# ◈ 기본사항

- 리그오브레전드(LOL) 게임에서 나의 솔로랭크 전적 데이터

- Attributes: 13개
  - Champion: (Jarvan IV, Sett, Modekaiser, Zac, Malphite, etc.)
  - Lane: (Top, Jungle, Mid, Bot, Support)
  - Team: (Red, Blue) - Win-Lose: (Win, Lose)
  - Playing Time, Kill, Death, Assist, KDA((Kill+Assist)/Death) //Death가 0일시 (KIll+Assist)*1.2
  - Damage, DPM(Damage Per Minute), CS(Creep Score), CS per Min

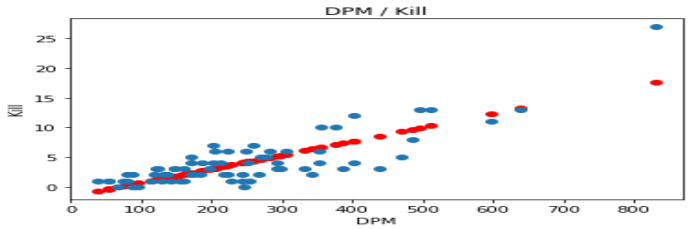
- Instances: 218개
  - Lane별 sample 수: Top(72), Jungle(106), Mid(3), Bot(2), Support(35)
  - Team별 sample 수: Red(114), Blue(104)
  - 승패 별 sample 수: Win(112), lose(106)

- 출처 : https://www.op.gg/summoner/userName=준혁신에서 솔로랭크 게임 데이터 직접 수집

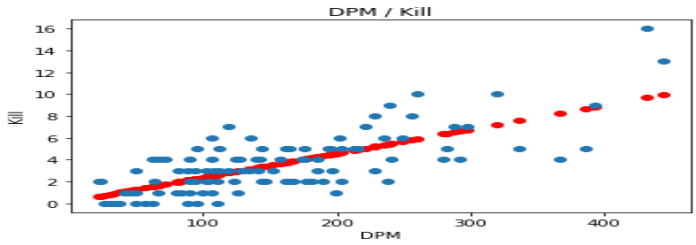
- 분석 방법: Regression / Clustering
  - Linear Regression, Multiple Linear Regression, Logistic Regression
  - K-means Clustering, Hierarchical Clustering

# ◈ Regression

## ◆ Linear Regression

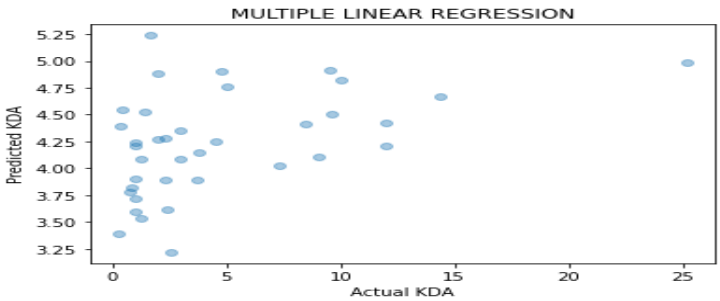



- Lane이 Top일 때의 DPM과 Kill의 데이터이다
- 우상향 하는 패턴을 보여준다
- 기울기는 [0.02325293] 절편은 -1.6304905687983862 이다

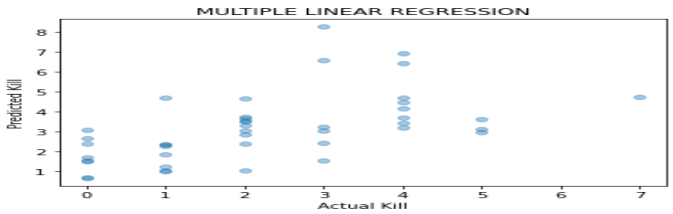



- Lane이 Jungle일 때의 DPM과 Kill의 데이터이다
- Lane이 Top일 때와 마찬가지로 우상향 패턴을 보여준다
- 기울기는 0.02208163 절편은 0.1687557242012252 이다

