# *PUBG*
# *Finish Placement Prediction*
# *Final Report*

**과목명** : Data Science
**담당교수**: 강재우

**제출 날짜**: 2019.06.13

**팀원**
2015410049 이강은
2015410051 김은주
2016320120 정소영
2016320204 박민서
2016320171 정예주


고려대학교
KOREA UNIVERSITY

# 목차

# I. 서론

## 1. PUBG

PUBG는 Player's Unknown Battleground의 약자로서, 1인칭과 3인칭을 모두 제공하는 배틀로얄 형식의 총 게임이다. 최대 100명의 플레이어는 지정된 맵에 낙하하여 총기나 회복 아이템 등을 파밍하여 서로를 죽인다. 주기적으로 데미지를 입히는 자기장을 피해 플레이어들은 지속적으로 이동을 하며, 최후의 한 팀이 남았을 때 그 플레이어(또는 플레이어가 속한 팀)의 우승으로 게임은 종료된다.

### 1-1. 게임의 종류

#### (1) 인칭

1인칭과 3인칭, 2종류를 제공한다. 3인칭은 1인칭으로는 보이지 않는 것을 볼 수 있게 해주며, 이는 게임 플레이에 장단점으로 모두 작용하여 1인칭과는 다른 재미를 부여한다. 현재 한국에서는 3인칭 서비스만을 제공하고 있다.

#### (2) 그룹

솔로(1인), 듀오(2인), 스쿼드(4인), 3종류의 개인 또는 그룹 플레이가 가능하다. 그룹의 경우 최대 인원일 뿐, 그 이하의 인원으로도 게임을 진행할 수 있다. 팀원은 서로를 엄호해주고 소생시켜주고, 아이템을 효율적으로 분배함으로서 팀의 생존율을 올릴 수 있으므로 솔로와는 다른 전략을 가지고 게임을 플레이할 수 있다.

#### (3) 맵

사녹(Sanhok, 16km$^2$), 비켄디(Vikendi, 36km$^2$), 미라마(Mirama, 64km$^2$), 에란겔(Erangel, 64km$^2$) 총 4가지 맵이 있다. 맵에 따라 크기가 다르고 스폰되는 무기나 아이템, 스폰률 등이 달라지기 때문에 각각 다른 전략을 이용하게 된다.

### 1-2. 아이템의 종류

#### (1) 총기

총의 종류는 크게 SG(산탄총), LMG(경기관총), AR(돌격소총), DMR(지정사수소총), SMG(기관단총), SR(저격총), 그리고 권총으로 나눌 수 있다. 각 종류에는 여러가지 총기가 있으며, 각 총기마다 발당 데미지, 탄속, 유효 거리 등이 다르며, 총기에 달 수 있는 파츠(보조품)의 종류도 다르다.

#### (2) 투척 무기

수류탄, 섬광탄, 화염병, 그리고 연막탄은 투척할 수 있는 무기로, 엄폐물 뒤에 숨어있는 상대방에게 데미지를 입히거나 시야를 흐리게 할 수 있어 전략적으로 중요한 역할을 한다.

#### (3) 기타 장착 아이템

프라이팬, 빠루, 마체테 등의 아이템을 장착하여 근접 무기로 사용할 수 있다. 대부분의 경우 총기가 없는 경우에만 쓰인다.

#### (4) 회복 아이템

회복 아이템은 크게 플레이어의 HP를 회복시켜주는 힐 아이템(구급상자, 의료용 키트, 붕대)와 체력을 회복시켜주는 부스트 아이템(에너지 드링크, 진통제, 주사기)가 있다.

**(5) 탈것**

탈것은 지상에서 탈 수 있는 2륜차와 3륜차가 있으며, 물 위에서 이동할 수 있도록 해주는 배 종류가 있다. 맵에 따라 스폰되는 탈 것의 종류가 달라지며, 각 탈 것의 최대 이동속도, 차량 HP, 구동 방식, 크기 등이 다르기 때문에 전략이나 지형에 따라 적절한 차량을 선택해야 한다.

## 2. Problem Definition

PUBG는 인칭, 팀 종류, 맵의 종류 등에 따라 전략이 다르며, 또 각 매치마다의 자기장의 위치, 파밍 상태, 플레이어 본인의 실력 등 수많은 요소들에 따라 승패가 갈린다. 플레이어들은 각기 1등을 하기 위해 다양한 전략을 펼친다. 우리는 이렇게 다양한 전략 속에서 어떤 요소들이 특히 플레이어의 순위에 영향을 미치는지 알아보고, 나아가 인게임 요소들을 분석하여 플레이어의 순위를 예측하고자 한다.

## 3. Data Analysis

우리는 누적된 인게임 데이터를 Kaggle 데이터베이스[1]에서 확보할 수 있었다. 이 데이터세트에는 총 28가지 Feature가 존재하며, 이를 통해 우리는 최종 순위를 예측하고자 한다.

### 3-1. winPlacePerc

winPlacePerc는 본 Problem의 Target Value이며, 0에서 1 사이의 값을 가지는 퍼센트 값이다. 1은 매치에서 1순위를, 0은 마지막 순위를 지칭한다. 한 매치의 인원수가 100명일때 winPlacePerc의 값이 0.04라면 이 플레이어의 순위는 4위라고 볼 수 있는 것이다.

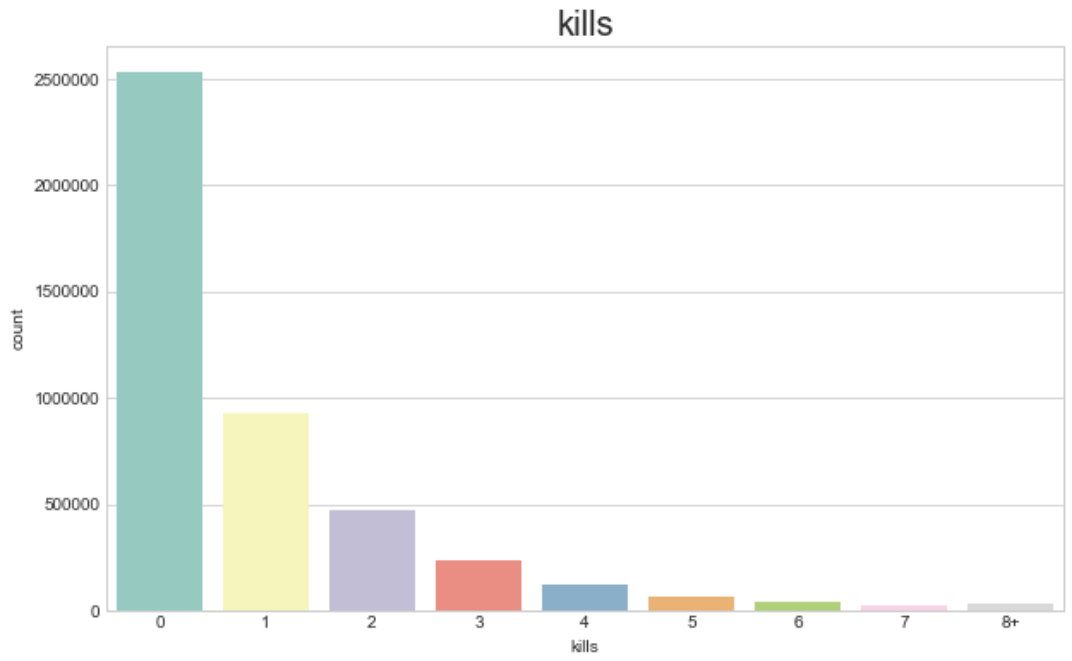
### 3-2. Kills & Damages

본 데이터셋에는 한 매치 동안 플레이어가 낸 킬과 데미지에 관련된 여러 Feature들이 존재한다.

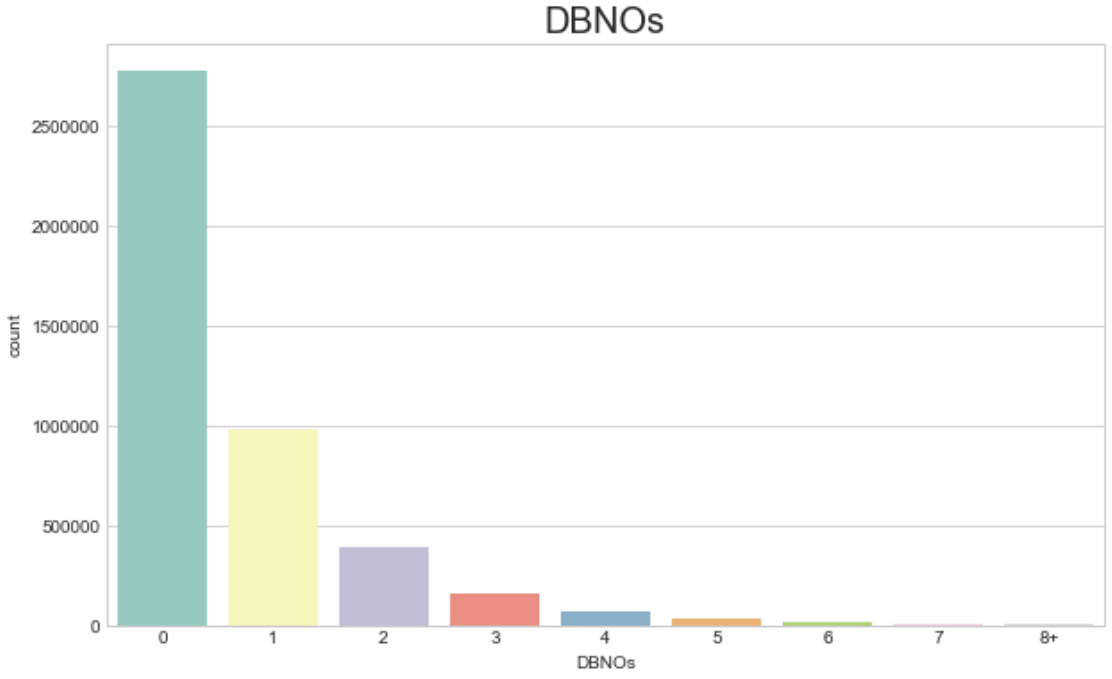
**(1) kills & DBNOs**

- kills: 플레이어가 한 매치 동안 죽인 적의 수를 지칭한다. 적이 죽기 직전 마지막으로 데미지를 넣은 플레이어에게 킬이 돌아간다. 아래 그림과 같이 분포하고 있다.

[1] https://www.kaggle.com/c/pubg-finish-placement-prediction/data에서 제공되는 데이터세트

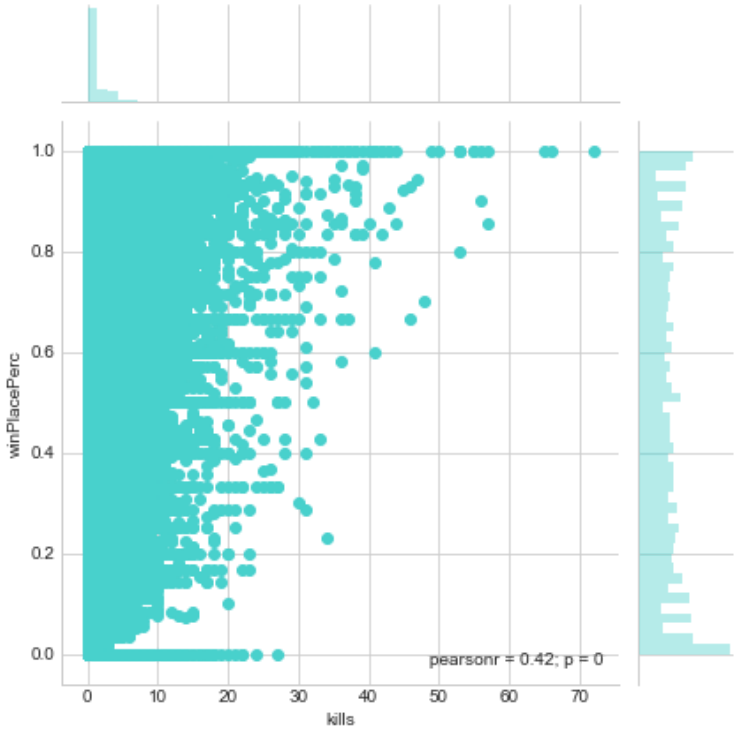



- DBNOs: 플레이어가 한 매치 동안 기절시킨 적의 수를 지칭한다. 기절은 듀오 또는 스쿼드에서 가능하며, 기절한 플레이어는 같은 팀 플레이어가 소생시킬 수 있다. 아래 그림과 같이 분포하고 있다.

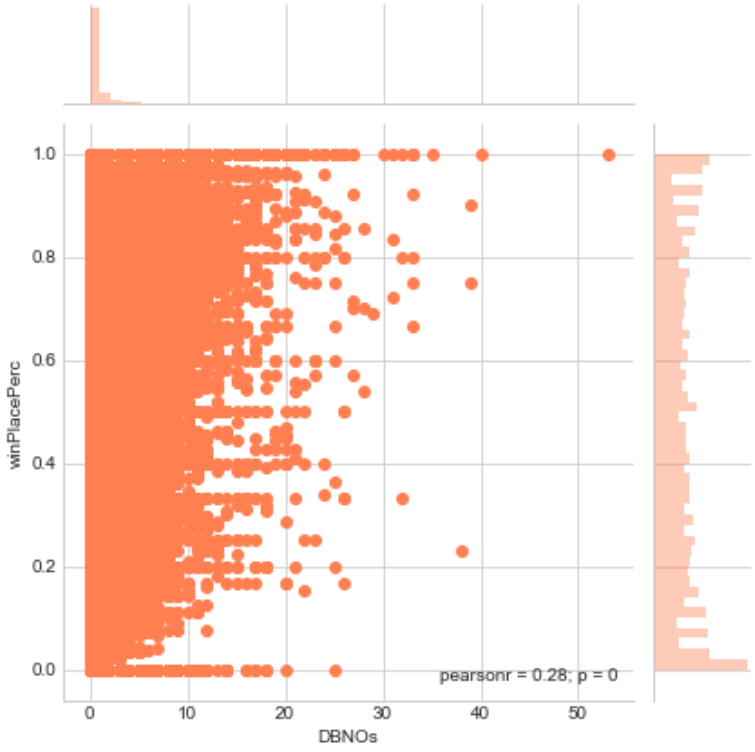



각 Feature의 분포에서 볼 수 있듯, 한 명의 적도 죽이거나 기질시키지 못한 플레이어가 대다수이다. 또한, kills과 DBNOs의 분포가 유사한 것을 확인할 수 있으며, 8킬 또는 9기절 이상은 거의 없는 것을 볼 수 있다.

kills와 DBNOs가 플레이어의 순위에 미치는 영향을 확인하기 위해 두 Feature와 우리의 Target value인 winPlacePerc과의 상관관계를 시각화를 통해 알아볼 수 있다.




먼저 kills의 경우, 0킬을 제외한다면 winPlacePerc와 양의 상관관계를 가지고 있음을 확인할 수 있다. 즉, 더 많은 킬을 한 플레이어 일수록 더 높은 순위에 위치할 확률이 높다는 것이다.

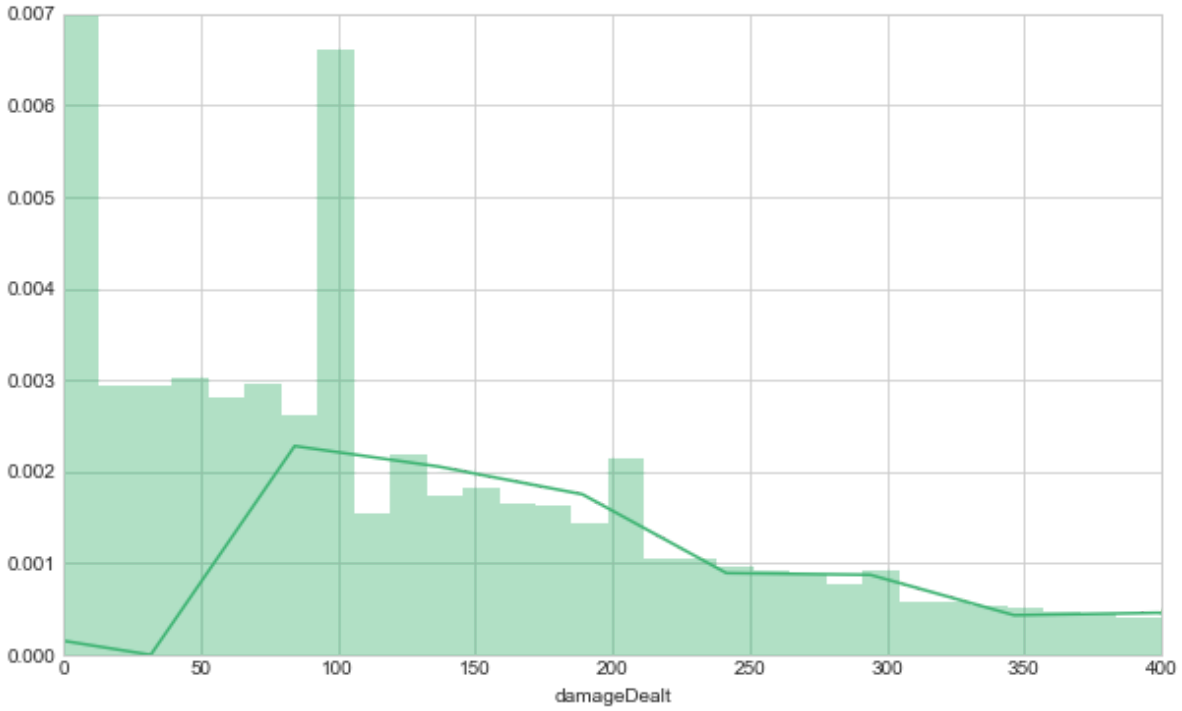



DBNOs의 경우에도 winPlacePerc와 양의 상관관계를 가지고 있는 것으로 볼 수 있다. kills와 마찬가지로 높은 DBNOs를 가지고 있는 플레이어는 높은 순위에 안착할 확률이

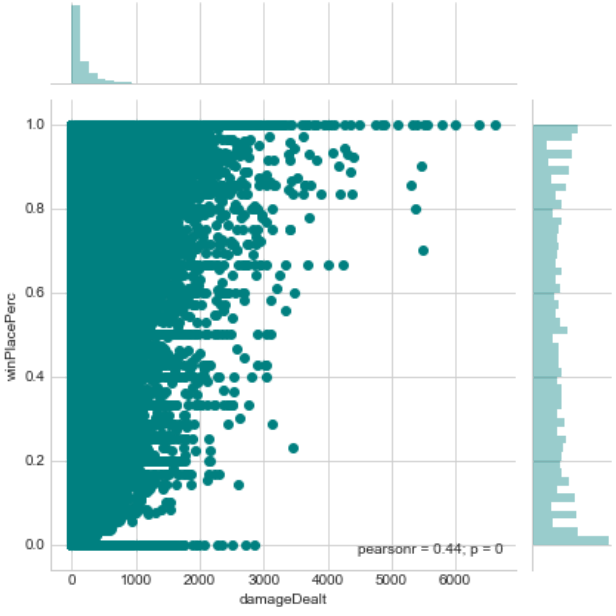
높다고 볼 수 있으며, kills와 DBNOs 모두 순위 예측에 중요한 역할을 할 것이라 예측할 수 있다.

**(2) damageDealt**

- damageDealt: 매치에서 플레이어가 가한 총 데미지량을 측정한다. 아래와 같이 분포한다.



