DOTA 2: Insights & Analysis

David Nguyen
Luke Watts
Joey Trevino
Lam Dinh Tran

Section 01

What is Dota?

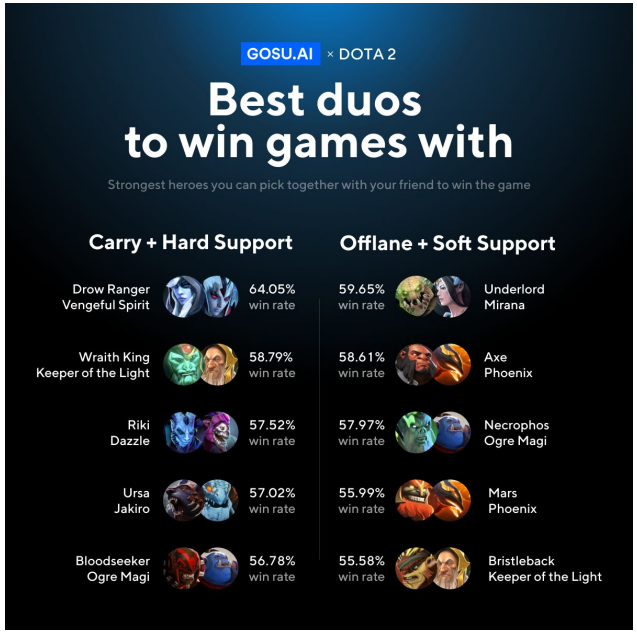
DATASET DESCRIPTION

Source: Kaggle

Finding Heroes Commonly Picked Together: Apriori

Background & Context (Why do we care?)

- Dota: a 5v5 game
- Any advantage we can get is necessary for increasing match win rate
- Certain Hero complement each other -> increases team synergy
- If we want to increase our chance to win, let's observe heroes commonly picked together



Why is Apriori appropriate for this question?

- We want to find heroes that are commonly picked **together**
- This suggests:
  - We will be analyzing ‘market baskets’
  - This case we call these market baskets -> team compositions
- Apriori deals with combinatorial sets
- In our case, each set will be a combination of heroes that appear often together in multiple matches

Implementation

- Gathering Data:
  - Using openDota API: grabbed over 40,000 matches
  - Over 20 csv datasets in total each over 700 MB in size
- Data Preparation:
  - Majority of time was data preparation
  - Analyzing which columns are necessary for analysis
  - Each hero is mapped to a unique ID
- Results:
  - Used apriori library: efficient-apriori
  - Mapping each hero unique ID to its respective in-game name
- Complications:
  - Reading massive csv files to Pandas dataframe created hardware limitations
  - Exploring different Jupyter development environments capable of running analysis in reasonable time frame

APRIORI

-An algorithm behind
"You may also like"

Results (Synergy and Counter-picks)

- Interpretation
  - The hero pudge synergizes with lion if they're on same team, but also strong against lion
  - Phantom Assassin is a strong counter pick against pudge
  - Pudge is one of Ogre Magi's counters

- npc_dota_hero_lion -> npc_dota_hero_pudge
- npc_dota_hero_pudge -> npc_dota_hero_lion
- npc_dota_hero_phantom_assassin -> npc_dota_hero_pudge
- npc_dota_hero_pudge -> npc_dota_hero_phantom_assassin
- npc_dota_hero_ogre_magi -> npc_dota_hero_pudge
- npc_dota_hero_pudge -> npc_dota_hero_ogre_magi

Can we determine game results (win/loss) using team strategy (heroes picked)?

Determining Match Outcome: Classification (C45) Initial Approach

- Have 5 independent variables and have 100+ different values is a bad idea.
- Generated a humongous tree
- Classification process took a really long time due to how big the tree is
- Not a good approach!

```
match_id,hero1,hero2,hero3,hero4,hero5,outcome
-,0,0,0,0,0,2
outcome
0,86,51,83,11,67,True
1,106,102,46,7,73,False
2,7,82,71,39,21,False
3,73,22,5,67,106,True
4,51,109,9,41,27,False
5,38,7,10,12,85,True
6,50,44,32,26,39,False
7,78,19,31,40,47,True
8,8,39,55,87,69,True
9,101,100,22,67,21,False
10,32,27,67,106,71,True
11,28,107,62,84,11,False
12,20,8,67,69,100,True
13,30,41,102,85,11,False
14,53,44,96,14,74,True
15,93,30,35,21,26,False
16,44,28,85,22,30,False
17,71,93,46,18,39,True
18,58,28,44,86,39,False
19,36,26,74,48,14,True
20,35,50,44,69,29,True
21,14,103,72,56,30,False
22,94,28,3,11,50,False
23,40,75,46,96,104,True
```

C45 modification

- Sparse vector of 1s and 0s
- Have 114 independent variables (since there are 114 heroes)
- 1s being classified as “picked hero”

DIFFERENCES COMPARED TO THE PREVIOUS VERSION?

- Have 100+ attributes instead of only 5
- Each column has only 2 distinct values (1s or 0s)
- Tree is much smaller

Problem:

- Too many attributes so runtime takes too long! (90 minutes)

Prediction Final Version

- Reduce the number of attributes to almost half!
  - Determine the rate that a specific hero is picked
  - Aggregate all the low percentage into one singular attribute category

- Able to reduce from 110+ attributes down to only 65
- Allows tree to be generated a lot faster
- Reduce overfitting!

```
match_id,hero1,hero2,hero5,hero7,hero8,hero9,hero11,hero12,hero14,hero16,hero18,hero19,hero20,hero21,hero22,hero25,hero26,hero27,hero28,h
-1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,5,2
outcome
0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,True
1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,False
2,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,False
3,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,True
4,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,False
5,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,2,True
6,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,1,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,False
7,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,True
8,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,True
9,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,False
10,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,True
11,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,False
12,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,True
13,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,False
14,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,True
15,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,False
16,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,False
17,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,True
```

Classified outcome (threshold = .001)

Predicted	False	True	All
Actual			
False	42366	7634	50000
True	39083	10917	50000
All	81449	18551	100000

Total Number Of Records Classified: 100000
Total Number Of Records Correctly Classified: 53283
Total number of records Incorrectly Classified: 46717
Overall accuracy and error rate of the classifier: 53.283 %

Predicted	False	True	All
Actual			
False	42856	7144	50000
True	39095	10905	50000
All	81951	18049	100000

Total Number Of Records Classified: 100000
Total Number Of Records Correctly Classified: 53761
Total number of records Incorrectly Classified: 46239
Overall accuracy and error rate of the classifier: 53.761 %

Another approach to determining match outcome?

Get total MMR of team and higher MMR team will most likely to win

```
(.venv) lamofgod@lams-mbp C45 % python3 players_rating_prediction.py
Predicted  False   True    All
Actual
False      12339  11718  24057
True       12757  13186  25943
All        25096  24904  50000
Total Number Of Records Classified:  50000
Total Number Of Records Correctly Classified:  25525
Total number of records Incorrectly Classified:  24475
Overall accuracy and error rate of the classifier:  51.05 %
```

Using Player Skill (abilities) to Classify Game Outcome

Classify matches using chosen abilities

unit_order_train_ability	count	percentage
25.0	43404	8.682068
16.0	41998	8.400827
17.0	41759	8.353020
18.0	41576	8.316414
19.0	39149	7.830943
15.0	36387	7.278463
20.0	36071	7.215253
14.0	33196	6.640169
21.0	31574	6.315722
22.0	27029	5.406589
13.0	24639	4.928520
11.0	23694	4.739492
23.0	22485	4.497657
24.0	18087	3.617928
12.0	13216	2.643586

	match_id	group-0	group-1	group-2	group-3	group-4	unpicked	dire	win
0	0	1.0	1.0	1.0	0.0	0.0	0	0.0	1.0
1	1	1.0	1.0	1.0	0.0	0.0	0	0.0	0.0
2	2	1.0	1.0	1.0	0.0	0.0	0	0.0	0.0
3	3	1.0	0.0	1.0	0.0	0.0	0	0.0	0.0
4	4	1.0	1.0	1.0	0.0	0.0	0	0.0	1.0
...	...	...	...	...	...	...	...	...	...
99995	49995	1.0	0.0	1.0	0.0	0.0	0	1.0	0.0
99996	49996	0.0	0.0	1.0	1.0	1.0	0	1.0	0.0
99997	49997	1.0	1.0	0.0	0.0	0.0	0	1.0	0.0
99998	49998	1.0	1.0	1.0	0.0	0.0	0	1.0	0.0
99999	49999	1.0	1.0	0.0	0.0	0.0	0	1.0	1.0

- Group abilities together to reduce column count
- Abilities hardly chosen grouped together
- Make predictions solely on these groupings

```
Accuracy rf: 0.5618857142857143
Accuracy knn: 0.5365714285714286
```

Visualizing Heroes: Clustering

Using in-game statistics, can we produce clusters of heroes (playable characters) that tell us more about those heroes' playstyles?

