UNIVERSITI MALAYA

WIH3003 Big Data Applications and Analytics

Project (20%)

Group 1

Small Data and Big Data in Football Events

	Name	Matric Number
1.	Amir Firdaus Bin Abdul Hadi	17204620
2.	Imran Asyraaf Bin Mohd Ikhalan	17202811
3.	Yasmin Hannah Binti Zainol Abidin	17203246

Dr. Hoo Wai Lam

Submission date: 01 February 2023

GDrive Link: Group 1 Big Data

Table of Contents

01 Introduction **2**

02 Problem Statements **2**

03 Methodology **2**

Dataset Used 2

Tools Used 3

04 Small Data Solution vs Big Data Solution **3**

I. Data Storage 3

II. Data Analysis and Visualizations 5

III. Machine Learning 9

05 Conclusion **11**

Limitations 11

Future Works 11

Summary 11

06 References **12**

07 Appendix **13**

01 Introduction

The sports market is vast and involves many different entities such as players, teams, leagues, fan clubs, and sponsors. All of these entities generate a lot of data, which can be used for various purposes. Some of the data is used internally to make better decisions, while other data is used by the media industry to create better products and attract viewers.

Football generates a lot of data from various sources such as player tracking, match statistics, and win probabilities. Collecting, storing and analyzing this large amount of data requires specialized software and hardware that small data tools may not have. Additionally, small data tools may not have the ability to process and analyze the data in real-time which is important for making quick decisions during a match. Furthermore, small data tools might not be able to perform advanced analytics, such as machine learning, which are necessary to gain insights from the data. The use of big data tools allows for more accurate predictions and better decision-making capabilities in football analytics.

The insights gained from this data allow for more accurate predictive models that can improve player predictions and help teams develop plans of action for player performance. This data is evidence-based and tailored to the statistical figures generated for each player, allowing teams to better understand how their players perform on the field and develop unique strategies while reducing risks. Hence, to extract insights from this big data, sports and media companies need to build end-to-end data pipelines that include data engineering, data analysis, and machine learning.

02 Problem Statements

- Limited scalability: Small data tools may not be able to handle large amounts of data, making it difficult to scale the data flow as the volume of data increases from sources to end user.
- Limited data storage and processing capabilities: Small data tools may not have the capability to process large amounts of data in real-time, which can cause delays in decision-making.

03 Methodology

Dataset Used

The dataset provides a granular view of 9,074 games, totaling 941,009 events from the biggest 5 European football (soccer) leagues: England, Spain, Germany, Italy, France from 2011/2012 season to 2016/2017 season. The dataset is organized in 3 files; events.csv that contains event data about each game, ginf.csv that contains metadata and market odds about each game and dictionary.txt that contains a dictionary with the textual description of each categorical variable coded with integers.

Tools Used

Aspect	Small Data Tools	Big Data Tools
Data Storage	Excel	Hadoop, Hive
Data Analysis	R Studio	PySpark
Machine Learning	Python	PySpark

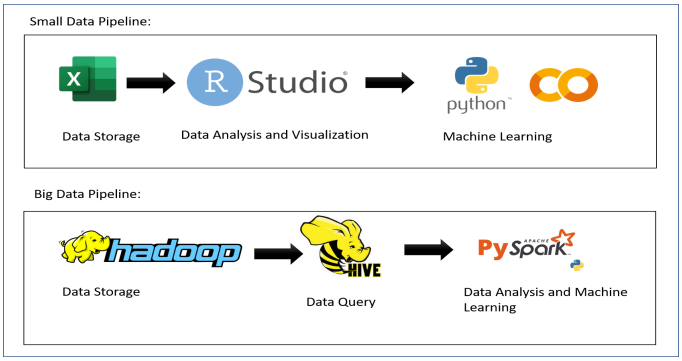


Figure 1: Pipeline for small data tools and big data tools

04 Small Data Solution vs Big Data Solution

I. **Data Storage**

In this section, we will discuss the tools that will be used in storing and querying data for small data and big data. For small data, due to low volume and velocity of data, we will use Excel directly to show how simple the data could be handled. For big data, we opted for ingested data into Hadoop to store the data and using Hive table to query it.

Big Data Storage Using Hadoop and Hive

1. Download data from the resource

   In this part, we put the data purposely in github as the medium to store the data online, Later, we will download the data from the github repository into our local machine. The zipped data then will be unzipped and moved to the root directory for easier access of the data.

