天津大学

Big Data Analysis Report

Big Data Analysis on CS:GO Game Mechanism based on Spark

Major: Smart Grid and Information Engineering

Name: Zhang Yiming

StudentID: 3019234068

□Independent Project ☑Team Project

2022. 6. 1

ABSTRACT

In this report, we mainly analyzed four part of the game mechanism and players behavior preferences in the video game CS:GO from Valve, which are weapon system prosperities, economics comparison and the impact on other factor, grenade using and its prosperities and finally the analysis of mechanism associated with locations on maps by using the PySpark interfaces of Apache Spark running on distributed cluster servers based on Hadoop. And from which we deduced and analyzed the design cretira and players preferences.

Keywords: CS:GO, Bigdata, Hadoop, Spark, Data Analysis

Contents

List of Figures

Chapter I

Big Data Analysis

At present time there are flood of data to be analyzed, and it is totally different from simple sample analyzing at a single node.

Big data refers to data sets that are too large or complex to be dealt with by traditional data-processing application software. Data with many fields (rows) offer greater statistical power, while data with higher complexity (more attributes or columns) may lead to a higher false discovery rate. Big data analysis challenges include capturing data, data storage, data analysis, search, sharing, transfer, visualization, querying, updating, information privacy, and data source. Big data was originally associated with three key concepts: volume, variety, and velocity. The analysis of big data presents challenges in sampling, and thus previously allowing for only observations and sampling. Therefore, big data often includes data with sizes that exceed the capacity of traditional software to process within an acceptable time and value.[2]

In order to store and process large scale of data, a highly available and reliable open source distributed computing platform Hadoop is needed. Hadoop has a complete set of ecosystem, HDFS for distributed file system, YARN for distributed resource managing, and we choose Apache Spark as computing programming platform.

Chapter II

Background

In this report we analyzed CS:GO game mechanics by choosing the original dataset of more than 500,000 game play data from ESEA demos.

2.1 Introduction

Video games are a rich area for data extraction due to their digital nature. Notable examples such as the complex EVE Online economy, World of Warcraft corrupted blood incident and even Grand Theft Auto self-driving cars tells us that fiction is closer to reality than we really think. Data scientists can gain insight on the logic and decision-making that the players face when put in hypothetical and virtual scenarios.

There is a Kaggle Dataset provided just over 1400 competitive matchmaking matches from Valve's game Counter-strike: Global Offensive (CS:GO). The data was extracted from competitive matchmaking replays submitted to csgo-stats.

We will analyze the mechanism and player behaviors by using these demo data.

2.2 About Counter-Strike: Global Offensive

Counter-Strike: Global Offensive is a first-person shooter game pitting two teams of 5 players against each other. Within a maximum of 30 rounds, the two teams find themselves on either side as a Counter Terrorist or Terrorist. Both sides are tasked with eliminating the opposition or, as the terrorist team, planting the C4 bomb at a bomb site and allowing it to explode. Rounds are played out until either of those two objectives or if the maximum time is reached (in which

the counter terrorists then win by default). At the end of the 15th round, the two teams switch sides and continue until one team reaches 16 round wins first. CS:GO is widely known for its competitive aspect of technical skill, teamwork and in-game strategies. Players are constantly rewarded with the efforts they put it in training and learning through advancing in rank.

More competitive mechanics can be found at Wikipedia.[3]

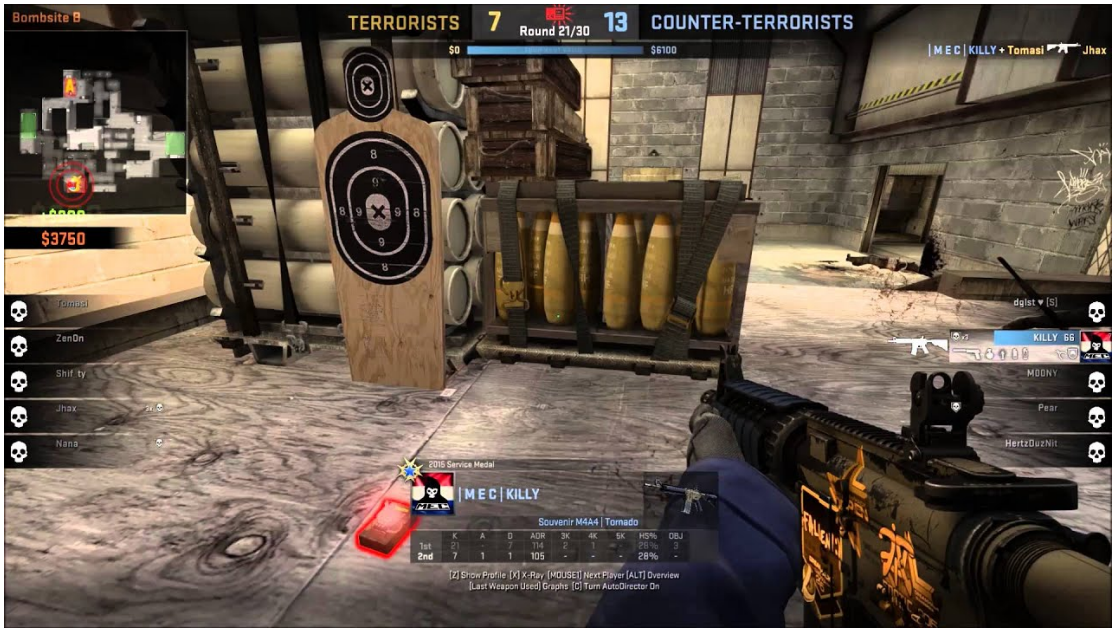


Fig. 2-1 CS:GO gaming display

2.3 Analyzing Propose

By analyzing the game mechanics and player behavior characteristics of CS:GO in terms of map win rate, weapon characteristics, attack points, economic impact, etc., we can help CS:GO players better win the game, and can also brings inspiration to game developers of the same type.

We used data counting analysis, data distribution density analysis, heat map analysis, relative analysis in this report.

Chapter III

Research Goal

There are several analysis targets around CS:GO game system design. Including weapon system analysis, the analysis of economic system and the association between economic system and game systems such as grenade buying and winning chances.

We still want to analyze gamer behavior in the game, including grenade using statistics, attacking position heat graph and winning chance associated with defending locations.

All analysis goals are shown as Fig. 3-1, there are 4 parts, ***25 research goals*** in total.

To expand, we mainly study the following four parts of game matching mechanism and gamer behaviors.

3.1 Weapon System

We mainly ranking total damage by different weapons to find out which weapons are most popular to the gamers. Here are these two targets.

- **Weapon Type Damage Rank**
- **Weapon Damage Rank**

Then we wanted to know which weapons resulted in the most kills, and analyzed the favorite weapons of players:

- **Kill counting of Weapons**
- **Most kills of Weapon when 1 CT alive**

These following targets shown which part of body dose players prefer to attack.

- **Hitbox Damage**

Fig. 3-1 Analysis Goals

- **Hitbox Records**

And we also care about the properties of different weapons, like head shooting rate and the average damage per shot.

- **Head Shooting Rate of Weapons**
- **Power of Weapons**

Lastly we want to analyze some target associated with game mechanism, like weapon choices in the ECO rounds.

- **ECO round kill weapons**

3.2 Economic Impact

First to find out what is the most type in all matching play rounds.

- **Round Types Count**

Then we collect the total equipment value of of both T side and CT side in the every matching play rounds, and compare the economics situation between both sides by analyzing the density distribution of both values and calculate the statistics numbers of the economics.

- **Density Curve**
- **Number Statistics**

And we also want to analyze the connection among the economics, winning chances and grenade buying situation, in order to see if they are relative.

- **Coupled with Winning Chances**
- **Coupled with Grenade Using**

3.3 Grenade Analysis

CS:GO is a hard core game basically depending on the level of aiming and positioning, but grenades are essential part of managing the game play. We wonder does grenade still matter a lot in matching game plays, and if *High-Explosive* grenades have any effect in the game.

We mainly have these two targets in this section:

- **Grenades Counting**
- **victims of 1 HE grenade**

And the grenades landing locations pointed on maps. Which is similar to the heat map analysis.

- **Grenade Locations**

3.4 Map Analysis

The first is the analysis of the winning percentage of different maps by analyzing the wins of both T and CT side in all rounds. The winning percentage of each map is calculated to judge the balance of maps.

Combined with the placement of the bombs, we can evaluate the impact of planting bombs at different points on different maps on the winning rate of both sides.

- **Overall Map Winning Chances**
- **Winning Chances associated with Bomb**
- **Planting site counts in different Maps**
- **Winning Chances of bomb planting situation**

According to the number advantage (or disadvantage) of T relative to CT after the bomb is installed, the winning rate under different numbers of people can be analyzed.

After the bomb is planted, the defensive position of the T side is also very important. When there are 2 people left in the T, we choose the T attack point as one person inside the bomb point and one person outside, and analyze the ADR in this situation.

- **Post-plant Win Probabilities by Advantages**
- **In/Out-of-site ADR**

Select the attack points of T and CT in different maps, mark them on the map, and draw heat maps of the most attack points of both sides.

- **Heatmaps of Frequency of Overall Damage**

Chapter IV

Experimental Data

4.1 Introduction

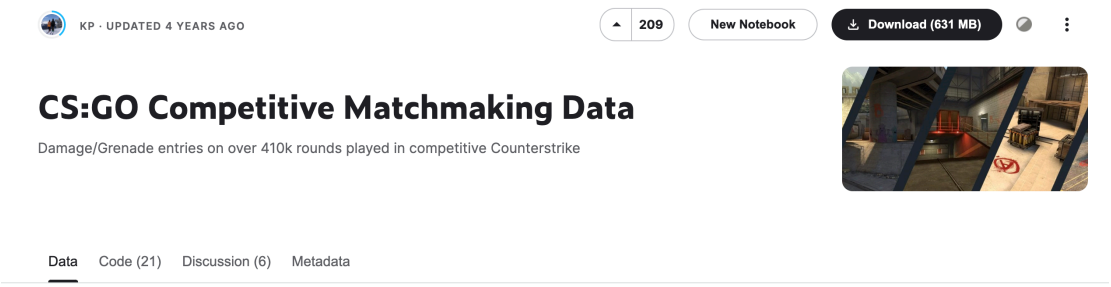


Fig. 4-1 Kagle Dataset

We use the dataset of ***CS:GO Competitive Matchmaking Data*** from Kaggle, including Damage/Grenade entries on over 410k rounds played in competitive Counterstrike.

This Dataset use CC BY-NC-SA 4.0 License, the Kagle dataset page is `https://www.kaggle.com/datasets/skihikingkevin/csgo-matchmaking-damage` you can visit the webpage and download the entire dataset.

This dataset includes 2D overhead view pictures of each map used for map point analysis, as well as kills, damages, and grenade entries of all rounds stored in `csv` files. The contents are shown as Fig. 4-2

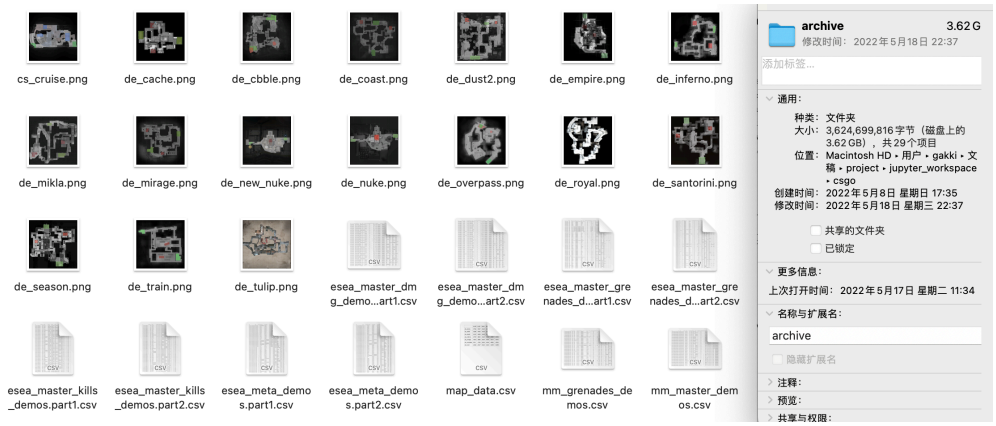


Fig. 4-2 Dataset Contents

There are two main part of the entire dataset.

One is the entries of ESEA matches in September 2018, which is new and stored in hierarchical multi tables since there are more than 500,000 damage records in average. The demos were collected over only a small span of two weeks in August and is definitely not representative of CS games over time.

According to the uploader of this dataset, the new ESEA demos are structured a bit differently to save space. The per-round information (e.g ct, t equipment value, winner, map) are all saved inside `esea_meta_demos.csv`. The other files have the same format, less the meta-columns. The kills data have also been added but follow a slightly different format, see the file detail for more info. Any demos that had less than 16 rounds played were excluded, no other data cleaning and assurance was done. It is sure there are a lot of matches that spat out weird data.[4]

The other is common matches entries in early time, and relatively has a small amount of data.

This dataset provides every successful entry of duels (or battle) that took place for a player. That is, each row documents an event when a player is hurt by another player (or World e.g fall damage).

4.2 Datasets Hierarchical Structure

The `csv` files are shown as Fig. 4-3, the dataset of ESEA matching records have `esea_` in front, the dataset of common rank-related competitive plays have `mm_` in front.

There is also a `map_data.csv` containing map bound position and map resolu-

Fig. 4-3 `csv` files in dataset

tion for converting person or grenade locations.

In the ESEA match dataset, there are four main tables, `meta_demos`, `master_dmg_demos` , `master_kills_demos` and `master_grenades_demos`, all separated into 2 parts, the connection of these four tables are shown as Fig. 4-4

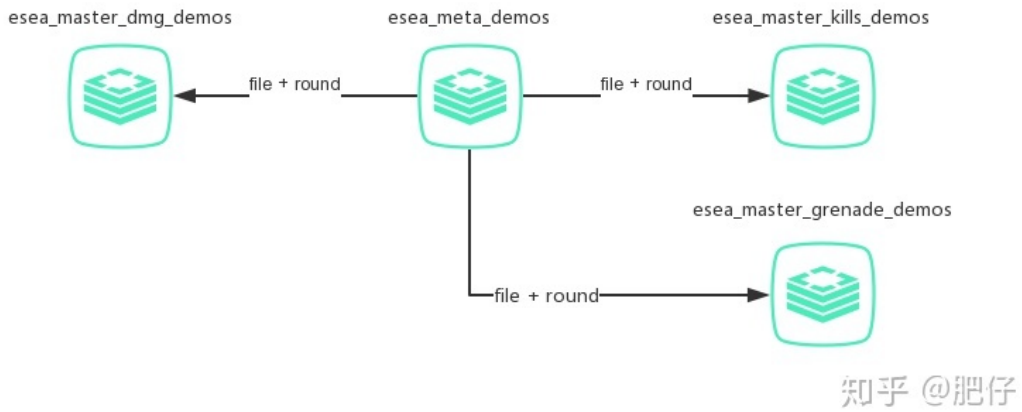


Fig. 4-4 Dataset Hierarchical Structure[1]

The `meta_demos` contains meta data of each play round, e.g. which play(`file`), `round`, `map`, `win_side`, total equipment value, etc.

In the `dmg_demos` table, each row is a damaging event took place by players, the meta information can be found in `meta_demos` table.

In the `kill_demos` table, each row represent a killing event, it can be found in

`dmg_demos` table as the last damage to some player.

The `grenades_demos` table contains grenade using event, has a field of `nade_land_pos` for grenade landing location, the damage records also can be found in `dmg_demos` table.

Use `file + round` field can locate same play round among 4 tables, and use `file + round + tick/seconds` field can locate a same event among these tables.

4.3 Tables display

The fields in ESEA match dataset table are shown as Fig. 4-5, each row stands for the same field among four dataset tables.

meta	master_dmg	master_kill	master_grenade
file	file	file	file
round	round	round	round
	tick	tick	
start_seconds	seconds	seconds(round)	seconds
end_seconds	att_team	att_team	att_team
winner_team	vic_team	vic_team	vic_team
winner_side	att_side	att_side	att_side
round_type	vic_side	vic_side	vic_side
ct_eq_val 装备价值	hp_dmg		hp_dmg
t_eq_val	arm_dmg		arm_dmg
	is_bomb_planted	is_bomb_planted	is_bomb_planted
	bomb_site		bomb_site
	hitbox		hitbox
	wp	wp	
	wp_type	wp_type	
	att_id		att_id
	att_rank 0		att_rank 0
	vic_id		vic_id
	vic_rank 0		vic_rank 0
	att_pos_x		att_pos_x
	att_pos_y		att_pos_y
map	vic_pos_x		vic_pos_x
	vic_pos_y		vic_pos_y
			nade
			nade_land_x
			nade_land_y
		ct_alive	
		t_alive	

Fig. 4-5 Fields in ESEA Dataset Tables

Among these the ESEA datasets, in `dmg` and `gre` tables there are position records of attack player and victim player, the `gre` tables also have the record of

nade_land position for map plotting analysis. Both two tables have hitbox records.

Three detailed tables have the is_bomb_planted flag, but only dmg and gre tables have the bomb_site information.

Both dmg and kill tables have weapon and weapon type records, which can be used for weapon prosperities analysis.

In heat map analysis we particularly care about the mm_master table and map_data, so here we merely list the fields of these two tables among common competitive demos dataset.

Tables Details:

- mm_master_demos:

The field in this table are as following:

```
,file,map,date,round,tick,seconds,att_team,vic_team,att_side,
vic_side,hp_dmg,arm_dmg,is_bomb_planted,bomb_site,hitbox,wp,
wp_type,award,winner_team,winner_side,att_id,att_rank,vic_id,
vic_rank,att_pos_x,att_pos_y,vic_pos_x,vic_pos_y,round_type,
ct_eq_val,t_eq_val,avg_match_rank
```

The common competitive plays are basically stored in one single giant table, which can carry out weapon system analysis, map plotting and economics analysis without joining the other tables.

It should be noticed that the alive players on both side are missing in the mm_master_demos table, but they can be evaluate through a series of processes.

- map_data:

Table 4-1 map_data table

	EndX	EndY	ResX	ResY	StartX	StartY
de_cache	3752	3187	1024	1024	-2031	-2240
de_cbble	2282	3032	1024	1024	-3819	-3073
de_dust2	2127	3455	1024	1024	-2486	-1150
de_inferno	2797	3800	1024	1024	-1960	-1062
de_mirage	1912	1682	1024	1024	-3217	-3401
de_overpass	503	1740	1024	1024	-4820	-3591
de_train	2262	2447	1024	1024	-2436	-2469

The `map_data` file are shown as Table 4-1, containing the start point and the end point positions, and the resolution pixels per each coordinate unit, providing a way to plot on maps.

- `meta_demos`:

The `meta_demos` table are shown as Fig. 4-6

	A	B	C	D	E	F	G	H	I	J
1	file	map	round	start_seconds	end_seconds	winner_team	winner_side	round_type	ct_eq_val	t_eq_val
2	esea_match_13770997.dem	de_overpass	1	94.30782	160.9591	Hentai Hooligans	Terrorist	PISTOL_ROUND	4300	4250
3	esea_match_13770997.dem	de_overpass	2	160.9591	279.3998	Hentai Hooligans	Terrorist	ECO	6300	19400
4	esea_match_13770997.dem	de_overpass	3	279.3998	341.0084	Hentai Hooligans	Terrorist	SEMI_ECO	7650	19250
5	esea_match_13770997.dem	de_overpass	4	341.0084	435.4259	Hentai Hooligans	Terrorist	NORMAL	24900	23400
6	esea_match_13770997.dem	de_overpass	5	435.4259	484.2398	Animal Style	CounterTerrorist	ECO	5400	20550
7	esea_match_13770997.dem	de_overpass	6	484.2398	604.5598	Hentai Hooligans	Terrorist	NORMAL	29650	25450
8	esea_match_13770997.dem	de_overpass	7	604.5598	684.2563	Hentai Hooligans	Terrorist	ECO	3200	25300
9	esea_match_13770997.dem	de_overpass	8	684.2563	763.8902	Animal Style	CounterTerrorist	ECO	4850	27600
10	esea_match_13770997.dem	de_overpass	9	763.8902	844.3071	Animal Style	CounterTerrorist	FORCE_BUY	32150	18200
11	esea_match_13770997.dem	de_overpass	10	844.3071	937.0176	Animal Style	CounterTerrorist	ECO	32950	9950
12	esea_match_13770997.dem	de_overpass	11	937.0176	1031.012	Animal Style	CounterTerrorist	ECO	30100	10350
13	esea_match_13770997.dem	de_overpass	12	1031.012	1089.536	Animal Style	CounterTerrorist	NORMAL	30150	22800
14	esea_match_13770997.dem	de_overpass	13	1089.536	1171.988	Animal Style	CounterTerrorist	NORMAL	31750	19900
15	esea_match_13770997.dem	de_overpass	14	1171.988	1238.029	Hentai Hooligans	Terrorist	SEMI_ECO	32750	14050
16	esea_match_13770997.dem	de_overpass	15	1238.029	1299.653	Animal Style	CounterTerrorist	NORMAL	22000	27450
17	esea_match_13770997.dem	de_overpass	17	1379.35	1513.482	Hentai Hooligans	CounterTerrorist	ECO	20450	5000
18	esea_match_13770997.dem	de_overpass	18	1513.482	1566.337	Hentai Hooligans	CounterTerrorist	ECO	21550	1500
19	esea_match_13770997.dem	de_overpass	19	1566.337	1670.479	Hentai Hooligans	CounterTerrorist	NORMAL	24650	24000
20	esea_match_13770997.dem	de_overpass	20	1670.479	1786.007	Hentai Hooligans	CounterTerrorist	ECO	31400	4450

Fig. 4-6 Contents of `meta` table

The data are stored in two separated tables, each row stand for a play round information, and there are 377,629 records in total.

- `dmg_demos`:

The `dmg_demos` table are shown as Fig. 4-7

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
1	file	rour	tick	seconds	att_team	vic_team	att_side	vic_side	hp_dmg	arm_dmg	is_bomb_planted	bor	hitbox	wp	wp_type	att_id	att_r	vic_id	vic_att_pos_x	att_pos_y	vic_pos_x	vic_pos_y	
2	esea_matc	1	14372	111.8476	World	Animal Style	None	CounterTerrorist	1	0	FALSE		Generic	Unknown	Unkown	0	0	8E+16	0	0	0	0	0
3	esea_matc	1	15972	124.3761	Animal Style	Hentai Hooligans	CounterTerrorist	Terrorist	18	9	FALSE		Stomach	USP	Pistol	7.7E+16	0	8E+16	0	-1499.69	63.33829	-669.5558	-79.76957
4	esea_matc	1	16058	125.0495	Animal Style	Hentai Hooligans	CounterTerrorist	Terrorist	100	0	FALSE		Head	USP	Pistol	7.7E+16	0	8E+16	0	-1066.874	3.44563	-614.1868	-91.70777
5	esea_matc	1	16066	125.1121	Hentai Hooligans	Animal Style	Terrorist	CounterTerrorist	12	7	FALSE		RightArm	Glock	Pistol	7.7E+16	0	8E+16	0	-747.3146	-49.32681	-1065.556	9.381622
6	esea_matc	1	16108	125.441	Animal Style	Hentai Hooligans	CounterTerrorist	Terrorist	15	7	FALSE		Chest	USP	Pistol	7.7E+16	0	8E+16	0	-1501.861	49.19798	-748.4188	-53.46922
7	esea_matc	1	16188	126.0674	Hentai Hooligans	Animal Style	Terrorist	CounterTerrorist	94	0	FALSE		Head	Glock	Pistol	7.7E+16	0	8E+16	0	-756.2186	-80.94859	-1500.078	60.65815
8	esea_matc	1	16210	126.2397	Hentai Hooligans	Animal Style	Terrorist	CounterTerrorist	6	0	FALSE		Head	Glock	Pistol	7.7E+16	0	8E+16	0	-751.6216	-55.69779	-1499.893	61.62376
9	esea_matc	1	16496	128.4791	Animal Style	Hentai Hooligans	CounterTerrorist	Terrorist	13	6	FALSE		Chest	USP	Pistol	7.7E+16	0	8E+16	0	-1878.242	527.7095	-632.5327	69.1092
10	esea_matc	1	16510	128.5888	Hentai Hooligans	Animal Style	Terrorist	CounterTerrorist	87	0	FALSE		Head	Glock	Pistol	7.7E+16	0	8E+16	0	-1025.829	-487.7901	-1051.822	98.66519
11	esea_matc	1	16514	128.6201	Hentai Hooligans	Animal Style	Terrorist	CounterTerrorist	9	5	FALSE		Chest	Glock	Pistol	7.7E+16	0	8E+16	0	-702.2363	93.98489	-1877.25	530.4081
12	esea_matc	1	16530	128.7454	Animal Style	Hentai Hooligans	CounterTerrorist	Terrorist	13	6	FALSE		RightArm	USP	Pistol	7.7E+16	0	8E+16	0	-1875.663	534.7057	-701.8441	95.10104
13	esea_matc	1	16530	128.7454	Animal Style	Hentai Hooligans	CounterTerrorist	Terrorist	3	1	FALSE		Chest	USP	Pistol	7.7E+16	0	8E+16	0	-1875.663	534.7057	-642.4961	60.08743
14	esea_matc	1	16562	128.9959	Animal Style	Hentai Hooligans	CounterTerrorist	Terrorist	13	6	FALSE		RightArm	USP	Pistol	7.7E+16	0	8E+16	0	-1880.191	522.7754	-705.1382	86.38733
15	esea_matc	1	17014	132.5352	Hentai Hooligans	Animal Style	Terrorist	CounterTerrorist	9	5	FALSE		Chest	Glock	Pistol	7.7E+16	0	8E+16	0	-789.0594	156.5308	-1878.863	518.6008
16	esea_matc	1	17048	132.8015	Hentai Hooligans	Animal Style	Terrorist	CounterTerrorist	9	5	FALSE		Chest	Glock	Pistol	7.7E+16	0	8E+16	0	-788.0546	159.6495	-1882.112	507.5257
17	esea_matc	1	17104	133.2399	Animal Style	Hentai Hooligans	CounterTerrorist	Terrorist	41	0	FALSE		Head	USP	Pistol	7.7E+16	0	8E+16	0	-1882.543	506.9022	-782.783	176.0159
18	esea_matc	1	17252	134.3988	Hentai Hooligans	Animal Style	Terrorist	CounterTerrorist	92	0	FALSE		Head	Glock	Pistol	7.7E+16	0	8E+16	0	-1089.135	-194.6018	-1890.576	-358.8826
19	esea_matc	1	17338	135.0722	Hentai Hooligans	Animal Style	Terrorist	CounterTerrorist	8	0	FALSE		Head	Glock	Pistol	7.7E+16	0	8E+16	0	-1104.001	-127.941	-1888.107	-367.5803
20	esea_matc	1	17426	135.7613	Hentai Hooligans	Animal Style	Terrorist	CounterTerrorist	7	3	FALSE		Chest	Glock	Pistol	7.7E+16	0	8E+16	0	-1154.541	-64.86525	-1849.73	579.1832

Fig. 4-7 Contents of `dmg` table

The data are stored in two separated tables, each row stand for a damaging event took place in the match, and there are 10,538,182 records in total.

- `kill_demos`:

The `kill_demos` table are shown as Fig. 4-8

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	file	round	tick	seconds	att_team	vic_team	att_side	vic_side	wp	wp_type	ct_alive	t_alive	is_bomb_planted
2	esea_match_13770997.dem	1	16058	30.74165	Animal Style	Hentai Hooligans	CounterTerrorist	Terrorist	USP	Pistol	5	4	FALSE
3	esea_match_13770997.dem	1	16210	31.93185	Hentai Hooligans	Animal Style	Terrorist	CounterTerrorist	Glock	Pistol	4	4	FALSE
4	esea_match_13770997.dem	1	16510	34.28094	Hentai Hooligans	Animal Style	Terrorist	CounterTerrorist	Glock	Pistol	3	4	FALSE
5	esea_match_13770997.dem	1	17104	38.93212	Animal Style	Hentai Hooligans	CounterTerrorist	Terrorist	USP	Pistol	3	3	FALSE
6	esea_match_13770997.dem	1	17338	40.76441	Hentai Hooligans	Animal Style	Terrorist	CounterTerrorist	Glock	Pistol	2	3	FALSE
7	esea_match_13770997.dem	1	17524	42.22083	Hentai Hooligans	Animal Style	Terrorist	CounterTerrorist	Glock	Pistol	1	3	FALSE
8	esea_match_13770997.dem	1	17814	44.49162	Animal Style	Hentai Hooligans	CounterTerrorist	Terrorist	USP	Pistol	1	2	FALSE
9	esea_match_13770997.dem	1	18662	51.13169	Animal Style	Hentai Hooligans	CounterTerrorist	Terrorist	USP	Pistol	1	1	FALSE
10	esea_match_13770997.dem	1	20000	61.60859	Hentai Hooligans	Animal Style	Terrorist	CounterTerrorist	Glock	Pistol	0	1	TRUE
11	esea_match_13770997.dem	2	26160	43.03519	Hentai Hooligans	Animal Style	Terrorist	CounterTerrorist	AK47	Rifle	4	5	FALSE
12	esea_match_13770997.dem	2	26190	43.27008	Hentai Hooligans	Animal Style	Terrorist	CounterTerrorist	AK47	Rifle	3	5	FALSE
13	esea_match_13770997.dem	2	26626	46.68408	Animal Style	Hentai Hooligans	CounterTerrorist	Terrorist	Deagle	Pistol	3	4	FALSE
14	esea_match_13770997.dem	2	26634	46.74673	Hentai Hooligans	Animal Style	Terrorist	CounterTerrorist	AK47	Rifle	2	4	FALSE
15	esea_match_13770997.dem	2	28032	57.69345	Animal Style	Hentai Hooligans	CounterTerrorist	Terrorist	CZ	Pistol	2	3	FALSE
16	esea_match_13770997.dem	2	28154	58.64874	Animal Style	Hentai Hooligans	CounterTerrorist	Terrorist	CZ	Pistol	2	2	FALSE
17	esea_match_13770997.dem	2	28406	60.62198	Hentai Hooligans	Animal Style	Terrorist	CounterTerrorist	AK47	Rifle	1	2	FALSE
18	esea_match_13770997.dem	3	40853	39.46457	Animal Style	Hentai Hooligans	CounterTerrorist	Terrorist	AK47	Rifle	5	4	FALSE
19	esea_match_13770997.dem	3	41141	41.7197	Hentai Hooligans	Animal Style	Terrorist	CounterTerrorist	AK47	Rifle	4	4	FALSE
20	esea_match_13770997.dem	3	42263	50.50528	Hentai Hooligans	Animal Style	Terrorist	CounterTerrorist	Negev	Heavy	3	4	FALSE

Fig. 4-8 Contents of `kill` table

The data are stored in two separated tables, each row stand for a killing event took place in the match, and there are 2,742,646 records in total.

- `grenades_demos`:

The `grenades_demos` table are shown as Fig. 4-9

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
1	file	round	seconds	att_team	vic_team	att_id	vic_id	att_side	vic_side	hp_dmg	arm_dmg	is_bomb_p	bomb_site	hitbox	nade	att_rank	vic_rank	att_pos_x	att_pos_y	nade_land	nade_land	vic_pos_x	vic_pos_y
2	esea_matc	1	153.1602	Animal Style		7.66E+16		CounterTerrorist		0	0	TRUE	B		Smoke	0		-1618.15	-66.0026	-949.857	-340.302		
3	esea_matc	2	184.7945	Hentai Hoo	Animal Sty	7.66E+16	7.66E+16	Terrorist	CounterTe	70	0	FALSE		Generic	HE	0	0	-1719.9	-2357.65	-2774.67	-1603.94	-2741.25	-1523.16
4	esea_matc	2	186.8617	Animal Style		7.66E+16		CounterTerrorist		0	0	FALSE			HE	0		-1036.35	492.1676	-466.868	-356.964		
5	esea_matc	2	187.1122	Animal Style		7.66E+16		CounterTerrorist		0	0	FALSE			HE	0		-855.077	438.6909	-459.015	-543.858		
6	esea_matc	2	191.0587	Hentai Hooligans		7.66E+16		Terrorist		0	0	FALSE			Molotov	0		-2617.49	-1832.41	-2743.56	-927.3		
7	esea_matc	2	191.4502	Animal Style		7.66E+16		CounterTerrorist		0	0	FALSE			Smoke	0		-854.771	427.4865	-490.528	-652.898		
8	esea_matc	2	191.6225	Animal Style		7.66E+16		CounterTerrorist		0	0	FALSE			Flash	0		-1948.91	-1184.2	-1780.66	-976.403		
9	esea_matc	2	204.7617	Hentai Hooligans		7.66E+16		Terrorist		0	0	FALSE			HE	0		-1182.09	-931.698	-1726.53	-428.848		
10	esea_matc	2	225.5276	Hentai Hooligans		7.66E+16		Terrorist		0	0	FALSE			Smoke	0		-1436.79	-720.22	-1122.9	-510.926		
11	esea_matc	3	304.3628	Hentai Hooligans		7.66E+16		Terrorist		0	0	FALSE			HE	0		-1734.94	-2341.19	-2860.17	-1505.67		
12	esea_matc	3	306.8371	Hentai Hooligans		7.66E+16		Terrorist		0	0	FALSE			Flash	0		-2115.31	-1922.65	-2684.47	-1329.79		
13	esea_matc	4	363.9355	Animal Sty	Hentai Hoo	7.66E+16	7.66E+16	CounterTe	Terrorist	1	0	FALSE		Generic	Incendiary	0	0	-2339.55	-358.022	-1823.1	-2271.76	-1872.84	-2442.81
14	esea_matc	4	363.9355	Animal Sty	Hentai Hoo	7.66E+16	7.66E+16	CounterTe	Terrorist	1	0	FALSE		Generic	Incendiary	0	0	-2339.55	-358.022	-1823.1	-2271.76	-1713.42	-2356.09
15	esea_matc	4	363.9355	Animal Sty	Hentai Hoo	7.66E+16	7.66E+16	CounterTe	Terrorist	2	0	FALSE		Generic	Incendiary	0	0	-2339.55	-358.022	-1823.1	-2271.76	-1748.88	-2336.29
16	esea_matc	4	363.9355	Animal Sty	Hentai Hoo	7.66E+16	7.66E+16	CounterTe	Terrorist	2	0	FALSE		Generic	Incendiary	0	0	-2339.55	-358.022	-1823.1	-2271.76	-1677.46	-2380.63
17	esea_matc	4	363.9355	Animal Sty	Hentai Hoo	7.66E+16	7.66E+16	CounterTe	Terrorist	3	0	FALSE		Generic	Incendiary	0	0	-2339.55	-358.022	-1823.1	-2271.76	-1773.39	-2305.95
18	esea_matc	4	363.9355	Animal Sty	Hentai Hoo	7.66E+16	7.66E+16	CounterTe	Terrorist	3	0	FALSE		Generic	Incendiary	0	0	-2339.55	-358.022	-1823.1	-2271.76	-1674.07	-2380.29
19	esea_matc	4	363.9355	Animal Sty	Hentai Hoo	7.66E+16	7.66E+16	CounterTe	Terrorist	4	0	FALSE		Generic	Incendiary	0	0	-2339.55	-358.022	-1823.1	-2271.76	-1788.36	-2264.08
20	esea_matc	4	363.9355	Animal Sty	Hentai Hoo	7.66E+16	7.66E+16	CounterTe	Terrorist	4	0	FALSE		Generic	Incendiary	0	0	-2339.55	-358.022	-1823.1	-2271.76	-1802.62	-2235.25

Fig. 4-9 Contents of `gre` table

The data are stored in two separated tables, each row stand for a event took place by grenades in the match, and there are 5,246,458 records in total.

Chapter V

Analysis Method

5.1 Architecture

The Apache™ Hadoop® project develops open-source software for reliable, scalable, distributed computing.

The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage.[5]

Hadoop has high availability and has distributed file storing and resource managing frameworks, and it has a grand ecosystem supporting many kinds of computing applications and APIs, such as Apache Spark, which supports Python programming.

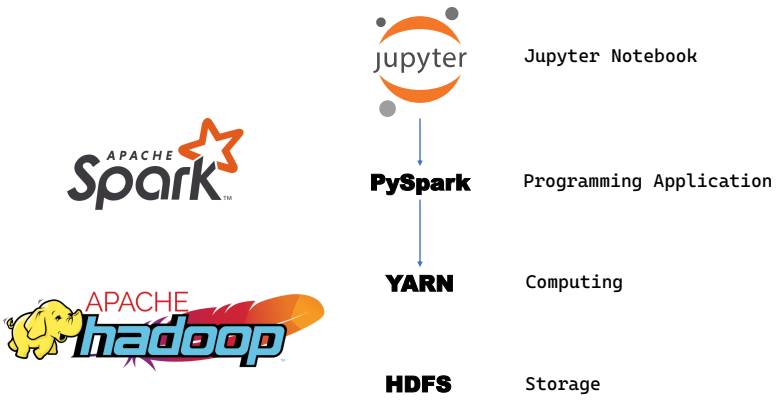


Fig. 5-1 Analysis Architecture

We use Spark running on YARN, and use Hadoop HDFS for distributed file system to store input files and output results.

