LEAGUE OF LEGENDS RANKED LADDER AND GAME ANALYSIS

Jacopo Mileto - Progetto per il corso di Social Media Mining - A.A. 2022/2023

INTRODUZIONE:

COS’É LEAGUE OF LEGENDS?

1

LEAGUE OF LEGENDS É UN MOBA

Partite 5v5: a ogni giocatore viene assegnato un ruolo che corrisponde alla posizione in cui gioca.: Top laner, Jungle, Mid laner, Adc & Support (entrambi giocano in Bot lane).

2

RANKED LADDER

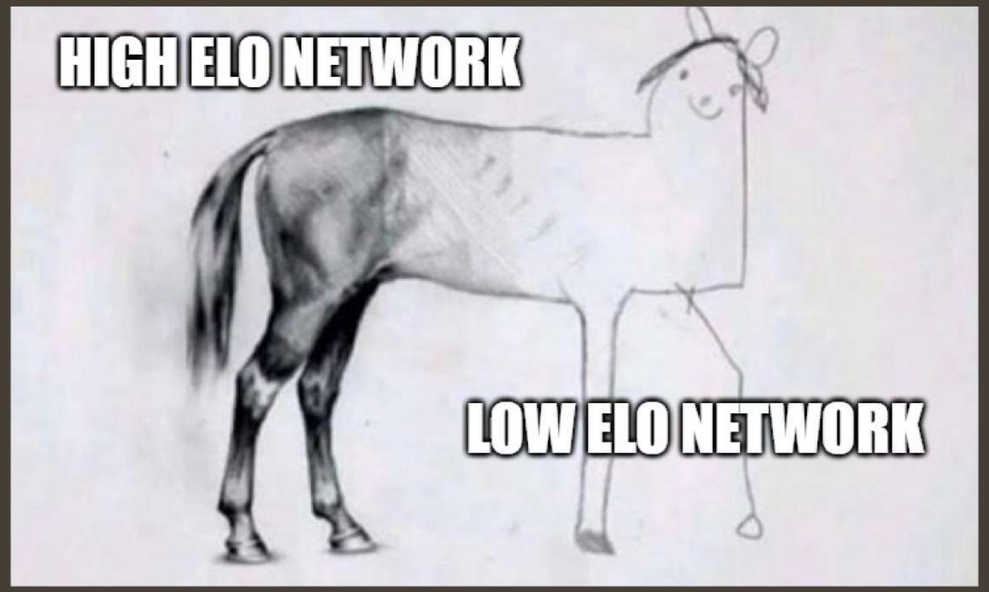
La modalità di gioco classificata è basata su un sistema ELO con diverse leghe e divisioni che i giocatori possono scalare vincendo le partite. Si parte da gradi bassi: Iron, Bronze etc. fino a: Master, Grandmaster e Challenger

3

IL JUNGLER

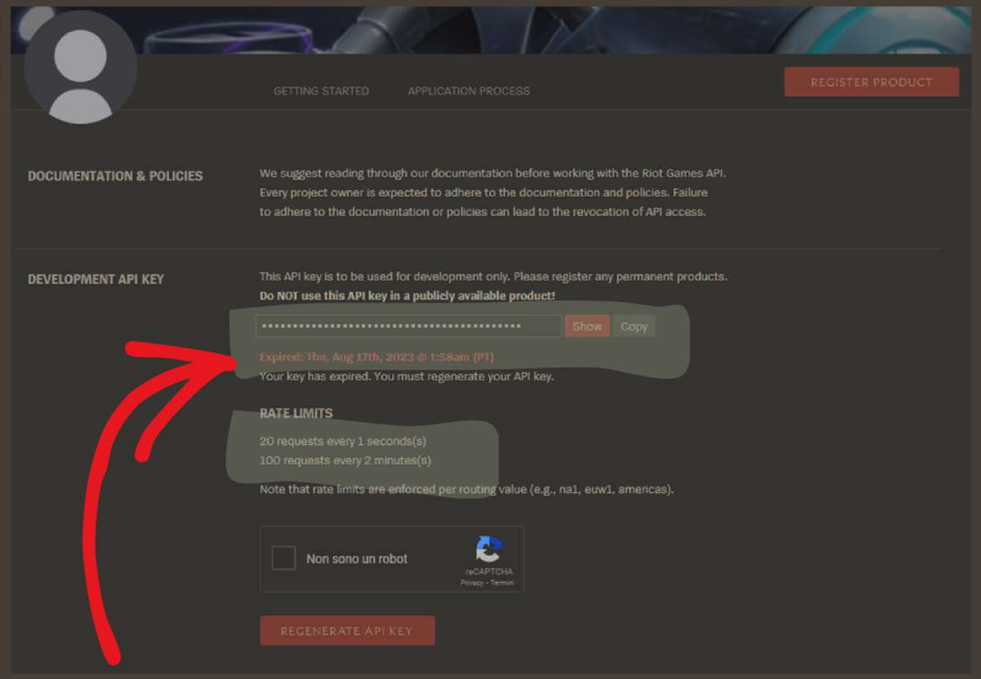
Il ruolo della giungla non è assegnato a una corsia e si aggira per la mappa, raccoglie exp uccidendo i mostri della giungla e il suo obiettivo primario è quello di fungere da supporto alle corsie controllando le zone obiettivo neutrali.

OBIETTIVI

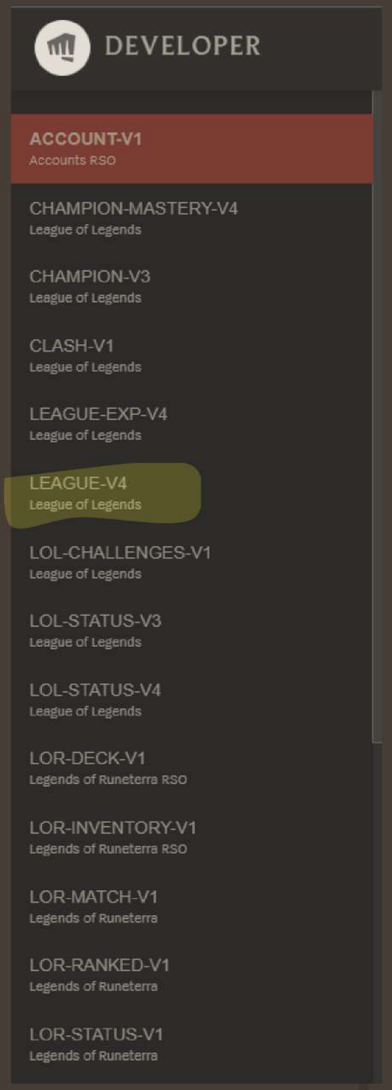


- **01** Mappatura delle partite dei jungler
- **02** Analisi struttura del network
- **03** Estrazione dei dati delle partite e dei giocatori
- **04** Confronto con la rete di basso elo
- **05** Match Prediction

RIOT GAMES APIs



Development key valida per 24h



Data Extraction

Starting player

GET /lol/summoner/v4/summoners/by-name/{summonerName}

Dato il nome dell'evocatore, estraiamo il puuid e il summonerID criptato.

Player info

GET /lol/league/v4/entries/by-summoner/{encryptedSummonerId}

Dal summonerID criptato possiamo estrarre le informazioni sul giocatore relative al suo rango, al suo livello e ai suoi LP.

Player matches

GET /lol/match/v5/matches/by-puuid/{puuid}/ids

Dal puuid si richiedono i match filtrati per: queue, modalità, data inizio, data di fine raccolta.

Match data

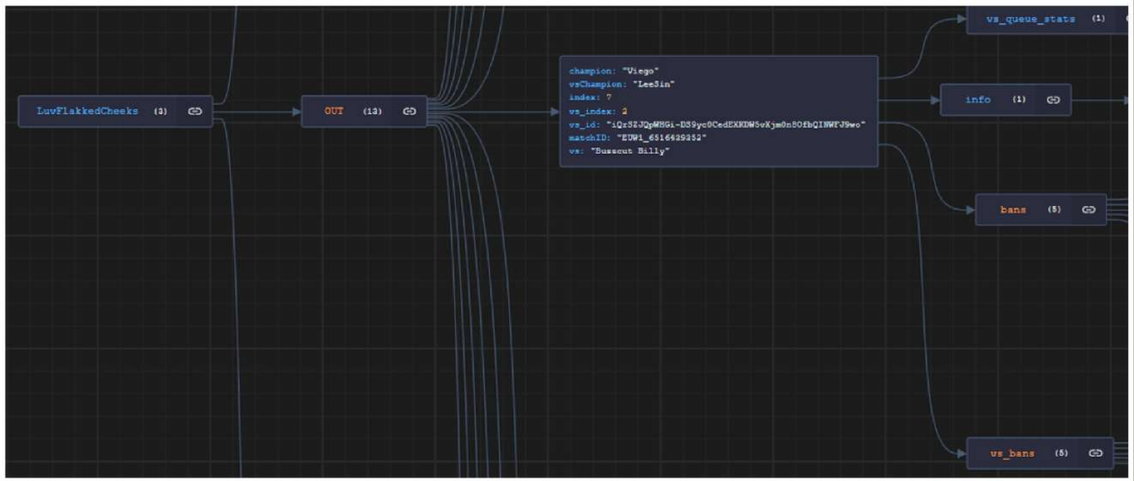
GET /lol/match/v5/matches/{matchId}

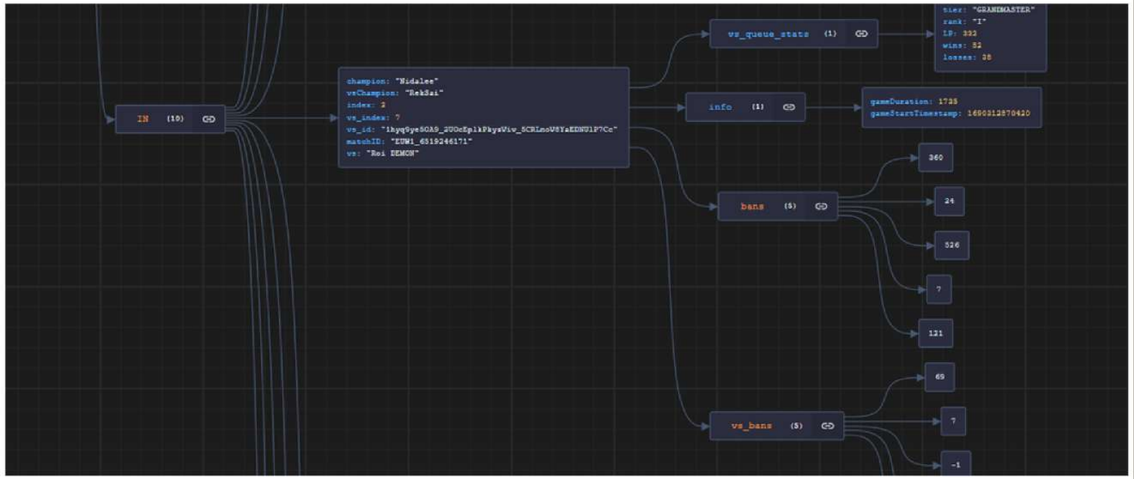
Una volta ottenuto l'elenco delle corrispondenze, possiamo utilizzare il matchID per richiedere informazioni specifiche del match.

