

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

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 nxjerrë me ndihmesën e Steam Public API, formati 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. Gjuhë programuese 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.

Në mënyrë që të ekzistojë qasja për në pikat fundore pa ndalesa, projekti është konvertuar në një kontejner dhe është hostuar në Cloud nëpërmjet Google Cloud Platform. Konvertimi në kontejner është realizuar nëpërmjet Docker. Lidhja për të kaluar në API realizohet nëpërmjet https://serverfinal3-qm4ka2ucaq-ew.a.run.app.

Tabela e figurave

[Fig 1: Algoritmi i një loje tic-tac-toe 7](#_Toc28087150)

[Fig 2: K-Means vizualisht 11](#_Toc28087151)

[Fig 3: Bisecting K-Means vizualisht 13](#_Toc28087152)

[Fig 4: Formacioni i lojtarëve të video-lojës Dota 2 16](#_Toc28087153)

[Fig 5: Organizimi i ekipit 17](#_Toc28087154)

[Fig 6: Mostra e data setit 18](#_Toc28087155)

[Fig 7: Kolonat e data setit me tipet përkatëse 18](#_Toc28087156)

[Fig 8: Rreshtat dhe kolonat e data setit të pre-procesuar 19](#_Toc28087157)

[Fig 9: Ngarkimi i data setit 19](#_Toc28087158)

[Fig 10: Zgjedhja dhe filtrimi në data set 19](#_Toc28087159)

[Fig 11: Grupimi dhe heqja e kolonës hero\_id 20](#_Toc28087160)

[Fig 12: Leximi gjatë aplikimit të algoritmit 20](#_Toc28087161)

[Fig 13: Funksioni për riemërimin e data setit 20](#_Toc28087162)

[Fig 14: Fazat nëpër të cilat ka kaluar algoritmi 21](#_Toc28087163)

[Fig 15: Përdorimi i fazave (gypit) sipas funksionit fit 22](#_Toc28087164)

[Fig 16: Ruajtja e modelit lokalisht 22](#_Toc28087165)

[Fig 17: Listimi i imazheve Docker 23](#_Toc28087166)

[Fig 18: Mundësia për zgjedhje dhe krijimit të projektit 24](#_Toc28087167)

[Fig 19: Shoqërimi i etiketës dhe shtyrja për në Cloud 24](#_Toc28087168)

[Fig 20: Mundësia për krijimin e një shërbimi në Cloud 25](#_Toc28087169)

[Fig 21: Imazhi i publikuar nga CLI 25](#_Toc28087170)

[Fig 22: Zgjedhja e regjionit më të përafërt 25](#_Toc28087171)

[Fig 23: Lejimi i qasjeve të pa autorizuara 26](#_Toc28087172)

[Fig 24: Memoria e alokuar në Cloud dhe maksimumi i kërkesave 26](#_Toc28087173)

[Fig 25: Porti i hostit në Cloud 26](#_Toc28087174)

[Fig 26: Ekzekutimi i imazhit me gabime gjatë procedurës 27](#_Toc28087175)

[Fig 27: Ekzekutimi i imazhit pa gabime gjatë procedurës 27](#_Toc28087176)

[Fig 28: Më shumë detaje rreth hostimit 27](#_Toc28087177)

[Fig 29: Lidhja përfundimtare e qasshme në cdo kohë 28](#_Toc28087178)

[Fig 30: Pamja hyrëse 29](file:///C:\Users\Labinot\Documents\GitHub\Raporti%20-%20NI.docx#_Toc28087179)

[Fig 31: Vartësia për rrëshqitje 29](file:///C:\Users\Labinot\Documents\GitHub\Raporti%20-%20NI.docx#_Toc28087180)

[Fig 32: Pamja e butoneve për klasifikim / kllasterim 30](file:///C:\Users\Labinot\Documents\GitHub\Raporti%20-%20NI.docx#_Toc28087181)

[Fig 33: Kolonat e diskretizuesit dhe rrëshqitësit 30](file:///C:\Users\Labinot\Documents\GitHub\Raporti%20-%20NI.docx#_Toc28087182)

[Fig 34: Diskretizuesi 30](file:///C:\Users\Labinot\Documents\GitHub\Raporti%20-%20NI.docx#_Toc28087183)

[Fig 35: Pamja e grafikut të Diskretizuesit 31](file:///C:\Users\Labinot\Documents\GitHub\Raporti%20-%20NI.docx#_Toc28087184)

[Fig 36: Pamja e parë e dy fazave 31](file:///C:\Users\Labinot\Documents\GitHub\Raporti%20-%20NI.docx#_Toc28087185)

[Fig 37: Faza Bucketizer 31](file:///C:\Users\Labinot\Documents\GitHub\Raporti%20-%20NI.docx#_Toc28087186)

[Fig 38: Pamja e vlerave në mostër 32](file:///C:\Users\Labinot\Documents\GitHub\Raporti%20-%20NI.docx#_Toc28087187)

[Fig 39: Pamja e kolonave në mostër 32](file:///C:\Users\Labinot\Documents\GitHub\Raporti%20-%20NI.docx#_Toc28087188)

[Fig 40: Matrica e korrelacionit të data setit Kaggle 33](file:///C:\Users\Labinot\Documents\GitHub\Raporti%20-%20NI.docx#_Toc28087189)

[Fig 41: Matrica e korrelacionit të data setit Steam 33](file:///C:\Users\Labinot\Documents\GitHub\Raporti%20-%20NI.docx#_Toc28087190)

[Fig 42: Vlerat fundore të korrelacionit 33](file:///C:\Users\Labinot\Documents\GitHub\Raporti%20-%20NI.docx#_Toc28087191)

[Fig 43: Struktura e data seteve 34](file:///C:\Users\Labinot\Documents\GitHub\Raporti%20-%20NI.docx#_Toc28087192)

[Fig 44: Pamja e klasifikimit në kohë reale 35](file:///C:\Users\Labinot\Documents\GitHub\Raporti%20-%20NI.docx#_Toc28087193)

[Fig 45: Kllasterët në mënyrë grafike 36](file:///C:\Users\Labinot\Documents\GitHub\Raporti%20-%20NI.docx#_Toc28087194)

[Fig 46: Grupimi i kllasterëve 39](file:///C:\Users\Labinot\Documents\GitHub\Raporti%20-%20NI.docx#_Toc28087195)

Tabela e tabelave

[Tab 1: Pika fundore index 40](#_Toc28087284)

[Tab 2: Pika fundore getColumns 40](#_Toc28087285)

[Tab 3: Pika fundore getSample 40](#_Toc28087286)

[Tab 4: Pika fundore getStages 41](#_Toc28087287)

[Tab 5: Pika fundore: getCorrelationMatrix 41](#_Toc28087288)

[Tab 6: Pika fundore getGroupAndCount 41](#_Toc28087289)

[Tab 7: Pika fundore getStages 41](#_Toc28087290)

[Tab 8: Pika fundore getSchema 42](#_Toc28087291)

[Tab 9: Pika fundore getDoubleGroup 42](#_Toc28087292)

[Tab 10: Pika fundore getClusterStats 42](#_Toc28087293)

[Tab 11: Pika fundore getClusterCount 42](#_Toc28087294)

Tabela e përmbajtjes

[1 Hyrja 6](#_Toc28087246)

[1.1 Motivimi 6](#_Toc28087247)

[1.2 Përshkrimi i Problemit 7](#_Toc28087248)

[2 Inteligjenca artificiale – Kllasterimi si mësim pa mbikëqyrje 9](#_Toc28087249)

[2.1 Algoritmet e mësimit pa mbikëqyrje 10](#_Toc28087250)

[2.1.1 K-Means 10](#_Toc28087251)

[2.1.2 Mean-Shift 12](#_Toc28087252)

[2.1.3 Bisecting K-Means 12](#_Toc28087253)

[3 Hyrje në projekt 15](#_Toc28087254)

[3.1 Loja kompjuterike Dota 2 16](#_Toc28087255)

[3.2 Para-procesimi i të dhënave 17](#_Toc28087256)

[3.3 Para-procesimi në Scala 19](#_Toc28087257)

[3.4 Aplikimi i Algoritmit 20](#_Toc28087258)

[3.5 Publikimi i imazhit Docker në Google Cloud Platform 23](#_Toc28087259)

[4 Zhvillimi i Aplikacionit 29](#_Toc28087260)

[4.1 Opsionet e menusë 30](#_Toc28087261)

[4.1.1 Mundësia për zgjedhje 30](#_Toc28087262)

[4.1.2 Diskretizuesi 30](#_Toc28087263)

[4.1.3 Fazat për Trajnim 31](#_Toc28087264)

[4.1.4 Mostrat e të Dhënave 32](#_Toc28087265)

[4.1.5 Matrica e Korrelacionit 33](#_Toc28087266)

[4.1.6 Struktura e Data Seteve 34](#_Toc28087267)

[4.1.7 Klasifikimi dhe kllasterimi në Kohë Reale 35](#_Toc28087268)

[4.1.8 Kllasterët 36](#_Toc28087269)

[4.1.9 Grupimi i Kllasterëve 39](#_Toc28087270)

[5 Lista e thirrjeve fundore të mundshme 40](#_Toc28087271)

[5.1 index 40](#_Toc28087272)

[5.2 getColumns 40](#_Toc28087273)

[5.3 getSample 40](#_Toc28087274)

[5.4 getStages 41](#_Toc28087275)

[5.5 getCorrelationMatrix 41](#_Toc28087276)

[5.6 getGroupAndCount 41](#_Toc28087277)

[5.7 getStages 41](#_Toc28087278)

[5.8 getSchema 42](#_Toc28087279)

[5.9 getDoubleGroup 42](#_Toc28087280)

[5.10 getClusterStats 42](#_Toc28087281)

[5.11 getClusterCount 42](#_Toc28087282)

[5.12 postCluster 43](#_Toc28087283)

# 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 së 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 (Fig 1).



Fig 1: Algoritmi i një loje tic-tac-toe

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:

* Të dhënat dhe algoritmi – fazë kjo që përmban hapat kryesorë në lidhje me mësimin e makinës. Duke filluar nga marrja e të dhënave, pastrimi i tyre, përkthimi i atributeve që të jenë të lexueshme për makinën dhe deri tek mundësia e përdorimit të algoritmit në data setin e përpunuar.
* Organizimi dhe Paraqitja në Flutter – kjo fazë përfshin komunikimin me thirrjet e krijuara në pjesën e Scala dhe përcaktimin e një një dizajni të përshtatshëm dhe përgjegjshëm për pajisje mobile e njëherësh të lexueshëm për përdoruesin. Në fund ka pasuar lidhja logjike e pamjes me të dhënat dhe testimi në pajisje të ndryshme.

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). Në thelb mësimi pa mbikëqyrje është sinonim për kllasterimin. Procesi i të mësuarit është i pa mbikëqyrur pasi shembujt hyrës nuk janë të etiketuar si klasë. Ne përdorim kllasterimin për të zbuluar klasa përbrenda të dhënave. Për shembull, një metodë e të mësuarit të pa mbikëqyrur mund të marr si hyrje një set të imazheve të numrave të shkruar më dorë. Supozojmë që i gjen 10 kllasterë me të dhëna. Këto kllasterë mund t’i korrespondojnë 10 numrave të veçantë nga 0 tek 9 respektivisht. Sidoqoftë, pasi që të dhënat e trajnimit nuk janë të etiketuar, modeli i mësuar nuk mund të na tregojë kuptimin semantik të kllasterëve të gjetur.

* 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 parim ky lloj i mësimit është sinonim i klasifikimit. Mbikëqyrja gjatë mësimit vjen nga shembujt e etiketuar në setin e trajnuar. Për shembull në problemin e njohjes së kodit postar, një set i kodeve postare të shkruara më dorë dhe përkthimet korresponduese të lexueshme nga makina përdoren si shembuj trajnues, të cilët mbikëqyrin mësimin e modelit klasifikues.

