



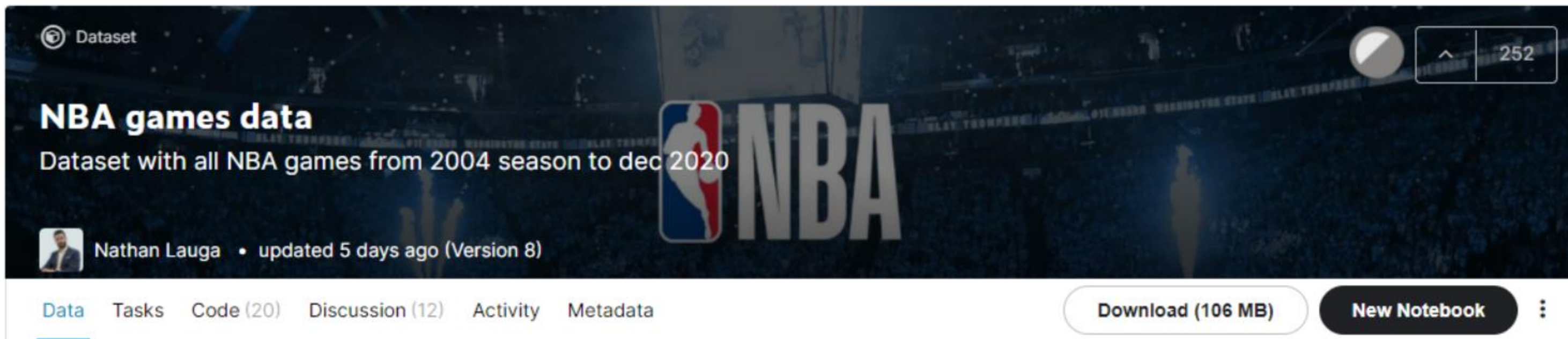
# 승부 예측

---

2019108278  
컴퓨터공학전공  
하지현

# Dataset

<https://www.kaggle.com/nathanlauga/nba-games>



**NBA games data**  
Dataset with all NBA games from 2004 season to dec 2020

Nathan Lauga • updated 5 days ago (Version 8)

Data Tasks Code (20) Discussion (12) Activity Metadata

Download (106 MB) New Notebook

## Data Explorer

105.65 MB

- games.csv
- games\_details.csv
- players.csv
- ranking.csv
- teams.csv

2003년~2021년 NBA 경기 데이터  
업데이트: 2021-11-18

games.csv: 경기에 관한 전반적인 데이터 - 21칼럼  
games\_details.csv: 보다 자세한 데이터(출전 선수 및 포지션 포함) - 29칼럼  
players.csv: 선수 데이터 - 4칼럼  
ranking.csv: 리그 당 팀 순위 데이터 - 13칼럼  
teams.csv: 팀 데이터 - 14칼럼

# 사용 데이터 및 내용

## games.csv

```
df = pd.read_csv("/kaggle/input/nba-games/games.csv")
df.head()
```

	GAME_DATE_EST	GAME_ID	GAME_STATUS_TEXT	HOME_TEAM_ID	VISITOR_TEAM_ID	SEASON	TEAM_ID_home	PTS_home	FG_PCT_home	FT_PCT_home	...	AST_home	REB_home	TEAM_ID_away	PTS_away	FG_PCT_away	FT_PCT_away
0	2021-11-17	22100213	Final	1610612766	1610612764	2021	1610612766	97.0	0.438	0.500	...	30.0	59.0	1610612764	87.0	0.367	0.813
1	2021-11-17	22100214	Final	1610612765	1610612754	2021	1610612765	97.0	0.425	0.750	...	16.0	42.0	1610612754	89.0	0.418	0.737
2	2021-11-17	22100215	Final	1610612737	1610612738	2021	1610612737	110.0	0.506	0.833	...	28.0	40.0	1610612738	99.0	0.440	0.824
3	2021-11-17	22100216	Final	1610612751	1610612739	2021	1610612751	109.0	0.458	0.840	...	29.0	47.0	1610612739	99.0	0.393	0.857
4	2021-11-17	22100217	Final	1610612748	1610612740	2021	1610612748	113.0	0.483	0.824	...	29.0	39.0	1610612740	98.0	0.440	0.786