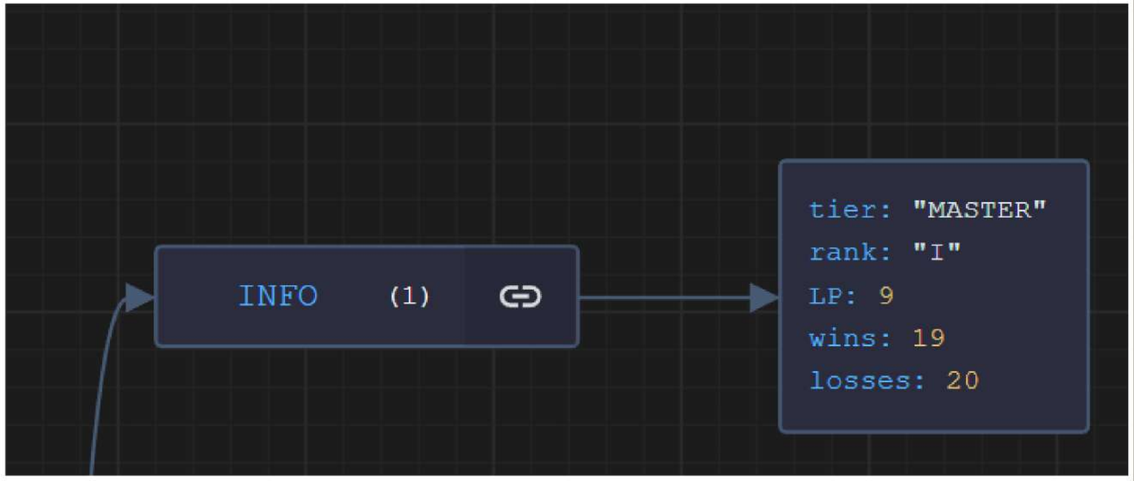
Match list Extraction parameters

```python
def get_matchlist(puuid, start_at):
    try:
        return lol_watcher.match.matchlist_by_puuid(
            "EUROPE",
            puuid,
            queue=420,
            type="ranked",
            start_time=1690101004,
            end_time=1690446600,
            start=start_at,  # dalle 8:30 del 26/07 al 27/07
        )
    except ApiError as err:
        if err.response.status_code == 429:
            print("We should retry in {} seconds.".format(err.headers["Retry-After"]))
            return get_matchlist(puuid)
        else:
            print("errore a questo punto")
            return None
```

```python
while len(account_search_set) > 0:
    print("\n nuova iterazione \n ")
    search_list = list(account_search_set)
    new_players = set()
    for search_account in search_list:
        searched_players.add(search_account)
        # inizializzo il dict e la lista
        player_node[search_account] = {Info: {}, Out: [], In: []}
        # ho lo username, estraggo i dati per ottenere il puuid
        account = get_account(search_account)
        if account is None:
            account_search_set.discard(search_account)
            continue
        puuid = get_puuid(account)
        if puuid is None:
            account_search_set.discard(search_account)
            continue
        encryptedID = account["id"]
        account_queue_stats = get_rankedQ_stats(get_account_info(encryptedID))
        player_node[search_account][Info] = account_queue_stats
        print("cerco i match di: ", search_account, " puuid: ", puuid)
        # ho il puuid estraggo i match giocati
        matches = get_matchlist(puuid=puuid, start_at=0)
        if matches is None:
            account_search_set.discard(search_account)
            continue
        print("matches : \n ", len(matches))
        if len(matches) == 20:
            matches.extend(get_matchlist(puuid=puuid, start_at=20))
        for match in matches:
            # per ogni match estraggo le informazioni per capire se il match è stato vinto
            if match in checked_matches:
                continue
            match_info = get_matchinfo(match)
            if match_info is None:
                account_search_set.discard(search_account)
                continue
            # dalle informazioni estraggo anche i dati fondamentali
            index = match_info["metadata"]["participants"].index(puuid)
            if match_info["info"]["participants"][index]["teamPosition"] != "JUNGLE":
                continue
            match_result = True if is_match_won(match_info, index) else False
            result_is = "WIN" if match_result else "LOSS"
            print("MATCH: ", match, " : ", result_is)
            result = get_opponent_in_role(match_info, index)
            if result is None:
                continue
            playerToExtract, vs_index = result
            champion = get_champion(match_info, index)
            general_info = get_general_info(match_info)
            vs_champion = get_champion(match_info, vs_index)
            bans = get_team_bans(match_info, index)
            vs_bans = get_team_bans(match_info, vs_index)
            vs_queue_stats = get_rankedQ_stats(get_account_info(get_summonerId(match_info,vs_index)))
            key_stats = get_player_key_performance(match_info, index)
            vs_key_stats = get_player_key_performance(match_info, vs_index)
            print("preso le key stats, aggiungo il dict del match: ", match)
            match_dict = {
                match: playerToExtract,
                "vs_queue_stats": vs_queue_stats,
                "champion": champion,
                "info": general_info,
                "vsChampion": vs_champion,
                "index": index,
                "vs_index": vs_index,
                "vs_id": get_summonerId(match_info, vs_index),
                "bans": bans,
                "vs_bans": vs_bans,
                "performance" : key_stats,
                "vs_performance" : vs_key_stats
            }

            print("Player to add: ", playerToExtract)
            new_players.add(playerToExtract)
            # Aggiorno il nodo
            if match_result:
                player_node[search_account][Out].append(match_dict)
            else:
                player_node[search_account][In].append(match_dict)
            time.sleep(0.1)
            progress_checkpoint["Current"] = list(account_search_set)
            #progress_checkpoint["Next"] = list(new_players)
            with open("jungler_network_2.json", "w") as f:
                json.dump(player_node, f, indent=4)

            with open("search_list.json", "w") as t:
                json.dump(progress_checkpoint, t, indent=4)
            checked_matches.append(match)
        account_search_set.discard(search_account)
    new_players.difference_update(
        searched_players
    )  # Rimuovi i giocatori già ricercati da new_players
    account_search_set.update(
        new_players
    )  # Aggiorna account_search_set con i nuovi giocatori
```





