## Project Framework

# 1. Introduction

## Project Description

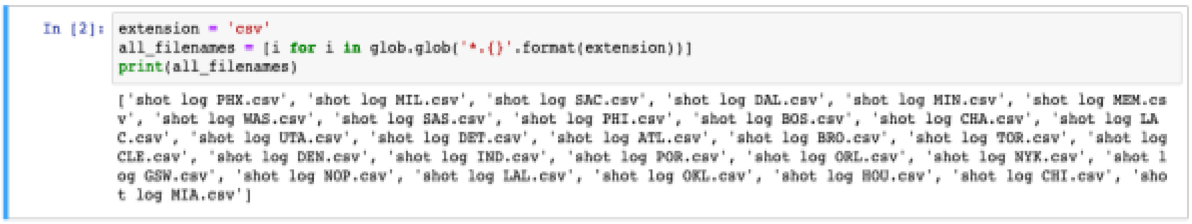
The purpose of this project is to process and visualize NBA 16-17 regular season shot log dataset by using various Big Data technologies and to see how performance of players and teams changes based on various factors.

## Data Definition

NBA 16-17 regular season shot log dataset has been retrieved from Kaggle Competition. The dataset include Game Schedule 16-17-Reg, Player Regular 16-17 Stats, Player Regular 16-17 Stats, NBA team name vs abbreviation, Notes for the shot data and shot log data for each team. In shot log data for each team, the columns were: self previous shot, player position, home game, location x, opponent previous shot, home team, shot type, points, away team, location y, time, date, shoot player, time from last shot, quarter, current shot outcome.

## 2. Data Preparation

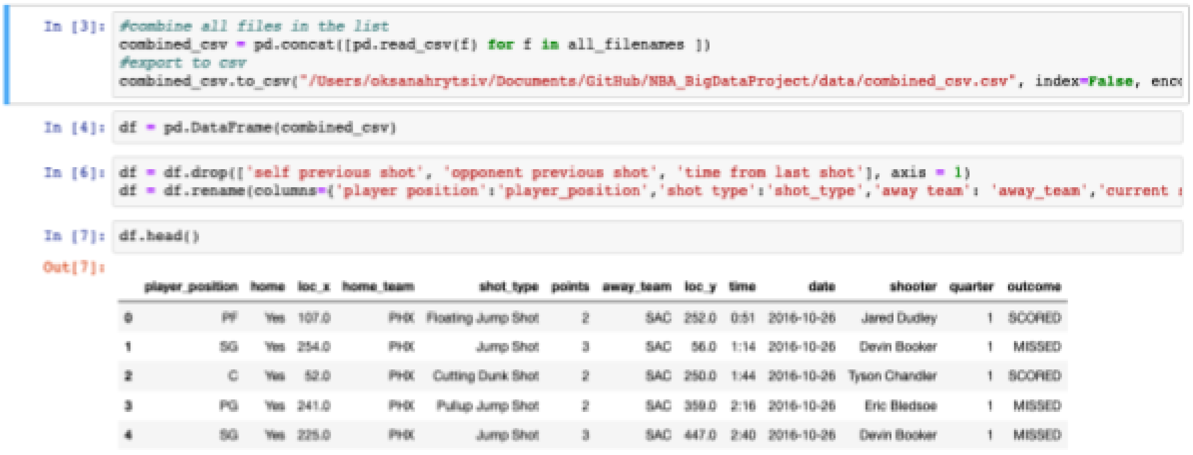
Python was used in the processing stage of data. The shot log data consists of *.csv files for each of the 30 teams.



```
In [2]: extension = 'csv'
        all_filenames = [i for i in glob.glob('*.{}'.format(extension))]
        print(all_filenames)

        ['shot log PHX.csv', 'shot log MIL.csv', 'shot log SAC.csv', 'shot log DAL.csv', 'shot log MIN.csv', 'shot log MEM.cs
        v', 'shot log WAS.csv', 'shot log SAS.csv', 'shot log PHI.csv', 'shot log BOS.csv', 'shot log CHA.csv', 'shot log LA
        C.csv', 'shot log UTA.csv', 'shot log DET.csv', 'shot log ATL.csv', 'shot log BRO.csv', 'shot log TOR.csv', 'shot log
        CLE.csv', 'shot log DEN.csv', 'shot log IND.csv', 'shot log POR.csv', 'shot log ORL.csv', 'shot log NYK.csv', 'shot l
        og GSW.csv', 'shot log NOP.csv', 'shot log LAL.csv', 'shot log OKL.csv', 'shot log HOU.csv', 'shot log CHI.csv', 'sho
        t log MIA.csv']
```

Figure 1. All shot log data files

In the NBA_Court notebook, all files were combined in the list by using the pandas concat function. Afterwards, all was exported to a csv file.

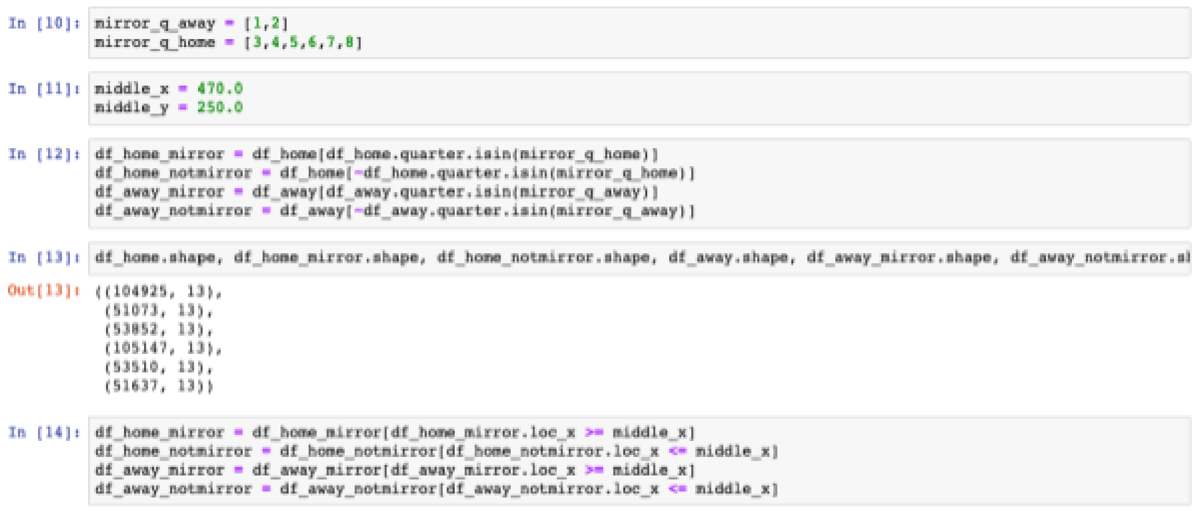


```
In [3]: #combine all files in the list
        combined_csv = pd.concat([pd.read_csv(f) for f in all_filenames ])
        #export to csv
        combined_csv.to_csv("/Users/oksanahrytsiv/Documents/GitHub/NBA_BigDataProject/data/combined_csv.csv", index=False, enc

In [4]: df = pd.DataFrame(combined_csv)

In [6]: df = df.drop(['self previous shot', 'opponent previous shot', 'time from last shot'], axis = 1)
        df = df.rename(columns={'player position':'player_position','shot type':'shot_type','away team': 'away_team','current :

In [7]: df.head()

Out[7]:
```

| | player_position | home | loc_x | home_team | shot_type | points | away_team | loc_y | time | date | shooter | quarter | outcome |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | PF | Yes | 107.0 | PHX | Floating Jump Shot | 2 | SAC | 252.0 | 0:51 | 2016-10-26 | Jared Dudley | 1 | SCORED |
| 1 | SG | Yes | 254.0 | PHX | Jump Shot | 3 | SAC | 56.0 | 1:14 | 2016-10-26 | Devin Booker | 1 | MISSED |
| 2 | C | Yes | 52.0 | PHX | Cutting Dunk Shot | 2 | SAC | 250.0 | 1:44 | 2016-10-26 | Tyson Chandler | 1 | SCORED |
| 3 | PG | Yes | 241.0 | PHX | Pullup Jump Shot | 2 | SAC | 359.0 | 2:16 | 2016-10-26 | Eric Bledsoe | 1 | MISSED |
| 4 | SG | Yes | 225.0 | PHX | Jump Shot | 3 | SAC | 447.0 | 2:40 | 2016-10-26 | Devin Booker | 1 | MISSED |

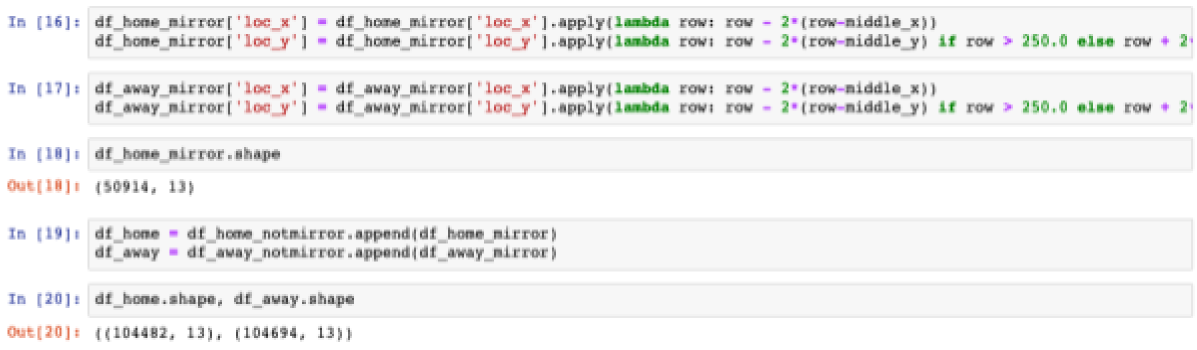
Figure 2. Combined *.csv file

There were some challenges to be resolved. For instance, in a basketball game teams switch sides after halftime. Therefore the basket on which they try to score changes as well. Since the original dataset had shot locations both in the right hand side and the left hand side of the court for the same team in the same match, the shot location plot will not give much information unless it's on the same x-y quadrant. For this we mirrored the data, specifically the shot locations, by calculating the middle point of both x and y axis in order to shift the quarters in which the home team was facing the right side of the court to the left side, looking to have a more homogeneous dataset.



```
In [10]: mirror_q_away = [1,2]
         mirror_q_home = [3,4,5,6,7,8]

In [11]: middle_x = 470.0
         middle_y = 250.0

In [12]: df_home_mirror = df_home[df_home.quarter.isin(mirror_q_home)]
         df_home_notmirror = df_home[~df_home.quarter.isin(mirror_q_home)]
         df_away_mirror = df_away[df_away.quarter.isin(mirror_q_away)]
         df_away_notmirror = df_away[~df_away.quarter.isin(mirror_q_away)]

In [13]: df_home.shape, df_home_mirror.shape, df_home_notmirror.shape, df_away.shape, df_away_mirror.shape, df_away_notmirror.sl

Out[13]: ((104925, 13),
          (51073, 13),
          (53852, 13),
          (105147, 13),
          (53510, 13),
          (51637, 13))

In [14]: df_home_mirror = df_home_mirror[df_home_mirror.loc_x >= middle_x]
         df_home_notmirror = df_home_notmirror[df_home_notmirror.loc_x <= middle_x]
         df_away_mirror = df_away_mirror[df_away_mirror.loc_x >= middle_x]
         df_away_notmirror = df_away_notmirror[df_away_notmirror.loc_x <= middle_x]
```

Figure 3. Mirroring Data



```
In [16]: df_home_mirror['loc_x'] = df_home_mirror['loc_x'].apply(lambda row: row - 2*(row-middle_x))
         df_home_mirror['loc_y'] = df_home_mirror['loc_y'].apply(lambda row: row - 2*(row-middle_y) if row > 250.0 else row + 2

In [17]: df_away_mirror['loc_x'] = df_away_mirror['loc_x'].apply(lambda row: row - 2*(row-middle_x))
         df_away_mirror['loc_y'] = df_away_mirror['loc_y'].apply(lambda row: row - 2*(row-middle_y) if row > 250.0 else row + 2

In [18]: df_home_mirror.shape

Out[18]: (50914, 13)

In [19]: df_home = df_home_notmirror.append(df_home_mirror)
         df_away = df_away_notmirror.append(df_away_mirror)

In [20]: df_home.shape, df_away.shape

Out[20]: ((104482, 13), (104694, 13))
```

Figure 4. Home and Away dataframes preprocess

Finally, we joined both home and away subsets of the original dataframe to form the new complete dataframe with the mirrored x and y shot locations on the same quadrant. Thereby, they could be plotted in the half-court matplotlib plot.

