The Biggest Disappointments of Modern Basketball

By Evan Meyer

TABLE OF CONTENTS

01 Overview

02 Datasets

03 Cleaning

04 Process

05 Scoring

06 Insights

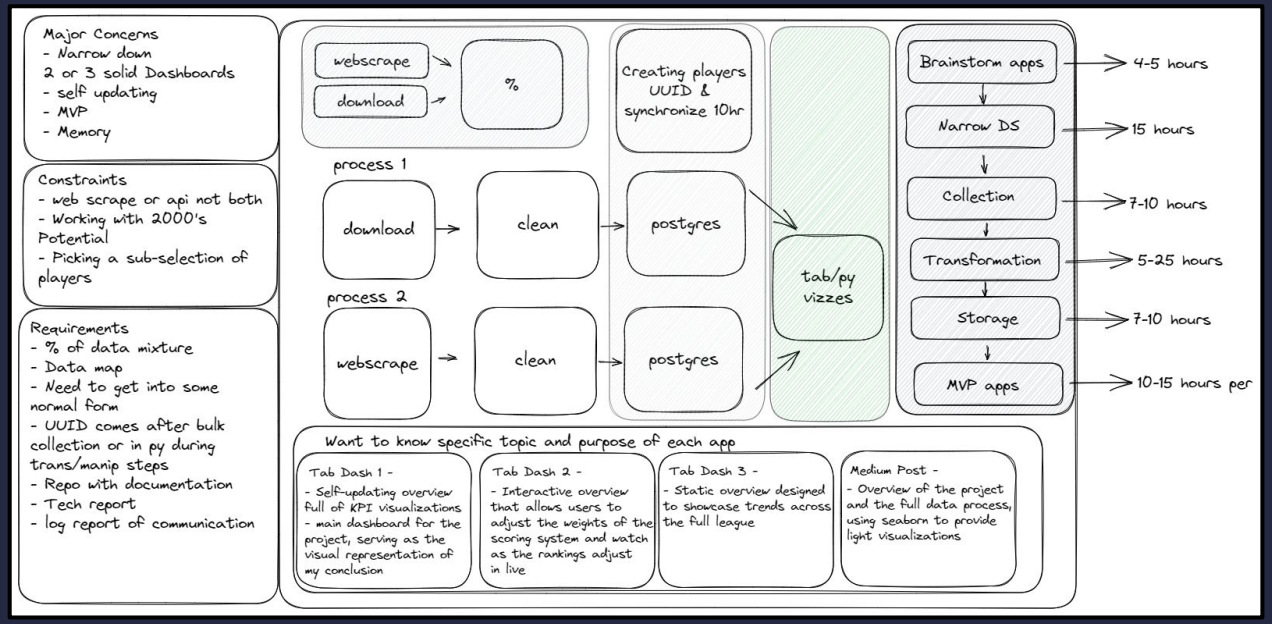
07 Dashboard

08 Next Steps

Project Overview

- Determining the NBA's worst season since 1996
- Attempted to define "disappointment" quantifiably, by comparing production to expectations
- Project took about 2 weeks and 1-200 hours to complete
- Tools Used:
  - Python for data cleaning and wrangling
  - postgreSQL for scoring and analysis
  - Tableau for interactive visualizations

Project Overview



Original Project Plan


Data

Datasets

- All datasets were webscrapped:
  - Hoopshype annual player salary data
  - Hoopshype team salary cap data
  - Basketball Reference end-of-season player stats
  - Basketball Reference draft + lottery order data
  - ESPN RPM + Wins data
  - ESPN annual awards data

Web Scraping

Standard Scrapping

```
var_name = f'stats_{year - 1}'
table_list.append(var_name)

table_soup = soup(requests.get(f'http://www.espn.com/nba/statistics/rpm/_/year/{year}/page/1').text, 'html.parser')
page_count = int(str(table_soup.select('div.page-numbers')).split('of ')[1].split('<')[0])
table = pd.read_html(str(table_soup.select('table.tablehead')))[0]
for i in range(2, page_count + 1):
    table_soup = soup(requests.get(f'http://www.espn.com/nba/statistics/rpm/_/year/{year}/page/{i}').text, 'html.parser')
    table = pd.concat([table, pd.read_html(str(table_soup.select('table.tablehead')))[0]])
```