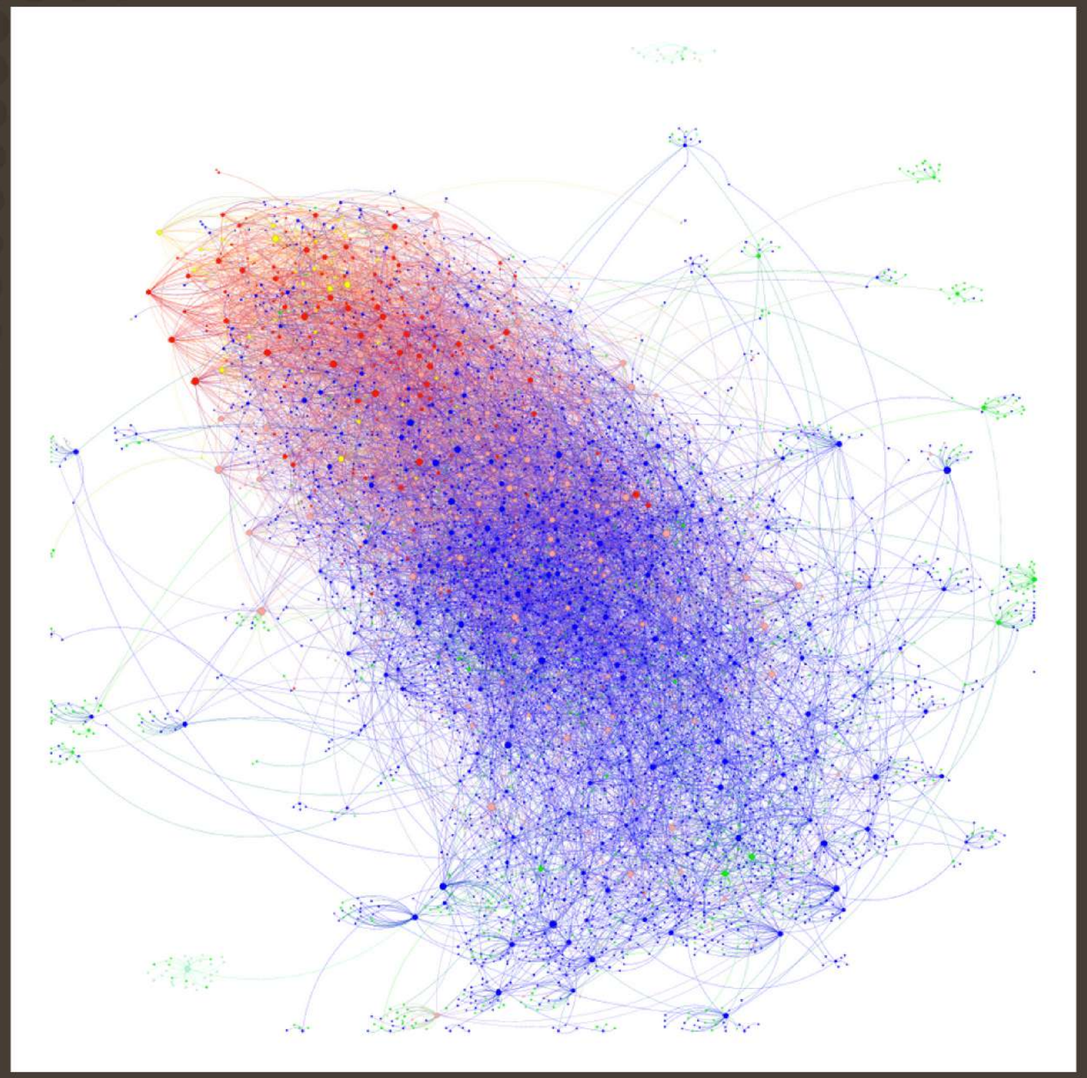


Building the Network's data

Generator Nodes
INFO: GN ranked stats

OUT : win IN: loss

Game stats: GN stats + adversary stats



Extracted Graph

Nodi: 4897

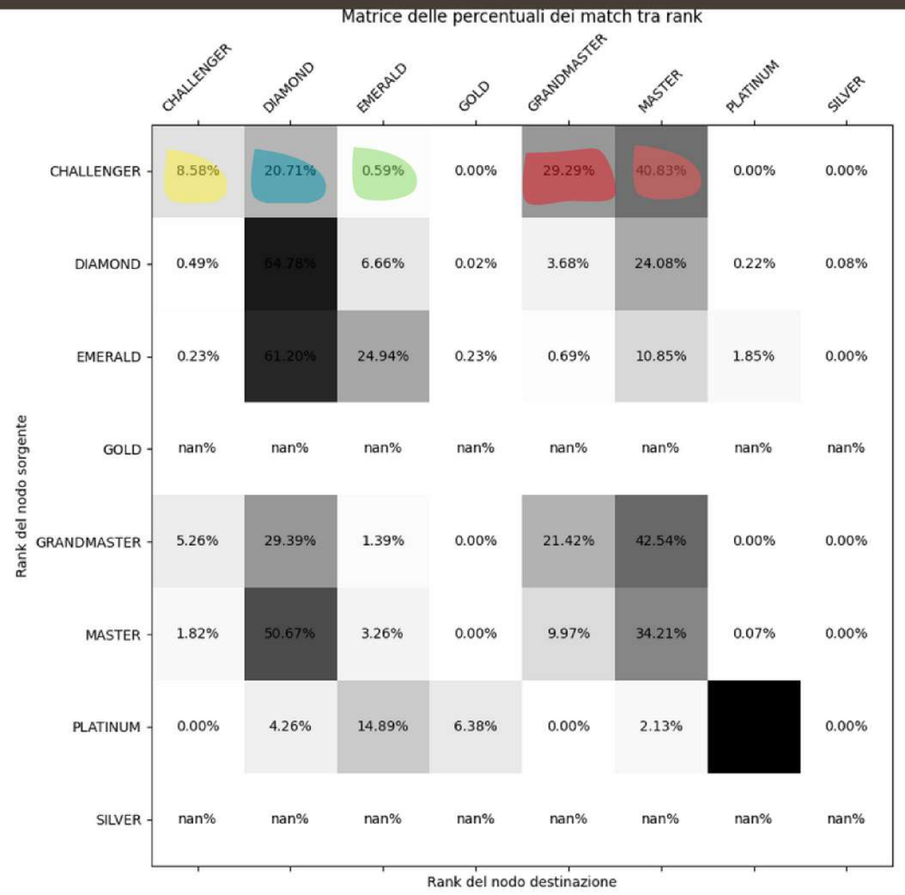
Archi: 11886 Densità: 0.000495751

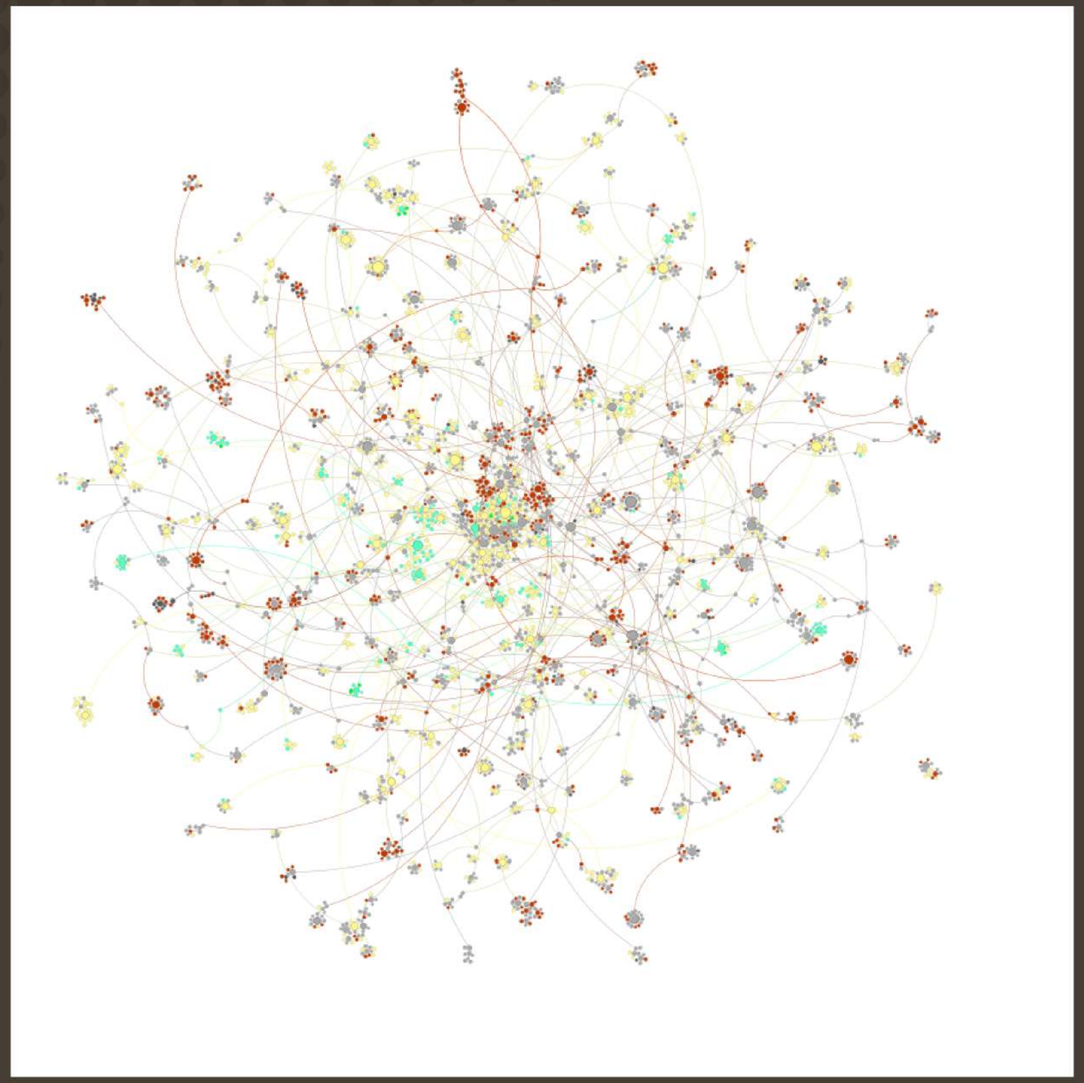
1024 Nodi Generatori ≈ 21%

1266 Nodi Foglia ≈ 25%

2607 Non Generatori in comune ≈ 53%

DIAMOND	(62,63%)
MASTER	(19,13%)
EMERALD	(11,8%)
GRANDMASTER	(3,96%)
PLATINUM	(1,25%)
CHALLENGER	(1,02%)
GOLD	(0,1%)
SILVER	(0,1%)


Matrice delle percentuali dei match tra rank
CHALLENGER
DIAMOND
EMERALD
GOLD
GRANDMASTER
MASTER
PLATINUM
SILVER
Rank del nodo sorgente
Rank del nodo destinazione
8.58% 20.71% 0.59% 0.00% 29.29% 40.83% 0.00% 0.00%
0.49% 64.78% 6.66% 0.02% 3.68% 24.08% 0.22% 0.08%
0.23% 61.20% 24.94% 0.23% 0.69% 10.85% 1.85% 0.00%
nan% nan% nan% nan% nan% nan% nan% nan%
5.26% 29.39% 1.39% 0.00% 21.42% 42.54% 0.00% 0.00%
1.82% 50.67% 3.26% 0.00% 9.97% 34.21% 0.07% 0.00%
0.00% 4.26% 14.89% 6.38% 0.00% 2.13% 0.00%
nan% nan% nan% nan% nan% nan% nan% nan%



Noob Graph

Nodi: 3436

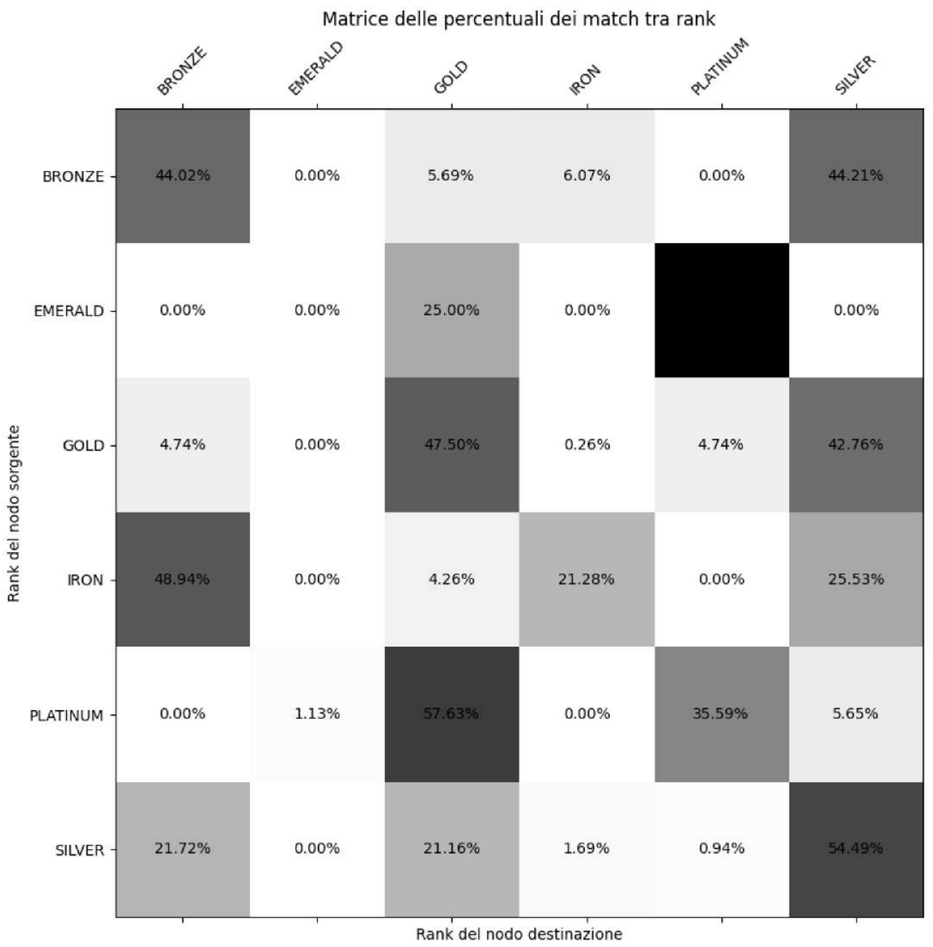
Archi: 3499 Densità: 0.000296458

995 Nodi Generatori ≈ 29%

1419 Nodi Foglia ≈ 41%

1022 Non Generatori in comune ≈ 30%

SILVER	(45,02%)
GOLD	(28,14%)
BRONZE	(19,21%)
PLATINUM	(4,86%)
IRON	(2,47%)
EMERALD	(0,29%)


Matrice delle percentuali dei match tra rank
BRONZE
EMERALD
GOLD
IRON
PLATINUM
SILVER
Rank del nodo sorgente
Rank del nodo destinazione
44.02% 0.00% 5.69% 6.07% 0.00% 44.21%
0.00% 0.00% 25.00% 0.00% 0.00%
4.74% 0.00% 47.50% 0.26% 4.74% 42.76%
48.94% 0.00% 4.26% 21.28% 0.00% 25.53%
0.00% 1.13% 57.63% 0.00% 35.59% 5.65%
21.72% 0.00% 21.16% 1.69% 0.94% 54.49%

Altre proprietà della rete

HIGH ELO:

Diametro della componente gigante: 13

Clustering locale medio: 0.004734807926971433

Clustering medio per nodi con CC > 0: 0.03819827746026212

LOW ELO:

Diametro della componente gigante: 10

Clustering locale medio: 2.9681061403995047e-05

Clustering medio per nodi con CC > 0: 0.033994708994708994

Transitività: 0.0051
Reciprocità: 0.0163

Rank	Reciprocità	Transitività
SILVER	nan%	0.00%
MASTER	1.35%	0.80%
PLATINUM	0.00%	0.00%
GRANDMASTER	3.86%	3.80%
EMERALD	0.00%	0.00%
CHALLENGER	0.00%	0.00%
GOLD	nan%	0.00%
DIAMOND	1.28%	0.19%

Transitività: 0.0000
Reciprocità: 0.0006

Rank	Reciprocità	Transitività
IRON	0.000%	0.000%
BRONZE	0.000%	0.000%
GOLD	0.000%	0.000%
PLATINUM	0.000%	0.000%
EMERALD	nan%	0.000%
SILVER	0.229%	0.000%

High Elo

Grado Medio: 4.8544006
Standard deviation: 7.2777559
Median: 2.0
Above Average player(W>L) 462/1024 ≈ 45 %
Max: 51
Max Degree Player : **Vedoluinim**

