**SAN JOSE STATE UNIVERSITY**

**CMPE 180B DBMS Project Report**

**Group 17**

**Kunal Keshav Singh Sahni**

**Zach Kuo**

**Sonali Lonkar**

**Phuong Duy Lam Nguyen**

**< April 30, 2025 >**

**Introduction:**

Our group chose to work with the NCAA Men’s Basketball dataset from Google’s BigQuery Repository. From the dataset, our group created an API to interact with the data that would be used for analyzing the performance of individual players or teams across different seasons, player comparisons and season winner predictions. This dataset has comprehensive historical data for the basketball teams and players making it an ideal choice for analytical and predictive applications. We have used the FastAPI python framework to build and test the APIs.

**Project Github Repository** : [Link](https://github.com/kunal768/NCAA_Basketball_Stats)

**Tech Stack** : Python FastAPI Server, Google BigQuery (Python Client)

**Applications:**

Our API utilizes FastAPI python framework to build a powerful service that can empower a variety of applications, where the in-depth analysis of various players could be used to set odds for various different scenarios:

* Sports Betting Platforms can integrate our APIs to inform odds calculation with up-to-date player and team performance metrics.
* Coaches & Scouts gain rapid, granular insights into player trends and head-to-head team histories.
* Fans & Managers can drill into favorite athletes' recent box scores or compare stars across seasons.

The degree of information kept about each player, game, and team, compounded across multiple seasons, provides a great opportunity to explore different trends within the league. And given the recent rise in popularity of sports betting, providing insights from a large wealth of data in an easily digestible manner could prove useful to those looking to enter into the field. This API could be used in consumer-facing products that are used by sports bettors to inform their decisions behind a bet, or even being used as a basis for the odds that are set for a given bet.

**Challenges:**

One of the difficulties of this project comes from its strength: the vast amount of different angles to explore within the data. Using this large data it was challenging to optimize queries and implement predictive models for season winners accurately. We chose SQL for its capabilities and RESTful APIs for scalability. It is important to balance how interesting a specific insight is with its digestibility and ease of access.

**Related Work:**

1. **Source:** KenPom.com

**Link:** <https://kenpom.com>

**Description:** KenPom.com, developed by Ken Pomeroy, is a prominent platform offering advanced analytics for NCAA basketball teams. It provides metrics such as adjusted offensive and defensive efficiency, tempo, and strength of schedule.

**Pros:**

* Widely recognized and utilized within the basketball analytics community.
* Offers possession-based metrics that provide deeper insights than traditional statistics.

**Cons:**

* Proprietary platform with limited access to raw data.
* Lacks a public API for developers to integrate data into custom applications.

**Comparison:**

While KenPom.com provides comprehensive analytics, our project emphasizes openness and scalability by offering a RESTful API, enabling real-time data access and integration into various applications.

1. **Source:** TeamRankings & PoolGenius

**Link:** <https://poolgenius.teamrankings.com/ncaa-bracket-picks/>

**Description:** TeamRankings and its tool PoolGenius provide predictive analytics and strategy advice for NCAA tournament pools, utilizing statistical models to forecast game outcomes.

**Pros:**

* Offers user-friendly tools for bracket predictions and pool strategies.
* Utilizes statistical models to inform user decisions.

**Cons:**

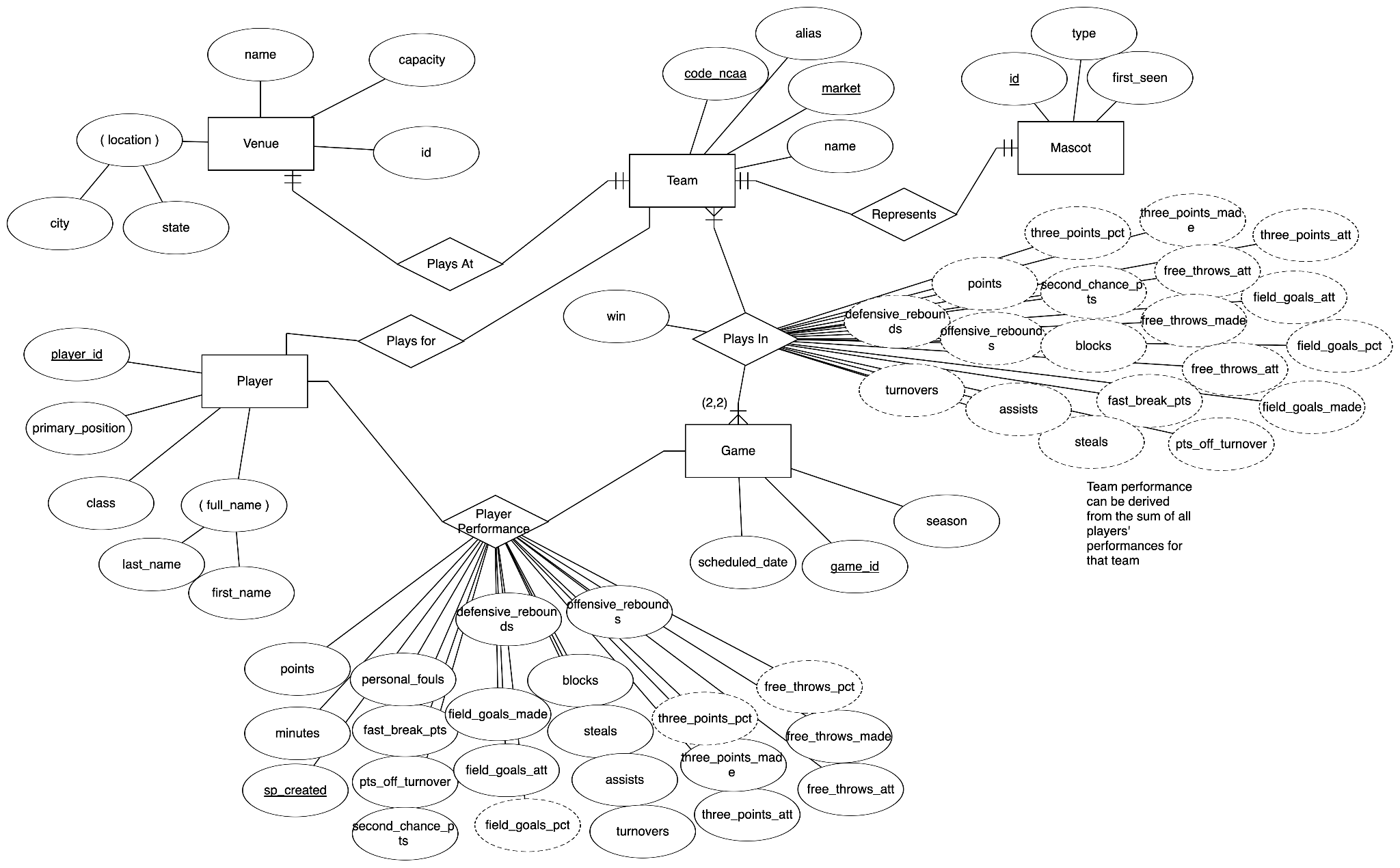
* Primarily consumer-focused with limited access to underlying data and models.
* Lacks customization options for advanced users or developers.

**Comparison:**

Our project caters to both developers and analysts by providing customizable access to data and models through an API, enabling deeper analysis and integration into diverse applications.

Our project improves existing approaches by combining easily scalable APIs which can be integrated with any relevant application and predictive modelling, for the need of both real-time and predictive analysis.

**ER Diagram**

****

\*table and/or column names abbreviated for legibility

**Application Features:**

Our application has several features listed below with all its details including goal, required tables from the dataset, columns used from the table, API endpoints and expected & tested results.

1. **Feature: Player Analytics**

* **Abstract**: Individual player statistics (points, rebounds, assists, etc.) will be retrievable through API calls, enabling granular analysis of player performance.
* **Goal:** College basketball coaches, scouts and fans need rapid, granular insight into individual player performance—both season-level trends and game-by-game details—to make recruitment decisions, game-planning adjustments, or simply follow their favorite athletes.
* **Solution**: Fetch the most recent *N* box-score lines for a given player, so that any client (web, mobile, analytics) can display or analyze each game’s stats (minutes, points, rebounds, assists, shooting splits, etc.) in chronological order.
* **Database Detail**

1. **Player By-Seasons Endpoint**
   * 1. **Table:** *bigquery-public-data.ncaa\_basketball.mbb\_players\_games\_sr*
     2. **Relevant columns:**
        1. full\_name (STRING): player’s exact name
        2. sp\_created (TIMESTAMP): when the box‐score was recorded (serves as game timestamp) → extract YEAR
        3. minutes\_int64 (INT64): minutes played
        4. points, rebounds, assists, steals, blocks (INT64): core counting stats
   1. **Player Game-by-Game Endpoint**
      1. **Table:** *bigquery-public-data.ncaa\_basketball.mbb\_players\_games\_sr*
      2. **Relevant columns:**
         1. full\_name (STRING)
         2. sp\_created (TIMESTAMP) → game timestamp
         3. minutes\_int64, points, rebounds, assists, steals, blocks
         4. shooting splits: field\_goals\_made/att/pct, three\_points\_made/att/pct, free\_throws\_made/att/pct

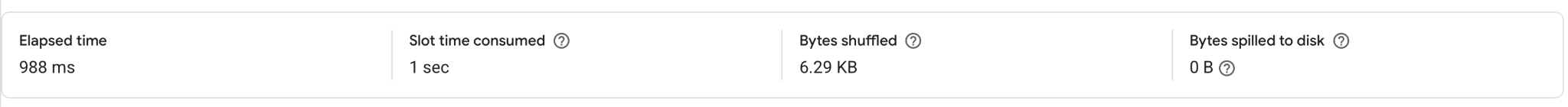
* **Query**
  1. **Player By-Seasons Endpoint**

**IO Cost: 43.65 MB**

| -- BQ Standard SQL WITH games AS (  **SELECT**  full\_name,  **EXTRACT**(**YEAR** **FROM** sp\_created) **AS** season\_year,  points, rebounds, assists, steals, blocks  **FROM**  `bigquery-public-data.ncaa\_basketball.mbb\_players\_games\_sr`  **WHERE**  full\_name = @player\_name  **AND** (@start\_year **IS** **NULL** **OR** **EXTRACT**(**YEAR** **FROM** sp\_created) >= @start\_year)  **AND** (@end\_year **IS** **NULL** **OR** **EXTRACT**(**YEAR** **FROM** sp\_created) <= @end\_year) ) **SELECT**  full\_name **AS** player\_name,  season\_year,  **COUNT**(1) **AS** games\_played,  **SUM**(points) **AS** total\_points,  **ROUND**(**AVG**(points), 2) **AS** avg\_points,  **SUM**(rebounds) **AS** total\_rebounds,  **ROUND**(**AVG**(rebounds), 2) **AS** avg\_rebounds,  **SUM**(assists) **AS** total\_assists,  **ROUND**(**AVG**(assists), 2) **AS** avg\_assists,  **SUM**(steals) **AS** total\_steals,  **ROUND**(**AVG**(steals), 2) **AS** avg\_steals,  **SUM**(blocks) **AS** total\_blocks,  **ROUND**(**AVG**(blocks), 2) **AS** avg\_blocks **FROM**  games **GROUP** **BY**  full\_name, season\_year **ORDER** **BY**  season\_year; |
| --- |

