

Universiteti i Prishtinës “Hasan Prishtina”

Tema: Kllasterimi në video-lojën Dota 2

Mentorë: Prof. Dr. Ang. Lule Ahmedi

Studenti: Nora Ibrahimi

01/01/2019

Abstrakti

Falenderime

Parathenie

# Hyrja

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:

* Aplikimi i algoritmit
* Marrja e te dhënave
* Organizimi ne Flutter
* Paraqitja?

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 – Kllasterimi si mësim pa 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 pa mbikëqyrje, teknikë kjo që do të përshkruhet më poshtë saktësisht me algoritmet përkatëse!

Mundemi edhe pak me shtu ktu per algoritmin pa mbikqyrje ...

## Algoritmet e mësimit pa mbikëqyrje

Kllasterimi është një teknikë në inteligjencën artificiale që përfshinë grupimin e të dhënave, pra ne mund të përdorim kllasterimin për të klasifikuar të dhënat në grupe të caktuara. Në teori, të dhënat në grup të njëjtë duhen të kenë karakteristika dhe cilësi të njëjta, për derisa të dhënat në grupe të ndryshme duhet të kenë atribute dalluese.

Kllasterimi përndryshe është një teknikë e të mësuarit pa mbikëqyrje dhe përdoret shumë për statistika dhe analiza të të dhënave në fusha të ndryshme.

Në aplikacionin tonë, si teknikë kllasterimi ne kemi përdorur K-Means, por do të shfaqim edhe teknikën tjetër shumë të përdorur Mean-Shiftnga e cila nuk ka nevojë fare të caktohet numri i pikave qendrore të quajtura ndryshe centroide*.*

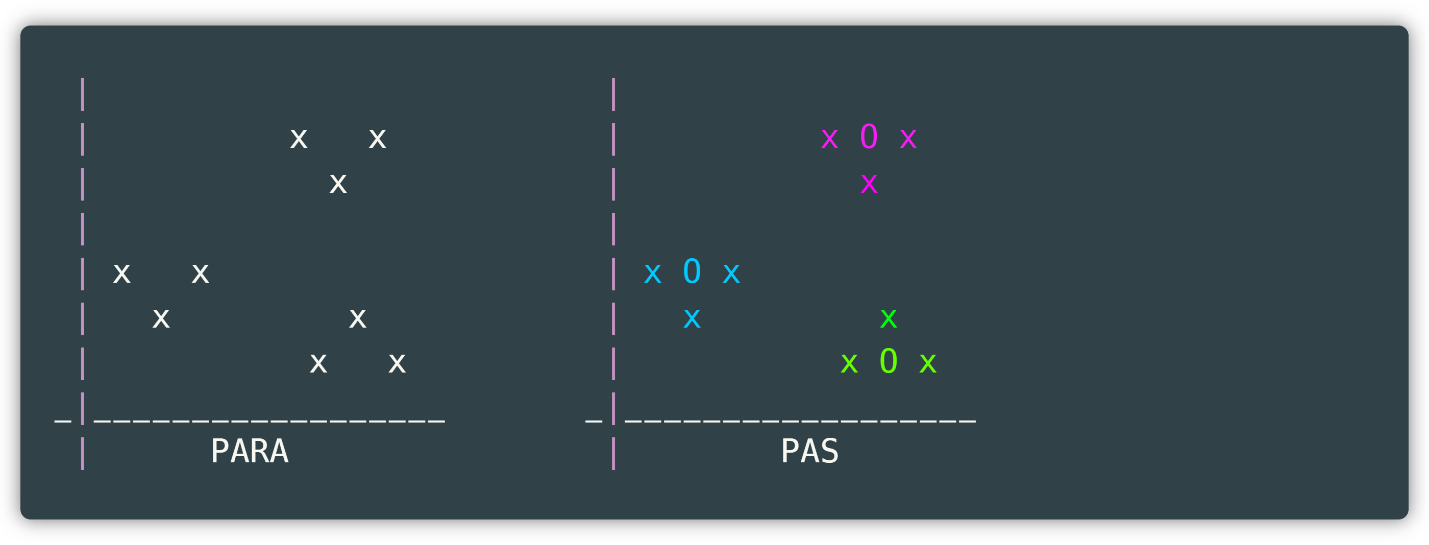
### K-Means

Është me shumë mundësi algoritmi më i përhapur dhe përdorur për kllasterim. Kjo për arsye se përdoret dhe kuptohet lehtë. Po përshkruajmë në hapa algoritmin për të sqaruar mënyrën se si funksionin.

* Fillimisht, zgjedhim një numër grupesh dhe në mënyrë të rastësishme vendosim pikat qendrore të tyre. Për të gjetur numrin që duam të përdorim, është praktikë e mirë që tu hedhim një sy të dhënave dhe të shohim ‘me sy të lirë’ nëse kanë ngjashmëri. Pikat qendrore janë po ashtu vektorë si çdo pikë tjetër aktuale.
* Secila pikë e të dhënave klasifikohet duket llogaritur distancën në mes pikës aktuale dhe secilit grup pikash qendrore, e pika e të dhënave klasifikohet në atë grup, distanca e së cilës është më e vogël ndër të gjitha grupet e pikave qendrore.
* Duke u bazuar në këto pika klasifikuese, ri-llogarisim pikat qendrore duke marrë vlerën mesatare të të gjitha grupeve qendrore.
* Ri përsërisim këto hapa për një numër të caktuar apo deri sa qendrat e grupeve nuk ndryshojnë shumë. Një mundësi tjetër është inicializimi në mënyrë të rastësishme i qendrave grupore disa herë dhe zgjedhim rezultatin që ka performuar më së miri.

K-Means ka avantazhin kryesorë në shpejtësi, pasi krejt çka bën është llogaritja e distancave ndërmjet pikave apo qendrave grupore; kalkulime shumë të vogla. Për këtë arsye ka kompleksitet linear O(n).

Një shembull i algoritmit K-Means duket kështu



ku grafiku i parë paraqet të dhënat para kllasterimit, pra të dhëna të rëndomta pa ndonjë grupim, ndërsa grafiku i dytë paraqet të dhënat e grupuara në bazë të ngjyrës, ku shihen 3 pika qendrore të shënuara me O(nën supozimin se kemi zgjedhur 3 pika qendrore, pra k = 3). E nëse vjen një e dhënë e re, ajo grupohet në atë kllaster, distanca ndaj të cilit është më e afërt.