Në këtë aplikacion do të përdoret saktësisht të kllasterimi si mënyrë e të mësuarit pa mbikëqyrje, teknikë kjo që do të përshkruhet më poshtë saktësisht me algoritmet përkatëse.

## 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 (Fig 2):

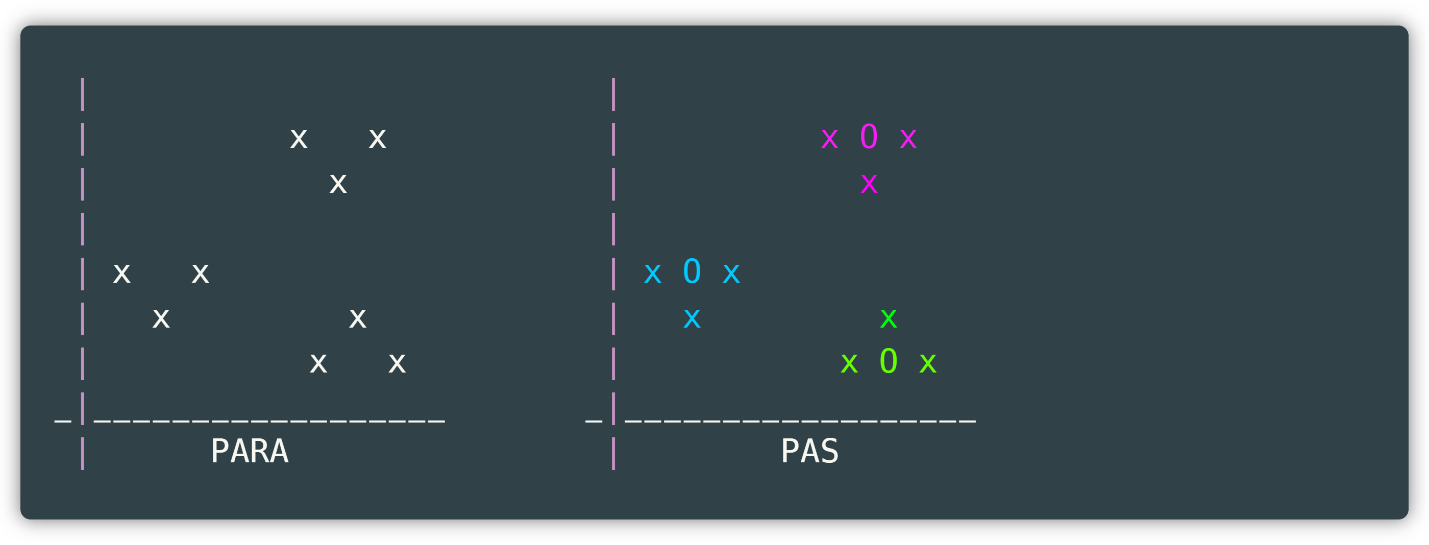


Fig 2: K-Means vizualisht

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

Mean Shift është një procedurë kllasterimi pa parametra, e famshme në fushën e pamjeve kompjuterike dhe përpunimit të imazheve, dalja e së cilës për dallim nga K-Means algoritmi, nuk varet nga supozimet eksplicite mbi formën e shpërndarjes së pikave, numrin e kllasterëve, ose çfarëdo forme të ndonjë inicializimi të rëndomtë.

Mean Shift ndërton mbi bazën e konceptit të vlerësimit të densitetit të bërthamës (ang. *kernel density estimation (KDE*). KDE është një metodë për të vlerësuar shpërndarjen themelore, e quajtur edhe funksioni i densitetit të probabilitetit, për një seri të të dhënave. Funksionon në atë mënyrë që vendosë një bërthamë (ang. *Kernel*) në secilën pikë të data setit. Bërthama është një shprehje matematikore për një funksion peshues. Ka lloje të ndryshme të këtyre bërthamave, por më i famshmi është bërthama Gausiane. Duke i mbledhur të gjitha bërthamat individuale do të gjenerohet një sipërfaqe e probabilitetit (p.sh *funksioni i densitetit*).

Mean Shift e shfrytëzon këtë ide të KDE, duke imagjinuar se çfarë do të ndodhte nëse pikat do të ngjiteshin lartë në majën më të afërt në sipërfaqen KDE. Kjo vazhdon deri sa të gjithë pikat kanë arritur të jenë në maje dhe se këto pika eventualisht do të bashkohen tek një seri e pikave, afër maksimës lokale të shpërndarjes. Ato pika që konvergjojnë tek makisma e njëjtë lokale konsiderohen anëtarë të kllasterit të njëjtë, numri i të cilëve varet nga gjerësia e brezit të sipërfaqes KDE, kështu po të merrej një vlerë shumë e vogël do të rezultonte me numër të kllasterëve aq sa kemi pika në data set, dhe në të kundërtën do të kishim vetëm një kllaster të formuar.

### Bisecting K-Means

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 (Fig 3).



Fig 3: Bisecting K-Means vizualisht

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 të kllasterimit si përfaqësues të të mësuarit pa mbikëqyrje të 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.
* *Postman* - i cili aktualisht është një nga mjetet më të njohura që përdoret në testimin e API, ku API qëndron për Ndërfaqe e Programimit të Aplikacioneve (ang. *Application Programming Interface*) dhe lejon aplikacionet softuerike të komunikojnë në mes vete përmes thirrjeve API.
* *Android Studio* - është mjedisi zyrtar i zhvillimit të integruar për sistemin operativ Android të Google, i krijuar posaçërisht për zhvillimin e Android. Ky mjedis është përdorur për zhvillimin e aplikacionit në Flutter, pasi ofron shumë lehtësi në kodim dhe gjithashtu është mjedis i rekomanduar nga Flutter.
* *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.
* *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.*

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* është një nga lojërat e tipit MOBA që ka kuptimin arenë e betejës me shumë lojtar në kohë reale (ang. *Multiplayer Online Battle Arena*), e njohur edhe si strategji veprimi në kohë reale, e cila me masat rafinerisë të aplikuara gjatë viteve ka arritur një nivel të lartë sofistikimi dhe me këtë ka tërhequr jo vetëm lojtar të shumtë si të rastësishëm ashtu edhe profesional, por edhe është bërë pjesë e studimeve të shumta dhe eksperimenteve të ndryshme sidomos në fushën e Inteligjencës Artificiale.

*Dota 2* qëndron për Mbrojtjen e Kullave (ang. *Defense of the Ancients*), ku lojtarët zgjedhin nga një grup prej më shumë se njëqind heronjsh, duke formuar dy ekipe që konsistojnë nga pesë lojtar secila, duke okupuar hapësirë të veçantë në hartë. Secili nga dhjetë lojtarët kontrollon në mënyrë të pavarur një karakter, të ashtuquajtur hero me aftësi unike dhe stile të ndryshme të lojës. Gjatë ndeshjes, lojtari dhe ekipi i tij grumbullojnë pikë eksperience dhe ari për të blerë artikuj për heronjtë e tyre, në mënyrë që të përshkojnë tek dymbëdhjetë kullat e ekipit kundërshtar. Një ekip fiton duke qenë e para për të shkatërruar bazën kryesore të armikut të quajtur *“Ancient”. Dota 2* ka një theks të veçantë tek taktika dhe koordinimi i ekipit, dhe një fokus të madh në strategjinë e ndërtimit të forcës sa më shpejt që është e mundur dhe zgjedhjen e rendit të azhurnimit të magjive të heroit.

Ekipet e lojës posedojnë një ndarje të një niveli më të lartë, pra mbajtësit (ang. *carries*) dhe mbështetësit (ang. *supports*)*.* Në esencë, çdo *support* mbështet një *carry* (Fig 4)*.*



Fig 4: Formacioni i lojtarëve të video-lojës Dota 2

Ndërsa në këndvështrim të lartë, ekipi organizohet në formacion të tillë (Fig 5):



Fig 5: 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 kllasterimi, pra postCluster.

## Para-procesimi i të dhënave

Si burim i të dhënave kemi përdorur Kaggle. Kaggle është një komunitet online për shkencëtarë të të dhënave dhe për ato që dëshirojnë të përvetësojnë aftësitë në të mësuar të makinës. 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 numrit të madh të 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. Nëntë 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 (Fig 6):

