

프로젝트 결과보고

목표 및 주제

한국에서 활동하는 외국인 선수의 정보를 분석한다.
그 방법으로 활동하는 외국인 선수의 MLB 시절 데이터와 현재 KBO 데이터의 연관성을 찾아 Decision Tree 를 제작한다.
이를 통해 MLB선수의 정보를 입력받아 KBO영입 시 수준을 예측해주는 웹 서비스를 구축한다.
MLB선수에 대한 정보는 이름만 입력받고 이를 기준으로 데이터를 수집하고, 전처리하고, 분석하여 결과를 출력하는 과정을 자동화하여 간편한 서비스를 제공한다.

이는 선수 스카우트를 할 때에 유용하게 사용될 것으로 생각된다.




조원 소개

1. 장경석 (조장)
데이터 수집, 전처리, 예측
2. 장지승
전처리, 분석, 예측
3. 김수연
데이터 수집, 시각화, 분석

프로젝트 진행

※ 데이터수집

2011~2019년도 야구선수에 대한 데이터는 csv파일 형태의 데이터를 사용했다.

 baseball_savant_foreigners_2011_201...	2019-12-17 오후 8:05
 fangraphs_foreigners_2011_2019.csv	2019-12-16 오후 8:42
 kbo_yearly_foreigners_2011_2019.csv	2019-12-16 오후 8:42

이는 야구선수의 MLB시절 스탯데이터, 던진 공에대한 데이터를 담고있고 마지막으로 KBO에서의 활약을 담고있는 데이터이다.

