=varAgent, command=SearchAndFilter)
agentCheckbox.grid(row=2, column=2)
agentCheckbox.select()


#-------------------------------------------------search--------------------
---------------------------
searchLabel = Label(window, text="Pretraga:")
searchLabel.place(x=10, y=330)


searchInput = Entry(window, width=20, highlightcolor="red",
highlightthickness=1)
searchInput.place(x=65, y=330)


searchButton = Button(window, text="Pretraži", command=SearchAndFilter)
searchButton.place(x=200, y=327)


#-------------------------------------------------reset--------------------
---------------------------
resetButton = Button(window, text="Resetiraj sve", command=ResetUI)
```

```
resetButton.place(x=255, y=327)


#-----------------------------------------------datatable-----------------
----------------------------

dataFrame = LabelFrame(window, text="Sadržaj datoteke:")

dataFrame.place(height=400, width=1200, x=0, y=360)


dataTable = ttk.Treeview(dataFrame)

dataTable.place(relheight=1, relwidth=1)


dataTableBarX = Scrollbar(dataFrame, orient="horizontal",
command=dataTable.xview)

dataTableBarY = Scrollbar(dataFrame, orient="vertical",
command=dataTable.yview)

dataTable.configure(xscrollcommand=dataTableBarX.set,
yscrollcommand=dataTableBarY.set)

dataTableBarX.pack(side="bottom", fill="x")

dataTableBarY.pack(side="right", fill="y")


#-----------------------------------------------statistics----------------
----------------------------

statisticsFrame = LabelFrame(window, text="Statistika:", height=120,
width=257, borderwidth=2, relief="groove")

statisticsFrame.place(x=5, y=140)


amountOfRecordsLabel = Label(statisticsFrame, text="Broj zapisa: /")

amountOfRecordsLabel.place(x=5, y=10)


megabytesLabel = Label(statisticsFrame, text="Ukupna količina prijenosa:
/")

megabytesLabel.place(x=5, y=30)


uniqueIPsLabel = Label(statisticsFrame, text="Broj jedinstvenih IP adresa:
/")

uniqueIPsLabel.place(x=5, y=50)


topIPLabel = Label(statisticsFrame, text="Najčešća IP adresa: /")

topIPLabel.place(x=5, y=70)


#-----------------------------------------------graphs--------------------
----------------------------

graphsFrame = LabelFrame(window, text="Vizualizacija:", height=120,
width=257, borderwidth=2, relief="groove")

graphsFrame.place(x=262, y=140)
```

```
comboboxValues = ["Udio po HTTP odgovoru", "Udio po HTTP metodi", "Top 5 po broju zahtjeva", "Količina zahtjeva po danu"]
graphsCombobox = ttk.Combobox(graphsFrame, values=comboboxValues)
graphsCombobox.current(0)
graphsCombobox.place(x=55, y=25)


graphsButton = Button(graphsFrame, text="Generiraj graf", command=DisplayGraph)
graphsButton.place(x=85, y= 55)


#-----------------------------------------------save----------------------------------------

quitButton = Button(window, text="Spremi podatke", command=SaveData)
quitButton.place(x=1010, y=770)


#-----------------------------------------------quit----------------------------------------

quitButton = Button(window, text="Izlaz", width=10, command=window.quit)
quitButton.place(x=1110, y=770)


window.mainloop()
```

Izvorni kod datoteke Parser.py

```
import re
import pandas as pd
import matplotlib.pyplot as plt
import copy

regex = r"([(\d\.)]+) (.*) (.*) \[(.*?)\] \"(.*?)\" (\d+) (\d+|.*) \"(.*)\"
\"(.*)\""
header = ["IP klijenta", "ID korisnika", "Korisničko ime", "Datum i
vrijeme", "Metoda i sadržaj", "HTTP kod odgovora", "Veličina u bajtovima",
"Referrer", "Korisnički agent"]

class Parser():

    def __init__(self):
        self.rawData = list()
        self.parsedData = list()
        self.dataFrameList = list()

    def SetFilename(self, filename):
        self.filename = filename
        self.ReadFile()

    def ReadFile(self):
        self.ReadData()
        self.ParseData()
        self.CreateDataFrameList()
        self.CreateDataFrame()

    def ReadData(self):
        with open(self.filename, "r") as file:
            self.rawData = file.readlines()

    def ParseData(self):
        for x in range(len(self.rawData)):
            line = self.rawData[x].replace("\n", "")
            self.parsedData.append(line)

