

Universiteti i Prishtinës “Hasan Prishtina”

Tema: Klasifikimi në video-lojën Dota 2

Mentorë: Prof. Dr. Ing. Lule Ahmedi

Studenti: Labinot Vila

01/01/2019

Abstrakti

IntelliDota është një projekt i realizuar me anë të gjuhëve programuese Scala dhe Flutter, gërshetimi i së cilave sjell një aplikacion të mençur, të qëndrueshëm, të shpejtë dhe ndihmues, me anë të së cilit mund të analizojmë burime të të dhënave, vizualizojmë dhe të aplikojmë metrika dhe algoritme të ndryshme mbi to.

Kemi saktësisht dy burime të të dhënave, njëri që është i i nxjerrë me ndihmesën e Steam Public API, format i të cilit është JSON ndërsa tjetri i marrë nga Kaggle, formati i të cilit është CSV – vlera të ndara me presje (ang. *comma seperated values*). Tek këto dy burime të të dhënave janë aplikuar teknika e klasifikimit dhe kllasterimitrespektivisht.

Përveç këtyre algoritmeve të mençura, aplikacioni lejon edhe diskretizimin e të dhënave bashkë me shumë metrika të tjera të cilat mund t’i aplikojmë në dy data setet. Si pjesë kryesore e aplikacionit është klasifikimi apo kllasterimi në kohë reale që realizohet nga modelet e trajnuara të krijuara po ashtu nga ne.

Pjesa e padukshme e aplikacionit, pra pjesa *backend* është kryer me anë të gjuhës programuese *Scala*, konkretisht *Spark* dhe *Play.* *Scala* shquhet për programim funksional dhe të orientuar në objekte, që rezulton në një sintaksë koncize dhe vetë-përshkruese. Si ambient punues ka *Java Virtual Machine*, që do të thotë se ka një bagazh të madh të librarive. Ndër libraritë më të njohura si për procesim të të dhënave, ashtu edhe për të mësuar të mençur është Spark. Krahas Spark, është përdorur edhe *Play*, që paraqet një skeletme anë të së cilit ofrohet arkitektura MVC (model, pamje, kontroller, ang*. model, view, controller*) e ndërtuar mbi *Akka.*

Ndërsa pjesa e dukshme e aplikacionit, prapjesa *frontend* është ndërtuar me anë të Flutter. Flutter është një vegël e ndërtuar nga *Google* për ndërtimin të aplikacioneve autokton (ang. *native*)për telefona të mençur, ekran apo shfletues. Lejon një zhvillim të ndërfaqes(ang. *interface*)në mënyrë të shpejtë, shprehëse dhe elastike. Si ambient punues ka *Dart*.

Ndërlidhja e këtyre dy komponentëve mundësohet nëpërmjet një ndërfaqe programore të aplikacionit e cila përdorë kërkesat HTTP (*GET, PUT, POST* dhe *DELETE*) për të kryer veprimet përkatëse me të dhëna.

Lista e figurave

Lista e tabelave

Tabela e përmbajtjes

[1 Hyrje 7](#_Toc23929301)

[2 Algoritmet kryesore për inteligjencë artificiale 9](#_Toc23929302)

[2.1 Decision Trees 10](#_Toc23929303)

[2.2 Naïve Bayes 11](#_Toc23929304)

[2.3 Random Forest 11](#_Toc23929305)

[2.4 Support Vector Machine 12](#_Toc23929306)

[3 Hyrje në projekt 13](#_Toc23929307)

[4 Koleksionimi i të dhënave për klasifikim 16](#_Toc23929308)

[4.1 Pamja e parë e strukturës 16](#_Toc23929309)

[4.2 Metrika gjenerike për data setin e derivuar 19](#_Toc23929310)

[4.3 Mënyra e *Scalas* 24](#_Toc23929311)

[4.4 Klasifikimi në Scala 28](#_Toc23929312)

[4.5 Largimi i të veçuarve 31](#_Toc23929313)

[5 Play në Scala 32](#_Toc23929314)

[5.1 Struktura 32](#_Toc23929315)

[5.2 Klasifikimi në kohë reale në Scala 33](#_Toc23929316)

[6 Lista e thirrjeve fundore të mundshme 37](#_Toc23929317)

[6.1 index 37](#_Toc23929318)

[6.2 getColumns 37](#_Toc23929319)

[6.3 getSample 37](#_Toc23929320)

[6.4 getStages 38](#_Toc23929321)

[6.5 getCorrelationMatrix 38](#_Toc23929322)

[6.6 getGroupAndCount 38](#_Toc23929323)

[6.7 getStages 39](#_Toc23929324)

[6.8 getSchema 39](#_Toc23929325)

[6.9 getDoubleGroup 39](#_Toc23929326)

[6.10 postPredict 40](#_Toc23929327)

[7 Fjalët e fundit 41](#_Toc23929328)

[8 References 42](#_Toc23929329)

# Hyrje

Ashtu siç kureshtja e njeriut nuk shuhet kurrë, ashtu edhe përparimi i teknologjisë nuk ka të ndalur. Megjithëse nuk është e gabuar nëse themi se teknologjia gati ka arritur majat e veta, akoma ka shtigje të pashkelura që mund të themi se paraqesin një botë teknologjike në vete. Ndër to, dhe ndër më të pëlqyerat e kërkuarat nga njerëzit është inteligjenca artificiale (ang. *artificial intelligence*).

## Motivimi

Inteligjenca artificiale, e referuar ndryshe si inteligjenca e makinave (ang. *machine intelligence*)ka të bëjë me stimulimin e inteligjencës të përvetësuar dhe zhvilluar nga makinat [1].

Kjo shkencë ndryshe definohet si fusha e studimit të agjentëve inteligjentë(ang. *intelligent agents*)*:* një pajisje që kupton ambientin ku ndodhet dhe ndërmerr veprime të tilla që mundësia e arritjes së qëllimit të jetë maksimale.

Për një agjent themi se është racional nëse bën gjënë e duhur, pra secili veprim në rrethana të caktuara është i saktë. Por si mund të definojmë se çfarë është e saktë dhe çfarë jo? Nëse agjenti kalon nëpër një sekuencë veprimesh dhe gjendjesh që na kënaqin themi se agjenti ka pasur një performancë të mirë [2].

Pra, makinat tentojnë që të përvetësojnë aftësitë njerëzore të të kuptuaritdhe zgjidhjes së problemeve(ang. *learning and problem solving*)*.*

Qëllimi i një agjenti inteligjentë mund të jetë i thjeshtë, si për shembull luajtja e një loje GO, apo kompleks siç është kryerja e operacioneve matematikore.

Parimi bazë i inteligjencës artificiale është përdorimi i algoritmeve. Algoritmet janë një grumbull instruksionesh që një makinë kompjuterike mund të ekzekutojë. Një algoritëm kompleks ndërtohet si bashkësi e algoritmeve të thjeshta (**Error! Reference source not found.**).



Disa algoritme, janë të aftë të mësojnë nga një grumbull të dhënash, si në rastin tonë, ku algoritmi mëson një strategji apo një mënyre të mirë(ang. *rule of thumb*) të cilin e zbaton në të dhëna të reja, e disa algoritme të tjerë mund të vetë shkruajnë algoritme tjera.

Disa nga algoritmet që mësojnë, siç janë *fqinji më i afërt* (ang. *nearest-neighboor*)*, pemët me vendime* (ang. *Decision Trees*)*,* apo *rrjeti Bajesian* (ang. *Bayesian network*)munden teorikisht të përafrojnë çfarëdo të dhënash në një funksion të caktuar (nëse kanë memorie dhe kohë të pafundme). Që në kohët e hershme është stimuluar inteligjenca artificiale, e ndër qasjet më të njohura janë:

* Simbolizmi, njohur ndryshe si logjika formale: nëse personi ka të ftohtë, atëherë ai ka grip.
* InterferenceBajesiane(ang. *Bayesian interference*)*:* nëse personi ka të ftohtë, atëherë ekziston një probabilitet që ai të ketë edhe grip.
* *Vektorit Mbështetës i Makinës* (ang. *Support Vector Machine*)apo *Nearest-neighboor:* pas shqyrtimit të të dhënave të personave që kanë të ftohtë, duke përfshirë moshën, simptomat dhe faktorët tjerë, dhe këto faktorë përkasin me pacientin aktual, themi se pacienti ka grip.

Për një makinë thuhet se është inteligjente nëse kalon testin e Turingut, test ky i cili u dizajnua në kohët e hershme që të sjell një përkufizim të kënaqshëm për inteligjencën. Ky test kalohet nëse është e pamundur të tregohet se a vijnë përgjigjet nga një njeri apo nga një makinë pas disa pyetjeve nga njeriu [3].

## Përshkrimi i problemit

Dihet se sot po thuajse në çdo fushë të jetës gjejmë aplikim të inteligjencës artificiale e sidomos në fushën e kompjuterikes, duke filluar nga sistemet rekomanduese, sistemet për predikim e shumë të tjera.

Duke pasur parasysh këto aplikime, ne kemi tentuar që të ndërtojmë një asistent për video-lojën e njohur Dota 2 pasi jo vetëm që do t’u ndihmonte njerëzve përgjatë lojës, por edhe do të pasuronte komunitetin me një data set unik (pasi po e krijojmë vet një të tillë nëpërmjet thirrjeve përkatëse) dhe kod burimorë që paraqet ecurinë e projektit dhe një shembull për të realizuar diçka të ngjashme si nga ana e inteligjencës artificiale, edhe nga ana e realizimit të thirrjeve dhe filtrimit të strukturës JSON.

Duhet cekur se ekzistojnë shumë data sete të lidhura me këtë temë në internet, por për arsye të teknikave dhe metodologjive që dëshirojmë të implementojmë, kemi zgjedhur që ta krijojmë vet një të tillë.

Problemi jonë mund të përshkruhet nëpër disa faza, ku më të rëndësishmet janë:

* Krijimi i data setit – duke kryer këtë hap me sukses, neve na jepet mundësia të aplikojmë metrika, krahasime e edhe algoritme për inteligjencë artificiale në burimin e të dhënave përkatës. Si çështje kryesore paraqitet mënyra e realizimit të kërkesave dhe filtrimi i tyre, në mënyrë që të marrim një data set të shëndoshë.
* Aplikimi i algoritmit – na krijohet një model i trajnuar, qoftë ai lokal apo jo me anë të së cilit mund të aplikojmë klasifikimin në kohë reale. Lokalisht, kjo kryhet lehtë pasi nuk ka nevojë për masa tjera për të ruajtur modelin, për derisa nëse dëshirojmë të publikojmë në një server nikoqir (ang. *host*) duhet ruajtur modelin si një imazh.
* Zgjedhja e gjuhës programuese – për të realizuar kërkesat e më sipërme na nevojitet një gjuhë programuese që ofron fuqi dhe metoda të shumta në procesim dhe para procesim të të dhënave, e edhe opsione për inteligjencë artificiale.
* Krijimi i pikave fundore – u jep thirrjeve që realizohen nga jashtë një adresë të caktuar se ku duhet referuar për të marrë rezultatet e dëshiruara.

Pra, për aplikacionin tonë themi se ka kryer punën me sukses nëse është në gjendje të klasifikoj dhe kllasteroj të dhënat që i japim si hyrje në kohë reale, në mënyrë të shpejtë dhe të organizuar.

Pjesa inteligjente e aplikacionit mund të kryhet në shumë mënyra, duke filluar nga gjuha programuese Python me libraritë përkatëse që mundësojnë krijimin e një aplikacioni të ngjashëm me atë se çfarë duam të bëjmë ne e deri te shkrimi i një algoritmi nga ana jonë. Por meqë nuk është parë e arsyeshme të ‘ri zbulojmë rrotën’ ne kemi vendosur të përdorim gjuhën programuese Scala bashkë me Spark që ofron një arsenal të fuqishëm për manovrim me të dhëna dhe inteligjencë artificiale. Një përshkrim më në detaje se përse kemi zgjedhur këtë gjuhë me libraritë përkatëse do të shpjegohet më vonë.

E sa i përket pjesës së krijimit të të dhënave, ekzistojnë mënyra të tjera (siç janë përdorimi i një gjuhe programuese për të krijuar thirrjet) por jo metodologji, që do të thotë se data seti që kemi krijuar ne mund të merret vetëm nëpërmjet ueb-faqes zyrtare Steam nëpërmjet thirrjeve dhe filtrimeve përkatëse.

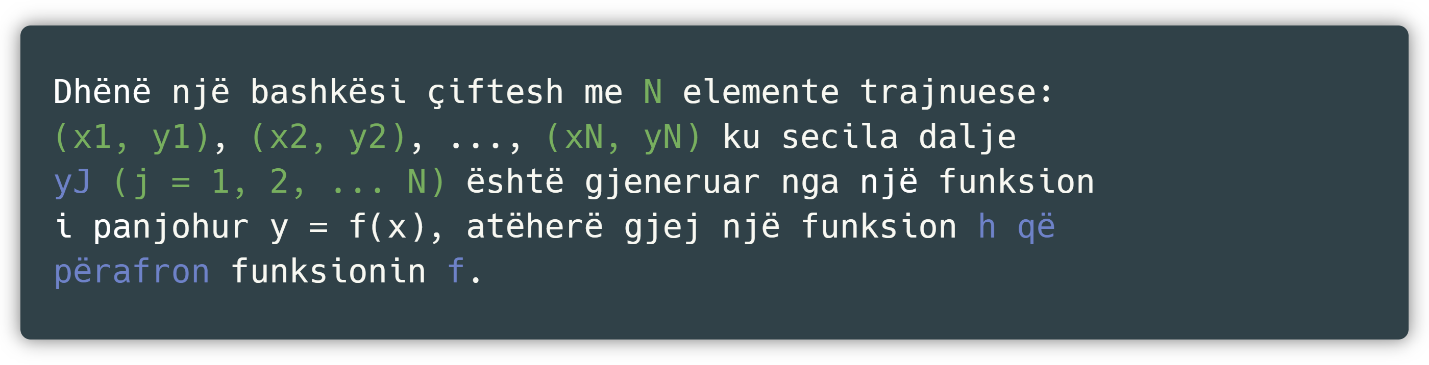
# Inteligjenca artificiale – Klasifikimi si mësim me mbikëqyrje

Në përgjithësi ekzistojnë forma të ndryshme të të mësuarit të makinës, por tri më konkretisht përfaqësojnë pothuajse çdo formë të të mësuarit. Kemi:

* Mësimin pa mbikëqyrje (ang. *unsupervised learning*) – agjenti mëson mënyra për të zgjidhur apo kalkuluar problemin e caktuar varësisht hyrjes edhe pse nuk ka reagime dalëse nga burime të caktuara (pra, algoritmi nuk shpërblehet e as nuk dënohet për veprimin e zgjedhur). Teknika më e zakonshme dhe e përhapur është kllasterimi, me anë të së cilës bëhet grupimi potencialisht i saktë nga të dhënat hyrëse. Për shembull, një taksi gradualisht zhvillon konceptin e një ‘dite të mirë trafiku’ apo ‘një dite jo të mirë trafiku’ pa i dhënë asnjëherë shembuj të mëparshëm nga një ‘mësues’.
* Mësimi i sforcuar (ang. *reinforcment learning*) – agjenti mëson nga një seri e veprimeve dalëse, qofshin ato veprime dënuese apo shpërblyese. Për shembull, një bakshish në fund të vozitjes me taksi i tregon algoritmit se ka vepruar mirë, përndryshe ka vepruar keq. I takon algoritmit të vendos se cilat pjesë të tij kanë bërë që të kryejë veprimin mirë.
* Mësimi me mbikëqyrje (ang. *supervised learning*) – agjenti mëson nga çiftet hyrëse – dalëse, dhe mëson një funksion me anë të së cilit krijon një skemë ku orientohen vlerat e ardhshme hyrëse në ato dalëse.

Në këtë aplikacion do të përdoret saktësisht të mësuarit me mbikëqyrje, teknikë kjo që do të përshkruhet më poshtë saktësisht me algoritmin përkatës!

Qëllimi i mësimit me mbikëqyrje është përshkruar si më poshtë ku x dhe y mund të jenë vlera të çfarëdoshme dhe funksioni h është hipoteza.



Për të masur saktësinë e hipotezës, ne japim një bashkësi testuese të dhënash të ndryshme nga të dhënat trajnuese. Për hipotezën themi se ka gjeneralizuar mirë nëse me saktësi ka predikuar vlerën *y* për llojin e ri të të dhënave [4].

Kur dalja *y* është një interval i vlerave të fundme, themi se problemi i të mësuarit është klasifikimi dhe quhet binarë apo Bulean nëse intervali ka vetëm dy vlera. Ndërsa kur *y* është një numër i vazhdueshëm themi se problemi i të mësuarit është regresioni. Teknikisht në regresion tentojmë të gjejmë një vlerë të afërt, sepse probabiliteti për t’ia qëlluar saktë vlerës është 0.

## Algoritmet e mësimit me mbikëqyrje

Ekzistojnë shumë algoritme të të mësuarit me mbikëqyrje, megjithatë, edhe pse punojnë në parimin e njëjtë të gjithë kanë përparësitë, të metat dhe rastet ku një algoritëm është më i përshtatshëm se tjetri. Nuk ka ndonjë algoritëm që punon më së miri në të gjitha problemet e mundshme. Ndër parametrat kryesorë që bëjnë që një algoritëm të ketë përparësi ndaj një algoritmi tjetër janë:

* Ndryshueshmëria në të dhëna – nëse të dhënat hyrëse kanë llojllojshmëri të lartë (të dhëna diskrete, të vazhdueshme, të vazhdueshme rritëse).
* Përsëritja në të dhëna – nëse të dhënat hyrëse kanë përsëritje të mëdha, kjo rezulton në një matrice korrelacionesh të lartë që do të thotë se disa algoritme si Regresioni Linearë apo Regresioni Logjistik do të performojnë dobët.
* Prezenca e pa varësisë dhe jo linearitetit – nëse secili atribut i të dhënave hyrëse ka një ndikim të pa varur në rezultat, atëherë algoritmet lineare performojnë në përgjithësi më mirë se sa të tjerët.

