메이저리거의 FA 적정 급여 책정

- 최종 보고서

2조 김영하 김지환 송규상 심석현 안호정 이용하

팀원소개

김영하

경영학과 4학년
데이터, 코딩 경험이 없어서 관련 지식을 습득하고자 이 수업을 수강함

송규상

경영학과 4학년
파이썬을 공부하고 혼자 캐글에 몇 번 참여해보다가 팀 단위로 프로젝트를 해보면 좋을 것 같아서 수업을 수강함

안호정

경영학과 4학년
데이터 분석 등에 대한 경험이 없으나, 빅데이터 분야에 대해 평소 관심을 갖고 있어 관련 지식들을 얻고자 수업을 수강함

김지환

경영학과 4학년
데이터 관련 경험 없으나, 인사 분야에서 데이터 분석이 어떻게 활용되는지 학습해보고 싶어서 수업을 수강함
이번 프로젝트에서 **프로젝트 매니저**를 담당

심석현

경영학과 4학년
매니지먼트 분야 중 인사 쪽에 흥미가 있고, 특히 인사 분야 중 평가/보상 분야에 대한 관심이 많아 수업을 수강함

이용하

경영학과 4학년
AI 채용 등 인사 분야에 빅데이터 기술이 도입되는 상황에 관심이 있어, 이의 방식을 알고 직접 구현해보며 유용한 인사이트를 얻을 수 있을 것 같아 수업을 수강함

개인별 학습상황

김영하

- Python for everyone 강좌와 교수님의 사이버 강좌를 통해 파이썬의 개념부터 function을 만드는 것까지 학습함.
- Numpy와 Pandas를 이용하여 간단한 수식 및 도식화하는 법을 학습함.
- Seaborn을 통한 시각화 하는 법을 학습함.
- 기초적인 머신러닝의 개념을 이해하고 Regression을 돌려봄.

김지환

- Python 기본 문법 및 자료구조 학습
- 웹 언어 자료구조 및 웹 접근, Data Crawling 기법 학습
- Numpy, Pandas, Seaborn, Matplotlib, sklearn 등 범용적인 3rd party module 사용 방법 학습
- Linear regression, Polynomial regression, PCA analysis 등의 통계적 분석 방법 학습.

송규상

- 데이터마이닝 수업을 한 번 수강하여 머신러닝과 관련한 분석 기법을 학습하는 데 큰 무리 없었음.
- pandas나 numpy, matplotlib 같은 모듈에 대해서는 써보긴 했으나 영상을 통해서 배운 점이 좋았음.
- 다양한 머신러닝 기법을 팀원들이 적용하여 코드를 짠 것을 보면서 더 쉽게 학습할 수 있었음.

개인별 학습상황

심석현

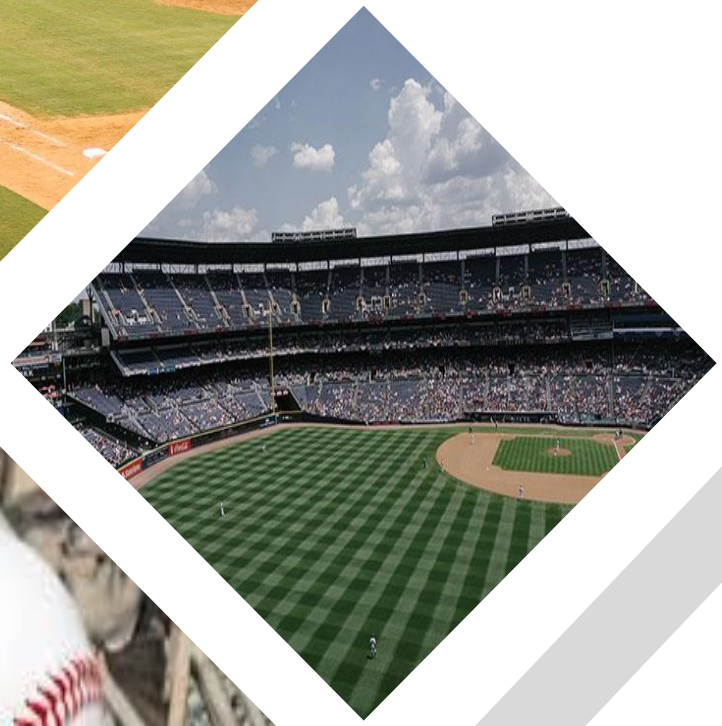
- R로는 머신러닝에 관련해 여러가지 학습을 진행하였으나 파이썬으로는 처음이었음. 그러나 이론적인 부분에 대한 학습은 진행해놓은 상황이어서 기술적인 부분에 대해서 따라잡는데 큰 무리는 없없음.
- 학습을 진행하며 머신러닝 기법 중 특히 Linear Regression, Decision Tree 그리고 Random Forest 부분을 집중적으로 학습하면서 관련된 지식을 풍부하게 쌓을 수 있었음.
- 다른 모델들, PCA나 NN에 대하여 조원들이 학습한 내용을 바탕으로 이해를 진행하였고 추후 다른 머신러닝 기법에 대한 학습과 시각화에 대한 학습을 더 진행할 계획.
- 추후 선수의 선발/모집이라는 측면에서 드래프트와 관련된 연구를 더 진행해보고 싶은 욕심도 생겨서, 이번에 배운 파이썬과 머신러닝을 기반으로 더욱 많은 데이터 분석을 진행할 예정.

안호정

- yscec 사이버 강의, PY4E 강의 및 제공된 자료, youtube 강의 등을 통해 Python 기본 문법들과 자료구조 등을 학습함.
- Pandas, Matplotlib, Numpy, Seaborn 등의 설치 및 사용방법을 학습함.
- 다양한 Machine learning 기법들의 개념을 이해하고 학습함
- 데이터 수집, 텍스트 전처리, 분석 및 그래프 시각화 방법을 학습함.

이용하

- selenium, beautifulsoup을 통한 crawling 실습과 개인 학습을 통해 효율적으로 crawling하는 방법을 알게 됨
- Python으로 몇개의 머신러닝 기법을 실행하는 방법은 알고 있었지만 이론적인 부분의 이해도는 조금 부족했는데, 팀원들과 학습, 프로젝트를 진행하며 Linear Regression, PCA, Random Forest 등의 기법의 이론을 이해할 수 있었음



INDEX

- 주제 선정 이유
- 데이터 출처 및 특성
- Methodology
- 데이터 분석 절차 및 결과
- 시사점

FA in Major League

"**류현진**, 다저스와 3년 696억 원 재계약" 美 뉴욕 매체 예상
스포츠조선 | 10분 전 | 네이버뉴스
공식적으로 올겨울 자유계약선수(**FA**) 신분을 선언한 **류현진**(32)이 끝내 LA 다저스에 잔류할 가능성이 크다는 미국 현지 언론매체의 예상이 나왔다. **류현진**은 현재 지난 7년간 몸담

MLB 전직 단장 예상, "**류현진** 3년 5550만 달러… 콜 역대 최고액"
스포티비뉴스 | 6시간 전 | 네이버뉴스
보든이 예상한 **류현진**의 **FA** 금액은 3년간 5550만 달러(약 644억 원)다. 연평균 1850만 달러 수준이다. 이는 올해 **류현진**의 연봉(1790만 달러)보다 소폭 높은 정도에 그친다. 올해 사이

3년 5930만 달러… 통계 전문가들이 예상한 **류현진** FA 금액
스포티비뉴스 | 6시간 전 | 네이버뉴스
그러면서 **류현진**의 올해 **FA** 예상 금액으로 연평균 1900만 달러, 총액 5930만 달러를 제시했다. 이 금액은 7명의 통계 전문가들이 각자 제시한 수치의 평균이다. 평균 계약기간은 3.12

선발강화 외친 MIN·SD, **류현진** 포함 FA 가치 상승 조짐
스포츠서울 | 21시간 전 | 네이버뉴스
미니애폴리스 스타 트리뷴은 이러한 미네소타 상황을 짚으면서 미네소타가 **FA** 매디슨 범가너, 잭 휠러, **류현진**을 바라볼 것으로 예상했다. 게릿 콜과 같은 최대어 영입은 현실적으로 어렵지만 콜과 스티븐…

[SC핫포커스] 스트라스버그의 FA 합류, **류현진**에 어떤 영향 미칠까
스포츠조선 | 1일 전 | 네이버뉴스
역시 **FA** 시장에 나온 **류현진**에게는 어떤 영향을 미칠까. 스트라스버그는 최대어 게릿 콜을 대체할 수 있는 선수다. 하지만 콜의 굳건한 입지를 흔들 만한 영향을 미치지는 못할 것이다. 오히려 **류현진**, 잭 휠러, 매디슨…

2019년을 끝으로 류현진과 LA 다저스의 계약이 만료되면서 FA시장 및 계약 금액 에 대한 관심이 뜨거워짐

+

FA 계약 가격 책정에 대한 기준이 모호하며 구단마다 천차만별이라 예측이 어려움

선수의 개인정보와 과거 기록을 바탕으로 적정한 FA 가격을 책정하는 모델의 필요성

FA Salary Prediction

성적 부진의 기준

기존 성적 및 나이 등 정량적인 요소 ✚ 갑작스런 부상 및 사고, 약물 복용 등의 개인적 외부효과가 작용 多

But, 천여명에 달하는 메이저리그 선수들의 개인적 외부효과를 일일이 반영하는 것은 현실적 한계 존재

" 기존의 성적 및 개인정보를 기반으로 적정급여를 책정하는 모델 제작"

"기존 2014~2018년까지의 스탯 및 급여 데이터를 바탕으로 급여를 예측하는 모델 제작"

FA Salary Prediction

연구 대상

투수 보다는 기록 및 변수가 다양한 타자를 대상으로 선정

연구 목표

1. 다양한 분석 기법을 통해 스탯을 기반으로 타자의 급여를 예측하는 다양한 모델을 만들고, 각 모델의 장점과 한계점을 분석

2. 예측 모델을 통해 2019년 12월에 시작되는 메이저리그 FA 시장에서 FA자격을 얻은 선수들의 계약 금액 예측

인사와의 연관성

1. 스포츠 선수와 같이 과업 실행 정도가 구체적으로 드러나는 경우 연봉 협상 시 급여 측정의 가이드라인 마련 가능

2. 일반 인사 상황에서 과업에 따른 성과 측정 시 상황에 따라 어떤 분석 모델을 적용하는 것이 알맞은지 파악 가능

FA Salary Prediction

연구 방법

Through Statistical Modeling

Simple Linear Regression

OLS 방식의
기본 선형회귀분석

PCA Analysis

변수간 관계에 영향
을 받지 않는 차원
축소를 통한 분석

Through Deep Learning

Neural Network Method

신경망을 통한
데이터 셋의 학습으로
모델 수립

Random Forest Method

모든 Components에
대한 Randomly
Picked Simulation

Literature Review

Meltzer, Josh. “Average Salary and Contract Length in Major League Baseball: When Do They Diverge?” 2005, Department of Economics, Stanford University, CA.