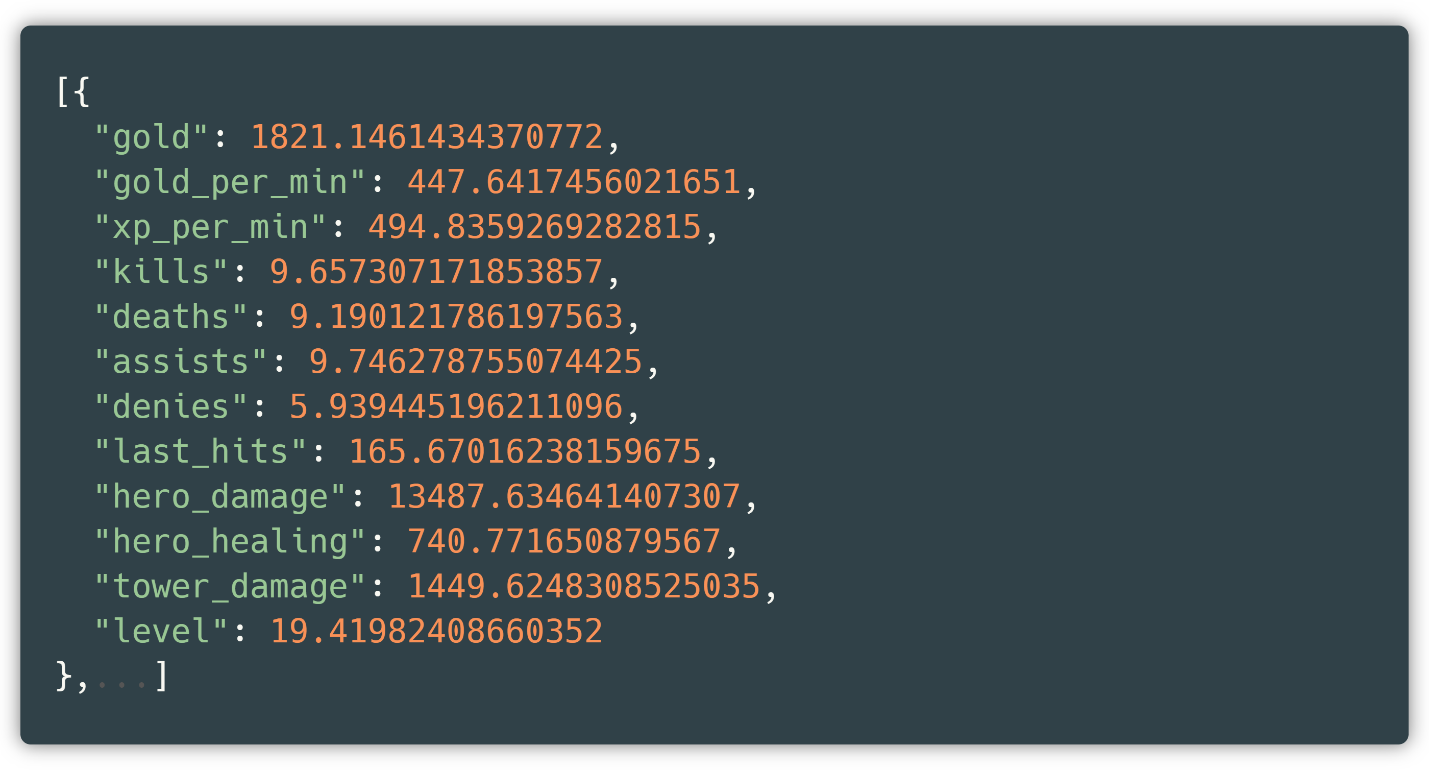


Fig 6: Mostra e data setit

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

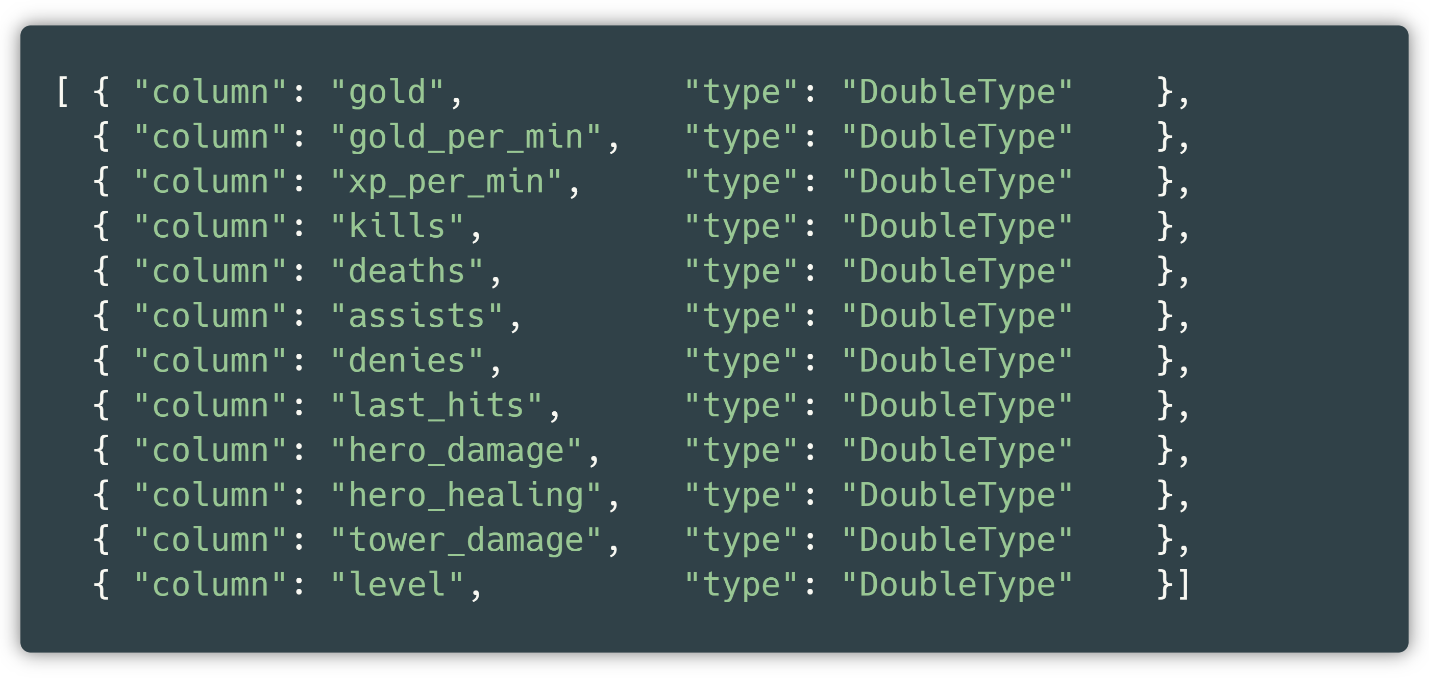


Fig 7: Kolonat e data setit me tipet përkatëse

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 (Fig 8):

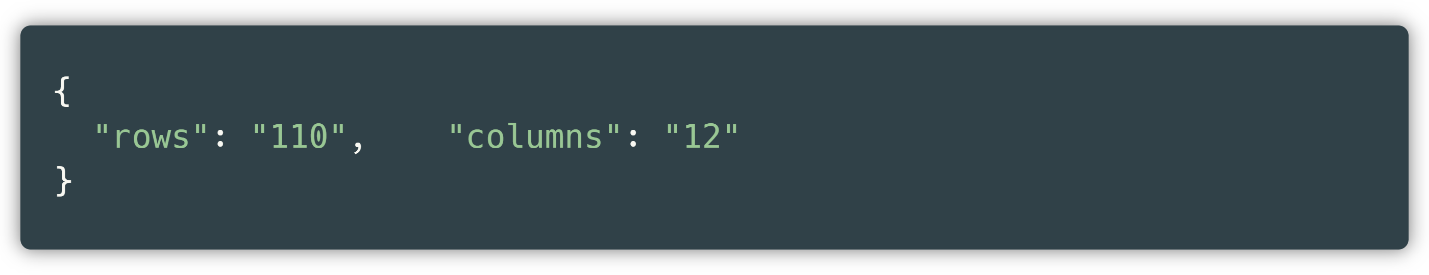


Fig 8: Rreshtat dhe kolonat e data setit të pre-procesuar

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 (Fig 9):



Fig 9: Ngarkimi i data setit

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 (Fig 10).

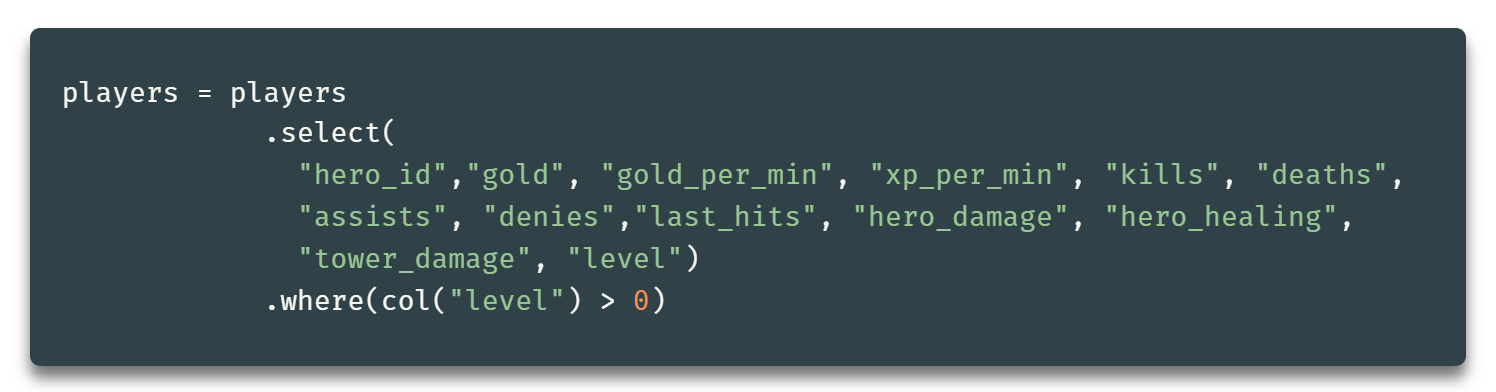


Fig 10: Zgjedhja dhe filtrimi në data set

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 madhe se zero 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 (Fig 11):

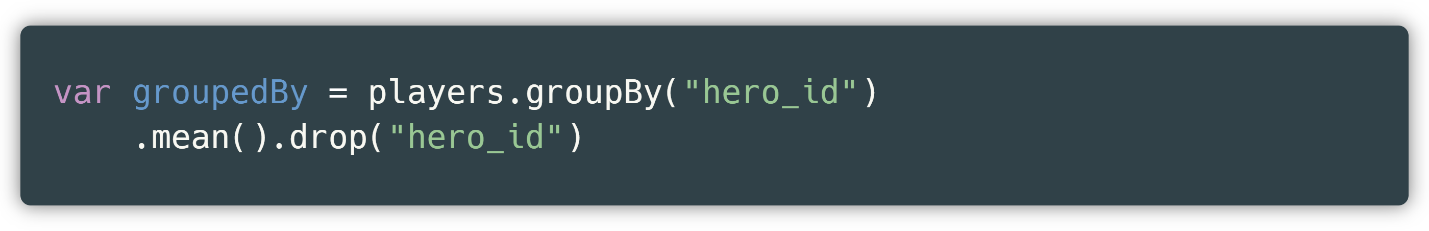


Fig 11: Grupimi dhe heqja e kolonës hero\_id

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 dublikatë 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.

## Aplikimi i Algoritmit

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 (Fig 12).



Fig 12: Leximi gjatë aplikimit të algoritmit

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*) me 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, kolona do të merr emrin që ka pasur para funksionit. Këtë metrikë ua aplikojmë të gjitha kolonave. Ky funksion duket si në vijim (Fig 13):



Fig 13: Funksioni për riemërimin e data setit

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 (Fig 14):

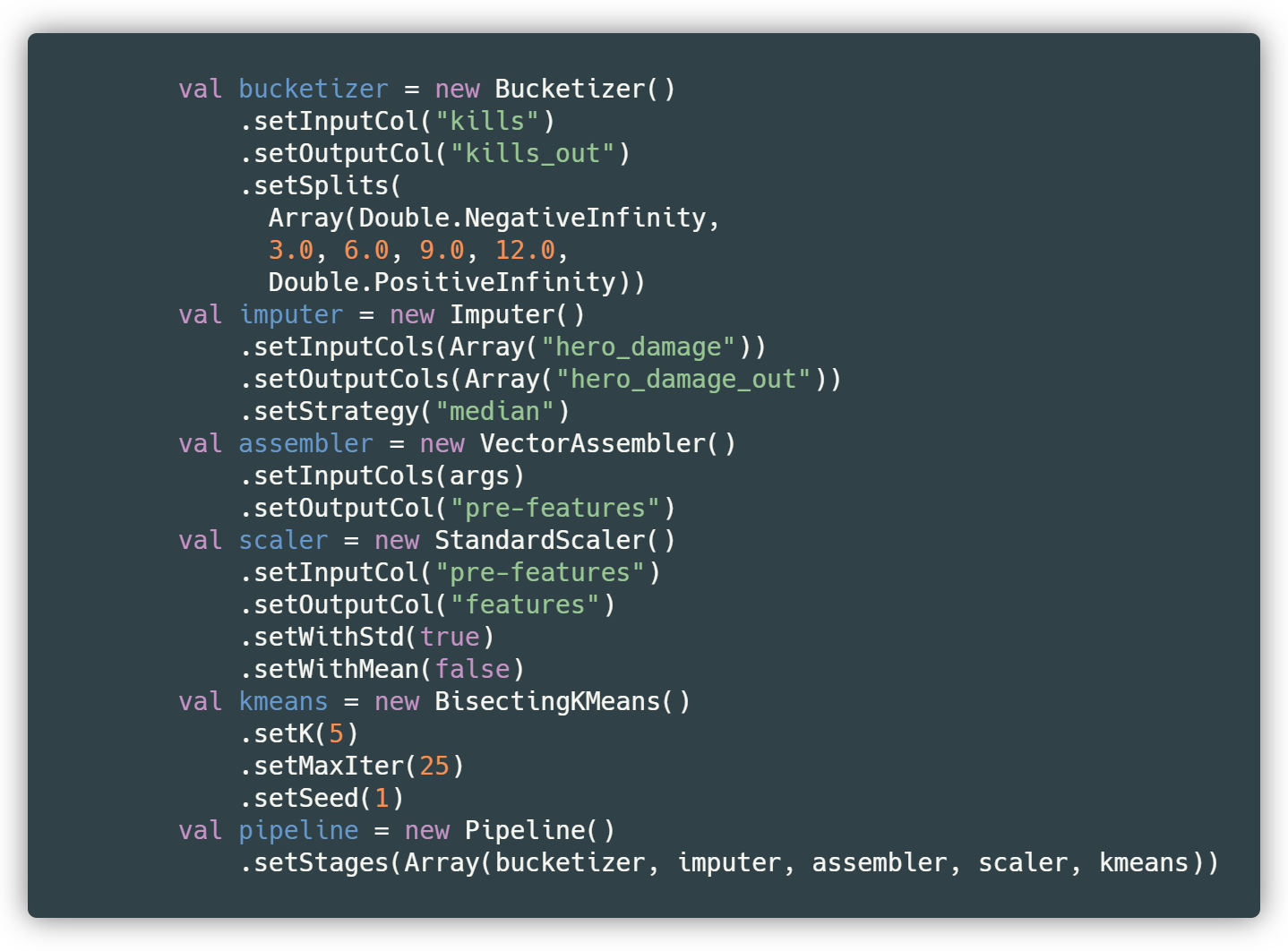


Fig 14: Fazat nëpër të cilat ka kaluar algoritmi

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 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 është përdorur për të shkallëzuar të dhënat. Ky lloj shkallëzuesi, ndër të shumtë të llojit të tij, shërben për të konvertuar secilën vlerë 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 parazgjedhura, 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 të grupohen në faza dhe të ekzekutohen një pas një në mënyre automatike para se të përdoret algoritmi. 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 (Fig 15):