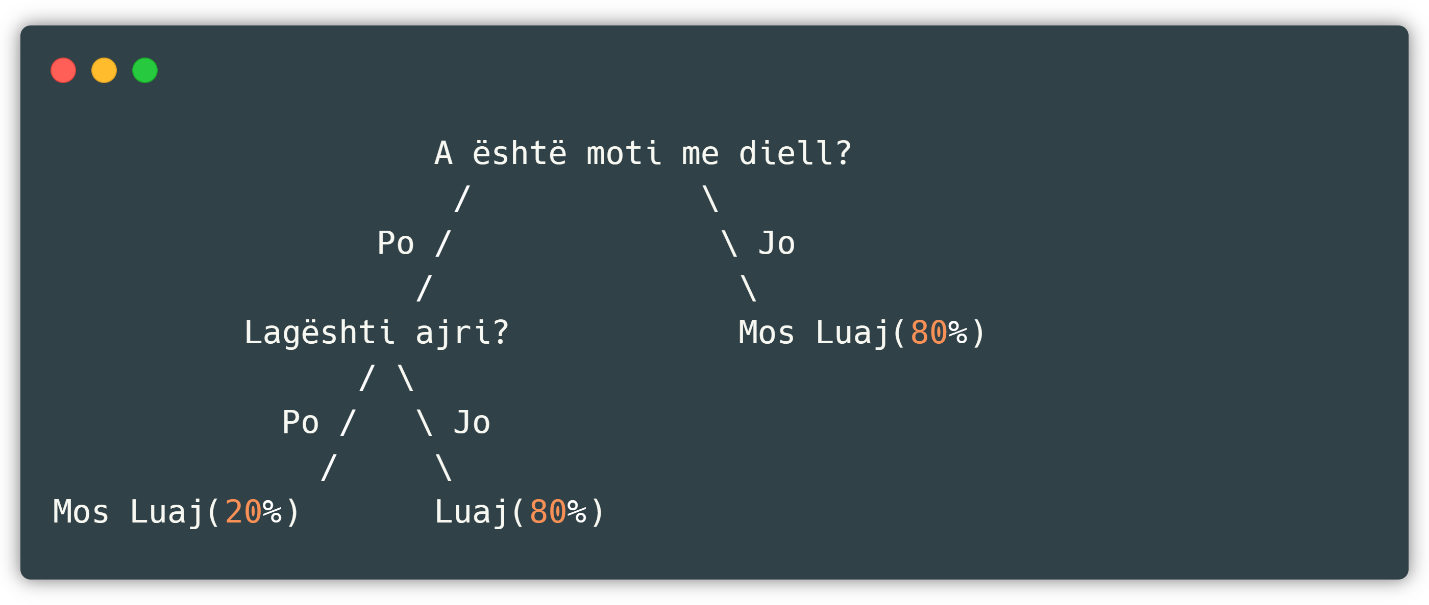
Më poshtë do të listojmë disa nga algoritmet më të përhapur e më të përshtatshëm pavarësisht data setit.

### Pemët vendim-marrëse

Pemët vendim-marrëse (ang. *Decision Trees*) përdoren për të klasifikuar një instancë duke e kaluar nga maja e pemës e deri tek një nyje gjethe, që jep rezultatin e instancës së klasifikuar. Qëllimi është të krijohet një model që predikon vlerën e një variable varësisht vlerave hyrëse të tjera.

Pra, ndërtohet një pemë duke ndarë bashkësinë e atributeve hyrëse në grupe të caktuara, ky proces, i quajtur ndarje rekursive(ang. *recursive splitting*)*,* përsëritet në mënyrë rekursive në secilën nën bashkësi të përfituar. Ky proces mbaron atëherë kur atributi i përfituar është i vetëm, që do të thotë se ndarjet e mëtutjeshme nuk ndikojnë në vlerën përfundimtare [5].

Pavarësisht të metave dhe përparësive, bazuar në përdorimin dhe dokumentimin masiv që ka ky algoritëm, lehtë themi se është ndër algoritmet më të fuqishme që po përdoren aktualisht.



Përparësitë dhe të metat e qasjes *Decision Trees*

+ Janë të afta të gjenerojnë rregulla të kuptueshme.

+ Janë të afta të kryejnë klasifikimin pa llogaritje të mëdha.

+ Kryejnë edhe klasifikimin edhe regresionin.

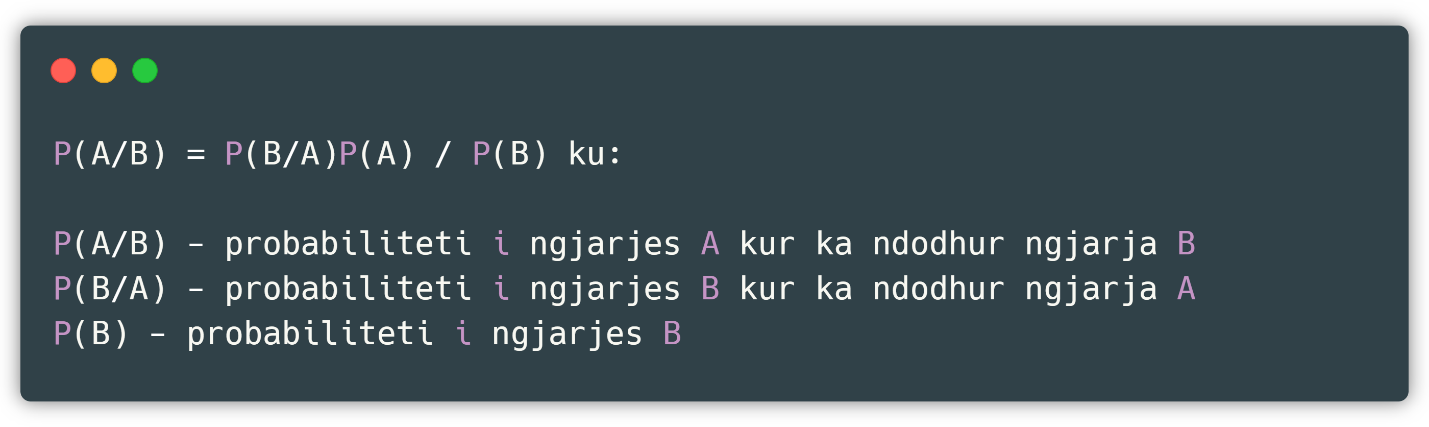
- Janë të dobëta kur duhet predikuar vlerën e një atributi të vazhdueshëm.

- Nëse kemi pak të dhëna trajnuese, atëherë mazha e gabimit është e madhe.

- Për të trajnuar të dhëna, duhet fuqi e madhe kompjuterike.

### Naïve Bayes

Hyn në familjen e klasifikuesve probabilistik(ang. *probabilistic classification*)ku si bazë përdoret teorema e Bajesitnën supozimin se asnjë atribut nuk është i varur nga tjetri, nga edhe e ka marrë emrin *naïve* [6].

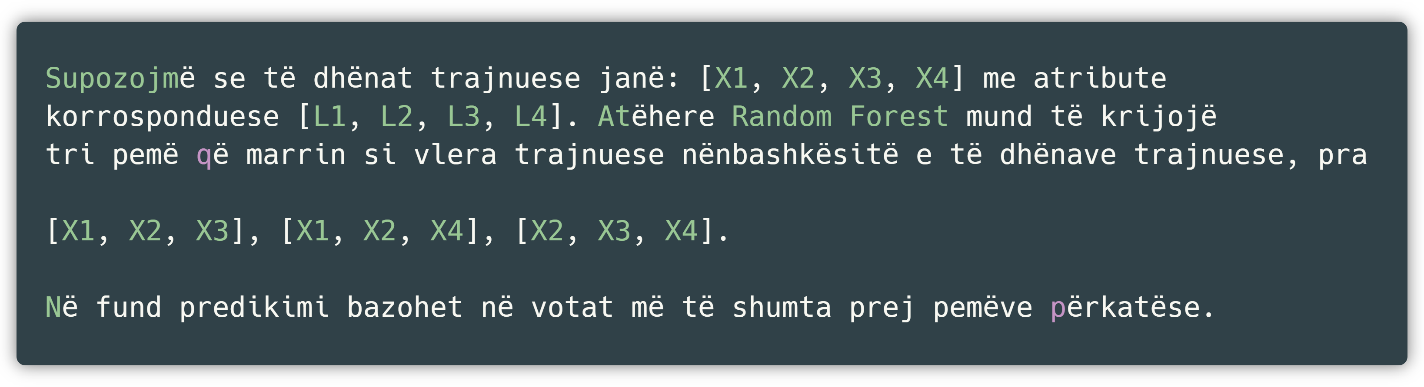


Ndërsa më së miri shpjegohet me një shembull, nëse kanceri varet nga mosha, atëherë me teoremën Bajesiane mund të përdoret ky informacion që ti jap peshë vartësisë së kancerit nga mosha në predikimin se a ka personi përkatës kancer apo jo.

Kjo metodë është e aplikueshme vetëm në të dhëna diskrete.

### Random Forest

Është një algoritëm ansambël (ang. *ensemble algorithm*), që do të thotë se kombinon një apo më shumë algoritme të të njëjtit lloj apo të ndryshëm për të klasifikuar objektin, pra është njëlloj sikur algoritmi të kalonte nëpër *SVM, Naïve Bayes* apo *Decision Tree* e në fund të votohej se cili algoritëm të merret si bazë. Ky lloj algoritmi krijon një bashkësi pemësh në mënyrë të rastësishme të zgjedhur nga të dhënat trajnuese, grumbullon votat nga bashkësi pemësh të ndryshme të votuara si vendim-marrëse dhe caktohet klasa finale e objektit [7].



Algoritmet e tilla, pra ansambël algoritmet, janë shumë më të sakta se sa llojet e algoritmeve tjera për arsye se grupet e pemëve mbrojnë njëra tjetrën nga gabimet individuale, përveç rastit kur të gjitha grupet e pemëve dështojnë në të njëjtin mënyrë. Për shembull, kur disa grupe pemësh janë gabim, shumica tjetër janë saktë, andaj në këtë mënyrë pemët janë të afta të lëvizin të gjitha në një drejtim të saktë së bashku.

### Support Vector Machine

*Support Vector Machines* në të mësuarit e makinës janë modele të mbikëqyrura me anë të së cilëve analizohen të dhënat të përdorura për klasifikim dhe regresion. Me një bashkësi të të dhënave trajnuese, ku secila i përket një kategorie, *SVM* mundet të ndërtoj një model që shembujve të ardhshëm t’ua shoqëroj kategorive përkatëse, duke e bërë këtë zgjedhje në mënyrë jo-probabilistike.

Konkretisht, një *SVM* është një reprezentim i pikave në hapësirë, të organizuara ashtu që secila pikë në hapësirë është e ndarë me një hapësirë që tentohet të jetë sa më e madhe e mundshme.

Më pas, shembujt e ri të predikuar ndjekin po këtë mënyrë dhe kategoria të cilën i përkasin varet nga hapësira në të cilën bien.

Si shembull kemi pjesën më poshtë ku pikat me ngjyrë të gjelbër reprezentojnë grupin e parë G1 ndërsa pikat me ngjyrë të verdhë reprezentojnë grupin e dytë G2. Çfarëdo e dhëne e re që, siç u tha edhe në definicionin më lartë i takon njërit ndaj këtyre grupeve, i takon grupit, atributet e së cilit përafrojnë më shumë.



### Algoritmi i zgjedhur për aplikacion

Duke pasur parasysh shumicën e algoritmeve, mënyrën e funksionimit si dhe rastet konkrete për aplikim, ne kemi zgjedhur algoritmin Random Forest Classification. Arsyet se përse kemi zgjedhur këto algoritme janë dhënë më poshtë:

* Me caktimin e thellësisë së pemës, ne nuk jemi pronë e mbi vendosjes (ang. *overfitting*) për arsye se modeli nuk trajnohet aq shumë sa që të jetë shumë dinamik, e edhe nuk është pak dinamik sa të ketë një saktësi të ulët (nëse data seti ka një numër të konsiderueshëm të rreshtave).
* Data seti jonë përbëhet nga vlera numerike (siç do të shihet më poshtë) dhe algoritmi në fjalë është i saktë në predikim pasi gjykohet nga një numër pemësh vendim-marrëse e jo vetëm një soj.
* Nga të gjitha provat e algoritmeve tjera në data setin tonë, ky algoritëm ka rezultuar më i sakti dukshëm (me rreth 5% diferencë).

Pra në vazhdim nënkuptohet që do të përdoret ky algoritëm.

# Hyrje në projekt

Me të përshkruar inteligjencën artificiale dhe algoritmet kryesore te klasifikimit si përfaqësues te te mësuarit me mbikëqyrje te makinës, tani mund të kalohet në përshkrimin e zgjidhjes së ofruar të problemit, si dhe të rezultateve te fituara.

Po fillojmë njëherë me përshkrimin e veglave softuerike që janë përdorur, në mënyrë që të kuptohet më vonë përdorimi i tyre varësisht rastit.

* *IntelliJ –* vegla për ndërtim(ang. *Integrated Development Kit*,IDE)në të cilin është shkruar aplikacioni, është ndërtuar(ang. *build*)dheprovuar(ang. *test*). Kjo vegël është e shkruar në gjuhën programuese Java dhe është zgjedhur ndër shumë tjera për arsye të organizimit të lartë, mundësive të shumta që ofron si dhe paketave të gatshme si për Scala ashtu edhe për Play.
* *Git –* platforma në të cilën është koordinuar ecuria e punës. Pasi kjo platformë dominon në tregun e punës, është parë e arsyeshme që të praktikohet edhe më shumë kjo mënyrë e punës. Funksionon nëpërmjet ndërfaqes komanduese (ang. *Command Line Interface, CLI*).
* *Github* – platforma në të cilën *Git* vendos dosjet burimore. Kjo platformë në përgjithësi mbledh të gjitha projektet publike të shkruara nga çdo programues në botë dhe ia ofron kërkuesit në mënyrë që ato të kenë qasje edhe në kode burimore edhe në aplikacione që ndërtohen si ndihmesa, siç është edhe vegla e përdorur më poshtë *Ngrok.*
* *Postman –* vegla nëpërmjet së cilës janë kryer HTTPkërkesat(ang. *requests*)si *GET* dhe *POST* me atributet përkatëse. Falë kësaj vegle, ne kemi mundur të stimulojmë kërkesat në mënyrë lokale.
* *Ngrok* – vegla nëpërmjet së cilës janë publikuar qasjet e mundshme të pikave fundore. Kjo është nevojitur për arsye se nuk ka pasur nevojë që të kopjohet projekti në çdo ambient punues por shërbimet në të cilat projekti po ekzekutohet lokalisht ekspozohen në një URI publike nëpërmjet së cilës është realizuar pjesa e dytë e projektit. Për shembull, pika fundore, qasja e së cilës realizohet nëpërmjet lidhjes localhost:9000/getColumns?kind=steam ekspozohet në lidhjen [http://338ab558.ngrok.io/getColumns?kind=steam](http://338ab558.ngrok.io/getColumns?kind=steam%20)dhe kjo lidhje mund të thërritet nga çfarë do burimi.
* *Sublime Text –* vegla me anë të së cilës janë shqyrtuar të dhënat me përmbajtje të madhe të strukturës JSON, nga edhe është bërë përzgjedhja e kolonave për klasifikim. Po ashtu, vegël e dobishme për kontrollin e vlefshmërisë së strukturës JSON (nëpërmjet tasteve CTRL + ALT + J).
* Excel – po ashtu vegël për të shikuar të dhënat e formatit *CSV,* në të cilin mund të aplikohen filtrime dhe kontrolle të ndryshme.
* Powershell – vegla me anë të së cilës është menaxhuar imazhi Docker. Ndryshe nga CLI (Command Line Interface) të tjera, kjo vegël është më funksionale, më e pasur me funksione si dhe ka një ndërfaqe më të përdorshme.
* Docker – vegla me anë të së cilës është krijuar imazhi në mënyrë që ti publikojmë shërbimet e tona në internet të qasshme për këdo.

IntelliDota është një aplikacion i ndërtuar në Scala dhe Flutter që tenton të *kllasteroj* (ang. *cluster)* dhe *klasifikoj* (ang. *cluster)* rezultatin e një video-loje që në rastin tonë është video-loja *Dota 2*. Do të emërtojmë me IntelliDota Classification pjesën e aplikacionit që kryen klasifikimin ndërsa IntelliDota Clustering pjesën e aplikacionit që kryen kllasterimin. Përpara se të vazhdojmë më tutje me aplikacionin, do të bëjmë një përshkrim të shkurtër se çfarë është kjo video-lojë dhe çfarë duam të predikojmë ne.

## Loja kompjuterike Dota 2

*Dota 2* qëndron për Mbrojtjen e Kullave (ang. *Defense of the Ancients)*, pra dy ekipe prej pesë lojtarësh tentojnë të shkatërrojnë bazën kryesore të armikut. Një ekip, që përbëhet prej pesë lojtarëve, posedon një ndarje të një niveli më të lartë, pra mbajtësit (ang. *carries)* dhe mbështetësit (ang. *supports).* Në esencë, çdo *support* mbështet një *carry* (Figura 1: Roli i lojtarëve të një ekipi)*.*



Ndërsa në këndvështrim të lartë, ekipi organizohet në formacion të tillë (Figura 2: Organizimi i ekipit):



Secili lojtarë, pavarësisht rolit ka qindra statistika, por ndër më të thepisurat dhe më të rëndësishmet janë *leaver\_status, gold\_per\_min, leaver\_status, xp\_per\_min, deaths, tower\_damage* etj. Lista e plotë mund të gjendet në kapitullin e mëposhtëm në të cilin realizohet predikimi postPredict, ku përshkruhet saktë funksioni dhe jepet përshkrimi se a është atributi i derivuar apo jo.

Pra, nëse mblidhen këto të dhëna, analizohen dhe aplikohen metrika në to, ne mund të fitojmë rezultate të ndryshme qoftë rreth lojës, apo edhe lojtarëve që luajnë këtë lojë.