Methodology
First stage regression, Second stage regression을 통해 contract length에 따른 average salary 측정

Variables
Average yearly salary, length of contract, OPS, all-star appearances, gold glove winnings, health status, age, position, contract status, payroll for the team

Result
Performance metrics are a significant predictor of player salary

Hakes, J.K. & Turner, C. (2011) Pay, productivity and aging in major league baseball. Journal of Productivity Analysis. 35, 61 – 74.

Methodology
Examined productivity and experience in determining salary for position players

Variables
Age, OPS, MVP awards, ALLSTAR appearances, population of market, indicators for negotiating freedom

Result
1. Salary peaked at least 1.8 years after hitting productivity in baseball and salary declined slightly before retirement.
2. Premier players had their performance decline slower than non-premier players

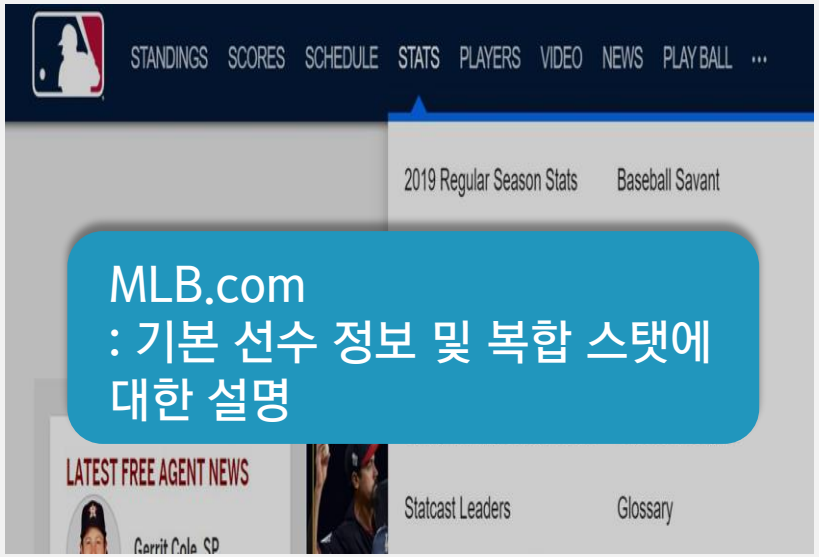
Hochberg, Daniel. “The Effect of Contract Year Performance on Free Agent Salary in Major League Baseball” 2011, Department of Economics, Haverford College, PA

Methodology
Linear model (Simple regression, interaction model)

Variables
On-base plus slugging (OPS), stolen base rate, fielding rate above replacement, age and position

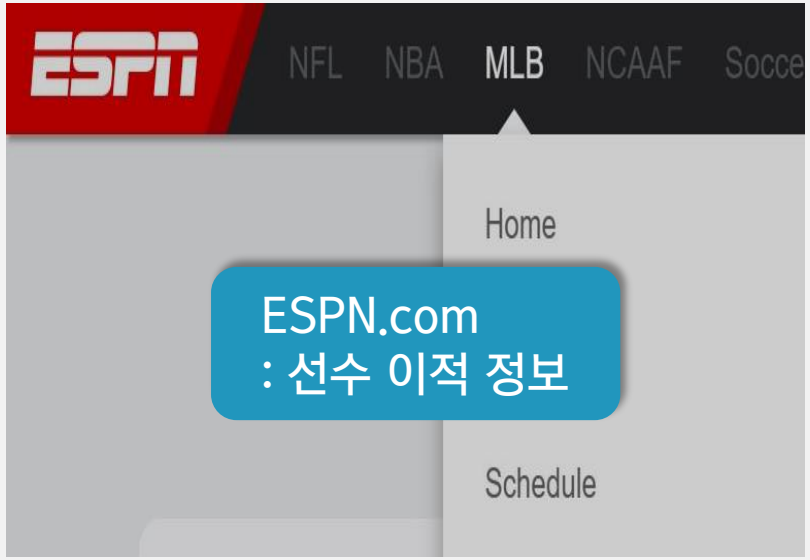
Result
1. Position did not influence the salary
2. Contract year performance is overvalued compared to the previous seasons before a contract was signed

데이터 출처

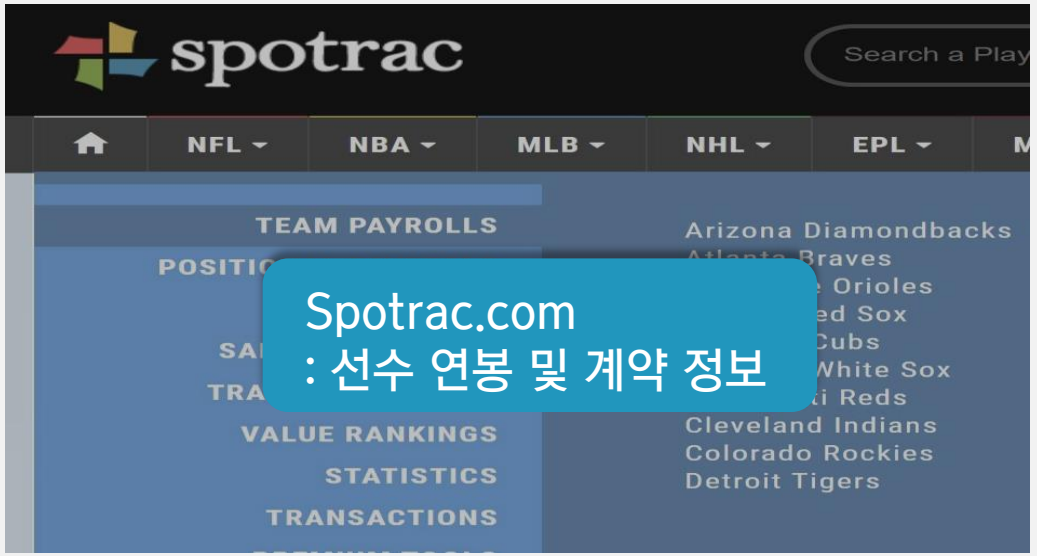


MLB.com
: 기본 선수 정보 및 복합 스탯에 대한 설명

BasebalReferences.com
: 선수 연도별 기록 및 세부 스탯



ESPN.com
: 선수 이적 정보



Spotrac.com
: 선수 연봉 및 계약 정보

데이터 특성

종속변수

Salary: 선수 연간 급여 (계약금 / 계약 연수)

→ 정수형

선수마다 계약 기간의 차이가 있기 때문에
계약금 / 계약기간 으로 연간급여를 계산하여 비교

데이터 특성

독립변수

- **Age**(정수형) : 선수 나이 (만 나이 기준). 계약금에 음의 관계가 있을 것으로 예상
- **TM** : Team name. 팀의 경제적 사정에 따라 FA 계약금 크기도 달라질 것으로 예상
- **Lg** : National League or American League
- **G**(정수형) : Games(경기 수). FA 계약금에 양의 상관관계 예상(이하 +)
- **PA**(정수형) : 타석 수. +
- **AB**(정수형) : 타수' 로서 타석 수에서 볼넷, 몸에 맞는 볼, 희생번트, 희생플라이, 타격방해, 주루방해 상황을 뺀 횟수로 타자가 타격행위를 완료한 횟수. +
- **R**(정수형) : Run, 득점. +
- **H**(정수형) : Hits, 안타. +
- **2B**(정수형) : Double Hits, 2루타. +
- **3B**(정수형) : Triple Hits, 3루타. +
- **BB**(정수형) : 볼넷. +
- **SO**(정수형) : 삼진. –
- **BA**(실수형) : 타율. +
- **OBP**(실수형) : 출루율. +
- **SLG**(실수형) : 장타율. +
- **OPS**(실수형) : 출루율+장타율. +
- **OPS+**(실수형) : 조정 OPS, 리그 평균적인 생산력을 가진 타자의 수치를 100으로 간주. +
- **TB**(정수형) : 총루타, 타자가 안타를 통해 획득한 루의 총계. +
- **GDP**(정수형) : 병살타. –
- **HBP**(정수형) : 데드볼.
- **SH**(정수형) : 희생번트
- **SF**(정수형) : 희생플라이
- **IBB**(정수형) : 고의사구. +
- **Pos** : 포지션

❖ 데이터 수집

1. 선수정보 로드
2. 투수 데이터 제외
3. 급여 데이터 크롤링

```
# Load data

import pandas as pd
import numpy as np

year = input('Year: ')
#path = input('Path that you saved txt file: ')
path = "C:/Users/JiWhan Kim/Desktop"
df_stats = pd.read_csv(path+"/stats_"+year+".txt", sep = ',')
try:
    for i in range(31):
        df_stats.rename(columns = {df_stats.columns[i] : df_stats[i][0]}, inplace=True)
except:
    pass
df_stats = df_stats.drop([0], axis=0)
df_stats = df_stats.drop("Rk", axis=1)
df_stats["Salary"] = 0
df_stats.head()
```

	Name	Age	Tm	Lg	G	PA	AB	R	H	2B	...
1	Bobby Abreu*\abreubo01	40.0	NYM	NL	78	155	133	12	33	9	...
2	Jose Abreu\abreujo02	27.0	CHW	AL	145	622	556	80	176	35	...
3	Tony Abreu#\abreuto01	29.0	SFG	NL	3	4	4	0	0	0	...
4	Alfredo Aceves\aceveal01	31.0	NYY	AL	4	0	0	0	0	0	...
5	Dustin Ackley*\ackledu01	26.0	SEA	AL	143	542	502	64	123	27	...