2. Make directory in HDFS

   After we successfully download our data inside the local virtual box, we need to make a directory to HDFS. When we create a directory in HDFS, it provides a logical namespace for organizing files within the HDFS file system. The

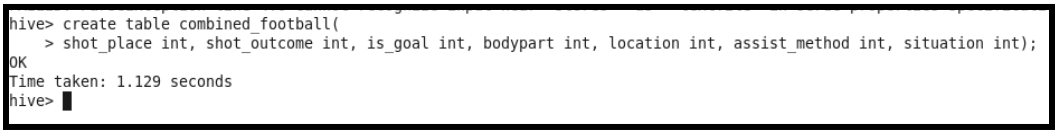
directories and files within HDFS are stored across multiple nodes in a cluster, providing scalable and fault-tolerant storage for big data applications.

3. Loading data into HDFS

   After we successfully create the directory, where it also means we create a file to store the data, we need to put our data from the local machine into the HDFS file. This could be done by making a directory from the data that we want to put from our local machine, to the directory of the file of the HDFS.

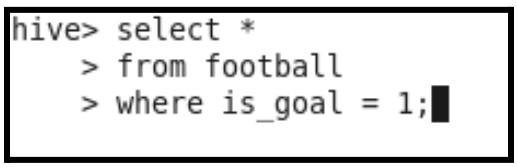
4. Create Hive table

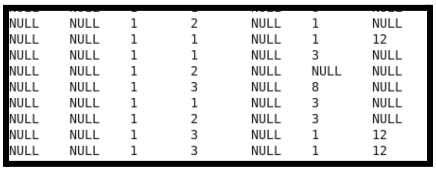
   To create a table in Hive, there will be a few simple steps. Start the Hive shell by running the command "hive" in the terminal.Then, use the CREATE TABLE statement to define the structure of the table. After we successfully create the table, we can inspect the table to show its properties. When we are satisfied with the table creation, we need to insert the data that has been located inside our HDFS file to our table.



5. Create simple query

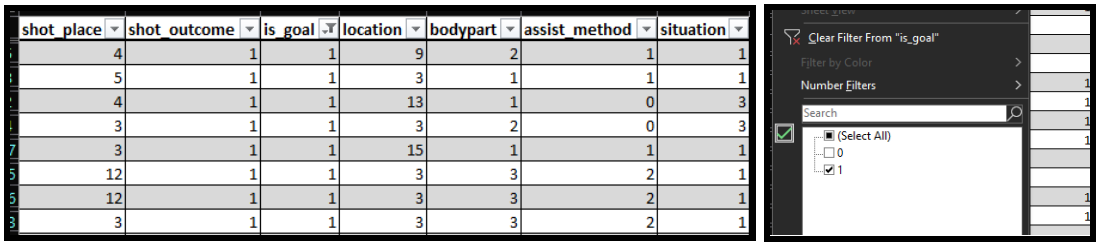
   Lastly, we could run a simple or complex sequel language to retrieve the data that we want. From here, we could see that a lot of time could be saved from one query to process thousands of lines of data.





Small Data Storage Using Excel

For small data, we want to indicate that simple small data can be stored using Excel files in the .csv format. The table shows how small data can be organized automatically by Microsoft Excel itself. The filtering features in excel can just filter the data that we want, replacing SQL language like big data.



shot_place	shot_outcome	is_goal	location	bodypart	assist_method	situation
4	1	1	9	2	1	1
5	1	1	3	1	1	1
4	1	1	13	1	0	3
3	1	1	3	2	0	3
3	1	1	15	1	1	1
12	1	1	3	3	2	1
12	1	1	3	3	2	1
3	1	1	3	3	2	1

Overall Comparison In Terms of Data Storage

Aspect	Excel	Hadoop & Hive
Data Scale	Design for small to medium dataset and has its own limit.	Designed for handling large amounts of data, in the terabytes or petabytes range.
Performance	Excel may become slow when dealing with large data sets.	Hadoop is designed for parallel processing, makes it faster and more efficient at handling large amounts of data
Data Extraction	Data can be filtered using the filter function	Hive can filter data using a query language with more complexity.
Accessibility	Excel files can be easily shared and accessed by multiple users	Data stored in Hadoop often requires specialized tools and knowledge to access and analyze

II. Data Analysis and Visualizations

For the purpose of exploring and understanding the data, these are the questions that are being investigated during the Exploratory Data Analysis (EDA) process:

- What are the minutes throughout the game that have the highest frequency of events?
- Which type of events had the most occurrences during the game?
- Who are the most frequent players and what types of events do they participate in?

