Izrada informacijsko komunikacijskog sustava za praćenje prometnih entiteta i procjenu parametra brzine entiteta u prometnoj mreži

Vizner, Valentino

Master's thesis / Diplomski rad

2018

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Transport and Traffic Sciences / Sveučilište u Zagrebu, Fakultet prometnih znanosti**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:119:958559

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

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

Repository / Repozitorij:

Faculty of Transport and Traffic Sciences - Institutional Repository

SVEUČILIŠTE U ZAGREBU
FAKULTET PROMETNIH ZNANOSTI

Valentino Vizner

IZRADA INFORMACIJSKO-KOMUNIKACIJSKOG SUSTAVA ZA PRAĆENJE PROMETNIH ENTITETA I PROCJENU PARAMETRA BRZINE ENTITETA U PROMETNOJ MREŽI

DIPLOMSKI RAD

Zagreb, 2018.

Sveučilište u Zagrebu
Fakultet prometnih znanosti

DIPLOMSKI RAD

IZRADA INFORMACIJSKO-KOMUNIKACIJSKOG SUSTAVA ZA PRAĆENJE PROMETNIH ENTITETA I PROCJENU PARAMETRA BRZINE ENTITETA U PROMETNOJ MREŽI

INFORMATION-COMMUNICATION SYSTEM FOR MONITORING TRAFFIC ENTITIES AND PARAMETER ESTIMATION OF AN ENTITY VELOCITY IN TRAFFIC NETWORK

Mentor: prof. dr. sc. Tonči Carić

Student: Valentino Vizner

JMBAG: 0135228192

Zagreb, rujan 2018.

Zahvaljujem se svom mentoru prof. dr. sc. Tončiju Carić̀u na čijim sam kolegijima tijekom studiranja stekao izvrsna znanja o relacijskim bazama podataka te na strpljenju, smirenost i pomoći pri izradi diplomskoga rada. Također, značajnu zahvalu upućujem Tomislavu Erdeliću, mag. ing. el. techn. inf. na tehničkoj pomoći pri izradi rada i nepresušnim korisnim savjetima i smjernicama bez kojih rad ne bi bio valjan. Zahvaljujem se svojoj obitelj, kolegama s fakulteta i djevojci koji su me podržavali tijekom cijelog trajanja studija i za vrijeme pisanja diplomskog rada.

IZRADA INFORMACIJSKO-KOMUNIKACIJSKOG SUSTAVA ZA PRAĆENJE PROMETNIH ENTITETA I PROCJENU PARAMETRA BRZINE ENTITETA U PROMETNOJ MREŽI

Sažetak:

Kvantitativna količina podataka osnova je za moguću obradu i pohranu podataka te grafički prikaz GPS podataka o prometnim entitetima unutar prometnog sustava. Za kvalitetnu procjenu prosječnih brzina, vremena putovanja i razinu pouzadnosti dobivenih rezultata tijekom dana, količinski je potreban zadovoljavajući broj podataka za svaki pojedini cestovni segment. Također, potreban je i odgovarajući infromacijsko-komunikacijski sustav koji će filtrirati, obraditi i pohraniti podatke u relacijsku bazu podataka te u aplikaciji prikazati rezultate krajnjem korisniku, što je ujedno i cilj ovog diplomskog rada. Prikazane su matematičke metode filtriranja podataka s odgovarajućim formulacijama, obrada podataka te grafički prikaz podataka. Nakon obrade, obrađeni podaci pohranjuju se u relacijsku bazu podataka PostgreSQL. Implementirano je grafičko sučelje u Python programskom jeziku za dohvaćanje i prikaz podataka spremljenih u bazu podatka na interaktivnoj karti. Time se stvara proširivi informacijsko-komunikacijski sustav informiranja korisnika o stanju prometa i pruža mogućnost optimiziranja ruta flote vozila.

Ključne riječi: relacijska baza podataka, SQL, Python, GPS, obrada i pohrana podataka, razina pouzdanosti, PostgreSQL, aplikacija s interaktivnom kartom

INFORMATION-COMMUNICATION SYSTEM FOR MONITORING TRAFFIC ENTITIES AND PARAMETER ESTIMATION OF AN ENTITY VELOCITY IN TRAFFIC NETWORK

Abstract:

The quantitative data are the basic step for processing, storage and a graphical representation of GPS data of traffic entities within the traffic system. For a good estimate of average speeds, travel time and the level of reliability during the day a sufficient number data are needed for each road segment. Also, adequate information-communication system is required, that will filter, process and store the data in the relational database and display the results to the end user in the developed application. Mathematical methods for filtering data with appropriate formulations are presented together with processing and graphical presentation of the results. After processing, the processed data are be stored in the relational database PostgreSQL together with processing and graphical presentation of the results. GUI application in Python programming language is implemented which is used to retrieve and display the data from database on interactive map. This creates an expandable information-communication system for informing the user about the traffic state and provides the possibility of optimizing the route for the fleet of vehicles.

Keywords: database, SQL, Python, GPS, processing and storage of data, level of reliability, PostgreSQL, interactive map application

SADRŽAJ

1. Uvod **1**

2. Načini prikupljanja podataka o prometnim entitetima **4**

3. Filtriranje, obrada i pohrana sirovih podataka **8**

3.1. Filtriranje podataka 8

3.1.1. Metoda interkvartila – IQR 10

3.1.2. Apsolutna devijacija medijana – MAD 13

3.1.3. Statičko filtriranje podataka 15

3.2. Obrada podataka 15

3.3. Pohrana podataka u bazu podataka 18

4. Izrada programske podrške za filtriranje, obradu i pohranu podataka u relacijsku bazu podataka **23**

4.1. Programski alat – Python 23

4.2. Programski alat – PostgreSQL 25

4.3. Razvojno okruženje i biblioteke 30

5. Obrada i pohrana podataka programskom podrškom te stvaranje aplikativnog rješenja **35**

5.1. Obrada, pohrana i grafički prikaz brzina entiteta u prometnoj mreži programskom podrškom 35

5.2. Obrada, pohrana i grafički prikaz geografskih podataka programskom podrškom 40

5.3. Stvaranje aplikativnog rješenja 42

6. Izračun razine pouzdanosti **49**

7. Studij slučaja **54**

7.1. Studija slučaja – prikupljanje podataka 54

7.2. Studija slučaja – analiza podataka 56

7.3. Studija slučaja – pitanja vrijednosti i pouzdanosti 59

8. Zaključak **64**

Literatura **66**

Popis kratica i akronima **70**

Popis slika **73**

Popis kôdova **75**

1. Uvod

Podaci su danas neizbježan dio svijeta i svaki dan ih se stvara sve više, a time se postavlja sljedeće pitanje. Što raditi sa svim generiranim podacima te mogu li se stvoriti nove informacije od tih podataka? Kako bi se stvorile nove informacije od sirovih podataka potrebno je poznavati područje na koje se ti podaci odnose. Uz to, potrebna je osnova deskriptivne statistike i poznavanje programskog jezika, kako bi se mogla stvoriti programska podrška koja će filtrirati, obrađivati, pohranjivati i grafički prikazivati novostvorene informacije. Upravo te informacije dobivene iz velike količine sirovih podataka, koje u sirovom obliku imaju minimalnu vrijednost, mogu optimizirati sustave, organizacije, povećati prihode kao i predviđati moguće događaje i kretanje tržišta te time spriječiti potencijalne gubitke. Podaci iz kojih se dobivaju informacije su uvijek imali veliku moć u društvu, a danas kada živimo u dobu gdje je sve oko nas vođeno informacijama i svuda oko nas se nalaze podaci, moguće je donositi veliki broj novih zaključaka i zakonitosti povezujući više skupova različitih podataka.

Svrha ovog istraživanja je opisati načine na koji su prikupljeni sirovi podaci o prometnim entitetima, te postupak dobivanja informacija na temelju sirovih podataka. Izradom programske podrške omogućuje se obrada velike količine sirovih podataka te se dobivene informacije pohranjuju u bazu podataka. Upotrebom upitnog jezika za relacijsku bazu podataka, uspostavlja se komunikacija između programske podrške i relacijske baze podataka. Dobivene informacije se statističkom analizom obrađuju za dobivanje dodatnih informacija o brzinama entiteta i izračun razine pouzdanosti dobivenih rezultata. Studijem slučaja se upotpunjuju i naglašavaju moguća poboljšanja te mogućnosti za proširivanje dobivenih rezultata.

Cilj istraživanja je izrada pravovaljanog informacijsko-komunikacijskog sustava koji će pružati nove informacije krajnjim korisnicima s određenom pouzdanošću brzine entiteta u prometnoj mreži.

Naziv diplomskog rada je **Izrada informacijsko-komunikacijskog sustava za praćenje prometnih entiteta i procjenu parametra brzine entiteta u prometnoj mreži** i podijeljen je na osam povezanih poglavlja:

1) Uvod,

2) Načini prikupljanja podataka o prometnim entitetima,

3) Filtriranje, obrada i pohrana sirovih podataka,

4) Izrada programske podrške za filtriranje, obradu i pohranu podataka u relacijsku bazu podataka,

5) Obrada i pohrana podataka programskom podrškom te stvaranje aplikativnog rješenja,

6) Izračun razine pouzdanosti,

7) Studija slučaja,

8) Zaključak.

U drugom poglavlju daje se pregled prijašnjih istraživanja vezana uz ovaj diplomski rad te se opisuju načini prikupljanja podataka o prometnim entitetima. Detaljno se opisuje na koji način se prikupljaju podaci, u kojem periodu te koliko je prometnih entiteta sudjelovalo u prikupljanju podataka. Na kraju se opisuju dva skupa podataka i njihovi atributi, jedan koji sadržava brzine za svaki cestovni segment, a drugi skup podataka koji istim tim cestovnim segmentima dodjeljuje geografske koordinate kako bi ih se moglo pridružiti digitalnoj karti.

Treće poglavlje teoretski objašnjava filtriranje, obradu i pohranu sirovih podataka. Kod filtriranja se pojašnjavaju matematičke metode koje definiraju pronalazak podataka koji kvare skup podataka. Također, objašnjava se koja je metoda bolja ovisno o podacima, tj. o distribuciji podataka. Objašnjavaju se tri metode: metoda interkvartila, medijan apsolutne devijacije i statičko filtriranje podataka; njihova matematička formulacija te koje su prednosti, a koji nedostaci pojedine metode. Zatim će se opisati obrada sirovih podataka, odnosno koraci potrebni za kvalitetnu obradu prije filtriranja podataka. Na kraju poglavlja se opisuje pohrana podataka u relacijsku bazu podataka, gdje se opisuju dva načina pohrane s prednostima i nedostacima pojedinog načina pohrane. Prilikom pohrane najbitnija su tri faktora, brzina dohvaćanja, količina stvorenih tablica i potrebna količina prostora za pohranu.

Četvrto poglavlje opisuje postupak izrade programske podrške za filtriranje, obradu i pohranu podataka, što je nastavak na prethodno poglavlje, gdje se konkretno navode alati koji su korišteni. Programski jezik koji se koristi za filtriranje i obradu podataka je Python te će se opisati razvojno okruženje Pythona, što ga izdvaja od ostalih programskih jezika te koje su mu njegove prednosti i mane. Zatim je opisan programski alat koji će se koristiti kao relacijska baza podataka, što ga razlikuje od ostalih programskih alata, prednosti, nedostaci i razlog zašto je odabran. Nakon što su objašnjeni programski alati, stvara se razvojno okruženje i odabire se platforma za stvaranje razvojnog okruženja te se opisuje izgled i način upotrebe. Također, potrebno je naglasiti koje će se Python biblioteke koristiti ovisno o skupu podataka.

U petom poglavlju se nakon teoretske osnove i stvaranja razvojnog okruženja konkretno izvršava filtriranje, obrada i pohrana podataka te se stvara aplikativno rješenje. Učitavaju se

sirovi podaci, filtriraju i obrađuju u Pythonu te se pohranjuju u PostgreSQL relacijsku bazu podataka. Prvo se odrađuju sirovi podaci za prvi skup podataka, tj. za brzine pojedinog cestovnog segmenta, a onda se isto to napravi i za drugi skup podataka koji daje geografska obilježja cestovnim segmentima. Zatim se stvori grafičko korisničko sučelje Qt dizajnerom i stvori aplikacija u Pythonu s interaktivnom kartom, gdje se pritiskom lijeve tipke miša mogu dohvatiti i grafički prikazati podaci za cestovni segment, ovisno o odabranom mjestu.

U šestom poglavlju izračuna se razina pouzdanosti za definiranu rutu koja se sastoji od definiranog broja cestovnih segmenata te se prvo dobije srednja brzina rute sa standardnom devijacijom oko srednje brzine. Nakon toga izračuna se s kojom sigurnošću se može garantirati da će srednja brzina biti upravo ta brzina tokom cijele rute, a to se izražava u postocima. Osim toga, izračuna se srednje vrijeme putovanja te kolika je standardna devijacija srednjeg vremena putovanja. Izračuni vezani uz rutu su iznimno korisni, a pogotovo prosječno vrijeme putovanja po dobu u danu.

Predzadnje, sedmo poglavlje objašnjava što je studija slučaja i koji elementi studija slučaja se koriste u diplomskom radu. Objašnjava se studija slučaja za prikupljanje podataka te koliko količina podataka utječe na krajnje dobivene rezultate. Također, objašnjava se kakvi su rezultati kada postoji manja količina podataka i kako prikupiti veću količinu podataka. U analizi podataka analiziraju se korištene metode te koje metode mogu još poboljšati dobivene rezultate. Na kraju evaluira se vrijednost dobivenih rezultata te koliko su rezultati pouzdani. Vrijednosti dobivenih rezultata najviše je vidljiva u stvorenoj aplikaciji, a pouzdanost je prethodno objašnjena u poglavlju šest te kako poboljšati razinu pouzdanosti.

Na kraju je u Zaključku dan pregled rada, s osvrtom na mogućnosti izrađene aplikacije i budućim poboljšanjima.

2. Načini prikupljanja podataka o prometnim entitetima

Za izradu informacijsko-komunikacijskog sustava za praćenje prometnih entiteta i procjenu parametra brzine entiteta u prometnoj mreži, prvo je potrebno prikupiti podatke nad kojima će se izvoditi operacije i donositi zaključci. Prikupljanje podataka o prometnim entitetima nije potrebno napraviti, jer će se koristiti povijesni podaci dobiveni u projektu Sustav za optimizaciju ruta u dinamičkom transportnom okruženju (engl. *System for Route Optimization in a Dynamic Transport Environment*, SORDITO). Te povijesne podatke čine Globalni pozicijski sustav (engl. *Global Positioning System*, GPS) tragovi (slika 2.1) vozila koji su prikupljeni pokretnim osjetilima iz navigacijskih uređaja, ugrađenih u prometne entitete. Dok se prometni entitet kreće prometnicom sirovi podaci se pohranjuju na server, a ti podaci sadrže informaciju o poziciji vozila, vremenskom trenutku zapisa, brzini vozila, smjeru kretanja, broju raspoloživih satelita itd.

Slika 2.1: Primjer GPS tragova, [1]

Važno je naglasiti kako će se za diplomski rad koristiti već postojeći podaci, dakle nije ih potrebno prikupiti već ih je potrebno obraditi i pohraniti te dobiti nove informacije koje će informacijsko-komunikacijski sustav prikazivati korisnicima. Potrebno je obraditi podatke zbog GPS-a koji unosi moguće greške te je potrebno filtrirati podatke i odbaciti one koje kvare set podataka. Filtriranje i pohrana podataka će se detaljno obraditi u idućem poglavlju. Kroz projekt SORDITO prikupljeno je oko sedam milijardi GPS zapisa koji opisuju kretanje vozila

u Republici Hrvatskoj (od kolovoza 2009. do listopada 2014. godine). Podaci su se prikupljali s oko 4.200 različitih vozila, ponajviše transportnih vozila, a neka od tih transportnih vozila su imali mogućnosti dostići i velike brzine što će se prikazati u poglavlju 3. S obzirom na to kako su podaci prikupljeni putem GPS uređaja, a većina GPS uređaja odašilje signal svake sekunde, jasno je kako je pohrana podataka izniman problem zbog velike količine pohranjenog prometa. Kako bi se smanjila količina pohranjenih podataka, u projektu SORDITO zapisi s GPS uređaja su pohranjeni otprilike svakih 5 *min*, tj. 100 m. Količina već tako smanjenog seta podataka iznosi 520 GB. Digitalna karta Republike Hrvatske sastoji se od 448.393 cestovnih segmenata u prosjeku duljine 80 m, a prevladavaju cestovni segmenti kraćih duljina. Cestovni segment je dio prometnice koji se nalazi između dva susjedna raskrižja, a još se može zvati i linkom koji će se češće koristiti u radu. Primjer linkova koji su izdvojeni različitom bojom moguće je vidjeti na slici 2.2.

Slika 2.2: Prikaz izdvojenih cestovnih segmenata, [1]

Za postizanje cilja diplomskog rada, potrebno je obraditi dva različita skupa podataka koji su u tekstualnom obliku (".txt"). Jedan sirovi skup podataka sastoji se od više tekstualnih datoteka za svaki pojedini link na području grada Zagreba, točnije njih 14.154. Količina seta podataka za grad Zagreb iznosi 9,35 GB. Primjer podataka za jedan link prikazan je na slici 2.3, a atributi su prikazani u tablici 2.1.

```
epoch_UTC;v1;v2;v3;v4
1364762871;59.0;59.6;59.6;59.6
1364770182;70.7;72.2;72.2;72.2
1375404798;69.5;77.0;77.0;77.0
1375456980;63.5;68.3;68.3;68.3
1375485308;65.2;57.7;57.7;57.7
1375489216;68.0;60.8;60.8;60.8
1375541058;70.7;59.2;59.2;59.2
1375546714;72.7;73.1;73.1;73.1
1375660850;64.0;65.0;65.0;65.0
1375692492;68.7;75.5;75.5;75.5
1375985349;70.5;70.9;70.9;70.9
1376065706;54.7;57.2;57.2;57.2
1376070326;61.0;61.5;61.5;61.5
1376200444;72.0;71.6;71.6;71.6
1376268635;74.5;74.8;74.8;74.8
1376284173;61.7;63.5;63.5;63.5
1376406245;54.7;56.6;56.6;56.6
1376708541;58.2;42.9;42.9;42.9
1376734830;56.0;57.0;57.0;57.0
1376739070;61.0;61.7;61.7;61.7
1376859018;66.2;67.0;67.0;67.0
1376864610;72.5;73.3;73.3;73.3
1376866622;71.5;120.1;120.1;120.1
```

Slika 2.3: Prikaz sirovih podataka u tekstualnom obliku s vremenom i brzinama svakog GPS zapisa

Tablica 2.1 Opis atributa prvog skupa podataka

Atribut	Primjer	Opis
epoch_UTC	1376866622	UTC u sirovom *epoch*[1] obliku iz kojega se može dobiti točan datum, te vrijeme u danu u sekundnoj rezoluciji
v_1	71.5	Uprosječena GPS brzina svih zapisa jednog zabilježenog prolaska vozila po linku $[km/h]$
v_2	120.1	Uprosječena jednolika brzina po segmentima izračunata na temelju prijeđene udaljenosti između susjednih GPS zapisa kretanja jednog vozila na linku $[km/h]$
v_3	120.1	Jednolika brzina kretanja izračunata na temelju prijeđene udaljenosti između prvog i zadnjeg GPS zapisa kretanja jednog vozila koja piše u podacima i protekloga vremena $[km/h]$
v_4	120.1	Jednolika brzina kretanja izračunata na temelju zračne prijeđene udaljenosti između prvog i zadnjeg GPS zapisa kretanja jednog vozila i protekloga vremena $[km/h]$

Drugi skup podataka sastoji se od jedne tekstualne datoteke i pruža dodatne informacije o pojedinom linku kao što je vidljivo sa slike 2.4, a svi atributi s opisima se nalaze u tablici 2.2.

```
IdLink;Flag;Duljina_m;Srednja_brzina;Ograničenje_brzine;Početna_z.d;Početna_z.š;Završna_z.d;Završna_z.š;Kategorija_linka;Naziv_linka
214675;2;260;38;0;15.947953462600745.7941600974532;15.944648981O944;45.7938085103048;1040;Zagrebačka avenija
214676;2;48;38;0;15.9446489810944;45.7938085103048;15.9440910816193;45.7939805638674;1040;
214677;2;26;38;0;15.9441554546356;45.7936663787004;15.943865776062;45.7937785878919;1040;
214678;1;38;38;0;15.943865776062;45.7937785878919;15.9442520141602;45.7939955250216;1040;
214679;1;41;38;0;15.9442520141602;45.7939955250216;15.9441661834717;45.794362072046;1040;Zagrebačka avenija
214680;1;93;38;40;15.9441661834717;45.794362072046;15.9439408779144;45.7951849239319;1040;Selska cesta
214681;1;56;38;40;15.9439408779144;45.7951849239319;15.9438121318817;45.7956786292316;1040;Selska cesta
214682;1;74;38;40;15.9438121318817;45.7956786292316;15.9436404705048;45.7963368961605;1040;Selska cesta
214683;2;46;38;0;15.9437370300293;45.7941975002108;15.9442520141602;45.7939955250216;1040;
214684;2;63;38;0;15.9442520141602;45.7939955250216;15.944412946701;45.7934344789886;1040;Selska cesta
214685;2;36;38;60;15.944412946701;45.7934344789886;15.9444987773895;45.7931128100489;1040;Selska cesta
214686;2;208;38;60;15.9444987773895;45.7931128100489;15.944949388504;45.7912650478512;1040;Selska cesta
214687;2;341;38;60;15.944949388504;45.7912650478512;15.9459149837494;45.7882800718157;1040;Selska cesta
214688;2;20;38;60;15.9459149837494;45.7882800718157;15.94602227211;45.7881154820209;1040;Selska cesta
214689;2;57;38;60;15.94602227211;45.7881154820209;15.9464406967163;45.7876965239862;1040;Selska cesta
214690;2;417;38;60;15.9464406967163;45.7876965239862;15.9512579441071;45.7865817810229;1040;Selska cesta
214691;2;14;38;60;15.9512579441071;45.7865817810229;15.951429605484;45.7866191888664;1040;Selska cesta
214692;2;69;38;60;15.951429605484;45.7866191888664;15.9520626068115;45.7862675541466;1040;
214693;2;58;38;60;15.9520626068115;45.7862675541466;15.9520620068115;45.78574383875;1040;Jadranski most
214694;2;97;38;60;15.9520626068115;45.78574383875;15.9521055221558;45.7848684748881;1040;Jadranski most
214695;2;340;38;60;15.9521055221558;45.7848684748881;15.95250248909;45.781823298653;1040;Jadranski most
214696;2;199;38;60;15.95250248909;45.781823298653;15.9527921676636;45.7800425042739;1040;Jadranski most
214697;2;172;38;60;15.9527921676636;45.7800425042739;15.9530818462372;45.7785085811673;1040;Jadranski most
214698;2;137;38;60;15.9526205062866;45.7788452996105;15.952513217926;45.7800724340611;1040;Jadranski most
214699;2;183;38;60;15.952513217926;45.7800724340611;15.9522342681885;45.7817110653926;1040;Jadranski most
214700;2;326;38;60;15.9522342681885;45.7817110653926;15.9515261650085;45.7845991293959;1040;Jadranski most
214701;2;160;38;60;15.9515261650085;45.7845991293959;15.951247215271;45.7860206603584;1040;Jadranski most
214702;2;53;38;50;15.951247215271;45.7860206603584;15.9507322311401;45.7862525909179;1040;
214703;2;27;38;50;15.9507322311401;45.7862525909179;15.9503996372223;45.7861927379626;1040;Selska cesta
214704;2;44;38;50;15.9503996372223;45.7861927379626;15.9498310089111;45.7861852563387;1040;Selska cesta
214705;2;230;38;60;15.9498310089111;45.7861852563387;15.9471273422241;45.7869783028865;1040;Selska cesta
214706;2;120;38;60;15.9471273422241;45.7869783028865;15.946033000946;45.7877414124977;1040;Selska cesta
214707;2;87;38;50;15.946033000946;45.7877414124977;15.9450244903564;45.7880556310658;1040;
```

Slika 2.4: Prikaz sirovih podataka u tekstualnom obliku s informacijama za pojedini link

[1] *epoch* odnosno *Unix time* je broj sekundi proteklih od ponoći 1. siječnja 1970. godine po UTC i često se koristi za zapisivanje točnog vremena kod uređaja koji primaju/šalju podatke o vremenu

Tablica 2.2 Opis atributa drugog skupa podataka

Atribut	Primjer	Opis
Link	-214695	Identifikacijski broj linka
Flag	2	Oznaka linka je zastavica (engl. *Flag*) koja definira je li link dvosmjeran (0), pozitivan smjer (1), negativan smjer (2) ili je link zatvoren (3)
Duljina linka	340	Prijeđena udaljenost po prometnici između početne i završne točke $[m]$
Srednja brzina	38	Srednja brzina linka po kojoj prometuju prometni entiteti $[km/h]$
Ograničenje brzine	60	Ograničenje dozvoljene brzine vozila po linku $[km/h]$
Početni X	15.9521055221558	Početna geografska dužina u decimalnim stupnjevima
Početni Y	45.7848684748881	Početna geografska širina u decimalnim stupnjevima
Završni X	15.95250248909	Završna geografska dužina u decimalnim stupnjevima
Završni Y	45.781823298653	Završna geografska širina u decimalnim stupnjevima
Kategorija linka	1040	Kategorija linka koja definira tip cestovnog segmenta, a kategorije su: autocesta - 1010, avenija - 1030, brza cesta - 1040, glavna gradska ulica - 1050, lokalna cesta i gradska ulica - 1060, kvartovska ulica - 1070 ili 1080, itd.
Naziv linka	Jadranski most	Naziv prometnice dodijeljen linku

Naime, to je samo dio podataka koji je prikupljen putem projekta SORDITO, a ti su podaci neupotrebljivi bez dodatne obrade podataka. Podaci koji su vidljivi iz slika 2.3 i 2.4 su obrađivani i prije ovakvog oblika, te su sortirani svi zapisi po linkovima. Izračunate su brzine na četiri različita načina prikazana u tablici 2.1 i smješteni u tekstualnu datoteku za svaki link, [1, 2, 3].

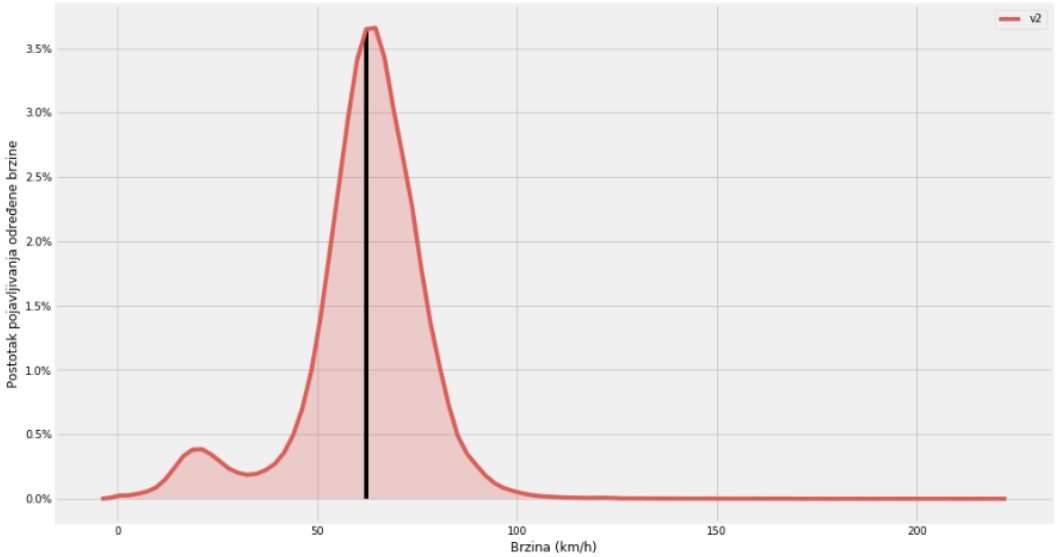
3. Filtriranje, obrada i pohrana sirovih podataka

Poglavlje opisuje tri radnje: filtriranje, obradu i pohranu sirovih podataka. Teorijski se opisuje principi i metodologija prema kojima je izvršeno filtriranje, obrada i pohrana sirovih podataka te koji su razlozi primjene izabrane metodologije.