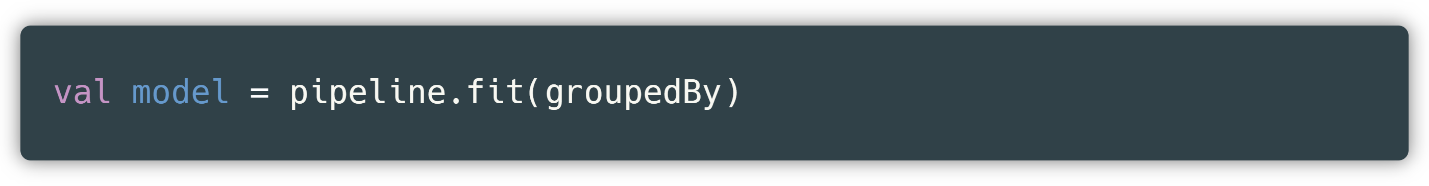


Fig 15: Përdorimi i fazave (gypit) sipas funksionit fit

Na mbetet vetëm të ruajmë modelin lokalisht (Fig 16):

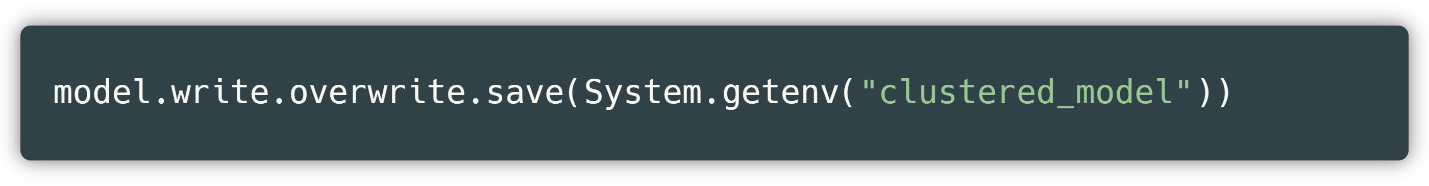


Fig 16: Ruajtja e modelit lokalisht

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 tonë 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, ku në këtë rast ne kemi pjesën 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 e makinës, një depo për dosje të ndryshme, vegla për DevOps si K8 e të tjera.

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 (Fig 17):



Fig 17: Listimi i imazheve Docker

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 (Fig 18).

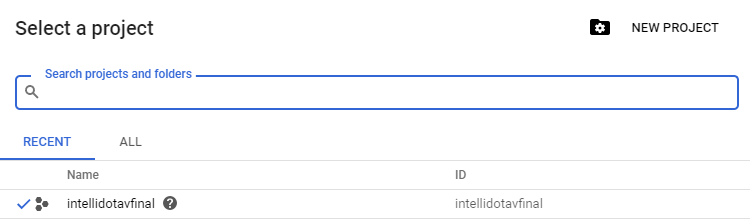


Fig 18: Mundësia për zgjedhje dhe krijimit të projektit

Me të krijuar projektin, hapim 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 (Fig 19).

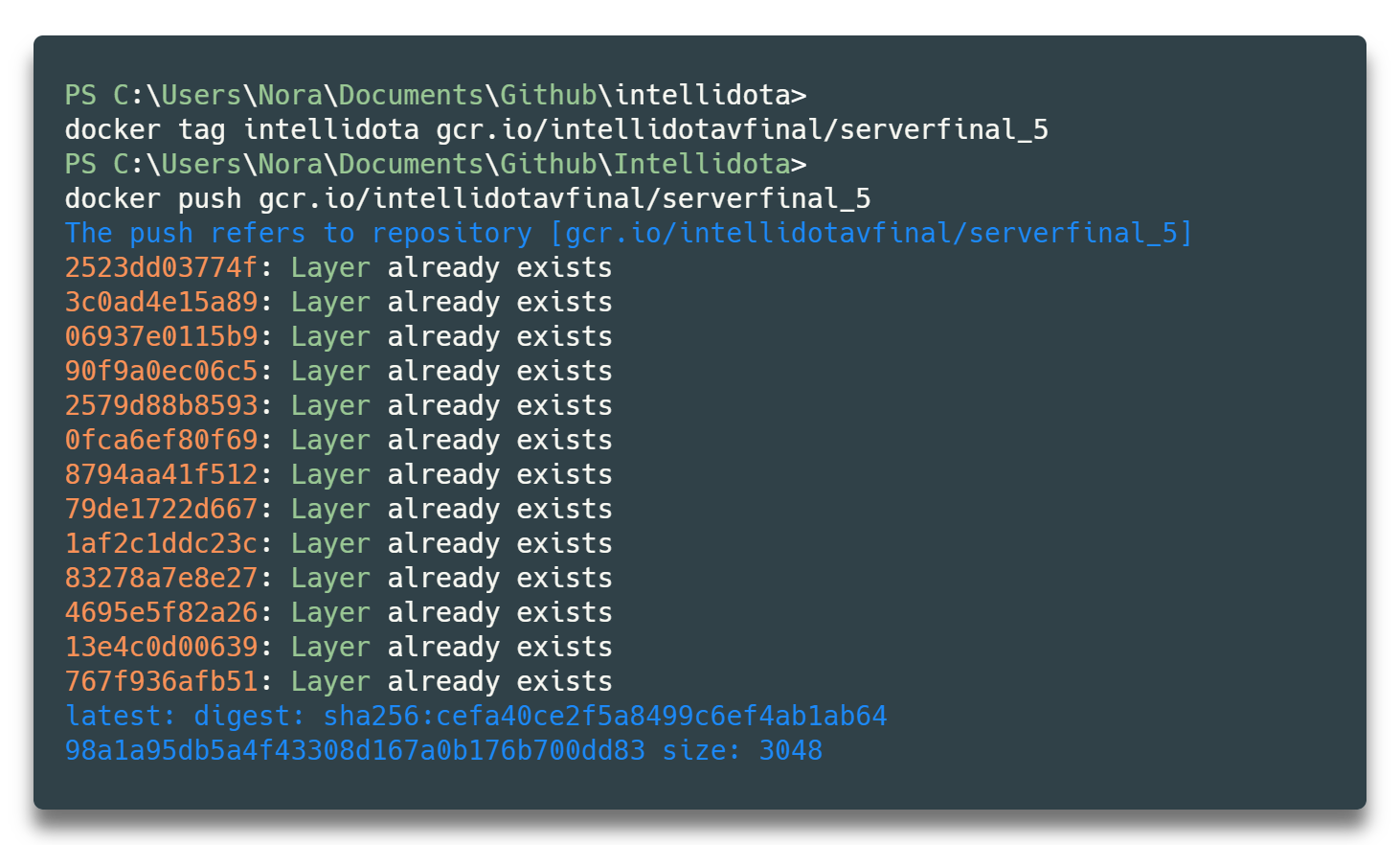


Fig 19: Shoqërimi i etiketës dhe shtyrja për në Cloud

Komanda e parë i shoqëron projektit 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* (Fig 20).

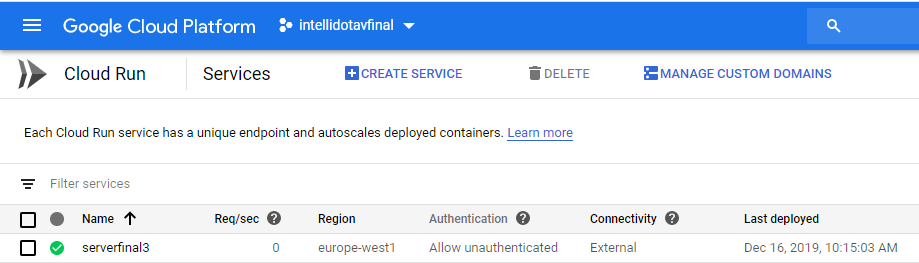


Fig 20: Mundësia për krijimin e një shërbimi në Cloud

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 (Fig 21, ne zgjedhim kontejnerin e publikuar më herët, pra super versionin serverfinal\_5 me etiketën *latest* – vetvetiu i shoqëruar nga Google).

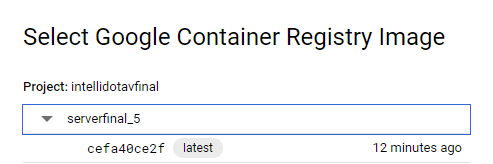


Fig 21: Imazhi i publikuar nga CLI

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 (Fig 22).

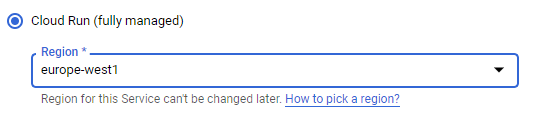


Fig 22: Zgjedhja e regjionit më të përafërt

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 (Fig 23).

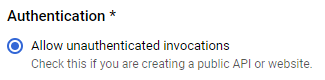


Fig 23: Lejimi i qasjeve të pa autorizuara

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 (Fig 24). Kjo praktikë aplikohet në mënyrë që të shmangen sulmet e ndryshme, ndër më të famshmit që është mohimi i shërbimeve (ang. *DOS – Denial Of Service Attacks*).

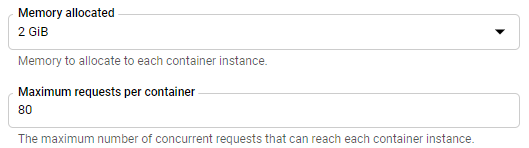


Fig 24: Memoria e alokuar në Cloud dhe maksimumi i kërkesave

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 (Fig 25). 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.

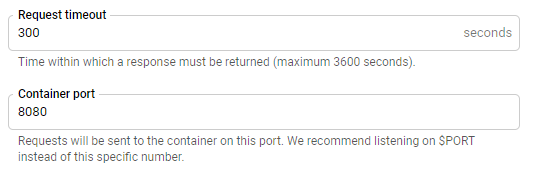


Fig 25: Porti i hostit në Cloud

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 (Fig 26).



Fig 26: Ekzekutimi i imazhit me gabime gjatë procedurës

Përndryshe, themi se shërbimet tona janë të gatshme në ueb-faqen e specifikuar nga GPC, saktësisht na shfaqet imazhi në vijim (Fig 27).



Fig 27: Ekzekutimi i imazhit pa gabime gjatë procedurës

Nëse na intereson të shohim më shumë të dhëna, kemi anën e djathë të ekranit, pra (Fig 28):



Fig 28: Më shumë detaje rreth hostimit

Përfundimisht kemi lidhjen https://serverfinal3-qm4ka2ucaq-eë.a.run.app të cilën nëse provojmë ti qasemi, shohim (Fig 29):

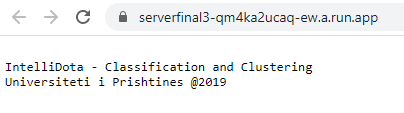


Fig 29: Lidhja përfundimtare e qasshme në cdo kohë

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

# Zhvillimi i Aplikacionit

Aplikacioni është zhvilluar në gjuhën programuese Dart e cila është gjuhë e orientuar në objekte, duke përdorur Flutter SDK. Flutter është një strukturë e zhvillimit të aplikacioneve mobile me burim të hapur e krijuar nga Google. Përdoret për të zhvilluar aplikacione për Android dhe IOS, si dhe është mënyra primare se si krijohen aplikacionet për Google Fuchsia. Flutter me mënyrën e krijimit të saj e lehtëson jashtëzakonisht punën duke ju dhënë zhvilluesve mundësi të shumta, gjë që krijon një eksperiencë të mirë dhe gjithashtu e përshpejton kohën e zhvillimit. Mbështet disa mjedise për të ndërtuar aplikacione, të cilat janë IntelliJ, Visual Studio Code dhe Android Studio. Një ndër karakteristikat e Flutter është edhe ri-ngarkimi i aplikacionit i cili çdo ndryshim të bërë në kod na mundëson ta vëzhgojmë në kohë reale.