분석을 할 데이터는 2011~2019년도 데이터를 사용하였고
예측할 데이터는 selenium을 기반으로 한 웹크롤링으로 직접 추출했다.

```
# 페이지 소스에서 실질적으로 사용할 값만 사용하고 저장하기 위한 dic 선언
data_list={
    'team': [], 'year': [], 'BABIP': [], 'ERA': [], 'WAR': [], 'TBF': [], 'H': [], 'HR': [], 'BB': [], 'HBP': [], 'SO': [], 'WHIP': [], 'FIP': [],
    'LD': [], 'GB': [], 'FB': [], 'IFFB': [], 'Swing': [], 'SWStr': []
}

# 각 테이블별로 원하는 정보를 추출해올
# for 문 내부의 알고리즘설명은 생략

bodys=contents[0].select('tbody')
for body in bodys:
    trs=body.select('tr')
    for tr in trs[:-2]:
        tds=tr.select('td')
        season=tds[0].text
        team=tds[1].text

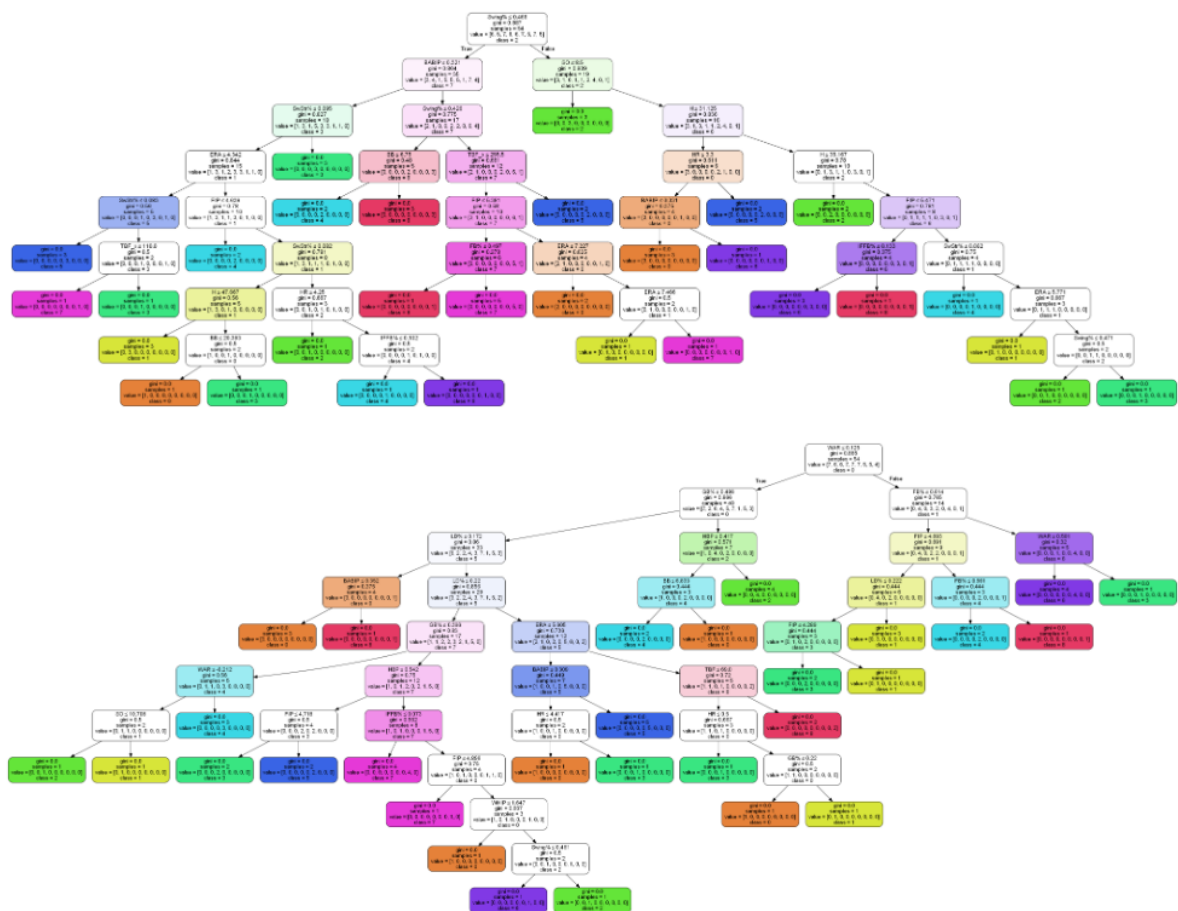
        year=tds[0].text
        BABIP=tds[11].text
        ERA=tds[15].text
        WAR=tds[18].text
        if comp.search(str(tr))!=None and comp3.search(str(tr))!=None and season!='2020' and season!='Total':
            if comp2.search(team)!=None:
                data_list['year'].append(year)
                data_list['BABIP'].append(BABIP)
                data_list['ERA'].append(ERA)
                data_list['WAR'].append(WAR)
                data_list['team'].append(team)
```

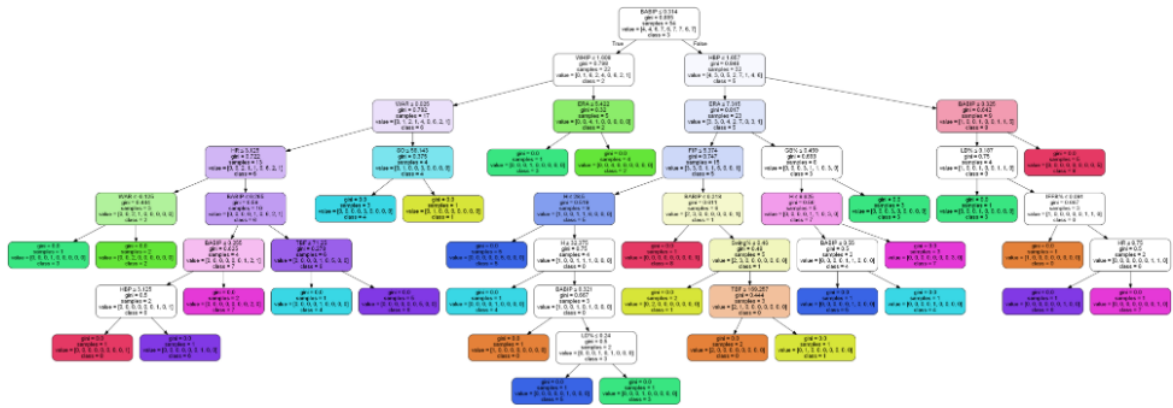
이와같은 알고리즘을 사용했으며 본문 내용이 길어 일부 생략했다.