Using PySpark APIs to program in Python on Jupyter Notebook, which will reduce the difficulty of writing programs, enhance the readability of the code, and speed up the development process.

5.1.1 Analyzing Environment

To maximize the use of distributed computing, we choose to use the distributed cluster of the Tianjin University laboratory as the platform to run the program. Which are 3 Linux virtual machine servers as our cluster nodes. One node for master node of Spark, namenode of HDFS and Resource Manager of YARN; and two nodes for worker of Spark, Datanode of HDFS.

The environment prosperities are listed below:

- All nodes are CentOS 8 Linux 4.18.0-383.el8.x86_64
- Hadoop: 2.7.5
- JAVA: 1.8.0_161
- Spark: 2.2.1, Scala: 2.11.8
- Python: 3.7.3
- Jupyter: 4.4.0

And system information are listed below:

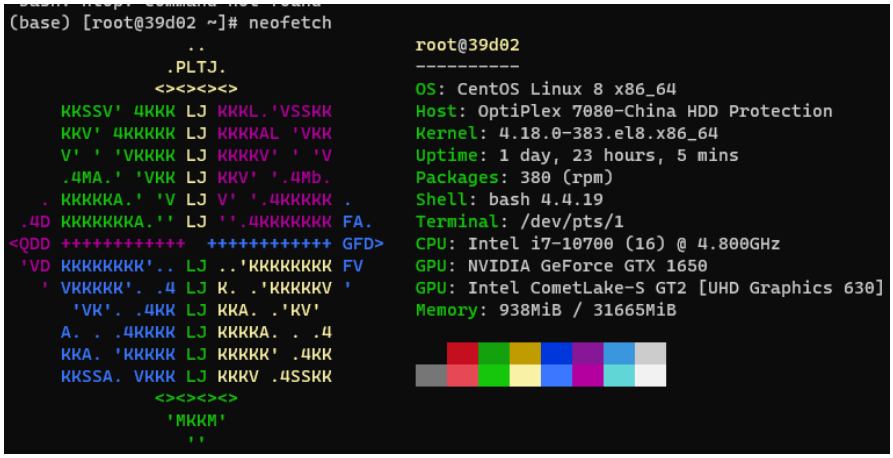


Fig. 5-2 System Information

As we constructed the distributed computing platform on the cluster, we run the pyspark application on one of the nodes by client mode, using the following command under the base environment of the root user:

```
PYSPARK_DRIVER_PYTHON=jupyter PYSPARK_DRIVER_PYTHON_OPTS='notebook --
    allow-root' HADOOP_CONF_DIR=/opt/bigdata/hadoop/etc/hadoop pyspark
     --master yarn --deploy-mode client
```

5.1.2 Data Storing

All dataset files are uploaded to the HDFS on cluster servers by using JAVA APIs. And the map pictures are copied from HDFS to local on the node for map plotting analysis.

Jupyter Notebooks are stored in the local node, which could be access from webpage on port `9595`.

5.2 Analyzing Flow

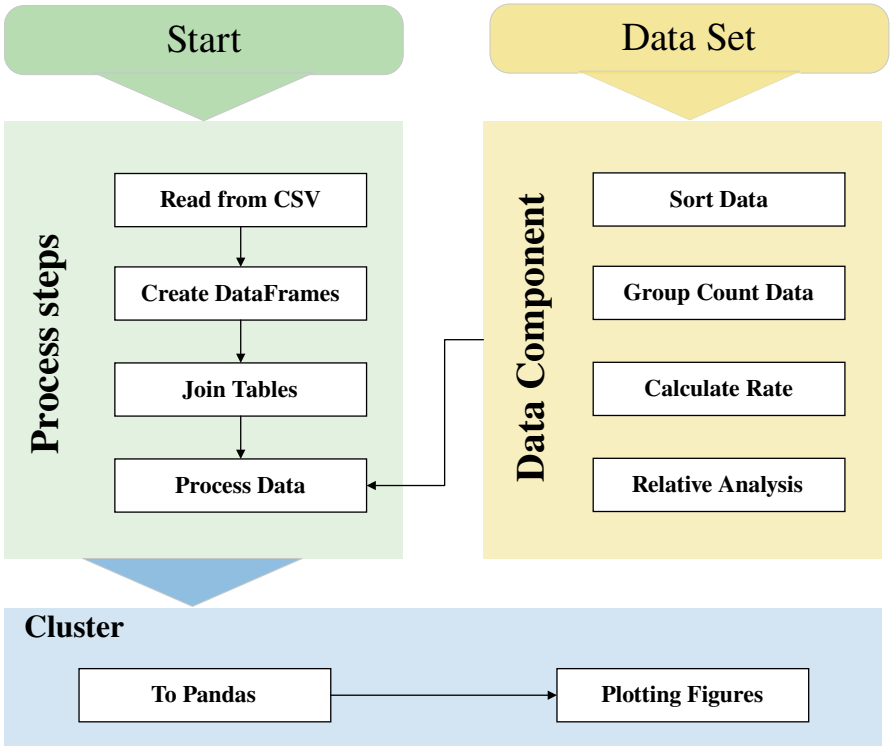


Fig. 5-3 Analyzing Flow

Use the sqlContext environment of Spark Session throughout the process, import data from csv files in HDFS, and create RDDs; create DataFrames from RDDs, and link different tables. Here we have numbers of DataFrames.

Perform operations such as grouping statistics, sorting, and calculating proportions on the DataFrame; after the calculation is completed, the results are

stored locally and sent to Pandas.

Here is a example of data process of counting bomb platting site of every rounds in the `mm_master_demos` table.

```
#统计炸弹安放点
site_count_pd=mm_rgl_df.filter("bomb_site!=''").select(['file', 'map'
    , 'round', 'bomb_site']).dropDuplicates().\
            groupby(['map', 'bomb_site']).count().toPandas()
#送入Pandas，转置，显示:
site_count_pd.set_index(['map','bomb_site']).unstack()
```

Convert Pandas output to LaTeX format, shown as Table 5-1.

Table 5-1 Example table of the bomb site counting to Pandas

	count	
bomb_site	A	B
map		
de_cache	1819	1375
de_cbble	346	434
de_dust2	2123	1537
de_inferno	1065	938
de_mirage	2544	1955
de_overpass	428	557
de_train	257	204

Data visualization: Use tools such as `plt` and `seaborn` to draw bar charts, pie charts, density distribution curves, and two-dimensional density distribution graphs for Pandas tabular data.

Here is the same example:

```
fig = plt.figure(figsize=(6, 8))
colors1=sns.color_palette("hls",2)
sns.barplot(y='count',x='map',hue='bomb_site',data=site_count_pd,
   palette=colors1)
plt.suptitle('Victims per HE', fontsize=16, fontweight='bold')
```

And output figure shown asFig. 5-4.

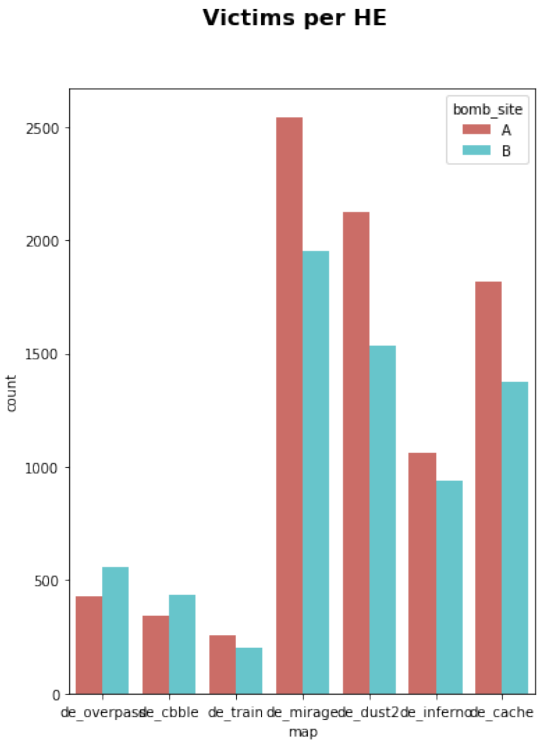


Fig. 5-4 Example Picture

5.3 Importing Data

Here explain and display the code relative to the data importing process, which is essential to the following steps in the flow.

Import the 6 `csv` tables required for the analysis into RDD, then fill in the fields according to the RDD, create a DataFrame, and finally form 6 DataFrames: `meta_df`, `dmg_df`, `kill_df`, `gre_df`, `mm_df` and the map bound data `map_bounds`.

Here we take example of the formation of **`meta_df`**, the rest 5 tables are similar.

First import `csv` files and cut *headers* then create RDDs.

Listing 5-1 Import `meta_demos`

```
#导入数据表
#表头
meta_header = sc.textFile("/ZHANGYIMING/csgodata/archive/
    esea_meta_demos.part1.csv").first()
#raw data 创建RDD
meta_1_RDD = sc.textFile("/ZHANGYIMING/csgodata/archive/
    esea_meta_demos.part1.csv").filter(lambda x:x !=meta_header).map(
    lambda x: x.split(","))
meta_2_RDD = sc.textFile("/ZHANGYIMING/csgodata/archive/
    esea_meta_demos.part2.csv").filter(lambda x:x !=meta_header).map(
```

```
    lambda x: x.split(","))
```

Union to one RDD, and print total entries:

```
meta_RDD=meta_1_RDD.union(meta_2_RDD)
meta_RDD.count()
```

Then fill the fields and create DataFrame from RDD:

```
sqlContext = SparkSession.builder.getOrCreate()
from pyspark.sql import Row
meta = meta_RDD.map(lambda p:
     Row(
         file=p[0],
         map=p[1],
         round=int(p[2]),
         start_seconds=p[3],
         end_seconds=p[4],
         winner_team=p[5],
         winner_side=p[6],
         round_type=p[7],
         ct_eq_val=int(p[8]),
         t_eq_val=int(p[9])
    )
)
meta_df = sqlContext.createDataFrame(meta)
meta_df.printSchema()
```

Here we can see the Schema of `meta_df`:

```
root
 |-- ct_eq_val: long (nullable = true)
 |-- end_seconds: string (nullable = true)
 |-- file: string (nullable = true)
 |-- map: string (nullable = true)
 |-- round: long (nullable = true)
 |-- round_type: string (nullable = true)
 |-- start_seconds: string (nullable = true)
 |-- t_eq_val: long (nullable = true)
 |-- winner_side: string (nullable = true)
 |-- winner_team: string (nullable = true)
```

Fig. 5-5 Schema of `meta_df`

And here is the top 5 records in the DataFrame:

```
meta_df.show(5)
```

ct_eq_val	end_seconds	file	map	round	round_type	start_seconds	t_eq_val	winner_side	winner_team
4300	160.9591	esea_match_137709...	de_overpass	1	PISTOL_ROUND	94.30781999999999	4250	Terrorist	Hentai Hooligans
6300	279.3998	esea_match_137709...	de_overpass	2	ECO	160.9591	19400	Terrorist	Hentai Hooligans
7650	341.0084	esea_match_137709...	de_overpass	3	SEMI_ECO	279.3998	19250	Terrorist	Hentai Hooligans
24900	435.4259	esea_match_137709...	de_overpass	4	NORMAL	341.0084	23400	Terrorist	Hentai Hooligans
5400	484.2398	esea_match_137709...	de_overpass	5	ECO	435.4259	20550	CounterTerrorist	Animal Style

only showing top 5 rows

Fig. 5-6 Show 5 records of `meta_df`

```
root
 |-- arm_dmg: string (nullable = true)
 |-- att_pos_x: string (nullable = true)
 |-- att_pos_y: string (nullable = true)
 |-- att_side: string (nullable = true)
 |-- att_team: string (nullable = true)
 |-- bomb_site: string (nullable = true)
 |-- file: string (nullable = true)
 |-- hitbox: string (nullable = true)
 |-- hp_dmg: string (nullable = true)
 |-- is_bomb_planted: string (nullable = true)
 |-- round: string (nullable = true)
 |-- seconds: string (nullable = true)
 |-- tick: string (nullable = true)
 |-- vic_pos_x: string (nullable = true)
 |-- vic_pos_y: string (nullable = true)
 |-- vic_side: string (nullable = true)
 |-- vic_team: string (nullable = true)
 |-- wp: string (nullable = true)
 |-- wp_type: string (nullable = true)
```

(a) `dmg` Schema

```
root
 |-- att_side: string (nullable = true)
 |-- att_team: string (nullable = true)
 |-- bomb_site: string (nullable = true)
 |-- ct_alive: string (nullable = true)
 |-- file: string (nullable = true)
 |-- is_bomb_planted: string (nullable = true)
 |-- round: string (nullable = true)
 |-- seconds: string (nullable = true)
 |-- t_alive: string (nullable = true)
 |-- tick: string (nullable = true)
 |-- vic_side: string (nullable = true)
 |-- vic_team: string (nullable = true)
 |-- wp: string (nullable = true)
 |-- wp_type: string (nullable = true)
```

(b) `kill` Schema

```
root
 |-- arm_dmg: string (nullable = true)
 |-- att_id: string (nullable = true)
 |-- att_pos_x: string (nullable = true)
 |-- att_pos_y: string (nullable = true)
 |-- att_side: string (nullable = true)
 |-- bomb_site: string (nullable = true)
 |-- file: string (nullable = true)
 |-- hitbox: string (nullable = true)
 |-- hp_dmg: string (nullable = true)
 |-- is_bomb_planted: string (nullable = true)
 |-- nade: string (nullable = true)
 |-- nade_land_x: string (nullable = true)
 |-- nade_land_y: string (nullable = true)
 |-- round: string (nullable = true)
 |-- seconds: string (nullable = true)
 |-- vic_id: string (nullable = true)
 |-- vic_pos_x: string (nullable = true)
 |-- vic_pos_y: string (nullable = true)
 |-- vic_side: string (nullable = true)
```

(c) `grenades` Schema

```
root
 |-- arm_dmg: string (nullable = true)
 |-- att_id: string (nullable = true)
 |-- att_pos_x: string (nullable = true)
 |-- att_pos_y: string (nullable = true)
 |-- att_side: string (nullable = true)
 |-- award: string (nullable = true)
 |-- bomb_site: string (nullable = true)
 |-- file: string (nullable = true)
 |-- hitbox: string (nullable = true)
 |-- hp_dmg: string (nullable = true)
 |-- is_bomb_planted: string (nullable = true)
 |-- map: string (nullable = true)
 |-- round: string (nullable = true)
 |-- round_type: string (nullable = true)
 |-- seconds: string (nullable = true)
 |-- tick: string (nullable = true)
 |-- vic_id: string (nullable = true)
 |-- vic_pos_x: string (nullable = true)
 |-- vic_pos_y: string (nullable = true)
 |-- vic_side: string (nullable = true)
 |-- winner_side: string (nullable = true)
 |-- wp: string (nullable = true)
 |-- wp_type: string (nullable = true)
```

(d) `mm` Schema

Fig. 5-7 Schemas of the rest of DataFrames

The rest four tables `dmg_df`, `kill_df`, `gre_df`, `mm_df` and `map_bounds` remain same as the formation of `meta_df`.

Here we put the Schemas of the rest of DataFrames except the `map_data` table, which is simple, as Fig. 5-7 shown.

We can join these DataFrames to more fields included DataFrame for the specific analysis, such as add field relating the map bounds info to the `mm_df` by joining the `mm_df` DataFrame and the converted map bounds information DataFrame.

Chapter VI

Weapon System Analysis

6.1 Damages Ranking

6.1.1 Weapon Type Damage Rank

There are 7 types of weapons, most records are common guns, according to Fig. 6-1.

```
#列出不同伤害的武器类型
dmg_df.select('wp_type').groupBy('wp_type').count().orderBy('count',
    ascending=0 ).toPandas()
```

	wp_type	count
0	Rifle	5338372
1	Pistol	2070588
2	Grenade	1280516
3	SMG	1133901
4	Sniper	490912
5	Heavy	110201
6	Unkown	103280
7	Equipment	10412

Fig. 6-1 Records of different weapon types

Group by `wp_type` and summarize the total damage, we can count the total damages of different weapon types.

Listing 6-1 Damages Rank by Weapon Types

```
#以wp_type区分的伤害，并计算总伤害
dmg_wp_list=dmg_df.select('wp','wp_type','hitbox','hp_dmg','arm_dmg'
    ,(dmg_df.hp_dmg+dmg_df.arm_dmg).alias("tot_dmg")).\
  orderBy('wp_type')

dmg_wp_type_sum=dmg_wp_list.groupBy('wp_type').agg({'hp_dmg': 'sum',
    'arm_dmg': 'sum','tot_dmg': 'sum'}).\
  withColumnRenamed("sum(tot_dmg)", "tot_dmg").\
  withColumnRenamed("sum(hp_dmg)", "hp_dmg").\
  withColumnRenamed("sum(arm_dmg)", "arm_dmg").\
  orderBy('tot_dmg',ascending=0)
dmg_wp_type_sum_pd=dmg_wp_type_sum.toPandas()
```

The sum result are shown as Fig. 6-2.

	wp_type	tot_dmg	hp_dmg	arm_dmg
0	Rifle	177745516.0	156448524.0	21296992.0
1	Pistol	76737824.0	67355345.0	9382479.0
2	Sniper	39695267.0	38710578.0	984689.0
3	SMG	27397275.0	23032905.0	4364370.0
4	Grenade	15249803.0	13021114.0	2228689.0
5	Heavy	2325035.0	2039414.0	285621.0
6	Unkown	1854692.0	1295327.0	559365.0
7	Equipment	583514.0	543252.0	40262.0

Fig. 6-2 Damages sum of different weapon types

Then draw graphs to illustrate the ranking of HP and ARM damages by different weapon types:

```
from matplotlib import ticker
fig = plt.figure(figsize=(18, 11))
plt.subplot(1,2,1)
plt.title('HP & Arm Demage of Different Weapon Types', fontsize=13,
    fontweight='bold')

plt.bar(dmg_wp_type_sum_pd['wp_type'], dmg_wp_type_sum_pd['hp_dmg'],
    label='HP')
plt.bar(dmg_wp_type_sum_pd['wp_type'], dmg_wp_type_sum_pd['arm_dmg'],
     bottom=dmg_wp_type_sum_pd['hp_dmg'], label='Arm')
plt.legend()

plt.subplot(1,2,2)
```

```
plt.title('Total Demage Percentage', fontsize=13, fontweight='bold',
    loc="left")
plt.pie(dmg_wp_type_sum_pd['tot_dmg'],labels=dmg_wp_type_sum_pd['
    wp_type'],startangle=90,autopct='%1.1f%%',explode =
    (0.01,0.02,0.03,0.05,0.07,0.15,0.3,0.5))
plt.suptitle('Weapon Type Demage Counting', fontsize=16, fontweight='
    bold')
```

The total damage bar plot and the occupation of total damage of differen weapon types are shown as Fig. 6-3. It can be seen that the damage of the long gun to the health value (HP) or the armor (Arm) is the highest, followed by the pistol, followed by the sniper rifle, submachine gun, props, heavy weapons and equipment.

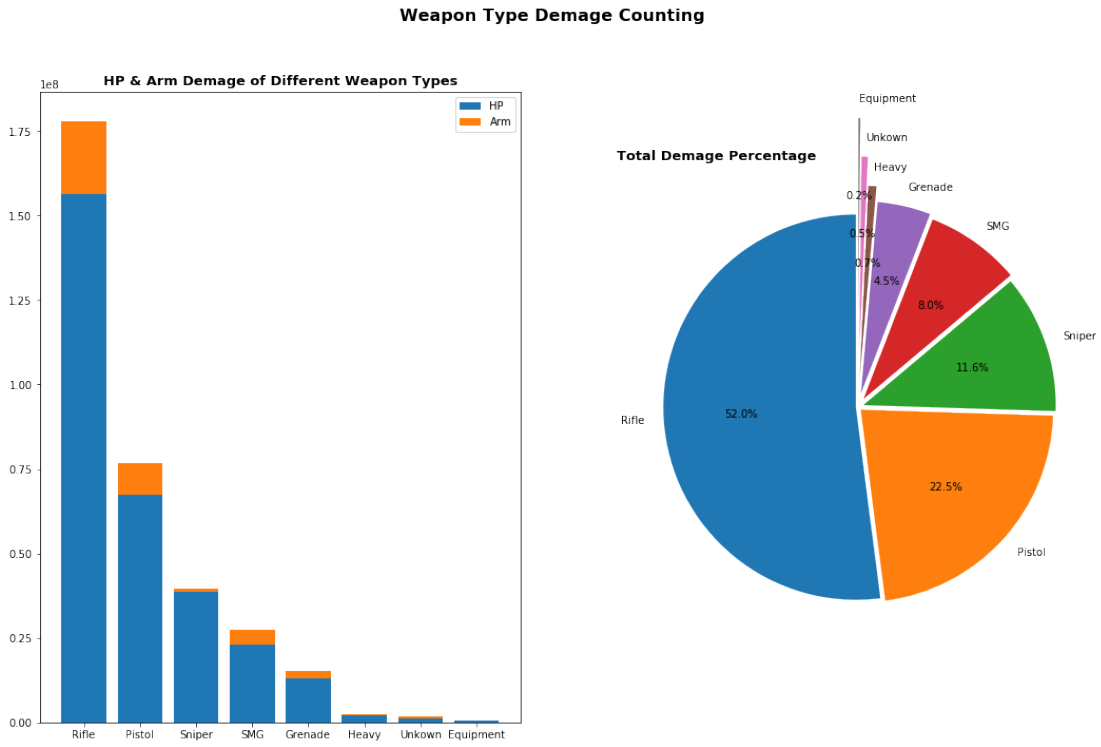


Fig. 6-3 Damages occupations of different weapon types

And display HP damages and Armor damages separately.

```
fig = plt.figure(figsize=(15, 8))
plt.subplot(1,2,1)
plt.title('HP Demage', fontsize=13, fontweight='bold',fontstyle='
    italic')
sns.barplot(x=dmg_wp_type_sum_pd['wp_type'],y=dmg_wp_type_sum_pd['
    hp_dmg'])
plt.subplot(1,2,2)
```

```
29 plt.title('Arm Demage', fontsize=13, fontweight='bold',fontstyle='
       italic')
30 sns.barplot(x=dmg_wp_type_sum_pd['wp_type'],y=dmg_wp_type_sum_pd['
       arm_dmg'])
31 plt.suptitle('Weapon Type Demage On HP & Arm', fontsize=16,
       fontweight='bold')
```

From ?? we can see:

The most damage to armor is the Rifles, followed by pistols, submachine guns, props, sniper rifles, heavy weapons and equipment.

It is worth noting that the damage of the submachine gun to the armor is higher than the damage to the HP. It may be that the players who use the submachine gun are more accustomed to charging, causing non-lethal damage to multiple enemies.

Sniper rifles do more damage to HP, but lower damage to armor, which reflects the characteristics of one-shot kills.

6.1.2 Weapon Damage Rank

There are 40 weapons according to Fig. 6-4

	wp_type	wp	count
0	Equipment	Knife	8673
1	Equipment	Zeus	1738
2	Equipment	Bomb	1
3	Grenade	Smoke	10237
4	Grenade	Flash	5664
5	Grenade	Decoy	8
6	Grenade	HE	469987
7	Grenade	Incendiary	782888
8	Grenade	Molotov	11732
9	Heavy	Nova	17900
10	Heavy	XM1014	40529
11	Heavy	M249	654
12	Heavy	SawedOff	4126
13	Heavy	Swag7	38449

(a)

14	Heavy	Negev	8543
15	Pistol	FiveSeven	80439
16	Pistol	Deagle	426307
17	Pistol	USP	462254
18	Pistol	DualBarettas	9659
19	Pistol	Tec9	9786
20	Pistol	CZ	340874
21	Pistol	P2000	56231
22	Pistol	Glock	435716
23	Pistol	P250	249322
24	Rifle	Famas	249257
25	Rifle	M4A4	1463705
26	Rifle	AK47	3115867
27	Rifle	AUG	24122

(b)

28	Rifle	Galil	98004
29	Rifle	M4A1	368527
30	Rifle	SG556	18890
31	SMG	MP7	427649
32	SMG	MP9	197649
33	SMG	P90	71122
34	SMG	UMP	208544
35	SMG	Bizon	10480
36	SMG	Mac10	218457
37	Sniper	G3SG1	3189
38	Sniper	Scout	64328
39	Sniper	Scar20	7870
40	Sniper	AWP	415525
41	Unkown	Unknown	103280

(c)

Fig. 6-4 Records of different weapons

Group by `wp` and summarize the total damage, we can count the total damages of different weapons.

Listing 6-2 Damages Rank by Weapons

```
dmg_wp_sum=dmg_wp_list.groupBy('wp').agg({'hp_dmg': 'sum', 'arm_dmg':
    'sum','tot_dmg': 'sum'}).\
  withColumnRenamed("sum(tot_dmg)", "tot_dmg").\
  withColumnRenamed("sum(hp_dmg)", "hp_dmg").\
  withColumnRenamed("sum(arm_dmg)", "arm_dmg").\
  orderBy('tot_dmg',ascending=0)
dmg_wp_sum_pd=dmg_wp_sum.toPandas()
```

The sum result are partly shown as Fig. 6-5

	wp	tot_dmg	hp_dmg	arm_dmg
0	AK47	109725947.0	98743694.0	10982253.0
1	M4A4	45141653.0	38014304.0	7127349.0
2	AWP	34445328.0	34013206.0	432122.0
3	Deagle	20967316.0	20171835.0	795481.0
4	USP	17441255.0	14527886.0	2913369.0
5	Glock	13562451.0	11582232.0	1980219.0

Fig. 6-5 HP and Armor damages of different weapons

Then draw graphs to illustrate the ranking of HP and ARM damages by different weapons.

```
#画图显示不同武器造成的伤害
fig = plt.figure(figsize=(16, 18))
sns.barplot(y=dmg_wp_sum_pd['wp'],\
            x=dmg_wp_sum_pd['tot_dmg'],\
            )

plt.suptitle('Demages from Different Weapon', fontsize=20, fontweight
    ='bold')

#画图显示不同武器造成的伤害
fig = plt.figure(figsize=(12, 12))

plt.pie(dmg_wp_sum_pd['tot_dmg'],labels=dmg_wp_sum_pd['wp'],\
        startangle=90,autopct='%1.1f%%')

plt.suptitle('Demages from Different Weapon', fontsize=16, fontweight
    ='bold')
```

The result shown as Fig. 6-6 and Fig. 6-7.

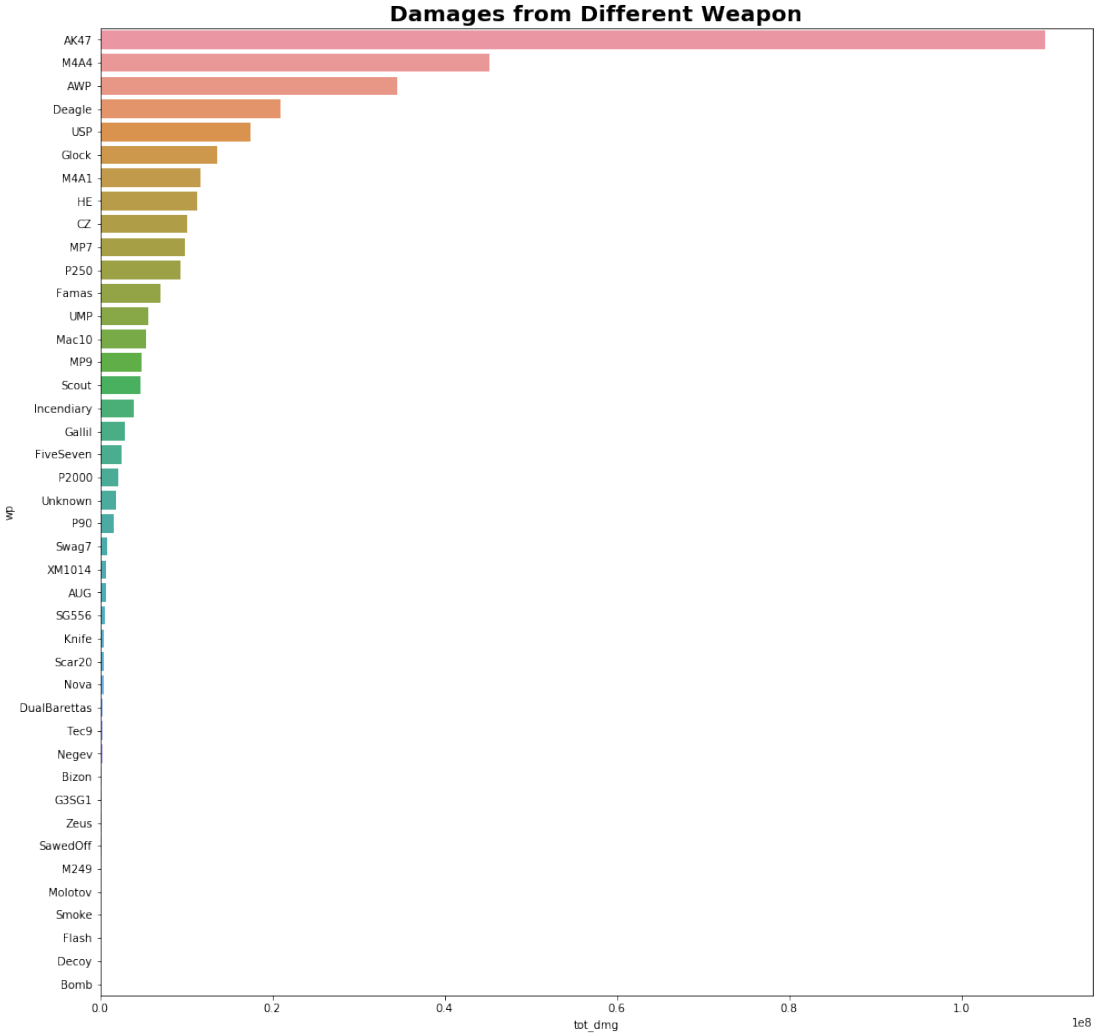


Fig. 6-6 Damages of different weapons

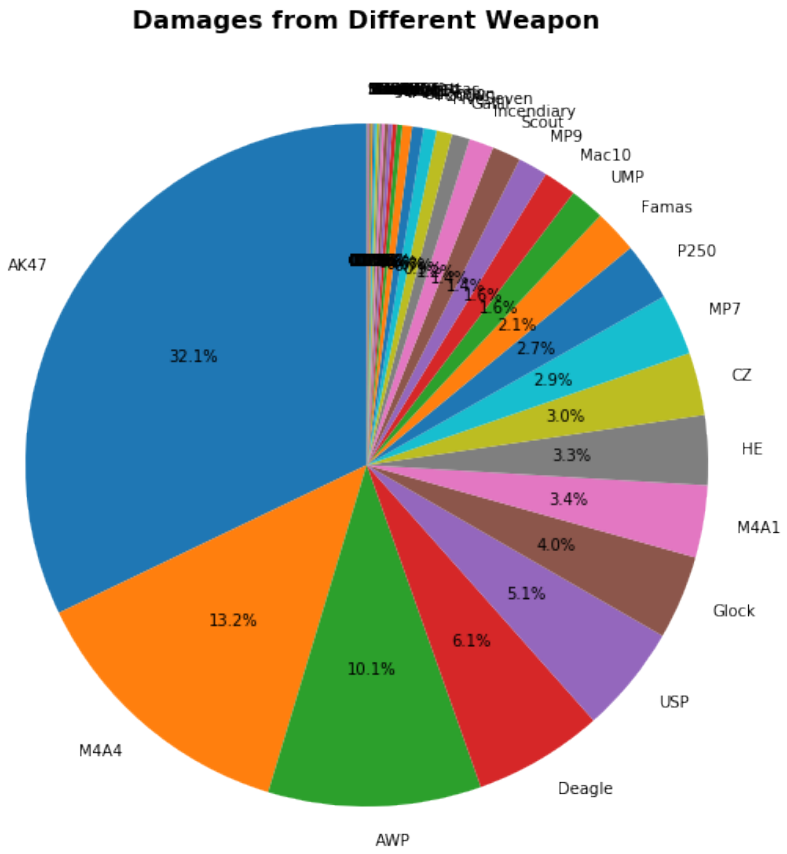


Fig. 6-7 Damage occupation of different weapons

From Fig. 6-7 we can see: The long guns, AK-47, M4A4, and AWP, Desert Eagle stand for the most of damages, noticing this is the dataset in late 2018, so M4A4 still take a big part than M4A1.

Here we show the top 10 and last 10 weapons particularly.

```
#画图显示前10名和后10名
fig = plt.figure(figsize=(13.5, 13))
plt.subplot(2,1,1)
plt.title('Top 10 Weapon Dameges', fontsize=14, fontweight='bold')
sns.barplot(y=dmg_wp_sum_pd.head(10)['wp'],\
            x=dmg_wp_sum_pd.head(10)['tot_dmg'],\
            palette=sns.color_palette("hls", 10)\
            )
plt.subplot(2,1,2)
plt.title('Last 10 Weapon Dameges', fontsize=14, fontweight='bold')
sns.barplot(y=dmg_wp_sum_pd.tail(10)['wp'],\
            x=dmg_wp_sum_pd.tail(10)['tot_dmg'],\
            palette=sns.color_palette("hls", 15)\
            )
```

```
36 plt.suptitle('Zoom of Weapon Dameges', fontsize=16, fontweight='bold'
       )
```

The results are shown as Fig. 6-8, the most popular weapon is `AK-47`.

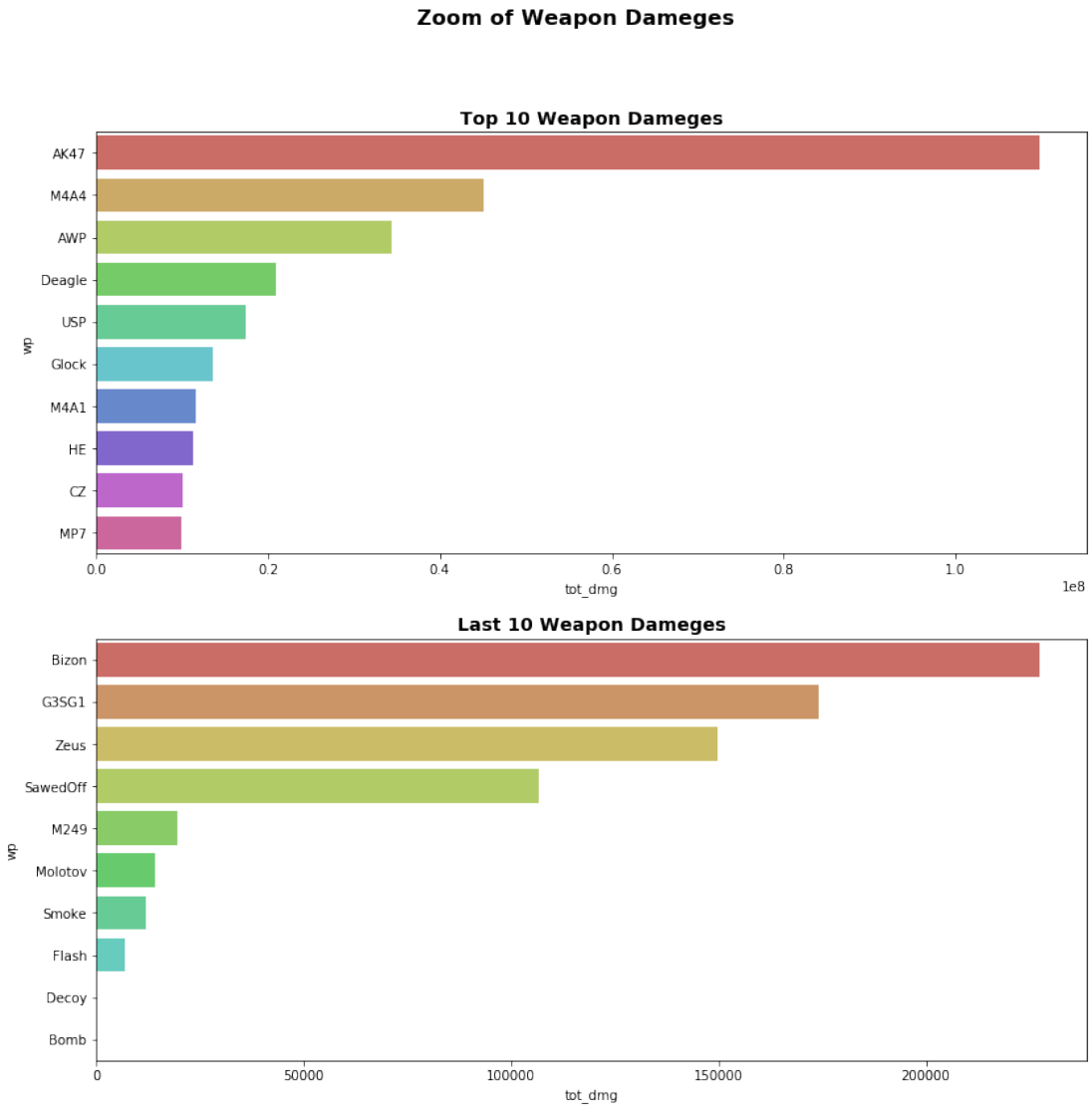


Fig. 6-8 Damage occupation of different weapons

6.2 Kill counting of Weapons

Count the killing records of different weapons, and draw the bar plot.