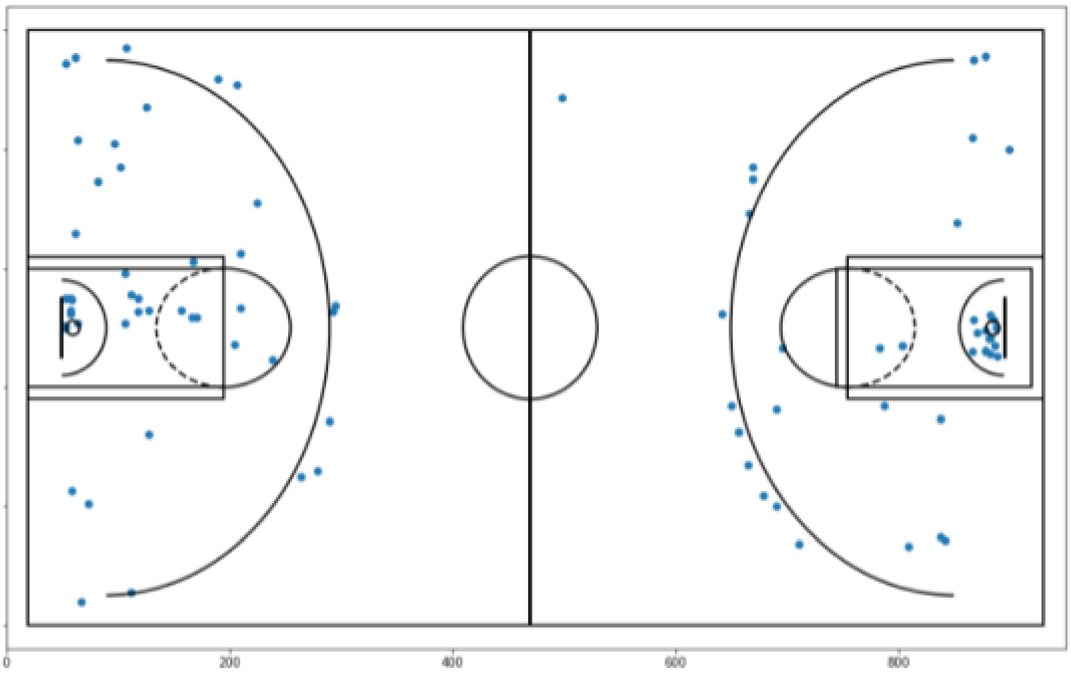


Figure 5. Full Court Plot of dataset with original X and Y shot locations

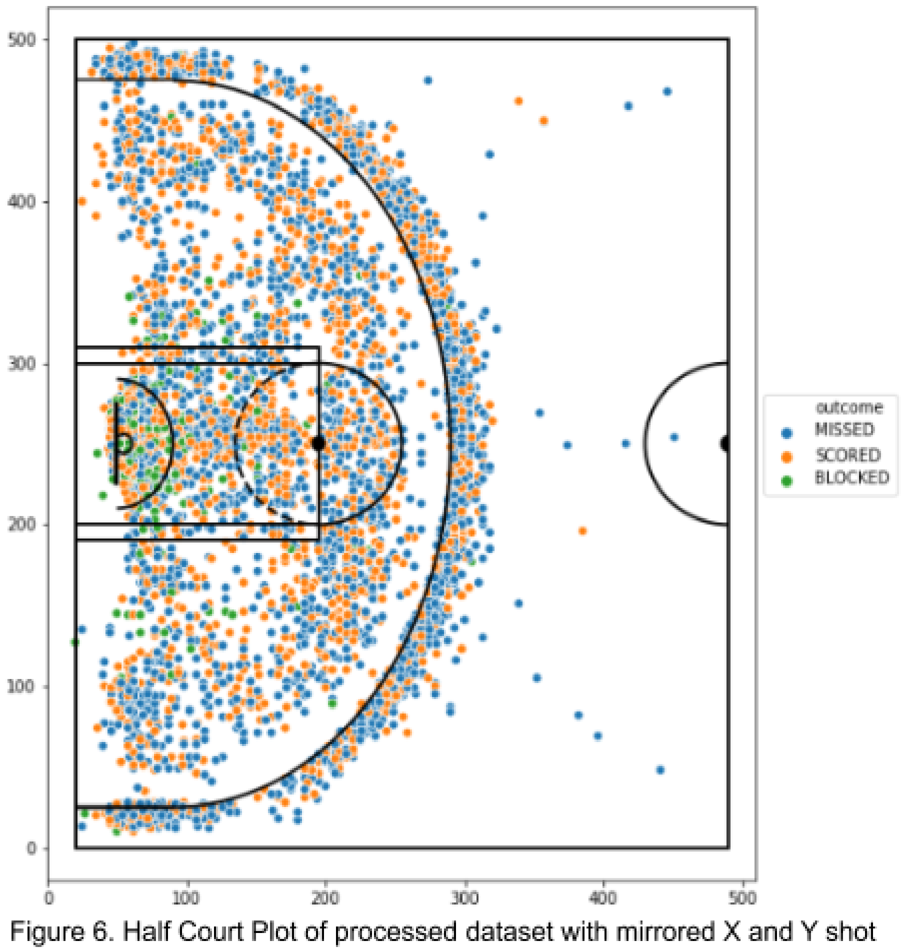


Figure 6. Half Court Plot of processed dataset with mirrored X and Y shot locations

## 3. Technologies Used

Obviously, there are a lot of options when trying to tackle a big data project. We decided to cover several of them in order to compare them with the same data set. The main technologies used for the development of this project were:

- **Github** - We used a **Github** repository to manage the project and collaborate as a team. All codes have been written with **Python 3.7**.

- **Google Cloud Storage** - GCP is a RESTful online storage mostly suitable for enterprises mostly comparable to Amazon S3.

- **Google BigQuery** - BigQuery is an enterprise data warehouse that enables super-fast SQL queries using the processing power of Google's infrastructure.

- **Apache Hadoop -** Hadoop is an open source distributed processing framework that manages data processing and storage for big data applications running in clustered systems.

- **Apache Hive** - Hive is a data warehouse software project built on top of Apache Hadoop for providing data query and analysis. Hive gives a SQL-like interface to query data stored in various databases and file systems that integrate with Hadoop.

- **Apache Spark** - Spark is an open-source distributed general-purpose cluster-computing framework. Spark provides an interface for programming entire clusters with implicit data parallelism and fault tolerance.

- **Apache Pig** - Apache Pig is a high-level platform for creating programs that run on Apache Hadoop. The language for this platform is called Pig Latin. Pig can execute its Hadoop jobs in MapReduce, Apache Tez, or Apache Spark.

- **Apache Zeppelin** - Zeppelin is a multi-purposed web-based notebook which brings data ingestion, data exploration, visualization, sharing and collaboration features to Hadoop and Spark.

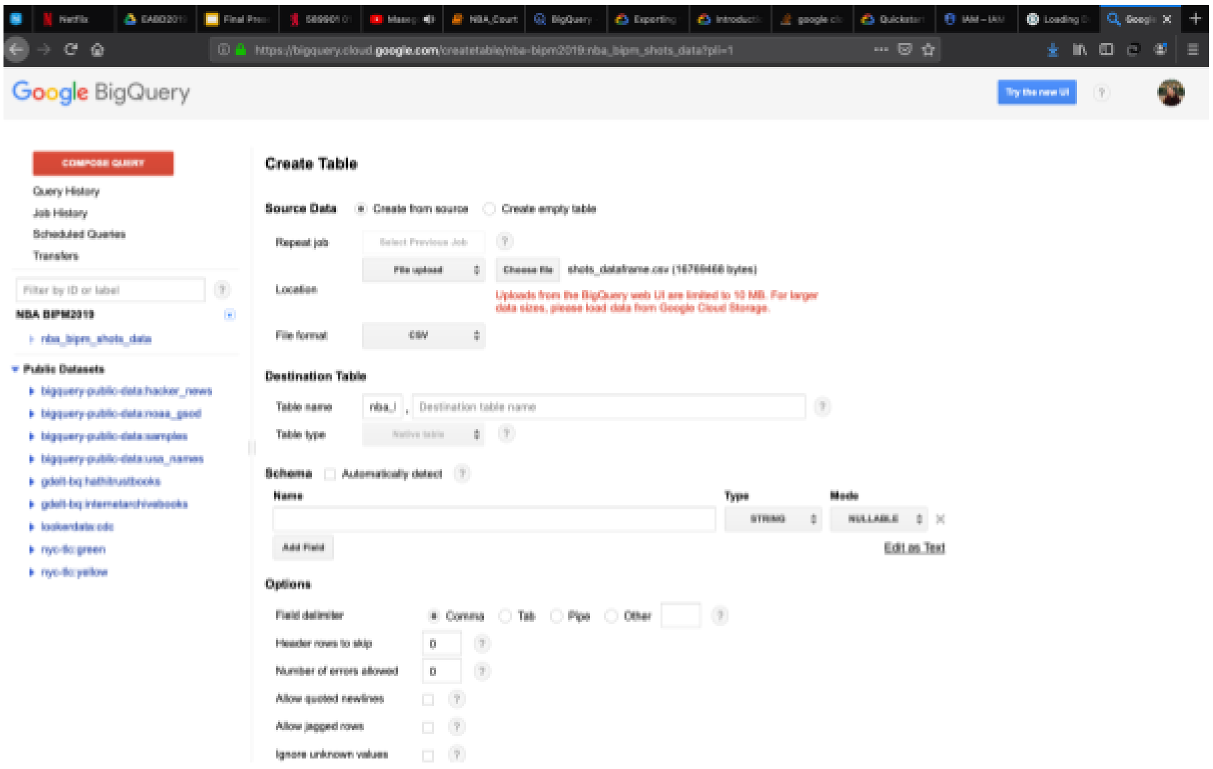
- **Data Visualization Tools -** The data visualization tools used were Tableau, Microsoft Power BI and the integrated capabilities of  Apache Zeppelin.

## 4. Use Cases:

### a. Scenario 1 (Google BigQuery + Tableau)

**GCP BigQuerry**

Open the BigQuery web UI. Click the blue arrow to the right of your project name and choose **Create new dataset**.


https://bigquery.cloud.google.com/createtable/nba-bipm2019:nba_bipm_shots_data?pli=1

Google BigQuery

COMPOSE QUERY
Query History
Job History
Scheduled Queries
Transfers

NBA BIPM2019
nba_bipm_shots_data

Public Datasets
bigquery-public-data:hacker_news
bigquery-public-data:noaa_gsod
bigquery-public-data:samples
bigquery-public-data:usa_names
gdelt-bq:hathitrustbooks
gdelt-bq:internetarchivebooks
lookerdata:cdc
nyc-tlc:green
nyc-tlc:yellow

**Create Table**

**Source Data** [✓ Create from source] Create empty table

Location: File upload — shots_dataframe.csv (16769468 bytes)

Uploads from the BigQuery web UI are limited to 10 MB. For larger data sizes, please load data from Google Cloud Storage.

File format: CSV

**Destination Table**

Table name: nba_l . Destination table name

Table type: Native table

**Schema** Automatically detect

| Name | Type | Mode |
|---|---|---|
| | STRING | NULLABLE |

**Options**

Field delimiter: [✓ Comma] Tab Pipe Other

Header rows to skip: 0

Number of errors allowed: 0

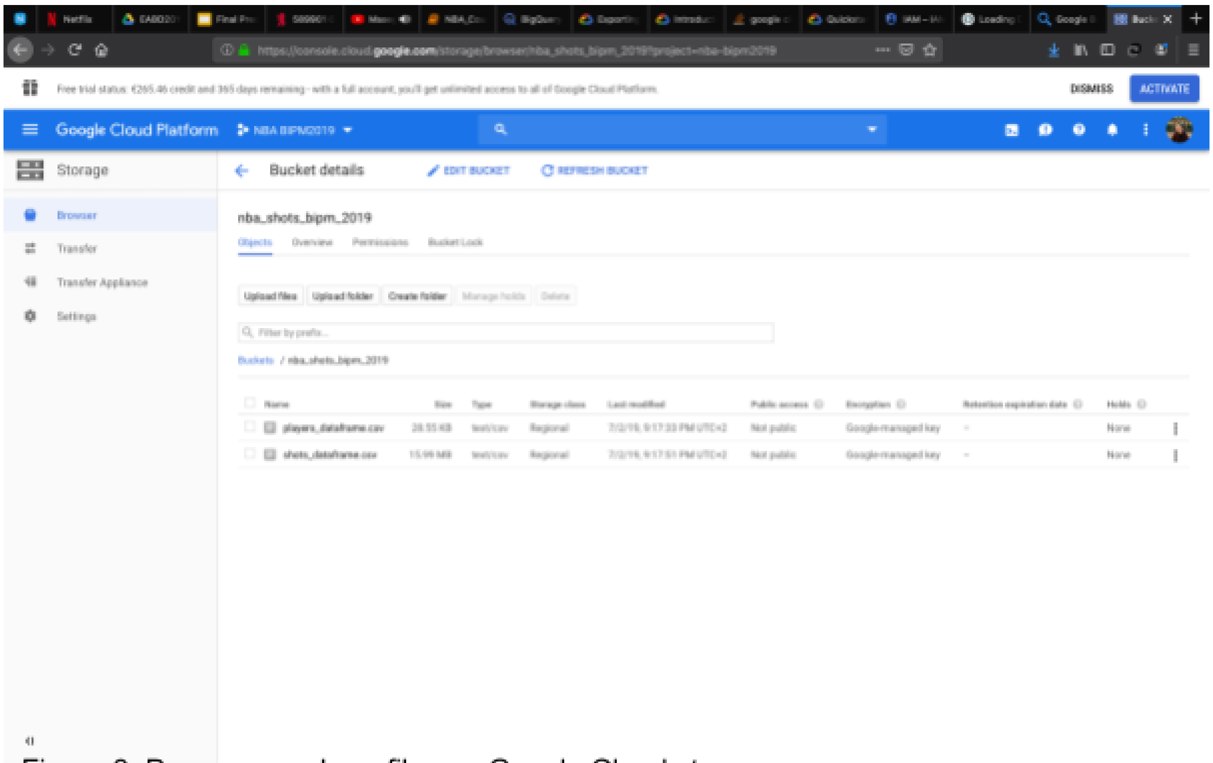
Allow quoted newlines: ☐

Allow jagged rows: ☐

Ignore unknown values: ☐

Figure 7. Loading .csv files

Since the data file was too large to load from a local machine, we added our preprocessed csv files on google cloud storage.