To analyze the dataset for small data tools implementation, we employed RStudio along with the R Programming language. Meanwhile for big data tools implementation, we utilized Google Colab in conjunction with Apache PySpark. The codes used for creating visualizations in the form of screenshots will be included in the Appendix section.

Steps of Data Analysis and Visualization using R vs PySpark:

I. Dependencies Used

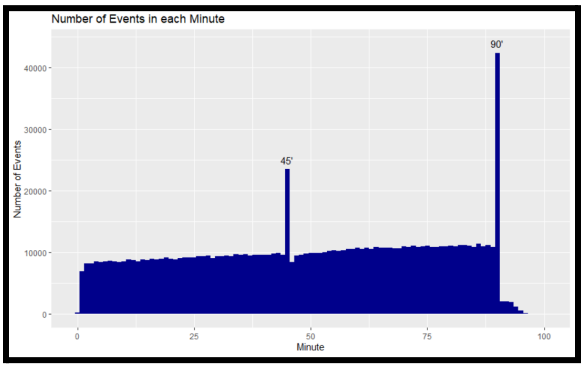
A. R

- tidyverse
- dplyr
- ggplot2

B. PySpark
- pyspark.sql
- pyspark.sql.types
- Pyspark.sql.functions (col, udf)
- plotly.express
- pandas

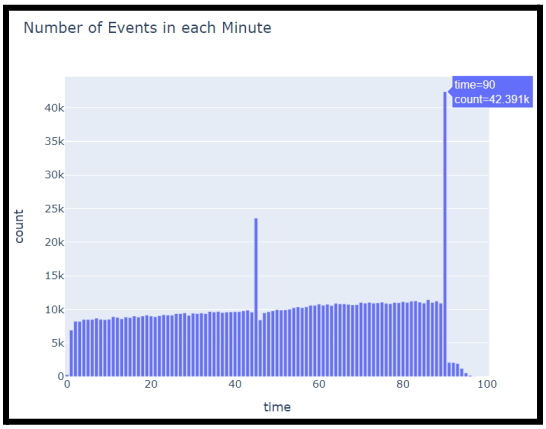
II. Question 1: What are the minutes with the highest frequency of events?

A. R



This bar chart, created using R (ggplot2), illustrates an overall upward trend in the number of events as the minutes progress. Notably, there are two significant peaks present in the plot, specifically at the 45th and 90th minutes. These spikes may be due to the fact that these moments often mark the end of a half in a soccer game, leading to an increase in events during these times. Most likely due to the adrenaline rush, substitutions, or injuries of the players.

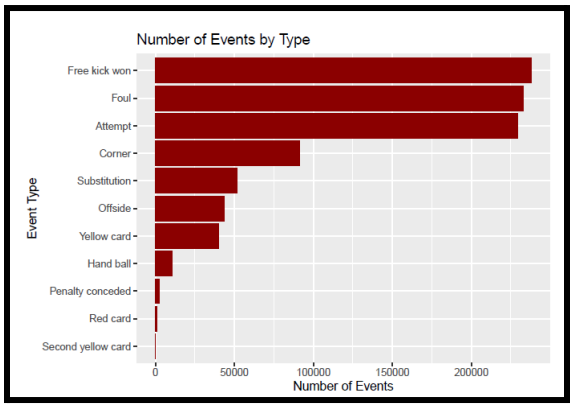
B. PySpark



In Pyspark, the dataframe must be transformed into a pandas dataframe before the Plotly package can generate the bar chart. It produces the same results as R, with the majority of events occurring in the 45th and 90th minutes.

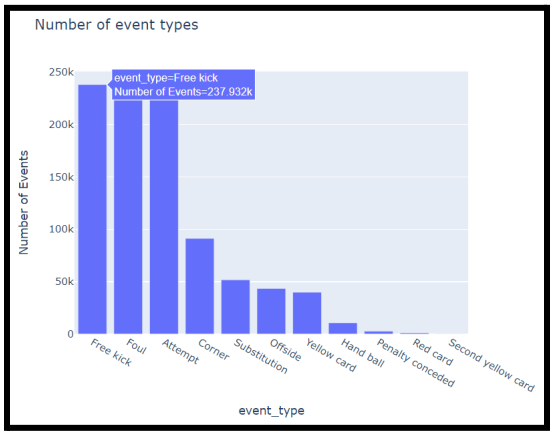
III. Question 2: Which type of events had the most occurrences during the game?