3.1. Filtriranje podataka

Pod pojmom filtriranje podataka smatra se proučavanje skupa podataka nad kojim se provodi filtriranje podataka i uočavaju zakonitosti unutar toga skupa te odstraniti podatke koji "kvare" skup podataka, tj. pronaći izvanpopulacijske podatke (engl. *outlier*). Izvanpopulacijski podatak je podatak koji se nalazi u skupu, ali odskače od većine podataka te tako kvari skup podataka. Skup podataka koji se proučava je prikazan na slici 2.3 te će se promatrati samo brzina v_2, zbog čega je tekst crvene boje u tablici 2.1, jer se kroz projekt SORDITO dokazala kao najpouzdanija, a brzine v_3 i v_4 potvrđuju u većini slučajeva kako je to istina jer su iste vrijednosti kao i za brzinu v_2. Grafički prikaz podataka za brzinu v_2 prikazan je na slici 3.1. Slika prikazuje distribuciju sirovih podataka, na kojoj su vidljiva dva karakteristična brijega, dakle bimodalna distribucija. Na x osi je brzina u kilometrima na sat, a na y osi su postotni udijeli brzine u decimalnom zapisu. Vidljivo je kako su brzine oko 60 kilometara na sat (km/h) najučestalije. Crna linija predstavlja prosjek svih brzina, a to je ujedno i očekivana brzina za link 214695 koji je opisan tablicom 2.2. Prvi zaključak koji se može donijeti je taj kako se radi o brzinama prometnih entiteta i kako se vrijednosti kreću od minimalnih nula do realno maksimalne brzine cestovnog vozila 400 km/h. Vjerojatnost te brzine je iznimno mala, ali je teorijski moguća. Realne brzine koje su uočene u skupovima podataka su do 250 km/h. Međutim, postavlja se pitanje kako te velike brzine utječu na cjelokupni skup podataka kada se računa prosječna brzina vozila. S druge strane, postavlja se pitanje kako tretirati brzine od 0 km/h. S obzirom na to kako se u diplomskom radu sva pažnja usmjerava prema dobivanju prosječnih brzina vozila na pojedinim cestovnim segmentima u točno određenom vremenu, brzine od 0 km/h, iako su pokazatelj zastoja u prometu, u nekim slučajevima mogu "pokvariti" skup podataka. Brzine od

0 km/h koje se pojavljuju u malim skupovima podataka mogu pokvariti rezultate, pogotovo ako nisu zabilježene veće brzine za ravnotežu manjih brzina. Može se umanjiti prosječna brzina, dok je stvarna situacija na linku drugačija. Primjer su linkovi koji se nalaze u stambenim područjima, gdje su brzine znatno niže i pojavljuju se nule zbog stajanja, a realno je moguća znatno veća brzina. Zakonitost i logično razmišljanje dovodi do zaključka ako velike brzine kvare skup podataka i odstrane se iz skupa, a ostave se male brzine u skupu podataka, može doći do nerealnih rezultata, odnosno manje prosječne brzine vozila od realnog stanja. Potreban je balans između malih i velikih brzina prometnih entiteta, a upravo prva metoda za filtriranje podataka koristi tu logičku zakonitost i predlaže kako korigirati skup podataka. Metode koje se korite su: metoda interkvartila (engl. *Interquartile range*, IQR), zatim apsolutna devijacija medijana (engl. *Medain absolute deviation*, MAD) i statično filtriranje podataka.



Slika 3.1: Distribucija brzina na razmatranom linku

3.1.1. Metoda interkvartila – IQR

Prije samog opisa metoda, matematički se opisuju srednja vrijednost i medijan, jer se koriste u radu. Srednja vrijednost ili aritmetička sredina najčešće se računa kod kvantitativnih podataka i dobiva se zbrojem numeričkih vrijednosti podataka koji se podjeli s brojem podataka u skupu. Za dobivanje srednje vrijednosti koristi se izraz (3.1):

$$\overline{x} = \frac{\sum_{i=1}^{n} x_i}{n} \tag{3.1}$$

gdje je:

- $\overline{x}$ – oznaka za aritmetičku sredinu, tj. srednju vrijednost,
- n – predstavlja broj vrijednosti/članova u skupu podataka,
- x_i – pojedinačni član skupa od prvog ($i = 1$) do zadnjeg člana ($i = n$).

Medijan je također srednja vrijednost, ali se dobije tako što se vrijednosti poredaju od najmanje prema najvećoj, a medijan je vrijednost koja se nalazi točno u sredini poredanog skupa podataka. S obzirom na to da skup podataka može imati paran i neparan broj vrijednosti, za neparan skup vrijednosti izraz je:

$$x_{\frac{n+1}{2}} \tag{3.2}$$

gdje je n ukupan broj u skupu podataka. Ako se primjerice skup podataka sastoji od 23 vrijednosti, medijan će biti X_{12}, odnosno 12-ta vrijednost u poredanom nizu. Kada je veličina skupa podataka parna, odnosno kada je n paran, izraz je:

$$\frac{x_{\frac{n}{2}} + x_{\frac{n+1}{2}}}{2} \tag{3.3}$$

Za veličinu skupa podataka od 24 poredane vrijednosti uzima se vrijednost na 12-oj (X_{12}) i 13-oj poziciji (X_{13}) te se njihova srednja vrijednost uzima kao medijan, [4].

Medijan dijeli skup podataka točno na pola, na lijevu i desnu stranu od medijana, a moguća je i daljnja podjela te se dobiju kvartili. Kvartili dijele skup podataka na četvrtine, što se može iskoristiti za odbacivanje podataka koji kvare skup podataka. Iz slike 3.2 na kojoj je *box plot*, koji se koristi za grafički prikaz podataka i detekciju izvanpopulacijskog podatka, vidljive su granice koje definiraju metodu interkvartila. Metoda interkvartila definira tri kvartila:

- Prvi ili donji kvartil (Q_1) je vrijednost u skupu podataka koja se nalazi točno na sredini najmanje vrijednosti i medijana cijeloga skupa podataka, odnosno 25 %-tna vrijednost u skupu podataka,
- Drugi kvartil (Q_2) je ujedno i medijan skupa podataka, dakle on je 50 %-tna vrijednost skupa podataka,

– Treći ili gornji kvartil (Q_3) je vrijednost u skupu podataka koja se nalazi točno na sredini najveće vrijednosti i medijana cijeloga skupa podataka, odnosno 75 %-tna vrijednost u skupu podataka.

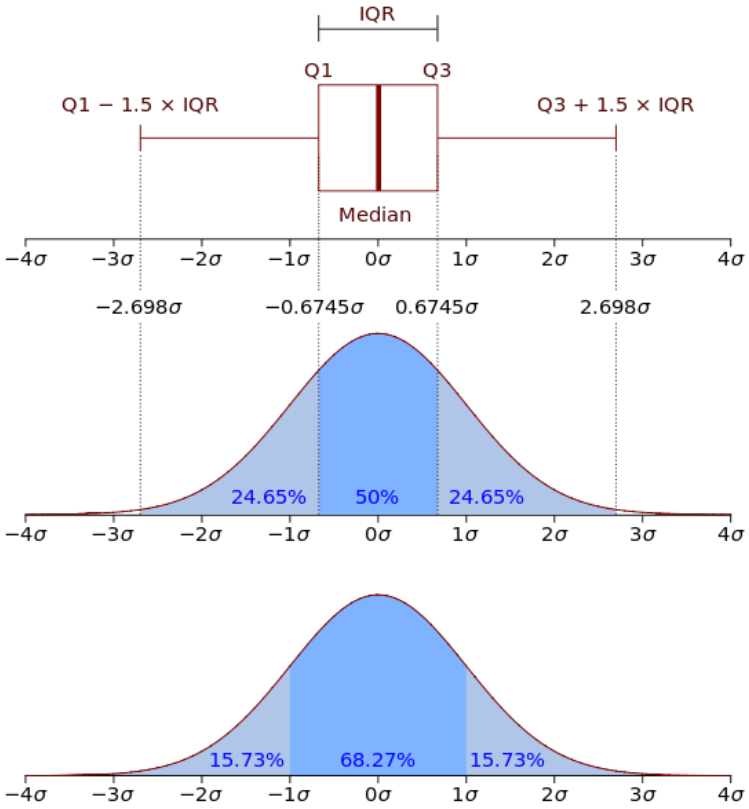
Sukladno tomu, prostor između Q_1 i Q_3 je IQR, što je prikazano izrazom (3.4):

$$IQR = Q_3 - Q_1 \tag{3.4}$$

Međutim, odbacivanje 50 % podataka u većini slučajeva nije dobar odabir, tako da se za odbacivanje izvanpopulacijskih podataka koristi sljedeći izraz (3.5):

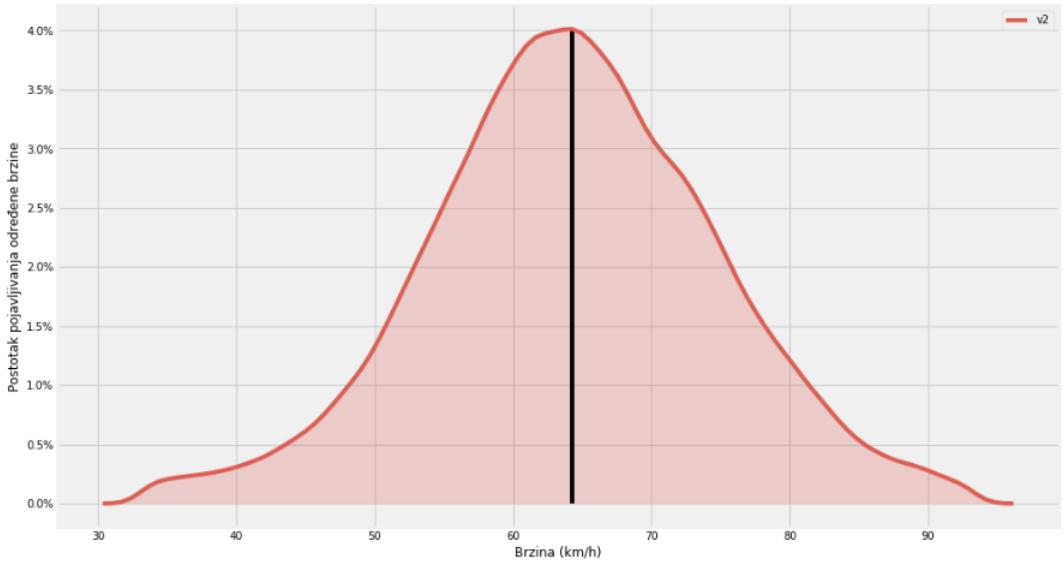
$$\begin{aligned} Outlier &< Q_1 - 1.5 \cdot IQR \\ Outlier &> Q_3 + 1.5 \cdot IQR \end{aligned} \tag{3.5}$$

kao što je i vidljivo sa slike 3.2 dodavanjem ove dvije zakonitosti iz skupa podataka odbacuju se samo ekstremi, a ostaje preko 99 % podataka.



Slika 3.2: Grafički prikaz metode interkvartila na normalnoj distribuciji podataka i prikaz putem *box plota*, [5]

Za pronalaženje izvanpopulacijskih podataka jedna od najkorištenijih metoda je upravo metoda interkvartila, jer se mogu eliminirati vrijednosti koje odskaču od ostalih vrijednosti u skupu. Međutim, postoje i robusnije metode koje također pronalaze izvanpopulacijske podatke, a za skup podataka koji se koristi u radu i nad kojim se radi analiza, testiranjem su se dokazale efikasnijim od metode interkvartila. Razlog je taj što kod velikog skupa podataka (primjerice 200.000 vrijednosti u skupu) u jedan posto podataka koji se odbacuju može biti velik broj vrijednosti koje su valjane, ali se izbacuju iz skupa. Nadalje, ako se distribucija podataka ne ravna prema normalnoj distribuciji, postoji asimetričnost podataka u odnosu na medijan. Upravo je to vidljivo iz slike 3.1, gdje se zapravo ne radi o normalnoj distribuciji, jer se s lijeve strane medijana (koji je prikazan crnom linijom) nalazi potencijalno još jedna normalna distribucija. Primjenom metode interkvartila na ovaj skup podataka 92,3 % podataka se zadržava, a 7.7 % odbacuje kao što je prikazano na slici. Razlog zašto se ne zadržava 99 % podataka je upravo taj što nije klasična normalna distribucija. Na slici 3.3 su prikazane filtrirane brzine i kreću se od 33 km/h do 93 km/h, što je iznimno loše, jer se odbacuju male brzine koje dobro opisuju gužve i velike brzine koje su valjane (do 120 km/h).

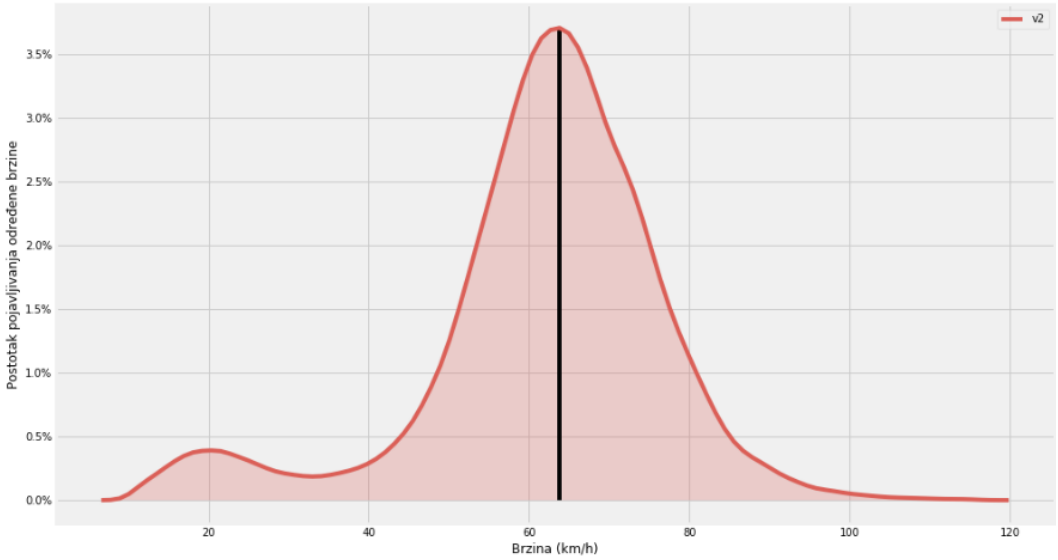


Slika 3.3: Grafički prikaz filtriranih podataka metodom interkvartila s normalnim granicama, prema izrazu (3.5)

Nešto bolji rezultati se dobiju proširenjem granica pomoću izraza (3.6):

$$Outlier < Q_1 - 3 \cdot IQR$$
$$Outlier > Q_3 + 3 \cdot IQR \tag{3.6}$$

te se dobiju rezultati prikazani na slici 3.4. Donja granica je 10,5 km/h, gornja granica 116 km/h, a sve ostale manje ili veće brzine su odbačene. Jedan posto podataka je odbačeno, što je prihvatljivo, [6].



Slika 3.4: Grafički prikaz filtriranih podataka metodom interkvartila s proširenim granicama, prema izrazu (3.6)

Kada se koristi filtriranje podataka proširenjem granica kod manjeg skupa podataka za pojedine linkove uključiti će se brzine i od 0 km/h. Zbog raznolikosti linkova, odnosno broja podataka po pojedinom linku, iznimno je teško pronaći najbolju metodu koja će kvalitetno filtrirati veliki i mali skup podataka. Metoda apsolutne devijacije medijana je testiranjem dokazana kvalitetnijom od metode interkvartila te će se sljedeća objasniti.

3.1.2. Apsolutna devijacija medijana – MAD

Apsolutna devijacija medijana je metoda za filtriranje podataka te je nešto robusnija od metode interkvartila. MAD je mjera disperzije, tj. varijacije podataka, kao što je to i metoda interkvartila. Prema izvoru [7], MAD posjeduje najbolju moguću točku loma, a to je do 50 %, što je duplo više od metode interkvartila koja ima točku loma 25 %. Točku loma određuje procjenitelj koji se definira kao proporcija netočnih promatranja, tj. to su proizvoljne vrijednosti koje imaju velike vrijednosti. Primjer je srednja vrijednost, čija je točka loma nula jer dodavanjem iznimno velike vrijednosti na postojeći skup podataka može značajno promijeniti srednju

vrijednost skupa podataka. Zbog toga je srednja vrijednost loša za pronalaženje *outliera*. Točka loma procjenitelja ne može biti veća od 50 %, jer ako je više od polovice podataka u skupu kontaminirano, teško je donijeti statističke zakone, distribuciju podataka, itd. Prvi korak MAD metode je poredati vrijednosti podataka od najmanjeg prema najvećem i pronaći medijan kao što je objašnjeno u prijašnjem potpoglavlju. Zatim se računa medijan od apsolutne razlike svake vrijednosti skupa s dobivenim medijanom, što je prikazano u izrazu (3.7):

$$MAD_n = b \cdot med_i \left| x_i - med_j x_j \right| \tag{3.7}$$

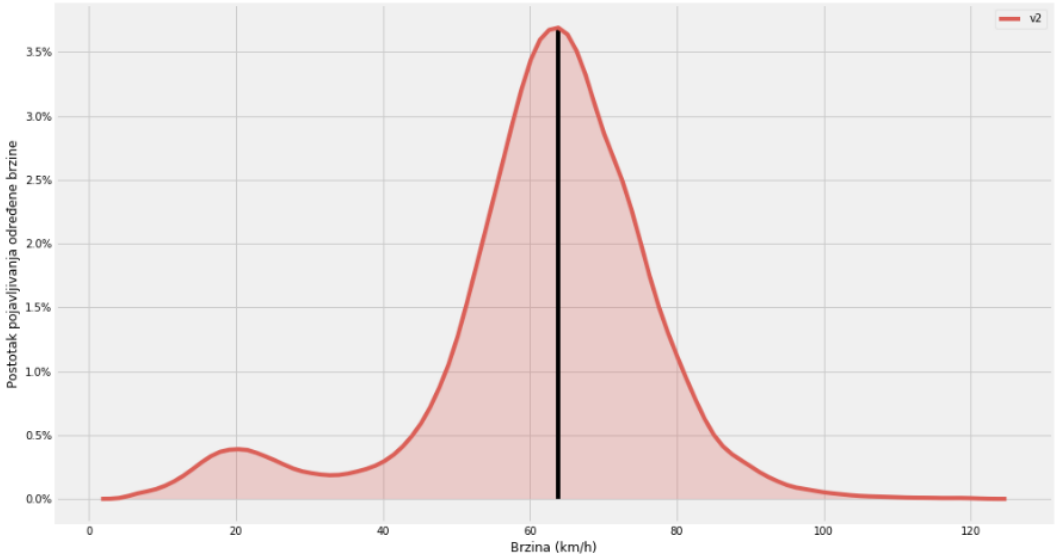
gdje je:

- b – konstanta koja se koristi kada je u pitanju normalna distribucija, za bilo koju drugu distribuciju, izraz je bez b konstante,
- x_i – pojedinačni član skupa od prvog ($i = 1$) do zadnjeg člana ($i = n$),
- ($med_j x_j$) – oznaka za izračun medijan nad originalnim skupom podataka,
- med_i – medijan koji se dobije apsolutnom razlikom svih članova skupa i izračunatoga medijana,
- MAD_n – brojčana vrijednost koja određuje maksimalnu donju i gornju granicu, a vrijednosti koje su preko granice se odbacuju iz skupa podataka.

Kako bi otkrili izvanpopulacijske podatke potrebno je dobiti apsolutnu razliku između svake vrijednosti skupa podataka (x_i) i medijana ($med_j x_j$) te podijeliti s izračunatim MAD, kao što je prikazano u izrazu (3.8):

$$\frac{\left| x_i - med_j x_j \right|}{MAD_n} \tag{3.8}$$

Izračun je najlakše shvatiti putem primjera, pa je tako izračun dan na primjeru skupa 1, 1, 2, 3, 3, 4, 4, 5, 5.5, 6, 6.4, 6.5, 7, 7.2, 7.5, 8, 10, 14, 56, 80. Izračuna se MAD prema izrazu (3.7) i dobije se vrijednost 2 te se pomnoži s konstantom b. S obzirom na to da se radi o normalnoj distribuciji dobije se rezultat MAD = 2.965. Nakon toga se primjeni drugi izraz (3.8) kako bi se uočili *outlieri* i dobiju se vrijednosti 2, 2, 1.5, 1.5, 1, 1, 0.5, 0.25, 0, 0, 0.25, 0.5, 0.6, 0.75, 1, 2, 2.5, 25, 37. Sve vrijednosti veće od izračunatog MAD-a (2,965) se odbacuju iz skupa podataka, a to su u ovom slučaju vrijednosti 56 i 80 iz izvornog skupa podataka. Kada se primjeni MAD metoda na skup podataka za pojedine cestovne segmente i brzine po linkovima dobiju se bolji rezultati, zbog robusnosti MAD metode i prilagodljivosti pojedinom skupu podataka. Kada se filtriraju sirovi podaci sa slike 3.1 pomoću MAD-a, minimalna brzina je 5,6 km/h, a najveća 121 km/h, što je prikazano slikom 3.5. Brzine dobivene metodom MAD su bolje od metode interkvartila jer postoje još manje brzine koje opisuju gužvu, a i uključuju se brzine do 120 km/h koje su prihvatljive za skup podataka.



Slika 3.5: Grafički prikaz filtriranih podataka pomoću metode apsolutna devijacija medijana

Postoji jedna mana MAD metode, a to je pojavljivanje 0 km/h kod manjih skupova podataka za pojedine linkove gdje postoji manji broj podataka, što je prihvatljivo. Postoji samo jedan način koji može garantirati filtriranje prema željenoj minimalnoj i maksimalnoj brzini, a to je statičko filtriranje podataka.

3.1.3. Statičko filtriranje podataka

Kao što i samo ime kaže, postavi se donja i gornja granica i uzimaju se svi podaci između statičko postavljenih granica. Kao minimalna donja granica uzima se brzina od 3 km/h, a maksimalna gornja granica 120 km/h. Time će se prema provedenoj analizi dobiti kvalitetni rezultati za većinu linkova, međutim, kako su sustavi dinamični, skup podataka je isto dinamičan kada mu se dodaju novi podaci. Sukladno tome, potrebno je koristiti dinamičku filtraciju podataka prema statističkim matematičkim metodama. Mogući su linkovi s iznimno malim brzinama i čestim zastojima, odnosno česta pojava brzine 0 km/h te je tada odabir statične metode iznimno loš. To je i glavni razlog uporabe dinamičkih metoda za filtriranje podataka zbog različitosti linkova, odnosno skupova podataka. MAD se pokazao kao metoda koja je najbliže željenim rezultatima, a ponekad su ovisno o skupu i brzine od 0 km/h dobar pokazatelj.

3.2. Obrada podataka

Filtrirane podatke je potrebno obraditi jer su i dalje u sirovom obliku te nije moguće donijeti zaključke proučavajući podatke sa slike 2.3. Prvi korak je pretvorba *epoch* UTC vremena u oblik čitak ljudima s godinom, mjesecom, danom, satima, minutama i sekundama. Kako se izvodi ta pretvorba razlikuje se ovisno o programskom jeziku u kojem se radi. Drugi je korak grupiranje

podataka prema sljedećim vremenskim razdobljima:

- prema **mjesecima** u godini, grupiraju se zabilježeni podaci za pojedini link te je moguće za pojedini mjesec izračunati: broj podataka, prosječnu brzinu po mjesecu, medijan, standardnu devijaciju, itd., kao što je prikazano na slici 3.7a (tablica vrijednosti slika 3.6a). Vidljive su sitne razlike između mjeseci te povećanje brzine tijekom ljeta. Zbog manjeg broja podataka iskoristivost dobivenih informacija je mala te je potrebno kombinirati različita vremenska razdoblja,
- grupiranje prema **danima u tjednu** donosi korisnije informacije za pojedini link, jer je vidljiva razlika u brzinama za vikend i tijekom radnih dana prema slici 3.7b (tablica vrijednosti slika 3.6b). Kombiniranjem mjeseci i dana u tjednu, mogu se dobiti još konkretniji zaključci, odnosno realniji podaci uz određeniji vremenski interval. Mogu se proučavati brzine za prvi dan u mjesecu ili zadnji dan u mjesecu, točno određeni datumi, itd.,
- vrlo korisno grupiranje je prema satima u danu, tj. još bolje grupiranje je brzina prema **minutnim intervalima** kao što je prikazano na slici 3.7c (tablica vrijednosti slika 3.6c). Kada korisnik želi znati prosječnu brzinu pojedinog linka, najčešće ga zanima prosječna brzina u trenutku koji gleda, tj. sada ili u sljedećih nekoliko sati. Za pružanje takvih informacija, najbolje je ako ima dovoljno podataka za raspodjeliti minutne intervale (2.5 min, 5 min, 10 min, 15 min), jer se na taj način dobiva veći uzorak za pojedini interval i točniji rezultati. U radu se koriste 5 i 15 minutni vremenski intervali,
- s obzirom na količinu podataka za pojedini link, kombinacija određenog dana s minutnim intervalima kvalitetnija je od prijašnjih navedenih kombinacija, jer je dovoljan broj podataka po pojedinom razmatranom minutnom intervalu, kao što je vidljivo sa slike 3.7d (tablica vrijednosti slika 3.6d) i stupca *count*, odnosno broj vrijednosti u razmatranom minutnom vremenskom intervalu (5 ili 15 min) iz skupa. S obzirom na to kako se dani u tjednu (za većinu zaposlenika) dijele na radne dane i neradne dane, za većinu linkova će biti vidljiva razlika u brzinama, manje brzine za radne dane i veće brzine za neradne dane. Zato je logično koristiti dane u tjednu zajedno s minutnim intervalima kako bi se dobili vrlo dobri rezultati za većinu linkova. Postoje linkovi koji ne posjeduju dovoljno podataka za 15 minutne intervale, a onda sigurno niti za 5 minutne intervale. O tim slučajevima će se kasnije raspravljati u radu.

Mjesec	count	mean	median	std
1	19946	61.253028	63.0	16.888863
2	18665	59.957273	62.1	17.078709
3	21033	60.937375	63.1	17.447066
4	19805	61.499470	63.1	17.079819
5	22048	63.082252	64.0	16.347225
6	21418	65.023910	65.0	15.790036
7	19251	64.024388	64.2	16.073839
8	13195	65.742311	65.6	14.363185
9	20864	60.816886	62.4	18.247838
10	21223	60.268935	62.7	19.215965
11	18942	57.782995	61.8	20.405334
12	22602	59.831736	62.3	18.372996

(a) Mjeseci na Jadranskom mostu u Zagrebu

Dani	count	mean	median	std
Ponedjeljak	30295	59.060238	62.0	18.543588
Utorak	31408	59.333173	61.8	18.277362
Srijeda	32698	59.436057	61.9	18.150836
Četvrtak	37160	60.124820	62.2	18.168101
Petak	36465	59.726072	61.9	17.977691
Subota	38151	66.036067	65.7	14.575732
Nedjelja	32815	66.773936	66.5	15.070075

(b) Dani u tjednu na Jadranskom mostu u Zagrebu

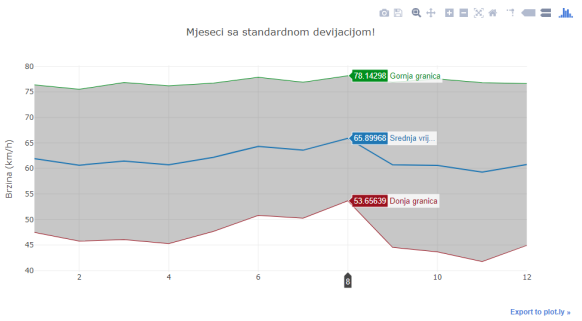
	count	mean	median	std
06:30:00	1713	68.124869	66.40	12.936409
06:45:00	1678	67.325030	66.30	12.778311
07:00:00	2215	61.409616	61.00	14.561357
07:15:00	2362	46.733362	47.00	21.891399
...	...	...	...	...
16:30:00	3054	30.636379	21.70	23.470633
16:45:00	2703	36.749131	27.50	24.159510
17:00:00	2572	44.179821	50.00	23.099289
17:15:00	2335	49.345225	54.30	21.233539
17:30:00	2136	55.481461	58.40	16.636938
17:45:00	1940	57.982113	59.30	15.221169
18:00:00	1896	59.760812	60.10	14.392671
18:15:00	1748	60.137529	60.50	13.512985
18:30:00	1798	59.970245	60.00	13.308109
18:45:00	1821	61.367271	61.30	12.458592
19:00:00	1915	60.797232	60.40	12.256189

(c) Svi dani, 15 minutni interval na Jadranskom mostu u Zagrebu

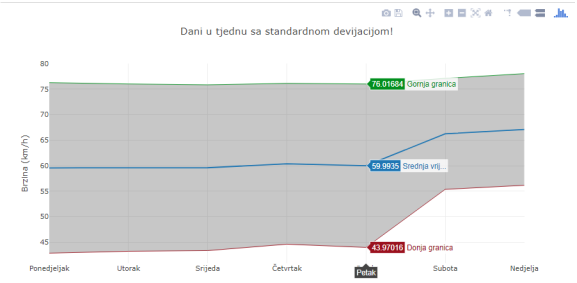
Ponedjeljak	count	mean	median	std
06:30:00	276	65.347101	64.10	10.907413
06:45:00	263	65.378327	63.80	11.313136
07:00:00	359	60.099443	59.80	16.032420
07:15:00	378	38.662169	33.70	20.048578
...	...	...	...	...
16:30:00	474	26.456540	19.35	22.400642
16:45:00	440	32.488182	24.30	21.511535
17:00:00	458	43.066594	48.30	23.165219
17:15:00	392	45.722194	51.25	22.295305
17:30:00	312	55.044551	58.15	16.124287
17:45:00	306	56.722549	58.65	17.396952
18:00:00	262	59.352290	59.10	13.721814
18:15:00	262	61.228244	60.70	12.797167
18:30:00	225	58.827556	59.50	11.978725
18:45:00	273	60.760440	61.60	14.550181
19:00:00	246	60.106098	58.80	10.802007

(d) Ponedjeljak, 15 minutni interval na Jadranskom mostu u Zagrebu

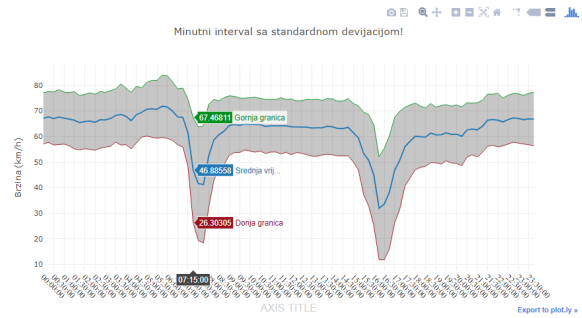
Slika 3.6: Grupirani podaci u tablicama prema različitim vremenskim periodima za link -216495 odnosni Jadranski most u Zagrebu



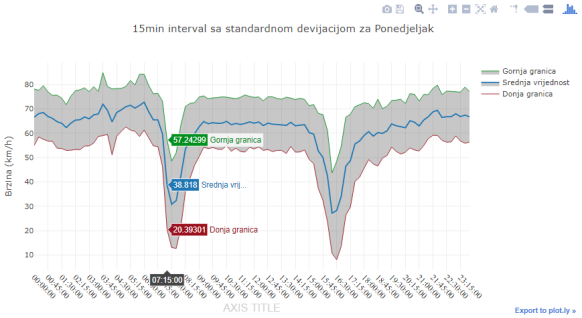
(a) Mjeseci



(b) Dani u tjednu



(c) Svi dani, 15 minutni interval



(d) Ponedjeljak, 15 minutni interval

Slika 3.7: Prikaz grupiranih podataka prema različitim vremenskim periodima za link -216495

Treći korak je priprema obrađenih podataka prikazanih na slici 3.6d za pohranu u bazu podataka. Potrebno je srednje vrijednosti, medijan i standardnu devijaciju pretvoriti u realni tip podataka, a brojač je prirodni broj. Radi boljeg pregleda, potrebno je zamijeniti stupac s vremenskim intervalima, tako da svaki vremenski interval bude zasebni stupac. To je zamjena redova u stupce. Također, potrebno je dodati stupac s nazivom linka, odnosno broj koji može biti i negativnog predznaka, što određuje zastavica sa slike 2.4. U nastavku je opisana pohrana podataka u bazu podataka i problematika vezana uz pohranu podataka.