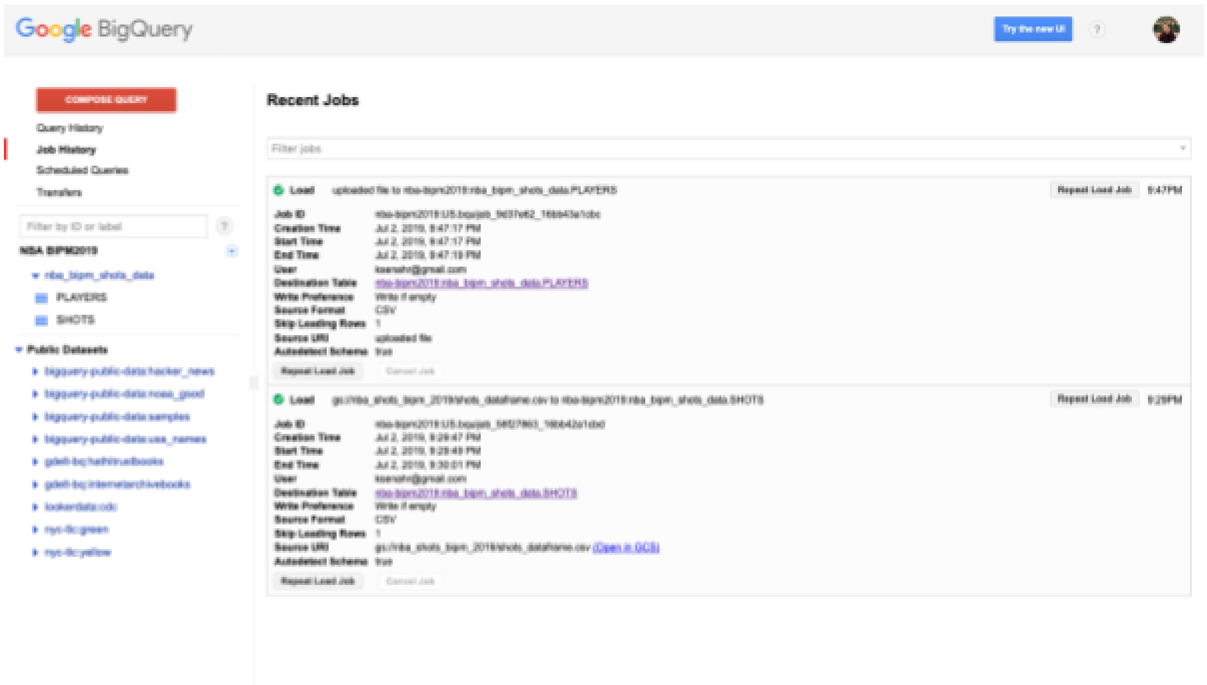


Buckets / nba_shots_bipm_2019

| Name | Size | Type | Storage class | Last modified | Public access | Encryption | Retention expiration date | Holds |
|---|---|---|---|---|---|---|---|---|
| players_dataframe.csv | 28.55 KB | text/csv | Regional | 7/2/19, 9:17:33 PM UTC+2 | Not public | Google-managed key | - | None |
| shots_dataframe.csv | 15.99 MB | text/csv | Regional | 7/2/19, 9:17:51 PM UTC+2 | Not public | Google-managed key | - | None |

Figure 8. Preprocessed csv files on Google Cloud storage

Now, we will upload a CSV file that was previously stored in gs to BigQuery using the BigQuery web UI. Since our csv file contains a header, we type **1** in the field **header rows to skip**. For **Field delimiter**, we should verify that **Comma** is selected. Now our table [nba-bipm2019:nba_bipm_shots_data.SHOTS](nba-bipm2019:nba_bipm_shots_data.SHOTS) is created. The same procedures have to be performed for the table [nba-bipm2019:nba_bipm_shots_data.PLAYERS](nba-bipm2019:nba_bipm_shots_data.PLAYERS). BigQuery creates a load job to create the table and upload data into the table. We can track job progress by clicking **Job History.**



Recent Jobs

Load uploaded file to nba-bipm2019:nba_bipm_shots_data.PLAYERS — 9:47PM

| | |
|---|---|
| Job ID | nba-bipm2019:US.bqjob_r5637e62_16bb43a1cbc |
| Creation Time | Jul 2, 2019, 9:47:17 PM |
| Start Time | Jul 2, 2019, 9:47:17 PM |
| End Time | Jul 2, 2019, 9:47:19 PM |
| User | ksenshr@gmail.com |
| Destination Table | nba-bipm2019:nba_bipm_shots_data.PLAYERS |
| Write Preference | Write if empty |
| Source Format | CSV |
| Skip Leading Rows | 1 |
| Source URI | uploaded file |
| Autodetect Schema | true |

Load gs://nba_shots_bipm_2019/shots_dataframe.csv to nba-bipm2019:nba_bipm_shots_data.SHOTS — 9:29PM

| | |
|---|---|
| Job ID | nba-bipm2019:US.bqjob_r5f27863_16bb42a1cbd |
| Creation Time | Jul 2, 2019, 9:29:47 PM |
| Start Time | Jul 2, 2019, 9:29:48 PM |
| End Time | Jul 2, 2019, 9:30:01 PM |
| User | ksenshr@gmail.com |
| Destination Table | nba-bipm2019:nba_bipm_shots_data.SHOTS |
| Write Preference | Write if empty |
| Source Format | CSV |
| Skip Leading Rows | 1 |
| Source URI | gs://nba_shots_bipm_2019/shots_dataframe.csv (Open in GCS) |
| Autodetect Schema | true |

Figure 9. Job History

On the **Table Details** page, click **Details** to view the table properties and then click **Preview** to view the table data.

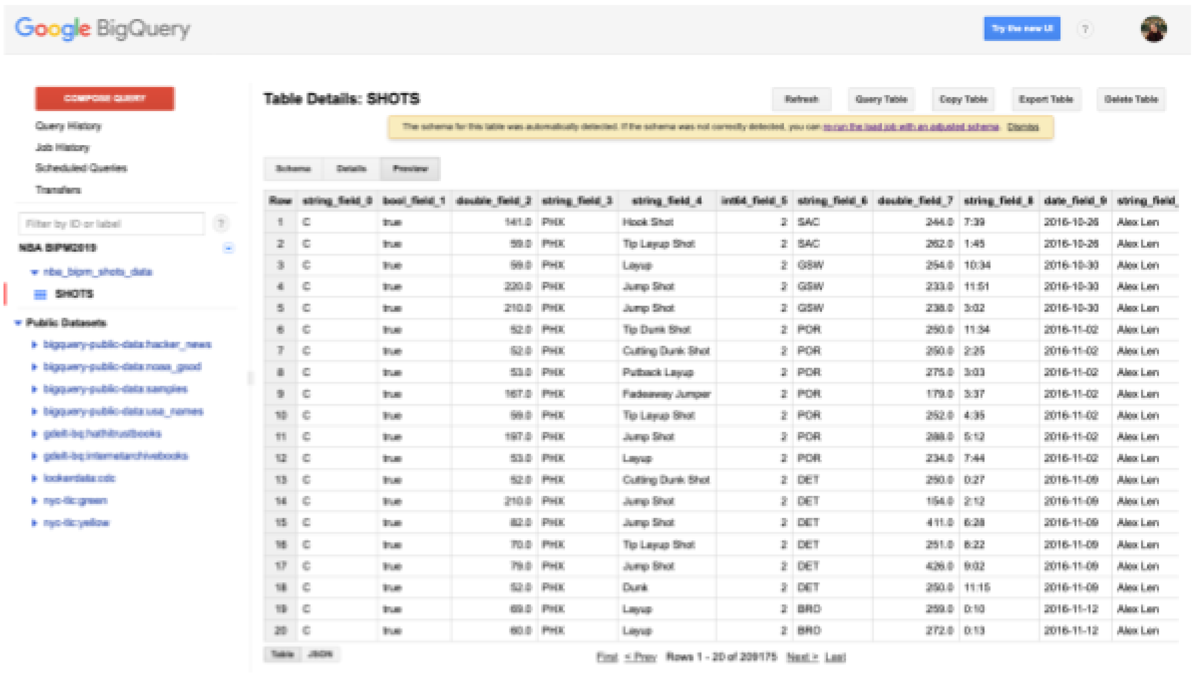


Table Details: SHOTS

| Row | string_field_0 | bool_field_1 | double_field_2 | string_field_3 | string_field_4 | int64_field_5 | string_field_6 | double_field_7 | string_field_8 | date_field_9 | string_field_ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | C | true | 141.0 | PHX | Hook Shot | 2 | SAC | 244.0 | 7:39 | 2016-10-26 | Alex Len |
| 2 | C | true | 59.0 | PHX | Tip Layup Shot | 2 | SAC | 262.0 | 1:45 | 2016-10-26 | Alex Len |
| 3 | C | true | 59.0 | PHX | Layup | 2 | GSW | 254.0 | 10:34 | 2016-10-30 | Alex Len |
| 4 | C | true | 220.0 | PHX | Jump Shot | 2 | GSW | 233.0 | 11:51 | 2016-10-30 | Alex Len |
| 5 | C | true | 210.0 | PHX | Jump Shot | 2 | GSW | 238.0 | 3:02 | 2016-10-30 | Alex Len |
| 6 | C | true | 52.0 | PHX | Tip Dunk Shot | 2 | POR | 250.0 | 11:34 | 2016-11-02 | Alex Len |
| 7 | C | true | 52.0 | PHX | Cutting Dunk Shot | 2 | POR | 250.0 | 2:25 | 2016-11-02 | Alex Len |
| 8 | C | true | 53.0 | PHX | Putback Layup | 2 | POR | 275.0 | 3:03 | 2016-11-02 | Alex Len |
| 9 | C | true | 167.0 | PHX | Fadeaway Jumper | 2 | POR | 179.0 | 3:37 | 2016-11-02 | Alex Len |
| 10 | C | true | 59.0 | PHX | Tip Layup Shot | 2 | POR | 252.0 | 4:35 | 2016-11-02 | Alex Len |
| 11 | C | true | 197.0 | PHX | Jump Shot | 2 | POR | 288.0 | 5:12 | 2016-11-02 | Alex Len |
| 12 | C | true | 53.0 | PHX | Layup | 2 | POR | 234.0 | 7:44 | 2016-11-02 | Alex Len |
| 13 | C | true | 52.0 | PHX | Cutting Dunk Shot | 2 | DET | 250.0 | 0:27 | 2016-11-09 | Alex Len |
| 14 | C | true | 210.0 | PHX | Jump Shot | 2 | DET | 164.0 | 2:12 | 2016-11-09 | Alex Len |
| 15 | C | true | 82.0 | PHX | Jump Shot | 2 | DET | 411.0 | 6:28 | 2016-11-09 | Alex Len |
| 16 | C | true | 70.0 | PHX | Tip Layup Shot | 2 | DET | 251.0 | 8:22 | 2016-11-09 | Alex Len |
| 17 | C | true | 79.0 | PHX | Jump Shot | 2 | DET | 426.0 | 9:02 | 2016-11-09 | Alex Len |
| 18 | C | true | 52.0 | PHX | Dunk | 2 | DET | 250.0 | 11:15 | 2016-11-09 | Alex Len |
| 19 | C | true | 69.0 | PHX | Layup | 2 | BRO | 259.0 | 0:10 | 2016-11-12 | Alex Len |
| 20 | C | true | 60.0 | PHX | Layup | 2 | BRO | 272.0 | 0:13 | 2016-11-12 | Alex Len |

First < Prev Rows 1 - 20 of 209175 Next > Last

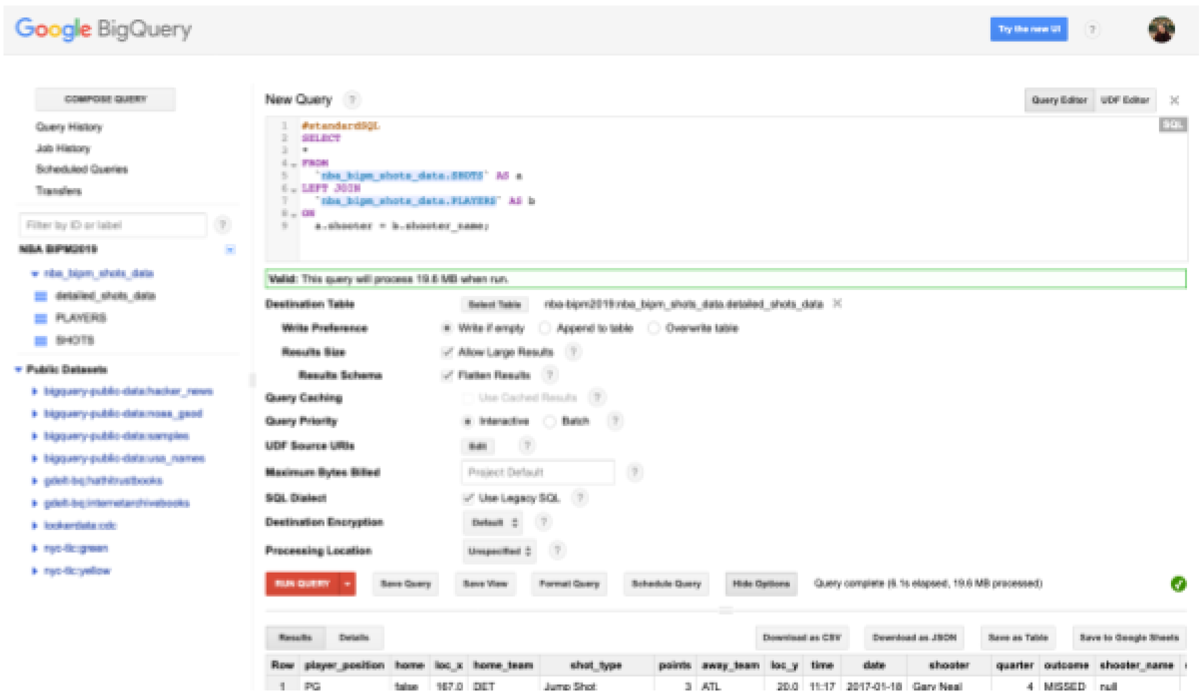
Figure 10. Table Details

In the next step, we will have to Transform and load data using web UI to transform and merge the data from the SHOTS and Players tables into a single denormalized table, which we upload to BigQuery.