Fig 30: Pamja hyrëse

Bazuar në pikat fundore të krijuara në Scala, janë krijuar thirrjet e veçanta në Flutter duke përdorur paketën *http* të zhvilluar nga Dart, ku në bazë të tyre është mundësuar krijimi i vizualizimeve të të dhënave.

Për të demonstruar më së miri mënyrën e funksionimit dhe strukturën e Flutter, fillojmë me pamjen hyrëse të aplikacionit në të cilën jepen detaje te përgjithshme për atë se çfarë është funksioni kryesor i tij (Fig 30). Në momentin që aplikacionit hapet, përdoruesit si pamje e parë i shfaqet ballina e aplikacionit, e cila është ndërtuar me anë të komponentëve (ang. *Widgets*)të cilat janë karakteristika kryesore e mbi të cilat është ndërtuar gjithçka në Flutter. Në këtë pjesë ne mund të fitojmë një kuptim të shkurtër mbi atë se çfarë përfshihet në projekt, kjo arrihet duke zvarritur panelin e poshtëm (Fig 31), i cili është krijuar duke përdorur paketën e mëposhtme e cila ofron një *Widget* të gatshëm që ofron këtë pamje.

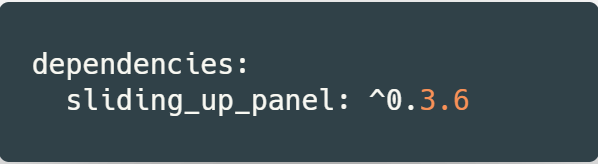
Për të manovruar tek pjesët tjera të aplikacionit ne mund të shfrytëzojmë menynë e pozicionuar në pjesën e epërme të ballinës e cila do të hap pamjen me listën e ekraneve, ku kemi gjithsej tetë ekrane të mundshme tek të cilat mund të navigojmë.

Fig 31: Vartësia për rrëshqitje

## Opsionet e menusë

### Mundësia për zgjedhje

Meqenëse ne kemi dy data sete të trajnuara, metrikët mund të aplikohen në data setin e krijuar vetë, pra Steam apo Kaggle. Se në cilin data set duam të operojmë, kemi krijuar dy butona (ang. *Tabs*, Fig 32) me anë të së cilëve vendoset lloji i data setit përkatës. Duhet cekur se ky opsion nuk ekziston tek menyja Kllasterët dhe tek menyja Grupimi i Kllasterëve që funksionojnë vetëm për kllasterim. Ne si shembull kemi zgjedhur 'Kllasterim' që do të thotë se metrikët operohen në data setin Kaggle dhe si kolona automatikisht rreshtohen të data setit përkatës, në rastin tonë të data setit Kaggle.

Fig 32: Pamja e butoneve për klasifikim / kllasterim

### Diskretizuesi

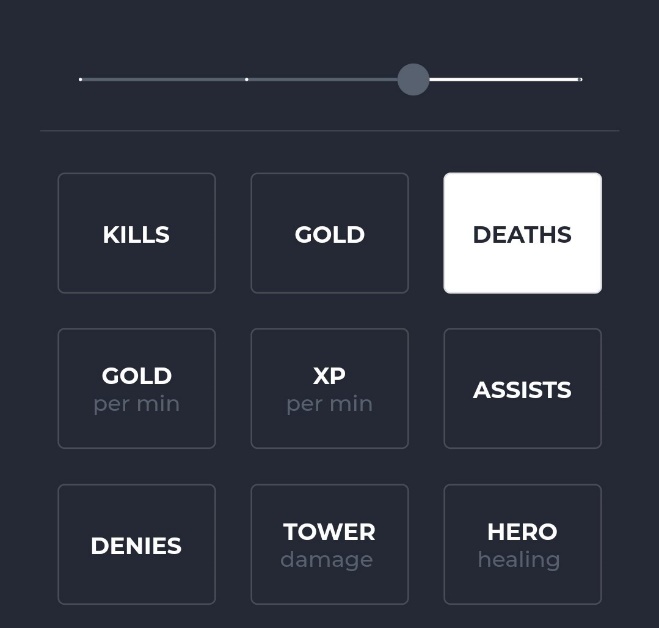
Diskretizuesi është një nga veglat më të dobishme në asetin tonë, që paraqet transformimin e vlerave të vazhdueshme në vlera të numërueshme. Meqenëse data setet përbëhen nga vlera të vazhdueshme me madhësi të lartë, ka qenë e arsyeshme ndërtimi i një vegle të tillë që diskretizon të dhënat në njërin nga grupet e zgjedhura nga rrëshqitësi (ang. *slider*). Ky ekran mund të hapet me zgjedhjen e *Diskretizuesi* nga pjesa e menysë siç shihet në vijim (Fig 33):

Fig 33: Kolonat e diskretizuesit dhe rrëshqitësit



Fig 34: Diskretizuesi

Tek ky ekran ne kemi mundësinë që të shohim shpërndarjen e të dhënave në mënyrë grafike.

Së pari nga lista e shfaqur në pjesën e poshtme të ekranit ne zgjedhim njërën nga kolonat për të cilën dëshirojmë të bëjmë këtë vizualizim dhe më pastaj me anë të rrëshqitësit (ang. *Slider*) ne mund të vendosim për numrin e ndarjeve në të cilat do të grupohen vlerat e kolonës përkatëse (Fig 33). Kemi vendosur që të lejojmë mundësinë e zgjedhjes të një nga 4 vlerat e ndryshme të rrëshqitësit, meqë nga vlerësimet e bëra këto vlera reprezentojnë më së miri voluminozitetin e të dhënave.

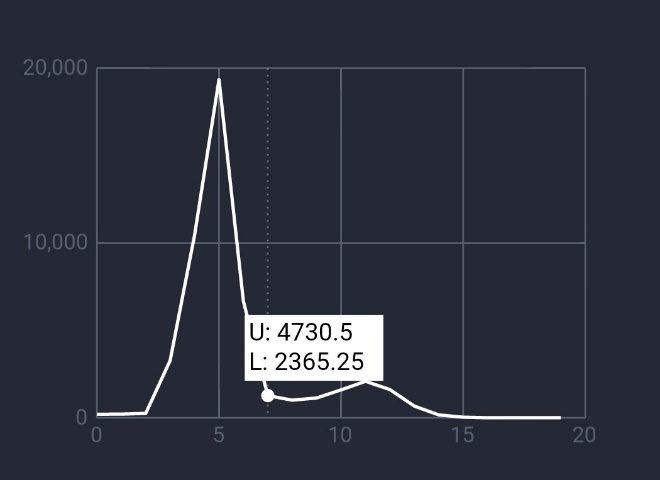
Me zgjedhjen e njërës prej kolonave, kërkesa do të procesohet dhe si rezulat do të shfaqet një reprezentim grafik i cili demostron në boshtin X numrin e grupimeve të zgjedhura nga ne, qofte ai 5, 10, 15 apo 20 grupime ndërsa në boshtin Y sasinë e grupimit (Fig 35). Për të parë se cili është rangu i vlerave të grupuara, klikojmë në ndonjë pikë në drejtëz dhe shfaqet vlera më e ulët dhe më e lartë, ku shkronja U qëndron për lartë (ang. *Upper*) dhe L për poshtë (ang. *Lower*). Si shembull kemi imazhin në fig ku vlera më e vogël është 2365.5 dhe vlera më e lartë 4730.5, ku në rastin tonë janë rreth 2000 vlera siç vërehet në boshtin Y, ndërsa grupimi i takon kovës me numër 7 në boshtin X. Kjo vlerë është fituar nga të dhënat e grumbulluara për klasifikim, duke përdorur atributin *gold\_per\_min*. Duhet cekur se ky ekran është i ndërtuar me anë të API-të *getGroupAndCount*.

Fig 35: Pamja e grafikut të Diskretizuesit

### Fazat për Trajnim

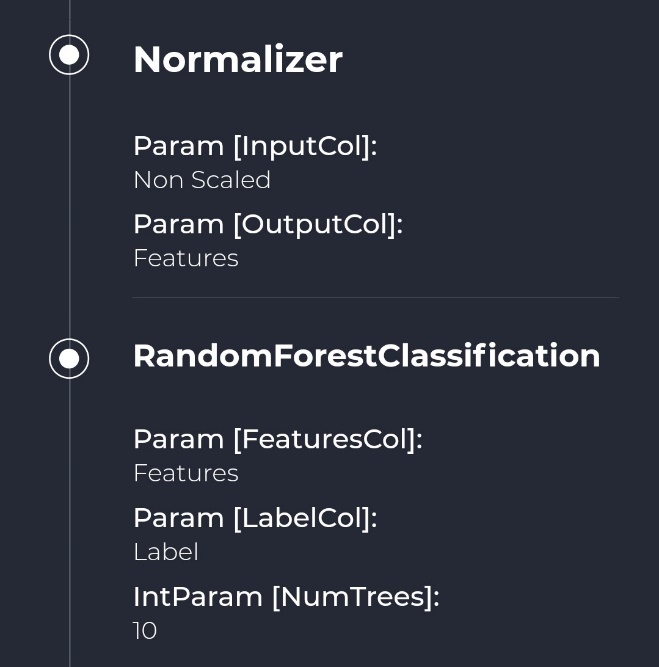
Gjatë para-procesimit të të dhënave deri tek përzgjedhja e algoritmit në mënyrë që saktë të shikohen fazat nëpër të cilat kanë kaluar algoritmet tona deri sa janë trajnuar, kemi vizualizuar ato me renditje nga lartë poshtë, që do të thotë se faza më e lartë është ekzekutuar e para ndërsa faza më poshtë paraqet fazën e fundit.

Fig 36: Pamja e parë e dy fazave

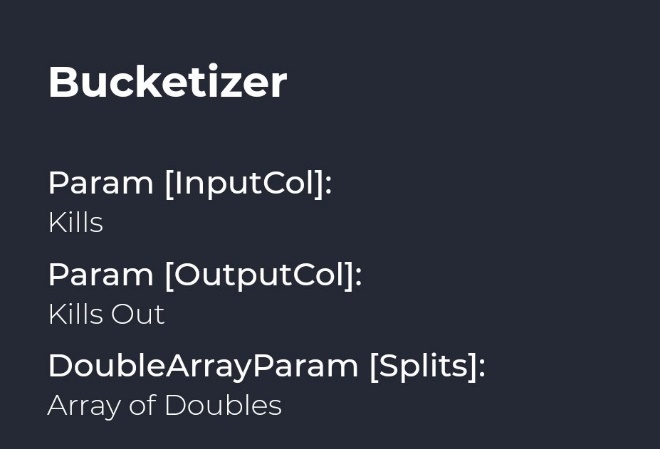


Fig 37: Faza Bucketizer

Nëse për instancë duam të shqyrtojmë se çfarë metodash janë përdorur në krijimin e një modeli të trajnuar, kemi, si shembull një Bucketizer ku kërkon disa parametra si kolona hyrëse, kolona dalëse dhe numri i ndarjeve (Fig 36). Duhet cekur se nëse një metodë është përdorur dy apo më shumë herë, metodës së përdorur më vonë i shtohet një numër i rastësishëm, për shembull, nëse përdorim një metodë tjetër të Bucketizer atëherë metoda tjetër emërohet Bucketizer4 (apo një numër të rastësishëm).