3.3. Pohrana podataka u bazu podataka

Prije samog opisa pohrane podataka u bazu podataka, potrebno je ukratko objasniti što su baze podataka i svrha baza podataka. Baza podataka je skup međusobno povezanih podataka, pohranjenih bez zalihosti, a koriste se za različite aplikacije, povezivanje s programskim alatima, itd. Podaci koji posjeduju ista obilježja, iste stupce, odnosno podaci koji su međusobno povezani, spremaju se u kreirane tablice (s definiranim stupcima) te im se dodjeljuje identifikacijski broj (Id) koji je ujedno i primarni ključ. Primarni ključ jedinstveno definira svaki red u tablici. Baza podataka se zasniva na upitima, gdje korisnik baze zahtjeva podatke iz baze te postavlja upit bazi za dohvaćanje podataka. U tom upitu mora biti točno definirano koji se podaci žele dobiti iz baze podataka. Administrator baze podataka vodi brigu o strukturi baze poda-

taka i po potrebi izmjenjuje i optimizira bazu te postavlja ograničenja pristupa bazi podataka. Administrator definira tko od korisnika baze podataka smije dohvatiti, unijeti, brisati i ažurirati podatke. Danas je teško pronaći sustav bez baze podataka u kojima se pohranjuju podaci korisnika, koji mogu biti zaposlenici organizacije ili klijenti organizacije. Logično je kako će zaposlenici moći dohvatiti, unijeti, brisati i ažurirati podatke, ovisno o vrsti posla koji obavljaju u organizaciji. Dok će krajnji korisnici/klijenti moći pristupiti bazi podataka kroz definirane poglede (gdje vide samo podatke koje definira organizacija), odnosno aplikacije koje pružaju samo određene podatke. Primjer je registracija na *web* stranicu, na kojoj korisnik upisuje svoje podatke i koji se spremaju u bazu podataka, a zatim može pristupiti određenim sadržajima/informacijama, ali samo onom sadržaju koji organizacija odluči dijeliti sa svima koji se registriraju.

Pohrana podataka je sve veći problem zbog sve većeg i većeg broja podataka koji se svakodnevno stvaraju. Sirovi podaci dobiveni putem projekta SORDITO prikupljani su do 2015. i trenutno se ne stvaraju novi podaci, tako da problem konstantne pohrane novih podataka nije problem u ovom radu. Nakon izrade programske podrške i kada se stvori grafičko korisničko sučelje (engl. *Graphical User Interface*, GUI) koje dohvaća podatke iz baze, pohrana ima značaj u obliku brzine izvršavanja dohvata podataka iz baze te će se zato testirati tri načina pohrane podataka.

Prvi način pohrane podataka je stvaranje tablice za svaki dan u tjednu i za svaki vremenski interval. Dakle, za dva minutna intervala (5 min i 15 min) stvoreno je 14 tablica. Tablice za 5 min interval sastoje se od 288 stupaca, odnosno intervala plus stupac s brojem linka te tablice za 15 min interval koji se sastoji od 96 stupaca plus stupac s brojem linka. S obzirom na to da ne postoje tablice koje se povezuju, upiti su vrlo jednostavni. Potrebno je samo navesti tablicu i navesti identifikacijski broj linka te se dohvaća jedan red sa svim stupcima za navedeni link. U jednu ćeliju pohranjuje se više vrijednosti, a te vrijednosti su srednja vrijednost, medijan i standardna devijacija. U većini slučajeva jedna ćelija posjeduje jednu vrijednost, ali u ovom slučaju jedna ćelija posjeduje tri vrijednosti koje su odvojene znakom ";" koji se zove graničnik (engl. *delimiter*).

Drugi način pohrane je stvaranje tri tablice koje su međusobno povezane. Prva tablica, najviše razine, se zove **linkovi** i sadržava informacije o linkovima, odnosno stupce identifikacijski broj (Idlink), broja linka (koji je bio identifikacijski broj linka u prvom načinu pohrane), kategorija linka i naziv linka kao što je prikazano na slici 3.8a. Druga tablica se naziva profili koja je niže razine od prve tablice te se sastoji od stupaca identifikacijski broj profila (Idprofil), stupca dan gdje se pohranjuju dani u tjednu za svaki minutni interval. Dakle, ako su dva minutna intervala 5 min i 15 min pohranit će se 14 redaka, tj. svaki dan po dva puta kao što je vidljivo iz slike 3.8b. Treći stupac se naziva interval i pohranjuje se interval za svaki dan u tjednu, dakle sedam redaka 5 minutnog intervala u obliku 00:05:00 i sedam redaka 15 minutnog intervala u obliku 00:15:00. Zadnji i najbitniji stupac je strani ključ, naziva linkId koji se veže

na tablicu više razine (tablicu linkovi) i dodjeljuju se identifikacijski brojevi linkova za sve dane u tjednu, puta različiti minutni intervali. S obzirom na to da se pohranjuju dva različita minutna intevala, jedan identifikacijski broj linka će se prepisati u 14 redaka u stupac linkId. Idlinka s brojem jedan iz tablice linkovi će biti upisan u tablicu profili pod stupac linkId 14 puta, dakle 14 jedinica, i svaki idući broj u Idlinku će se upisivati po 14 puta.

Treća tablica pod nazivom **vrijednosti** je još jednu razinu ispod tablice profili i sastoji se od šest stupaca. Prvi stupac je identifikacijski broj vrijednosti (Idvrijednost), drugi stupac je naziva vrijeme i tu se upisuju vremena u danu za pojedini minutni interval. Za 5 min interval, upisat će se najviše 288 redaka, a za 15 min interval najviše 96 redaka. Treći stupac je srednja vrijednost, četvrti stupac medijan, a peti stupac je standardna devijacija. Zadnji, odnosno šesti stupac se zove profilId koji je strani ključ i veže se na tablicu profili. Na primjer, ako je identifikacijski broj u profilima jedan i 5 minutni je interval, u stupac profilId, odnosno šesti stupac će se upisati najmanje 288 redaka, tj. najmanje 288 jedinica kao što se vidi iz slike 3.8c. Ako je 15 minutni interval onda će se u stupac profilId upisati najmanje 96 iste brojke koja odgovara identifikacijskom broju profila.

Data Output | Explain | Messages | Query History

	idlink [PK] integer	broj bigint	kategorija integer	naziv character varying (255)
7445	7445	-214688	1040	Selska cesta
7446	7446	-214689	1040	Selska cesta
7447	7447	-214690	1040	Selska cesta
7448	7448	-214691	1040	Selska cesta
7449	7449	-214692	1040	[null]
7450	7450	-214693	1040	Jadranski most
7451	7451	-214694	1040	Jadranski most
7452	7452	-214695	1040	Jadranski most
7453	7453	-214696	1040	Jadranski most
7454	7454	-214697	1040	Jadranski most
7455	7455	-214698	1040	Jadranski most
7456	7456	-214699	1040	Jadranski most
7457	7457	-214700	1040	Jadranski most
7458	7458	-214701	1040	Jadranski most
7459	7459	-214702	1040	[null]
7460	7460	-214703	1040	Selska cesta
7461	7461	-214704	1040	Selska cesta

(a) Tablica linkova

Data Output | Explain | Messages | Query History

	idprofil [PK] bigint	dan character varying (255)	interval interval	linkid integer
104272	104272	Nedjelja	00:15:00	7451
104273	104273	Ponedjeljak	00:05:00	7452
104274	104274	Utorak	00:05:00	7452
104275	104275	Srijeda	00:05:00	7452
104276	104276	Četvrtak	00:05:00	7452
104277	104277	Petak	00:05:00	7452
104278	104278	Subota	00:05:00	7452
104279	104279	Nedjelja	00:05:00	7452
104280	104280	Ponedjeljak	00:15:00	7452
104281	104281	Utorak	00:15:00	7452
104282	104282	Srijeda	00:15:00	7452
104283	104283	Četvrtak	00:15:00	7452
104284	104284	Petak	00:15:00	7452
104285	104285	Subota	00:15:00	7452
104286	104286	Nedjelja	00:15:00	7452
104287	104287	Ponedjeljak	00:05:00	7453
104288	104288	Utorak	00:05:00	7453
104289	104289	Srijeda	00:05:00	7453

(b) Tablica profila

Data Output | Explain | Messages | Query History

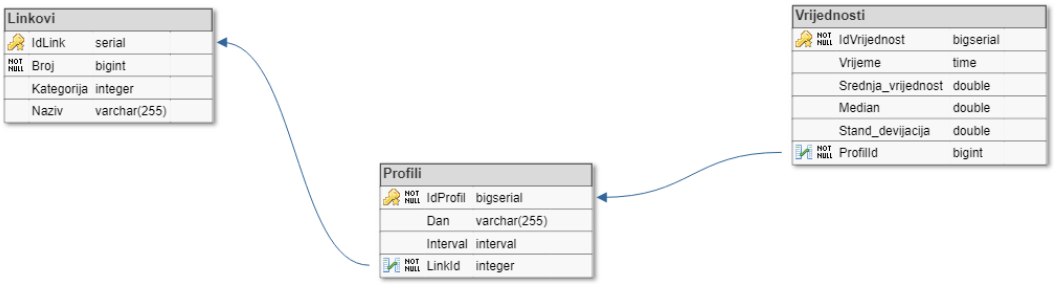
	idvrijednost integer	vrijeme time without time zone	mean double precision	median double precision	std double precision	profilid bigint
1	10107894	00:00:00	68.8031746031746	68.6	10.9617070758805	104273
2	10107895	00:05:00	66.8968253968254	67	11.1240072881332	104273
3	10107896	00:10:00	63.8898305084746	63.2	12.3659250238272	104273
4	10107897	00:15:00	66.769014084507	66.3	8.40509355007504	104273
5	10107898	00:20:00	69.6358208955224	69.1	9.85266771277426	104273
6	10107899	00:25:00	67.8392857142857	66.35	10.5882899052153	104273
7	10107900	00:30:00	69.1203703703704	68.7	10.5297132077562	104273
8	10107901	00:35:00	69.4575757575757	69.85	10.7330841593223	104273
9	10107902	00:40:00	67.0968253968254	66.9	11.7463922842219	104273
10	10107903	00:45:00	68.1552631578947	67.55	8.73685518916796	104273
11	10107904	00:50:00	65.45	64.95	10.4997385588367	104273
12	10107905	00:55:00	66.5549019607843	64.9	11.5169842185442	104273
13	10107906	01:00:00	67.7385714285714	68.15	9.66123724871631	104273
14	10107907	01:05:00	65.8529411764706	65.1	8.84823943937269	104273
15	10107908	01:10:00	64.0458333333333	63.3	9.57134371796228	104273
16	10107909	01:15:00	66.84	65.3	9.88845956926868	104273
17	10107910	01:20:00	62.7732142857143	64.9	11.0388486071432	104273
18	10107911	01:25:00	64.9	64.3	11.5312799231771	104273

(c) Tablica vrijednosti

Slika 3.8: Tablice za drugi način pohrane podataka

Sve tri tablice moguće je vidjeti putem dijagram entiteta i veza (engl. *Entity Relationship Diagram*, ERD) na slici 3.9 i relacijski model baze na slici 3.10. ERD dijagram prikazuje veze između tablica, odnosno entiteta gdje **1** link može posjedovati **više**, tj. **N** profila, a **1** profil može posjedovati **više**, tj. **N** vrijednosti. Relacijski model predstavlja tablice koje su prije opisane za drugi način pohrane.

Slika 3.9: Prikaz dijagrama eniteta i vrste veza između njih



Slika 3.10: Prikaz relacijskog dijagrama putem tablice sa svim imenima stupaca i tipovima stupaca

Treći način je pohrana sirovih podataka u bazu podataka bez predobrade, osim pretvorbe *epoch* UTC u oblik datuma čitak ljudima. Pohranjuju se svi podaci u jednu tablicu sa šest stupaca. Prvi stupac je datum, drugi je brzina jedan (v_1), treći je brzina dva (v_2), četvrti je brzina tri (v_3), peti je brzina četiri (v_4) i šesti stupac je broj linka. Broj redaka je ovisan o broju redaka za svaki link, tj. broju redaka u svakoj tekstualnoj datoteci. S obzirom na to da tekstualne datoteke mogu sadržavati i više od 200.000 redaka, a obraditi će se preko 14.000 redaka, jasno je kako će broj redaka biti iznimno velik. S obzirom na navedeno ovaj način se neće dalje opisivati, jer je nepregledan i brzina dohvaćanja i pohrane je znatno lošija, odnosno sporija uspoređujući s dva načina pohrane prije objašnjena.

Koristiti će se prvi način, stvaranje 14 tablica, jer je preglednije i brže od drugog i trećeg načina, međutim, u slučaju da se grupiraju podaci u više od dva intervala, već tada je potrebno razmotriti drugi način kao bolju opciju. Razlog je prevelik broj tablica koji se stvara svakim novim intervalom.

4. Izrada programske podrške za filtriranje, obradu i pohranu podataka u relacijsku bazu podataka

Za izradu programske podrške, prvo su opisani programski alati koji su se koristili za izradu programske podrške i zbog čega su odabrani upravo ti programski alati. Nakon toga će se opisanim programskim alatima obraditi sirovi podaci te pohraniti u odabranu bazu podataka, koja će se također opisati.

4.1. Programski alat – Python

Python je nastao krajem 1980-te, prema riječima Guida van Rossuma koji je radeći u istraživačkoj grupi za distributivni operativni sustav *Amoeba*, u slobodno vrijeme razvijao skriptni jezik[1] za administrativne poslove. Programski jezik koji su koristili u istraživanju zove se sve osnovni kôd (engl. *All Basic Code*, ABC) koji nije bio proširiv i pogodan za poboljšanja. Tada je Guido van Rossum, koji je bio dio istraživačke grupe, stvorio interpretirajući programski jezik visoke razine spajajući dobre karakteristike ABC-a i drugih programskih jezika te ga nazvao Python. Python ne nudi revolucionarna rješenja na programske probleme, već je spoj više jezika, skriptnih i sistematskih programskih jezika te ga čini jednostavnim i razumljivim. Bitno je naglasiti kako otvorenost kôda[2] (engl. *open source*) omogućava razvoj biblioteka (engl. *library*) od strane svakoga. Razlozi odabira Pythona kao programske podrške u diplomskom radu, kao i velikog broja velikog broja korisnika su sljedeći:

– **kvaliteta programske podrške** u smislu čitljivosti i logičkoj povezanosti kôda te ga je jednostavno razumjeti, ponovno koristiti i održavati. Velika je odlika razumjeti kôd i

[1] skriptni jezik je način programiranja drugačiji od klasičnih programskih jezika (C, C++, Pascal, Fortran). Namjena im je sastavljanje programa od gotovih aplikacija, odnosno komponenti, njihovim lijepljenjem čime se postiže viša razina programiranja i brži razvoj aplikacija.

[2] Otvoreni kôd ne znači samo pristup izvornom kôdu nego i slobodnu redistribuciju, modifikaciju i izvedene oblike programa, integritet autorskog izvornog kôda, licenca ne smije diskriminirati pojedinačne osobe, skupine ljudi ili djelatnosti, licenca ne smije ograničavati drugu programsku podršku, itd. izvor: `http://otvorenikod.nsk.hr/definicija-otvorenog-koda/`

onda kada nije samostalno napisan, a i krasi ga mogućnost napredne ponovne upotrebe za objektno orijentirano programiranje (engl. *Object-Oriented Programming*, OOP),

- **razvojna produktivnost** je povećana uporabom Pythona, jer je Python kôd često za trećinu manji od kôda programskih jezika primjerice C++, Java, itd. Količinski manje linija kôda, lakše je za pronaći potencijalne pogreške i održavati kôd bez gubitka funkcionalnosti kôda,
- **interoperabilnost**, odnosno mogućnost korištenja praktički istog kôda na najkorištenijim računalnim operacijskim sutavima (Windows, Linux, Macintosh). Potrebno je samo uskladiti verzije Pythona i posjedovati iste programske pakete na operacijskim sustavima koji se koriste u kôdu,
- **bogatstvo biblioteka** je najveća prednost Pythona u odnosu na druge programske jezike jer postoji veliki broj biblioteka koji proširuju funkcionalnosti Pythona. Standardne biblioteke[3] koje dolaze uz instalaciju Pythona već nude velik broj funkcionalnosti, a to je samo mali djelić svih biblioteka. Biblioteke su često napravljene za uspostavu informacijsko-komunikacijske veze između Pythona i drugih aplikacija, programa, programskih jezika, itd. Koristi se za izradu *web*-a, videoigara, obradu numeričkih vrijednosti i podataka, vizualizaciju podataka, a obrada i vizualizacija podataka je upravo područje gdje se značajno ističe. Znanost o podacima je područje gdje se Python smatra najjačim programskim jezikom, zbog biblioteka kao što su *NumPy*, *Pandas*, *Matplotlib*, *Plotly*, *SQLAlchemy*, itd.

Naravno postoje i mane kod Pythona, a najčešća mana koja se spominje je brzina izvođenja u odnosu na programske jezike C i C++. Razlog je taj što Python prevodi izvorni kôd u *byte* kôd[4] i onda povlači iz *cache bytea* kôd te interpretira napisani kôd. Upravo zbog toga što se Python kôd ne prevodi sve do strojnog jezika, odnosno do nula i jedinica, neki programi će biti sporiji u Pythonu nego u programskom jeziku tipa C koji prevodi kôd do razine strojnog jezika. Ta brzina izvođenja često nije toliko vidljiva i Python je već puno puta optimiziran. Potrebno je istaknuti manu pri instaliranju svih biblioteka ponekad nije moguće koristiti automatski upravljač paketima (engl. *Pip Installs Packages*, pip) za instaliranje paketa već je potrebno samostalno pronaći biblioteke u obliku binarne datoteke[5] (engl. *binaries*) i instalirati ga. Također, još uvijek se službeno koristi Python verzija 2.7 (navodno do 2020. godine) koja je nešto drugačija od Python verzije 3.0 i najnovije verzije 3.7. Za korištenje kôda verzije 2.7, potrebno je znati koje su razlike između verzija 2.7 i 3.0 te preinačiti kôd kako bi bio funkcionalan na verzijama 3.0

[3] Popis standardne biblioteke vidljivo na stranici: `https://docs.python.org/3/library/`

[4] *byte* kôd je dizajniran za efikasnije izvršavanje interpretatora programske podrške i ne sastoji se od niza nula i jedinica, već od niza brojeva, konstanti i adresa koji definiraju semantiku kôda, koji je tip kojeg objekta, itd.

[5] Binarna datoteka je vrsta računalne datoteke koja pohranjuje podatke u binarnom obliku za spremanje ili daljnju obradu. Uglavnom je neshvatljiv čovjeku te je potrebno provesti datoteku putem računalnom programa, kod Pythona *wheel* se koristi za ugrađivanje paketa, a to su upravo binarne datoteke s nastavkom .whl

pa nadalje.

Python danas koristi velik broj organizacija kao što su Google, YouTube, Intel, Cisco, NASA, ESRI i druge organizacije. U zadnjih nekoliko godina Pythonu je dodatnu vrijednost donijelo područje koje proučava podatke i nastoji na temelju postojećih podataka uočiti zakonitosti, predviđati buduće ishode i stvarati prediktivne modele koristeći algoritme strojnog učenja, a to područje se naziva znanost o podacima (engl. *Data science*). Upravo zbog biblioteka koje Python posjeduje za obradu, pohranu, analizu i grafičko prikazivanje podataka se izdvaja od ostalih programskih jezika, kada je pitanje znanosti o podacima, [8].

4.2. Programski alat – PostgreSQL

PostgreSQL je iznimno kvalitetna aplikacija, odnosno objektna relacijska baza podataka, koja je otvorenog kôda, a to znači kako je izvorni kôd i dizajn programske podrške dostupan svima na korištenje, izmjene i to bez novčane naknade. PostgreSQL koristi strukturirani upitni jezik (engl. *Structured Query Language*, SQL) i dodaje dodatnu vrijednost SQL-u koji osiguravaju pohranu, zaštitu podataka, skaliranje i brojne druge mogućnosti koje će se prikazivati u diplomskom radu. Postoji već 30 godina te je u zadnjih nekoliko godina stekao visoku reputaciju u zajednici koja svakodnevno koristi i održava baze podataka. PostgreSQL, kao i Python, podržava sve najpoznatije računalne operacijske sustave i poznat je po kvalitetnoj arhitekturi, pouzdanosti, čuva integritet podataka i posjeduje robusne značajke. Prema izvorima [9], [10], [11], [12] i [13], što ocrtava opće prihvaćeno mišljenje o PostgreSQL-u, prednosti su:

- otvorenog je kôda i bez novčane naknade omogućava neograničenu veličinu baze podataka maksimalnu veličinu tablice od 32 terabajta, neograničen broj redaka u tablici i 250 do 1600 stupaca po tablici (ovisno o tipu stupca),
- snažna podrška zajednice koja odgovara na pitanja bez novčane naknade,
- ima bolju skalabilnost od konkurencije, jer koristi paralelne upite, integritet podataka je kvalitetniji, a što se tiče sigurnosti posjeduje uloge preko kojih se mogu postaviti i održavati korisničke ovlasti. Također, posjedujuje izvornu podršku protokola za sigurnu komunikaciju (engl. *Secure Sockets Layer*, SSL) s dodatnim poboljšanjem koje kontrolira pristup, zvano SE-PostgreSQL,
- PostgreSQL jako dobro podnosi kompleksnost baze i upravljanje s velikim brojem podataka, pogotovo kada se koriste prilagođene procedure, što je izvrsno za razvijanje usluga. Može se reći kako je PostgreSQL najbolji za izvođenje kompleksnih upita,
- podržava materijalizirane poglede, privremene tablice, velik broj programskih jezika (C/C++, Java, Javascript, .Net, R, Perl, Python, itd.) zajedno s programskim paketima proširenja u samom PostgreSQL-u. Također, podržava JavaScript objekt notacija (engl.

JavaScript Object Notation, JSON) i NoSQL značajke kao što je izvorni proširivi jezik za označavanje (engl. *Extensible Markup Language*, XML),

- moguće je zapisati podatak tipa red (engl. *array*) u pojedinu ćeliju tablice te zbog otvorenosti kôda i proširivosti, PostgreSQL pruža korisniku mogućnost stvaranja prilagođenog tipa podataka koji nije predefiniran, što izdvaja PostgreSQL od konkurencije,
- zadnja prednost je pohrana i obrada zemljopisnih podataka koji se sastoje od točaka, linija, poligona i drugih geografskih objekta s koordinatama, a za to je potrebno preuzeti i instalirati dodatak PostgreSQL-u zvan PostGIS. PostGIS je prostorna baza podataka koja uz standardne tipove podataka posjeduje i različite vrste prostornih objekata kao i operacije nad njima, a aktivira se u samom PostgreSQL-u upisivanjem prikazanog kôda 4.1.[6]

```
1 CREATE EXTENSION postgis;
```

Kôd 4.1: Aktivacija PostGIS-a

Nedostaci koje donosi PostgreSQL su:

- najveći nedostatak je brzina za pokretanje jednostavnijih upita i kod manje količine podataka,
- trenutno ne posjeduje različite vrste tablica,
- prijašnje verzije nisu posjedovale dobru replikaciju baze podataka, međutim, od nedavno je i to poboljšano te se može reći kako i to funkcionalno. Iako je replikacija pouzdanija kod drugih aplikativnih rješenja SQL-a, zbog dugogodišnjeg iskustva korištenja replikacije.

Vidljivo je koliko posjeduje prednosti, a malo nedostataka PostgreSQL. Iz navedenih izvora moguće je vidjeti i još više razloga zašto je PostgreSQL zapravo najbolja opcija. Za upotrebu u diplomskom radu najbitnije su bile dvije značajke koje posjeduje PostgreSQL, a to su dobra podrška za Python, obrada i pohrana velike količine podataka te upotreba zemljopisnih podataka. U tablici 4.1 moguće je uočiti odličnu povezanost PostgreSQL-a s Pythonom te koliko su podržane Python vrste podataka u PostgreSQL-u, a osim navedenih vrsta podataka u tablici, postoji još vrsta koje podržava PostgreSQL [7].