A. R



From the bar chart, the information that can be derived is that most of the events occurred are free kicks won, fouls and attempts respectively, each with over 200,000 events. Generally, this suggests that the best events to analyze would be fouls (including free kicks won) or attempts, since these are the events for which there is the most data.

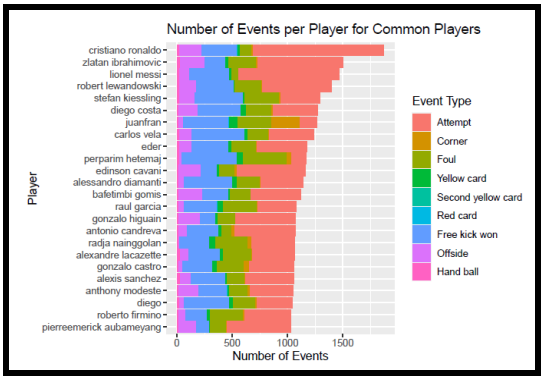
B. PySpark



Similar to the bar chart generated by R, this bar chart displays the frequency of different events that occurred during a game, with the majority of the events being free kicks, followed by fouls and attempts. This chart is useful for understanding the types of events that were most common during the game and can be used to identify patterns or trends in the way the game was played.

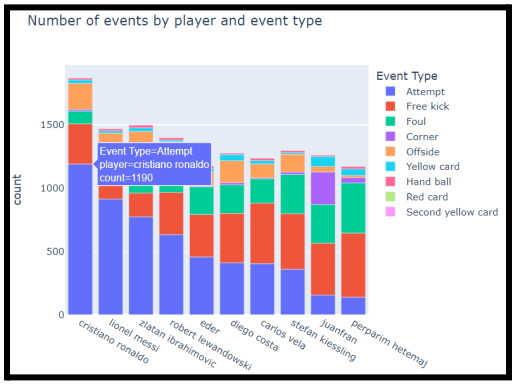
IV. Question 3: Who are the most frequent players and what types of events do they participate in?

A. R



This stacked bar chart displays the most frequent players in the data, filtered to only include those who have more than 1,000 events. Typically, these players are world-class forwards who have a high number of attempts. However, there are some notable exceptions in the chart. For example, Stefan Kiessling, a center forward who played in the Bundesliga, is the fifth most frequent player in the data, but has far fewer attempts than the players above him. Instead, his events were primarily winning free kicks and fouling other players.

B. PySpark



This stacked bar chart illustrates the distribution of events across the top 10 players who have the most events in a league. The events are separated by type and stacked to show the proportion of each event a player is involved in. The top three players on the chart are Cristiano Ronaldo, Lionel Messi, and Zlatan Ibrahimovic, who all have a high number of events across different types of play, such as free kicks, fouls, and attempts. This chart provides a useful way to quickly compare the involvement of different players in a league and identify patterns or trends in their play.

V. Overall Comparison

In summary, both R and PySpark are powerful tools for analyzing and visualizing data. While R is relatively easy to use, as it is a familiar tool, PySpark can be a bit more challenging to set up, particularly if you're not familiar with the environment. However, once the environment is configured, PySpark is just as easy to use as R.

Aspect	R	PySpark
Setup and Configuration	Does not require extensive setup and configuration. All that is needed is to import the necessary libraries for the analysis and visualization tasks.	Required a few steps to install and configure the Spark environment on Google Colab. Also need to import necessary libraries used during the analysis and visualizations.
Ease of use	It requires knowledge of the R programming language but there are many resources available for learning and troubleshooting. It is an user-friendly programming language with a large community of users.	It requires the knowledge of Python and SQL, and has a smaller community. Finding and learning resources and tools may be harder.
Visualizations	Provide more advanced and sophisticated packages like ggplot2, lattice, etc	Also provide visualization libraries that integrate with Python such as matplotlib, plotly, seaborn, etc.
Distributed computing	Not designed to handle data processing in a distributed environment	Able to process large amounts of data in a distributed environment

III. Machine Learning

For the machine learning part, we are going to make a multivariate logistic regression to predict a binary classification whether it is a goal or not (1 is goal and 0 is not goal). We are going to compare the steps used by PySpark (big data tool) and Python(small data tool) on Google Colab. The features that we used for the independent variables are "side", "shot_place", "location", "assist_method", and "situation".