5 rows × 30 columns

▶ Baseballreference.com 에서 제공하는 전체 선수 데이터 셋 사용

▶ 급여항목 포함 X

❖ 데이터 수집

- 1. 선수정보 로드
- 2. 투수 데이터 제외
- 3. 급여 데이터 크롤링

```
#Remove pitchers & Re-index

pit_list = []
for i in range(len(df_stats)):
    pos = str(df_stats.iloc[i,28])
    if pos.find("1") != -1:
        pit_list.append(i+1)

df_stats = df_stats.drop(pit_list, axis=0)
df_stats.index = range(len(df_stats))
print(df_stats.shape)
df_stats.head(10)
```

```
(747, 30)
```

NL 리그의 경우 투수도 타격을 하기 때문에 소량의 타자 데이터 존재

타자 데이터만 보기 위해 Position Summary에 1이 포함된 데이터 삭제

❖ 데이터 수집

- 1. 선수정보 로드
- 2. 투수 데이터 제외
- 3. 급여 데이터 크롤링

```
# Find salaries and add to the df
from bs4 import BeautifulSoup, Comment
import requests
import csv
import re

# Pandas 내 선수 개인 페이지에 접근
count = 0
for i in df_stats["Name"]:
    print(count, end=" ")
    code = i.split("\\")[-1]
    print(code, end=" ")
    url = "https://www.baseball-reference.com/players/"+code[0]+"/"+code+".shtml"
    #print(url)
    r = requests.post(url)
    soup = BeautifulSoup(r.text, "html.parser")

    comments = soup.findAll(text=lambda text:isinstance(text, Comment))
    location = 0
```

```
# 개인 페이지 내 Salary 정보에 접근하여 값 추출
for j in comments:
    #print(location)
    #print(j)
    #print('-------------')
    if j.find('<div class="overthrow table_container" id="div_batting_value">') != -1:
        #print(j)
        break
    location += 1
try:
    #print(comments[location])
    commentsoup = BeautifulSoup(comments[location] , 'lxml')
    value = commentsoup.find('tr', attrs={'id':'batting_value.'+year}).find('td', attrs={'data-stat':'Salary'}
```

급여 데이터는 스탯 데이터와 별도로 선수 개인페이지에 제공됨

타 페이지 or 타 사이트의 선수 개인페이지에 일일이 접근하여 크롤링

❖ 데이터 수집

- 1. 선수정보 로드
- 2. 투수 데이터 제외
- 3. 급여 데이터 크롤링

```
    try:
        salary = re.search('([0-9,]+)', str(value)).group()
        salary = salary.replace(',', '')
    except:
        salary = 0
    print(salary)
    df_stats['Salary'][count] = salary
    count += 1
except:
    df_stats.tail(10)
```

```
26 ariasjo01 1150000
27 arrueba01 3000000
28 ascheco01 500000
29 avilaal01 4150000
30 avilemi01 3500000
31 aybarer01 8500000
32 baezja01 0
33 bakerje03 1600000
34 barmecl01 2000000
35 barnebr02 501000
```

	Name	Age	Tm	Lg	G	PA	AB	R	H	2B	...	OPS	OPS+	TB	GDP	HBP	SH	SF	IBB	Pos Summary	Salary
737	Rafael Ynoa#\ynoara01	26.0	COL	NL	19	71	67	5	23	6	...	0.843	122.0	31	1	0	0	0	0	5/64	0
738	Chris Young\youngch04	30.0	TOT	MLB	111	366	325	40	72	20	...	0.683	95.0	125	3	5	1	3	2	78/9D	7250000
739	Chris Young\youngch04	30.0	NYM	NL	88	287	254	31	52	12	...	0.630	81.0	88	3	4	1	3	2	78/9	7250000
740	Chris Young\youngch04	30.0	NYY	AL	23	79	71	9	20	8	...	0.876	145.0	37	0	1	0	0	0	7/D9	7250000
741	Delmon Young\youngde03	28.0	BAL	AL	83	255	242	27	73	11	...	0.779	117.0	107	6	3	0	0	0	D7/9	0
742	Eric Young Jr.#\younger03	29.0	NYM	NL	100	316	280	48	64	10	...	0.610	77.0	87	2	5	5	2	1	7/48D	1850000

동명이인 선수 페이지에 접근하는 경우 존재

선수명을 코드화하여 중복 접근 문제 방지 및 데이터 개별화

❖ 데이터 전처리

1. 허수처리
2. 중복 데이터 처리

```
import pandas as pd
import numpy as np
```

2014

```
data = pd.read_csv("data/stsal_2014.csv")
data.shape
```

```
(747, 30)
```

```
data = data[lambda x: x.Salary > 0]
data = data.reset_index(drop=True)

data.shape
```

```
(480, 30)
```

▶ 2014~2019년의 data 모두 불러와 동일 코드 적용

▶ 선수 Salary가 미공개여서 0으로 처리되는 경우들 제거

❖ 데이터 전처리

1. 허수처리
2. 중복 데이터 처리

```
def g_sum(index):
    name = data.iloc[index]['Name']

    sum = data.iloc[index]['G']

    for i in range(index+1, len(data)):
        if data.iloc[i]['Name'] == name:
            sum += data.iloc[i]['G']

    return sum
```

▶ ‘G’ column을 기준으로 가중 평균을 구할 때 사용하기 위한 함수로,
‘Name’ column이 같은 중복 행들의 ‘G’ column 값의 합을 계산한다

❖ 데이터 전처리

1. 허수처리
2. 중복 데이터 처리

변수(Column)별 처리

```
i = 0
while i < len(data):
    name = data.loc[i,'Name']
    gamesum = g_sum(i)

    for j in range(17, 22):
        data.iloc[i,j] = data.loc[i,'G'] * data.iloc[i,j]

    k = i + 1
    while k < len(data):
        if data.loc[k,'Name'] == name:

            if data.loc[k,'Tm'] not in data.loc[i,'Tm']:
                data.loc[i,'Tm'] = data.loc[i,'Tm'] + "/" + data.loc[k,'Tm']
            if data.loc[k,'Lg'] not in data.loc[i,'Lg']:
                data.loc[i,'Lg'] = data.loc[i,'Lg'] + "/" + data.loc[k,'Lg']

            for l in range(4, 17):
                data.iloc[i,l] += data.iloc[k,l]

            for m in range(17, 22):
                data.iloc[i,m] += data.loc[k,'G'] * data.iloc[k,m]

            for n in range(22, 28):
                data.iloc[i,n] += data.iloc[k,n]

            data.loc[i,'Salary'] += data.loc[k,'Salary']

            data = data.drop(data.index[k]).reset_index(drop=True)

        else:
            k += 1

    for o in range(17, 22):
        data.iloc[i,o] = data.iloc[i,o] / gamesum

    i += 1
```

- ‘Name’ 이 같은 행을 찾아 중복 행 정리
- ‘Tm’ 과 ‘Lg’ column은 concatenate
- 4~16 index의 column은 sum
- 17~21 index의 column은 가중 평균 (weighted average)
- 22~27 index와 ‘Salary’ 의 column은 sum
- 중복 행 제거

❖ 다중선형회귀 분석

"두개 이상의 독립 변수들과 종속 변수(Salary)들의 관계를 파악하기 위해 다중 선형 회귀(multiple linear regression) 모델을 활용"

데이터 전처리

- 선수 Salary가 미공개여서 0인 경우 제거
- str 타입은 상관 관계를 계산할 수 없으므로, str 타입의 column 제거
- X와 y로 각각 독립변수와 종속변수를 선언

다중 선형 회귀(multiple linear regression)

Python package 로드 및 matplotlib 출력 옵션 설정

```
from sklearn import linear_model
import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline
matplotlib.style.use('ggplot')
```

데이터 생성

```
data = pd.read_csv("MLB_2014_2017.csv")
```

```
data = data[lambda x: x.Salary > 0]
data = data.reset_index(drop=True)
```

```
data.drop(['Name', 'Id', 'Tm', 'Lg'], axis=1, inplace=True)
```

```
# 독립변수를 X라는 변수에 저장
X = data.iloc[:, :30]

# 종속변수를 y라는 변수에 저장
y = data['Salary']
```

❖ 다중선형회귀 분석

데이터 학습

- linear_model.LinearRegression 함수를 통해 선형회귀모델을 만들어 linear_regression 변수 안에 저장
- linear_regression.fit 함수를 이용해 모델을 학습
- linear_regression.predict 함수를 통해 학습된 선형회귀모델에 "x"값을 입력값으로 해서 y값을 예측. 예측된 y값은 prediction 변수에 저장
- 회귀 계수 출력 : a 계수, b 계수

데이터 학습

```
linear_regression = linear_model.LinearRegression()
linear_regression.fit(X = pd.DataFrame(X), y = y)
prediction = linear_regression.predict(X = pd.DataFrame(X))

print('a value = ', linear_regression.intercept_) # 1개
print('b value = ', linear_regression.coef_)      # 독립변수 개수만큼 생성
```

```
a value =  -10982619.233690117
b value =  [ 1.36274882e+05 -6.55047234e+04  8.88326689e+05 -8.67090215e+05
  6.09615272e+04 -8.98084718e+02 -5.53756730e+04  2.91844230e+04
 -8.02233018e+03  2.38050459e+04  3.31505919e+03 -5.26620343e+04
 -8.82066171e+05  5.18100036e+03 -1.67796135e+07 -1.59147922e+08
 -1.72594585e+08  1.77843098e+08 -7.35489334e+03 -2.19718950e+04
  7.45915469e+04 -9.58883268e+05 -1.00137062e+06 -8.56512489e+05
  8.19168027e+04  4.67854155e+05  2.40100149e+04 -7.35742210e+03
  6.62619326e+05  7.97688563e+04]
```

