­

NCAA ETL Project

Authors: Rusty Gentile, Minal Jajoo, Scott Nubar, Axander Wilson

December 2018

Contents

[1 Introduction 1](#_Toc532650277)

[2 Architecture 1](#_Toc532650278)

[3 Production Database Design 3](#_Toc532650279)

[3.1 Design Process 3](#_Toc532650280)

[3.2 Design Requirements 3](#_Toc532650281)

[3.3 Tables and Columns 3](#_Toc532650282)

[3.4 Data Relations 4](#_Toc532650283)

[4 Extraction Processes 6](#_Toc532650284)

[4.1 Data Sources 6](#_Toc532650285)

[4.2 Team Mascot Extraction 7](#_Toc532650286)

[4.3 Game Data Extraction 7](#_Toc532650287)

[4.4 Game Data Staging 10](#_Toc532650288)

[5 Transformation and Loading 11](#_Toc532650289)

[5.1 Loading the Production Database 11](#_Toc532650290)

[5.2 Issues and Resolutions 11](#_Toc532650291)

[5.3 Database Insights 12](#_Toc532650292)

[5.4 Mining the Database for Insights 12](#_Toc532650293)

[5.5 Database Visualizations 13](#_Toc532650294)

[6 Web Application Functionality 16](#_Toc532650295)

[7 API Documentation 17](#_Toc532650296)

List of Figures

[Figure 2‑1 Web Application 1](#_Toc532649724)

[Figure 2‑2 Team Mascot ETL 2](#_Toc532649725)

[Figure 2‑3 Game Data ETL 2](#_Toc532649726)

[Figure 3‑1 Entity Relation Diagram 5](#_Toc532649727)

[Figure 4‑1 Tournament Bracket Page 6](#_Toc532649728)

[Figure 4‑2 Sample Game Box Score Page 7](#_Toc532649729)

[Figure 4‑3 Mt. St. Mary’s vs. Villanova Box Score 7](#_Toc532649730)

[Figure 4‑4 Villanova Players Box Score 8](#_Toc532649731)

[Figure 4‑5 Box Score Page Layout 8](#_Toc532649732)

[Figure 4‑6 Box Score Page Video Ad 9](#_Toc532649733)

[Figure 4‑7 Sample Game Results DataFrame 9](#_Toc532649734)

[Figure 4‑8 Sample Away Team Stats DataFrame 9](#_Toc532649735)

[Figure 4‑9 Sample Home Team Stats DataFrame 10](#_Toc532649736)

[Figure 4‑10 ETL Staging Space 10](#_Toc532649737)

[Figure 5‑1 Mapping Document 11](#_Toc532649738)

[Figure 5‑2 Top 5 Scoring Players 12](#_Toc532649739)

[Figure 5‑3 Top 10 Free-Throw Percentages 13](#_Toc532649740)

[Figure 5‑4 Final Four Game A Performance 13](#_Toc532649741)

[Figure 5‑5 Final Four Game B Performance 14](#_Toc532649742)

[Figure 5‑6 Championship Performance 14](#_Toc532649743)

[Figure 5‑7 Free Throw Attempts vs. Total Points Scored 15](#_Toc532649744)

[Figure 5‑8 Turnovers vs. Total Points Scored 15](#_Toc532649745)

[Figure 5‑9 Minutes Played vs. Total Points Scored 15](#_Toc532649746)

[Figure 7‑1 API Web Page 17](#_Toc532649747)

# Introduction

The purpose of this project was to model and demonstrate a successful ETL process. To achieve this, data from the 2017 and 2018 men’s basketball tournaments was extracted from the NCAA.com website.

# Architecture

The end product for this project is a bootstrap website which displays the extracted data in visually appealing tables [Figure 2‑1]. Flask is used to connect to the transformed data and serve the web pages. The Flask application also serves up an API which queries both a relation Production Database and uses non-relational data from a set of static .json files.

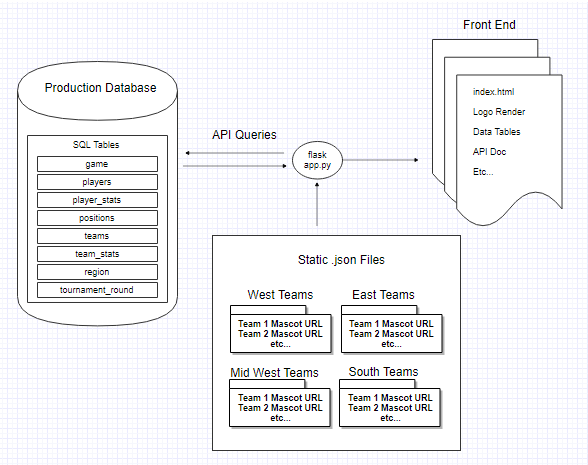


Figure ‑ Web Application

The static .json file data was retrieved using a local workstation which ran a mascot\_scraper script [Figure 2‑2]. This retrieved mainly an image URL for each team mascot. It did not retrieve actual image files. A no-SQL design was selected for this. An optional MongoDB connection was also developed but found to be unnecessary.

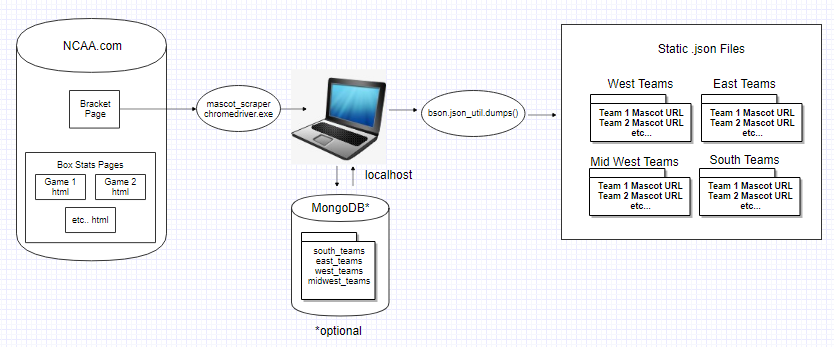


Figure ‑ Team Mascot ETL

A separate elaborate ETL process was developed for the production database [Figure 2‑3]. Here, table data was extracted from several pages and a data.world workspace was used as an ETL staging server. This is further explained in section 4.3.