A Look at Player Data

50k games of 5v5, each player plays a unique hero from a pool of 110

```
[ ] players.columns

Index(['match_id', 'account_id', 'hero_id', 'player_slot', 'gold',
       'gold_spent', 'gold_per_min', 'xp_per_min', 'kills', 'deaths',
       'assists', 'denies', 'last_hits', 'stuns', 'hero_damage',
       'hero_healing', 'tower_damage', 'item_0', 'item_1', 'item_2', 'item_3',
       'item_4', 'item_5', 'level', 'leaver_status', 'xp_hero', 'xp_creep',
       'xp_roshan', 'xp_other', 'gold_other', 'gold_death', 'gold_buyback',
       'gold_abandon', 'gold_sell', 'gold_destroying_structure',
       'gold_killing_heros', 'gold_killing_creeps', 'gold_killing_roshan',
       'gold_killing_couriers', 'unit_order_none',
       'unit_order_move_to_position', 'unit_order_move_to_target',
       'unit_order_attack_move', 'unit_order_attack_target',
       'unit_order_cast_position', 'unit_order_cast_target',
       'unit_order_cast_target_tree', 'unit_order_cast_no_target',
       'unit_order_cast_toggle', 'unit_order_hold_position',
       'unit_order_train_ability', 'unit_order_drop_item',
       'unit_order_give_item', 'unit_order_pickup_item',
       'unit_order_pickup_rune', 'unit_order_purchase_item',
       'unit_order_sell_item', 'unit_order_disassemble_item',
       'unit_order_move_item', 'unit_order_cast_toggle_auto',
       'unit_order_stop', 'unit_order_taunt', 'unit_order_buyback',
       'unit_order_glyph', 'unit_order_eject_item_from_stash',
       'unit_order_cast_rune', 'unit_order_ping_ability',
       'unit_order_move_to_direction', 'unit_order_patrol',
       'unit_order_vector_target_position', 'unit_order_radar',
       'unit_order_set_item_combine_lock', 'unit_order_continue'],
      dtype='object')
```

	match_id	account_id	hero_id	player_slot	gold	gold_spent	gold_per_min	xp_per_
0	0	0	86	0	3261	10960	347	
1	0	1	51	1	2954	17760	494	
2	0	0	83	2	110	12195	350	
3	0	2	11	3	1179	22505	599	
4	0	3	67	4	3307	23825	613	
...	...	...	...	...	...	...	...	
499995	49999	0	100	128	2718	17735	468	
499996	49999	0	9	129	3755	20815	507	
499997	49999	0	90	130	1059	16225	371	
499998	49999	0	73	131	3165	31015	780	
499999	49999	158360	53	132	2972	21195	520	

500000 rows × 73 columns

Identifying Intriguing Attributes

14 Continuous Features of Interest:

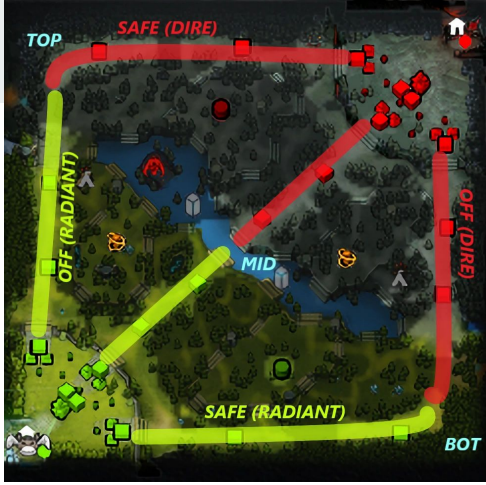
name	hero_id	gold	gold_per_min	xp_per_min	kills	deaths	assists	denies	last_hits	stuns	hero_damage	hero_healing	tower_damage	level	k/d
Anti-Mage	1.0	2392.246275	554.192742	580.392827	7.340996	5.449979	6.722222	10.741699	298.673585	4.774384	10162.878033	135.156769	3081.401767	20.574819	2.295656
Axe	2.0	1682.507933	400.109107	442.938274	8.525321	9.365138	10.498153	1.327755	136.269724	0.171975	11945.333188	14.217779	453.616605	18.033254	1.258256
Bane	3.0	1512.149628	295.293380	336.720329	4.766941	7.721112	11.811986	3.867607	32.764591	96.947184	7838.933803	167.141794	381.018801	15.563259	0.920094
Bloodseeker	4.0	1821.146143	447.641746	494.835927	9.657307	9.190122	9.746279	5.939445	165.670162	1.239450	13487.634641	740.771651	1449.624831	19.419824	1.457787
Crystal Maiden	5.0	1532.348713	314.668876	346.943411	4.483686	8.946215	14.027402	1.828830	48.836477	3.011080	8805.037726	142.610375	288.910018	15.789064	0.708599
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
Earth Spirit	107.0	1719.227564	332.400997	375.771011	5.356838	7.667023	15.289530	2.331553	52.301282	64.817438	10249.472222	731.684117	333.606125	16.221154	1.077073
Terrorblade	109.0	2469.381689	488.280464	493.556415	7.531915	7.255964	9.275306	7.647324	212.918117	0.052731	12729.525467	28.301741	3663.286267	18.959381	1.770451
Phoenix	110.0	2073.953120	364.729944	428.591945	6.465170	7.988775	15.667217	4.650050	80.664906	11.759792	13959.607131	1450.888082	457.317927	17.762298	1.208532
Oracle	111.0	1617.788900	305.821606	336.724480	5.663033	7.887017	9.775025	3.813677	40.501487	1.528767	7572.126858	4086.590684	466.964321	15.305253	1.059136
Winter Wyvern	112.0	1737.437573	309.383266	335.962973	3.637911	7.674678	12.228401	2.266078	52.165649	89.919248	6869.851111	313.957646	314.619852	15.408601	0.708318