Master I
155LP

Main Champion:
Hecarim

Low Elo

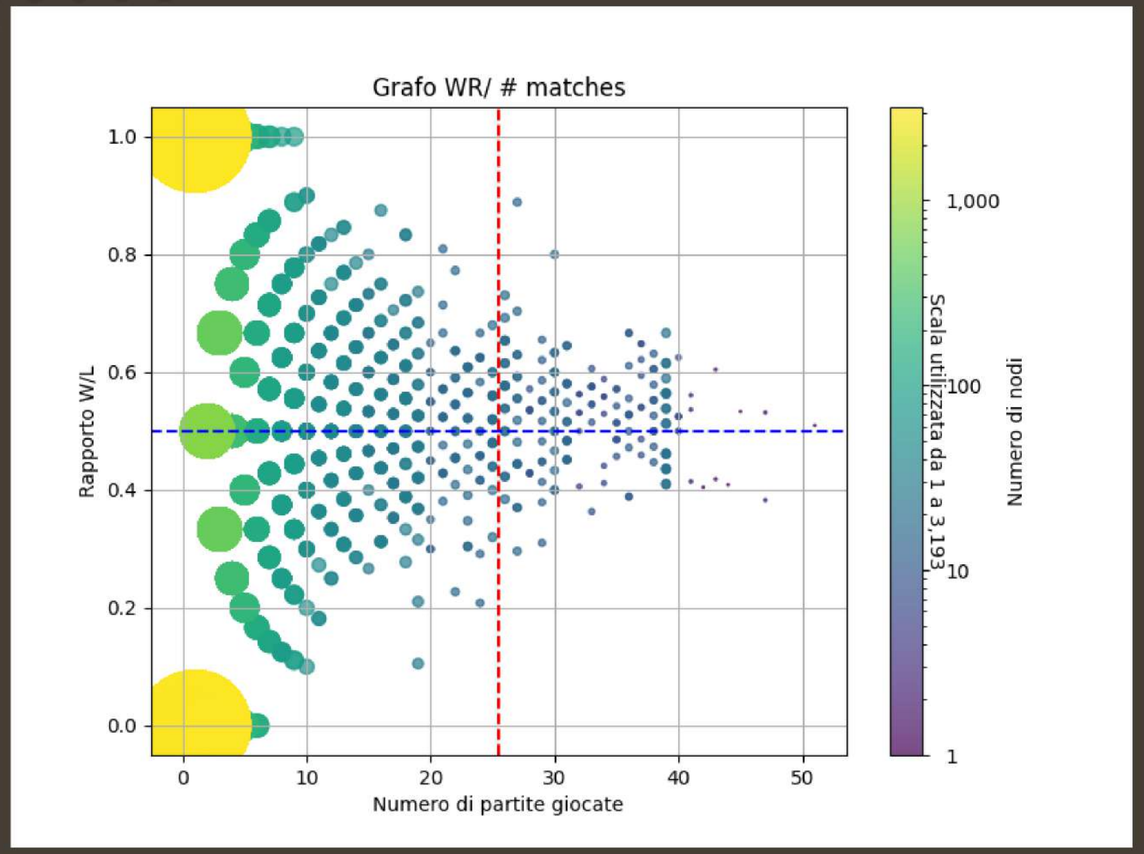
Grado Medio: 2.0366705
Standard deviation: 2.4731020
Median: 1.0
Above Average player(W>L) 413/995 ≈ 41 %
Max: 24
Max Degree Player : **STAND NAME**

Silver I
10 LP

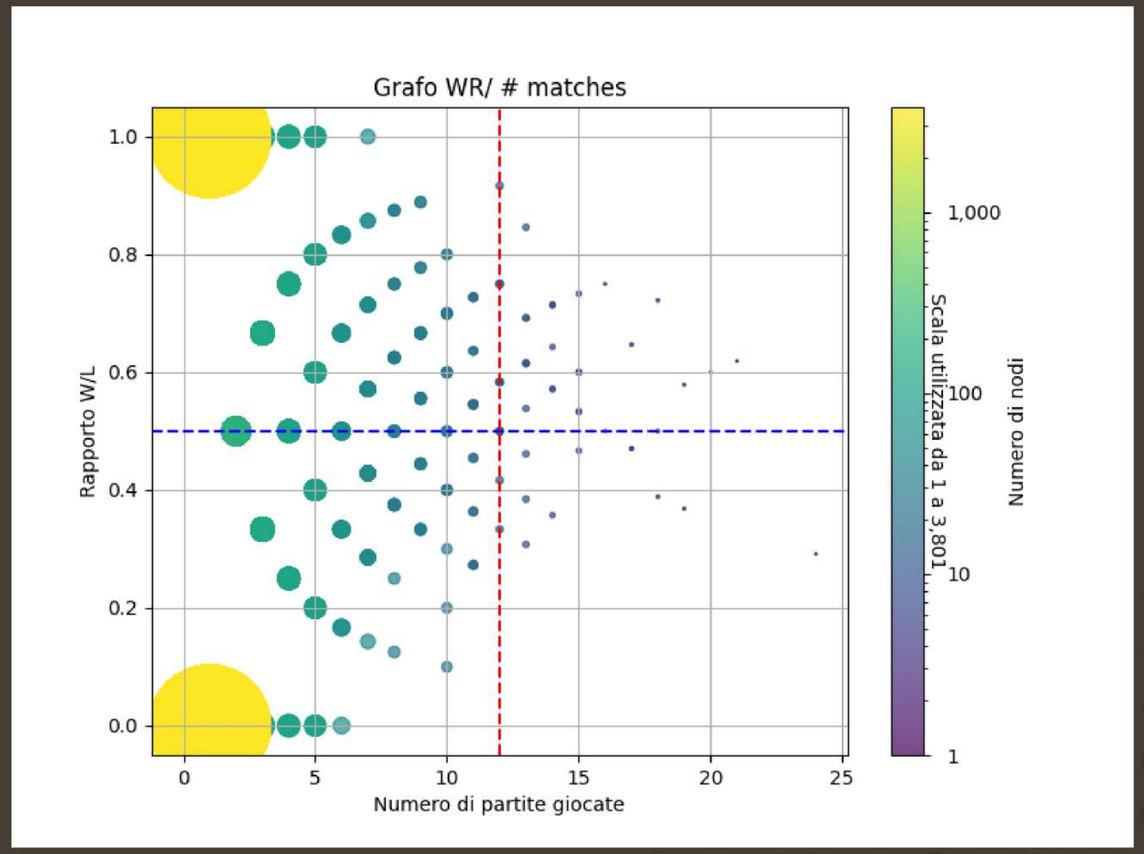
Main Champion:
Rengar

Distribuzione Rapporto Vittoria Sconfitta su Match giocati
Calcolato usando out degree/degree

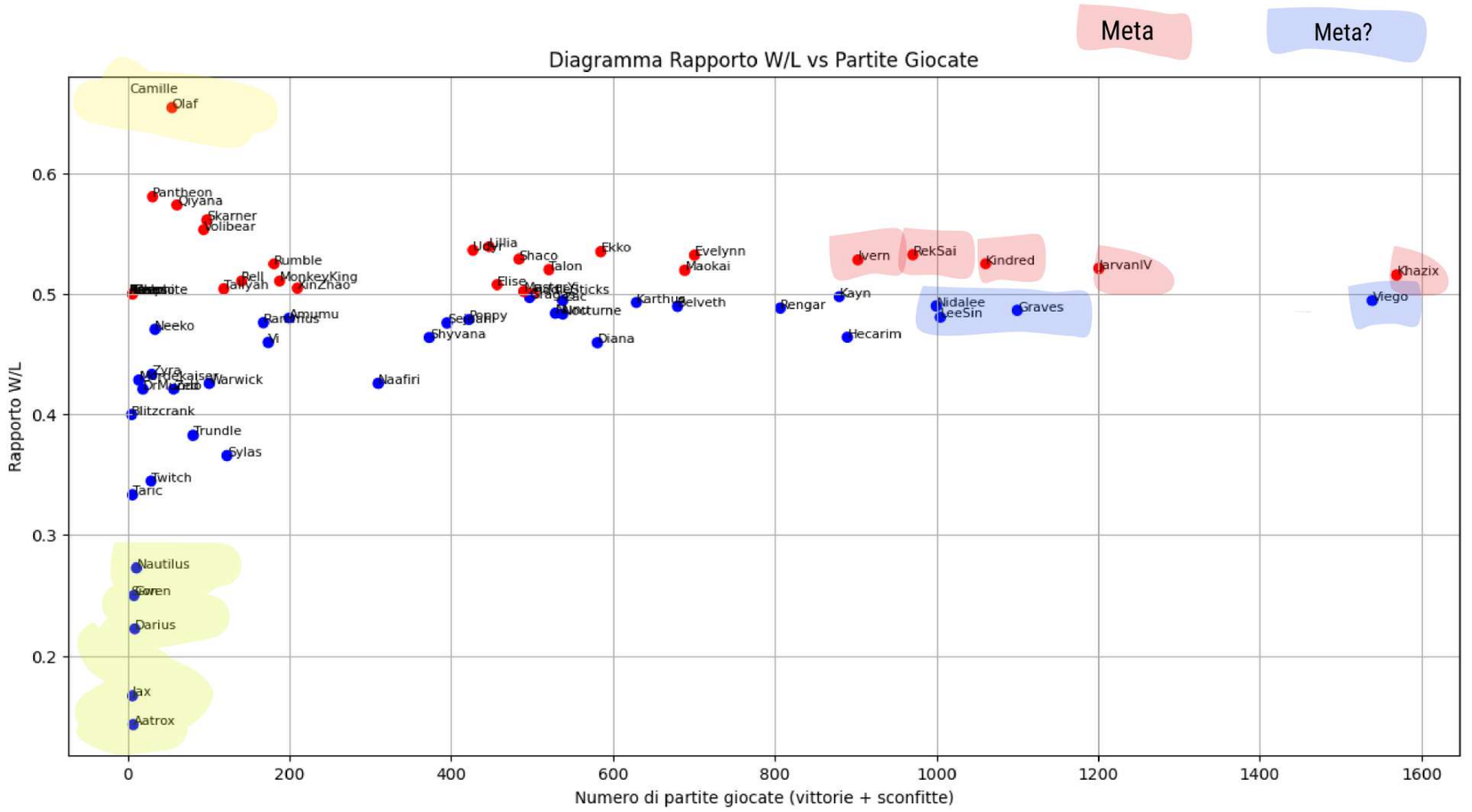
High Elo



Low Elo



Distribuzione Champion



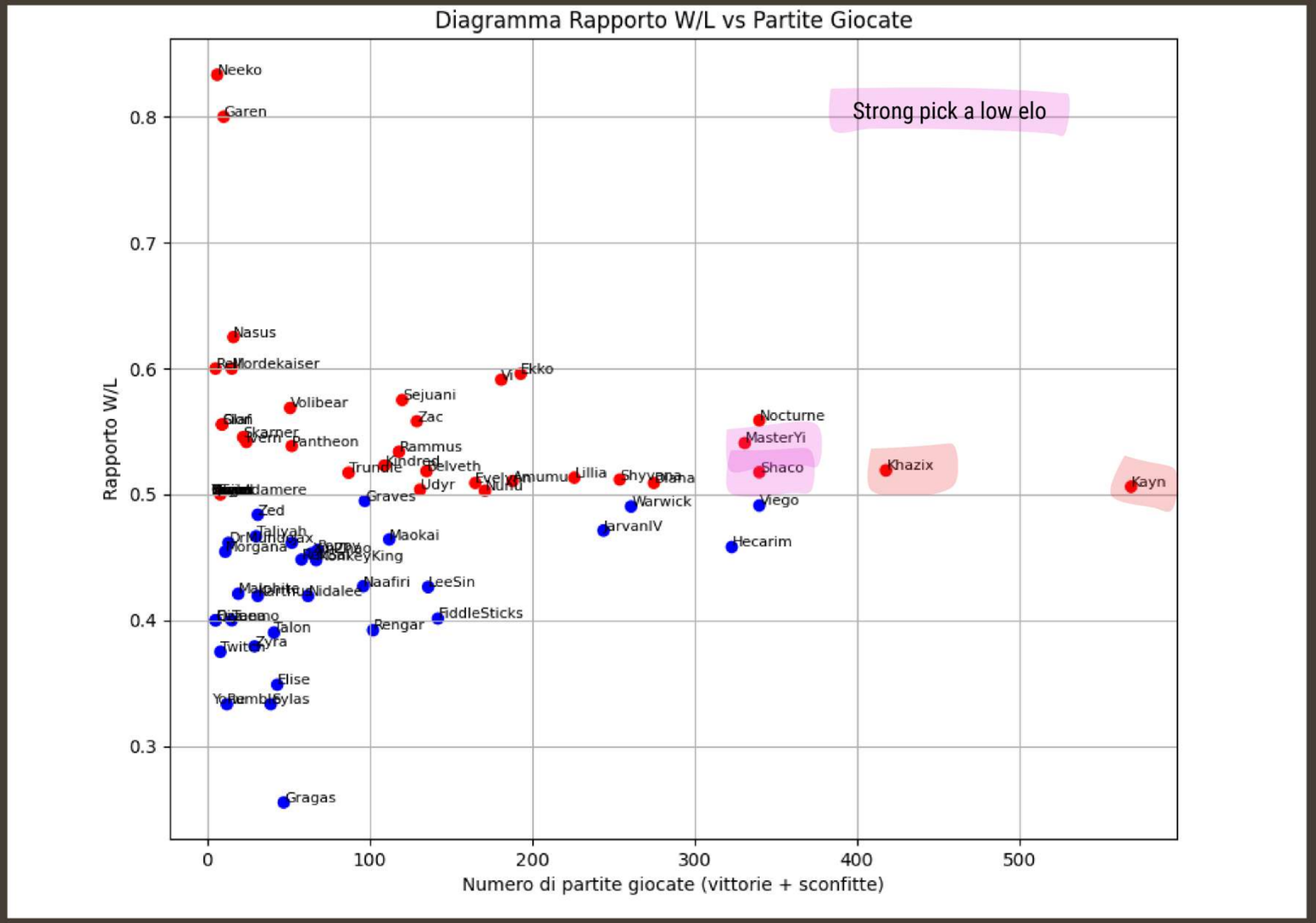


Diagramma Rapporto W/L vs Partite Giocate
Strong pick a low elo
Rapporto W/L
Numero di partite giocate (vittorie + sconfitte)
0.8
0.7
0.6
0.5
0.4
0.3
0
100
200
300
400
500
Neeko
Garen
Nasus
Mordekaiser
Ekko
Volibear
Sejuani
Zac
Pantheon
Rammus
Trundle
Udyr
Graves
Zed
Maokai
Naafiri
LeeSin
Nidalee
FiddleSticks
Rengar
Talon
Zyra
Elise
Sylas
Gragas
Nocturne
MasterYi
Shaco
Khazix
Kayn
Lillia
Warwick
Viego
JarvanIV
Hecarim