Grabbing headers to set table names

```
#set variable name to the first two capitalized words of the header
if len(name) > 1:
    for word in name:
        if word[0] != word[0].upper():
            name.remove(word)
    var_name = (name[0] + '_' + name[1]).lower().replace('-', '_')
else:
    var_name = ''.join(name).lower()

table_list.append(var_name)

#grab the datset
globals()[var_name] = pd.read_html(requests.get(f'http:{link}').text)[0]
```

Iterating through links

Wrote functions that first grabbed a list of website links and scrapped a unique dataframe from each, concatenating them all into one table

```
for i in range(start_year, end_year + 1):
    #define our variables
    year = f'{i}-{i + 1}/'
    var_name = f'team_salaries_{i}'
    teams = []

    #scrap all of the website links to the per-team data
    main = soup(requests.get(f'https://hoopshype.com/salaries/{year}').text, 'html.parser')
    links = [a for a in main.find_all('tbody')[0].find_all('a', href=True)]

    #iterate through each link to grab website + team name
    for link in links:
        website = str(link).split('"')[1]
        team = str(link).split('"')[-1].replace('\t', '').replace('</a>', '').replace('>\n', '')
        table = pd.read_html(website)[0]
```

```
    #iterate through and concat our teams list to get one final table for the year
    globals()[var_name] = pd.concat(teams)
    print(f'{var_name} cleaned!')
    table_list.append(var_name)
    temp_list.append(globals()[var_name])
```

Data Cleaning

UUIDS

Generated unique ids for all players using the python ‘uuid’ package

```
def generate_index(table):
    player_index = pd.DataFrame(table, columns = ['player'])
    for name in player_index['player'].unique():
        player_index.loc[player_index['player'] == name, 'uuid'] = uuid.uuid4()
    return player_index
```

Fixing columns + more

```
def clean_table(table):
    table.columns = table.iloc[0].str.lower()
    table = table.tail(-1)
    table['temp'] = table['name']
    table['name'] = table.apply(just_name, axis=1)
    table['temp'] = table.apply(just_pos, axis=1)
    table.drop(columns=['rk'], inplace=True)
    table.rename(columns={'temp': 'pos', 'name': 'player'}, inplace=True)
    cols = table.columns.values
    cols = [cols[i] for i in [0, 8, 1, 2, 3, 4, 5, 6, 7]]
    table = table[cols]
    table = pd.merge(left=table, right=player_index, how='left', on='player')
    table = table[table['player'] != 'NAME']

    for bad_name in table['player']:
        if bad_name in [x for x in null_players['original_name']]:
            table.loc[table[table['player'] == bad_name].index, 'player'] = [
                x for x in null_players[null_players['original_name'] == bad_name]['new_name']][0]

    #table.dropna(inplace=True)
    table.reset_index(drop=True)

    table = update_wins(table)
    return table
```

Apply

Applying functions to create new categorical rows

```
def perc_score(row):
    if round(row['perc'] * 10) == 0:
        outcome = (row['perc'] * 100)
    else:
        outcome = (row['perc'] * 100) * row['perc'] * 10
    return round(outcome, 3)
```

```
def just_name(row):
    return str(row['name']).split(', ')[0].split('  ')[-1]

def just_pos(row):
    return str(row['temp']).split(', ')[-1]
```

Handling Nulls

Wrote a function to read in and locate all nulls from all tables:

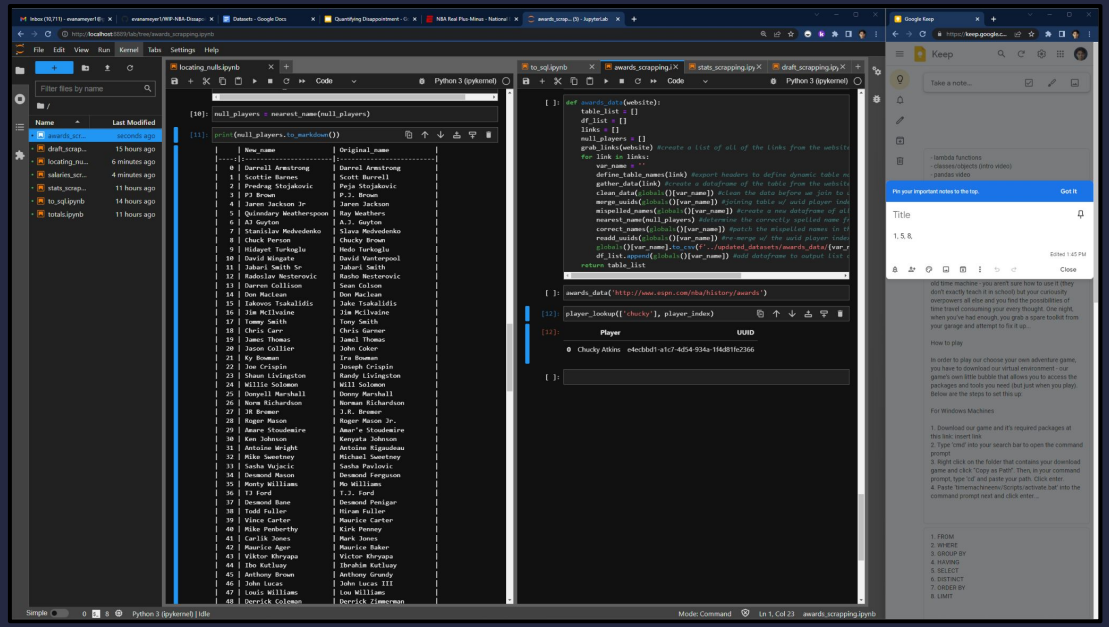
```
def locate_nulls():
    null_players = awards_data()
    null_players = pd.concat([stats_data(), null_players]).drop_duplicates()
    null_players = pd.concat([draft_data(), null_players]).drop_duplicates().reset_index(drop=True)
    null_players = null_players[null_players['Player'] != 'NAME']
    return null_players
```

Then, handled each specific null differently:

- Replaced statistical nulls with 0s
- Dropped rows with nulls implying lack of play time (nulls for salary)
- Patched misspelled names with their correct spelling (below)

```
def nearest_name(players):
    nearest_name = []
    original_name = []
    for name in players['Player']:
        close_matches = difflib.get_close_matches(name, player_index['Player'], 1)
        for x in close_matches:
                nearest_name.append(x)
                original_name.append(name)
    nearest_name = pd.DataFrame(nearest_name, columns=['New_name'])
    nearest_name['Original_name'] = original_name
    return nearest_name
```

Handling Nulls



Using python markdown to locate misspelled names

Handling Duplicates

```python
def remove_duplicates(tables):
    cleaned_df_list = []
    for i in range(len(tables)):
        years = [year for year in range(1996, 2022)]
        var_name = f'season_totals_{years[i]}'
        globals()[var_name] = tables[i][tables[i].duplicated(subset=['player'], keep='first') == False].reset_index(drop=True)
        cleaned_df_list.append(globals()[var_name])
        globals()[var_name].to_csv(f'../updated_datasets/season_totals/{var_name}.csv')
        print(f'{var_name} is done!')
    return cleaned_df_list
```

Removing all but the first iteration of a player for players that were on multiple teams for one season

To SQL

Read in csvs and loaded them into SQL as tables

```python
def load_to_sql(tables, directory):
    table_list = []
    for i in [year for year in range(1996, 2022)]:
        track = []
        var_name = f'{tables}_{i}'
        #load in the csvs
        globals()[var_name] = pd.read_csv(f'../updated_datasets/{directory}/{tables}_{i}.csv')
        globals()[var_name] = globals()[var_name][globals()[var_name].columns[1:]]
        track.append(var_name)
        #upload the csvs to sql
        db = create_engine('postgresql+psycopg2://postgres:password@localhost:5432/nba_disappointments')
        conn = db.connect()

        start_time = time.time()
        globals()[var_name].to_sql(f'{var_name}', con=conn, if_exists='replace', index=False)
        track.append(f'to_sql duration: {time.time() - start_time} seconds')
        #output a list of table names and upload durations
        table_list.append(track)
    return table_list
```

Used list comprehension to union table across all years together

```python
with conn.cursor() as cur:

    # Generate the SELECT statement to union all tables from 1996 to 2021
    select_query = ' UNION ALL '.join([f'SELECT * FROM player_scores_{year}' for year in range(1996, 2022)])

    # Insert the selected data from all tables into the new table
    cur.execute(f'''create table player_scores as (
    SELECT * FROM ({select_query}) AS all_scores);''')

    # Commit the changes to the database
    conn.commit()

# Close the database connection
conn.close()
```