5 rows × 21 columns

경기 id, 날짜, 홈 팀 id, 원정 팀 id, 시즌, 득점 정보, 홈팀 승리 여부 외 21칼럼

SVM 모델 사용 - 완전한 예측X, 승리 여부를 분류하는 식으로 진행



# 데이터 전처리

2003년 데이터 NaN 값 有 -> 2004년 데이터부터 사용

id(숫자) -> Nickname(문자열)으로 replace

	GAME_DATE_EST	GAME_ID	GAME_STATUS_TEXT	HOME_TEAM_ID	VISITOR_TEAM_ID	SEASON
0	2004-01-02	20300458	Final	Timberwolves	Hawks	2003
1	2004-01-02	20300451	Final	Wizards	Warriors	2003
2	2004-01-02	20300454	Final	Knicks	Bulls	2003
3	2004-01-02	20300457	Final	Pistons	Suns	2003
4	2004-01-02	20300456	Final	Heat	Magic	2003

2004년 이후 데이터 확인

id값 변경 확인

```
df = df.sort_values(by='GAME_DATE_EST').reset_index(drop = True)

df = df.loc[df['GAME_DATE_EST'] >= "2004-01-01"].reset_index(drop=True)
#null 체크
df.isnull().values.any()
null 확인
```

False

```
# 'HOME_TEAM_ID' 'VISITOR_TEAM_ID' name으로 변경

df_names = df_names[['TEAM_ID', 'NICKNAME']]
home_names = df_names.copy()
home_names.columns = ['HOME_TEAM_ID', 'NICKNAME']

result_1 = pd.merge(df['HOME_TEAM_ID'], home_names, how = "left", on="HOME_TEAM_ID")
df['HOME_TEAM_ID'] = result_1['NICKNAME']

visitor_names = df_names.copy()
visitor_names.columns = ['VISITOR_TEAM_ID', 'NICKNAME']
result_2 = pd.merge(df['VISITOR_TEAM_ID'], visitor_names, how = "left", on="VISITOR_TEAM_ID")
df['VISITOR_TEAM_ID'] = result_2['NICKNAME']
```

# 데이터 전처리

```
df = df.loc[df['GAME_DATE_EST'] < '2020-08-01'].reset_index(drop=True)|
```

```
df_ = df.loc[df['GAME_DATE_EST'] > '2019-10-21'].reset_index(drop=True)
```

2019-2020 시즌 데이터

```
selected_features = [  
    'FG_PCT_home', 'FT_PCT_home', 'FG3_PCT_home', 'AST_home', 'REB_home',  
    'FG_PCT_away', 'FT_PCT_away', 'FG3_PCT_away', 'AST_away', 'REB_away',  
]
```

경기 데이터 내의 특성들  
(경기 점수에 직접적 영향 내용)

```
X = df[selected_features]  
y = df['HOME_TEAM_WINS']
```

```
X_train, X_test, y_train, y_test = train_test_split(  
    X,  
    y,  
    test_size=0.5, random_state=42)
```

훈련, 테스트 셋 분리

# 학습

---

```
clf = svm.SVC(kernel='linear')  
clf.fit(X_train, y_train) # fi
```

학습 SVM - SVC 사용  
(classification)

```
y_pred = clf.predict(X_test)  
print('balanced accuracy score:', balanced_accuracy_score(y_test, y_pred))  
balanced accuracy score: 0.6880731208944275
```

```
clf.score(X_test, y_test)
```