Në anën tjetër, ka disa mangësi, duke filluar prej asaj se duhet caktuar në mënyrë manuale numrin e grupeve të pikave qendrore (variabla k). Kjo nuk është diçka e mirë sepse ne duam të nxjerrim statistika të shëndosha nga data seti dhe do të ishte mirë po të merrej algoritmi me këtë pjesë.

Përveç kësaj, siç u cek edhe më lartë, K-Means i cakton qendrat grupore në mënyrë të rastësishme, për këtë arsye rezulton ndryshe nga secili ambient që ekzekutohet algoritmi. Pra, rezultatet nuk janë të ri përdorshëm dhe kemi mungesë qëndrueshmërie. Metodat tjera janë më të qëndrueshme.

### Mean-Shift

Shkruj per ket algoritem pak

### Bisecting K-Means (E kemi përdor ne)

Ideja prapa këtij algoritmi është pothuajse e njëjtë me atë të algoritmit K-Means, por diferencon në disa detaje të caktuara.

Ndryshe nga algoritmi K-Means, ky algoritëm nuk zgjedh pika të rastësishme për të matur më pas distancën, por ndan mostrën tonë përgjysmë duke vazhduar me pjesën me marzhë gabimi më të madhe deri sa të formohen K grupime.

Po përshkruajmë më poshtë hapat e saktë që ndërmerr ky algoritëm:

* Mbledh të dhënat në një grup të vetëm, që do të thotë se të gjitha të dhënat e data setit gjenden në një kllaster të vetëm.
* Aplikohet K-Means me vlerë 2, pra kllasteri jonë ndahet në dy pjesë relativisht të barabarta, pjesë këto të caktuara nga vetë algoritmi.
* Kalkulohet marzha e gabimit sipas metrikës RMSE dhe zgjidhet data seti me marzhë të gabimit më të madhe për tu ndarë përsëri në grupime më të vogla.
* Përsëritet hapi paraprak deri sa numri i grupimeve të jetë i barabartë me K.

Një shembull se si duket kjo grafikisht paraqitet më poshtë



Ndër plot algoritme për kllasterim, ne kemi zgjedhur, siç u cek edhe më lartë, algoritmin Bisecting K-Means. Ndër arsyet kryesore se përse kemi e kemi zgjedhur këtë algoritëm mbi të tjerët është se ka saktësi të lartë, mënyra e funksionimit i përshtatet data setit tonë dhe se fenomeni i rastësisë është më pak i pranishëm, duke mos anashkaluar faktin se është ndër algoritmet më të rinj të prezantuar në Spark.

# Hyrje në projekt

Me të përshkruar inteligjencën artificiale dhe algoritmet kryesore te klasifikimit si përfaqësues të të 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.
* Vazhdo me tjera

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 (mbushe hala se vyn)

*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* (**Error! Reference source not found.**)*.*



Ndërsa në këndvështrim të lartë, ekipi organizohet në formacion të tillë (**Error! Reference source not found.**):



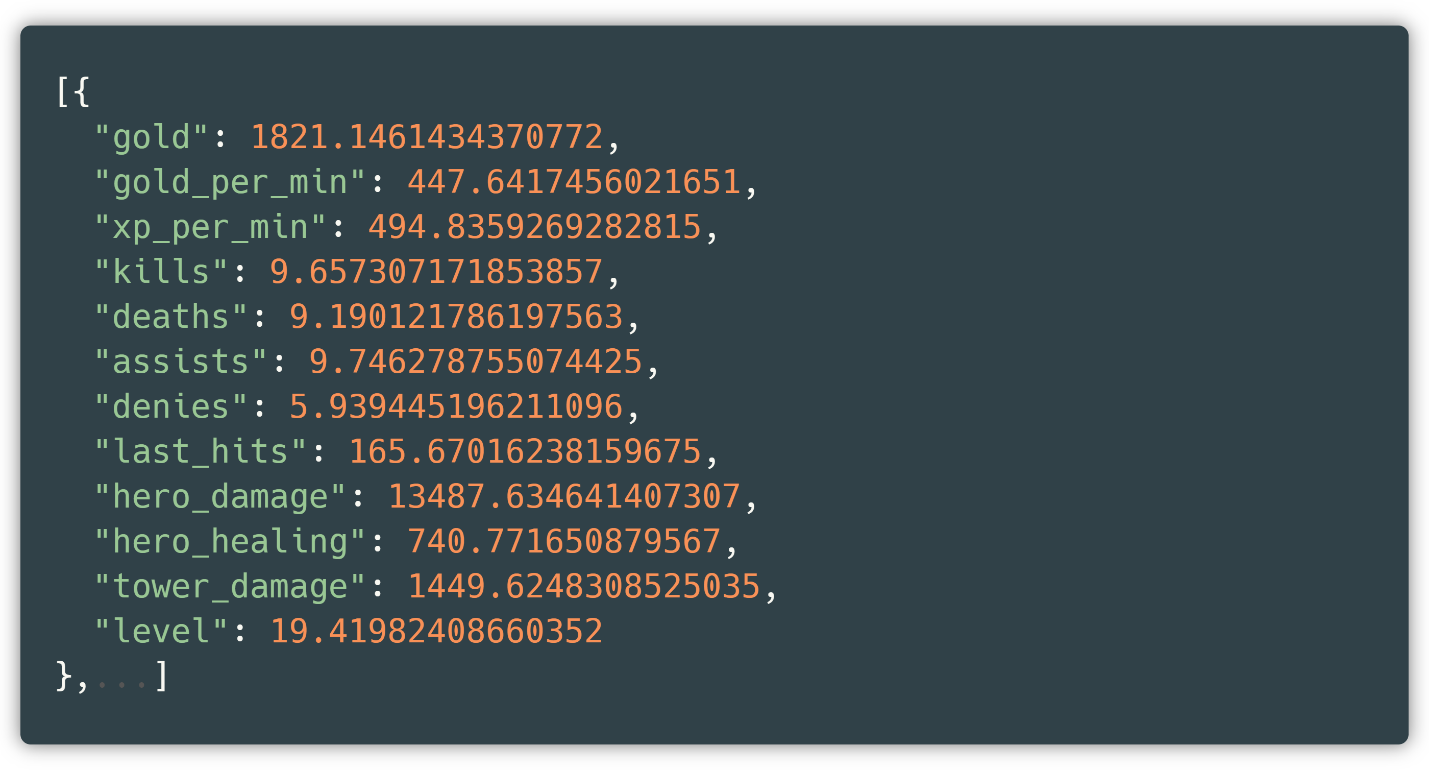
Secili lojtarë, pavarësisht rolit ka qindra statistika, por ndër më të thepisurat dhe më të rëndësishmet janë *leaver\_status, gold\_per\_min, leaver\_status, xp\_per\_min, deaths, tower\_damage* etj. Lista e plotë mund të gjendet në kapitullin e mëposhtëm në të cilin realizohet kllasterimi, pra postCluster.