For that we go to [BigQuery web UI]() and click on 'Compose Query'. There, we should type the following query in the New Query window:

```
#standardSQL
SELECT
*
FROM
  `nba_bipm_shots_data.SHOTS` AS a
LEFT JOIN
  `nba_bipm_shots_data.PLAYERS` AS b
ON
  a.shooter = b.shooter_name;
```

At the screenshot below you may see that a new table is added. This query will process 19.6 MB in 4.3 sec:



```
#standardSQL
SELECT
*
FROM
  `nba_bipm_shots_data.SHOTS` AS a
LEFT JOIN
  `nba_bipm_shots_data.PLAYERS` AS b
ON
  a.shooter = b.shooter_name;
```

**Valid:** This query will process 19.6 MB when run.

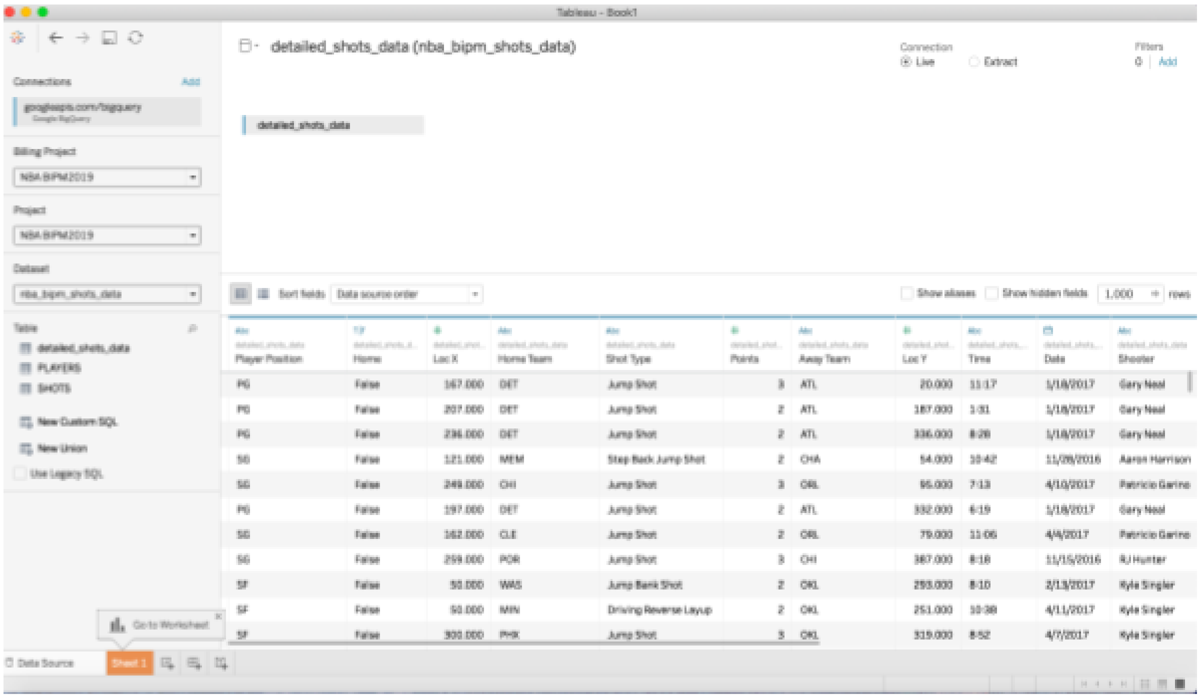
| | |
|---|---|
| Destination Table | Select Table nba-bipm2019:nba_bipm_shots_data.detailed_shots_data |
| Write Preference | [✓ Write if empty] Append to table Overwrite table |
| Results Size | ✓ Allow Large Results |
| Results Schema | ✓ Flatten Results |
| Query Caching | Use Cached Results |
| Query Priority | [✓ Interactive] Batch |
| UDF Source URIs | Edit |
| Maximum Bytes Billed | Project Default |
| SQL Dialect | ✓ Use Legacy SQL |
| Destination Encryption | Default |
| Processing Location | Unspecified |

Query complete (6.1s elapsed, 19.6 MB processed)

| Row | player_position | home | loc_x | home_team | shot_type | points | away_team | loc_y | time | date | shooter | quarter | outcome | shooter_name |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | PG | false | 167.0 | DET | Jump Shot | 3 | ATL | 20.0 | 11:17 | 2017-01-18 | Gary Neal | 4 | MISSED | null |

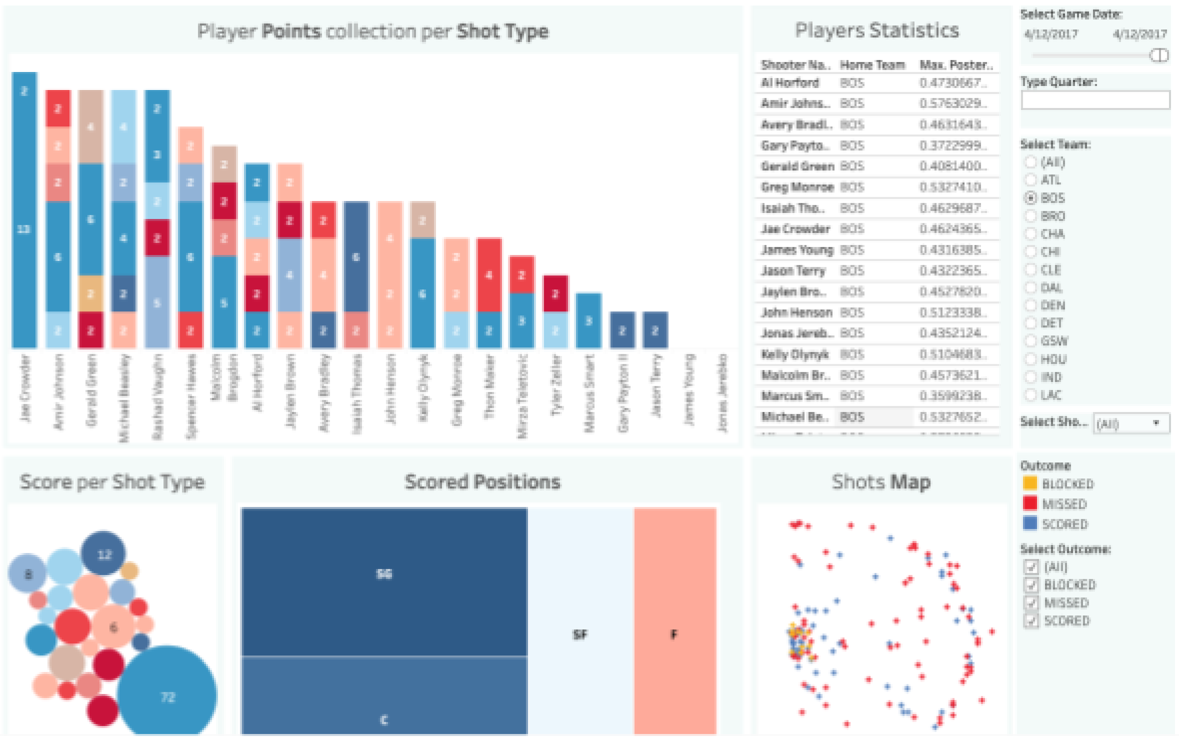
Figure 11. Added a new table

After the tables were successfully added, we can start connecting to BigQuery from Tableau and begin with visualizations:



| Player Position | Home | Loc X | Home Team | Shot Type | Points | Away Team | Loc Y | Time | Date | Shooter |
|---|---|---|---|---|---|---|---|---|---|---|
| PG | False | 167.000 | DET | Jump Shot | 3 | ATL | 20.000 | 11:17 | 1/18/2017 | Gary Neal |
| PG | False | 207.000 | DET | Jump Shot | 2 | ATL | 187.000 | 1:31 | 1/18/2017 | Gary Neal |
| PG | False | 236.000 | DET | Jump Shot | 2 | ATL | 336.000 | 8:28 | 1/18/2017 | Gary Neal |
| SG | False | 121.000 | MEM | Step Back Jump Shot | 2 | CHA | 54.000 | 10:42 | 11/28/2016 | Aaron Harrison |
| SG | False | 249.000 | CHI | Jump Shot | 3 | ORL | 95.000 | 7:13 | 4/10/2017 | Patricio Garino |
| PG | False | 197.000 | DET | Jump Shot | 2 | ATL | 332.000 | 6:19 | 1/18/2017 | Gary Neal |
| SG | False | 162.000 | CLE | Jump Shot | 2 | ORL | 79.000 | 11:06 | 4/4/2017 | Patricio Garino |
| SG | False | 259.000 | POR | Jump Shot | 3 | CHI | 387.000 | 8:18 | 11/15/2016 | RJ Hunter |
| SF | False | 50.000 | WAS | Jump Bank Shot | 2 | OKL | 293.000 | 8:10 | 2/13/2017 | Kyle Singler |
| SF | False | 50.000 | MIN | Driving Reverse Layup | 2 | OKL | 251.000 | 10:38 | 4/11/2017 | Kyle Singler |
| SF | False | 300.000 | PHX | Jump Shot | 3 | OKL | 319.000 | 8:52 | 4/7/2017 | Kyle Singler |

Figure 12. Connect BigQuery and Tableau



Players Statistics

| Shooter Na.. | Home Team | Max. Poster.. |
|---|---|---|
| Al Horford | BOS | 0.4730667.. |
| Amir Johns.. | BOS | 0.5763029.. |
| Avery Bradl.. | BOS | 0.4631643.. |
| Gary Payto.. | BOS | 0.3722999.. |
| Gerald Green | BOS | 0.4081400.. |
| Greg Monroe | BOS | 0.5327410.. |
| Isaiah Tho.. | BOS | 0.4629687.. |
| Jae Crowder | BOS | 0.4624365.. |
| James Young | BOS | 0.4316385.. |
| Jason Terry | BOS | 0.4322365.. |
| Jaylen Bro.. | BOS | 0.4527820.. |
| John Henson | BOS | 0.5123338.. |
| Jonas Jereb.. | BOS | 0.4352124.. |
| Kelly Olynyk | BOS | 0.5104683.. |
| Malcolm Br.. | BOS | 0.4573621.. |
| Marcus Sm.. | BOS | 0.3599238.. |
| Michael Be.. | BOS | 0.5327652.. |

Select Game Date: 4/12/2017 4/12/2017

Type Quarter:

Select Team: (All) ATL [✓ BOS] BRO CHA CHI CLE DAL DEN DET GSW HOU IND LAC

Select Sho...: (All)