Listing 6-3 Weapon Killing Counts

```
kill_count=kill_df.select('wp','wp_type').groupBy('wp_type','wp').
    count().orderBy('count',ascending=0).toPandas()

fig = plt.figure(figsize=(16, 18))
#plt.subplot(1,2,1)
colors1=sns.color_palette("hls",40)
sns.barplot(y=kill_count['wp'],x=kill_count['count'],palette=colors1)
    \
            # .xaxis.set_major_formatter(ticker.PercentFormatter(xmax
    =1, decimals=1))
plt.suptitle('Kill Counting of Weapons', fontsize=16, fontweight='
    bold')
```

The result as the Fig. 6-9 shows.

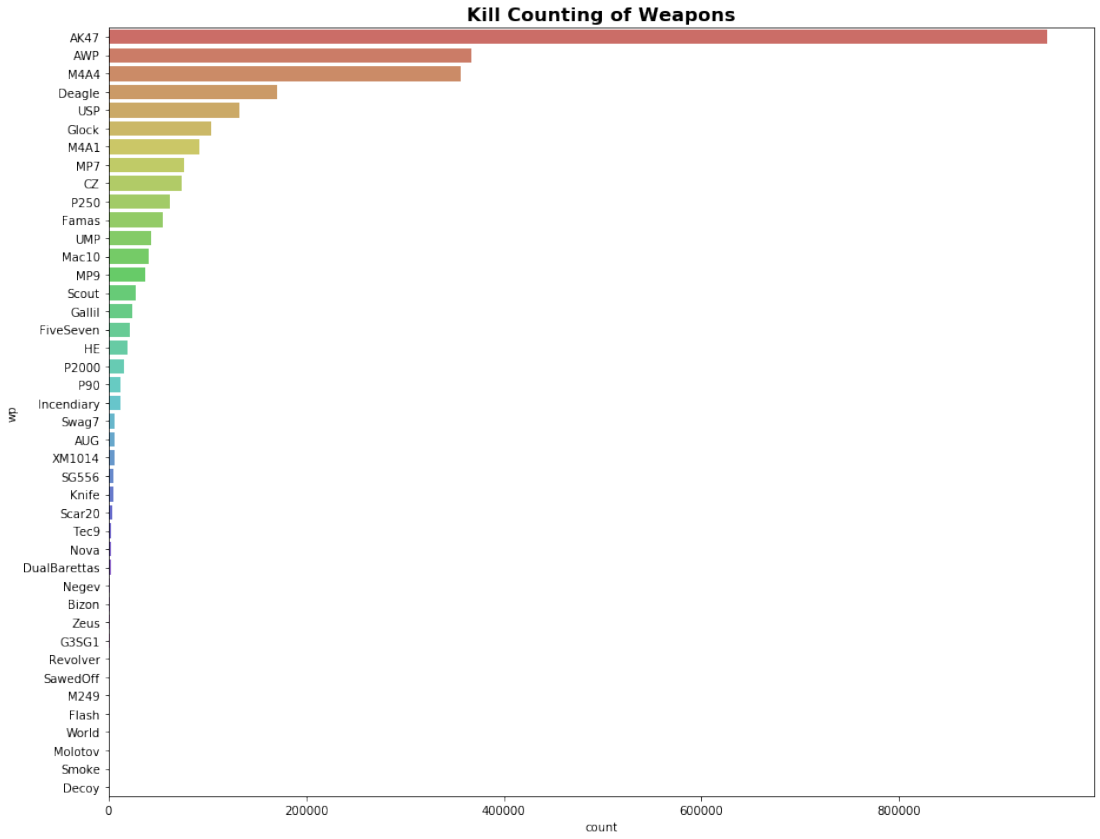


Fig. 6-9 Killing counting of different weapons

6.3 Preferences

6.3.1 Most kills of Weapon when 1 CT alive

Filter out the data when there is 1 CT left and the attacking side is CT.

Listing 6-4 Weapon Killing When 1 CT alive

```
kill_ct_1_pd=kill_df.select('wp','wp_type','att_side','ct_alive','
    t_alive','is_bomb_planted').\
  filter("ct_alive='1' and att_side='CounterTerrorist'").groupBy('wp'
    ).\
  count().orderBy('count',ascending=0).toPandas()
```

The Fig. 6-10 drawing shows the ranking of the guns that resulted in kills by CT when there is 1 person left in the CT.

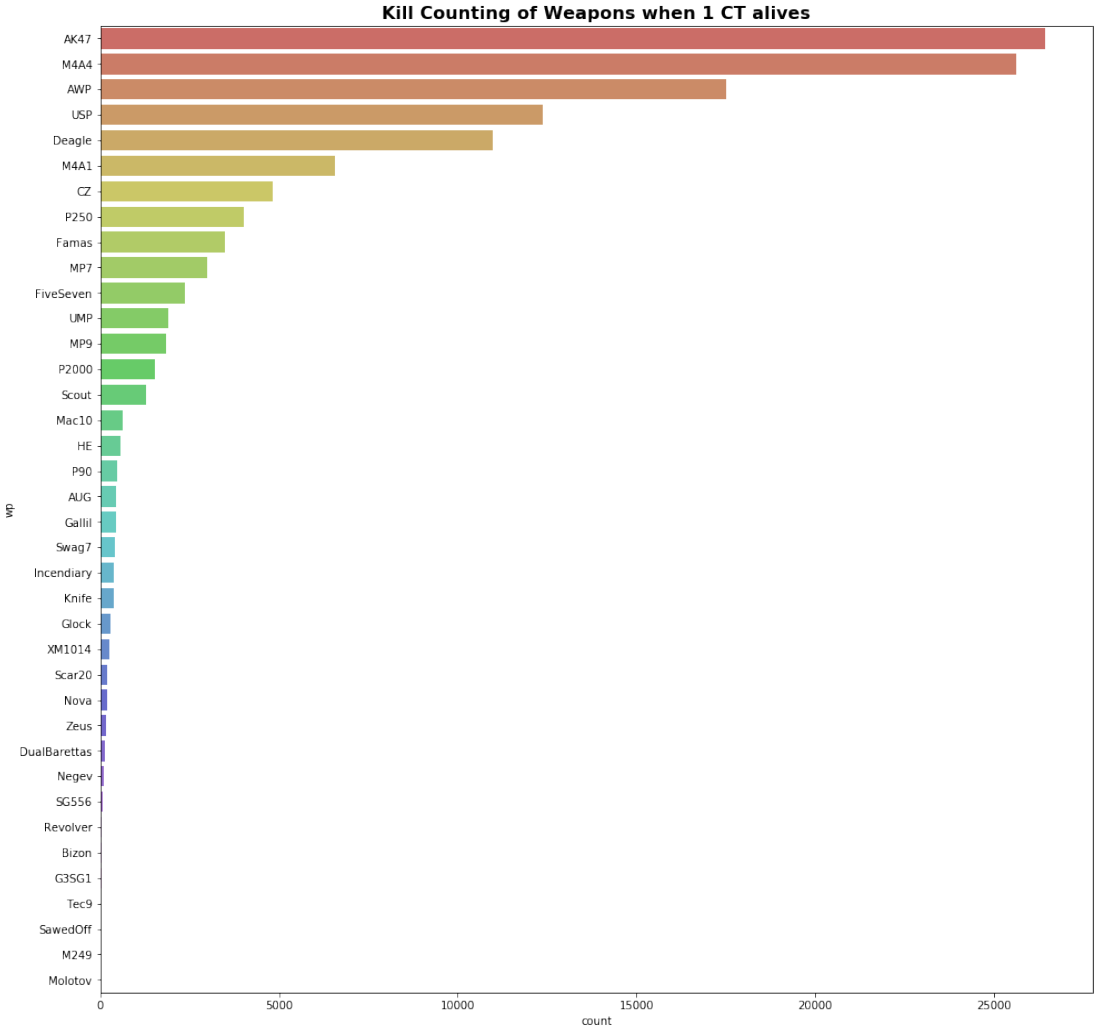


Fig. 6-10 Killing counting of CT when 1 alive

It is clearly to see that CTs are most likely to give a kill by AK-47 picked up from T side.

6.3.2 ECO round kill weapons

Join kill and meta table, as the most original data. Add `round_type` to determine the party with the least economy, and filter out the kills of the party with the least economy in the ECO grounds.

We collect T & CT side separately, filtered with ECO rounds, and count the killings.

Listing 6-5 Weapon Killing of ECO rounds

```
meta_part=meta_df.select('file','round','round_type','ct_eq_val','t_eq_val').withColumnRenamed("file", "file1").withColumnRenamed("round", "round1")
kill_count=kill_df.join(meta_part,on=[(kill_df.file==meta_part.file1) & (kill_df.round==meta_part.round1)],how="left").\
        filter("round_type='ECO'")
kill_count_ct=kill_count.filter(kill_count['ct_eq_val']< kill_count['t_eq_val']).filter("att_side='CounterTerrorist'").\
    groupBy('wp_type','wp').count().orderBy('count',ascending=0)
kill_count_t=kill_count.filter(kill_count['ct_eq_val']>= kill_count['t_eq_val']).filter("att_side='Terrorist'").\
    groupBy('wp_type','wp').count().orderBy('count',ascending=0)

kill_count_ct=kill_count_ct.toPandas()
kill_count_t=kill_count_t.toPandas()
```

And draw the pictures, shown as Fig. 6-11, the most popular weapon of ECO rounds is the Desert Eagle.

```
fig = plt.figure(figsize=(16, 25))
plt.subplot(2,1,1)
colors1=sns.color_palette("hls",40)
sns.barplot(y=kill_count_ct['wp'],x=kill_count_ct['count'],palette=colors1)
plt.title('Killings of ECO rounds on CT side', fontsize=16, fontweight='bold')

plt.subplot(2,1,2)
colors1=sns.color_palette("hls",40)
sns.barplot(y=kill_count_t['wp'],x=kill_count_t['count'],palette=colors1)
```

```
20 plt.title('Killings of ECO rounds on T side', fontsize=16, fontweight
       ='bold')
```

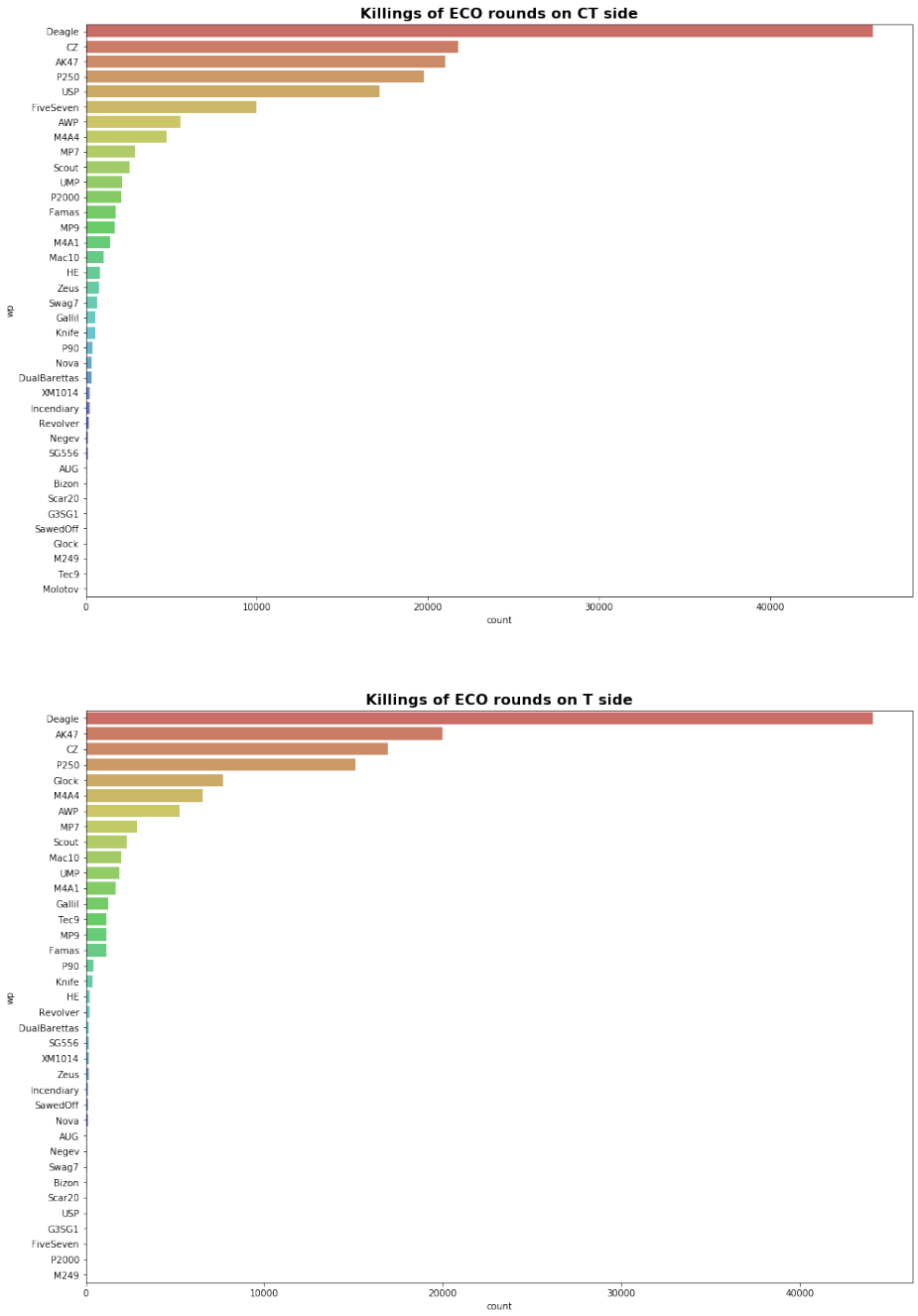


Fig. 6-11 Killing counting of CT when 1 alive

6.4 Head Shooting Rate Analysis

In the first we want to find how many hitboxes there are, and sum the damages and the records of different hitbox. Finally we can calculate the percentage of heat shots of different weapons.

6.4.1 Hitbox Damage

we use `dmg_hitbox_pd` and `dmg_hitbox_sum_pd` for records and total damages storing. The result tables are shown as fig:hitbox-raw.

Listing 6-6 Hitbox Analysis

```
dmg_hitbox_pd=dmg_wp_list.groupBy('hitbox').count().\
  orderBy('count',ascending=0 ).toPandas()

dmg_hitbox_sum=dmg_wp_list.groupBy('hitbox').agg({'hp_dmg': 'sum', '
    arm_dmg': 'sum','tot_dmg': 'sum'}).\
  withColumnRenamed("sum(tot_dmg)", "tot_dmg").\
  withColumnRenamed("sum(hp_dmg)", "hp_dmg").\
  withColumnRenamed("sum(arm_dmg)", "arm_dmg").\
  orderBy('tot_dmg',ascending=0)
dmg_hitbox_sum_pd=dmg_hitbox_sum.toPandas()
```

	hitbox	count
0	Chest	4225631
1	Stomach	1811283
2	Head	1478655
3	Generic	1394708
4	RightArm	715480
5	RightLeg	333583
6	LeftLeg	298783
7	LeftArm	258125
8	8	21934

(a) hitbox records

	hitbox	tot_dmg	hp_dmg	arm_dmg
0	Chest	115286584.0	102746528.0	12540056.0
1	Head	110507583.0	96296187.0	14211396.0
2	Stomach	57601555.0	50948744.0	6652811.0
3	RightArm	19570309.0	17501191.0	2069118.0
4	Generic	17688686.0	14860317.0	2828369.0
5	RightLeg	7112794.0	7112794.0	0.0
6	LeftArm	6935885.0	6165103.0	770782.0
7	LeftLeg	6319436.0	6319436.0	0.0
8	8	566094.0	496159.0	69935.0

(b) hitbox damages

Fig. 6-12 Records and damages counting of different hitbox

And we draw bar plot and pie chart of hitboxes.

First is the total damages of different hitbox, shown as Fig. 6-13

```
fig = plt.figure(figsize=(17, 9))
plt.subplot(1,2,1)
colors1=sns.color_palette("hls", 9)
sns.barplot(y=dmg_hitbox_sum_pd['hitbox'],x=dmg_hitbox_sum_pd['tot_dmg'],palette=colors1)
#.xaxis.set_major_formatter(ticker.PercentFormatter(xmax=1, decimals=1))

plt.subplot(1,2,2)
colors=sns.color_palette("hls", 9)
plt.pie(dmg_hitbox_sum_pd['tot_dmg'],labels=dmg_hitbox_sum_pd['hitbox'],\
        colors=colors1,startangle=180,\
        autopct='%1.1f%%',explode =(0.01,0.02,0.03,0.04,0.05,0.06,0.08,0.12,0.25))
plt.suptitle('Demage on Different Hitboxs', fontsize=16, fontweight='bold')
```

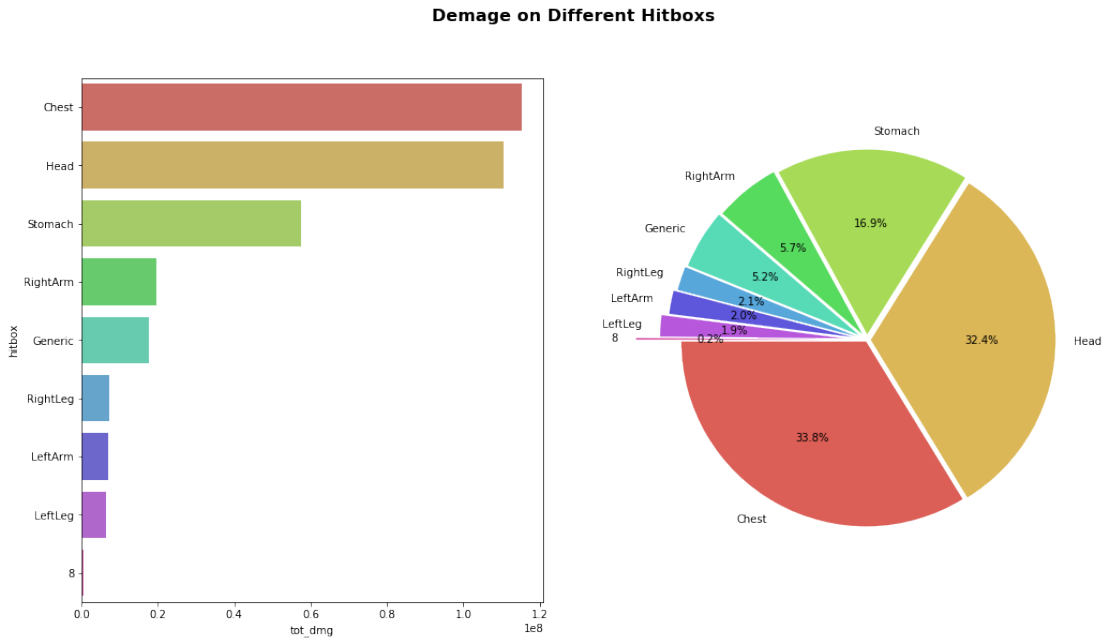


Fig. 6-13 Total damages of different hitbox

6.4.2 Hitbox Records

Then analyze the records of hitbox, the result shown as Fig. 6-14.

```
fig = plt.figure(figsize=(17, 9))
plt.subplot(1,2,1)
```

```
colors1=sns.color_palette("Paired",10)
sns.barplot(y=dmg_hitbox_pd['hitbox'],x=dmg_hitbox_pd['count'],
    palette=colors1)
#.xaxis.set_major_formatter(ticker.PercentFormatter(xmax=1, decimals
    =1))

plt.subplot(1,2,2)
colors=sns.color_palette("hls", 9)
plt.pie(dmg_hitbox_pd['count'],labels=dmg_hitbox_pd['hitbox'],\
        colors=colors1,startangle=180,\
        autopct='%1.1f%%',explode =
    (0.01,0.02,0.03,0.04,0.05,0.06,0.08,0.12,0.25))
plt.suptitle('Records of Different Hitboxs', fontsize=16, fontweight=
    'bold')
```

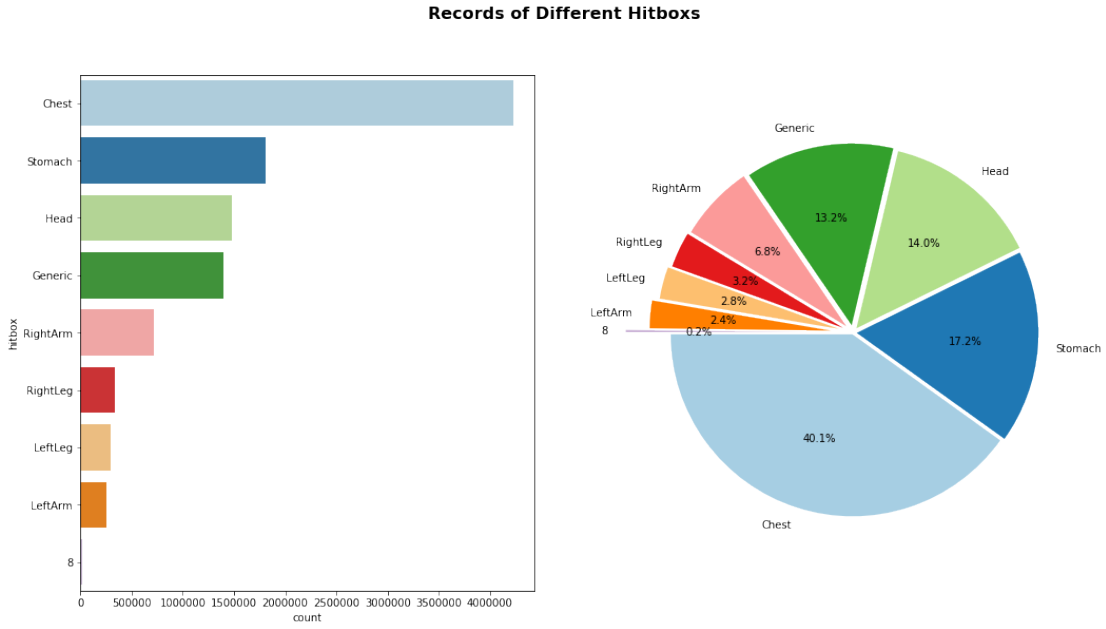


Fig. 6-14 Records of different hitbox

It can be seen that the stomach has more damage entries than the head, but the total damage amount of the head is higher than that of the stomach.

Kicks have no armor.

From Fig. 6-14, It can be seen that the comprehensive headshot rate of all weapons (which can also be understood as the comprehensive headshot rate of all players in the game) is only 16%. More damage in the game is still generated by the long gun hitting the chest and abdomen.

6.4.3 Head Shooting Rate of Weapons

Derivate the head shooting rate by using eq. (6-1), by collect the head hit records and all hit records, which shown as Listing 6-7.

$$\text{Head Shot Rate} = \frac{\text{Count of Head}}{\text{Count of ALL Hitbox}} \tag{6-1}$$

Listing 6-7 Head shooting Rate Analysis

```
#全部伤害条目
dmg_all_hit=dmg_wp_list.groupBy('wp').count().orderBy('count',
    ascending=0 ).toPandas().head(40)
#击中头部的条目
dmg_head_hit=dmg_wp_list.filter("hitbox='Head'").groupBy('wp').count
    ().withColumnRenamed("count", "count_h").\
       orderBy('count_h',ascending=0 ).toPandas()

#链接表
#在Pandas中join
dmg_head_rate=dmg_head_hit.join(dmg_all_hit.set_index('wp'), on='wp')
#计算爆头率
dmg_head_rate['head_rate']=dmg_head_rate['count_h']/dmg_head_rate['
    count']
dmg_head_rate=dmg_head_rate.sort_values('head_rate',ascending=0 )
```

The result are shown as Table 6-1.

And we draw the plot as Fig. 6-15 shows. USP is the highest heat shooting rate weapon, and P200, Glock, Desert Eagle take the rest places.

Rifle guns has less head shooting rate than pistols. And AK-47 has head shooting rate in the middle.

```
#from matplotlib import ticker
fig = plt.figure(figsize=(17, 13))
#plt.subplot(1,2,1)
colors1=sns.color_palette("hls",40)
sns.barplot(y=dmg_head_rate['wp'],x=dmg_head_rate['head_rate'],
    palette=colors1)\
             .xaxis.set_major_formatter(ticker.PercentFormatter(xmax
    =1, decimals=1))
plt.suptitle('Head Shot Rate of Weapons', fontsize=16, fontweight='
    bold')
```

Table 6-1 Example table of the bomb site counting to Pandas

	wp	count_h	count	head_rate
2	USP	130791	462254	0.282942
15	P2000	15398	56231	0.273835
3	Glock	116769	435716	0.267993
4	Deagle	100688	426307	0.236187
16	Scout	14469	64328	0.224925
6	P250	52273	249322	0.209661
24	DualBarettas	1976	9659	0.204576
14	FiveSeven	15793	80439	0.196335
25	Tec9	1868	9786	0.190885
29	SawedOff	713	4126	0.172807
19	Swag7	6556	38449	0.170512
11	MP9	31833	197649	0.161058
5	CZ	54508	340874	0.159907
21	Nova	2828	17900	0.157989
0	AK47	473126	3115867	0.151844
10	Mac10	32604	218457	0.149247
23	SG556	2785	18890	0.147433
12	UMP	30610	208544	0.146780
1	M4A4	194640	1463705	0.132978
31	M249	86	654	0.131498
8	M4A1	48242	368527	0.130905
17	Gallil	12705	98004	0.129638
20	XM1014	5060	40529	0.124849
7	MP7	51855	427649	0.121256
13	Famas	29552	249257	0.118560
26	Bizon	1224	10480	0.116794
22	AUG	2813	24122	0.116616
30	G3SG1	358	3189	0.112261
18	P90	7791	71122	0.109544
27	Negev	809	8543	0.094697
28	Scar20	732	7870	0.093011
9	AWP	37200	415525	0.089525

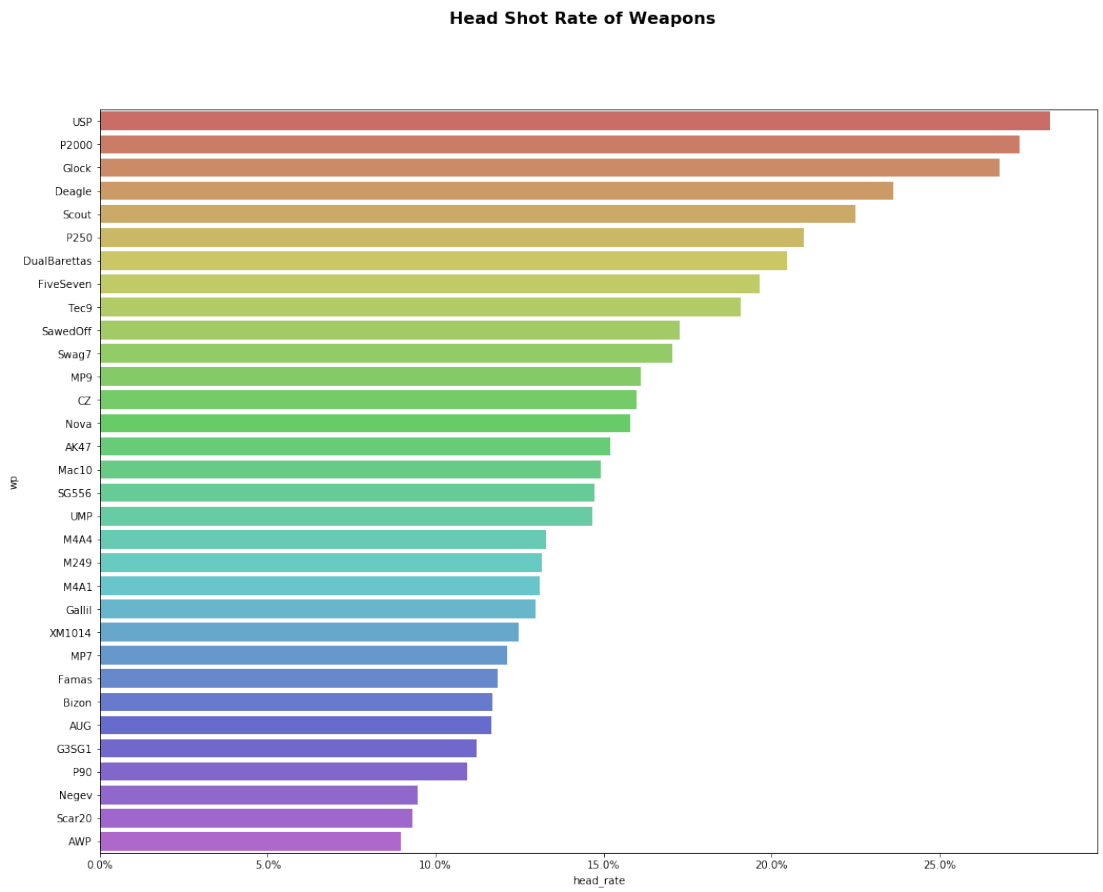


Fig. 6-15 Head Shooting Rate of Weapons

6.5 Power of Weapons

We calculate Attack Damage Ratio, which is all_dmg/all_count, represent the power of weapons.

Listing 6-8 Weapon ADR Calculation

```
#链接表
wp_dmg_rate=dmg_wp_sum_pd.join(dmg_count.set_index('wp'), on='wp')
#计算单发伤害
wp_dmg_rate['dmg_rate']=wp_dmg_rate['tot_dmg']/wp_dmg_rate['count']
wp_dmg_rate=wp_dmg_rate.sort_values('dmg_rate',ascending=0 )

fig = plt.figure(figsize=(17, 13))
#plt.subplot(1,2,1)
colors1=sns.color_palette("hls",40)
sns.barplot(y=wp_dmg_rate['wp'],x=wp_dmg_rate['dmg_rate'],palette=
    colors1)\
```

```
            # .xaxis.set_major_formatter(ticker.PercentFormatter(xmax
    =1, decimals=1))
plt.suptitle('Average Demage per Shot of Weapons', fontsize=16,
    fontweight='bold')
```

The result picture are shown as Fig. 6-16.

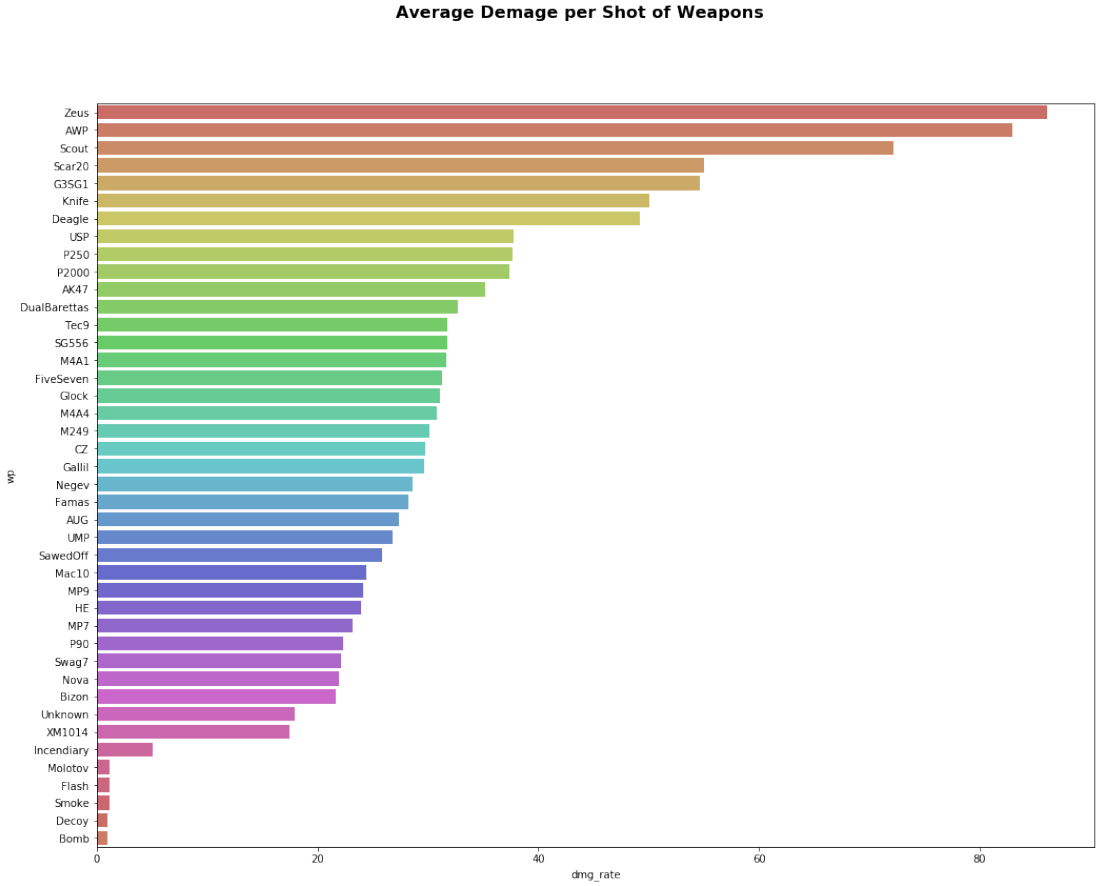


Fig. 6-16 Average Damage per shot of Weapons

Chapter VII

Economic Impact Analysis

7.1 Round Types Counts

We first collect fields relating to economics, and count how many round types there are.

Listing 7-1 Round Type Counting

```
economics_RAW=meta_df.select('file', 'map', 'round','round_type','
    ct_eq_val','t_eq_val')
round_type_pd=economics_RAW.groupBy('round_type').count().toPandas()
round_type_pd
```

	round_type	count
0	ECO	128613
1	PISTOL_ROUND	28114
2	SEMI_ECO	24444
3	FORCE_BUY	40876
4	NORMAL	155582

Fig. 7-1 Round Types Counting

Then plot bar chart and pie chart so show the occupation of these type rounds, shown as Fig. 7-2

```
import seaborn as sns
fig = plt.figure(figsize=(14, 6))
#画条形图
plt.subplot(1,2,1)
sns.barplot(x=round_type_pd['round_type'],y=round_type_pd['count'])
#画饼图
```

```
plt.subplot(1,2,2)
plt.pie(round_type_pd['count'],labels=round_type_pd['round_type'],
    autopct='%1.1f%%',explode = (0.02,0.03,0.04,0.05,0.06))
plt.suptitle('Round Types Statistics', fontsize=16, fontweight='bold'
    )
```

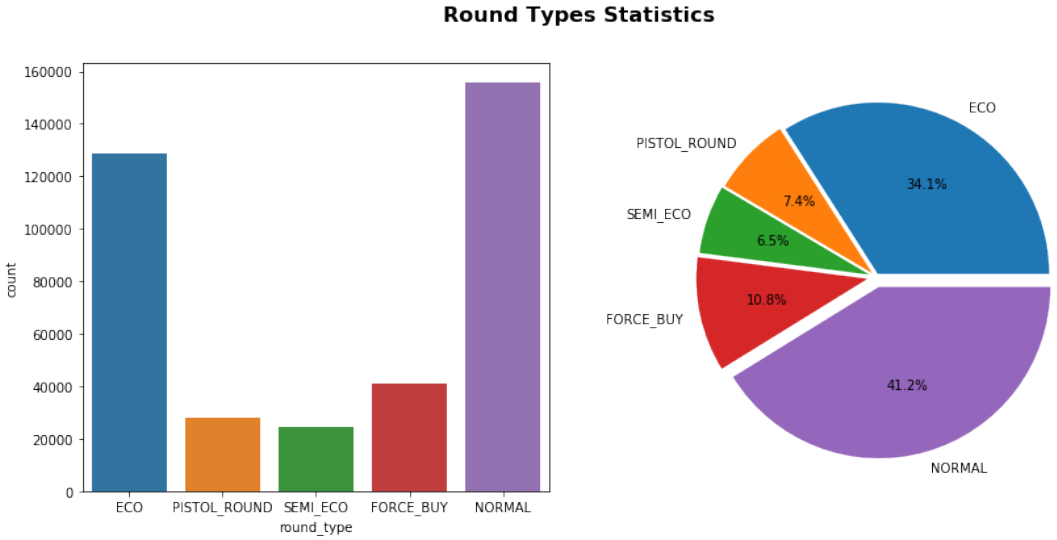


Fig. 7-2 Round Types Counting

Most rounds are normal rounds, and over a quarter rounds are ECO rounds. pistol rounds, semi pistol rounds and force buy rounds share the similar parts around 10 percent. We are about to see the connection between economics and round types.

7.2 Density Curve

Let compare the total economics between T side & CT side.

Collect equipment value field of each rounds, directly sending then to Pandas, and draw the density distributing curve and the accumulating distributing curve.