평가

```
0.7252901864227929
```



# simulation

```
unique_teams = df['HOME_TEAM_ID'].unique() # extract all the unique teams
|
all_team_sim_data = {}

for team_name in unique_teams:

    df_team = df_.loc[(df_['HOME_TEAM_ID'] == team_name) | (df_['VISITOR_TEAM_ID'] == team_name)]
    df_1 = df_team.loc[df_team['HOME_TEAM_ID'] == team_name][selected_features[:5]]
    df_0 = df_team.loc[df_team['VISITOR_TEAM_ID'] == team_name][selected_features[5:]]

    df_0.columns = df_1.columns
    df_s = pd.concat([df_1, df_0], axis = 0)

    # convert the pandas.DataFrame to numpy array
    all_team_sim_data[team_name] = df_s.to_numpy()
```

```
group_list = [
    # Eastern Conference
    ('Bucks', 'Magic'), # group A
    ('Pacers', 'Heat'), # group B

    ('Celtics', '76ers'), # group C
    ('Raptors', 'Nets'), # group D

    # Western Conference
    ('Lakers', 'Trail Blazers'), # group E
    ('Rockets', 'Thunder'), # group F
    |
    ('Nuggets', 'Jazz'), # group G
    ('Clippers', 'Mavericks')] # group H
```

홈 경기 / 원정 경기에 따라 위의 특성들을  
나눠서 넣어줌

```
selected_features = [
    'FG_PCT_home', 'FT_PCT_home', 'FG3_PCT_home', 'AST_home', 'REB_home',
    'FG_PCT_away', 'FT_PCT_away', 'FG3_PCT_away', 'AST_away', 'REB_away',
]
```

동부/서부 컨퍼런스 그룹 나눔

```

def __init__(self, random_state = None):

    self.random_state = random_state # keep this to None for making simulations

def predict(self, team1, team2, num_games = 1):

    ''' predict the win or loss of n game(s) played by two teams'''

    assert num_games >= 1, "at least one game must be played"
    # output numpy array
    team_1_feature_data = DATA[team1]
    team_2_feature_data = DATA[team2]
    features = []
    for feature_paras_1 in team_1_feature_data:
        sample_1 = self.sampling(feature_paras_1, size = num_games) # gives a list if num_games> 1
        features.append(sample_1)

    for feature_paras_2 in team_2_feature_data:
        sample_2 = self.sampling(feature_paras_2, size = num_games) # gives a list if num_games> 1
        features.append(sample_2)

    features = np.array(features).T
    win_loss = DIS.predict(features)

    return list(win_loss) # a list of win/loss from num_games

def sampling(self, dic, size = 1, random_state = None):

    '''generate feature values used for making win/loss prediction'''

    dis_name = list(dic.keys())[0] # get the type
    paras = list(dic.values())[0] # get the paras

    # get sample
    sample = GEN[dis_name](*paras, size = size, random_state = random_state)

    return sample

```

```

playoff = FinalTournament()
# simulate the playoff 5,000 times
playoff.simulate(group_list, n_simulation = 5000)

```



```

def simulate(self, group_list, n_simulation = 1, probs = True):

    ''' simulate the entire playoff n times and also record the accumulated wins'''

    # update the list of teams
    self.rounds = {}
    self.team_list = [i[0] for i in group_list] + [i[1] for i in group_list]

    for i in range(n_simulation):
        cham = self.one_time_simu(group_list)
    if probs:
        self.rounds_probs = self._compute_probs()

def one_time_simu(self, group_list, verbose = False, probs = False):

    ''' simulate the entire playoff once and also record the accumulated wins'''

    # update the list of teams if haven't done so
    if self.team_list == None:
        self.team_list = [i[0] for i in group_list] + [i[1] for i in group_list]
    round_number, done = 0, 0
    while not done:
        all_group_winners, group_list = self.play_round(group_list)
        # retrieve round stats
        try:
            updated_round_stats = self.rounds[round_number]
        except KeyError:
            updated_round_stats = {}
            for team in self.team_list:
                updated_round_stats[team] = 0
        # if a team wins, record + 1
        for winner in all_group_winners:
            try:
                updated_round_stats[winner] += 1
            except KeyError:
                pass
        self.rounds[round_number] = updated_round_stats
        if verbose:
            print('{} round played'.format(round_number))
        if probs:
            self.rounds_probs = self._compute_probs()
        if type(group_list) != list: # if it becomes the final
            done = 1
        round_number += 1

    return group_list