## ◆ Multiple Linear Regression




- Lane이 Support가 아닐 때 Playing Time, Damage, CS를 통해 KDA를 예측하고자 하였음
- 학습 데이터와 테스트 데이터를 8:2로 분리함
- 예측 값과 실제 값이 일치한다면 정확히 선으로 나타내어 질 것이다
- 예측 KDA가 실제 KDA보다 높은 것으로 보아 잘 못 되었음을 알 수 있음




- Lane이 Support가 아닐 때 Playing Time, Damage, CS를 통해 Kill을 예측하고자 하였음
- 학습 데이터와 테스트 데이터를 8:2로 분리함
- KDA를 예측했을 때 보다 정확한 값이 나옴

```
virtual_data = [[25, 20000, 180]]
predict_data = mrg.predict(virtual_data)
print(predict_data)
```

```
[[7.50960547]]
```

- Playing Time, Damage, CS를 25, 20000, 180으로 가정했을 때 Kill은 7.5로 예측되었다

### ◆ Rogistic Regression

```
df['Win-Lose'] = df['Win-Lose'].replace(['Win', 'Lose'], [1, 0])
```

- Win-Lose 열에서 win을 1 Lose를 0으로 변경함
- 승패에 관련이 있을 것이라 예상되는 KDA, DPM, CS를 선택함
- 학습 데이터와 테스트 데이터를 7.5:2.5로 분리함

```
print(Lor.score(train_feats, train_wl))
```

```
0.8220858895705522
```

- 학습 세트의 정확도는 약 82% 정도로 나타남

```
print(Lor.coef_)
```

```
[[2.82457771 0.18624795 0.05352452]]
```

- 각 feature들의 계수를 보면 KDA가 승패에 가장 큰 요인임을 알 수 있음

```
data1 = np.array([4, 400, 200])
data2 = np.array([3.0, 300, 200])
data3 = np.array([2.5, 400, 220])
sample_datas = np.array([data1, data2, data3])
sample_datas = scaler.transform(sample_datas)
print(Lor.predict(sample_datas))
print(Lor.predict_proba(sample_datas))
```

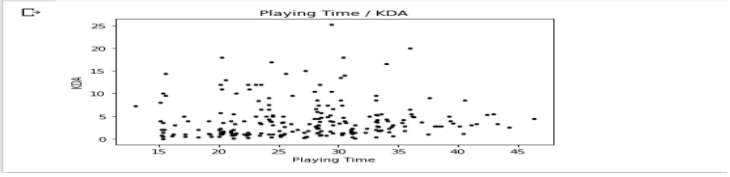
```
[1 1 0]
[[0.27996585 0.72003415]
 [0.48735249 0.51264751]
 [0.53627368 0.46372632]]
```