❖ 다중선형회귀 분석

적합도 검증

- 잔차
  - 실제 값 y에서 prediction에 저장된 예측값 y를 빼 잔차를 구함
  - 잔차에 대한 다양한 요약 통계 생성

- 결정계수
  - SSE, SST 변수를 이용해 결정계수 값을 구함

적합도 검증

```
# 잔차
residuals = y - prediction
residuals.describe()
```

```
count    1.042000e+03
mean     2.953776e-08
std      4.719838e+06
min     -1.645846e+07
25%     -2.498309e+06
50%     -5.940571e+05
75%      1.780610e+06
max      2.540903e+07
Name: Salary, dtype: float64
```

```
# 결정계수
SSE = (residuals**2).sum()
SST = ((y-y.mean())**2).sum()
R_squared = 1 - (SSE/SST)
print('R_squared = ', R_squared)
```

```
R_squared =  0.4882322835165951
```

❖ 다중선형회귀 분석

성능 평가

- R-squared : 0.488

- R-squared, 즉 예측력이 작고 오차값이 큰 모델

성능 평가

```
from sklearn.metrics import mean_squared_error
```

```
print('score = ', linear_regression.score(X =pd.DataFrame(X), y=y))
print('Mean_Squared_Error = ', mean_squared_error(prediction, y))
print('RMSE = ', mean_squared_error(prediction, y)**0.5)
```

```
score =  0.4882322835165951
Mean_Squared_Error =  22255487538669.773
RMSE =  4717572.208103419
```

❖ 다중선형회귀 분석

장점

- 학습 속도가 빠르고 예측 또한 빠름
- 매우 큰 데이터셋과 희소한 데이터셋에도 잘 작동
- 결과의 해석이 쉽고 직관적

한계점

- Nonlinear data에는 적합하지 않다

  → data가 linear이라는 가정 필요
  → 대안 : Polynomial regression, Generalized Additive Model(GAM) 등
- Data가 많을 경우 underfit한 경향을 보임
- 독립 변수가 많을 경우 실행이 안되거나 결과가 좋지 않은 경우가 많음
- 독립 변수 간에 상관성이 강할 경우(다중공선성), 다른 조치가 필요하다

  → 대안 : PCA, Lasso regression, Ridge regression 등

❖ PCA 분석

"기존 선형회귀분석 시 변수들 간의 다중공선성 문제를 해결하기 위해 주성분분석(PCA) 모델을 활용"

데이터 전처리

- 주성분분석은 자동으로 Bad data를 제거하지 못하므로, 전처리 시에 데이터에 악영향을 줄 수 있는 Bad data들을 제거해줘야 함

  → 게임수가 20보다 적거나, 급여가 0인 표본들을 제외

- 차원축소를 위해 모든 데이터를 0-1범위로 Scaling

주성분분석(PCA)를 사용한 회귀모델

```
import numpy as np
import pandas as pd
import statsmodels.api as sm
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.decomposition import PCA
```

```
data = pd.read_csv("data/MLB_2014_2017.csv")
```

게임 수 적은 표본 및 연봉정보 없는 표본 제거

```
data = data[lambda x: x.G >= 20]
data = data[lambda x: x.Salary > 0]
data = data.reset_index(drop=True)
data.shape
```

```
(1020, 35)
```

명목변수, 중복적용변수 제거

```
data = data.drop(["Name", "Id", "Tm", "Lg", "OPS", "OPS+"], axis=1)
data.shape
```

```
(1020, 29)
```

데이터 0-1 범위로 Scaling

```
x = data.values #returns a numpy array
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(x)
df = pd.DataFrame(x_scaled)

df.head()
```

❖ PCA 분석

주성분 결정

- 다섯 번째 주성분 축까지에 원 데이터셋 분산의 80% 정도가 놓여 있음

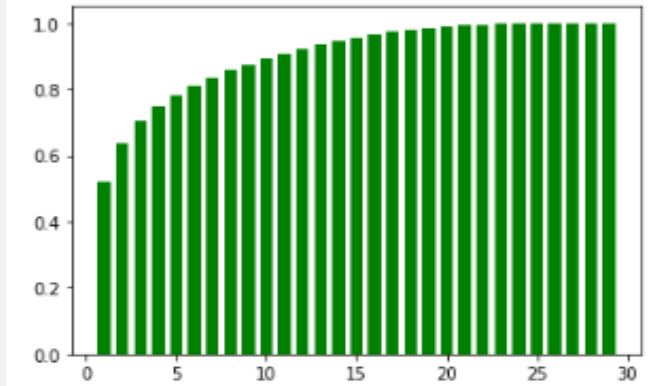
- 이후의 주성분 축은 축 당 1% 미만의 분산이 놓여 있어 매우 적은 정보를 포함함

- 높은 설명력을 위해 지나치게 많은 주성분 축을 포함하면, 주성분분석의 의미가 적어질 뿐만 아니라 Adjusted R-Square 점수가 나빠질 우려가 있음

➡ 따라서 다섯 번째 주성분 축까지만 모델에 포함하기로 결정

주성분 설명력 확인

```
pca = PCA().fit(df)
var = pca.explained_variance_
y = np.cumsum(var)/np.sum(var)

plt.bar(np.arange(1,len(var)+1), y, color='green', width=0.7)
plt.show()
```



주성분 차원 결정 및 설명력

```
pca = PCA(n_components=10)
pca.fit(df)
v = pca.explained_variance_ratio_

# Adjusted R-Square를 최소화하고 설명력을 최대화하는 5차원으로 결정
dim = 5
Sum = 0
# 설명력 출력
for i in range(dim):
    Sum += v[i]
print(Sum)
```

```
0.7833417395818499
```

❖ PCA 분석

축소된 차원 바탕 선형회귀 실시 결과

- 다섯 번째 주성분 축까지를 새로운 데이터 셋으로 설정

- P 값이 유의한 1, 2, 3, 5번째 축을 변수로 OLS 실시

R-squared: 0.754
Adjusted R-squared: 0.753
모든 Variable의 P-value 〈 0.0005 으로,
유의성 높은 모형 완성

PCA 분석을 위한 데이터 세팅

```
pca = PCA(n_components=5)
pca_x = pd.DataFrame(pca.fit_transform(df))
pca_x.columns = ['pc1', 'pc2', 'pc3', 'pc4', 'pc5']
salary = data['Salary']

data_pca = pd.concat([pca_x, salary], axis = 1)
data_pca.columns = ['pc1', 'pc2', 'pc3', 'pc4', 'pc5', 'Salary']
print(data_pca)
```

...

PCA 데이터를 바탕으로 OLS 실행 결과

```
model = sm.OLS.from_formula('Salary ~ pc1 + pc2 + pc3 + pc5', data=data_pca)
fit = model.fit()
print(fit.summary())
```

OLS Regression Results

Dep. Variable:	Salary	R-squared:	0.754
Model:	OLS	Adj. R-squared:	0.753
Method:	Least Squares	F-statistic:	777.8
Date:	Sun, 17 Nov 2019	Prob (F-statistic):	3.05e-307
Time:	16:02:13	Log-Likelihood:	-16751.
No. Observations:	1020	AIC:	3.351e+04
Df Residuals:	1015	BIC:	3.354e+04
Df Model:	4		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	6.192e+06	1.03e+05	60.150	0.000	5.99e+06	6.39e+06
pc1	-4.502e+06	1.49e+05	-30.175	0.000	-4.8e+06	-4.21e+06
pc2	-1.186e+07	3.18e+05	-37.334	0.000	-1.25e+07	-1.12e+07
pc3	8.716e+06	4.06e+05	21.490	0.000	7.92e+06	9.51e+06
pc5	1.06e+07	5.71e+05	18.570	0.000	9.48e+06	1.17e+07

Omnibus:	87.329	Durbin-Watson:	2.028
Prob(Omnibus):	0.000	Jarque-Bera (JB):	486.173
Skew:	0.087	Prob(JB):	2.68e-106
Kurtosis:	6.378	Cond. No.	5.54

❖ PCA 분석

교차검증 실시

- Overfitting 여부를 판단하기 위해 교차 검증을 실시

- K-Fold cross validation 사용, Fold 수는 5개 사용

평균 점수 0.75로,
전체적으로 유의한 모형임을 검증

```
from sklearn.model_selection import KFold
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
```

```
data_pca_f = pd.DataFrame(data_pca,
                          columns = ['pc1', 'pc2', 'pc3', 'pc4', 'pc5','Salary'])
data_pca_x = pd.DataFrame(data_pca_f,
                          columns = ['pc1', 'pc2', 'pc3', 'pc4', 'pc5'])
data_pca_y = pd.DataFrame(data_pca_f,
                          columns = ['Salary'])
```

```
cv = KFold(5, shuffle=True, random_state=0)
model = LinearRegression()

scores = np.zeros(5)
for i, (train_index, test_index) in enumerate(cv.split(data_pca_y)):
    X_train = data_pca_x.loc[train_index]
    y_train = data_pca_y['Salary'][train_index]
    X_test = data_pca_x.loc[test_index]
    y_test = data_pca_y['Salary'][test_index]
    model.fit(X_train, y_train)
    y_pre = model.predict(X_test)
    scores[i] = r2_score(y_test, y_pre)

print(scores)
print(np.mean(scores))
```

```
[0.70362776 0.75772391 0.78363194 0.75370999 0.75776754]
0.7512922277884148
```

❖ PCA 분석

장점

- PCA분석을 통한 차원축소는 데이터셋의 밀도를 높일 수 있음
- 데이터를 간소화하여 복잡도를 낮출 수 있으며, 이를 통해 머신러닝의 Overfitting 위험을 낮출 수 있음
- 축소된 차원을 바탕으로, 고도화된 분석을 적은 리소스로 활용할 수 있음