* 1. **Player Game-by-Game Endpoint**

**IO Cost: 91.6 MB**



| -- BQ Standard SQL **SELECT**  full\_name **AS** player\_name,  sp\_created **AS** game\_timestamp,  **CAST**(minutes\_int64 **AS** INT64) **AS** minutes\_played,  points, rebounds, assists, steals, blocks,  field\_goals\_made **AS** fgm,  field\_goals\_att **AS** fga,  **ROUND**(field\_goals\_pct, 3) **AS** fg\_pct,  three\_points\_made **AS** three\_pt\_made,  three\_points\_att **AS** three\_pt\_att,  **ROUND**(three\_points\_pct, 3) **AS** three\_pt\_pct,  free\_throws\_made **AS** ftm,  free\_throws\_att **AS** fta,  **ROUND**(free\_throws\_pct, 3) **AS** ft\_pct **FROM**  `bigquery-public-data.ncaa\_basketball.mbb\_players\_games\_sr` **WHERE**  full\_name = @player\_name  **AND** (@start\_year **IS** **NULL** **OR** **EXTRACT**(**YEAR** **FROM** sp\_created) >= @start\_year)  **AND** (@end\_year **IS** **NULL** **OR** **EXTRACT**(**YEAR** **FROM** sp\_created) <= @end\_year) **ORDER** **BY**  sp\_created **DESC** **LIMIT**  @**limit**; |
| --- |

* **API Endpoint**
  1. **Player By-Seasons Endpoint**

| GET /v1/**data**/player/{player\_name}/seasons Query parameters:  start\_year (INT64, **optional**)  end\_year (INT64, **optional**) |
| --- |

* 1. **Player Game-by-Game Endpoint**

| GET /v1/**data**/player/{player\_name}/games Query parameters:  start\_year (INT64, **optional**)  end\_year (INT64, **optional**)  limit (INT64, **optional**, **default**=20) |
| --- |

* **Result JSON**
  1. **Player By-Seasons Endpoint**

| {  "seasons": [  {  "player\_name": "A'Torri Shine",  "season\_year": 2017,  "games\_played": 28,  "total\_points": 412,  "avg\_points": 14.71,  "total\_rebounds": 198,  "avg\_rebounds": 7.07,  "total\_assists": 85,  "avg\_assists": 3.04,  "total\_steals": 32,  "avg\_steals": 1.14,  "total\_blocks": 18,  "avg\_blocks": 0.64  },  {  "player\_name": "A'Torri Shine",  "season\_year": 2018,  "games\_played": 30,  "total\_points": 397,  "avg\_points": 13.23,  "total\_rebounds": 212,  "avg\_rebounds": 7.07,  "total\_assists": 92,  "avg\_assists": 3.07,  "total\_steals": 29,  "avg\_steals": 0.97,  "total\_blocks": 22,  "avg\_blocks": 0.73  }  ] } |
| --- |

* 1. **Player Game-by-Game Endpoint**

| {  "games": [  {  "player\_name": "A'Torri Shine",  "game\_timestamp": "2018-03-01T02:00:00Z",  "minutes\_played": 35,  "points": 21,  "rebounds": 8,  "assists": 3,  "steals": 2,  "blocks": 1,  "fgm": 8,  "fga": 16,  "fg\_pct": 0.500,  "three\_pt\_made": 2,  "three\_pt\_att": 5,  "three\_pt\_pct": 0.400,  "ftm": 3,  "fta": 4,  "ft\_pct": 0.750  },  {  "player\_name": "A'Torri Shine",  "game\_timestamp": "2018-02-25T01:00:00Z",  "minutes\_played": 32,  "points": 18,  "rebounds": 6,  "assists": 4,  "steals": 1,  "blocks": 0,  "fgm": 7,  "fga": 14,  "fg\_pct": 0.500,  "three\_pt\_made": 1,  "three\_pt\_att": 4,  "three\_pt\_pct": 0.250,  "ftm": 3,  "fta": 3,  "ft\_pct": 1.000  }  /\* ... up to @limit games ... \*/  ] } |
| --- |

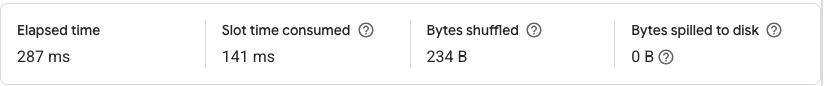
1. **Feature: Team Versus Team Win-Loss**

* **Abstract**: Historical performance of Team 1 vs. Team 2.
* **Goal:** Predicting the outcome of a given game requires analyzing various aspects of how teams match up against one another. The relative strength of two given programs also changes over time, which can complicate simply finding out which team has an advantage in overall record.
* **Solution**: Returns a record of the wins and losses of Team 1 vs. Team 2 that can be used to analyze the relative strength of the two programs. By selecting a starting and ending season (both inclusive), the end user can more accurately gauge trends in the competition between two teams over time.
* **Database Detail**
  1. **Table:** *bigquery-public-data.ncaa\_basketball.mbb\_historical\_teams\_games*
  2. **Relevant columns:**
     1. win (BOOLEAN): indicates if Team 1 won against Team 2
     2. team\_code (STRING): NCAA identifier code of Team 1
     3. opp\_code (INT): NCAA identifier code of Team 2
     4. season (INT): The seasons over which the games were played

\* team codes can be found for a given school [here](http://stats.ncaa.org/game_upload/team_codes)

* **Query**

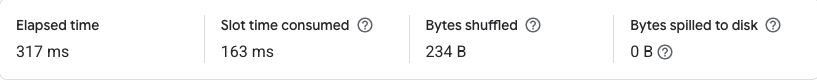
**Initial, Unoptimized IO Cost:** 35.96 MB



| -- initial query  **SELECT   market as team,  COALESCE(SUM(CASE WHEN win THEN 1 ELSE 0 END)) as wins,   COALESCE(SUM(CASE WHEN win THEN 0 ELSE 1 END)) as losses,  opp\_market as opposing\_team FROM   `bigquery-public-data.ncaa\_basketball.mbb\_historical\_teams\_games`  WHERE   team\_code = CAST(@team1\_code AS string) AND   opp\_code = @team2\_code AND   season >= @starting\_season AND   season <= @ending\_season GROUP BY team, opposing\_team LIMIT 1000** |
| --- |

* **Query After Optimization: Attempt 1**

**IO Cost:** 28.76 MB

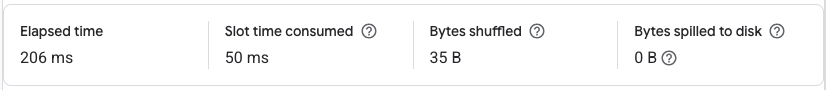


The original query uses the team\_code and opp\_code as a unique identifier for each team. Surprisingly, whether caused by bigquery not indexing the table of relatively small size or needing to fetch the team/school name as part of the result output, using the code actually increases the IO cost compared to using the school name for filtering.

| **SELECT**   market **as** team,  **COALESCE**(**SUM**(**CASE** **WHEN** win **THEN** 1 **ELSE** 0 **END**)) **as** wins,   **COALESCE**(**SUM**(**CASE** **WHEN** win **THEN** 0 **ELSE** 1 **END**)) **as** losses,  opp\_market **as** opposing\_team **FROM**   `bigquery-public-data.ncaa\_basketball.mbb\_historical\_teams\_games` **WHERE**   market="San Jose State University" **AND**   opp\_market="San Diego State University" **AND**   season **BETWEEN** 2000 **AND** 2017 **GROUP** **BY** team, opposing\_team **LIMIT** 1000 |
| --- |

* **Query After Optimization: Attempt 2**

**IO Cost: 12.1 MB**

****

| **-- optimization attempt 2 SELECT   COALESCE(SUM(CASE WHEN win THEN 1 ELSE 0 END)) as wins,   COALESCE(SUM(CASE WHEN win THEN 0 ELSE 1 END)) as losses, FROM   `bigquery-public-data.ncaa\_basketball.mbb\_historical\_teams\_games`  WHERE   team\_code = CAST(@team1\_code AS string) AND   opp\_code = @team2\_code AND   season >= @starting\_season AND   season <= @ending\_season LIMIT 1000** |
| --- |

If the team names were dropped entirely in favor of using codes, the query would see a substantial decrease in IO cost down to just 12.1 MB, a third of the original IO cost. However, this version does significantly hinder readability if the end user is directly inputting the data, as they would need to manually look up the codes for each team or utilize another query to find the corresponding code for the teams they wish to compare. Depending on the use case, either optimization attempt could be useful. For the case of being integrated within the backend of some sport betting or sports analysis app, we found the abstraction to use NCAA codes over a university name is acceptable. Since the team code is used across multiple tables in the dataset, it is reasonable to assume that the application would be able to use a hashmap or another similar data structure to map the university name to its respective code.

Further optimization could be done by converting the team\_code to be an integer instead of the string that the NCAA dataset uses, leading to improved comparison performance. Exploring the impact of partitioning the table to logically group by team could also yield a performance benefit.

I explored partitioning the table by team\_code to see how that affected performance.

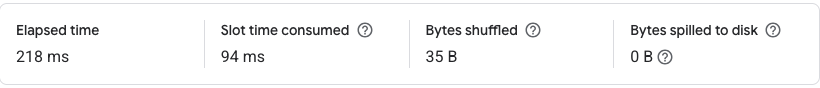
| **CREATE** **TABLE**  `shining-sign-456416-g2.ncaa\_basketball.optimized\_mbb\_games\_sr` **PARTITION** **BY** RANGE\_BUCKET(**CAST**(team\_code **AS** INT64), GENERATE\_ARRAY(0, 600000, 10000)) -- Partition by team\_code as INT64 CLUSTER **BY** opp\_code OPTIONS (  description = "Clustered table on team\_pair for efficiency" ) **AS** **SELECT**  **CAST**(team\_code **AS** INT64) **AS** team\_code,  **CONCAT**(**LPAD**(**CAST**(team\_code **AS** **STRING**), 6, '0'), '-', **LPAD**(**CAST**(opp\_code **AS** **STRING**), 6, '0')) **AS** team\_pair, -- custom column for searching  opp\_code,  market,  opp\_market,  win **FROM**   `bigquery-public-data.ncaa\_basketball.mbb\_historical\_teams\_games` **WHERE**   team\_code **IS** **NOT** **NULL**  **AND** opp\_code **IS** **NOT** **NULL** |
| --- |

Now that the table is partitioned, we can test out some queries to see if that improves our IO cost:

Partitioned Query 1:

| **-- query on partitioned table SELECT  COALESCE(SUM(CASE WHEN win THEN 1 ELSE 0 END)) as wins,  COALESCE(SUM(CASE WHEN win THEN 0 ELSE 1 END)) as losses, FROM  `shining-sign-456416-g2.ncaa\_basketball.optimized\_historical\_teams\_games` WHERE  team\_code = 630 AND  opp\_code = 626 AND  season BETWEEN 2000 AND 2017 LIMIT 1000** |
| --- |

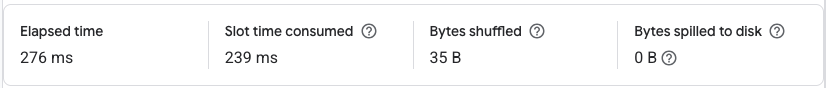
**IO Cost: 12.57MB**



Partitioned Query 2 - Custom Column Search

| -- query w/ custom column search **SELECT**  **COALESCE**(**SUM**(**CASE** **WHEN** win **THEN** 1 **ELSE** 0 **END**)) **as** wins,  **COALESCE**(**SUM**(**CASE** **WHEN** win **THEN** 0 **ELSE** 1 **END**)) **as** losses, **FROM**  `shining-sign-456416-g2.ncaa\_basketball.optimized\_historical\_teams\_games` **WHERE**  team\_pair = "000630-000626"  **AND** season **BETWEEN** 2000 **AND** 2017 **LIMIT** 1000 |
| --- |

**IO Cost: 13.08 MB**

****

Unfortunately, it would appear that the attempt to partition the table did not improve the performance of the query. In fact, the custom column search was substantially more expensive than the others, at least in terms of the Slot Time reported by BigQuery, at 239 ms compared to 94ms and 50ms for Partition Query 1 and the Query Optimization Attempt 2, respectively. The runtimes of each did not differ significantly, though admittedly I did not extensively test the runtimes of each out of fear that I would run up to the end of my credits.