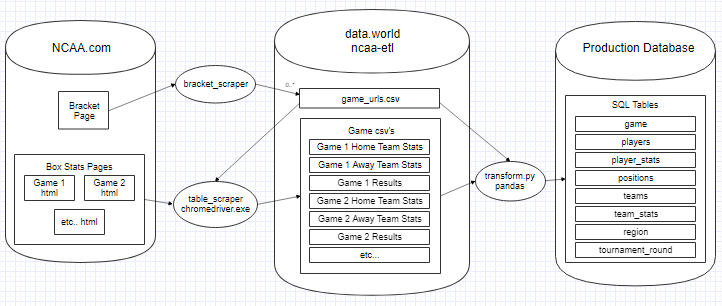


Figure ‑ Game Data ETL

# Production Database Design

## Design Process

The first step in designing the database was to determine its design requirements. Next, the tables and columns needed to fulfill this purpose identified. Lastly, the relationship between each table and column was defined.

## Design Requirements

The purpose of the 2018 NCAA Men’s basketball database is to store player statistics and team statistics.

## Tables and Columns

In order to meet the requirements, the database designer used the following:

Player person data -

* Player name
* Player’s team
* Players position
* Player height
* Player weight
* Date of Birth

Player game data -

* Minutes played per game
* Field goals made per game
* Field goals attempted per game
* Three -point shots attempted per game
* Three-point shots made per game
* Free throws attempted per game
* Free throws made per game
* Offensive rebounds per game
* Defensive rebounds per game
* Assists per game
* Steals per game

Team data -

* Team name
* Team region
* Team ranking

Team game data -

* Minutes played per game
* Field goals made per game
* Field goals attempted per game
* Three -point shots attempted per game
* Three-point shots made per game
* Free throws attempted per game
* Free throws made per game
* Offensive rebounds per game
* Defensive rebounds per game
* Assists per game
* Steals per game

## Data Relations

The database designer referred to the requirements to determine key insights and trends of interest to client. These were identified as follows:

* Shooting percentage per game
* Shooting percentage per player
* Turnover percentage per game
* Turnover percentage per player
* Efficiency percentage: Efficiency = (points/possessions) \* 100

To achieve this, the table relations shown in [Figure 3‑1] were used. The database was implemented in mySQL workbench sql scripts. Full code can be found in ‘database/create\_database/create\_ncaam2018\_db.sql’.

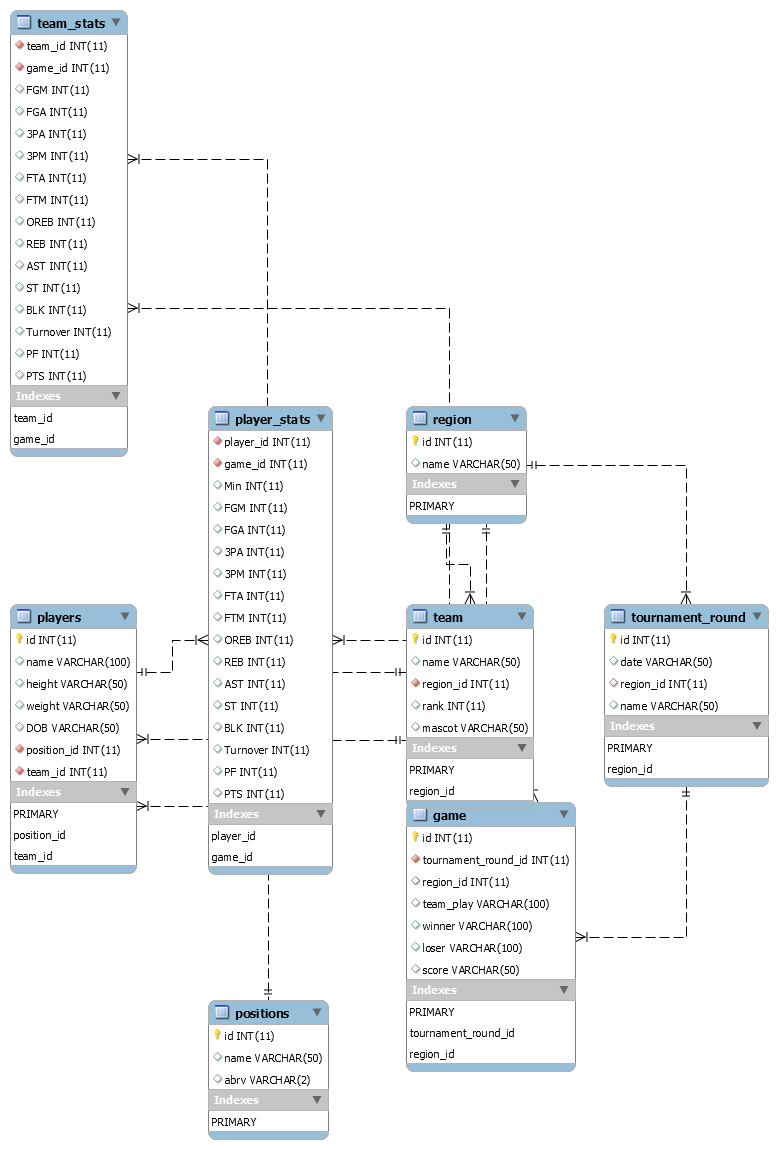


Figure ‑ Entity Relation Diagram

# Extraction Processes

Data extraction was performed using the python splinter library to automate visiting web pages. This required Google chrome and the latest chromedriver.exe (version 2.45 as of this writing).