## Para-procesimi i të dhënave

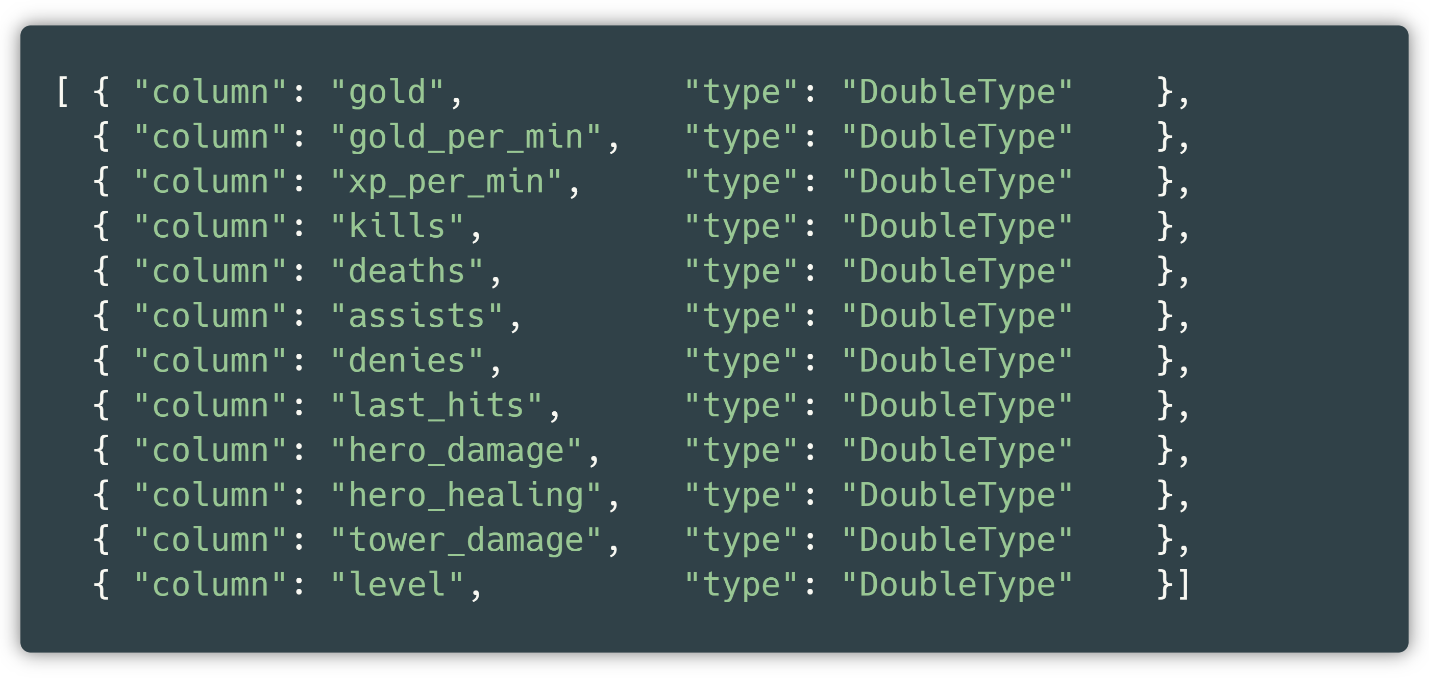
Si burim i të dhënave kemi përdorur Kaggle. Kaggle është një ... Data seti përbëhet nga dhjetëra kolona dhe rreth 500.000 rreshta. Megjithëse procesi i kllasterimit nuk kërkon shumë fuqi llogaritëse, jemi munduar të marrim kolonat më përshkruese dhe më të shëndosha pavarësisht sasisë së rreshtave dhe kolonave.

Fillojmë me analizimin e data setit nga i cili fillimisht heqim kolonën *match\_id dhe account\_id* për arsye se nuk kanë peshë në kllasterim, pra janë të dhëna ‘jo të mençura’. Shohim se kolona *hero\_id* i përket një heroi përkatës, dhe secili hero në lojë shquhet me atributet përkatëse, pra kemi heronj që shkaktojnë dëme, që shërojnë, që shtyjnë kulla e të tjera, të dhëna këto të nxjerra nga pamja e parë e data setit.

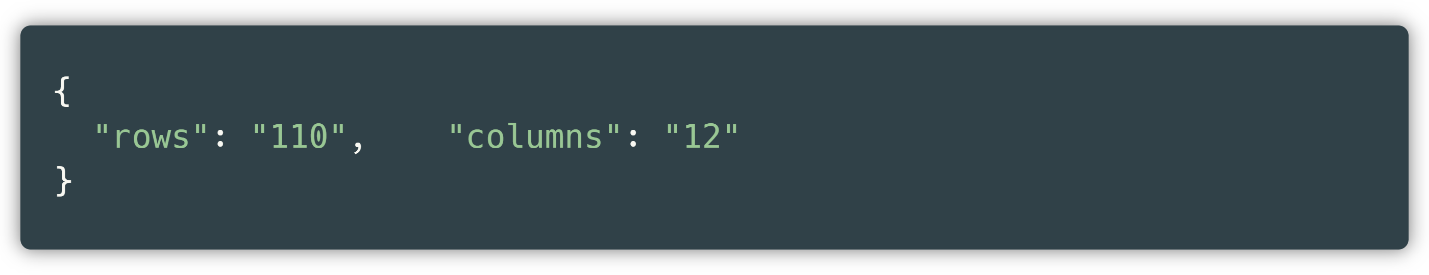
Vazhdojmë me kolonën *player\_slot* të cilën nuk e marrim parasysh po ashtu, pasi shërben vetëm për të treguar inventarin e heroit. 9 kolonat e ardhshme janë shumë përshkruese dhe që të gjitha do të jenë pjesë e data setit tonë të pastër, ato kolona janë *gold, gold\_per\_min, xp\_per\_min, kills, deaths, assists, denies, last\_hits, hero\_damage, hero\_healing, tower\_damage* dhe *level*. Ndërsa kolonat tjerë përshkruajnë veglat e lojtarit që nuk janë përshkruese ngase nuk ndryshojnë varësisht rolit. Gjithsesi, tani më kemi një data set të llojit



Shohim tani skemën e data setit, pasi duhet ditur tipet e të dhënave në mënyrë që të ndryshojmë diçka apo të vazhdojmë më tutje.