데이터 예측에 필요한 컬럼을 Dic 형태로 미리 선언을 하여 필요한 값만 받았다.

이에 pd.DataFrame(data_list) 함수를 사용하여 데이터프레임을 제작했다.

※ 시각화





이와 같이 Decition Tree를 시각화 하여 구조를 확인해 봤습니다.

※ 전처리

```
df=pd.read_csv('데이터_new/baseball_savant_foreigners_2011_2019.csv',encoding='EUC-KR') #2011~2019년 MLB선수의 경기데이터
df
```

one	stand	p_throws	bb_type	...	plate_x	plate_z	ax	ay	az	launch_speed	launch_angle	release_spin_rate	pitch_name	pitcher_name
4.0	R	R	popup	...	-0.626	2.771	-6.404	26.077	-9.694	NaN	NaN	NaN	4-Seam Fastball	니퍼트
5.0	L	R	fly_ball	...	-0.112	2.401	-10.168	28.786	-8.895	NaN	NaN	NaN	4-Seam Fastball	니퍼트
2.0	L	R	NaN	...	0.272	3.052	-5.605	26.657	-7.794	NaN	NaN	NaN	4-Seam Fastball	니퍼트
13.0	L	R	NaN	...	-1.193	0.632	-9.099	30.273	-6.371	NaN	NaN	NaN	4-Seam Fastball	니퍼트
14.0	L	R	NaN	...	0.370	1.383	-5.846	27.290	-11.262	NaN	NaN	NaN	4-Seam Fastball	니퍼트
13.0	L	R	NaN	...	-1.233	2.059	-10.897	18.539	-20.286	NaN	NaN	NaN	Changeup	니퍼트
9.0	L	R	NaN	...	0.510	1.722	-6.251	27.786	-10.031	NaN	NaN	NaN	4-Seam Fastball	니퍼트

스탯데이터와는 다르게 던진 공에 대한 데이터를 분석에 사용하기 위해서는 가공이 필요했습니다.

우선 선수가 던질 수 있는 구종별로 데이터를 확인하기 위하여 groupby함수를 사용했습니다.

```
df2=df.groupby([df['pitcher_name'],df['pitch_name']]).mean() #경기데이터를 선수명,구종 으로 groupby하여 평균값으로 추려냄
df2
```

		release_speed	batter	pitcher	zone	balls	strikes	pfx_x	pfx_z	plate_x	plate_z
pitcher_name	pitch_name										
니퍼트	2-Seam Fastball	92.223077	392827.961538	430962.0	9.923077	0.692308	0.500000	-1.188737	1.739510	-1.217808	2.706231
	4-Seam Fastball	93.842708	400768.851562	430962.0	8.307292	1.065104	0.937500	-0.587416	1.711267	-0.016130	2.822839
	Changeup	82.770370	407691.975309	430962.0	9.876543	1.259259	1.012346	-1.036095	1.053453	-0.740568	2.391840
	Curveball	75.119512	435796.560976	430962.0	8.634146	0.573171	0.695122	0.380708	-0.550000	0.123439	2.369098
	Cutter	93.033333	453475.333333	430962.0	10.000000	2.000000	1.166667	0.073879	1.597022	-0.179500	2.543833
	Intentional Ball	72.325000	279913.000000	430962.0	11.000000	1.500000	0.000000	-0.647004	1.891692	-3.584250	5.119000
	Pitch Out	86.100000	444379.000000	430962.0	11.000000	0.000000	1.000000	-0.486267	1.078633	-3.708000	4.313000
	Slider	85.050000	444235.750000	430962.0	10.250000	0.750000	0.500000	0.158075	0.540775	0.650500	1.889000

위와 같이 선수에 대한 구종별 데이터를 평균을 내어 표현했습니다.