Me algoritme të ndryshme, ne mund të shohim qartë se çfarë po luhet më shumë, çka po pëlqehet e si këto atribute bashkë me shumë të tjera ndikojnë në rezultatin e lojës. Ndër tërë këto aspekte, ne kemi vendos të predikojmë atributin fitorja e radiantëve(ang. *radiant\_win)*, që është asgjë më shumë se një nga këto kolona të mësipërme që identifikojnë një lojë të posaçme, me dallim e vetëm se kjo kolonë (si dhe disa të tjera) nuk i përket një lojtari konkret, por qëndron si një atribut që identifikon lojën në tërësi.

Nga këto të thëna, ne mund të vërejmë se në këto lojëra që duam të analizojmë, kemi dy lloje atributesh:

* Atributet që i përkasin lojtarëve, si shembull atributet *gold\_per\_min*, *xp\_per\_min* etj. apo
* Atributet që i përkasin lojës, që janë si shembull kohëzgjatja e lojës, atributi i së cilës emërohet *duration*, atributi *radiant\_win* që aktualisht dëshirojmë të predikojmë, atributi *teamfights* e shumë të tjerë.

Duke bërë kombinimin e këtyre dy atributeve, ne mund të krijojmë një burim të dhënash me të dhëna të shëndosha në të cilin mund të aplikojmë algoritme inteligjente. Një të tillë ne e kemi realizuar bashkë me ndihmesën nga Ndërfaqja Publike e Aplikacionit Programorë Steam(ang. *Steam Public Application Programming Interface, i njohur si Steam Public API)*.

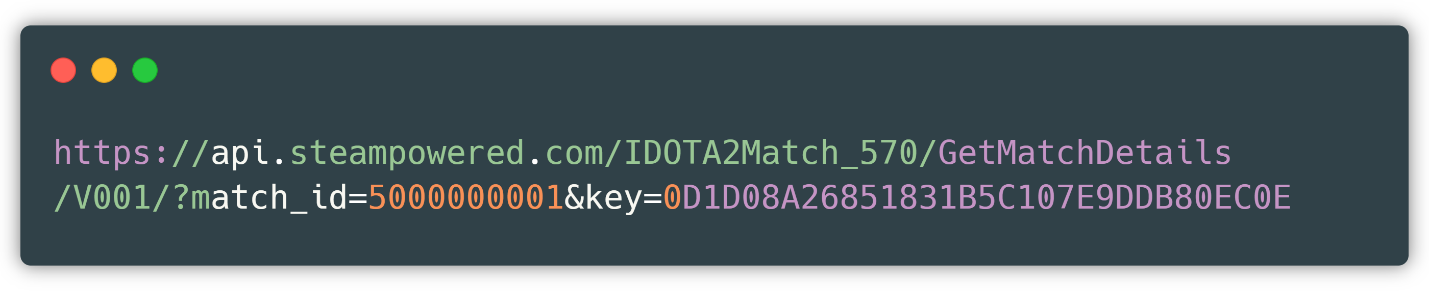
*Valve* (kompania që ka zhvilluar këtë ueb-faqe) ofron këto *API* në mënyrë që zhvilluesit të mund të përdorin të dhënat në mënyra interesante e të dobishme. Pra, zhvilluesit lejohen që të ekzekutojnë kërkesa e të realizojnë pyetësorë të ndryshëm. Për momentin lista aktuale është e kufizuar në vetëm disa video lojëra, por kjo listë pritet të zgjerohet shpejtë [8].

# Koleksionimi i të dhënave për klasifikim dhe para procesimi

## Pamja e parë e strukturës Json nga e cila duam të marrim të dhënat

Si lidhje e burimit të të dhënave për klasifikim është përdorur, siç u tha edhe më lartë, ueb-faqja zyrtare e Valve, pra Steam nëpërmjet Steam Public API *(https://steamapi.xpaw.me)* [9]. Përpara se të qasemi në këtë faqe, duhet pasur një llogari në Steam *(https://steampowered.com)* në mënyrë që të fitojmë një çelës (ang. *Steam Key) dhe* një *domain* emër(ang. *domain name)* nga i cili na lejohet që të marrim të dhëna nga *API* publiki ofruar nga to [10]. Një dyshe e tyre duket kështu, dhe është vendosur në mënyrë që përdoruesi që përdor të ketë një limit të caktuar në marrjen (ang. *fetching)* e të dhënave (e zgjeruar më poshtë, Figura 3: Çifti çelës - domain name).

Me marrjen e këtyre kredencialeve ne mund të bëjmë një kërkesë të thjeshtë nëpërmjet lidhjes më poshtë (Figura 4: Lidhja për një lojë të caktuar).

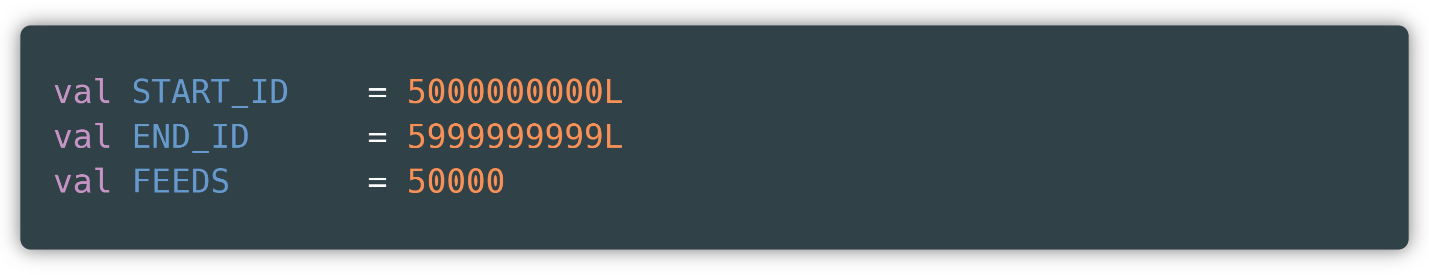


ku si *match­ id­* jepet vlera numerike e një loje me *id* përkatëse, që në rastin e mësipërm është 5000000001, dhe *key* është çelësi i gjeneruar më lartë.

Formati i të dhënave i reprezentuar nga lidhja e mësipërme është JSON *(Javascript Object Notation)* dhe ky na lejon ruajtjen dhe transportimin e të dhënave [11]. Një format i vlefshëm i një loje gjendet nga numri 1 deri te numri 6 miliardë, pra, një *match id* do të ishte e vlefshme nga vlera *match id 1* deri tek *match id 6000000000*.

Dihet që për të grumbulluar kaq shumë të dhëna, duhen vite e vite duke marrë parasysh që një video-lojë zhvillohet për rreth 45 minutash. E për një periudhë kohore kaq të gjatë, edhe struktura e të dhënave, që në rastin tonë është JSON ndërron dukshëm, pasi gjithmonë është tentuar të krijohet një strukturë që është më përshkruese dhe më e kuptueshme se ajo paraprakja. Çfarë duam të themi është se struktura JSON me *match id 1* nuk është e njëjtë me strukturën JSON me *match id 5* miliard*.* Për këtë arsye, ne kemi zgjedhur një interval në të cilin struktura JSON nuk ndryshon dhe posedon aq shumë të dhëna sa që ne të mund të krijojmë një data set të mirëfilltë.

Intervali që kemi zgjedhur ne është (Figura 5: Të dhëna rreth data setit):

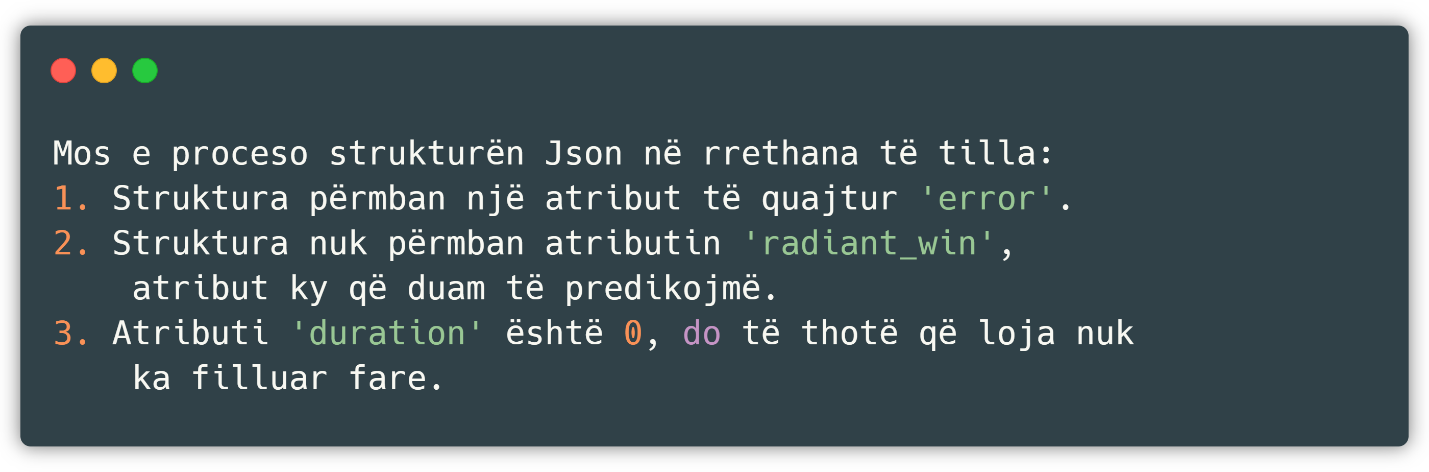


Ku *start id* paraqet fillimin e intervalit që duam të marrim të dhënat dhe *end id* paraqet mbarimin e intervalit, gjatë këtij intervali, për shkak të automatizimit kemi vendosur që vlera e numrit të të dhënavetë jetë dinamike, pra në rastin tonë *feeds*, që është 50.000.

Varësisht numrit *FEEDS*, aq të dhëna do të ketë data seti jonë. Do i referohemi me fjalën data setburimit tonë të të dhënave të cilin jemi duke e vetë-krijuar.

## Para procesimi i të dhënave të mbledhura

Gjatë marrjes së të dhënave, kemi hasur në *game id* që nuk janë të përshtatshme apo janë të korruptuara në kuptimin që dëmtojnë algoritmin predikues apo nuk tregojnë të dhëna reale. Atribute të tilla ka vendosur vetë Steamkur ndodhin rrethana të caktuara në lojë.

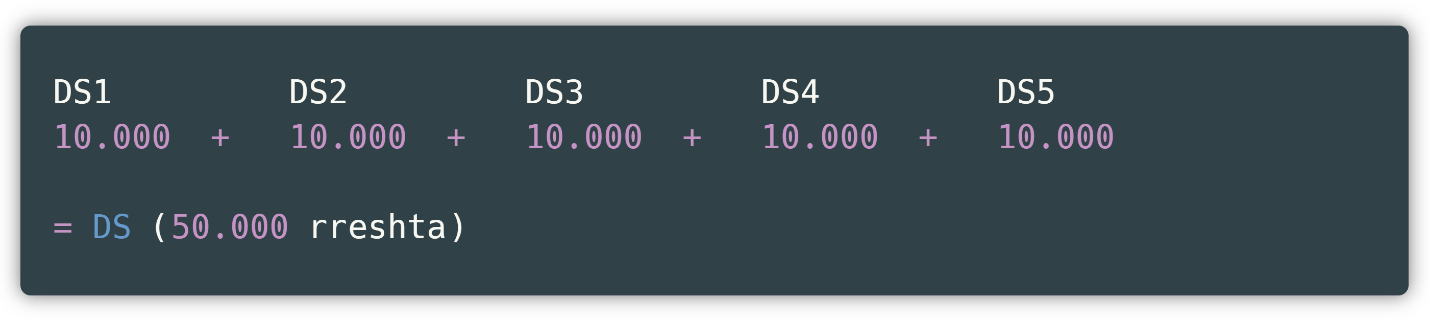


Me të përjashtuar këto raste, jemi siguruar që secila JSON strukturë të jetë e vlefshme dhe të ketë të dhëna përshkruese, në mënyrë që algoritmi të jetë sa më i saktë.

Ky ka qenë rasti i vetëm në të cilin ka qenë nevoja për intervenim nga jashtë, përndryshe këto të dhëna nuk përmbajnë asnjë vlerë asgjëje (ang. *null)*, vlerë jo-përshkrueseapo të ndonjë lloji tjetër që do ta dëmtonte data setin tonë në çfarëdo mënyre.

Duhet cekur gjithashtu se të gjitha të dhënat e këtij data setijanë të formës së vazhdueshme, për këtë arsye ka qenë më se e nevojshme aplikimi i heqjes së të veçuarve (ang. *outliers),* ndërsa atributet *leaver\_status* dhe *level* nuk janë prekur nga kjo procedure, ngase respektivisht njëri atribut është binarëndërsa tjetri është diskret*.*

Meqenëse ekziston një kufi prej rreth 11.000 thirrjeve të njëpasnjëshme në lidhjen e mësipërme që është i paraparë ta ndaloj abuzimet dhe sulmet e ndryshme, ne e kemi ndarë procesin e dërgimit të kërkesave në 5 pjesë, ku secila pjesë përmban 10.000 thirrje të njëpasnjëshme që rezulton në pothuajse çerekun e data setit përfundimtarë. E më pas këto 5 pjesë i kemi bashkuar për të krijuar një data set të plotë, në të cilin po operojmë tani.



Ky data set i plotë përmban 13 kolona dhe 50.000 rreshta. Se si është realizuar kjo procedurë do të tregohet më vonë pasi të hyjmë në pjesën e realizimit të konceptit përmes gjuhës programuese përkatëse

## Metrika gjenerike për data setin e derivuar

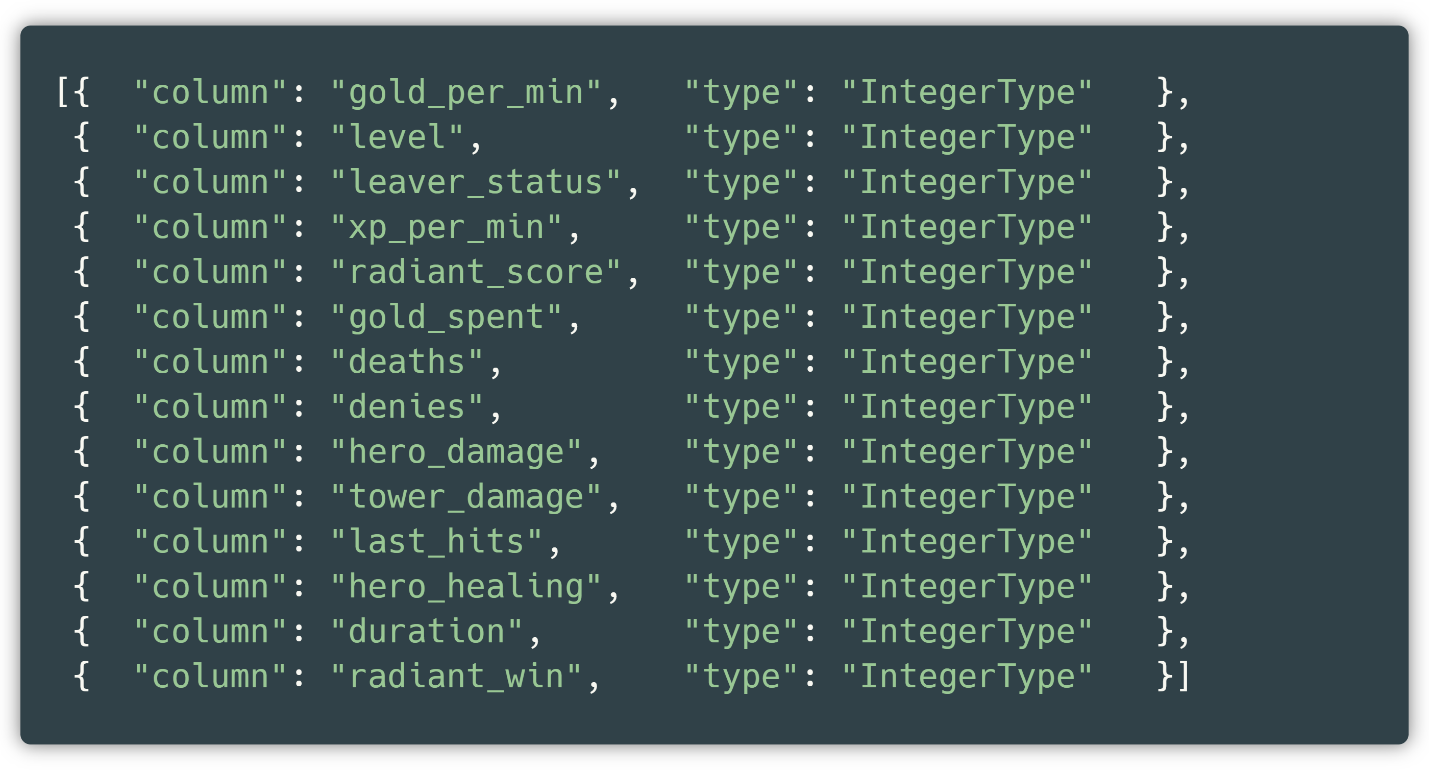
Për të pasur më shumë njohuri rreth data setit, lidhjes dhe te dhënave në përgjithësi, aplikojmë disa nga metriket e krijuara. Duhet pasur parasysh se të gjitha këto metrika po kryhen në një mostërme 1000 rreshta për arsye se procedimi në data setine plotë do të kërkonte më shumë kohë.

Fillojmë me thirrjen fundore getSample nga e cila shohim një mostër të data setit tonë (Figura 7: Mostra e data setit), nga shohim se kemi vlera numerike të vazhdueshme në shumicën e atributeve, përveç *leaver\_status* dhe *level* që mund të konsiderohen vlera diskrete, e njëra konkretisht binare.