.

## Data Sources

Two main data sources were used for web scraping. First, a tournament bracket page: <https://www.ncaa.com/brackets/basketball-men/d1> [Figure 4‑1]. From this page, a URL was scraped for each of the 67 games in the tournament.

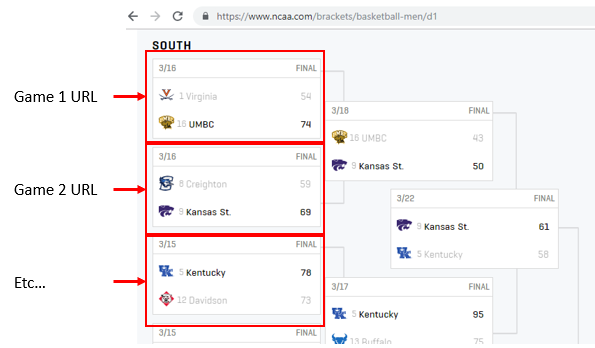


Figure ‑ Tournament Bracket Page

From here, each game URL leads to a page containing table-like box score data [Figure 4‑2].

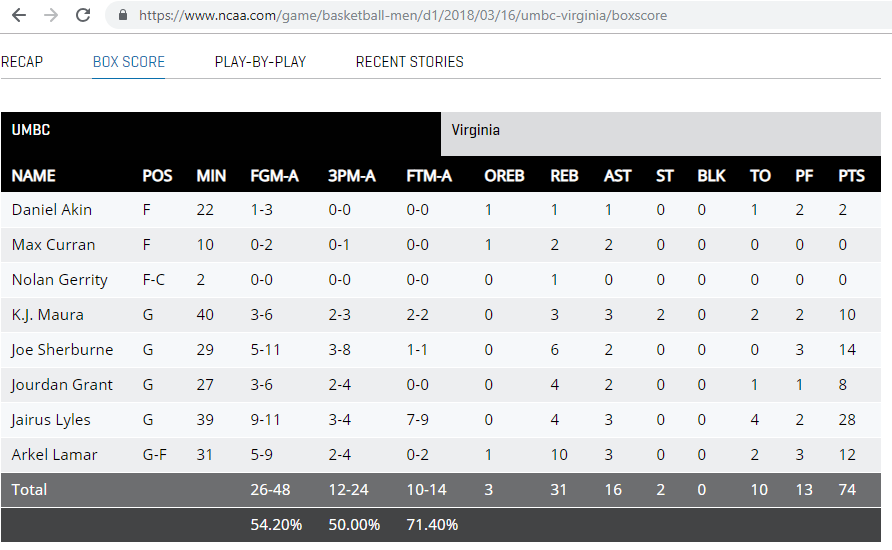


Figure ‑ Sample Game Box Score Page

## Team Mascot Extraction

The first ETL process used only the tournament bracket page. Html data was parsed using the python Beautiful Soup (version 4) library. For each team a name, seed, region, and mascot image URL were retrieved. Region was inferred by the position of each team in the html document.

## Game Data Extraction

The next ETL process was to retrieve game data from the Box Score pages. This presented some challenges. The main difficulty was that the html delivered by each game URL contained player stats for only one of the teams. To reach the other team’s stats, a mouse click is needed on the page. For example, in the Mt. St. Mary’s vs. Villanova game, a user must click on “Villanova” [Figure 4‑3] to reach the Villanova player stats for that game [Figure 4‑4].

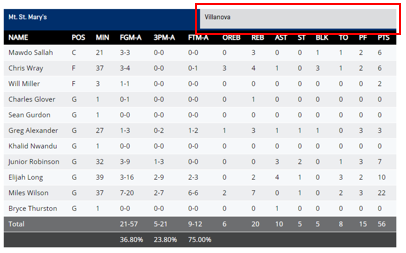


Figure ‑ Mt. St. Mary’s vs. Villanova Box Score

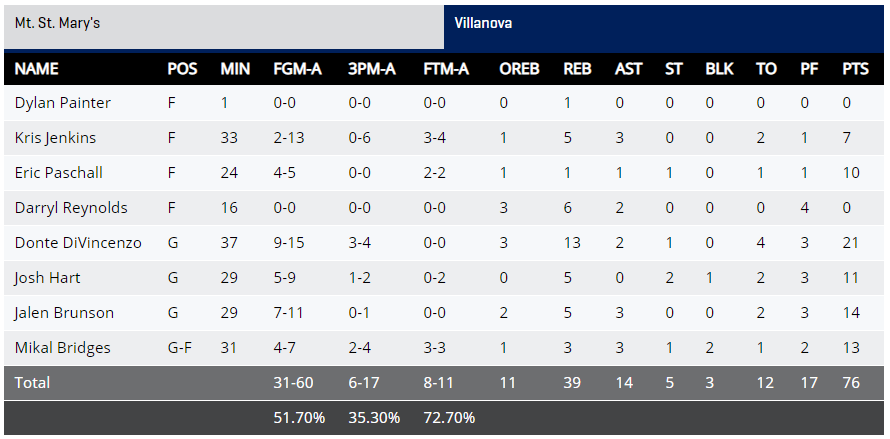


Figure ‑ Villanova Players Box Score

Mouse clicks are indeed possible to automate using Splinter. However, another difficulty was due to the layout of game pages. To make the “Villanova” button visible to click in this example, the user needs to scroll down [Figure 4‑5].