또한 이는 decition tree에 적용하기 위하여 한줄의 데이터로 표시하기로 결정했습니다.

각 구종별 데이터를 '구종_데이터컬럼' 이러한 방식으로 네이밍했습니다.

```

ball_list2=[]
for ball in ball_list:
    ball_list2.append(ball+'_release_speed')
    ball_list2.append(ball+'_release_spin_rate')
    ball_list2.append(ball+'_pfx_x')
    ball_list2.append(ball+'_pfx_z')
    ball_list2.append(ball+'_ax')
    ball_list2.append(ball+'_ay')
    ball_list2.append(ball+'_az')
ball_list2

['Changeup_release_speed',
'Changeup_release_spin_rate',
'Changeup_pfx_x',
'Changeup_pfx_z',
'Changeup_ax',
'Changeup_ay',
'Changeup_az',
'Pitch Out_release_speed',
'Pitch Out_release_spin_rate',
'Pitch Out_pfx_x',
'Pitch Out_pfx_z',
'Pitch Out_ax',
'Pitch Out_ay',
'Pitch Out_az',
'Split Finger_release_speed',
'Split Finger_release_spin_rate',
'Split Finger_pfx_x',
'Split Finger_pfx_z',
'Split Finger_ax',
'Split Finger_ay',
'Split Finger_az']

test_dic={t:[] for t in ball_list2}
# ball_list2 라는 이름으로 생성된 컬럼명 list를
# DataFrame으로 가공하기 위하여 Dic형태로 만들어줌

```

이처럼 DataFrame형성을 위하여 dic형태로 재선언했습니다.

```

# 각 선수별 데이터를 일련의 과정을 통해 test_dic에 삽입해줄
# 각 선수가 사용하지 않는 구종의 데이터에는 nan값을 삽입해줄
for name in name_list:
    test_dic['pitcher_name'].append(name)
    for ball in ball_list:
        try:
            dfdf=df2.loc[name].loc[ball]
            release_speed=dfdf['release_speed']
            release_spin_rate=dfdf['release_spin_rate']
            print(release_spin_rate)
            pfx_x=dfdf['pfx_x']
            pfx_z=dfdf['pfx_z']
            ax=dfdf['ax']
            ay=dfdf['ay']
            az=dfdf['az']
            test_dic[ball+'_release_speed'].append(release_speed)
            test_dic[ball+'_release_spin_rate'].append(release_spin_rate)
            test_dic[ball+'_pfx_x'].append(pfx_x)
            test_dic[ball+'_pfx_z'].append(pfx_z)
            test_dic[ball+'_ax'].append(ax)
            test_dic[ball+'_ay'].append(ay)
            test_dic[ball+'_az'].append(az)
        except:
            test_dic[ball+'_release_speed'].append(np.nan)
            test_dic[ball+'_release_spin_rate'].append(np.nan)
            test_dic[ball+'_pfx_x'].append(np.nan)
            test_dic[ball+'_pfx_z'].append(np.nan)
            test_dic[ball+'_ax'].append(np.nan)
            test_dic[ball+'_ay'].append(np.nan)
            test_dic[ball+'_az'].append(np.nan)

```

마지막으로 네이밍한 컬럼명이 될 dic의 value에 각각 데이터를 삽입했습니다.

```
def value_encode(origin_df, col_name, range_list):
    result_df = pd.DataFrame.copy(origin_df)
    #result_df = result_df.drop(list(range(0, len(result_df))))
    for df_index in result_df.index:
        val = result_df.loc[df_index, col_name]
        for range_index in range(0, len(range_list)):
            if range_list[range_index] <= val:
                result_df.loc[df_index, col_name] = range_index:
    return result_df
```

```
def make_range(origin_df, col_name):
    range_num = 10
    range_list = []
    num_per_range = int(len(origin_df.index)/range_num) + 1
    print(num_per_range)
    sorted_df = origin_df.sort_values(col_name)
    appended_df_index = 0
    for i in range(0, range_num):
        try:
            range_list.append(sorted_df.iloc[appended_df_index:appended_df_index+1][col_name].values[0])
        except:
            print('error')
        appended_df_index = appended_df_index + num_per_range
    return range_list
```