```

```

def play_round(self, group_list):

    '''play a round of games based of a list of paired teams'''

    all_group_winners = []
    # play each group and get the group winner
    for group in group_list:
        winner = self.play_n_games(group[0], group[1])
        all_group_winners.append(winner)

    if len(all_group_winners) > 1:
        new_group_list = []
        for index in range(0, len(all_group_winners), 2):
            # first winner, second winner
            new_group = [all_group_winners[index], all_group_winners[index + 1]]
            new_group_list.append(new_group)

        return all_group_winners, new_group_list
    else:
        return all_group_winners, winner

def play_n_games(self, team1, team2):

    '''simulate data, and then use our classifier to predict win/loss'''

    result = Game().predict(team1, team2, self.n_games_per_group)
    if sum(result[:4]) == self.winning_threshold or sum(result) >= self.winning_threshold:
        winner = team1 # home team wins
    else:
        winner = team2 # visitor team wins

    return winner

def _compute_probs(self):

    '''prob = wins for a team / sum of wins for all teams at a particular round'''

    rounds_probs = copy.deepcopy(self.rounds)
    for round_number, round_stats in rounds_probs.items():
        m = np.sum(list(round_stats.values()))
        for k, v in rounds_probs[round_number].items():
            rounds_probs[round_number][k] = v / m

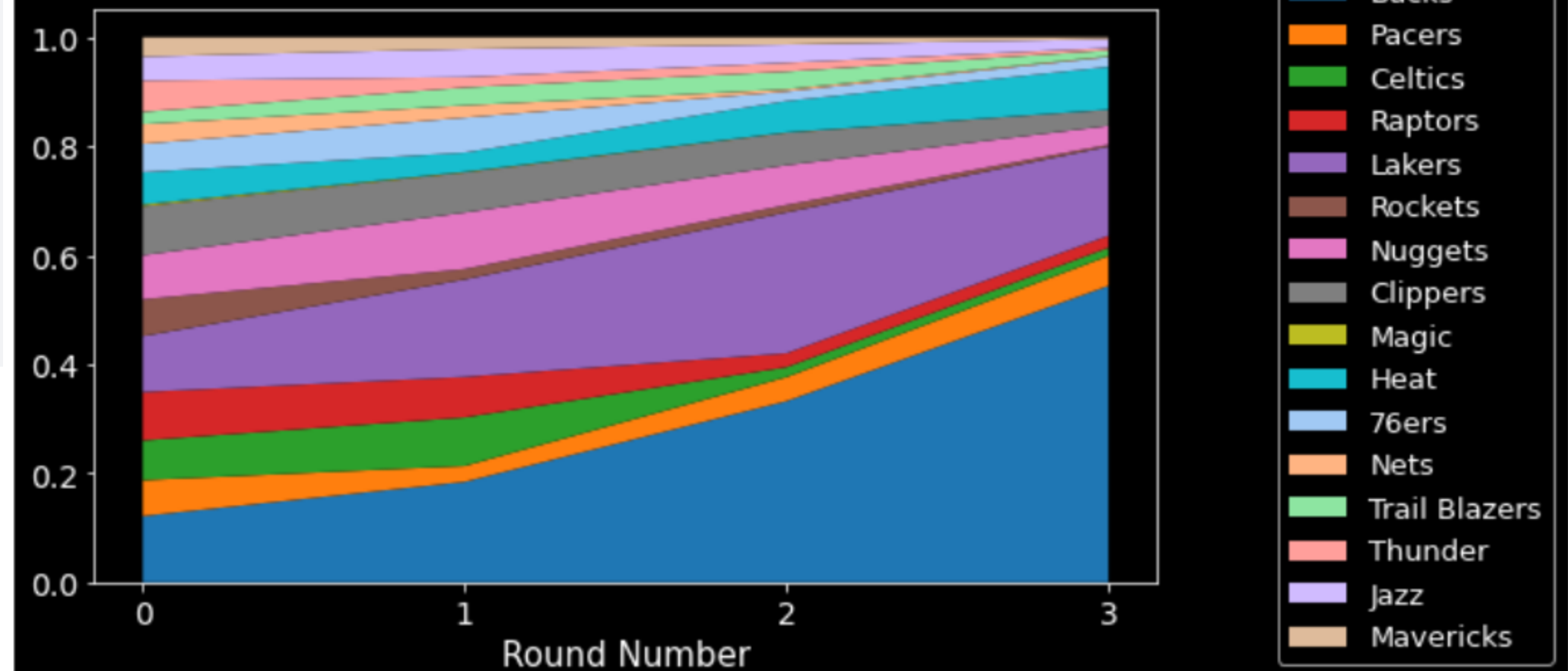
    return rounds_probs