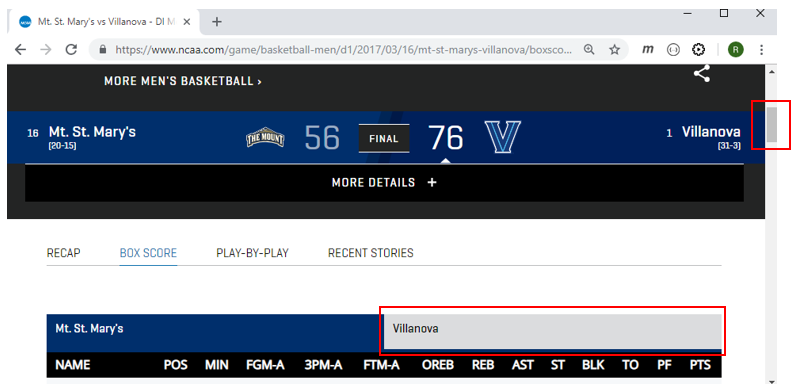


Figure ‑ Box Score Page Layout

Scrolling can also be automated by with a small javascript command:

browser.execute\_script("window.scrollTo(0, 200)")

This, however, did not work as intended since some of the pages contained a video ad in their header [Figure 4‑6]. The extra page space from the ad made it impossible to predict how far the user needs to scroll down. To mitigate this, a sleep command was added to the *table\_scraper* script. The script user simply supervised the execution, then scrolled and clicked in the appropriate place for each page as the script ran.

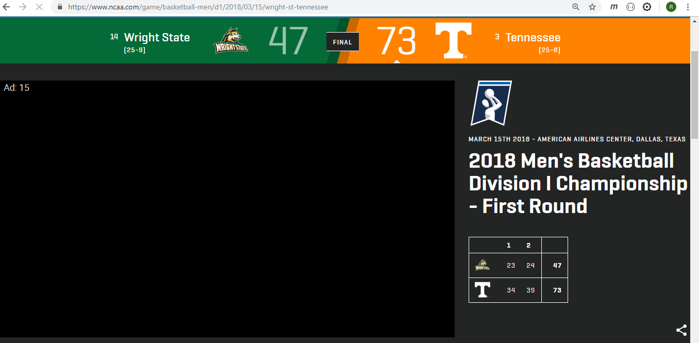


Figure ‑ Box Score Page Video Ad

Finally, to extract the raw data, the pandas read\_html() function was used. This yielded three DataFrames for each page, a game results DataFrame [Figure 4‑7], an away team stats DataFrame [Figure 4‑8], and a home team stats table [Figure 4‑9].



Figure ‑ Sample Game Results DataFrame

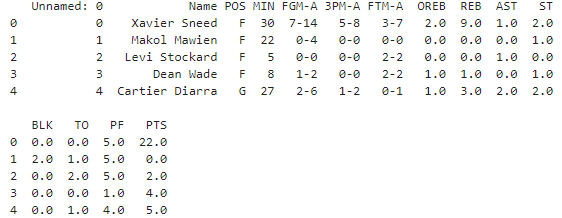


Figure ‑ Sample Away Team Stats DataFrame

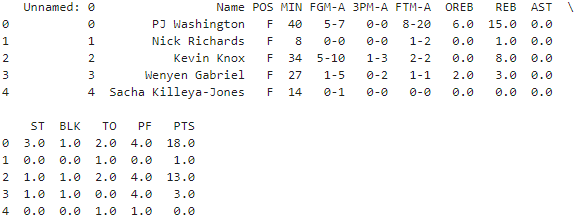


Figure ‑ Sample Home Team Stats DataFrame

## Game Data Staging

Raw data was staged prior to doing any cleansing or transformation processes. This was done to separate the extraction step. For each of the 67 tournament games, the three DataFrames described above were extracted for a total of 201. The pandas DataFrames were stored as .csv files using a dataset hosted at data.world [Figure 4‑10]. File names for the .csv’s were derived from the game URL, yielding three .csv’s per game as an example:

* 2018\_03\_22\_kansas\_st\_kentucky\_boxscore\_game\_results.csv
* 2018\_03\_22\_kansas\_st\_kentucky\_boxscore\_Kansas St..csv
* 2018\_03\_22\_kansas\_st\_kentucky\_boxscore\_Kentucky.csv

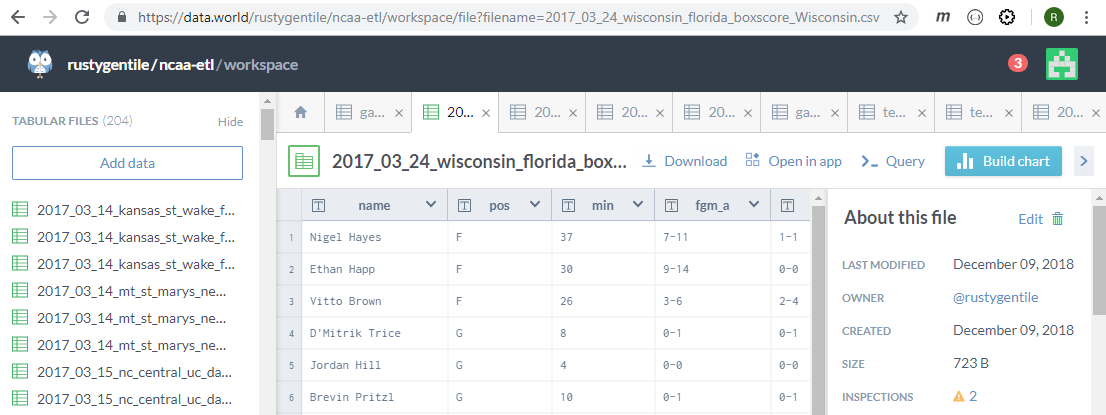


Figure ‑ ETL Staging Space

# Transformation and Loading

## Loading the Production Database

The greatest challenge to loading the database was transforming external data sources to meet the structure requirements of the database. The external data sources were scraped from the NCAA website and the raw csv files did not have the full columns or information needed to load the csv files to the database. The raw csv files did not contain the team name, winner or loser information available. Therefore, a python script was created to transform the raw data using jupyter notebook. See data\_transform\_load.iynb for full code.