110 rows × 15 columns

Initial Idea

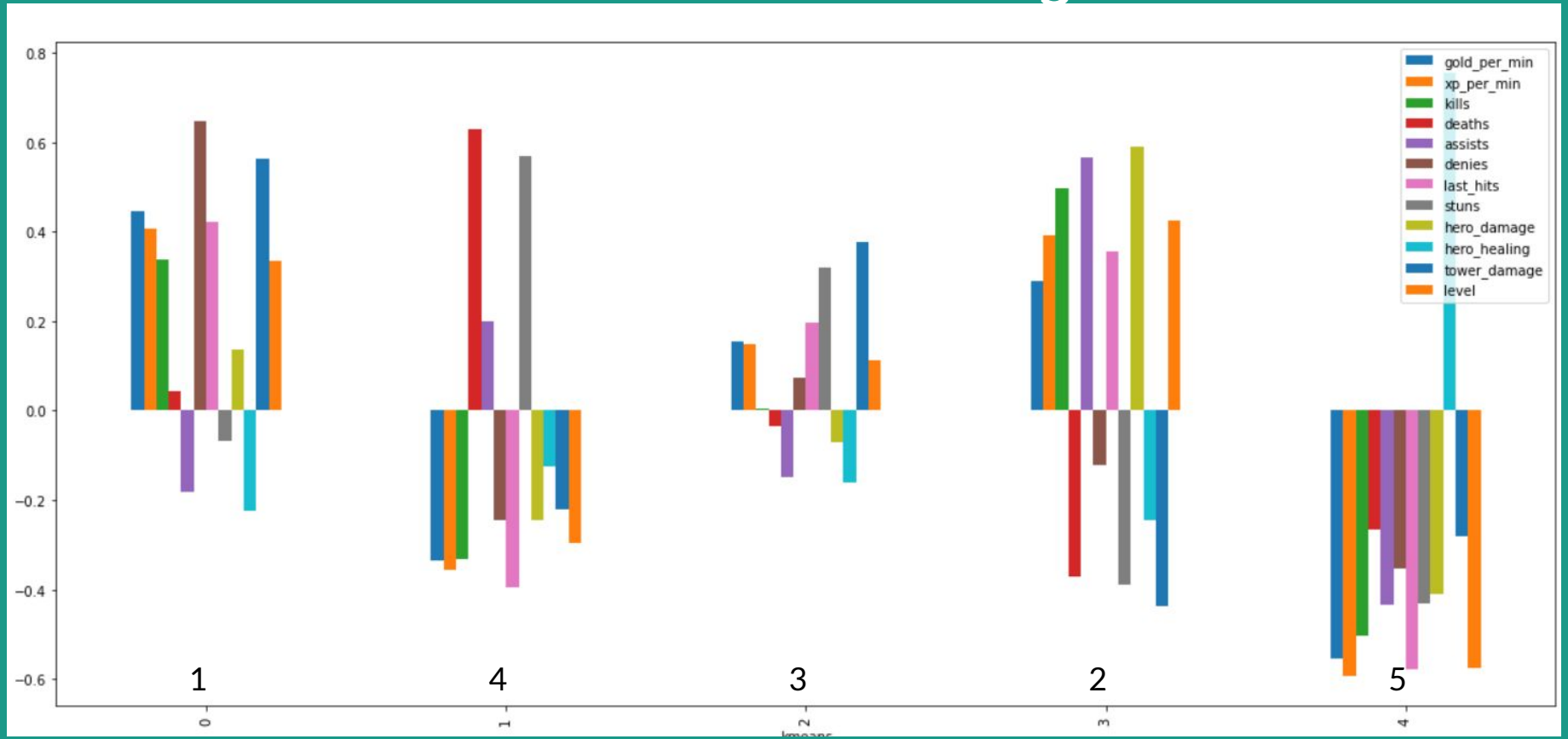
$$Z = \frac{x - \mu}{\sigma}$$

Score, Mean, SD

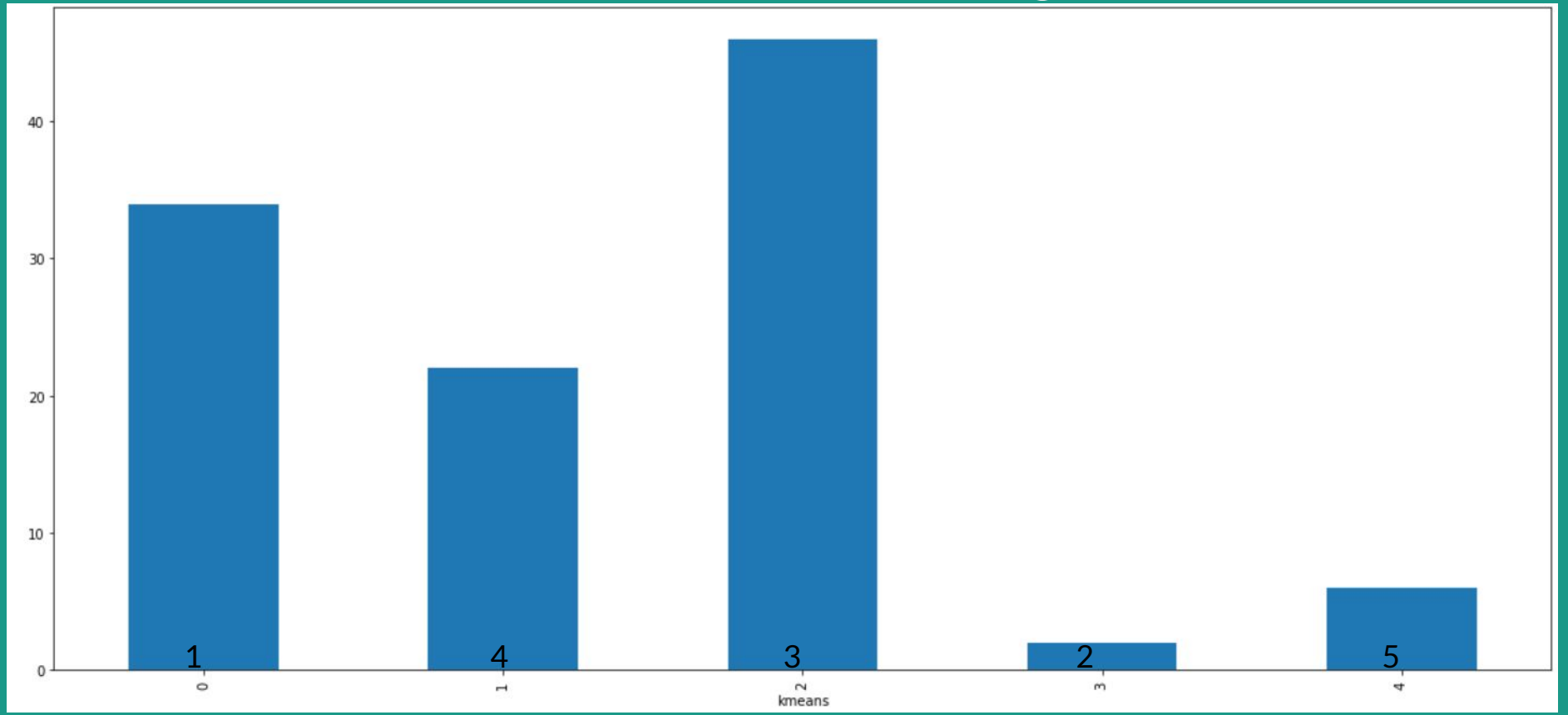


- Group player data by hero_id
  - Get mean average of statistics for each game with said hero
  - Normalize numeric data using z-score
- Compute Clustering Algorithm
  - Identify “meta” positions 1 -5
    - 1 = Hard Carry/Core (usually safelane), needs resources
    - 2 = Early Carry/Core (usually midlane), left alone early
    - 3 = Initiator (usually offlane), harass other teams pos. 1
    - 4 = Soft Support (offlane/roaming), wanders early to assist
    - 5 = Hard Support (safelane), keep pos. 1 alive and happy

First Test Using K Means Clustering with k = 5 (with normalization AFTER clustering)



First Test Using K Means Clustering with k = 5 (with normalization AFTER clustering)