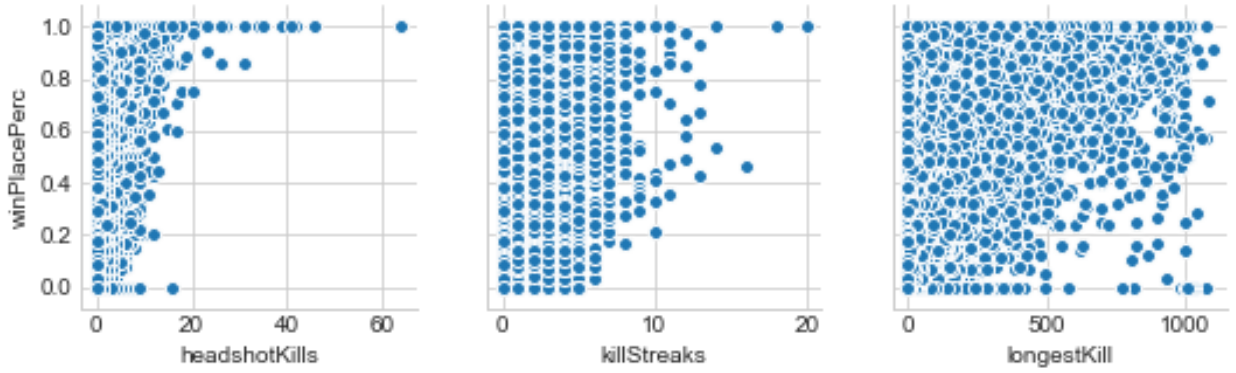

kills나 DBNOs와 마찬가지로 0 데미지를 가한 플레이어가 대다수이며, 점차 감소하는 추세를 보인다. 또한 100, 200, 그리고 300이라는 수치에서 피크를 확인할 수 있는데, 이는 한 플레이어의 HP가 총 100임을 고려했을 때 이 수치들은 1킬, 2킬, 그리고 3킬로 볼 수 있을 것이다.



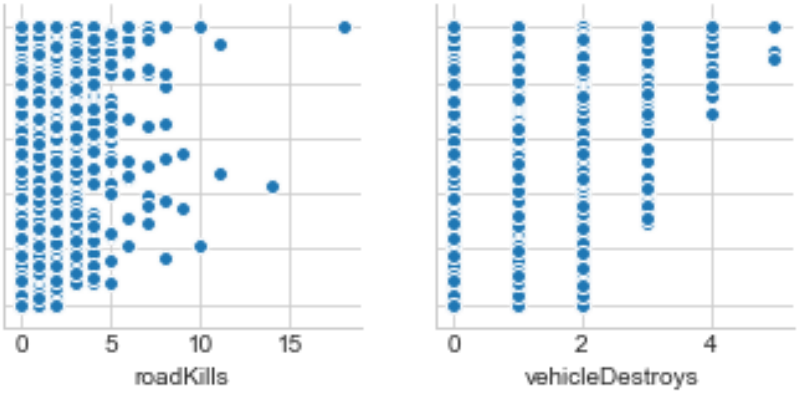
위 그림은 damageDealt와 winPlacePerc의 상관관계를 확인하기 위한 분포도인데, kills와 DBNOs와 마찬가지로 양의 상관관계를 확인할 수 있다.

**(3) 기타**

kills, DBNOs, 그리고 damaeDealt 이외에도 kills이나 damage에 관련된 여러 Feature들이 있다. 이들 각각 winPlacePerc와의 상관관계를 확인해볼 수 있다.




- headshotKills: 플레이어가 헤드샷(적의 머리를 맞춤)을 통해 처치한 적의 수를 지칭한다.
- killStreaks: 짧은 시간동안 플레이어가 빠르게 죽인 적의 수를 지칭한다.
- longestKill: 플레이어와 죽은 적 사이의 최대 거리를 지칭한다.

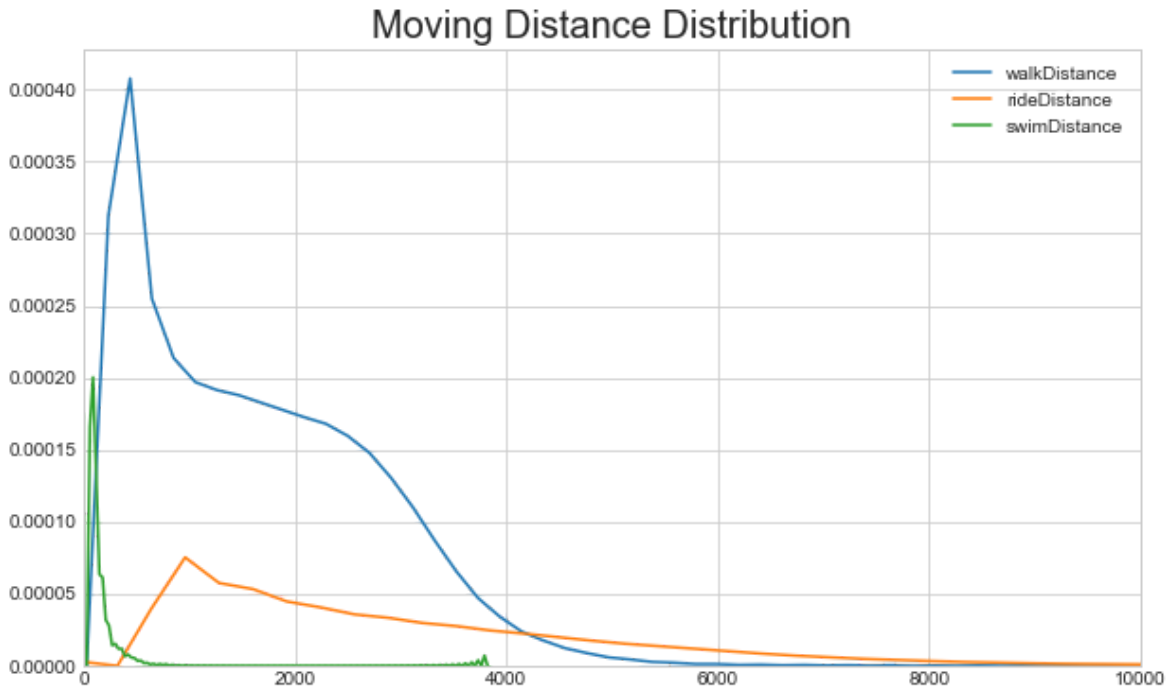



- roadKills: 차량을 타고 이동하면서 차로 죽인 적의 수를 지칭한다.
- vehicleDestroys: 매치 내에서 파괴한 차량의 수를 지칭한다.

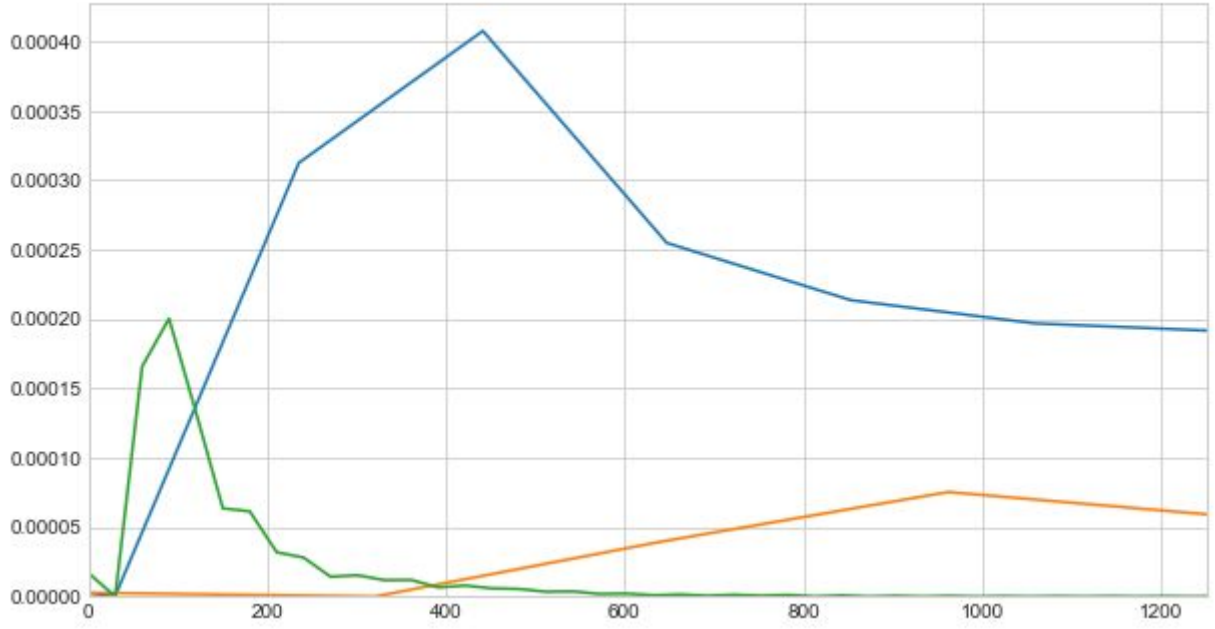
### 3-3. Player Movement

매치가 진행됨에 따라 플레이어는 자기장을 피해서, 적을 피하거나 찾아서, 또는 전략적으로 유리한 지점을 먼저 차지하기 위해서 계속 이동하곤 한다. 반대로 이동을 하지 않고 같은 위치를 고수하는 플레이어도 존재한다

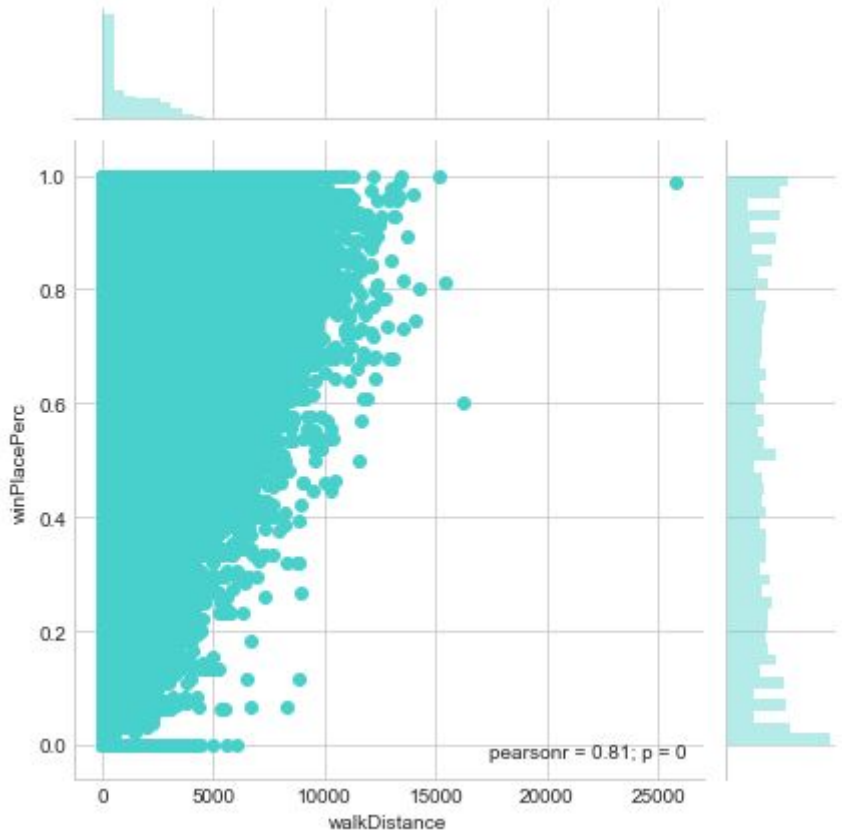
이동하는 방법으로는 지형 또는 전략에 따라 걷거나, 차량을 타거나, 수영을 할 수 있다 이렇게 이동한 거리는 3가지 Feature를 통해 알아볼 수 있는데, 총 걸은 거리는 walkDistance, 차량을 타고 이동한 거리는 rideDistance, 그리고 강이나 바다를 수영하여 이동한 거리 swimDistance에서 측정된다.




이들의 분포는 위 그림과 같은데, 이동거리가 6km이상 되는 플레이어는 거의 없다고 볼 수 있다.




이렇게 0과 1250m 사이를 확대하여 더 자세한 분포를 확인해보면, 걸은 거리가 0m인 플레이어는 없다. 차량을 탄 플레이어는 최소 300m 정도를 이동하였으며, 수영을 한 플레이어는 약 100m 정도를 이동하였다. 대부분의 플레이어는 걷는 것을 통해 주로 이동하는 것을 확인할 수 있다.



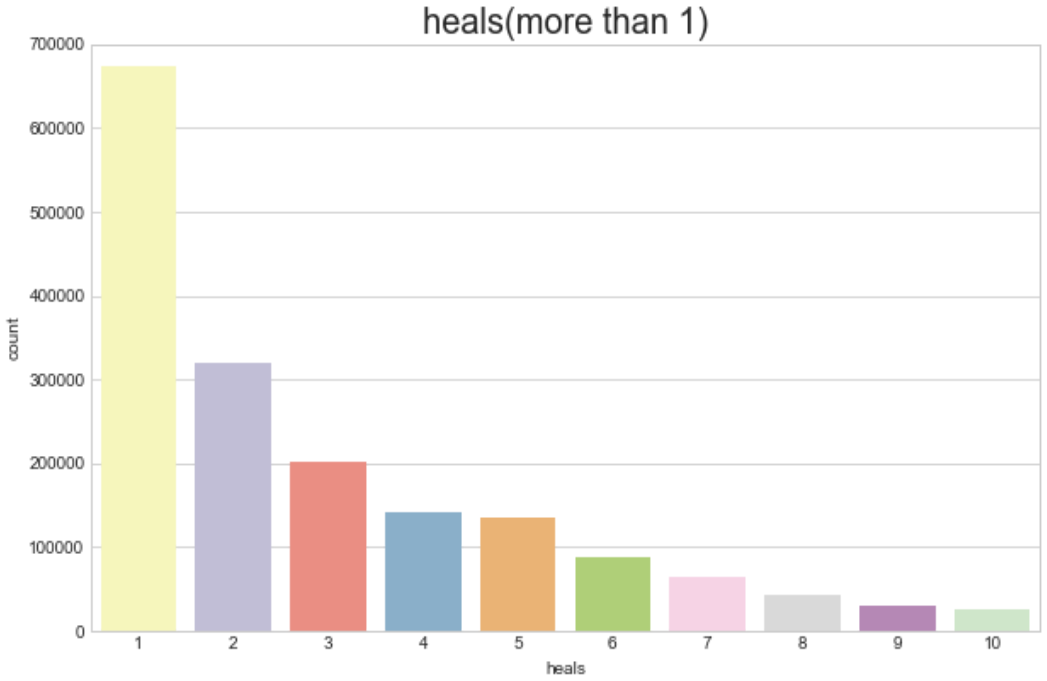

이동 거리 관련 Feature 중 가장 분포가 돋보였던 walkDistance와 winPlacePerc와의 상관관계를 위 그림을 통해 확인할 수 있는데, 확실한 양의 상관관계를 보이고 있다.

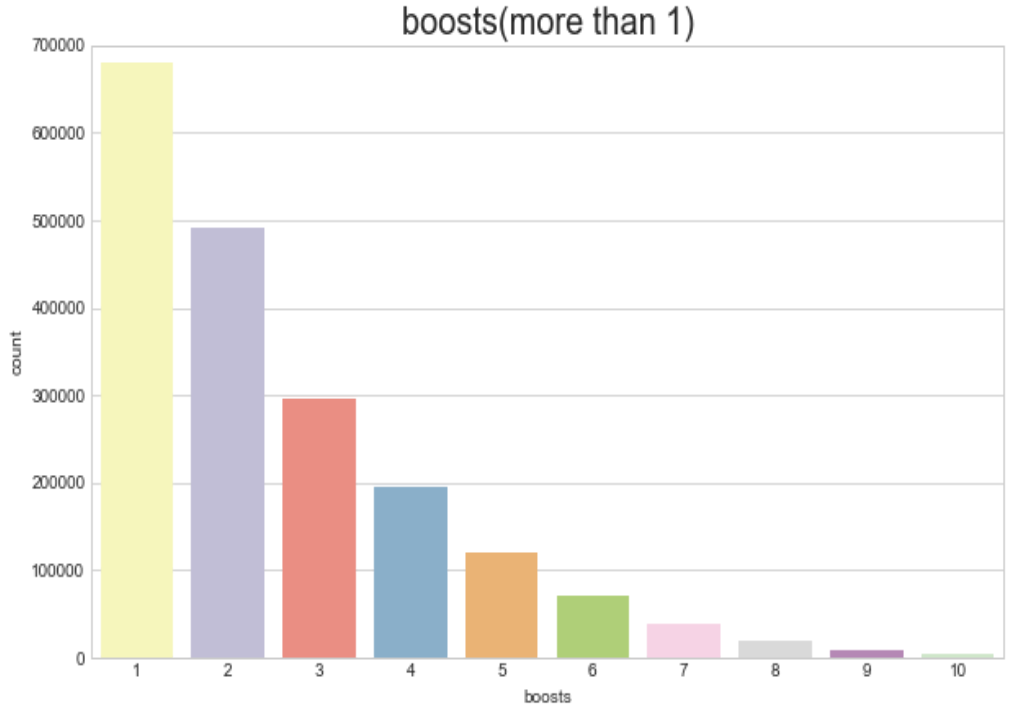
### 3-4. Items Acquired

#### (1) heals & boosts

- heals: 회복 아이템 중 HP를 복구해주는 아이템 사용 수를 지칭한다.
- boosts: 회복 아이템 중 체력을 복구해주는 아이템 사용 수를 지칭한다.

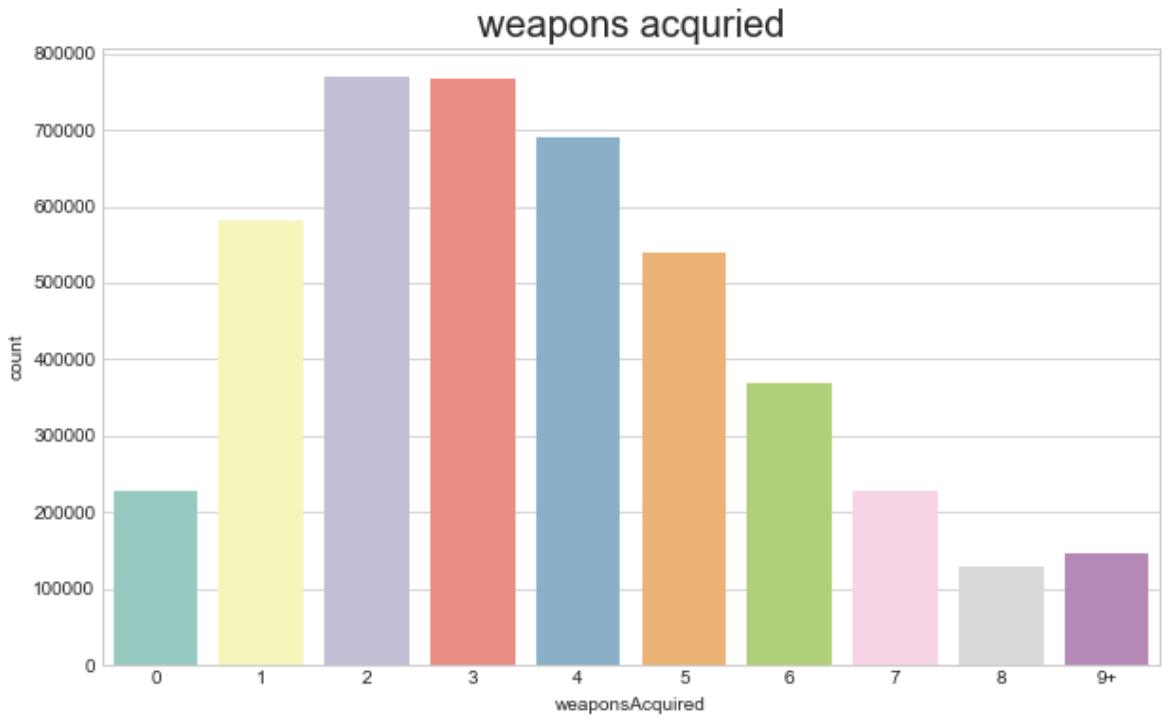
대부분의 경우 0개의 아이템을 사용했기 때문에 1이상의 아이템을 사용한 플레이어의 분포를 확인해 보는 것이 조금 더 효과적이다. 아래 두 그래프가 그것인데, heals와 boosts 모두 지속적으로 감소하는 추세를 보이고 있음을 확인할 수 있으며, 10개 이상의 아이템을 이용한 경우는 거의 없다고 볼 수 있다.