Listing 7-2 Economic Distribution Curves

```
eeco_pd=economics_RAW.select('round_type','ct_eq_val','t_eq_val').\
        orderBy('t_eq_val',ascending=0 ).toPandas()

fig = plt.figure(figsize=(14, 17))

#画概率密度函数
plt.subplot(2,1,1)
```

```
plt.title('Economics Distribution Density', fontsize=14, fontweight='bold')
sns.kdeplot(eco_pd['ct_eq_val'].rename('Counter-Terrorists'))
sns.kdeplot(eco_pd['t_eq_val'].rename('Terrorists'))


#画概率分布函数
plt.subplot(2,1,2)
plt.title('Economics Accumulating Distribution', fontsize=14, fontweight='bold')
sns.kdeplot(eco_pd['ct_eq_val'].rename('Counter-Terrorists'), cumulative=True,shade=True,color='blue')
sns.kdeplot(eco_pd['t_eq_val'].rename('Terrorists'),cumulative=True, shade=True,color='red')

plt.suptitle('Economics Distribution Estimation', fontsize=18, fontweight='bold')
```

The Fig. 7-3 shows the estimation curve of the economic probability density distribution of T and CT. From this figure, it can be seen that the highest economy of CT is higher than that of T.

Both T and CT tend to buy "few equipment" or "a lot of equipment",

In the low economic stage (ECO), the most probable economic values of T and CT economies are basically the same;

In the high economy stage (over $20000), the most likely economy value of CT is higher than the economy of T, which may also be that the equipment price of CT is higher than that of T

This law can also be seen from the economic cumulative distribution estimation curve in the figure below.

As you can see in the Fig. 7-3, teams are very much inclined to buy "barely" or "a lot" of gear. Interestingly, Counter-Strikes tend to cost slightly more than Terrorists during purchase rounds, probably because they have a higher limit on the maximum amount of gear that can be purchased.

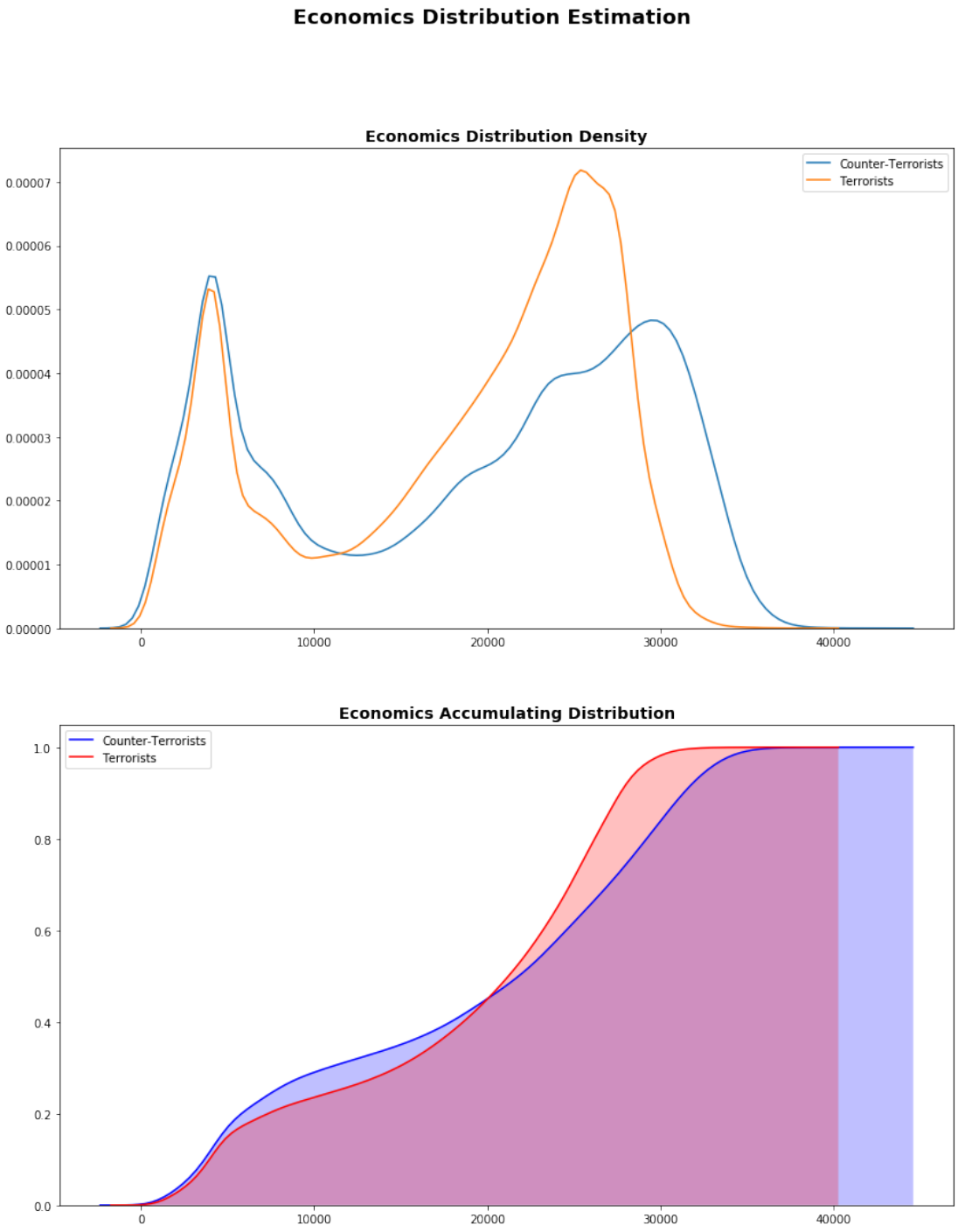


Fig. 7-3 Economic Distribution Curves

7.3 Number Statistics

Draw the box picture of T & CT economics:

Listing 7-3 Economic Statiscs

```
fig = plt.figure(figsize=(14, 8))

#画箱线图
bplot=plt.boxplot([eco_pd['ct_eq_val'],eco_pd['t_eq_val']],
                  patch_artist=True,
                  boxprops={'color': 'black'},
                  notch=True,
                  showmeans=True,
                  labels=['CT', 'T'])
colors = ['lightblue','orange']
for patch, color in zip(bplot['boxes'], colors):
    patch.set_facecolor(color)

plt.suptitle('Economics Statiscs', fontsize=18, fontweight='bold')
```

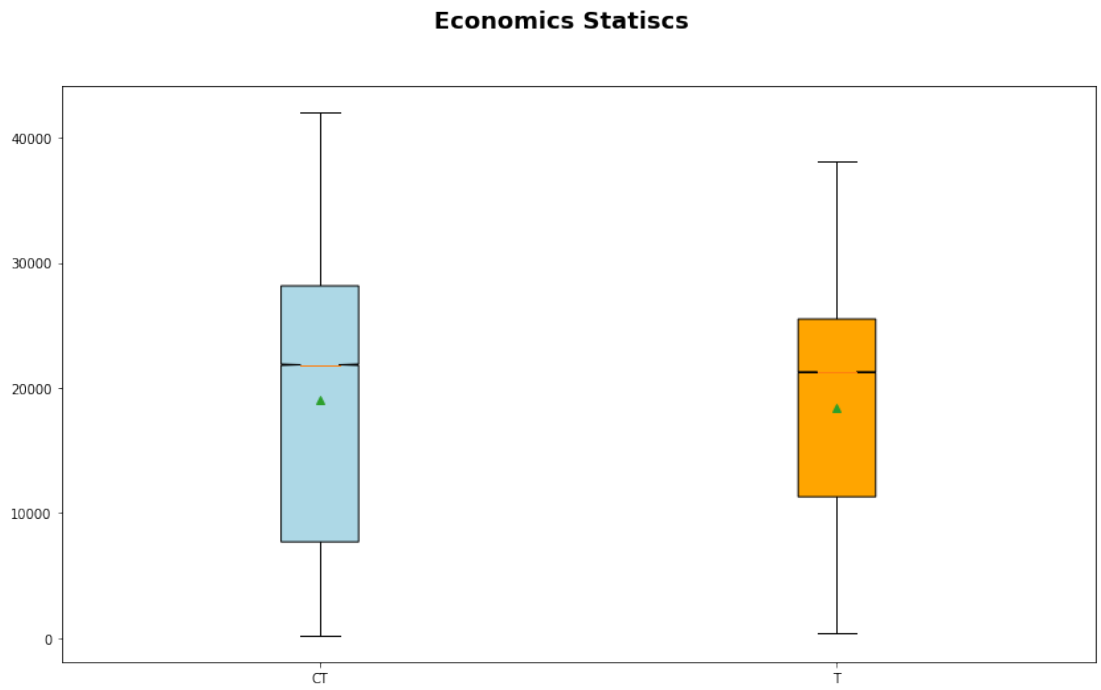


Fig. 7-4 Economic Box Plot

As the Fig. 7-4 shows, The economy of CT is higher than that of T, and it is also more volatile and easier to ECO, which is designed by the game mechanics.

The CT economy is slightly higher than the T economy, whether it is the

median or the average, which is related to the game mechanics. And it is clear that the CT economy is more volatile. Why do you think you can make a gun with a T? Why do police often ECO? The results show that CT is more difficult than T to control the economy.

7.4 Correlation between economics and winning rate

We draw the distribution density curve of the winning rounds on the difference of economics on T & CT side.

We use the `meta` table as our data source. First calculate the economics differences of both sides.

Listing 7-4 Correlation between economic and winrate

```
#以T减去CT的经济作为经济差
ecowin=meta_df.select('winner_side',(meta_df.t_eq_val-meta_df.
    ct_eq_val).alias("eco_diff"))
ecowin_t=ecowin.filter("winner_side='Terrorist'")
ecowin_ct=ecowin.filter("winner_side='CounterTerrorist'").withColumn(
    'eco_diff',-ecowin.eco_diff)

#Sending to Pandas
ecowin_t=ecowin_t.toPandas()
ecowin_ct=ecowin_ct.toPandas()
```

Now we have the records of rounds and eco differences. Then draw the density curves of the economics differences.

```
fig = plt.figure(figsize=(14, 8))

#画概率密度函数
plt.title('Distribution Density of Winning Rounds on Economics
    Difference ', fontsize=14, fontweight='bold')
sns.kdeplot(ecowin_t['eco_diff'].rename('T won'),shade=False,color='
    red')
sns.kdeplot(ecowin_ct['eco_diff'].rename('CT won'),shade=False,color=
    'darkblue')
```

The result are shown as Fig. 7-5.

Most winning rounds distributed at the economics on both sides are nearly the same, and winning rounds by economics advantages are simply more than disadvantages.

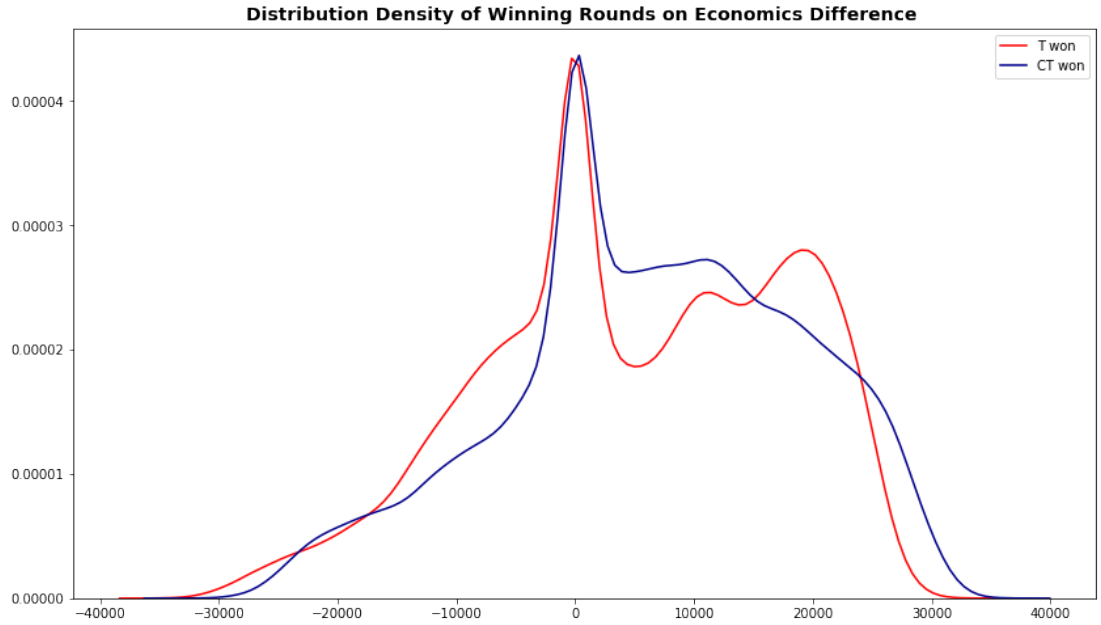


Fig. 7-5 Winning Rounds on Economics Difference

And when come economics disadvantages, the winning rounds of T are more than CT. When economics advantages, T usually need more positive economics difference to win the round.

To some extent, it reflects the positive correlation between economy and win rate on T & CT sides.

7.5 Correlation between economics and grenade using

We draw the scatter plot and the distribution density curve of the grenade using records on the difference of economics.

First count gre use records per round ,and add field `economics`, drop duplicated records of a same grenade to avoid redundant counting, count the grenade using count by economics, which means in the same round.

Listing 7-5 Correlation between economic and grenade using

```
meta_part=meta_df.select('file','round','ct_eq_val','t_eq_val').\
withColumnRenamed("file", "file1").withColumnRenamed("round", "round1
    ")
#将经济字段添加到gre中
gre_2=gre_df.join(meta_part,on=[(gre_df.file==meta_part.file1)& (
    gre_df.round==meta_part.round1)],how="left")
```

```
#对于同一个道具，只记录一条
gre_2=gre_2.select('file','round','att_id','nade','ct_eq_val','
    t_eq_val').dropDuplicates()
#对于CT和T，分别统计他们的道具数与经济
greco_ct=gre_2.filter("att_side='CounterTerrorist'").groupBy('file','
    round').count()
greco_ct=greco_ct.join(meta_part,on=[(greco_ct.file==meta_part.file1)
    & (greco_ct.round==meta_part.round1)],how="left").\
withColumnRenamed("ct_eq_val", "eco").select('file','round','count','
    eco')

greco_t=gre_2.filter("att_side='Terrorist'").groupBy('file','round').
    count()
greco_t=greco_t.join(meta_part,on=[(greco_t.file==meta_part.file1)& (
    greco_t.round==meta_part.round1)],how="left").\
withColumnRenamed("t_eq_val", "eco").select('file','round','count','
    eco')

greco=greco_t.union(greco_ct)
```

We draw the scatter plot of grenades using and total equipment value, the result is Fig. 7-7.

```
greco_pd=greco.toPandas()

import matplotlib.pyplot as plt
import seaborn as sns
fig = plt.figure(figsize=(32,16))
#colors1=sns.color_palette("hls",40)
plt.ylim(0,25)
sns.scatterplot(data=greco_pd,x='eco',y='count')
plt.title('Relative of Economics and Grenades Using', fontsize=20,
    fontweight='bold')
```

Then we collect grenade using records by T & CT side separately, then plot the density curve.

```
gre_den_ct=gre_2.filter("att_side='CounterTerrorist'").toPandas()
gre_den_t=gre_2.filter("att_side='Terrorist'").toPandas()

fig = plt.figure(figsize=(14, 8))

#画概率密度函数
plt.title('Distribution Density of Grenade Using on Economics',
    fontsize=14, fontweight='bold')
```

```
33 sns.kdeplot(gre_den_t['t_eq_val'].rename('T Grenade Using'),shade=
       False,color='red')
34 sns.kdeplot(gre_den_ct['ct_eq_val'].rename('CT Grenade Using'),shade=
       False,color='darkblue')
```

Here we show some record of T side, Fig. 7-6.

```
gre_den_t
```

	file	round	att_id	nade	ct_eq_val	t_eq_val
0	esea_match_13823203.dem	2	76561198070531027	Decoy	21150	1950
1	esea_match_13822924.dem	15	76561198059968024	Decoy	4250	3000
2	esea_match_13821656.dem	16	76561198198547942	Decoy	4450	3000
3	esea_match_13788924.dem	1	76561198843726342	Decoy	3450	3050
4	esea_match_13780894.dem	16	76561198120751708	Decoy	3550	3450
5	esea_match_13819673.dem	1	76561198148606816	Decoy	3000	3500
6	esea_match_13788050.dem	1	76561198826890445	Decoy	4150	3500

Fig. 7-6 Grenade using records with equipment value of T side

The density curve as the Fig. 7-8 shows.

From the scatterplot (Fig. 7-7) we can see the top edge of the data seem a growing curve, grenades using are more frequently when economics are sufficient.

And same as Fig. 7-8, here we are clearly to see more economics lead to more grenade using, and CT need more economics to get the peak than T does.

When economics are low of both T & CT side, the grenade using are nearly same.

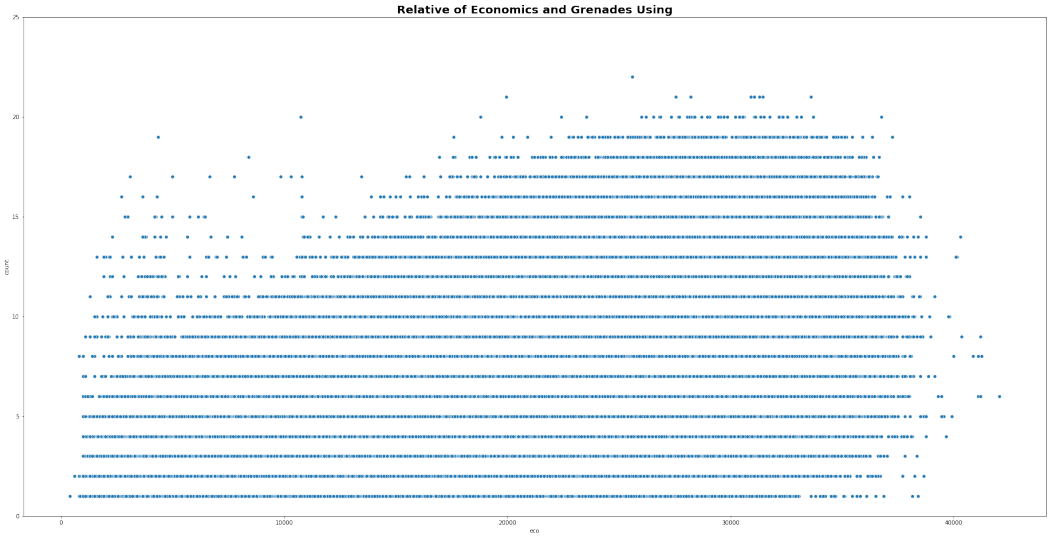


Fig. 7-7 Scatterplot of grenades using and total equipment value

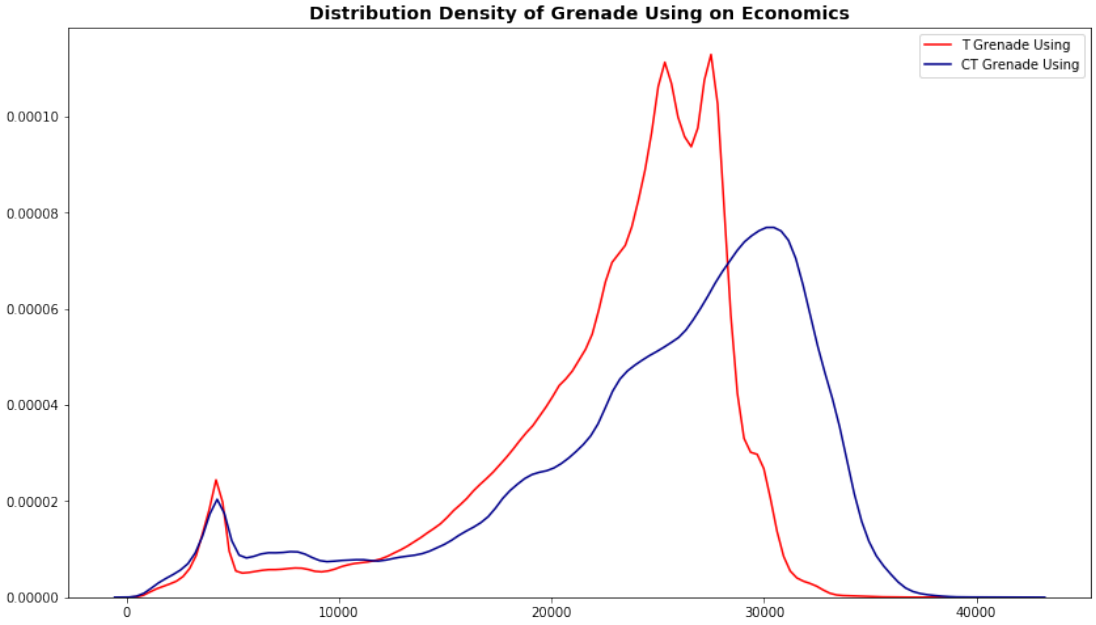


Fig. 7-8 Distributing density of grenade using records on equipment value

Chapter VIII

Grenade Analysis

8.1 Grenades Counting

Counting the throwing records of every kind of grenades. Group by `nade` and sort the data, then draw bar plots.

Listing 8-1 Grenade Using Count

```
gre_count_pd=gre_df.select('nade','hp_dmg','arm_dmg','nade_land_x','
    nade_land_y').groupBy('nade').count().\
orderBy('count',ascending=0 ).toPandas()

fig = plt.figure(figsize=(8, 6))
#plt.subplot(1,2,1)
colors1=sns.color_palette("hls",6)
sns.barplot(x=gre_count_pd['nade'],y=gre_count_pd['count'],palette=
    colors1)\
            # .xaxis.set_major_formatter(ticker.PercentFormatter(xmax
    =1, decimals=1))
plt.title('Grenades Counting', fontsize=16, fontweight='bold')
```

As the Fig. 8-1 shows, It can be seen that flash bombs are used the most, followed by smoke, HE grenades, CT incendiaries, T Molotov, and decoys.

The CT use incendiary is 30% more frequent than the T use Molotov, but the economy of the CT incendiaries is higher than that of the T Molotov. Afterwards, the correlation between the use of grenades and the total economic value of both sides will be analyzed.

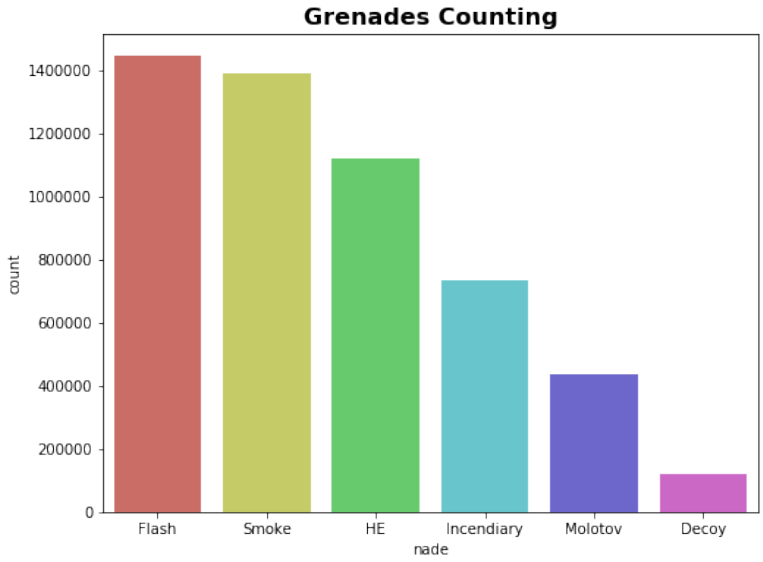


Fig. 8-1 Grenade Using Records Count

8.2 Victims of 1 HE grenade

This part is rather difficult, counting how man victims does ***one single High-Explosive grenade*** inflicted, by which can we deduce whether it is useful to use HE grenades in competitive game plays.

One single HE grenade typically means in the same play same round, at the same time, landed on the same position, throwing by the same player, etc.

We choose `'file','seconds','nade_land_x','nade_land_y'` four variables to determine a specific grenade.

First select HE grenade records from the main table, stored in `HE_list`, then drop the invalid records that the `secons` field is empty, as you can see in the Listing 8-2.

Listing 8-2 Counting victims per one HE grenade

```
from pyspark.sql.functions import *
import pyspark.sql.functions as F

gre_list=gre_df.select('file','seconds','nade','hp_dmg','arm_dmg','
    nade_land_x','nade_land_y',\
     'att_pos_x','att_pos_y','vic_id','vic_pos_x','vic_pos_y')
#筛选出HE手雷的数据
HE_list=gre_list.filter("nade='HE'").select('file','seconds','
    nade_land_x','nade_land_y','vic_id','vic_pos_x','vic_pos_y').\
     orderBy('nade_land_x','nade_land_y')
```

```

#过滤掉无效项
HE_list=HE_list.filter("seconds!=''").\
                orderBy('seconds')
HE_list.show(50)
```

It is necessary to replace the empty value `''` in the data with None, that is, `null` , so that the number of people who are not injured can be counted, that is, the number of people who have been injured is 0.

After data regularized, according to `file, seconds, nade_land_x` and `nade_land_y` , the number of different `vic_ids` is grouped and counted, and dropping duplicated records of the same `vic_ids`.And then count how many people a single grenade blew up.

```
#将vic_id为空值的, 替换为null
HE_null_replaced=HE_list.withColumn("vic_id",F.when(F.col("vic_id")==
    '',F.lit(None)).\
     otherwise(F.lit(F.col("vic_id"))))

#对于vic_id去重, 针对每一颗手雷分组查找
HE_count=HE_null_replaced.groupBy('file','seconds','nade_land_x','
    nade_land_y').agg(F.countDistinct("vic_id")).\
     withColumnRenamed("count(DISTINCT vic_id)", "num").\
     orderBy("num", ascending=0)
```

Counting the records of different victims does a HE grenade inflicted, and draw bar plot and pie pictures.

```
#统计炸伤人数的条目
HE_num_pd=HE_count.groupBy('num').count().orderBy('num').toPandas()

fig = plt.figure(figsize=(16, 6))
plt.subplot(1,2,1)
colors1=sns.color_palette("hls",6)
sns.barplot(y="num",x="count",data=HE_num_pd,orient="h",palette=
    colors1) #使用orient="h"参数控制列为主值
plt.subplot(1,2,2)
colors=sns.color_palette("hls", 6)
plt.pie(HE_num_pd['count'],labels=HE_num_pd['num'],\
     colors=colors1,\
     autopct='%1.1f%%',explode = (0.01,0.02,0.03,0.08,0.25,0.45))
plt.suptitle('Victims per HE', fontsize=16, fontweight='bold')
```

The result is shown by the Fig. 8-2

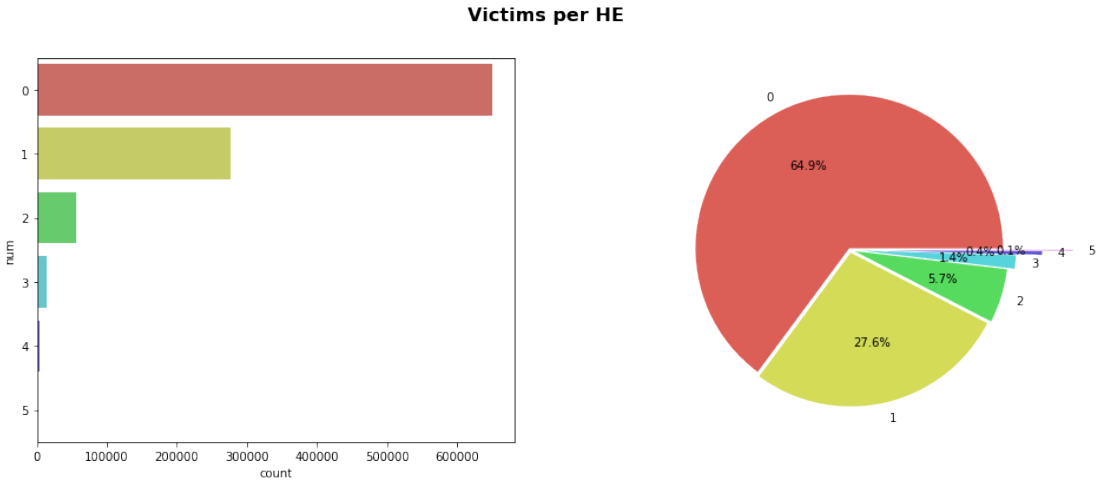


Fig. 8-2 Victims per one HE grenade

It can be seen that most of the grenades did not blow up, about 40% of them injured 1 person, and the proportion of people who blew up more than 2 people was very small.

This shows that especially in the professional league, the role of HE grenades is more to repel the enemy, or to attack the enemy who is alone. HE grenades are really not very necessary in daily match rounds.

8.3 Data Fidelity Validating

In section 8.2 we filtered the dataset by dropping the records `seconds` field are empty. That is because these entries have `0` in the `land_pos_x` and `land_pos_y`, and mostly don't cost a victim damage. Therefore we can consider these data invalid.

As for the records that the `land_pos_x` and `land_pos_y` field are `'0.0'`, they usually have the valid time value and victim value, we care about what are these data entries mean.

Listing 8-3 Grenades Data Validating

```
#查看是不是炸自己了
gre_list.select('file','seconds','nade_land_x','nade_land_y','
    att_pos_x','att_pos_y','vic_pos_x','vic_pos_y').\
     filter("nade_land_x='0.0'").orderBy('file').show(100)

#查看是不是位置为0.0的全是空爆？
gre_list.select('nade','nade_land_x','nade_land_y','att_pos_x','
    att_pos_y','vic_pos_x','vic_pos_y').\
```

```
7     filter("nade_land_x='0.0'").groupBy('nade').count().orderBy('
    count').show(100)
```

file	seconds	nade_land_x	nade_land_y	att_pos_x	att_pos_y	vic_pos_x	vic_pos_y
esea_match_137797...	994.6065	0.0	0.0	842.7267	452.0041		
esea_match_137797...	4016.6040000000003	0.0	0.0	1353.4260000000002	875.76		
esea_match_137797...	4020.938	0.0	0.0	30.899690000000003	1460.731		
esea_match_137797...	983.56	0.0	0.0	-211.6366	1816.471		
esea_match_137797...	1892.595	0.0	0.0	863.3273	425.2411		
esea_match_137797...	4041.075	0.0	0.0	36.4708	-1348.49		
esea_match_137797...	4058.63	0.0	0.0	670.9488	1425.984		
esea_match_137797...	1913.75	0.0	0.0	360.10900000000004	-942.9717		
esea_match_137797...	4152.181	0.0	0.0	-305.5988	-501.2994		
esea_match_137797...	4011.7690000000002	0.0	0.0	-50.88208	-584.2587		
esea_match_137797...	976.7381	0.0	0.0	-27.033070000000002	-109.1463		
esea_match_137797...	1762.731	0.0	0.0	1697.6329999999998	1.9715470000000002		
esea_match_137797...	2145.755	0.0	0.0	816.6481	2320.6820000000002	40.59077	896.6697
esea_match_137797...	2145.755	0.0	0.0	816.6481	2320.6820000000002	10.67295	900.1696

Fig. 8-3 Check the victim locations of '0.0' landed grenades

The first results are shown as Fig. 8-3, these records have normal values in time and victim position. And we count the types of grenade of this records, shown as Fig. 8-4, they have every kind of grenades.

nade	count
HE	2486
Decoy	3404
Flash	7824
Smoke	9707
Molotov	34529
Incendiary	40009

Fig. 8-4 Group Count of '0.0' landed grenades

Considering only High-Explosive grenades, Molotov and incendiaries can explode in the air, we have reason to believe that these records are somehow lost the data of the grenade locations, or thrown out of map bounds, but still considered as valid entries.

8.4 Damages per Grenades

We draw density plot of grenade damage records to see how much HP or ARM damages taken each time per grenade.

Fist to select five kinds of grenade, group count the damages records, store in five Pandas DataFrames separately.

Listing 8-4 Grenades Damage Counting

```
#查看这些投掷物，都造成了多少伤害？
#下面在filter内筛选出不同的nade种类
he_dmg_count=gre_list.select('nade','hp_dmg','arm_dmg').filter("nade='HE'")

he_dmg_count_pd=he_dmg_count.toPandas()
mo_dmg_count_pd=gre_list.select('nade','hp_dmg','arm_dmg').filter("nade='Molotov'").toPandas()
in_dmg_count=gre_list.select('nade','hp_dmg','arm_dmg').filter("nade='Incendiary'").toPandas()
fl_dmg_count=gre_list.select('nade','hp_dmg','arm_dmg').filter("nade='Flash'").toPandas()
sm_dmg_count=gre_list.select('nade','hp_dmg','arm_dmg').filter("nade='Smoke'").toPandas()
dc_dmg_count=gre_list.select('nade','hp_dmg','arm_dmg').filter("nade='Decoy'").toPandas()
```

Then Draw the distributing density curves (Fig. 8-5)to find out what the most likely damages taken by each record.

Since Decoy and Flash may do no damage to players, we only show the rest 3 grenades.

```
fig = plt.figure(figsize=(14, 15))

#画概率密度函数
plt.subplot(2,2,1)
plt.title('HE Grenade HP Damage Density', fontsize=14, fontweight='bold')
sns.kdeplot(he_dmg_count_pd['hp_dmg'].rename('HP'),shade=True,color='red')

plt.subplot(2,2,2)
plt.title('HE Grenade ARM Damage Density', fontsize=14, fontweight='bold')
sns.kdeplot(he_dmg_count_pd['arm_dmg'].rename('ARM'),shade=True,color='orange')

plt.subplot(2,2,3)
plt.title('Molotov HP Damage Density', fontsize=14, fontweight='bold')
```

```
sns.kdeplot(mo_dmg_count_pd['hp_dmg'].rename('HP'),shade=True,color='blue')

plt.subplot(2,2,4)
plt.title('Incendiary HP Damage Density', fontsize=14, fontweight='bold')
sns.kdeplot(in_dmg_count['hp_dmg'].rename('HP'),shade=True,color='green')
```

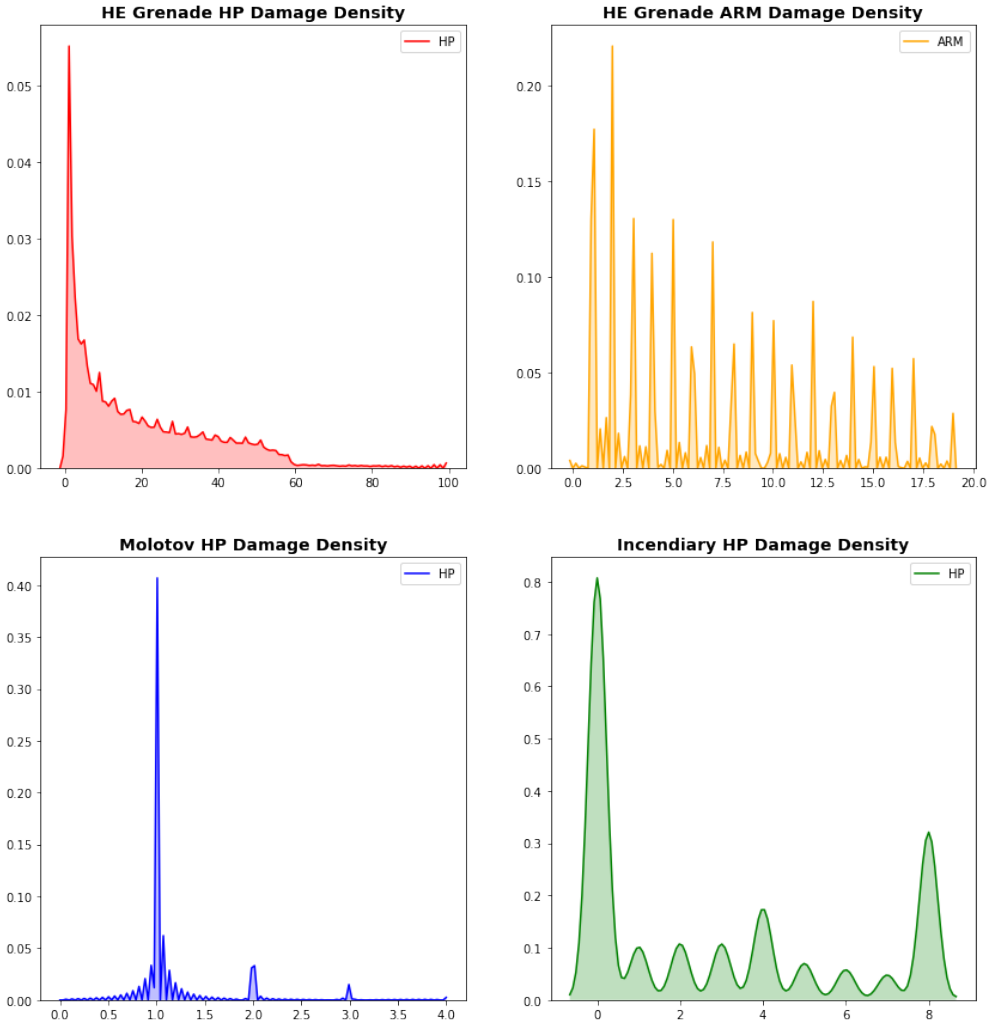


Fig. 8-5 Damage Density per Grenade

As we can see in the Fig. 8-5, most High-Explosive grenades don't cost zero HP damage, and the ARM damages remain below 20. The Molotov has mostly damages at 1, that because it can cause 1 damage at a time, and max damage is 4,but incendiaries of CT has damages records from 0 to 8.