Initial Test

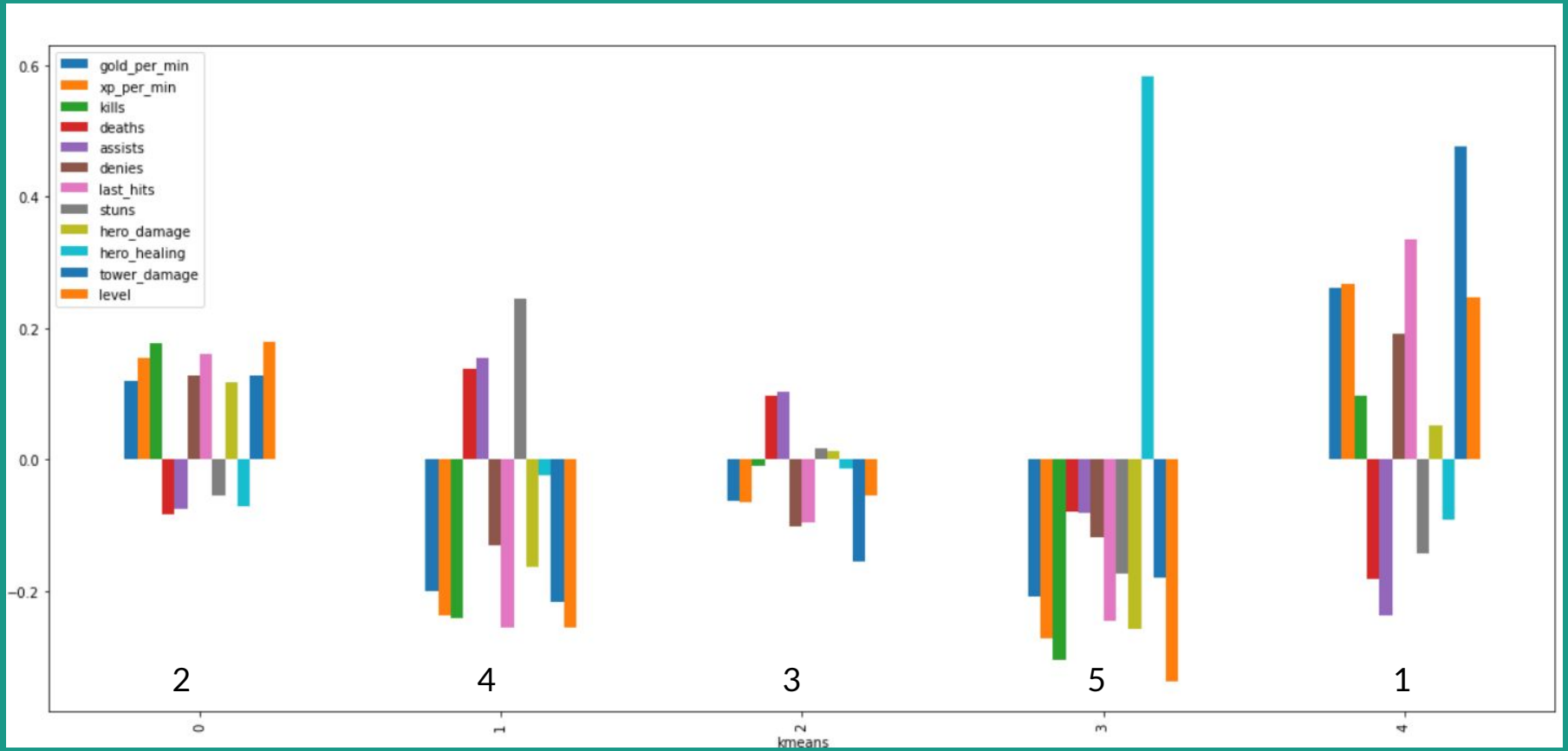
What went well:

- Identifying extremes:
  - Healers
  - High gold/xp carries

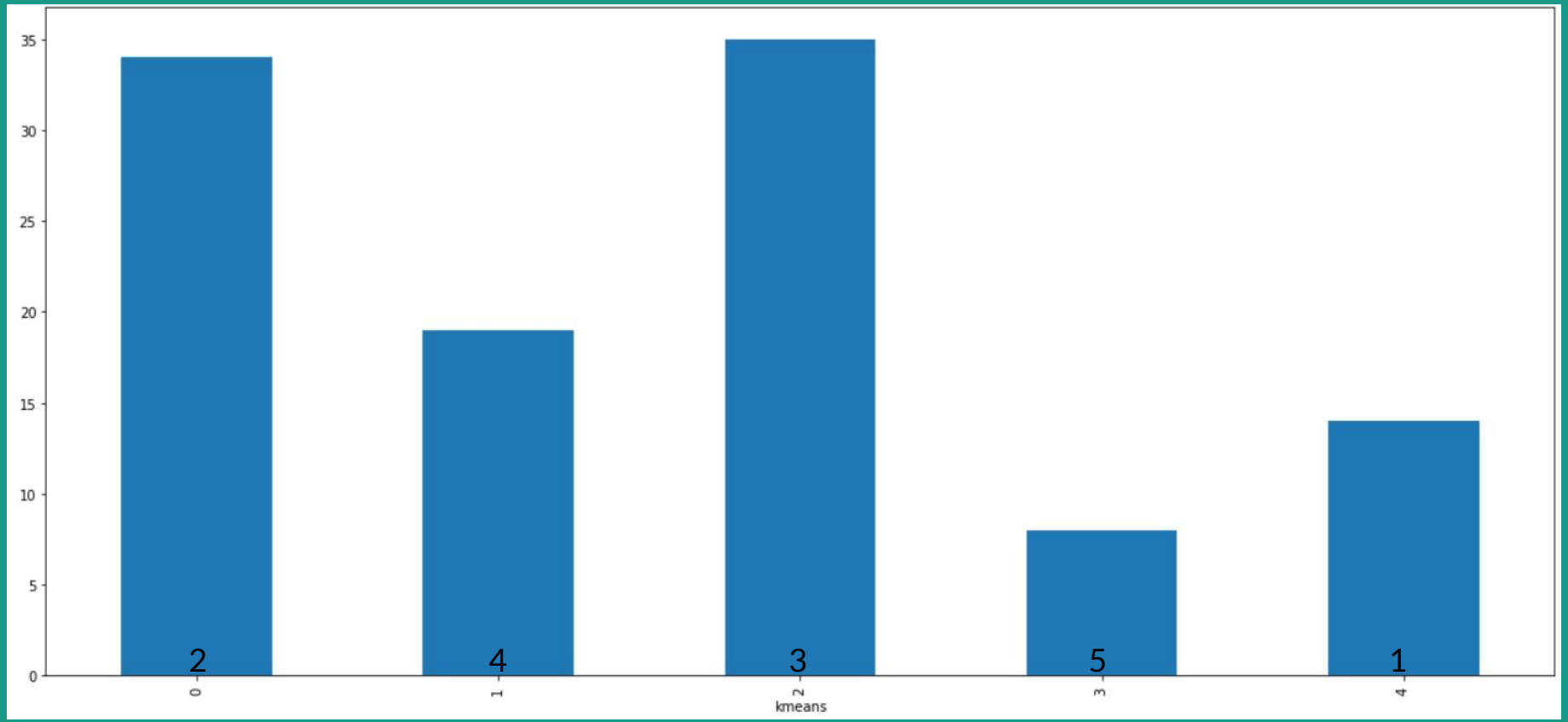
What went not so well:

- Bulked most heroes in the third cluster
  - Didn’t know how to differentiate
- Small group of two heroes
  - Tinker and Zeus

First Test Using K Means Clustering with k = 5 (with normalization BEFORE clustering)



First Test Using K Means Clustering with k = 5 (with normalization BEFORE clustering)



With Normalization Changes

What went well:

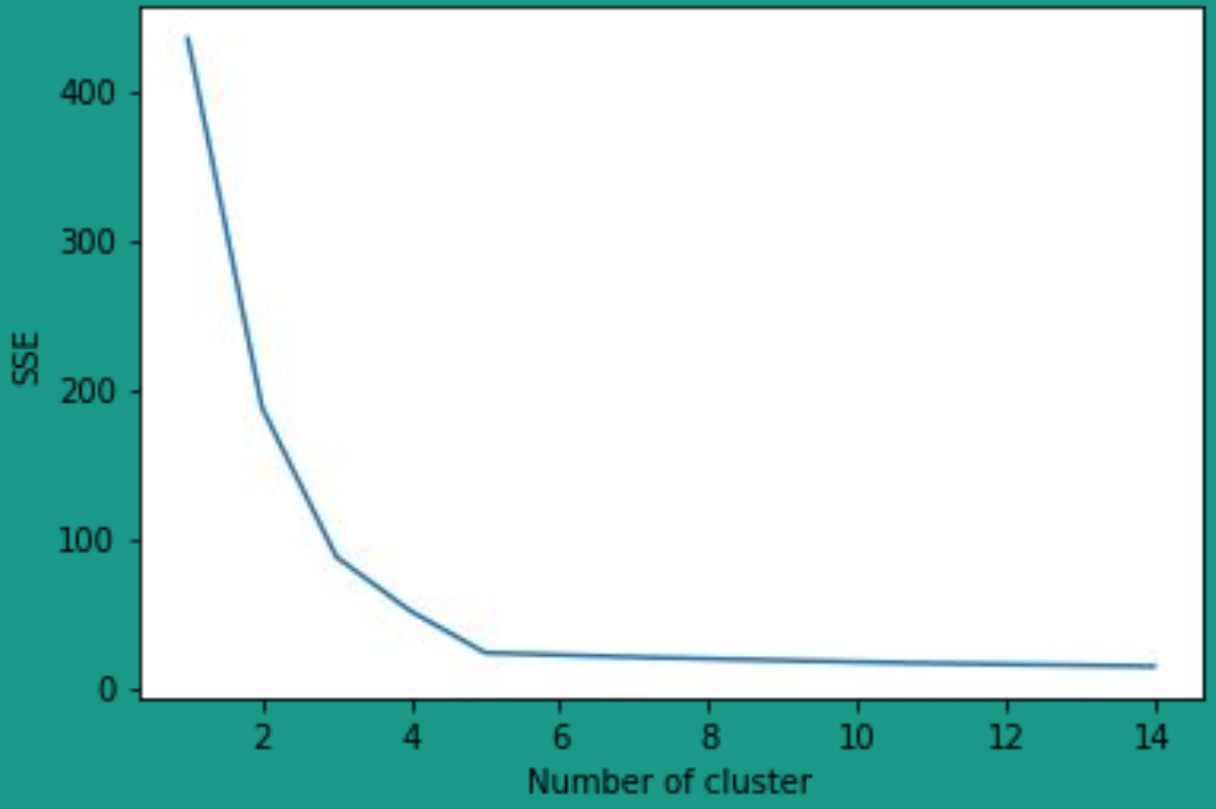
- Identifying extremes:
  - Healers
  - High gold/xp carries
- More balanced clustering

What went not so well:

- 5 clusters didn't seem to represent positions 1-5 exactly
  - Change concept of clusters?
  - Hyper-parameter tuning?

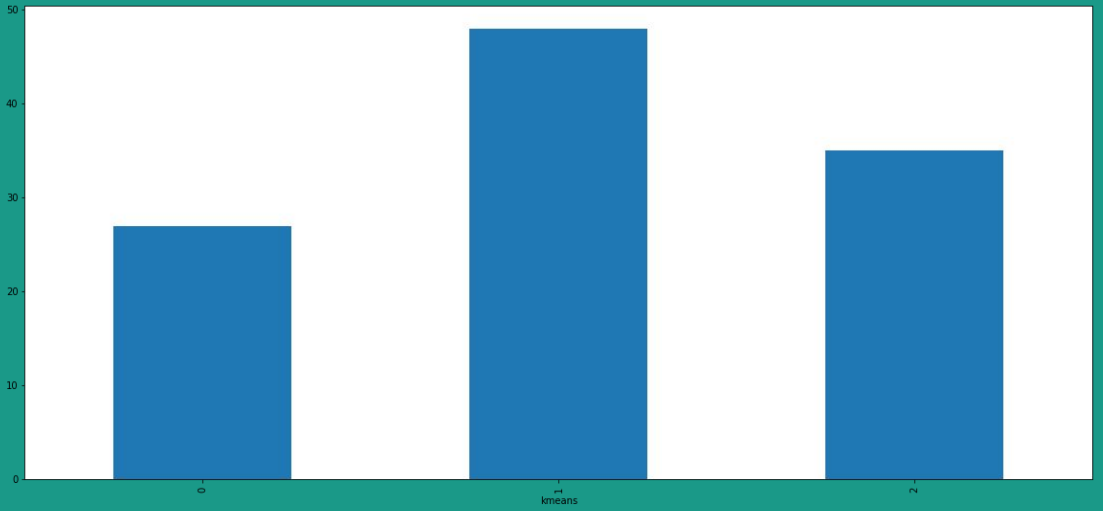
Tuning k with Elbow Criterion

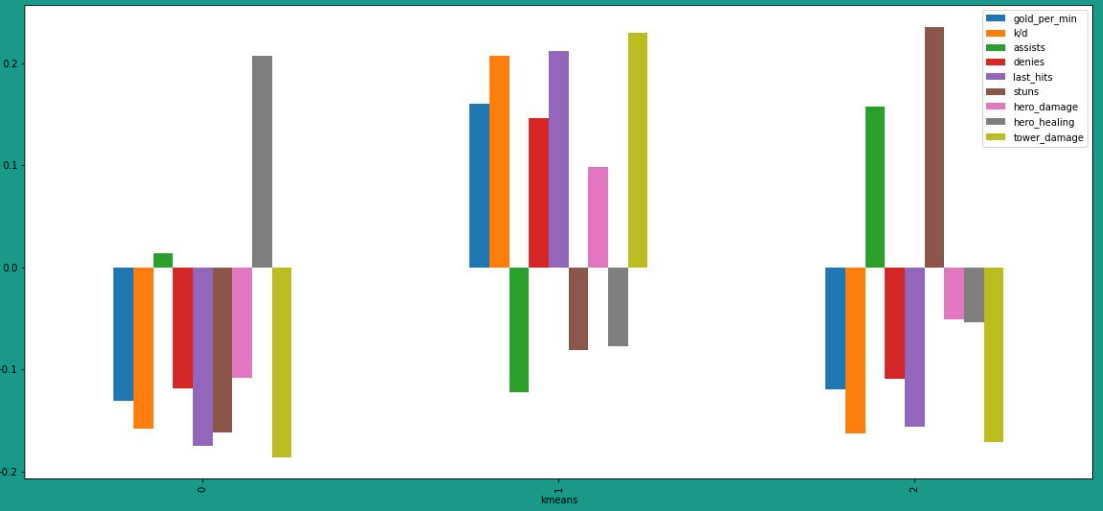
- Graph Inertia vs k
- Inertia: Sum of distances of samples to their closest cluster center
- Ideal k = x-axis at the “elbow” of graph
- k = 3