Steps of Machine Learning using Python vs PySpark:

I. ML Algorithm

A. Python

```
[ ] import timeit
    start = timeit.default_timer()

    # instantiate the model (using the default parameters)
    logreg = LogisticRegression(random_state=16)
    # fit the model with data
    logreg.fit(X_train, y_train)
    y_pred = logreg.predict(X_test)
    stop = timeit.default_timer()
    print("time:", stop-start)
```

```
time: 10.2342343 seconds
```

B. PySpark

```
[ ] import timeit
    # Create Spark ML pipeline using a GBT classifier
    gbtClassifier = GBTClassifier(labelCol="is_goal", featuresCol="features", maxDepth=5, maxIter=20)
    pipelineStages = stringIndexers + encoders + [featureAssembler, gbtClassifier]
    pipeline = Pipeline(stages=pipelineStages)

    # Split dataset into training/test, and create a model from training data
    (trainingData, testData) = gamesDf.randomSplit([0.75, 0.25])
    start = timeit.default_timer()
    model = pipeline.fit(trainingData)
    stop = timeit.default_timer()
    print("time:", stop-start)
```

```
time: 123.32785144300033
```

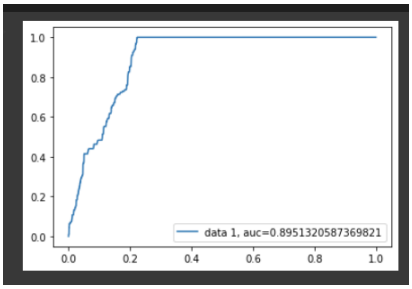
II. Evaluation

A. Python

```
[ ] acc = metrics.accuracy_score(y_test, y_pred)
    print("Accuracy: ", round(acc,2)*100, "%")

    Accuracy:  97.0 %

[ ] y_pred_proba = logreg.predict_proba(X_test)[::,1]
    fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
    auc = metrics.roc_auc_score(y_test, y_pred_proba)
    plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
    plt.legend(loc=4)
    plt.show()
```



B. PySpark

```
70] bcEvaluator = BinaryClassificationEvaluator(labelCol="is_goal", rawPredictionCol="prediction", metricName="areaUnderROC")
    print(f"Area under ROC curve: {bcEvaluator.evaluate(predictionslr)}")

    mcEvaluator = MulticlassClassificationEvaluator(labelCol="is_goal",metricName="accuracy")
    print(f"Accuracy: {mcEvaluator.evaluate(predictionslr)}")

    Area under ROC curve: 0.7780702110316613
    Accuracy: 0.9838811388446232
```

In terms of dependencies used, both Python and PySpark have different libraries for pre-processing and machine learning algorithms where PySpark used built-in MLlib. Although PySpark needs to use Python libraries for visualization as PySpark lacks built-in visualization tools. From the figure in the algorithm section, although using different libraries, the algorithm steps are still similar where we chose the columns to use, split into training and testing, then fit into the logistic model. As the dataset is not big enough to be considered big data, python processed faster than PySpark due to unnecessary overhead created where Python takes around 10 seconds while PySpark took around 123 seconds.

For the evaluation section, both Python and PySpark have built-in metrics for Area Under Curve (AUC) and Accuracy. From the results, Python Scikit-learn logistic regression has accuracy of 97% and AUC of 0.895 while PySpark MLlib logistic regression has accuracy of 98.4% and AUC of 0.78. Both tools manage to get good results in goal prediction using logistic regression. Because both tools have similar approaches and results in terms of machine learning, then it will come to other aspects as shown in table below to make comparisons.

Overall Comparisons in terms of Machine Learning

Aspect	Python	PySpark
Small Data	Works very well with small data.	Using PySpark for small datasets may introduce unnecessary overhead, as the distributed computing capabilities of Spark may not be needed for such small datasets.
Built-in ML support	Has a wide range of machine learning libraries such as scikit-learn, Tensorflow and Pytorch, which provides a lot of flexibility in terms of the algorithms and tools that you can use for machine learning.	Has built-in support for machine learning through its MLlib library, which provides a range of machine learning algorithms and tools for data preprocessing, feature extraction, and model evaluation. Although it is not as flexible as Python.

05 Conclusion

Limitations

One of the limitations of this project is the lack of big data size of the dataset. This makes it difficult to showcase the true capabilities of big data tools in terms of performance compared to small data tools. Other than that, limited representation of the end-to-end pipeline is another advantage of using big data tools that cover the flow from the source to the end-user in more efficient and automated ways.