8.5 Locations of grenades landed

In this section, we draw heatmaps of the grenade landing locations, similarly in section 9.1.

8.5.1 Preprocess of Data

First add map field to grenade DataFrame by joining the `meta` table with the `gre` table.

Listing 8-5 Adding map Locations

```
meta_part=meta_df.select('file','round','map').withColumnRenamed("file", "file1").withColumnRenamed("round", "round1")
gre_1=gre_list.join(meta_part,on=[(gre_list.file==meta_part.file1)& (gre_list.round==meta_part.round1)],how="left").\
                select('file','round','seconds','map','nade','hp_dmg','arm_dmg','nade_land_x','nade_land_y',\
                       'att_pos_x','att_pos_y','vic_id','vic_pos_x','vic_pos_y')
gre_1=gre_1.join(map_bounds,on='map',how='left')
```

Then convert positions on map using the `map_bound` data, using the algorithm of eq. (9-1), like the process in section 9.1.1.

```
gre_rgl_df=gre_1.withColumn("att_pos_x",(gre_1['ResX']*(gre_1['att_pos_x']-gre_1['StartX']))/(gre_1['EndX']-gre_1['StartX'])).\
withColumn("att_pos_y",(gre_1['ResY']*(gre_1['att_pos_y']-gre_1['StartY']))/(gre_1['EndY']-gre_1['StartY'])).\
withColumn("vic_pos_x",(gre_1['ResX']*(gre_1['vic_pos_x']-gre_1['StartX']))/(gre_1['EndX']-gre_1['StartX'])).\
withColumn("vic_pos_y",(gre_1['ResY']*(gre_1['vic_pos_y']-gre_1['StartY']))/(gre_1['EndY']-gre_1['StartY'])).\
withColumn("nade_land_x",(gre_1['ResX']*(gre_1['nade_land_x']-gre_1['StartX']))/(gre_1['EndX']-gre_1['StartX'])).\
withColumn("nade_land_y",(gre_1['ResY']*(gre_1['nade_land_y']-gre_1['StartY']))/(gre_1['EndY']-gre_1['StartY']))
```

8.5.2 Heatmap Plotting

Take the `de_dust2` for example, draw the heatmap of Smoke, Flash, Molotov and Incendiaries landing locations, as the Listing 8-6, like in the section 9.1.2

Listing 8-6 Heatmap Drawing of Grenade Locations

```
import imageio

#使用kdeplot画等高线
smap = 'de_dust2'

plot_df1 = gre_rgl_df.filter((gre_rgl_df.map == smap) & (gre_rgl_df.nade == 'Smoke')).toPandas()
plot_df2 = gre_rgl_df.filter((gre_rgl_df.map == smap) & (gre_rgl_df.nade == 'Flash')).toPandas()
plot_df3 = gre_rgl_df.filter((gre_rgl_df.map == smap) & (gre_rgl_df.nade == 'Molotov')).toPandas()
plot_df4 = gre_rgl_df.filter((gre_rgl_df.map == smap) & (gre_rgl_df.nade == 'Incendiary')).toPandas()

bg = imageio.imread('./archive/'+smap+'.png')
fig = plt.figure(figsize=(14, 15))

plt.subplot(2,2,1)
plt.grid(b=True, which='major', color='w', linestyle='--', alpha=0.25)
plt.imshow(bg, zorder=0, extent=[0.0, 1024, 0.0, 1024])
plt.xlim(0,1024)
plt.ylim(0,1024)
sns.kdeplot(plot_df1['nade_land_x'], plot_df1['nade_land_y'], cmap='YlOrBr',levels=30,bw=10)
plt.title('Smoke Landing Position')

plt.subplot(2,2,2)
plt.grid(b=True, which='major', color='w', linestyle='--', alpha=0.25)
plt.imshow(bg, zorder=0, extent=[0.0, 1024, 0.0, 1024])
plt.xlim(0,1024)
plt.ylim(0,1024)
sns.kdeplot(plot_df2['nade_land_x'], plot_df2['nade_land_y'], cmap='Blues',levels=30,bw=10)
plt.title('Flash Landing Position')

plt.subplot(2,2,3)
plt.grid(b=True, which='major', color='w', linestyle='--', alpha=0.25)
plt.imshow(bg, zorder=0, extent=[0.0, 1024, 0.0, 1024])
plt.xlim(0,1024)
```

```
plt.ylim(0,1024)
sns.kdeplot(plot_df3['nade_land_x'], plot_df3['nade_land_y'], cmap='Greens',levels=30,bw=10)
plt.title('Molotov Landing Position')

plt.subplot(2,2,4)
plt.grid(b=True, which='major', color='w', linestyle='--', alpha=0.25)
plt.imshow(bg, zorder=0, extent=[0.0, 1024, 0.0, 1024])
plt.xlim(0,1024)
plt.ylim(0,1024)
sns.kdeplot(plot_df4['nade_land_x'], plot_df4['nade_land_y'], cmap='Purples',levels=30,bw=10)
plt.title('Incendiary Landing Position')
```

Here we only post result of `de_dust2`, as the Fig. 8-6 shows.

The landing places are basically fixed to several frequent position.

On the attacking side, the smoke often landed to the CT near A site, cover on the short to A, and the gate of B side. The Flash mostly in the A long which are the cover of both sides. And the Molotov of T mostly fire on the car on the A site, break of B, and the inside of A.

On the defending side, smoke are landed to the tunnel of B, the certral gate. The incendiaries for CT are mostly landed on the tunnel of B, the gate of A, short to A and the central gate.

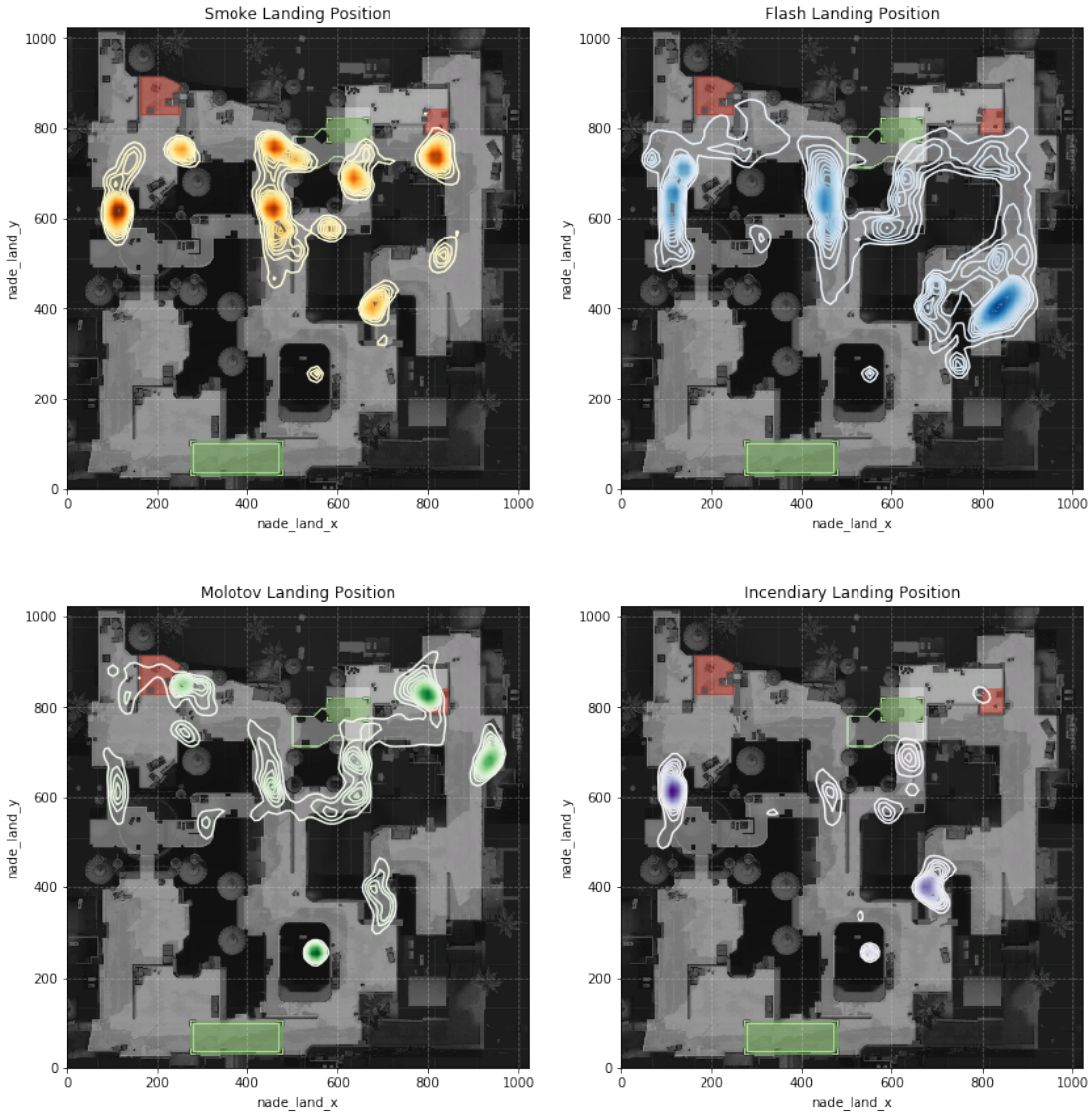


Fig. 8-6 Grenade locations on `de_dust2`

Chapter IX

Map Analysis

9.1 Heatmaps of Frequency of Overall Damage

We care about the attack positions of both T and CT side in each maps. In this section we are going to draw heatmaps of overall damage attacking positions on each map.

To simplify the records and avoid redundant data, we choose the dataset of common competitive rounds, which is the `mm_df` DataFrame.

9.1.1 Converting map locations

Let's first only isolate for active duty maps as they are the maps that most competitive players really care about. I also want to first convert the in-game coordinates to overhead map coordinates typically using eq. (9-1).

$$\text{绝对坐标} = \frac{\text{分辨率} \times \left(\text{像素坐标} - \text{起始像素坐标}\right)}{\text{地图长宽度}} \tag{9-1}$$

The calculation of ***Map Length*** (or width) as the eq. (9-2) shows.

$$\text{终止像素坐标} - \text{起始像素坐标} \tag{9-2}$$

The converting algorithm can be explained like Fig. 9-1

First select duty maps from `mm_df`, then left join the `map_bounds` DataFrames, which will add map bounds fields to `mm_df`.

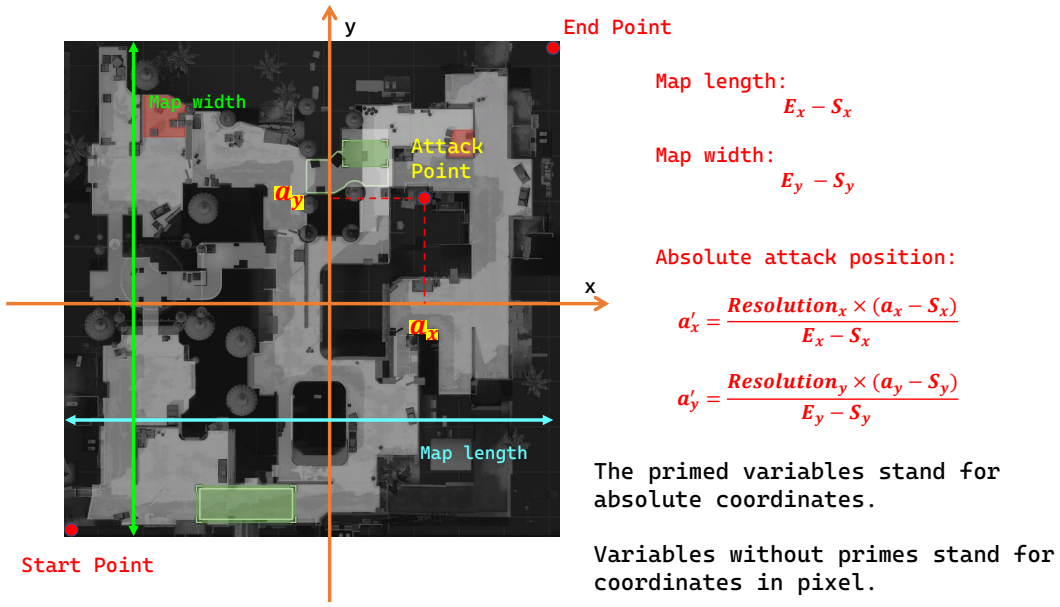


Fig. 9-1 Coordinates Transferring

```
#修正地图坐标
active_duty_maps = ['de_cache', 'de_cbble', 'de_dust2', 'de_inferno',
    'de_mirage', 'de_overpass', 'de_train']
mm_valid = mm_df[mm_df['map'].isin(active_duty_maps)]
md=mm_valid.join(map_bounds,on='map',how='left')
```

Then calculate the absolute coordinates replacing the original data,and saved as regulared DataFrame `mm_rgl_df`.

```
mm_rgl_df=md.withColumn("att_pos_x", (md['ResX']*(md['att_pos_x']-md['StartX']))/(md['EndX']-md['StartX'])).\
    withColumn("att_pos_y", (md['ResY']*(md['att_pos_y']-md['StartY']))/(md['EndY']-md['StartY'])).\
    withColumn("vic_pos_x", (md['ResX']*(md['vic_pos_x']-md['StartX']))/(md['EndX']-md['StartX'])).\
    withColumn("vic_pos_y", (md['ResY']*(md['vic_pos_y']-md['StartY']))/(md['EndY']-md['StartY']))
```

Fig. 9-2 Coordinates Regularizing

9.1.2 Plotting Heatmaps

Take *dust2* for example, filtered rows matching the de_dest2 map, then plot the 2-D density figure of attack position x and y to draw a frequency heatmap of both T & CT sides, shown as 9-3(a), with map picture backgrounded and grid tagged.

Change the `smap` variable into different maps, we can get frequency heat maps in different match play maps. In total we plotted the frequency heatmaps of

de_dust2, de_mirage, de_inferno, de_train, de_cache and de_overpass map, here we only show two of them, as Fig. 9-3 shown.

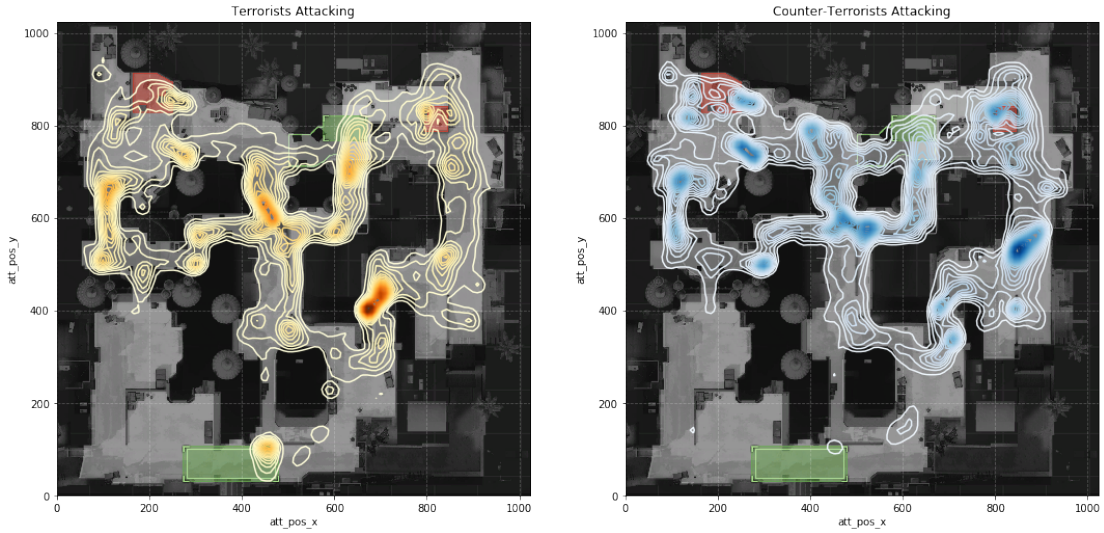
Listing 9-1 Drawing Heatmap[6]

```
import imageio
#使用kdeplot画等高线
smap = 'de_dust2'

plot_df1 = mm_rgl_df.filter((mm_rgl_df.map == smap) & (mm_rgl_df.
    att_side == 'Terrorist')).toPandas()
plot_df2 = mm_rgl_df.filter((mm_rgl_df.map == smap) & (mm_rgl_df.
    att_side == 'CounterTerrorist')).toPandas()

bg = imageio.imread('./archive/'+smap+'.png')
fig, (ax1, ax2) = plt.subplots(1,2,figsize=(18,16))
ax1.grid(b=True, which='major', color='w', linestyle='--', alpha
    =0.25)
ax2.grid(b=True, which='major', color='w', linestyle='--', alpha
    =0.25)
ax1.imshow(bg, zorder=0, extent=[0.0, 1024, 0., 1024])
ax2.imshow(bg, zorder=0, extent=[0.0, 1024, 0., 1024])
plt.xlim(0,1024)
plt.ylim(0,1024)
#sns.kdeplot(plot_df1['att_pos_x'], plot_df1['att_pos_y'], cmap='
    YlOrBr',levels=25, thresh=.1,bw_adjust=11,ax=ax1)
sns.kdeplot(plot_df1['att_pos_x'], plot_df1['att_pos_y'], cmap='
    YlOrBr',levels=30,bw=10,ax=ax1)
ax1.set_title('Terrorists Attacking')

sns.kdeplot(plot_df2['att_pos_x'], plot_df2['att_pos_y'], cmap='Blues
    ',levels=30,bw=10,ax=ax2)
ax2.set_title('Counter-Terrorists Attacking')
```

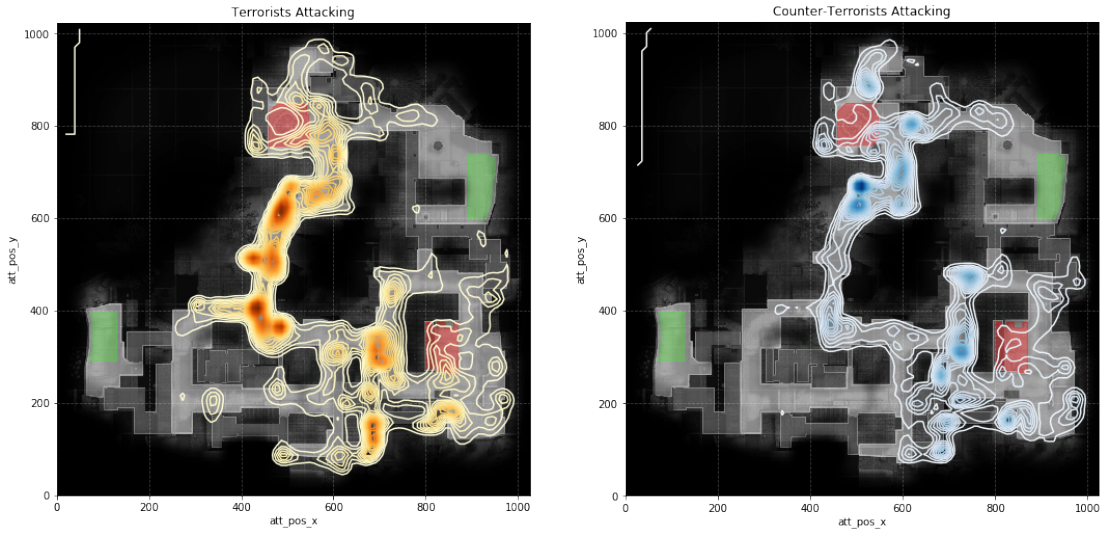
Noticing the `bw` variable is out of date in the late version of the `seaborn` package, but it works here due to the old version of Anaconda on the server.

It is clearly to see in Fig. 9-3(a) of de_dust2, the attacking and defending position mainly focus on gate of A long, the midway and tunnel to B site, which are keys to manage the game. Same result came to the banana and short to A in de_inferno according to Fig. 9-3(b).

More heatmap analysis can be found at Fig. 10-25.



(a) Dust2



(b) Inferno

Fig. 9-3 Frequency Heatmap of two maps. Orange lines stand for attack position of T, and blue lines stand for CT. The depth of the color indicates the frequency of attack points

9.2 Winning Chances Analysis

In this section we analyze of the winning percentage of different maps. By analyzing the wins of both T and CT in all rounds, the winning percentage of each map T and CT is calculated to judge the balance of the map.

9.2.1 Overall Map Winning Chances

Use `meta_df` for analyzing, group count the winning rounds of different maps, and evaluate the overall winning rate by rounds.

Listing 9-2 Overall Winning Chance

```
#显示地图和胜利的一方
map_winrate=meta_df.select(meta_df.file, meta_df.map, meta_df.
    winner_side ).\
filter("winner_side!='None'").\
groupby("map","winner_side").count().\
orderBy("map","winner_side")
```

Sent result to Pandas, and plot pie charts of overall winning chances in different maps by `pyplot`.

```
map_winrate_pd=map_winrate.toPandas()
import matplotlib.pyplot as plt

fig=plt.figure(figsize=(16,10))
#axes = fig.subplots(nrows=2, ncols=4)

plt.suptitle('Winning Rate of All Maps', fontsize=16, fontweight='
    bold')
#设置了j后面一次循环结束就加一，可以对应画布上的位置
j=1
#用unique()方法去重
for map in map_winrate_pd.map.unique():
    plt.subplot(2,4,j)
    plt.title('Winning Rate of %s' % map)
    Winrate_Pie=map_winrate_pd[map_winrate_pd['map']==map]
    plt.pie(Winrate_Pie['count'],labels=Winrate_Pie['winner_side'],
    autopct='%1.2f%%')
    j=j+1

labels = map_winrate_pd['winner_side'].unique()
#plt.tight_layout()
```

```
fig.legend(labels,loc='upper center',bbox_to_anchor=(0.39, 0.8))
plt.show()
```

The result as the Fig. 9-4 shows, most map are relatively balanced, nuke is most tend to CT, and dus2 is most tend to T side, mirage seems is the most balanced map.

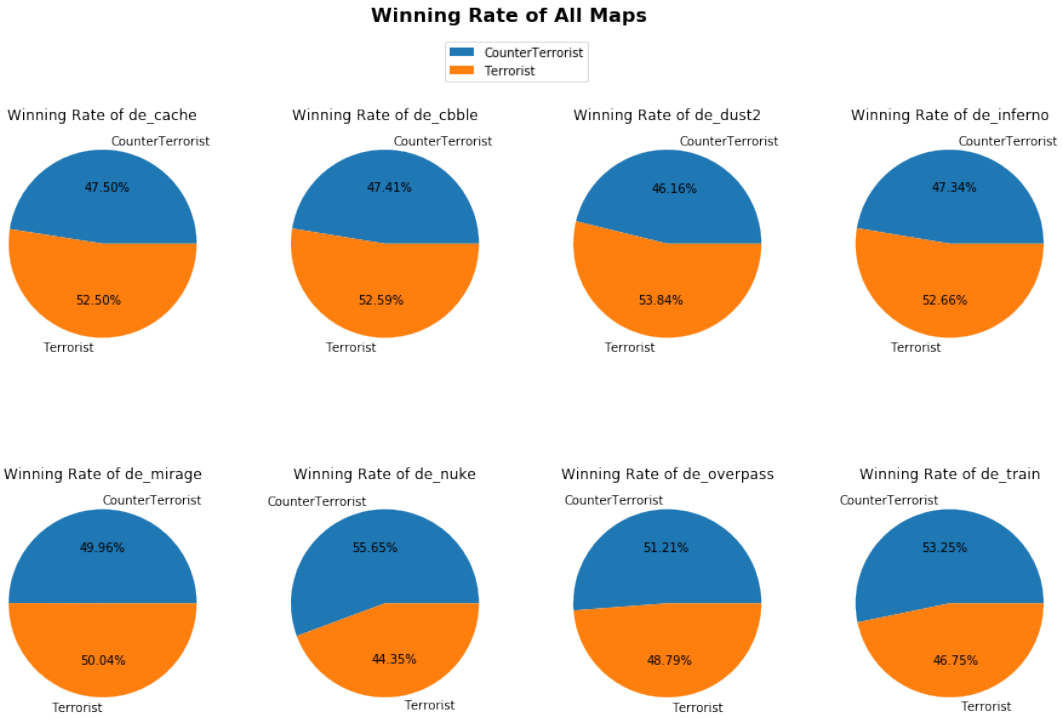


Fig. 9-4 Overall map winning chances

9.2.2 Winning chances associated with bomb

Left join the meta table with kill table, we can add a column of bomb planted flag to group count the winning chances by bomb planting situation .

```
#合并File和Round字段
meta_part=meta_df.select(F.concat_ws("_", F.col("file"), F.col("round").alias("round_key")),'winner_side').\
                withColumnRenamed("concat_ws(_, file, round AS `round_key`)", "round_key")

#选择出is_bomb_planted为True的条目
kill_part=kill_df.select(F.concat_ws("_", F.col("file"), F.col("round").alias("round_key")),'is_bomb_planted').\
               withColumnRenamed("concat_ws(_, file, round AS `round_key`)", "round_key").\
               filter("is_bomb_planted='True'")

#整合到meta表中，将未join上的替换成False
meta_joined=meta_part.join(kill_part ,on="round_key", how="left").fillna({"is_bomb_planted":'False'})
```

Fig. 9-5 Join table of meta and kill

As there may be True and False entries of the `is_bomb_planted` field in the same round, so we have to drop those duplicated records to join the tables correctly.

Different from the following steps, we join on a key newly contact the `file` and `round` fields, and replace Null values to `'False'`, which can avoid filling wrongly.

Group count by bomb flag and winning side field, then send to Pandas and draw bar chart using `sns.barplot`, as shown in Fig. 9-7.

```
bomb_win_pd=meta_joined.groupBy('is_bomb_planted','winner_side').count().orderBy('is_bomb_planted').toPandas()
```

```
bomb_win_pd.drop(bomb_win_pd[(bomb_win_pd.winner_side=='None')].index,inplace=True)
bomb_win_pd
```

	is_bomb_planted	winner_side	count
0	False	Terrorist	58948
1	False	CounterTerrorist	142887
2	True	CounterTerrorist	114185
4	True	Terrorist	344351

```
fig = plt.figure(figsize=(6, 7))
colors1=sns.color_palette("hls",2)
sns.barplot(x="is_bomb_planted", y="count", hue="winner_side",data=bomb_win_pd,palette=colors1) #使用orient="h"参数控制列为主值
plt.title('Winning Chance of T', fontsize=16, fontweight='bold')
```

Fig. 9-6 Group Sorting and Plotting Chart

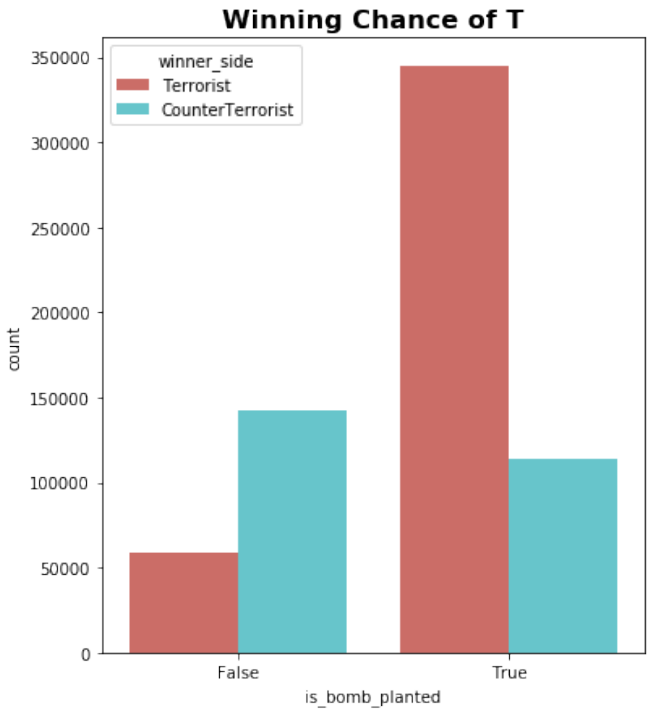


Fig. 9-7 Winning Chances by Bomb Planting Situation

9.2.3 Winning chances of bomb planting site

Combined with the placement of the bombs, we can make the impact of placing bombs at different points on different maps on the winning rate of both sides.

Choose `mm_df` as data source, since it is a dataset with fully filed including map and bomb information. First counting the bomb planting rounds in A and B site.

9.2.3.1 Planting counts in Maps

Let's now look at the Number of bomb plants by site. This statistic tells us the T's preferences for deciding which site to take during the round. Although the possibility of rotates are always there, it gives us a good idea of what to expect.

Group count by map and bombsite, then sent to Pandas.

Listing 9-3 Winning Chances by bomb and maps

```
#统计炸弹安放点
site_count_pd=mm_rgl_df.filter("bomb_site!=''").select(['file', 'map', 'round', 'bomb_site']).dropDuplicates().\
            groupby(['map', 'bomb_site']).count().toPandas()
site_count_pd.set_index(['map','bomb_site']).unstack()
```

Drawing bar plots of the result, as the Fig. 9-8 shows.

```
fig = plt.figure(figsize=(6, 8))
colors1=sns.color_palette("hls",2)
sns.barplot(y='count',x='map',hue='bomb_site',data=site_count_pd, palette=colors1)
plt.suptitle('Bomb Plant Site Counting', fontsize=16, fontweight='bold')
```

9.2.3.2 Winning Chances

The win rates for the safe positions are listed here, and calculate the winning chances in different maps. Drop null and duplicated records, then group count by `'map', 'bomb_site', 'winner_side'`.

```
bomb_prob_overall = mm_rgl_df.filter("bomb_site!=''").select(['file', 'map', 'round', 'bomb_site','winner_side']).\
dropDuplicates().groupby(['map', 'bomb_site', 'winner_side']).count().toPandas().\
set_index(['map','bomb_site','winner_side'])
```

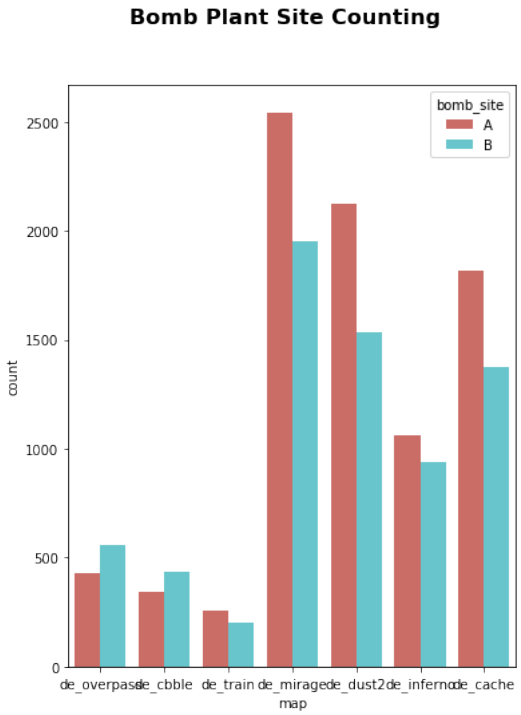
Then calculate winning percentage using Pandas `apply` method.

```
#计算百分率
bomb_prob_overall_pct = bomb_prob_overall.groupby(level=[0,1]).apply(
    lambda x: 100 * x / float(x.sum()))
bomb_prob_overall_pct.unstack('map')
```

The result are shown as Fig. 9-9.

	count	
bomb_site	A	B
map		
de_cache	1819	1375
de_cbble	346	434
de_dust2	2123	1537
de_inferno	1065	938
de_mirage	2544	1955
de_overpass	428	557
de_train	257	204

(a) Pandas



(b) Barplot

Fig. 9-8 Bomb Planting Counts

		count						
	map	de_cache	de_cbble	de_dust2	de_inferno	de_mirage	de_overpass	de_train
bomb_site	winner_side							
A	CounterTerrorist	24.079164	22.543353	21.479039	19.436620	22.366352	22.897196	19.844358
	Terrorist	75.920836	77.456647	78.520961	80.563380	77.633648	77.102804	80.155642
B	CounterTerrorist	24.218182	17.972350	24.137931	17.803838	26.496164	25.493716	44.117647
	Terrorist	75.781818	82.027650	75.862069	82.196162	73.503836	74.506284	55.882353

Fig. 9-9 Winning Chances of Bomb Planting Sites in Different Maps

9.2.4 Post-plant Win Probabilities by Advantages

According to the number advantage (or disadvantage) of T relative to CT after the bomb is installed, the winning rate under different numbers of people can be analyzed.

This one could be further disseminated but we want to be able to look at the win probabilities post plant given the context of how many Ts and CTs are alive at that time.

We can first find for each round the post-plant situation (if it was planted at all) and calculate advantages. I've given two options (XvX, X alive T vs X alive CT) or more generally by differences (e.g 5 Ts − 3 CTs = 2).[6]

Still using regularized `mm_rgl_df` DataFrame, add XvX field to the original DataFrame.

To evaluate the alive players on T & CT side after the bomb planted, we have to count players been killed on both side until the bomb just planted. Which can be done by summarizing total damages.

Group summarize total HP damages on both victim side in round when bomb has not been planted, when damage counts to 100, named as 1 player died. Then stored to a grouped list table `mm_xvx`, which will be joined with the original table later.

Listing 9-4 XvX Winning Chances

```
from pyspark.sql.functions import *
import pyspark.sql.functions as F

mm_xvx=mm_rgl_df.filter(mm_rgl_df.is_bomb_planted == False).groupBy('file','round','vic_side').agg({'hp_dmg':'sum'}).\
                        withColumnRenamed("sum(hp_dmg)", "tot_dmg")
mm_xvx=mm_xvx.withColumn('alive',(5-(mm_xvx.tot_dmg/100)).cast("Int"))
```

Then seperate the `mm_xvx` table into `mm_aliveT` and `mm_aliveCT`, stored T and CT side alive players separately.

We did a special tip, changing the key fields, which are `file` and `round` here, into different names, which are `file1` and `round2`, for joining tables easily.

```
mm_aliveT=mm_xvx.filter("vic_side='Terrorist'").withColumn('aliveT',mm_xvx.alive).select('file','round','aliveT')
mm_aliveCT=mm_xvx.filter("vic_side='CounterTerrorist'").withColumn('aliveCT',mm_xvx.alive).\
```

```
9         withColumnRenamed("file", "file1").withColumnRenamed("round",
    "round1").select('file1','round1','aliveCT')
```

Then join the alive players to the original table, and calculate player advantages or disadvantages on T over CT.

```
10 mm_xvx=mm_aliveT.join(mm_aliveCT,on=[(mm_aliveCT.file1==mm_aliveT.
       file)& (mm_aliveCT.round1==mm_aliveT.round)],how="left")
11 mm_xvx=mm_xvx.withColumn('XvX',(mm_xvx.aliveT - mm_xvx.aliveCT)).
       select('file1','round1','XvX')
12 mm_adwin=mm_rgl_df.join(mm_xvx,on=[(mm_xvx.file1==mm_rgl_df.file)& (
       mm_xvx.round1==mm_rgl_df.round)],how="left")
```

Now we can calculate the Win probabilities by advantages, note that I isolate for just Terrorist because having CT columns (which is redundant), muddles the table.

Group count and evaluate the winning chances over advantages in player number, and on different bomb plating site of different maps.

```
13 bomb_prob = mm_adwin.filter(F.col("XvX").isNotNull()).filter("
       bomb_site!=''").\
14 select(['file', 'round', 'map', 'bomb_site', 'XvX', 'winner_side']).
       dropDuplicates().\
15 groupby(['XvX', 'map', 'bomb_site', 'winner_side']).count().orderBy('
       XvX','bomb_site').toPandas().\
16 set_index(['XvX', 'bomb_site', 'map','winner_side'])
17 bomb_prob_pct = bomb_prob.groupby(level=[0,1,2]).apply(lambda x: 100
       * x / float(x.sum()))
18 bomb_prob_pct.xs('Terrorist', level=3).unstack().fillna(0)
```

The results are shown as Fig. 9-10

		count						
	map	de_cache	de_cbble	de_dust2	de_inferno	de_mirage	de_overpass	de_train
XvX	bomb_site							
-3	A	16.666667	0.000000	18.750000	10.000000	4.545455	0.000000	0.000000
	B	6.250000	33.333333	0.000000	0.000000	16.666667	0.000000	0.000000
-2	A	17.567568	22.222222	27.536232	34.146341	27.027027	23.076923	11.111111
	B	17.857143	16.666667	20.000000	44.186047	27.480916	21.428571	25.000000
-1	A	44.696970	53.061224	47.826087	57.534247	41.284404	46.296296	36.000000
	B	45.495495	45.454545	49.600000	50.370370	46.604938	40.000000	31.372549
0	A	72.093023	68.539326	72.359155	75.666667	71.735791	65.178571	70.149254
	B	73.958333	72.641509	71.794872	82.671480	74.368231	71.724138	59.649123
1	A	88.888889	89.108911	89.126853	90.460526	90.558511	91.489362	93.750000
	B	90.463215	89.510490	88.805970	93.560606	88.653846	86.585366	75.000000
2	A	96.099291	98.333333	96.569921	97.905759	96.995708	98.360656	100.000000
	B	95.141700	97.727273	96.825397	98.101266	97.967480	96.739130	87.500000
3	A	100.000000	100.000000	100.000000	100.000000	99.367089	100.000000	100.000000
	B	98.214286	100.000000	100.000000	100.000000	98.437500	100.000000	100.000000
4	A	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	0.000000
	B	100.000000	100.000000	100.000000	100.000000	100.000000	0.000000	0.000000

Fig. 9-10 T side Winning Chances over advantages in player number of Bomb Planting Sites in Different Maps

9.3 Post-Plant ADR of Defending position

I've always wondered if it's better to play post-plants in-site our out-of-site during a one-man up/down or equal situation. In-site has the advantage of peeking when the CTs are clearing outer-site spots but the con of being in a spot where you are forced to duel the CTs. Outer-site has pro of baiting shots and playing time but having to peek into the CT when he is defusing. Let's isolate for only 1v1, 2v1, 1v2, 2v2 situations and look at average ADR differential when you play inner site or outer site.