K Means Clustering
k = 3

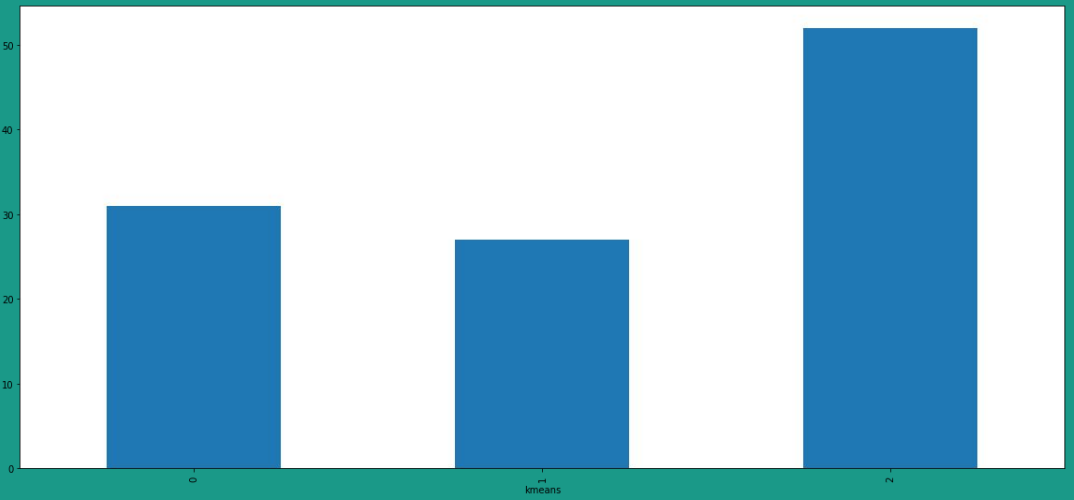
(with normalization BEFORE clustering)

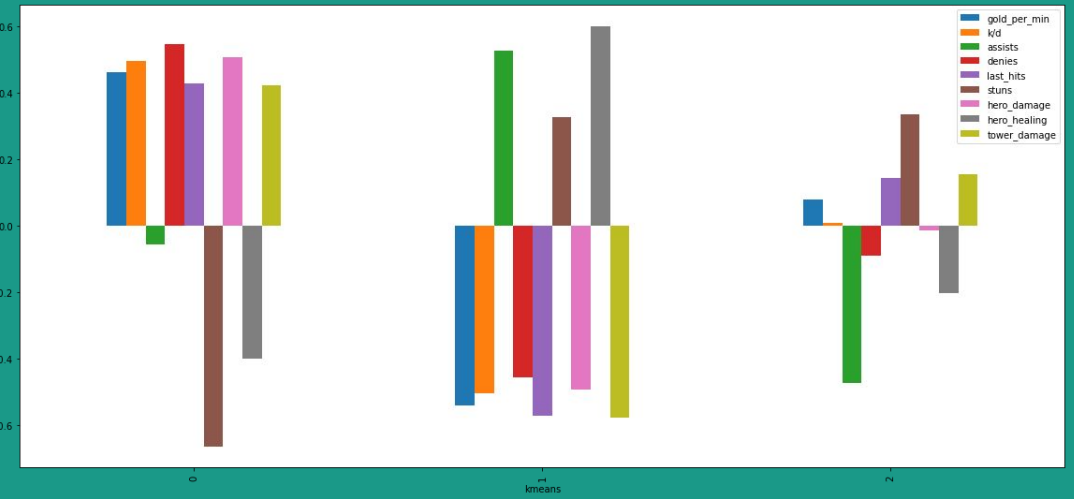




K Means Clustering
k = 3

(with normalization AFTER clustering)

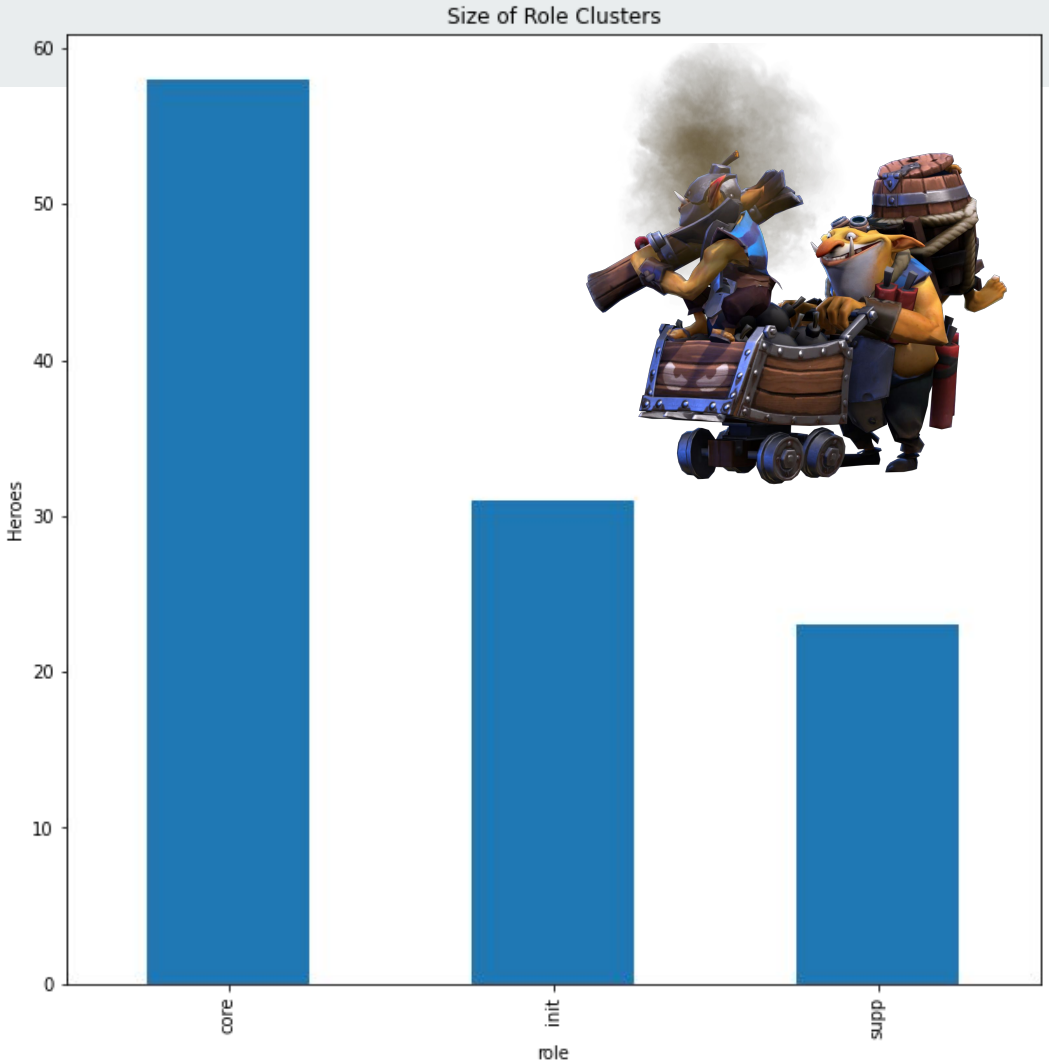




“Ground Truth”

(human judgement)

- Asked 3 friends on Steam to classify all 110 heroes as “core”, “support”, or “initiator”
- Used plurality of their decisions for final clusterings
- “Techies doesn’t belong in DOTA” - Nathan