FONTE: OP.GG

Possiamo confrontare la champion distribution della nostra rete per capire quanto rispecchia il meta (ovvero quei campioni che sono avvantaggiati dalla patch attuale)

Rank	Role	Champion	Tier	Win Rate	Pick Rate	Ban Rate	Counter Picks	Matches
1		Kha'Zix	S+	51.53%	14.7%	27.9%		77,489
2		Rek'Sai	S+	53.18%	3.9%	14.4%		20,329
3		Evelynn	S+	51.28%	4.9%	14.6%		25,745
4		Kindred	S+	51.64%	4.5%	12.5%		23,575
5		Nidalee	S+	51.57%	7.2%	11.0%		38,122
6		Kayn	S+	50.45%	10.6%	15.6%		55,584
7		Shaco	S+	50.76%	6.1%	17.4%		32,179
8		Fiddlestic...	S	52.58%	4.0%	4.9%		21,046
9		Jarvan IV	S	50.79%	10.6%	4.5%		55,632
10		Master Yi	S	50.08%	5.0%	9.5%		26,409
11		Bel'Veth	S	51.44%	3.2%	4.6%		16,995
12		Karthus	S	51.38%	2.8%	6.1%		14,527
13		Rammus	A	51.17%	1.6%	4.5%		8,662
14		Ekko	A	50.32%	8.5%	5.4%		44,735

Rank	Role	Champion	Tier	Win Rate	Pick Rate	Ban Rate	Counter Picks	Matches
33		Trundle	B	48.41%	0.8%	0.5%		4,464
34		Gragas	B	49.33%	3.5%	1.8%		18,501
35		Talon	B	48.84%	1.9%	2.9%		9,948
36		Shyvana	C	48.22%	2.4%	1.9%		12,697
37		Elise	C	48.25%	2.7%	2.6%		13,966
38		Zed	D	45.98%	0.8%	37.3%		4,143
39		Sylas	D	48.24%	3.3%	8.4%		17,123
40		Sejuani	D	47.47%	2.9%	0.4%		15,168
41		Wukong	D	46.18%	1.8%	0.9%		9,677
42		Vi	D	47.31%	2.7%	0.8%		14,020
43		Viego	D	48.92%	9.0%	2.4%		47,345
44		Hecarim	D	49.03%	7.2%	16.2%		38,099
45		Lee Sin	D	48.21%	12.9%	6.7%		67,641
46		Naafiri	D	42.29%	1.1%	47.8%		5,923

Matchup più frequenti

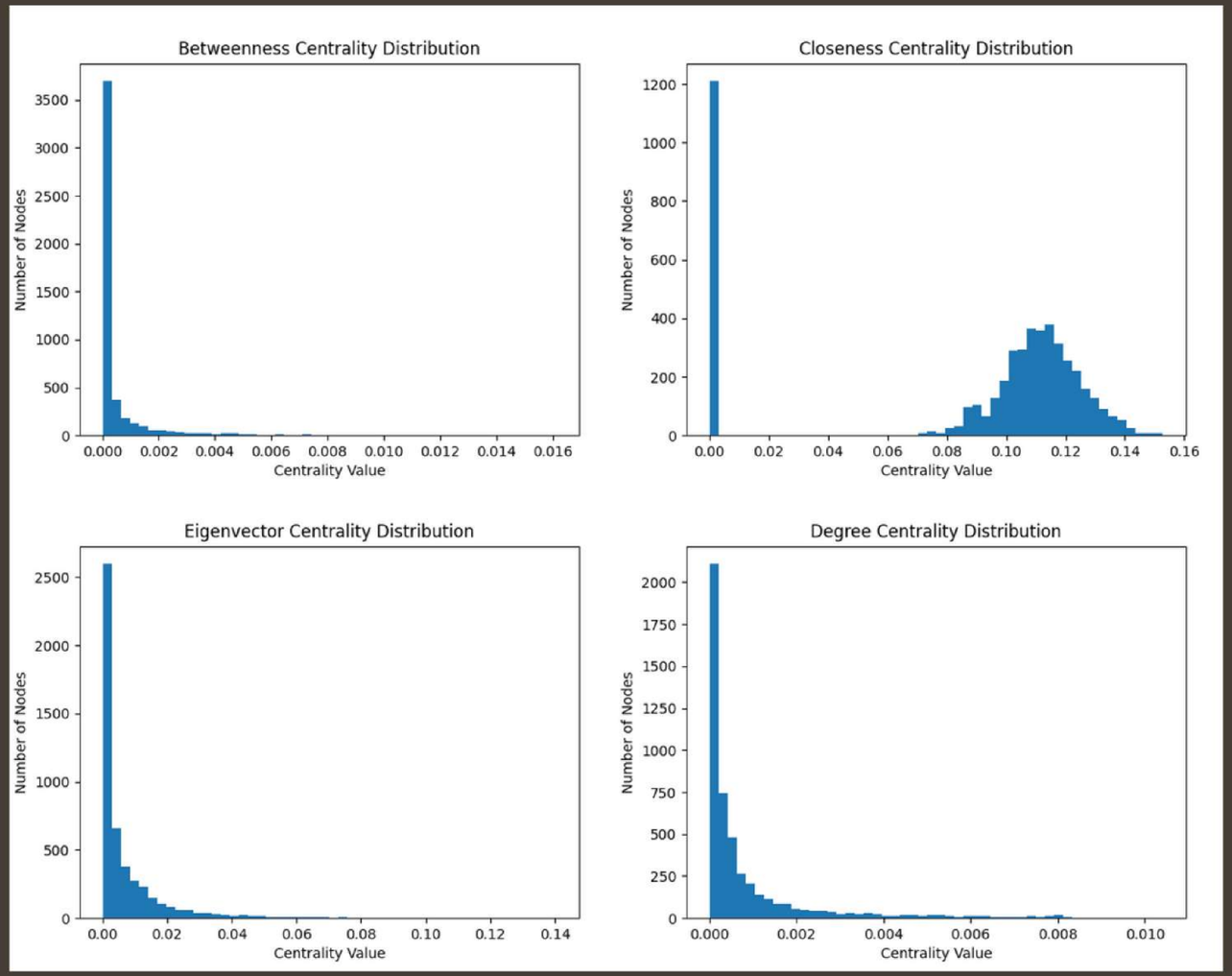
```
Top 5 matchups:
('LeeSin', 'Khazix') 54 ~ 48
('Nidalee', 'Khazix') 52 ~ 26
('Khazix', 'Viego') 49 ~ 42
('Kindred', 'Nidalee') 49 ~ 28
('Khazix', 'LeeSin') 48 ~ 54
Matchup meno bilanciato: ('Nidalee', 'Khazix')
Numero di vittorie: 52
Numero di sconfitte: 26
Numero di partite affrontate: 78
Percentuale di vittorie sul numero di partite affrontate: 66.66666666666666
```

VS

```
Top 5 matchups:
('MasterYi', 'Kayn') 21 ~ 15
('Nocturne', 'Khazix') 20 ~ 13
('Kayn', 'Shyvana') 19 ~ 8
('Nocturne', 'Kayn') 17 ~ 7
('Kayn', 'Diana') 17 ~ 14
Matchup meno bilanciato: ('Kayn', 'Shyvana')
Numero di vittorie: 19
Numero di sconfitte: 8
Numero di partite affrontate: 27
Percentuale di vittorie sul numero di partite affrontate: 70.37037037037037
```

VS

Centrality Distribution



Analisi Centralità

Degree Centrality

La centralità di un nodo è proporzionale al suo grado

	generator, non	rank counts	node degree	total degree	total out degree	total in degree	average out degree	average in degree	node
0	(14, 12)	{'DIAMOND': 19, 'MASTER': 6, 'GRANDMASTER': 1}	51	367	176	191	6.769230769230769	7.346153846153846	Vedoluinim
1	(12, 13)	{'GRANDMASTER': 5, 'DIAMOND': 9, 'MASTER': 11}	47	432	202	230	8.08	9.2	HORSETOCHALL
2	(9, 9)	{'GRANDMASTER': 6, 'MASTER': 5, 'DIAMOND': 4, 'CHALLENGER': 3}	47	331	180	151	10.0	8.38888888888889	FA Warrior
3	(7, 17)	{'DIAMOND': 19, 'MASTER': 2, 'EMERALD': 2, 'GRANDMASTER': 1}	45	189	82	107	3.4166666666666665	4.458333333333333	Famous Fingers 5
4	(5, 13)	{'MASTER': 6, 'DIAMOND': 12}	44	161	77	84	4.277777777777778	4.666666666666667	PayzBack

```
[
    ("Vedoluinim", 0.010416666666666668),
    ("HORSETOCHALL", 0.00959967320261438),
    ("FA Warrior", 0.00959967320261438),
    ("Famous Fingers 5", 0.009191176470588236),
    ("PayzBack", 0.008986928104575164),
]
```

Betweeness Centrality

Un nodo è più centrale più compare nei cammini minimi tra un nodo ed un altro.