Before we do that though, we have to define what is considered Inner/Outer site. Using some basic rectangles, I can draw sites on the map and then define them via simple top-left, bottom-right coordinates.

First determine the boundaries of bomb sites. Defining the callout and make up a function returning callout flag.

Listing 9-5 Callout Defining

```
callouts = {
'de_cache': {
    'B inner': [[310,782,413,865]],
    'A inner': [[278,165,388,320]]
},
'de_cbble': {
    'B inner': [[625,626,720,688]],
    'A inner': [[134,746,225,861]]
},
'de_train': {
    'B inner': [[405,754,607,812]],
    'A inner': [[582,462,713,539]]
},
'de_dust2': {
    'B inner': [[162,99,256,199]],
    'A inner': [[786,182,846,239]]
},
'de_mirage': {
    'B inner': [[188,245,286,345]],
    'A inner': [[498,737,610,835]]
},
'de_inferno': {
    'B inner': [[410,115,548,320]],
    'A inner': [[783,638,877,765]]
},
'de_overpass': {
    'B inner': [[686,294,745,359]],
    'A inner': [[452,174,560,272]]
},
}

def find_callout(x,y,m,buffer=10):
callout = 'N/A'
for c,coord in callouts[m].items():
    for box in coord:
        if ((box[2]+buffer >= x >= box[0]-buffer) &
            (buffer+(1024-box[1]) >= y >= (1024-box[3])-buffer)):
                callout = c
return callout
```

For example, plot the bomb site aera on `de_dest2` map.

Listing 9-6 Bomb Site Aera Plotting

```
smap = 'de_dust2'

def calc_plot_coord(l):
    tx,ty,bx,by = l
    by = 1024-by; ty=1024-ty;
    w = bx-tx; h= ty-by;
    return (tx,by,w,h)

bg = imread('./archive/'+smap+'.png')
fig, ax = plt.subplots(figsize=(10,10))
ax.grid(b=True, which='major', color='w', linestyle='--', alpha=0.25)
ax.imshow(bg, zorder=0, extent=[0.0, 1024, 0., 1024])
plt.xlim(0,1024)
plt.ylim(0,1024)
patches = []
for k,coords in callouts[smap].items():
    for c in coords:
        x,y,w,h = calc_plot_coord(c)
        patches.append(mpatches.Rectangle((x,y),w,h))
        plt.text(x+w/2.3,y+h/2.3, s=k, size= 8, color='w')
colors = np.linspace(0, 1, len(patches))
collection = PatchCollection(patches, cmap=plt.cm.hsv, alpha=0.4)
collection.set_array(np.array(colors))
ax.add_collection(collection)
```

Now let's convert the coordinates of T attackers or victims to callouts (either N/A: outer site or Inner A/Inner B)

```
bomb_dist = mm_adwin[(mm_adwin['XvX'].isin([-1, 0, 1]))
&(mm_adwin['bomb_site']!='')
&((mm_adwin['vic_side'] == 'Terrorist') | (mm_adwin['att_side'] == '
    Terrorist'))].toPandas()
bomb_dist['att_callout'] = bomb_dist.apply(lambda x: find_callout(x['
    att_pos_x'], x['att_pos_y'], x['map'], buffer=5), axis=1)
bomb_dist['vic_callout'] = bomb_dist.apply(lambda x: find_callout(x['
    vic_pos_x'], x['vic_pos_y'], x['map'], buffer=5), axis=1)
```

Now calculate ADR by site:

```
bomb_dist_total_dmg_att = bomb_dist.groupby(['file', 'map', 'round',
    'att_callout', 'att_id'])['hp_dmg'].sum()
bomb_dist_total_dmg_vic = bomb_dist.groupby(['file', 'map', 'round',
    'vic_callout', 'vic_id'])['hp_dmg'].sum()
dmg_dealt = bomb_dist_total_dmg_att.groupby(['map', 'att_callout']).
    agg(['count', 'mean'])
```

```
dmg_rec = bomb_dist_total_dmg_vic.groupby(['map', 'vic_callout']).agg
    (['count', 'mean'])
dmg_diff = dmg_dealt['mean'] - dmg_rec['mean']
dmg_diff.unstack('att_callout')
```

The final result are shown as Fig. 9-11

att_callout	A inner	B inner	N/A
map			
de_cache	4.053875	8.481475	5.486609
de_cbble	-33.550420	6.402715	12.267927
de_dust2	-3.421429	5.076923	5.599861
de_inferno	11.939250	4.642857	8.242179
de_mirage	-1.317458	5.058519	7.965681
de_overpass	1.577273	-8.491991	4.336491
de_train	-16.496970	-7.260684	9.366601

Fig. 9-11 In and Out ADR

Chapter X

Experimental Results and Analysis

We show final results and chart figures in this chapter, mainly divided into the following four parts: Weapon system, Economics impact, Grenade analysis and Map analysis.

10.1 Weapon System

There are 8 types of weapon in the CS:GO game, the summarized damages are shown as Table 10-1.

Table 10-1 Damages of Different weapon types

wp_type	tot_dmg	hp_dmg	arm_dmg
Rifle	177745516.0	156448524.0	21296992.0
Pistol	76737824.0	67355345.0	9382479.0
Sniper	39695267.0	38710578.0	984689.0
SMG	27397275.0	23032905.0	4364370.0
Grenade	15249803.0	13021114.0	2228689.0
Heavy	2325035.0	2039414.0	285621.0
Unknown	1854692.0	1295327.0	559365.0
Equipment	583514.0	543252.0	40262.0

The occupation of total damage of different weapon types are shown as Fig. 10-1. It can be seen that the damage of the Rifles to the health value (HP) or

the armor (Arm) is the highest, followed by the pistol, the sniper rifle, submachine gun, props, heavy weapons and equipment.

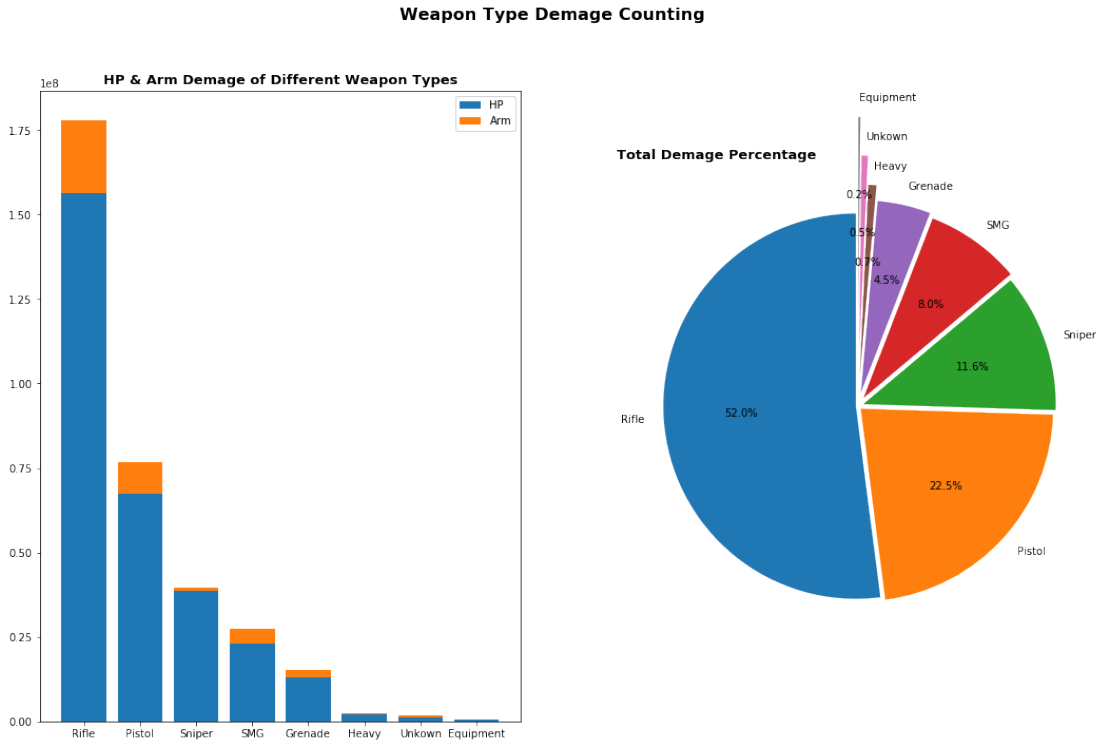


Fig. 10-1 Damages occupations of different weapon types

And to see HP and Armor damages separately, as Fig. 10-2 shows,

The most damage to armor is the Rifles, followed by pistols, submachine guns, props, sniper rifles, heavy weapons and equipment. It is worth noting that the damage of the submachine gun to the armor is higher than the damage to the HP. It may be that the players who use the submachine gun are more accustomed to charging, causing non-lethal damage to multiple enemies. Sniper rifles do more damage to HP, but lower damage to armor, which reflects the characteristics of one-shot kills.

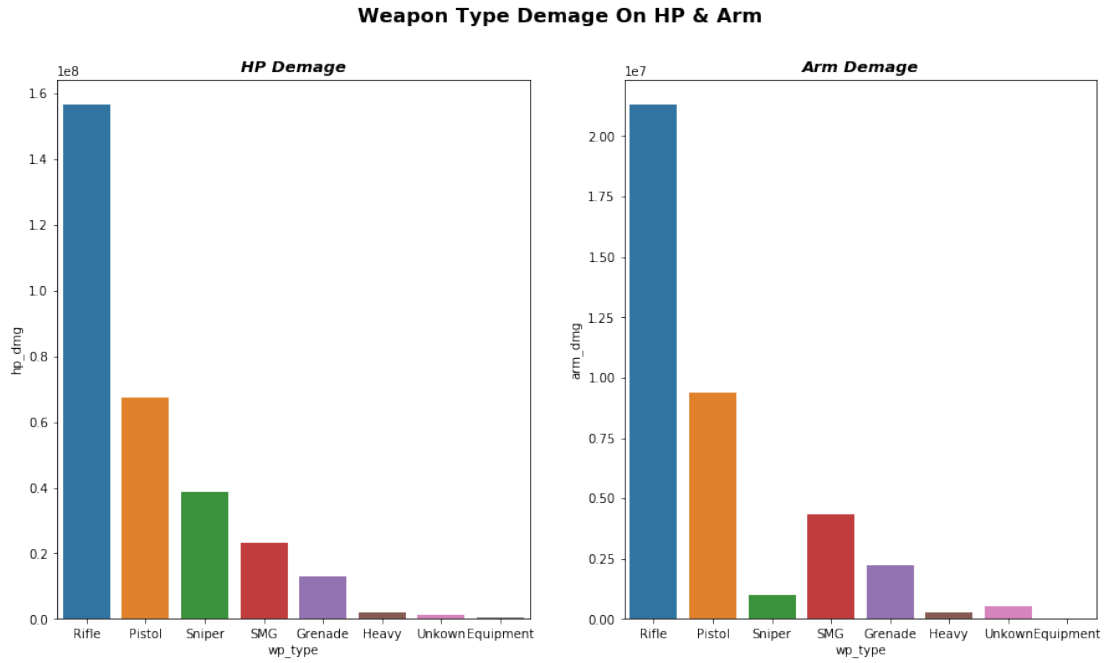


Fig. 10-2 HP and Armor damages of different weapon types

10.1.1 Weapon Ranking

And there are 40 weapons in 7 weapon types, Fig. 10-3 and Fig. 10-4 show the total damages and the occupation of those weapons.

AK47 takes the highest in Rifles, AWP takes the most in Snipers, Desert Eagle takes the most in pistols, and it leads the rest of guns, including other Rifles and all of SMGs.

High-Explosive grenades get higher than all SMGs and the most damages in SMG is MP7.

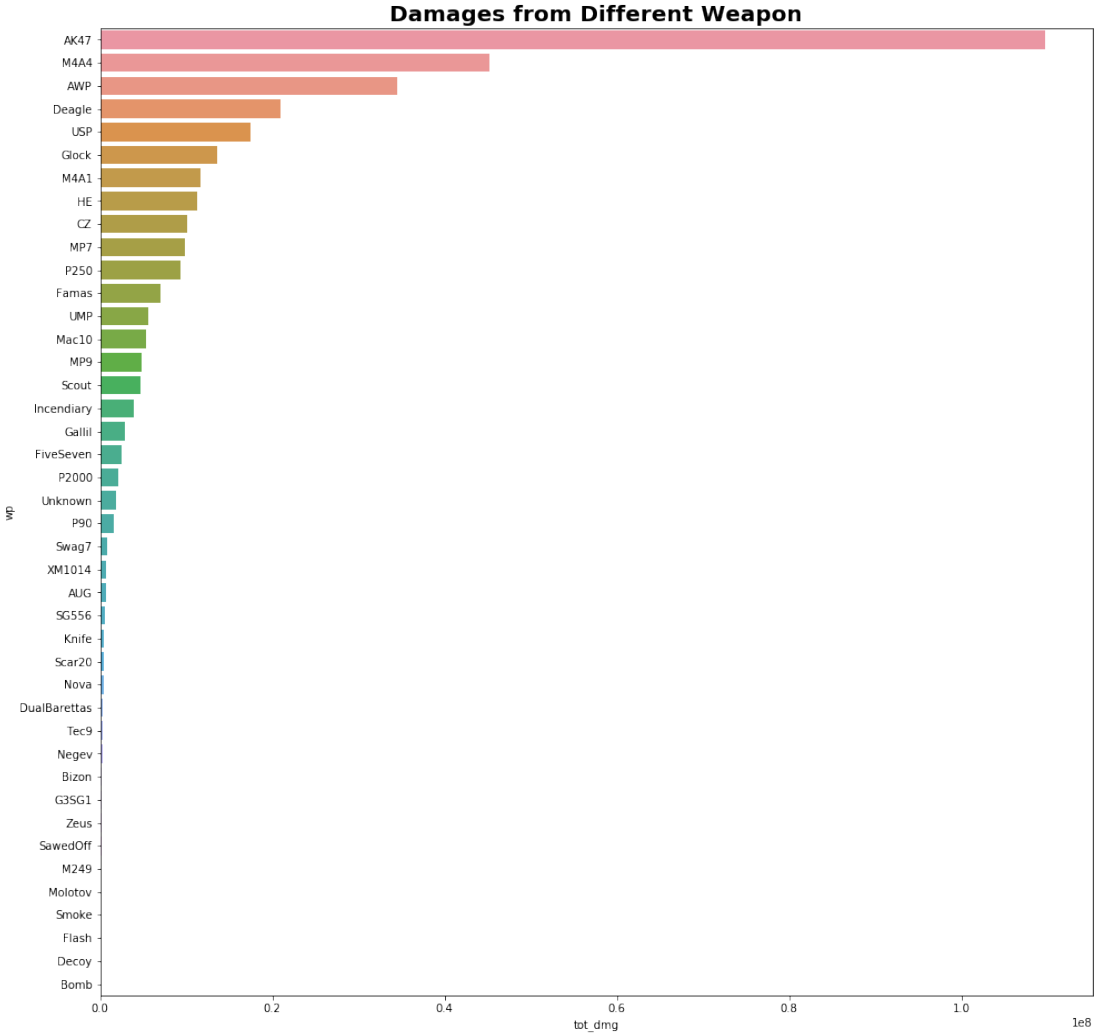


Fig. 10-3 Damages of different weapons

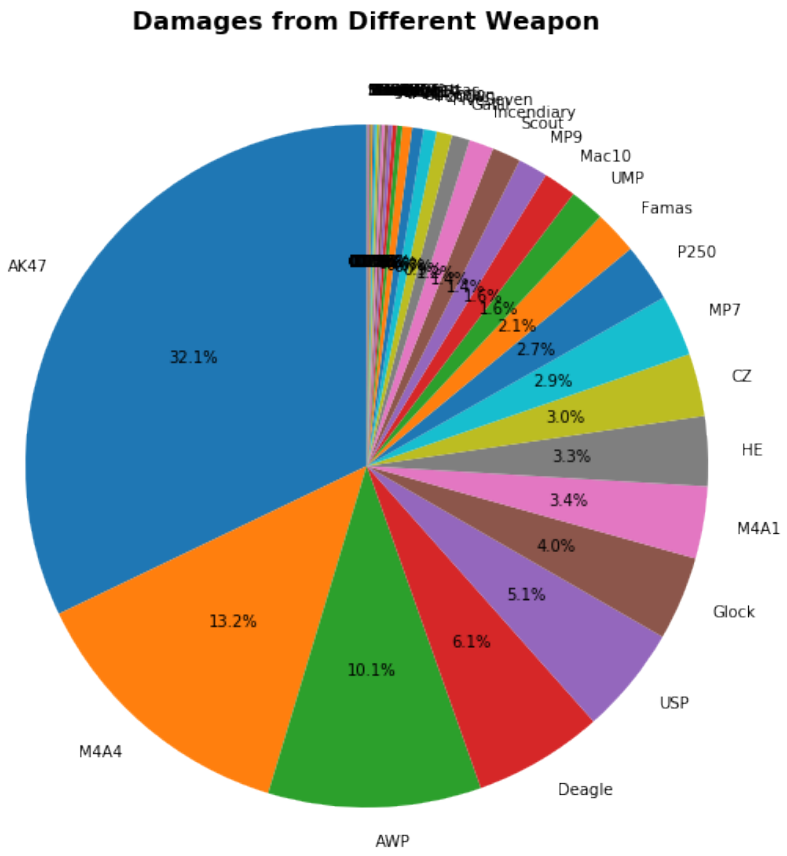


Fig. 10-4 Damage occupation of different weapons

Now showing the top 10 and last 10 weapons particularly.

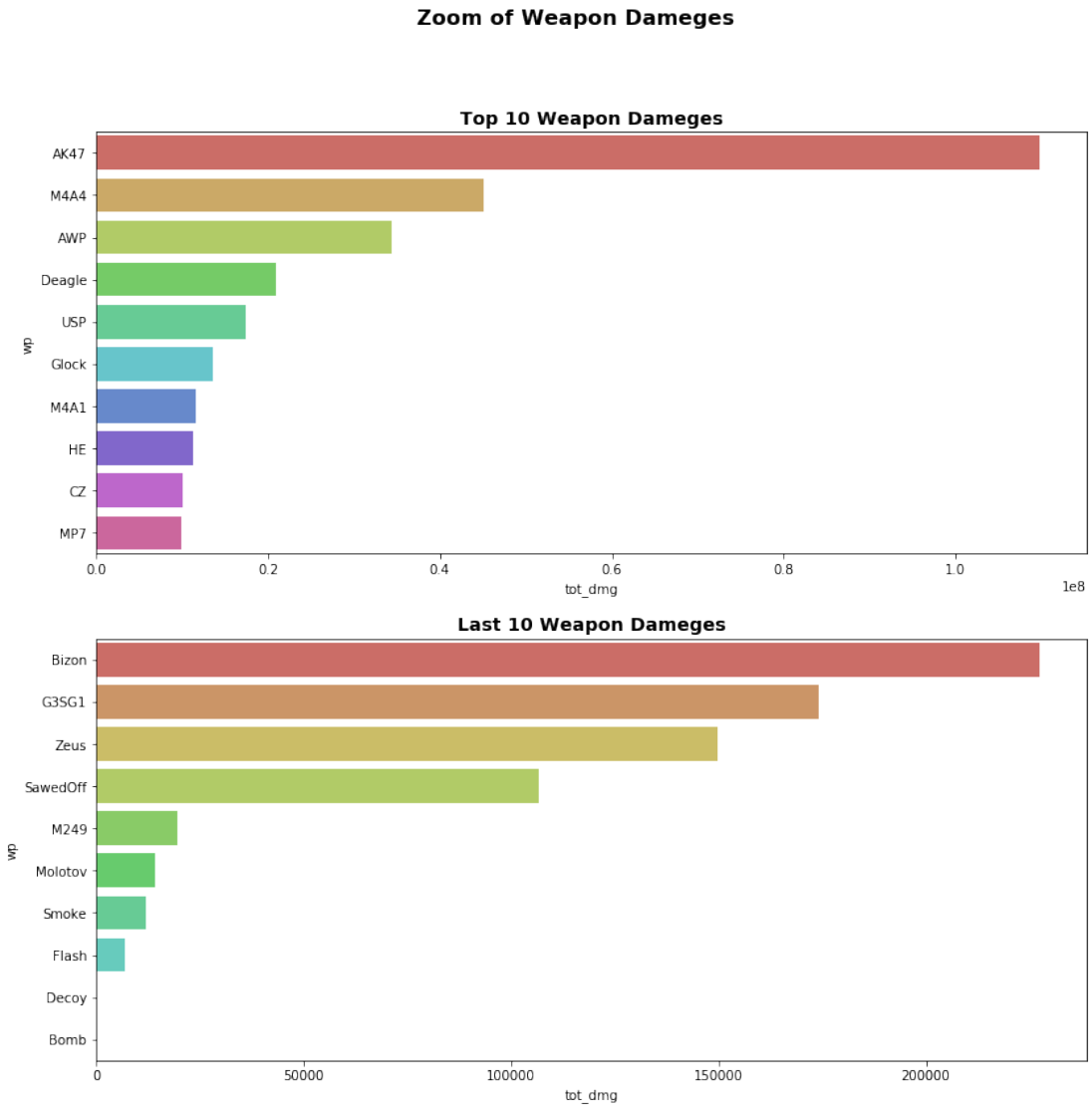


Fig. 10-5 Damage occupation of different weapons

The long guns, AK-47, M4A4, and AWP, Desert Eagle stand for the most of damages, noticing this is the dataset in late 2018, so M4A4 still take a big part than M4A1.

As we can see from Fig. 10-6, the most weapon of killing is still AK-47, then is AWP, M4A4, Desert Eagle, USP and Glock, M4A1, MP7, CZ-75.

We can make a deduction that when common rounds the players often buy AK-47, M4A4 and AWP, and when ECO and Semi-ECO rounds often buy Desert Eagle and MP7 etc.

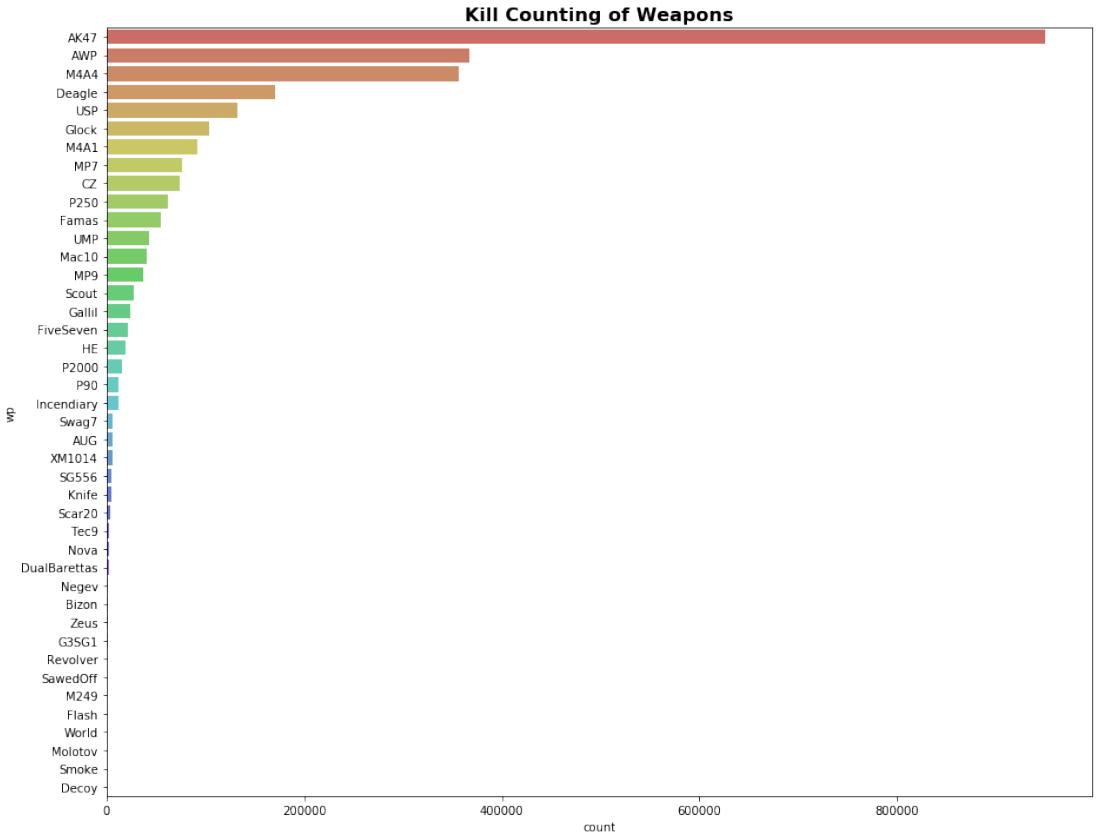


Fig. 10-6 Killing counting of different weapons

10.1.2 Preferences

In order to find the popularity of the exclusive weapons of T side, we ranked the most killing weapons on the CT side when there is only 1 CT alive, as Fig. 10-7 shows.

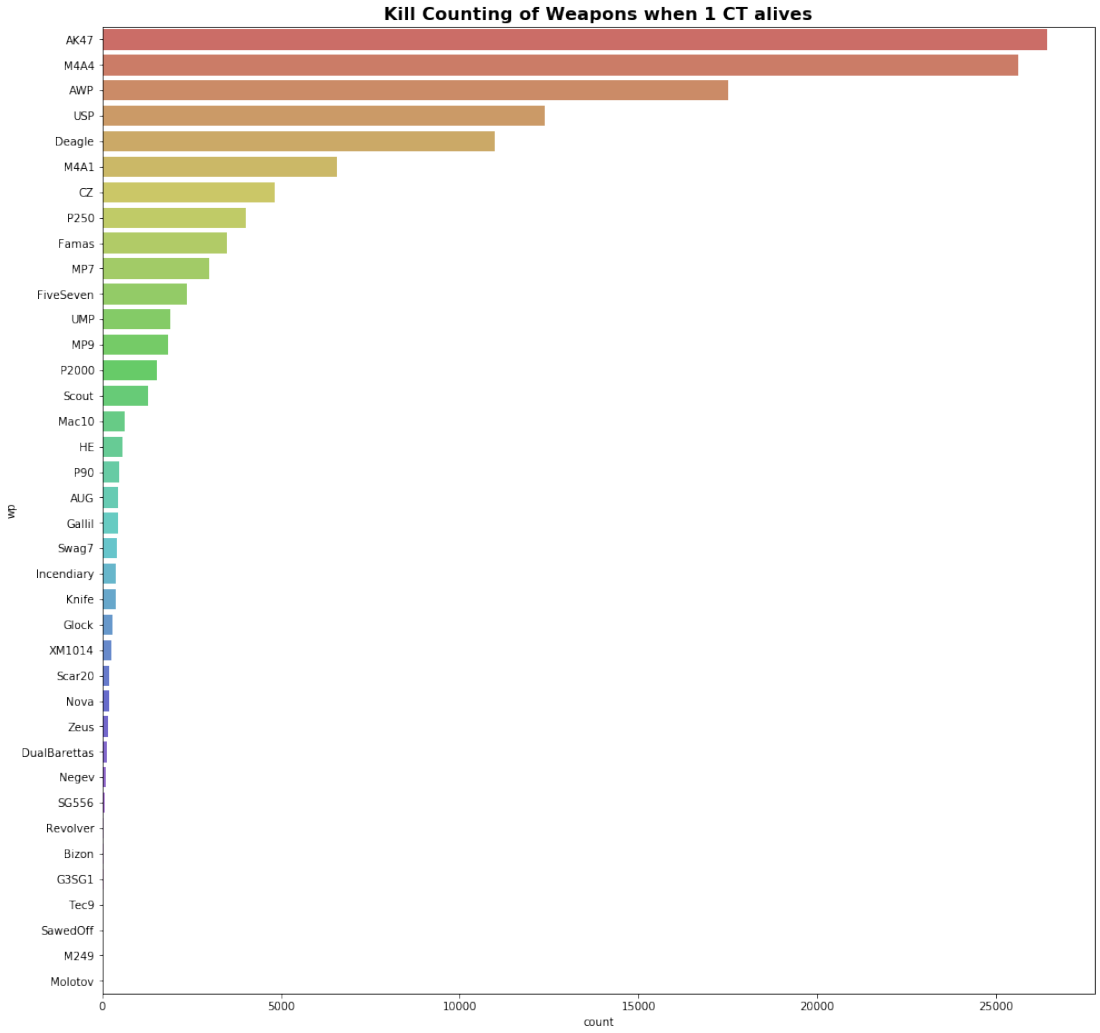


Fig. 10-7 Killing counting of CT when 1 alive

It is clearly to see that CTs are most likely to give a kill by AK-47 picked up from T side.

Then we ranked the most killing weapons on the economic weak side in all ECO rounds, as Fig. 10-8 shows.

We find out the most popular weapon of ECO rounds is the Desert Eagle.

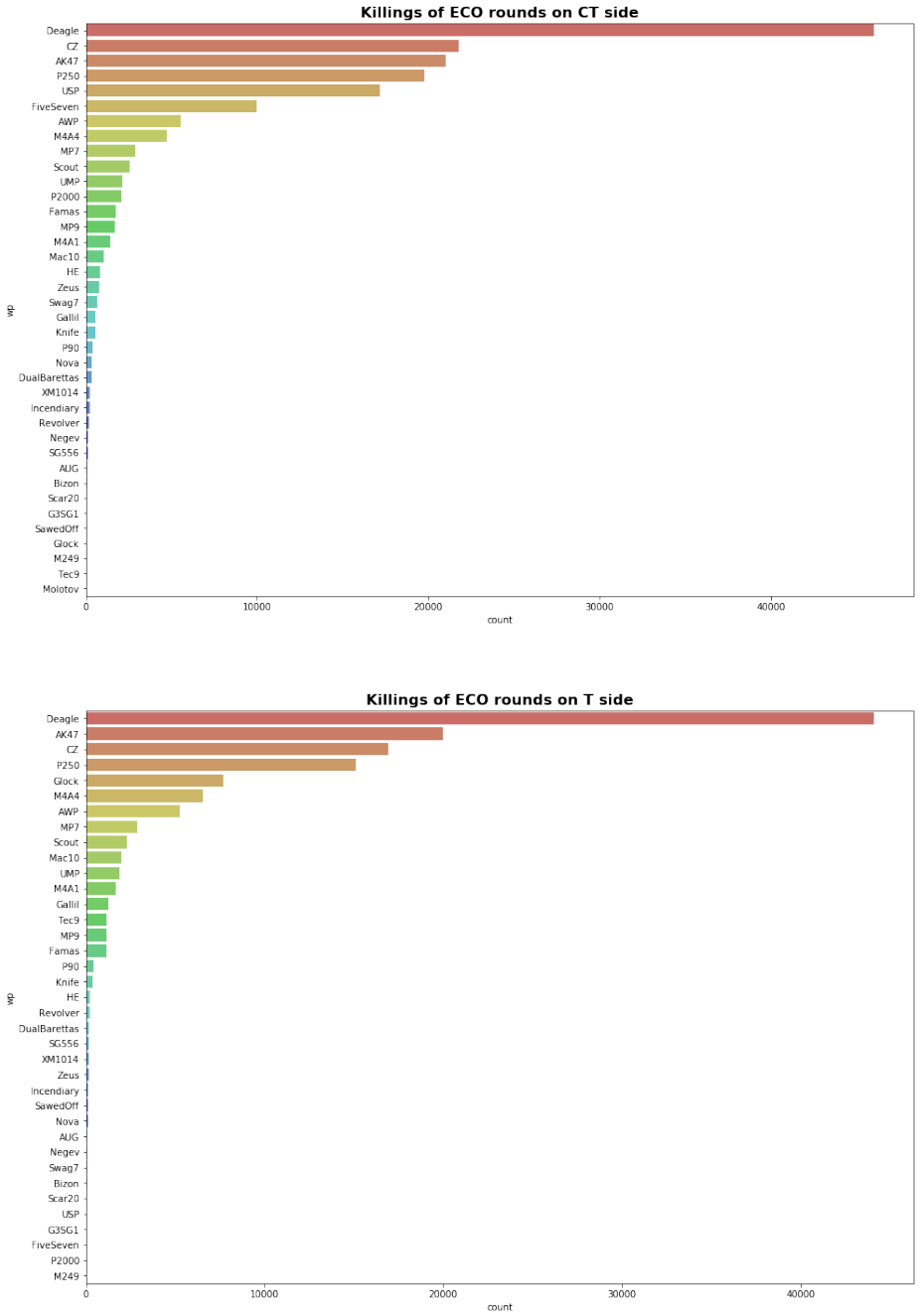


Fig. 10-8 Killing weapons in ECO rounds

10.1.3 Weapon Prosperities

There are 9 part of body could cause damage. The records and the damages of each hitbox are shown as Table 10-2, and the occupation chart are shown as the following figures.

Table 10-2 Hitboxes Statistics

hitbox	count
Chest	4225631
Stomach	1811283
Head	1478655
Generic	1394708
RightArm	715480
RightLeg	333583
LeftLeg	298783
LeftArm	258125
8	21934

(a) records

hitbox	tot_dmg	hp_dmg	arm_dmg
Chest	115286584.0	102746528.0	12540056.0
Head	110507583.0	96296187.0	14211396.0
Stomach	57601555.0	50948744.0	6652811.0
RightArm	19570309.0	17501191.0	2069118.0
Generic	17688686.0	14860317.0	2828369.0
RightLeg	7112794.0	7112794.0	0.0
LeftArm	6935885.0	6165103.0	770782.0
LeftLeg	6319436.0	6319436.0	0.0
8	566094.0	496159.0	69935.0

(b) damages

It can be seen from Fig. 10-9 and Fig. 10-10, that the stomach has more damage entries than the head, but the total damage amount of the head is higher than that of the stomach.

Kicks have no armor.

It can be seen that the comprehensive headshot rate of all weapons (which can also be understood as the comprehensive headshot rate of all players in the game) is only 16%. More damage in the game is still generated by the Rifles hitting the chest and abdomen.

The head shooting rate of all weapons are shown as Table 10-3 and Fig. 10-11.

USP is the highest heat shooting rate weapon, and P200, Glock, Desert Eagle take the rest places. Rifle guns has less head shooting rate than pistols. the Rifle AK-47 has head shooting rate in the middle.