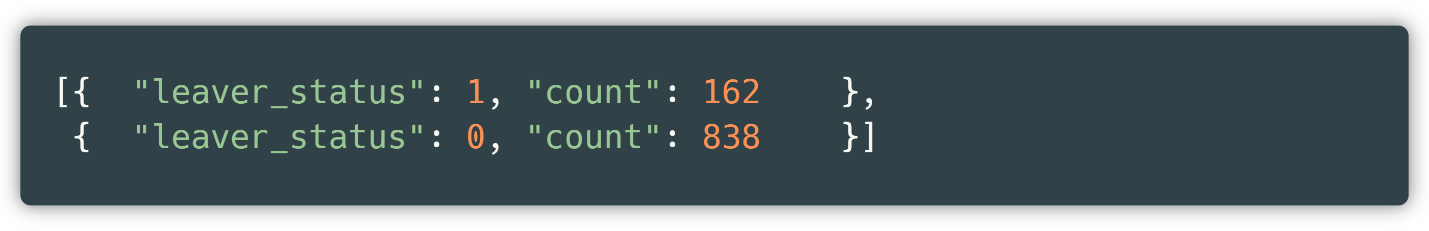
Meqë vetëm se kemi pasur njohuri paraprake rreth kësaj loje dhe se si funksionon ajo, përzgjedhja nuk ka qenë pjesa më e vështirë, ku në një strukturë JSON me rreth 65 atribute, ne kemi zgjedhur atribute që i përmban ky JSON file dhe atribute që ne kemi nxjerrë vetë, dhe ka rezultuar të jetë një veprim i suksesshëm.



Shohim secilën kolonë dhe tipin përkatës nëpërmjet thirrjes getSchema dhe shohim se të gjitha kolonat janë të tipit numër (Figura 8: Kolonat me tipet përkatëse. Pra, na lejohen metriket numërore si krahasime, filtrime numërore e të tjera. Mos të harrojmë që në klasën *Match* secilit atribut i kemi shoqëruar tipin numër.



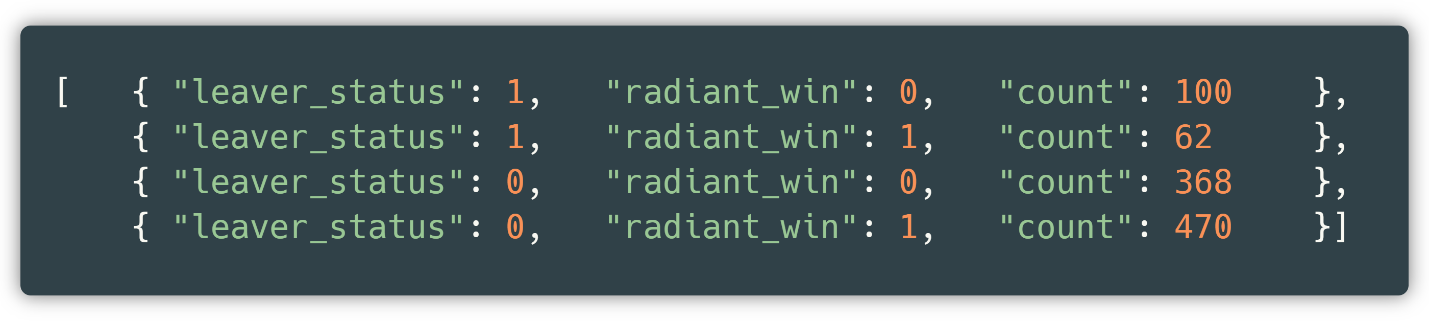
Nëse aplikojmë grupimin dhe njehsimin në *leaver\_status*, nëpërmjet getGroupAndCount (parametër nënkuptohet që vendoset *leaver\_status)* shohim se ndarja nuk është një ndarje e mirëfilltë, pra rreth 80% e të dhënave janë në *leaver\_status* 0dhe rreth 20% janë në *leaver\_status* 1



Nëse aplikojmë metrikën tjetër, e cila është një thirrje e brendshme në të cilën duam të kontrollojmë se si qëndron raporti i *leaver\_status* me kolonën të cilën ne duam të predikojmë, pra getDoubleGroup, marrim.

Pra, kur *leaver\_status* nuk është zero do të thotë që loja nuk është përfunduar si ekip dhe si rrjedhojëtë bardhëtkanë humbur pothuajse 65% të lojërave.

Gjithsesi, se përzgjedhja e kolonave ka qenë e mirë, mund të vërtetohet nëpërmjet matricës së korrelacioneve, getCorrelationMatrix, një matricëe cila paraqet lidhshmërinë e kolonave ndaj njëra tjetrës.



Pra kemi edhe korrelacione, edhe korrelacione inverse, ku një raport i plotë i kësaj matrice është i bashkangjitur në *Github*. Le të shohim tash atributet tjerë, vetëm sa të marrim një parafytyrim se me çfarë të dhënash kemi të bëjmë.

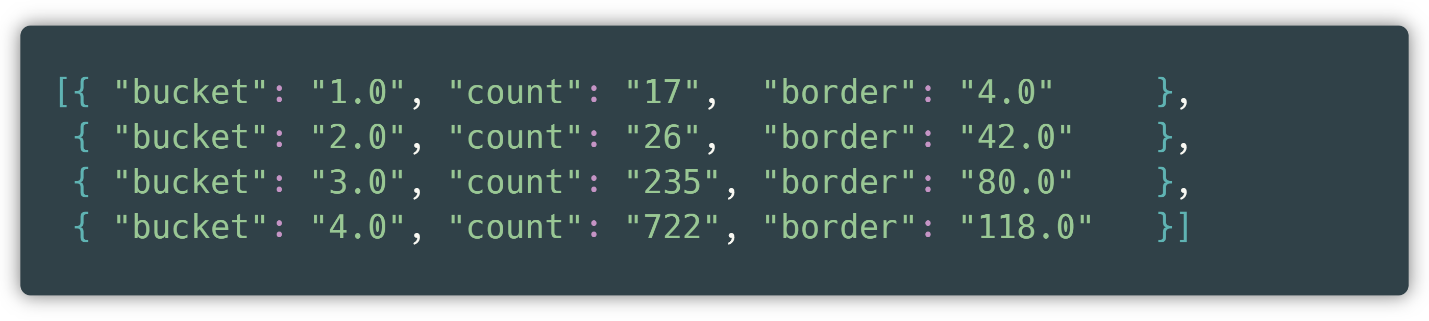
* Funksioni: grupimi sipas kolonës *xp\_per\_min* në 3 ndarje (Figura 11: Grupimi sipas kolonës xp\_per\_min në 3 ndarje)

Rezultati: fitojmë një ndarje relativisht të mirë ku në intervalit 0 ~ 472 i takojnë 828 elemente, intervalit 472 ~ 4755 i takojnë 113 elemente ndërsa intervalit 4755 e tutje i takojnë rreth 60 elemente.



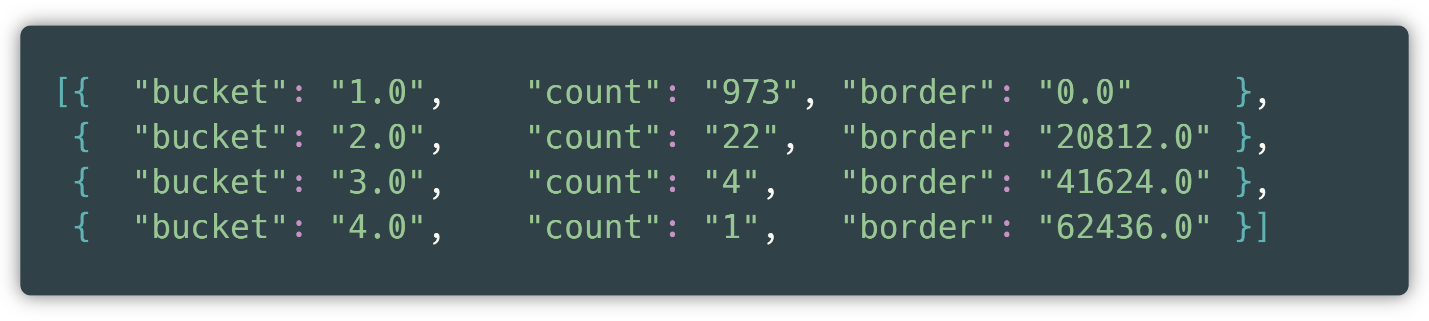
* Funksioni: grupimi sipas kolonës *level* në 4 ndarje (Figura 12: Grupimi sipas kolonës level në 4 ndarje)

Rezultati: edhe njëherë vërteton se data seti jonë është i shëndetshëm, ngase është lehtë e kuptueshme nga lojtarët e zakonshëm të kësaj loje se pjesa dominuese e lojërave mbaron me të gjithë lojtarët me *level* maksimal, që vërtetohet nga ky diagram.



* Funksioni: grupimi sipas kolonës *hero\_healing* në 3 ndarje (Figura 13: Grupimi sipas kolonës hero\_healing në 3 ndarje)

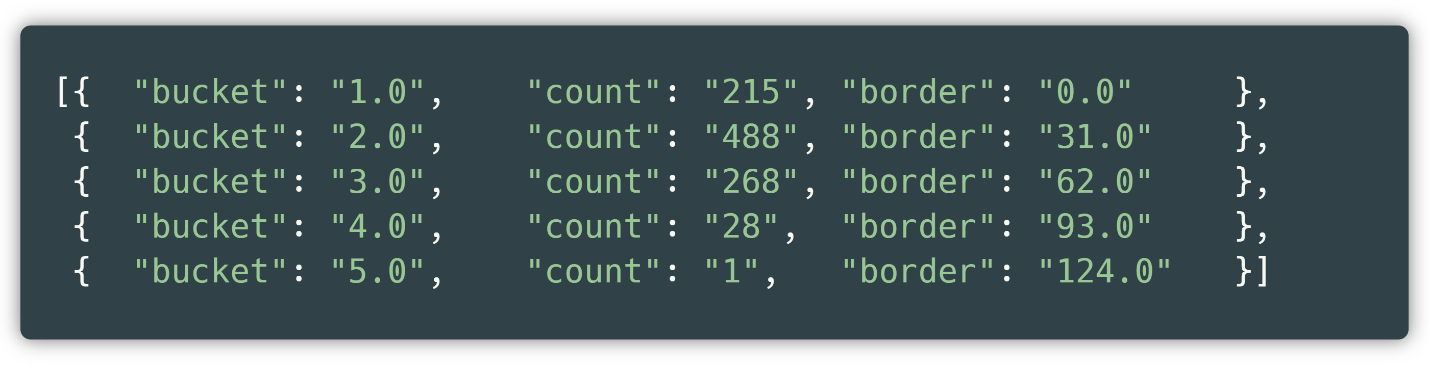
Rezultati: automatikisht krijohet edhe një ndarje e nivelit 0, ngase ka popullim të shumtë. Përndryshe, kjo ndarje na len të kuptohet se ekzistojnë dy grupe kryesore, ato që shërojnë dhe ato që nuk shërojnë.



* Funksioni: grupimi sipas kolonës *denies* në 4 ndarje (Figura 14: Grupimi sipas kolonës denies në 4 ndarje)

Rezultati: krijohet një ndarje në 5 grupime, për arsye se edhe niveli 0 konsiderohet.

Mund të shihet se intervali kryesorë qëndron në mes numrave 0 – 100. Për derisa një pjesë e konsiderueshme gjendet në tri grupimet e para. Po ashtu këto të dhëna mund të thuhet se janë diskrete.



Grupimet e realizuara më sipër nuk janë praktikë e mirë pasi numri i kategorizimeve është shumë i vogël e nuk mund të nxirren të dhëna të shëndosha por është realizuar në këtë mënyrë për arsye të prezantimit (hapësirës), përndryshe, nëse dëshirojmë të nxjerrim të dhëna analitike rreth kolonave përkatëse është shumë më e arsyeshme të bëhen me vlera dhjetore apo edhe me qindëshe.

Pra, në total kemi zgjedhur 14 atribute nga disa dhjetëra (Figura 15: Atributet e zgjedhura nga struktura).

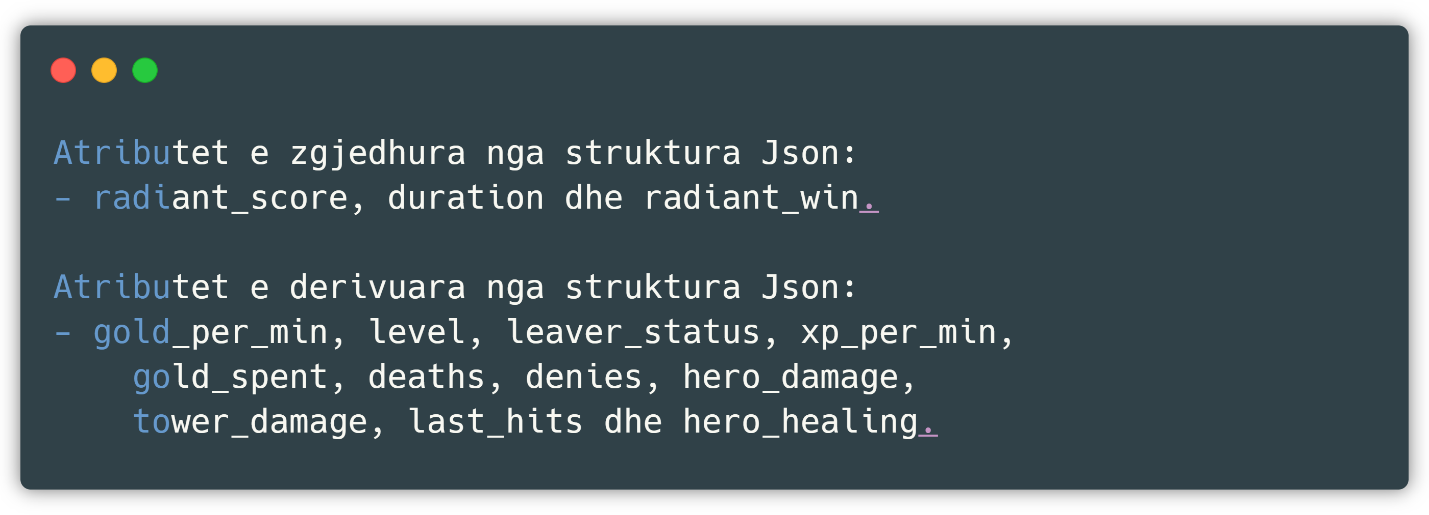


Figura 15: Atributet e zgjedhura nga struktura

Atributet e strukturës janë ato atribute të cilat nuk ka pasur nevojë të mëtutjeshme për përpunim përpara se të futen në data set, ndryshe nga atributet e nxjerra të strukturës, të cilat i kemi përbë nga copëza të vogla.

Duhet cekur se edhe atributet e përbëra janë përsëri atribute të strukturës JSON, por jo që mund t’i qasemi në mënyrë të drejtpërdrejtë. Më poshtë po përshkruajmë se çfarë duam të themi me qasje të drejtpërdrejtë.

Forma e strukturimit të një skeme JSON paraqitet më poshtë (Figura 16: Struktura e një objekti JSON). Shihet se si atribut kryesorë është *result*, që konsiderohet prind për secilin atribut tjetër. Fëmijët e këtij atributi mund të kategorizohen në dy grupe: atributet me nën atribute siç është *players* i cili përbëhet nga 10 objekte, secili objekt me veçori përkatëse për një lojtarë specifik.



Figura 16: Struktura e një objekti JSON

Këto objekte i kemi agreguar me shumë në çelësa përkatës dhe kemi krijuar të dhëna për data setin tonë të ri. Ndryshe nga ky lloj i atributeve, kemi edhe atributet vlerat e të cilëve mund t’i marrin direkt, pa agregime apo përpunime të mëtutjeshme. Operimi me këto atribute është shumë më i lehtë sesa me atributet të cilat duhet derivuar apo përpunuar, por ky numër qëndron në raport 2/8 ndaj atributeve që duhet përpunuar, numër ky i vogël.

Manovrimet dhe manipulime me data setin aktual i kemi kryer për arsye se Scala me libraritë e saj ofron mundësi shumë të mira për të kaluar të dhënat nga skema JSON për në klasën tonë Match, përshtatje kjo që realizohet me kod minimal e konciz, pra në njëfarë mënyre na është imponuar një mënyrë e tillë e punës. Le të shohim se si i kemi realizuar dhe organizuar të dhënat dhe kodin në projektin aktual në Scala.

## Mënyra e *Scalas*

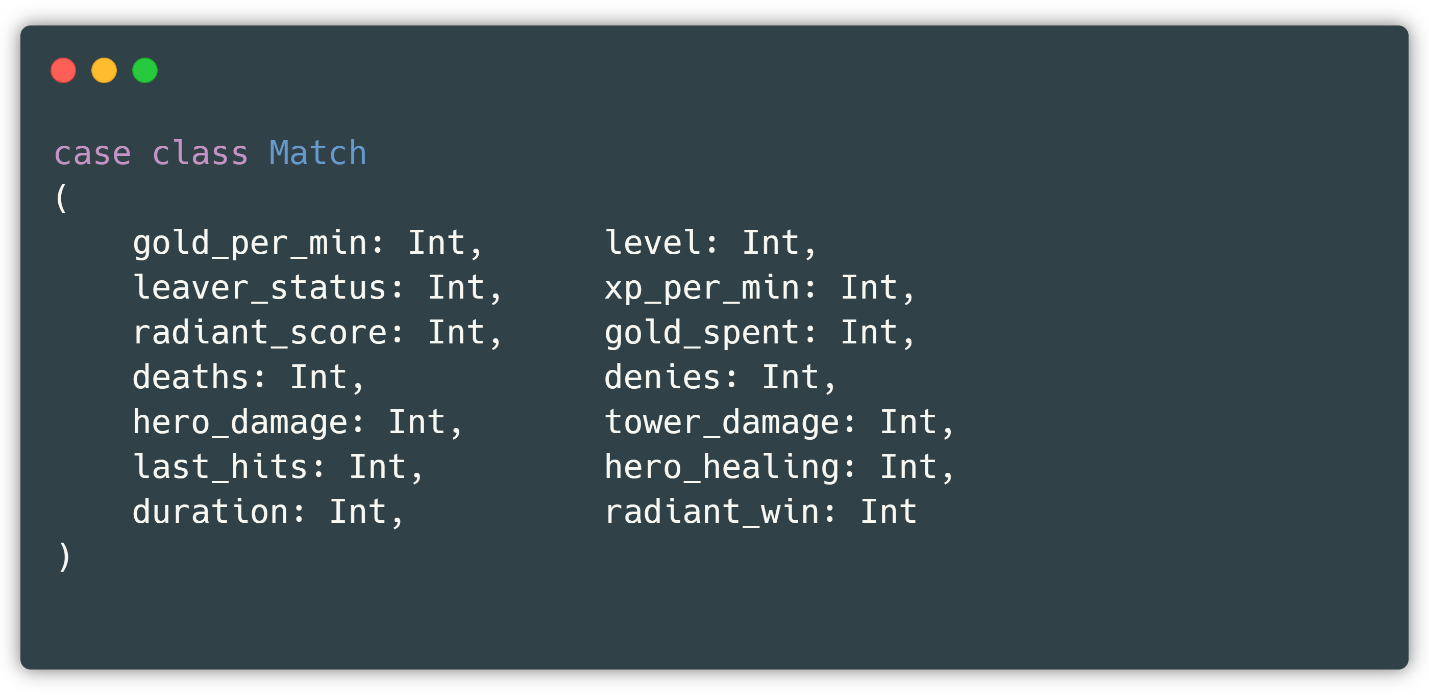
Përpara se të vazhdojmë me shpjegimin e projektit, duhet zgjeruar në pika të shkurtra konceptin Gson.