Një bashkësi e këtyre metodave formon një gyp (ang. *pipeline*) të tërë që prodhon një model të trajnuar. Nëse shqyrtojmë një gyp, për shembull Klasifikimin, mund të themi se së pari ndodhë normalizimi i të dhënave, ku si parametër hyrës kemi një numër kolonash ndërsa si dalje, jep një kolonë të vetme të quajtur Features ku vlerat janë të shkallëzuara ndërmjet 0 dhe 1. E pas kësaj përdoret algoritmi Random Forest Classifier me numër pemësh 10 (Fig 36). Duke vazhduar kështu, konstruktohet gypi i tërë. Gypat janë të dobishëm jo vetëm që procesi të jetë optimal por edhe për lexueshmëri. Gypi formohet në mënyrë dinamike dhe varësisht metodave të cilat i përdorim gjatë periudhës fillestare të mësimit të makinës, krijohen fazat përkatëse. Ky ekran është realizuar me anë të API-të *getStages.*

### Mostrat e të Dhënave

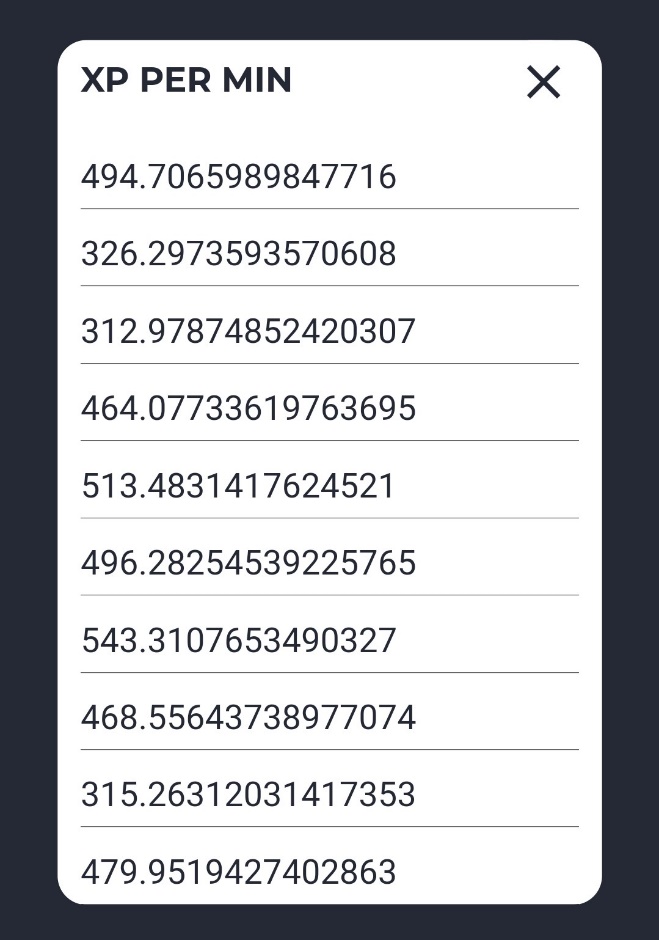
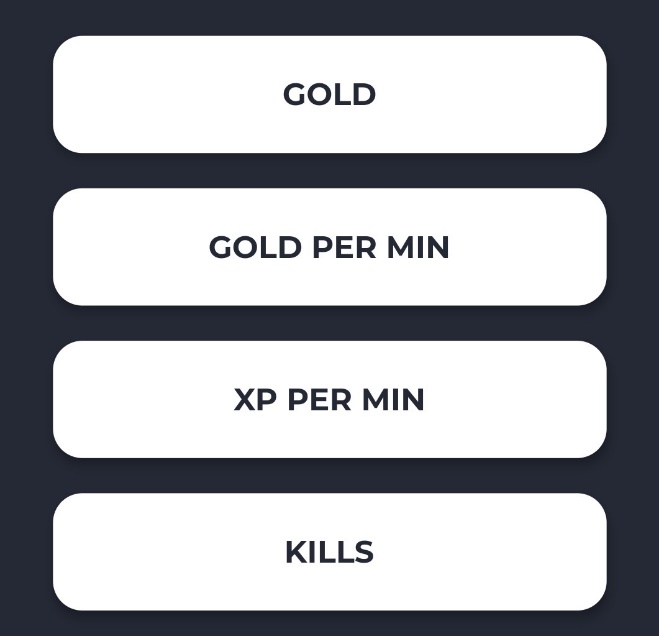
Për të shfaqur më mirë për përdoruesin të dhënat e grumbulluara, kemi krijuar një listë me emrat e atributeve të data seteve përkatëse, listë kjo e cila gjenerohet dinamikisht, ku secili element përfshin në vete një përqindje (20%) te vlerave të cilat japin një reprezentim për llojin e të dhënave të cilat shtrihen përgjatë fushave (Fig 38, Fig 39). Një shembull i kolonave të gjeneruara dhe të vizualizuara duket kështu.

Fig 38: Pamja e vlerave në mostër

Fig 39: Pamja e kolonave në mostër

Ndërsa me klikimin e atributit *XP PER MIN* do te hapet një fushe me vlerat e këtij atributi, përshtypja e parë e së cilave na len të kuptohet se janë vlera numerike decimale dhe se i përkasin një rangu nga 300 deri 600. Ky ekran është realizuar me anë të API-të *getSample* që pranon dy parametra, data setin që në këtë rast është Kaggle dhe një përqindje që në këtë rast është 20. Meqenëse është e vështirë të vizualizohen të dhënat e një mostre në tabelë në një sistem operativ iOS apo Android atëherë për shkak të dukjes, është parë më e arsyeshme që të shfaqen kolonat vertikalisht.

### Matrica e Korrelacionit

Matrica e korrelacioni është një lloj tabele që tregon koeficientet e korrelacionit ndërmjet variablave. Secila celulë në tabelë tregon lidhjen midis dy variablave. Matrica është në formë katrore, duke figuruar të njëjtat emra atributesh në të dyja boshtet. Shërben për të përmbledhur sasi të mëdha të të dhënave, ku qëllimi kryesor është që të fitohet një model, e ku tek ne në rastin e data setit të Kllasterimit vërehet se atributet janë shumë të lidhura me njëra tjetrën, ndërsa tek Klasifikimi mbizotëron më shumë pavarësia ndërmjet atributeve. Kodimi i korrelacioneve është i rangut të vlerave nga -1 deri në 1, ku 0 (Fig 42) tregon se vlerat aspak nuk korrelojnë me njëra-tjetrën, 1 do të thotë se variablat korrelojnë shumë ndërsa -1 do të thotë se ekziston korrelacion invers i lartë.

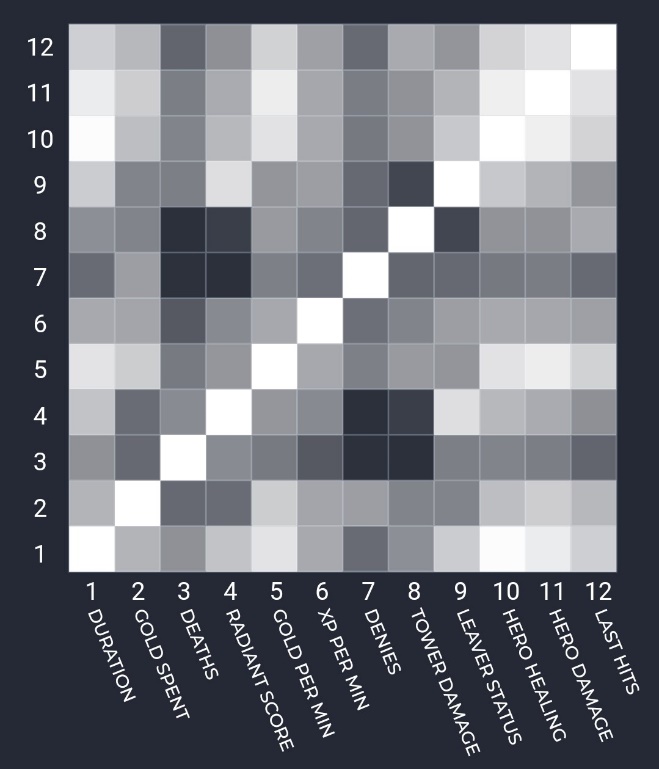
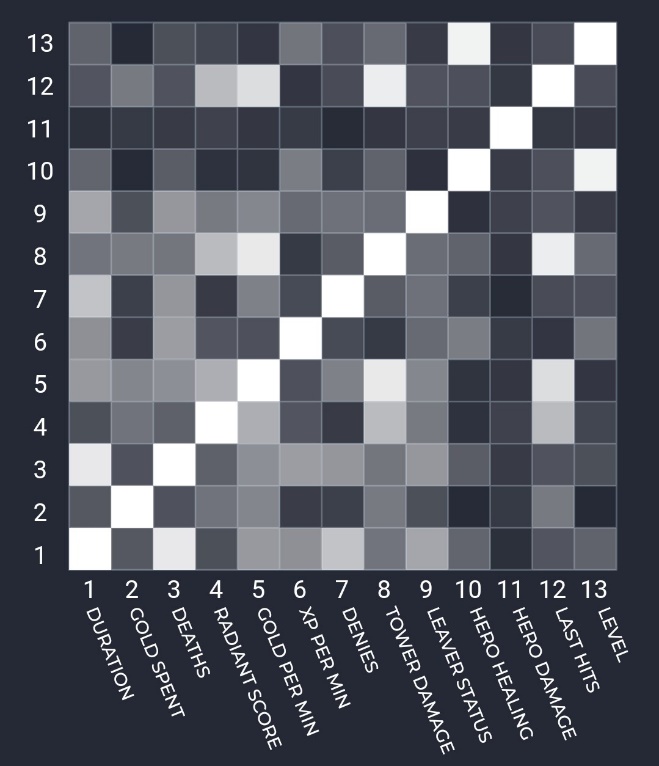


Fig 40: Matrica e korrelacionit të data setit Kaggle

Fig 41: Matrica e korrelacionit të data setit Steam

Fig 42: Vlerat fundore të korrelacionit

Kodimi mund të bëhet edhe në rangje të tjera por kjo nuk ka efekt të madh pasi çfarë do vlerë mund të pasqyronte vartësinë në mënyrë të duhur.

Për të shkallëzuar variablat nuk është përdorur ndonjë librari për këtë qëllim por metoda *Correlation* në Scala ka një shkallëzues unik të integruar në vete.

Meqë në Flutter nga paketa e cila është përdorur për vizualizim nuk ka pasur ndonjë *Widget* të gatshme për krijimin e matricës së korrelacionit, është përdorur një klasë e avancuar e Flutter, *CustomPaint*, e cila ofron një kanvacë (ang. *Canvas*) për vizatimin e formave të ndryshme të cilat nuk do të mund të krijoheshin në mënyrë të saktë me anë të *Widgets* të cilat ekzistojnë. Kështu nga kjo klasë të cilën ne e trashëgojmë dhe përdorim metodën vizato (ang. *paint*), tek e cila edhe shënojmë funksionet për të vizatuar në mënyrë dinamike celulat e vartësisë. Ky ekran është realizuar me anë të API-të *getCorrelationMatrix* që pranon si parametër data setin që duam të aplikojmë matricën (Fig 40, Fig 41).