- 3개의 데이터를 가정해 추측해 본 결과 승, 승, 패 로 결과가 나왔다

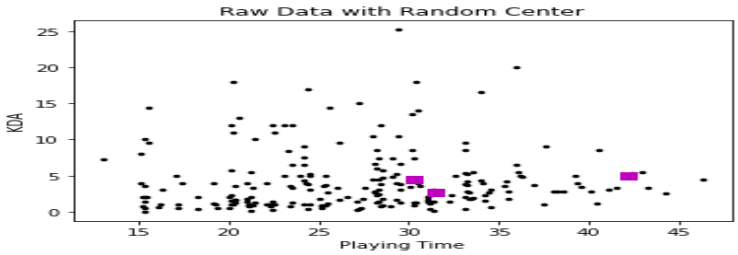
# ◈ Clustering

## ◆ K-means

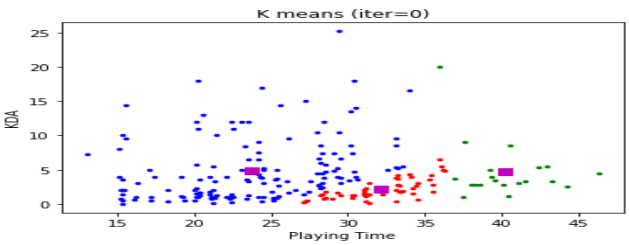
```
# Clustering
# 데이터 로드, 산점도 그래프
X = df.iloc[:,[4,8]].values
plt.scatter(X[:,0], X[:,1], c='black', s=7, label='LOL Data')
plt.xlabel('Playing Time')
plt.ylabel('KDA')
plt.title('Playing Time / KDA')
plt.show()
```

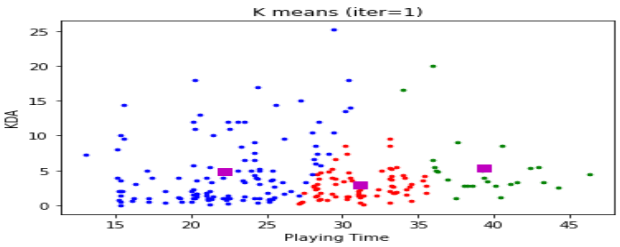


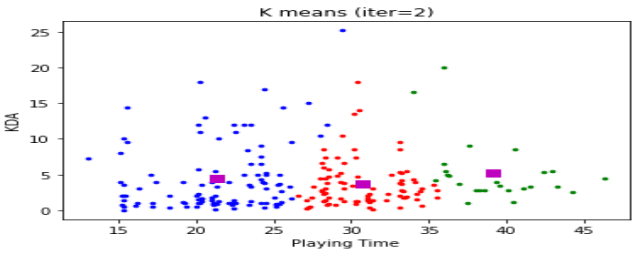

- 게임시간에 따라 KDA가 어떻게 나뉘는지 알고 싶어 x축은 Playing Time, y축은 KDA로 설정함
- 데이터를 로드하고 산점도 그래프를  그려보니 위와 같은 모양을 나타냈다

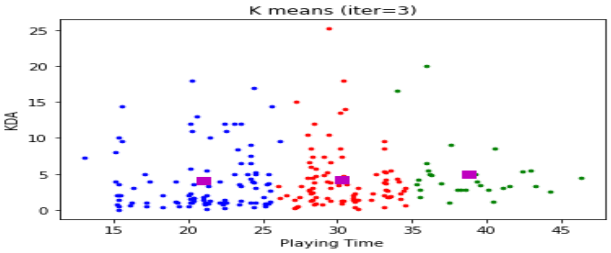


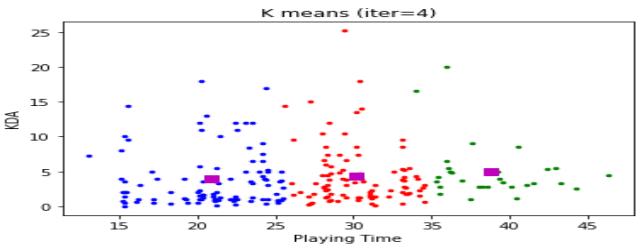

- 군집의 개수는 3개로 설정
- 초기 대표벡터 3개를 랜덤으로 설정함.