Gson [12] është një librari e gjuhës programuese Java që ndër shumë funksione tjera, na lejon konvertimin e Javaobjekteve në reprezentimin përkatës të strukturës JSON. Meqë Java ekzekutohet në Makinën Virtuale Java (ang. *Java Virtual Machine, JVM)*, do të thotë që edhe Scalamund të përdorë po këtë librari [13].

Në rastin tonë, ne kemi përdorur Gson që të mund të kalojmë të dhënat nga struktura JSON në një Scala klasë situate (ang. *Scala case class).* Një *case class* na lejon që të definojmë modelin apo strukturën e data setit tonë, ku një rresht i data setit është një objekt i ri i kësaj klase. Gson na është mundësuar pasi kemi shtuar vartësinë në përbërjen e projektit si *gson* nëpërmjet lidhjes *com.google.code.gson,* versioni i të cilit është 2.8.5.

Për të shpjeguar më mirë deklarimin e më sipërm, përdorim një shembull. Le të themi se kemi një JSON skemë me një atribut të vetëm të quajtur atributi, vlera e të cilit është 10 dhe krijojmë një *case class* me emrin Klasa me një atribut të quajtur atributi dhe e përdorim metodën *fromJSON* atëherë na krijohet një objekt i ri i klasës Klasa me variabël atributi që ka vlerën 10. Kjo është një praktikë shumë e mirë dhe e përdorshme për të krijuar klasa në mënyrë automatike, e një seri e gjatë e këtyre klasave rezulton në një data set.

Modeli jonë, i quajtur Ndeshja (ang. *Match*, Figura 17: Modeli i klasës Match)përbëhet nga 14 atribute, e secili nga këto atribute duhet gjithsesi të figurojë edhe në strukturën JSON të marrë nëpërmjet lidhjeve të cekura më sipër (nëpërmjet Steam API). Secilit atributit i përket tipi numër (ang. *Integer, Int)*.



Siç u cek edhe më lartë, kjo klasë shërben vetëm për të krijuar një objekt të ri për secilin rekord të sjellë nëpërmjet lidhjeve të realizuara në klasën kryesore në intervalet e caktuara. Objekti i krijuar i klasës Match më pas ruhet në një sekuencë. Sekuenca paraqet një koleksion të Scalas që me importimin e një funksioni të caktuar të librarisë Spark, i lejohet konvertimi në një data set dhe anasjelltas. Meqë data seti është i pa ndryshueshëm, neve na nevojitet që të krijohet data seti komplet në një sekuencë e më pas ta konvertojmë. Gjatë kësaj procedure, ne gjurmojmë shumën e objekteve të sekuencës në mënyrë që nëse arrin një numër të caktuar të ndaloj, pasi është arritur madhësia e data setit të dëshiruar. Ky numër në kodin tonë është konstantja *FEEDS*. Funksionet e Spark i marrim nga vartësitë *spark-core* dhe *spark-sql* nga *org.apache.spark,* versioni i së cilës është në këtë projekt 2.4.4.

Kjo mënyrë e përshkrimit është e nivelit të lartë, ndërsa më poshtë (Figura 18: Unaza kryesore e aplikacionit) përshkruhet saktësisht rrjedha e programit, variablat e përdorura dhe logjika kryesore e krijimit të një rekordi të vlefshëm. Fillojmë me krijimin e një lidhje të saktë në mënyrë që të kryejmë kërkesën. Kjo lidhje është krijuar nëpërmjet tre stringjeve: *steam api, gameId* dhe *steam key*, ku *steam api* dhe *steam key* janë konstanta ndërsa *gameId* ndryshon varësisht intervalit të mëposhtëm, që fillon me *start id* dhe mbaron me *end id*.



Figura 18: Unaza kryesore e aplikacionit

*Args* paraqet një varg që i kemi dhënë vlerat përmes *argumenteve programore (ang. program arguments* [14]të IntelliJ, vegla softuerike me të cilin është përpunuar aplikacioni), e ky varg përmban të gjitha vlerat e përfituaratë cilat janë gold\_per\_min, level, leaver\_status, xp\_per\_min, gold\_spent, deaths, denies, hero\_damage, tower\_damage, last\_hits dhe hero\_healing (si më lartë)

Krahas këtyre informatave, shihet edhe objekti me metodën përkatëse *Fetcher.fetchGames* (Figura 19: Struktura e metodës fetchGames),metodë statike (ang. *static method)* kjo e cila na ndihmon që të mbledhim të dhënat. Në këtë metodë definohet kërkesa (ang. *request)* e cila mundëson qasjen në Steam Public API, kontrollohet se a është kërkesa e vlefshme, dhe nëse po, atëherë kthehet (ang. *returns)* një instancë e re e klasës Match me të gjitha atributet e specifikuara sipër të marrura nga JSON struktura në të cilën është bërë kërkesa. Mundësia për kërkesa na është bërë e mundshme pasi të shtojmë *requests* nga *com.lihaoyi*, versioni më i ri i së cilës është 0.1.8.

Ky objekt më pas shtohet tek sekuenca e quajtur *matches* dhe kështu përsëritet procesi deri sa të formulohet një sekuencë me 500.000 të dhëna.



Figura 19: Struktura e metodës fetchGames

Pas këtij procesi, krejt çka mbetet është konvertimi në një *Spark Dataframe* dhe ruajtja e dosjes në mënyrë të pastër (ang. *cleaned-version)* në mënyrë lokale (Figura 20: Konvertimi dhe ruajtja e data setit në makinën lokale).



Figura 20: Konvertimi dhe ruajtja e data setit në makinën lokale

Ku krijohet një dosje e re në shtegun (ang. *path)* përkatës, i cili është definuar po ashtu nëpërmjet *IntelliJ -* variablat e mjedisit (ang. *environment variable)*. Ky përcaktim lehtëson punën në grup, ku ekipi punues në vend që të ndryshojë vlerat e variablave statike në mjedisin lokal, ndryshon vetëm variablat e mjedisit.

Pra, aktualisht kemi në makinën tonë lokale një *data set* të gatshëm dhe të pastër, në të cilin mund të aplikojmë metrika të ndryshme e edhe algoritme të mençura. Një mostër e marrjes dhe analizimit të të dhënave duket si më poshtë (Figura 21: Pamja e marrjes së të dhënave).

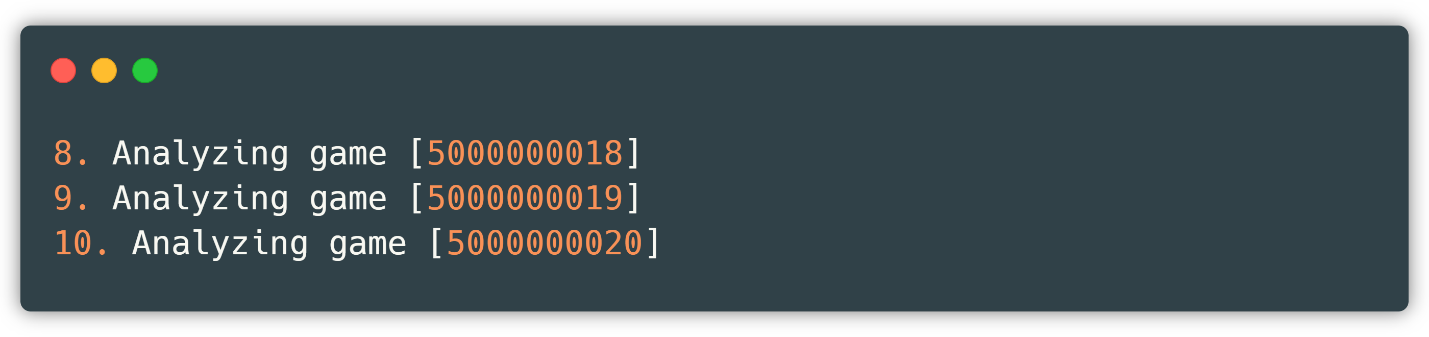


Figura 21: Pamja e marrjes së të dhënave

## Klasifikimi në Scala

Me krijimin e data setit, ne tashmë jemi ne gjendje të bëjmë një predikim në atributin *radiant\_win*.

Si hap i parë, normalisht që importohet data setinëpërmjet librarisë Spark (Figura 22: Leximi i data setit të krijuar).

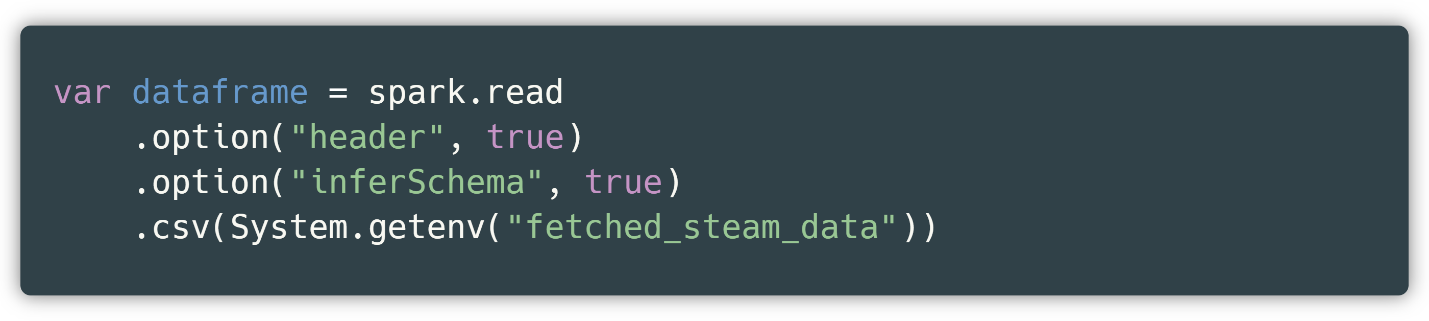


Figura 22: Leximi i data setit të krijuar

Më pas duhet riemëruar atributi *radiant\_win* në etiketë (ang. *label)* pasi mënyra se si algoritmi funksionon është ashtu që si hyrje duhet të ketë një atribut të vetëm, të quajtur cilësitë (ang. *features)* kurse atributi që predikohet duhet quajtur *label*.

Përveç kësaj cilësie, një data setime të dhëna të vazhdueshme (ang. *continual)* është gjithmonë e arsyeshme të përdoret një shkallëzues (ang. *scaler)* [15]bashkë me një metodë për largimin e të veçuarve*,* në mënyrë që algoritmi të mos dominohet nga një atribut i vetëm.

Si një veçori e re që i është shtuar Sparkut janë fazat (ang. *pipelines).* Por çfarë saktësisht na ofron neve kjo veçori? Në vend që të aplikohen këto tre hapa në mënyrë të veçantë, ne mund t’i vendosim në një fazë dhe të aplikojmë algoritmin në fazën kryesorë e jo në një hap të posaçëm (Figura 23: Renditja e fazave sipas realizimit).

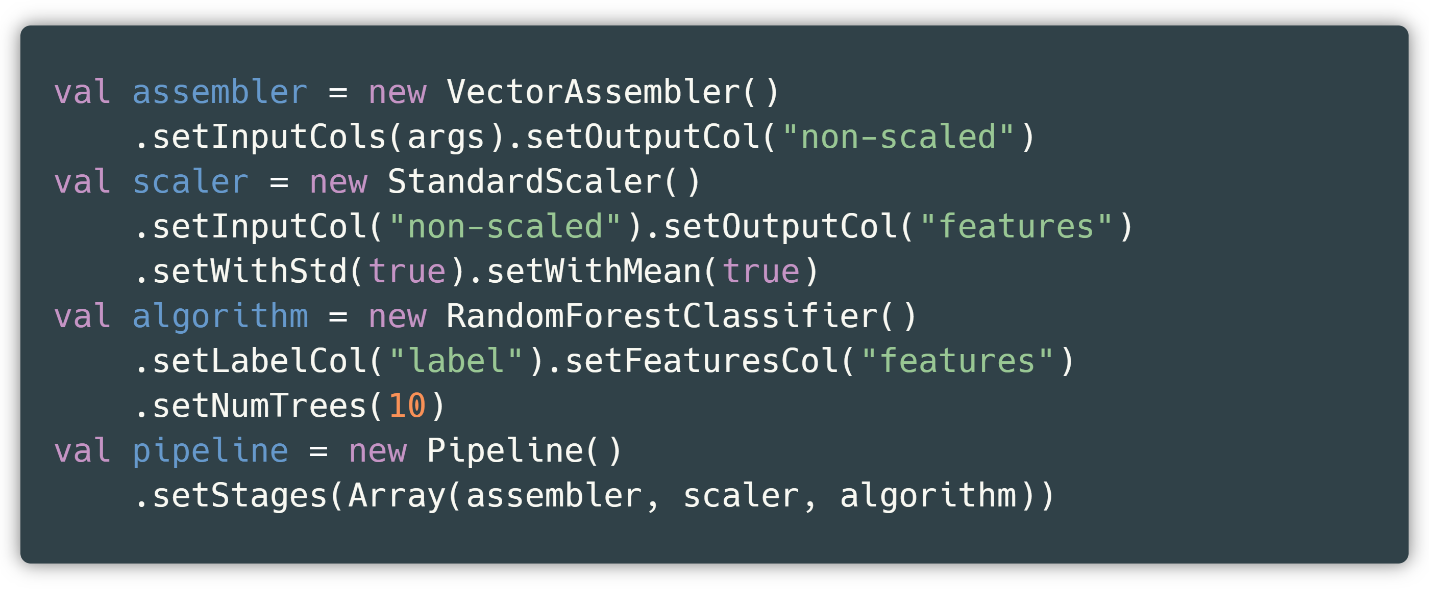


Figura 23: Renditja e fazave sipas realizimit

Shohim se kemi tre funksione kryesore:

* *vecAssembler\_id* – qëndron për *VectorAssembler* dhe shërben që një vargtë kolonave të grumbullojë në një atribut të vetëm, të quajtur në rastin tonë tëpa shkallëzuarat (ang. *non-scaled).* Pra, si kolona hyrëse (ang. *inputColumns)* pranon kolonat që duam t’i trajnojmë dhe si kolonë dalëse (ang. *outputColumn)* është kolona e re e krijuar *non-scaled*.
* *stdScal* – qëndron për *standardScaler* dhe shërben që të kthejë të gjithë numrat relativisht në një interval të caktuar, në rastin tonë ndërmjet 0 dhe 1. Pra, secila kolonë ka vetëm vlera të transformuar në intervalin 0 – 1. Përmban dy metoda statike që janë *withMean* dhe *withStd*. E para përgatitë të dhënat duke ua marrë vlerën mesatare para se ti shkallëzoj, ndërsa e dyta cakton devijimin standard. Gjithsesi, pesha e kolonave nuk ndryshon pavarësisht këtyre mutacioneve.
* *rfc* – qëndron për *Random Forest Classifier* (i përshkruar më lartë), ofron metodat përkatëse me anë të së cilave mund të trajnojmë dhe aplikojmë metrika në të dhëna të reja.

Pra, nëse bëjmë një përmbledhje të shkurtër rreth hapave të një data setitqë kalon deri në predikim; atributet që shërbejnë në predikim bëhen një dhe riemërohen si *të pa shkallëzuara (ang. non-scaled),* ky atribut më pas shkallëzohet dhe riemërohet në *features,* e në fund aplikohet algoritmi *Random Forest Classifier* ku fitohet një data seti cili përmban një kolonë më shumë se parapraket të quajtur *prediction,* vlera e të cilit është ajo çfarë ne dëshirojmë.

Fazat jo vetëm që lehtësojnë procedurën, por edhe bëjnë kodin më konciz, të lexueshëm dhe më të manovrueshëm. Një vizualizim i gjërave që u thanë më sipër duket kështu (Figura 24: Vizualizimi i fazave):



Figura 24: Vizualizimi i fazave

Saktësia e algoritmit tonë me një ndarje të të dhënave 3/10 aktualisht sillet deri 90%, siç shihet edhe më poshtë (Figura 25: Saktësia e algoritmit Random Forest Classifier):



Figura 25: Saktësia e algoritmit Random Forest Classifier

Si metrika për vlerësim kemi përdorur atë të librarisë Spark që quhet *accuracy* e cila realizohet nëpërmjet klasë *MulticlassClassificationEvaluator* që përdor dy metoda kryesore që shërbejnë për marrjen e predikimeve aktuale dhe krahasimin me ato realet. Kthen probabilitetin e saktësisë, pra një numër prej 0 deri 1.