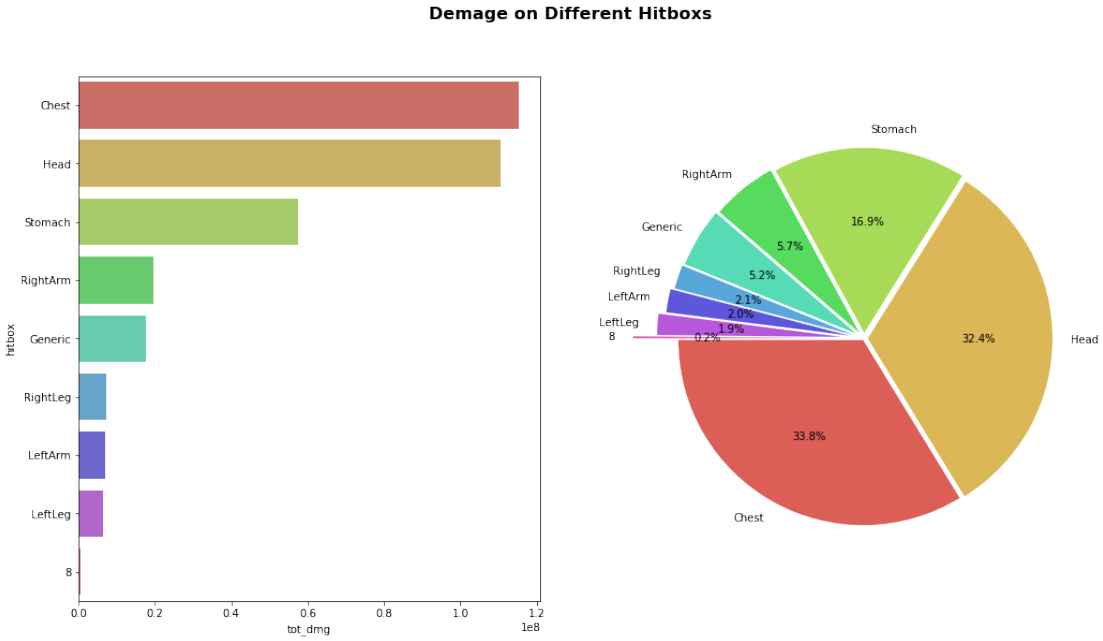


Fig. 10-9 Total damages of different hitbox

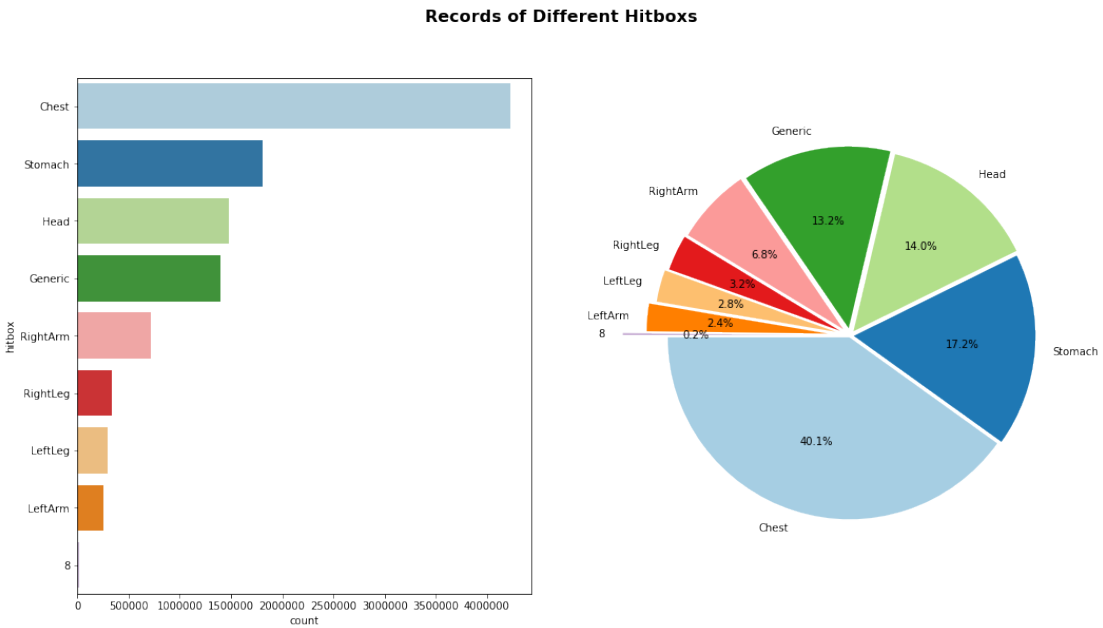


Fig. 10-10 Records of different hitbox

Table 10-3 Example table of the bomb site counting to Pandas

	wp	count_h	count	head_rate
2	USP	130791	462254	0.282942
15	P2000	15398	56231	0.273835
3	Glock	116769	435716	0.267993
4	Deagle	100688	426307	0.236187
16	Scout	14469	64328	0.224925
6	P250	52273	249322	0.209661
24	DualBarettas	1976	9659	0.204576
14	FiveSeven	15793	80439	0.196335
25	Tec9	1868	9786	0.190885
29	SawedOff	713	4126	0.172807
19	Swag7	6556	38449	0.170512
11	MP9	31833	197649	0.161058
5	CZ	54508	340874	0.159907
21	Nova	2828	17900	0.157989
0	AK47	473126	3115867	0.151844
10	Mac10	32604	218457	0.149247
23	SG556	2785	18890	0.147433
12	UMP	30610	208544	0.146780
1	M4A4	194640	1463705	0.132978
31	M249	86	654	0.131498
8	M4A1	48242	368527	0.130905
17	Gallil	12705	98004	0.129638
20	XM1014	5060	40529	0.124849
7	MP7	51855	427649	0.121256
13	Famas	29552	249257	0.118560
26	Bizon	1224	10480	0.116794
22	AUG	2813	24122	0.116616
30	G3SG1	358	3189	0.112261
18	P90	7791	71122	0.109544
27	Negev	809	8543	0.094697
28	Scar20	732	7870	0.093011
9	AWP	37200	415525	0.089525

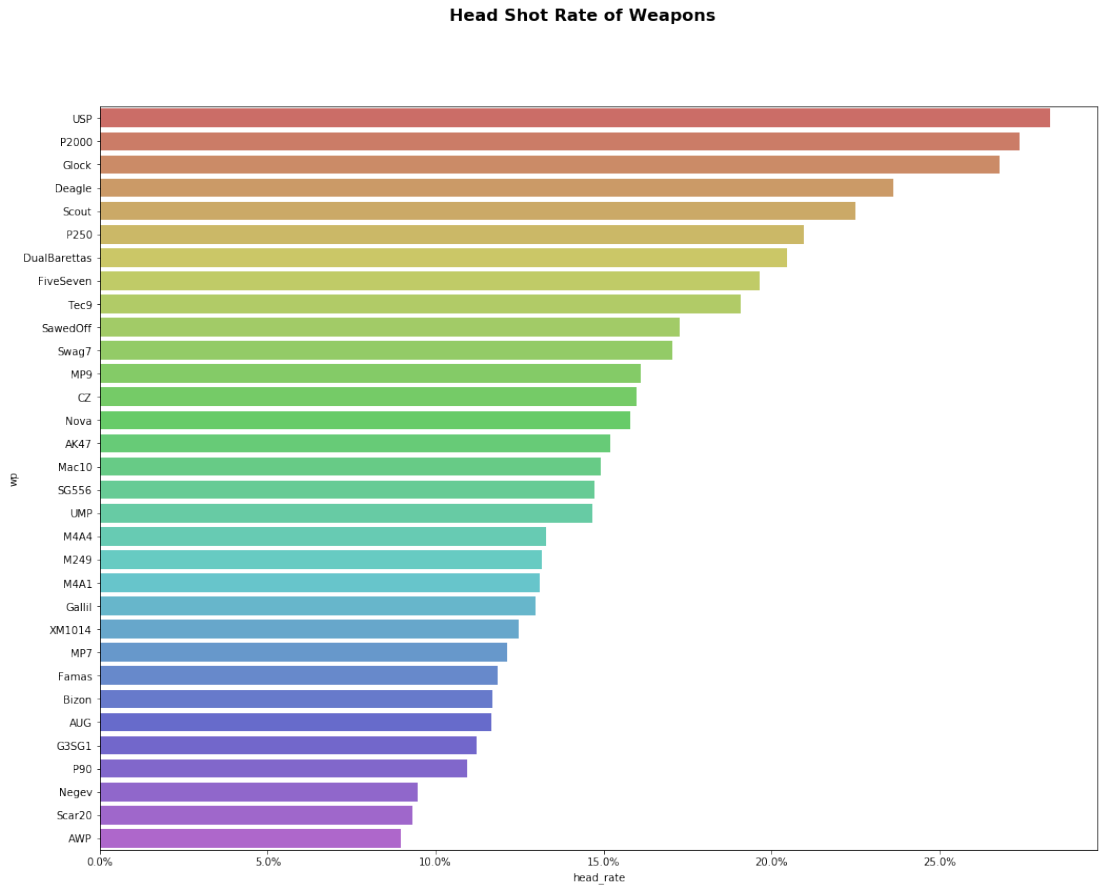


Fig. 10-11 Head Shooting Rate of Weapons

The power of each weapon can be seemed as the average damage per shot, shown as Fig. 10-12.

The stun gun Zeus takes the highest because of it can kill a player in a short distance. Then is the sniper AWP and Scout and Scar20 takes the most power, due to the one-shot nature of the sniper rifle.

The knife occupies a high position, for the most kill in the back.

Desert Eagle get highest power in pistols, and AK-47 takes the most in common Rifles.

Considering different hitbox cause different damages, this overall average power rank result are for reference only.

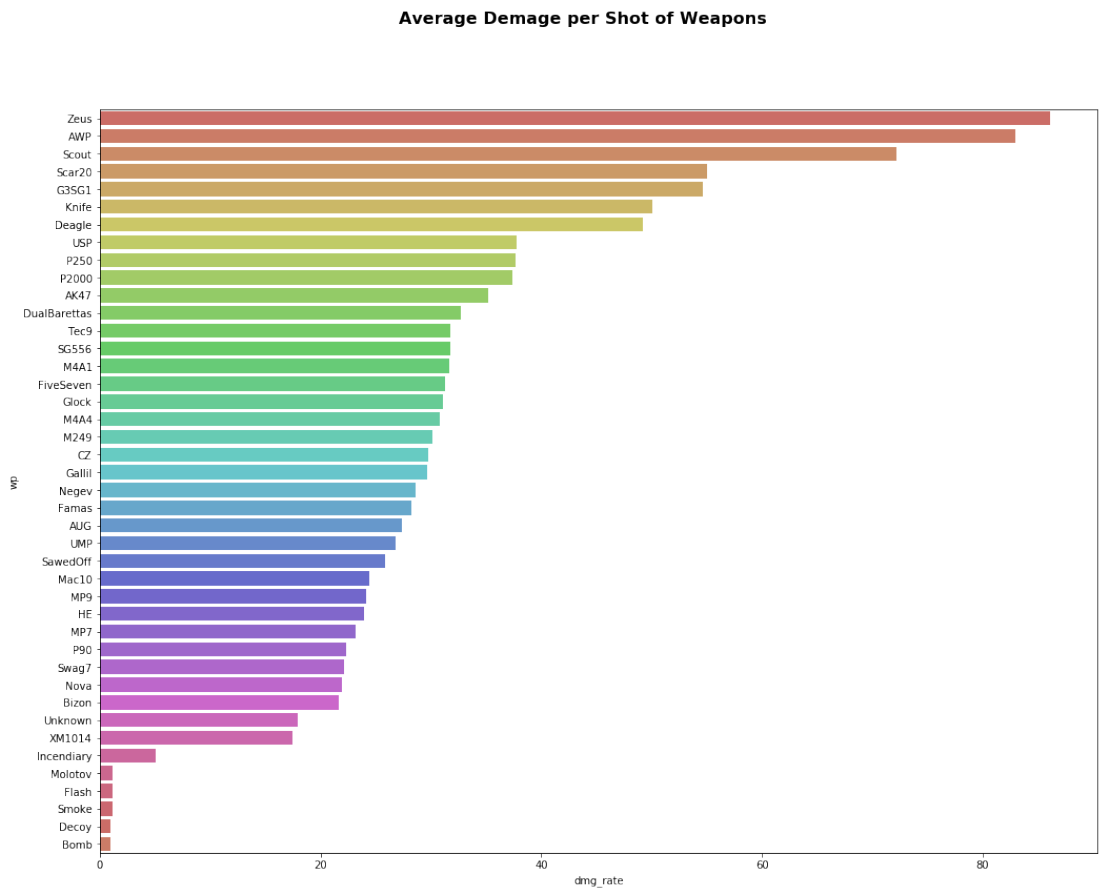


Fig. 10-12 Average Damage per shot of Weapons

10.2 Economic Impact

Table 10-4 Round Type Counting

round_type	count
ECO	128613
PISTOL_ROUND	28114
SEMI_ECO	24444
FORCE_BUY	40876
NORMAL	155582

There are five types of rounds in the CS:GO game, and the normal rounds take 41.2% of all rounds, and over a quarter rounds are ECO rounds. Pistol

rounds, semi pistol rounds and force buy rounds share the similar parts around 10 percent. Because the killing and economic effects are variant in the game, ECO rounds often appears in a play.

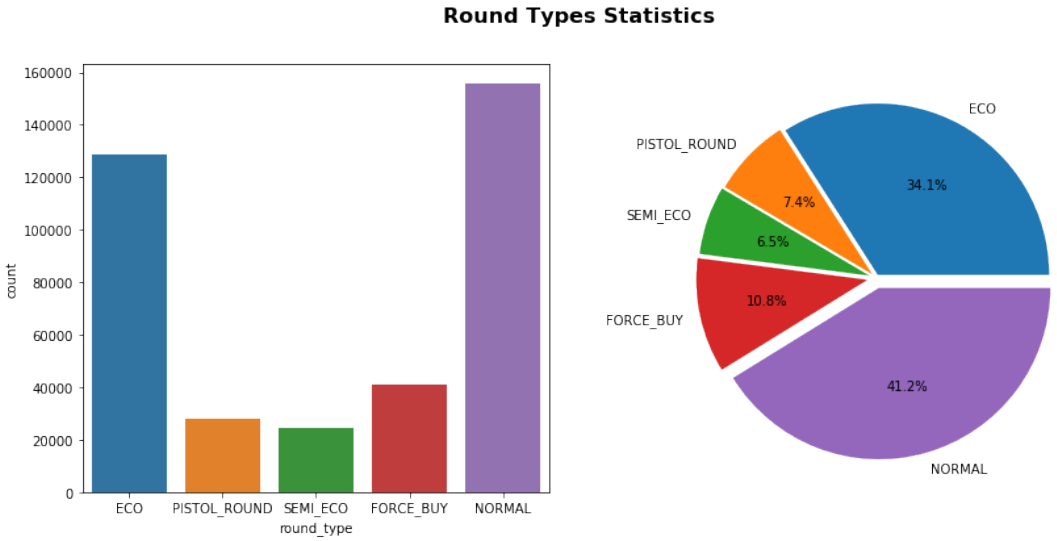


Fig. 10-13 Round Types Counting

Next we see the economics in the game through the total equipment value. Fig. 10-14 shows the estimation curve of probability density distribution and the estimated accumulative curve of the economics of T and CT side.

It can be seen that the highest economy of CT is higher than that of T.

Both T and CT tend to buy ”few equipment” or ”a lot of equipment”, In the low economic stage (ECO), the most probable economic values of T and CT economies are basically the same; In the high economy stage (over $20000), the most likely economy value of CT is higher than the economy of T, which may also be that the equipment price of CT is higher than that of T This law can also be seen from the economic cumulative distribution estimation curve in the figure below.

Teams are very much inclined to buy ”barely” or ”a lot” of gear. Interestingly, Counter-Strikes tend to cost slightly more than Terrorists during purchase rounds, probably because they have a higher limit on the maximum amount of gear that can be purchased.

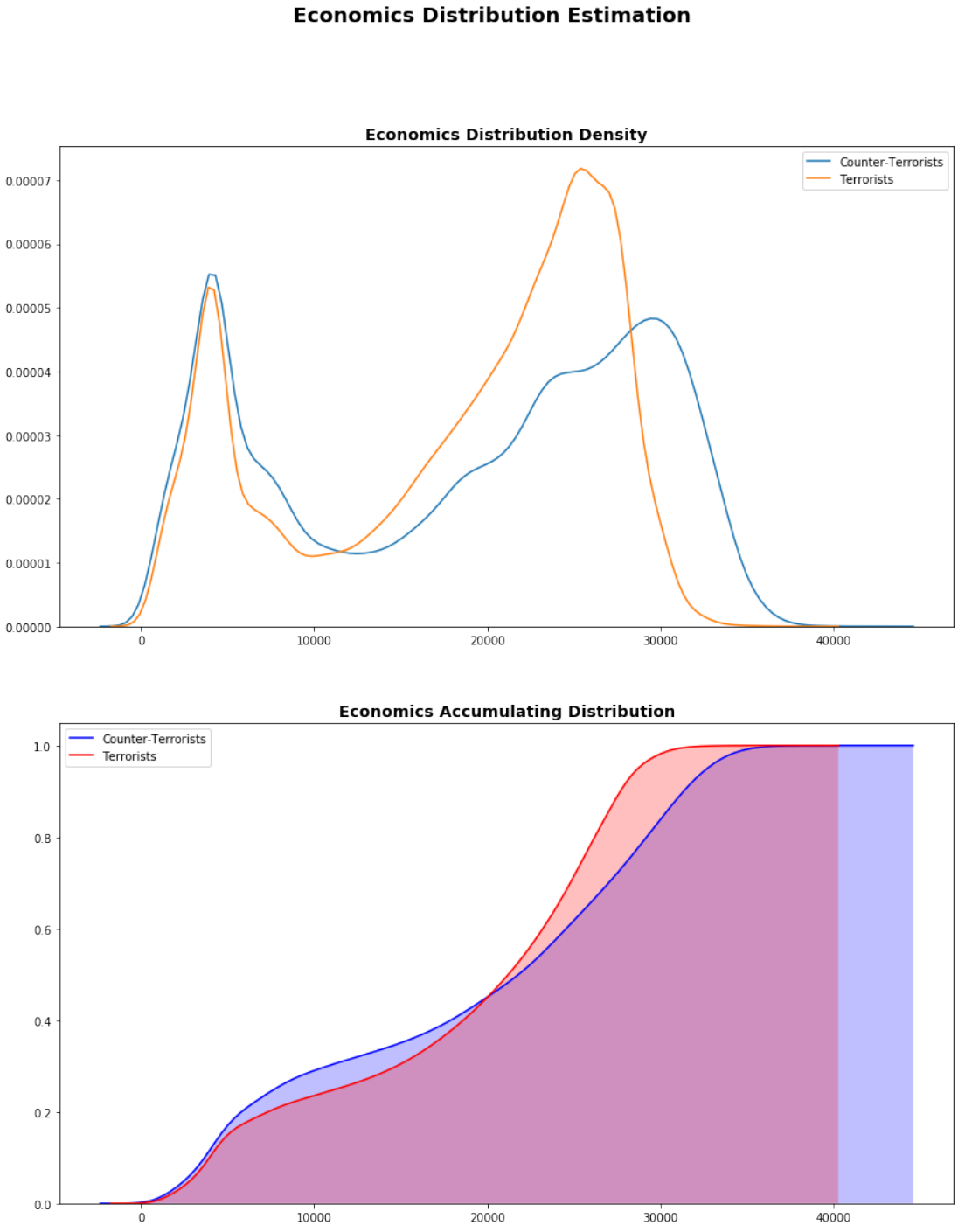


Fig. 10-14 Economic Distribution Curves

As the Fig. 10-15 shows, The economy of CT is higher than that of T, and it is also more volatile and easier to ECO, which is designed by the game mechanics.

The CT economy is slightly higher than the T economy, whether it is the median or the average, which is related to the game mechanics too. And it is clear that the CT economy is more volatile.

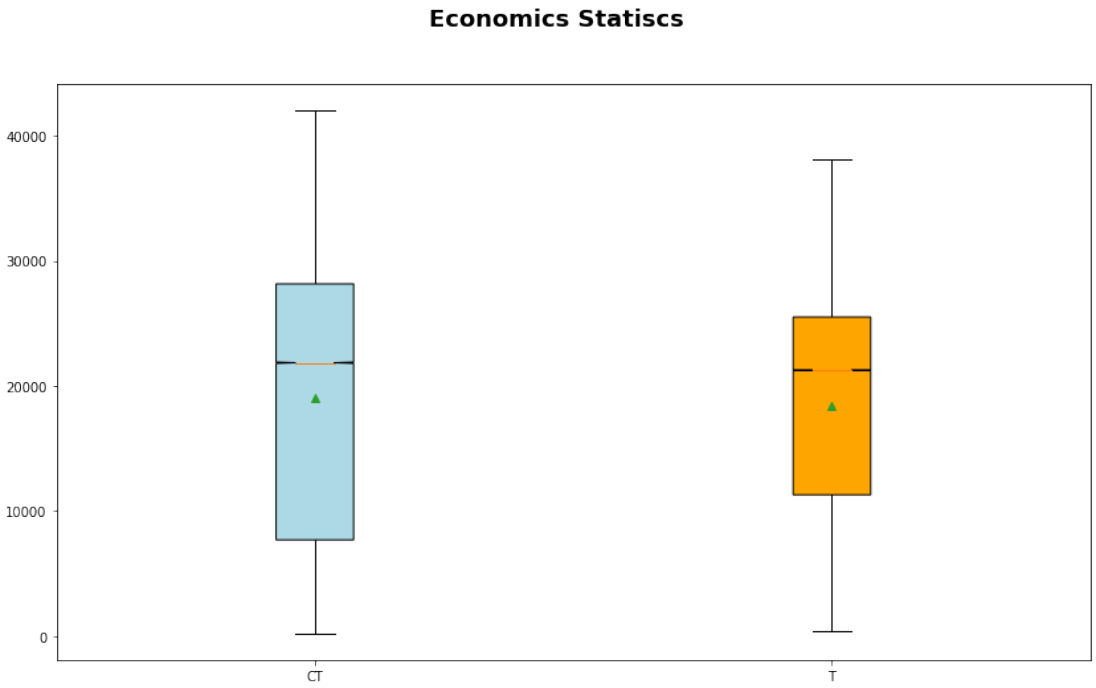


Fig. 10-15 Economic Box Plot

Impacts

The mechanism of economics are important to several affairs, such as the winning rate and grenade using.

Fig. 10-16 is the estimated distributing density curve of the number of winning rounds on different economics.

Most winning rounds distributed at the economics on both sides are nearly the same, and winning rounds by economics advantages are simply more than disadvantages.

And when come economics disadvantages, the winning rounds of T are more than CT. When economics advantages, T usually need more positive economics difference to win the round.

To some extent, it reflects the positive correlation between economy and win rate on T & CT sides.

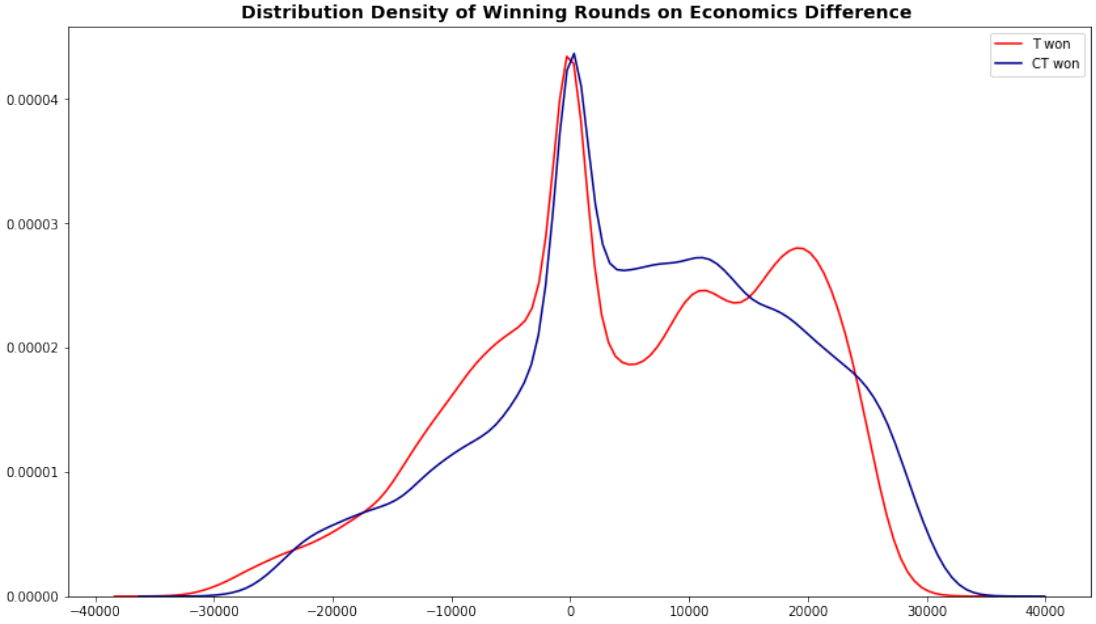


Fig. 10-16 Winning Rounds on Economics Difference

As for the grenade using related to economics, here we post the scatterplot of the number of grenades used in a same round and the economics, as well as the estimated distributing density curve of the number of grenades used on different economics on both T & CT side.

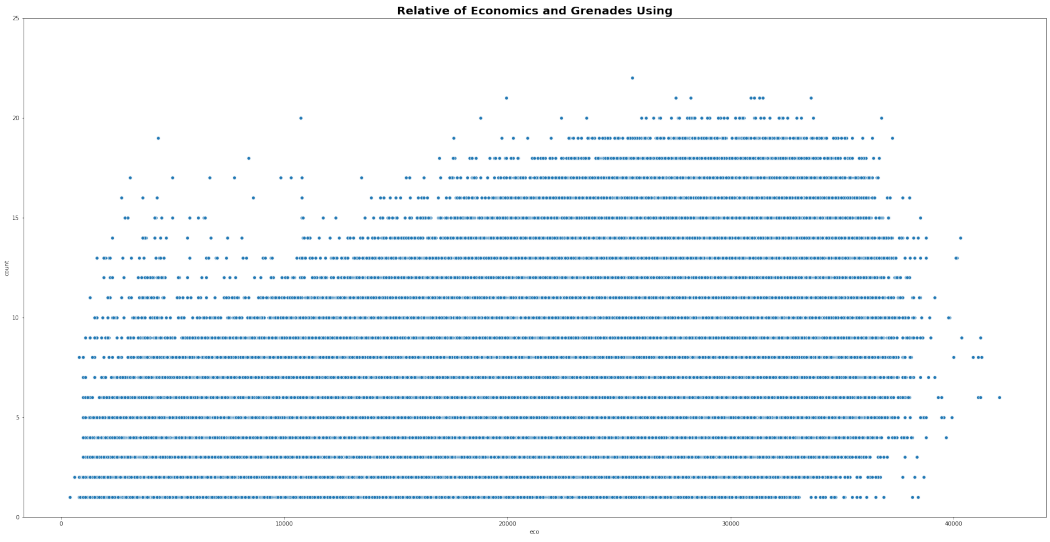


Fig. 10-17 Scatterplot of grenades using and total equipment value

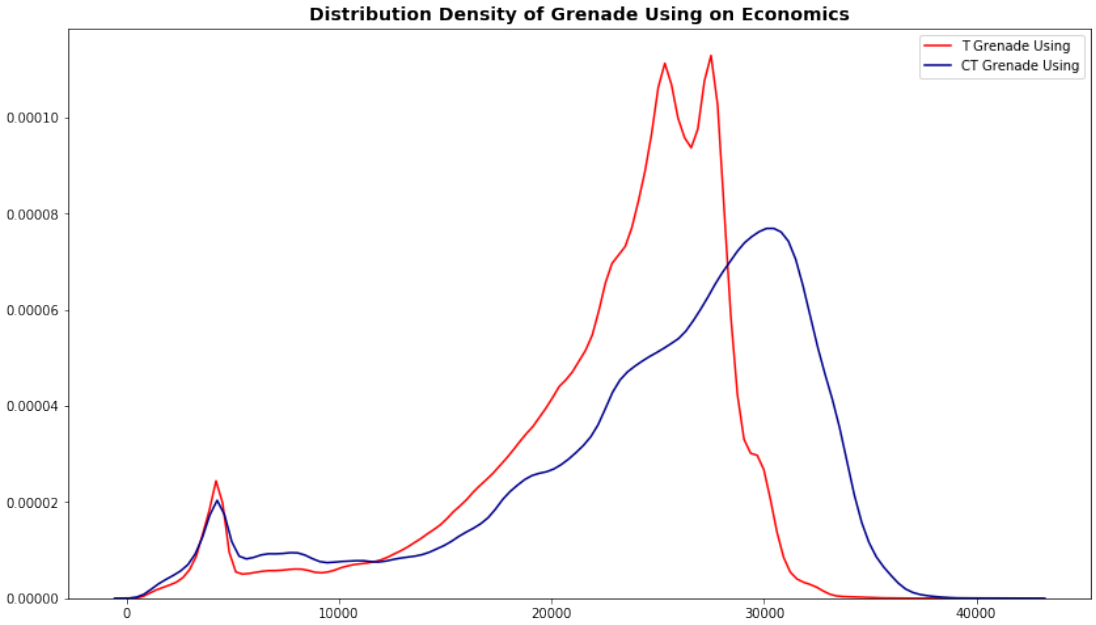


Fig. 10-18 Distributing density of grenade using records on equipment value

From the scatterplot (Fig. 10-17) we can see the top edge of the data seem a growing curve, grenades using are more frequently when economics are sufficient.

Same as Fig. 10-18, we are clearly to see more economics lead to more grenade using, and CT need more economics to get the peak than T does.

When economics are low of both T & CT side, the grenade using have a same peak.

10.3 Grenade Analysis

Grenades are auxiliary props in CS:GO games, and there are 6 kind of grenades in this game. The total using records of these grenades are shown as Fig. 10-19.

Flash bombs are used the most, followed by smoke bombs, grenades, CT incendiaries, T Molotov, and decoys.

The CT use incendiary is 30% more frequent than the T use Molotov, but the economy of the police incendiaries is higher than that of the T Molotov.

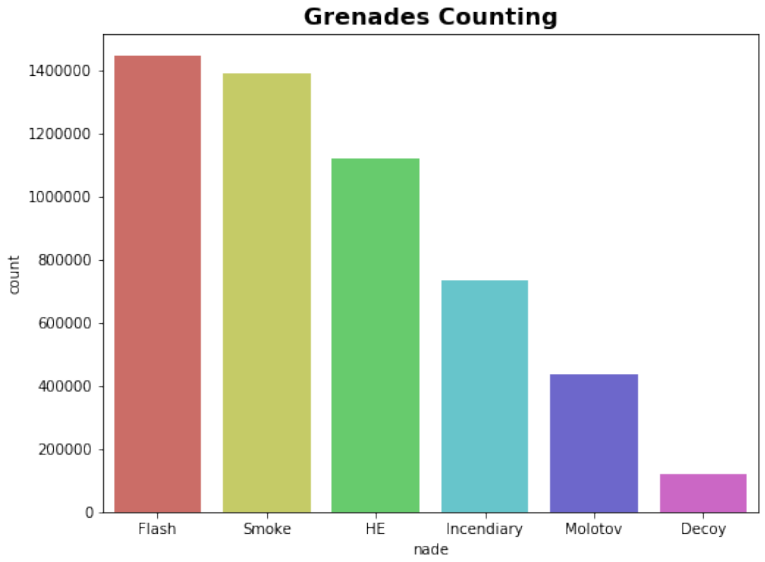


Fig. 10-19 Grenade Using Records Count

Damages

Players often think about the High-Explosive grenades are hardly to hurt enemies. By analyzing how many players could be hurt by a HE grenade can we deduce whether it is useful to use HE grenades in competitive game plays.

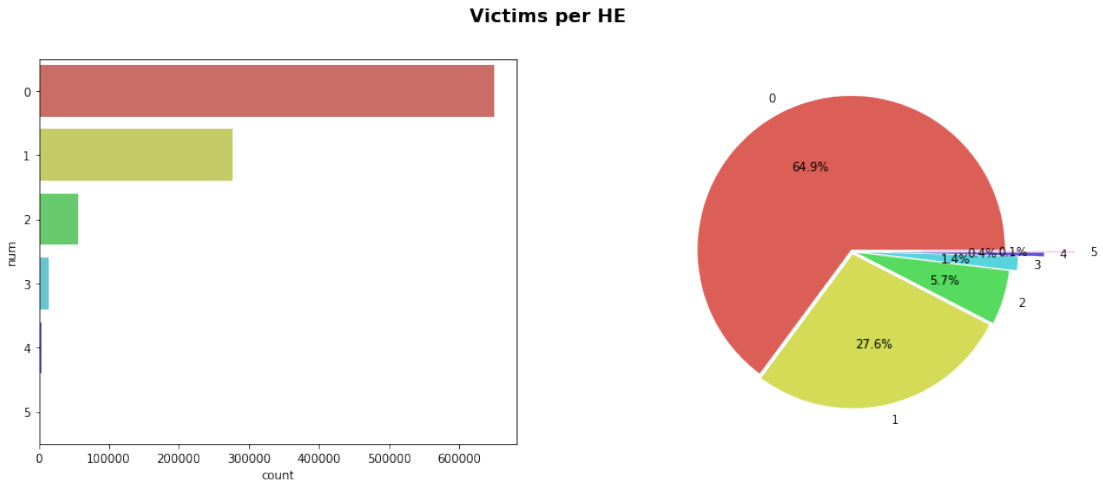


Fig. 10-20 Victims per one HE grenade

It can be seen that most of the grenades did not blow up, about 40% of them injured 1 person, and the proportion of people who blew up more than 2 people was very small.

This shows that especially in the professional league, the role of HE grenades is more to repel the enemy, or to attack the enemy who is alone. HE grenades might are really not very necessary in daily match rounds.

Then is the damages of all grenades, Fig. 10-21 shows the estimated distributing density curves of the HP damage and Armor damages by HE grenades, the HP damages by the fire grenades on T and CT side.

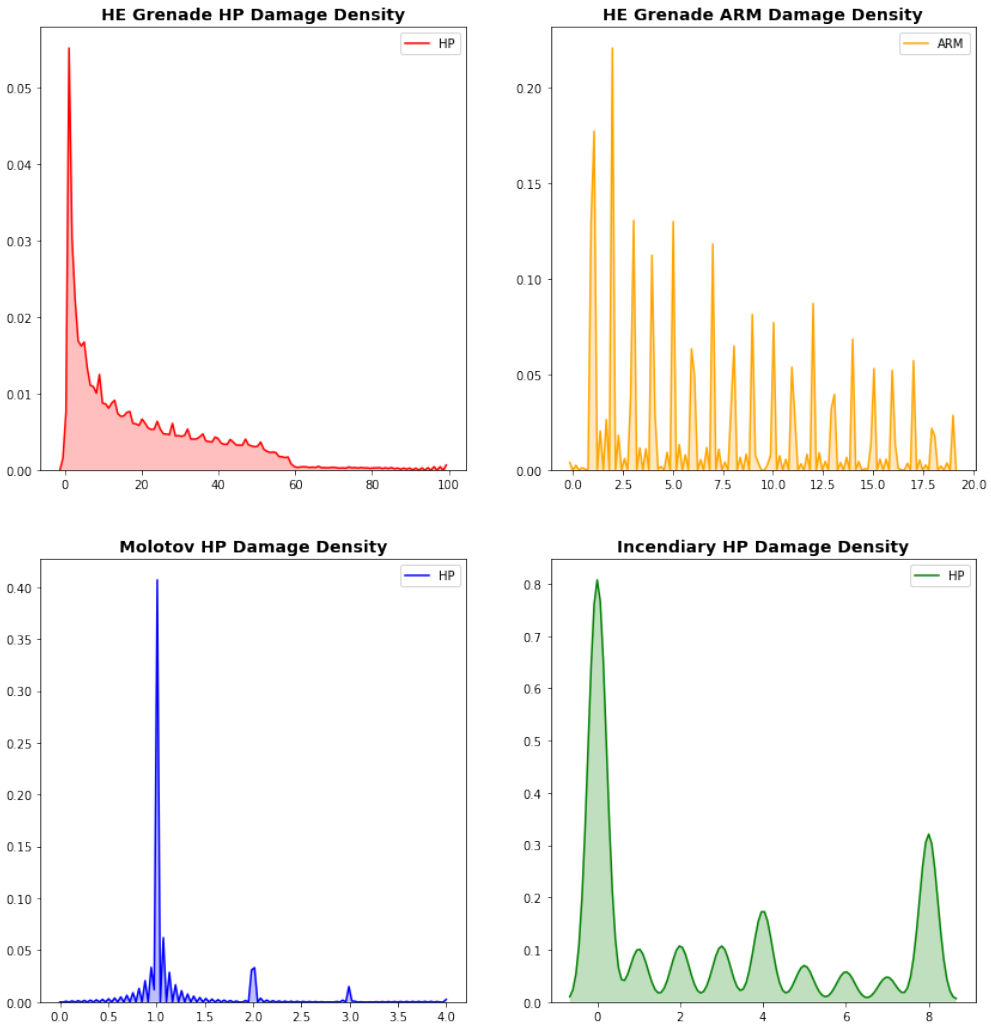


Fig. 10-21 Damage Density per Grenade

Most High-Explosive grenades cost zero HP damage, and the ARM damages remain below 20. The Molotov has mostly damages at 1, that because it can cause 1 damage at a time, and max damage is 4, but incendiaries of CT has damages records from 0 to 8.

Locations

Here we post the heatmaps of the grenade landing locations. Mainly analyzing the Dust2, Mirage and Inferno maps.