As for a reason for this behavior, I am rather puzzled. Logically, grouping the rows of the same team together should lead to a more localized search compared to a purely random search. Since I didn’t look at the data extensively outside of previewing a few hundred entries, perhaps the table is already organized in a way that localizes these groups and I accidentally undid some of that optimization? That is my best guess for now.

* **API Endpoint - Optimization 2**

| GET /v1/**data**/teams/win-loss Query parameters:  team1\_code (INT64)  team2\_code (INT64)  starting\_season (INT64)  ending\_season (INT64) |
| --- |

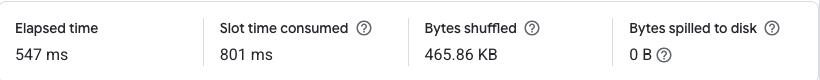
* **Result JSON - Optimization 2**

| {  "message": "win-loss request successful",  "error": "",  "data": {  "data": [  {  "wins": 23,  "losses": 20  }  ]  },  "status\_code": 200,  "success": true } |
| --- |

1. **Feature: Three-Point Percent**

* **Abstract**: Three-Point Percentage Leaders by Season
* **Goal:** As the three-point shot becomes ever more present in modern basketball strategy, a simple metric for comparing three-point shooters across the league can be helpful for gauging performance of players. Using a simple percentage for this metric can cause bias towards those who made few three-pointers but had a high percentage.
* **Solution**: Returns the top three-point shooters by percentage in a season, as well as accounting for a minimum number of shots taken so as to not be skewed by players who have an abnormally high percentage based on a few shots.
* **Database Detail**
  1. **Table:** *bigquery-public-data.ncaa\_basketball.mbb\_players\_games\_sr*
  2. **Relevant columns:**
     1. full\_name (STRING): full name of the player
     2. player\_id (STRING): unique identifier code the player
     3. opp\_code (INT): NCAA identifier code of Team 2
* **Query**

**IO Cost: 61.36 MB**

****

| **SELECT**   full\_name,   player\_id,   **SUM**(three\_points\_made) \* 1.0 / **SUM**(three\_points\_att) **as** three\_point\_pct **FROM**   `bigquery-public-data.ncaa\_basketball.mbb\_players\_games\_sr` **WHERE**   season = @season  **GROUP** **BY**   player\_id, full\_name  **HAVING**   **SUM**(three\_points\_att) > @minimum\_shots **ORDER** **BY** three\_point\_pct **DESC**  **LIMIT** 10 |
| --- |

* **API Endpoint**

| GET /v1/**data**/players/three-point-percent Query parameters:  season: (INT64, **default**=2015)  minimum\_shots: (INT64, **default**=10) |
| --- |

* **Result JSON**

| {  "message": "3pt percent request successful",  "error": "",  "data": {  "data": [  {  "full\_name": "David Levitch",  "player\_id": "c3899f37-1eaa-464b-8542-1b6ce591d180",  "three\_point\_pct": 0.6666666666666666  },  {  "full\_name": "Kason Harrell",  "player\_id": "b1afb40a-66da-4222-b7b9-aaf5c81ffb93",  "three\_point\_pct": 0.6428571428571429  },  {  "full\_name": "Darryl Smith",  "player\_id": "6cdc3338-ac42-47b8-8339-06606775d20b",  "three\_point\_pct": 0.5945945945945946  },  {  "full\_name": "Jubril Adekoya",  "player\_id": "6efd1825-74c8-4d45-87f6-a547edfa592f",  "three\_point\_pct": 0.5806451612903226  },  {  "full\_name": "Reid Shackelford",  "player\_id": "7568aac7-5dba-408c-bc63-50bd7d3a298d",  "three\_point\_pct": 0.5789473684210527  },  {  "full\_name": "Robert Mischler",  "player\_id": "db853d24-2396-4624-9be9-2f58dfbbeb6f",  "three\_point\_pct": 0.5789473684210527  },  {  "full\_name": "Evan Fisher ",  "player\_id": "9cec4377-8433-4414-b5e1-bf91b7c4eb2c",  "three\_point\_pct": 0.5714285714285714  },  {  "full\_name": "Connor Burchfield",  "player\_id": "5549e767-2aeb-43b6-baec-d2988e28ffe5",  "three\_point\_pct": 0.5633802816901409  },  {  "full\_name": "Corey Redman",  "player\_id": "5c92391a-2a7e-45b9-8326-207eedbb3124",  "three\_point\_pct": 0.5555555555555556  },  {  "full\_name": "Nick Mayo",  "player\_id": "2662aabd-dd20-4acf-8024-39c66411b3f6",  "three\_point\_pct": 0.5555555555555556  }  ]  },  "status\_code": 200,  "success": **true** } |
| --- |

1. **Feature: Fetch All Player Names**

* **Abstract**: Fetch players' names limited by result size. This is an important subquery to fetch names of players which are to be used as input for subsequent queries.
* **Goal**: To fetch name of all players playing in the tournament (limiting result by size)
* Database Detail
  1. **Table:** *bigquery-public-data.ncaa\_basketball.mbb\_players\_games\_sr*
  2. **Relevant columns:**
     1. full\_name (STRING): player’s exact name
* **Query**

**IO Cost: 12.73 MB**

****

| **SELECT**   **DISTINCT** full\_name **AS** full\_name  **FROM**   `bigquery-public-data.ncaa\_basketball.mbb\_players\_games\_sr`  **ORDER** **BY**   full\_name  **LIMIT** @result\_size |
| --- |

* **API Endpoint**

| curl -X 'GET' \  'http://localhost:8000/v1/data/players/10' \  -H 'accept: application/json' |
| --- |

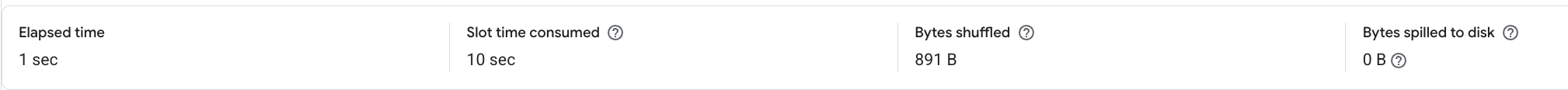
* **Result JSON**

| {  "message": "players fetched successfully",  "error": "",  "data": {  "players": [  {  "full\_name": " Jordan Van Ommering"  },  {  "full\_name": "A'Ram Johnson"  },  {  "full\_name": "A'Torey Everett"  },  {  "full\_name": "A'Torri Shine"  },  {  "full\_name": "A'kanni White"  },  {  "full\_name": "A.C. Reid"  },  {  "full\_name": "A.J. Astroth"  },  {  "full\_name": "A.J. Avery"  },  {  "full\_name": "A.J. Bowers"  },  {  "full\_name": "A.J. Bullard"  }  ]  },  "status\_code": 200,  "success": **true** } |
| --- |

1. **Feature: Player Performance Comparison**

* **Abstract**: Performance comparison between players based on their metrics enabling a deeper understanding of individual contributions, strengths and weaknesses. (**Key metrics for comparison - scoring efficiency, playmaking and passing, defensive impact, overall efficiency**)
* **Goal**: To compare performance between two players based on key metrics
* **Database Detail**
  1. **Table:** *bigquery-public-data.ncaa\_basketball.mbb\_players\_games\_sr*
  2. **Relevant columns:**
     1. full\_name (STRING): player’s exact name
     2. team\_name (STRING) : player’s team name
     3. primary\_position
     4. field\_goals\_made
     5. assists
     6. offensive\_rebounds
     7. defensive\_rebounds
     8. steals
     9. blocks
     10. turnovers
* **Query**

**IO Cost: 58.39 MB**

****

| WITH PlayerData AS (  **SELECT**  full\_name,  team\_name,  primary\_position,  **SUM**(field\_goals\_made) **AS** total\_goals,  **SUM**(assists) **AS** total\_assists,  **SUM**(offensive\_rebounds) **as** total\_orebs,  **SUM**(defensive\_rebounds) **as** total\_drebs,  **SUM**(steals) **as** total\_steals,  **SUM**(blocks) **as** total\_blocks,  **SUM**(turnovers) **as** total\_turnovers  **FROM**  `bigquery-public-data.ncaa\_basketball.mbb\_players\_games\_sr`  **WHERE**  full\_name **IN** (@player1\_name, @player2\_name)  **GROUP** **BY** full\_name, team\_name, primary\_position  ),  Player1 **AS** (  **SELECT** \* **FROM** PlayerData **WHERE** full\_name = @player1\_name  ),  Player2 **AS** (  **SELECT** \* **FROM** PlayerData **WHERE** full\_name = @player2\_name  )  **SELECT**  P1.full\_name **AS** player1\_name,  P2.full\_name **AS** player2\_name,  P1.primary\_position **as** player1\_position,  P2.primary\_position **as** player2\_position,  P1.team\_name **AS** player1\_team,  P2.team\_name **AS** player2\_team,  P1.total\_goals **AS** player1\_goals,  P2.total\_goals **AS** player2\_goals,  P1.total\_assists **AS** player1\_assists,  P2.total\_assists **AS** player2\_assists,  (P1.total\_goals + P1.total\_assists + P1.total\_orebs + P1.total\_drebs + P1.total\_steals + P1.total\_blocks - P1.total\_turnovers) **AS** player1\_efficiency,  (P2.total\_goals + P2.total\_assists + P2.total\_orebs + P2.total\_drebs + P2.total\_steals + P2.total\_blocks - P2.total\_turnovers) **AS** player2\_efficiency  **FROM**  Player1 P1,  Player2 P2 |
| --- |

* **API Endpoint**

| curl -X 'POST' \  'http://localhost:8000/v1/data/players/compare' \  -H 'accept: application/json' \  -H 'Content-**Type**: application/json' \  -d '{  "player1\_name": {  "full\_name": "A.C. Reid"  },  "player2\_name": {  "full\_name": "A.J. Avery"  } }' |
| --- |

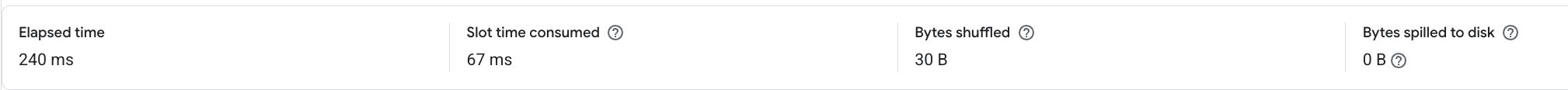
* **Result JSON**

| {  "message": "player comparison successful",  "error": "",  "data": {  "data": [  {  "player1\_name": "A.C. Reid",  "player2\_name": "A.J. Avery",  "player1\_position": "NA",  "player2\_position": "NA",  "player1\_team": "Flames",  "player2\_team": "Broncos",  "player1\_goals": 177,  "player2\_goals": 24,  "player1\_assists": 136,  "player2\_assists": 4,  "player1\_efficiency": 457,  "player2\_efficiency": 120  },  {  "player1\_name": "A.C. Reid",  "player2\_name": "A.J. Avery",  "player1\_position": "NA",  "player2\_position": "NA",  "player1\_team": "Tigers",  "player2\_team": "Broncos",  "player1\_goals": 4,  "player2\_goals": 24,  "player1\_assists": 4,  "player2\_assists": 4,  "player1\_efficiency": 15,  "player2\_efficiency": 120  }  ]  },  "status\_code": 200,  "success": **true** } |
| --- |

1. **Feature: Fetch Team Stats For Season**

* **Abstract**: Fetch season stats for a team given team name.
* **Goal**: By fetching stats of a team for a current season, we can make better predictions for winners of the current season. It also gives us insight upon generic team gameplay by highlighting stats such as - assists, rebounds, turnovers etc. Later we will also touch upon to see which has the most impact on a team's success.
* **Database Details**
  1. **Table:** *bigquery-public-data.ncaa\_basketball.mbb\_teams\_games\_sr*
  2. **Relevant columns:**
     1. team\_id (STRING): team name
     2. season (INT) : season for which stats is to be retrieved
     3. points
     4. field\_goals\_pct
     5. three\_points\_pct
     6. free\_throws\_pct
     7. turnovers
     8. assists
     9. steals
     10. blocks
     11. personal\_fouls
     12. fast\_break\_pts
     13. second\_chance\_pts
     14. points\_off\_turnovers
* **Query**