Fattori osservati: average degree nei vicini, total degree

	generator, non	rank counts	node degree	total degree	total out degree	total in degree	average out degree	average in degree	node
0	(14, 12)	{'DIAMOND': 19, 'MASTER': 6, 'GRANDMASTER': 1}	51	367	176	191	6.769230769230769	7.346153846153846	Vedoluinim
1	(12, 13)	{'GRANDMASTER': 5, 'DIAMOND': 9, 'MASTER': 11}	47	432	202	230	8.08	9.2	HORSETOCHALL
2	(9, 9)	{'MASTER': 7, 'DIAMOND': 10, 'EMERALD': 1}	39	275	134	141	7.444444444444445	7.833333333333333	Cochon à 8h
3	(10, 11)	{'MASTER': 13, 'DIAMOND': 8}	40	277	143	134	6.809523809523809	6.380952380952381	MetaSlaveJungler
4	(4, 21)	{'DIAMOND': 18, 'EMERALD': 4, 'MASTER': 3}	40	142	56	86	2.24	3.44	Ciamajda

```
[
    ("Vedoluinim", 0.016139108643345 69),
    ("HORSETOCHALL", 0.015428245174604035),
    ("Cochon à 8h", 0.015041498642427906),
    ("MetaSlaveJungler", 0.013992144581473036),
    ("Ciamajda", 0.013351049198755545),
]
```

Closeness Centrality

Centralità correlata con il numero di hop necessari a raggiungere gli altri nodi.

	generator, non	rank counts	node degree	total degree	total out degree	total in degree	average out degree	average in degree	node
0	(10, 12)	{'GRANDMASTER': 5, 'MASTER': 10, 'DIAMOND': 7}	39	422	219	203	9.954545454545455	9.227272727272727	Bolshoi Booze
1	(14, 5)	{'MASTER': 7, 'GRANDMASTER': 7, 'DIAMOND': 5}	36	420	214	206	11.263157894736842	10.842105263157896	ME LittleLaudi
2	(12, 13)	{'GRANDMASTER': 5, 'DIAMOND': 9, 'MASTER': 11}	47	432	202	230	8.08	9.2	HORSETOCHALL
3	(11, 13)	{'GRANDMASTER': 2, 'MASTER': 15, 'DIAMOND': 6, 'CHALLENGER': 1}	39	385	197	188	8.208333333333334	7.833333333333333	matrix agent 2
4	(10, 13)	{'GRANDMASTER': 3, 'MASTER': 10, 'EMERALD': 2, 'DIAMOND': 7, 'CHALLENGER': 1}	39	338	171	167	7.434782608695652	7.260869565217392	Villano Elmo

```
[
    ("Bolshoi Booze", 0.18882996445900438),
    ("ME LittleLaudi", 0.18445959816266666),
    ("HORSETOCHALL", 0.18358052144386128),
    ("matrix agent 2", 0.18266165160523287),
    ("Villano Elmo", 0.18140718935011116),
]
```

Eigenvector Centrality

Centralità influenzata dall’importanza dei vicini nella rete

	generator, non	rank counts	node degree	total degree	total out degree	total in degree	average out degree	average in degree	node
0	(13, 11)	{'GRANDMASTER': 8, 'DIAMOND': 6, 'CHALLENGER': 5, 'MASTER': 5}	30	375	195	180	8.125	7.5	Groinze
1	(8, 7)	{'GRANDMASTER': 8, 'MASTER': 3, 'DIAMOND': 1, 'CHALLENGER': 3}	28	308	159	149	10.6	9.933333333333334	FaithXD
2	(11, 13)	{'GRANDMASTER': 2, 'MASTER': 15, 'DIAMOND': 6, 'CHALLENGER': 1}	39	385	197	188	8.208333333333334	7.833333333333333	matrix agent 2
3	(9, 9)	{'GRANDMASTER': 6, 'MASTER': 5, 'DIAMOND': 4, 'CHALLENGER': 3}	47	331	180	151	10.0	8.38888888888889	FA Warrior
4	(9, 9)	{'CHALLENGER': 3, 'MASTER': 7, 'DIAMOND': 4, 'GRANDMASTER': 4}	26	281	148	133	8.222222222222221	7.388888888888889	Maître 0

```
[
    ("Groinze", 0.17081319094397215),
    ("FaithXD", 0.15195576324537874),
    ("matrix agent 2", 0.13865007316096215),
    ("FA Warrior", 0.12801691617251587),
    ("Maître 0", 0.12096204634653036),
]
```

```
StanPelly : : 37
matrix agent 2 : : 39
meistro : : 25
Vol : : 23
last arc : : 11
seul : : 7
DKRQGSCR : : 10
Desireful59 : : 4
unhappy Pony : : 4
Egotistical JG : : 6
Razørk Activoø : : 13
Traypo : : 2
tent is back : : 29
stipe69 : : 10
qwqw : : 2
Pura fantasia : : 43
Jebeni Ovisnik : : 33
Theocacs : : 33
```

```
Roi DEMON : : 38
Cow1337killr : : 13
Kevin Pearson : : 26
matrix agent 2 : : 39
tent is back : : 29
Goocha : : 8
FA Warrior : : 47
NO BYSTANDERS9 : : 15
SzymusJD : : 24
Fɑllɛn 1 : : 6
Kirei sama : : 17
KVIHNLCM : : 9
MafroMafroMafro : : 9
TMAX VROUM VROUM : : 17
Dopaminw : : 11
```

```
Roi DEMON : : 38
SzymusJD : : 24
Tezhak : : 3
MUSKETOON : : 4
tempbbnbwovvonqq : : 11
NO BYSTANDERS9 : : 15
Maître 0 : : 26
HACKSLASHXXXXX : : 8
lCherkessl : : 32
Razørk Activoø : : 13
Cow1337killr : : 13
KaniaWest : : 18
Cosmic Silence : : 2
Dîamondprox : : 4
Dopaminw : : 11
akksor : : 27
PikAsyou : : 37
KaynKîng : : 14
CrÃzY : : 27
Hyweedri : : 3
Pipibaat Yarr : : 9
verypositive : : 19
G2 Nezuko : : 3
EagleSbroccato : : 14
```

Top 5 giocatori per metrica di centralità

Eigenvector Centrality	Betweenness Centrality	Closeness Centrality	Degree Centrality
Groinze	Vedoluinim	Bolshoi Booze	Vedoluinim
FaithXD	HORSETOCHALL	ME LittleLaudi	HORSETOCHALL
matrix agent 2	Cochon à 8h	HORSETOCHALL	FA Warrior
FA Warrior	MetaSlaveJungler	matrix agent 2	Famous Fingers 5
Maître 0	Ciamajda	Villano Elmo	PayzBack

MATCH PREDICTION

Verifichiamo se a 15' di un match è possibile predirne l'esito

"WIN LANE WIN GAME"

UTILIZZO:

KNN CLASSIFIER

RANDOM FOREST

DEEP LEARNING

Come navigare il response body di timeline

GET /lol/match/v5/matches/{matchId}/timeline

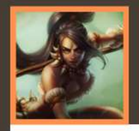
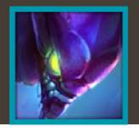
Metadata : contiene i puuid dei giocatori partecipanti (in Info trovi anche i rispettivi participantID)

Info : composto da Frames ovvero snapshot del match ad ogni minuto.

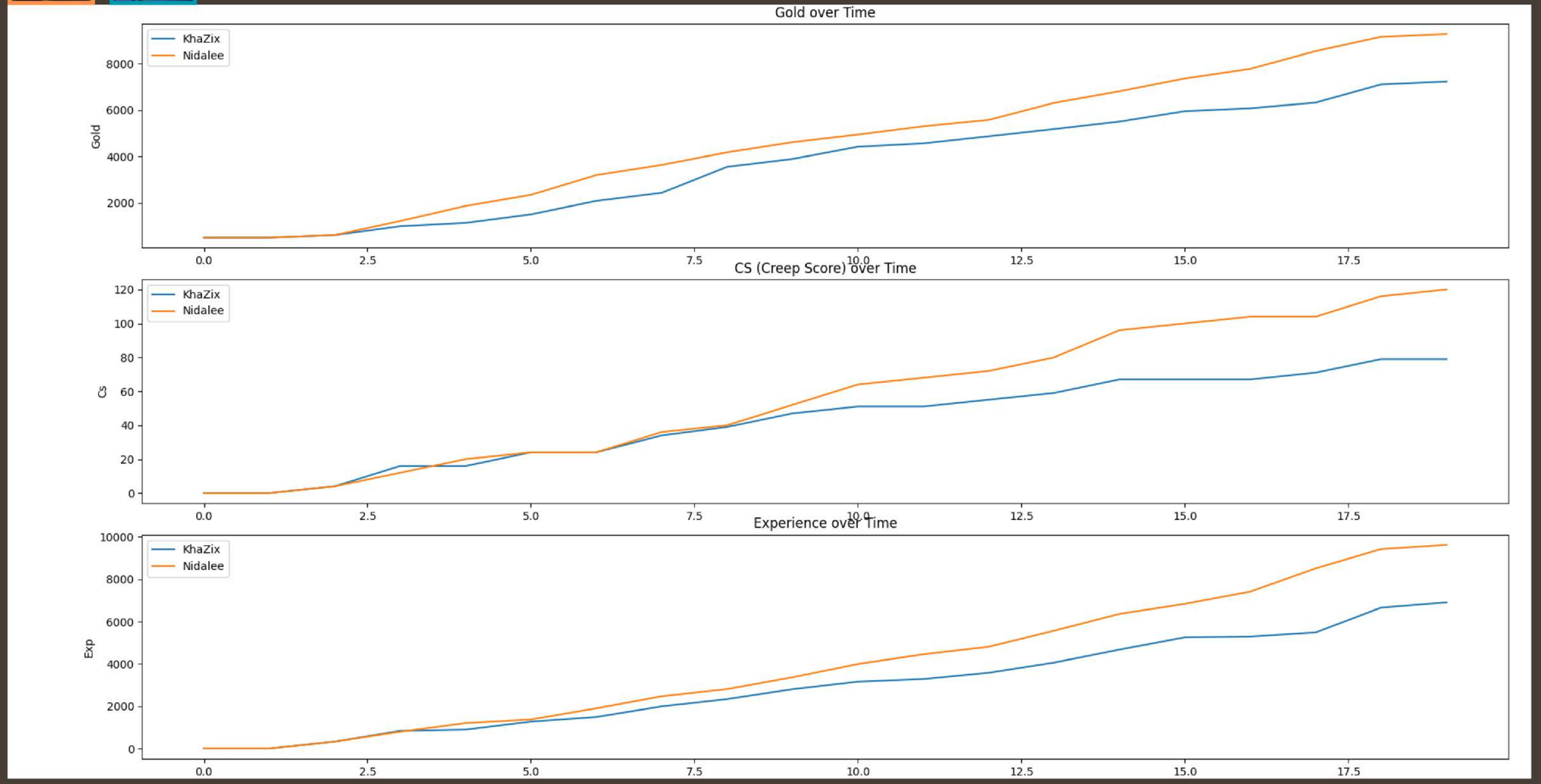
Ogni Frame è composto da Events e Participant Frames

Events: log di azioni avvenute in quel minuto(Frame) tipo: ChampionKill, ItemDestroyed etc..

Participant Frame: snapshot dei vari partecipanti con informazioni sulle statistiche del campione e inforamzioni generali come minion uccisi, gold in tasca e totali, xp, livello etc.