Select Outcome: ✓ (All) ✓ BLOCKED ✓ MISSED ✓ SCORED

Figure 13. Overall View

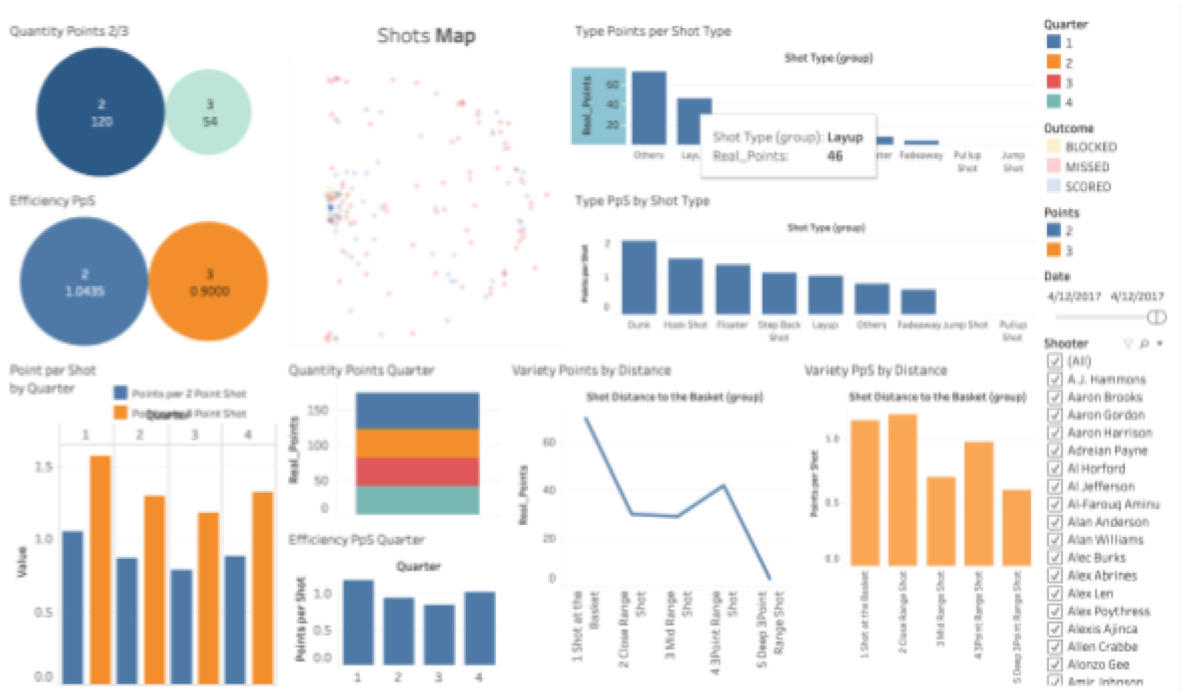
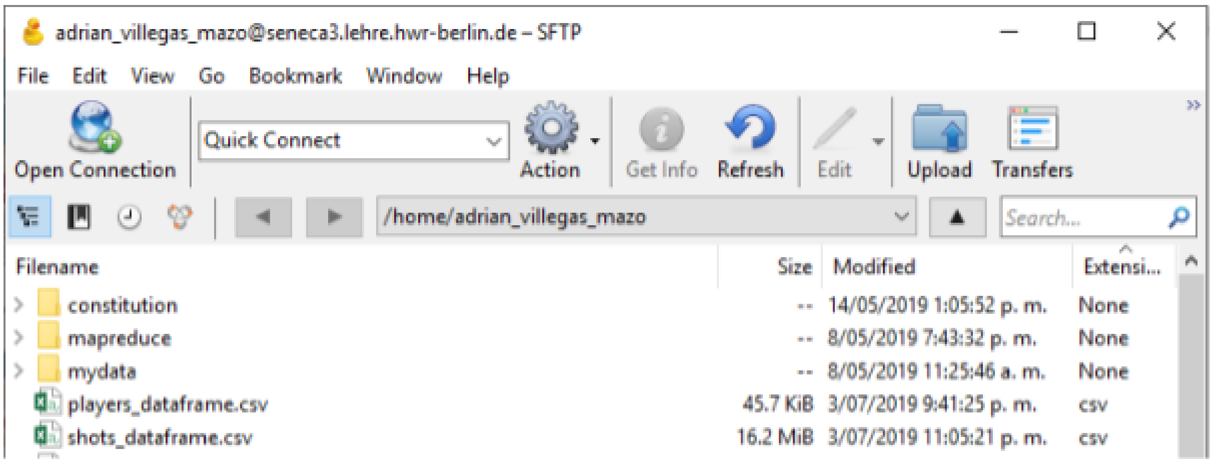


Figure 14. Player Production dashboard

## b. Scenario 2 (Apache Hive + Power BI)

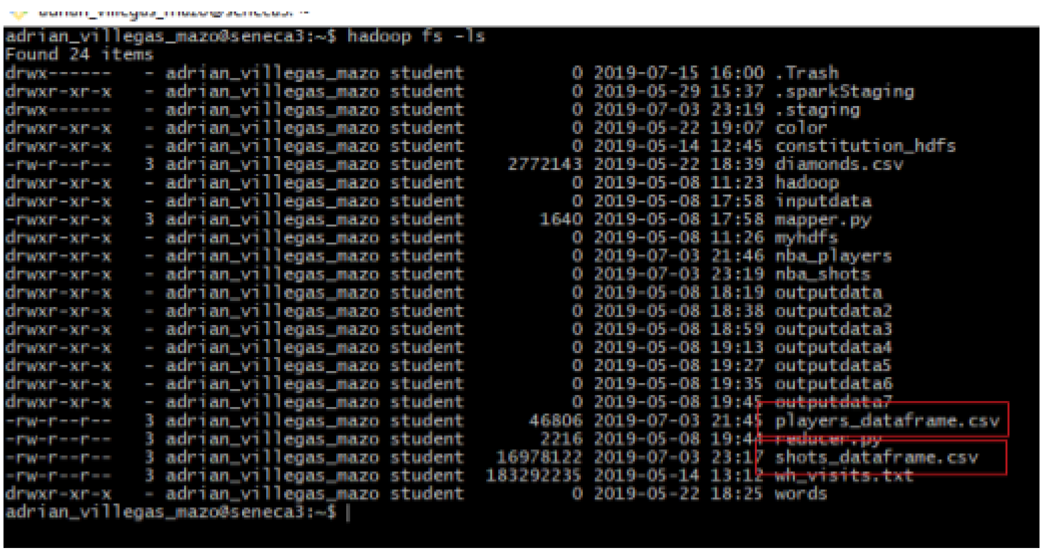
To use Hive as a datasource for Power BI, first we had to create the databases in the application. For this, we had to upload the .csv files into Seneca by using Cyberduck:


adrian_villegas_mazo@seneca3.lehre.hwr-berlin.de – SFTP
File Edit View Go Bookmark Window Help
Open Connection
Quick Connect
Action
Get Info
Refresh
Edit
Upload
Transfers
/home/adrian_villegas_mazo
Search...

| Filename | Size | Modified | Extensi... |
|---|---|---|---|
| constitution | -- | 14/05/2019 1:05:52 p. m. | None |
| mapreduce | -- | 8/05/2019 7:43:32 p. m. | None |
| mydata | -- | 8/05/2019 11:25:46 a. m. | None |
| players_dataframe.csv | 45.7 KiB | 3/07/2019 9:41:25 p. m. | csv |
| shots_dataframe.csv | 16.2 MiB | 3/07/2019 11:05:21 p. m. | csv |

Figure 15. File upload on Seneca

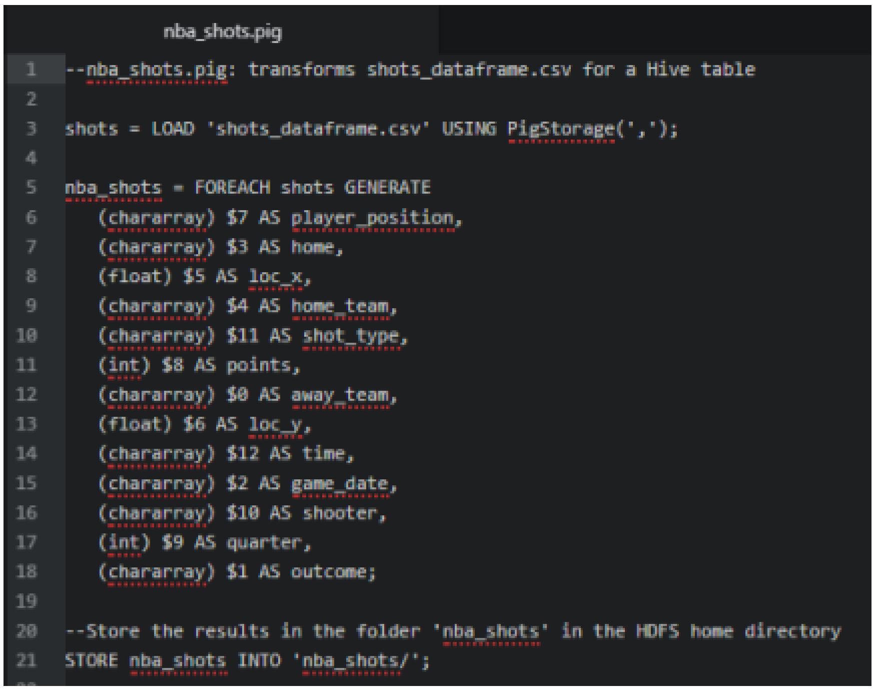
Then, by using command line, we upload the file into the Hadoop File System. The command used for this was “**hadoop fs -put ____.csv**” and after this we can check the HDFS to see if the upload was done correctly:



```
adrian_villegas_mazo@seneca3:~$ hadoop fs -ls
Found 24 items
drwx------   - adrian_villegas_mazo student          0 2019-07-15 16:00 .Trash
drwxr-xr-x   - adrian_villegas_mazo student          0 2019-05-29 15:37 .sparkStaging
drwx------   - adrian_villegas_mazo student          0 2019-07-03 23:19 .staging
drwxr-xr-x   - adrian_villegas_mazo student          0 2019-05-22 19:07 color
drwxr-xr-x   - adrian_villegas_mazo student          0 2019-05-14 12:45 constitution_hdfs
-rw-r--r--   3 adrian_villegas_mazo student    2772143 2019-05-22 18:39 diamonds.csv
drwxr-xr-x   - adrian_villegas_mazo student          0 2019-05-08 11:23 hadoop
drwxr-xr-x   - adrian_villegas_mazo student          0 2019-05-08 17:58 inputdata
-rwxr-xr-x   3 adrian_villegas_mazo student       1640 2019-05-08 17:58 mapper.py
drwxr-xr-x   - adrian_villegas_mazo student          0 2019-05-08 11:26 myhdfs
drwxr-xr-x   - adrian_villegas_mazo student          0 2019-07-03 21:46 nba_players
drwxr-xr-x   - adrian_villegas_mazo student          0 2019-07-03 23:19 nba_shots
drwxr-xr-x   - adrian_villegas_mazo student          0 2019-05-08 18:19 outputdata
drwxr-xr-x   - adrian_villegas_mazo student          0 2019-05-08 18:38 outputdata2
drwxr-xr-x   - adrian_villegas_mazo student          0 2019-05-08 18:59 outputdata3
drwxr-xr-x   - adrian_villegas_mazo student          0 2019-05-08 19:13 outputdata4
drwxr-xr-x   - adrian_villegas_mazo student          0 2019-05-08 19:27 outputdata5
drwxr-xr-x   - adrian_villegas_mazo student          0 2019-05-08 19:35 outputdata6
drwxr-xr-x   - adrian_villegas_mazo student          0 2019-05-08 19:45 outputdata7
-rw-r--r--   3 adrian_villegas_mazo student      46806 2019-07-03 21:45 players_dataframe.csv
-rw-r--r--   3 adrian_villegas_mazo student       2216 2019-05-08 19:44 reducer.py
-rw-r--r--   3 adrian_villegas_mazo student   16978122 2019-07-03 23:17 shots_dataframe.csv
-rw-r--r--   3 adrian_villegas_mazo student  183292235 2019-05-14 13:12 wh_visits.txt
drwxr-xr-x   - adrian_villegas_mazo student          0 2019-05-22 18:25 words
adrian_villegas_mazo@seneca3:~$ |
```

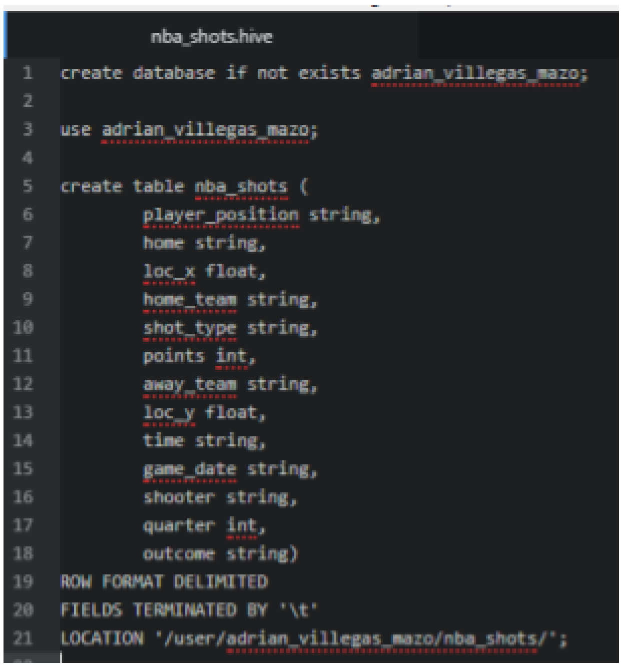
Figure 16. HDFS list of files

Then, by using the following .pig and .hive scripts, we uploaded the data into Hive and then created a HiveSQL table automatically:



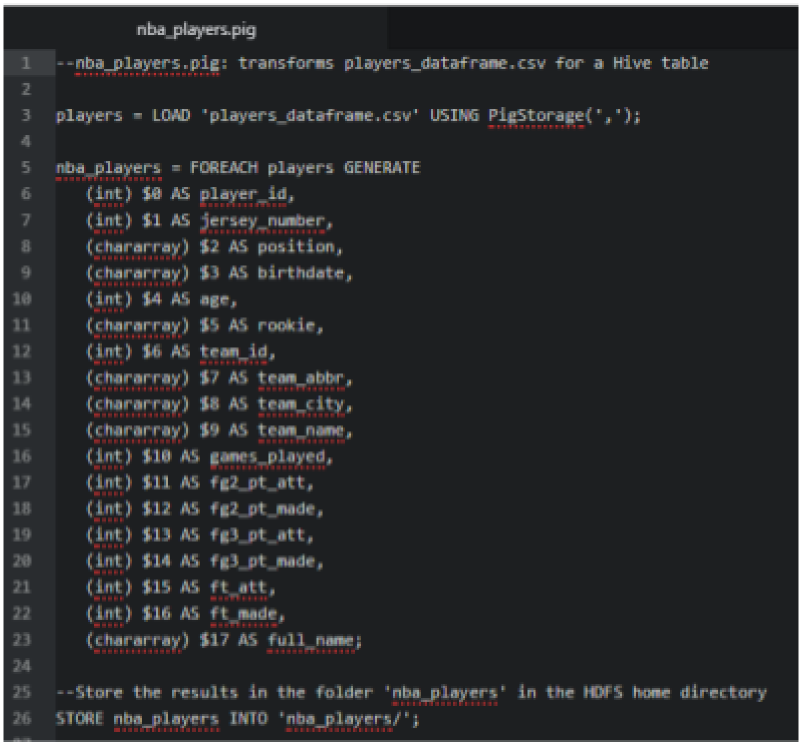
```
--nba_shots.pig: transforms shots_dataframe.csv for a Hive table

shots = LOAD 'shots_dataframe.csv' USING PigStorage(',');

nba_shots = FOREACH shots GENERATE
   (chararray) $7 AS player_position,
   (chararray) $3 AS home,
   (float) $5 AS loc_x,
   (chararray) $4 AS home_team,
   (chararray) $11 AS shot_type,
   (int) $8 AS points,
   (chararray) $0 AS away_team,
   (float) $6 AS loc_y,
   (chararray) $12 AS time,
   (chararray) $2 AS game_date,
   (chararray) $10 AS shooter,
   (int) $9 AS quarter,
   (chararray) $1 AS outcome;

--Store the results in the folder 'nba_shots' in the HDFS home directory
STORE nba_shots INTO 'nba_shots/';
```