**IO Cost: 7.75 MB**

****

| **SELECT   AVG(CASE WHEN home\_team THEN 1.0 ELSE 0.0 END) as home\_team,   AVG(points) as points,  AVG(field\_goals\_pct) as field\_goals\_pct,  AVG(three\_points\_pct) as three\_points\_pct,  AVG(free\_throws\_pct) as free\_throws\_pct,  AVG(rebounds) as rebounds,  AVG(assists) as assists,  AVG(turnovers) as turnovers,  AVG(steals) as steals,  AVG(blocks) as blocks,  AVG(personal\_fouls) as personal\_fouls,  AVG(fast\_break\_pts) as fast\_break\_pts,  AVG(second\_chance\_pts) as second\_chance\_pts,  AVG(points\_off\_turnovers) as points\_off\_turnovers  FROM `bigquery-public-data.ncaa\_basketball.mbb\_teams\_games\_sr`  WHERE team\_id = @team\_id AND season = @season** |
| --- |

* **API Endpoint**

| **curl -X 'POST' \  'http://localhost:8000/v1/data/teams/stats' \  -H 'accept: application/json' \  -H 'Content-Type: application/json' \  -d '{  "team\_name": "Duke",  "season": 2016 }'** |
| --- |

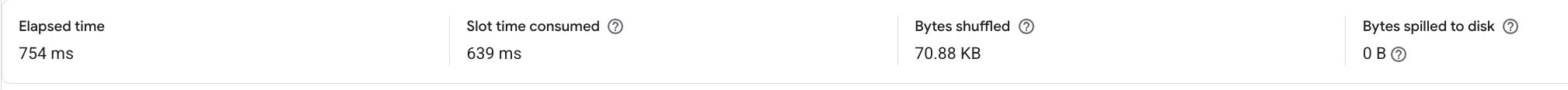
* **Result JSON :**

| **{  "message": "teams stats fetched successfully",  "error": "",  "data": {  "data": {  "home\_team": 0.6216216216216215,  "points": 80.83783783783782,  "field\_goals\_pct": 47.556756756756755,  "three\_points\_pct": 37.837837837837824,  "free\_throws\_pct": 75.89189189189187,  "rebounds": 33.94594594594595,  "assists": 13.081081081081082,  "turnovers": 11.162162162162163,  "steals": 6.108108108108108,  "blocks": 4.64864864864865,  "personal\_fouls": 18.243243243243235,  "fast\_break\_pts": 5.162162162162162,  "second\_chance\_pts": 10.432432432432432,  "points\_off\_turnovers": 14.486486486486486  }  },  "status\_code": 200,  "success": true }** |
| --- |

1. **Feature: Fetch Historical Matches Between Teams**

* **Abstract**: Query Historical Matches and their game data between two teams
* **Goal** : This query is used to provide additional insights when the winner between two teams is to be predicted.
* **Database Detail**
  1. **Tables:**
     1. *bigquery-public-data.ncaa\_basketball.mbb\_games\_sr*
     2. *bigquery-public-data.ncaa\_basketball.mbb\_teams\_games\_sr*
  2. **Relevant columns:**
     1. team\_id (STRING): team name
     2. game\_id(INT) :
     3. scheduled\_date
     4. market
     5. season
* **Query**

**IO Cost: 6.66 MB**

****

| **WITH** **matchups** **AS** (  SELECT  g.game\_id,  g.scheduled\_date,  t1.market AS team1,  t2.market AS team2,  CASE WHEN t1.win THEN t1.market ELSE t2.market END AS winner  FROM `bigquery-public-data.ncaa\_basketball.mbb\_games\_sr` g  JOIN `bigquery-public-data.ncaa\_basketball.mbb\_teams\_games\_sr` t1  ON g.game\_id = t1.game\_id  JOIN `bigquery-public-data.ncaa\_basketball.mbb\_teams\_games\_sr` t2  ON g.game\_id = t2.game\_id AND t1.team\_id < t2.team\_id  WHERE ((t1.team\_id = @team1\_id AND t2.team\_id = @team2\_id)  OR (t1.team\_id = @team2\_id AND t2.team\_id = @team1\_id))  AND g.season = @season  )  **SELECT** \* **FROM** **matchups** |
| --- |

* **API Endpoint**

| curl -X 'POST' \  'http://localhost:8000/v1/data/teams/past-matchups' \  -H 'accept: application/json' \  -H 'Content-**Type**: application/json' \  -d '{  "team1\_name": "Duke",  "team2\_name": "North Carolina",  "season": 2015 }' |
| --- |

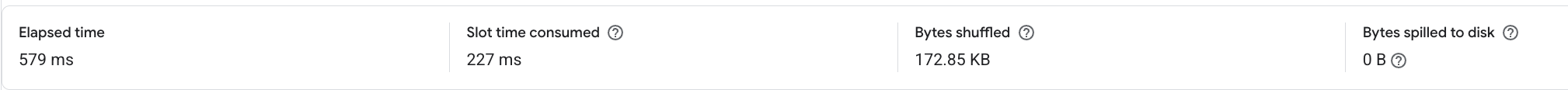
* **Result JSON :**

| **{  "message": "team matchups data fetched successfully",  "error": "",  "data": {  "data": [  {  "game\_id": "9b16a7dc-6f8b-46bf-be85-1bf0e88023b0",  "scheduled\_date": "2016-02-18",  "team1": "North Carolina",  "team2": "Duke",  "winner": "Duke"  },  {  "game\_id": "ad743c4c-2625-407a-8950-023e093f5119",  "scheduled\_date": "2016-03-05",  "team1": "North Carolina",  "team2": "Duke",  "winner": "North Carolina"  },  {  "game\_id": "b4113b16-0838-466e-970e-319057ec82df",  "scheduled\_date": "2015-03-08",  "team1": "North Carolina",  "team2": "Duke",  "winner": "Duke"  },  {  "game\_id": "b0ab87d6-6f00-4eba-a8ef-a1d8a6644575",  "scheduled\_date": "2014-02-21",  "team1": "North Carolina",  "team2": "Duke",  "winner": "North Carolina"  },  {  "game\_id": "0c0d9e2d-d7c1-44dd-922b-85f0647c07f1",  "scheduled\_date": "2014-03-09",  "team1": "North Carolina",  "team2": "Duke",  "winner": "Duke"  },  {  "game\_id": "8a6416e9-3295-4f1d-b3e4-d6507114261e",  "scheduled\_date": "2015-02-19",  "team1": "North Carolina",  "team2": "Duke",  "winner": "Duke"  }  ]  },  "status\_code": 200,  "success": true}** |
| --- |

1. **Feature: Fetch All Teams**

* **Abstract**: Fetch all team names playing in the tournament
* **Goal**: This is essential data which is used to supplement other queries. It was also used to create a ***‘team-mappings.json’*** file in the project as opposed to lazy loading team names each time for a query.
* **Database Detail**
  1. **Tables:**
     1. *bigquery-public-data.ncaa\_basketball.mbb\_teams\_games\_sr*
  2. **Relevant columns:**
     1. team\_id (STRING): team name
     2. market
* **Query**

**IO Cost: 2.91 MB**

****

| **SELECT**   **DISTINCT** team\_id,   market **AS** team\_name  **FROM** `bigquery-public-data.ncaa\_basketball.mbb\_teams\_games\_sr`  **WHERE** market **IS** **NOT** **NULL**  **ORDER** **BY** team\_name |
| --- |

* **API Endpoint**

| curl -X 'GET' \  'http://localhost:8000/v1/data/teams' \  -H 'accept: application/json' |
| --- |

* **Result JSON**

| **{  "message": "all teams fetched successfully",  "error": "",  "data": {  "teams": {  "Abilene Christian": "a52b2ece-1f87-45b5-ae1e-8d0920479965",  "Adams State": "b4b87d2c-9af2-455b-adc1-4e484c078912",  "Adrian": "992f4869-3d63-4ddf-888a-e7bd25f61b2a",  "Air Force": "aa7af640-5762-4686-9181-39f7b8a8186e",  "Akron": "56fe0ab2-e4f0-47b9-8726-9ce23ebcde20",  "Alabama": "c2104cdc-c83d-40d2-a3cd-df986e29f5d3",  "Alabama A&M": "949c3398-85e4-4c63-ba71-9a82e06ddea4", ............ ............ ............. "status\_code": 200,  "success": true }** |
| --- |

1. **Feature: Winner Prediction API (Based on ML Model)**

* **Abstract**: Given two team names, and season (optional), this prediction method helps predict who likely will be winner in a hypothetical matchup based on a pre-trained ML model trained on NCAA Basketball team\_games dataset.
* **Goal**: To compare probable winner between two teams
* **Database Detail**

**Table** : bigquery-public-data.ncaa\_basketball.mbb\_teams\_games\_sr

* **Query:** Internally uses Fetch Team Stats Query with ML Model for prediction.

Also provides additional insights using historical matchups query.

* **API Endpoint**

| curl -X 'POST' \  'http://localhost:8000/v1/data/teams/predict-winner' \  -H 'accept: application/json' \  -H 'Content-**Type**: application/json' \  -d '{  "team1\_name": "Duke",  "team2\_name": "Arkansas" }' |
| --- |

* **Result JSON**

| {  "message": "prediction result generation successful",  "error": "",  "data": {  "result": {  "team1": "Duke",  "team2": "Arkansas",  "team1\_win\_prob": 0.99,  "team2\_win\_prob": 0.77,  "historical\_matchups": [  []  ]  }  },  "status\_code": 200,  "success": **true** } |
| --- |

1. **Feature: Team Performance Metrics**

* **Abstract**: This query is designed to analyze the performance of a NCAA’s basketball teams for a specific or multiple seasons.
* **Goal:** The main purpose of the query is to summarize the performance of teams in such a way that it is easy to understand for users like fans, analysts or coaches.
  + Determine success of a team: with wins and win percentages
  + Offensive and defensive efficiency of a team
  + Compare teams: with different seasons or different teams
  + Head-to-head analysis of a team through different seasons
* **Solution**: This query is used to calculate team performance metrics such as Win Percentage, Average points scored and allowed(they could have scored or gave up), Points per possession(how efficiently team scored per offensive plays), Offensive and defensive efficiency (how team scored and prevented points with respect to possessions).
* **Database Detail**

1. **Table:** *bigquery-public-data.ncaa\_basketball.mbb\_games\_sr*
2. **Relevant columns:**
   1. h\_market (STRING): The name of the home team
   2. a\_market (STRING): The name of the away team
   3. season (INT64): The year of the game (e.g., 2015), used to filter games by season and group results
   4. h\_points\_game (INT64): Total points scored by the home team in a game, used to calculate wins and points scored/allowed.
   5. a\_points\_game (INT64): Total points scored by the away team in a game, used to calculate wins and points scored/allowed.
   6. h\_field\_goals\_att (INT64): Number of field goal attempts by the home team, used in the possession calculation.
   7. a\_field\_goals\_att (INT64): Number of field goal attempts by the away team, used in the possession calculation.
   8. h\_offensive\_rebounds (INT64): Number of offensive rebounds by the home team, used to adjust possessions
   9. a\_offensive\_rebounds (INT64): Number of offensive rebounds by the away team, used to adjust possessions.
   10. h\_turnovers (INT64): Number of turnovers by the home team, used in the possession calculation.
   11. a\_turnovers (INT64): Number of turnovers by the away team, used in the possession calculation.
   12. h\_free\_throws\_att (INT64): Number of free throw attempts by the home team, used in the possession calculation (weighted by 0.475).
   13. a\_free\_throws\_att (INT64): Number of free throw attempts by the away team, used in the possession calculation.