Confusion Matrices

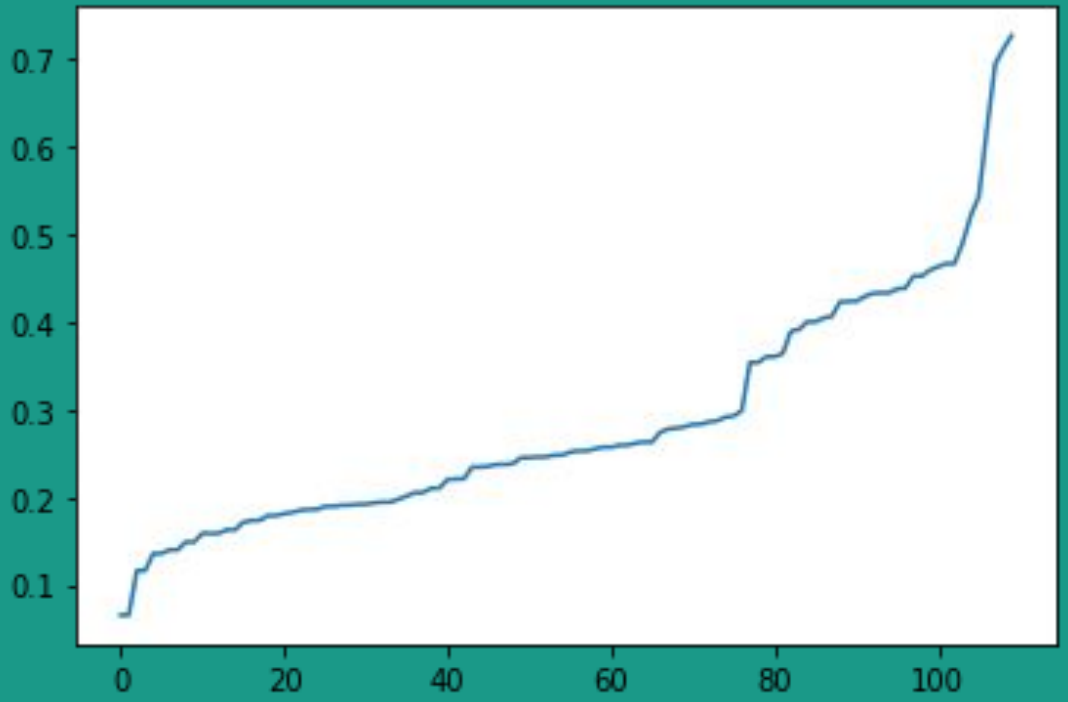
- K = 3 Normalized Before
  - ~81% accurate
  - Cores had 1.00 precision
    - (no FP)
  - Cores were most mistaken with 9 misclassifications (5 Init, 4 Supp)
  - Techies as a Support
- K = 3 Normalized After
  - Only ~ 56% accurate
  - 28 Cores were mistaken for Initiators
  - Supports had highest recall
  - Techies as an Initiator (agreed with ground truth)

```
K = 3, Normalizing Before
Actual (index) vs Predicted (columns)
      core  init  supp
core  48.0   5.0   4.0
init   0.0  24.0   6.0
supp   0.0   6.0  17.0
FP: [ 0. 11. 10.], total: 21.0
FN: [9. 6. 6.], total: 21.0
TP: [48. 24. 17.], total: 89.0
pre: [1.         0.68571429 0.62962963], overall: 0.8090909090909091
rec: [0.84210526 0.8        0.73913043], overall: 0.8090909090909091
f1: [0.91428571 0.73846154 0.68      ], overall: 0.8090909090909091
overall accuracy : 0.8090909090909091


K = 3, Normalizing After
Actual (index) vs Predicted (columns)
      core  init  supp
core  28.0  28.0   1.0
init   1.0  18.0  10.0
supp   2.0   6.0  16.0
FP: [ 3. 34. 11.], total: 48.0
FN: [29. 11.  8.], total: 48.0
TP: [28. 18. 16.], total: 62.0
pre: [0.90322581 0.34615385 0.59259259], overall: 0.5636363636363636
rec: [0.49122807 0.62068966 0.66666667], overall: 0.5636363636363636
f1: [0.63636364 0.44444444 0.62745098], overall: 0.5636363636363636
overall accuracy : 0.5636363636363636
```

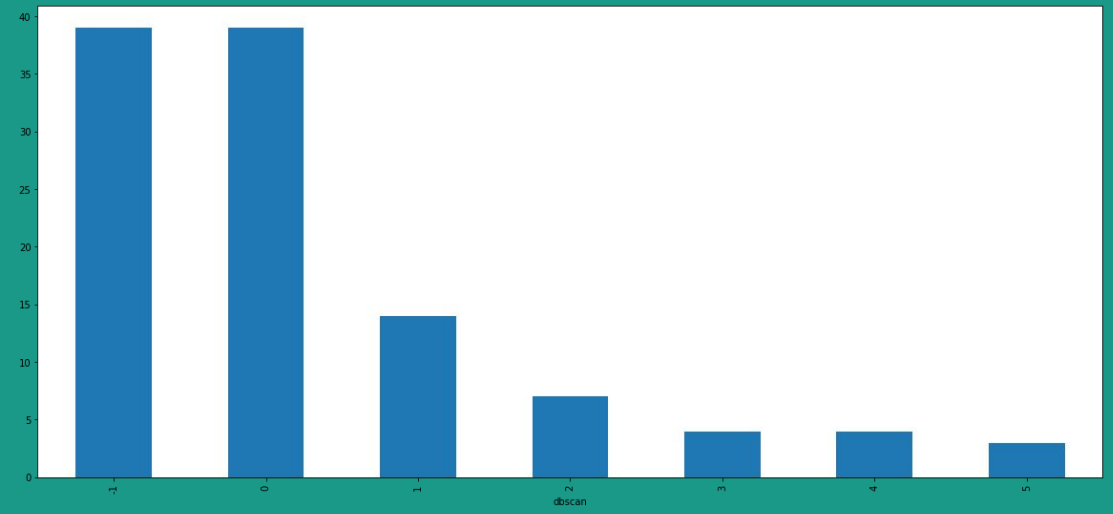
Tuning epsilon with K-Nearest-Neighbors for DBScan

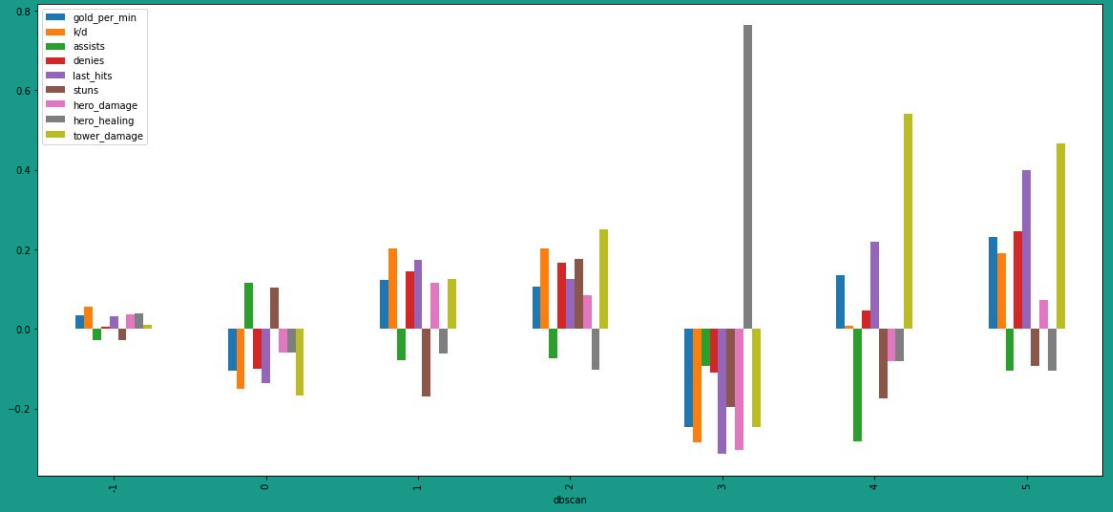
- Use K-Nearest-Neighbors to find distance of points to their nearest neighbors
- Graph all points in ascending distance from nearest neighbors
- Ideal epsilon = y-axis at the “elbow” of graph
- Epsilon = 0.305