또한 정리된 데이터를 빈도에 맞게 범주를 나누어주고 그 값을 범주의 값으로 변경해주는 함수를 제작했습니다.

```
test_df1=pd.merge(stat_data, kbo_data[['pitcher_name','ERA']], on='pitcher_name')
test_df1=value_encode(test_df1, 'ERA_y', make_range(test_df1, 'ERA_y'))
test_df1
```

	H	HR	BB	HBP	SO	WHIP	BABIP	FIP	LD%	GB%	FB%	IFFB%	SwStr%	Swing%	ERA_y
	61.000000	7.000000	34.000000	5.000000	47.000000	1.680000	0.320000	5.090000	0.267000	0.320000	0.666000	0.099000	0.082000	0.423000	1.0
	100.000000	10.250000	21.500000	1.250000	40.250000	2.125000	0.333750	5.660000	0.208250	0.466250	0.641250	0.044250	0.059000	0.440250	3.0
	90.000000	10.500000	36.500000	2.333333	73.666667	1.560000	0.317833	4.548333	0.205167	0.442667	0.632167	0.104333	0.077500	0.428500	5.0
	31.000000	7.333333	14.666667	0.333333	14.666667	1.630000	0.263667	6.703333	0.173000	0.345667	0.618000	0.101000	0.052667	0.454000	8.0
	16.500000	2.000000	5.000000	0.000000	10.500000	1.585000	0.303500	6.235000	0.162500	0.479500	0.590500	0.034500	0.101500	0.472500	8.0
	63.000000	7.666667	17.333333	2.333333	29.666667	2.496667	0.385667	7.310000	0.352333	0.357333	0.535000	0.042333	0.059000	0.471000	7.0
	22.000000	4.500000	9.500000	1.000000	15.000000	1.585000	0.287000	5.965000	0.254000	0.425500	0.541500	0.099500	0.094000	0.433500	3.0
	88.333333	10.000000	29.666667	4.333333	63.833333	1.593333	0.342833	4.446667	0.224833	0.448500	0.595667	0.094333	0.088500	0.455000	1.0
	45.750000	4.250000	5.750000	1.500000	41.000000	1.677500	0.412750	5.267500	0.193750	0.464750	0.561750	0.126500	0.060750	0.440500	6.0

이를통해 Decition Tree 의 y값이 될 ERA_y 가 범주의 값으로 변환된 것을 볼 수 있습니다.

※ 분석

```
dt_prediction1 = dt_clf1.predict(X1_test)
accuracy1 = accuracy_score(y1_test, dt_prediction1)
dt_prediction2 = dt_clf2.predict(X2_test)
accuracy2 = accuracy_score(y2_test, dt_prediction2)
dt_prediction3 = dt_clf3.predict(X3_test)
accuracy3 = accuracy_score(y3_test, dt_prediction3)
dt_prediction4 = dt_clf4.predict(X4_test)
accuracy4 = accuracy_score(y4_test, dt_prediction4)
dt_prediction5 = dt_clf5.predict(X5_test)
accuracy5 = accuracy_score(y5_test, dt_prediction5)
dt_prediction6 = dt_clf6.predict(X6_test)
accuracy6 = accuracy_score(y6_test, dt_prediction6)
dt_prediction7 = dt_clf7.predict(X7_test)
accuracy7 = accuracy_score(y7_test, dt_prediction7)
print('ERA정확도:', accuracy1, ' TBF정확도:', accuracy2, ' H정확도:', accuracy3)
print('HR정확도:', accuracy4, ' BB정확도:', accuracy5, ' HBP정확도:', accuracy6)
print('SO정확도:', accuracy7)
```

```
ERA정확도: 0.1111111111111111 TBF정확도: 0.0 H정확도: 0.0555555555555555
HR정확도: 0.0555555555555555 BB정확도: 0.0555555555555555 HBP정확도: 0.1111111111111111
SO정확도: 0.1666666666666666
```