* **Query (old version):**

**IO Cost: 3** MB

| WITH TeamStats AS (  **SELECT**  team\_name,  season,  SUM(games\_played) **AS** games\_played,  SUM(wins) **AS** wins,  SUM(points\_scored) **AS** total\_points\_scored,  SUM(points\_allowed) **AS** total\_points\_allowed,  SUM(possessions) **AS** total\_possessions  **FROM** (  **SELECT**  h\_market **AS** team\_name,  season,  COUNT(\*) **AS** games\_played,  SUM(**CASE** **WHEN** h\_points\_game > a\_points\_game **THEN** 1 **ELSE** 0 **END**) **AS** wins,  SUM(h\_points\_game) **AS** points\_scored,  SUM(a\_points\_game) **AS** points\_allowed,  SUM(h\_field\_goals\_att - h\_offensive\_rebounds + h\_turnovers + 0.475 \* h\_free\_throws\_att) **AS** possessions  **FROM**  `bigquery-public-data.ncaa\_basketball.mbb\_games\_sr`  **WHERE**  {season\_filter}  **AND** h\_points\_game **IS** **NOT** **NULL**  **AND** a\_points\_game **IS** **NOT** **NULL**  **AND** h\_market **IS** **NOT** **NULL**  {team\_filter\_home}  **GROUP** **BY**  h\_market, season  **UNION** **ALL**  **SELECT**  a\_market **AS** team\_name,  season,  COUNT(\*) **AS** games\_played,  SUM(**CASE** **WHEN** a\_points\_game > h\_points\_game **THEN** 1 **ELSE** 0 **END**) **AS** wins,  SUM(a\_points\_game) **AS** points\_scored,  SUM(h\_points\_game) **AS** points\_allowed,  SUM(a\_field\_goals\_att - a\_offensive\_rebounds + a\_turnovers + 0.475 \* a\_free\_throws\_att) **AS** possessions  **FROM**  `bigquery-public-data.ncaa\_basketball.mbb\_games\_sr`  **WHERE**  {season\_filter}  **AND** a\_points\_game **IS** **NOT** **NULL**  **AND** h\_points\_game **IS** **NOT** **NULL**  **AND** a\_market **IS** **NOT** **NULL**  {team\_filter\_away}  **GROUP** **BY**  a\_market, season  )  **GROUP** **BY**  team\_name, season ), TeamMetrics **AS** (  **SELECT**  team\_name,  season,  games\_played,  wins,  ROUND(wins / NULLIF(games\_played, 0), 3) **AS** win\_percentage,  ROUND(total\_points\_scored / NULLIF(games\_played, 0), 1) **AS** avg\_points\_scored,  ROUND(total\_points\_allowed / NULLIF(games\_played, 0), 1) **AS** avg\_points\_allowed,  ROUND(total\_points\_scored / NULLIF(total\_possessions, 0), 3) **AS** points\_per\_possession,  ROUND(100 \* total\_points\_scored / NULLIF(total\_possessions, 0), 1) **AS** offensive\_efficiency,  ROUND(100 \* total\_points\_allowed / NULLIF(total\_possessions, 0), 1) **AS** defensive\_efficiency  **FROM**  TeamStats  **WHERE**  total\_possessions > 0  **AND** games\_played >= 10 ) **SELECT**  team\_name,  season,  win\_percentage,  avg\_points\_scored,  avg\_points\_allowed,  points\_per\_possession,  offensive\_efficiency,  defensive\_efficiency **FROM**  TeamMetrics {order\_by} **LIMIT** {**limit**}; |
| --- |
|  |

* **API Endpoint**
  1. **Team performance Endpoint**

| GET /v1/data/teams/performance  Query parameters:  season (INT64, optional) seasons (ARRAY<INT64>, optional) |
| --- |

* 1. **Analyze a team Endpoint**

| GET /v1/data/teams/performance/**analyze**  **Query** **parameters**:  team\_name (**STRING**, **required**) season (INT64, optional) seasons (ARRAY<INT64>, optional) |
| --- |

* 1. **Head-to-head analysis Endpoint**

| **GET** /v1/data/teams/performance/head-**to**-head  Query parameters:  team\_names (ARRAY<STRING>, required) season (INT64, **optional**) seasons (ARRAY<INT64>, **optional**) |
| --- |

* 1. **Get Top offensive team Endpoint**

| GET /v1/**data**/teams/top/offensive  Query parameters:  season (INT64, **optional**) seasons (ARRAY<INT64>, **optional**) limit (INT64, **optional**) |
| --- |

* 1. **Get Top defensive team Endpoint**

| GET /v1/**data**/teams/top/defensive Query parameters: season (INT64, **optional**) seasons (ARRAY<INT64>, **optional**)  limit (INT64, **optional**) |
| --- |

* **Result JSON**

1. **Team performance Endpoint**

<http://127.0.0.1:8000/v1/data/teams/performance>

| {  "message": "Team performance data retrieved successfully",  "error": "",  "data": {  "teams": [  {  "team\_name": "Abilene Christian",  "season": 2013,  "win\_percentage": 0.355,  "avg\_points\_scored": 71.4,  "avg\_points\_allowed": 71.9,  "points\_per\_possession": 8.224,  "offensive\_efficiency": 822.4,  "defensive\_efficiency": 827.6  },  {  "team\_name": "Air Force",  "season": 2013,  "win\_percentage": 0.4,  "avg\_points\_scored": 66,  "avg\_points\_allowed": 69.1,  "points\_per\_possession": 7.396,  "offensive\_efficiency": 739.6,  "defensive\_efficiency": 774.7  },  {  "team\_name": "Akron",  "season": 2013,  "win\_percentage": 0.618,  "avg\_points\_scored": 68.6,  "avg\_points\_allowed": 66.9,  "points\_per\_possession": 8.715,  "offensive\_efficiency": 871.5,  "defensive\_efficiency": 850.6  },  {  "team\_name": "Alabama",  "season": 2013,  "win\_percentage": 0.406,  "avg\_points\_scored": 67.8,  "avg\_points\_allowed": 67.2,  "points\_per\_possession": 1.022,  "offensive\_efficiency": 102.2,  "defensive\_efficiency": 101.2  },  {  "team\_name": "Appalachian State",  "season": 2013,  "win\_percentage": 0.3,  "avg\_points\_scored": 67.8,  "avg\_points\_allowed": 71.7,  "points\_per\_possession": 9.651,  "offensive\_efficiency": 965.1,  "defensive\_efficiency": 1021.1  }  ]  },  "status\_code": 200,  "success": **true** } |
| --- |

1. **Analyze a team Endpoint**

<http://127.0.0.1:8000/v1/data/teams/performance/analyze?team_name=American%20University&seasons=2013&seasons=2014>

| **{  "message": "Performance data for American University retrieved successfully",  "error": "",  "data": {  "teams": [  {  "team\_name": "American University",  "season": 2013,  "win\_percentage": 0.606,  "avg\_points\_scored": 63.1,  "avg\_points\_allowed": 59.1,  "points\_per\_possession": 16.328,  "offensive\_efficiency": 1632.8,  "defensive\_efficiency": 1530.8  },  {  "team\_name": "American University",  "season": 2014,  "win\_percentage": 0.515,  "avg\_points\_scored": 58.7,  "avg\_points\_allowed": 58.6,  "points\_per\_possession": 1.01,  "offensive\_efficiency": 101,  "defensive\_efficiency": 100.7  }  ]  },  "status\_code": 200,  "success": true }** |
| --- |

1. **Head-to-head analysis Endpoint**

<http://127.0.0.1:8000/v1/data/teams/performance/head-to-head?team_names=American%20University&team_names=Alcorn%20State&seasons=2013&seasons=2014&seasons=2015>

| **{  "message": "Head-to-head performance data for American University, Alcorn State retrieved successfully",  "error": "",  "data": {  "teams": [  {  "team\_name": "Alcorn State",  "season": 2013,  "win\_percentage": 0.387,  "avg\_points\_scored": 64,  "avg\_points\_allowed": 67.8,  "points\_per\_possession": 13.806,  "offensive\_efficiency": 1380.6,  "defensive\_efficiency": 1461.3  },  {  "team\_name": "American University",  "season": 2013,  "win\_percentage": 0.606,  "avg\_points\_scored": 63.1,  "avg\_points\_allowed": 59.1,  "points\_per\_possession": 16.328,  "offensive\_efficiency": 1632.8,  "defensive\_efficiency": 1530.8  },  {  "team\_name": "Alcorn State",  "season": 2014,  "win\_percentage": 0.188,  "avg\_points\_scored": 64.6,  "avg\_points\_allowed": 74.3,  "points\_per\_possession": 0.928,  "offensive\_efficiency": 92.8,  "defensive\_efficiency": 106.8  },  {  "team\_name": "American University",  "season": 2014,  "win\_percentage": 0.515,  "avg\_points\_scored": 58.7,  "avg\_points\_allowed": 58.6,  "points\_per\_possession": 1.01,  "offensive\_efficiency": 101,  "defensive\_efficiency": 100.7  },  {  "team\_name": "Alcorn State",  "season": 2015,  "win\_percentage": 0.5,  "avg\_points\_scored": 67.1,  "avg\_points\_allowed": 72.1,  "points\_per\_possession": 0.948,  "offensive\_efficiency": 94.8,  "defensive\_efficiency": 101.9  },  {  "team\_name": "American University",  "season": 2015,  "win\_percentage": 0.387,  "avg\_points\_scored": 57.6,  "avg\_points\_allowed": 65.7,  "points\_per\_possession": 0.919,  "offensive\_efficiency": 91.9,  "defensive\_efficiency": 104.9  }  ]  },  "status\_code": 200,  "success": true }** |
| --- |

1. **Get Top offensive team Endpoint**

<http://127.0.0.1:8000/v1/data/teams/top/offensive?season=2013&limit=2>

| {  "message": "Top offensive teams retrieved successfully",  "error": "",  "data": {  "teams": [  {  "team\_name": "IPFW",  "season": 2013,  "win\_percentage": 0.694,  "avg\_points\_scored": 77.1,  "avg\_points\_allowed": 71,  "points\_per\_possession": 41.295,  "offensive\_efficiency": 4129.5,  "defensive\_efficiency": 3803.5  },  {  "team\_name": "Murray State",  "season": 2013,  "win\_percentage": 0.676,  "avg\_points\_scored": 77.9,  "avg\_points\_allowed": 71.9,  "points\_per\_possession": 40.121,  "offensive\_efficiency": 4012.1,  "defensive\_efficiency": 3707.5  }  ]  },  "status\_code": 200,  "success": **true** } |
| --- |

1. **Get Top defensive team Endpoint**

<http://127.0.0.1:8000/v1/data/teams/top/defensive?seasons=2014&seasons=2015&limit=5>

| {  "message": "Top defensive teams retrieved successfully",  "error": "",  "data": {  "teams": [  {  "team\_name": "Kentucky",  "season": 2014,  "win\_percentage": 0.974,  "avg\_points\_scored": 74.4,  "avg\_points\_allowed": 54.3,  "points\_per\_possession": 1.134,  "offensive\_efficiency": 113.4,  "defensive\_efficiency": 82.8  },  {  "team\_name": "San Diego State",  "season": 2014,  "win\_percentage": 0.75,  "avg\_points\_scored": 61.8,  "avg\_points\_allowed": 53.9,  "points\_per\_possession": 0.978,  "offensive\_efficiency": 97.8,  "defensive\_efficiency": 85.2  },  {  "team\_name": "Virginia",  "season": 2014,  "win\_percentage": 0.882,  "avg\_points\_scored": 65.4,  "avg\_points\_allowed": 51.5,  "points\_per\_possession": 1.092,  "offensive\_efficiency": 109.2,  "defensive\_efficiency": 86  },  {  "team\_name": "North Carolina Central",  "season": 2014,  "win\_percentage": 0.758,  "avg\_points\_scored": 69,  "avg\_points\_allowed": 55.2,  "points\_per\_possession": 1.079,  "offensive\_efficiency": 107.9,  "defensive\_efficiency": 86.3  },  {  "team\_name": "Sam Houston State",  "season": 2014,  "win\_percentage": 0.743,  "avg\_points\_scored": 74.1,  "avg\_points\_allowed": 60,  "points\_per\_possession": 1.066,  "offensive\_efficiency": 106.6,  "defensive\_efficiency": 86.4  }  ]  },  "status\_code": 200,  "success": **true** } |
| --- |

* **Optimization Opportunities:**

This query can be optimized using indexes and partitioning to improve performance considering the large size of the NCAA dataset(29805 records for 5 seasons) and complexity of the query involving multiple aggregations, unions and joins.