Champion Performance Tracking



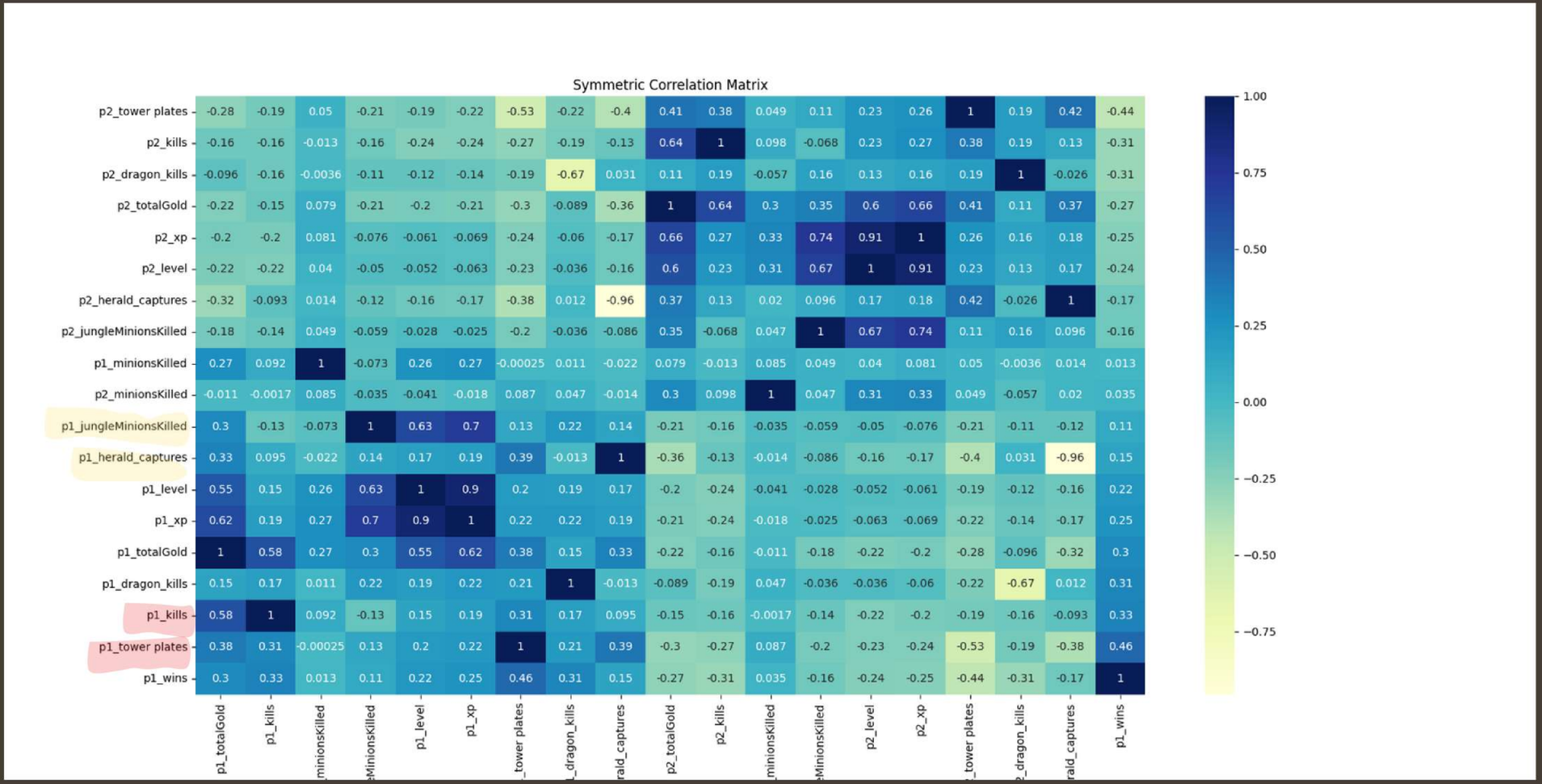
Kill map

```
domain = {
            min: {x: -120, y: -120},
            max: {x: 14870, y: 14980}
    },
    width = 512,
    height = 512,
    bg = "https://s3-us-west-1.amazonaws.com/riot-developer-portal/docs/map11.png",
    xScale, yScale, svg;

color = d3.scale.linear()
    .domain([0, 3])
    .range(["white", "steelblue"])
    .interpolate(d3.interpolateLab);

xScale = d3.scale.linear()
  .domain([domain.min.x, domain.max.x])
  .range([0, width]);

yScale = d3.scale.linear()
  .domain([domain.min.y, domain.max.y])
  .range([height, 0]);

svg = d3.select("#map").append("svg:svg")
    .attr("width", width)
    .attr("height", height);

svg.append('image')
    .attr('xlink:href', bg)
    .attr('x', '0')
    .attr('y', '0')
    .attr('width', width)
    .attr('height', height);

svg.append('svg:g').selectAll("circle")
    .data(cords)
    .enter().append("svg:circle")
        .attr('cx', function(d) { return xScale(d[0]) })
        .attr('cy', function(d) { return yScale(d[1]) })
        .attr('r', 5)
        .attr('class', 'kills');    transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
    ])
}
```

Timeline raccolte : 1053

p1_totalGold	p1_kills	p1_minior	p1_jungle	p1_level	p1_xp	p1_tower	p1_dragor	p1_herald	p2_totalGold	p2_kills	p2_minior	p2_jungle	p2_level	p2_xp	p2_tower	p2_dragon	p2_herald	p1_wins	id
7,294	10	12	93	10	6,806	6	1	0	6,662	5	10	92	9	6,067	9	0	1	1	EUW1_6521300440
6,868	4	6	76	9	5,207	3	0	1	5,849	6	2	79	9	5,187	10	1	0	1	EUW1_6521138617
6,35	2	5	113	10	7,288	2	2	0	4,634	2	2	76	9	5,353	9	0	1	1	EUW1_6519511169
5,985	4	12	84	9	5,733	11	1	0	5,987	6	17	56	8	4,924	2	0	1	1	EUW1_6519340361
6,305	5	6	92	10	6,44	7	1	1	6,7	5	14	84	9	5,72	13	0	0	1	EUW1_6519181941
5,509	4	4	80	9	5,336	3	1	1	5,343	3	2	80	8	5,014	10	1	0	1	EUW1_6516439352
5,727	3	20	80	9	6,019	8	1	0	7,005	5	10	119	10	6,992	6	1	1	1	EUW1_6516369292
7,028	9	17	89	9	6,031	6	1	0	6,914	11	5	74	9	5,741	5	0	1	1	EUW1_6516331731
7,142	5	8	96	10	6,805	4	1	1	5,888	2	14	101	10	6,276	13	1	0	1	EUW1_6516264386
5,703	7	5	88	9	5,562	6	1	0	5,557	4	5	74	9	5,361	8	0	1	1	EUW1_6516231073
5,305	5	8	80	8	4,93	4	0	1	5,738	7	17	68	8	4,868	4	2	0	1	EUW1_6514441704
5,765	4	11	97	9	5,911	6	2	0	5,762	5	7	81	9	5,636	6	0	1	1	EUW1_6514029725
4,699	2	16	78	9	5,372	5	1	0	5,573	6	6	80	9	5,181	7	0	1	1	EUW1_6518970306
5,701	3	1	83	9	5,538	1	1	1	5,461	1	13	91	9	5,687	14	0	0	1	EUW1_6517559307
6,557	4	20	80	10	6,671	9	1	0	7,229	5	31	94	11	7,668	12	0	1	1	EUW1_6521640782
5,404	6	10	60	9	5,244	3	0	1	6,575	8	13	98	9	6	6	1	0	1	EUW1_6521588302
6,722	8	19	88	10	6,296	6	0	0	6,12	8	20	67	10	6,159	4	1	1	1	EUW1_6521248964
7,673	6	15	68	10	6,131	4	2	1	5,428	4	13	64	9	5,141	10	0	0	1	EUW1_6520804187
6,264	3	15	84	9	5,866	4	1	1	6,174	2	8	128	11	7,545	7	1	0	1	EUW1_6520742895
5,276	3	15	88	9	5,827	4	1	0	5,612	3	8	84	9	5,593	11	0	1	1	EUW1_6520693596
7,638	5	6	80	9	5,846	3	2	1	5,773	5	9	83	9	6,095	12	0	0	1	EUW1_6520393563
5,966	6	25	64	10	6,203	4	1	1	5,943	3	15	103	10	6,897	9	1	0	1	EUW1_6520297256
6,451	6	17	88	10	6,398	7	1	1	7,274	7	9	92	10	6,51	8	1	0	1	EUW1_6520069131
6,243	8	7	86	10	6,447	3	1	1	4,216	0	4	81	9	5,268	15	0	0	1	EUW1_6519134517
8,084	6	29	80	10	6,564	10	1	1	7,333	7	15	95	10	7,206	12	0	0	1	EUW1_6518922080
6,608	5	16	99	10	6,737	7	1	0	7,782	6	23	76	9	5,435	5	0	1	1	EUW1_6518520647
6,033	5	15	90	9	5,837	2	2	0	5,186	4	12	80	9	5,299	7	0	1	1	EUW1_6518401894
5,967	3	8	72	9	5,712	8	0	1	5,574	6	7	87	9	5,375	7	1	0	1	EUW1_6518195325
6,737	7	2	86	9	5,492	1	2	1	6,262	5	6	76	8	4,424	10	0	0	1	EUW1_6518168294
6,828	3	29	91	10	6,72	2	2	0	5,083	1	5	88	9	5,937	6	0	0	1	EUW1_6518140840
6,531	7	8	80	10	6,672	7	2	0	6,297	4	13	96	9	6,078	10	0	1	1	EUW1_6516779240
5,745	7	19	60	9	5,325	7	1	0	6,435	4	11	95	9	6,064	1	0	1	1	EUW1_6516629108
5,488	9	15	81	10	6,141	13	1	0	7,296	7	21	72	9	6,084	5	0	1	1	EUW1_6521573307
5,806	10	9	69	8	4,995	7	1	0	5,561	4	17	73	9	5,631	8	0	1	1	EUW1_6520953854

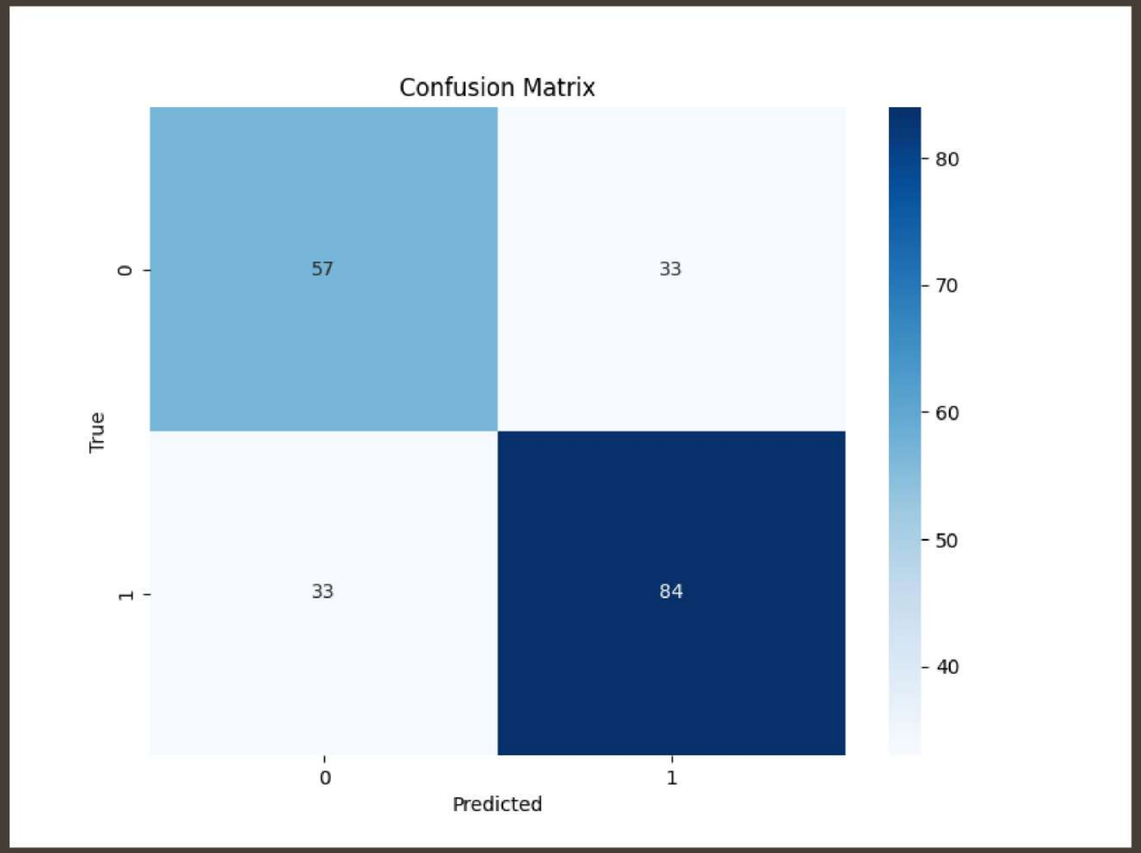
Winning stats correlation



Knn classifier

```
from sklearn.model_selection import train_test_split

df_selected = df.drop(columns=["id"])

X, y = df_selected.drop(["p1_wins"], axis=1), df_selected["p1_wins"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
print(len(X_train))

from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.fit_transform(X_test)
knn = KNeighborsClassifier()
knn.fit(X_train_scaled, y_train)
print(knn.score(X_test_scaled, y_test))
```

```
0.6811594202898551
```