Figure 18. NBA Shots pig scripts



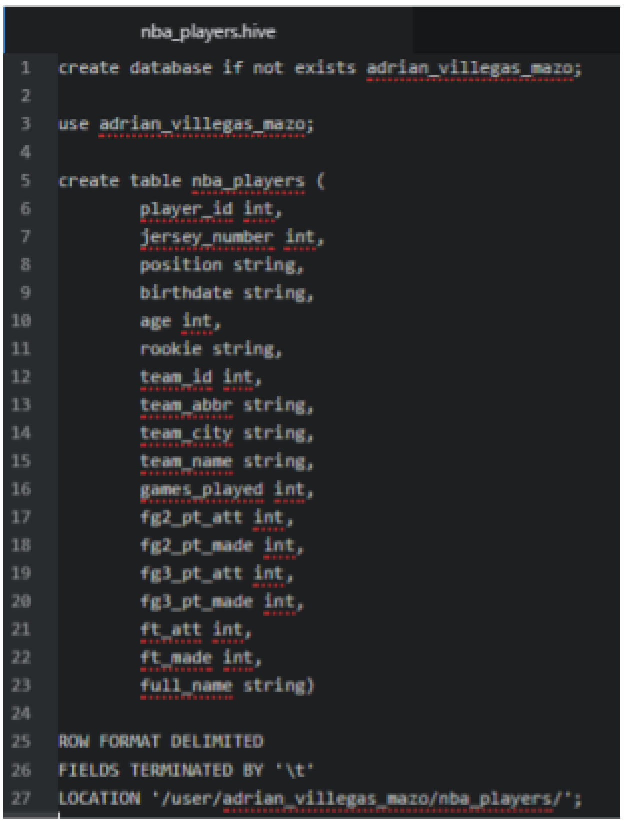
```
create database if not exists adrian_villegas_mazo;

use adrian_villegas_mazo;

create table nba_shots (
        player_position string,
        home string,
        loc_x float,
        home_team string,
        shot_type string,
        points int,
        away_team string,
        loc_y float,
        time string,
        game_date string,
        shooter string,
        quarter int,
        outcome string)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
LOCATION '/user/adrian_villegas_mazo/nba_shots/';
```

Figure 19. NBA Shots hive script



```
nba_players.pig
--nba_players.pig: transforms players_dataframe.csv for a Hive table

players = LOAD 'players_dataframe.csv' USING PigStorage(',');

nba_players = FOREACH players GENERATE
   (int) $0 AS player_id,
   (int) $1 AS jersey_number,
   (chararray) $2 AS position,
   (chararray) $3 AS birthdate,
   (int) $4 AS age,
   (chararray) $5 AS rookie,
   (int) $6 AS team_id,
   (chararray) $7 AS team_abbr,
   (chararray) $8 AS team_city,
   (chararray) $9 AS team_name,
   (int) $10 AS games_played,
   (int) $11 AS fg2_pt_att,
   (int) $12 AS fg2_pt_made,
   (int) $13 AS fg3_pt_att,
   (int) $14 AS fg3_pt_made,
   (int) $15 AS ft_att,
   (int) $16 AS ft_made,
   (chararray) $17 AS full_name;

--Store the results in the folder 'nba_players' in the HDFS home directory
STORE nba_players INTO 'nba_players/';
```

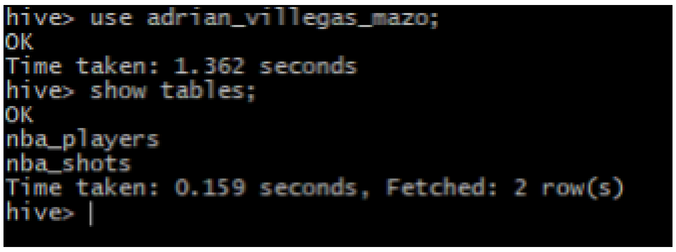
Figure 20. NBA Players pig script



```
nba_players.hive
create database if not exists adrian_villegas_mazo;

use adrian_villegas_mazo;

create table nba_players (
        player_id int,
        jersey_number int,
        position string,
        birthdate string,
        age int,
        rookie string,
        team_id int,
        team_abbr string,
        team_city string,
        team_name string,
        games_played int,
        fg2_pt_att int,
        fg2_pt_made int,
        fg3_pt_att int,
        fg3_pt_made int,
        ft_att int,
        ft_made int,
        full_name string)

ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
LOCATION '/user/adrian_villegas_mazo/nba_players/';
```

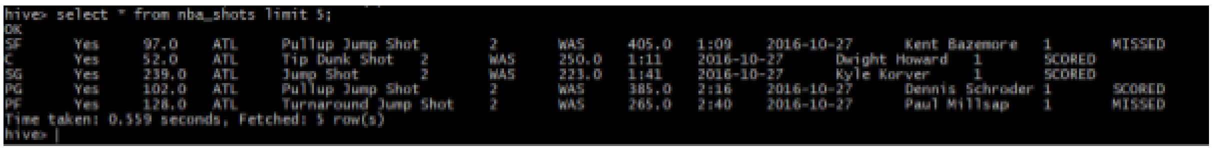
Figure 21. NBA Players hive script

After executing the scripts for both csv files, we can check in hive if the tables were created successfully and we can make a test query:



```
hive> use adrian_villegas_mazo;
OK
Time taken: 1.362 seconds
hive> show tables;
OK
nba_players
nba_shots
Time taken: 0.159 seconds, Fetched: 2 row(s)
hive> |
```

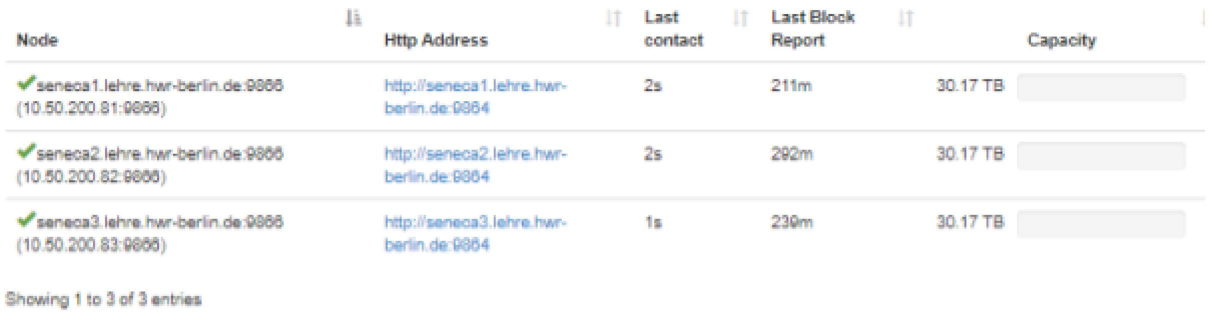
Figure 22. Hive created tables



```
hive> select * from nba_shots limit 5;
OK
SF      Yes     97.0    ATL     Pullup Jump Shot        2       WAS     405.0   1:09    2016-10-27      Kent Bazemore   1       MISSED
C       Yes     52.0    ATL     Tip Dunk Shot   2       WAS     250.0   1:11    2016-10-27      Dwight Howard   1       SCORED
SG      Yes     239.0   ATL     Jump Shot       2       WAS     223.0   1:41    2016-10-27      Kyle Korver     1       SCORED
PG      Yes     102.0   ATL     Pullup Jump Shot        2       WAS     385.0   2:16    2016-10-27      Dennis Schroder 1       SCORED
PF      Yes     128.0   ATL     Turnaround Jump Shot    2       WAS     265.0   2:40    2016-10-27      Paul Millsap    1       MISSED
Time taken: 0.559 seconds, Fetched: 5 row(s)
hive> |
```

Figure 23. Hive test query

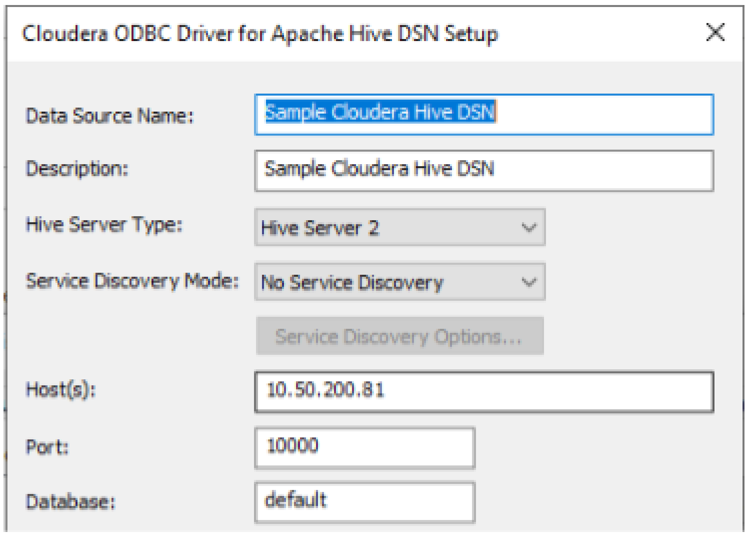
Now that we had the tables created in Hive, we had to figure out how to connect the Seneca cluster with a local client version of Microsoft Power BI. For this we had to download the Cloudera ODBC Driver for Apache Hive DSN Setup (x64) and configure it with the IP Address of the masternode in the cluster that we got from the Hadoop Web Interface:



| Node | Http Address | Last contact | Last Block Report | Capacity |
|---|---|---|---|---|
| ✓seneca1.lehre.hwr-berlin.de:9866 (10.50.200.81:9866) | http://seneca1.lehre.hwr-berlin.de:9864 | 2s | 211m | 30.17 TB |
| ✓seneca2.lehre.hwr-berlin.de:9866 (10.50.200.82:9866) | http://seneca2.lehre.hwr-berlin.de:9864 | 2s | 292m | 30.17 TB |
| ✓seneca3.lehre.hwr-berlin.de:9866 (10.50.200.83:9866) | http://seneca3.lehre.hwr-berlin.de:9864 | 1s | 239m | 30.17 TB |

Showing 1 to 3 of 3 entries

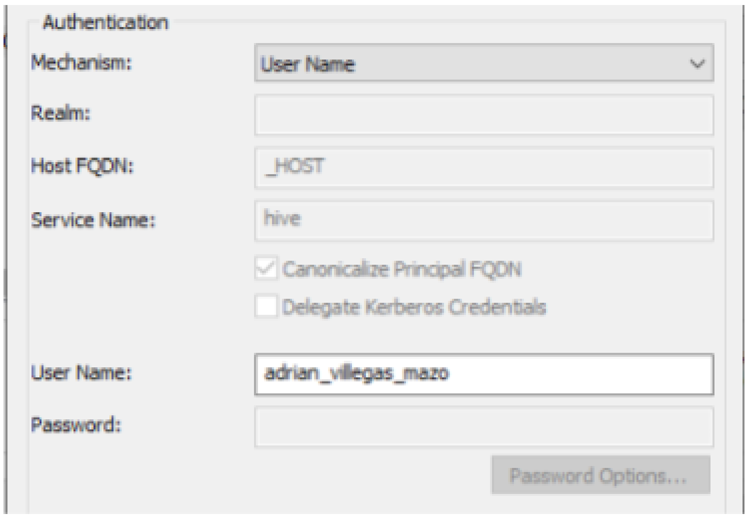
Figure 24. Master node and slave nodes information on Seneca cluster


Cloudera ODBC Driver for Apache Hive DSN Setup
Service Discovery Options...

| | |
|---|---|
| Data Source Name: | Sample Cloudera Hive DSN |
| Description: | Sample Cloudera Hive DSN |
| Hive Server Type: | Hive Server 2 |
| Service Discovery Mode: | No Service Discovery |
| Host(s): | 10.50.200.81 |
| Port: | 10000 |
| Database: | default |

Figure 25. ODBC Driver Configuration (I)

We also had to specify the default username that the driver uses to connect to the hive cluster:



| | |
|---|---|
| Mechanism: | User Name |
| Realm: | |
| Host FQDN: | _HOST |
| Service Name: | hive |
| | ✓ Canonicalize Principal FQDN |
| | Delegate Kerberos Credentials |
| User Name: | adrian_villegas_mazo |
| Password: | |

Figure 26. ODBC Driver Configuration (II)