1. **Indexing**:
   1. B+ Tree Index on season in mbb\_games\_sr table to speed up the {season\_filter} condition, as it is used commonly for filtering the results.
   2. Hash Index on h\_market and a\_market to speedup the lookups for specific teams in {team\_filter\_home} or {team\_filter\_away}.
2. **Partitioning**:
3. Partition the mbb\_games\_sr table by season to reduce the data scanned for season-specific queries, improving I/O efficiency.

* **Estimated IO Cost:** 
  + **Table Size:** 
    - Total records: 29,805, total seasons: 5
    - Logical size: 48.08 MB.
    - Records per season: ~5,961 (29,805 ÷ 5)
    - Size per season: ~9.62 MB (48.08 MB ÷ 5)
    - Row size: ~1.61 KB (48.08 MB ÷ 29,805)
  + **Without Optimization:** 
    - The query scans the entire table twice (home and away subqueries via UNION ALL): 29,805 × 2 = 59,610 rows.
    - Overhead for GROUP BY, aggregations, and filtering (~10%): 59,610 × 1.1 = ~65,571 I/O operations.
    - Data scanned: 59,610 rows × 1.61 KB = ~95.97 MB
    - **Estimated I/O Cost**: ~65,571 I/O operations.
  + **WIth Optimization**
    - B+ Tree Index on season: Reduces scans to ~5,961 rows (~9.62 MB) for a single-season query.
    - Hash Indexes on h\_market, a\_market: Limits team-specific lookups to ~100 rows (~0.161 MB, assuming ~60 teams per season).
    - Partitioning by season: Restricts scans to ~5,961 rows (~9.62 MB).
    - **Estimated I/O Cost:** ~300–600 I/O operations (for indexed/partitioned scans).

**Optimization Results**

The IO cost for the Team performance query was 3 MB.

For query optimization, there are 2 options as indexing and partitioning. As traditional indexing used in databases is not supported in bigquery datasets. So we can optimize it using only required columns in select instead of all columns, which was already done for existing query.

Now, the second option, optimizing the query using Partitioning.

Partitioning is done on the season column and using RANGE\_BUCKET.

* **Query:**

**IO Cost:** 3 MB (One time)

| **CREATE TABLE `my-project-180b-456416.ncaa\_basketball.optimized\_mbb\_games\_sr` PARTITION BY RANGE\_BUCKET(season, GENERATE\_ARRAY(2012, 2018, 1)) CLUSTER BY h\_market, a\_market OPTIONS (  description = "Partitioned and clustered table for NCAA basketball game data",  require\_partition\_filter = true ) AS SELECT  season,  h\_market,  a\_market,  h\_points\_game,  a\_points\_game,  h\_field\_goals\_att,  a\_field\_goals\_att,  h\_offensive\_rebounds,  a\_offensive\_rebounds,  h\_turnovers,  a\_turnovers,  h\_free\_throws\_att,  a\_free\_throws\_att FROM  `bigquery-public-data.ncaa\_basketball.mbb\_games\_sr` WHERE  h\_points\_game IS NOT NULL  AND a\_points\_game IS NOT NULL  AND h\_market IS NOT NULL  AND a\_market IS NOT NULL;** |
| --- |

Using above partition the original query is modified as below:

IO Cost:

The data is spread across 5 seasons so to run a query as per seasons IO cost is as per below table.

| **Seasons** | **IO Cost** |
| --- | --- |
| 1 season | 665 KB |
| 2 seasons | 1.1 MB |
| 3 seasons | 1.7 MB |
| 4 seasons | 2.35 MB |
| 5 seasons | 3 MB |

| **WITH GameStats AS (  SELECT  team\_name,  season,  COUNT(\*) AS games\_played,  SUM(wins) AS wins,  SUM(points\_scored) AS total\_points\_scored,  SUM(points\_allowed) AS total\_points\_allowed,  SUM(possessions) AS total\_possessions  FROM (  SELECT  h\_market AS team\_name,  season,  1 AS games\_played,  CASE WHEN h\_points\_game > a\_points\_game THEN 1 ELSE 0 END AS wins,  h\_points\_game AS points\_scored,  a\_points\_game AS points\_allowed,  (h\_field\_goals\_att - h\_offensive\_rebounds + h\_turnovers + 0.475 \* h\_free\_throws\_att) AS possessions  FROM  `my-project-180b-456416.ncaa\_basketball.optimized\_mbb\_games\_sr`  WHERE  season = 2015  AND h\_market IS NOT NULL  AND h\_market = 'Duke'  UNION ALL  SELECT  a\_market AS team\_name,  season,  1 AS games\_played,  CASE WHEN a\_points\_game > h\_points\_game THEN 1 ELSE 0 END AS wins,  a\_points\_game AS points\_scored,  h\_points\_game AS points\_allowed,  (a\_field\_goals\_att - a\_offensive\_rebounds + a\_turnovers + 0.475 \* a\_free\_throws\_att) AS possessions  FROM  `my-project-180b-456416.ncaa\_basketball.optimized\_mbb\_games\_sr`  WHERE  season = 2015  AND a\_market IS NOT NULL  AND a\_market = 'Duke'  ) AS games  GROUP BY  team\_name, season ), TeamMetrics AS (  SELECT  team\_name,  season,  games\_played,  wins,  ROUND(wins / NULLIF(games\_played, 0), 3) AS win\_percentage,  ROUND(total\_points\_scored / NULLIF(games\_played, 0), 1) AS avg\_points\_scored,  ROUND(total\_points\_allowed / NULLIF(games\_played, 0), 1) AS avg\_points\_allowed,  ROUND(total\_points\_scored / NULLIF(total\_possessions, 0), 3) AS points\_per\_possession,  ROUND(100 \* total\_points\_scored / NULLIF(total\_possessions, 0), 1) AS offensive\_efficiency,  ROUND(100 \* total\_points\_allowed / NULLIF(total\_possessions, 0), 1) AS defensive\_efficiency  FROM  GameStats  WHERE  total\_possessions > 0  AND games\_played >= 10 ) SELECT  team\_name,  season,  win\_percentage,  avg\_points\_scored,  avg\_points\_allowed,  points\_per\_possession,  offensive\_efficiency,  defensive\_efficiency FROM  TeamMetrics ORDER BY offensive\_efficiency DESC LIMIT 10;** |
| --- |

* **Running time:**

**Comparison of run times for both versions:**

| **Metric** | **Query(Old version)** | **Optimized Query** |
| --- | --- | --- |
| Elapsed Time | 491 ms | 346 ms |
| Slot Time Consumed | 469 ms | 554 ms |
| Bytes Shuffled | 268.51 KB | 135.05 KB |
| Bytes Spilled to Disk | 0 B | 0 B |

**Execution Details for Optimized Query:**

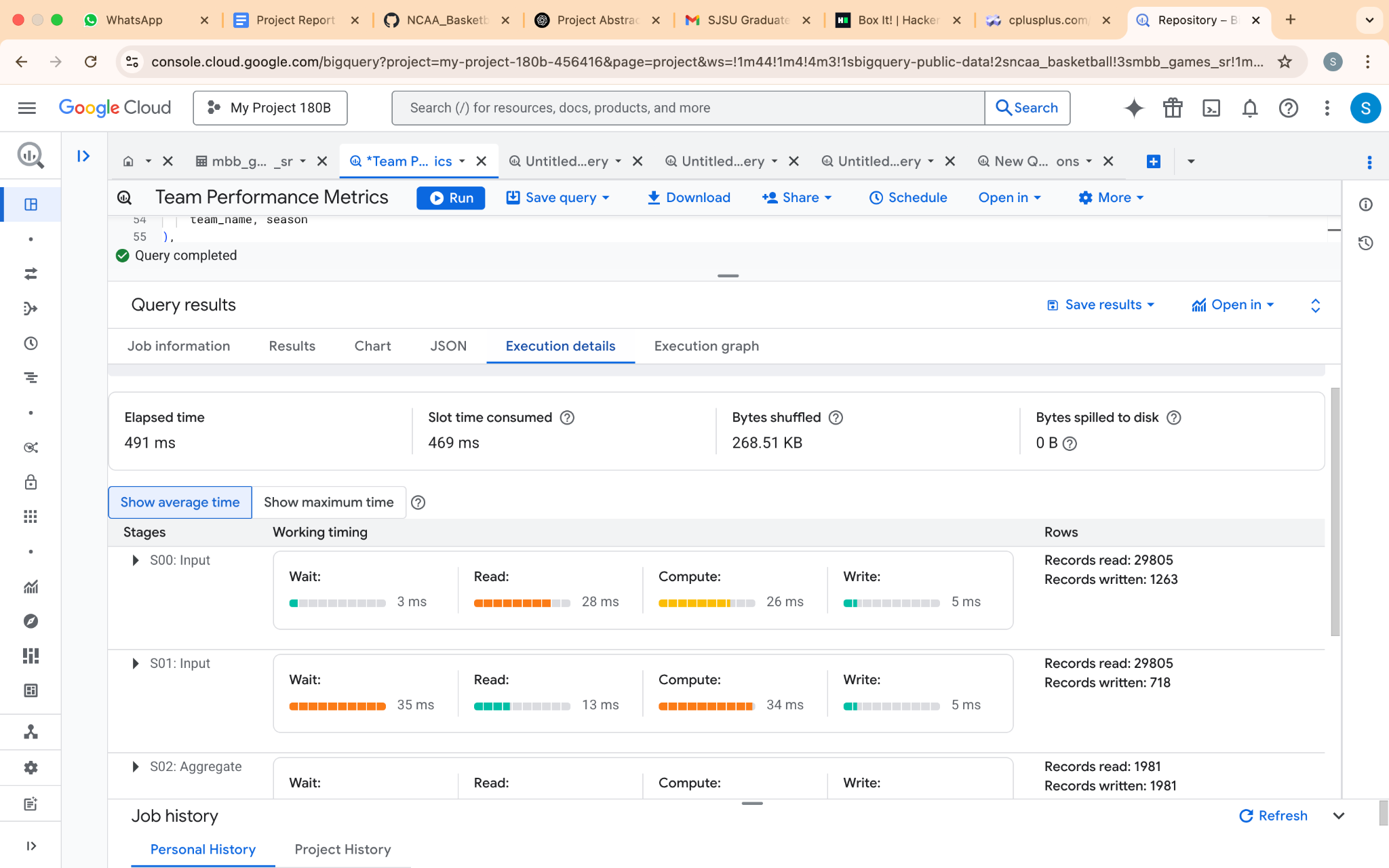
| **Stage** | **Wait (ms)** | **Read (ms)** | **Compute (ms)** | **Write (ms)** | **Records Read** | **Records Written** |
| --- | --- | --- | --- | --- | --- | --- |
| S00: Input | 12 | 63 | 17 | 4 | 23,752 | 1,981 |
| S01: Sort+ | 2 | 0 | 15 | 2 | 1,981 | 10 |
| S02: Output | 2 | 0 | 12 | 18 | 10 | 10 |