**(2) weaponsAcquired**

- weaponsAcquired: 한 매치 내에서 플레이어가 습득한 무기의 수를 지칭한다.
  보통의 플레이어의 경우, 권총을 제외하고 최대 습득할 수 있는 총기 2자루를 무조건 습득한다고 예상할 수 있다. 따라서 0또는 1개의 무기를 습득한 플레이어는 오랫동안 생존하지 못했다고 예측할 수 있다.

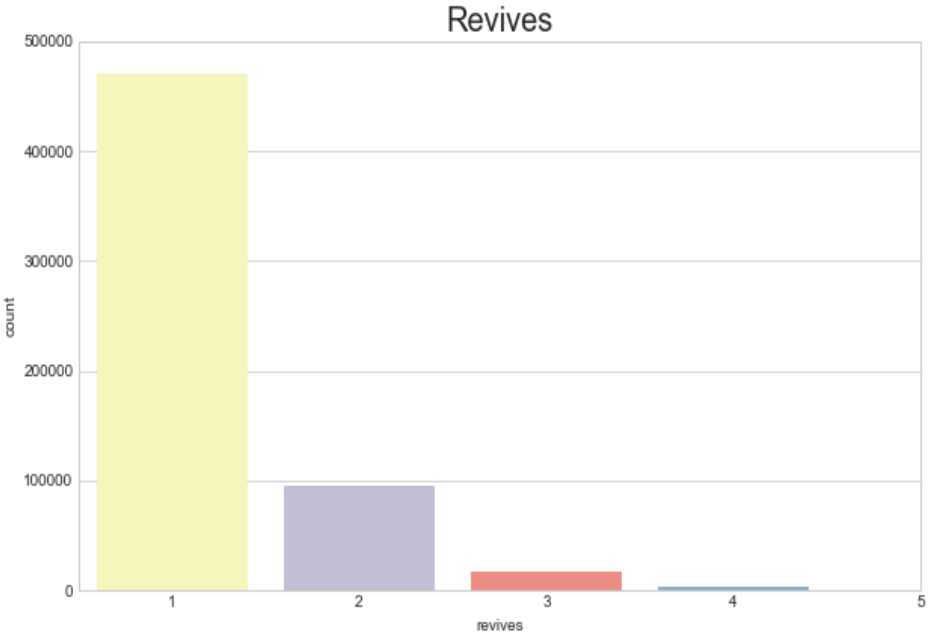



### 3-5. Team Play

PUBG는 팀게임이기 때문에 좋은 팀플레이는 보통 좋은 성과(순위)로 이어질 것이다. 따라서 인게임 데이터 중 팀플레이를 보여주는 Feature들이 존재한다.

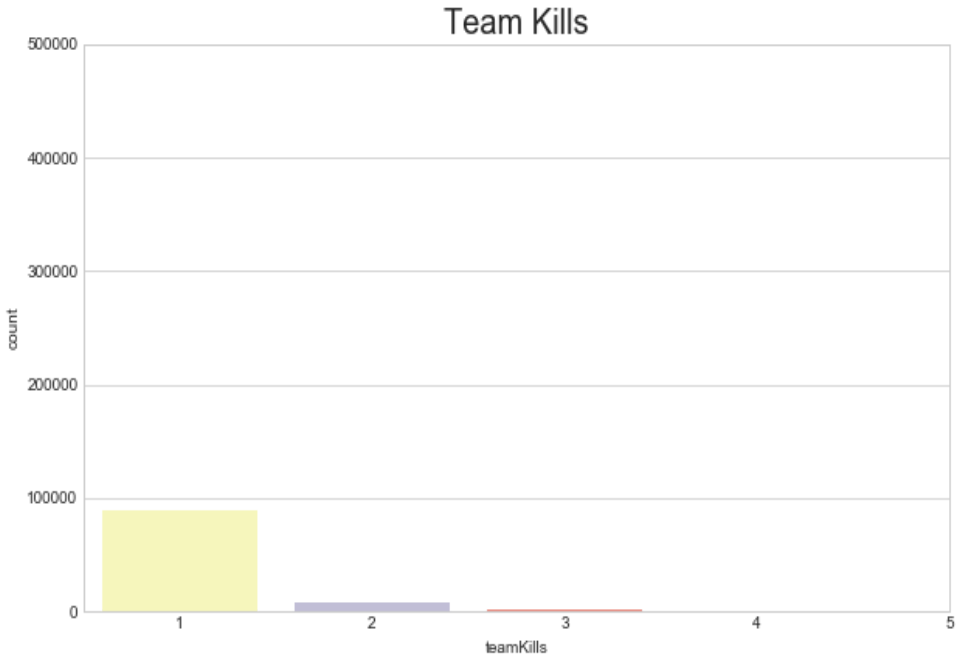
**(1) revives**

- revives: 소생시킨 기절한 팀원 수를 지칭한다.



**(2) teamKills**

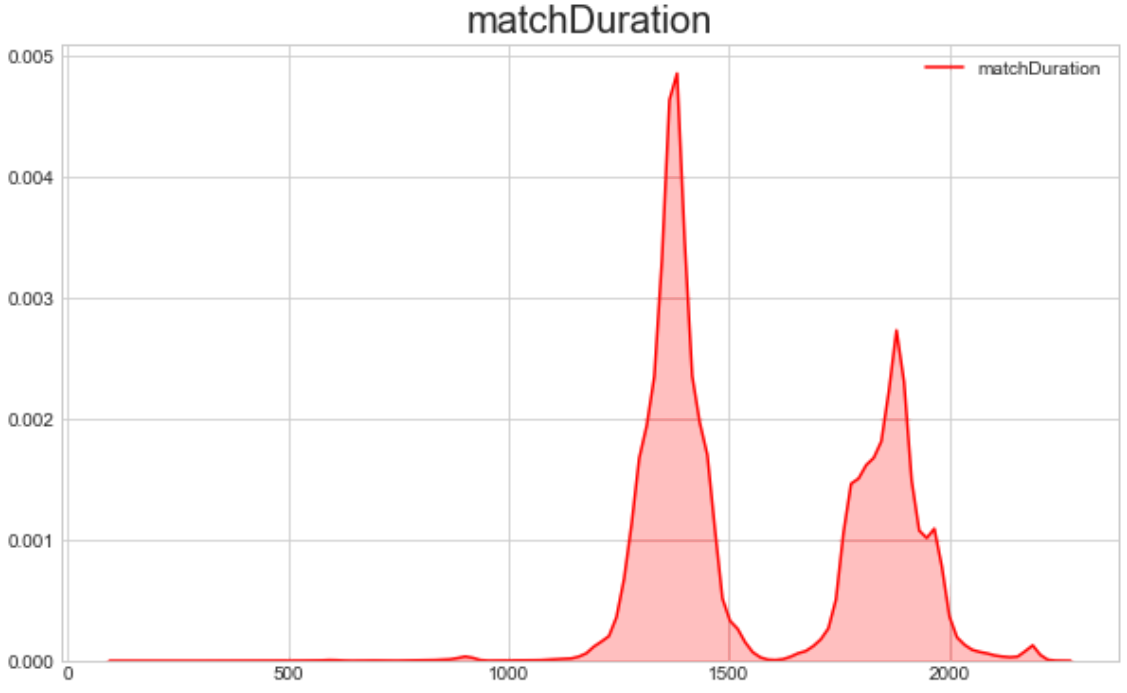
- teamKills: 플레이어가 죽인 같은 팀원의 수를 지칭한다.




## 3-6. Match Information

**(1) matchDuration**

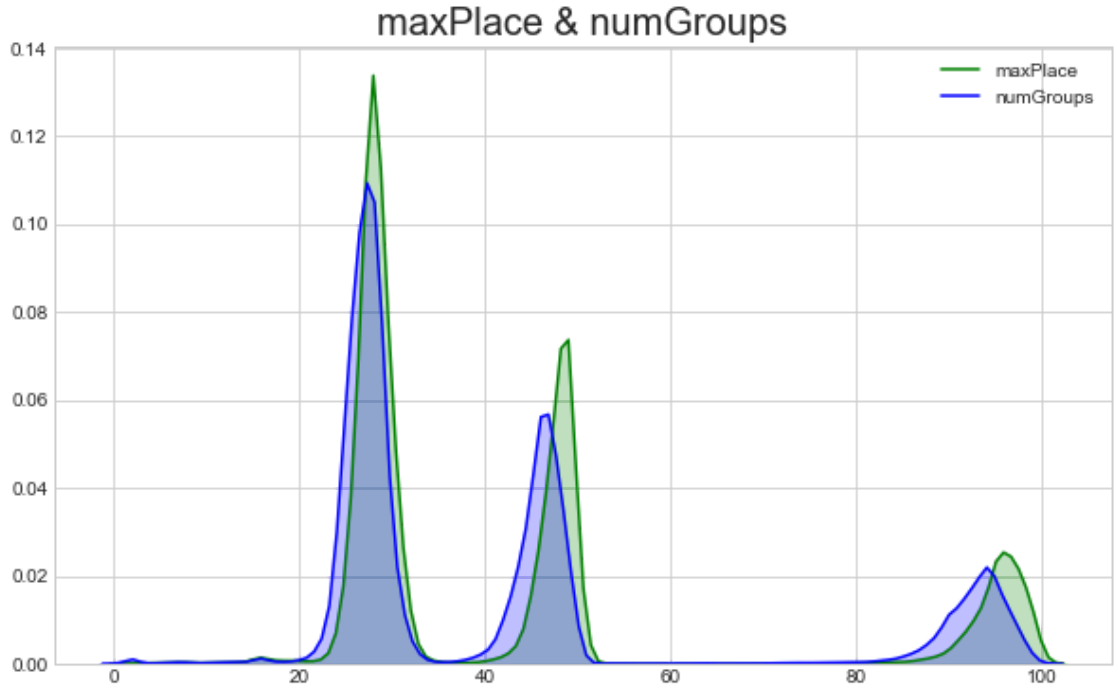
- matchDuration: 한 매치가 지속된 총 시간을 측정한다.



이 분포에서 직관적으로 2개의 클래스를 확인할 수 있다. 이는 맵의 종류가 크게 2종류[2]라는 사실에 의한 것으로 볼 수 있다. 매치가 지속되는 시간은 보통 마지막 자기장이 잡히는 시간이므로, 맵의 종류(크기)에 따라 2가지 클래스로 나눌 수 있다는 것이다. 즉, 왼쪽에 분포하고 있는 데이터는 작은 맵인 사녹에서, 오른쪽 분포는 에란겔이나 미라마에서 이루어지는 매치를 나타내는 것이다.

**(2) numGroups & maxGroups**

- numGroups: 매치 내에 존재하는 그룹의 수를 지칭한다.
- maxGroup: 매치 내에 존재하는 그룹 중 데이터를 가지고 있는 그룹의 수를 지칭한다.

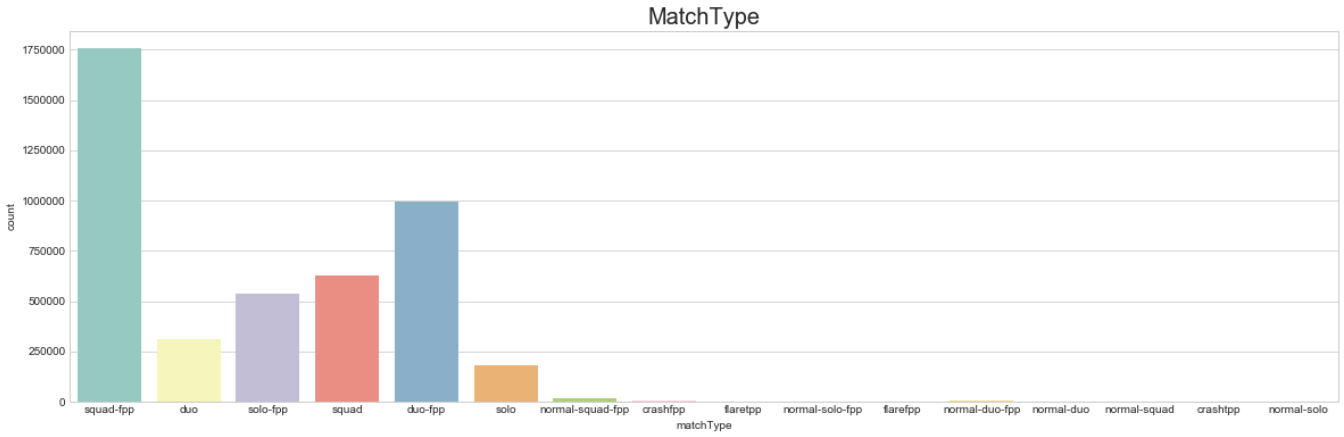



앞선 matchDuration과 마찬가지로, 직관적으로 3개의 클래스를 확인할 수 있다. 그룹의 수와 관련된 Feature이므로 이는 그룹의 종류에 의한 차이라고 볼 수 있다. 가장 왼쪽 클래스는 스쿼드, 중간은 듀오를, 가장 오른쪽 클래스는 솔로를 지칭한다고 볼 수 있다.

**(3) matchType**

- matchType: 그룹 종류(솔로, 듀오, 스쿼드)과 인칭(fpp[3], tpp[4])을 나타내며, 기타 이벤트 모드나 커스텀 모드에 대한 정보도 포함되어 있다.

[2] 비켄디(Vikendi)는 2018년 12월에 출시된 맵으로서, 이 데이터셋이 만들어진 후이다.
[3] First Person Perspective의 약자로, 1인칭을 뜻한다.
[4] Third Person Perspective의 약자로, 3인칭을 뜻한다.




matchType은 categorical한 Feature로서, 총 16가지 종류가 있다. 특히 1인칭과 3인칭 솔로, 듀오, 스쿼드 모드에 대한 데이터가 많음을 확인할 수 있으며, 이벤트 모드나 커스텀 모드에 대한 데이터는 상대적으로 적다.

**(4) IDs**

매치 내의 정보를 고유한 ID로 저장하고 있는 Feature들이 있다.

- matchId: 매치의 고유 ID이다.
- groupId: 매치 내에서 그룹(팀)의 고유 ID이다.
- Id: 매치 내에서 플레이어의 고유 ID이다.

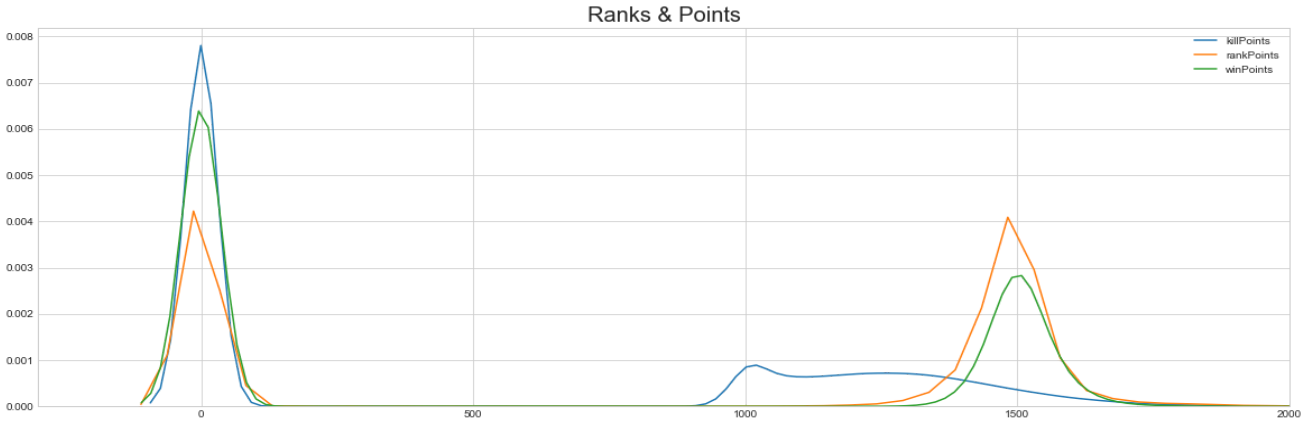
같은 그룹이나 플레이어도 다른 매치라면 다른 ID를 부여받으므로, 각 ID는 매치 내에서 고유하다고 볼 수 있다.

## 3-7. Points

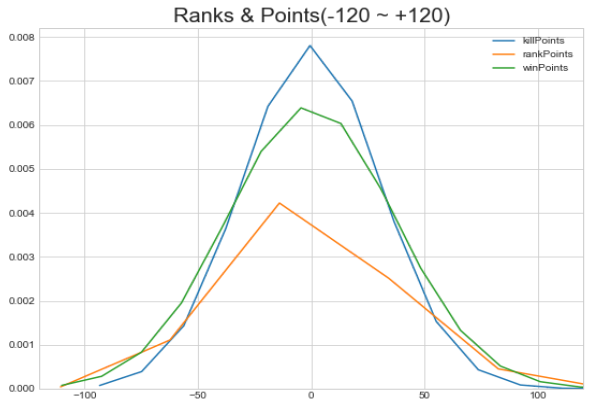
플레이어의 실력을 점수로서 평가하는 Feature도 존재한다. 이는 Elo-ranking 방법을 이용하여 외부 데이터와 비교하여 점수를 매기는 방식으로 측정된 데이터이다.

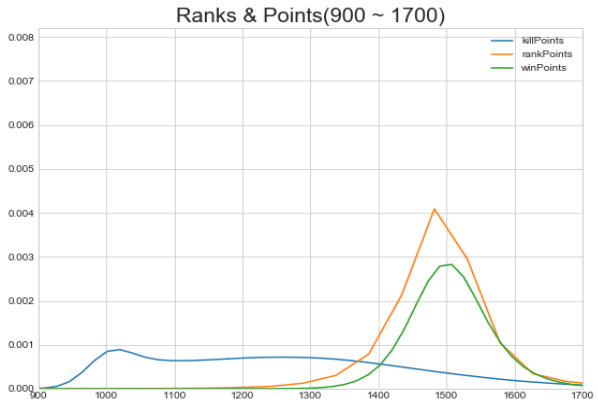
- killPoints: 킬 수를 바탕으로 한 Elo-ranking을 측정한다.
- winPoints: 승리만을 고려한 Elo-ranking을 측정한다.
- rankPoints: 순위를 바탕으로 한 Elo-ranking을 측정한다.

이 Feature들은 아래와 같이 분포한다.



위에서 확인할 수 있듯이, point와 관련된 Feature의 경우 0 값이 굉장히 많다.




확대해 보면 데이터가 있는 경우보다 없는 경우(0)가 더 많다. 또한, 이 외에도 확인을 해보면 NULL값을 가진 데이터도 굉장히 많다. 따라서 이 3가지 Feature는 모델 학습 과정에서 주요하지 않게 작용할 것이라 예상할 수 있다.

## 4. Hypothesis

데이터를 조작하고 모델을 만들기 전에, 분석한 데이터에 대한 간단한 가설 몇가지를 세울 수 있다. 이를 바탕으로 우리는 데이터를 어떻게 조작할지에 대한 Wrangling Hypothesis를 구체화하고, 나아가 더 효과적인 모델을 만들고자 한다.

### 4-1. Player Skills

총 게임의 특성 상, 더 좋은 에임이나 반응 속도를 가지고 있는 숙련된 플레이어가 더 오래 생존할 확률이 높을 것이다. 또한, 전략이 중요한 게임이므로 게임 상황을 보고 좋은 전략을 세우고 빠른 판단을 내릴 수 있는 플레이어가 높은 순위를 차지할 것이다. 따라서 우리는 숙련된 플레이어가 누구인지 찾아내야 한다. Kills이나 damage 등과 관련된 Feature에서 높은 수치를 보인 플레이어는 상대적으로 숙련된 플레이어라고 판단할 수 있을 것이다. 즉, 우리는 kills, DBNOs, damageDealt등의 Feature들이 winPlacePerc을 결정하는데에 결정적인 영향을 끼칠 요소이며, 이들이 높을 수록 플레이어의 최종 순위는 높을 것이라고 예상한다.

### 4-2. Teamwork

PUBG는 솔로를 제외하고는 팀게임이므로, 팀원 간의 협동심과 교전에서의 효과적인 팀플레이를 보여주는 팀이 높은 순위를 차지할 것이다. Assists와 revives의 경우, 높을 수록 팀원 간의 엄호 사격과 지원이 잘 이루어지고 있다고 볼 수 있으며, team kill의 경우 반대로 높을 수록 팀원 간 불화가 일어나고 있다고 예상할 수 있다. 따라서 우리는 이들이 생존율에 영향을 줄 것이라고 본다. 다만, 이 수치들은 대체적으로 적었기 때문에, 결정적인 요인이 되지는 않을 것으로 본다.