Më pas ruajmë modelin e trajnuar lokalisht në këtë mënyrë (Figura 26: Ruajtja e modelit të trajnuar në makinën lokale).



Figura 26: Ruajtja e modelit të trajnuar në makinën lokale

Përparësi e madhe e këtij veprimi është se nuk ka nevojë që të trajnohet modeli për secilën të dhënë që duam të predikojmë, por algoritmi ngarkon algoritmin e trajnuar dhe aplikon veprimin e caktuar në të, që kryhet në disa mili sekonda.

Përndryshe, nëse nuk do të ruanim algoritmin e trajnuar, do të duhej ta trajnonim atë për secilin predikim, që për data setin tonë, i duhen përafërsisht 5-6 minuta. E kjo do të rezultonte në eksperiencë të tmerrshme për këdo që tenton të aplikoj metrika në të.

## Largimi i të veçuarve

Siç edhe kemi cekur më lartë, për arsye të vlerave të vazhdueshme, mundësia e paraqitjes të *outliers* është e madhe andaj është mirë që të largohen të dhënat me këto veti përpara aplikimit të algoritmit *Random Forest.*

Për këtë pjesë ne kemi përdorur metodën për intervale ndër kuartile (ang. *interquartile range)*. Kjo metodë i është aplikuar secilës kolonë të data setit tonë, e metoda funksionon në atë mënyrë që së pari radhiten të dhënat në varg rritës dhe merren medianat e gjysmës së parë dhe gjysmës së dytë. Le të themi se Q1 është gjysma e medianës së parë ndërsa Q3 gjysma e medianës së dytë. Gjejmë diferencën në mes këtyre dy vlerave, IQR = Q3 – Q1, vlerë kjo e quajtur vlera ndër kuartile (nga edhe e ka marrë emrin) dhe ia shtojmë përkatësisht zbresim vlerës Q3 dhe Q1, pra, A = Q3 + 1.5 \* IQR dhe B = Q1 – 1.5 \* IQR ku 1.5 është një vlerë rritëse apo zvogëluese e intervalit që duam të marrim. Secilën vlerë në data set më të vogël se B dhe më të madhe se A i largojmë (Figura 27: Shembull i gjetjes e një intervali ndër kuartile).

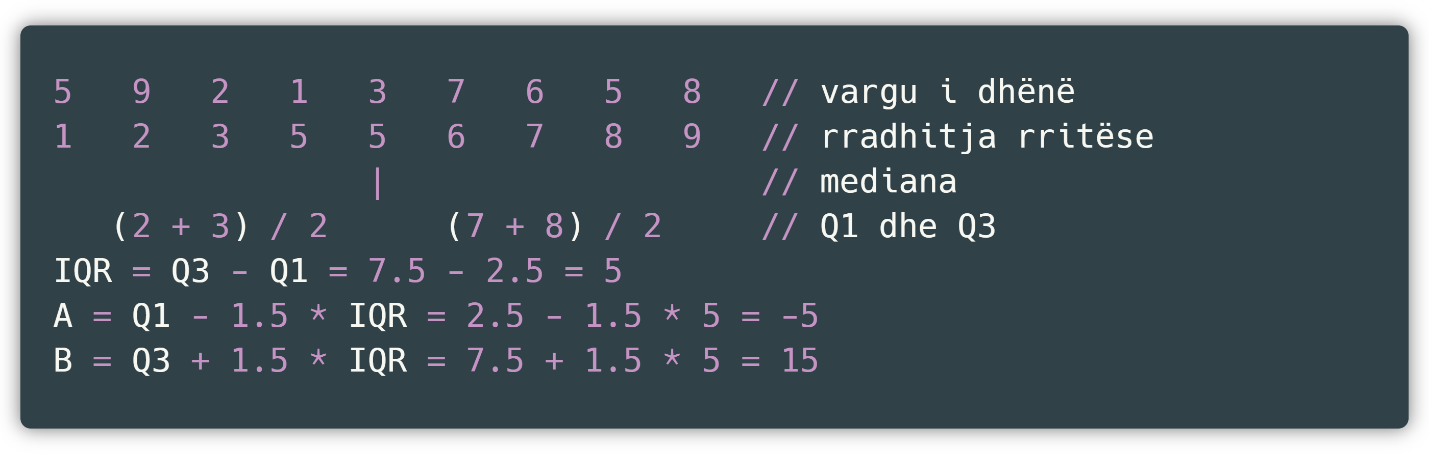


Figura 27: Shembull i gjetjes e një intervali ndër kuartile

# Klasifikimi në kohë reale dhe ruajtja si model i trajnuar

## Struktura e aplikacionit

Siç e kemi cekur edhe më lartë, arkitektura në të cilin Play është ndërtuar është MVC, pra *Model, View* dhe *Controller.* Në rastin tonë, për arsye të integrimit të aplikacionit me Flutter, kemi çkyqur komponentin *View*, ndërsa komponenti *Model* nuk është përdorur pasi nuk kemi nevojë për lidhje me bazën e të dhënave, pasi të dhënat po i ruajmë në mënyrë lokale. Diçka si komponenti *model* mund të konsiderohet klasa *Match* e përmendur më lartë, që paraqet një strukturë në të cilën kalohen informatat JSON*.*

Pra, ne aktualisht përdorim një kontrollerdhe një pako (ang. *package)* të quajtur veglat (ang. *utilities)* nëpërmjet së cilës ekzekutojmë metoda të shumta (Figura 28: Struktura MVC në Scala).

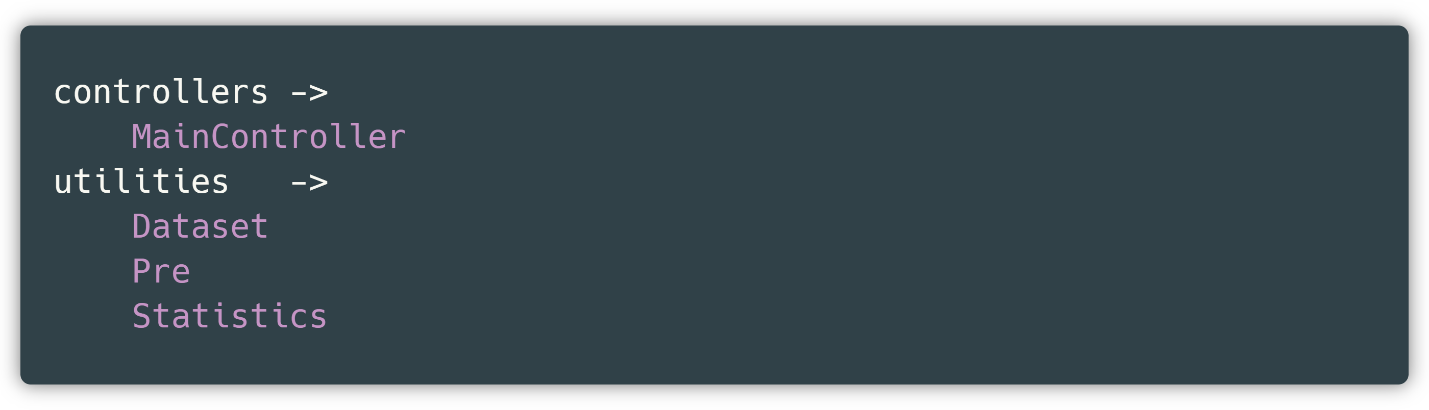


Figura 28: Struktura MVC në Scala

*MainController* paraqet kontrollerin kryesorë, që drejton secilën kërkesë në rrugën e duhur. Pra, secila metodë me rrugën (ang. *routing)* përkatëse menaxhohet nga ky kontroller.

*Pre* (Figura 29: Paraqitja e objektit Pre) paraqet po ashtu një objekt në të cilin ekzekutohen metodat globale për Sparkdhe për ngarkim të data seteve *steam* dhe *Kaggle.*

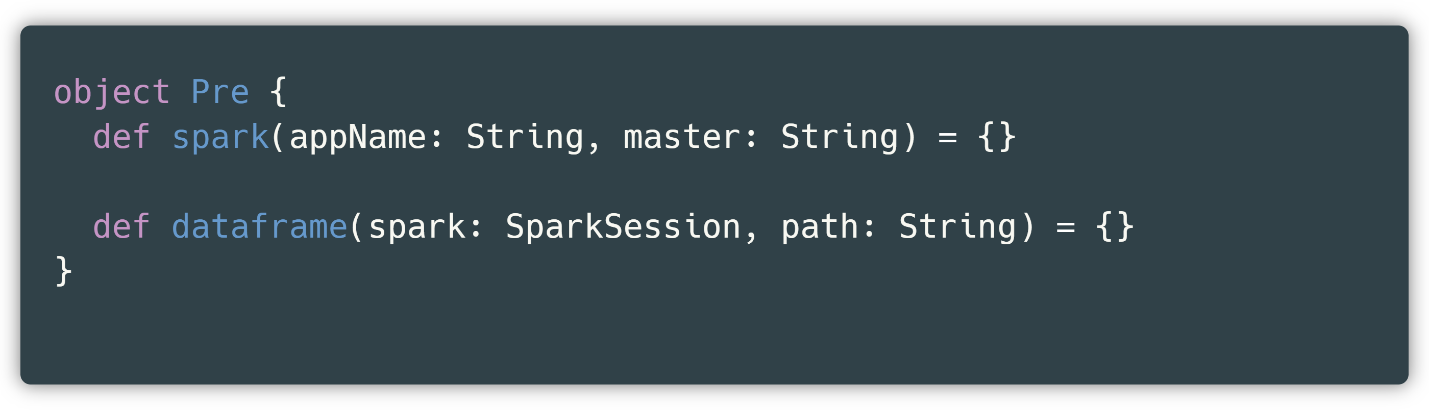


Figura 29: Paraqitja e objektit Pre

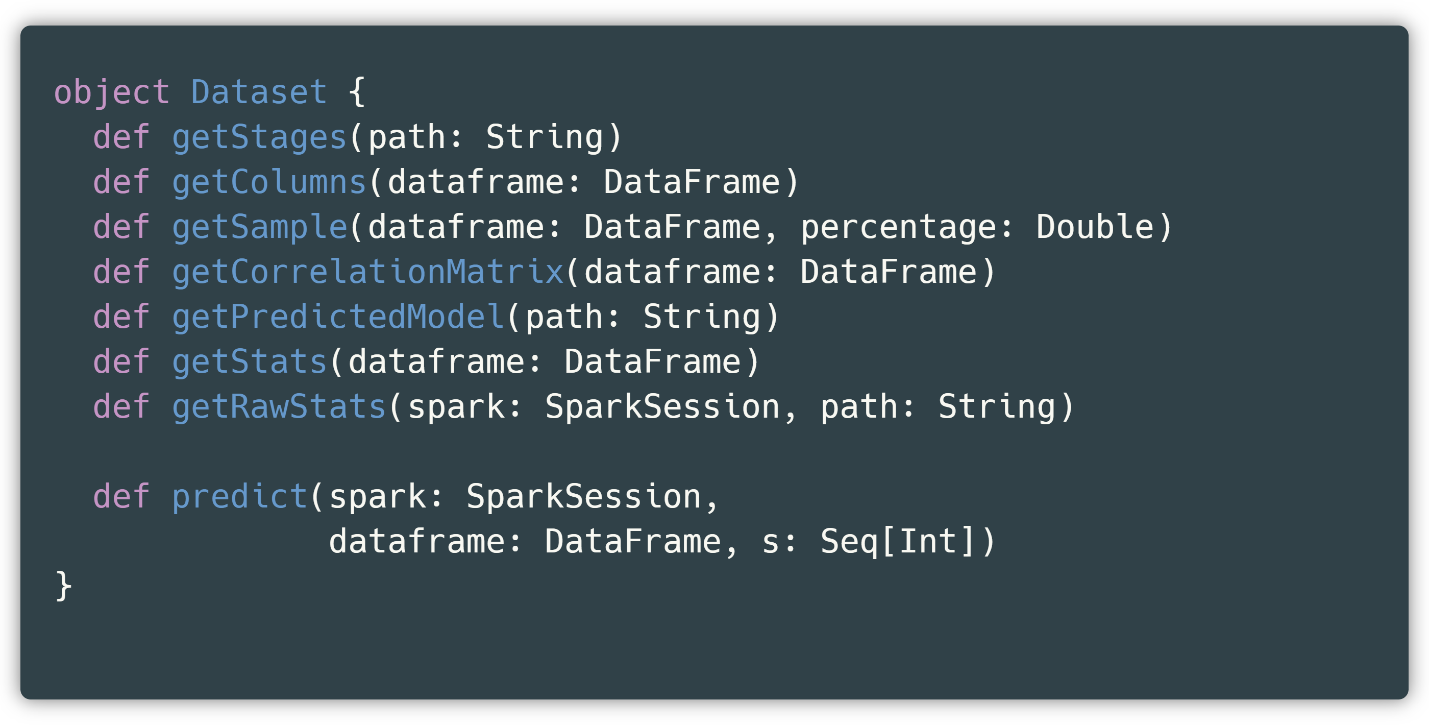
*Dataset* paraqet një objekt të Scalas që duket kështu (Figura 30: Paraqitja e objektit Dataset)

Figura 30: Paraqitja e objektit Dataset

në të cilin janë të organizuara të gjitha metodat me parametrat të kontrollerit përkatës *MainController.* Secila rrugë e kontrollerit pra ekzekuton njërën nga metodat e lartë cekura.

Dhe në fund *Statistics* përmban metodat e kontrollerit *MainController,* konkretishtmetodat *getGroupByAndCount* dhe *getBinary*. Meqë kompleksiteti i kësaj metode është më i madh, atëherë ka qenë nevoja që të krijohet një klasë e re për funksionalizim.

## Realizimi i klasifikimi në kohë reale

Rikujtojmë se ne gjatë programit tonë, kemi ruajtur modelin e klasifikuar. Çfarë do të thotë kjo?

Dihet që data seti duhet trajnuar, e më pas të testohet me të dhëna testuese, në rastin tonë, patëm një saktësi rreth 90% me të dhëna trajnuese. Por për një aplikacion normal, të dhënat trajnuese shërbejnë vetëm si provë (ang. *testing)* për të treguar se a ja vlen algoritmi jonë të sprovohet me të dhëna reale apo jo.

Algoritmi jonë ka rezultuar shumë i suksesshëm me të dhëna trajnuese të të njëjtit burim, por le të provojmë të testojmë me një vargtë dhënash krejt të një natyre tjetër.

Në aplikacionin tonë ofrojmë edhe një thirrje fundore të quajtur getStages, thirrje kjo që na ndihmon të kujtojmë parametrat dhe funksionet që kemi përdorur në algoritmin tonë, thënë ndryshe, mundësohet zgjatja e fazave të planifikuara varësisht algoritmit (Figura 31: Fazat të reprezentuar si JSON objekte – data seti 1).



Figura 31: Fazat të reprezentuar si JSON objekte – data seti 1

ku shihen emrat e klasave (për shembull *vecAssembler\_id)* dhe metodave të përdorura me vlerat përkatëse (për shembull *outputCol* me vlerën *non-scaled)*. E nëse krahasojmë me funksionin origjinal, gjithçka merr kuptim, pasi emrat e mësipërm përkasin komplet me metodat e shkruara më lartë. Si shembull po ilustrojmë edhe një rast tjetër, ku është përdorur algoritmi K-Means me parametra të caktuar (Figura 32: Fazat të reprezentuar si JSON objekte - data seti 2).

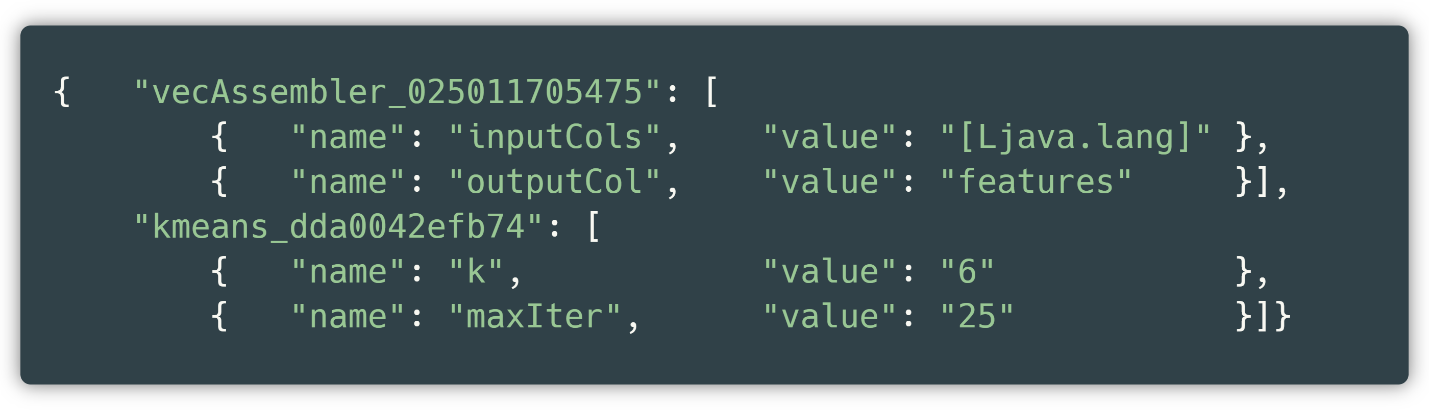


Figura 32: Fazat të reprezentuar si JSON objekte - data seti 2

Ku në bazë të këtij rasti, mund të themi se është përdorur klasa *vecAssembler* dhe klasa *kmeans* me metodat statike me metodat e tyre përkatëse (për shembull *name* me vlerën 6). Kjo është realizuar si në vijim

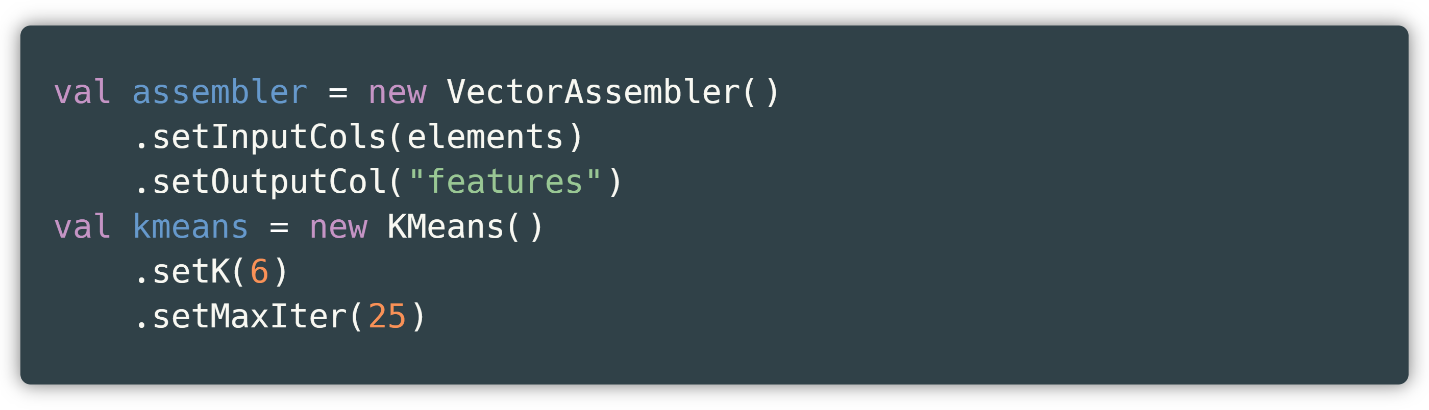


Figura 33: K-Means në Scala

Kalojmë në pjesën e predikimit ku duhet të krijojmë një kërkesë për aplikacionin tonë, përgjigjja e të cilit do të na tregoj rezultatin në formatin JSON (Figura 34: Forma e predikimit POST).