**Execution Details for Old Query:**

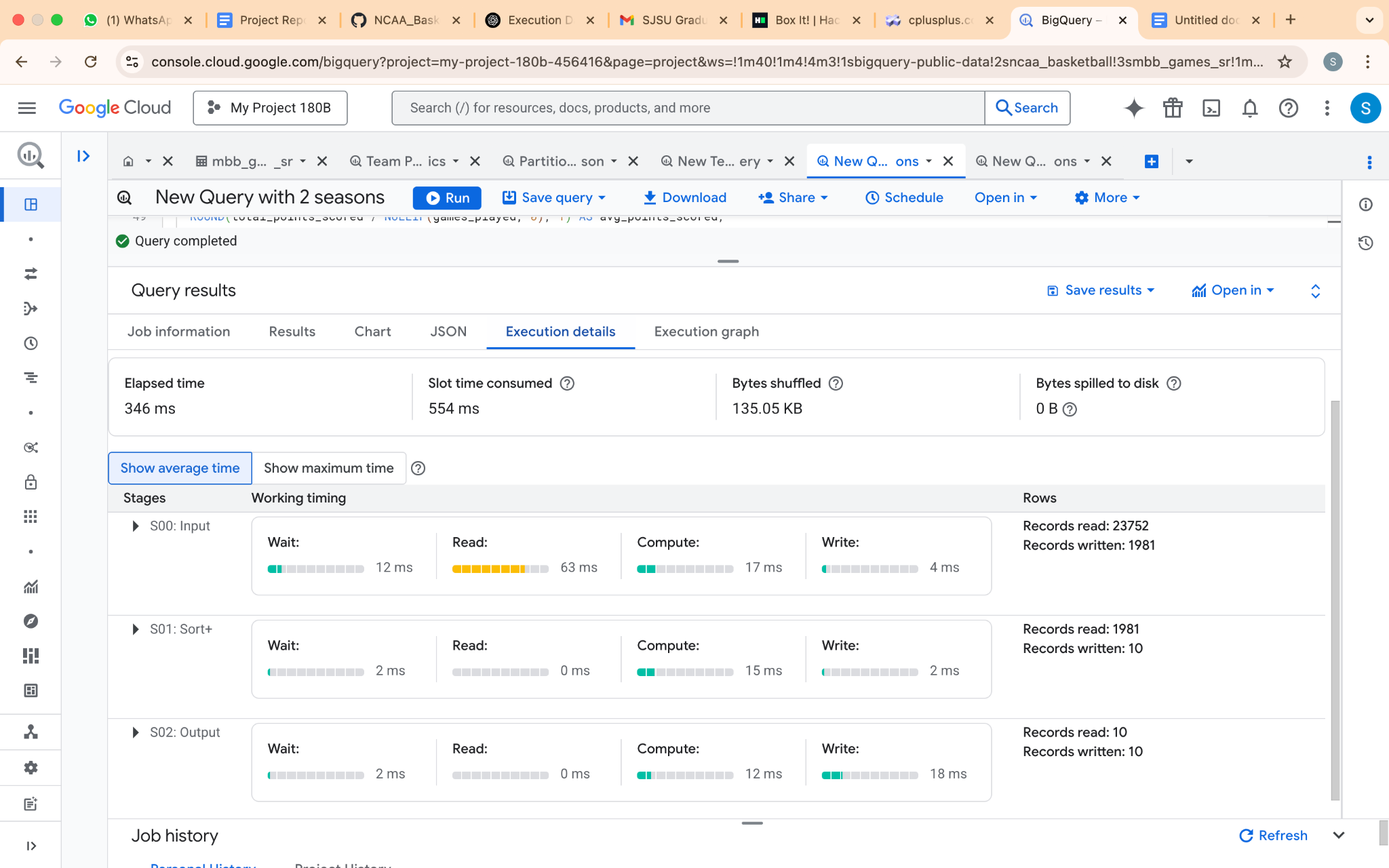
| **Stage** | **Wait (ms)** | **Read (ms)** | **Compute (ms)** | **Write (ms)** | **Records Read** | **Records Written** |
| --- | --- | --- | --- | --- | --- | --- |
| S00: Input | 3 | 28 | 26 | 5 | 29,805 | 1,263 |
| S01: Input | 35 | 13 | 34 | 5 | 29,805 | 718 |
| S02: Aggregate | 3 | 0 | 17 | 17 | 1,981 | 1,981 |
| S03: Sort+ | 1 | 0 | 14 | 11 | 1,981 | 10 |
| S04: Output | 1 | 0 | 10 | 5 | 10 | 10 |

* **Execution Graphs:**

Execution details before query optimization



Execution details after query optimization

****

| Execution graph **Before** query optimization | Execution graph **After** query optimization |
| --- | --- |
|  |  |

* **Impact of optimization:**

Partitioning the ncaa\_basketball’s “mbb\_games\_sr” table by season using RANGE\_BUCKET(season, GENERATE\_ARRAY(2012, 2018, 1)) divides the table into 5 distinct sections. This considerably limits the amount of data scanned when queries include a season/seasons filter.

* + For a single season query (e.g., season = 2013), BigQuery scans only the relevant partition (~9.62 MB) instead of the full table (48.08 MB), reducing data scanned by ~80%.
  + Queries with season filters scan only relevant partitions, improving the performance in predictive analysis even after the table grows.
* **Future improvements:**

To further improve the performance of a query beyond partitioning and clustering below are some ways.

* + Materialized view for Pre-aggregate team metrics such as games\_played, wins, total\_points\_scored, etc. by team\_name and season eliminating “UNION ALL” and “GROUP BY”.
  + Caching: Using Redis to cache frequent team metrics, bypassing BigQuery for repeated queries resulting in 0 query cost for cache hits.
  + Scheduled Queries for Result Caching: Schedule a query to run daily, storing results in a table to avoid real-time data calculations.

1. **Feature: Winner Prediction Model - Training, Testing and Insights**

* **Abstract:** This initiative focuses on creating a machine learning model aimed at forecasting the results of NCAA basketball games by utilizing historical team performance data sourced from **Google BigQuery**. A **Random Forest Classifier** demonstrated an **accuracy of 84.2%**, **precision of 81.4%**, and **recall of 88.7%** when **tested on data from the 2017 season**, pinpointing essential performance indicators such as points scored, rebounds, and shooting efficiency as significant predictors. The model underscores the strategic relevance of home-court advantage and the efficiency of both offensive and defensive plays in determining winning outcomes.
* **Goal:** Develop a predictive model to anticipate the outcomes of NCAA basketball games based on team-level statistics, facilitating data-driven insights into the performance elements that affect winning.
* **Classification Model** : **Random Forest**

Using a **Random Forest Classifier** for sports outcome prediction is effective due to its ability to handle complex, high-dimensional data while minimizing overfitting. Random Forests average multiple decision trees, reducing variance and improving stability compared to single decision trees.

They automatically identify key predictors, helping analysts focus on impactful variables.

Sports data often contain complex interactions (e.g., team dynamics, momentum shifts), which Random Forests model effectively without strict assumptions.By aggregating predictions from many trees trained on random subsets of data, they generalize well to unseen matches.

* **Data Pipeline**
  + **Training and Testing Data Queries**
    - **Table** : bigquery-public-data.ncaa\_basketball.mbb\_teams\_games\_sr
    - **Training Data**: All seasons up to 2016 (47,600 samples)
    - **Testing Data**: 2017 season (12,008 samples)
* **Query:**
* **IO Cost : 10.42 MB**

1. **Training Query (seasons ≤ 2016)**

| SELECT  game\_id,  season,  scheduled\_date,  team\_id,  home\_team,  win,  points,  field\_goals\_pct,  three\_points\_pct,  free\_throws\_pct,  rebounds,  assists,  turnovers,  steals,  blocks,  personal\_fouls,  fast\_break\_pts,  second\_chance\_pts,  points\_off\_turnovers FROM `bigquery-public-data.ncaa\_basketball.mbb\_teams\_games\_sr` WHERE season <= 2016  **AND** win IS **NOT** NULL |
| --- |

1. **Testing Query (season = 2017)**
   * + Game Data For 2017-18 Season

| SELECT  game\_id,  season,  scheduled\_date,  team\_id,  home\_team,  win,  points,  field\_goals\_pct,  three\_points\_pct,  free\_throws\_pct,  rebounds,  assists,  turnovers,  steals,  blocks,  personal\_fouls,  fast\_break\_pts,  second\_chance\_pts,  points\_off\_turnovers FROM `bigquery-public-data.ncaa\_basketball.mbb\_teams\_games\_sr` WHERE season = 2017 # Key change: Using 2017 instead of 2018  **AND** win IS **NOT** NULL |
| --- |

## **Data Insights**

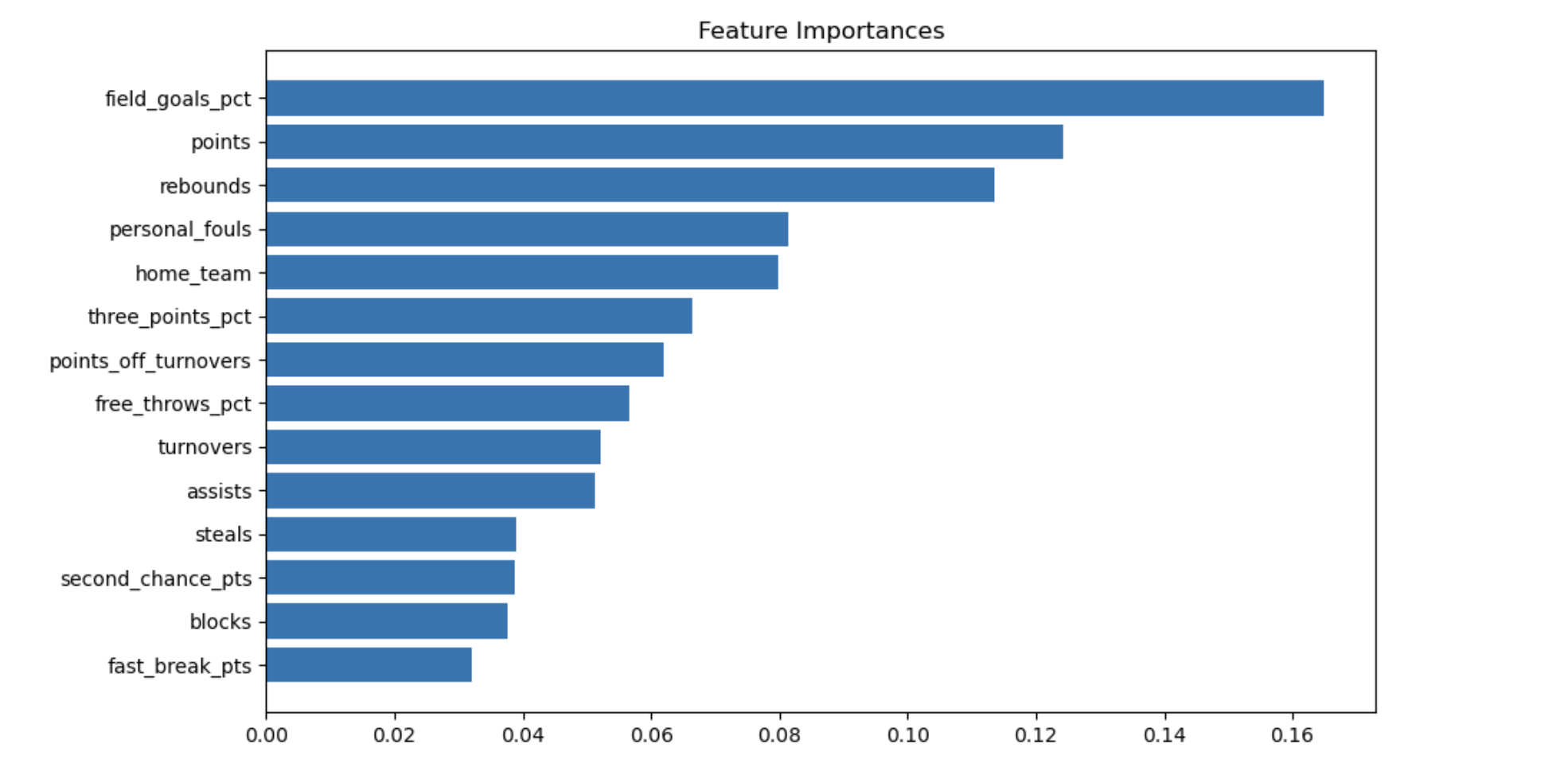
1. **Feature Selection**:
   1. 14 features used, including home\_team, points, rebounds, assists, turnovers, and shooting percentages.
   2. scheduled\_date dropped to avoid temporal bias.
2. **Key Observations**:
   1. **Home Advantage**: Home teams (home\_team=1) won more frequently.
   2. **Shooting Efficiency**: High field goal/three-point percentages strongly correlated with wins.
   3. **Turnover Impact**: Fewer turnovers linked to higher win probability.
3. **Data Preprocessing**:
   1. Boolean columns (home\_team, win) converted to integers.
   2. Numeric NaNs filled with 01.