```

# 시각화

```
def plotting(rounds_data):  
  
    rounds_stats = list(rounds_data.values())  
    team_names = list(rounds_stats[0].keys())  
  
    # x is number of rounds used for labels, y is a 2-D array of (n_teams, n_rounds) used for data  
    x = list(rounds_data.keys())  
    y = np.array([list(r.values()) for r in rounds_stats]).T  
  
    # we need at least 16 different colors, one for each team  
    c_1 = sns.color_palette('tab10', n_colors = 10)  
    c_2 = sns.color_palette("pastel", n_colors = 10)  
    color_map = c_1 + c_2  
  
    fig = plt.figure()  
    plt.stackplot(x, y, labels = team_names, colors = color_map)  
    plt.legend(bbox_to_anchor=(1.1, 1.1), loc = 'upper left', fontsize=13)  
    plt.xticks(x, fontsize=14)  
    plt.yticks(fontsize=14)  
    plt.xlabel('Round Number', fontsize = 15)  
    plt.title('Winning probabilities by all Teams & Rounds', pad = 20, fontsize = 24)  
    plt.tight_layout()  
    plt.show()  
  
    return fig
```

Winning probabilities by all Teams & Rounds





# 승리 확률

```
df_br_1 = df.loc[(df['HOME_TEAM_ID'] == 'Bucks') & (df['VISITOR_TEAM_ID'] == 'Raptors')].reset_index(drop=True)
df_br_2 = df.loc[(df['HOME_TEAM_ID'] == 'Raptors') & (df['VISITOR_TEAM_ID'] == 'Bucks')].reset_index(drop=True)
```

```
print('Bucks won {} out of {} when being the home team'.format(
    sum(df_br_1['HOME_TEAM_WINS']), df_br_1.shape[0]))
print('Bucks won {} out of {} when being the away team'.format(
    df_br_2.shape[0] - sum(df_br_2['HOME_TEAM_WINS']), df_br_2.shape[0]))
```

```
Bucks won 18 out of 35 when being the home team
Bucks won 16 out of 36 when being the away team
```

```
print("Bucks against Raptors: ", (sum(df_br_1['HOME_TEAM_WINS'])+df_br_2.shape[0] - sum(df_br_2['HOME_TEAM_WINS']))/(df_br_1.shape[0]+df_br_2.shape[0]))
```

```
Bucks against Raptors: 0.4788732394366197
```



# 소감

---

처음 생각했던 것과 다르게 가야해서 계산 위주로 하게 된 점, 선택한 데이터의 코드가 생각보다 많이 없기도 하고, 내용이 어려워서 코드를 100% 이해하지 못한 점들이 아쉬웠다. 특히나 예상했던 예측보다는 분류 후 계산의 과정이라는 점이 제일 아쉽고 부족한 부분이었던 것 같다. 그냥 보고 정해진 튜토리얼을 따라가는 것보다 분석하면서 진행하게 되어 이해도가 많이 올라갔다고 생각한다. 생각보다 재미있어서 즐겁게 진행했다.



**Q&A?**

---