Shohim se të gjitha tipet e të dhënave janë të tipit *Double* që i bie që mund të aplikohen metrika të ndryshme numërore dhe kjo është ajo që na nevojitet neve. Po ashtu, shohim statistikat e data setit tonë:



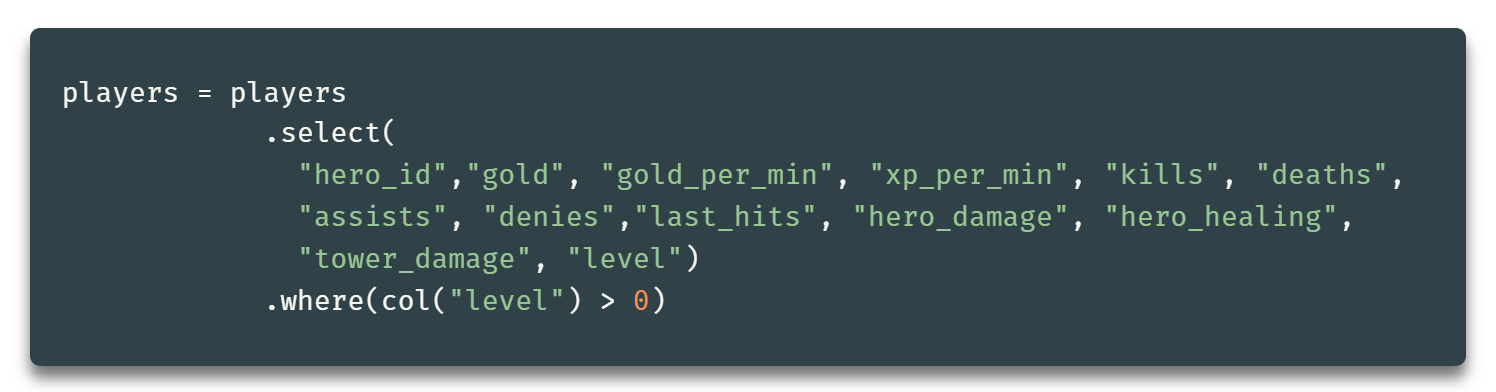
Pra, data seti jonë i përpunuar përmban 110 rreshta dhe 12 kolona. Kemi vetëm 110 rreshta për arsye se ekzistojnë 110 heronj, pasi kemi grupuar data setin në bazë të heronjve, ndërsa 12 kolona janë atributet e përzgjedhura nga ne.

## Para-procesimi në Scala

Së pari ngarkojmë të dhënat e shkarkuara nga Kaggle nëpërmjet Spark duke përdorur komandat:

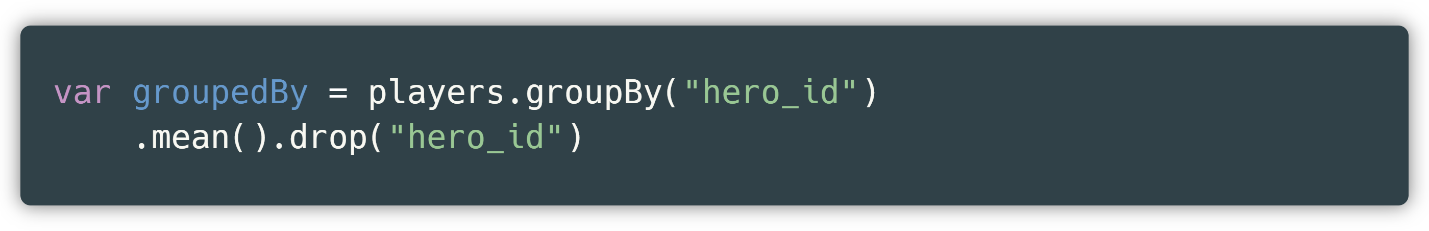


Pra, *players* aktualisht është një data set me 500.000 rreshta dhe rreth 30 kolona, ndërsa *raw\_kaggle\_data* është emri i variablës së mjedisit që e kemi emëruar ne për shkaqe të lehtësimit. Së pari zgjedhim kolonat e caktuara dhe filtrojmë ato, niveli (ang. *level*) i së cilave është më i madh se zero.



Pastaj aplikojmë ato në data setin *players* të ngarkuar më lartë. Tani meqë *players* është një element i pa ndryshueshëm (ang. *immutable*), ne mbishkruajmë data setin *players* me data setin e filtruar. Është një ide e mirë të filtrohet data seti në nivel më të ma pasi një nivel i tillë në lojë nuk ekziston dhe është mbushur hapësira si e korruptuar nga zhvilluesit e video lojës.

Si hap i ardhshëm është grupimi dhe mesatarja e secilës kolonë, e cila është realizuar si në vijim



e më pas vetëm kemi ruajtur lokalisht data setin në një lokacion të përshtatshëm për përdorim të më vonshëm. Funksionet e paraqitura më lartë, pra *mean* dhe *drop* i ofrohen koleksionit të data setit pas përdorimit të *group by*, kjo pasi mesatarja është vlera me e denjë për të përfaqësuar kolonën përkatëse, në rastin tone *mean* gjen mesataren aritmetike për secilën kolonë numerike, këtij funksioni i shtohet edhe largimi i *hero\_id* pasi pas grupimit, na krijohet edhe një *avg(hero\_id)*, qe është asgjë më shumë se sa duplikate e paraprakes

Pra tash kemi data setin gati për të aplikuar metrika të tjera ose për të aplikuar algoritmin kryesorë, që është edhe qëllimi i këtij aplikacioni.