→ 본 프로젝트에서는 간단히 OLS 모델을 사용하였지만, 더 고도화된 모델을 사용하면 더 좋은 결과를 얻을 수 있을 것이라 판단됨

한계점

- 차원축소 과정에서 분산의 손실이 일어나기 때문에, 원 데이터와의 재구성 오차가 발생함

→ 실제 본 프로젝트에서도 약 20%의 분산이 손실되었음

- 원 데이터의 차원의 수가 많다면, 원 데이터의 보존과 차원 축소 사이의 딜레마가 발생함

→ 본 프로젝트에서는 간소화 및 Adjusted R-square 점수를 위해 약 20개의 차원을 기각하였으며, 이로 인해 분산이 20% 손실됨

❖ NN 분석

"다양한 열이 포함된 복잡한 자료 속에서, 기존의 모델에 데이터를 집어넣는 방식이 아니라 대량의 데이터를 통해 예측 모델을 만들어 내기 위해 NN분석을 활용"

필요한 모듈 import

- Pandas
- Numpy
- Kearas
- matplotlib

데이터 전처리

- Lg 열의 NL과 관련한 항목을 1로, AL과 관련한 항목을 0으로 변경
- None값 제거
- Salary가 0인 행 제거
- Salary를 10000으로 나누어 단위를 '만 달러'로 맞춤

- x_data와 y_data로 각각 독립변수와 종속변수를 선언

```
import pandas as pd
import numpy as np
from keras import optimizers
from keras import models
from keras import layers
import matplotlib.pyplot as plt


data = pd.read_csv("MLB_2014_2017.csv")
data = data.drop(['Name', 'Id', 'Tm'], axis = 1)
data['Lg'] = data['Lg'].replace(['NL','MLB/NL','MLB/AL/NL'], 1)
data['Lg'] = data['Lg'].replace(['AL','MLB/AL','MLB/NL/AL'] ,0)

data = data.dropna()
data = data[data['Salary'] != 0]

data = data.astype(np.float32)
data['Salary'] = data['Salary']/10000

y_data = data['Salary']
x_data = data.drop(['Salary'], axis=1)

def norm(dataset):
    stats = dataset.describe()
    normed_dataset = (dataset - stats.mean())/(stats.std())
    return normed_dataset

normed_x_data = norm(x_data)
```

❖ NN 분석


```
model = models.Sequential()
model.add(layers.Dense(64, activation='relu', input_shape=(31,)))
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(1))
model.compile(loss='mse',optimizer=optimizers.RMSprop(lr=2e-5), metrics=['mae', 'mse'])
```

초기 딥러닝 모델 형성

- shallow 모델 (깊이 : 2)
- Optimizer를 RMS prop
  (root mean square propagation)
- Output shape을 2의 배수로 맞춤 (일반적인 방법)

⇒ 학습결과가 좋지 못함

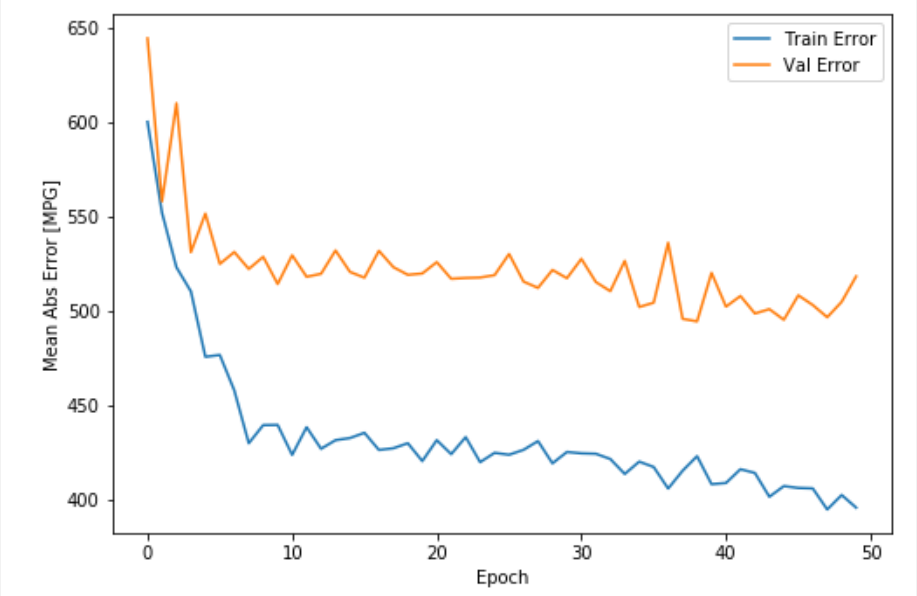
❖ NN 분석


```
model = models.Sequential()
model.add(layers.Dense(62, activation='relu', input_shape=(31,)))
model.add(layers.Dropout(0.2))
model.add(layers.Dense(124, activation='relu'))
model.add(layers.Dropout(0.2))
model.add(layers.Dense(248, activation='relu'))
model.add(layers.Dropout(0.2))
model.add(layers.Dense(496, activation='relu'))

model.add(layers.Dense(1))

model.compile(loss='mse',optimizer='adam', metrics=['mae', 'mse'])

history = model.fit(normed_x_data, y_data, epochs=50, batch_size=60, validation_split=0.2)
```

각종 시행착오를 거쳐 만든 최종 모델

- Deep learning 모델 (깊이 : 4)
- Optimizer를 가장 좋다고 알려진 ‘adam optimizer’ 로 변경
- Output shape을 31의 배수로 맞춤 (input shape과 맥락이 통하도록 함)
- 과적합을 방지하기 위해 Drop out을 사용 (cross validation과 유사한 개념)

❖ NN 분석



MAE (Mean Absolute Error)를 통해 판단

- Train Error와 Valid Error의 간격이 크게 벌어지는 지점이 과적합점
- Epochs 수를 50 / 100 / 200 / 300 / 500 등 다양하게 변경해본 결과 약 50 epoch에서 과적합이 시작된다고 판단, 최종 epoch의 수를 50으로 선정
- 최종 MAE 약 480 정도

```
hist = pd.DataFrame(history.history)
hist['epoch'] = history.epoch
plt.figure(figsize=(8,12))

plt.subplot(2,1,1)
plt.xlabel('Epoch')
plt.ylabel('Mean Abs Error [MPG]')
plt.plot(hist['epoch'], hist['mean_absolute_error'],
         label='Train Error')
plt.plot(hist['epoch'], hist['val_mean_absolute_error'],
         label = 'Val Error')
plt.legend()


plt.subplot(2,1,2)
plt.xlabel('Epoch')
plt.ylabel('Mean Square Error [$MPG^2$]')
plt.plot(hist['epoch'], hist['mean_squared_error'],
         label='Train Error')
plt.plot(hist['epoch'], hist['val_mean_squared_error'],
         label = 'Val Error')
plt.legend()


plt.show()
```

❖ NN 분석

장점

- 딥러닝 분석 방법이 상당히 고도화되었기 때문에 shallow learning에서부터 deep learning 까지 다양하게 모델을 변경할 수 있음.
  자유롭게 아주 많은 모델을 실험 가능함.
- 과적합 시점을 잘 찾으면 모델의 성능을 쉽게 파악할 수 있음.

한계점

- 딥러닝 기법에 대한 깊은 이해가 수반되지 않으면 코드를 짤 수 있다고 해도 알맞은 방향으로 분석이 되고 있는지 판단하기 어려움.
- 400대 후반 이하로 MAE가 더 이상 떨어지지 않아, 야구 데이터에는 딥러닝이 적합하지 않은 모델일 수도 있음.
- 모델의 깊이를 늘릴수록 시간이 오래 걸려서 빠른 분석에는 적합하지 않을 수 있음.

❖ **Random Forest**

"공분산을 줄임과 동시에 데이터 뿐만 아니라, 변수도 random하게 추출하여 다양한 모델을 만들기 위해 Random Forest 사용"

필요한 모듈 import

- Os, pandas, numpy, sklearn, matplotlib
- x_data와 y_data로 각각 독립변수와 종속변수를 선언
- x_data를 normalize하여 -1~1 사이값으로 변경해줌. 독립변수의 수치단위에 학습이 휘둘리는 것을 방지

```
import os
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import matplotlib as mpl
import matplotlib.pylab as plt
%matplotlib inline
```

```
# 데이터 불러오기
data = pd.read_csv("MLB_2014_2017.csv") # Test data
data.head() # 데이터 확인
```

설명변수와 타겟변수를 분리, 학습데이터와 평가데이터 분리

```
feature_columns = list(data.columns.difference(['Salary'])) # target을 제외한 모든 행
X = data[feature_columns] # 설명변수
y = data['Salary'] # 타겟변수
train_x, test_x, train_y, test_y = train_test_split(X, y, test_size = 0.2, random_state = 42)
print(train_x.shape, test_x.shape, train_y.shape, test_y.shape) # 데이터 개수 확인
```

❖ **Random Forest**


```
from sklearn.ensemble import RandomForestRegressor
random_forest_model1 = RandomForestRegressor(n_estimators = 20, # 20번 추정
                                             max_depth = 5, # 트리 최대 깊이 5
                                             random_state = 42) # 시드값 고정
model1 = random_forest_model1.fit(train_x, train_y) # 학습 진행
print(random_forest_model1.feature_importances_)
print("Accuracy on training set: {:.3f}".format(random_forest_model1.score(train_x, train_y)))
print("Accuracy on test set: {:.3f}".format(random_forest_model1.score(test_x, test_y)))
```

```
[0.00697188 0.00240423 0.00350026 0.0381704  0.36118698 0.02110064
 0.00496623 0.06979346 0.00439188 0.23168156 0.0088406  0.01792651
 0.01122099 0.00783826 0.00245681 0.01196798 0.03346579 0.00177266
 0.00966732 0.00903826 0.00509933 0.02586437 0.01587905 0.02725959
 0.01140486 0.00770285 0.00229909 0.01535602 0.00749252 0.00508639
 0.01819324]
Accuracy on training set: 0.742
Accuracy on test set: 0.435
```