Future Works

For future works, using real-time data ingestion from data sources to do analysis can mitigate the lack of big data size and be able to simulate the true capabilities of using big data tools for football events. Then, to be able to design and construct a pipeline from source to end-user can provide better flow and can be comparable to small data tools that are difficult to do integration and pipelining.

Summary

The results of this project demonstrate the capabilities of both small and big data tools in storing, analyzing, visualizing, and performing machine learning algorithms on the datasets. However, as the dataset grows, it may become challenging to handle football events data that generated almost every day using only small data tools such (Excel, R, Python), indicating that big data tools (Hadoop, Hive, PySpark) are more suitable and practical in the long term to solve the scalability and processing capabilities issues arise.

06 References

K, U. P., & Bhavanam, L. R. (2020). Usage of HIVE tool in Hadoop ECO system with loading data and user defined functions. *International Journal of Psychosocial Rehabilitation*, *24*(04), 1058-1062. doi:10.37200/ijpr/v24i4/pr201080

Shaw, S., Vermeulen, A. F., Gupta, A., & Kjerrumgaard, D. (2016). Setting the stage for hive: Hadoop. *Practical Hive*, 1-22. doi:10.1007/978-1-4842-0271-5_1

Chen, J. (n.d.). Analysis tools for small and big data problems. doi:10.33915/etd.5351

Loshin, D. (2013). Big data tools and techniques. *Big Data Analytics*, 61-72. doi:10.1016/b978-0-12-417319-4.00007-7

Mishra, R. K. (2017). PySpark MLlib and linear regression. *PySpark Recipes*, 235-259. doi:10.1007/978-1-4842-3141-8_9

Singh, P. (2021). Manage data with PySpark. *Machine Learning with PySpark*, 15-37. doi:10.1007/978-1-4842-7777-5_2

Kronthaler, F., & Zöllner, S. (2020). Data analysis basics with RStudio. *Data Analysis with RStudio*, 13-29. doi:10.1007/978-3-662-62518-7_2

Ramsingh, J. (2021). PySpark toward data analytics. *Big Data Applications in Industry 4.0*, 297-330. doi:10.1201/9781003175889-14

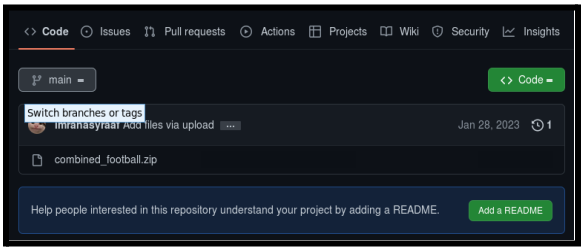
Hadoop - HDFS operations. (2014, June 26). TutorialsPoint. https://www.tutorialspoint.com/hadoop/hadoop_hdfs_operations.htm

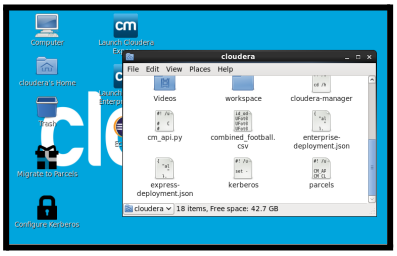
What is HDFS? Hadoop distributed file system overview. (2021, March 2). Data Management. https://www.techtarget.com/searchdatamanagement/definition/Hadoop-Distributed-File-System-HDFS

07 Appendix

Snippets of code used for Data Storing

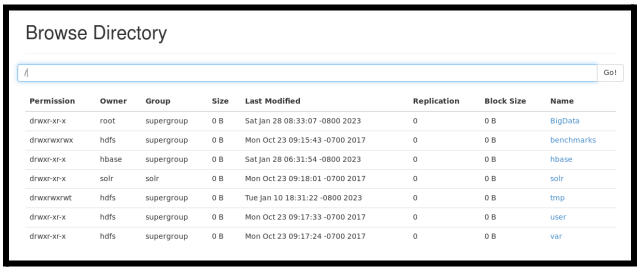
1. Download data from the resource





2. Make directory in HDFS

```
root@quickstart cloudera]# hadoop fs -mkdir /BigData
root@quickstart cloudera]# hadoop fs -mkdir /BigData/csv
root@quickstart cloudera]#
```



3. Loading data into HDFS

```
root@quickstart cloudera]# hadoop fs -put /home/cloudera/combined_football.csv
/BigData/csv
root@quickstart cloudera]#
```

4. Create Hive table

```
hive> create table combined_football(
    > shot_place int, shot_outcome int, is_goal int, bodypart int, location int, assist_method int, situation int);
OK
Time taken: 1.129 seconds
hive>
```