Team size 또한 중요한 요소가 될 것인데, 최대 인원을 채우지 못한 팀은 그렇지 않은 팀보다 파밍 효율성, 아이템 배분, 지원 사격 등에서 부족한 부분을 보일 수 밖에 없을 것이다. 따라서 이는 팀의 순위에 큰 영향을 미칠 것이다.

**4-3. Items Acquired**

더 많은 아이템을 소유하고 있는 플레이어는 더 오래 생존할 것이며, 또 더 오래 생존한 플레이어는 더 많은 아이템을 소유하고 있을 수 밖에 없다. 따라서 heals, boosts, 그리고 weaponsAcquired가 높은 플레이어의 경우 높은 순위에 안착할 것이다.

# II. Data Wrangling

## 1. Wrangling Hypothesis

### 1-1. Anomaly 제거

불법 프로그램은 유저의 상태 등을 교묘하게 조작하여 게임 내에서 불가능한 상황을 초래한다. 따라서 data에서 극단적인 값을 갖거나 불가능한 값을 갖는 경우 제거해야 예측에 더 도움이 된다고 판단하였다.

### 1-2. Data 정규화

전체적인 데이터가 정규분포를 갖도록 하는 정규화가 아니다. 본 게임의 경우 각 경기마다 참여 인원수가 다르기 때문에 kill수, damage수 등이 전체 플레이어 수에 따라 차지하는 그 비중이 달라질 수 있다. 따라서 전체 플레이어 수로 나누어 '정규화'를 주었을 때 더 정확한 예측이 가능할 것이다.

### 1-3. Feature 생성

비슷한 속성을 갖는 feature들의 경우 각자의 영향력은 다소 낮지만 합할 경우 높아지는 경우가 종종 있다. 이러한 경우를 고려하여 새로운 feature를 생성하면 정확성이 높아질 것이다.

### 1-4. Feature 제거

상관 관계가 높은 feature들의 경우 대표적인 feature만 남기고 삭제하는 것이 더 좋을 것이다. (Occam's razor) 또한 중요도가 아주 낮은 feature들의 경우 제거하는 것이 더 좋을 것이다.

### 1-5. Feature 선택

모든 feature들을 반영하여 학습시키는 것보다 중요한 feature들만 남겨 학습시키는 것이 정확도 상승에 더 좋을 것이다.

### 1-6. 중요하지 않은 데이터 제거

이벤트 모드와 같은 특수한 경우에서의 데이터는 그 수도 적고 중요도도 낮기 때문에 그러한 데이터를 제거할 때 variance를 낮출 수 있을 것이다.

### 1-7. 데이터 그룹화

본 게임의 경우 solo, duo, squad 등 그룹으로도 경기 참여가 가능하다. duo나 squad의 경우 본인이 죽더라도 팀원이 살아남는다면 우승이 가능하므로 groupId로 그룹화를 하면 정확도가 더 높아질 것이다.

위의 가정들을 바탕으로 다양한 전처리 방법을 활용하여 8개의 data frame을 만들었다. 각 data frame의 유의미함을 판단하기 위해 baseline모델인 linear regression 모델을 사용하였다.

## 2. Data Frames

### 2-1. Data Frame #1

**(1)** 첫번째 data frame의 경우 전체 data frame들의 baseline으로서 사용하였다. Raw data를 조금 수정하여 사용하였다.

**(2)** 먼저 winPlacePerc, 즉 target값이 Nan인 data를 삭제하였다.

**(3)** 다음으로 matchType feature를 one-hot encode하였다. matchType의 경우 categorical 값을 갖고 있기 때문에 모델에 적용하기 위해 수치화해주었다.

```
In [46]: # One hot encode matchType
df_wrangle = pd.get_dummies(df_wrangle, columns=['matchType'])
train = pd.get_dummies(train, columns=['matchType'])
# Take a look at the encoding
matchType_encoding = df_wrangle.filter(regex='matchType')
matchType_encoding_train = train.filter(regex='matchType')
```

| matchType |
|---|
| squad-fpp |
| squad-fpp |
| duo |
| squad-fpp |
| solo-fpp |

→

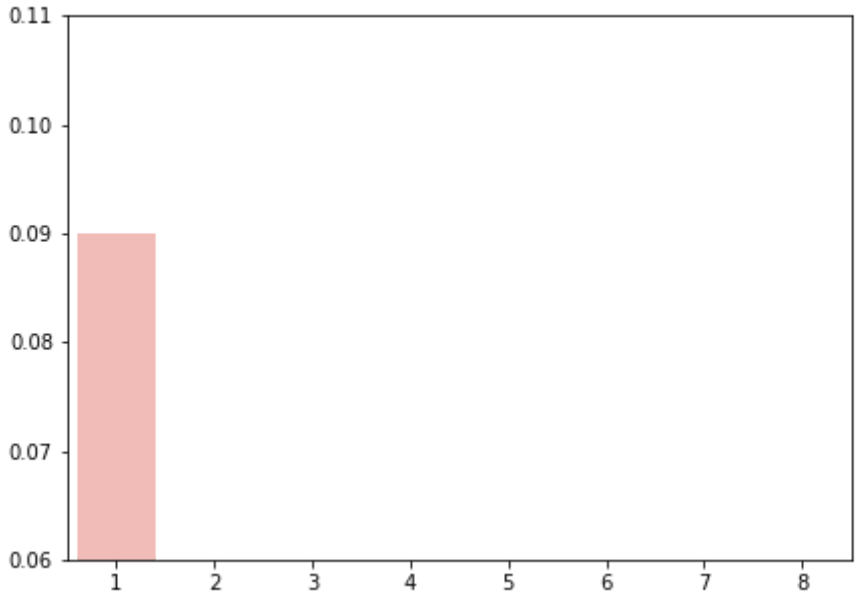
| matchType_normal-solo-fpp | matchType_normal-squad | matchType_normal-squad-fpp | matchType_solo | matchType_solo-fpp | matchType_squad | matchType_squad-fpp |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 |

**(4)** 마지막으로 Id, groupId, matchId의 경우 고유값이므로 중요하지 않다고 판단하여 3 feature를 삭제하였다.

**(5)** 이러한 처리가된 첫번째 data frame을 바탕으로 다양한 전처리가 이루어졌다.

**(6) Errors**

다음 그래프는 첫번째 data frame의 validation error이다.



**2-2. Data Frame #2**

두번째 data frame의 경우 전처리에 대한 가정의 대부분을 반영하였다.

**(1) Anomaly 제거**

불법 프로그램 이용자로 의심되는 anomaly들을 제거하였다.

→ 움직임이 없는 kills

→ kills가 20 이상인 경우

→ kills수에서 headShot의 비율이 0.9 이상이거나 headShot이 10 이상인 경우

→ longestKill이 1000 이상인 경우

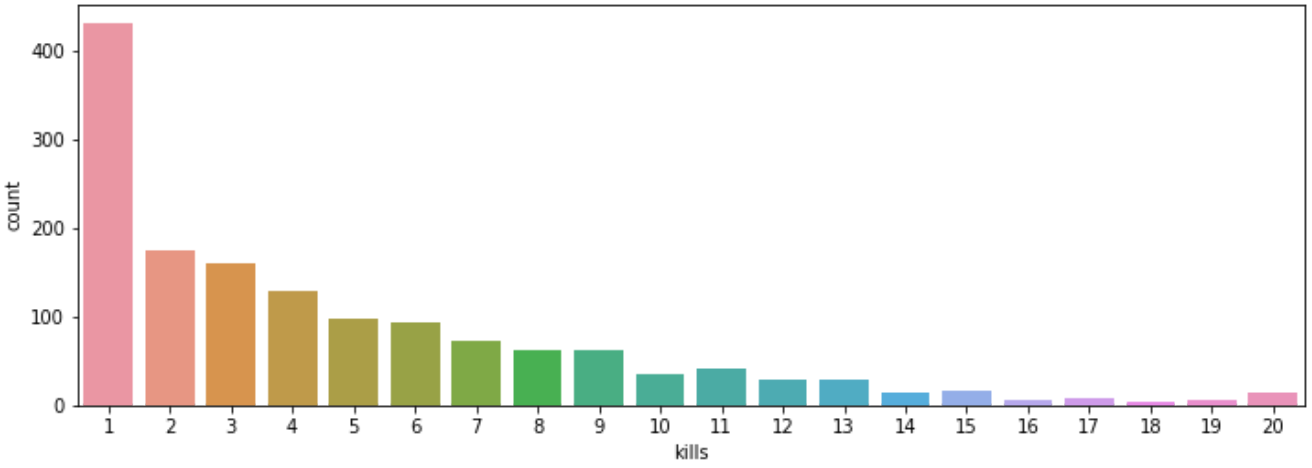
→ walkDistance가 10000 이상인 경우

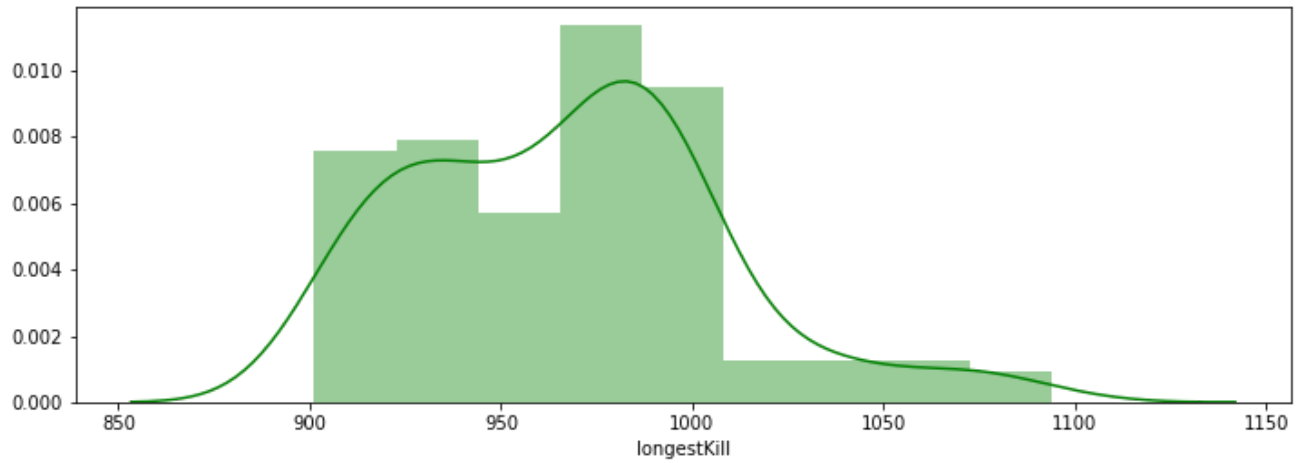
→ rideDistance가 20000 이상인 경우

→ swimDistance가 2000 이상인 경우

→ weaponsAcquired가 80 이상인 경우 heals가 40 이상인 경우 등

예를 들어 다음 분포를 통해 움직임이 없는 kills수가 실제로 있음을 알 수 있다.

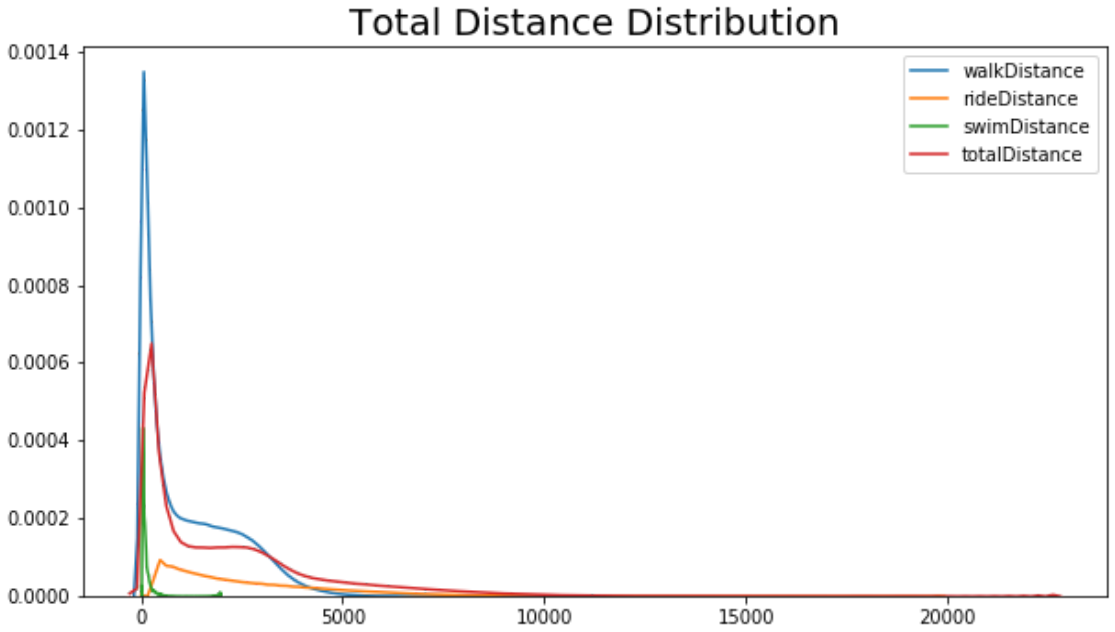


**(2) Feature Generation : totalDistance, healsandboosts**

두가지 feature를 추가하였다. totalDistance와 healsandboosts이다.

→ totalDistance는 3개의 거리 feature (rideDistance, walkDistance, swimDistance)를 합친 feature이다. 세 feature는 비슷한 분포도를 나타내고 player가 움직인 거리를 나타낸다는 공통점이 있다.



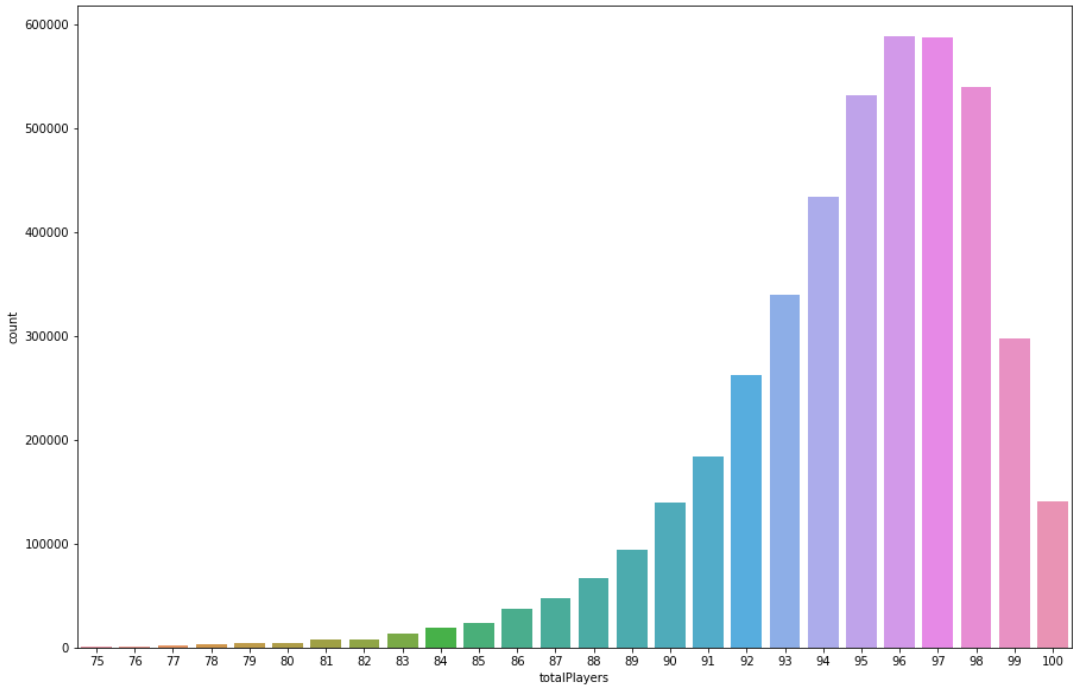

> healsandboosts는 heals와 boosts feature를 합친 것이다. heals와 boosts는 player가 더 오래 생존할 수 있도록 돕는 아이템이다.

**(3) Normalization**

각 매치에 참여하는 player 수가 상이하기 때문에 총 player 수에 따라 player의 활약이 달라질 수 있는 feature들은 총 player 수로 scale돼야 한다. 예를 들어 50 명중 5명을 죽이는 것과 100명 중 5명을 죽이는 것은 다르기 때문이다. 총 player 수로 scale된 feature들은 다음과 같다.

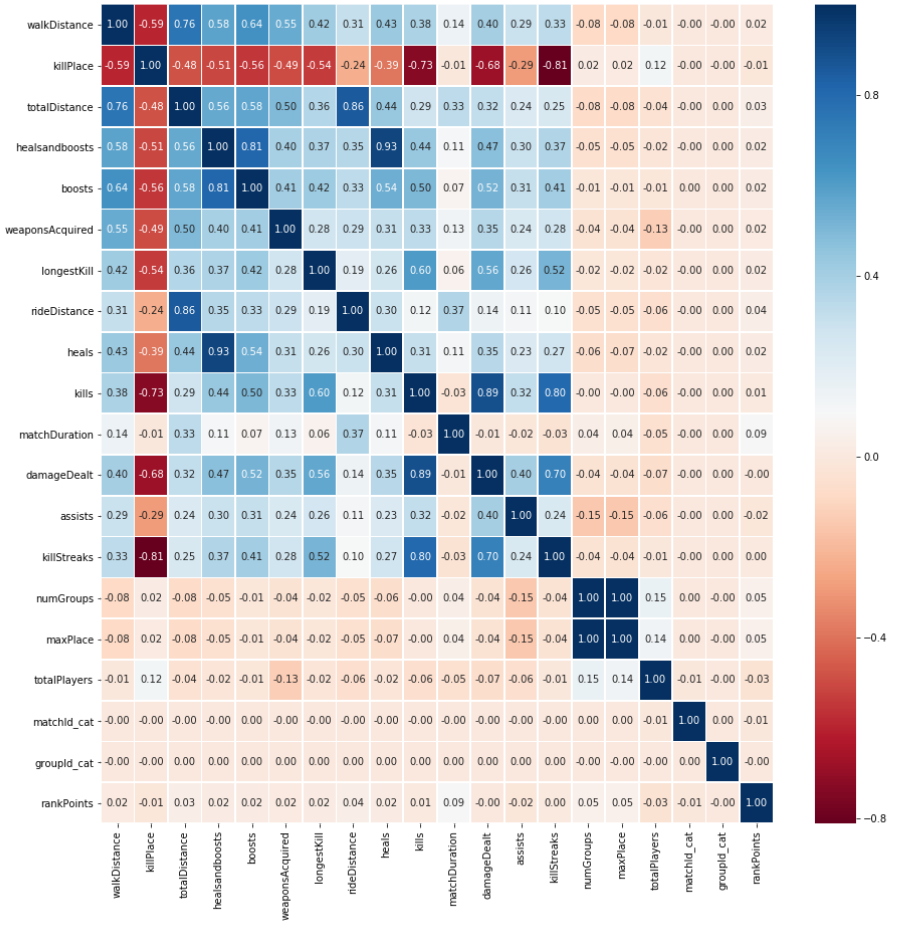
In [45]:
```
# Create normalized  val.
df_wrangle['killsNorm'] = df_wrangle['kills']*((100-df_wrangle['totalPlayers'])/100 + 1)
df_wrangle['matchDurationNorm'] = df_wrangle['matchDuration']*((100-df_wrangle['totalPlayers'])/100 + 1)
df_wrangle['damageDealtNorm'] = df_wrangle['damageDealt']*((100-df_wrangle['totalPlayers'])/100 + 1)
df_wrangle['maxPlaceNorm'] = df_wrangle['maxPlace']*((100-df_wrangle['totalPlayers'])/100 + 1)
df_wrangle['weaponsAcquiredNorm'] = df_wrangle['weaponsAcquired']*((100-df_wrangle['totalPlayers'])/100 + 1)
```