After this is done, when we open Power BI and we click on the option "Get Data" and select the ODBC Driver, we can take the previously configured Apache Hive option:

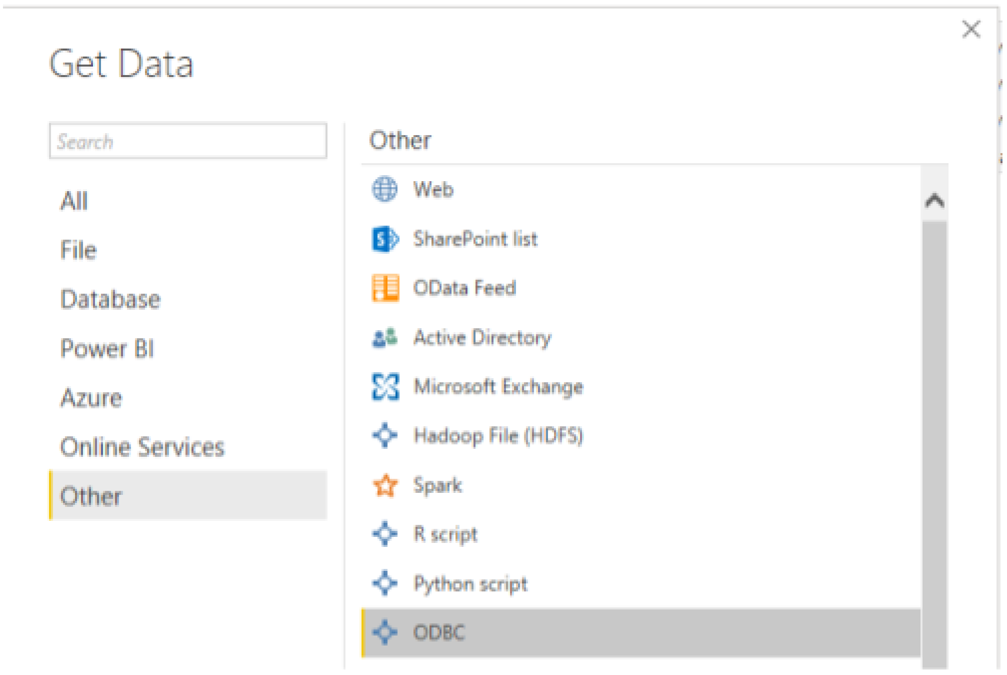
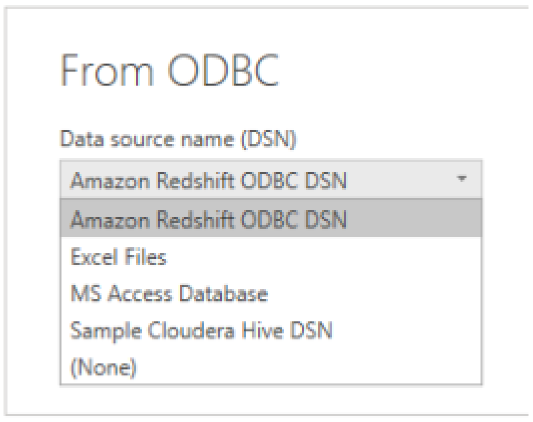


Figure 27. Power BI data source selection


From ODBC

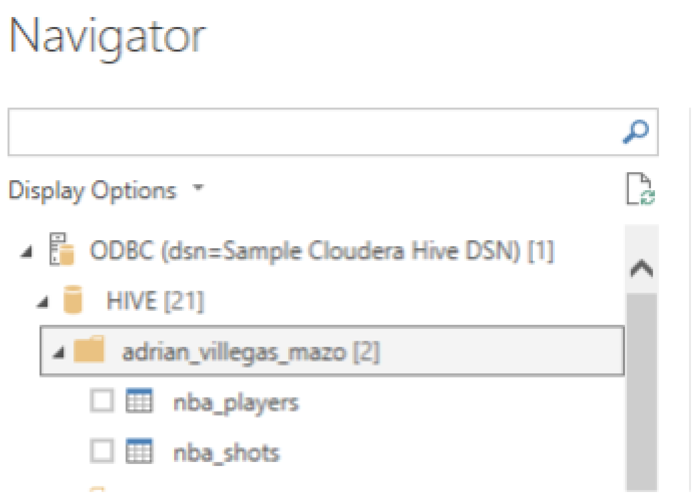
Data source name (DSN)

Amazon Redshift ODBC DSN

Amazon Redshift ODBC DSN
Excel Files
MS Access Database
Sample Cloudera Hive DSN
(None)

Figure 28. ODBC Driver Selection

After this we can see the two tables we created before and we can select them as the input into our Power BI model:

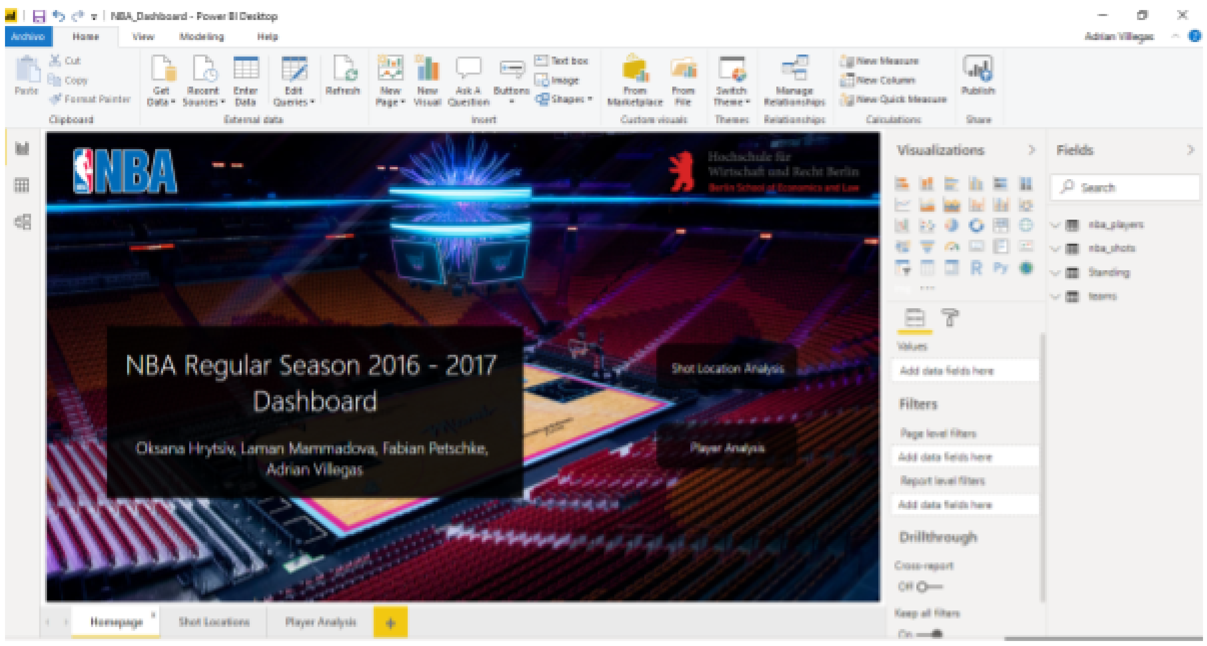

Navigator

Display Options

- ODBC (dsn=Sample Cloudera Hive DSN) [1]
  - HIVE [21]
    - adrian_villegas_mazo [2]
      - ☐ nba_players
      - ☐ nba_shots

Figure 29. Power BI Table Selection

From here on, we just got creative in order to make the dashboard, integrating both tables and adding some auxiliary tables for the visualizations:



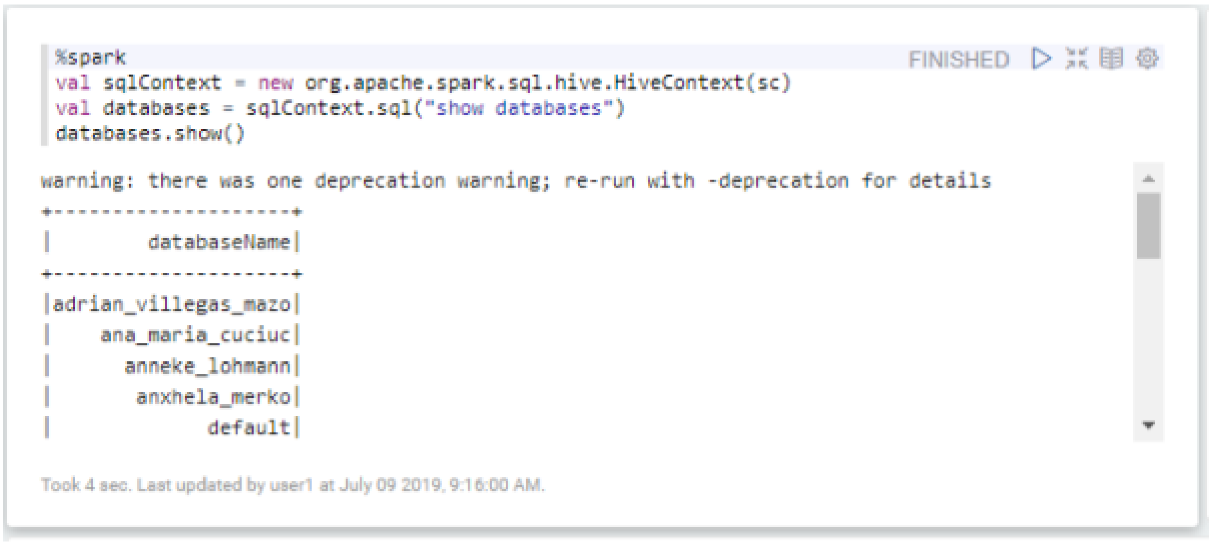
NBA Regular Season 2016 - 2017 Dashboard

Oksana Hrytsiv, Laman Mammadova, Fabian Petschke, Adrian Villegas

Figure 30. Power BI Dashboard Creation

### c. Scenario 3 (PySpark + Zeppelin)

In order to use Zeppelin as a visualization tool, first we need to connect to the Hive cluster located on Seneca. For this, we use spark to create a HiveContext that allows the Spark driver to access the cluster by using a resource manager, in this case YARN. Here we show our first test to see if the spark interpreter was able to connect to the Hive cluster by using the query “**show databases**”:

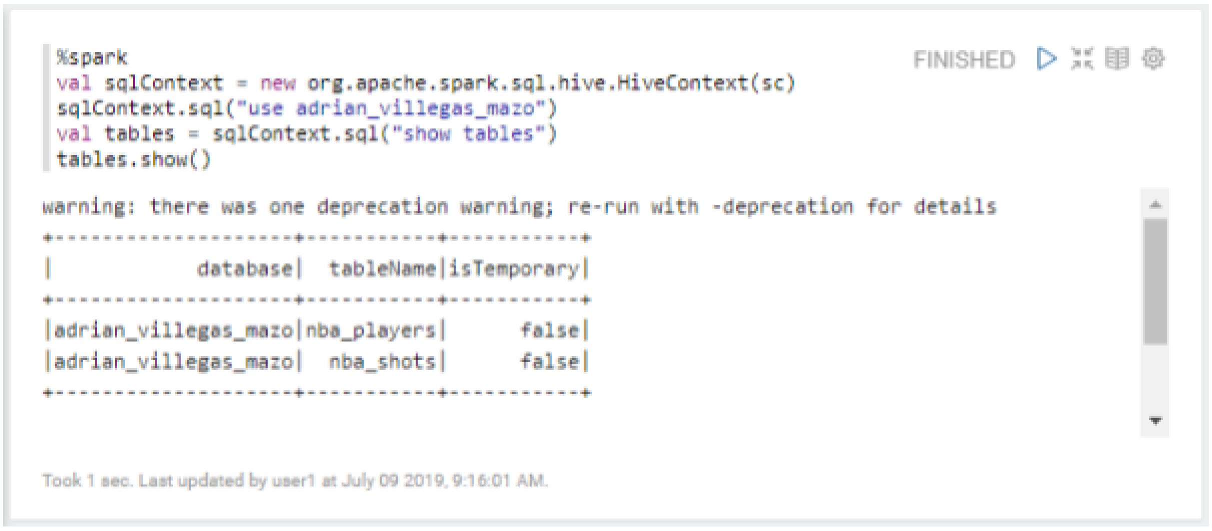


```
%spark
val sqlContext = new org.apache.spark.sql.hive.HiveContext(sc)
val databases = sqlContext.sql("show databases")
databases.show()

warning: there was one deprecation warning; re-run with -deprecation for details
+--------------------+
|        databaseName|
+--------------------+
|adrian_villegas_mazo|
|    ana_maria_cuciuc|
|      anneke_lohmann|
|       anxhela_merko|
|             default|
```

Took 4 sec. Last updated by user1 at July 09 2019, 9:16:00 AM.