[6] Za uključivanje ostalih dodataka za PosGIS pogledati `https://postgis.net/install/`

[7] PostgreSQL službena dokumentacija o vrstama podataka `https://www.postgresql.org/docs/current/static/datatype.html`

Tablica 4.1 Tablica koja prikazuje relaciju vrsta podataka između Pythona i PostgreSQL-a

Python	PostgreSQL	Opis
None	NULL	Nema podatka/vrijednosti
bool	bool	Logički tip podataka, True ili False, ali PostgreSQL podržava i "t" ili "f", "y" ili "n", "yes" ili "no" te "1" ili "0"
float	real double	to je realni broj i ima maksimalno 4 bajta, preciznost od 6 znamenki realni broj, 8 bajta, preciznost 15 znamenaka
int	smallint	maksimalno 2 bajta pri pohrani, raspon od -32768 do +32767
long	integer	uobičajen odabir za pohranu, 4 bajta
long	bigint	velik raspon brojeva, 8 bajta
Decimal	numeric ili decimal	korisnik definira preciznost iza zareza, do 16383 znamenaka iza zareza
str	varchar	duljina ograničena
unicode	text	neograničena duljina
date	date	4 bajta, datum bez doba u danu
time	time timetz	8 bajta, dob dana bez datuma, od 00:00:00 do 24:00:00 12 bajta, dob dana bez datuma s vremenskom zonom od 00:00:00+1459 do 24:00:00-1459
datetime	timestamp timestamptz	8 bajta, datum i dob dana bez vremenske zone 8 bajta, datum i dob dana s vremenskom zonom
timedelta	interval	16 bajta, vremenski interval
list	ARRAY	niz podataka pohranjen u jednu ćeliju
tuple	Composite types	korisnički definirani tip podataka koji može posjedovati više različitih tipova podataka
dict	hstore	pohrana rječnika
Anything™	json	pohrana JSON-a pomoću kojeg se može bilo što pohraniti u bazu podataka

Izvor: [14]

Što se tiče sintakse, postoje razlike u odnosu na ostale ekstenzije/programske alate koji koriste SQL jezik. Korišteni su primjeri koji su se koristili u daljnjem radu u idućim poglavljima. Osnovna sintaksa koja se često koristi za dohvaćanje svih podataka iz primjerice tablice geometrija je opisana kôdom 4.2.

```
1 SELECT * FROM geometrija;
```

Kôd 4.2: Dohvaćanje svih podataka iz tablice geometrija

Ako se žele dohvatiti samo određeni stupci, umjesto znaka "*" upišu se imena stupaca i odvoje zarezom. Kada tablica nema u bazi podataka, potrebno ih je kreirati, tako da se dodijeli ime stupca, vrsta podataka koji će se pohranjivati u imenovani stupac te ograničenje koje definira smije li biti ćelija bez podataka, a sintaksa za kreiranje tablice je prikazana kôdom 4.3.

```
1 CREATE TABLE Linkovi
2 (IdLink serial PRIMARY KEY,
3 Broj BIGINT NULL,
4 Kategorija integer NULL,
5 Naziv varchar(255) NULL);
```

Kôd 4.3: Kreiranje tablice u relacijskoj bazi

Stvori se tablica naziva Linkovi sa stupcem IdLink koji je identifikacijski broj tipa *serial*[8], a ujedno i primarni ključ koji jedinstveno definira redove u tablici. Ostali stupci su Broj, Kategorija i Naziv sa svojim vrstama podataka i kojima je dopuštena "prazna" ćelija, odnosno *NULL*

[8]dodatno objašnjeno na: http://www.postgresqltutorial.com/postgresql-serial/

vrijednost. Vrsta podataka pod nazivom *serial* se koristi za identifikaciji broj u PostgreSQL-u, što je razlika u odnosu na druge programske alate koji koriste SQL jezike, jer se kod drugih često samo dodaje *integer* kao vrsta podataka za primarni ključ. Iako se možda na prvu ne čini kako je potrebna posebna vrsta podataka za primarni ključ, PostgreSQL nudi dodatnu mogućnost pomoću pseudotipa *serial*, a to je promjena slijeda dodjeljivanja identifikacijskih brojeva prikazano kôdom 4.4.

```
1 ALTER SEQUENCE Linkovi_IdLink_seq INCREMENT 10 RESTART with 100;
```

Kôd 4.4: Promjena slijeda dodjeljivanja identifikacijskog broja

Time se mijenja slijed od primarnog ključa IdLink iz tablice Linkovi, tako da će iduća vrijednost krenuti od broja sto, a svaka iduća vrijednost će se uvećati za deset. Tako će nakon promjene primarni ključevi biti 100, 110, 120, 130, itd. Osim toga moguće je postaviti minimalnu i maksimalnu vrijednost slijeda i nije potrebno navesti ograničenje **NOT NULL**, a sve opcije, primjer i prednosti korištenja slijeda moguće je vidjeti na izvorima [15], [16]. Još jedna razlika PostgreSQL-a je što dohvaća tablicu i stupce malim slovima, a ako tablica ili stupac imaju kombinaciju malih i velikih slova, stavljanjem u navodnike jednostavno se dohvati pravilno, primjer prikazan kôdom 4.5.

```
1 SELECT * FROM ponedjeljak_5min WHERE "IdLink"  = -214695
```

Kôd 4.5: Dohvaćanje stupca s detekcijom malih i velikih slova iz tablice

Kod drugih programskih alata za relacijske baze podataka to je kompleksnije i potrebno je znati više sintakse. Koliko je PostgreSQL sintaksa dobro promišljena i napravljena na ruku korisnika prikazuje sljedeći primjer gdje se kreira nova tablica na temelju upita za dohvaćanje podataka iz druge tablice kao što je prikazano kôdom 4.6.

```
1 CREATE TABLE izvrsni_filmovi AS
2 SELECT * FROM filmovi
3 WHERE imdb_ocjena >= 8;
```

Kôd 4.6: PostgreSQL stvaranje tablice pomoću upita iz postojeće tablice

Kod Microsoft SQL Servera takav upit je prikazan kôdom 4.7.

```
SELECT * INTO izvrsni_filmovi
FROM filmovi
WHERE imdb_ocjena >= 8;
```

Kôd 4.7: Microsoft SQL Server stvaranje tablice kroz upit iz postojeće tablice

Osim što je znatno logičnija sintaksa PostgreSQL upita, jer i početnik može zaključiti na temelju uporabljene sintakse što upit radi, moguće je označiti upit za dohvaćanje i vidjeti rezultat upita. Kod upita za Microsoft SQL Servera mora se napisati dodatni upit za provjeru što dohvaća upit `SELECT*FROM filmovi WHERE imdb_ocjena >= 8` te pruža li zadovoljavajuće rezultate. Još jedan primjer jednostavne sintakse PostgreSQL-a je pri provjeri postoji li baza i ako postoji neka se izbriše, odnosno odbaci kako je prikazano kôdom 4.8.

```
DROP TABLE IF EXISTS Linkovi;
```

Kôd 4.8: Brisanje tablice ako postoji

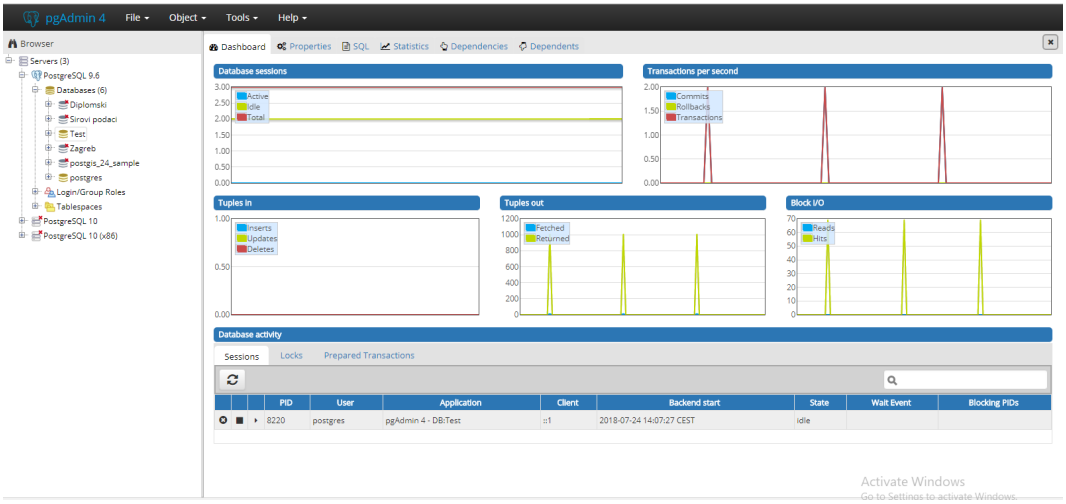
Ostali proizvođači programskih alata za relacijske baze podataka su uvidjeli jednostavnost ove sintakse PostgreSQL-a te su usvojili i implementirali sintaksu u svoje programske alate. Stvaranje druge tablice koja je niže razine od tablice Linkovi i povezana na nju sa stranim ključem prikazana je kôdom 4.9, a gdje se LinkId povezuje s IdLinkom iz tablice Linkovi i tako LinkId postaje strani ključ.

```
CREATE TABLE Profili
(IdProfil bigserial PRIMARY KEY,
Dan varchar(255) NULL,
Intervali interval NULL,
LinkId int REFERENCES Linkovi(IdLink));
```

Kôd 4.9: Kreiranje druge tablice sa stranim ključem

Prikazane su brojne prednosti i osnovna sintaksa za početno razumijevanje PostgreSQL-a, u radu je prošireno osnovno znanje, a izgled PostgreSQL aplikacije koja se od nedavno otvara putem *web* pretraživača[9] je vidljiv iz slike 4.1.

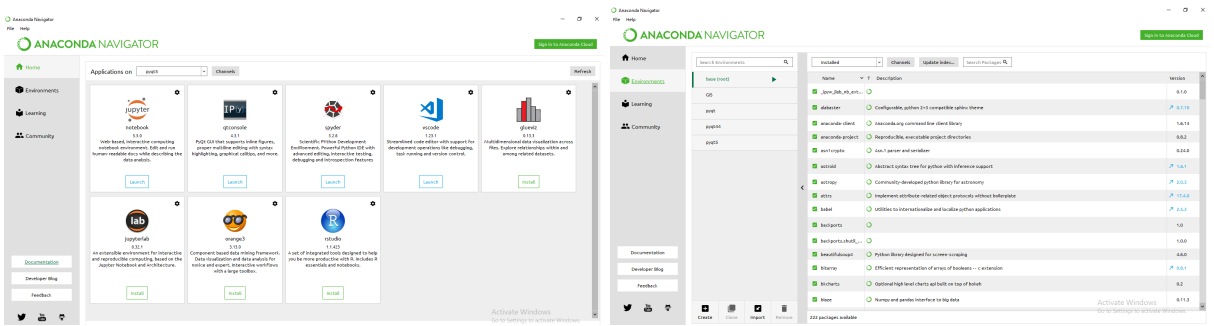
[9]Razlog zašto se otvara preko *web* pretraživača na izvoru: `https://www.postgresql.org/about/news/1846/`



Slika 4.1: Prikaz PostgreSQL grafičkog sučelja

4.3. Razvojno okruženje i biblioteke

Prvi korak je stvaranje razvojnog okruženja, dakle preuzimanje i instaliranje odabranih programskih alata, a to su Python i PostgreSQL. Za Python je odabrana jedna od najpopularnijih platformi, Anaconda, koja je otvorenog kôda i nudi razvojno okruženje sa svim potrebnim alatima za područje znanosti o podacima. Kao integrirano razvojno okruženje (engl. *Integrated Development Environment*, IDE) za Python se koristi Jupyter bilježnica (engl. *Notebook*) i Spyder koji su dio Anacondinog navigatora na kojem se nalaze sva virtualna okruženja i podržani programski alati, a izgleda kao na slici 4.2a. Virtualna okruženja su iznimno korisna, jer je moguće stvoriti virtualna okruženja s različitim verzijama Pythona, programskim paketima koji nisu ažurirani na novijim Python verzijama te drugim situacijama gdje se događaju nekompatibilnosti. Prvo je potrebno odabrati okruženje (engl. *Evironments*), kako je prikazano na slici 4.2b, pritisnuti stvaranje (engl. *create*) virtualnog okruženja te dodijeliti ime i verziju Pythona koji će se koristiti. Zatim se vratiti na početnu stranicu (4.2a) odabrati stvoreno virtualno okruženje i instalirati Jupyter bilježnicu i Spyder.



(a) Prikaz instaliranih aplikacija kroz Anaconda navigator

(b) Prikaz virtualnih okruženja i instaliranih biblioteka

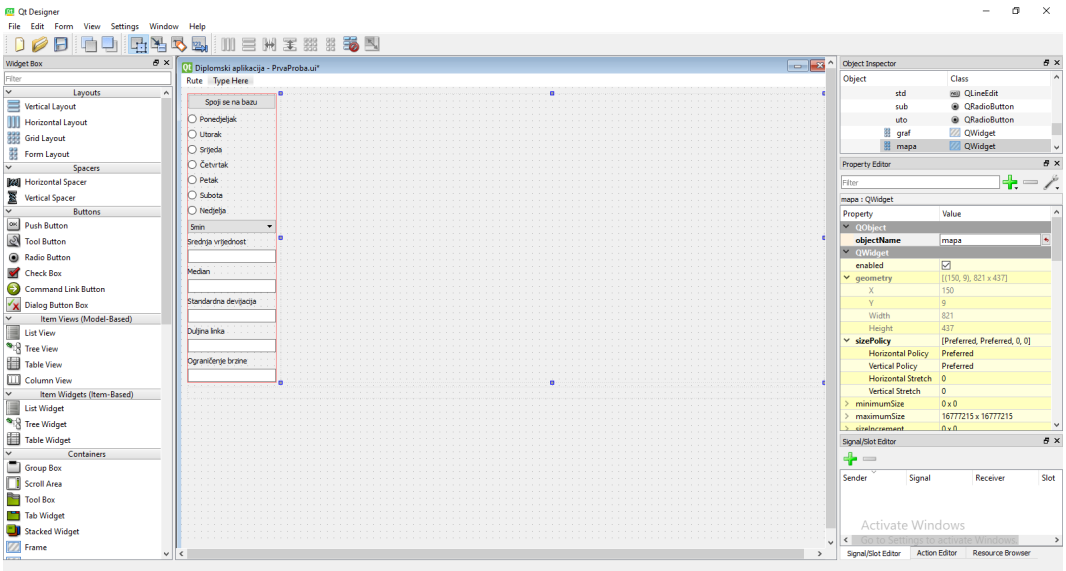
Slika 4.2: Anaconda razvojno okruženje

Sljedeći korak je ponovno odabrati okruženje, točnije novo stvoreno virtualno okruženje i otvoriti terminal te instalirati potrebne biblioteke. U većini slučajeva za instalaciju se upiše `pip install <ime biblioteke>` i automatski se instalira (pojedine biblioteke je bolje ručno instalirat, a to će se naglasiti). Biblioteke će se grupirati u dvije cjeline. Prva skupina su biblioteke za obradu i prezentaciju podataka, a druga skupina su biblioteke za obradu i prezentaciju geografskih podataka. **Biblioteke za obradu i prezentaciju podataka**:

- ***pandas*** – iznimno moćna biblioteka za analizu podataka pomoću koje se može praktički učitati bilo koja datotečna vrsta podataka i stvoriti Python objekt sa stupcima i recima, primjer su slike 3.6. Vrsta objekta koja se stvara s *pandas* se zove *dataframe* pomoću kojeg se mogu koristiti brojne metode za pregled podataka, dohvaćanje redaka/stupaca, filtriranje, sortiranje, grupiranje, itd., [17],
- ***NumPy*** – jedan od osnovnih paketa za matematičku obradu podataka, koji se najčešće upotrebljava za matrice i višedimenzionalne nizove te je iznimno koristan za područje matematike koje se zove linearna algebra, [18],
- ***matplotlib*** – biblioteka za grafički prikaz podataka u dvije dimenzije putem grafova u obliku histograma, grafikona, dijagrama, mapa, itd. Primjer korištenja *matplotlib*-a je prikazano na slici 3.3, a za pregled velikog broja primjera i bogate dokumentacije pogledati izvor [19],
- ***Plotly*** – nadogradnja na biblioteku *matplotlib* zbog značajno boljih grafičkih prikaza i veliki naglasak na izvrsnu interaktivnost grafova. Razlog je što se temelji na poznatoj Javascript biblioteci koja se zove D3 te prezentacijski jezik (engl. *Hypertext Markup Language*, HTML) i stilski jezik (engl. *Cascading Style Sheets*, CSS), a time se mogu prikazati sve vrste grafova. U *matplotlibu* se postiže interaktivnost koja je izvrsna za postaviti graf na *web* ili u aplikaciju. Za grafički prikaz podataka u diplomskom radu najviše se koristio *plotly*, a obilje primjera i cjelokupnu dokumentaciju pogledati na [20],
- ***psycopg2*** – je najpopularniji adapter za spajanje Pythona i PostgreSQLa, ostvarenje komunikacije koja omogućava pisanje SQL jezika u Python razvojnom okruženju. Mogu se kreirati tablice, umetati podaci u tablice, brisati tablice, stvarati nove vrste podataka i sve ostale mogućnosti koje su moguće u PostgreSQLu, a samo povezivanjem s PostgreSQLom moguće je odraditi u Pythonu. Cjelokupnu dokumentaciju vidjeti na [21],
- ***SQLAlchemy*** – biblioteka za povezivanje Pythona s bazom podataka koja omogućava razvijateljima aplikacija potpunu kontrolu i fleksibilnost korištenja SQL programskog jezika u Pythonu. S obzirom na to kako sve današnje aplikacije posjeduju relacijsku bazu podataka i objektno orijentirani jezik, a te dvije tehnologije su nekompatibilne, objektno-relacijsko preslikavanje (engl. *Object Relational Mapper*, ORM) nastoji "pomiriti" te dvije tehnologije tako da mogu uspješno komunicirati, a *SQLAlchemy* upravo

to i radi. Automatizira preslikavanje tabličnih podataka iz relacijske baze u Python i omogućava automatsko stvaranje tablica i stupaca, dodjelu tipova podataka i umetanje *pandas* podataka automatski, [22],

– ***PyQt*** – biblioteka koja koristi Python programski jezik za povezivanje s najpopularnijim višeplatformskim alatom Qt koji služi za stvaranje GUI-a, a otud i naziv *PyQt*. Postoji velik broj alata za stvaranje GUI-a u Pythonu, poput *Kivya*, *Pyformsa*, *PySide*, *wxPythona*, *Tkinter*, itd., ali ono što izdvaja *PyQt* je mogućnost izrade GUI-a na principu *"drag and drop"* kroz Qt dizajner. Stvori se GUI kao primjerice prikazan na slici 4.3 te se bez ručnog pisanja kôda, kroz terminal može pretvoriti stvoreni GUI u Python tip podatka (".py") i dobiti apsolutno cijeli kôd za stvaranje GUI-a sa slike 4.3. Kod ostalih navedenih biblioteka potrebno je napisati kôd ručno, dakle postaviti veličine za svaki *widget*[10], povezati grupu *widgeta* koji se zajedno grupiraju, dodijeliti imena, promjeniti veličine *widgeta* kod povećanja i smanjenja GUI-a, itd. Pretvorbom stvorenoga GUI-a kroz Qt dizajner u Python kôd potrebno je još samo animirati sve *widgete* GUI-a kroz Python kôd, odnosno povezati s relacijskom bazom podataka, HTML, CSS, Javascript kôdom za prikaz primjerice *weba*, itd. Cijelu dokumentaciju i razlike između verzija *PyQt*-a moguće je vidjeti na [23], a za diplomski rad se koristila verzija PyQt5.



Slika 4.3: Primjer GUI-a stvorenoga u aplikaciji Qt dizajner

Druga skupina su **biblioteke za obradu i prezentaciju geografskih podataka**:

– ***GeoPandas*** – proširena *pandas* biblioteka za rad s geografskim podacima, odnosno s točkama, linijama i poligonima koji posjeduju geografske koordinate. Geometrijske

[10]Riječ *widget* koja je nastala stapanjem engleskih riječi *window* i *gadget*. *Widget* je dio grafičkog korisničkog sučelja i služi za interakciju s aplikacijama i operativnim sustavom za prikaz informacija i poziva korisnika na akcije, u obliku dijaloških kutija, privremenih prozora, tipki, padajućih izbornika, ikona, vrpca za pomicanje stranice, vrpca s izbornicima, prozora i dr.

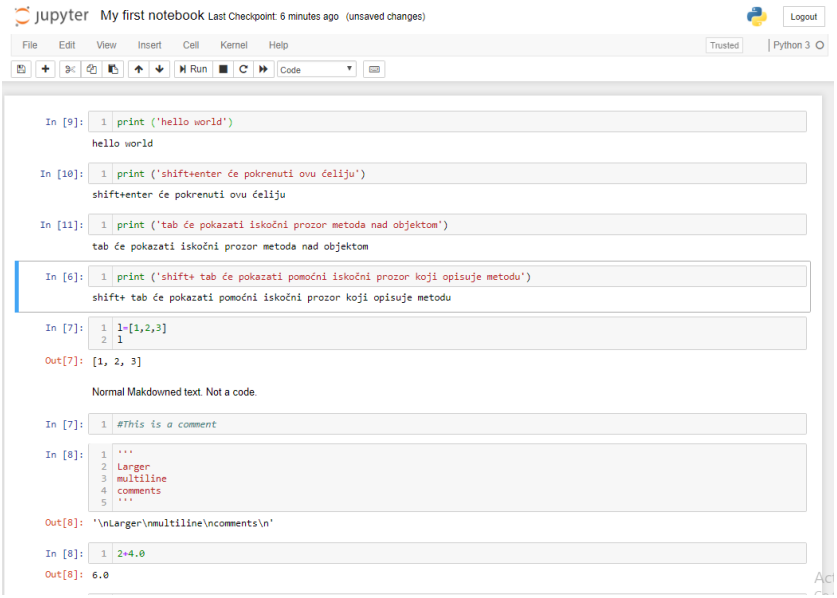
operacije izvršavaju se pomoću biblioteke *shapely*-a što znači stvaranje točka, linija i poligona i drugih oblika od koordinata koje su u sirovom obliku samo realni brojevi. Osim *shapely*a *GeoPandas* ovisi o biblioteci *fiona* koja omogućava čitanje i pisanje geografskih datoteka, a za to koristi OpenGIS implementaciju referentnih jednostavnih značajki (engl. *OpenGIS Simple Features Reference Implementation*, OGR) kao najpopularniju biblioteku za pretvorbu geografskih vektorskih informacija. Za grafički prikaz koristi se već opisani *matplotlib*, a ostale biblikoteke koje dolaze uz *GeoPandas* moguće vidjeti iz izvora [24],

- ***Folium*** – biblioteka koja omogućava smještanje obrađenih podataka kroz *GeoPandas* na interaktivnu kartu, a biblioteka je napravljena prema uzoru na *Leaflet* JavaScript biblioteku. S obzirom na to kako je JavaScript skriptni jezik te je napravljen sa svrhom postizanja interaktivnosti na *webu*, JavaScript biblioteke su trenutačno najbolje za interaktivnost karata. Karta koja se koristila je s *weba*. Moguće je dodavati markere na kartu, tj. atribute geografskim objektima pomoću GeoJSON-a (proširenje JSON-a za manipuliranje geografskih podataka), itd. [25],
- ***GeoAlchemy 2*** – biblioteka koja se nadograđuje na *SQLAlchemy* biblioteku i omogućava ubacivanje *GeoPandas dataframea* u relacijsku bazu podataka, kao i dohvaćanje podataka iz relacijske baze podataka. Iako *SQLAlchemy* podržava većinu različitih ekstenzija/programskih alata koje su relacijske baze podataka, *GeoAlchemy 2* se najviše fokusira na PostGIS, a upravo se PostgreSQL koristi kao relacijska baza podataka u radu. [26].

Nakon instalacije biblioteka za obradu i prezentaciju geografskih podataka prema izvoru [24] dolazi do pogrešaka, tako da je potrebno ručno preuzeti biblioteke *fiona*, *gdal*, *psycopg2* i *SQLAlchemy* s *web* stranice: `https://www.lfd.uci.edu/~gohlke/pythonlibs/` te instalirati terminalom.

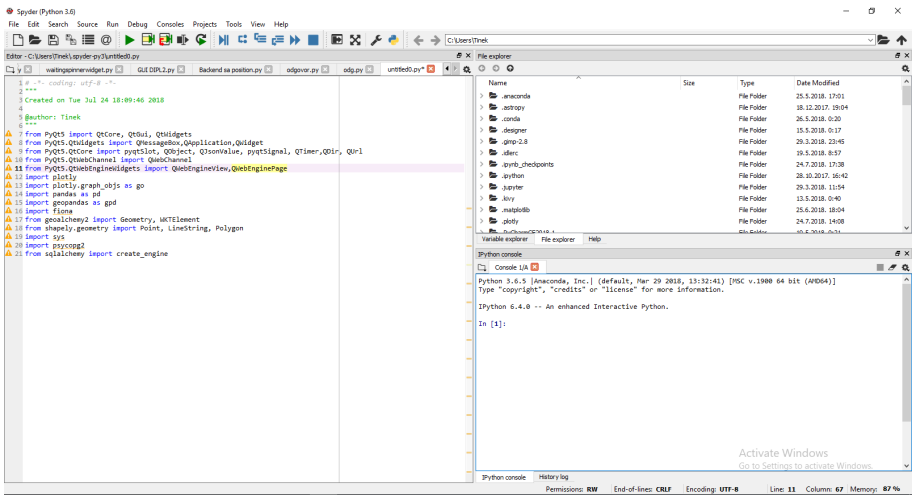
Time je razvojno okruženje spremno za rad. Prije početka razvoja na slici 4.4 moguće je vidjeti Jupyter bilježnicu koja je drugačija od Spydera i ostalih IDE-ova po tome što su bazirani na *webu*, odnosno otvara se preko *web* pretraživača. Dokumenti koje Jupyter stvara je JSON dokument sa shemom verzije te se sastoji od ćelija, a svaka ćelija se može gledati kao jedan ".py" dokument, jer pokretanjem ćelije se pokreće samo kôd unutar ćelije. Vrsta datoteke koju stvara Jupyter bilježnica ima nastavak ".ipynb", upravo zato što je drugačiji i zato što posjeduju veću interaktivnost jer svaka ćelija može prikazati graf ili rezultat neke klase, funkcije, metode. Pri razvoju ovo donosi ogromnu prednost, jer je jednostavno segmentirati kôd na manje dijelove i pronaći potencijalne greške, optimizirati kôd, itd. Prednost je i ta što Jupyter bilježnica može biti razvojno okruženje i za sve druge programske jezike, kao i za Python. Za sve ostale informacije pogledati izvor [27].

Spyder je klasični IDE i krase ga brojne prednosti kao što su IPython konzola koji je pret-



Slika 4.4: Primjer Jupyter bilježnice s više ćelija koje objašnjavaju osnovne funkcije

hodnica Jupyter bilježnice, a dobar je jer podržava više programskih jezika i pokazuje povijest izlaznih rezultata. Spyder se sastoji od tri glavna prozora kako je vidljivo sa slike 4.5 te je istovremeno jednostavnije gledati kôd na lijevoj strani, a u drugom prozoru proučavati vodič pomoću nekih od biblioteka ili tražiti drugu Python datoteku. Također, vidljivo je iz slike Spyder IDE-a sintaksu za uvođenje (engl. *import*) biblioteka te je moguće dodijeliti kraticu za biblioteku. Spyder se koristi u radu, jer bolje pokreće finalnu verziju kôda.



Slika 4.5: Izgled Spyder IDE-a

Iduće poglavlje obuhvaća sve objašnjeno do sada i uspostavlja razvojno okruženje za postizanje rezultata i cilja diplomskog rada.

5. Obrada i pohrana podataka programskom podrškom te stvaranje aplikativnog rješenja

U ovom poglavlju prikazana je obrada, pohrana i grafički prikaz podataka programskom podrškom za skupove podatak iz tablica 2.1 i 2.2. Na kraju, dobiveni rezultati su implementirani u aplikaciju, odnosno GUI koji povezuje rezultate te omogućuje interaktivno dohvaćanje podataka i grafički prikaz podataka.

5.1. Obrada, pohrana i grafički prikaz brzina entiteta u prometnoj mreži programskom podrškom

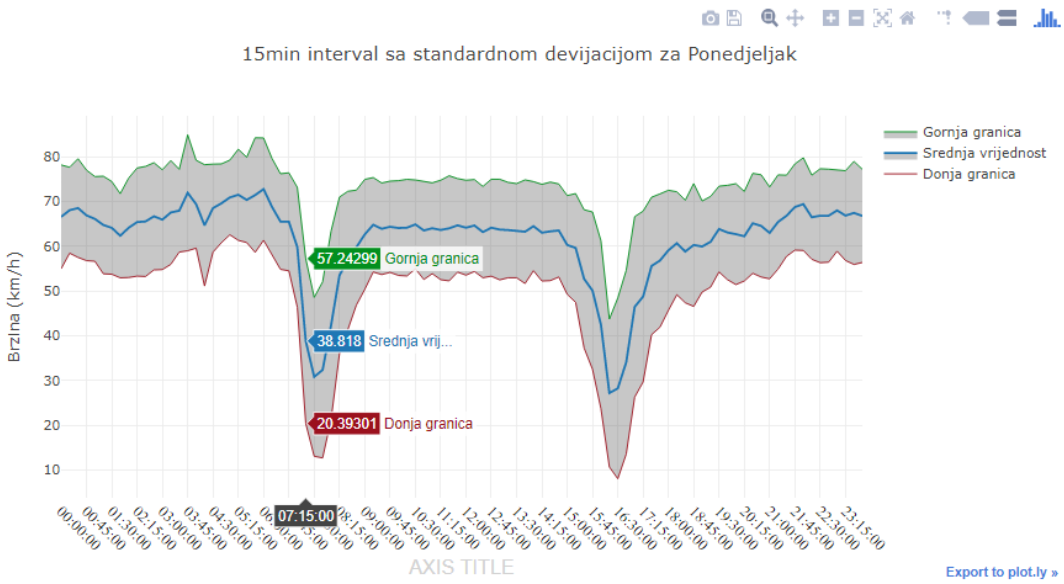
Za početak, potrebno je otvoriti Jupyter te stvoriti novu bilježnicu i uvesti sve biblioteke za obradu i prezentaciju podataka. Kao što je prije spomenuto obrađeno je 14.154 linkova koji su na području Zagreba, a svaki link ima svoju zasebnu datoteku u kojoj su pohranjene brzine u obliku opisanom u tablici 2.1. Gledajući količinski i koliko je potrebno mjesta za pohranu podataka, prvi skup podataka je veličine 9,35 GB. To je prihvatljiva količina podataka, međutim, pri pohrani 448.393 linka, jasno je kako je potreban veći prostor za pohranu od lokalne relacijske baze podataka koja se pokreće na osobnom računalu. Cilj ovog potpoglavlja je stvoriti klasu u kojoj se nalazi skup metoda koje će filtrirati i obrađivati sirove podatke i na kraju ih pohranjivati. Prvo je potrebno oformiti metodu koja će dohvaćati putanju do datoteke, kako bi se mogla dohvatiti svaka tekstualna datoteka za različite linkove i sirovi podaci unutar dototeke za daljnju obradu. Ta metoda će stvoriti listu svih putanja do tekstualnih datoteka, a drugom metodom moguće je dohvatiti broj linka koristeći dohvaćeni put do datoteke i dodijeliti ga obrađenim podacima. Sljedeća je metoda za učitavanje sirovih podatke u *pandas* biblioteku tako što se preda putanja datoteke i postavi graničnik (u ovom slučaju je to znak ";"). Još jedna bitna funkcija koja se izvršava unutar metode za učitavanje je pretvorba *epoch*_UTC u datum s vremenom čitak ljudima, a također mora biti lokaliziran s obzirom na to gdje su zabilježeni GPS zapisi.