## Algoritmi (ma titullin e mire)

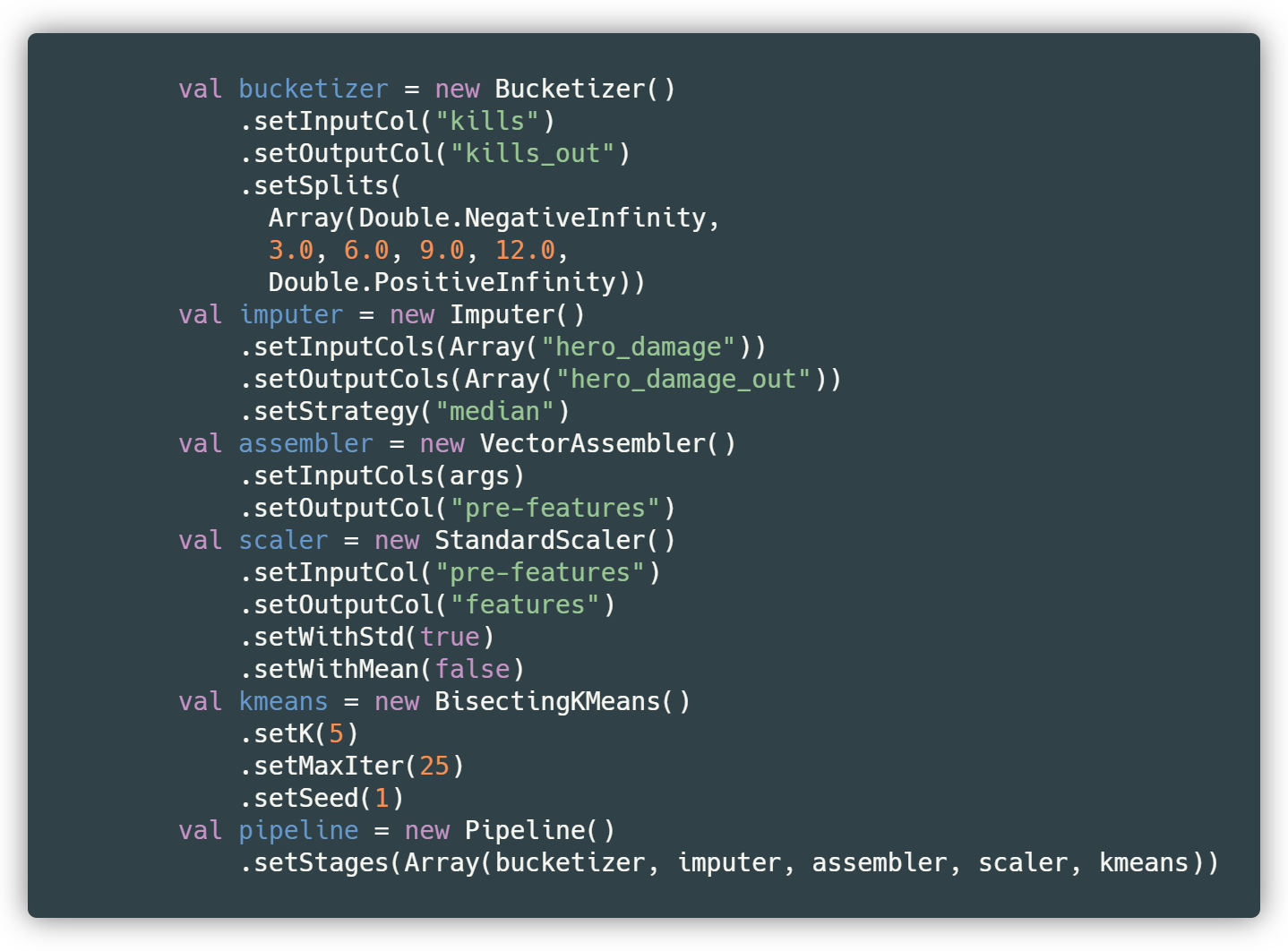
Ri ngarkojmë data setin tonë të pastër nëpërmjet librarisë Spark në mënyrë që të aplikojmë kllasterimin. Rikujtojmë se ky data set i ri përmban 110 rreshta dhe rreth 13 kolona. Së pari nëpërmjet këtyre rreshtave ngarkojmë data setin e pastruar:



dhe ua heqim emrat e pa përshtatshëm kolonave përkatëse. Me emra të pa përshtatshëm duam të themi se nëse aplikojmë metrika si grupimin (ang. *group by*) më shumë, apo me mesatare siç e kemi në rastin tonë, atëherë kolonave ju shtohet parashtesa e funksionit, në rastin tonë, secila kolonë merr emrin *avg(...)*. E me anë të funksionit *RemoveBadNaming* e krijuar nga ne, funksioni do të merr emrin që ka pasur para funksionit. Këtë metrikë ua aplikojmë të gjitha kolonave. Ky funksion duket si në vijim:



ku *groupedBy* paraqet data setin e filtruar dhe tanimë edhe të riemëruar. Pas këtij hapi, ne konstruktojmë fazat nëpër të cilat do të kalojë algoritmi jonë. Këto faza duken si në vijim:



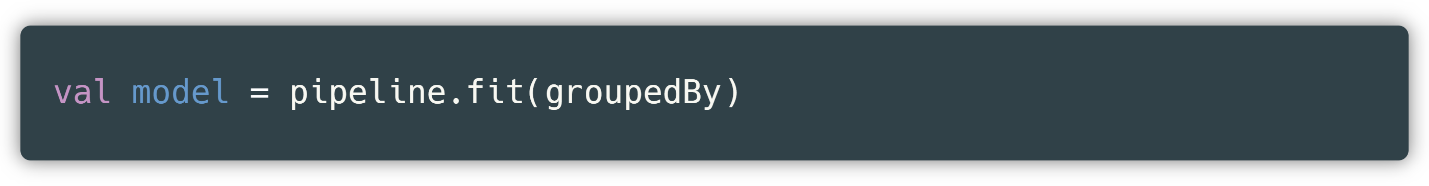
Nga e cila mund të shohim se përdorim gjashtë klasa kryesore, ku:

* Klasa e parë, pra *Bucketizer* mundëson grupimin e të dhënave nga kolona *kills* në kolonën dalëse *kills\_out*. Si metodë shihet që përdoret edhe *setSplits* nga e cila mundësohet që kolona të grupohet varësisht ndarjeve që kemi specifikuar ne. Ky grupim është një lloj klasifikuesi jo binarë që ndihmon në kllasterimin e mëtutjeshëm të të dhënave, pasi kolona *kills* vetëm se ka vlera nga zero deri në mesatarisht 50. Saktësisht, ndarja i takon grupimeve të më poshtëm:

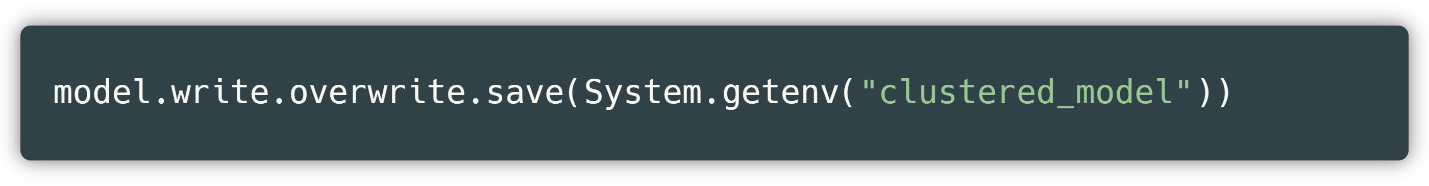
1. Prej intervalit infinitiv negativ deri në 3 të dhënat grupohen në vlerën 0.
2. Prej intervalit 3 deri në 6 të dhënat grupohen në vlerën 1.
3. Prej intervalit 6 deri në 9 të dhënat grupohen në vlerën 2.
4. Prej intervalit 9 deri në 12 të dhënat grupohen në vlerën 3.
5. Prej intervalit 12 e deri në infinitiv pozitiv të dhënat grupohen në vlerën 4.

* Klasa e dytë e quajtur *Imputer* përdoret për të mbushur vlerat e zbrazëta në data set, ku në rastin tonë përdoret strategjia *median* në atributin *hero\_damage*. Në rastin tonë konkret nuk gjejmë ndonjë aplikim të kësaj por është përdorur si demonstrim.
* Klasa e tretë *VectorAssembler* që mundëson grumbullimin e atributeve në një atribut të vetëm të quajtur *features* në rastin tonë. Ky funksion nevojitet në secilin algoritëm të mençur për arsye se mënyra e funksionimit të tyre është e tillë që algoritmi kërkon një kolonë hyrëse dhe jep një kolonë dalëse. Funksioni ka pra dy metoda, *setInputCols* që kërkon një varg me elementet që dëshirojmë të grumbullojmë në një atribut dhe metodën *setOutputCol* nga e cila mundësohet kolona dalëse.
* Klasa e katërt është *StandardScaler* dhe përdorur për të shkallëzuar të dhënat. Ky lloj shkallëzuesi, ndër të shumtë të llojit të tij, shërben për të konvertuar secilën vlere në intervalin -1 deri në 1. Si kolonë hyrëse kemi marrë *pre-features* të përgatitura nga hapi paraprak dhe si kolonë dalëse kemi *features*.
* Klasa e pestë quhet *Bisecting* *Kmeans* dhe është funksioni kryesorë me anë të së cilit aplikohet algoritmi i kllasterimit. Ndër shumë metoda tjerë me vlera të para zgjedhura, ne po aplikojmë dy parametra të ri në metodën *setK*. Kjo metodë cakton numrin e pikave qendrore të kllasterave që në rastin tonë është 5. Krahas kësaj, kemi edhe metodën *setMaxIter* që cakton numrin e unazave që në rastin tonë është 25, konkretisht kjo metodë provon 25 kombinime të mundshme të algoritmit dhe zgjedh rezultatin më të mirë.
* Klasa e gjashtë quhet *Pipeline* dhe mundëson që të gjitha klasat ndihmëse para se të përdoret algoritmi të grupohen në faza dhe të ekzekutohen një pas një në mënyre automatike. Kjo teknikë lehtëson përdorimin e algoritmeve kryesore si dhe bën kodin më konciz, më të lexueshëm dhe më elastik. Nëse nuk do të kishim këtë teknikë, ne do të duhej të aplikonim secilën klasë të cekur më sipër në data set e kjo është jo praktike dhe redudante.

Me të aplikuar këto algoritme në data setin tonë me anë të funksionit



Na mbetet vetëm të ruajmë modelin lokalisht



Ne ruajmë modelin lokalisht për arsye se është më logjike dhe më praktike që të ruhet modeli i kllasteruar, në mënyrë që nëse duam të nxjerrim statistika apo të aplikojmë funksione në të, të mos trajnohet modeli përsëri pasi kërkon kohë dhe fuqi kompjuterike përpunuese. Kjo po ashtu na mundëson neve që të kryejmë kllasterime në kohë reale, që do të thotë mundemi të kllasterojmë çfarëdo rreshti të ri në kohë shumë të shkurtër.

## Publikimi i imazhit Docker në Google Cloud Platform

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.

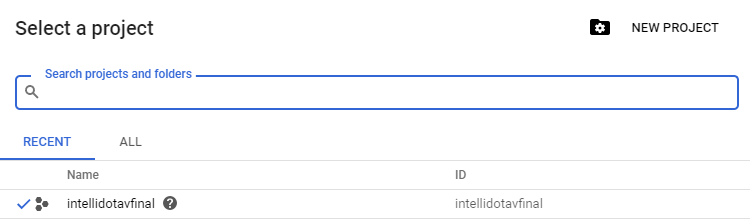
Krahas Docker, kemi edhe Google Cloud Platform, që paraqet një aset të fuqishëm të ofruar nga Google për të kryer shumë operacione, duke filluar nga të mësuarit të makinës, një depo për dosje të ndryshme, vegla për DevOps si K8 etj.

Ajo se çfarë ne kemi bërë me Google Cloud Platform është hostimi i API të ofruara nga ne. Përse na është nevojitur diçka e tillë? Për arsye që aplikacioni të mos varet nga ekzekutimi lokal i aplikacionit, pra thirrjet e pjesës frontend të kryhen pavarësisht se a po ekzekutohet aplikacioni lokalisht, pasi shërbimet e aplikacionit që është i ndezur nga një makinë lokale humben nëse ajo fiket. Me të cekur të gjitha këto detaje, kalojmë te pjesa e realizimit të kësaj procedure. Nga pjesa IntelliDota Classification, vetëm se kemi krijuar një Docker imazh, verifikojmë se a ekziston një i tillë:



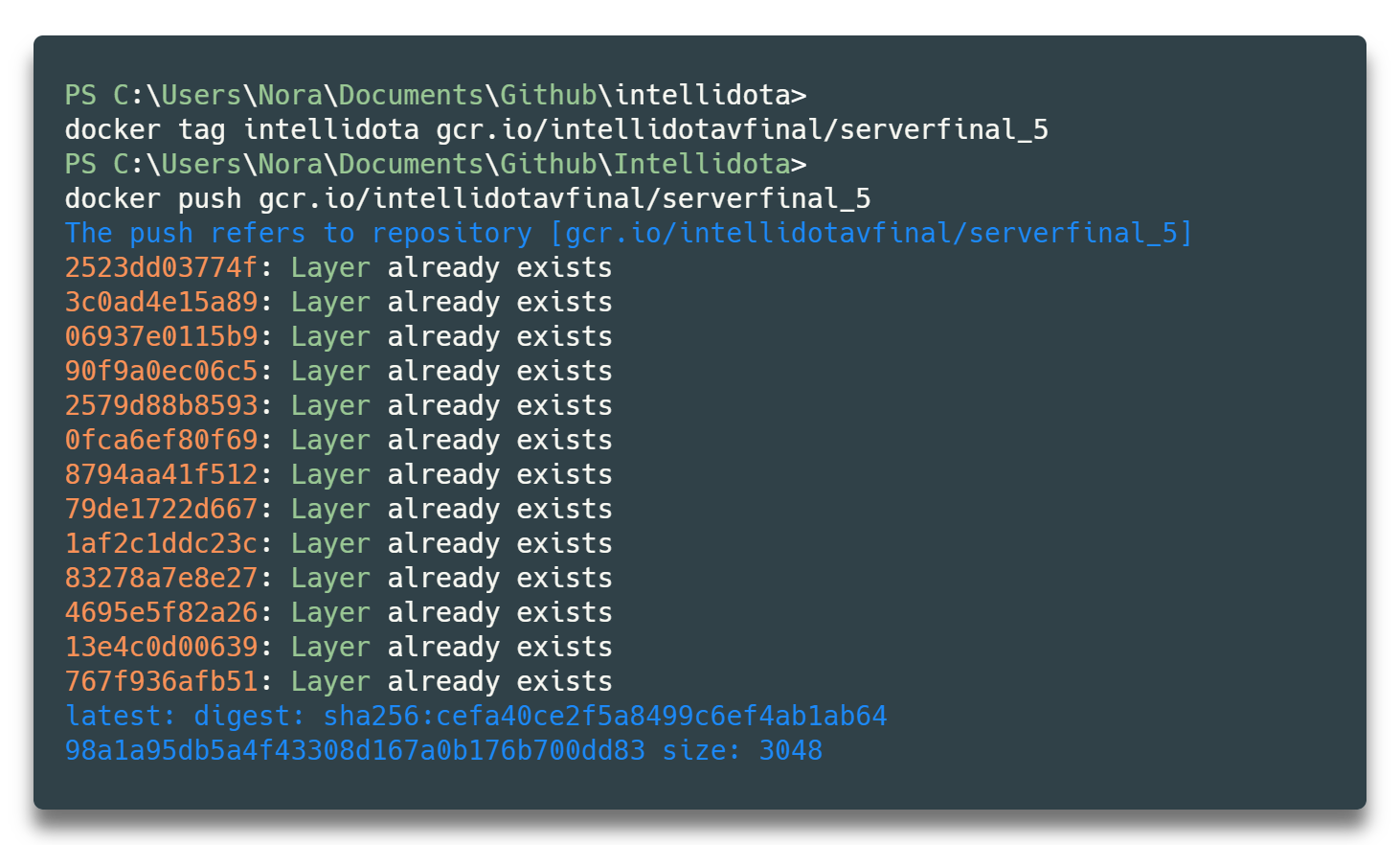
Pra shohim se kemi imazhin e quajtur intellidota me numër identifikues 5534d846e77e me madhësi rreth 525 MB. Kjo është e tëra që na nevojitet për ta kaluar në Google Cloud.

Kalojmë te pjesa e Google Cloud Platform, ku hapi i parë që duhet bërë është të krijojmë një projekt, në rastin tonë, ne vetëm se kemi krijuar diçka të tillë të emëruar *intellidotavfinal*. Krijimi i një projekti lehtë mund të realizohet nëpërmjet ndërfaqes kryesore si në vijim, pra tek butoni *New Project*. Shihet se ne kemi zgjedhur projektin *intellidotavfinal*, prandaj vazhdojmë tutje.



Me të krijuar projektin, hapin navigatorin dhe zgjedhim opsionin Cloud Run. Në esencë, Cloud Run është një platformë që lejon manipulimin me kontejnerë, në rastin tonë me imazhet Docker.

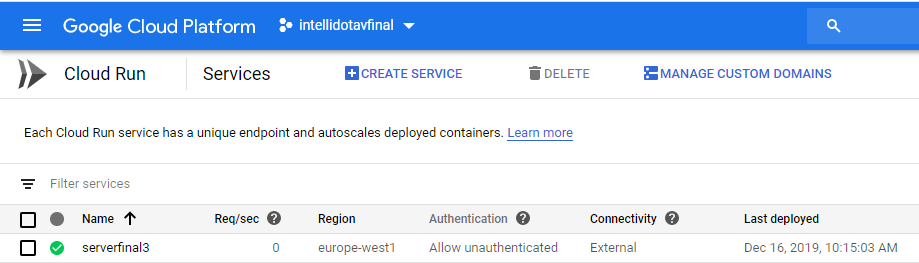
Më pas, me anë të një CLI (ang. *Command Line Interface*) publikojmë imazhin tonë lokal në Google Cloud Platform.



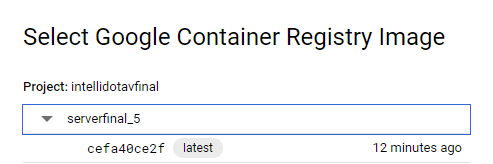
Komanda e parë i shoqëron projektin tonë lidhjen e faqes gcr.io, ku GCR qëndron për Google Cloud Repository, dhe më pas e shtyjmë projektin tonë nëpërmjet lidhjes së shoqëruar. Lidhja në vazhdim përbëhet nga emri i projektit të GCP-së që në rastin tonë është *intellidotavfinal* ndërsa *serverfinal\_5* është emri i degës, apo një super version në të cilën po bëhet publikimi.

Ky proces, pra ngarkimi, merr përafërsisht 10 minuta pasi madhësia e imazhit Docker është rreth 500 MB. Më poshtë vërejmë se shfaqen disa mesazhe si *Layer already exists*, kjo ndodhë për arsye se ne vetëm se kemi publikuar po të njëjtin version më herët (versionin në të cilin aktualisht është i ndezur serveri).