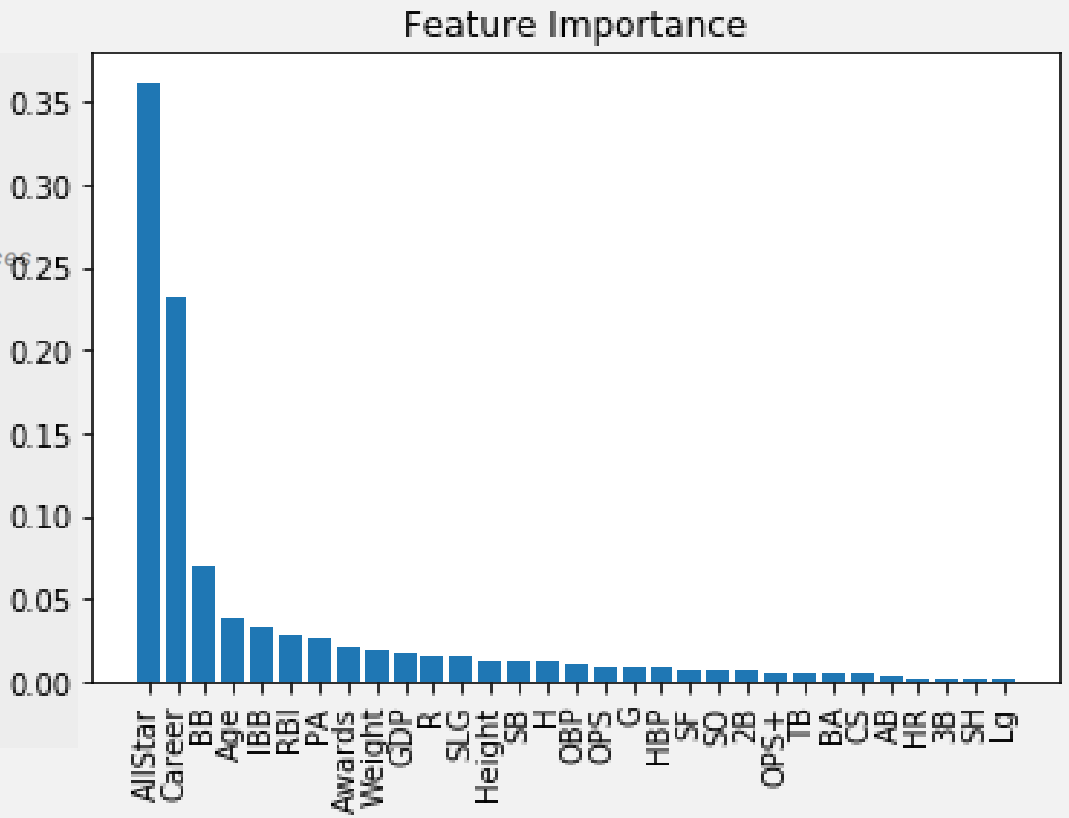
초기 학습모델 형성

- shallow 모델 (20번 추정, 트리 최대 깊이 5)
- Training set에 대해서 74.2%, test set에 대해서 43.5%의 R^2 값을 보여줌

❖ Random Forest

- 1. 분석방법
- 2. 분석결과
- 3. 장점 및 한계점

```
# Calculate feature importances
importances = random_forest_model1.feature_importances_
# Sort feature importances in descending order
indices = np.argsort(importances)[::-1]
# Rearrange feature names so they match the sorted feature importances
names = [train_x.columns[i] for i in indices]
# Create plot
plt.figure()
# Create plot title
plt.title("Feature Importance")
# Add bars
plt.bar(range(train_x.shape[1]), importances[indices])
# Add feature names as x-axis lables
plt.xticks(range(train_x.shape[1]), names, rotation = 90)
# Show plot
plt.show
```



분석결과 중요한 변수들

- 중요한 변수들 순서대로 AllStar, Career, BB, Age, IBB, RBI, PA 등이 있음을 알 수 있음

❖ **Random Forest**


```
from sklearn.ensemble import GradientBoostingRegressor
gbrt = GradientBoostingRegressor(max_depth = 5, random_state = 42) # Default로 m
gbrt.fit(train_x, train_y)
print(gbrt.feature_importances_)
print("Accuracy on training set: {:.3f}".format(gbrt.score(train_x, train_y)))
print("Accuracy on test set: {:.3f}".format(gbrt.score(test_x, test_y)))
```

```
[0.0160697  0.00714692 0.0142569  0.03174893 0.28674723 0.05343906
 0.01459373 0.05427365 0.00249906 0.17622463 0.01673498 0.0179709
 0.01948699 0.01385566 0.00234114 0.00705815 0.02977323 0.00250487
 0.01486886 0.02484369 0.0155652  0.01856065 0.02317834 0.0328289
 0.02084005 0.01443742 0.00352773 0.00842405 0.02050552 0.00783746
 0.02785641]
Accuracy on training set: 0.968
Accuracy on test set: 0.385
```

Gradient Boosting을 통해 더욱 많은 tree를 생성해 봄

- Default로 100번 추정을 하는 Gradient Boosting을 했을 때,
  Training은 96.8%로 overfitting 문제가 발생하였지만 test set에 대해서는 38.5%의 설명력을 보여줌

❖ **Random Forest**


```
random_forest_model2 = RandomForestRegressor(n_estimators = 300, # 300번 추정
                                             max_depth = 5, # 트리 최대 깊이 5
                                             random_state = 42) # 시드값 고정
model2 = random_forest_model2.fit(train_x, train_y) # 학습 진행
print(random_forest_model2.feature_importances_)
print(random_forest_model2.score(train_x, train_y)*100, "%")
print(random_forest_model2.score(test_x, test_y)*100, "%")
```

```
[0.00702047 0.00317683 0.00706912 0.03029456 0.3920263  0.03000181
 0.00931346 0.06268485 0.00367241 0.21052764 0.00896512 0.01468221
 0.01048839 0.00762911 0.00574767 0.0052934  0.02622426 0.00109299
 0.01005128 0.01455938 0.00898149 0.01733003 0.02120443 0.02097315
 0.00787584 0.00497173 0.00165853 0.01160104 0.01037064 0.00672612
 0.02778576]
74.82459285724605 %
45.60025856919962 %
```

추정횟수를 300번까지 늘려봄

- Training은 74.8%의 설명력을, test set에 대해서는 45.6%의 설명력을 보여줌

추정횟수를 300번까지 유지한채 깊이를 20까지 늘려봄

- Training은 94%로overfitting 문제가 발생하였지만 test set에 대해서는 44.6%의 설명력을 보여줌

```
random_forest_model3 = RandomForestRegressor(n_estimators = 300, # 300번 추정
                                             max_depth = 20, # 트리 최대 깊이 20
                                             random_state = 42) # 시드값 고정
model3 = random_forest_model3.fit(train_x, train_y) # 학습 진행
print(random_forest_model3.feature_importances_)
print(random_forest_model3.score(train_x, train_y)*100, "%")
print(random_forest_model3.score(test_x, test_y)*100, "%")
```

```
[0.01555242 0.00792879 0.01301619 0.03306209 0.29704112 0.03141397
 0.01684911 0.05666627 0.00898354 0.17050076 0.01723209 0.02092811
 0.01597669 0.01504389 0.01141041 0.01060543 0.02592968 0.0030554
 0.01858733 0.01905701 0.01579739 0.018763   0.02335433 0.02617407
 0.01429012 0.01235147 0.00348703 0.01647774 0.01765011 0.01165259
 0.03116184]
94.02125991739094 %
44.61338959428707 %
```

❖ Random Forest


```
random_forest_model4 = RandomForestRegressor(n_estimators = 300, # 300번 추정
                                             max_depth = 100, # 트리 최대 깊이 100
                                             random_state = 42) # 시드값 고정
model4 = random_forest_model4.fit(train_x, train_y) # 학습 진행
print(random_forest_model4.feature_importances_)
print(random_forest_model4.score(train_x, train_y)*100, "%")
print(random_forest_model4.score(test_x, test_y)*100, "%")
```

```
[0.01610103 0.0080371  0.01289933 0.03339374 0.29677866 0.03224529
 0.01699123 0.05694403 0.00871609 0.17044648 0.01669337 0.02074528
 0.01605167 0.0153315  0.01161965 0.0105832  0.02568282 0.0029191
 0.01809333 0.01845312 0.01565748 0.01901053 0.02312622 0.02553891
 0.01447508 0.01237041 0.00356635 0.01699631 0.01775056 0.01181039
 0.03097173]
93.98856431572172 %
44.75982573386506 %
```

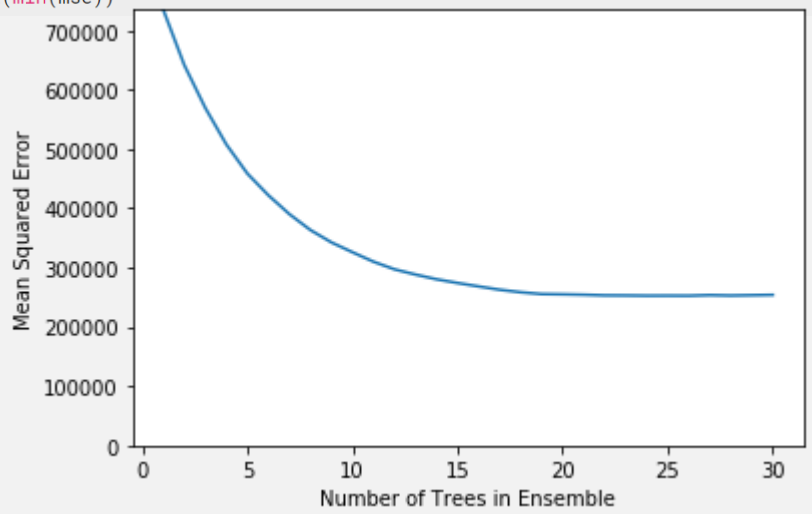
추정횟수를 300번까지 유지한채 깊이를 100까지 늘려봄

- Training은 94%로overfitting 문제가 발생하였지만 test set에 대해서는 44.7%의 설명력을 보여줌
- 유의미한 설명력의 증가가 있지 않았다