Nakon što su podaci učitani u Jupyter bilježnicu, odraditi će se filtriranje podataka metodom MAD, a razlog odabira navedene metode opisan je u potpoglavlju 3.1. Algoritam je stvoren na temelju izraza iz potpoglavlja 3.1 te se prvo ubaci *pandas dataframe* dobiven metodom za učitavanje sirovih podatka, sortiraju se vrijednosti prema brzini v_2 (što je ulazni argument "col_name") te se pronalazi medijan samo za brzinu v_2. Nakon što su brzine poredane po rastućim vrijednostima, računa se MAD. Zatim se stvara novi *pandas dataframe* kao apsolutna razlika vrijednosti s medijanom te se podijeli s dobivenim MAD-om i ponavlja se za svakog člana skupa. Tako se dobije *pandas dataframe* s novim vrijednostima, ali s istim brojem članova u skupu. Na kraju se uspoređuje novo stvoreni *pandas dataframe* s MAD-om, a svaki član koji ima veću vrijednost od MAD-a se odbacuje te se dobije novi filtrirani *pandas dataframe*. Kako algoritam izgleda u Pythonu vidljivo je iz kôda 5.1.

```
def MAD(self, col_name):
    data_in=self.df
    poBrzini1=data_in.sort_values(by=[col_name])
    median = pd.DataFrame.median(poBrzini1[col_name])
    Mad=pd.DataFrame.median(pd.DataFrame.abs(poBrzini1[col_name]-median))
    nesto=pd.DataFrame.abs(poBrzini1[col_name]-median)/Mad
    self.data_out = poBrzini1.loc[(pd.DataFrame.abs(nesto) < Mad)]
    return self.data_out
```

Kôd 5.1: Python kôd za filtriranje podataka pomoću MAD metode

Sljedeće metode unutar klase grupiraju podatke kao što je vidljivo na slici 3.6, a svaka metoda posjeduje opciju grafičkog prikaza podataka, što treba biti ulazni argument za metodu. Za grafički prikaz koristi se biblioteka *plotly* gdje će se na apscisi nalaziti vremenska razdoblja po kojima su grupirani podaci, a na ordinati se nalaze brzine u kilometrima na sat. Primjer četiri vremenska razdoblja je moguće vidjeti na slici 3.6, a kako je objašnjeno u poglavlju 3.2, korišena je kombinacija grupiranja prema danima u tjednu s minutnim intervalima. Primjer grafičkog prikaza u biblioteci *plotly* prikazan je na slici 5.1.



Slika 5.1: Grafički prikaz 15 minutnog intervala sa standardnom devijacijom za ponedjeljak

Grafički prikaz podataka je ljudima iznimno prihvatljiv,popularan i oku ugodan, jer se i iz samih brojki vide rezultati, ali kada se grafički prikažu brojke sve je jednostavnije za vidjeti. Tako je odmah vidljivo iz grafa kako u dva doba dana dolazi do znatnog smanjenja brzina zbog velikih gužvi. Razlog su vršni sati gdje prometni entiteti, odnosno ljudi odlaze na posao u jutarnjim satima od otprilike 7 sati do 9 sati, a odlaze s posla u intervalu od 15 sati do 18 sati. Vidljive su gornje i donje granice, koje su zapravo standardne devijacije i označavaju raspršenost podataka u skupu podataka. Standardna devijacija za skup se interpretira kao prosječno odstupanje od prosjeka, odnosno srednje vrijednosti ili medijana u apsolutnom iznosu, a formulacija je prikazana izrazom (5.1), [28]:

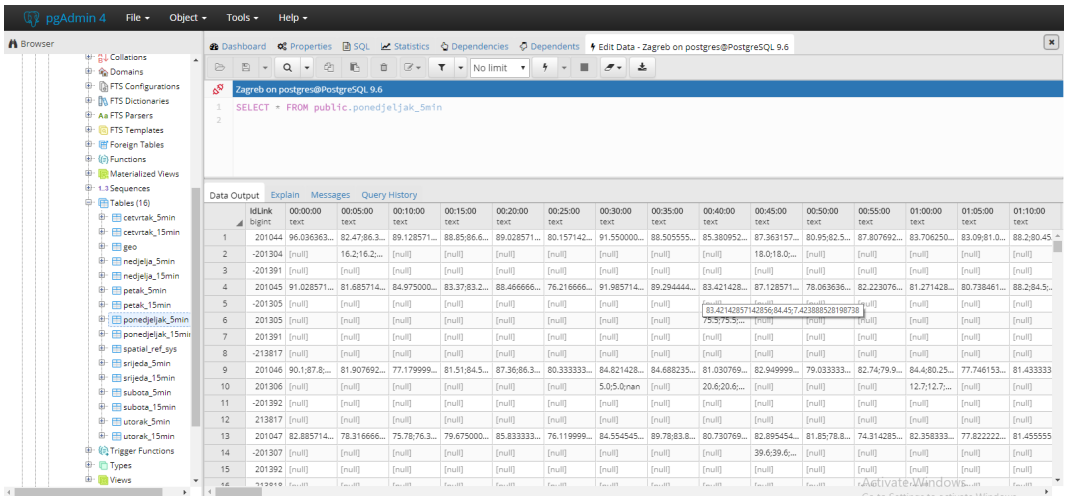
$$\sigma = \sqrt{\frac{\sum_{i=1}^{N}(x_i - \overline{x})^2}{N}} \tag{5.1}$$

gdje je:

- σ – znak za standardnu devijaciju skupa,
- $(x_i - \overline{x})^2$ – kvadrat razlike između promatranih vrijednosti $\{x_1, x_2, \ldots, x_N\}$ i srednje vrijednosti $\overline{x}$,
- N – označava broj vrijednosti/članova skupa.

U potpoglavlju 3.3 je objašnjeno kako će se podaci pohranjivati u relacijsku bazu podaci, a pri tome će se koristiti prije opisane metode. Međutim, za pohranu podataka u relacijsku bazu podataka potrebno je napraviti posebne metode koje će ovisno o načinu pohrane pohranjivati obrađene podatke. Kod prvog načina pohrane za dva vremenska intervala (5 min i 15 min) potrebno je napraviti automatsku metodu gdje je samo potrebno upisati željeni minutni interval i stvori se sedam tablica po intervalu za svaki dan u tjednu s minutnim intervalom u imenu tablice. Unutar metode nalazi se rječnik (engl. *dictionary*) s danima u tjednu koji kroz petlju omogućava

prolazak kroz sve linkove i upis u tablicu za pojedini dan. Također, unutar metode potrebno je spojiti stupce koji su prikazani na slici 3.6d, a to su srednja vrijednost, medijan i standardnu devijacija (bez stupca koji broji broj podataka po retku) te im dodijeliti graničnik. Tako se dobije jedan stupac sa spojenim podacima tipa *string* i stupac index s vremenskim intervalima, a za dobivanje krajnjeg oblika *pandas dataframea* potrebno je transponirati retke i stupce. Time se dobiva željeni format za pohranu u bazu, jedan redak s brojem linka i maksimalno 288 stupaca za tablice s minutnim intervalom 5 min i maksimalno 96 stupaca za tablice s minutnim intervalom 15 min. Pohranjeni podaci u PostgreSQL-u vidljivi su sa slike 5.2. Također, na slici se vide sve tablice s lijeve strane, a s desne strane gore se vidi prozor za upite, a ispod tog prozora nalazi se tablica "ponedjeljak_5min" u kojoj su vidljivi linkovi sa stupcima kao intervalima, dok svaka ćelija posjeduje tri vrijednosti. Bitno je naglasiti kako se tablice stvaraju sa svim atributima i tipovima podataka automatski pomoću *pandas* metode *to_sql*. Prije korištenja automatske metode potrebno je pretvoriti pojedini stupac u Python tip podataka koji odgovara tipu podataka u PostgreSQL-u, te upravo tome i služi tablica 4.1.



Slika 5.2: Primjer izglda tablice u PostgreSQL-u

Kod drugog načina pohrane podataka tablice s atributima/stupcima i tipovima podataka bit će kao na slici 3.10. Najveća razlika između prvog i drugog načina pohrane je ta što je kod drugog načina potrebno napisati kôd koji stvara tablice i kôd koji ubacuje podatke za svaku tablicu. Prikazana je sintaksa kreiranja tablice Linkovi (kôd 4.3) i tablice Profili (kôd 4.9), a sada će se prikazati i stvaranje tablice Vrijednosti, ali kroz Python biblioteku *psycopg2*. Potrebno je uspostaviti vezu s lokalnom PostgreSQL aplikacijom unoseći ime relacijske baze podataka, ime korisnika, lozinku i port 5432 koji je zadan po izvornim postavkama te se stvara pokazivač i tvrdnja u koju se upisuje SQL kôd, a zatim se izvrši tvrdnja i potvrdi kako je prikazano kôdom 5.2.

```
conn=psycopg2.connect("dbname='Diplomski_Zg' user='postgres'"
  ↪  "host='localhost' password='postgres' port=5432")
cursor = conn.cursor()
statement='create table Vrijednosti (IdVrijednost serial PRIMARY KEY,'
  ↪  ' Vrijeme Time NULL, mean double NULL, median double NULL,'
  ↪  'std double NULL, ProfilId int REFERENCES Profili(IdProfil));'
cursor.execute(statement)
conn.commit()
```

Kôd 5.2: Python kôd u biblioteci *psycopg2* za stvaranje tablice

Zatim, potrebno je ubaciti podatke u novo stvorene tablice preko nekoliko petlji. Prvo se ubacuju podaci u tablicu Linkovi, onda se popuni tablica Profili i u stupac LinkId koji je strani ključ upisuje se IdLink koji je zapamčen. Nakon toga se pamti IdProfil i upisuje u stupac ProfilId koji se nalazi u tablici Vrijednosti i tako za svaki link. Primjer ubacivanja podataka moguće vidjeti kroz kôd 5.3 koji za svaku iteraciju dohvaća broj linka, kategoriju i naziv iz drugog skupa podataka koji imaju te potrebne atribute, što je vidljivo iz tablice 2.2. Dakle, potrebno je učitati i drugi skup podataka kako bi se moglo pohraniti podatke na ovaj način s ovim stupcima. Zadnja linija kôda pamti trenutni IdLinka i spremiti će se pod LinkId po sedam puta za svaki minutni interval.

```
cursor = conn.cursor()
sql_string="INSERT INTO Linkovi (broj, Kategorija,Naziv)"
  ↪  "VALUES (%s,%s,%s) RETURNING idlink;"
cursor.execute(sql_string, (broj, Kategorija, Naziv))
linkid = cursor.fetchone()[0]
```

Kôd 5.3: Python kôd u biblioteci *psycopg2* za umetanje podataka u tablicu

Osim toga potrebno je i povezati tablice, tako što će tablice profili i vrijednosti posjedovati strani ključ, što nije potrebno kod prvog načina pohrane podataka. Drugi način pohrane je znatno kompleksniji, ali je bolji za održavanje i izmjenjivanje kôda metode za pohranu jer je ručno napravljen, a ne automatskom metodom za pohranu. Moguće je programskom podrškom dokazati kako je prvi način pohrane brži od drugog načina pohrane. Uzeto je prvih 30 linkova te se mjeri vrijeme dohvata podataka iz PostgreSQL-a, zatim pretvorba *pandas dataframea* u oblik prikazan na slici 5.1 i na kraju grafički prikaz. Za prvi način pohrane prosječno vrijeme dohvata podataka, pretvorba u *pandas dataframea* i grafički prikaz je potrebno 0,055 sekunde, a za drugi način pohrane prosjek je 0,38 sekunde.

Oba načina sadržavaju podosta Python kôda, ali on se neće prikazivati u radu jer je fokus na SQL jeziku i stjecanje osnovnih znanja obrade, pohrane i grafičkog prikaza podataka.

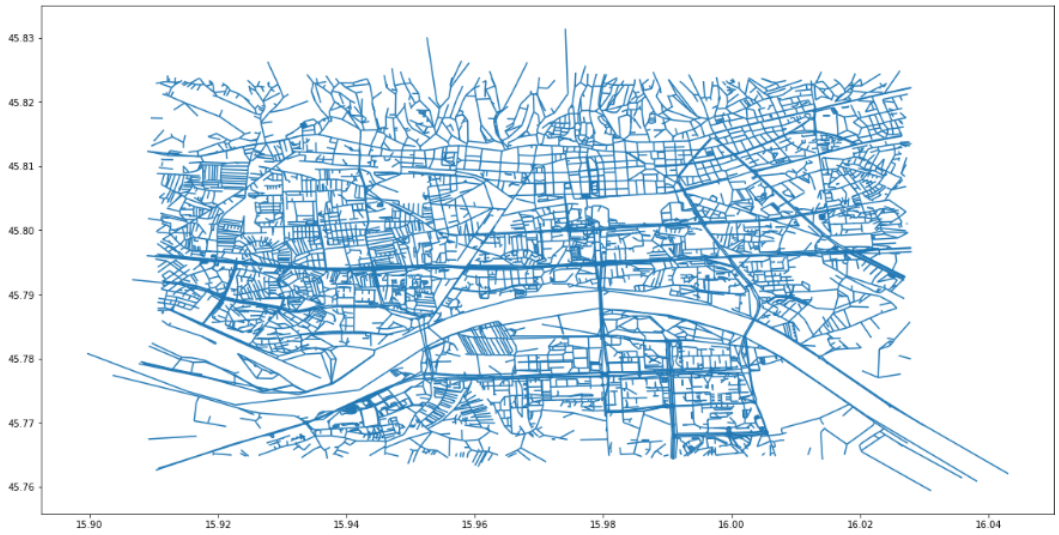
5.2. Obrada, pohrana i grafički prikaz geografskih podataka programskom podrškom

Obrada, pohrana i grafički prikaz geografskih podataka programskom podrškom odnosi se na drugi skup podataka koji opisuje linkove, prikazano u tablici 2.2. Za razliku od prvog skupa podataka, drugi skup podataka posjeduje geografske koordinate, ali su u decimalnom zapisu. Dok su u sirovom decimalnom zapisu to nisu geografski podaci, već su samo brojevi koje nije moguće smjestiti na kartu u ovakvom obliku. Tu dolazi do izražaja Python biblioteka *geopandas* koja sadržava biblioteku *shapely* za pretvorbu sirovih decimalnih zapisa u geografske podatke, točnije u točke (engl. *Point*), linije (engl. *Linestring*) i poligone (engl. *Polygon*). Prvo se učitaju sirovi podaci i prije same pretvorbe u geografske podatke potrebno je uzeti u obzir oznaku linka, odnosno atribut *flag*, jer on označava predznak broja linka i smjernost linka. Dakle, ako je *flag* 2, onda treba dodati znak minus ispred broja linka, jer u sirovom *pandas dataframeu* svi su brojevi linkova pozitivni. Kada je *flag* 0 ili 1 nije potrebno mijenjati broj linka, ostaje pozitivan, tj. bez minusa ispred. Sljedeći je korak stvaranje dva nova stupca koji će predstavljati početnu i završnu točku gdje također *flag* ima veliku ulogu. Kada je *flag* 2 onda početna točka ima koordinate (Završni_X, Završni_Y), a krajnja točka (Početni_X, Početni_Y), što je obrnuti redoslijed u odnosu na linkove gdje je *flag* 0 i 1. Kada se stvore početna i završna točka za svaki link, onda se stvori novi *geopandas geodataframe* u kojem se može definirati stupac geometrija (engl. *geometry*) i koordinatni referentni sustavi (engl. *Coordinate reference systems*, CRS) te se dobije *geopandas dataframe* sa slike 5.3.

	IdLink	Flag	Duljina_m	Srednja_brzina	Ogranicenje_brzine	Kategorija_linka	Naziv_linka	geometry
14124	-394928	2	88	24	0	1080	Trg Milovana Zoričića	LINESTRING (16.0242462158203 45.8149745908398,...
14125	-394929	2	63	24	0	1080	Lavoslava Hartmanna	LINESTRING (16.0239779949188 45.8144361895467,...
14126	-394930	2	12	24	0	1080	NaN	LINESTRING (16.0239887237549 45.8143240219555,...
14127	394931	0	76	24	0	1080	Ante Jakšića	LINESTRING (16.0239243507385 45.813882827238, ...
14128	394932	0	74	24	0	1080	Ante Jakšića	LINESTRING (16.024181842804 45.8132247676055, ...
14129	394933	0	5	24	0	1080	Ante Jakšića	LINESTRING (16.0251152515411 45.81336684932071...
14130	394934	0	82	24	0	1080	Ante Jakšića	LINESTRING (16.0251796245575 45.81338180526961...
14131	394935	0	113	24	0	1080	Brckovljanska	LINESTRING (16.0262095928192 45.81347154087889...
14132	394951	0	46	24	0	1080	Maksimirsko naselje III	LINESTRING (16.026531457901 45.81669444898721,...
14133	394952	0	34	24	0	1080	Maksimirsko naselje III	LINESTRING (16.026166677475 45.8170159817907, ...
14134	394953	0	100	24	0	1080	Hegedušićeva	LINESTRING (16.0261881351471 45.8209714311113,...
14135	394955	0	138	24	0	1080	Ravenska	LINESTRING (16.0280227661133 45.8134416290252,...
14136	394956	0	28	24	0	1080	Ante Jakšića	LINESTRING (16.0262739658356 45.8136958792695,...
14137	394957	0	118	24	0	1080	Prikrajska	LINESTRING (16.0263061523438 45.8134491069901,...
14138	394959	0	219	24	0	1080	Ante Jakšića	LINESTRING (16.0276901721954 45.8137781364529,...

Slika 5.3: Prikaz obrađenih u *geopandas dataframeu*

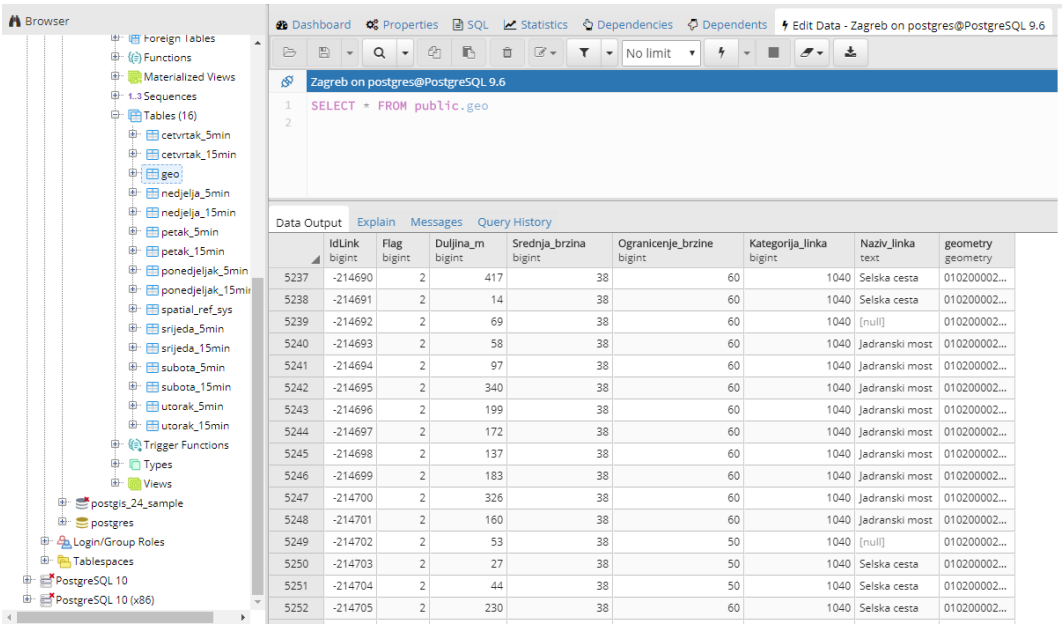
Koordinatni referentni sustav koji se koristi je svjetski geodetski sustav (engl. *World Geodetic System*, WGS 84) koji koristi i GPS, a EPSG je način koji identificira WGS 84 i to je broj 4326 [29]. Time je sama obrada podataka za skup 2 gotova te se može grafički prikazati pomoću biblioteke *matplotlib* kako je prikazano na slici 5.4.



Slika 5.4: Grafički prikaz svih linkova koji ocrtavaju grad Zagreb

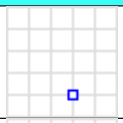
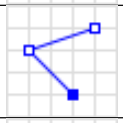
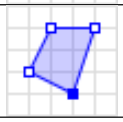
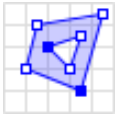
Pohrana podataka je jednostavna, zahvaljujući dvijema biblioteke. Prije su to bile *pandas* i *SQLAlchemy*, a kod geografskih podataka su to *geopandas* i *geoalchemy* 2. Također, može se koristiti automatska metoda koja pohranjuje *geopandas dataframe* u PostgreSQL, odnosno u PostGIS. Potrebno je prije same pohrane svaki stupac pretvoriti u željeni tip podataka koji se želi pohraniti u relacijsku bazu podataka. Međutim, kao što se i vidi iz tablice 4.1 nema spomena o geografski podacima i kako ih pohraniti, što je uredu, jer se zapravo podaci pohranjuju u PostGIS. PostGIS prepoznaje geografske podatke, odnosno geografske objekte (točke, linije, poligone prikazane u tablici 5.1) koji su stvoreni kroz *geopandas*, ali potrebno ih je pretvoriti u opisni jezik za reprezentaciju vektorskih geometrijskih objekata (engl. *Well-known text*, WKT) prije same pohrane. Tu pomaže implementirana WKT metoda u biblioteci *geoalchemy* 2 unutar koje je potrebno navesti stupac geometrije gdje se nalaze geografski objekti i broj CRS-a. Na kraju je potrebno iskoristiti automatsku metodu koju pruža *geopandas*, a ona stvara tablicu i ubacuje podatke iz prije obrađenog *geopanas dataframea* te se dobije tablica u bazi podataka prikazana na slici 5.5. Iz slike je vidljivo kako u PostgreSQL bazi, odnosno POstGIS-u stupac geometrija, tj. geometrijski objekti nisu spremljeni onako kako je prikazano u tablici 5.1, već su pohranjeni u heksadecimanlnom zapisu[1].

[1] `http://www.fpz.unizg.hr/hgold/es/de/heksadecimalni.htm`



Slika 5.5: Prikaz tablice sa stupcima iz tablice 2.2

Tablica 5.1 Tablica koja prikazuje relaciju vrsta podataka između Pythona i PostgreSQL-a

Tip	Primjeri	
Točka		Point(30 10)
Linija		LineString (30 10, 10 30, 40 40)
Poligon		Polygon (30 10, 40 40, 20 40, 10 20, 30 10)
		POLYGON ((35 10, 45 45, 15 40, 10 20, 35 10) (20 30, 35 35, 30 20, 20 30))

Izvor: [30]

Time je završena obrada, pohrana i grafički prikaz podataka programskom podrškom za oba skupa podataka. U sljedećem potpoglavlju su iskorišteni dobiveni rezultati i pohranjeni podaci za stvaranje GUI-a kroz biblioteku *PyQt5* i aplikaciju Qt dizajner.

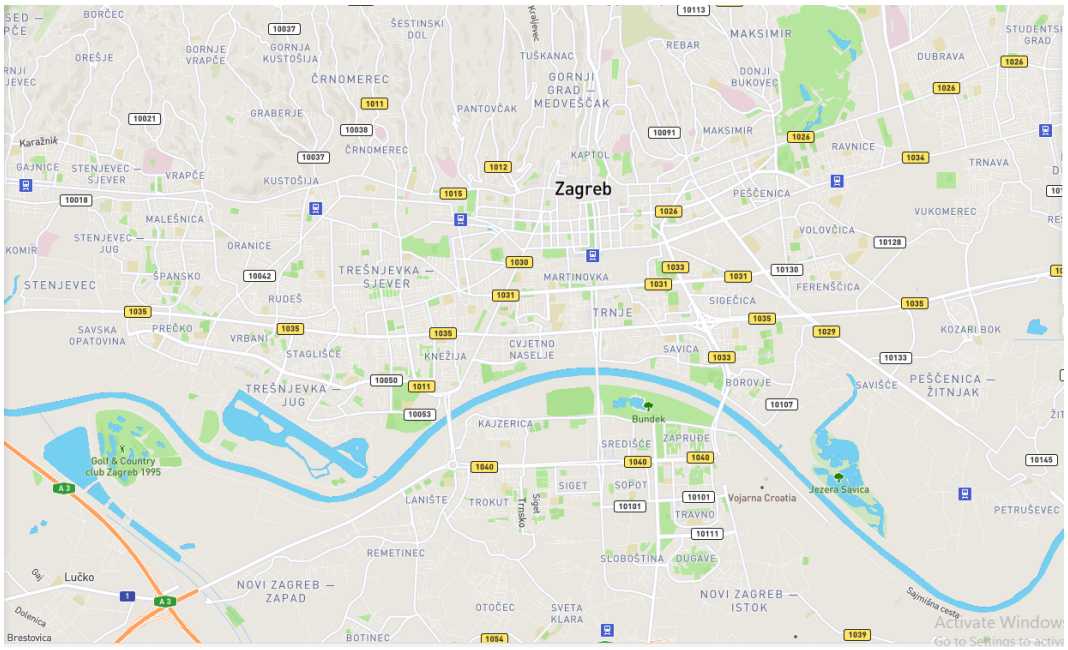
5.3. Stvaranje aplikativnog rješenja

Stvaranje GUI-a, odnosno aplikativnog rješenja odradit će se u aplikaciji Qt dizajner i Python biblioteci *PyQt5*. Prvi je korak stvaranje GUI-a u aplikaciji Qt dizajner koji se instalira

bibliotekom *PyQt5*. Slika 4.3 prikazuje napravljeni *GUI* koji će se koristiti za prikaz pohranjenih podataka u bazi. Svi *widgeti* vidljivi iz slike su:

- gumb za spajanje na relacijsku bazu podataka,
- sedam selektora koji predstavljaju dane u tjednu i moguće je odabrati samo jedan od njih sedam za svaki upit prema bazi,
- padajući izbornik koji prikazuje minutne intervale koje je moguće odabrati,
- četiri pravokutnika gdje će se ispisati srednja vrijednost, medijan, standardna devijacija, duljina linka i ograničenje brzine,
- dva prozora, gornji koji se nalazi desno od opisanih *widgeta* i gdje će se prikazati karta te donji prozor u kojem će se prikazivati grafički prikaz podataka za odgovarajući link kao što je grafički prikazan link -214695 na slici 5.1.

Kako bi napravljeni GUI bio upotrebljiv, potrebno je svakom od *widgeta* dodijeliti funkcionalnost kroz kôd, a upravo je to uloga bibliotete *PyQt5*. Prvo je potrebno pretvoriti stvoreni GUI u Python kôd, a taj kôd predstavlja imena, veličine i druge atribute pojedinog *widgeta* te se može uređivati automatski dobiveni kôd. Sljedeći je korak dodavanje karte u gornji prozor, moguće je postaviti statičnu sliku 5.4 ili stvoriti interaktivnu kartu cijelog svijeta što znatno komplicira stvari, jer za postizanje interaktivnosti karte potrebno je poznavanje programskih jezika HTML, Javascript i CSS. Odabran je kompliciraniji način, jer je interaktivna karta budućim korisnicima aplikacije znatno pristupačnija i ljepšeg izgleda. Najbolja praksa pri razvijanju i kasnijem održavanju kôda je razdvajanje sva četiri programskih jezika. Upravo tako je napravljena CSS datoteka (".css") u kojoj se opisuje stilski izgled varijabla u Javascriptu te kôd unutar HTML-a. Javascript datoteka (".js") je ključ interaktivne karte i komunikacije s Python kôdom. Najbitnije je unutar Javascripta stvoriti funkciju koja komunicira s Pythonom, funkciju za stvaranje karte te dodavanje karte otvorenog kôda. Dobar primjer karte otvorenog kôda je OpenStreetMap čiji je cilj stvaranje svima dostupne karte koju svatko može koristiti i dorađivati po vlastitoj potrebi, a drugi primjer je Mapbox koji se koristi u radu, jer posjeduje studio u kojem se mogu unijeti podaci u GeoJSON formatu i prikazati na karti. Primjer karte stvorene u Mapbox studiju je vidljiv iz slike 5.6 gdje se prikazuje grad Zagreb.