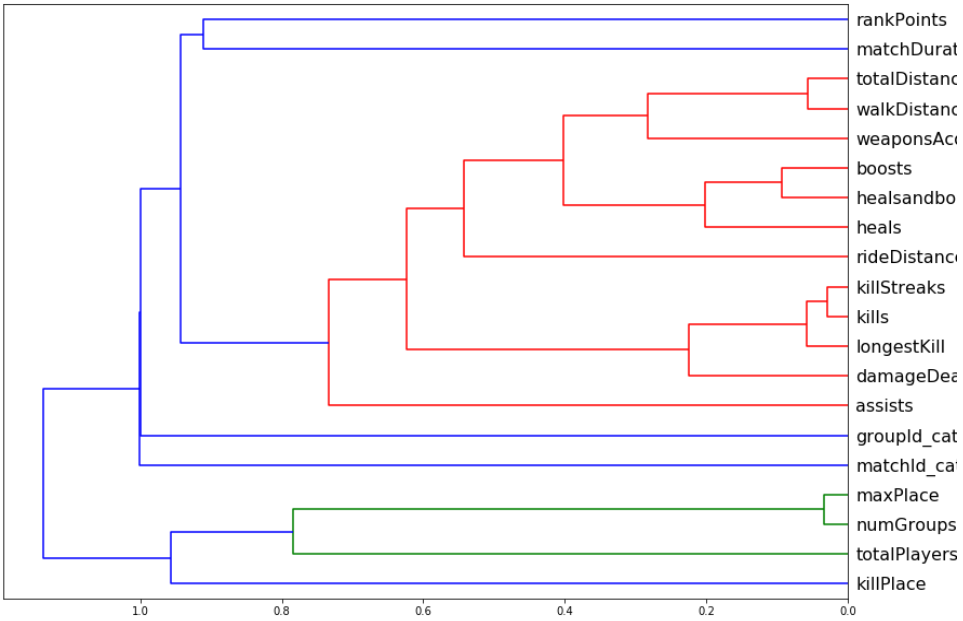
→ kills, matchDuration, damageDealt, maxPlace, weaponsAcquired



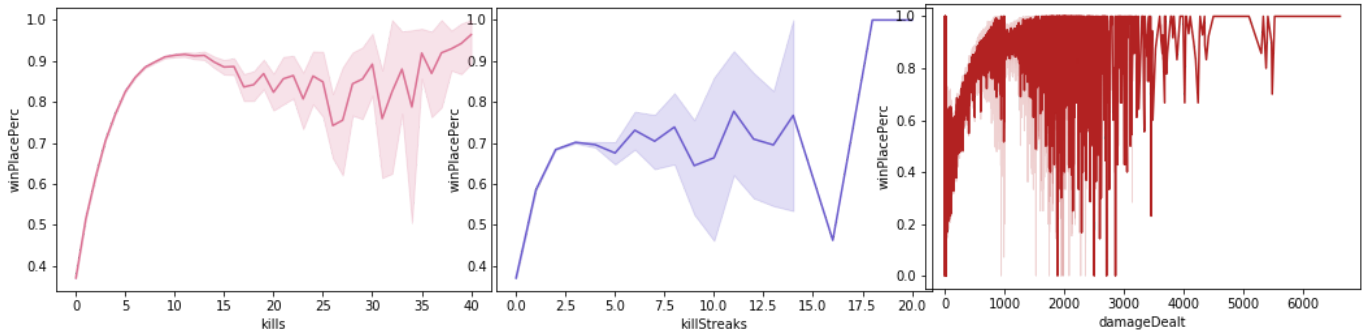

**(4) Correlation**

Correlation에 따라 feature를 제거하였다. Heatmap과 dendrogram을 통해 계수들 간의 correlation을 구하였다






→ 실제로 kills, killStreaks, damageDealt 분포를 비교하면 correlation이 높음을 알 수 있다. 이 중 kills를 제외한 killStreaks, damageDealt를 삭제하였다.

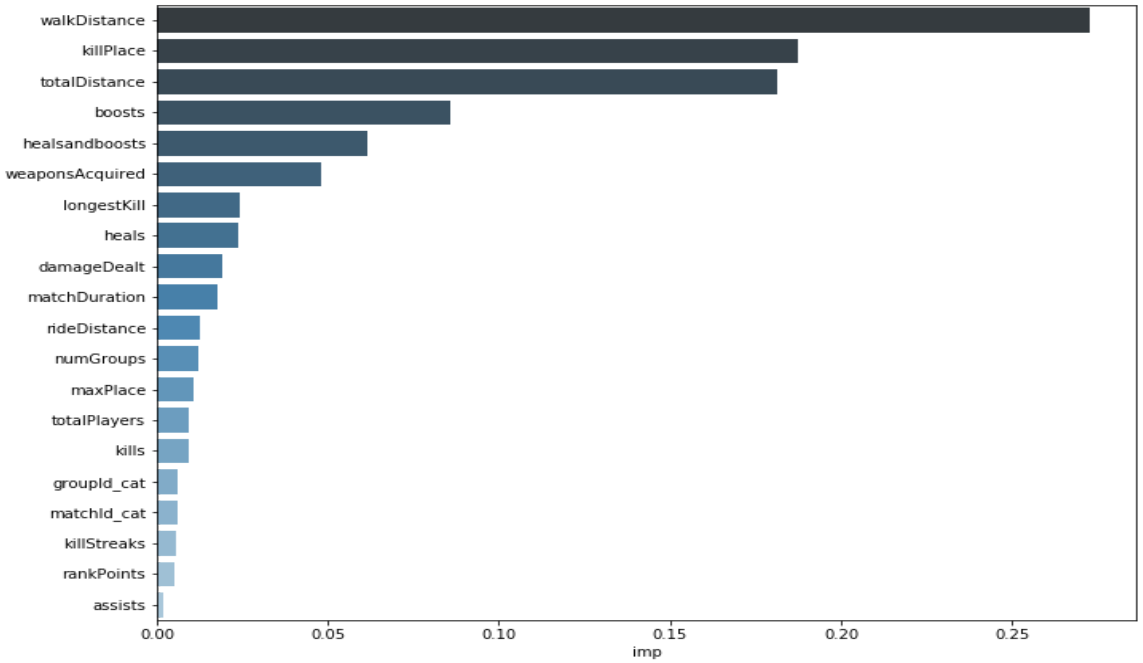



→ maxPlace와 numGroups는 거의 같다. 또한 이미 totalPlayer로 scale을 해주었기 때문에 두 feature는 중요하지 않다고 판단하여 제거하였다. 두 feature는 totalPlayer와 비슷한 성질을 갖기 때문이다.

**(5) Feature Selection**

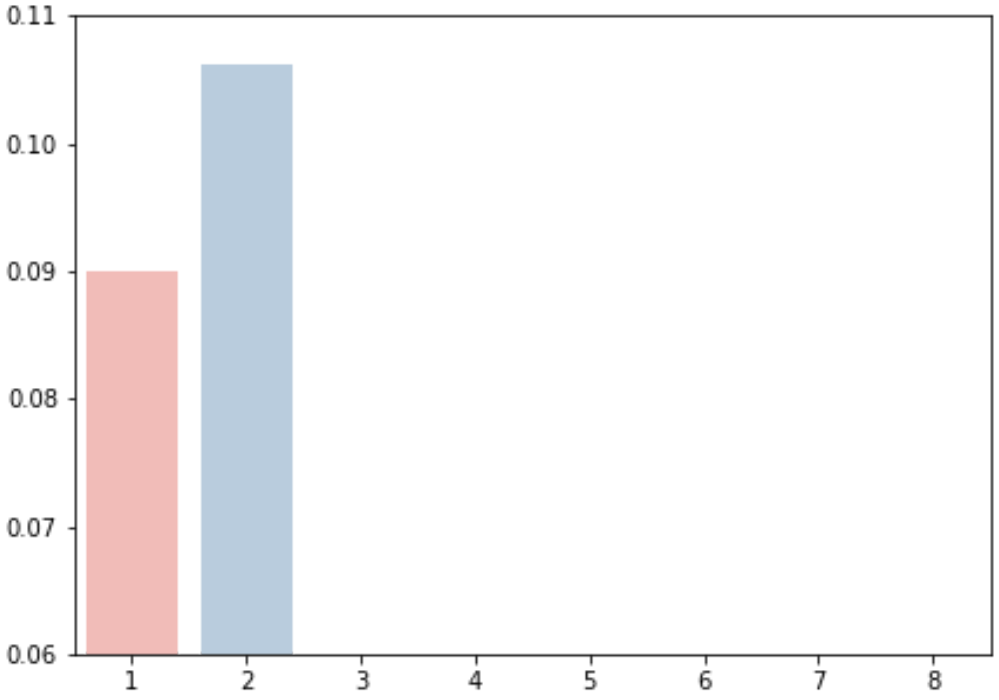
중요한 feature들만 모델에 반영하기 위해 selectKBest와 f_regression 라이브러리를 사용하였다.

In [58]:
```
from sklearn.feature_selection import SelectKBest, f_regression

m_fi = SelectKBest(f_regression, k=21).fit(X_train, y_train)
feature_idx = m_fi.get_support()
to_keep = original.columns[feature_idx]

sample = df_wrangle.iloc[0:100000]
# Split sample into training data and target variable
original = sample[to_keep] #all columns except target
target = sample['winPlacePerc'] # Only target variable
```




**(6) Errors**

그러나 이러한 전처리 과정에도 불구하고 validation error가 증가하였다. 따라서 위의 전처리 과정을 검토하였다.



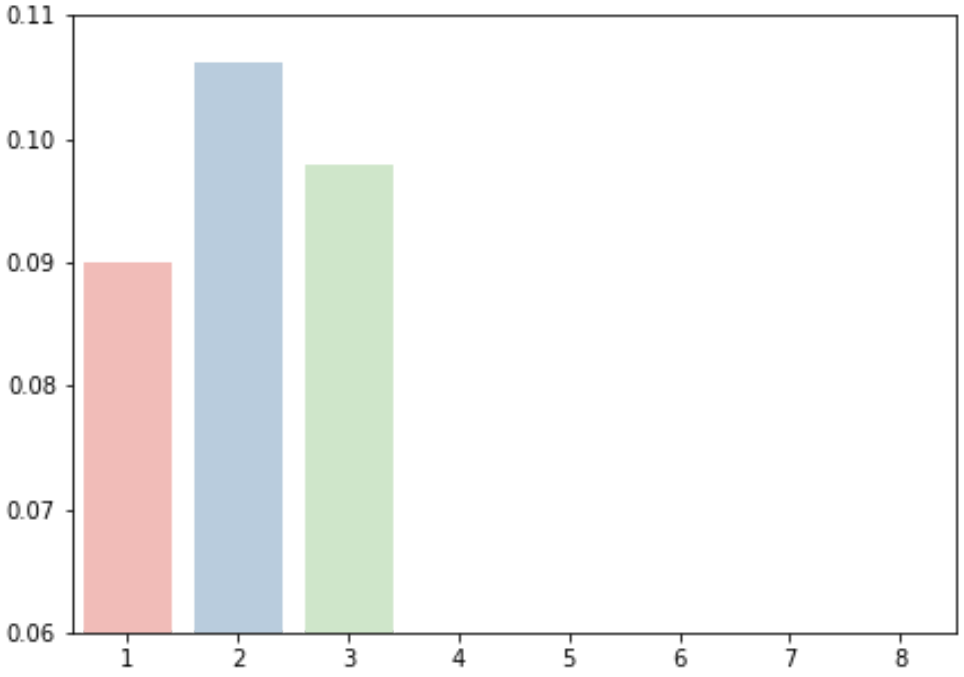
### 2-3. Data Frame #3

#### (1) Correlation 기반 feature 제거 제외

먼저 correlation을 기반으로 한 feature 제거로 data 손실이 발생해 성능이 저하되었다고 판단하여 이 과정을 제외하였다.

#### (2) Errors

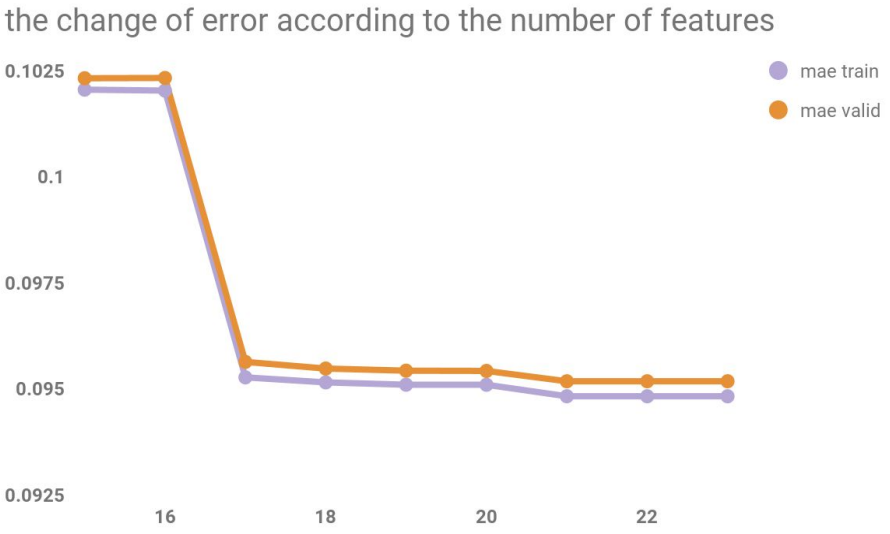
두번째 data frame에 비해 validation error가 감소하였다. 하지만 여전히 baseline data frame 보다 error가 높아 다른 과정을 더 검토하였다.



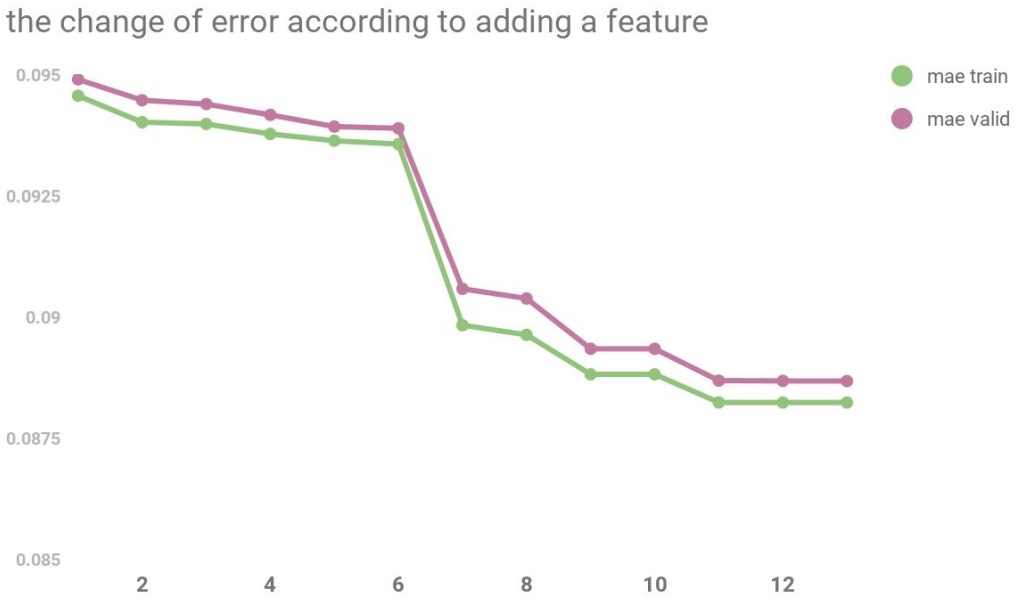

### 2-4. Data Frame #4

#### (1) Feature selection 제외

→ 21개의 feature를 가지고 모델을 학습시켰던 이유는 selectKBest 에서 K = 21 일 때 가장 error가 낮았기 때문이다.

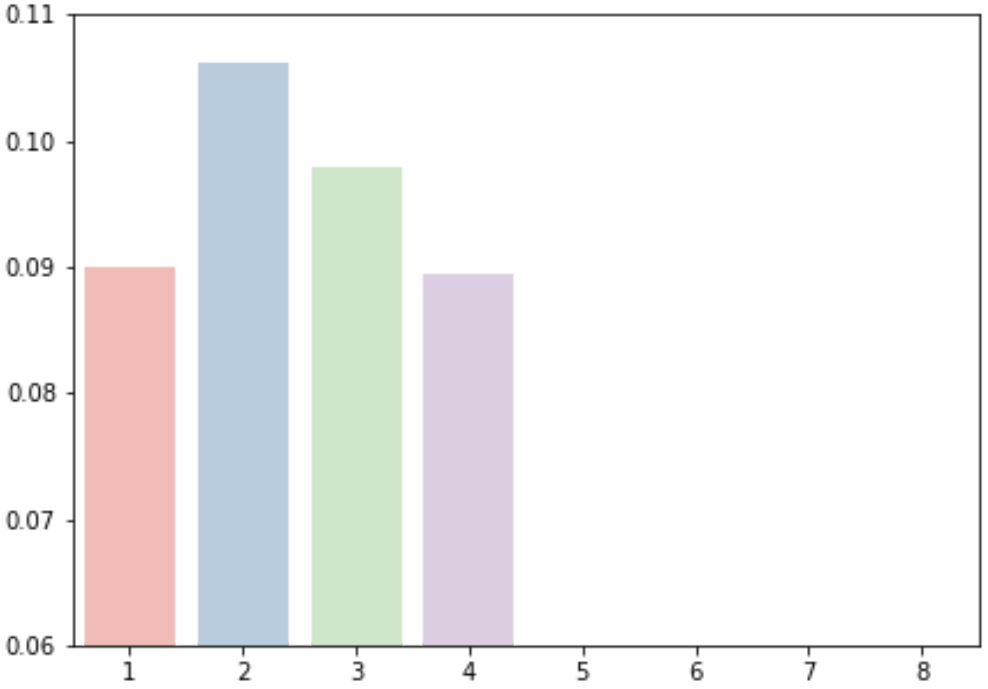


→ 하지만 선택된 21개의 feature에 우리의 추측으로 중요하다 생각했던 feature들이 없어 하나씩 추가할 때마다  error가 줄어들기 시작했다. 뿐만 아니라 중요하다고 생각했던 feature들뿐만 아니라 다른 feature들을 추가할 때도 error가 계속해서 감소하였다. 따라서 우리는 모든 feature를 사용하기로 하였다.




**(2) Errors**

네번째 data frame부터 baseline data frame보다 더 낮은 error를 갖게 되었다. 하지만 여기서 멈추지 않고 더 낮은 error를 위해 wrangling 작업을 계속하였다.



## 2-5. Data Frame #5

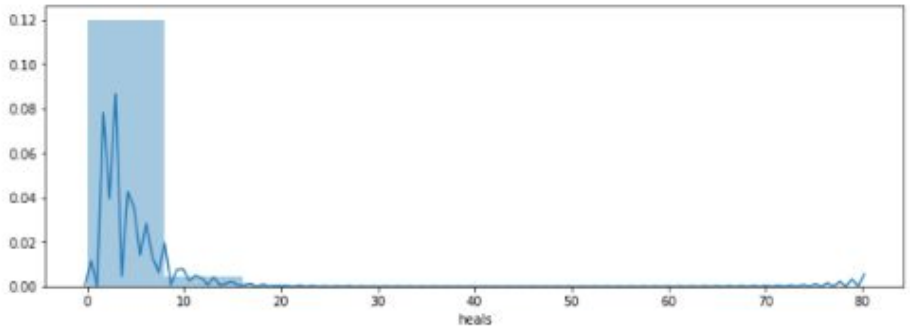
### (1) Anomaly

Anomaly 제거로 인해 많은 데이터가 손실되었다고 판단하여 anomaly 제거를 제외하였다

```
In [20]: df_wrangle[(df_wrangle['totalDistance'] == 0) & (df_wrangle['kills']<=20)&(df_wrangle['kills']>0)].shape

Out[20]: (1489, 32)
```

- kills without moving 을 제거 했을 경우에만 약 1500 개의 데이터가 사라진다.