Used for loops to write queries for each table

```python
for year in range(1996, 2022):
    with conn.cursor() as cur:
        cur.execute(f'''
        CREATE TABLE player_scores_{year} AS (
            select
                sub.uuid,
                sub.year,
                sub.team,
                sub.pos,
                sub.player,
                sub.salary,
                sub.percent_of_team,
                sub.wins as wins_contributed,
                case when sub.relative_wins = 0 then 0 else (sub.percent_of_team / sub.r
                from (
                    with t1 as (
                        select
                            avg(adjusted_wins)::Decimal(6,2) as avg_wins
                            from stats_{year}
                    ),

                    t2 as (
                        select
                            sal.uuid,
                            sal.year,
                            sal.team,
                            stat.pos,
                            sal.player,
                            salary,
                            (perc)::decimal(6,4) as percent_of_team,
                            perc_score as salary_score,
                            wins,
                            (adjusted_wins)::Decimal(6,2) as adjusted_wins
                            from team_salaries_{year} sal
                            left join stats_{year} stat
                                using(uuid)
                    )

                    select
                        t2.*,
                        (t2.adjusted_wins / t1.avg_wins)::decimal(6,4) as relative_wins
                        from t1, t2
                ) sub
                where wins IS NOT NULL
                order by 8 desc
        );
        ''')

    # Commit the changes to the database
    conn.commit()
```


Scoring +
Insights

Scoring

Expectations:

- Salary-based
- Relative to their team salary cap

Production:

All success metrics were pulled directly from ESPN's website. While they don't disclose the formula used to get these numbers, they offer the following explanations.

- RPM
  - Player's estimated on-court impact on team performance, measured in net point differential per 100 offensive and defensive possessions. RPM takes into account teammates, opponents and additional factors
- WINS
  - "RPM Wins" provide an estimate of the number of wins each player has contributed to his team's win total on the season. RPM Wins include the player's Real Plus-Minus and his number of possessions played.

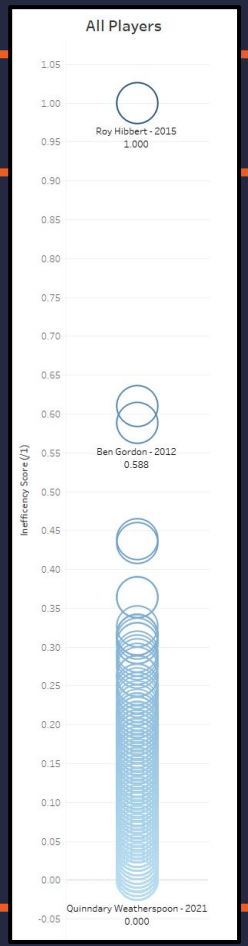
Scoring

- Each player received an “inefficiency score” - expectation score / production score
- Due to the WINS column being negative as well as positive, the scores all had to be normalize to be properly compared:

```
def update_wins(table):
    table['adjusted_wins'] = pd.to_numeric(table['wins']) + abs(min(pd.to_numeric(table['wins']))) + 1
    return table
```

- Once I unioned all the player inefficiency scores (labeled production score in the code below), I had to normalize the production score so everything was out of 1

```
-- normalize scores --
UPDATE player_scores
SET production_score = ((production_score - min_score) / (max_score - min_score))::decimal(5,4)
FROM (SELECT MIN(production_score) AS min_score, MAX(production_score) AS max_score FROM player_scores) AS sub;
```