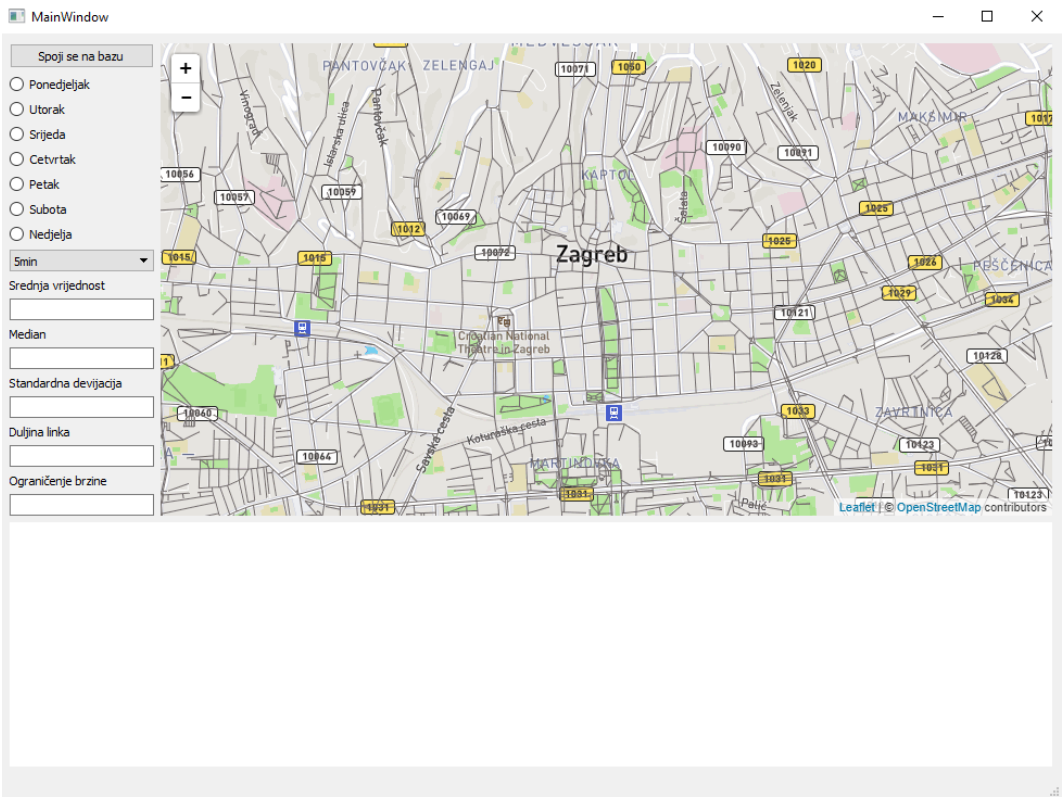
Slika 5.6: Prikaz karte stvorene u Mapbox studiju

Moguće je odabrati razne stilove prikaza karte, a isto tako i samostalno dodavati slojeve koji će prikazivati objekte na karti te im se može dodijeliti različite boje kako bi se bolje vidjeli, itd. Kada se dodaju linkovi na kartu u GeoJSON obliku koji su se obradili i pohranili u bazu podataka dobije se slika 5.7.

Slika 5.7: Prikaz karte stvorene u Mapbox studiju s dodanim slojem linkova

Može se reći kako prikaz linkova ometa vizualno, s obzirom na to da nije proveden algoritam podudaranja s kartom (engl. *map matching*), o kojem se može više pročitati iz izvora [31], linkovi se u nekim slučajevima ne nalaze na prometnici koja je prikazana na karti, a radi se o istoj prometnici. Kada je napravljena karta i dodana u Javascript potrebno je još dodijeliti funkciju koja će pritiskom lijeve tipke na mišu dohvatiti koordinatu oblika (geografska širina (x), geografska dužina(y)) i proslijediti do Python metode. Unutar HTML kôda povezuju se stvorene datoteke, dakle Javascript datoteke i CSS datoteka te se dobije datoteka oblika (".html") koja će se učitati u Python datoteku (".py").

U prvotno stvorenu Python datoteku (".py") koja je stvorena od pretvorbe napravljenog GUI-a u Qt dizajneru, stvara se samostalna klasa koja učitava HTML datoteku koja je lokalno pohranjena. Nije potrebno učitavati posebno Javascript, CSS i HTML datoteke, već samo HTML datoteku, jer ona povezuje ostale datoteke, a s obzirom na to kako je kôd segmentiran, lako ga je održavati i nadograđivati. Kada je učitan HTML u Python datoteku, karta postaje vidljiva unutar GUI-a kako je prikazano slikom 5.8.



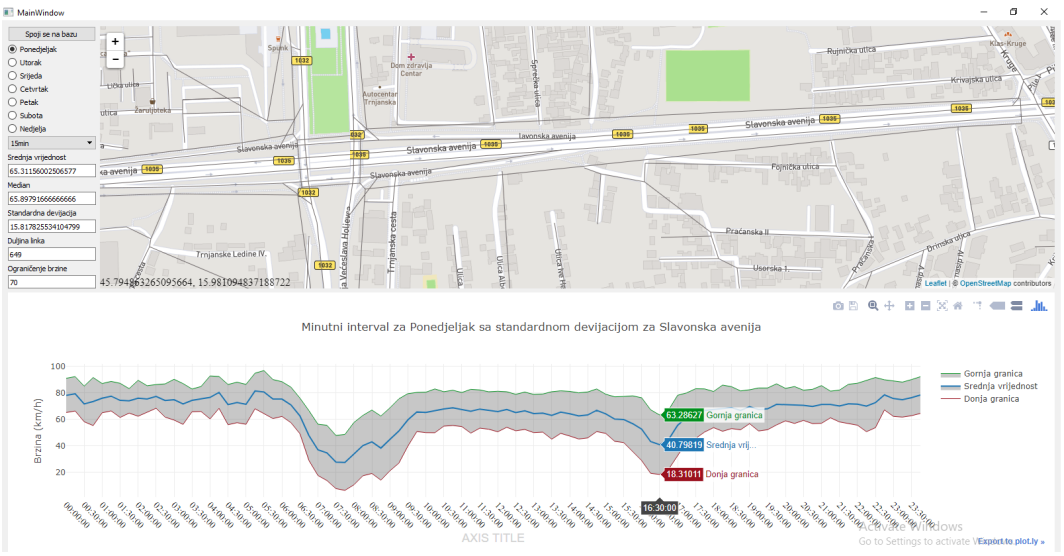
Slika 5.8: Prikaz GUI-a s implementiranom kartom

Sljedeći je korak animiranje gumba za spajanje na relacijsku bazu podataka kako bi se pritiskanjem gumba obavilo spajanje na bazu podataka, a za to se koristi biblioteka *SQLAlchemy*. Ako je spajanje uspješno prikaže se obavijest kako je spajanje na bazu uspješno, a ako je neuspješno prikaže da je došlo do pogreške pri spajanju na bazu podataka. Nakon pritiska gumba za spajanje, odabire se dan u tjednu te se sprema tekstualna vrijednost odabranoga gumba, a isti princip je i za odabir minutnog intervala iz padajućeg izbornika. Sljedeći je korak stvaranje metode koja prima dvije vrijednosti u decimalnom zapisu, a te vrijednosti su zapravo geografska širina i dužina. Zatim se povlače svi podaci iz tablice geo koji su prikazani na slici 5.5 i pohranjuju u *geopanas dataframe* pomoću automatske implementirane metode. Razlog povlačenja cijele tablice je taj što je potrebno usporediti koordinate koje su dobivene klikom miša sa svim koordinatama iz tablice geo i pronaći najbliži geometrijski objekt. Kako bi se pronašao najbliži geometrijski objekt izračuna se udaljenost od kliknute koordinate do svih linija odnosno linkova i pronađe najbliži link te se zabilježi IdLinka kao što je prikazano kroz kôd 5.4.

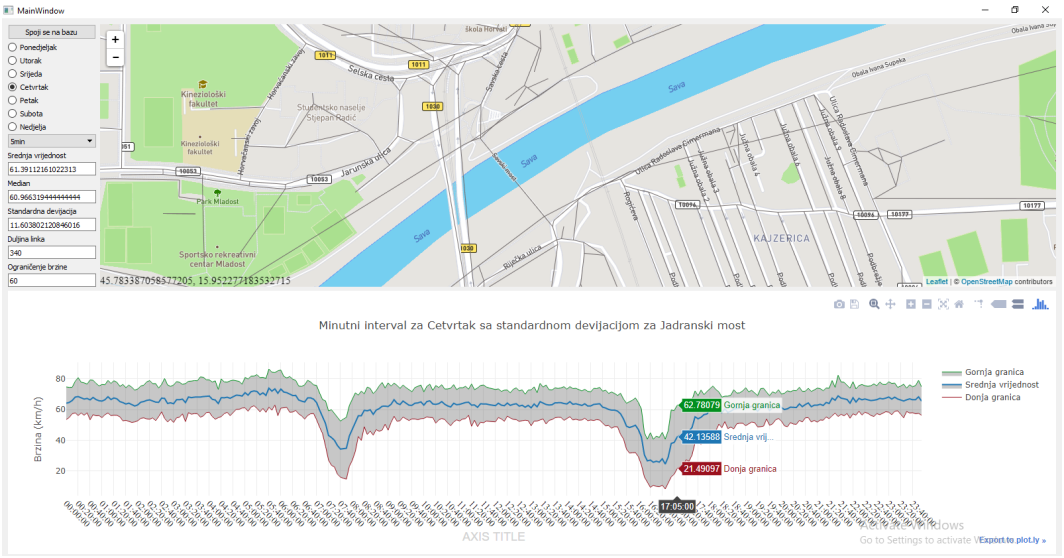
```
@QtCore.pyqtSlot(float,float)
    def pointClicked(self, x, y):
        sql = 'select * from geo'
        self.df = gpd.GeoDataFrame.from_postgis(sql, self.connection,
          ↪  geom_col='geometry')
        def min_dist(tocka, gpd2):
            self.df['Dist'] = self.df.apply(lambda row:
              ↪  tocka.distance(row.geometry),axis=1)
            geoseries = self.df.iloc[gpd2['Dist'].idxmin()]
            return geoseries['IdLink']
        tocka=Point(y,x)
        self.brojka=min_dist(tocka,self.df)
        broj=str(self.brojka)
```

Kôd 5.4: Python kôd u biblioteci *PyQt5* koji omogućava asinkronu komunikaciju sa Javascriptom i dohvaćanje kliknutih koordinati

Time se dobiju sve potrebne varijable kako bi se mogao napraviti novi upit prema bazi podataka (PostgreSQL-u) koji izgleda ovako: `SELECT * FROM ponedjeljak_15min where "IdLink" = -214695`, a ponedjeljak u upitu može biti i drugi dan u tjednu, ovisno što korisnik odabere. Isto vrijedi i za 15 min, može biti i drugi minutni interval, a IdLink se dobije iz kôda 5.4. Taj upit dohvaća redak iz odgovarajuće tablice za definirani IdLinka i pohranjuje ih u *pandas dataframe* te se razdvajaju srednja brzina, medijan i standardna devijacija kako bi se dobio isti *pandas dataframe* kao na slici 3.6d. Zapravo se radi obrnuti proces koji je opisan u potpoglavlju 5.1, ali bez filtracije, jer podaci koji su pohranjeni su već filtrirani. Kada se dobije takav oblik *pandas dataframea* onda je moguće u drugi prozor dodati i grafički prikaz. Finalni produkt je prikazan na slikama 5.10 i 5.9 gdje su vidljivi i podaci unutar pravokutnika, odnosno srednja vrijednost, medijan, standardna devijacija, duljina linka te ograničenje brzine.

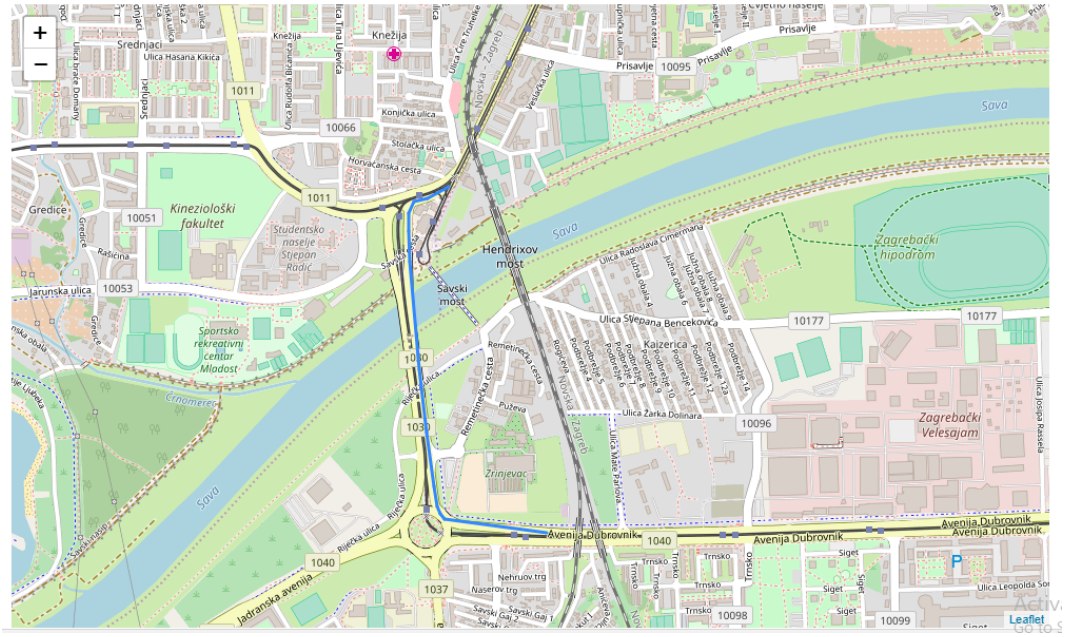


Slika 5.9: Prikaz aplikacije za link -218749, odnosno Slavonska avenija



Slika 5.10: Prikaz aplikacije za link -214695, odnosno Jadranski most

Također, napravljena je kratka animacija kako aplikacija radi, a moguće ju je vidjeti na internet stranici: `https://imgur.com/a/SyItOvh`. Komunikacija koja se odvija između Javascripta i Pythona, odnosno biblioteke *PyQt5* je asinkrona, što znači da nije moguće upisivati podatke u Javascript putem *PyQt5* već je potrebno podići server koji bi bio posrednik za komunikaciju. Putem servera bi se mogla ostvariti sinkrona komunikacija i time bi se mogla poboljšati ova aplikacija, označiti link drugom bojom, itd. Također, moguće je bibliotekom *folium* prikazati rute. Potrebno je učitati tekstualnu datoteku s brojevima linka pomoću *pandas dataframea* i dobije se prikaz kao na slici 5.11.



Slika 5.11: Prikaz rute od 12 linkova

6. Izračun razine pouzdanosti

Izračunati razine pouzdanosti brzina, teoretski je moguće izračunati na više načina, a s obzirom na postojeće podatke za izračun razine pouzdanosti brzine koristiti će se standardna devijacija. Prije samog izračuna razine pouzdanosti brzine, objasnit će se ukratko teoretski razina pouzdanosti. Izračunata je samo razina pouzdanosti za definirane rute, jer računati samostalno za svaki link nema prevelikog smisla. Korisnike uglavnom neće zanimati razina pouzdanosti samo jednog linka, već tokom cijele rute po kojoj namjeravaju proći. Glavna ideja je pronaći srednju vrijednost cijele rute u određenom dobu dana, ovisno o minutnom intervalu te najbitnijem parametru, a to je standardna devijacija. Standardna devijacija govori koliko je odstupanje od srednje vrijednosti, a to je ključan parametar pomoću kojeg se može procijeniti pouzdanost izračunate srednje brzine. Drugim riječima, s kolikom sigurnošću se može reći da je izračunata srednja brzina upravo ta brzina u određenom dobu dana, a to se može izraziti postotcima. Primjer je koliki je postotak da će u četvrtak u 16 sati i 30 minuta biti izračunata srednja brzina od 32,99 km/h za minutni interval od 5 min. Kako bi dobili te rezultate koriste se sljedeći izrazi (6.1):

$$\begin{aligned} Gornja &= \overline{x} + \sigma \\ Donja &= \overline{x} - \sigma \\ Pouzdanost &= \frac{\left(\frac{\overline{x}}{Gornja} + \frac{Donja}{\overline{x}}\right) \cdot 100}{2} \end{aligned} \tag{6.1}$$

gdje je:

- Gornja – predstavlja gornju granicu kao zbroj srednje vrijednosti i standardne devijacije [km/h],
- Donja – predstavlja donju granicu kao razliku srednje vrijednosti i standardne devijacije [km/h],
- $\overline{x}$ – aritmetička sredinu, tj. srednju vrijednost [km/h],
- σ – standardna devijacija populacije,
- Pouzdanost – izražena u postocima [$\%$].

Za početak je potrebno dohvatiti podatke za definiranu rutu, a primjer rute koja će se obraditi se sastoji od 12 linkova, prikazanih na slici 5.11. Kada se dohvate podaci, računa se srednja vrijed-

nost pojedinog linka, ali kako bi se dobila srednja vrijednost rute, potrebno je poznavati sljedeće zakonitosti. Svaki link se gleda kao slučajnu varijablu koja posjeduje očekivanje i varijancu. S pretpostavkom kako svaki link rute posjeduje normalnu distribuciju podataka te su distribucije međusobno nezavisne korelacija linkova. Moguće je sumirati prosječne brzine svakog linka i dobiti prosječnu brzinu rute. U slučaju da svi linkovi unutar rute nemaju približno normalnu distribuciju, nije moguće sumirati slučajne varijable, odnosno linkove te nije moguće dobiti ispravne rezultate. Izraz (6.2) dokazuje zakonitost prema kojoj je moguće sumirati slučajne varijable, tj. linkove.

$$X_i \sim N(\mu_i, \sigma_i^2), \qquad i = 1, \ldots, n$$

ako nezavisne varijable

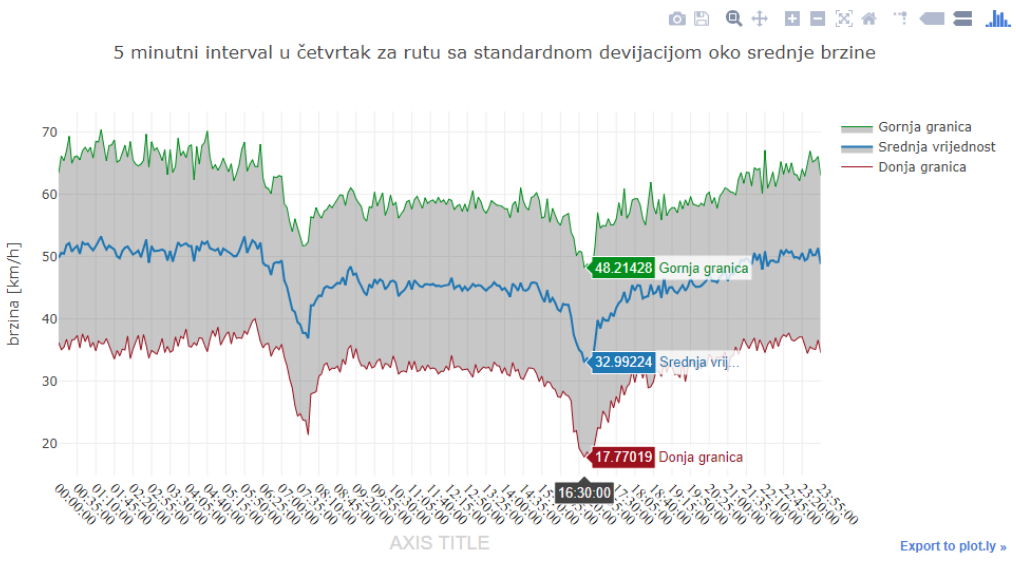
$$\{X_1, X_2, \ldots, X_n\}$$

imaju normalnu distribuciju, onda:

$$\sum_{i=1}^{n} X_i \sim N\left(\sum_{i=1}^{n} \mu_i, \sum_{i=1}^{n} \sigma_i^2\right) \tag{6.2}$$

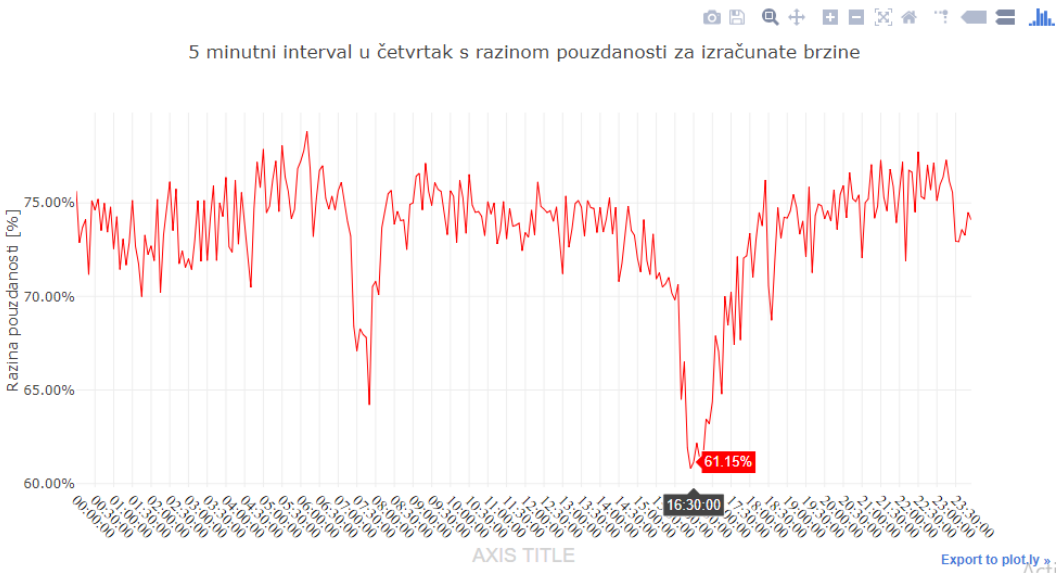
gdje je:

- X_i – nezavisna varijabla,
- σ_i^2 – varijanca $[m]$,
- μ_i – očekivanje,
- N – oznaka za normalnu distibuciju,
- n – broj vrijednosti u skupu.



Slika 6.1: Prikaz srednje vrijednosti kroz doba dana za linkove s rute

Kada je dobiven *pandas dataframe* sa srednjim brzinama i standardnom devijacijom za sve linkove koji su dio rute, kao i cjelokupne rute, onda se realizira izračun razine pouzdanosti po uzoru na izraz 6.1 te se dobiju rezultati grafički prikazani na slici 6.2.



Slika 6.2: Prikaz izračunate razine pouzdanosti brzine za rutu od 12 linkova

Razina pouzdanosti za četvrtak u 16 sati i 30 minuta je 61,15 % te se može reći s pouzdanošću da će brzina biti 33 km/h, što drugim riječima znači kako brzina varira, ali je uglavnom oko 30 km/h. Na nekim dijelovima rute brzina će biti manja, na nekima veća, ali prosjek je oko 30 km/h za četvrtak u 16 sati i 30 minuta s lošijom pouzdanošću. To doba dana pripada u takozvane vršne sate gdje je najviše prometnih entiteta na prometnicama, zbog odlaska s posla, a zbog toga je i najmanja pouzdanost, jer prometni entiteti dolaze u valovima i u jednom trenutku je moguća normalna brzina, a u drugom male brzine ili zagušenja. Sa slike 6.2 vidljivo je kako u vršnim satima prometni entiteti, odnosno ljudi odlaze na posao u jutarnjim satima približno od 7 sati do 9 sati , a odlaze s posla u intervalu od 15 sati do do 18 sati, a upravo tu su najvarijabilnije brzine zbog periodičnih gužvi u prometu. U ostalim dobima dana vidljivo je kako je pouzdanost uglavnom preko 70 % i s većom sigurnošću se može reći da će brzina biti prosječna/srednja brzina.

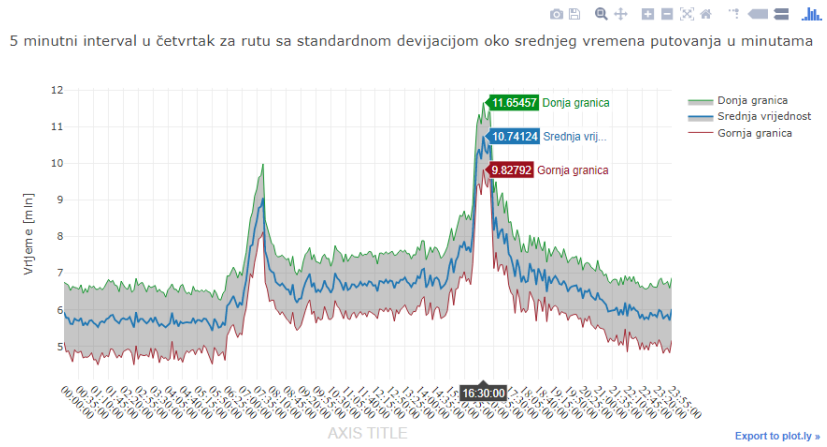
Razlog izračuna razine pouzdanosti je mogućnost odabira rute koja nudi najbolju opciju za korisnika kao omjer prosječne brzine i visoku razinu pouzdanosti. Osim toga, moguće je izračunati i vrijeme putovanja u Pythonu tako što se prvo izračuna duljina rute, sumirajući duljinu svakog linka, a duljina linkova se preuzima iz drugog skupa podataka (slika 5.3). Kada su nezavisne varijable normalno distribuirane, moguće je izračunati vrijeme putovanja. Vrijeme putovanja se računa prema izrazu (6.3):

$$t = \sum_{i=1}^{n} \frac{\frac{s_i}{1000}}{\frac{\overline{x_i}}{3.6}} \cdot 60 \tag{6.3}$$

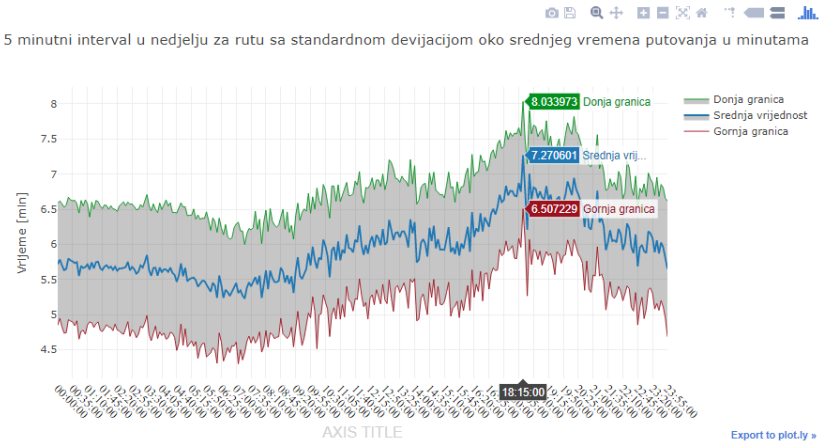
gdje je:

- t – vrijeme (putovanja) $[min]$,
- s_i – oznaka za duljinu pojedinog linka $[m]$,
- $\overline{x_i}$ – srednja vrijednost brzine po pojedinom linku $[km/h]$.

Kada se u izraz (6.3) uvrsti jedna srednja vrijednost, onda je ta vrijednost samo jednog linka. Upravo tome služe programski jezici kao što je Python, gdje nije potrebno ručno računati za svaki link, već se iskoristi ovaj ozraz i iterativno prođe za sve linkove koji su dio rute i dobije srednje vrijeme putovanja rute kroz cijeli dan. Dodatno se mogu izračunati gornja i donja granica, tako što se u izraz (6.3) u nazivniku doda izračunata prosječna vrijednost standardne devijacije za rutu. Gornja granica se dobije tako što se srednja vrijednost oduzme za standardnu vrijednost. Donja granica se dobije dodavanjem standardne devijacije srednjoj brzini, što je obrnuta logika, jer gornja granica predstavlja najduže vrijeme putovanja, a donja granica najkraće vrijeme putovanja. Grafički prikaz će bolje dočarati prije objašnjeno, a razlika između radnog dana i neradnog dana je vidljiva iz slike 6.3a za četvrtak i slike 6.3b za nedjelju.