❖ Random Forest


```
#train a series of models on random subsets of the training data
#collect the models in a list and check error of composite as list grows
#maximum number of models to generate
numTreesMax = 30
#tree depth - typically at the high end
treeDepth = 5
#initialize a list to hold models
modelList = []
predList = []
eps = 0.1
#initialize residuals to be the labels y
residuals = list(train_y)
for iTrees in range(numTreesMax):
    modelList.append(DecisionTreeRegressor(max_depth=treeDepth))
    modelList[-1].fit(train_x, residuals)
    #make prediction with latest model and add to list of predictions
    latestInSamplePrediction = modelList[-1].predict(train_x)
    #use new predictions to update residuals
    residuals = [residuals[i] - eps * latestInSamplePrediction[i] \
        for i in range(len(residuals))]
    latestOutSamplePrediction = modelList[-1].predict(test_x)
    predList.append(list(latestOutSamplePrediction))
```

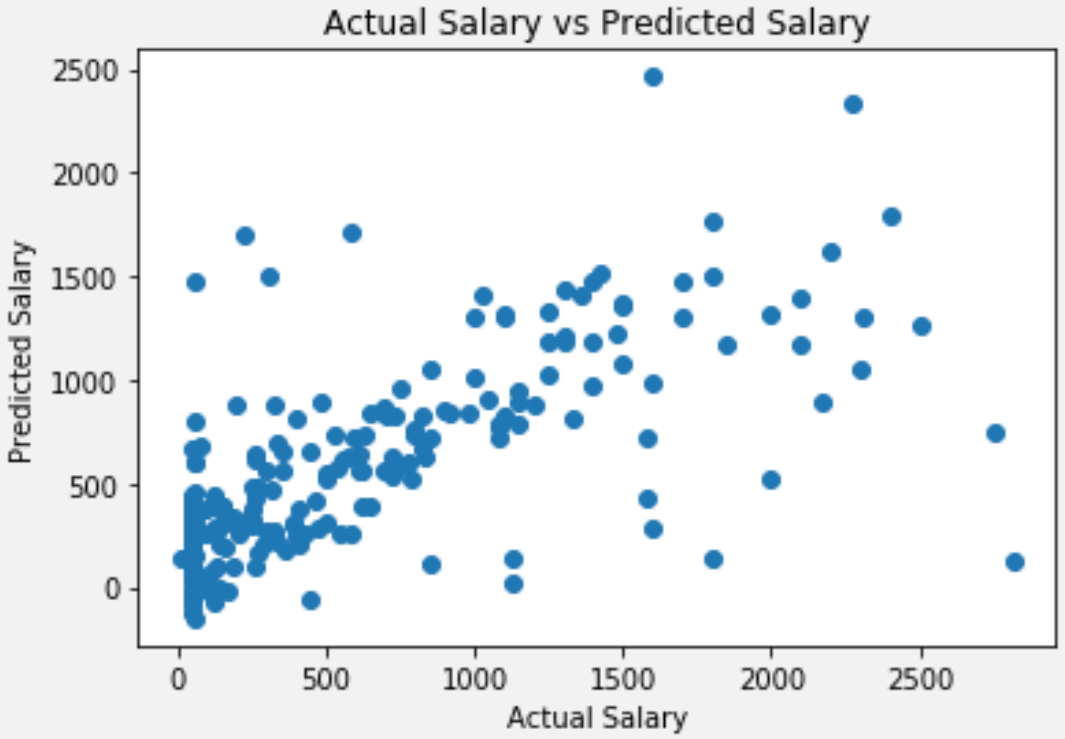
```
#build cumulative prediction from first "n" models
mse = []
allPredictions = []
for iModels in range(len(modelList)):
    #add the first "iModels" of the predictions and multiply by eps
    prediction = []
    for iPred in range(len(test_x)):
        prediction.append(sum([predList[i][iPred]
            for i in range(iModels + 1)]) * eps)
    allPredictions.append(prediction)
    errors = [(test_y.iloc[i] - prediction[i]) for i in range(len(test_y))]
    mse.append(sum([e * e for e in errors]) / len(test_y))
nModels = [i + 1 for i in range(len(modelList))]
plot.plot(nModels,mse)
plot.axis('tight')
plot.xlabel('Number of Trees in Ensemble')
plot.ylabel('Mean Squared Error')
plot.ylim((0.0, max(mse)))
plot.show()
print('Minimum MSE')
print(min(mse))
```



Random Forest가 아닌 Decision Tree Regression을 통해 추정

- 20개 Tree정도부터 의미 있는 설명력 증가가 이루어 지지 않음

❖ **Random Forest**




```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(train_x, train_y)
salary_pred = model.predict(test_x)
print('R-Squared: %.4f' % model.score(test_x,test_y))
print(model.intercept_)
print(model.coef_)
```

```
R-Squared: 0.4425
```

```
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(test_y, salary_pred)
print(mse)
plt.scatter(test_y, salary_pred)
plt.xlabel("Actual Salary")
plt.ylabel("Predicted Salary")
plt.title("Actual Salary vs Predicted Salary")
```

```
238001.53920699662
```

- Linear Regression의 경우도 44.25%의 설명력을 보여줌

❖ **Random Forest**

장점

- 과대적합(Overfitting 문제를 극복)
  : 통상적으로 의사결정나무 (Decision Tree)는 훈련 데이터에 대해서는 과대 적합을 보여주지만, 테스트 데이터에 대해서는 낮은 설명력을 지니곤 하는데 Random Forest는 과대적합된 트리를 여러 개 생성해 냄으로써 예측 성능은 유지하면서 과대적합을 줄일 수 있음

- 깊이와 추정 개수의 조절을 통해 안정적인 예측 가능

- 표본 및 변수에 따라 모형의 변화폭 및 예측력의 차이가 매우 심하게 나타나는 의사 결정 나무의 한계를 극복
  : 다른 머신러닝 기법에 비해 샘플링 되지 않은 데이터를 테스트 데이터로 이용할 수 있기 때문에 데이터 전체를 학습에 사용할 수 있으며, 의사결정 트리와 달리 일반화도 적용 가능.

한계점

- 실시간 예측을 위한 많은 수의 나무가 알고리즘의 전체 모델을 느려지거나 비 효과적으로 만들 수 있음

- 예측 모델링에는 이용되기 좋지만, 변수들 간의 설명 관계에 대해서 부적합

- 다른 머신 러닝 기법에 비해 더 이상 설명력이 늘어나지 않는 포인트가 빠르게 찾아옴. 실제 모델에서도 깊이와 추정횟수를 비약적으로 늘려봐도 설명력의 유의미한 증가가 나타나지 않음.

시사점

스포츠 선수와 같이 과업 실행 정도가 구체적으로 드러나는 경우 데이터 분석을 통해 충분히 유의한 연봉 책정 모델을 만들 수 있음

- 인사 담당자 입장에서 연봉 협상 시에 연봉 책정의 가이드라인 역할 수행
- 시즌 중의 성적을 통해 실시간으로 차년 급여 지급 계획 수립 가능
- 피고용자에게 연봉 책정의 근거 제시하여 급여 관련 불만족 완화 및 동기부여 가능

일반 인사 상황에서도 과업 활동이 구체적으로 기록되어 있다면, 성과에 따른 급여 책정 모델을 적용 가능

- 다만, 과업 및 성과의 정의가 수치적으로 치밀하게 설정되어야 함
- 실제 급여 책정 시스템 혹은 급여 협상 시의 가이드라인으로 활용 가능
- 모델의 선정에 있어서 Deep learning이 사용된 정교한 모델일수록 예측 정확도 및 유의성이 높아지지만, 절차의 직관성이 떨어지므로 급여 협상에서의 설득력이 떨어질 수 있음
- 때문에 적용 상황별로 절차의 직관성과 예측 정확도를 동시에 최대화하는 모델 파악 필요

참고자료

Meltzer, Josh. “Average Salary and Contract Length in Major League Baseball: When Do They Diverge?” 2005, Department of Economics, Stanford University, CA.

Hakes, J.K. & Turner, C. (2011) Pay, productivity and aging in major league baseball. Journal of Productivity Analysis. 35, 61 – 74.

Hochberg, Daniel. “The Effect of Contract Year Performance on Free Agent Salary in Major League Baseball” 2011, Department of Economics, Haverford College, PA