앞서 전처리가 완료된 데이터들의 x값에 대한 y값의 연관성을 계산해봤으나 연관성이 높지 않은 것을 확인했습니다.

이에 우리 팀은 상대한 타자수와 피안타수를 조합하기로 결정했습니다.

```
X9 = np.array(pd.DataFrame(test_df9, columns = ['ERA', 'WAR', 'TBF_x', 'H_x', 'HR', 'BB', 'HBP', 'SO', 'WHIP', 'BABIP', 'FIP', 'LD%', 'GB%', 'FB%', 'IFFB%', 'H/TBF']))
y9 = np.array(pd.DataFrame(test_df9, columns = ['H/TBF_class']))
X9_train, X9_test, y9_train, y9_test = train_test_split(X9, y9)
dt_clf9 = DecisionTreeClassifier()
dt_clf9 = dt_clf9.fit(X9_train, y9_train)
dt_dot_data9 = tree.export_graphviz(dt_clf9, out_file=None,
                                   feature_names = ['ERA', 'WAR', 'TBF_x', 'H_x', 'HR', 'BB', 'HBP', 'SO', 'WHIP', 'BABIP', 'FIP', 'LD%', 'GB%', 'FB%', 'IFFB%', 'H/TBF'],
                                   class_names = ['good', 'bad'],
                                   filled = True, rounded = True, special_characters = True)
dt_graph9 = pydotplus.graph_from_dot_data(dt_dot_data9)
dt_prediction9 = dt_clf9.predict(X9_test)
accuracy9 = accuracy_score(y9_test, dt_prediction9)
print('정확도:', accuracy9)
```

정확도: 0.8333333333333334

피안타수(H) / 상대한 타자수(TBF) 를 y값으로 준 후 Decition Tree를 생성 후 정확도를 계산하니 지속적으로 높은 정확도가 출력되는 것을 확인했습니다.

```
import pickle

with open('trees.p', 'wb') as file:
    pickle.dump(dt_clf11, file)

# H/TBF(피안타수/상대한타자수) 기준으로 0, 1 나누기 평균이상이 0, 평균미만이 1
with open('trees.p', 'wb') as file:
    pickle.dump(dt_clf9, file)
    pickle.dump(dt_clf11, file)
```

앞서 설명한 방식으로 생성한 Decition Tree 객체를 pickle 라이브러리를 사용하여 저장했습니다.

이를 통해 flask로 구현할 웹을 구동할 때 Decition Tree를 생성하는 일련의 과정을 생략할 수 있었습니다.

```
In [72]: list_for_drop = list(stat_data.columns)
list_for_drop = list_for_drop[3:]
list_for_drop
dropped_df300=test_df300.drop(columns=list_for_drop)
dropped_df300
```

Changeup_release_speed	Changeup_release_spin_rate	Changeup_pfx_x	Changeup_pfx_z	Changeup_ax	Changeup_ay	Changeup_az	Pitch Out_release_speed	Out_release_sp
82.770370	0.000000	-1.036095	1.053453	-9.660741	20.266074	-23.271333	86.100000	
82.770370	0.000000	-1.036095	1.053453	-9.660741	20.266074	-23.271333	86.100000	
82.770370	0.000000	-1.036095	1.053453	-9.660741	20.266074	-23.271333	86.100000	
82.770370	0.000000	-1.036095	1.053453	-9.660741	20.266074	-23.271333	86.100000	
82.770370	0.000000	-1.036095	1.053453	-9.660741	20.266074	-23.271333	86.100000	
82.770370	0.000000	-1.036095	1.053453	-9.660741	20.266074	-23.271333	86.100000	
82.770370	0.000000	-1.036095	1.053453	-9.660741	20.266074	-23.271333	86.100000	
82.770370	0.000000	-1.036095	1.053453	-9.660741	20.266074	-23.271333	86.100000	
84.234876	2028.250000	0.912363	1.424642	11.366478	24.148745	-19.145179	79.300000	
84.234876	2028.250000	0.912363	1.424642	11.366478	24.148745	-19.145179	79.300000	