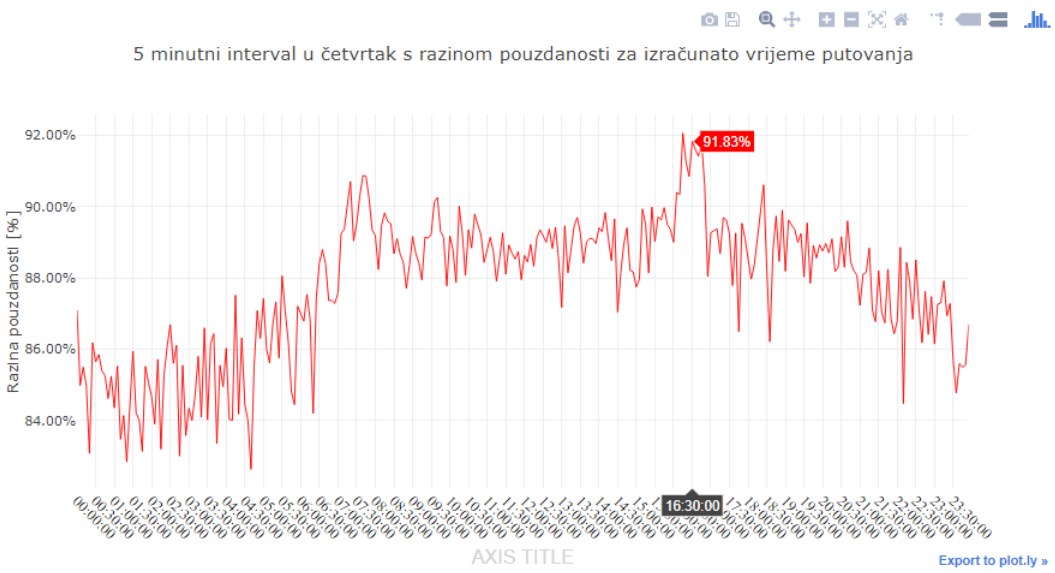
(a) Grafički prikaz vremena putovanja za četvrtak



(b) Grafički prikaz vremena putovanja za nedjelju

Slika 6.3: Srednje vrijeme putovanja sa standardnom devijacijom

Iz slika je vidljivo kako je veća devijacija, odnosno varijabilnost za nedjelju, ali je manje vrijeme putovanja nego za četvrtak. Što je i logično, jer u nedjelju ima manje prometnih entiteta, voze svojim tempom i brzinom, ali je potrebno poštivati ograničenje brzine. Također, može se zaključiti kako je u nedjelju, odnosno za neradne dane vrijeme putovanja uglavnom uravnoteženo, nema naznaka vršnih sati. Predvečer je vidljivo povećanje, a pretpostavlja se da je razlog odlazak ljudi svojim domovima. Međutim, kada se izračuna razina pouzdanosti za vrijeme putovanja u četvrtak i grafički prikaže, dobije se veća razina pouzdanosti u vršnim satima kao što se vidi iz slike 6.4. Razlog veće razine pouzdanosti vremena putovanja u vršnim satima je zbog konstantnog prometa tokom dana, dok u večernjim i jutarnjim satima ima izrazito malo prometa. Ovisno o vozačima prometnih entiteta, neki će voziti umjerenom brzinom, neki većom, a neki malom brzinom, tako da je manja pouzdanost vremena putovanja. Iako je potrebno naglasiti kako su male razlike u vremenu putovanja, odnosno mala je razlika između vršnih sati i noćnih, radi se o desetinkama ili stotinkama.



Slika 6.4: Prikaz izračunate razine pouzdanosti vremena putovanja za rutu od 12 linkova

Time je završen izračun razine pouzdanosti i analiza dobivenih rezultata. Potrebno je naglasiti kako se može izračunati i razina pouzdanosti brzine za svaki link, međutim, ima više smisla izračunati pouzdanost brzine za rutu, jer prometni entitet prometuje preko više linkova, od izvora do odredišta.

7. Studij slučaja

Ratko Zelenika definira metodu studija slučaja kao postupak kojim se izučava neki pojedinačni slučaj iz odabranog znanstvenog područja, međutim, ne predstavlja znanstvenu metodu u pravom smislu te riječi, jer se samo promatranjem više slučajeva mogu izvući određene zakonitosti iz dobivenih rezultata. Nastoji se sagledati problematiku iz svih aspekata, a za to je potrebno uključiti sve varijable, mogućnosti i potencijalna rješenja kako bi se steklo razumijevanje o dobivenim rezultatima. Razlika metode studije slučaja u odnosu na druge kvantitativne metode istraživanja je to što se postavljaju pitanja kako i zašto, što su ključna pitanja za problematiku ovog diplomskog rada. Studij slučaja koristi istraživaču kada ima minimalnu kontrolu nad događajima, što često podrazumijeva stvarne probleme stavljenim u specifičnom okruženju, [32]. Prema izvoru [33], navodi se šest koraka koje bi trebalo slijediti, a tri koraka su direktno vezana uz ovaj diplomski rad, a to su:

1) Prikupljanje podataka,

2) Analiza podataka,

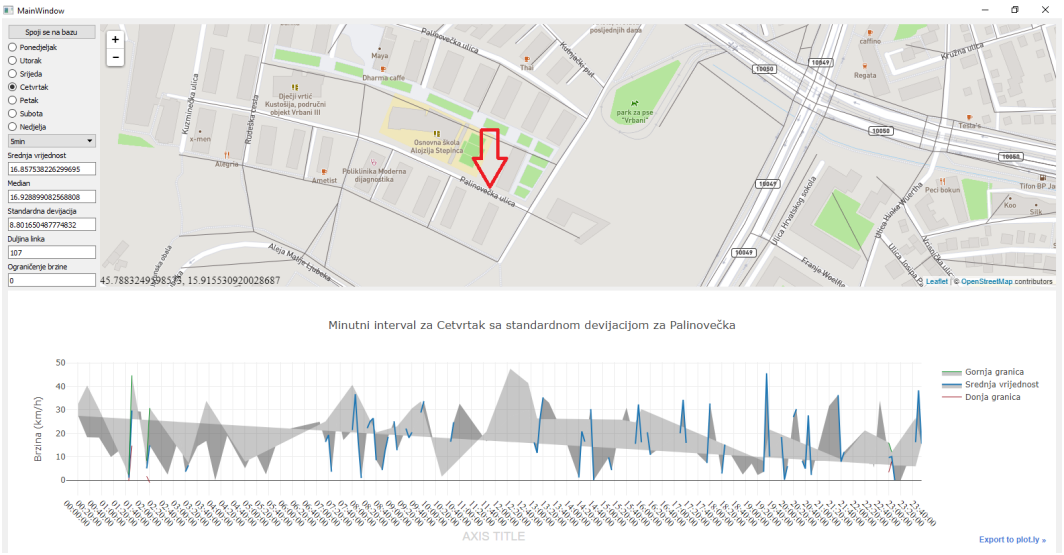
3) Pitanja vrijednosti i pouzdanosti.

Tako je ovo poglavlje strukturirano u tri potpoglavlja u kojima su uspoređeni dobiveni rezultati te kako bi se mogli poboljšati i unaprijediti dobiveni rezultati.

7.1. Studija slučaja – prikupljanje podataka

Prilikom prikupljanja podataka koji će se koristiti za analizu, jedna je tvrdnja uvijek istinita, a to je tvrdnja što više podataka to su bolji rezultati. Posjedovanje velike količine podataka za svaki link omogućava smanjivanje minutnih intervala i dobivanje preciznijih prosječnih brzina. Primjerice kada se pogleda slika 5.1 vidi se kako je standardna devijacija $\sim 18,4\ km/h$, ali kada se minutni interval smanji na 1 min standardna devijacija je $\sim 16,6\ km/h$. Kada se minutni interval još smanjuje dobije se još manja standardna devijacija, ali ne smanjuje se drastično, a razlog je sve manji broj brzina. Moguće je napraviti i kontraefekt te će se dobiti pogrešna srednja brzina, jer će se promatrati mali broj brzina, a moguće je da te brzine ne ocrtavaju

realno stanje. Već pri minutnom intervalu od 1 min neki od tih intervala posjeduju manje od 10 zapisa, odnosno manje od 10 različitih brzina. Kada bi bilo više podataka moguće je da bi se standardna devijacija smanjila, jer statistički gledajući bi prevladala brzina koja se najčešće pojavljuje u tom danu u to doba dana. Kada postoji samo 10 brzina za pojedini interval to je iznimno malen uzorak i srednja brzina i dalje neće imati dobru pouzdanost, odnosno neće se prevladavajuća brzina koja je najčešća istaknuti i time smanjiti standardnu devijaciju. Dakle, to je i razlog ne korištenja manjih minutnih intervala od 5 min, pogotovo kada se uzme u obzir kako su prije dobiveni spomenuti rezultati za jedan od linkova s najvećim brojem podataka. Primjer linka za 5 min interval koji ima znatno manji broj zapisa od linka -214695 (cestovni segment Jadranskog mosta) je prikazan slikom 7.1. Ovo je i dobar primjer korisnosti stvorene aplikacije gdje se na jednostavan način mogu pronaći linkovi s malim brojem podataka, a graf prikazuje rezultate malog broja podataka, gdje je graf jednostavno neprepoznatljiv i praktički je nemoguće donijeti zaključke na temelju njega. Razlog je što je prometnica u naseljenom dijelu i nije dio onih prometnica po kojima prolazi velik broj prometnih entiteta tijekom dana. Iako bi bilo korisno posjedovati podatke i za tu prometnicu (nije nužno), ali bez obzira na dob dana vjerojatnost gužve je praktički nepostojeća. Slični su i grafovi za ostale cestovne segmente u ulici Palinovečka, kao i za okolne cestovne segmente koji su dio stambenog područja.



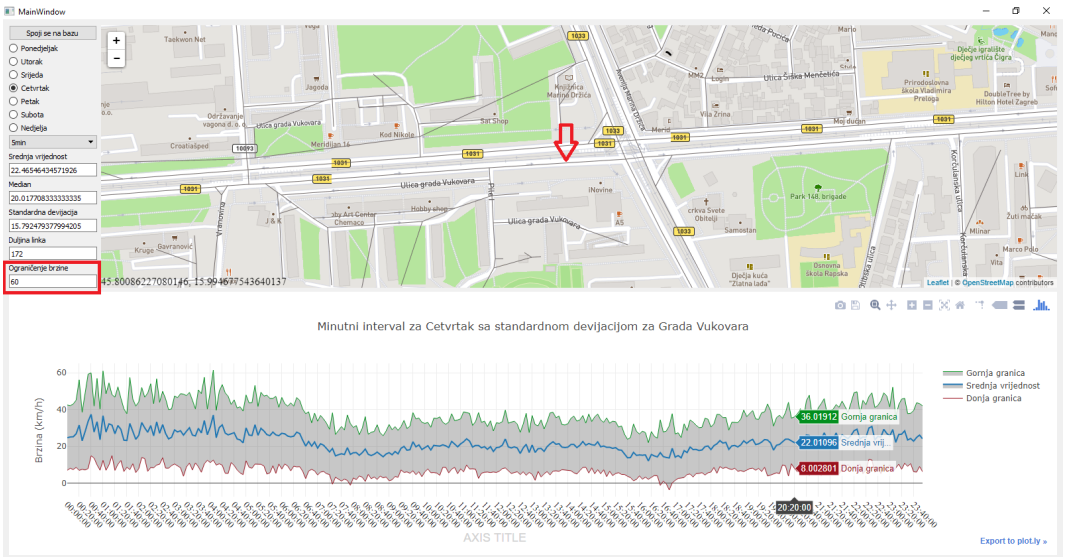
Slika 7.1: Prikaz linka u aplikaciji s malim brojem podataka

Zaključak je da je potrebno prikupiti veći skup podataka, kako bi se mogli dobiti što precizniji rezultati unutar manjih minutnih intervala za kritične linkove koji su najkorišteniji u dnevnim migracijama prometnih entiteta. Također, poželjno bi bilo idealno prikupljati podatke od što većeg broja prometnih entiteta, odnosno više od 4.200 vozila korištenih u projektu SORDITO. Potrebno je i naglasiti kako se u ovom slučaju proučava samo prikupljanje podataka na području grada Zagreba. Iskustveno je poznato kako je količina podataka najveća na području grada Zagreba, no pitanje je koliko je podataka prikupljeno u manjim mjestima, primjerice Ku-

tina, Požega, Vir, itd. Gdje također mogu postojati gužve u nekim dijelovima dana ili godine. Primjerice grad Vir će sigurno u ljetnim mjesecima biti znatno prometniji u određenim minutnim intervalima, a korisniku može pomoći znanje kada su moguće gužve, mala brzina te može zaobići kritični link/rutu i otići na alternativni link/rutu.

7.2. Studija slučaja – analiza podataka

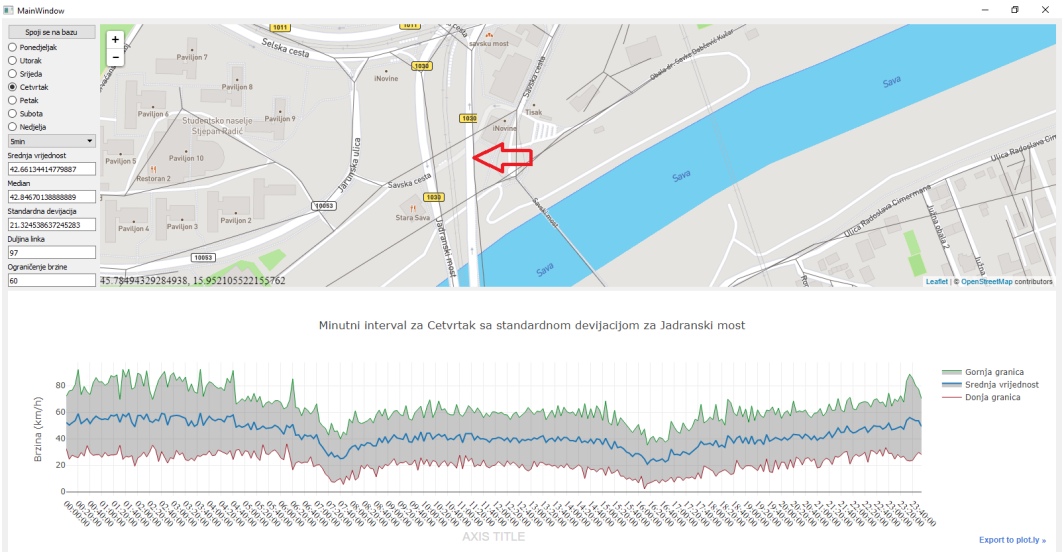
Prilikom analize podataka potrebno je prvo razumijeti podatke nad kojima se izvodi analiza podataka te njihova distribucija. Distribucija podataka je iznimno bitna kod filtriranja podataka, jer ovisno o distribuciji podataka odabrat će se metoda za filtriranje podataka. Podaci o brzinama prometnih entiteta teže normalnoj distribuciji te s tom pretpostavkom se koristi MAD metoda za filtriranje podataka. Ograničenje brzine na linku često definira i prosječnu brzinu prometnih entiteta. Međutim, zbog mogućih zagušenja u prometnom sustavu, točnije na specifičnim linkovima u vršnim satima, smanjuje se prosječna brzina ispod ograničenja brzine. Analizom podataka moguće je detektirati točne periode smanjenja prosječnih brzina, samim time i zagušenja, odnosno gužve na prometnicama. Neki cestovni segmenti su opterećeni do te mjere da im je u cijelom danu prosječna brzina znatno manja od ograničenja brzine. Primjer je link u ulici grada Vukovara prikazan u aplikaciji na slici 7.2, gdje je prosječna brzina uvijek ispod ograničenja brzine, čak i vikendom.



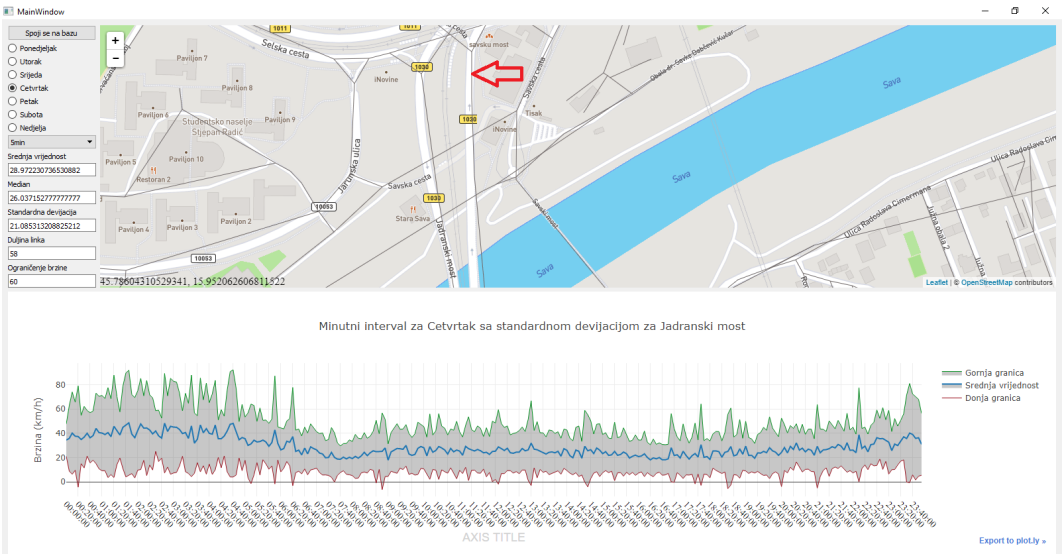
Slika 7.2: Prikaz brzina za link u ulici grada Vukovara

Razlozi tako niskih brzina su centar grada, prometnica preko koje se horizontalno povezuje cijeli grad i okolica, a drugi razlog je što upravo taj link dolazi do velikog križanja te tu nastaju gužve, odnosno čekanje na semaforu. Analizom podataka link se pokazao kao loša sastavnica rute te je potrebno predložiti alternativnu rutu koji bi zaobišla ovakve kritične linkove. Za prijedlog alternativne rute, potrebno je prikupiti podatke upravo za one linkove koji su naglašeni

kao linkovi s malim brojem podataka. To su sporedni linkovi koji mogu ubrzati put od izvora do odredišta, ako postoji zadovoljavajući broj podataka. Bez zadovoljavajućeg broja podataka nije moguće donijeti niti preciznu analizu podataka niti predložiti alternativne rute. Analizom dobivenih rezultata i jednostavnim prikazom rezultata u aplikaciji moguće je detektirati linkove na ruti s malim brojem podataka te s dodatnim prikupljanjem podataka mogu se poboljšati trenutni rezultati. Definira se ruta i zatim se u aplikaciji mogu pogledati pojedini linkovi i kakva im je brzina sa standardnim devijacijama. Iz rute definirane slikom 5.11, razlika između pojedinih linkova je vidljiva usporedbom dva linka: link neposredno prije semaforiziranog raskrižja (slika 7.3b) i drugim linkom koji prethodi prvom linku (slika 7.3a).



(a) Grafički prikaz linka Jadranski most prije križanja na ruti



(b) Grafički prikaz linka Jadranski most povezan s križanjem na ruti

Slika 7.3: Primjer brzina dva linka na Jadranskom mostu

Brzine linkova koji prilaze velikim križanjima imat će manje prosječne brzine, upravo zbog nakupljanja velikog broja vozila i čekanja na prometnu signalizaciju. Za optimizaciju tih linkova, moguće je analizirati kvalitetu križanja i njegove prometne signalizacije te na temelju podataka poboljšati i prosječne brzine kritičnih linkova. Analiza koja je korištena za dobivanje rezultata je proširiva, međutim, zbog opsega diplomskog rada, vremenski jednostavno nije bilo moguće proširiti moguću analizu i metode za dobivanje alternativnih rezultata. Primjer proširenja analize je uključivanje godišnjih doba pri grupiranju podataka, čime se mogu dobiti dani u tjednu s minutnim intervalima te selekcija po godišnjim dobima, što bi svakako pridonijelo preciznijim rezultatima. Očekivano je kako će brzine na području grada Zagreba biti veće u ljetnim mjesecima, a manje u zimskim mjesecima, ali još bitnije je grupiranje prema godišnjim dobima za mjesta i gradove u Dalmaciji i Istri, jer turizam značajno utječe na zagušenja u prometu. U tim područjima velika je razlika između ljeta i drugih godišnjih doba.

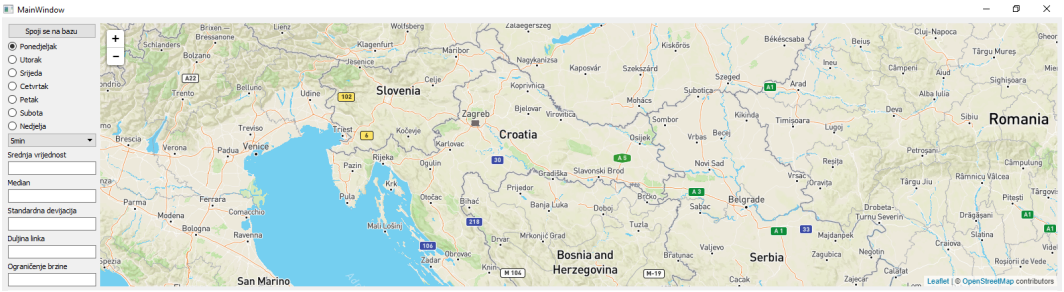
Druga mogućnost kod analize podataka je korištenje strojnog učenja za filtriranje podataka, a ponajviše Python biblioteke *scikit-learn* pomoću koje se mogu iskoristiti algoritmi strojnog učenja za detekciju *outliera*. Jedna od metoda je linearna regresija, odnosno smještanje linije u koordinatnom sustavu tako da najbolje odgovara svim podacima, odnosno tako da je maksimalan broj podataka što bliži liniji, tj. minimalna udaljenost što većeg broja podataka. Prema izvoru [34], koji je i ujedno službena dokumentacija od biblioteke *scikit-learn* postoji još načina detekcije *outliera*, a zasniva se na grupiranju podataka u obliku šume (engl. *forest*). Neće se ići u detalje vezano za algoritme strojnog učenja za pronalazak *outliera*, jer nisu korišteni. Pretragom službene dokumentacije za *scikit-learn* i primjera moguće je vidjeti teoretski dio i praktične primjere. Osim za detekciju *outliera* strojno učenje se može koristiti i za stvaranje prediktivnih modela. Što upravo to dobiveni podaci i nastoje stvoriti, međutim, strojno učenje donosi novu dimenziju gdje se mogu implementirati razne druge varijable za kvalitetniji prediktivni model. Primjer varijabli koje mogu poboljšati prediktivni model je utjecaj vremenskih (ne)prilika, događaja kao što su koncerti, nesreće, blokirani dijelovi prometnica, itd. Kako bi se dohvatio takav tip podataka potrebno je ugraditi dodatne senzore u sama vozila koji generiraju GPS zapise koji se pohranjuju u bazu ili je potrebno napraviti jedan kompletan inteligentni informacijski sustav. Taj sustav bi podacima analiziranim na prezentirani način u diplomskom radu, dodao iz drugih izvora trenutnu vremensku prognozu i događaje u okolici zabilježenog GPS zapisa. To je iznimno kompleksan sustav, koji postoji u nekim državama svijeta i omogućio bi nadzor i preusmjeravanje prometa, kao i mogućnost preciznije predikcije prosječnih brzina za doba dana. Time bi se sigurno povećala razina pouzdanosti.

Zadnje unaprijeđenje dobivenih rezultata bi bilo provesti algoritam podudaranja s kartom, tj. *map matching* algoritam koji bi dobivene geometrijske objekte, odnosno linije pridružio stvarnim prometnicama na karti. Kroz projekt SORDITO korišteni su GPS uređaji s preciznosti, odnosno odstupanje od točne pozicije, s procjenom na 10-ak metara. Zbog toga je potrebno

provesti *map matchin* algoritam koji će pridružiti link odgovarajućoj prometnici.

7.3. Studija slučaja – pitanja vrijednosti i pouzdanosti

Vrijednost aplikacije koja je stvorena je velika, jer se primjenom istih postupaka mogu dobiti informacije o brzini i pouzdanosti i o drugim gradovima, a ne samo za Zagreb. Dobiveni rezultati možda nisu primjerice toliko korisni privatnim korisnicima, već poslovnim korisnicima koji prevoze svoje proizvode ili im je glavna djelatnost distribucija proizvoda. Primjer organizacija, odnosno poslovnih korisnika kojima ovakva aplikacija može pomoći su pošta, logistička poduzeća, uglavnom bilo koja organizacija koja posjeduje poseban odjel za dostavu. Mogu planirati koje rute koristiti, tj. koje rute daju najbolji omjer brzine i razinu pouzdanosti za vrijeme u danu kada se obavlja dostava te tako smanjiti troškove dostave. Mogu smanjiti potreban broj dostavnih vozila, a da pri tome ostane ista kvaliteta dostave. Što se tiče same aplikacije, ona je nadogradiva u smislu kako je ovo prva verzija koja se pokreće lokalno bez *web* stranice na kojoj se pokreće aplikacija. Kada bi se aplikacija postavila na *web* stranicu, moglo bi se pomoću Ajaxa ostvariti dvostruka komunikacija između *PyQt5* i Javascripta te bi se time mogao označiti i istaknuti pritisnuti link u aplikaciji. Velika odlika je što je moguće smještanje podataka za cijelu Hrvatsku, a i za cijeli svijet kao što je vidljivo iz slike 7.4. Kako

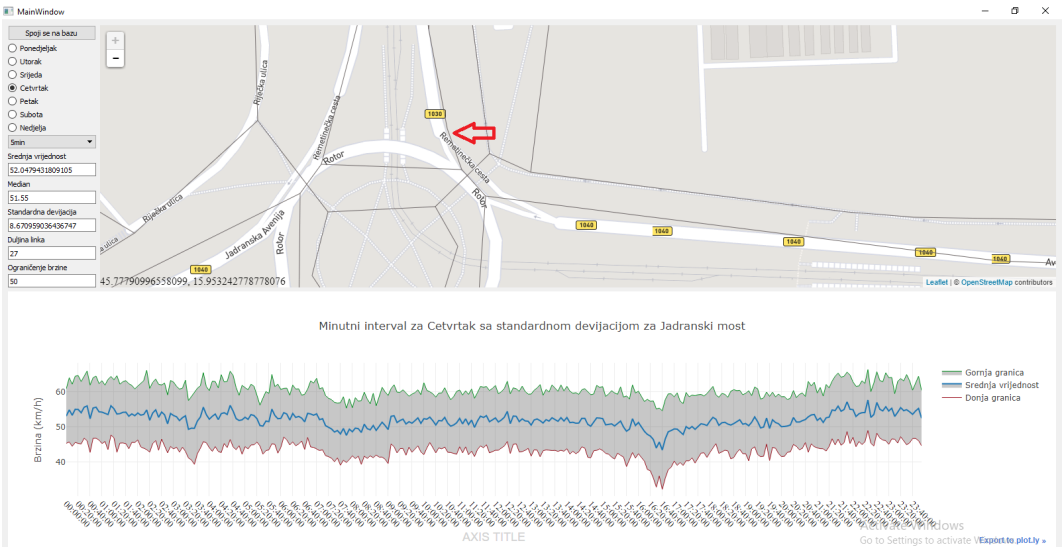


Slika 7.4: Prikaz interaktivne karte sa smanjenim zumom

bi se smjestili drugi podaci na kartu, potrebno je posjedovati dva skupa opisana u tablicama 2.1 i 2.2, filtrirati, obraditi i pohraniti ih u bazu te ih smjestiti na interaktivnu kartu. Sve metode za filtriranje, obradu i pohranu podataka te programska podrška su napravljeni za ponovnu uporabu i dodavanje novih podataka. Svrha stvorene informacijsko-komunikacijske programske podrške je ponovna uporaba i proširivanje rezultata, odnosno dodavanje svih linkova koji se nalaze u Republici Hrvatskoj, a po potrebi i proširenje na više zemalja. Upravo ponovna upotreba je velika i neupitna vrijednost dobivenih rezultata.

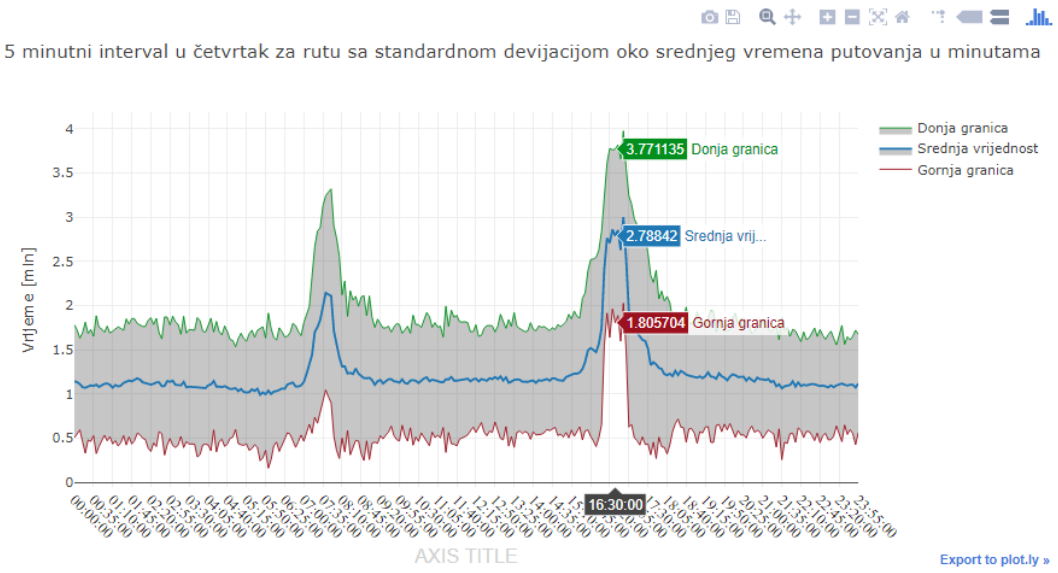
Veliku vrijednost predstavljaju informacije koje je moguće dobiti putem stvorene aplikacije. Kao što je vidljivo iz slike 7.5, gdje se vidi srednja vrijednost, medijan i standardna devijacija brzine linka za cijeli dan. Srednja vrijednost, medijan i standardna devijacija prikazana u aplikaciji su uprosječene vrijednosti kroz cijeli dan za odabrani link. Također, aplikacija daje na uvid

duljinu i ograničenje brzine linka. Pomoću duljine linka može se procijeniti gdje je potrebno pritisnuti za prikazivanje susjednih linkova. S obzirom na to da interaktivna karta pritiskom pokazivača ne osvjetljava bojom početak i kraj linka, potrebno je procijeniti gdje se nalazi idući link. Dok ograničenje brzine omogućava usporedbu dobivenih rezultata, odnosno dobivenih prosječnih brzina s ograničenjem brzine. Iz slike 7.5 vidljivo je da brzina kroz dan poprima vrijednosti približno jednake ograničenju brzine od 50 km/h, dok je sa slike 7.3b vidljivo da je brzina kroz dana značajno manja od postavljenog ograničenja brzine. Idealno bi bilo kada bi prometni entiteti prometovali oko brzine ograničenja, što bi značilo da prometni sustav funkcionira odlično!

