

Project Framework

- 1. Introduction**
- 2. Data Analysis and Processing**
- 3. Big Data Technologies Used**
- 4. Use Cases**
 - a. Scenario 1 (Google BigQuery + Tableau)**
 - b. Scenario 2 (Apache Hive + Power BI)**
 - c. Scenario 3 (PySpark + Zeppelin)**
- 5. Technologies Evaluation**
- 6. Conclusion**

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.csv', '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 log GSW.csv', 'shot log NOP.csv', 'shot log LAL.csv', 'shot log OKL.csv', 'shot log HOU.csv', 'shot log CHI.csv', 'shot 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, encoding='utf-8')

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 score': 'current_score'})

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	Putup 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.s
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.

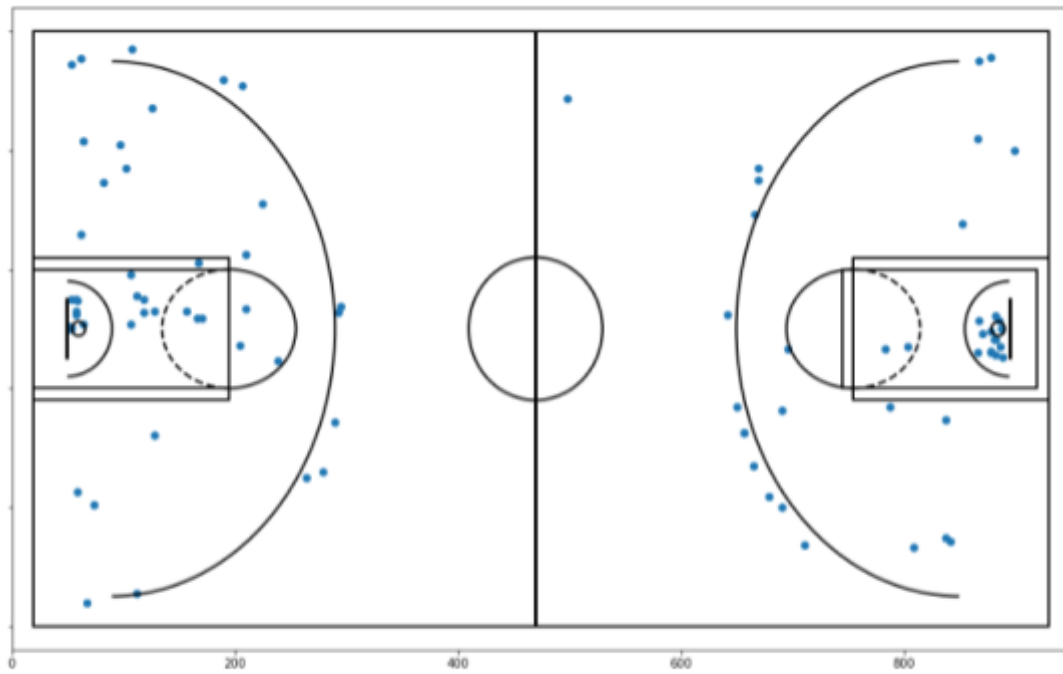


Figure 5. Full Court Plot of dataset with original X and Y shot locations

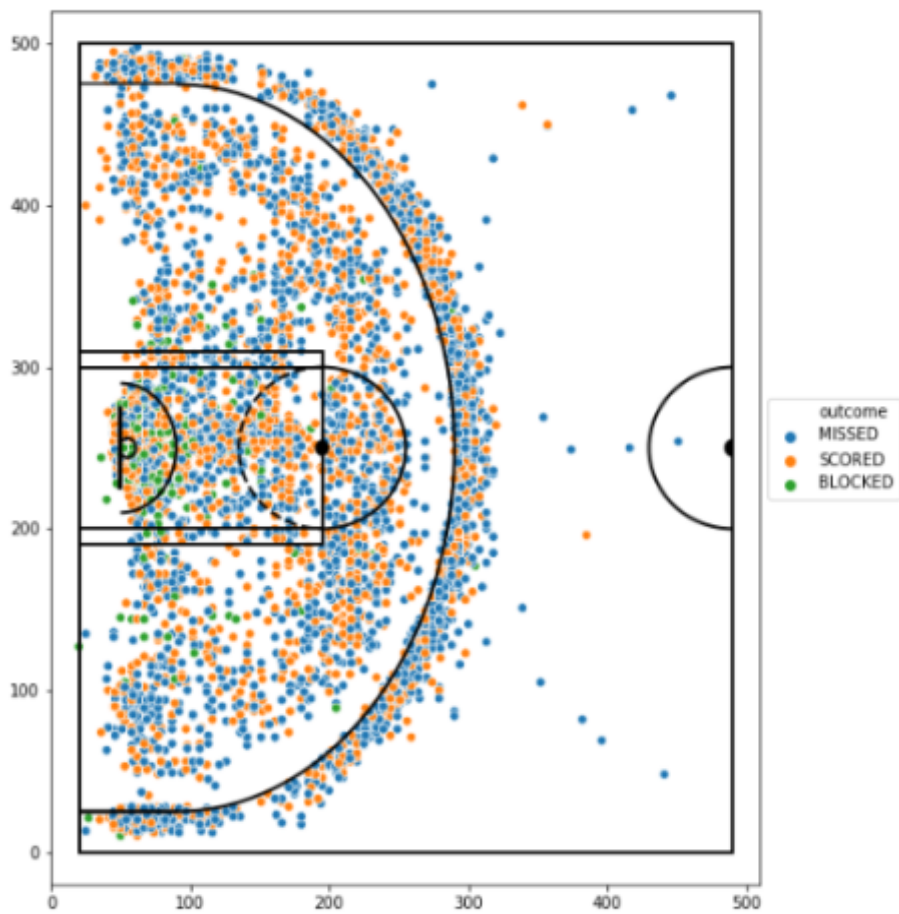


Figure 6. Half Court Plot of processed dataset with mirrored X and Y shot locations

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** - GCS 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 BigQuery

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

The screenshot shows the Google BigQuery 'Create Table' interface. The 'Source Data' section is set to 'Create from source' with 'File upload' selected. The file 'shots_dataframe.csv' (16766466 bytes) is chosen. The 'Destination Table' section shows the table name 'nba_1' and type 'Native table'. The 'Schema' section is set to 'Automatically detect'. The 'Options' section shows 'Field delimiter' as 'Comma', 'Header rows to skip' as '0', and 'Number of errors allowed' as '0'.