MLB선수의 투구의 구질에 대한 데이터 또한 가공하여 KBO성적과 대응시켰을 때 정확도가 크게 높은 것을 확인할 수 있었습니다.

```

X300 = np.array(pd.DataFrame(test_df300, columns = col_list))
y300 = np.array(pd.DataFrame(test_df300, columns = ['ERA_class']))
X300_train, X300_test, y300_train, y300_test = train_test_split(X300,y300)
dt_clf300 = DecisionTreeClassifier()
dt_clf300 = dt_clf300.fit(X300_train, y300_train)
dt_dot_data300 = tree.export_graphviz(dt_clf300,out_file=None,
                                     feature_names =col_list ,
                                     class_names = ['good','bad'],
                                     filled = True,rounded = True,special_characters = True)
dt_graph300 = pydotplus.graph_from_dot_data(dt_dot_data300)
dt_prediction300 = dt_clf300.predict(X300_test)
accuracy300 = accuracy_score(y300_test, dt_prediction300)
print('정확도:', accuracy300)

```

정확도: 0.7816091954022989

※ 예측

결과 및 보완점

결과물을 만들기 위한 코드 테스트는 **jupyter notebook** 환경에서 작성하고 테스트를 거쳐 사용할 코드는 **visualStudioCode**를 통해 **flask** 기능을 적용한 환경에 작성했습니다.



Donald Zackary Greinke

분석하기

KBO 사이트

위의 검색창 UI에 분석할 MLB선수의 이름을 적게 되면

선수에 대한 데이터 크롤링, 데이터프레임 가공의 일련이 과정이 자동으로 처리되며 그 결과값을

앞서 미리 저장해놓은 **Decition Tree**에 적용하여 결과값을 예측합니다.

그 결과로는

Donald Zackary Greinke

선수는 KBO에서 평균이상의 성적을 낼 것으로 예측됩니다.

결과값 = [0.]

상위 15% KBO 기준성적 : 피안타수/상대한 타자수 = 0.25 , 평균자책점 = 5

Donald Zackary Greinke의 역대 MBL 성적

팀	년도	승리기여도	타석수	피안타	피홈런	피볼넷	피사구	삼진수	출루허용률	타구 안타비율	자책점	라인드라이브	땅볼	플라이볼	인필드플라이볼	헛스윙	스윙
Royals	2004	3.97	1.8	599.0	143.0	26.0	26.0	8.0	100.0	1.17	4.7	22.1	34.6	43.2	13.2	1.8	0.06
Royals	2005	5.8	2.0	829.0	233.0	23.0	53.0	13.0	114.0	1.56	4.49	23.4	39.2	37.3	11.3	-0.1	-0.79
Royals	2006	4.26	0.0	28.0	7.0	1.0	3.0	0.0	5.0	1.58	5.04	35.0	35.0	30.0	16.7	nan	3.38
Royals	2007	3.69	2.1	507.0	122.0	12.0	36.0	3.0	106.0	1.3	3.74	22.3	32.1	45.5	8.6	nan	0.44
Royals	2008	3.47	4.2	851.0	202.0	21.0	56.0	4.0	183.0	1.28	3.56	19.0	42.7	38.3	8.7	nan	2.41
Royals	2009	2.16	8.7	915.0	195.0	11.0	51.0	4.0	242.0	1.07	2.33	19.4	40.0	40.5	9.5	nan	2.69
Royals	2010	4.17	4.9	919.0	219.0	18.0	55.0	7.0	181.0	1.25	3.34	17.8	46.0	36.3	8.8	nan	1.75
Brewers	2011	3.83	3.3	715.0	161.0	19.0	45.0	4.0	201.0	1.2	2.98	22.0	47.3	30.8	5.0	nan	2.07
2 Teams	2012	3.48	4.8	868.0	200.0	18.0	54.0	2.0	200.0	1.2	3.1	21.7	49.2	29.1	9.1	-0.6	0.07
Dodgers	2013	2.63	3.4	717.0	152.0	13.0	46.0	7.0	148.0	1.11	3.23	23.8	45.6	30.6	11.2	1.2	-2.97
Dodgers	2014	2.71	4.5	821.0	190.0	19.0	43.0	2.0	207.0	1.15	2.97	22.8	48.7	28.5	13.2	nan	0.03
Dodgers	2015	1.66	5.3	843.0	148.0	14.0	40.0	5.0	200.0	0.84	2.76	19.1	48.0	32.9	9.3	nan	2.98