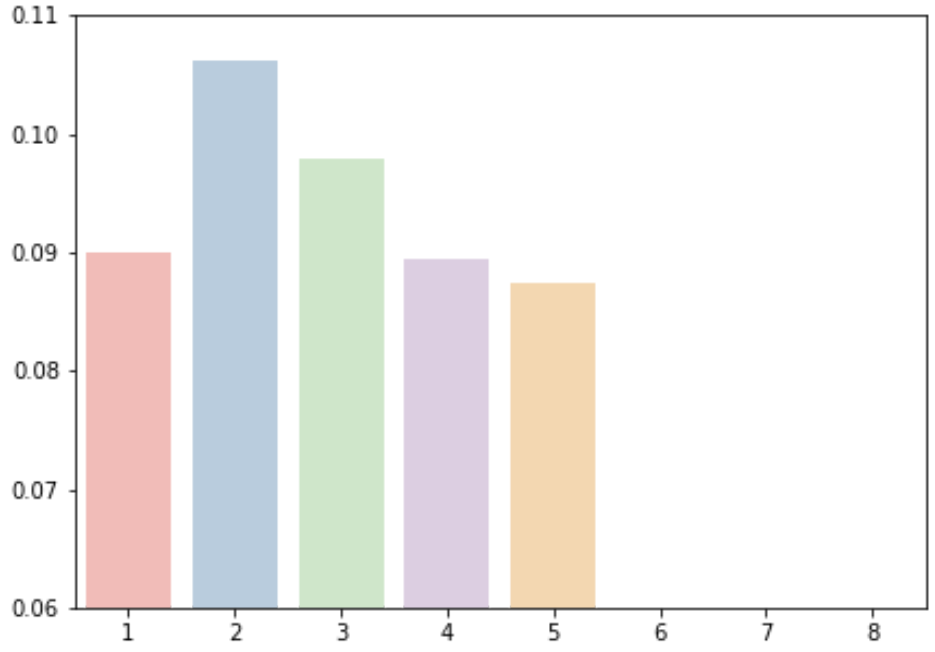
```
In [36]: #heals (40개 이상 제거)

plt.figure(figsize=(12,4))
sns.distplot(df_wrangle['heals'], bins=10)
plt.show()

display(df_wrangle[df_wrangle['heals'] >= 40].shape)

df_wrangle=df_wrangle.drop(df_wrangle[df_wrangle['heals']>=40].index)
```




(135, 30)

### (2) Errors

Data Frame #4 보다 낮은 0.0873693458927027 의 에러 rate 가 나왔다.

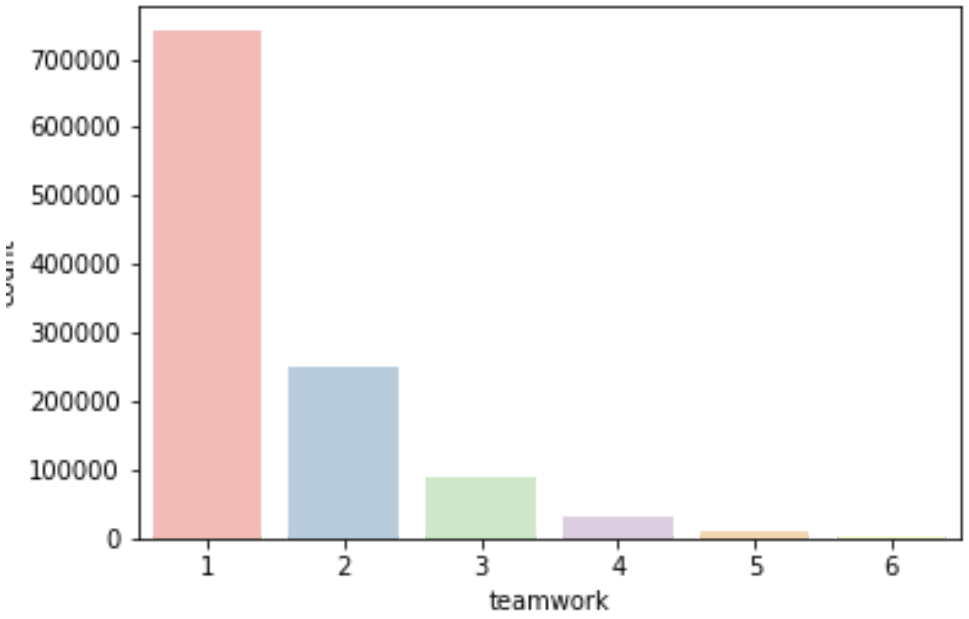


## 2-6. Data Frame #6

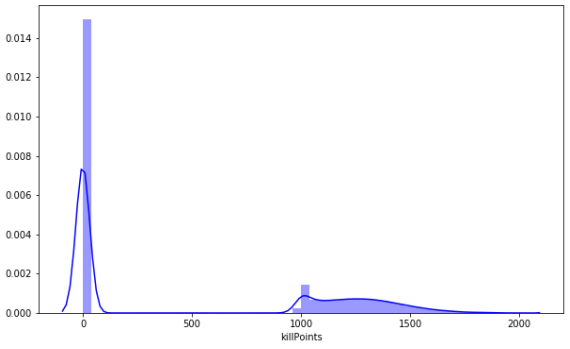
### (1) Feature Generation : teamwork

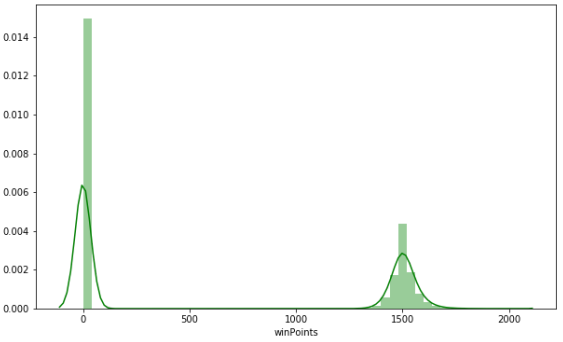
teamwork = assists + revives - teamkills

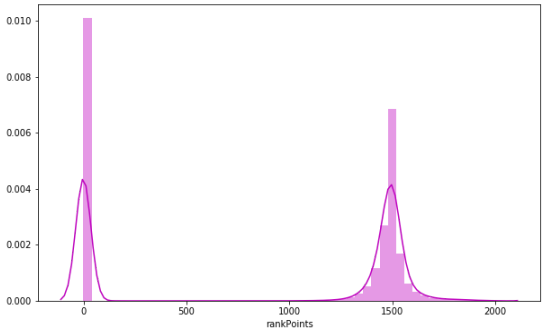
팀에서의 기여도도 중요한 feature 이라고 생각해 assists, revive 를 합하고, 팀에 안좋은 영향을 미치는 teamkills 를 뺀 teamwork 를 만들었다




### (2) Removal of  unimportant Features



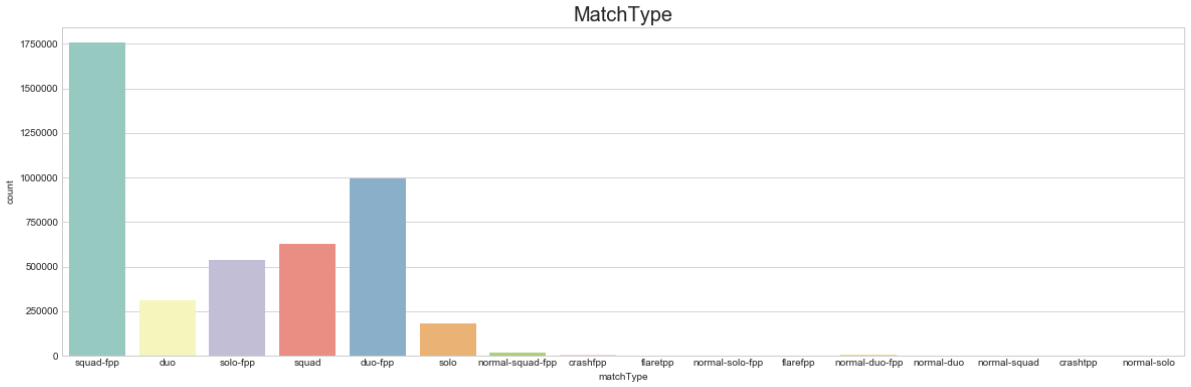



In [ ]:
```
df_wrangle.drop(columns = ['rankPoints'], inplace=True)
df_wrangle.drop(columns = ['killPoints'], inplace=True)
df_wrangle.drop(columns = ['winPoints'], inplace=True)
```

→ rankPoints, killPoints, winPoints 는 0 값이 너무 많아 feature 제거를 해주었다.

In [ ]:
```
df_wrangle.drop(columns = ['rankPoints'], inplace=True)
df_wrangle.drop(columns = ['killPoints'], inplace=True)
df_wrangle.drop(columns = ['winPoints'], inplace=True)
```

→ 이벤트 모드는 수도 적고 경기 진행 방식이 매우 달라 제거를 해주었다.

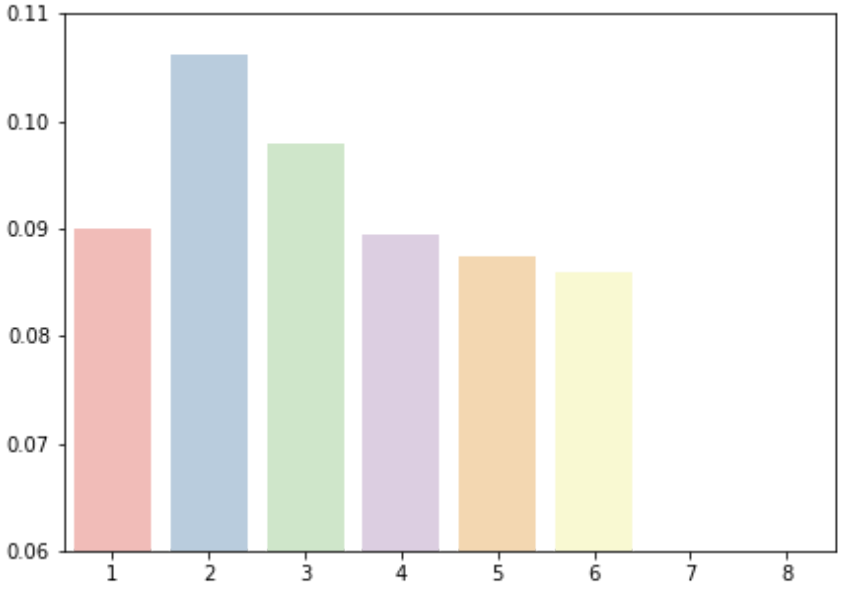



In [ ]:
```
df_wrangle = df_wrangle[df_wrangle.matchType!='flarefpp']
df_wrangle = df_wrangle[df_wrangle.matchType!='flaretpp']
df_wrangle = df_wrangle[df_wrangle.matchType!='crashfpp']
df_wrangle = df_wrangle[df_wrangle.matchType!='crashtpp']
df_wrangle = df_wrangle[df_wrangle.matchType!='normal-squad-fpp']
df_wrangle = df_wrangle[df_wrangle.matchType!='normal-solo-fpp']
df_wrangle = df_wrangle[df_wrangle.matchType!='normal-duo-fpp']
df_wrangle = df_wrangle[df_wrangle.matchType!='normal-squad']
df_wrangle = df_wrangle[df_wrangle.matchType!='normal-solo']
df_wrangle = df_wrangle[df_wrangle.matchType!='normal-duo']
```

→ flare-fpp, flare-tpp, crash-fpp, crash-tpp, normal-squad-fpp, normal-duo-fpp, normal-solo-fpp, normal-squad, normal-duo, normal-solo 총 10가지 모드를 제거해주었다.

**(3) Errors**

Data Frame #6 보다 낮은0.08596319969235391 의 에러 rate 가 나왔다.



### 2-7. Data Frame #7

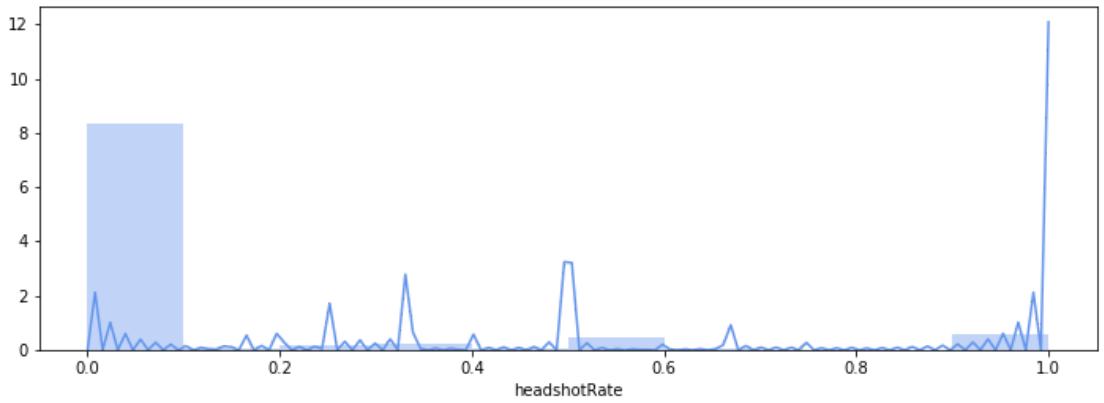
#### (1) Feature Generation

→ headshotRate, killPlacePerc

headshotRate 은 플레이어의 에임 정확도를 알 수 있다고 생각하여 추가해 주었고, killPlacePerc 은 단순히 킬 순위보다, 전체 팀수에 대한 비율이 더 의미있다고 생각하여 추가해주었다.

In [ ]:
```
#create headshot_rate feature
df_wrangle['headshotRate'] = df_wrangle['headshotKills']/df_wrangle['kills']
df_wrangle['headshotRate'] = df_wrangle['headshotRate'].fillna(0)
```

In [ ]:
```
#create killPlacePerc
df_wrangle['killPlacePerc'] = df_wrangle['killPlace'] / df_wrangle['maxPlace']
df_wrangle['killPlacePerc'] = df_wrangle['killPlacePerc'].fillna(0)
```




: totalDistance_over_{healsandboosts, kills, damage, DBNOS, weapons}

플레이어가 가만히 있는 것보다 많이 이동하면 그만큼 다른 팀을 만날 가능성도 증가하고, 더 많은 아이템을 주울 수 있다. 총 이동거리에 대한 feature 들을 추가해주었다.

In [57]:
```
wrangled['totalDistance_over_heals'] = wrangled['totalDistance'] / wrangled['heals']
wrangled['totalDistance_over_heals'].fillna(0, inplace=True)
wrangled['totalDistance_over_heals'].replace(np.inf, 0, inplace=True)
```

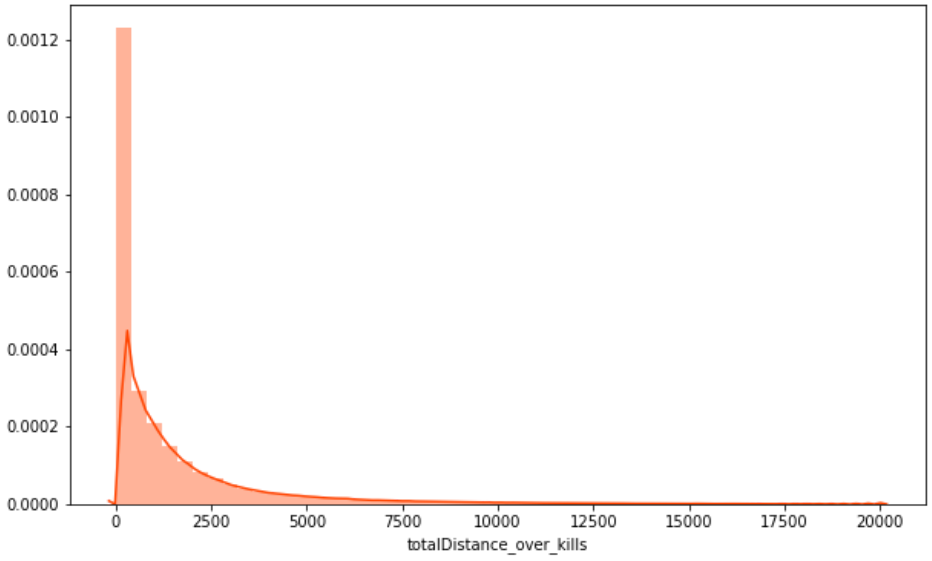
In [58]:
```
wrangled['totalDistance_over_boosts'] = wrangled['totalDistance'] / wrangled['boosts']
wrangled['totalDistance_over_boosts'].fillna(0, inplace=True)
wrangled['totalDistance_over_boosts'].replace(np.inf, 0, inplace=True)
```

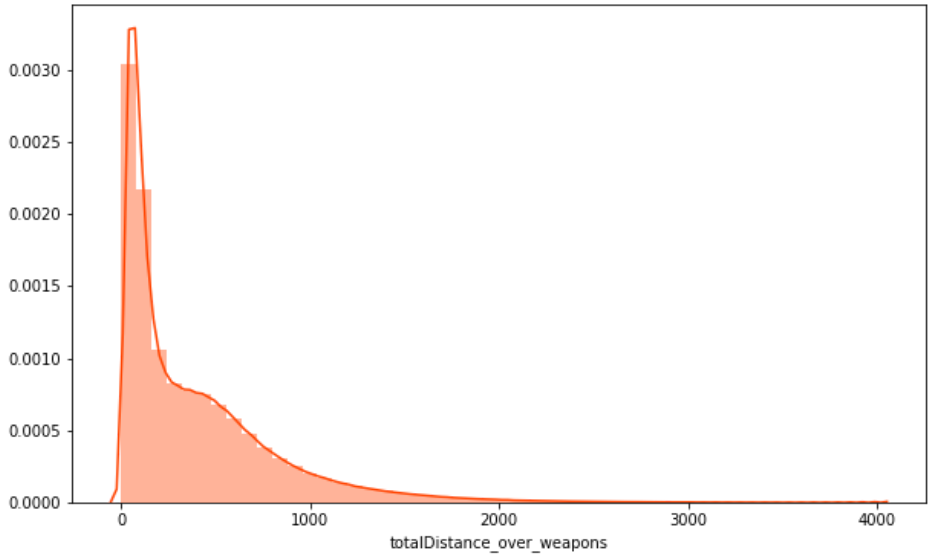
In [59]:
```
wrangled['totalDistance_over_kills'] = wrangled['totalDistance'] / wrangled['kills']
wrangled['totalDistance_over_kills'].fillna(0, inplace=True)
wrangled['totalDistance_over_kills'].replace(np.inf, 0, inplace=True)
```

In [60]:
```
wrangled['totalDistance_over_damage'] = wrangled['totalDistance'] / wrangled['damageDealt']
wrangled['totalDistance_over_damage'].fillna(0, inplace=True)
wrangled['totalDistance_over_damage'].replace(np.inf, 0, inplace=True)
```

In [70]:
```
wrangled['totalDistance_over_DBNOs'] = wrangled['totalDistance'] / wrangled['DBNOs']
wrangled['totalDistance_over_DBNOs'].fillna(0, inplace=True)
wrangled['totalDistance_over_DBNOs'].replace(np.inf, 0, inplace=True)
```

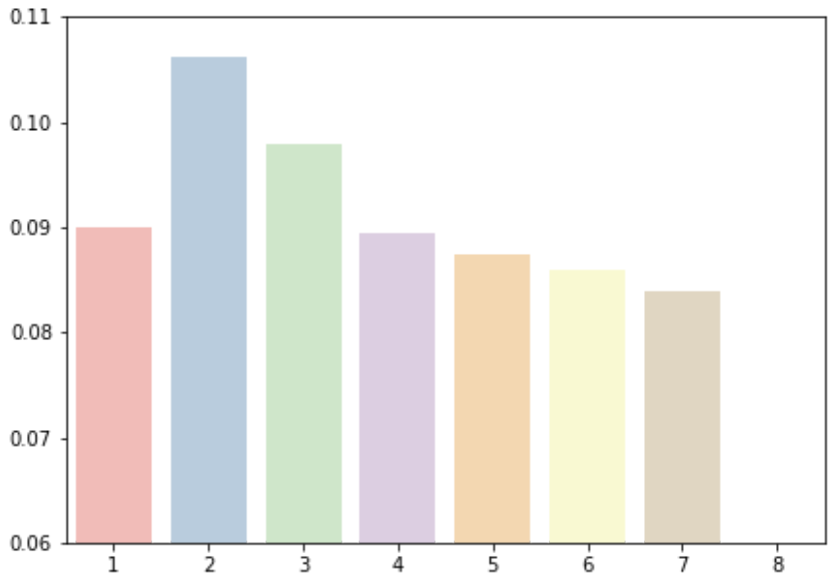
In [72]:
```
wrangled['totalDistance_over_weapons'] = wrangled['totalDistance'] / wrangled['weaponsAcquired']
wrangled.fillna(0, inplace=True)
wrangled.replace(np.inf, 0, inplace=True)
```



**(2) Errors**

Data Frame #6 보다 낮은 0.08390298049343939 의 에러 rate 가 나왔다.