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.

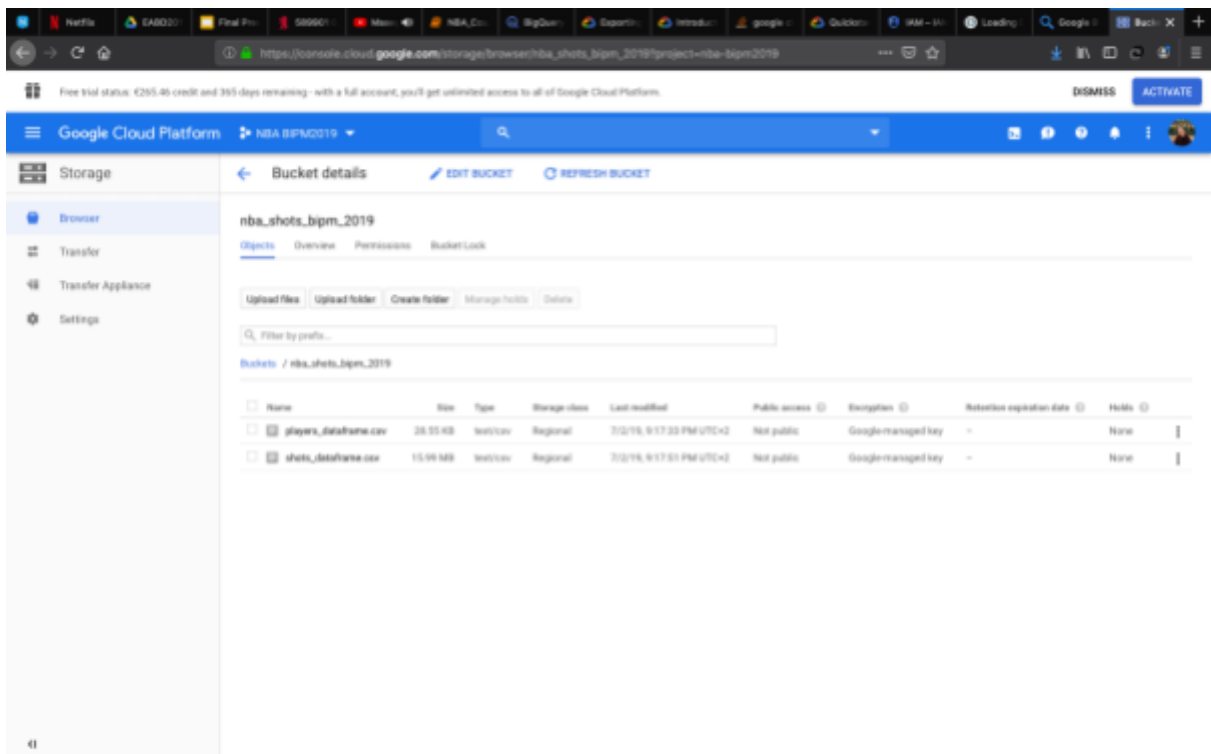


Figure 8. Preprocessed csv files on Google Cloud storage

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](#) is created. The same procedures have to be performed for the table [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**.