## 

## **Model Training**

* + **Algorithm**: RandomForestClassifier with 100 estimators and random\_state=42
  + **Feature Importance**:

1. Points
2. Rebounds
3. Field Goal %
4. Three-Point %
5. Assists
6. Turnovers



## **Evaluation Metrics**

| **Metric** | **Score** | **Interpretation** |
| --- | --- | --- |
| Accuracy | 84.2% | Overall correct prediction rate |
| Precision | 81.4% | Reliability of "win" predictions |
| Recall | 88.7% | Effectiveness in capturing actual wins |

## 

## **Predictions and Probabilities**

* + **Top Predictions Example**:

| **game\_id** | **team\_id** | **Predicted Win** | **Win Probability** |
| --- | --- | --- | --- |
| 1a689aee-fec2-49df-822d... | db6e1cab-3fa3-4a93-a673... | 1 | 1.00 |
| dfc8f2a4-ce1c-4023-b53a... | f38b58de-3273-4723-a78b... | 1 | 1.00 |

High-confidence predictions (e.g., 99% probability) aligned with actual outcomes.

## **Key Findings**

1. **Offensive Dominance**: Points and shooting efficiency are the strongest win predictors.
2. **Home-Court Edge**: Home teams win 5-10% more frequently, likely due to crowd support and familiarity.
3. **Defensive Impact**: Rebounds and steals significantly reduce opponent scoring opportunities.

## **Challenges**

* + **Data Limitations**: Missing values in older seasons required imputation.
  + **Computational Load**: Training on 47k samples with 100 estimators demanded significant resources.

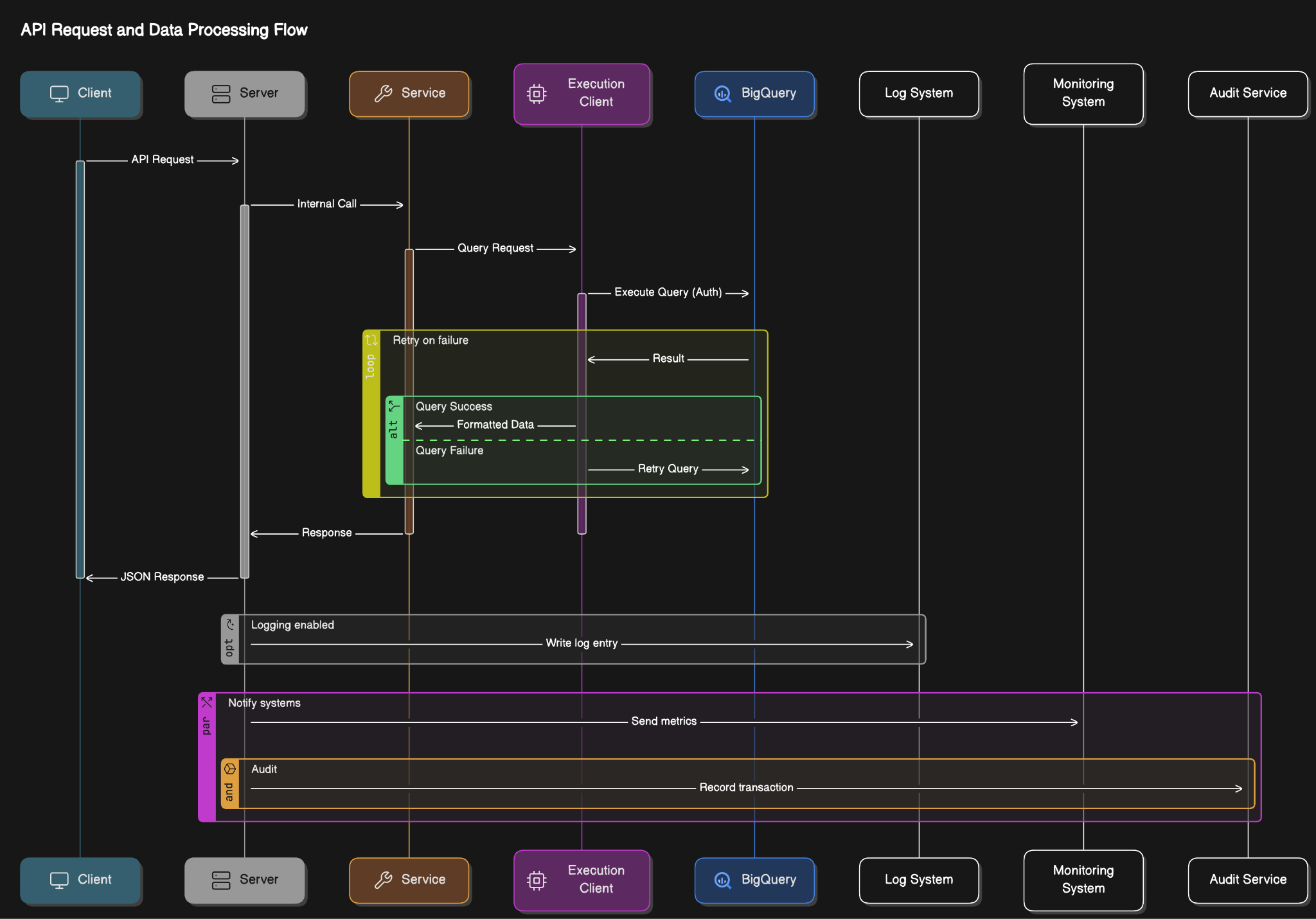
## **Future Work**

1. Expand training data to include recent seasons.
2. Optimize hyperparameters (e.g., tree depth, ensemble size).
3. Deploy as a real-time prediction tool for bettors/analysts.

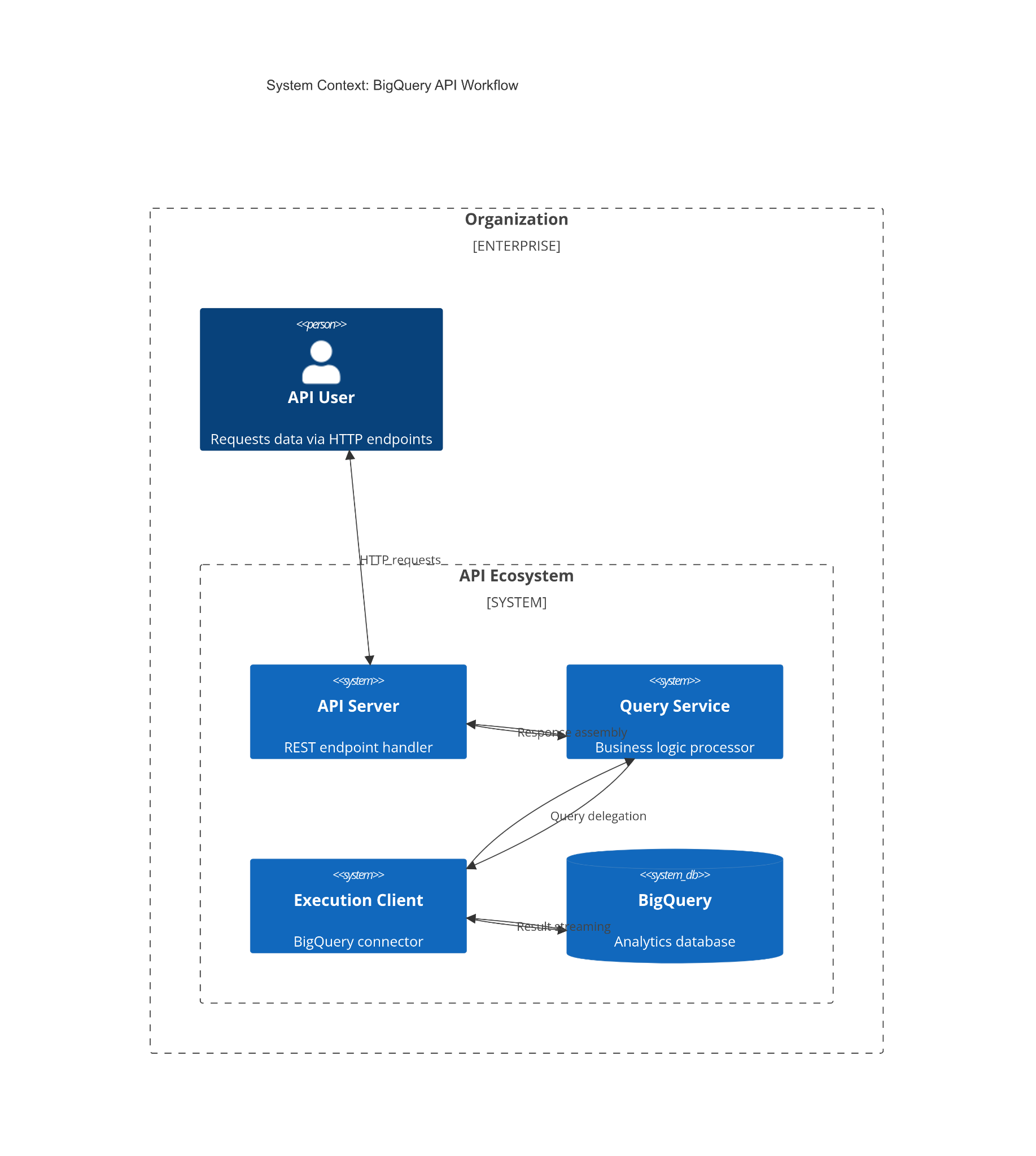
* **Conclusion**: This model offers practical insights into team performance, showcasing the effectiveness of machine learning in sports analytics. Further improvements could enhance both accuracy and practical application.

**Application Design**

**Sequence Diagram :**

****

**Architecture:**

****

**Code Structure:**

****

**Project Conclusion:**

Sports analytics can be a daunting field, especially for those looking to begin their journey of analysis or betting; the sheer amount of stats collected in modern sports is almost incomprehensible. There’s a tricky balance between providing enough relevant information about a given matchup or player and flooding the user with information that only makes their head spin. Competitors within the field have managed to walk the tightrope somewhat, though many use their feat as reason to charge for their services or otherwise privatize them. By creating an open-source API, we would be able to provide information for those who are looking to get into the field who don’t want to shell out money for something they may not have a long-term interest in. Additionally, creating many different API endpoints segment itself out into varying levels of complexity, allowing users or app designers to utilize whatever volume of data fits best. And by including a machine learning model, we have also provided an option to offload the analysis entirely, leading to a minimal knowledge barrier for those looking to enter.

**Citations:**

Below are some of the resources we referred for our project.

1. <https://github.com/search?q=ncaa_basketball&type=repositories>
2. <https://adeshpande3.github.io/Applying-Machine-Learning-to-March-Madness>
3. <https://github.com/jflancer/bigballR>
4. <https://github.com/minimaxir/ncaa-basketball>
5. <https://github.com/grdavis/college-basketball-elo>
6. <https://github.com/timdagostino/NCAAB-Game-Simulator>
7. <https://www.kaggle.com/datasets/ncaa/ncaa-basketball/code>
8. <https://poolgenius.teamrankings.com/ncaa-bracket-picks/>
9. <https://kenpom.com/>