## 2-8. Data Frame #8

### (1) Feature Generation : groupBy

듀오, 스쿼드는 2인, 4인으로 이루어진 팀 경기이다. 한 팀원이 먼저 죽어도 다른 팀원이 잘하면 높은 등수를 받을 수 있다. 개인의 성적( 킬 수, 이동 거리 등) 보다 팀의 성적이 더 중요하다고 생각했다. 팀의 최고 성적, 팀의 평균성적, 팀의 성적의 합을 kills, DBNOs, damageDealt, assists, headshotKills, heals, boosts, distance, weaponsAquired 등에 대하여 구하고 feature 로 추가해 주었다.

In [39]:
```
def get_max(df, max_of, to):
    return df.merge(df.groupby(['groupId'])[max_of].sum().to_frame(to), how='left', on=['groupId'])

def get_mean(df, mean_of, to):
    return df.merge(df.groupby(['groupId'])[mean_of].sum().to_frame(to), how='left', on=['groupId'])

def get_sum(df, sum_of, to):
    return df.merge(df.groupby(['groupId'])[sum_of].sum().to_frame(to), how='left', on=['groupId'])
```

In [41]:
```
# 팀 전체 킬 수
wrangled = get_sum(wrangled, 'kills', 'total_kills')
# 팀 최고 킬 수
wrangled = get_max(wrangled, 'kills', 'max_kills')
```

In [42]:
```
# 팀 전체 기절 수
wrangled = get_sum(wrangled, 'DBNOs', 'total_DBNOs')
# 팀 최고 기절 수
wrangled = get_max(wrangled, 'DBNOs', 'max_DBNOs')
```

In [43]:
```
# 팀 전체 데미지량
wrangled = get_sum(wrangled, 'damageDealt', 'total_damage')
# 팀 최고 데미지량
wrangled = get_max(wrangled, 'damageDealt', 'max_damage')
```

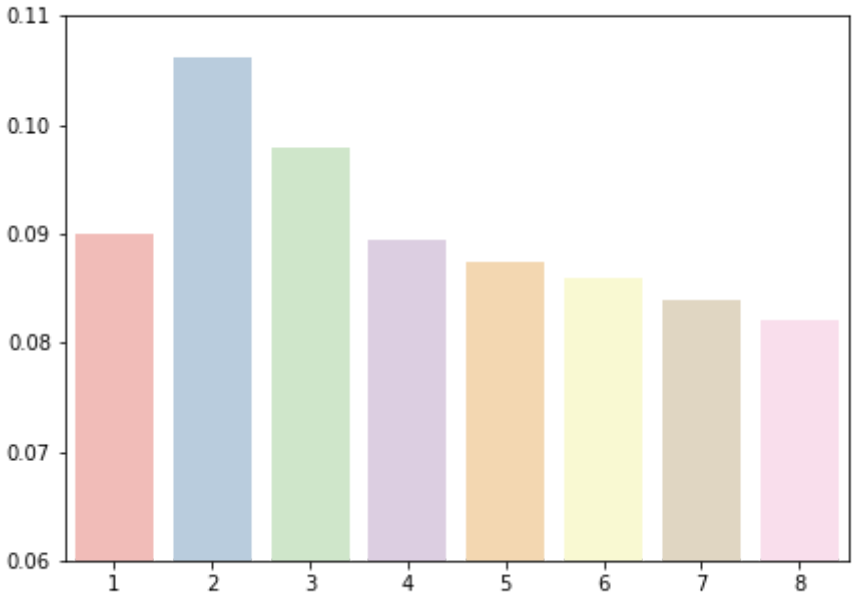
In [44]:
```
# 팀 전체 어시스트 수
wrangled = get_sum(wrangled, 'assists', 'total_assists')
# 팀 평균 어시스트 수
wrangled = get_mean(wrangled, 'assists', 'avg_assists')
```

In [45]:
```
# 팀 최고 헤드샷
wrangled = get_max(wrangled, 'headshotKills', 'max_headshot')
```

In [46]:
```
# 팀 최고 kill place
wrangled = get_max(wrangled, 'killPlace', 'max_killplace')
```

**(2) Errors**

Data Frame #6 보다 낮은 0.0819940040822313 의 에러 rate 가 나왔다.



## 3. Wrangling Conclusion

### 3-1. Final Data Frame

Remove hacker data (Anomaly Detection)

→ 과도하게 높은 킬 수나 이동거리, 너무 높은 헤드샷 정확도 등은 불법 프로그램 이용자라고 생각하여 데이터에서 제거하였으나, 오류가 증가하였다. 오류가 증가한 이유를 데이터의 손실 때문이라고 판단하여 제거하지 않았다.

**Data Normalization**

→ 같은 킬수여도 총 플레이어 수가 100명일때 10킬을 하는 것과 50명일때 10킬을 하는 것은 다르다. 두 상황에 대해 차별을 두기 위해 총 플레이어수에 따라 달라질 수 있는 kills, damageDealt 등의 feature 들을 정규화 시켜주었다.

**Feature Generation**

→ heashotKill 자체 보다 kill을 나누어준 headshotKillRate 이 플레이어가 얼마나 높은 정확도를 가지고 있는지 나타내 줄 수있다. 또한 비슷한 기능을 하는 heals 와 boosts를 합친 healsandboosts 가 더 큰 영향을 미치기도 한다. 여러 feature 들을 조합해 더 의미있는 feature 들을 만들어주었다.

Feature Removal - based on Feature Correlation

→ 높은 correlation을 가진 feature 들은 비슷하다고 판단해 그 중 가장 영향력이 높은 feature 만 남기고 제거해주었었다. 하지만 모든 feature 을 사용하는 것이 오류가 더 낮아 제거해주지 않았다.

Feature Selection - based on Feature Importance

→ 원래의 feature과 feature generation 을 통해 생성한 총 feature의 수가 매우 많았다. 이 중 winPlacePerc 에 적은 영향을 주는 feature 들을 빼주면 더 정확하게 예측할 수 있을 것이라고 생각하여 중요한 feature들만 남겨주었다. feature의 importance 별로, 개수 별로 나누어 feature selection 을 해보았지만 모든 feature 을 사용하는 것이 오류가 가장 낮아 feature selection 을 하지 않았다.

**Data Removal - Unimportant Data**

→ 이벤트 모드는 종류가 매우 다양하고, 데이터의 수도 매우 적었다. 이벤트 모드는 모드별로 게임 규칙, 우승 조건 등이 매우 달라서 일일이 고려할 수 없어 데이터에서 제외를 시켜주었다.

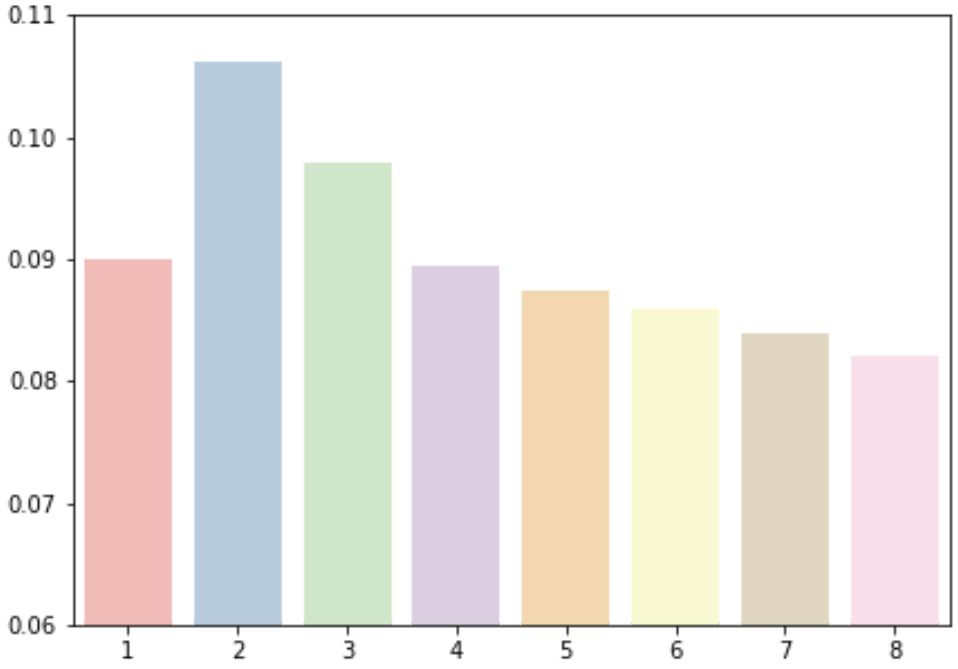
**Group data**

→ 듀오나 스쿼드 같은 경우에는 개인 순위가 아니라 팀 순위를 받는다. 그렇기 때문에 플레이어 1이 먼저 죽어도 플레이어 2가 끝까지 살아남으면 플레이어 1, 2 모두 1등을 받게 된다. 팀의 성적도 중요하다고 판단해 팀의 최고 점수, 평균, 합 등을 구한 feature 을 추가해 주었다.

### 3-2. Errors

Data Frame #2가 0.1062083416871684으로 에러가 제일 높게 나왔고, Data Frame #8이 에러가 0.0819940040822313으로 제일 낮게 나와 Data Frame #8을 최종 data frame 으로 선택했다.

| | MAE Train | MAE Validation |
|---|---|---|
| **Data Frame #1** | 0.0900413873170658 | 0.09004087988062638 |
| **Data Frame #2** | 0.10613482548194682 | 0.1062083416871684 |
| **Data Frame #3** | 0.09791443528865511 | 0.09786979593273051 |
| **Data Frame #4** | 0.08887464204521152 | 0.08934394580354478 |
| **Data Frame #5** | 0.08725870042000705 | 0.0873693458927027 |
| **Data Frame #6** | 0.08606482442274331 | 0.08596319969235391 |
| **Data Frame #7** | 0.08422211741981636 | 0.08390298049343939 |
| **Data Frame #8** | 0.08182221428860487 | 0.0819940040822313 |



## III. Model Selection

### 1. Linear Regression

#### 1-1. Linear Regression 모델 설명

→ x가 input이고 y가 output일 때 y=ax+b 라는 식으로 새로운 x에 대해 y를 예측하고자 한다. 따라서 학습을 통해 적절한 a와 b를 정한다.

→실제 y값과 ax+b의 차이가 에러가 된다.

→ 간단하면서도 직관적인 머신러닝 모델이므로 linear regression 모델을 baseline으로 선정했다.

### 2. Random Forest

```
m = RandomForestRegressor(n_estimators=200, max_features=0.5, n_jobs=-1)
m.fit(X_train, y_train)
print_score(m)
```

#### 2-1. Random Forest 모델 설명

→ 랜덤포레스트는 분류, 회귀 분석 등에 사용되는 앙상블 학습 방법의 일종으로, 훈련과정에서 구성한 다수의 decision tree 로부터 분류 또는 평균 예측치를 출력함으로써 동작한다.

→ random forest 방법을 사용하지 않고 decision tree를 만들면 overfitting 되기 쉽다. 따라서 random forest는 랜덤성을 추가시켜서 overfitiing을 방지하면서 decision tree를 생성한다.

→ 랜덤성을 부여하는 방식은 크게 2가지 방법이 있다. 첫번째는 트리를 만들 때 사용 하는 데이터를 랜덤으로 선택하는 방법이고 두번째는 트리를 만들 때 사용하는 특성을 랜덤으로 선택하는 방법이다.

→ 여러개의 decision tree를 만들고, 투표를 시켜 다수결로 최종 결과를 결정하는 방식이다.

#### 2-2. Random Forest의 hyperparameter tuning

**(1) 주요 hyperparameter**

→ n_estimators : 생성할 트리의 개수

→ max_features : 최대 선택할 특성의 수

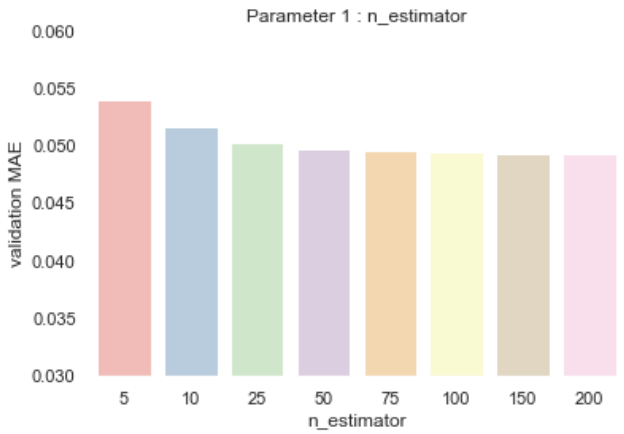
max_features 의 값을 크게 하면 random forest의 트리들은 같은 특성을 고려하므로 트리들이 매우 비슷해지고 가장 두드러진 특성을 이용해 데이터에 잘 맞는 트리를 구성한다.

max_features 의 값을 작게 하면 트리들이 많이 달라지고 각 트리는 데이터에 맞추기 위해 트리의 깊이가 싶어진다.

**(2) hyperparameter tuning**

데이터에 대해 가장 적절한 hyperparameter 를 찾아내기 위해 max_featrues를 고정한 상태로 여러 n_estimators 값에 대해 validation 성능을 측정하였고, 반대로 n_estimators 값을 고정 시킨 상태에서 여러 max_features 값에 대한 validation 성능을 측정해보았다.

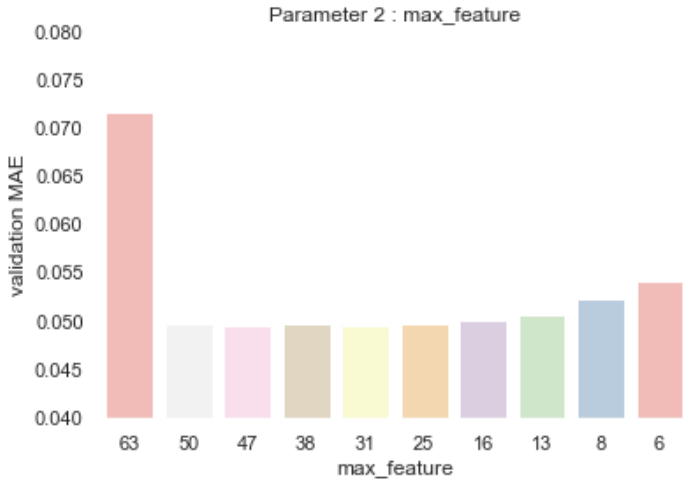
→ n_estimators




| n_estimator | MAE |
|---|---|
| 5 | 0.054022 |
| 10 | 0.051616 |
| 25 | 0.050335 |
| 50 | 0.049701 |
| 75 | 0.049578 |
| 100 | 0.049504 |
| 150 | 0.049373 |
| **200** | **0.049259** |

n_estimators값이 증가할 수록 아주 작은 변화이지만 validation 성능이 점점 증가하는 것을 확인할 수 있었다.

→ max_features




| max_feature | MAE |
|---|---|
| 1 (63개) | 0.071564 |
| 0.8 (50개) | 0.049805 |
| 0.75 (47개) | 0.049636 |
| 0.6 (38개) | 0.049665 |
| **0.5 (31개)** | **0.049562** |
| 0.4 (25개) | 0.049789 |
| 0.25 (16개) | 0.050057 |
| 0.2 (13개) | 0.050675 |
| sqrt (8개) | 0.052251 |
| log2 (6개) | 0.054085 |

max_features 가 1일 때, 즉 특성 선택에 랜덤성이 전혀 없을 때 에러가 현저히 높은 것을 확인 할 수 있었다. 또한 max_features는 값이 작아질수록 validation 성능이 좋아지다가 어느 부분을 기준으로 다시 validation 성능이 나빠지는 것을 확인 하였다.

각 경우 최적의 성능을 냈던 경우를 합쳐서 validation 성능을 확인했을 때, 즉 n_estimators=200, max_featrues=0.5 일때 MAE값은 0.049271 이었다.

### 3. Extreme Gradient Boosting

```
m = xgboost.XGBRegressor(random_state=42,n_estimators=400, subsample = 0.8, colsample_bytree=1,max_depth=7, learning_rate=0.08)
m.fit(X_train, y_train)
print_score(m)
```

#### 3-1. Extreme Gradient Boosting 모델 설명

→ XGboost 모델은 여러 결정 트리를 묶어 강력한 모델을 만드는 또 다른 방법이다. 하지만 random forest와는 달리 이전 트리의 오차를 보완하는 방식으로 순차적으로 트리를 생성한다.

→ XGBoost 모델에는 무작위성이 없는 대신, 강력한 사전 가지치기가 사용된다. 보통 깊지 않은 트리를 사용하므로 메모리를 적게 사용하고 예측도 빠르다.
→ XGBoost 모델의 근본 아이디어는 약한 학습기 (weak learner)을 많이 연결하는 것이다. 각각의 트리는 데이터의 일부에 대해서만 예측을 잘 수행할 수 있으므로, 트리가 많이 추가될수록 성능이 좋아진다.
→ Greedy-algorithm을 사용하여 자동 가지치기가 가능하기에, Overfitting이 잘 일어나지 않는다.

### 3-2. Extreme Gradient Boosting hyperparameter tuning

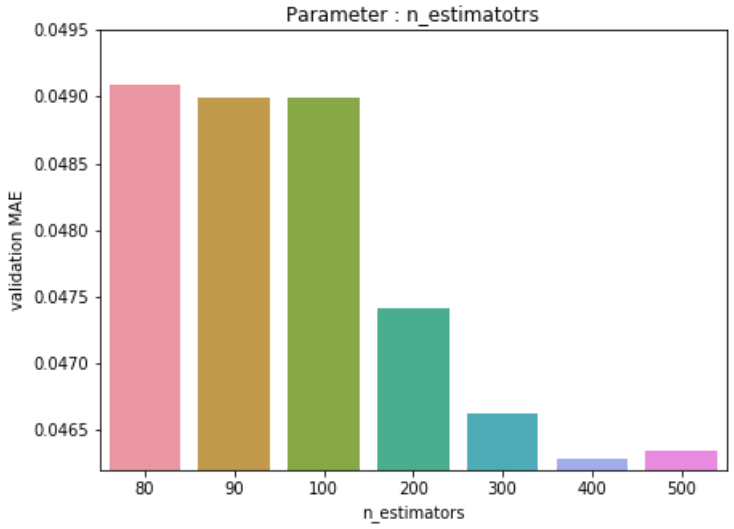
#### (1) 주요 hyperparameter

→ n_estimators: 트리 개수 지정
→ learning rate: 학습률
→ gamma: 트리에서 가지를 추가로 치기 위해 필요한 최고산의 손실 감소 기준. 기준값이 클수록 모델이 더 단순해진다. (>0)
→ max_depth: 트리의 최대 깊이
→ min_child_weight: 트리에서 가지를 추가로 치기 위해 필요한 최소한의 사례 수
→ colsample_bytree: 각각의 트리를 만들 때 사용할 column의 비율
→subsample: 각각의 트리를 만들 때 사용할 sample의 비율

.