### Struktura e Data Seteve

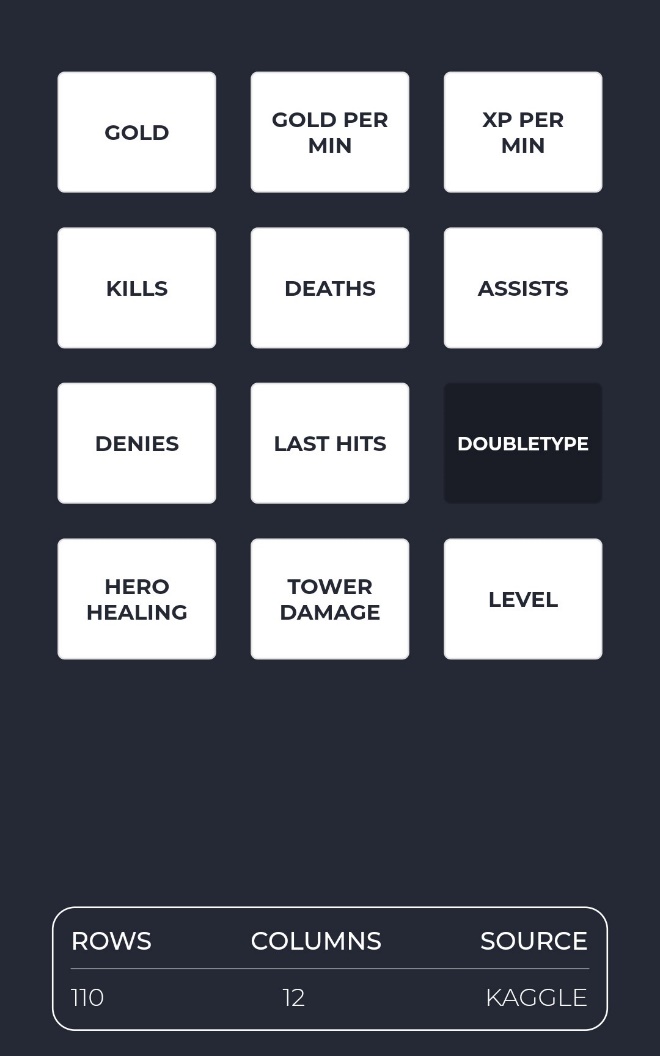
Në këtë pjesë të aplikacionit kemi ofruar një pamje e cila na informon për numrin e atributeve dhe fushave të përdorura për secilin data set, si dhe kemi mundësinë të shohim tipin e të dhënave për secilin atribut, ndonëse thuajse të gjithë kanë tip të njëjtë të të dhënave dhe dallojnë vetëm nga data seti në data set. Në rastin aktual (Fig 43), si data set po përdorim Kaggle nga shohim se tipi i atributit të klikuar është Double ndërsa data seti përbëhet nga 12 kolona dhe 110 rreshta. Rikujtojmë se ky data set është i para-procesuar dhe se data seti origjinal përmban përafërsisht 500.000 rreshta dhe rreth 70 kolona.

Fig 43: Struktura e data seteve

Ky ekran është realizuar me gërshetimin e dy API të përdorur, që janë *getSchema* që mundëson listimin e kolonave bashkë me tipin e tyre, ndërsa *getStats* ofron numrin e kolonave, rreshtave dhe burimin e data setit për të cilin është gjeneruar statistika.

Duhet cekur se ky ekran kërkon kohën më të lartë për procesim dhe hapje, pasi kalkulimi i rreshtave dhe kolonave bëhet në kohë reale, pra me klikimin e butonit, njihen këto të dhëna dhe shfaqen në ekran, që do të thotë se nuk ekzistojnë vlera të ruajtura paraprakisht në aplikacion.

### Klasifikimi dhe kllasterimi në Kohë Reale

Ndër pjesët kryesore të aplikacionit është klasifikimi dhe kllasterimi në kohë reale. Me kohë reale nënkuptojmë mundësinë që të përdoren modelet e trajnuara kurdo që klikojmë butonin *Predict* me kolonat e caktuara. Pamja e parëduket kështu si për klasifikim edhe për kllasterim. Shohim () se tek secila hapësirë në të cilën mund të shkruajmë kemi emrin e kolonës që i përket hapësira dhe një vlerë numerike, që paraqet vlerën e para definuar të kolonës. Ky ekran përbëhet nga 3 thirrje fundore, të cilat janë:

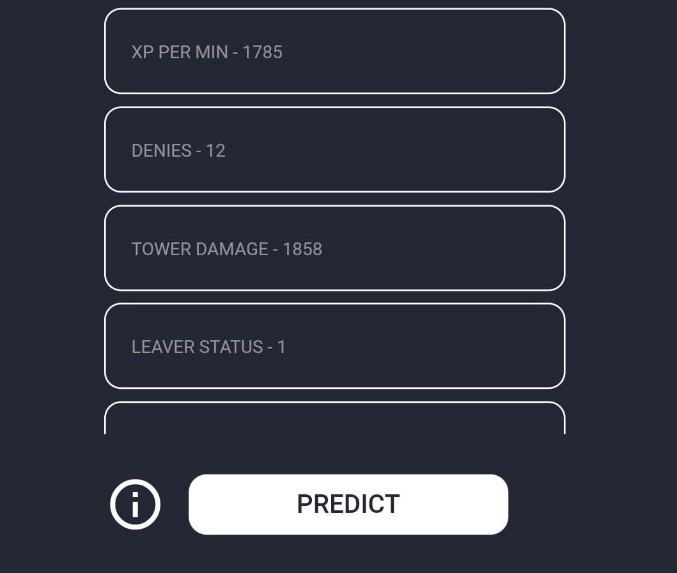
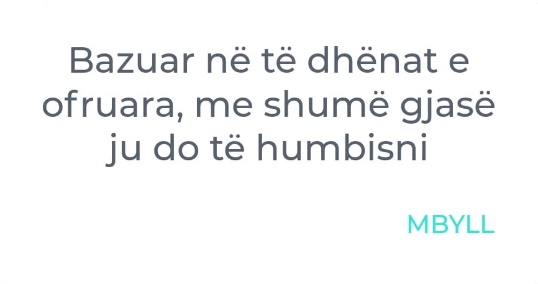
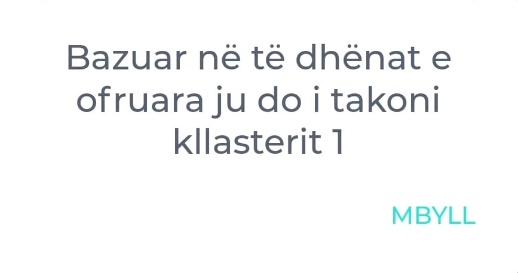
* postPredict - për të realizuar klasifikimin ku si hyrje jepen kolonat e data setit ndërsa si dalje ofrohet një tekst ku tregohet se a e kemi fituar lojën apo humbur.

Fig 44: Pamja e klasifikimit në kohë reale

* postCluster - për të realizuar kllasterimin ku si hyrje jepen kolonat e data setit ndërsa si dalje ofrohet numri i kllasterit në të cilin janë grupuar të dhënat.
* getColumns – që liston emrat kolonave bashkë me vlerat e para definuara.

Me kryerjen e procesit të klasifikimit apo kllasterimit, na shfaqet një tekst në të cilin tregohet se sa është vlera e predikuar apo se cilit kllaster i përkasin vlerat e reja.

Tek pamja e parë shihet se me këto atribute të para definuara ne do të humbim lojën, siç mund të kuptohet edhe nga teksti që shfaqë ‘Bazuar në të dhënat e ofruara, me shumë gjasë ju do të humbisni’.



Ndërsa me atributet e para definuara të kllasterimit, pra pamjes së dytë, shohim se do t’i takojmë grupimit numër 1, pra ‘Bazuar në të dhënat e ofruara ju do i takoni kllasterit 1’.

Se çfarë të dhënash ka grupimi 1, do të mund të kuptohet nëse lexohet pjesa e Kllasterët.

### Kllasterët

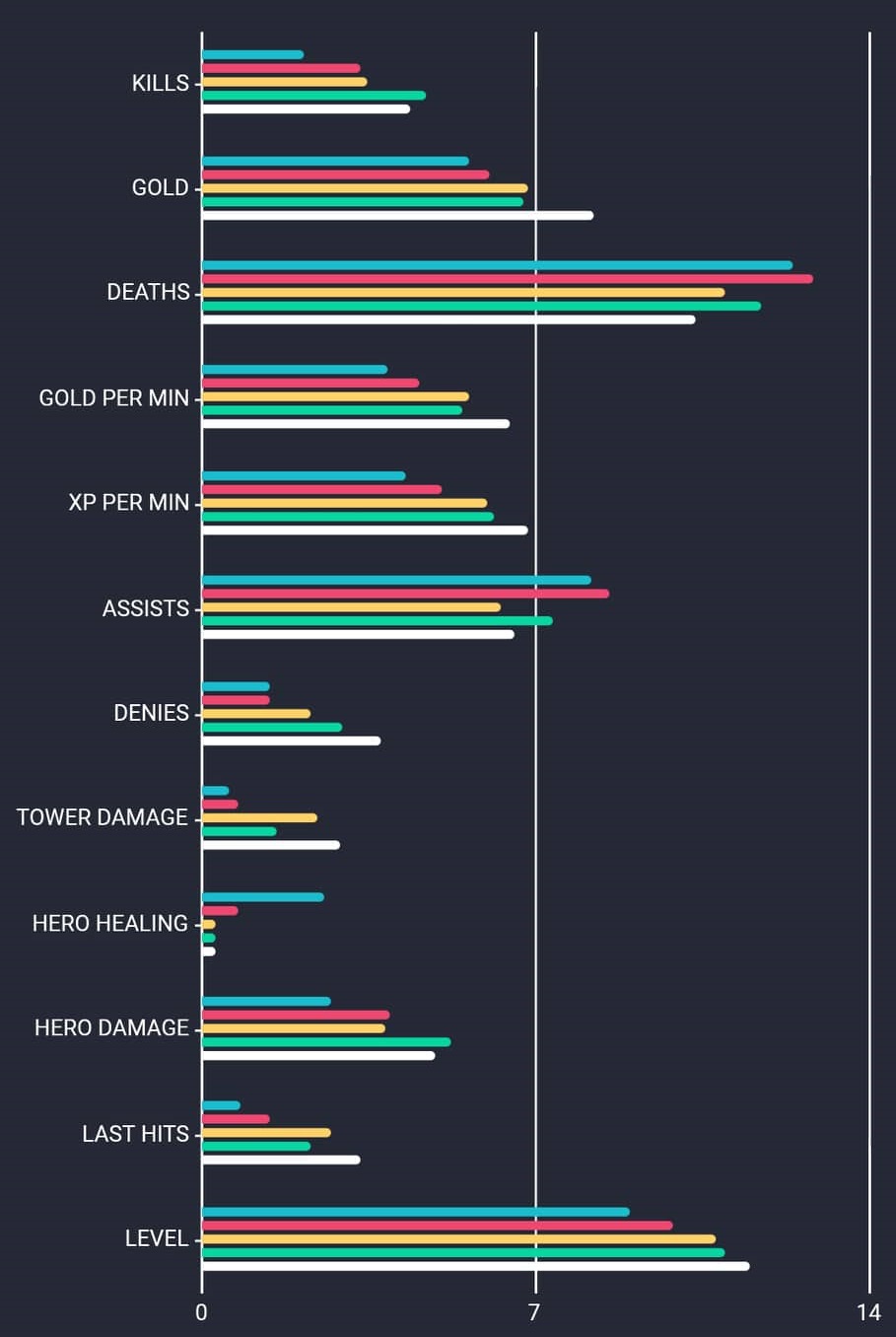
Ekrani i kllasterimit paraqet në mënyrë grafike të dhënat e kllasteruara në një renditje të caktuar. Shpjegojmë këtë menu duke u bazuar tek pamja më poshtë (Fig 45). Kjo menu vlenë vetëm për procesin e kllasterimit, pra nuk ekziston opsioni për rrëshqitje. Shohim se në të majtë, kemi një varg emrash që reprezentojnë atributet e data setit Kaggle. Secili atribut shoqërohet me 5 ngjyra të ndryshme, ku renditja prej lart – poshtë paraqet numrin e kllasterit, që do të thotë se ngjyra e kaltër i përket kllasterit 1, ngjyra e kuqe i përket kllasterit 2, ngjyra e verdhë kllasterit 3, ngjyra e gjelbër kllasterit 4 dhe në fund ngjyra e bardhë i përket kllasterit 5.

Fig 45: Kllasterët në mënyrë grafike

Me të identifikuar kllasterët, fillojmë me analizën e tyre deri sa të nxjerrim të dhëna të shëndosha dhe praktike për secilin grupim. Rikujtojmë se Dota 2 është një video lojë ku lojtarët janë të ndarë në role të caktuara, diçka që edhe ne do të stimulojmë të nxjerrim varësisht atributeve.