DBScan Clustering
epsilon = 0.305
MinPts = 3

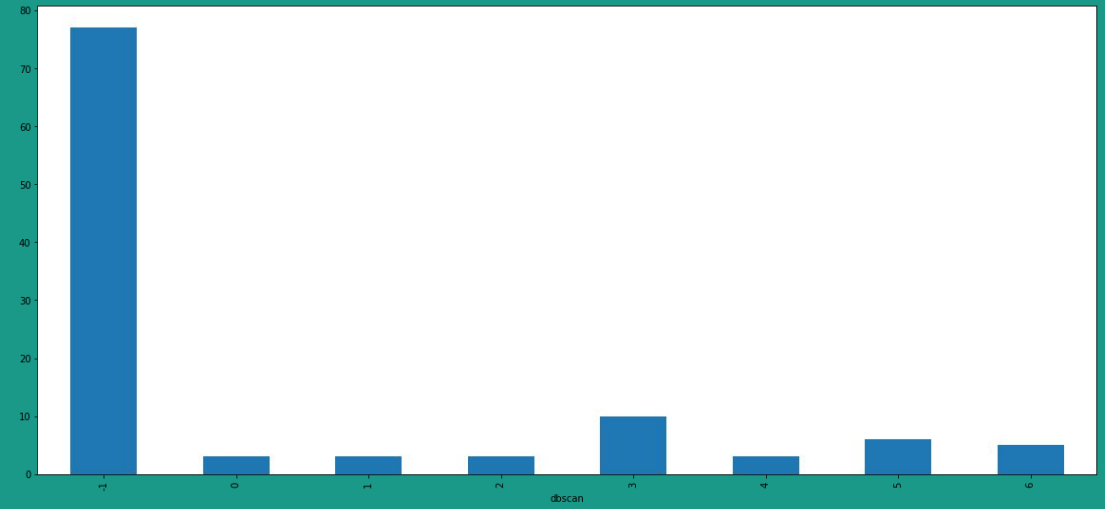
(with normalization BEFORE clustering)

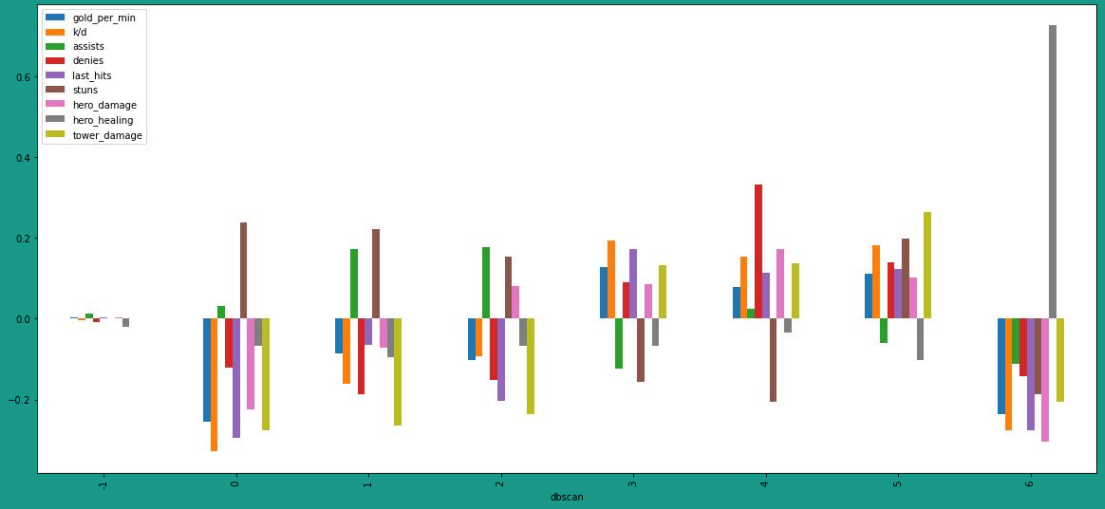




OPTICS Clustering
max epsilon = 0.5
MinPts = 3

(with normalization BEFORE clustering)

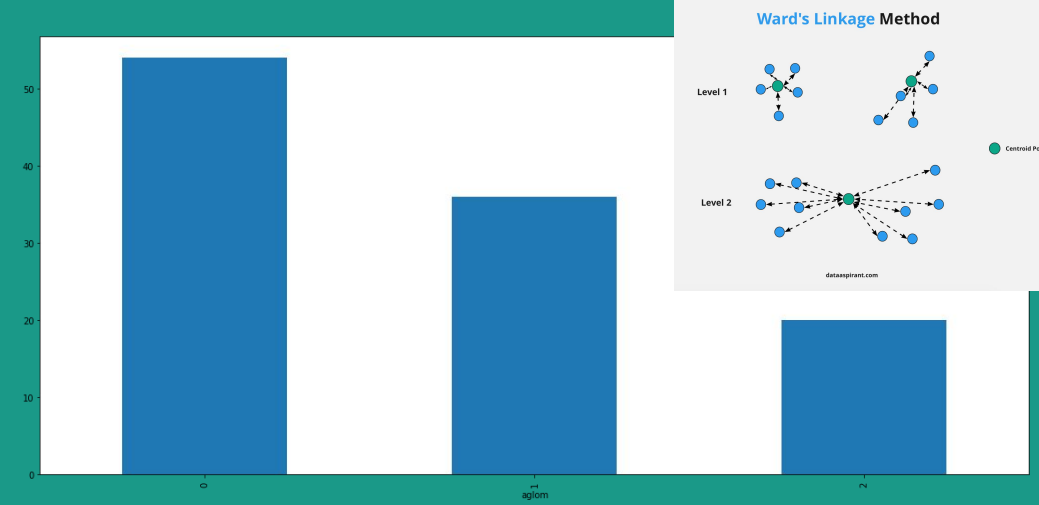


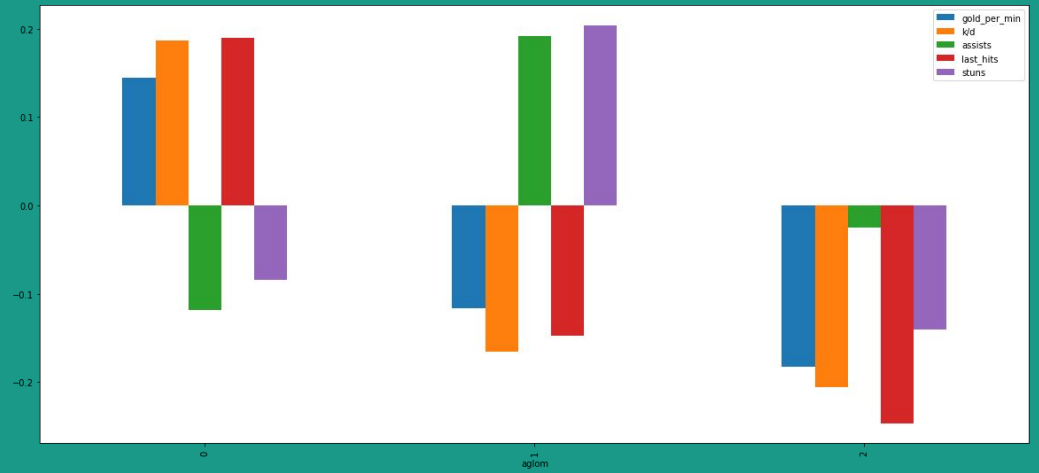


Agglomerative Clustering
n_clusters = 3
linkage = ‘ward’

(with normalization BEFORE clustering)

```
Actual (index) vs Predicted (columns)
      core  init  supp
core  50.0   5.0   2.0
init   1.0  23.0   4.0
supp   3.0   8.0  14.0
FP: [ 4. 13.  6.], total: 23.0
FN: [ 7.  5. 11.], total: 23.0
TP: [50. 23. 14.], total: 87.0
pre: [0.92592593 0.63888889 0.7       ], overall: 0.7909090909090909
rec: [0.87719298 0.82142857 0.56      ], overall: 0.7909090909090909
f1: [0.9009009  0.71875    0.62222222], overall: 0.7909090909090909
overall accuracy : 0.7909090909090909
```





Clustering Summary

- Normalizing data is important for identifying clusters
- K-Means was most effective with Agglomerative close behind
- DBScan/OPTICS can identify outliers but may be overzealous with dataset of sparse clusters
    - Still useful for identifying small, pure clusters
- Viewing heroes by fluid roles rather than strict meta positions will lead to better clustering
    - May be insightful for players drafting heroes