#### (2) hyperparameter tuning

Extreme Gradient Boosting의 주요한 hyperparameter를 tuning하기 위해 validation 과정을 여러 차례 진행했다. 40만개의 데이터중 90%의 데이터를 학습에 사용했고, 10%의 데이터를 hyperparameter 검증에 사용했다. 다양한 hyperparameter tuning 과정을 진행했지만, 본 보고서에서는 다양한 hyperparameter중 n_estimators과 subsample의 validation 결과를 다루었다.
→ n_estimators




| n_estimators | val MAE |
|---|---|
| 80 | 0.049087 |
| 90 | 0.048997 |
| 100 | 0.048987 |
| 200 | 0.047416 |
| 300 | 0.046624 |
| **400** | **0.046286** |
| 500 | 0.046341 |

n_estimators 값이 증가할수록 성능이 좋아지는 경향을 보였으며, 최적의 값은 400임을 확인했다.

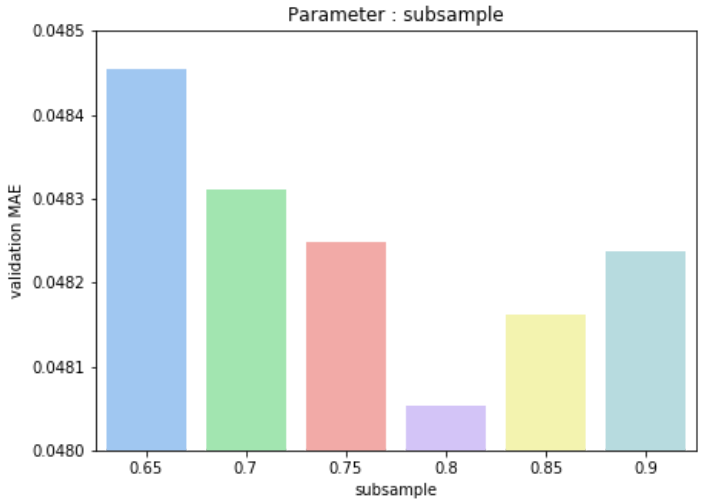
→ subsample
subsample 값은 0.8 이전까지는 error가 감소했지만, 0.8 이후에는 error가 증가하는 경향을 확인할 수 있었다.

Hyperparameter tuning 과정에서 얻은 최적의 hyperparameters는 **random_state=42, n_estimators=400, subsample = 0.8, colsample_bytree=1, max_depth=7, learning_rate=0.08** 이었으며, MAE는 **0.0460962849633646** 이었다.

### 4. Final Model Selection

Linear regression 모델, Random Forest 모델, XGBoost 모델 중 가장 좋은 성능을 보인 XGBoost 모델을 선택하였다. XGBoost 모델을 train하기 위해 360,0000개의 데이터를 사용했고, test하기 위해 4,0000개의 데이터를 사용했다.

## IV. 결론




| subsample | val MAE |
|---|---|
| 0.65 | 0.048454 |
| 0.7 | 0.048311 |
| 0.75 | 0.048248 |
| **0.8** | **0.048054** |
| 0.85 | 0.048162 |
| 0.9 | 0.048237 |

### 1. Model Prediction Accuracy

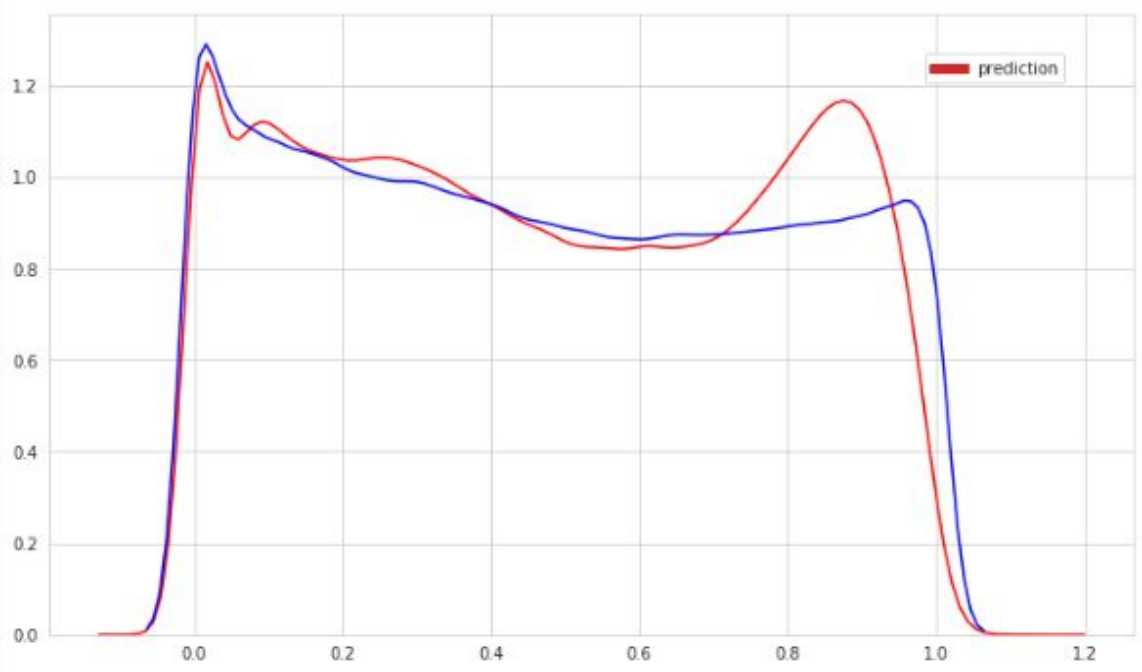
모델의 정확도를 평가하기 위해, MAE와 visualization을 사용했다.

#### 1-1. MAE

모델이 예측한 결과의 MAE는 0.045326012533503 42였다.

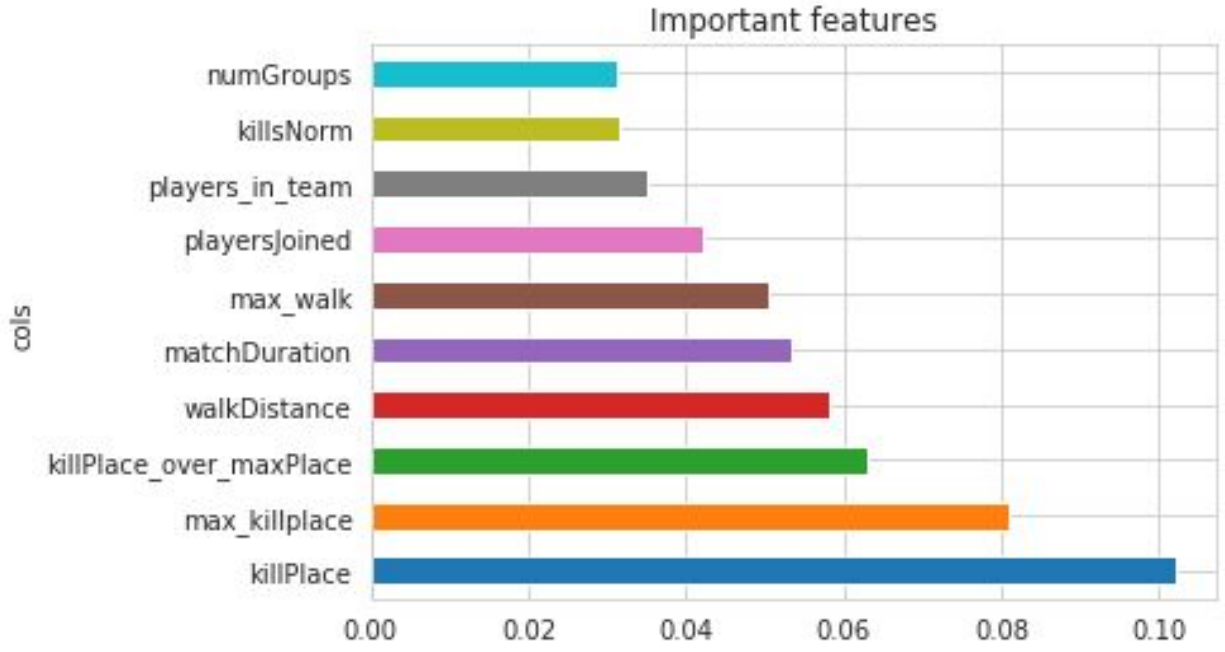
#### 1-2. visualization

최종적으로 선택한 모델이 예측한 결과와 실제 값의 분포를 시각화하였다. 붉은 선은 모델이 예측한 결과의 분포를 나타내며, 푸른 선은 실제 값의 분포를 나타낸다.




**2. Result Analysis**

결과를 예측하는데 중요하게 사용된 10개의 feature들은 다음과 같다.




Kill과 walk distance에 관련된 feature가 많이 선택되었으며, ‘groups’에 기반하여 생성한 새로운 feature들 또한 강력한 feature로 선택되었음을 확인할 수 있다.

## 3. Conclusion

### 3-1. Hypothesis 검증

해당 프로젝트를 시작할 때 세웠던 가설은 다음과 같다.

(1) **플레이어의 숙련도**를 나타내는 **kills, DBNOs, damageDealt** features이 winPlacePerc을 결정하는데에 결정적인 영향을 끼칠 것이다.
(2) **팀원간의 협동심**을 나타내는 **Assists, revives, team size** features이 winPlacePerc을 결정하는데에 결정적인 영향을 끼칠 것이다.
(3) 플레이어가 **소유하고 있는 아이템**을 나타내는 **heals, boosts, weaponsAcquired**가 높은 플레이어의 경우 높은 순위에 안착할 것이다.

모델을 통한 실험 결과, winPlacePerc를 예측하는데 결정적인 영향을 끼친 feature들은 **Kill, walk distance**, 그리고 **groups에 기반하여 생성**된 feature들이었다.

해당 결과를 통해 앞서 세운 가설들을 평가해보았다.

(1) 플레이어의 숙련도는 결과를 예측하는데 중요한 영향을 끼쳤음을 확인했지만, **kills과 관련된 feature**들이 예측했던 DBNOs, damageDealt feature들보다 더 많은 영향력을 가진다.
(2) 팀과 관련된 feature들은 결과를 예측하는데 중요한 영향을 가진다는 사실을 확인했다. 하지만, 가설에서는 Assits, revives, team size와 같이 팀의 협동심을 나타내는 기존의 feature들의 영향력이 높을 것이라 예상했던 반면, 모델링 결과 **팀의 성적과 관련된 feature**들이 영향력이 높았다.
(3) 플레이어가 소유하고 있는 아이템과 관련된 feature들보다 플레이어가 걸은 거리를 나타내는 **walk distance와 관련된 feature**들이 결과를 예측하는데 더 중요한 영향을 끼쳤다.

### 3-2. Data preprocessing 검증

데이터의 전처리 없이 baseline model인 linear regression model로 얻은 MAE는 0.09004087988062638 였고, 전처리 후 linear regression model이 얻은 MAE는 0.08199400408223l3였다. 이를 통해, **데이터 전처리를 통한 모델의 성능 향상**을 확인했다.

### 3-3. Model selection 검증

최종 모델인 XGBoost model이 얻은 MAE는 0.04532601253350342였다. 즉, 실제 등수 차이는 100등까지 있을 때 약 4등의 차이가 발생하고, 스쿼드의 경우 1등의 차이가 발생함을 의미한다. 또한, 최종 모델의 결과를 baseline 모델 결과와 비교했을 때, **적절한 모델의 선택을 통해 예측 결과가 성공적으로 향상**되었음을 확인했다.

## V. 부록

### 1. 참고문헌

(1) https://www.kaggle.com/c/pubg-finish-placement-prediction/data
(2) https://www.kaggle.com/deffro/eda-is-fun
(3) https://www.kaggle.com/carlolepelaars/pubg-data-exploration-rf-funny-gifs

### 2. Final Source Code

```
# coding: utf-8
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.utils import shuffle

wrangled = pd.read_csv('./train_V2.csv')

# ### Drop row with NaN value for winPlacePerc
wrangled.drop(2744604, inplace=True)

# ### Anomaly

# 움직이지 않고 킬 한 경우
wrangled['killsWithoutMoving'] = (wrangled['kills'] > 0) & (wrangled['walkDistance'] == 0)
wrangled = wrangled.drop(wrangled[wrangled['killsWithoutMoving']==True].index)
wrangled = wrangled.drop(columns=['killsWithoutMoving'])

# Drop columns with 'flare' or'crash' or 'normal' matchtype
wrangled = wrangled[wrangled.matchType!='flarefpp']
wrangled = wrangled[wrangled.matchType!='flaretpp']
wrangled = wrangled[wrangled.matchType!='crashfpp']
wrangled = wrangled[wrangled.matchType!='crashtpp']
wrangled = wrangled[wrangled.matchType!='normal-squad-fpp']
wrangled = wrangled[wrangled.matchType!='normal-duo-fpp']
wrangled = wrangled[wrangled.matchType!='normal-solo-fpp']
wrangled = wrangled[wrangled.matchType!='normal-squad']
wrangled = wrangled[wrangled.matchType!='normal-duo']
wrangled = wrangled[wrangled.matchType!='normal-solo']

# ### Group

def get_max(df, max_of, to):
    return df.merge(df.groupby(['groupId'])[max_of].sum().to_frame(to), how='left',
on=['groupId'])

def get_mean(df, mean_of, to):
    return df.merge(df.groupby(['groupId'])[mean_of].sum().to_frame(to), how='left',
on=['groupId'])

def get_sum(df, sum_of, to):
    return df.merge(df.groupby(['groupId'])[sum_of].sum().to_frame(to), how='left',
on=['groupId'])
```

```
# 팀원 수
wrangled = wrangled.merge(wrangled.groupby(['groupId']).size().to_frame('players_in_team'),
how='left', on=['groupId'])
# 팀 전체 킬 수
wrangled = get_sum(wrangled, 'kills', 'total_kills')
# 팀 최고 킬 수
wrangled = get_max(wrangled, 'kills', 'max_kills')
# 팀 전체 기절 수
wrangled = get_sum(wrangled, 'DBNOs', 'total_DBNOs')
# 팀 최고 기절 수
wrangled = get_max(wrangled, 'DBNOs', 'max_DBNOs')
# 팀 전체 데미지량
wrangled = get_sum(wrangled, 'damageDealt', 'total_damage')
# 팀 최고 데미지량
wrangled = get_max(wrangled, 'damageDealt', 'max_damage')
# 팀 전체 어시스트 수
wrangled = get_sum(wrangled, 'assists', 'total_assists')
# 팀 평균 어시스트 수
wrangled = get_mean(wrangled, 'assists', 'avg_assists')
# 팀 최고 헤드샷
wrangled = get_max(wrangled, 'headshotKills', 'max_headshot')
# 팀 최고 kill place
wrangled = get_max(wrangled, 'killPlace', 'max_killplace')
# 팀 최고 heals & boosts 수
wrangled = get_max(wrangled, 'heals', 'max_heals')
wrangled = get_max(wrangled, 'boosts', 'max_boosts')
# totalDistance
wrangled['totalDistance'] = wrangled['walkDistance'] + wrangled['swimDistance'] +
wrangled['rideDistance']
wrangled = get_max(wrangled, 'totalDistance', 'max_distance')
# 팀 최고 이동거리
wrangled = get_max(wrangled, 'walkDistance', 'max_walk')
wrangled = get_max(wrangled, 'rideDistance', 'max_ride')
wrangled = get_max(wrangled, 'swimDistance', 'max_swim')
# 팀 최고 무기 수
wrangled = get_max(wrangled, 'weaponsAcquired', 'max_weapons')
# 팀 평균 소생 & 팀킬 수
wrangled = get_mean(wrangled, 'revives', 'avg_revives')
wrangled = get_mean(wrangled, 'teamKills', 'avg_teamKills')
wrangled = get_max(wrangled, 'killStreaks', 'max_killStreaks')
wrangled = get_max(wrangled, 'longestKill', 'max_longestKill')
wrangled = get_max(wrangled, 'roadKills', 'max_roadKills')
wrangled = get_max(wrangled, 'vehicleDestroys', 'max_vehicleDestroys')

# ### Create features
# teamplay
wrangled['teamwork'] = wrangled['avg_assists'] + wrangled['avg_revives'] -
wrangled['avg_teamKills']
# duo나 squad인데 인원수가 적은 경우(솔듀오 솔쿼드 듀쿼드 삼쿼드)
wrangled['less'] = 0
wrangled.loc[((wrangled['matchType'] == 'duo') | (wrangled['matchType'] == 'duo-fpp')) &
(wrangled['players_in_team'] < 2), 'less'] = 1
wrangled.loc[((wrangled['matchType'] == 'squad') | (wrangled['matchType'] == 'squad-fpp'))
& (wrangled['players_in_team'] < 4), 'less'] = 1
```

```
# matchType one hot encoding
wrangled = pd.get_dummies(wrangled, columns=['matchType'])
matchType_encoding = wrangled.filter(regex='matchType')

wrangled['headshotKills_over_kills'] = wrangled['max_headshot'] / wrangled['max_kills']
wrangled['headshotKills_over_kills'].fillna(0, inplace=True) # replace NaN value
wrangled['headshotKills_over_kills'].replace(np.inf, 0, inplace=True) # replace infinite
value
wrangled['killPlace_over_maxPlace'] = wrangled['max_killplace'] / wrangled['maxPlace']
wrangled['killPlace_over_maxPlace'].fillna(0, inplace=True)
wrangled['killPlace_over_maxPlace'].replace(np.inf, 0, inplace=True)

wrangled['totalDistance_over_heals'] = wrangled['max_distance'] / wrangled['max_heals']
wrangled['totalDistance_over_heals'].fillna(0, inplace=True)
wrangled['totalDistance_over_heals'].replace(np.inf, 0, inplace=True)

wrangled['totalDistance_over_boosts'] = wrangled['max_distance'] / wrangled['max_boosts']
wrangled['totalDistance_over_boosts'].fillna(0, inplace=True)
wrangled['totalDistance_over_boosts'].replace(np.inf, 0, inplace=True)

wrangled['totalDistance_over_kills'] = wrangled['max_distance'] / wrangled['max_kills']
wrangled['totalDistance_over_kills'].fillna(0, inplace=True)
wrangled['totalDistance_over_kills'].replace(np.inf, 0, inplace=True)

wrangled['totalDistance_over_damage'] = wrangled['max_distance'] / wrangled['max_damage']
wrangled['totalDistance_over_damage'].fillna(0, inplace=True)
wrangled['totalDistance_over_damage'].replace(np.inf, 0, inplace=True)

wrangled['totalDistance_over_DBNOs'] = wrangled['max_distance'] / wrangled['max_DBNOs']
wrangled['totalDistance_over_DBNOs'].fillna(0, inplace=True)
wrangled['totalDistance_over_DBNOs'].replace(np.inf, 0, inplace=True)

wrangled['totalDistance_over_weapons'] = wrangled['max_distance'] / wrangled['max_weapons']
wrangled['totalDistance_over_weapons'].fillna(0, inplace=True)
wrangled['totalDistance_over_weapons'].replace(np.inf, 0, inplace=True)

wrangled['playersJoined'] = wrangled.groupby('matchId')['matchId'].transform('count')

#normalized features
wrangled['killsNorm'] = wrangled['max_kills']*((100-wrangled['playersJoined'])/100 + 1)
wrangled['damageDealtNorm'] = wrangled['max_damage']*((100-wrangled['playersJoined'])/100 +
1)

# ### Drop Features
# ID
wrangled.drop(columns = ['Id'], inplace=True)
wrangled.drop(columns = ['groupId'], inplace=True)
wrangled.drop(columns = ['matchId'], inplace=True)

#0이 많은 column 삭제
wrangled.drop(columns = ['killPoints'], inplace=True)
wrangled.drop(columns = ['rankPoints'], inplace=True)
wrangled.drop(columns = ['winPoints'], inplace=True)
```

```
# ## Model

import sklearn
from sklearn import metrics
from sklearn.metrics import mean_absolute_error
import xgboost

train = wrangled.iloc[:3600000]
test = wrangled.iloc[3600000:]

train_data = train.drop(columns=['winPlacePerc'])
train_target = train['winPlacePerc']
test_data = test.drop(columns=['winPlacePerc'])
test_target = test['winPlacePerc']

model = xgboost.XGBRegressor(random_state=42, n_estimators=400, subsample = 0.8,
colsample_bytree=1,max_depth=7, learning_rate=0.08)
final_model = model.fit(train_data, train_target)
final_result = final_model.predict(test_data)

mean_absolute_error(final_result, test_target)
```