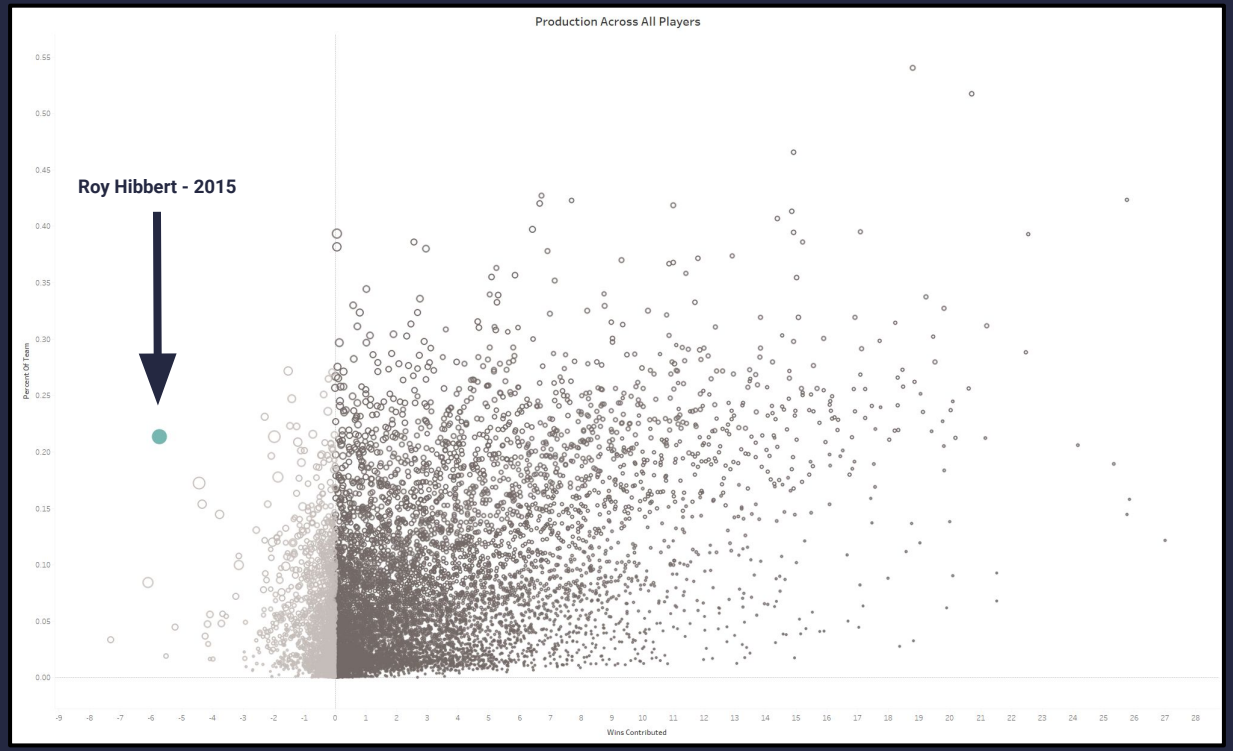
Insights

- Roy Hibbert in 2015 was the NBA's most inefficient player by FAR
    - The second place player had a score that was 61% Roy Hibbert's score
- A large majority of players were in the lower quartile
- The average inefficiency score of players was 0.032
- Roy Hibbert's 2015 stats compared with 2015's averages are below:

Player	Team	Position	Pts/Game	Ast/Game	Reb/Game	Salary	Percent Of Team	Wins Contributed	
Roy Hibbert - 2015	LAL	C	5.94	1.17	4.91	15,514,031	0.2134	-5.71	1.000

Season	Avg. Pts/Game	Avg. Ast/Game	Avg. Reb/Game	Avg. Salary	Avg. Percent Of Team	Avg. Wins Contributed	
2015	8.304004525	1.849570136	3.585769231	4,614,451.26696833	0.058450452	1.990497738	0.03118

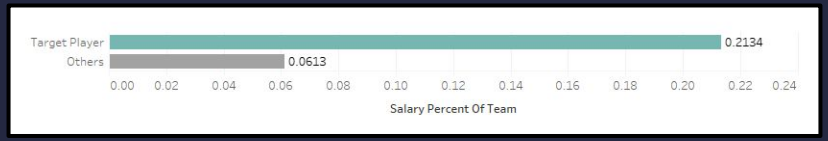
Insights



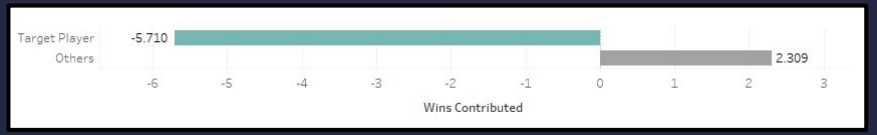
The production of all players compared with their percent of their team's salary

Just How Bad Was He?