Po fillojmë me kllasterin 1, duke vazhduar tutje, duke krijuar një tabelë për secilin anëtarë

#### Kllasteri me ngjyrë të kaltër - *Mbështetësit* *mjekues*

Nga grafiku, shohim se ky grupim shquhet me shumë pak vrasje (*kills*), pak para (*gold*), shumë vdekje (*deaths*), pak para për minutë (*gold per min*) dhe shumë asistime (*assists*). Me këto statistika, konstatojmë se ky grupim nuk është prej vrasësve, pasi vrasësit do të kishin para me anë të së cilave do të furnizoheshin me vegla dhe do të vrisnin hero tjerë, kjo do të thotë që me këto aftësi, numri i vdekje do të ishte më i vogël (pasi vetë mbrojtja do të ishte më e madhe). Me një numër të madhe asistimesh, themi se ky grup i takon më shumë mbështetësve, por le të vazhdojmë me analizim të mëtutjeshëm që saktë të emërojmë këtë grupim. Themi se ky grup ka pak mohime, ju shkakton pak dëme kullave dhe mjekojnë shkëlqyeshëm. Me këto të dhëna themi se ky grup me saktësi i takon mbështetësve pa vazhduar më tutje, pasi krahasimi i këtij grupimi me të tjerë në aspektin e mjekimit ka një avantazh të dukshëm. Pak goditje të fundit dhe një level i ulët vetëm sa i përforcojnë këto që thamë. Konkludojmë se ky grupim i takon mbështetësve, atyre që sakrifikojnë veten për një rol tjetër.

Do e quajmë këtë grup *Mbështetësit* *mjekues* pasi karakterizohen dukshëm me një atribut të ndryshëm. Duhet cekur se mbështetësit mund të mos jenë mjekues tërë kohën, në lojë mund të mbështetesh një hero edhe pa e mjekuar.

#### Kllasteri me ngjyrë të kuqe - *Mbështetësit e dorës së dytë*

Me pak vrasje, pak para dhe një numër konsiderueshëm të lartë të vdekjeve, themi se as ky grup nuk është një që kryen vrasje. Megjithatë, duke u bazuar në atribute, ky grup akoma mund t'i takoj një grupi që shtynë kulla, që mbështesin apo që janë të parët në linjë. Vazhdojmë me një numër të vogël të eksperiencës për minutë dhe një numër të madh asistimesh. Pak mohime dhe pak shtyrje kullash. Përjashtohen mundësitë që grupimi ti takoj shtyrësve të kullave. Mbetet të shikohet se a është grup që janë të parët në linjë apo një mbështetës por jo mjekues. Vazhdojmë me atributet dhe shohim se ky grupim mjekon në një sasi të caktuar dhe shkakton dëme heronjve të tjerë në një masë. Ka pak goditje përfundimtare dhe pak nivele. Me këto të dhëna, themi se edhe ky grup i takon mbështetësve, por në këtë rast jo mjekuesve, por *Mbështetësve të dorës së dytë*.

#### Kllasteri me ngjyrë të verdhë – *Të parët në linjë*

Ky grupim shquhet me vrasje mesatare, dhe para mbi mesataren, pak numër vdekjesh dhe eksperiencë relativisht të madhe. Me shumë pak asistime, shumë mohime dhe shtyrje kullash të madhe, por pak mjekim. Me këto informacione, bie teoria se ky grupim mund t'i takojë ndonjë grupi tjetër të mbështetësve, pasi ato shquheshin për mjekim apo asistime dhe relativisht pak para dhe eksperiencë. Ekzistojnë akoma të hapura mundësitë që ky grup ti takojë vrasësve, shtyrësve të kullave apo të parëve në linjë. Vazhdojmë me atributet tjera; Pak dëmtime heronjve të tjerë dhe shumë goditje përfundimtare, dhe nivel të lartë. Me të thëna këto, emri i këtij grupi mund të jetë *Të parët në linjë.* Bazuar nga eksperienca në lojë, këto heronj, përveç këtyre atributeve, shquhen për lojë të vetmuar (pak asistime) dhe shtyjnë kulla por jo më së shumti, megjithëse roli i tyre e kërkon këtë.

#### Kllasteri me ngjyrë të gjelbër – *Vrasësit* */* *Playmakerët*

Numër i madh vrasjesh, numër i parave dhe vdekjeve janë karakteristikat që e dallojnë këtë grupim për të vazhduar me eksperiencë të lartë dhe numër të madh asistimesh. Një eksperiencë e lartë dhe një numër i madh asistimesh nuk janë diçka që i karakterizon mbështetësit, andaj ky opsion nxirret nga loja. Shumë mohime dhe konsiderueshëm dëme kullave por jo mjekim. Vazhdojmë me dëmtim masiv ndaj heronjve dhe numër të lartë të goditjeve përfundimtare dhe nivel shumë të lartë. Ne mund të identifikojmë këtë grupim me emrin *Vrasës* apo *Playmaker* për disa arsye. Vrasës pasi i ka të thepisura atributet që identifikojnë këtë emër, bashkë me paratë dhe eksperiencën. Ndërsa *Playmaker* pasi ka numër të madh vdekjesh dhe nuk shquhet për shtyrje masive kullash. Numri i asistimeve e ndihmon këtë emërtim që i kemi vënë e edhe niveli i lartë.

#### Kllasterimi me ngjyrë të bardhë – *Ndjekësit e objektivave*

Me një numër të lartë të vrasjeve, një numër maksimalisht të lartë të parave dhe numër të vogël të vdekjeve, shohim se ky grupim i të dhënave është unik, pasi për secilin atribut të lartcekur ka një marzhë ideale. Me shumë eksperiencë dhe numër mesatarë të asistimeve, themi se ky grup definitivisht nuk i takon mbështetësve. Vazhdojmë me një numër të madh mohimesh dhe numër maksimal dëmesh të kullave, shumë pak mjekim dhe shumë dëmtime heronjve. Shumë goditje përfundimtare dhe maksimalisht nivele. Ekzistojnë shumë emërime që mund të ketë ky atribut, por nuk mund të grupohet tek të parët në linjë pasi ka numër të madh të vrasjeve dhe dëme heronjve. Mund të futet në kategorinë e vrasësve por as ky emërim nuk e identifikon këtë kategori në mënyrë ideale pase ka shtyrje maksimale ndaj kullave. Ky kllaster ka të dhëna ideale për secilin atribut, të dhëna që fitohen vetëm nëse loja luhet në mënyrë perfekte dhe ndiqen objektivat në mënyrë të saktë, prandaj do e quajmë *Ndjekësit e objektivave.*

Nëse bëjmë një përmbledhje të këtyre grupimeve dhe emrave, themi se:

|  |  |  |
| --- | --- | --- |
| *Kllasteri* | *Ngjyra* | *Emri* |
| I | E kaltër | *Mbështetësit* *mjekues* |
| II | E kuqe | *Mbështetësit e dorës së dytë* |
| III | E verdhë | *Të parët në linjë* |
| IV | E gjelbër | *Vrasësit* */* *Playmakerët* |
| V | E bardhë | *Ndjekësit e objektivave* |

### Grupimi i Kllasterëve

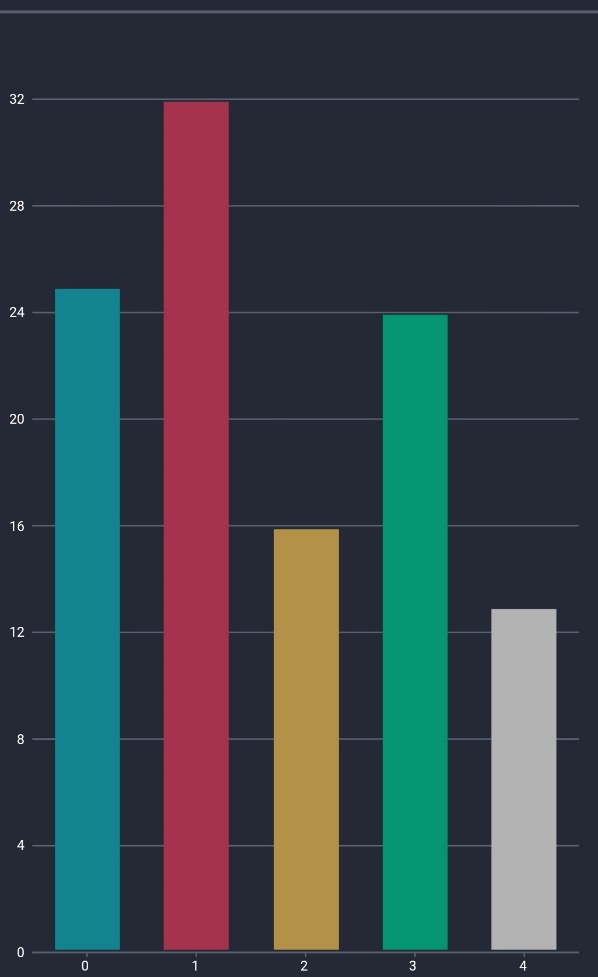
Në fund, në mënyrë që të shohim disa statistika bazike për kllasterimin e realizuar më lartë, kemi ekranin Grupimi i Kllasterëve (Fig 46). Ky ekran mundësohet nga API *getClusterCount* që në princip paraqet një numërues të thjeshtë. Nga figura shohim se në:

Fig 46: Grupimi i kllasterëve

* Kllasterin numër 0, të quajtur *Mbështetësit* *mjekues*, janë grupuar rreth 25 vlera (me ngjyrë të gjelbër të hapur).
* Kllasterin numër 1, të quajtur *Mbështetësit e dorës së dytë*, janë grupuar rreth 32 vlera (me ngjyrë të kuqe).
* Kllasterin numër 2, të quajtur *Të parët në linjë*, janë grupuar rreth 16 vlera (me ngjyrë të verdhë).
* Kllasterin numër 3, të quajtur *Vrasësit* */* *Playmakerët*, janë grupuar rreth 24 vlera (me ngjyrë të gjelbër)
* Kllasterin numër 4, të quajtur *Ndjekësit e objektivave*, janë grupuar rreth 13 vlera (me ngjyrë të bardhë)

# Lista e thirrjeve fundore të mundshme

## index

host / index

Parametrat:

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

Tab 1: Pika fundore index

## getColumns

host / getColumns

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

Përshkrimi: kthen një listë të kolonave të datasetit përkatës

Shembull: /getColumns?kind=steam

Tab 2: Pika fundore getColumns

## getSample

host / 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: /getSample?kind=steam&percentage=10

Tab 3: Pika fundore getSample

## getStages

host / getStages

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

Përshkrimi: kthen një varg të fazave nëpër të cilët kanë kaluar të dhënat

Shembull: /getStages?kind=kaggle

Tab 4: Pika fundore getStages

## getCorrelationMatrix

host / getCorrelationMatrix

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

Përshkrimi: kthen një varg numrash që tregojnë ndërlidhjen mes kolonave të datasetit përkatës

Shembull: /getCorrelationMatrix?kind=kaggle

Tab 5: Pika fundore: getCorrelationMatrix

## getGroupAndCount

host / 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: /getGroupAndCount?attribute=xp\_per\_min&partitions=3

Tab 6: Pika fundore getGroupAndCount

## getStages

host / getStats

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

Përshkrimi: kthen sasinë e rreshtave dhe kolonave të datasetit përkatës

Shembull: /getStats?kind=steam

Tab 7: Pika fundore getStages

## getSchema

host / getSchema

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

Përshkrimi: kthen kolonat dhe tipin përkatës të datasetit të caktuar

Shembull: /getSchema?kind=steam

Tab 8: Pika fundore getSchema

## getDoubleGroup

host / 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: /getDoubleGroup?kind=steam&col1=leaver\_status& col2=radiant\_win

Tab 9: Pika fundore getDoubleGroup

## getClusterStats

host / getClusterStats

Parametrat:

Përshkrimi: Kthen pikat qendrore të të dhënave për secilin kllaster, pra informata për secilin atribut të data setit

Shembull: /getClusterStats

Tab 10: Pika fundore getClusterStats

## getClusterCount

host / getClusterCount

Parametrat:

Përshkrimi: Kthen numrin e sasisë së të dhënave për secilin kllaster

Shembull: /getClusterCount

Tab 11: Pika fundore getClusterCount

## postCluster

host / getClusterCount

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: /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