    def CreateDataFrameList(self):
        for x in range(len(self.parsedData)):
```

```
            regexMatch = re.match(regex, self.parsedData[x])
            self.dataFrameList.append(list(regexMatch.groups()))

    def CreateDataFrame(self):
        self.DataFrame = pd.DataFrame(self.dataFrameList, columns=header)
        self.ModifiedDataFrame = self.DataFrame

    def ClearData(self):
        self.rawData = list()
        self.parsedData = list()
        self.dataFrameList = list()
        self.DataFrame = None
        self.ModifiedDataFrame = None

    def SearchAndFilter(self, term, checkboxList):
        df = self.DataFrame
        self.ModifiedDataFrame = df.loc[(df["IP
klijenta"].str.contains(term)) | (df["ID korisnika"].str.contains(term)) |
(df["Korisničko ime"].str.contains(term)) | (df["Datum i
vrijeme"].str.contains(term)) | (df["Metoda i sadržaj"].str.contains(term))
| (df["Veličina u bajtovima"].str.contains(term)) |
(df["Referrer"].str.contains(term)) | (df["Korisnički
agent"].str.contains(term))]

        dropList = list()

        for x in range(len(checkboxList)):
            if checkboxList[x] == 0:
                dropList.append(header[x])

        self.ModifiedDataFrame =
self.ModifiedDataFrame.drop(columns=dropList)

    def GetStatistics(self):
        statisticsDict = dict()
        statisticsDict["records"] = self.GetAmountOfRecords()
        statisticsDict["mbytes"] = self.GetTotalMBytes()
        statisticsDict["uniqueIPs"] = self.GetAmountOfUniqueIP()
        statisticsDict["topIP"] = self.GetTopIP()

        return statisticsDict

    def GetAmountOfRecords(self):
```

```
    df = self.DataFrame

    return str(len(df.index))

def GetTotalMBytes(self):
    df = self.DataFrame["Veličina u bajtovima"]
    totalBytes = 0

    for value in df:
        if value.isdigit():
            totalBytes += int(value)

    return str(round(totalBytes/1000000, 2))

def GetAmountOfUniqueIP(self):
    df = self.DataFrame
    uniqueIPs = df["IP klijenta"].nunique()

    return str(uniqueIPs)

def GetTopIP(self):
    df = self.DataFrame
    topIP = df["IP klijenta"].value_counts().idxmax()

    return topIP

def DisplayGraph(self, ind):
    plt.style.use("ggplot")

    if ind == 0:
        self.DisplayByHTTPResponse()
    elif ind == 1:
        self.DisplayByMethod()
    elif ind == 2:
        self.DisplayTopRequestIPs()
    else:
        self.DisplayAmountPerDay()

def DisplayByMethod(self):
    df = copy.deepcopy(self.DataFrame)
```

```
        fig, ax = plt.subplots()
        splitColumn = df["Metoda i sadržaj"].str.split(" ", expand = True)
        df["Metoda"] = splitColumn[0]

        fig.set_size_inches(8, 7)
        fig.suptitle("Podjela zahtjeva po korištenoj HTTP metodi")
        ax.set_xlabel("Broj zahtjeva")

        df["Metoda"].value_counts().plot(kind="barh")
        plt.show()

    def DisplayByHTTPResponse(self):
        df = copy.deepcopy(self.DataFrame)

        fig, ax = plt.subplots()

        fig.set_size_inches(8, 7)
        fig.suptitle("Podjela zahtjeva po HTTP kodu odgovora")
        ax.set_xlabel("HTTP statusni kod")

        df["HTTP kod odgovora"].value_counts().plot(kind="bar")
        plt.show()

    def DisplayAmountPerDay(self):
        df = copy.deepcopy(self.DataFrame)

        fig, ax = plt.subplots()
        splitColumn = df["Datum i vrijeme"].str.split(":", expand = True)
        df["Datum"] = splitColumn[0]

        fig.set_size_inches(8, 7)
        fig.suptitle("Količina zahtjeva po danu")

        df["Datum"].value_counts().plot(kind="pie")
        plt.show()

    def DisplayTopRequestIPs(self):
        df = copy.deepcopy(self.DataFrame)
```

```
    fig, ax = plt.subplots()
    df = df["IP klijenta"].value_counts(ascending=False).head(5)

    fig.set_size_inches(10, 7)
    fig.suptitle("Top 5 IP adresa po broju zahtjeva")

    df.plot(kind="barh")
    plt.show()

def SaveDataFrameAsCSV(self, dataFrame, path):
    dataFrame.to_csv(path, index = False, header=True)
```