Dust2

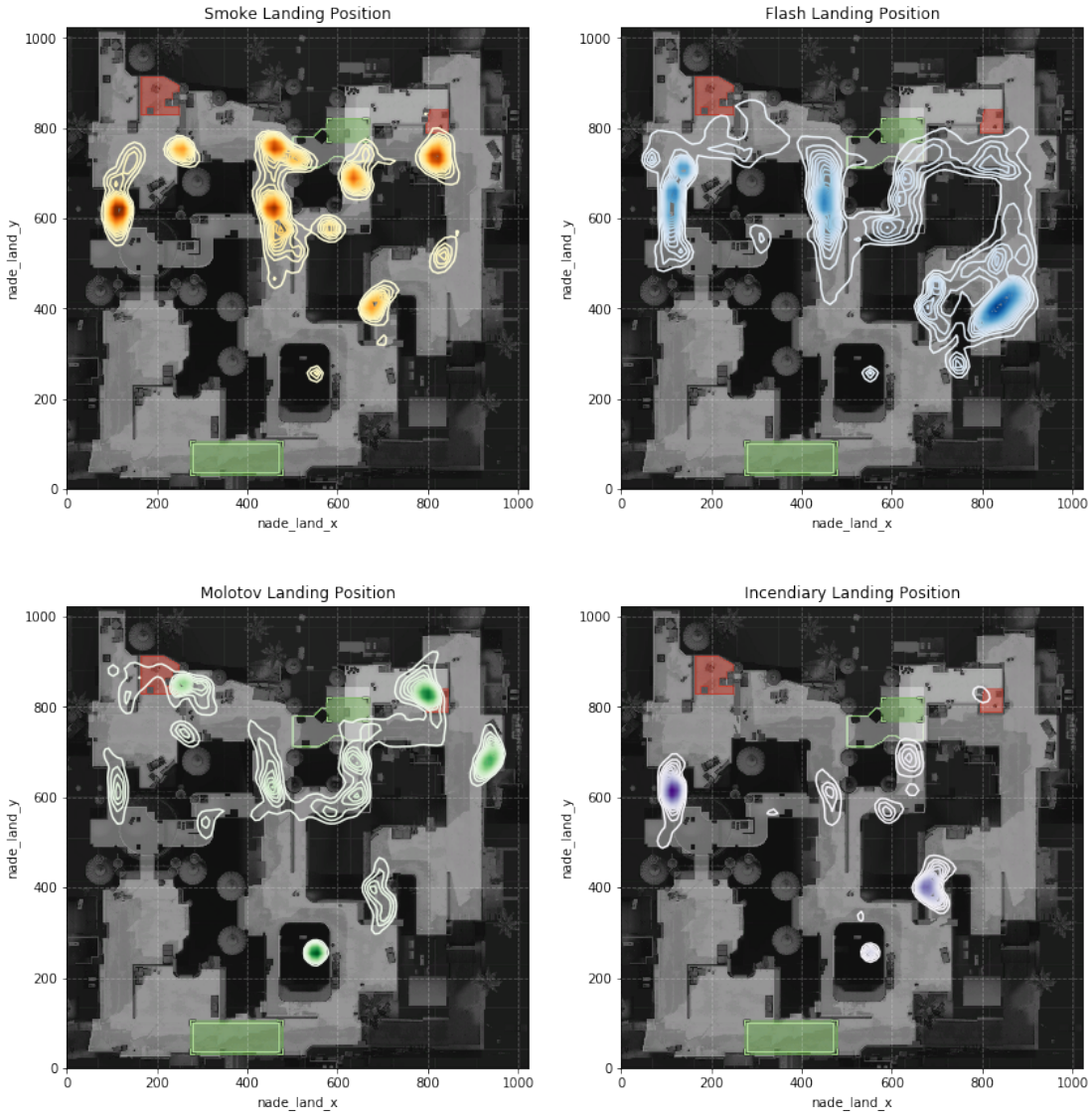


Fig. 10-22 Grenade locations on `de_dust2`

The landing places are basically fixed to several frequent position.

On the attacking side, the smoke often landed to the CT near A site, cover on the short to A, and the gate of B side. The Flash mostly in the A long which are the cover of both sides. And the Molotov of T mostly fire on the car on the A site, break of B, and the inside of A.

On the defending side, smokes are landed to the tunnel of B, the central gate. The incendiaries for CT are mostly landed on the tunnel of B, the gate of A, short to A and the central gate.

Mirage

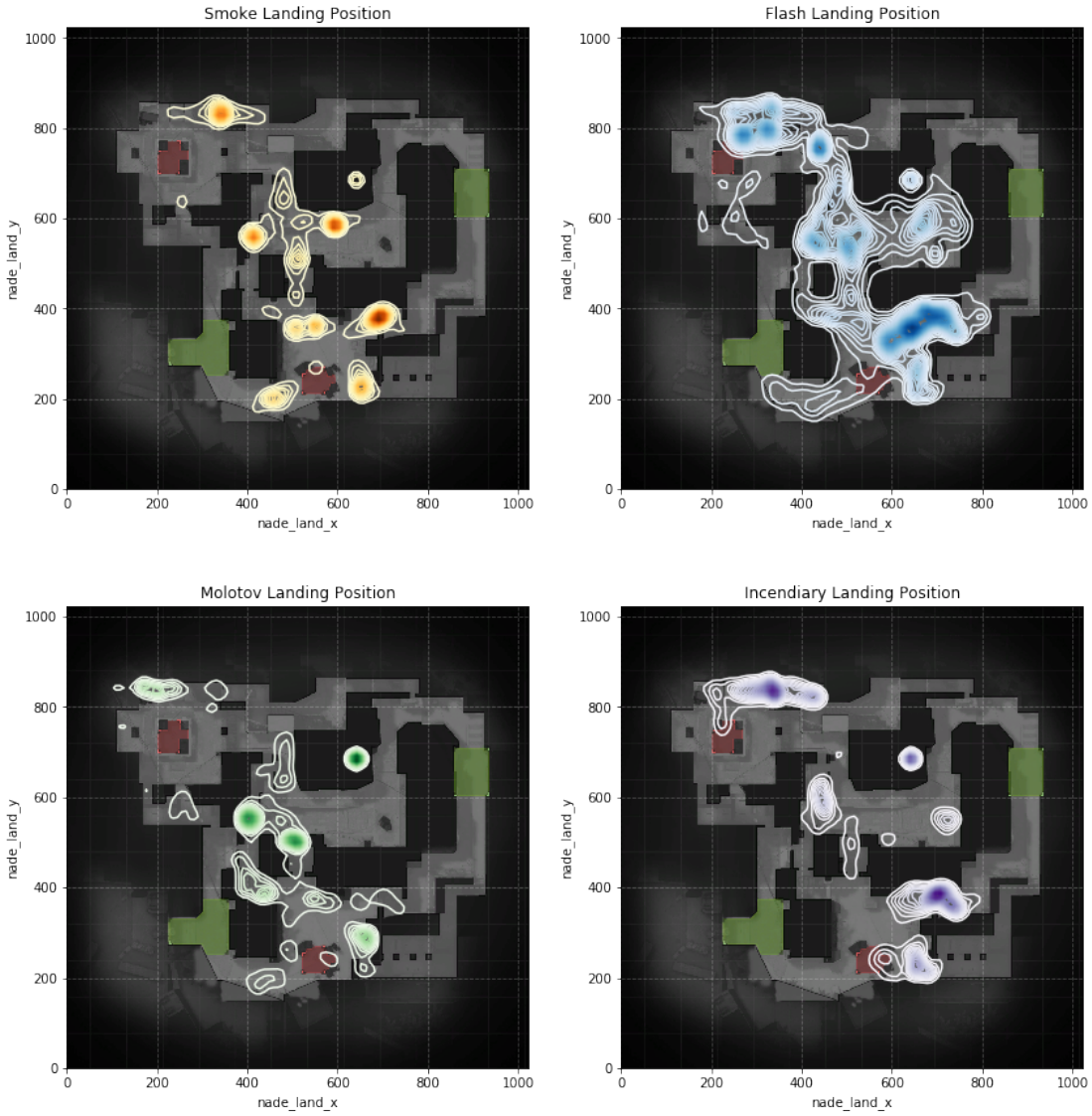


Fig. 10-23 Grenade locations on `de_mirage`

As to Mirage, the defending side mainly land smoke on the second floor of A, A1, entries of center road, and second floor of B site. And T often land smoke on CT of A, arch of B, supermarket at B to prevent CT from returning. On jungle and outside of arch mainly for covering of attacking.

The Flash: most flashes exploded on the area near A1 for clearing and reverse

clearing, then is B2 and short of B. Jungle, outside of arch and entry of center road are the rest frequent locations of flash landed.

T mostly throw fire grenade on jungle and arch, then is the car of B, supermarket, below A2, and up of arch.

CT mostly throw fire on A1 and B2, then is the jungle, A2 and sniper pos at central road.

Inferno

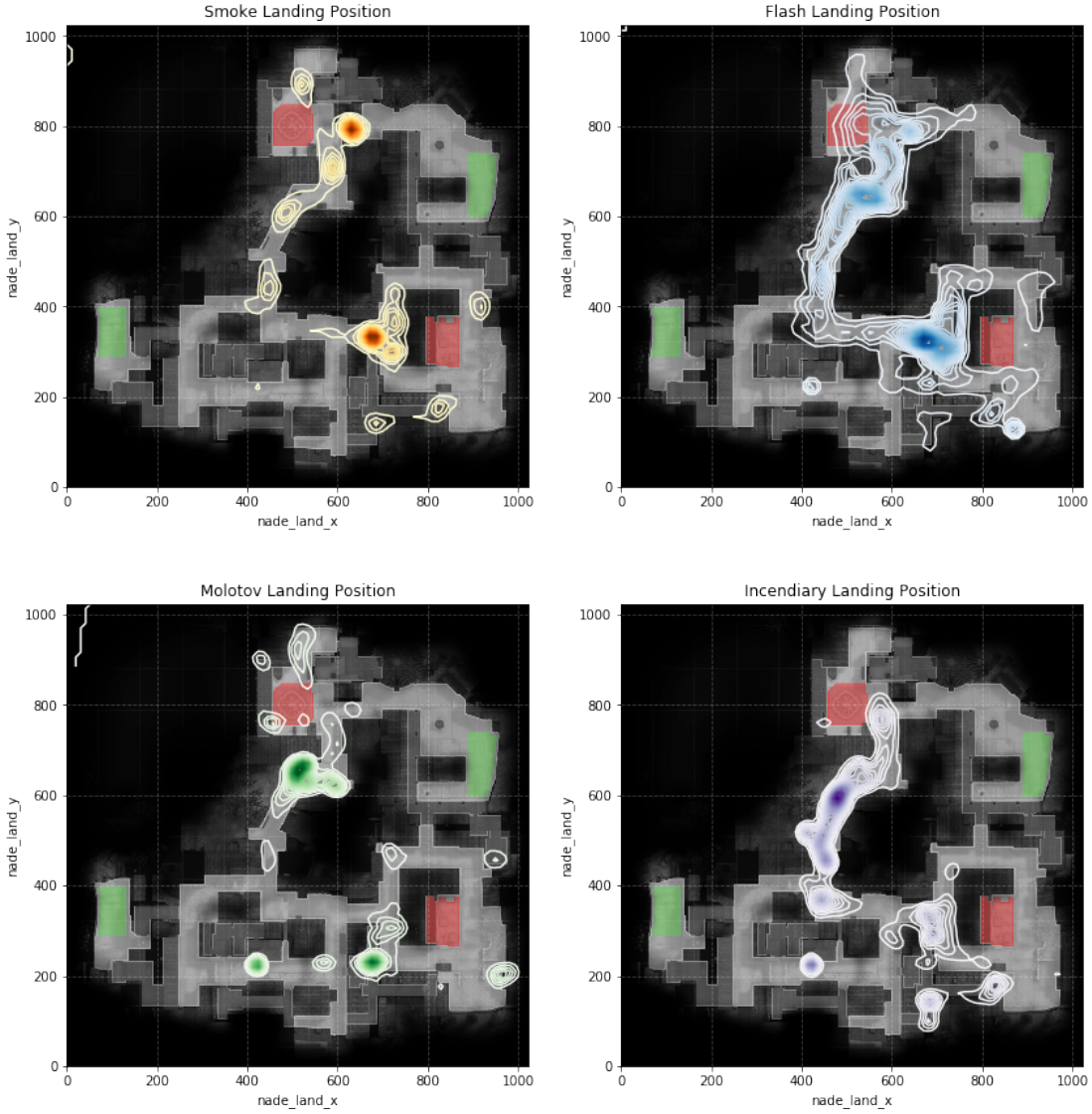


Fig. 10-24 Grenade locations on `de_inferno`

Smoke are mostly landed on A long and the CT to the B site. Then the banana, coffin position, library, A2 etc.

The Flash: most flashes exploded on the mid road to A long and banana to B site.

T mostly throw fire grenade on sniper position in banana, jungle of A, boxes at B, main hole at A and the church at B.

CT mostly throw fire on the central of banana, mid road near A long, stairs to A, and A2.

10.4 Map Analysis

10.4.1 Heat Maps of attacking positions

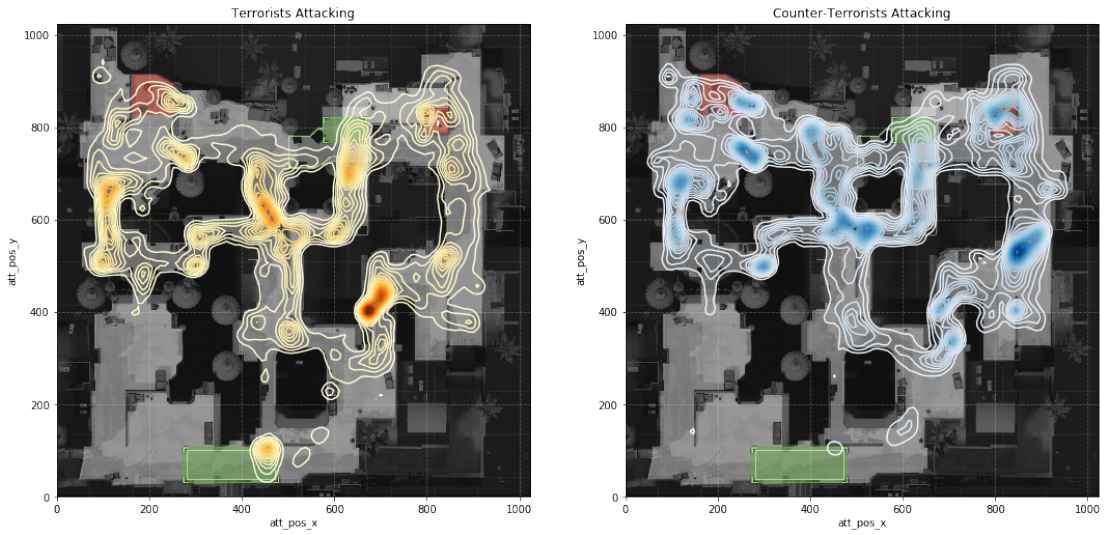
The heatmaps of frequent attacking positions on different maps are shown as Fig. 10-25, orange lines stand for attack position of T, and blue lines stand for CT. The depth of the color indicates the frequency of attack points.

In Dust2, the attacking and defending position are mainly focused on gate of A long, the midway and tunnel to B site, which are keys to manage the game.

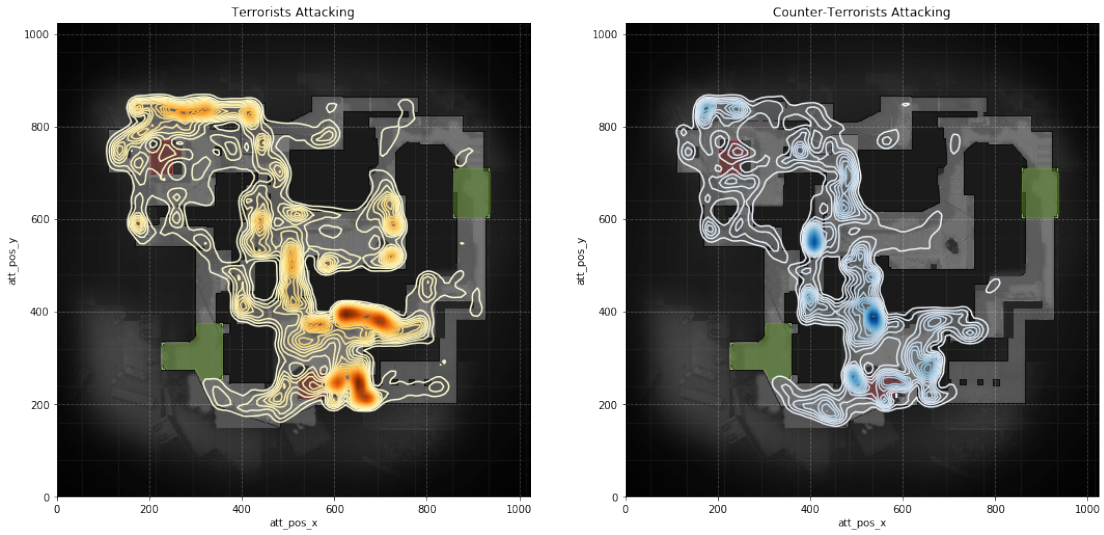
In Mirage, T mainly attack A2, A1, B2 and mid-way near arch, CT mainly defend at stairs, long box at A, jungle, car at B and B short near mid-way.

In Inferno, T literally attack banana most frequently, and then is A long and A2. And CT often defend at sniper position at banana near B, corner at A long, the coffin and CT to the B site. (Fig. 10-25(a), Fig. 10-25(b) and Fig. 10-25(c))

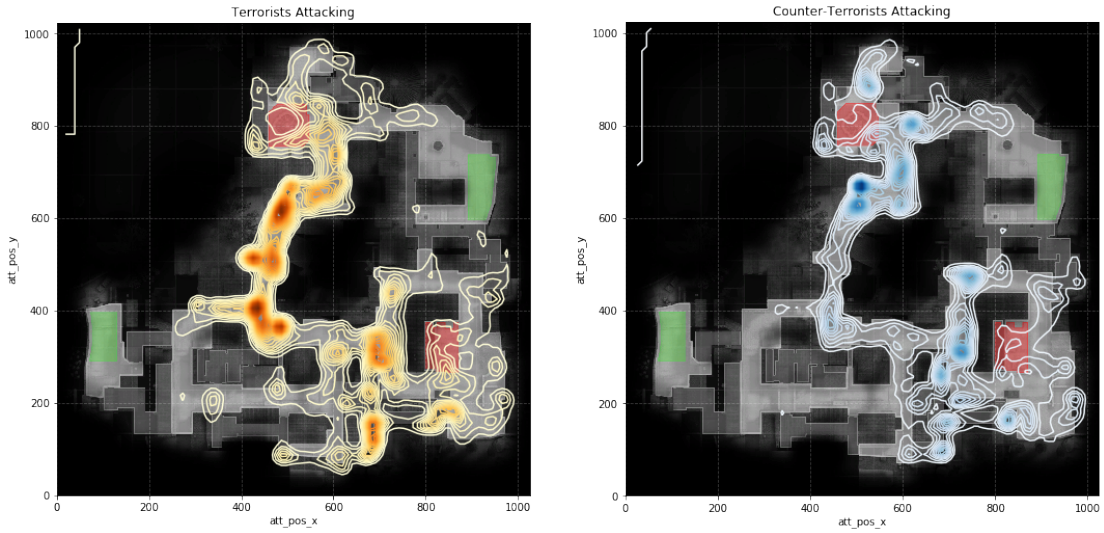
The frequent attacking locations of other maps can be found below, like Train (Fig. 10-25(d)), Cache (Fig. 10-25(e)) and Overpass (Fig. 10-25(f)).



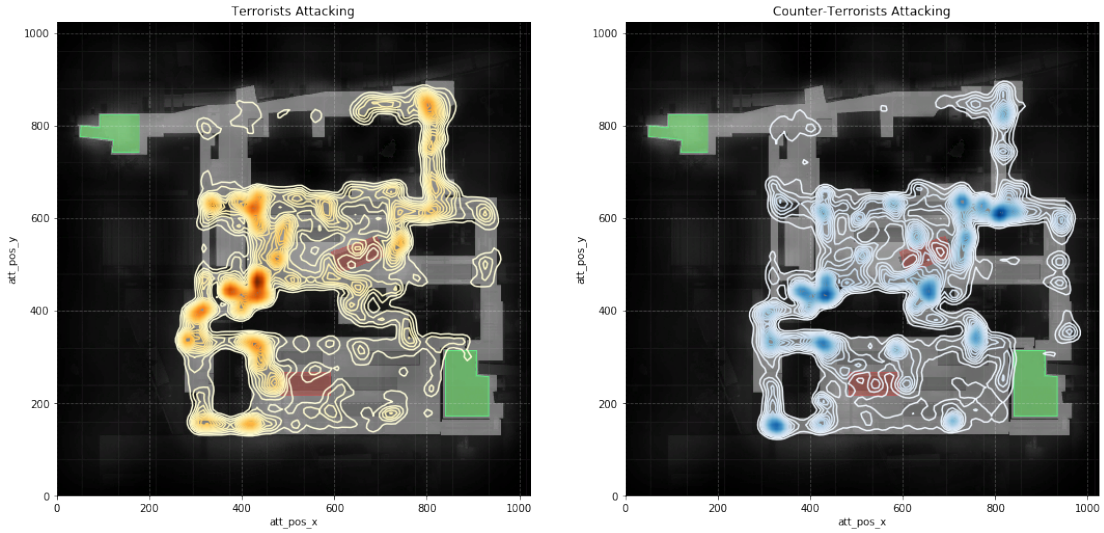
(a) Dust2



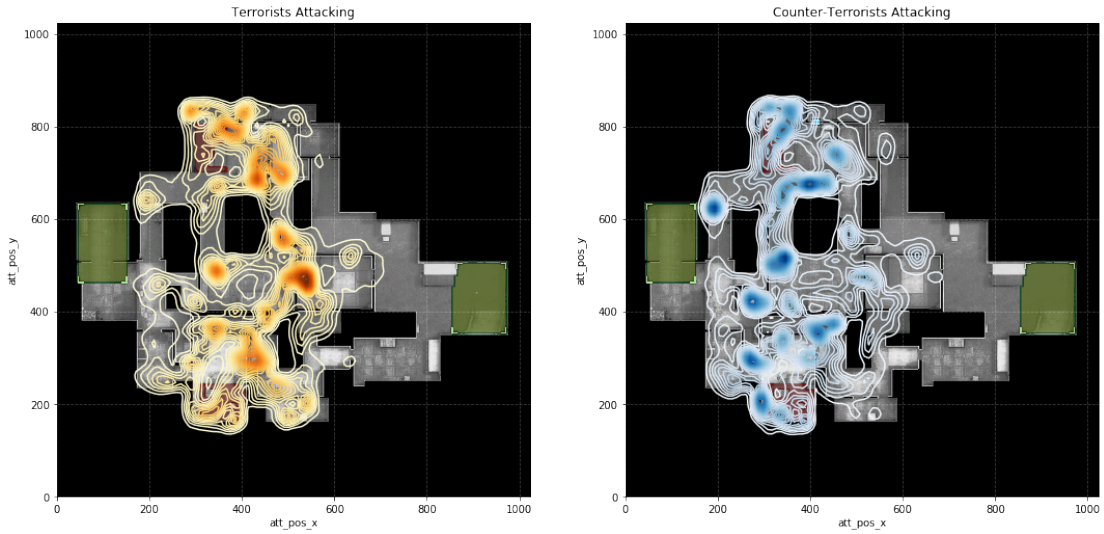
(b) Mirage



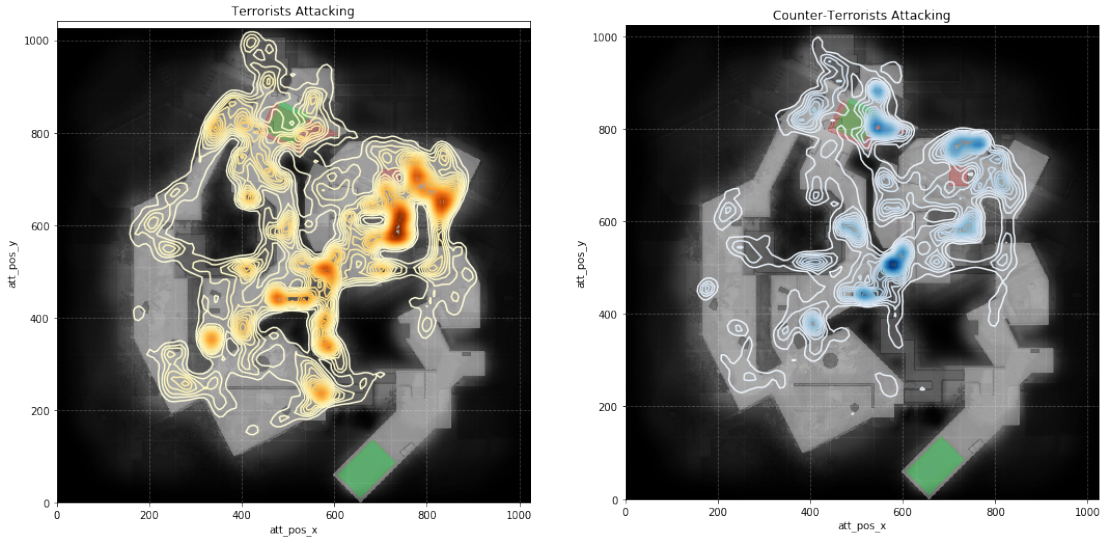
(c) Inferno



(d) Train



(e) Cache



(f) Overpass

Fig. 10-25 Frequency Heatmaps of Duty Maps

10.4.2 Winning Chances

The winning percentage of rounds in different maps suggest the balance of different maps. The overall winning chances are shown as Fig. 10-26, most map are relatively balanced, Nuke is most tend to CT, and Dus2 is most tend to T side, Mirage seems is the most balanced map.

Planting bomb is definitely helpful with winning, according to the winning rounds by bomb planting (Fig. 10-27), the winning rate is 6 times higher after planting the bomb, and for the CT side, winning chance just makes less than 20% higher when Terrorists don't have chances to plant bomb.

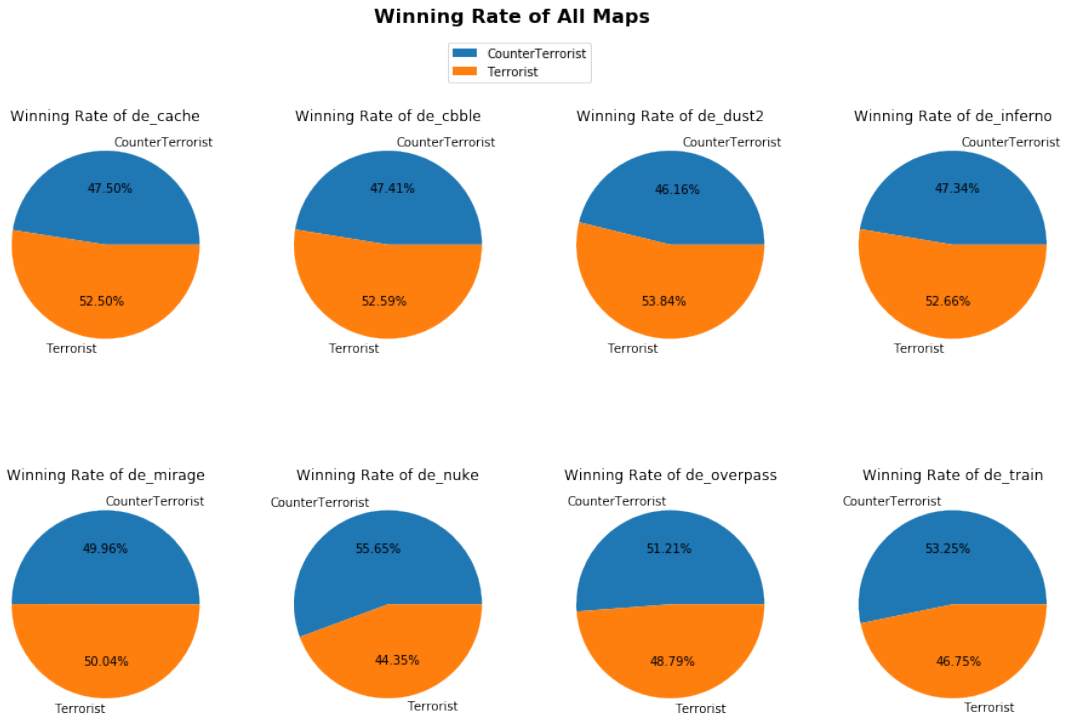


Fig. 10-26 Overall map winning chances

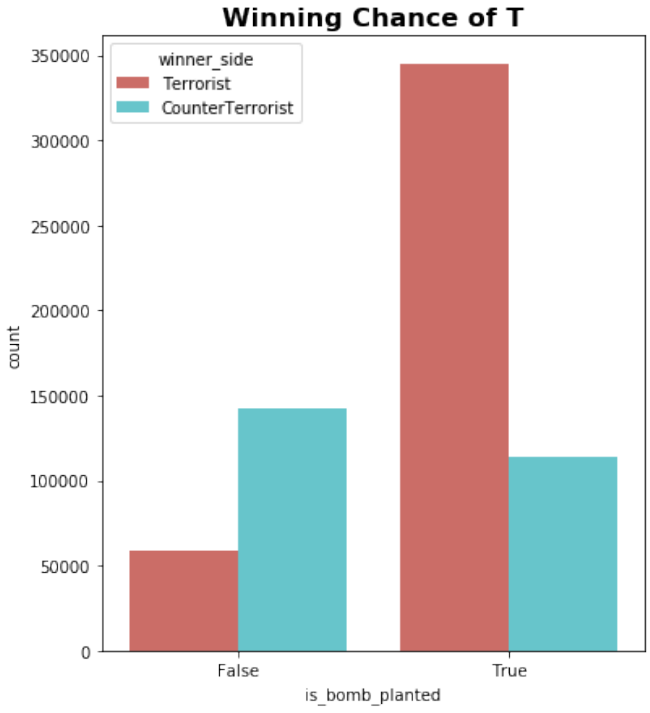


Fig. 10-27 Winning Chances by Bomb Planting Situation

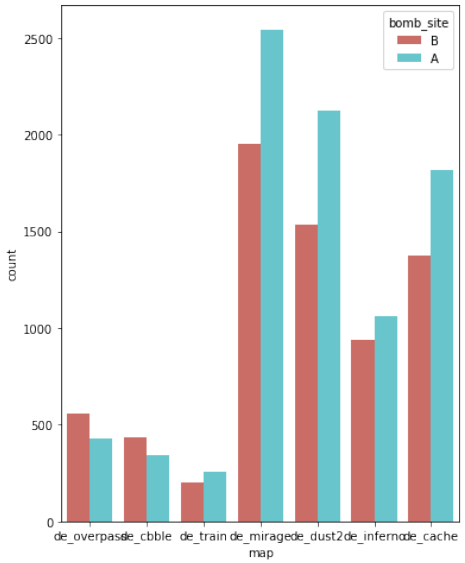
Next, we analyze the prosperities in common competitive game plays.

The winning rounds of T side by planting bomb at different site are shown as Fig. 10-28.

Table 10-5 Winning rounds

bomb_site map	count A	B
de_cache	1819	1375
de_cbble	346	434
de_dust2	2123	1537
de_inferno	1065	938
de_mirage	2544	1955
de_overpass	428	557
de_train	257	204

Fig. 10-28 Bomb Planting Counts



In Mirage, Dust2 and Cache, planting at bomb site A a significant increase in win rate, all above 20%. In Train and Inferno, planttng at A give a little advantage in winning. And panting at B site have advantages in winning the round in the maps of Overpass and Cbble.

The winning chances of both T and CT side when planted to different site in different maps are shown as Table 10-6.

And the advantage of the number of people can improve the winning rate, the winning percentage of T side when have `XvX` player advantages or disadvantages after planting the bomb at different bomb site in different maps are shown as Table 10-7.

There nearly no winning round when have 3 players less, and have 20% winning chance when have 2 players less, average at 45% when have 1 plyer less. There are average at 70% when both T & CT have equal number of alive players. When having 1 more player than CT side, the average winning chance goes to 90%, average at 95% when 2, and nearly 100% when having 3 or 4 players advance.

Table 10-6 Winning chances on bomb planted at different site

	map	count de _cache	de _cbble	de _dust2	de _in-ferno	de _mi-rage	de _over-pass	de _train
site	win							
A	CT	24.079 164	22.543 353	21.479 039	19.436 620	22.366 352	22.897 196	19.844 358
	T	75.920 836	77.456 647	78.520 961	80.563 380	77.633 648	77.102 804	80.155 642
B	CT	24.218 182	17.972 350	24.137 931	17.803 838	26.496 164	25.493 716	44.117 647
	T	75.781 818	82.027 650	75.862 069	82.196 162	73.503 836	74.506 284	55.882 353

Table 10-7 Winning chances when have player differences

	map	count de_cache	de_cbble	de_dust2	de_inferno	de_mirage	de_overpass	de_train
XvX	bomb_site							
-3	A	16.666667	0.000000	18.750000	10.000000	4.545455	0.000000	0.000000
	B	6.250000	33.333333	0.000000	0.000000	16.666667	0.000000	0.000000
-2	A	17.567568	22.222222	27.536232	34.146341	27.027027	23.076923	11.111111
	B	17.857143	16.666667	20.000000	44.186047	27.480916	21.428571	25.000000
-1	A	44.696970	53.061224	47.826087	57.534247	41.284404	46.296296	36.000000
	B	45.495495	45.454545	49.600000	50.370370	46.604938	40.000000	31.372549
0	A	72.093023	68.539326	72.359155	75.666667	71.735791	65.178571	70.149254
	B	73.958333	72.641509	71.794872	82.671480	74.368231	71.724138	59.649123
1	A	88.888889	89.108911	89.126853	90.460526	90.558511	91.489362	93.750000
	B	90.463215	89.510490	88.805970	93.560606	88.653846	86.585366	75.000000
2	A	96.099291	98.333333	96.569921	97.905759	96.995708	98.360656	100.000000
	B	95.141700	97.727273	96.825397	98.101266	97.967480	96.739130	87.500000
3	A	100.000000	100.000000	100.000000	100.000000	99.367089	100.000000	100.000000
	B	98.214286	100.000000	100.000000	100.000000	98.437500	100.000000	100.000000
4	A	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	0.000000
	B	100.000000	100.000000	100.000000	100.000000	100.000000	0.000000	0.000000

10.4.3 Defending Positions

There is a question that if it's better to play post-plants in-site our out-of-site during a one-man up/down or equal situation. In-site has the advantage of peeking when the CTs are clearing outer-site spots but the con of being in a spot where you are forced to duel the CTs. Outer-site has pro of baiting shots and playing time but having to peek into the CT when he is defusing. Let's isolate for only 1v1, 2v1, 1v2, 2v2 situations and look at average ADR differential when you play inner site or outer site.

By analyzing that, we use the ADR when inside or outside the bombsite for defending, as shows in Table 10-8.

Except the A site in Inferno, defending outside of bombsite will have a higher ADR, defending inside the bombsite usually have a negative ADR, which means being killed. So it will be wise to defending out of the bombsite after planted a bomb.

Table 10-8 ADR of different defending positions

att_callout map	A inner	B inner	N/A
de_cache	4.053875	8.481475	5.486609
de_cbble	-33.550420	6.402715	12.267927
de_dust2	-3.421429	5.076923	5.599861
de_inferno	11.939250	4.642857	8.242179
de_mirage	-1.317458	5.058519	7.965681
de_overpass	1.577273	-8.491991	4.336491
de_train	-16.496970	-7.260684	9.366601

Chapter XI

Conclusion

In this report, we mainly analyzed four part of the game mechanism and players behavior preferences, which are weapon system prosperities, economics comparison and the impact on other factor, grenade using and its prosperities and finally the analysis of mechanism associated with locations on map.

Players tend to buy weapons that can almost kill in one hit, which also means good penetration, the most popular weapon in the CS:GO game is AK-47, the most popular pistol is Desert Eagle, these two weapon still caused most kills even in economic weak rounds.

The economics system is a great design to balance the difference of both sides due to the different characteristics of weapons. CT have to buy weapons more expensively, while T have more cheaper weapons with higher power. Higher economics leads to more winning chances, and more grenade using.

Grenades take a significant part in the game, reasonable and effective use of grenades will increase the probability of mastering the game and winning. High-Explosive grenades takes a mediocre place in game. Most locations of smoke, flash and fire grenades on both T & CT side are frequently fixed. The intent and method could be deduced and learned from grenade landing positions on different maps.

The duty maps are relatively balanced in winning chances. And the attacking and defending pattern could be seen from the heatmaps of frequent attacking positions. Planting bombs have a great advantages in winning chances, and it is better to hold positions outside the bombsite to stop CT's returning.

More details can be found in four main chapters and the Results and Analysis chapter.

References

[1] 反恐精英：全球攻势多维度战斗数据分析（机制类分析）. https://zhuanlan.zhihu.com/p/66158085.

[2] Big data. Wikipedia. https://en.wikipedia.org/wiki/Big_data.

[3] Counter-Strike: Global Offensive. Gameplay. https://en.wikipedia.org/wiki/Counter-Strike:_Global_Offensive#Gameplay.

[4] CS:GO Competitive Matchmaking Data. https://www.kaggle.com/datasets/skihikingkevin/csgo-matchmaking-damage.

[5] Apache Hadoop. https://hadoop.apache.org/.

[6] Pistol Round Analyses. https://www.kaggle.com/code/skihikingkevin/pistol-round-analyses.