https://www.baseball-reference.com/

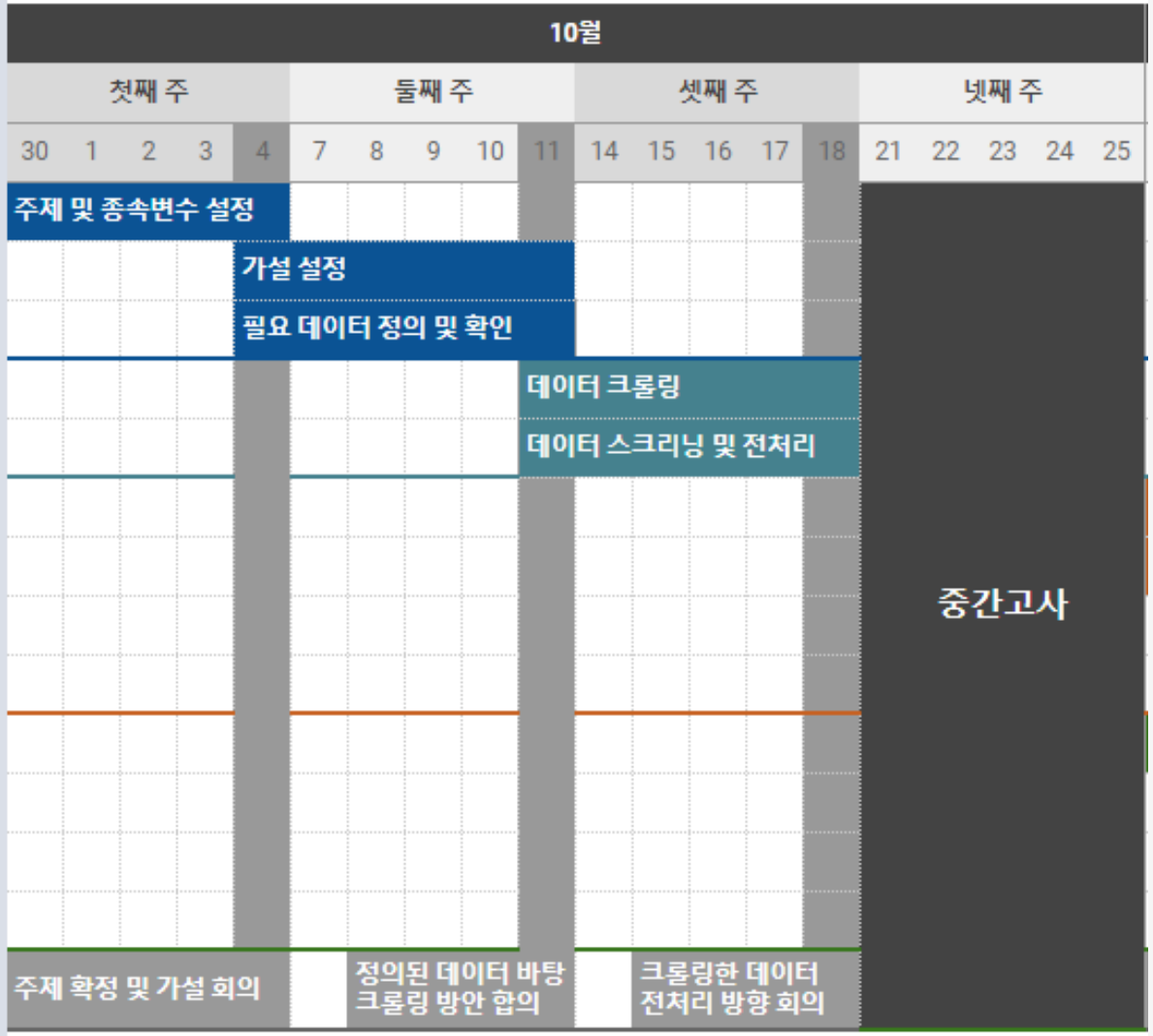
https://www.spotrac.com/

https://www.mlb.com/

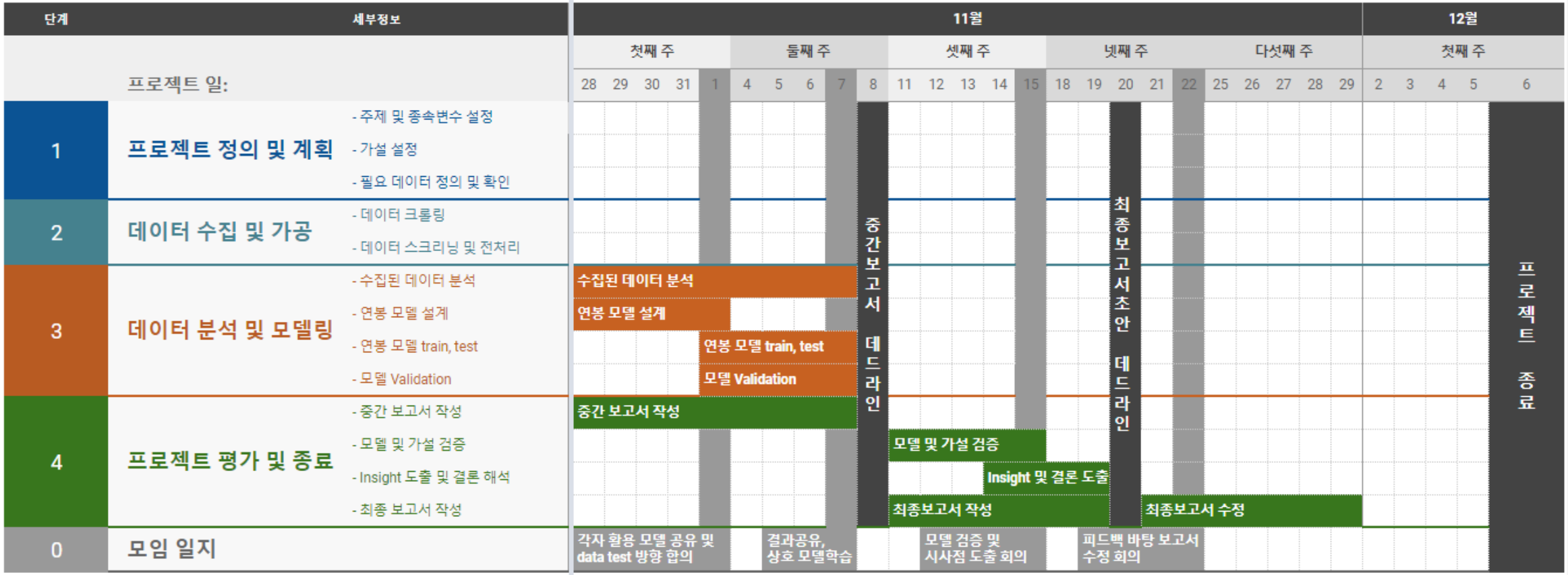
https://www.espn.com/mlb/

타임라인 및 모임일지

단계		세부정보
	프로젝트 일:	
1	프로젝트 정의 및 계획	- 주제 및 종속변수 설정 - 가설 설정 - 필요 데이터 정의 및 확인
2	데이터 수집 및 가공	- 데이터 크롤링 - 데이터 스크리닝 및 전처리
3	데이터 분석 및 모델링	- 수집된 데이터 분석 - 연봉 모델 설계 - 연봉 모델 train, test - 모델 Validation
4	프로젝트 평가 및 종료	- 중간 보고서 작성 - 모델 및 가설 검증 - Insight 도출 및 결론 해석 - 최종 보고서 작성
0	모임 일지	



타임라인 및 모임일지


단계
세부정보
11월
12월
첫째 주
둘째 주
셋째 주
넷째 주
다섯째 주
첫째 주
프로젝트 일:
28 29 30 31 1 4 5 6 7 8 11 12 13 14 15 18 19 20 21 22 25 26 27 28 29 2 3 4 5 6
1
프로젝트 정의 및 계획
- 주제 및 종속변수 설정
- 가설 설정
- 필요 데이터 정의 및 확인
2
데이터 수집 및 가공
- 데이터 크롤링
- 데이터 스크리닝 및 전처리
3
데이터 분석 및 모델링
- 수집된 데이터 분석
- 연봉 모델 설계
- 연봉 모델 train, test
- 모델 Validation
4
프로젝트 평가 및 종료
- 중간 보고서 작성
- 모델 및 가설 검증
- Insight 도출 및 결론 해석
- 최종 보고서 작성
0
모임 일지
수집된 데이터 분석
연봉 모델 설계
연봉 모델 train, test
모델 Validation
중간 보고서 작성
중간보고서 데드라인
모델 및 가설 검증
Insight 및 결론 도출
최종보고서 작성
최종보고서초안 데드라인
최종보고서 수정
프로젝트 종료
각자 활용 모델 공유 및 data test 방향 합의
결과공유, 상호 모델학습
모델 검증 및 시사점 도출 회의
피드백 바탕 보고서 수정 회의

소감

김영하

데이터 분석 및 코딩에 관한 경험이 거의 없었지만, 팀과제를 하면서 코딩 및 머신러닝에 대한 흥미가 생겼음.
그리고 데이터를 다루는데 능숙한 팀원들을 보면서 향후 산업 전반에서 필요한 인재상에 대해 생각해보게 되었음.

김지환

데이터 분석은 마케팅 혹은 재무의 특정 분야에서 주로 활용되는 technical tool 로 알고 있었지만, 이번 프로젝트를 통해 인사와 기타 모든 영역에서도 적극적으로 활용 해야 하는 필수적인 요소임을 알게 되었음. 이번 기회를 통해 데이터 분석 방법을 익힐 수 있어서 매우 만족스러움.

송규상

코드를 짤 줄은 알았지만, 데이터분석을 위해서는 분석하는 대상에 대한 배경지식이 중요하다는 것을 깨달음.
야구에 대해서 거의 알지 못해서 꽤 어려움을 겪었음.

심석현

실제로 데이터를 가지고 주제를 탐색한 뒤 구체적인 결론을 도출하는 것에 대한 경험을 조금이나마 해볼 수 있었음.
조원들과 함께 주제의 적합성은 물론, 모델의 방향과 구체적인 사항들을 심도깊게 논의하며 다양한 시각으로부터 많은 것을 얻고 배울 수 있었음. 특히, 각각의 머신러닝 기법으로 주제에 접근해가는 방법에 있어서 더 넓은 시야를 견지하게 되었고 추후 매니지먼트 분야에서 people analytics를 공부할 때 폭넓은 사고를 할 수 있는 자양분이 되었다고 생각함.

안호정

단순히 빅데이터에 대한 관심과 흥미만 갖고 수강하여, 데이터를 수집하고 분석하는 등 모든 과정이 처음이어서 조금 벅차고 매주 진도를 따라가기에 급급했으나, 이번 프로젝트르 통해 실제 활용방법을 함께 배울 수 있어서 좋았음. 단순히 빅데이터가 중요하다는 추상적인 생각을 구체화시킬 수 있었으며, 실력 있는 팀원들과 함께하면서 다양한 사고방식, 여러 분석 기법 등을 학습할 수 있었음.

이용하

데이터를 수집하고 분석하는 과정도 중요하지만, 그 이전에 연구의 목적과 이를 위한 방식을 확실하게 정하는 것이 생각보다 어렵고 중요하다는 것을 확실히 알게 된 수업이었음. 주제, 가설을 정하고 이를 검증하기 위한 방식을 정하는 과정에서 세세하게 점검하지 않아 모델을 구축하고 분석하는 과정에서 큰 모순이 발견되는 경우가 많았고, 이에 주제의 방향성과 적합성 등을 다시 한 번 점검하며 심도 있게 계획을 세우는 것이 무엇보다 중요함을 알게 되었음. 또한 다양한 머신러닝 기법을 통해 같은 데이터를 다른 방식으로 분석하여, 그 결과를 각각 비교하는 것이 중요하며, 모델들의 장점과 단점을 잘 파악하고 적절히 결합해 사용하는 방법을 알아야 한다는 것을 깨닫게 되었음.