## Issues and Resolutions

A common challenge of loading data from external sources is a different naming convention used to refer to the same data point. In this case, this was needed due to the limitations of special characters. For example, Duke University player Alex O'Connell. mySQL had difficulty loading the players name due to the apostrophe. As another example, the mySQL database used the name ‘Mount St. Mary’ while external data sources used names like ‘Mt. St. Mary’s’. To overcome these challenges, a mapping document was created [Figure 5‑1]. The mapping document outlines and tracks the names used by each data source and how item is mapped to the internal database.

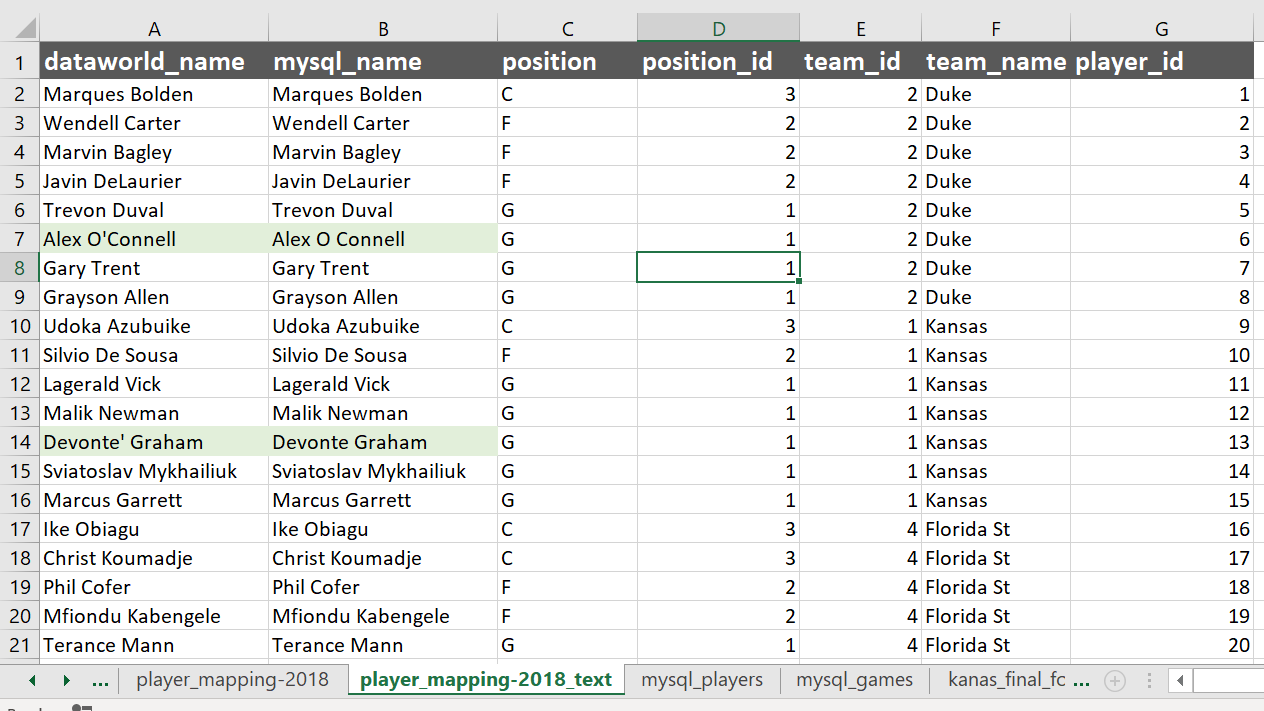


Figure ‑ Mapping Document

## Database Insights

Once the hard work of designing and loading the database is complete, sql queries can be used to probe the database for helpful insight. For example, determine each team’s most, least efficient game and the opponent; determine the number of times a lower ranking team won and which categories did the lower ranking team outperform a higher ranking team; who was the conference leading three-point shooter, shot blocker, and free throw shooter. The possibilities are endless. Sample queries were provided to the client in the ‘sql\_queries’ folder as a good starting point.

## Mining the Database for Insights

With such valuable data now stored in the client’s database, the next step is mining the database for hidden gems. The file ‘performance\_queries.sql’ is a full list of queries provided to breakdown team performance and individual player performance. For example [Figure 5‑2] and [Figure 5‑3] show two of the top five scoring players in the final four and championship games and the top ten free-throw percentage. For a full list of queries available see ‘performance\_queries\_. xlsx