5. Create simple query

```
hive> select *
    > from football
    > where is_goal = 1;
```

NULL	NULL	1	2	NULL	1	NULL
NULL	NULL	1	1	NULL	1	12
NULL	NULL	1	1	NULL	3	NULL
NULL	NULL	1	2	NULL	NULL	NULL
NULL	NULL	1	3	NULL	8	NULL
NULL	NULL	1	1	NULL	3	NULL
NULL	NULL	1	2	NULL	3	NULL
NULL	NULL	1	3	NULL	1	12
NULL	NULL	1	3	NULL	1	12

```
Time taken: 0.05 seconds, Fetched: 8439 row(s)
```

<u>Snippets of code used for Data Analysis and Visualizations</u>

Question 1

A. R

```r
library(ggplot2)
ggplot(events) + geom_bar(aes(x = time), fill = "dark blue", width = 1) +
  labs(x = "Minute", y = "Number of Events", title = "Number of Events in each Minute") +
  annotate("text", x = 45, y = 25000, label = "45'") +
  annotate("text", x = 90, y = 44000, label = "90'")
```

B. PySpark

```
+----+-----+
|time|count|
+----+-----+
|  90|42391|
|  45|23587|
|  86|11446|
|  83|11255|
|  88|11228|
|  82|11215|
|  80|11150|
|  84|11079|
|  75|11068|
|  72|11048|
|  70|11035|
|  87|11025|
|  81|11020|
|  78|11018|
|  79|10987|
|  74|10975|
|  85|10928|
|  73|10922|
|  89|10917|
|  71|10913|
+----+-----+
only showing top 20 rows
```

```python
from pyspark.sql.functions import col
import plotly.express as px

# group the events by time and count the number of events in each group
event_counts = df.groupBy("time").count()

# sort the result by time
event_counts = event_counts.orderBy(col("time"))

# show the result in descending order
event_counts = event_counts.sort(col("count").desc())
event_counts.show()

# convert the event counts dataframe to a pandas dataframe
event_counts_pd = event_counts.toPandas()

# create the bar chart
fig = px.bar(event_counts_pd, x='time', y='count', title='Number of Events in each Minute')

# show the chart
fig.show()
```

Question 2

A. R

```r
event_type.labs = c("0" = "Announcement", "1" = "Attempt", "2" = "Corner", "3" = "Foul", "4" = "Yellow card",
                    "5" = "Second yellow card", "6" = "Red card", "7" = "Substitution", "8" = "Free kick won",
                    "9" = "Offside", "10" = "Hand ball", "11" = "Penalty conceded")
events %>% add_count(event_type) %>% ggplot() +
  geom_bar(aes(y = reorder(event_type, n)), fill = "dark red") +
  scale_y_discrete("Event Type", labels = event_type.labs) +
  labs(x = "Number of Events", title = "Number of Events by Type")
```

B. PySpark

```
event_type_count = df.groupBy("event_type").count()
event_type_count = event_type_count.sort("count", ascending=False)
event_type_count.show()
```

```
from pyspark.sql.functions import udf

event_type_count = event_type_count.withColumnRenamed("count", "Number of Events")

event_type_labs = {
    0 : "Announcement",
    1 : "Attempt",
    2 : "Corner",
    3 : "Foul",
    4 : "Yellow card",
    5 : "Second yellow card",
    6 : "Red card",
    7 : "Substitution",
    8 : "Free kick",
    9 : "Offside",
    10 : "Hand ball",
    11 : "Penalty conceded"
}

event_type_label = udf(lambda x: event_type_labs[x])
event_type_count = event_type_count.withColumn("event_type", event_type_label(event_type_count.event_type))
event_type_count.show()
```

```
# convert the event counts dataframe to a pandas dataframe
event_type_count_pd = event_type_count.toPandas()

# create the bar chart
fig2 = px.bar(event_type_count_pd, x='event_type', y='Number of Events', title='Number of event types')

# show the chart
fig2.show()
```

```
+----------+------+
|event_type| count|
+----------+------+
|         8|237932|
|         3|232925|
|         1|229135|
|         2| 91204|
|         7| 51738|
|         9| 43476|
|         4| 39911|
|        10| 10730|
|        11|  2706|
|         6|  1152|
|         5|   100|
+----------+------+
```

```
+------------------+----------------+
|        event_type|Number of Events|
+------------------+----------------+
|         Free kick|          237932|
|              Foul|          232925|
|           Attempt|          229135|
|            Corner|           91204|
|      Substitution|           51738|
|           Offside|           43476|
|       Yellow card|           39911|
|         Hand ball|           10730|
| Penalty conceded|            2706|
|          Red card|            1152|
|Second yellow card|             100|
+------------------+----------------+
```