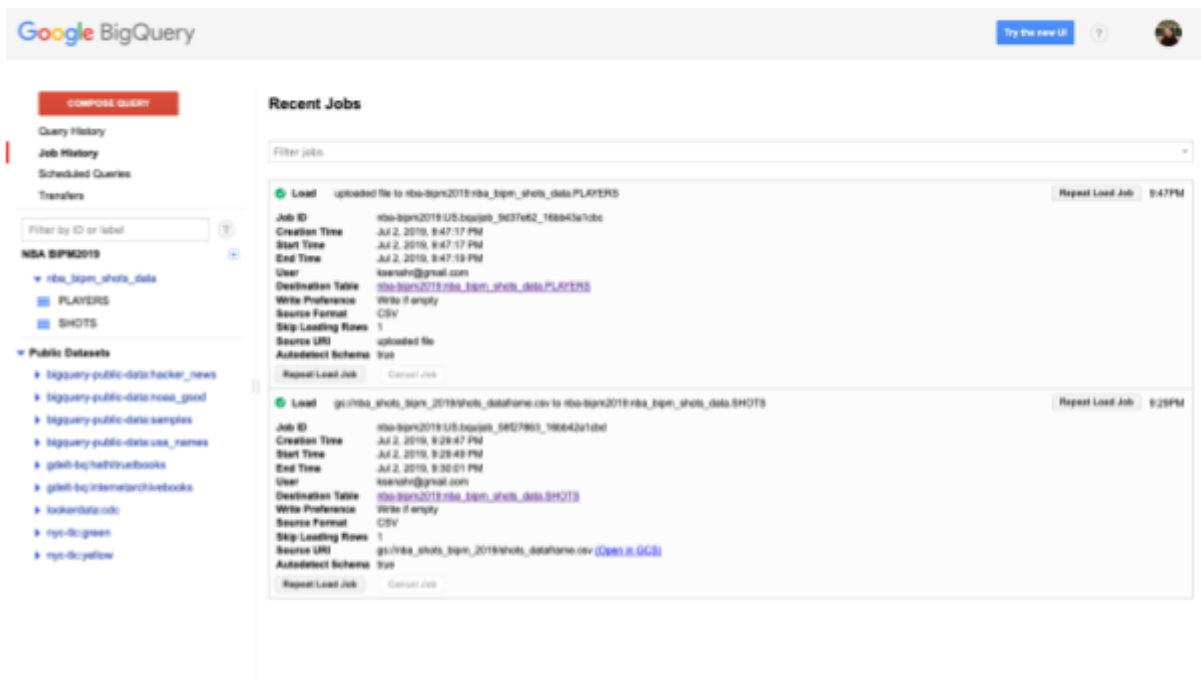


Figure 9. Job History

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

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_10
1	C	true	141.0	PHX	Hook Shot	2	SAC	244.0	7:39	2016-10-26	Alex Len
2	C	true	99.0	PHX	Tip Layup Shot	2	SAC	262.0	1:45	2016-10-26	Alex Len
3	C	true	99.0	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	99.0	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	154.0	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	BRD	259.0	0:10	2016-11-12	Alex Len
20	C	true	60.0	PHX	Layup	2	BRD	272.0	0:13	2016-11-12	Alex Len

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:

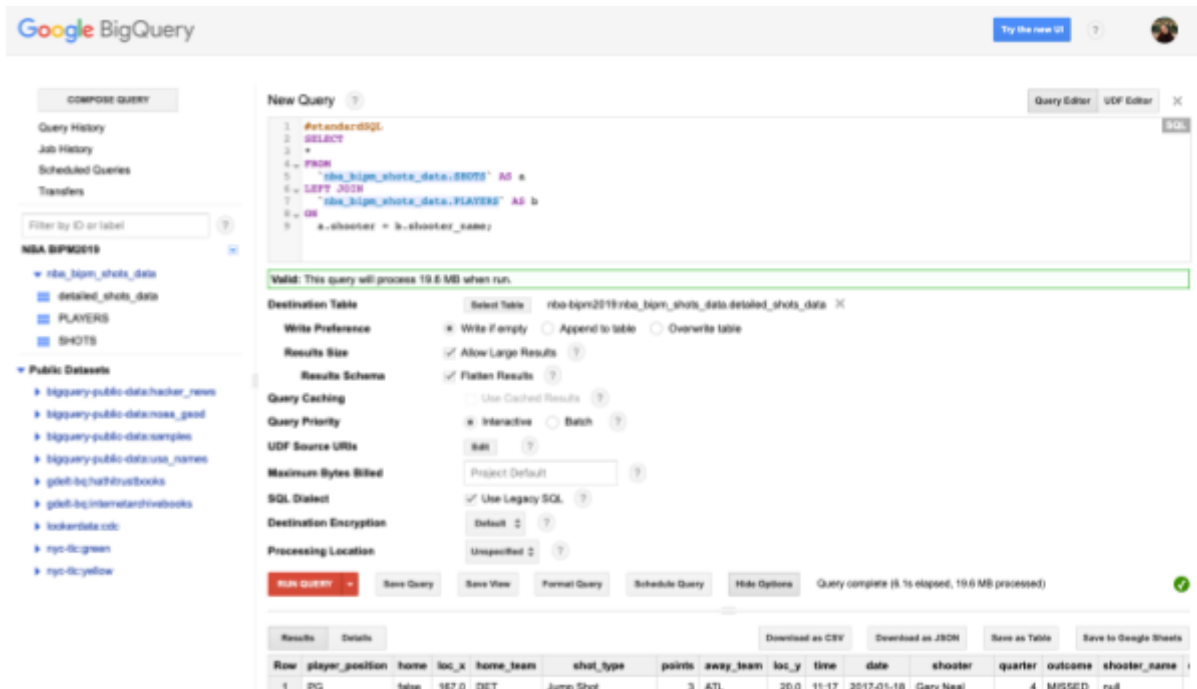


Figure 11. Added a new table

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

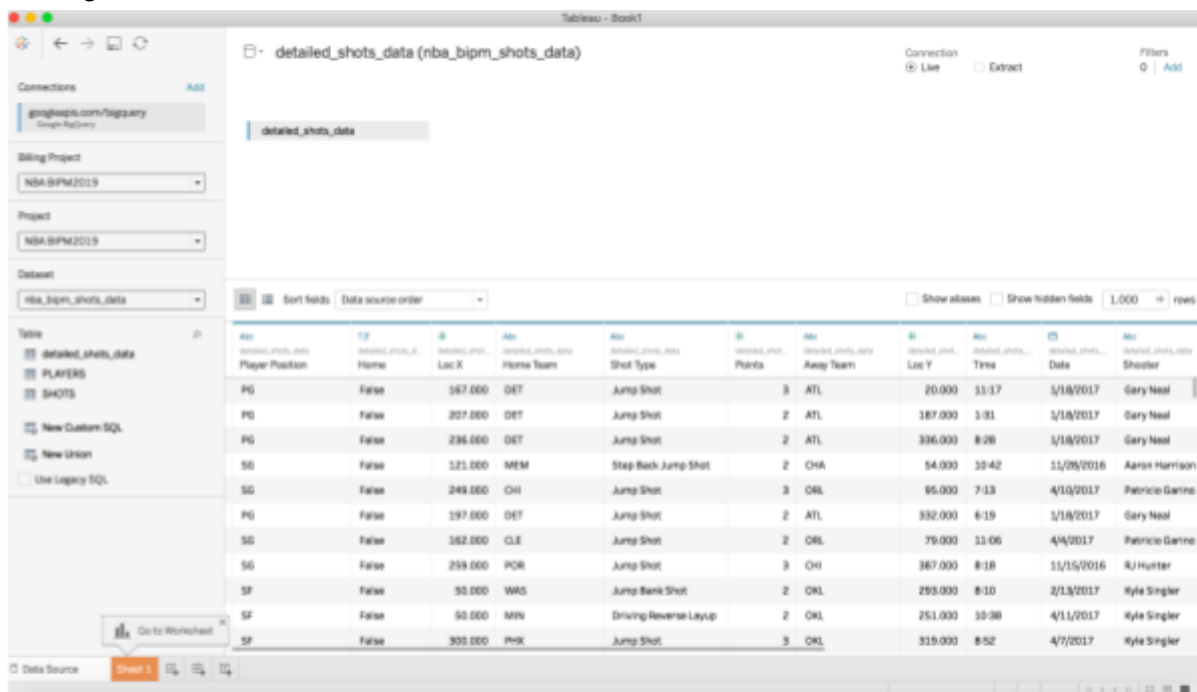


Figure 12. Connect BigQuery and Tableau

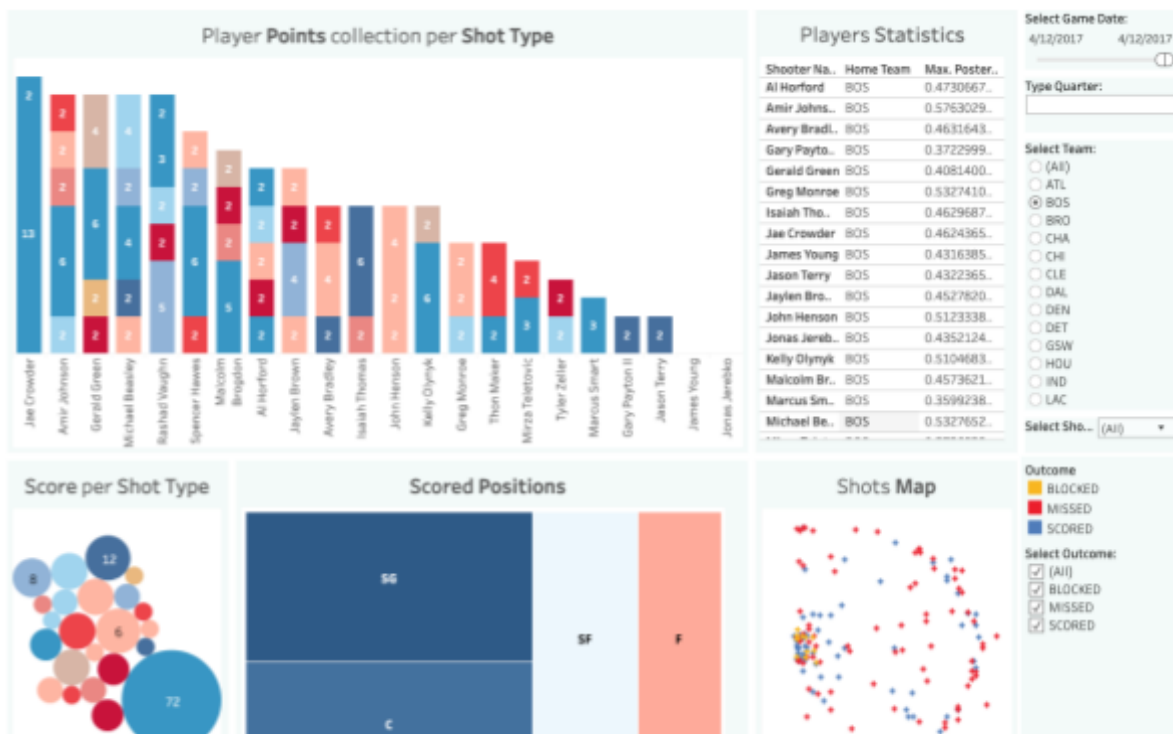


Figure 13. Overall View

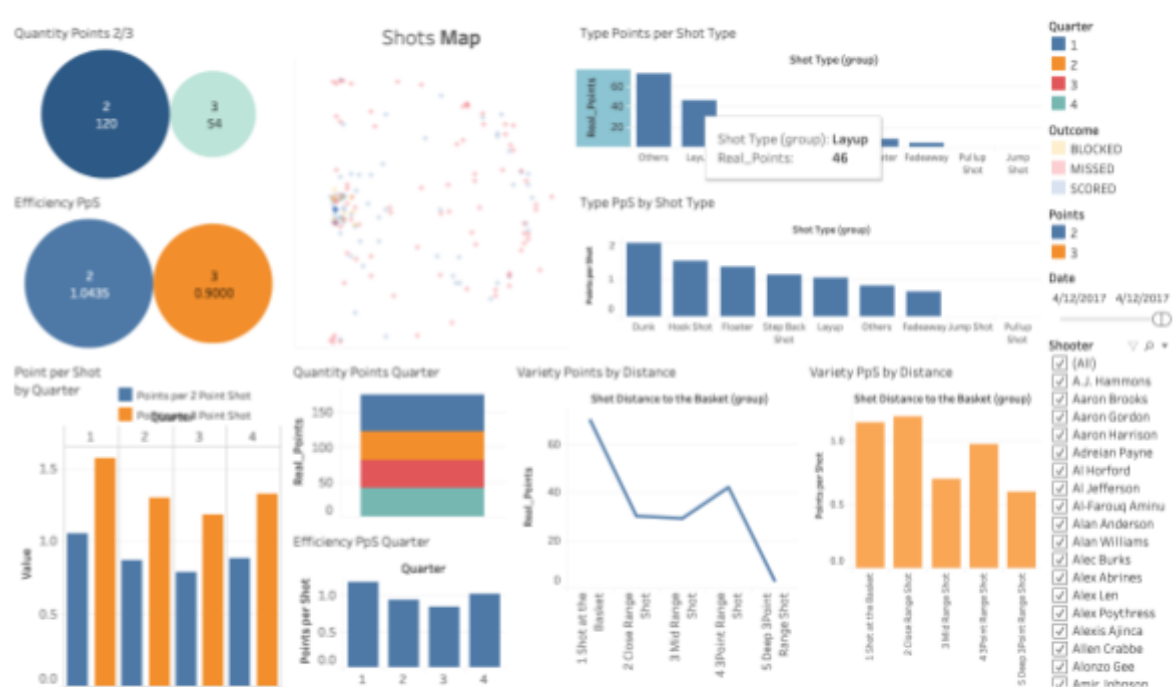


Figure 14. Player Production dashboard

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:

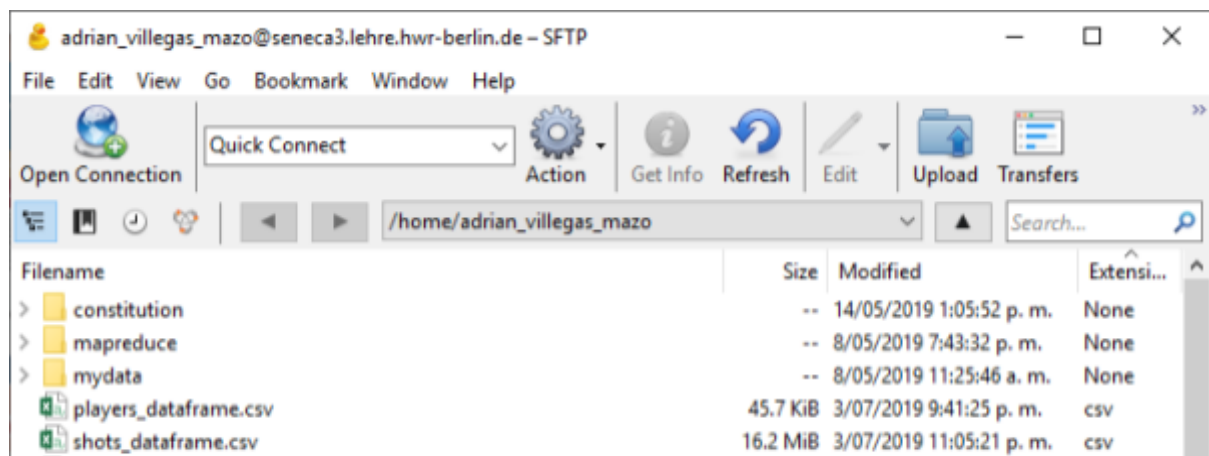


Figure 15. File upload on Seneca

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:

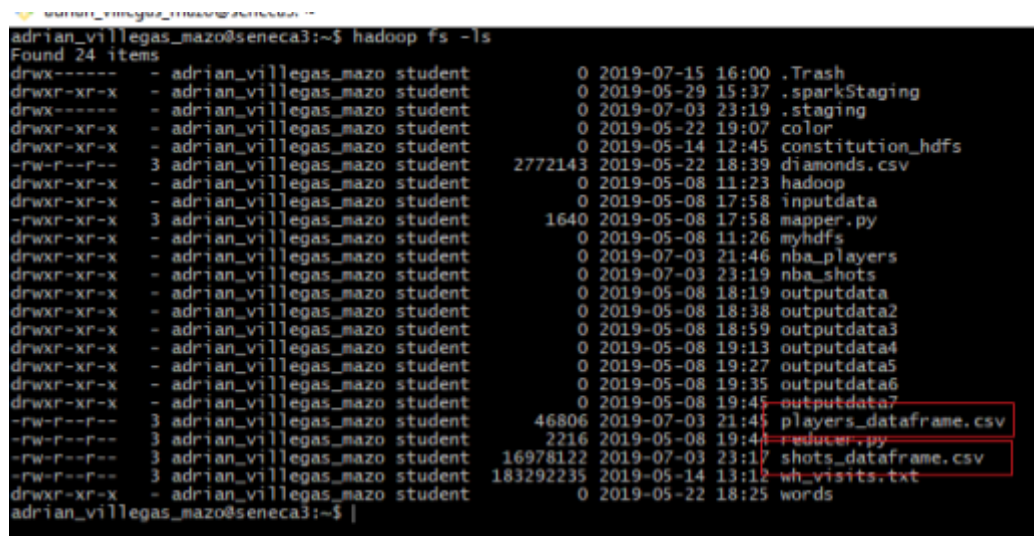


Figure 16. HDFS list of files

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
1  --nba_shots.pig: transforms shots_dataframe.csv for a Hive table
2
3  shots = LOAD 'shots_dataframe.csv' USING PigStorage(',');
4
5  nba_shots = FOREACH shots GENERATE
6      (chararray) $7 AS player_position,
7      (chararray) $3 AS home,
8      (float) $5 AS loc_x,
9      (chararray) $4 AS home_team,
10     (chararray) $11 AS shot_type,
11     (int) $8 AS points,
12     (chararray) $0 AS away_team,
13     (float) $6 AS loc_y,
14     (chararray) $12 AS time,
15     (chararray) $2 AS game_date,
16     (chararray) $10 AS shooter,
17     (int) $9 AS quarter,
18     (chararray) $1 AS outcome;
19
20 --Store the results in the folder 'nba_shots' in the HDFS home directory
21 STORE nba_shots INTO 'nba_shots/';
```

Figure 18. NBA Shots pig scripts