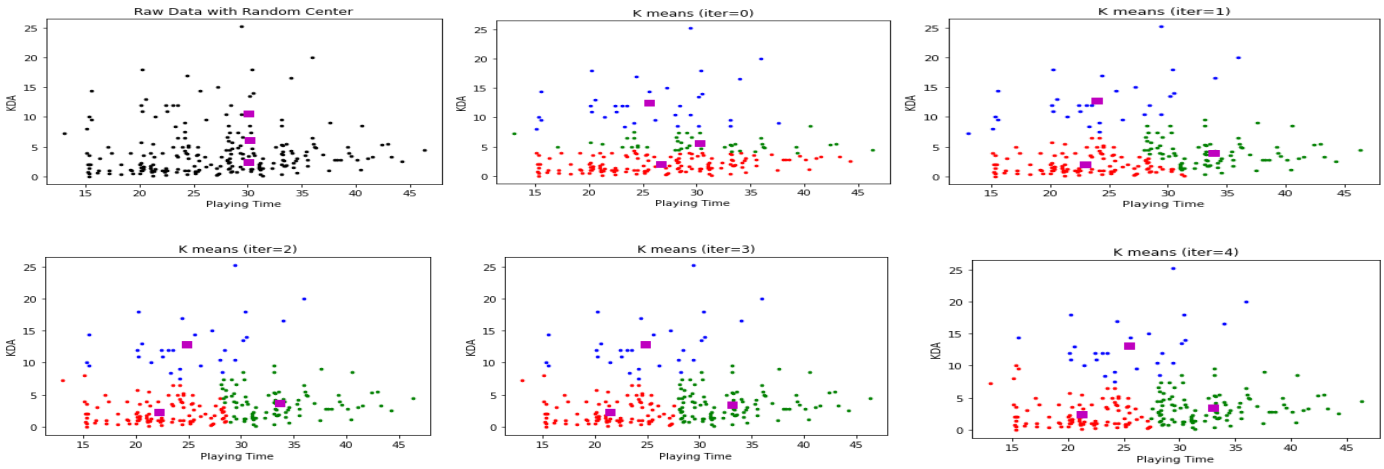



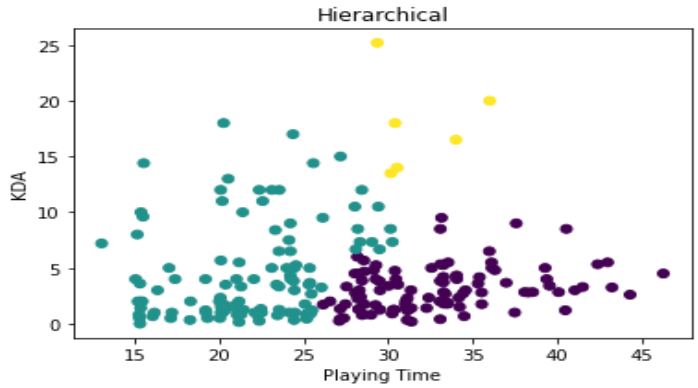




- 반복횟수는 5번으로 설정
- 처음의 클러스터링의 모양은 제대로 되지 않음
- 반복이 진행될수록 모양을 잡아가고 있다
- 계속 반복을 진행하니 수렴해서 대표벡터가 움직이지 않는다

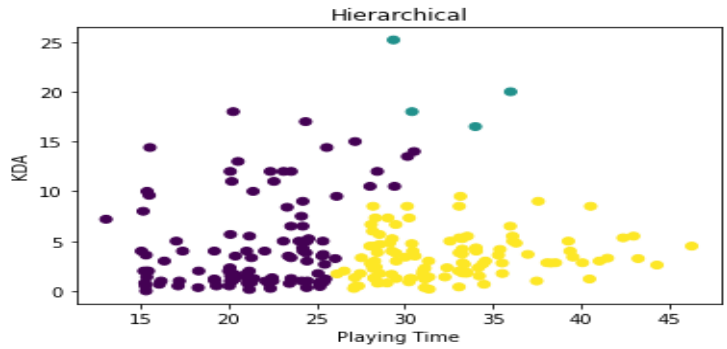



- 초기 대표벡터를 다르게 설정하니 전혀 다르게 군집화가 되었다
- 초기 대표벡터를 어떻게 설정하느냐가 중요함을 알 수 있음

### ◆Hierarchical Clustering




- Hierarchical Clustering을 통해 군집화한 모양이다
- 완전 거리를 이용함
- K-means와는 조금 차이가 있는 모양을 보여준다




- 평균 거리를 이용함
- 완전거리를 이용해 군집화 한 것보다 군집화가 잘 되지 않은 것으로 보인다