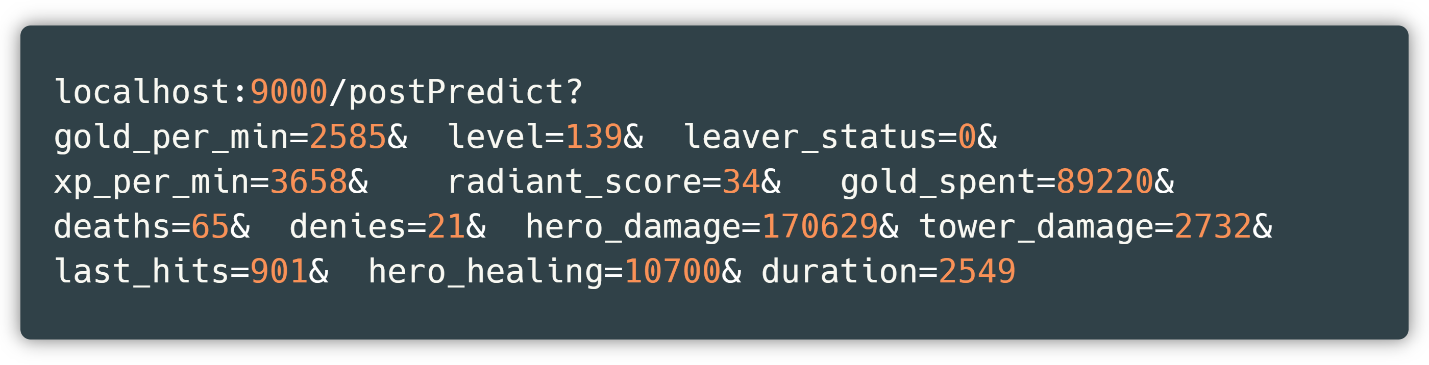


Figura 34: Forma e predikimit POST

Nga mund të shihet se atributeve gold\_per\_min, level, leaver\_status, xp\_per\_min, radiant\_score, gold\_spent, deaths, denies, hero\_damage, tower\_damage, last\_hits, hero\_healing dhe duration ua kemi shoqëruar vlerat përkatëse numerike. Dihet që numri i atributeve në lidhje është i njëjtë me numrin e atributeve që pranon pika fundore. Ekzekutojmë thirrjen dhe na vjen përgjigjja përkatëse (Figura 35: Rezultati i POST kërkesës për predikim).

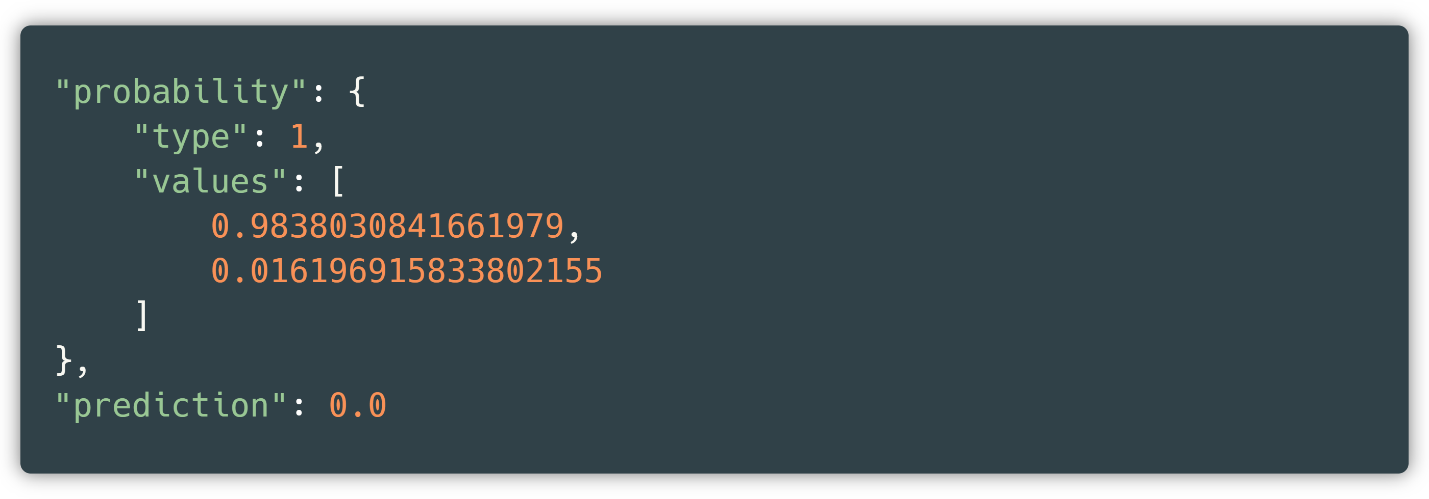


Figura 35: Rezultati i POST kërkesës për predikim

Nëse analizojmë këtë përgjigje, shohim se na kthehet një çelës (ang. *key)* me një tip (ang. *type)* që paraqet llojin e algoritmit që në rastin tonë është klasifikim, një listë vlerash që paraqesin probabilitetin dhe vetë predikimi.

Pra, algoritmi ka klasifikuar vlerat hyrëse nga ne me 0 me një probabilitet 0.983 që i bie 98%, një saktësi shumë e lartë.

Ndërsa se çfarë ndodhë në prapa skenë kur realizohet thirrja nëpërmjet lidhjes më sipër është se krijohet një Spark RDD – që paraqesin një formë tjetër të koleksioneve më primitive sesa data setet (Figura 36: Krijimi i RDD nga lista e atributeve hyrëse).



Figura 36: Krijimi i RDD nga lista e atributeve hyrëse

e më pas simulohet krijimi i një data seti me kolonat hyrëse që janë dhënë nëpërmjet pjesës hyrëse *(frontend)* nga RDD (Figura 37: Krijimi i data setit nga RDD)



Figura 37: Krijimi i data setit nga RDD

tani, *df* paraqet data setin tonë me një rresht. Me të krijuar këtë data set, mundemi që ta aplikojmë në modelin e trajnuar më herët (Figura 38: Modeli i trajnuar dhe klasifikimi në kohë reale).



Figura 38: Modeli i trajnuar dhe klasifikimi në kohë reale

Ku *getPredictedModel* kthen modelin e trajnuar në shtegun e caktuar të quajtur *classified\_model* dhe përmes metodës *transform*, realizohet klasifikimi në kohë reale dhe kthehet përgjigjja në formën JSON.

# Përgatitja e imazhit për ekspozimin e shërbimeve

## Çka është Docker?

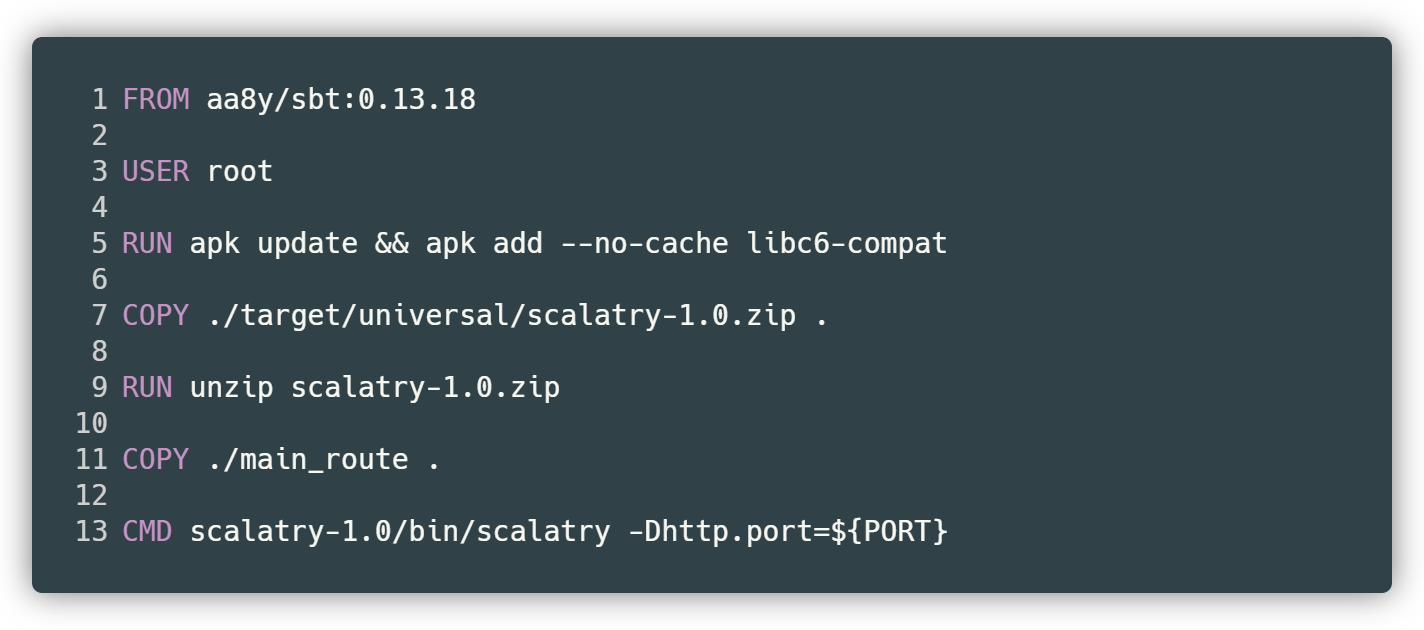
Përpara se të vazhdojmë më tutje, duhet të përshkruajmë se çka është Docker dhe si funksionon ai. Docker është një vegël që ndihmon në krijimin, zhvillimin dhe publikimin e aplikacionit në ueb. Këtë e bën duke paketuar softuerin tonë në paketa të quajtur kontejnerë. Kontejnerët janë të izoluar nga njëri tjetri, që do të thotë se posedojnë pavarësi të plotë. Motivimi për Docker, ndër shumë arsye tjera, ka ardhur nga ajo se për tu ekzekutuar programi nga dy sisteme të ndryshme është shumë më e lehtë të krijohet një Docker imazh që mund të ekzekutohet nga pala tjetër sesa pasimi i kodit burimorë. Arsyeja tjetër dhe kryesore është ajo se për të publikuar një sistem në ueb, që në rastin ka qenë publikimi i API të krijuara nga pjesa backend, Docker është vegla më e përshtatshme, sidomos kur aplikacioni është i shpërndarë në disa pjesë, si në rastin tonë, pjesa e data seteve, modelet e trajnuara për klasifikim dhe kllasterim e të tjera.

Po përshkruajmë më poshtë hapat kryesorë qe kemi marrë deri në krijim e një imazhi si të lartcekur.

## Krijimi dhe shfletimi i kontejnerit

Si hap i parë që duhet të kryejmë është të instalojmë Docker nga faqja zyrtare.Çdoherë kur krijojmë një imazh Docker, duhet të ekzistojë një tekst i quajtur Dockerfile në direktoriumin e projektit kryesorë. Në rastin tonë, ne e kemi krijuar këtë dosje në direktoriumin rrënjë të projektit (ang. *root directory*).

Duhet konfiguruar Dockerfile ashtu që ekzekutimi i tij, të mundësoj krijimin e imazhit tonë. Përmbajtja e Dockerfile tonë është si vijon:

****

ku rreshti numër 1 paraqet imazhin kryesorë dhe ambientin në të cilin do të ekzekutohet aplikacioni. Meqë projekti jonë është i ndërtuar në Scala dhe si bazë për ndërtim (ang. *build*) ka SBT-në, atëherë ne përdorim po të njëjtin imazh me po të njëjtin version si projekti, pra 0.13.18. Më pas, caktojmë përdoruesin si *root* (hapi 3) në mënyrë që të kemi qasje administrative. Duhet theksuar se ky imazh nuk është një imazh zyrtarë nga Docker por i krijuar nga një person apo persona të caktuar, që në rastin tonë është zhvilluesi *aa8y*. Pra, ne po krijojmë një imazh nga një imazh tjetër që po përdorim si mjedis zhvillues në imazhin tonë.

Përditësojmë ambienton tonë dhe shtojmë *libc6-compat* që mundëson që libraritë të mos kenë konflikte dhe siguron që aplikacioni të mos ketë gabime gjatë ekzekutimit (hapi 5).

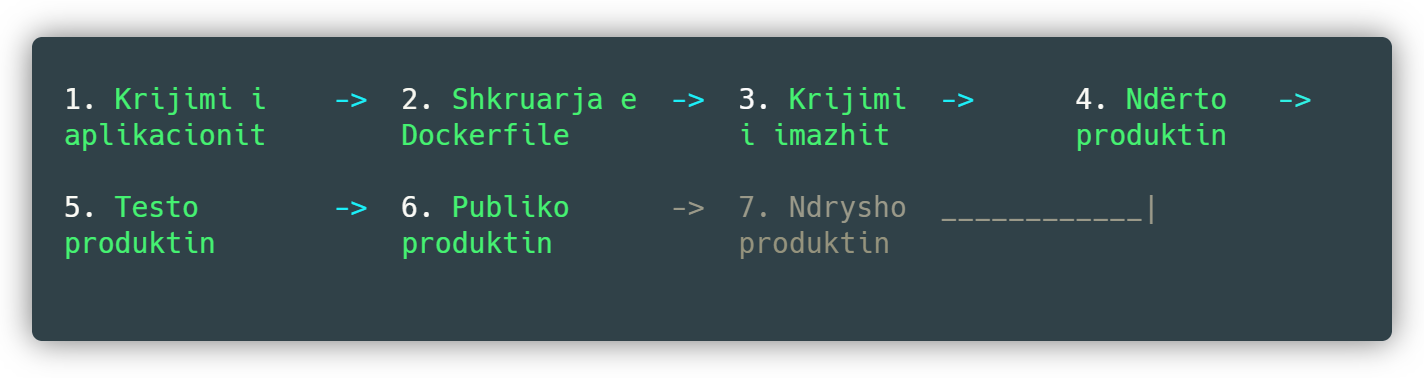
Kopjojmë shtegun ./target/universal/scalatry-1.0.zip në direktoriumin bazë të imazhit tonë që nuk e kemi krijuar akoma (hapi 7). Shtegu i lartcekur mundësohet nga *sbt shell* me anë të komandës *dist* që krijon një aplikacion të kompresuar gati për publikim, pra me librari dhe vartësi të gjitha të gatshme dhe të paketuara në të.

Pas kopjimit të dosjes së projektit në direktoriumin bazë të imazhit, e dekompresojmë atë (hapi 9). Ky hap duhet kryer në mënyrë që të kemi dosjen e hapur për ekzekutim.

Më në fund kopjojmë dosjen tonë lokale *main\_route* dhe e vendosim po ashtu në direktoriumin bazë të imazhit Docker (hapi 11). Rikujtojmë se *main\_route* paraqet dosjen në të cilën janë të vendosura të gjitha data setet e nevojitur për aplikacion bashkë me modelet e trajnuara, pra shtegu ./main\_route/fetched\_steam\_data paraqet data setin e krijuar si më lartë.

Hapi 13 paraqet një aksion abstrakt, që do të thotë se pas ekzekutimit të hapave më lartë, duhet vizituar shtegu scalatry-1.0/bin/ dhe ekzekutuar BIN fajlli scalatry. Prapashtesa -Dhttp.port paraqet një aksion që detyron aplikacionin të ekzekutohet në portin ${PORT}. Ky port vjen nga *GCP (Google Cloud Platform*), pasi si rregull i GCP është që porti në të cilën do të publikohet ueb-faqja të jetë dinamike, e jo të caktohet nga zhvilluesit e aplikacionit (në rastin tonë ne).