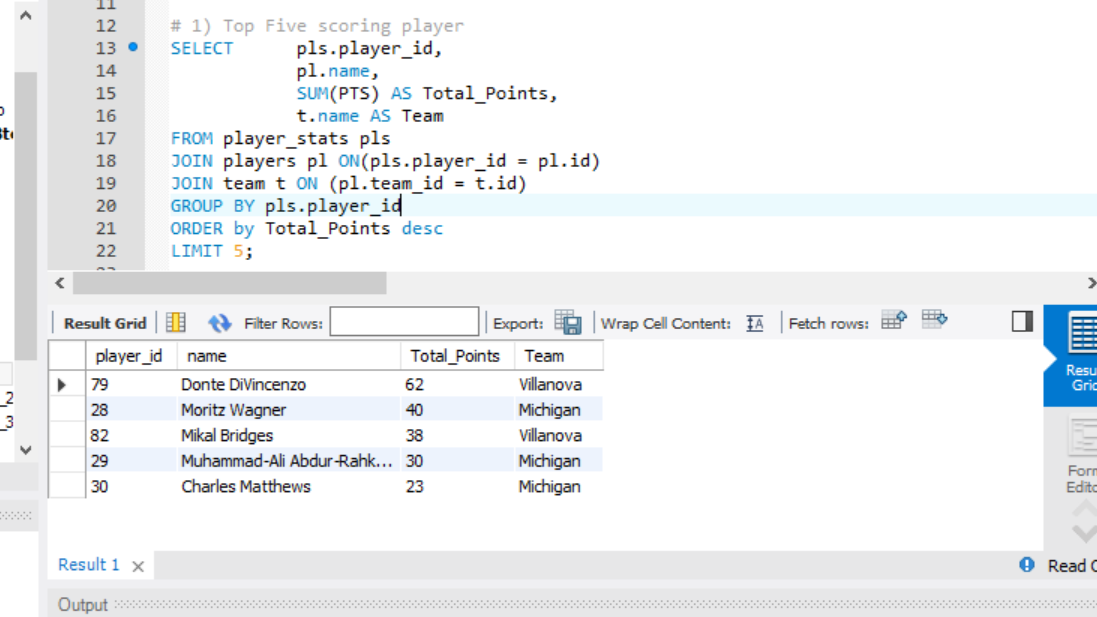


Figure ‑ Top 5 Scoring Players

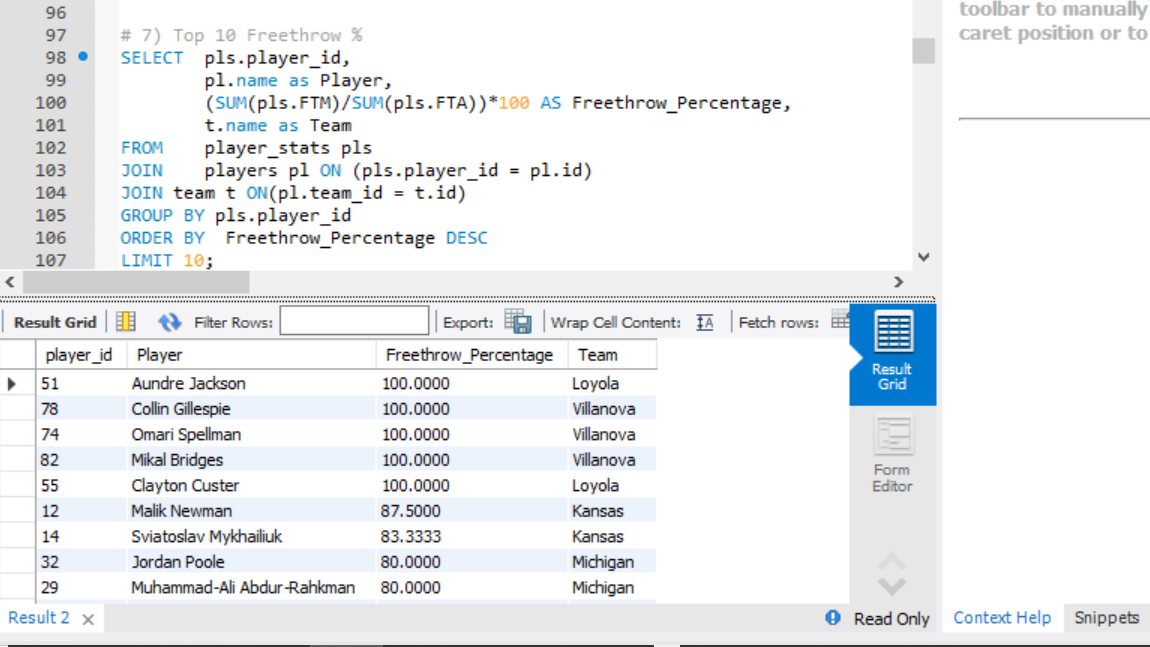


Figure ‑ Top 10 Free-Throw Percentages

## Database Visualizations

The final step is a visual analysis of a graphical visualizations of the data. Final Four game performance, Championship game performance, the relationship between of free-throw attempts and overall scoring, the relationship between turnovers and overall scoring, minutes played and scoring productivity is provided in the jupyter notebook ‘stats\_visualzation’. Sample visualizations are provided in [Figure 5‑4] though [Figure 5‑9].