Accuracy

0.6812

Precision

0.7179

F1

0.7179

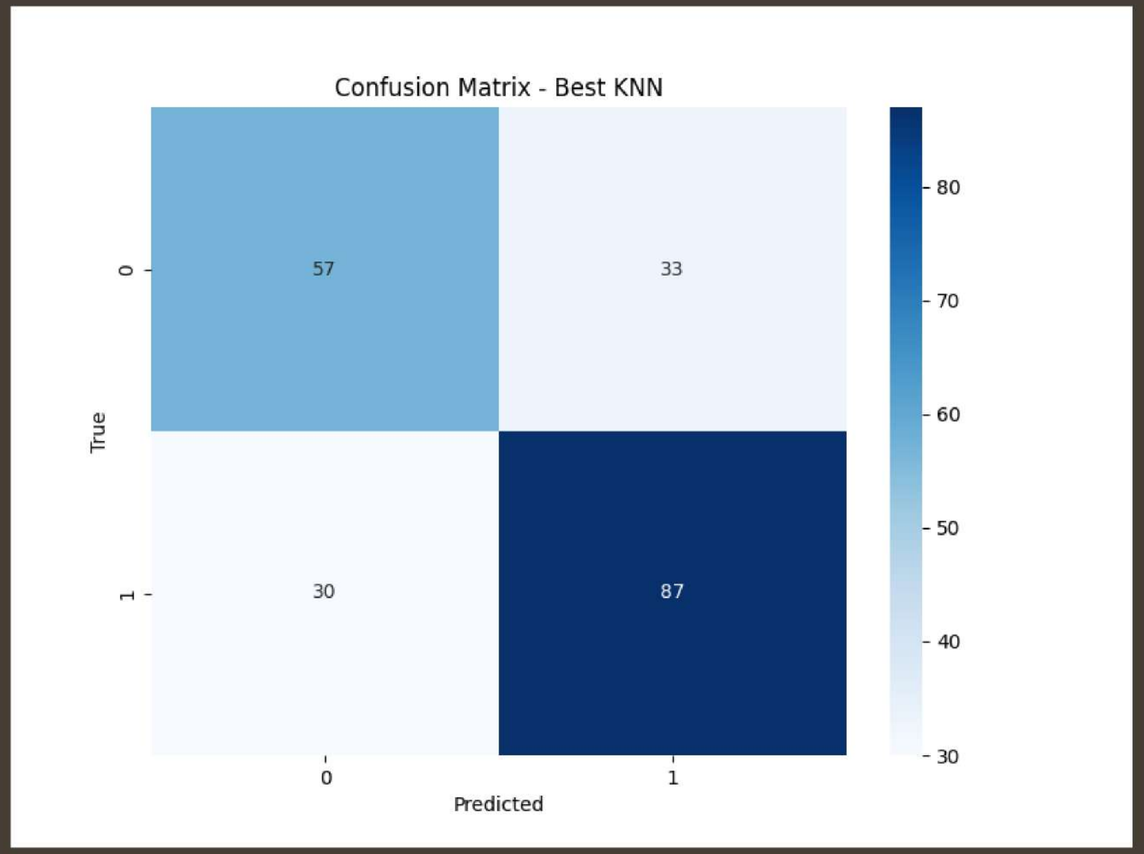
Recall

0.7179

Best Knn estimator

```python
from sklearn.model_selection import RandomizedSearchCV

param_grid = {"n_neighbors": list(range(5, 17, 2)), "weights": ["uniform", "distance"]}
knn_2 = KNeighborsClassifier(n_jobs=4)

clf = RandomizedSearchCV(knn_2, param_grid, n_jobs=4, n_iter=3, verbose=2, cv=3)
clf.fit(X_train_scaled, y_train)
knn_2 = clf.best_estimator_
print("best estimator: ", knn_2, " scored: ", knn_2.score(X_test_scaled, y_test))
```

```
Fitting 3 folds for each of 3 candidates, totalling 9 fits
[CV] END ......................n_neighbors=9, weights=uniform; total time=   0.0s
[CV] END .....................n_neighbors=11, weights=uniform; total time=   0.0s
[CV] END ......................n_neighbors=9, weights=uniform; total time=   0.0s
[CV] END ......................n_neighbors=9, weights=uniform; total time=   0.0s
[CV] END .....................n_neighbors=11, weights=uniform; total time=   0.0s
[CV] END .....................n_neighbors=11, weights=uniform; total time=   0.0s
[CV] END ....................n_neighbors=15, weights=distance; total time=   0.0s
[CV] END ....................n_neighbors=15, weights=distance; total time=   0.0s
[CV] END ....................n_neighbors=15, weights=distance; total time=   0.0s
best estimator:  KNeighborsClassifier(n_jobs=4, n_neighbors=9)  scored:  0.6956521739130435
```



Accuracy

0.6957

Precision

0.7436

F1

0.7250

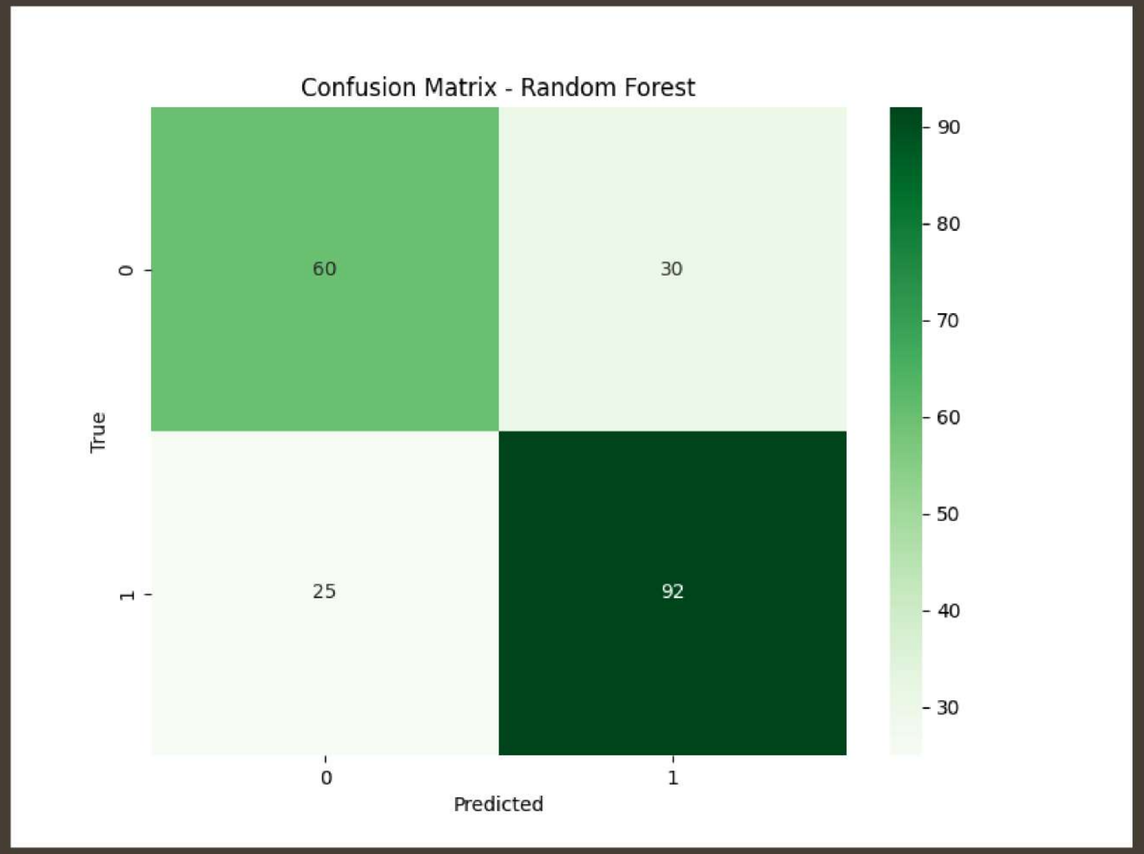
Recall

0.7436

Alberi di decisione :: Random Forest

```python
from sklearn.ensemble import RandomForestClassifier

forest = RandomForestClassifier(n_jobs=4)
forest.fit(X_train_scaled, y_train)
print("forest score: ", forest.score(X_test_scaled, y_test))
```

```
forest score:  0.7342995169082126
```



Accuracy

0.7343

Precision

0.7541

F1

0.7699

Recall

0.7863

Neural Network

```python
from tensorflow import keras

model = keras.models.Sequential()
model.add(keras.layers.Input(shape=(18,)))
model.add(keras.layers.Dense(200, activation="relu"))
model.add(keras.layers.Dense(100, activation="relu"))
model.add(keras.layers.Dense(100, activation="relu"))
model.add(keras.layers.Dense(1, activation="sigmoid"))

model.compile(loss="binary_crossentropy", optimizer="adam", metrics=["accuracy"])
early_stopping_cb = keras.callbacks.EarlyStopping(
    patience=10
)  # stops at 10 epochs without improvement

X_train_scaled_train, X_valid, y_train_train, y_valid = train_test_split(
    X_train_scaled, y_train, test_size=0.15
)

model.fit(
    X_train_scaled_train,
    y_train_train,
    epochs=30,
    callbacks=[early_stopping_cb],
    validation_data=(X_valid, y_valid),
)

print(model.evaluate(X_test_scaled, y_test))
```

```
[0.6514018177986145, 0.6908212304115295]
```

3 dense layer

Sigmoid activation function

```
Epoch 1/30
22/22 [==============================] - 1s 11ms/step - loss: 0.5607 - accuracy: 0.7208 - val_loss: 0.4769 - val_accuracy: 0.7600
Epoch 2/30
22/22 [==============================] - 0s 3ms/step - loss: 0.4808 - accuracy: 0.7692 - val_loss: 0.4667 - val_accuracy: 0.7840
Epoch 3/30
22/22 [==============================] - 0s 3ms/step - loss: 0.4593 - accuracy: 0.7778 - val_loss: 0.4611 - val_accuracy: 0.7760
Epoch 4/30
22/22 [==============================] - 0s 3ms/step - loss: 0.4360 - accuracy: 0.7920 - val_loss: 0.4681 - val_accuracy: 0.7920
Epoch 5/30
22/22 [==============================] - 0s 2ms/step - loss: 0.4156 - accuracy: 0.8006 - val_loss: 0.4689 - val_accuracy: 0.7760
Epoch 6/30
22/22 [==============================] - 0s 3ms/step - loss: 0.4029 - accuracy: 0.8077 - val_loss: 0.4759 - val_accuracy: 0.7920
Epoch 7/30
22/22 [==============================] - 0s 3ms/step - loss: 0.3850 - accuracy: 0.8234 - val_loss: 0.4936 - val_accuracy: 0.7680
Epoch 8/30
22/22 [==============================] - 0s 2ms/step - loss: 0.3658 - accuracy: 0.8362 - val_loss: 0.5077 - val_accuracy: 0.7680
Epoch 9/30
22/22 [==============================] - 0s 3ms/step - loss: 0.3450 - accuracy: 0.8519 - val_loss: 0.5069 - val_accuracy: 0.7840
Epoch 10/30
22/22 [==============================] - 0s 2ms/step - loss: 0.3246 - accuracy: 0.8604 - val_loss: 0.6170 - val_accuracy: 0.7360
Epoch 11/30
22/22 [==============================] - 0s 3ms/step - loss: 0.3032 - accuracy: 0.8832 - val_loss: 0.5772 - val_accuracy: 0.7520
Epoch 12/30
22/22 [==============================] - 0s 3ms/step - loss: 0.2720 - accuracy: 0.8932 - val_loss: 0.5843 - val_accuracy: 0.7600
Epoch 13/30
22/22 [==============================] - 0s 3ms/step - loss: 0.2605 - accuracy: 0.8974 - val_loss: 0.6899 - val_accuracy: 0.7200
7/7 [==============================] - 0s 1ms/step - loss: 0.6514 - accuracy: 0.6908
```

FINE

Grazie per l’attenzione