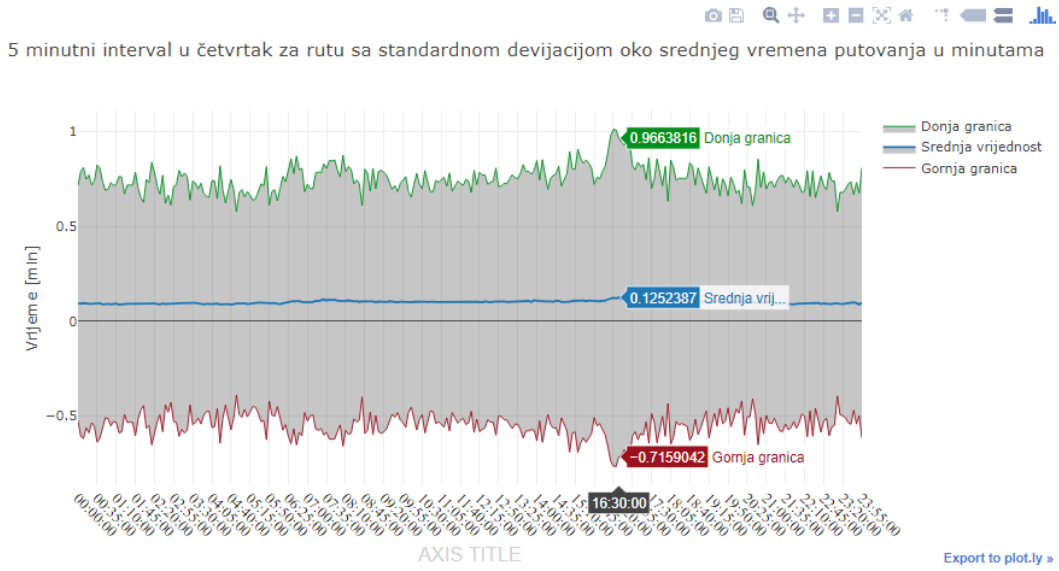


Slika 7.5: Prikaz stvorene aplikacije

Kroz stvorenu programsku podršku, moguće je usporediti linkove koje su dio rute te vidjeti pojedino vrijeme putovanja za svaki link. Na slici 7.6 može se vidjeti razlika vremena putovanja između najdužeg i najkraćeg linka koji se nalazi na definiranoj ruti. Slika 7.6b stvorena programskoj podršci nije sasvim točna, a razlog je malo vrijeme putovanja, a velika standardna devijacije. Donja granica ide u minus, a vrijeme ne može biti negativno, tako da ovaj graf treba gledati do nule.



(a) Prikaz vremena putovanja za najduži link u ruti

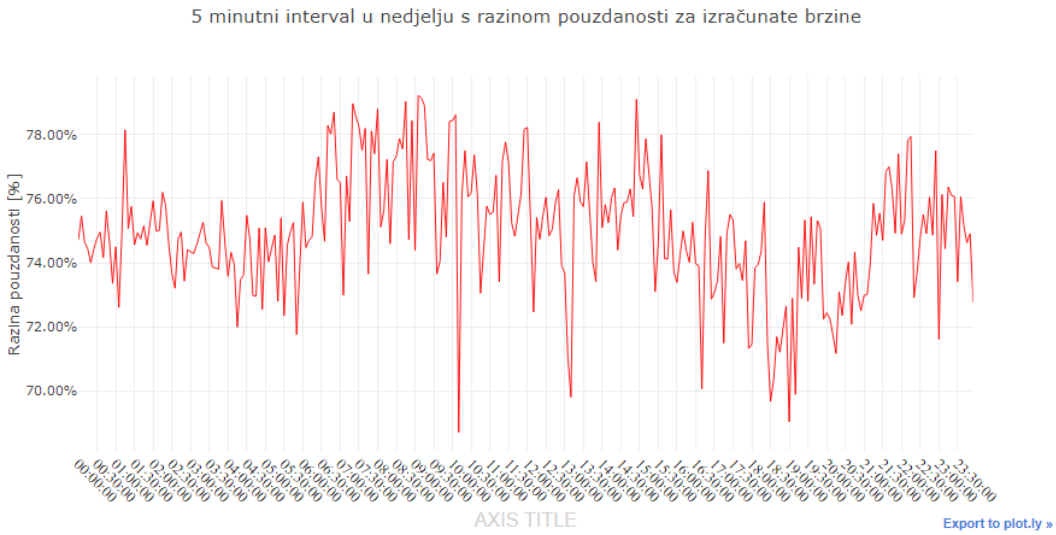


(b) Prikaz vremena putovanja za najkraći link u ruti

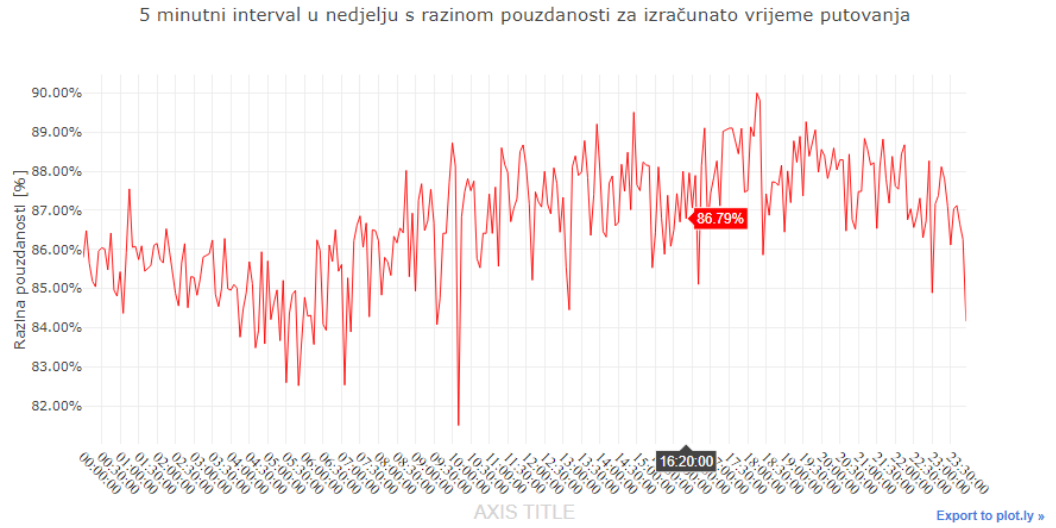
Slika 7.6: Prikaz vremena putovanja za najduži i najkraći link u ruti

Prava vrijednost stvorene programske podrške i biblioteke *plotly* vidi se na primjeru slike 7.6, gdje se čini kako je standardna devijacija veća za najkraći link (7.6b), a manja za najduži link (7.6a). A zapravo prijelazom pokazivača preko svih pet minutnih intervala uviđa se kako je veća vrijednost standardne devijacije za najduži link. Kroz stvorenu programsku podršku, moguće je uspoređivati pouzdanosti za različite dane u tjednu te dodati nove rute i uspoređivati rute. Usporedbom ruta moguće je uočiti zakonitosti i donositi ispravne zaključke. Pouzdanost brzina i vremena putovanja za vikend značajno se razlikuje od radnih dana u tjednu. Kao što je vidljivo iz slike 7.7a, ne postoji vizualna razlika za vršne sate kao na slici 6.2. Za vikend nema vršnih sati, tako da će razina pouzdanosti brzine za svaku rutu biti različita. Također,

razina pouzdanosti za vrijeme putovanja je manja vikendom i manja je razlika između noćnih i dnevnih minutnih intervala, kao što se vidi na slici 7.7b. Dok je razina pouzdanosti vremena putovanja za radne dane veća i konstantnija u dnevnim minutnim intervalima, a u noćnim i jutarnjim minutnim intervalima je razina pouzdanosti manja, kao što je vidljivo iz slike 6.4.



(a) Prikaz razine pouzdanosti brzine rute za nedjelju



(b) Prikaz razine pouzdanosti vremena putovanja rute za nedjelju

Slika 7.7: Prikaz brzine i vremena putovanja rute za nedjelju

Osim pouzdanosti izračunatih brzina za rutu, može se postaviti pitanje pouzdanosti pojedinih linkova s malim brojem podataka. To je nažalost problem, jer neka mjesta i pojedini linkovi nemaju veliku količinu prometa i nije moguće dobiti realne rezultate. Primjer su linkovi koji se nalaze u stambenim područjima grada/mjesta gdje se može dobiti prosječna brzina od 15 km/h, a razlog je vozilo koje je tek krenulo s odredišta ili se vraća. Vozilo traži slobodno mjesto na parkiralištu i vozi niskom brzinom. Iako je ograničenje 50 km/h često će se vozit značajno manjom brzinom, jer je stambeno područje. Tada se postavlja pitanje kako vrednovati dobivene rezultate, jer ponekad može biti stvarna gužva, a nije razlog stambeno područje gdje se nalazi link. Pravo pitanje je kolika je pouzdanost tih rezultata i kako prikazati korisniku prosječnu brzinu u aplikaciji, kada odabire optimalnu rutu. Hoće li biti prikazana izmjerena prosječna brzina od 15 km/h, hoće li se prikazati ograničenje kao prosječna brzina, računajući kako ne postoje gužve ili izračunati srednju vrijednost dobivenih srednjih brzina. To zapravo ostaje kao opcija za daljnji razvoj aplikacije i nadogradnja svega opisanog i napravljenog u diplomskom radu.

8. Zaključak

Obrada, pohrana i analiza velikih količina podataka je jedno od najaktualnijih područja za sektor informacijsko-komunikacijska tehnologija (engl. *Information and Communication Technology*, ICT) te se sve više i više implementira u različite poslovne organizacije. Ovaj diplomski rad pokazao je veliki značaj i potencijal podataka o prometnim entitetima, odnosno njihovim GPS zapisima pomoću kojih se može s velikom pouzdanošću predvidjeti brzina te vrijeme putovanja. Kroz projekt SORDITO trebalo je uložiti vremena i novca kako bi se prikupilo podatke, dok postojeće aplikacije i organizacije koje pružaju usluge lociranja, kao primjerice Google, posjeduju ogromnu količinu potrebnih podataka pomoću koji također mogu predviđati prosječne brzine i predlagati alternative rute za brži dolazak do cilja. S obzirom na to kako napredak tehnologije omogućava pohranu i sve bržu obradu sirovih podataka, jasno je kako organizacije koje prikupljaju najveći broj podataka, mogu donositi najtočnija predviđanja. Govoreći o podacima o brzini entiteta ili o gledanosti televizijskih programa ili broju igrača na dnevnoj razini u video igrici, njih je moguće obraditi, analizirati i grafički prikazati putem statističkih metoda i programskih jezika te donijeti kvalitetne zaključke. Na temelju tih zaključaka moguće je vidjeti postoje li problemi, naznake problema, pad popularnosti te pronaći rješenje koje će preventivno ukloniti moguće poteškoće ili optimizirati trenutno stanje. Svrha je kako se opisana metodologija filtriranja, obrade, pohrane i grafičkog prikaza podataka može iskoristiti za bilo koju vrstu podataka. Samo je potrebno modificirati određene korake obrade i pohrane te se mogu dobiti nove informacije na temelju sirovih podataka.

Cilj diplomskog rada je postignut izračunom razine pouzdanosti i stvaranjem informacijsko-komunikacijskog sustava, odnosno stvaranjem aplikacije pomoću koje korisnici mogu saznati informacije o prosječnim brzinama po pojedinim linkovima i rutama koje dodatno prikazuju prosječno vrijeme putovanja. Cilj je postignut zahvaljujući korištenju rezultata projekta SORDITO koji je omogućio osnovno razumijevanje problematike prometnog sustava te omogućio sirove podatke o prometnim entitetima. Bez sirovih podataka nije moguće obaviti filtriranje, obradu i pohranu podataka te stvarati nove informacije koje će poboljšati razumijevanje prometnog sustava. Sirovi podaci opisuju cestovne segmente grada Zagreba, odnosno 14.154 cestovna segmenta. Nakon što se teorijski objasne metode filtriranja, obrada i pohrana podataka, stvara se programska podrška, odnosno informacijsko-komunikacijski sustav za filtriranje, obradu i

pohranu podataka. Filtriranje podataka je uspješno obavljeno MAD metodom nad skupom podataka koji se sastoji od brzina. Obrađeni podaci, njih 9,35 GB, pohranjuju se na dva različita načina. Kod prvog načina se u relacijsku bazu podataka - PostgreSQL, stvaraju tablice za svaki dan u tjednu te za svaki vremenski interval. Što je prihvatljivo u ovom radu, jer se promatraju dva vremenska intervala (5 min i 15 min) i neće se stvoriti više od 14 tablica. Drugi način se koristi kada postoji više vremenskih intervala, čime se smanjuje broj tablica na tri tablice: linkovi, profili i vrijednosti, koje su međusobno hijerarhijski povezani stranim ključevima. Razlog odabira prvog načina je brzina dohvata podataka iz PostgreSQL-a. Nakon pohrane podataka stvara se GUI s interaktivnom kartom i grafičkim prikazom podataka za pojedini cestovni segment. Nakon toga se definira razina pouzdanosti koja govori s kojom pouzdanošću će biti izračunata prosječna brzina za definiranu rutu. Osim razine pouzdanosti za prosječnu brzinu, izračunat će se razina pouzdanosti za prosječno vrijeme putovanje. Iz rezultata je vidljivo kako je u vršnim satima razina pouzdanosti manja, a razlog tome su periodične gužve koje nastaju u određenim trenucima i smanjuju brzine prometnih entiteta. Studija slučaja detaljno opisuje dobivene rezultate u stvorenom informacijsko-komunikacijskom sustavu. Uspoređuju se podaci za različite linkove, koje se metode mogu koristiti za poboljšavanje dobivenih rezultata. U studiji slučaja se prikazuju primjeri linkova s malim brojem podataka te zbog čega nemaju veću količinu podataka pojedini linkovi. Najbitniji dio studije slučaja je ocjena vrijednosti i pouzdanosti dobivenih rezultata, u kojim područjima i slučajevima su korisni dobiveni rezultati te kako poboljšati dobivene rezultate.

Rezultate dobivene aplikacijom mogu koristiti svi korisnici, međutim, od najveće su koristi organizacijama koje posjeduju flotu vozila i prevoze resurse, proizvode, ljude, itd. Na temelju podataka može se optimizirati ruta kretanja od izvora prema odredištu te tako smanjiti troškove isporuke/dostave. Vidljiva je prosječna brzina s pouzdanošću, kao i vrijeme putovanja te na temelju razine pouzdanosti i vremena putovanja moguće je odabrati najoptimalniju rutu. Međutim, moguća su i poboljšanja same aplikacije, kao i dobivenih rezultata. Za dobivanje preciznije razine pouzdanosti i kvalitetnijih rezultata, potreban je još veći broj podataka po pojedinom linku, upotreba algoritama strojnog učenja i uključivanje dodatnih informacija za pojedini GPS zapis.

LITERATURA

[1] Rožić L, Carić T, Matulin M, Ravlić M, Fosin J, Milošević A, et al. Tehnički izvještaj rezultata eksperimentalnog razvoja projekta SORDITO. Sveučilište u Zagrebu, Fakultet prometnih Znanosti; 2016.

[2] Erdelić T, Vrbančić S, Rožić L. A model of speed profiles for urban road networks using G-means clustering. In: 2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO); 2015. p. 1081–1086.

[3] Erdelić T, Ravlić M, Carić T. Travel time prediction using speed profiles for road network of Croatia. In: 2016 International Symposium ELMAR; 2016. p. 97–100.

[4] Nandi K. Basic statistics: Mode, Mediaj, Mean. Make me analyst;. Preuzeto sa: http://makemeanalyst.com/explore-your-data-mode-median-and-mean/. [Pristupljeno: rujan 2018.].

[5] Jhguch. Wikipedia;. Preuzeto sa: https://commons.wikimedia.org/wiki/File:Boxplot_vs_PDF.svg. [Pristupljeno: rujan 2018.].

[6] What are outliers in the data? NIST/SEMATECH;. Preuzeto sa: https://www.itl.nist.gov/div898/handbook/prc/section1/prc16.htm. [Pristupljeno: rujan 2018.].

[7] Rousseeuw PJ, Croux C. Alternatives to the Median Absolute Deviation. Journal of the American Statistical Association. 1993;88(424):1273–1283. Preuzeto sa: https://www.tandfonline.com/doi/abs/10.1080/01621459.1993.10476408. [Pristupljeno: rujan 2018.].

[8] Lutz M. Learning Python. 2nd ed. Sebastopol, CA, USA: O'Reilly & Associates, Inc.; 2003.

[9] Liberty Center One. PostGRESQL vs MySQL vs SQL Server vs Oracle: Which DBMS Is The Best Choice For You?; 2018. Preuzeto sa: https://www.libertycenterone.com/blog/postgresql-vs-mysql-vs-sql-server-vs-oracle/. [Pristupljeno: rujan 2018.].

[10] Tezer OS. SQLite vs MySQL vs PostgreSQL: A Comparison Of Relational Database Management Systems; 2014. Preuzeto sa: https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql-a-comparison-of-relational-database-management-systems. [Pristupljeno: rujan 2018.].

[11] Quora. What are pros and cons of PostgreSQL and MySQL? With respect to reliability, speed, scalability, and features.; 2018. Preuzeto sa: https://www.quora.com/What-are-pros-and-cons-of-PostgreSQL-and-MySQL-With-respect-to-reliability-speed-scalability-and-features. [Pristupljeno: rujan 2018.].

[12] Chander G. PostgreSQL vs. MS SQL Server. LinkedIn; 2017. Preuzeto sa: https://www.linkedin.com/pulse/postgresql-vs-ms-sql-server-girish-chander/. [Pristupljeno: rujan 2018.].

[13] Smith L. What PostgreSQL has over other open source SQL databases: Part I. Compose Articles; 2015. Preuzeto sa: https://www.compose.com/articles/what-postgresql-has-over-other-open-source-sql-databases/. [Pristupljeno: rujan 2018.].

[14] Gregorio FD, Varrazzo D. Basic module usage;. Preuzeto sa: http://initd.org/psycopg/docs/usage.html#adaptation-of-python-values-to-sql-types. [Pristupljeno: rujan 2018.].

[15] Imran M. How sequence works in SQL Server 2012; 2012. Preuzeto sa: https://raresql.com/2012/04/29/how-sequence-works-in-sql-server-2012/. [Pristupljeno: rujan 2018.].

[16] Imran M. Difference between Identity and Sequence in SQL Server 2012; 2012. Preuzeto sa: https://raresql.com/2012/05/01/difference-between-identity-and-sequence/. [Pristupljeno: rujan 2018.].

[17] Bronshtein A. A Quick Introduction to the "Pandas" Python Library. Towards Data Science; 2017. Preuzeto sa: https://towardsdatascience.com/a-quick-introduction-to-the-pandas-python-library-f1b678f34673. [Pristupljeno: rujan 2018.].

[18] NumPy. NumPy;. Preuzeto sa: http://www.numpy.org/. [Pristupljeno: rujan 2018.].

[19] Matplotlib: Python plotting - Matplotlib 2.2.2 documentation;. Preuzeto sa: https://matplotlib.org/gallery/index.html. [Pristupljeno: rujan 2018.].

[20] Plotly;. Preuzeto sa: https://plot.ly/python/. [Pristupljeno: rujan 2018.].

[21] Gregorio FD, Varrazzo D. Psycopg – PostgreSQL database adapter for Python;. Preuzeto sa: http://initd.org/psycopg/docs/. [Pristupljeno: rujan 2018.].

[22] Krebs B. SQLAlchemy ORM Tutorial for Python Developers. Auth0 - Blog; 2017. Preuzeto sa: https://auth0.com/blog/sqlalchemy-orm-tutorial-for-python-developers/. [Pristupljeno: rujan 2018.].

[23] PyQt. The Python Wiki;. Preuzeto sa: https://wiki.python.org/moin/PyQt. [Pristupljeno: rujan 2018.].

[24] Tenkanen H. Intro Python GIS;. Preuzeto sa: https://automating-gis-processes.github.io/CSC18/course-info/Installing_Anacondas_GIS.html. [Pristupljeno: rujan 2018.].

[25] Python-visualization. Folium. GitHub; 2018. Preuzeto sa: https://github.com/python-visualization/folium. [Pristupljeno: rujan 2018.].

[26] GeoAlchemy 2 Documentation. GeoAlchemy 2;. Preuzeto sa: https://geoalchemy-2.readthedocs.io/en/latest/. [Pristupljeno: rujan 2018.].

[27] Jupyter Documentation. Project Jupyter;. Preuzeto sa: http://jupyter.org/documentation. [Pristupljeno: rujan 2018.].

[28] Nandi K. Variance and standard deviation. Make me analyst;. Preuzeto sa: http://makemeanalyst.com/explore-your-data-variance-and-standard-deviation/. [Pristupljeno: rujan 2018.].

[29] World Geodetic System (WGS84). GIS Geography; 2018. Preuzeto sa: https://gisgeography.com/wgs84-world-geodetic-system/. [Pristupljeno: rujan 2018.].

[30] Well-known text. Wikimedia Foundation; 2018. Preuzeto sa: https://en.wikipedia.org/wiki/Well-known_text. [Pristupljeno: rujan 2018.].

[31] Pereira FC, Costa H, Pereira NM. An off-line map-matching algorithm for incomplete map databases. European Transport Research Review. 2009 Oct;1(3):107–124. Preuzeto sa: https://doi.org/10.1007/s12544-009-0013-6. [Pristupljeno: rujan 2018.].

[32] Zelenika R. Metodologija i tehnologija izrade znanstvenog i stručnog djela. Udžbenici Sveučilišta u Rijeci. Ekonomski fakultet; 2000.

[33] Stake RE. The Art of Case Study Research. SAGE Publications; 1995.

[34] Novelty and Outlier Detection;. Preuzeto sa: http://scikit-learn.org/stable/modules/outlier_detection.html. [Pristupljeno: rujan 2018.].

POPIS KRATICA I AKRONIMA

Kratica	Engleski	Hrvatski
ABC	All Basic Code	sve osnovni kôd
CRS	Coordinate reference systems	koordinatni referentni sustavi
CSS	Cascading Style Sheets	stilski jezik
ERD	Entity Relationship Diagram	dijagram entiteta i veza
GPS	Global Positioning System	Globalni pozicijski sustav
GUI	Graphical User Interface	grafičko korisničko sučelje
HTML	Hypertext Markup Language	prezentacijski jezik
ICT	Information and Communication Technology	informacijsko-komunikacijska tehnologija
IDE	Integrated Development Environment	integrirano razvojno okruženje
IQR	Interquartile range	metoda interkvartila

Kratica	Engleski	Hrvatski
JSON	JavaScript Object Notation	JavaScript objekt notacija
MAD	Medain absolute deviation	apsolutna devijacija medijana
OGR	OpenGIS Simple Features Reference Implementation	OpenGIS implementaciju referentnih jednostavnih značajki
OOP	Object-Oriented Programming	objektno orijentirano programiranje
ORM	Object Relational Mapper	objektno-relacijsko preslikavanje
pip	Pip Installs Packages	upravljač paketima
SORDITO	System for Route Optimization in a Dynamic Transport Environment	Sustav za optimizaciju ruta u dinamičkom transportnom okruženju
SQL	Structured Query Language	strukturirani upitni jezik
SSL	Secure Sockets Layer	protokola za sigurnu komunikaciju
UTC	Coordinated Universal Time	usklađeno svjetsko vrijeme
WGS 84	World Geodetic System	svjetski geodetski sustav
WKT	Well-known text	opisni jezik za reprezentaciju vektorskih geometrijskih objekata

Kratica	Engleski	Hrvatski
XML	Extensible Markup Language	proširivi jezik za označavanje

POPIS SLIKA

2.1. Primjer GPS tragova, [1] 4
2.2. Prikaz izdvojenih cestovnih segmenata, [1] 5
2.3. Prikaz sirovih podataka u tekstualnom obliku s vremenom i brzinama svakog GPS zapisa 5
2.4. Prikaz sirovih podataka u tekstualnom obliku s informacijama za pojedini link . 6

3.1. Distribucija brzina na razmatranom linku 9
3.2. Grafički prikaz metode interkvartila na normalnoj distribuciji podataka i prikaz putem *box plota*, [5] 11
3.3. Grafički prikaz filtriranih podataka metodom interkvartila s normalnim granicama, prema izrazu (3.5) 12
3.4. Grafički prikaz filtriranih podataka metodom interkvartila s proširenim granicama, prema izrazu (3.6) 13
3.5. Grafički prikaz filtriranih podataka pomoću metode apsolutna devijacija medijana 15
3.8. Tablice za drugi način pohrane podataka 21
3.9. Prikaz dijagrama eniteta i vrste veza između njih 22
3.10. Prikaz relacijskog dijagrama putem tablice sa svim imenima stupaca i tipovima stupaca 22

4.1. Prikaz PostgreSQL grafičkog sučelja 30
4.2. Anaconda razvojno okruženje 30
4.3. Primjer GUI-a stvorenoga u aplikaciji Qt dizajner 32
4.4. Primjer Jupyter bilježnice s više ćelija koje objašnjavaju osnovne funkcije . . . 34
4.5. Izgled Spyder IDE-a 34

5.1. Grafički prikaz 15 minutnog intervala sa standardnom devijacijom za ponedjeljak 37
5.2. Primjer izglda tablice u PostgreSQL-u 38
5.3. Prikaz obrađenih u *geopandas dataframeu* 40
5.4. Grafički prikaz svih linkova koji ocrtavaju grad Zagreb 41
5.5. Prikaz tablice sa stupcima iz tablice 2.2 42

5.6. Prikaz karte stvorene u Mapbox studiju 44
5.7. Prikaz karte stvorene u Mapbox studiju s dodanim slojem linkova 44
5.8. Prikaz GUI-a s implementiranom kartom 45
5.9. Prikaz aplikacije za link -218749, odnosno Slavonska avenija 47
5.10. Prikaz aplikacije za link -214695, odnosno Jadranski most 47
5.11. Prikaz rute od 12 linkova 48

6.1. Prikaz srednje vrijednosti kroz doba dana za linkove s rute 50
6.2. Prikaz izračunate razine pouzdanosti brzine za rutu od 12 linkova 51
6.3. Srednje vrijeme putovanja sa standardnom devijacijom 52
6.4. Prikaz izračunate razine pouzdanosti vremena putovanja za rutu od 12 linkova . 53

7.1. Prikaz linka u aplikaciji s malim brojem podataka 55
7.2. Prikaz brzina za link u ulici grada Vukovara 56
7.3. Primjer brzina dva linka na Jadranskom mostu 57
7.4. Prikaz interaktivne karte sa smanjenim zumom 59
7.5. Prikaz stvorene aplikacije 60
7.6. Prikaz vremena putovanja za najduži i najkraći link u ruti 61
7.7. Prikaz brzine i vremena putovanja rute za nedjelju 62

POPIS KÔDOVA

4.1. Aktivacija PostGIS-a 26
4.2. Dohvaćanje svih podataka iz tablice geometrija 27
4.3. Kreiranje tablice u relacijskoj bazi 27
4.4. Promjena slijeda dodjeljivanja identifikacijskog broja 28
4.5. Dohvaćanje stupca s detekcijom malih i velikih slova iz tablice 28
4.6. PostgreSQL stvaranje tablice pomoću upita iz postojeće tablice 28
4.7. Microsoft SQL Server stvaranje tablice kroz upit iz postojeće tablice 29
4.8. Brisanje tablice ako postoji 29
4.9. Kreiranje druge tablice sa stranim ključem 29

5.1. Python kôd za filtriranje podataka pomoću MAD metode 36
5.2. Python kôd u biblioteci *psycopg2* za stvaranje tablice 39
5.3. Python kôd u biblioteci *psycopg2* za umetanje podataka u tablicu 39
5.4. Python kôd u biblioteci *PyQt5* koji omogućava asinkronu komunikaciju sa Javascriptom i dohvaćanje kliknutih koordinati 46