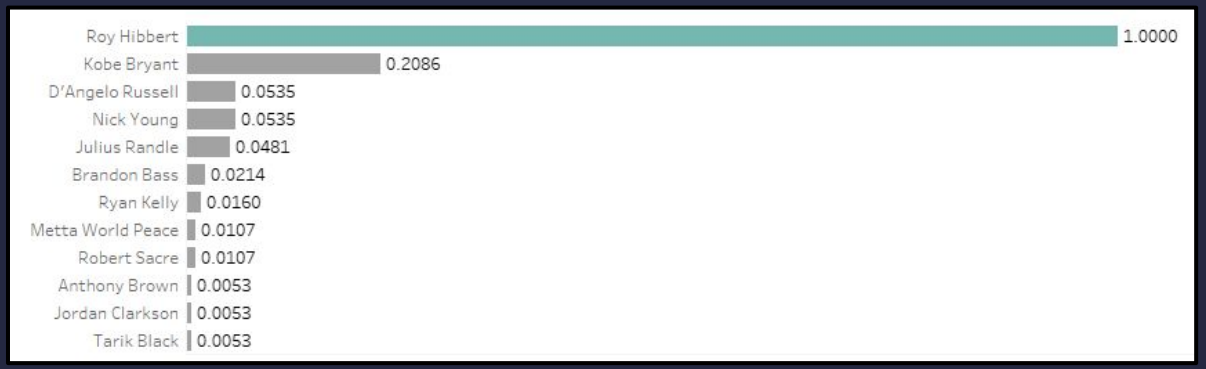
Expectation Score



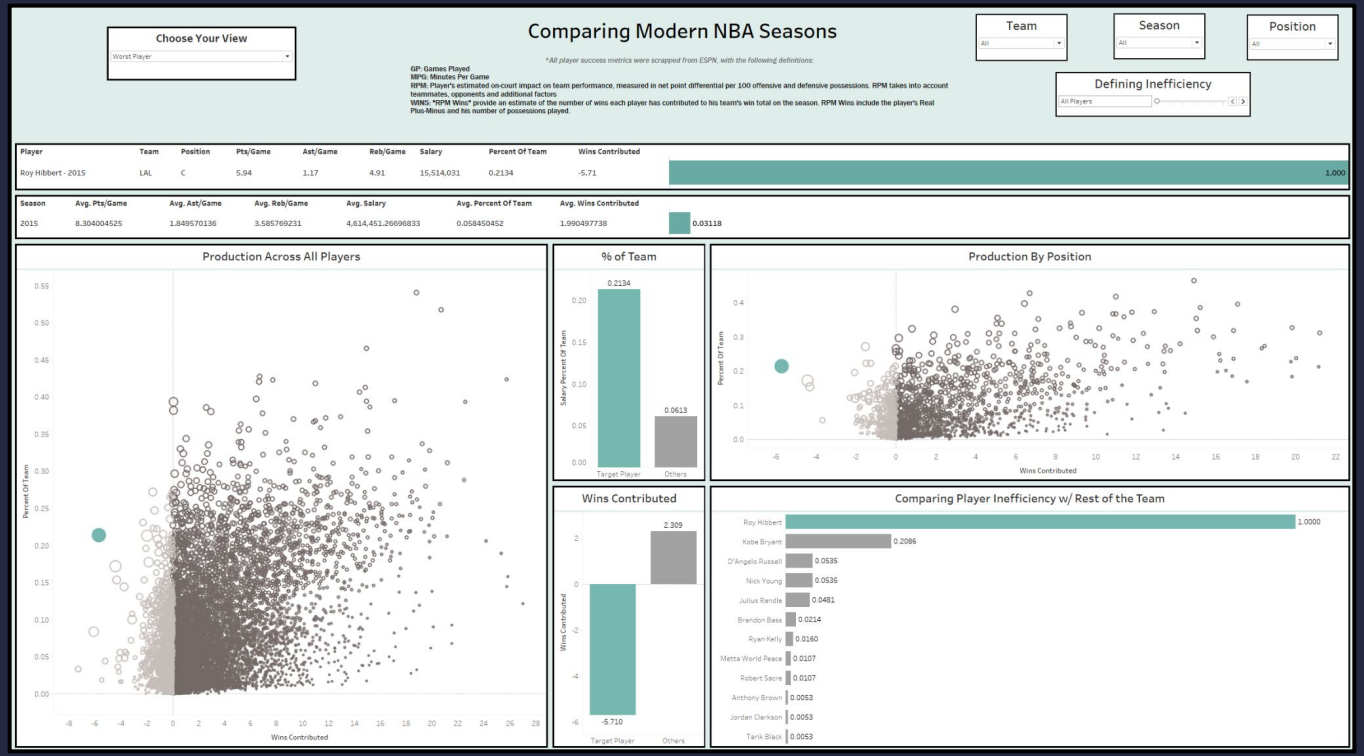
Production Score



Comparing Roy Hibbert with the rest of his team



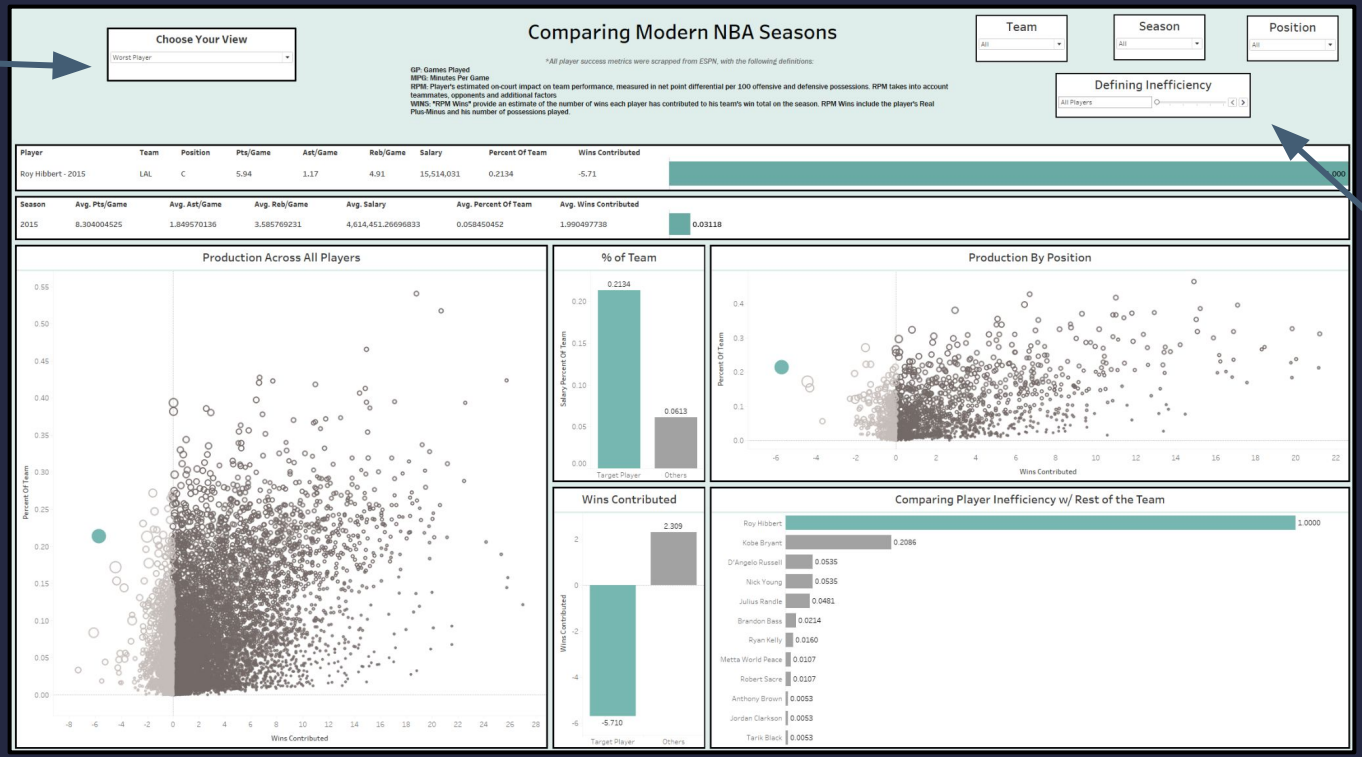
Dashboard



The default view of the dashboard shows us Roy Hibbert and his stats compared with others

Interactivity

Adjust view to return the season's worst or best player



Set Team, Season, and Position through these parameters

Control what the minimum wins contributed is

The interactivity of the dashboard allows you to control the target player you are focusing

Next Steps

- Design new scoring system using draft picks + awards
- Dashboard 2: Summing numbers over full careers
- Recreate ESPN's ML Algorithm to score player success
- Dashboard 3: Interactive 2-player comparison
- MJ Vs. Lebron


Thank You!