```
nba_shots.hive
1  create database if not exists adrian_villegas_mazo;
2
3  use adrian_villegas_mazo;
4
5  create table nba_shots (
6      player_position string,
7      home string,
8      loc_x float,
9      home_team string,
10     shot_type string,
11     points int,
12     away_team string,
13     loc_y float,
14     time string,
15     game_date string,
16     shooter string,
17     quarter int,
18     outcome string)
19 ROW FORMAT DELIMITED
20 FIELDS TERMINATED BY '\t'
21 LOCATION '/user/adrian_villegas_mazo/nba_shots/';
```

Figure 19. NBA Shots hive script

```

nba_players.pig
1  --nba_players.pig: transforms players_dataframe.csv for a Hive table
2
3  players = LOAD 'players_dataframe.csv' USING PigStorage(',');
4
5  nba_players = FOREACH players GENERATE
6      (int) $0 AS player_id,
7      (int) $1 AS jersey_number,
8      (chararray) $2 AS position,
9      (chararray) $3 AS birthdate,
10     (int) $4 AS age,
11     (chararray) $5 AS rookie,
12     (int) $6 AS team_id,
13     (chararray) $7 AS team_abbr,
14     (chararray) $8 AS team_city,
15     (chararray) $9 AS team_name,
16     (int) $10 AS games_played,
17     (int) $11 AS fg2_pt_att,
18     (int) $12 AS fg2_pt_made,
19     (int) $13 AS fg3_pt_att,
20     (int) $14 AS fg3_pt_made,
21     (int) $15 AS ft_att,
22     (int) $16 AS ft_made,
23     (chararray) $17 AS full_name;
24
25  --Store the results in the folder 'nba_players' in the HDFS home directory
26  STORE nba_players INTO 'nba_players/';

```

Figure 20. NBA Players pig script

```

nba_players.hive
1  create database if not exists adrian_villegas_mazo;
2
3  use adrian_villegas_mazo;
4
5  create table nba_players (
6      player_id int,
7      jersey_number int,
8      position string,
9      birthdate string,
10     age int,
11     rookie string,
12     team_id int,
13     team_abbr string,
14     team_city string,
15     team_name string,
16     games_played int,
17     fg2_pt_att int,
18     fg2_pt_made int,
19     fg3_pt_att int,
20     fg3_pt_made int,
21     ft_att int,
22     ft_made int,
23     full_name string)
24
25  ROW FORMAT DELIMITED
26  FIELDS TERMINATED BY '\t'
27  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

Data Source Name: Sample Cloudera Hive DSN

Description: Sample Cloudera Hive DSN

Hive Server Type: Hive Server 2

Service Discovery Mode: No Service Discovery

Service Discovery Options...

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:

Authentication

Mechanism: User Name

Realm:

Host FQDN: _HOST

Service Name: hive

☒ Canonicalize Principal FQDN

☐ Delegate Kerberos Credentials

User Name: adrian_villegas_mazo

Password:

Password Options...

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:

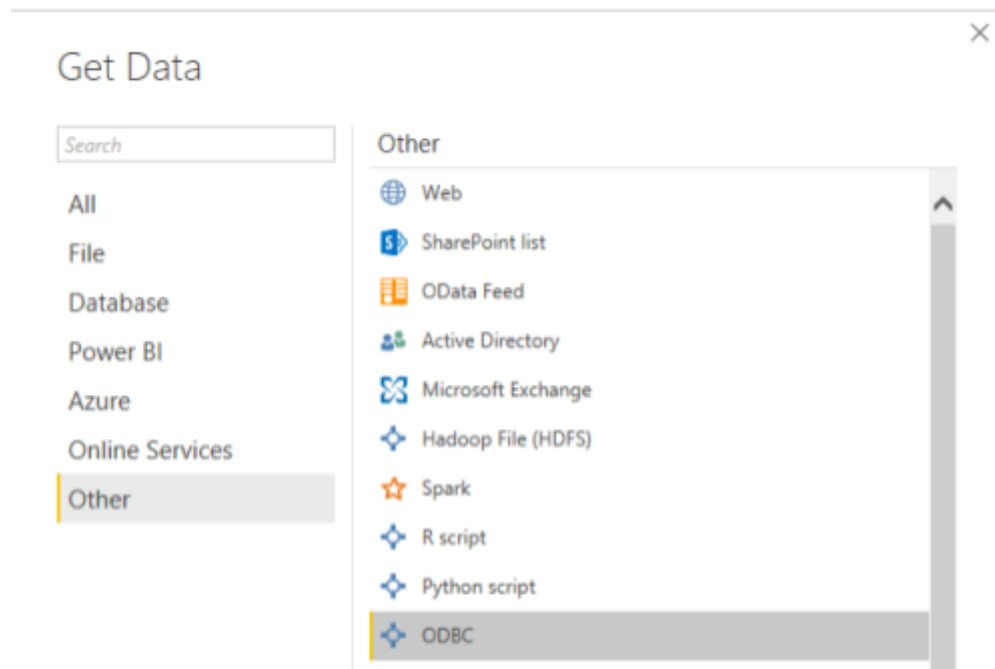


Figure 27. Power BI data source selection

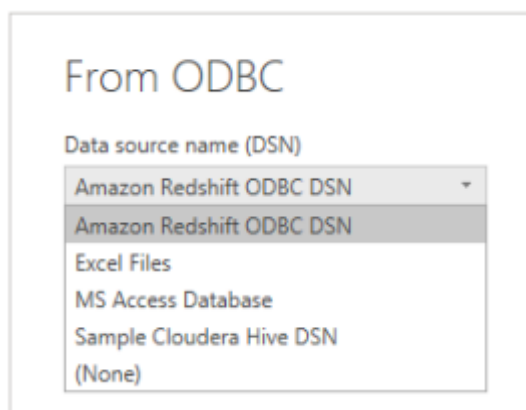


Figure 28. ODBC Driver Selection

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

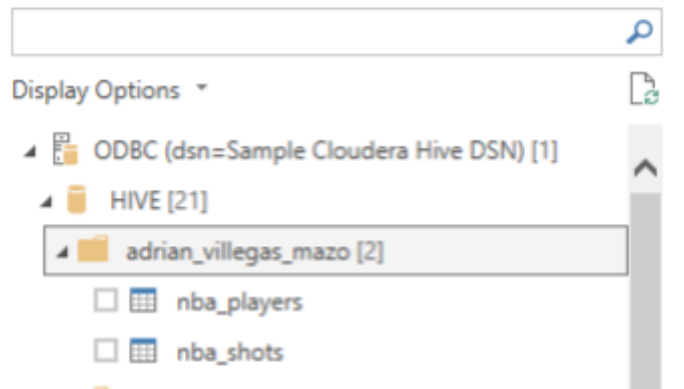


Figure 29. Power BI Table Selection

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

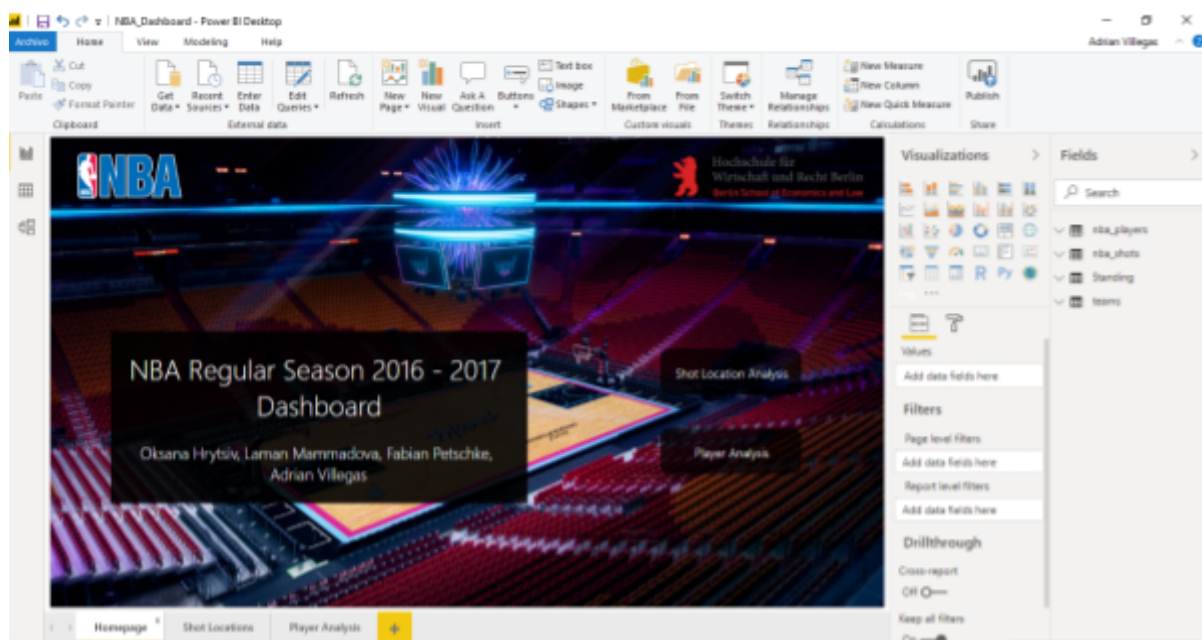
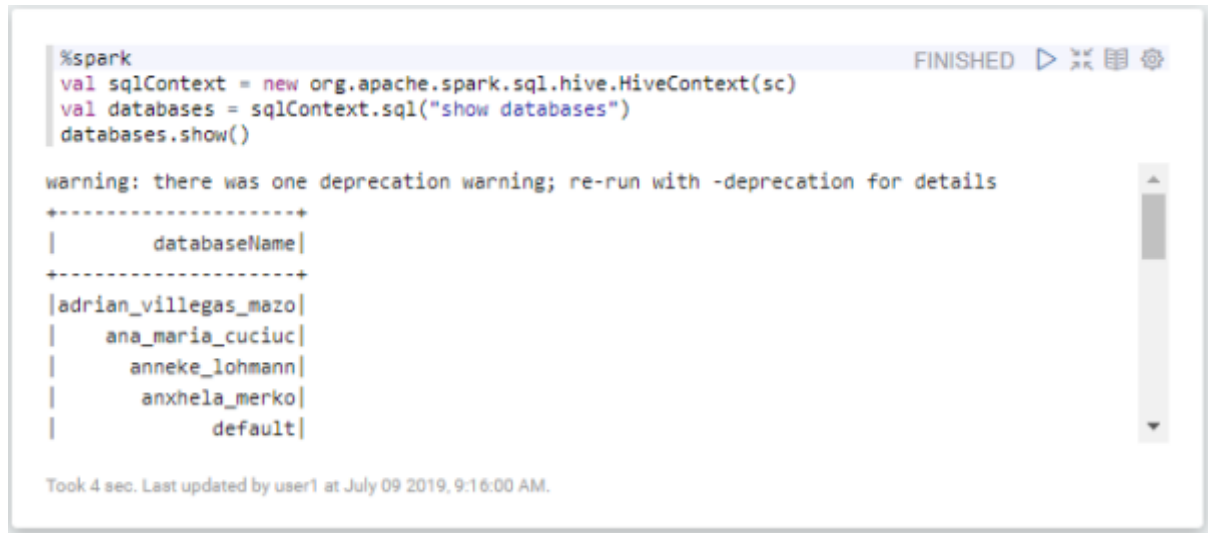


Figure 30. Power BI Dashboard Creation

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")
```

FINISHED ▶ ⌵ 📖 ⚙

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")
```