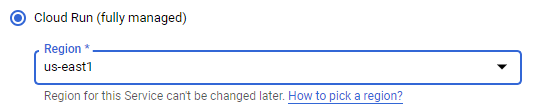
Pasi të kryejmë këtë veprim, kalojmë përsëri në GCP dhe vazhdojmë procesin në Cloud Run. Së pari krijojmë një shërbim të ri nëpërmjet butonit *Create Service*



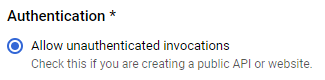
Me klikimin e këtij butoni, na shfaqet dritarja në të cilën duhet të zgjedhim kontejnerin nga lista e të gjithë kontejnerëve të publikuar nga ne (ne zgjedhim kontejnerin e publikuar më herët, pra super versionin serverfinal\_5 me etiketën *latest* – vetvetiu i shoqëruar nga Google pasi ne nuk i kemi specifikuar ndonjë të tillë).



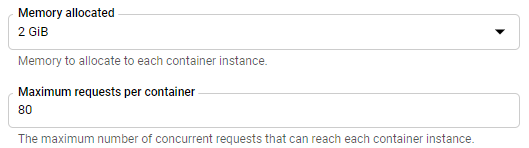
Caktojmë regjionin në mënyrë që thirrjet të realizohen në mënyrë më të shpejtë, me këtë duam të themi se nëse caktojmë një regjion me distancë të largët nga se ku jemi kyçur ne në internet, koha e transmetimit dhe pranimit të të dhënave nëpërmjet rrjetit është e lartë.



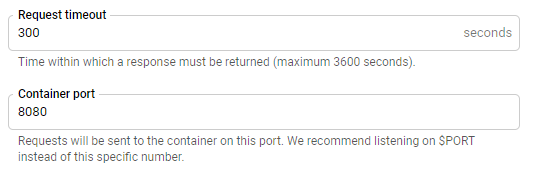
Lejojmë që thirrjet në shërbimet tona të jenë të realizueshme edhe nëse nuk janë të autentifikuara. Në këtë mënyrë, çdokush që dëshiron mund t’u qaset Json dokumenteve në mënyrë të drejtpërdrejtë duke shfrytëzuar lidhjen e aktuale që kemi mundësuar ne.



Ia ndajmë kontejnerit tonë 2GB memorie për arsye se aq kërkon edhe Spark në backend, që do të thotë se nëse i ndahen më pak njësi të memories, nuk mund të ndodhë procesimi, ndërkaq do të hidhen gabime, pasi nevojiten më shumë fuqi procesuse. Krahas kësaj, lejojmë edhe që maksimumi i kërkesave të përnjëhershme të jetë 80. Kjo praktikë aplikohet në mënyrë që të shmangen e ndryshme, ndër më të famshmit që është mohimi i shërbimeve (ang. *DOS – Denial Of Service Attacks*).



Më pas, kufizojmë një kohë që të kthehen përgjigjet që në rastin tonë është 300 sekonda dhe specifikohet porta e kontejnerit, që Google ia ka caktuar 8080. Kjo mund të ndryshohet, por duhet cekur se kjo portë dhe porta në të cilën është ndërtuar Docker imazhi duhet të jenë të njëjta, për te realizuar këtë gjë gjithmonë në mënyrë dinamike, kemi specifikuar portin në Dockerfile siç është përshkruar në IntelliDota Classification si variabël mjedisi.



Pra nëse ky proces kryhet saktë, shërbimi jonë do të jetë i qasshëm në portin 8080. Me këto të dhëna, klikojmë butonin Create dhe presim që të krijohet shërbimi jonë.

Nëse kemi gabuar përgjatë procedurës, na shfaqet ky imazh bashkë me gabimin përgjatë procedurës.



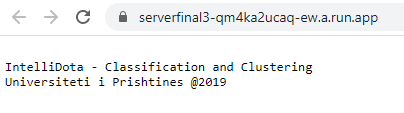
Përndryshe, themi se shërbimet tona janë të gatshme në ueb-faqen e specifikuar nga GPC, saktësisht na shfaqet imazhi në vijim.



Nëse na intereson të shohim më shumë të dhëna, kemi anën e djathë të ekranit, pra:



Përfundimisht kemi lidhjen https://serverfinal3-qm4ka2ucaq-ew.a.run.app të cilën nëse provojmë ti qasemi, shohim:



Pra, shërbimet tona janë të qasshme nga kushdo në çfarëdo kohe.

# 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/ |

## getColumns

|  |  |
| --- | --- |
| 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 |

## 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 |

## 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 |

## 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 |

## 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 |

## 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 |

## getClusterStats

|  |  |
| --- | --- |
| Emri | getClusterStats |
| URL | localhost/getClusterStats |
| Parametrat |  |
| Përshkrimi | Kthen pikat qendrore të të dhënave për secilin kllaster, pra informata për secilin atribut të data setit |
| Shembull | localhost:9000/getClusterStats |

## getClusterCount

|  |  |
| --- | --- |
| Emri | getClusterCount |
| URL | localhost/getClusterCount |
| Parametrat |  |
| Përshkrimi | Kthen numrin e sasisë së të dhënave për secilin kllaster |
| Shembull | localhost:9000/getClusterCount |

## postCluster

|  |  |
| --- | --- |
| Emri | postCluster |
| URL | localhost/postCluster |
| Parametrat | gold - paratë  gold\_per\_min - paratë për minutë  xp\_per\_min - eksperienca për minutë  kills - vrasjet gjatë lojës  deaths - vdekjet gjatë lojës  assists - asistimet  denies – mohimet e vrasjeve  last\_hits - goditja përfundimtare për vrasje  hero\_damage - dëmtimet ndaj kundërshtarëve  hero\_healing - ndihmesa ndaj ekipit  tower\_damage - dëmtimet ndaj kullave  level – nivelet |
| Përshkrimi | Kthen atributet përkatëse bashkë me një atribut të ri që tregon llojin e kllasterimit në të cilin përket data seti |
| Shembull | localhost:9000/postCluster?  gold=2000.0&gold\_per\_min=5113.0&xp\_per\_min=1231.0&kills=21.2  &deaths=3.3&assists=10.0&denies=34.3&last\_hits=411.1  &hero\_damage=56000.2&hero\_healing=0.0&tower\_damage=22000.0  &level=25.0 |