이와같은 화면을 출력하게 됩니다.

보완점

호스팅한 웹 서비스에 접속하여 검색 진행 시 **selenium** 호스팅 pc에서는 **selenium**이 직접적으로 실행되어 다수에 대한 서비스가 부적절할 것으로 예상됩니다. 크롤링 하는 부분에 있어 다른 방법으로의 접근이 필요하다고 생각했습니다.

실질적으로 KBO에서 활동하는 MLB선수의 절대량이 적기 때문에 데이터를 분석하고 예측하는 방법에 있어 평균값을 내어 조사한다거나 1:1 매핑이 되는 **Decition Tree**를 사용한 것에 한계점이 있다고 느꼈습니다.

후기

장경석

기존의 데이터를 사용하는 방법, 크롤링을 통한 데이터 수집을 하는 방법, 분석도구를 사용하는 방법, 시각화하는방법, 예측하는방법, 웹서비스를 구현하는 방법 등 전반적인 분석 서비스의 요소들을 이해하게 된 것 같아 흡족한 프로젝트였습니다. 또한 분업을 통해 진행되었는데 소통이 잘 되어 하나의 프로그램으로 잘 융합되어 만족스럽습니다.

장지승

의미있는 값을 도출하기 위해 여러차례 시도하고 원하는 결과가 나오지 않아 다시 시작해야 하는 상황이 등장해서 상심하는 상황을 팀원들의 함께하고자 하는 의지로 극복할 수 있어서 뿌듯한 시간 이였습니다. 또 데이터처리를 효율적으로 하기위해 파이썬 코드를 정리하고 데이터를 관리하는 것에 조금은 익숙해질 수 있어서 보람있었습니다. 과정에서 웹에 관심이 많은 저로써는 크롤링을 하는 라이브러리를 사용하는데 익숙해질 수 있어서 즐거웠습니다. **selenium** 과 **beautifulsoup** 을 활용하면서 브라우저와 소통하고 **html**코드를 다루는 것이 좋았습니다.

김수연

이번 수업에서는 데이터 수집 및 저장에 필요한 기본적인 방법을 익혔습니다. 필요한 데이터를 크롤링하여 수집하고, 수집한 정형데이터와 비정형데이터를 **db**에 저장하고, 그 데이터를 전처리과정을 통해 분석하며 시각화하고, 예측하는 일련의 과정을 최대한 포함하는 것이 이번 프로젝트의 목표였습니다. 각 요소들을 잘 포함하여 프로젝트를 완성한 듯 하여 목표를 잘 달성했다고 생각합니다. 목표달성 과정에서 의견충돌이 일어났을때 혹은 예기치않은 상황에 부딪혔을 때 팀원들과 충분한 소통으로 문제를 잘 해결해나가 만족스러운 팀활동이었습니다.