Një përmbledhje e të gjithë hapave më sipër është përshkruar më poshtë



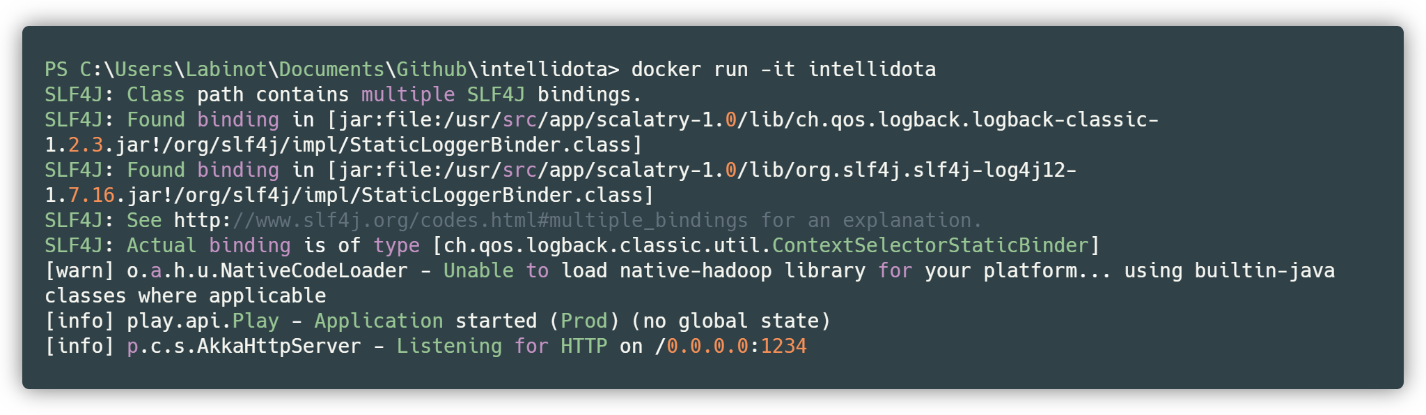
Me të kryer këto hapa, imazhi jonë është gati për tu krijuar. Për të vazhduar më tutje, hapim një *CLI* (*Command Line Interface*) në mënyrë që të ndërtojmë imazhin dhe përdorim komandat



Komanda e parë mundëson kalimin te direktoriumi ku gjendet projekti, ndërsa komanda e dytë ndërton imazhin. Parashtesa *-t* mundëson qëndron për etiketues (ang. *tag*) dhe lejon që imazhi i krijuar në këtë rast të njihet si intellidota, ndërsa pika tregon që shtegu që duhet përdorur për të krijuar imazhin është ky ku gjendemi aktualisht.Pas ekzekutimit të asaj komande, marrim rezultatet si në vijim:

**

Dhe nëse duam që të shfaqim imazhet e krijuara deri më tani, përdorim komandën *docker images*. Pas krijimit të imazhit, ekzekutojmë atë me anë të komandës: *docker run -it intellidota* ku parashtesa ­*-it­* qëndron për ndërfaqen (ang. *interface*) dhe marrim rezultatet si më poshtë:



Pra, serveri është i ngritur në 0.0.0.0 me portin 1234, pra vizitimi i *localhost:1234* na shpien te aplikacioni jonë. Me këtë rezultat, edhe publikimi në çfarëdo platforme të madhe Cloud nuk do të na shfaq problem.

Për fund, le të shohim përmbajtjen e kontejnerit tonë.

Fillojmë me komandën *docker container ps –-all*. Kjo mundëson që të listohen të gjithë kontejnerët bashkë me numrin identifikues përkatës dhe rezultati duket si më poshtë:

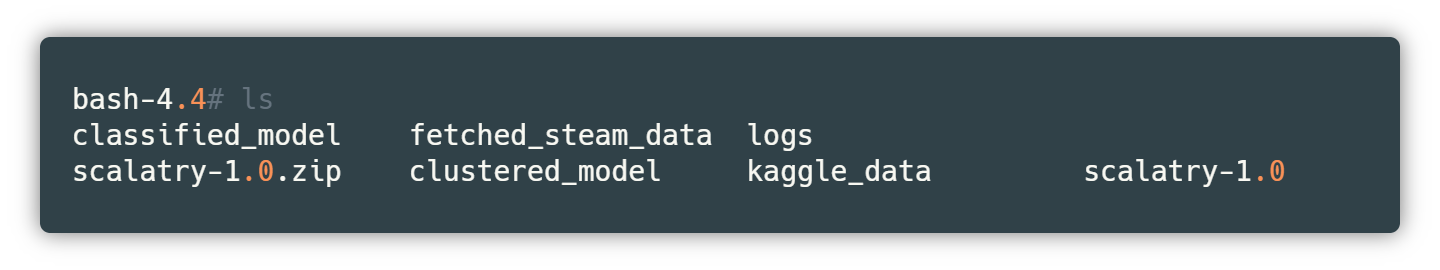


Që do të thotë se kontejneri jonë ka numrin identifikues c26ebae69ab5 dhe ka emrin intellidota. Me të pranuar këto informacione, ekzekutojmë komandën *docker exec -it c26ebae69ab5 bash* për të kaluar në përmbajtje të kontejnerit. Fjala *exec* qëndron për ekzekutues (ang. *ekzekutues*).

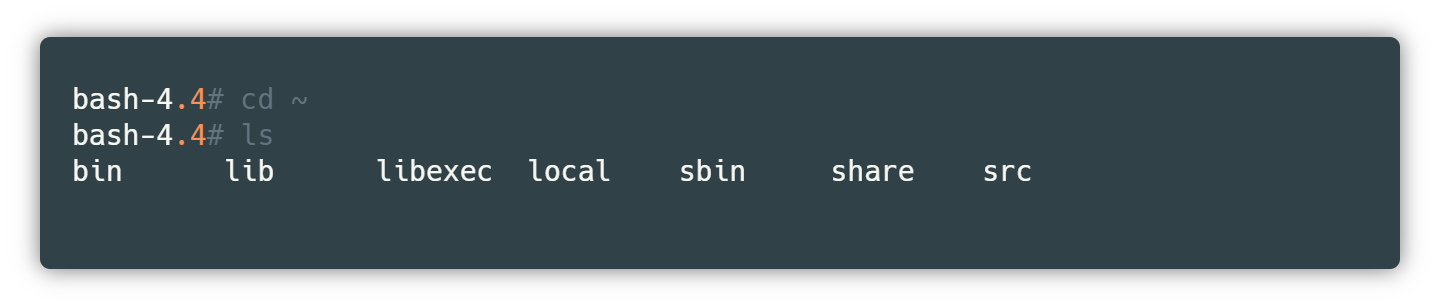
Tanimë jemi brenda kontejnerit tonë. Fillojmë me komandën PWD (Point Working Directory) për të parë shtegun në të cilin jemi aktualisht:



Shikojmë se çfarë ka brenda direktoriumit aktual si më poshtë:



Pra, kemi modelet e trajnuara, projektin e kompresuar dhe të dekompresuar (scalatry-1.0.zip dhe scalatry-1.0 respektivisht). Nëse kthehemi tek direktoriumi bazë, pra root, shohim se kemi:



Duke u bazuar në këtë rezultat, shohim se brenda imazhit aa8y/sbt:0.13.18 është i përdorur një sistem operativ Unix, nga se ne nuk kemi përdorur një imazh të tillë në mënyrë të drejtpërdrejtë.

Kjo është edhe një arsye se pse Docker është aktualisht një ndër veglat më të përdorura aktualisht, pra ofron fleksibilitet dhe është lehtë i përdorshëm dhe menaxhueshëm.

# Lista e thirrjeve fundore të mundshme

## Index

Emri: index

URL: localhost/

Parametrat:

Përshkrimi: Kthen përmbajtjen e faqes filluese, që është lista e pikave fundore të ofruara nga ne.

Shembull: localhost:9000/

Emri: getColumns

URL: localhost/getColumns

Parametrat: kind – lloji i datasetit [steam / Kaggle]

Përshkrimi: kthen një listë të kolonave të datasetit përkatës

Shembull: localhost:9000/getColumns?kind=steam

## getSample

|  |  |
| --- | --- |
| Emri | getSample |
| URL | localhost/getSample |
| Parametrat | kind – lloji i datasetit [steam / Kaggle]  percentage – përqindja e monstrës [0 deri 100] |
| Përshkrimi | kthen një monstër të datasetit përkatës, varësisht përqindjes |
| Shembull | localhost:9000/getSample?kind=steam&percentage=10 |

## getStages

|  |  |
| --- | --- |
| Emri | getStages |
| URL | localhost/getStages |
| Parametrat | kind – lloji i datasetit [steam / Kaggle] |
| Përshkrimi | kthen një varg të fazavenëpër të cilët kanë kaluar të dhënat |
| Shembull | localhost:9000/getStages?kind=kaggle |

Tabela 4: pika fundore getStages

## getCorrelationMatrix

|  |  |
| --- | --- |
| Emri | getCorrelationMatrix |
| URL | localhost/getCorrelationMatrix |
| Parametrat | kind – lloji i datasetit [steam / Kaggle] |
| Përshkrimi | kthen një varg numrash që tregojnë ndërlidhjen mes kolonave të *datasetit* përkatës |
| Shembull | localhost:9000/getCorrelationMatrix?kind=kaggle |

Tabela 5: pika fundore getCorrelationMatrix

## getGroupAndCount

|  |  |
| --- | --- |
| Emri | getGroupAndCount |
| URL | localhost/getGroupAndCount |
| Parametrat | kind – lloji i datasetit [steam / Kaggle]  attribute - grupimi ndodhë sipas këtij atributi  partitions – numri i ndarjeve të të dhënave numerike |
| Përshkrimi | kthen një varg grupesh dhe intervalin përkatës që të dhënat i përkasin |
| Shembull | localhost:9000/getGroupAndCount?attribute=xp\_per\_min&partitions=3 |

Tabela 6: pika fundore getGroupAndCount

## getStages

|  |  |
| --- | --- |
| Emri | getStats |
| URL | localhost/getStats |
| Parametrat | kind – lloji i datasetit [steam / Kaggle] |
| Përshkrimi | kthen sasinë e rreshtave dhe kolonave të *datasetit* përkatës |
| Shembull | localhost:9000/getStats?kind=steam |

Tabela 7: pika fundore getStats

## getSchema

|  |  |
| --- | --- |
| Emri | getSchema |
| URL | localhost/getSchema |
| Parametrat | kind - lloji i datasetit [steam / Kaggle] |
| Përshkrimi | kthen kolonat dhe tipin përkatës të *datasetit* të caktuar |
| Shembull | localhost:9000/getSchema?kind=steam |

Tabela 8: pika fundore getSchema

## getDoubleGroup

|  |  |
| --- | --- |
| Emri | getDoubleGroup |
| URL | localhost/getDoubleGroup |
| Parametrat | kind – lloji i datasetit [steam / Kaggle]  col1 - kolona e parë  col2 - kolona e dytë |
| Përshkrimi | kthen grupimin dhe njehsimin sipas kolonave përkatëse |
| Shembull | localhost:9000/getDoubleGroup?kind=steam&col1=leaver\_status& col2=radiant\_win |

Tabela 9: pika fundore getDoubleGroup

## postPredict

|  |  |
| --- | --- |
| Emri | postPredict |
| URL | localhost/postPredict |
| Parametrat | gold\_per\_min – paratë për minutë - derivuar  level – nivelet - derivuar  leaver\_status – statusi i përfundimit të ndeshjes - derivuar  xp\_per\_min – eksperienca për minutë - derivuar  radiant\_score - pikët e të bardhëve - drejtpërdrejtë  gold\_spent – paratë e harxhuara - derivuar  deaths – vdekjet gjatë lojës - derivuar  denies – mohimet e vrasjeve - derivuar  hero\_damage - dëmtimet ndaj kundërshtarëve - derivuar  tower\_damage- dëmtimet ndaj kullave - derivuar  last\_hits - goditja përfundimtare për vrasje - derivuar  hero\_healing - ndihmesa ndaj ekipit - derivuar  duration - gjatësia e lojës - drejtpërdrejtë |
| Përshkrimi | Kthen dy atribute të reja në data set që tregojnë se a kanë fituar të bardhët apo jo dhe saktësisht përqindjen e përgjigjes së dhënë |
| Shembull | localhost:9000/postPredict?  gold\_per\_min=2585&level=139&leaver\_status=0&xp\_per\_min=3658  &radiant\_score=34&gold\_spent=89220&deaths=65&denies=21  &hero\_damage=170629&tower\_damage=2732&last\_hits=901  &hero\_healing=10700&duration=2549 |

Tabela 10: pika fundore postPredict

# Fjalët e fundit

Meqenëse projekti në përgjithësi ka tentuar të përmbledh sa më shumë detaje, funksione dhe karakteristika, normalisht që është dashur punë dhe kujdes më i madh.

Dihet që projekti është i ndarë në dy pjesë, pjesa *frontend* dhe pjesa *backend* nga edhe kanë ardhur vështirësitë më të shumta, pasi vizualizimi dhe përshkrimi i funksioneve kryesore është dashur të bëhet në koordinim të plotë me njëri tjetrin. Gjithsesi, eksperienca e krijimit të një platforme të tillë në të cilën koordinohen pjesa *frontend* dhe pjesa *backend* ka qenë e veçantë dhe do të hyjë në përdorim shumë në të ardhmen, pasi në ditët e sotme rrallë herë punohet me *frontend* të integruar në skeletin (ang. *framework)* përkatës.

Gjithsesi, rikthimi në gjuhën programuese *SQL* ishte sfidues sidomos kur shkruhet në një mënyrë tjetër siç ka qenë Spark Dataset API, nga i cili fjalët kyqe të gjuhës SQL janë funksione në Spark Dataset API.

Po ashtu, kënaqësia më e veçantë e këtij projekti ka qenë pa dyshim vënia e duarve në gjuhën programuese Scala bashkë me libraritë Spark dhe për mësim të makinës, ku përveç funksioneve të realizuara më sipër, kemi tentuar mënjanimin e unazave të shpeshta dhe kodit përsëritës në një kod më konciz dhe më të lexueshëm. Kjo është realizuar sidomos me *llambda expressions* dhe programin funksional ku ka qenë e mundur, pasi Scala shquhet për këto veçori.

Elemente tjera me rëndësi edhe për projektin edhe për të ardhmen kanë qenë përdorimi serioz i strukturës JSON, pra manipulimi i atributeve nëpërmjet Scalas, përdorimi i Gson si librari ndërmjetësuese e JSON dhe Scalas dhe marrja e të dhënave nëpërmjet vartësive të Scalas, si *lihaoyi* me anë të së cilës kemi realizuar kërkesat. Ja vlen të ceket edhe njëherë aplikacioni *ngrok* me anë të së cilit kemi mundur të publikojmë pikat fundore tonat ashtu që të mund të realizohet pjesa e dytë, pra ajo *frontend*.

Ky projekt ka qenë një shtytje e madhe për ne si në aftësi programuese, ashtu edhe në bashkëpunim. Këto njohuri të fituara gjithsesi që do të ripërdoren në projekte tjera reale e pse jo edhe t’i përhapim në rrethin tonë. [16]

Në fund, një falënderim i veçantë i takon familjes dhe kompanisë aktuale në të cilën jam aktualisht, Prime [17], posaçërisht mentorit Endrit Bytyqi për ndihmesën dhe mbështetjen e vazhdueshme përgjatë këtyre muajve.

# References

|  |  |
| --- | --- |
| [1] | P. Norvig dhe S. J. Russell, Artificial Intelligence: A Modern Approach, Harlow, United Kingdom: Prentice Hall, 2009. |
| [2] | P. Norvig dhe S. J. Russell, «Artificial Intelligence: A Modern Approach,» në *Artificial Intelligence: A Modern Approach*, Harlow, United Kingdom, Prentice Hall, 2019, p. 37. |
| [3] | P. Norvig dhe S. J. Russell, «Artificial Intelligence: A Modern Approach,» në *Artificial Intelligence: A Modern Approach*, Harlow, United Kingdom, Prentice Hall, 2009, p. 2. |
| [4] | P. R. S. J. Norvig, «Artificial Intelligence: A Modern Approach,» në *Artificial Intelligence: A Modern Approach*, Harlow, United Kingdom, Prentice Hall, 2009, p. 695. |
| [5] | J. R. Quinlan, «Wikipedia,» University of Sydney, 1975. [Në linjë]. Available: https://en.wikipedia.org/wiki/Decision\_tree\_learning. [Qasja 6 October 2019]. |
| [6] | Bayes, «Wikipedia,» [Në linjë]. Available: https://en.wikipedia.org/wiki/Naive\_Bayes\_classifier. [Qasja 6 October 2019]. |
| [7] | Ho dhe . L. Breiman, «Wikipedia,» 1995. [Në linjë]. Available: https://en.wikipedia.org/wiki/Random\_forest. [Qasja 11 September 2019]. |
| [8] | Valve, «Steam,» 8 October 2016. [Në linjë]. Available: https://steamcommunity.com/dev. [Qasja 4 November 2019]. |
| [9] | neilatsyracuse, «Reddit,» Reddit, 2018. [Në linjë]. Available: https://www.reddit.com/r/DotA2/comments/7yzlu4/dota2\_api/. [Qasja 2 October 2019]. |
| [10] | S. Steam Developers, «Steam,» 12 September 2003. [Në linjë]. Available: https://steamcommunity.com/dev. [Qasja 5 October 2019]. |
| [11] | «w3school,» w3, 2019. [Në linjë]. Available: https://www.w3schools.com/js/js\_JSON\_intro.asp. [Qasja 1 October 2019]. |
| [12] | «Github,» Google, 22 May 2008. [Në linjë]. Available: https://github.com/google/gson. [Qasja 11 October 2019]. |
| [13] | Oracle, «Wikipedia,» OpenJDK, 8 May 2007. [Në linjë]. Available: https://en.wikipedia.org/wiki/Java\_virtual\_machine. [Qasja 28 September 2019]. |
| [14] | «Jetbrains,» JetBrains, 1 January 2001. [Në linjë]. Available: https://www.jetbrains.com/help/idea/2016.1/intellij-idea-help.pdf. [Qasja 1 October 2019]. |
| [15] | M. Zaharia, «Apache Spark,» 9 September 2019. [Në linjë]. Available: https://spark.apache.org/docs/latest/ml-features. [Qasja 16 October 2019]. |
| [16] | P. Bugnion, P. R. Nicolas dhe A. Kozlov, Scala: Applied Machine Learning, Packt Publishing, 2017. |
| [17] | Prime, «Prime,» Prime, November 2016. [Në linjë]. Available: https://goprime.io/. [Qasja June 2019]. |
| [18] | Databricks, The Data Scientist's Guide to Apache Spark™, Berkley: Apache Spark, 2019. |