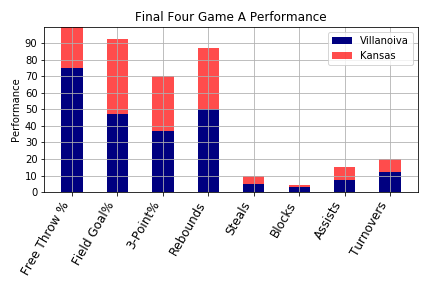


Figure ‑ Final Four Game A Performance

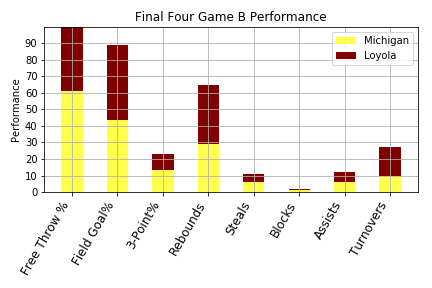


Figure ‑ Final Four Game B Performance

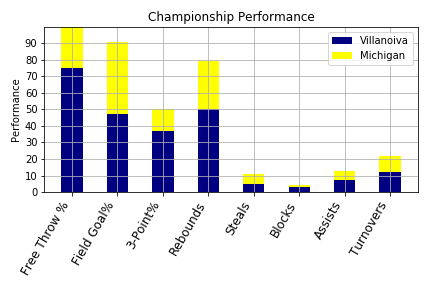


Figure ‑ Championship Performance

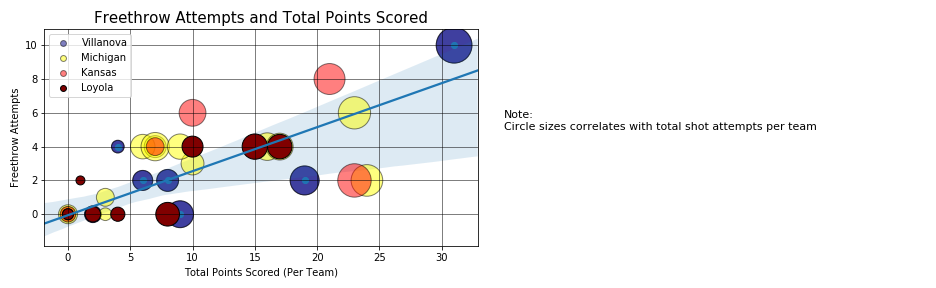


Figure ‑ Free Throw Attempts vs. Total Points Scored

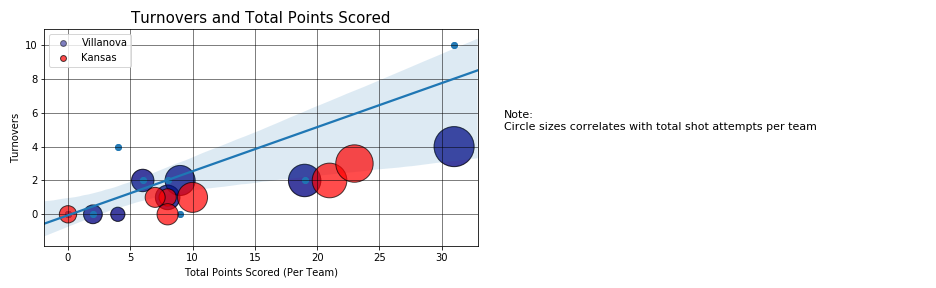


Figure ‑ Turnovers vs. Total Points Scored

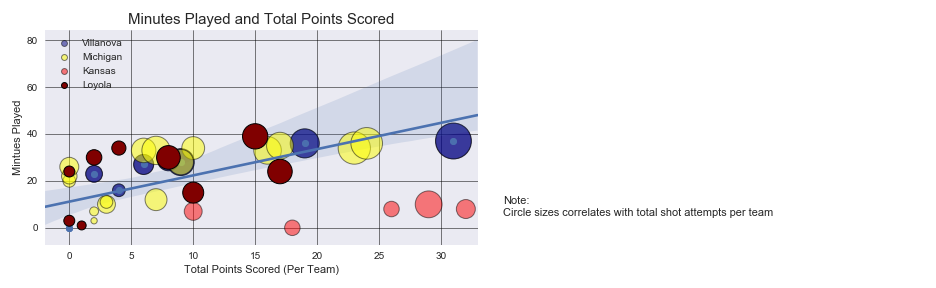


Figure ‑ Minutes Played vs. Total Points Scored

With more time and resources available, one possible area of improvement would be conducting tests for data validation and reasonability. For example, pulling the statistics and validating that each player’s aggregated statistics matches the team’s statistics and the overall score.

# Web Application Functionality

The website is initialized through a flask application which delivers the static pages as well as the API routes. Bootstrap 4 and CSS were utilized for styling, visualization, customization, and organization.

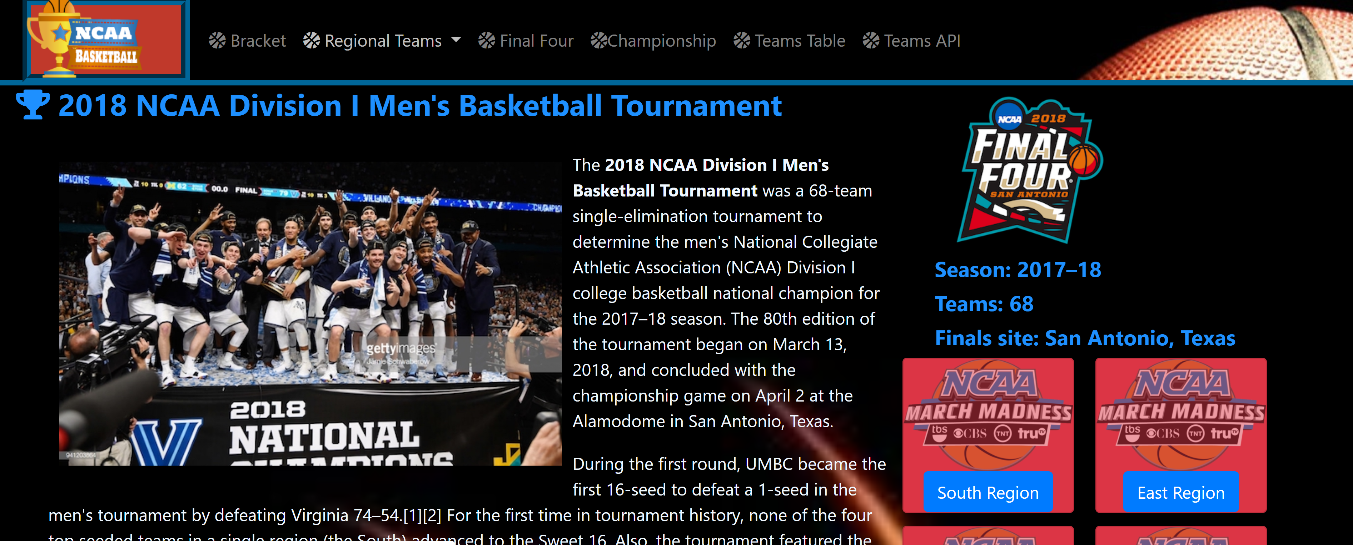


Figure ‑ Website Landing Page

The Website consists of 10 total pages including the API:

* Home
* Bracket
* Regional Teams: South, West, East, and Midwest
* Final Four
* Championship
* Teams Table
* API

All pages are called via a render template statement in the flask application for each route defined per page.