FINISHED ▶ ⌵ 📖 ⚙

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()
```

SPARK JOBS FINISHED ▶ ⌵ 📖 ⚙

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:

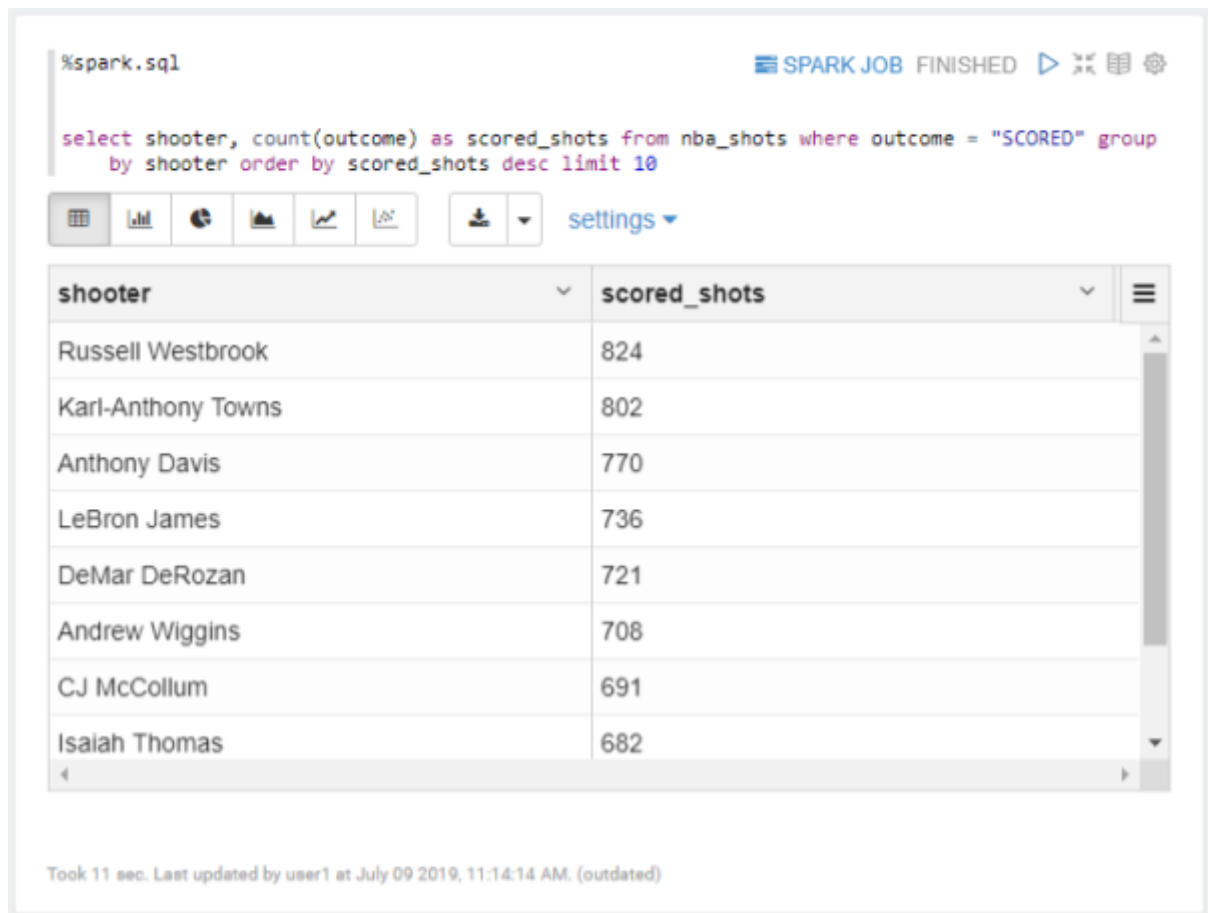


Figure 36. SQL query on Spark object

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

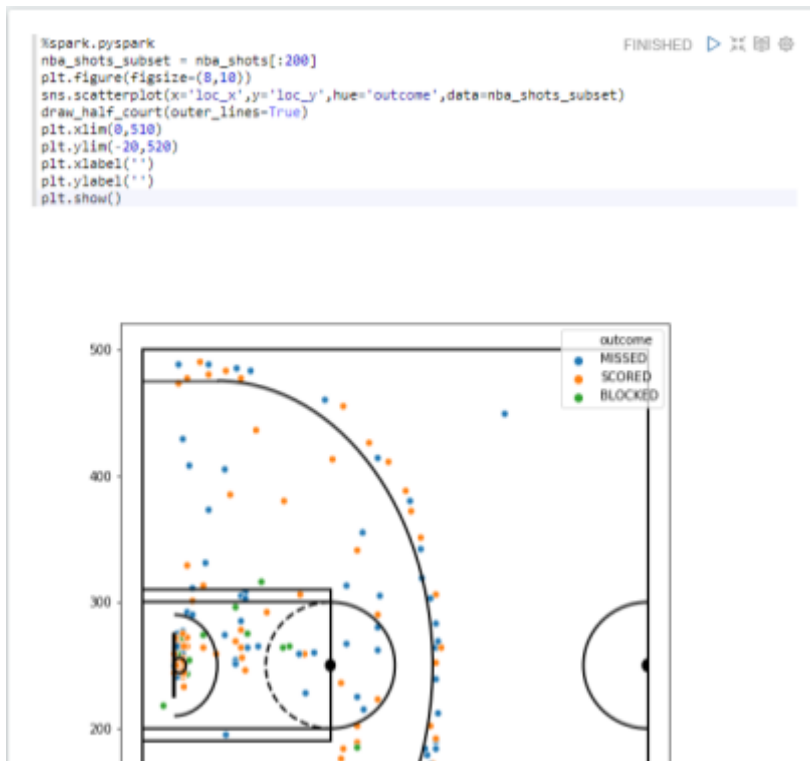


Figure 37. Pyspark half court plot

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.I.]: CreateSpace Independent Publishing Platform; [s.n.]