Question 3

A. R

```
events[!(is.na(events$player)) & !(is.na(events$event_type)),] %>% add_count(player) %>% filter (n > 1000) %>%
  ggplot() + geom_bar(aes(y = reorder(player, n), fill = event_type)) +
  scale_fill_discrete("Event Type", labels = event_type.labs) +
  labs(y = "Player", x = "Number of Events", title = "Number of Events per Player for Common Players")
```

B. PySpark

```python
# Rename the event types
event_type_count_pd["event_type"] = event_type_count_pd["event_type"].replace({
    0 : "Announcement",
    1 : "Attempt",
    2 : "Corner",
    3 : "Foul",
    4 : "Yellow card",
    5 : "Second yellow card",
    6 : "Red card",
    7 : "Substitution",
    8 : "Free kick",
    9 : "Offside",
    10 : "Hand ball",
    11 : "Penalty conceded"
})

# Sort the dataframe by the count column in descending order
event_type_count_pd = event_type_count_pd.sort_values("count", ascending=False)

# Get the top 10 players with the highest number of events
top_10_players = event_type_count_pd.groupby("player").sum().sort_values("count", ascending=False).head(10)

# Filter the dataframe to only include events from the top 10 players
event_type_count_pd = event_type_count_pd[event_type_count_pd["player"].isin(top_10_players.index.values)]

# Create the stacked bar chart
fig = px.bar(event_type_count_pd, x='player', y='count', color='event_type',
             title='Number of events by player and event type',
             labels={'event_type': 'Event Type'})

fig.show()
```

Snippets of code used for Machine Learning

III. ML Algorithm

A. Python

```python
[ ] #split dataset in features and target variable
    feature_cols = ["side", "shot_place", "location", "assist_method", "situation"]
    X = df[feature_cols] # Features
    y = df.is_goal # Target variable

    # split X and y into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=16)

[ ] # instantiate the model (using the default parameters)
    logreg = LogisticRegression(random_state=16)

    # fit the model with data
    logreg.fit(X_train, y_train)
    y_pred = logreg.predict(X_test)
```

B. PySpark

```python
[67] # Create a list of variables to used
     variables = ["side", "shot_place", "location", "assist_method", "situation"]

     # Combine all variables into a single feature vector
     featureAssemblerlr = VectorAssembler(inputCols = variables, outputCol = "features")

[68] #instantiate the model and pipeline
     lr = LogisticRegression(featuresCol="features", labelCol="is_goal", regParam=1.0)
     pipelineStageslr = [featureAssembler, lr]
     pipelinelr = Pipeline(stages=pipelineStages)

     # Split dataset into training/test, and create a model from training data
     (trainingData, testData) = gamesDf.randomSplit([0.75, 0.25])
     lrmodel = pipelinelr.fit(trainingData)
```

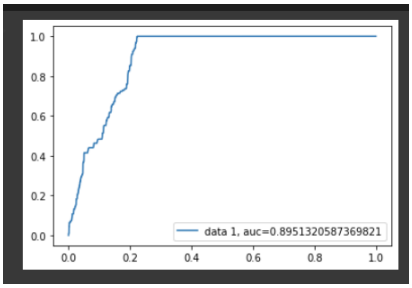
IV. Evaluation

A. Python

```
[ ]  acc = metrics.accuracy_score(y_test, y_pred)
     print("Accuracy: ", round(acc,2)*100, "%")

     Accuracy:  97.0 %

[ ]  y_pred_proba = logreg.predict_proba(X_test)[::,1]
     fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
     auc = metrics.roc_auc_score(y_test, y_pred_proba)
     plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
     plt.legend(loc=4)
     plt.show()
```



B. PySpark

```
[70] bcEvaluator = BinaryClassificationEvaluator(labelCol="is_goal", rawPredictionCol="prediction", metricName="areaUnderROC")
     print(f"Area under ROC curve: {bcEvaluator.evaluate(predictionslr)}")

     mcEvaluator = MulticlassClassificationEvaluator(labelCol="is_goal",metricName="accuracy")
     print(f"Accuracy: {mcEvaluator.evaluate(predictionslr)}")

     Area under ROC curve: 0.7780702110316613
     Accuracy: 0.9838811388446232
```