All images on the website are rendered links which were obtained via a python scraping script. The script uses splinter via the google driver to scrape the images URLs from ncaa.com, for the championship teams, for 2018 specifically. 2017 data was scraped and featured in the database, however the look and feel of the website was intended to be modern and up to date with the most recent stats and games.

The tables featured on the site were created via a csv to html conversion tool (<http://convertcsv.com/csv-to-html.htm>) in order to display on each page where a table is shown. Some javascript was utilized on the API for user input to accommodate easy browsing, which is further explained in the API documentation.

An alteration was made to the Final Four tab at the last moment as we ran into combability issues between Macs and PC’s. We were using PyMongo with a python scraper (scrape\_four.py) to live scrape Final Four statistics to display on the website. This was disabled on the site, but the files are still in the project for further development later on.

# API Documentation

This API page provides the ability to retrieve the transformed 2017 and 2018 data that was scraped from ncaa.com for both championship brackets.

Available paths:

* /api/2017/team\_names: Displays only the 2017 team names.
* /api/2017/team: Displays all available team info for 2017.
* /api/2017/team/<name>: Allow for the user to input a specific team name to search.
* /api/2017/player\_names: Displays only the 2017 player names.
* /api/2017/player: Displays all available info for the 2017 players.
* /api/2017/player/<name>: Allow for the user to input a specific player name to search.
* /api/2018/elite\_player\_names: Display only the elite eight player names.
* /api/2018/elite\_eight: Display all available team info for 2017.
* /api/2018/elite\_eight/<name>: Allow for the user to input a specific elite eight player to search.
* /api/2018/Seeds\_region/<region>: Returns a list of all teams in the region.
* /api/2018/player\_rank/<top\_number>: Display a list of ranked players up to the number specified.

This API interface was built using a flask application, but the GUI was implemented via a html file that is rendered to 127.0.0.1:5000/API.

The API Home page was built using a combination of HTML and JavaScript. Bootstrap was used to bring the visual theme of the page in line with the rest of the website for a seamless experience to the user [Figure 7‑1].

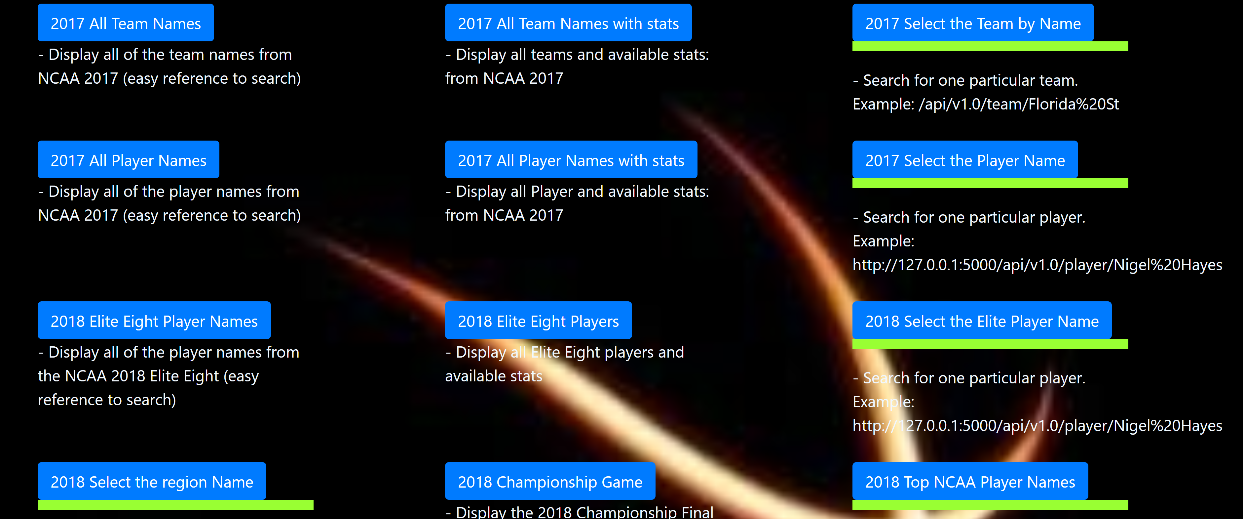


Figure ‑ API Web Page

Plain blue buttons have the paths programmed so that the user can simply click a button and retrieve the information. Buttons highlighted in green allow for user input and to run a specific search. A box will generate on screen via javascript which enables the user input. The development team programmed the API routes and the javascript to convert any spaces between names to a %20 for a seamless URL to appear in the browser.

To keep the web application self-contained without the need for a SQL server, the API accesses the transformed data in two separate manners. The first is via a set of JSON files stored in /website/static/Resources. JSON was selected to simplify the website logic, since the API delivers data in JSON format.

The second type of API access is via a sqlite implementation of a few select tables. The final two API routes utilize this:

* /api/2018/Seeds\_region/<region>: Returns a list of all teams in the region.
* /api/2018/player\_rank/<top\_number>: Display a list of ranked players up to the number specified.