Figure 31. Hive Context show database query

Then we test if we can pass two different queries, to select the database that has the tables and also to show them:

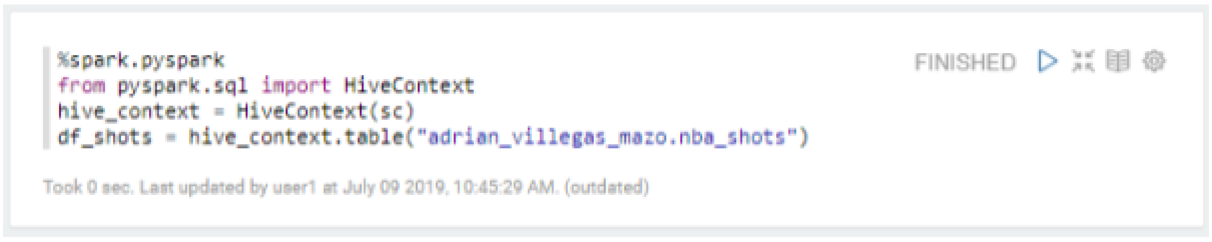


```
%spark
val sqlContext = new org.apache.spark.sql.hive.HiveContext(sc)
sqlContext.sql("use adrian_villegas_mazo")
val tables = sqlContext.sql("show tables")
tables.show()

warning: there was one deprecation warning; re-run with -deprecation for details
+--------------------+-----------+-----------+
|            database|  tableName|isTemporary|
+--------------------+-----------+-----------+
|adrian_villegas_mazo|nba_players|      false|
|adrian_villegas_mazo|  nba_shots|      false|
+--------------------+-----------+-----------+
```

Took 1 sec. Last updated by user1 at July 09 2019, 9:16:01 AM.

Figure 32. Hive Context show databases query

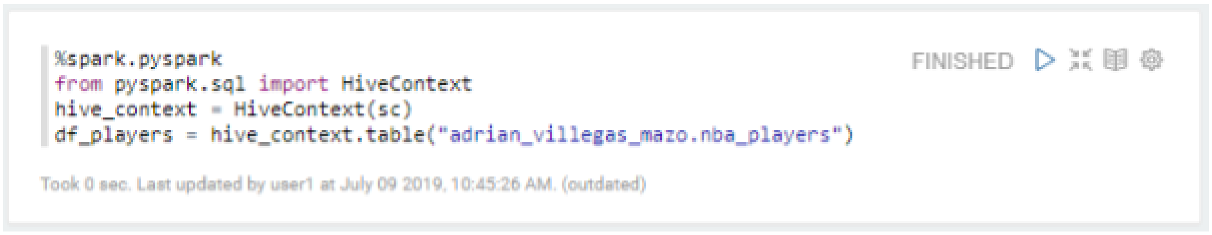
Knowing that the tables can be accessed, we use PySpark to create Pandas Dataframes in order to be able to plot them in following steps:



```
%spark.pyspark
from pyspark.sql import HiveContext
hive_context = HiveContext(sc)
df_shots = hive_context.table("adrian_villegas_mazo.nba_shots")
```

```
Took 0 sec. Last updated by user1 at July 09 2019, 10:45:29 AM. (outdated)
```

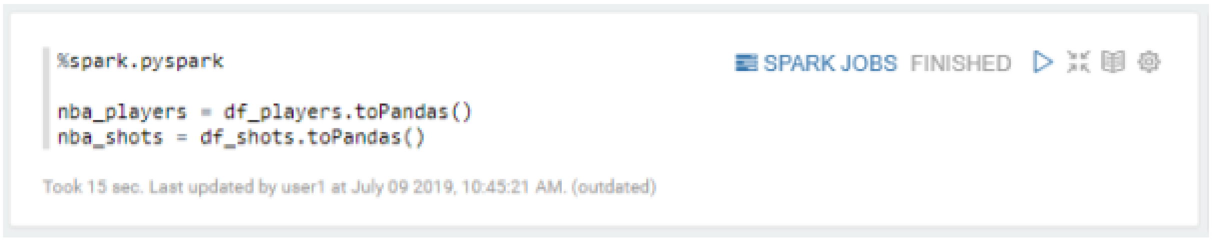
Figure 33. Shots dataframe creation



```
%spark.pyspark
from pyspark.sql import HiveContext
hive_context = HiveContext(sc)
df_players = hive_context.table("adrian_villegas_mazo.nba_players")
```

```
Took 0 sec. Last updated by user1 at July 09 2019, 10:45:26 AM. (outdated)
```

Figure 34. Players dataframe creation

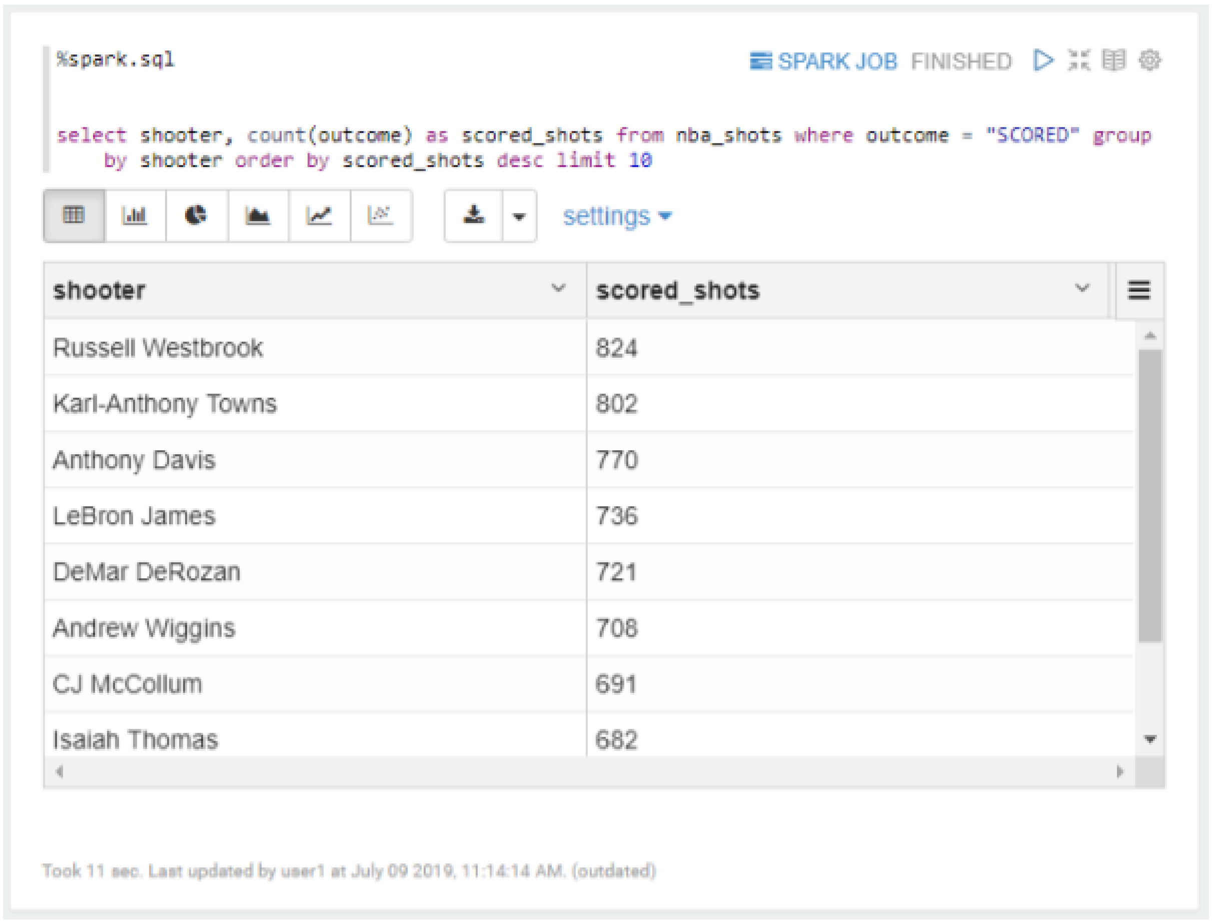


```
%spark.pyspark

nba_players = df_players.toPandas()
nba_shots = df_shots.toPandas()
```

```
Took 15 sec. Last updated by user1 at July 09 2019, 10:45:21 AM. (outdated)
```

Figure 35. Pandas dataframe creation

Using Apache Zeppelin, we can use the Spark.SQL interpreter to query the Pandas dataframes that we created from hive, using SQL queries like the following:



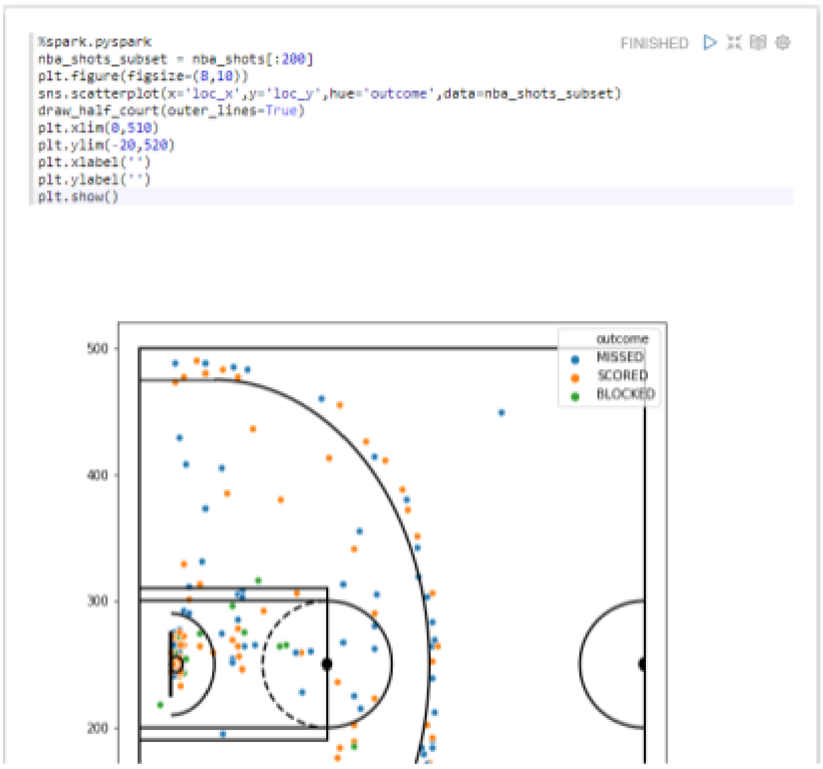
```
%spark.sql

select shooter, count(outcome) as scored_shots from nba_shots where outcome = "SCORED" group
    by shooter order by scored_shots desc limit 10
```

| shooter | scored_shots |
|---|---|
| Russell Westbrook | 824 |
| Karl-Anthony Towns | 802 |
| Anthony Davis | 770 |
| LeBron James | 736 |
| DeMar DeRozan | 721 |
| Andrew Wiggins | 708 |
| CJ McCollum | 691 |
| Isaiah Thomas | 682 |

Took 11 sec. Last updated by user1 at July 09 2019, 11:14:14 AM. (outdated)

Figure 36. SQL query on Spark object

We can also use the Spark.Python interpreter to plot the data using the common python plotting packages like matplotlib and seaborn.



```
%spark.pyspark
nba_shots_subset = nba_shots[:200]
plt.figure(figsize=(8,10))
sns.scatterplot(x='loc_x',y='loc_y',hue='outcome',data=nba_shots_subset)
draw_half_court(outer_lines=True)
plt.xlim(0,510)
plt.ylim(-20,520)
plt.xlabel('')
plt.ylabel('')
plt.show()
```

Figure 37. Pyspark half court plot

# 5. Technologies Evaluation

## 5.1 Storage technologies

| Google Cloud Storage | Apache Hive |
|---|---|
| **BigQuery** | **PySpark** |
| **Pros** | |
| Integration with Tableau | Python is slow as compared to Scala for Spark Jobs |
| On-demand pricing | Installation of ODBC drives to connect Power BI or Tableau |
| Free processing of small amount of data | |
| **Cons** | |
| BigQuery is an asset to analyze billions of rows | Installation of ODBC driver to connect PowerBI or Tableau |

| | |
|---|---|
| 3-5 seconds of response time | Limited available number of functions built-in into the package |
| Storing images in Google Storage to be processed by Google BigQuery is not possible | |

Table 1. Comparison of BigQuery and PySpark

### 5.2 Visualization Tools

| Tableau | Power BI | Zeppelin |
|---|---|---|
| **Pros** | | |
| Integration with BigQuery | Integration with Hive | Integration with Hive |
| User-friendly interface | User-friendly interface | User-friendly interface |
| | Creating visuals using Python | Creating visuals using Python |
| | | Huge amount of interpreters for different types of data and different types of querying/programming languages |
| **Cons** | | |
| Creating visuals using Python is not possible | Free version does not support publishing dashboards with Python visualizations to Web | Free version does not support publishing dashboards with Python visualizations to Web |
| Integration with Hive only through ODBC connector | | |

Table 2. Comparison of Tableau, Power BI and Zeppelin

## 6. Conclusion

In this project we used both cloud storage technologies and also Hadoop distributed file system with data stored in Hive. We have built 2 different data warehouses: one on the cloud using Google BigQuery and another on Hive using Apache Pig/Hive.
Coming from the business perspective, NBA Coaching Staffs, Managers, Owners and Analysts can use those technologies and the information gained from the visualizations to improve their approach to the game. Some applicable areas may be team planning (player

hirings, trades and cuts), team coaching (roster and lineup) and player development (focus in practice).
Traditionally, the eye test was used to evaluate players. Now, decision makers can improve their team performance and increase team success, leading to more wins, which in turn leads to more fan engagement and a higher profit, all by applying statistics and Big Data Technologies. The goal is to find undervalued players, as well as avoid overvalued players. Naturally, one wants to build a championship team at a reasonable price. Some expensive players may contribute less value than they are paid, whereas there are commonly unknown players that contribute high value at a relatively low price range.

## References

Shea,  S. M. (2014). *Basketball Analytics // Basketball analytics: Spatial Tracking // Spatial tracking*. S.l.]: CreateSpace Independent Publishing Platform; [s.n.]