UE18CS322 – BIG DATA
FINAL PROJECT REPORT

PREMIER LEAGUE ANALYSIS USING BIG DATA TECHNIQUES

SUBMITTED BY:

Sl no	Name	SRN
1	Sunad Suhas	PES2201800511
2	Khushdeep Kaur	PES2201800063

ABSTRACT:

The global sports market is huge, comprised of players, teams, leagues, fan clubs, sponsors, etc., and all of these entities interact in myriad ways generating an enormous amount of data. Some of that data is used internally to help make better decisions, and there are a number of use cases within the media industry that use the same data to create better products and attract/retain viewers.

A few ways that the sports and media industries have started utilizing big data are:

- ❖ Analyse on-field conditions and events (passes, player positions, etc.) that lead to soccer goals, football touchdowns, or baseball home runs etc.
- ❖ Assess the win-loss percentage with different combinations of players in different on-field positions.
- ❖ Track a sportsperson’s or team’s performance graph over the years/seasons.

In our analysis, we process the streamed events data and rate players based on 4 parameters:

1. Pass Accuracy:

A pass is signified by eventId = 8
Pass Accuracy must be bound between 0 and 1
If the “tags” field in the events JSON has:

i. ‘id’ = 1801, it is an accurate pass
ii. ‘id’ = 1802, it not an accurate pass
iii. ‘id’ = 302, it is a key pass

Formula:
Pass accuracy = (no of accurate normal passes+(2* no of accurate key passes))/(number of normal passes + (no of key passes *2))

2. Duel Effectiveness

A duel is signified by eventId = 1
Duel Effectiveness must be bound between 0 and 1
If the “tags” field in the events JSON has:

i. ‘id’ = 701, duel is lost
ii. ‘id’ = 702 , duel is neutral
iii. ‘id’ = 703, duel is won

Formula:
Duel effectiveness = (No of duels won + (no of neutral duels * 0.5))/total no of duels

3. Free Kick Effectiveness

A free kick is signified by eventId = 3.
Free kick effectiveness must be bound between 0 and 1.
If the “tags” field in the events JSON has:

i. ‘id’ = 1801 it is an accurate pass

ii. ‘id’ = 1802 it not an accurate pass.

If the subEventId = 35, the free kick is a penalty and in such a case if tags has Id = 101, the penalty was a goal.

Some penalties can be effective but may not be a goal.

Formula:

Free Kick Effectiveness: (No of effective free kicks + no of penalties scored)/total no of free kicks

4. Shots on Target

A shot is signified by eventId = 10

Shots on target must be bound between 0 and 1

If the “tags” field in the events JSON has:

i. ‘id’ = 1801, shot is on target

ii. ‘id’ = 1802, shot is not on target

iii. ‘id’ = 101, shot was a goal

Formula:

Shots effectiveness = (shots on target and goals +(0.5* shots on target but not goals))/ total shots

The amount of data being generated regarding European soccer players is massive. This calls for improvised methods of storage and processing of data.

Spark, specifically pyspark, is used for processing and analysing the data. Spark is an easier alternative to the cumbersome map-reduce programs.

The data streamed is in JSON format and for every match first a match JSON object is sent followed by event JSON objects.

CODE:

1. stream.py

```
def send_data_to_spark(tcp_connection, eve, mat):
    m = eve[0]['matchId']
    fl = 0
    msg = ''
    for i in eve:
        if i['matchId'] == m:
            msg = json.dumps(i)
        else:
            fl = 0
            time.sleep(5)
            m = i['matchId']
            msg = json.dumps(i)
        if fl == 0:
            fl = 1
            for j in mat:
                if j['wyId'] == m:
                    tcp_connection.send((json.dumps(j)+'\n').encode())
                    break

        tcp_connection.send((msg+'\n').encode())


TCP_IP = 'localhost'
TCP_PORT = 6100
conn = None
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
s.bind((TCP_IP, TCP_PORT))

with open('eve.txt', 'rb') as (f):
    t = f.read()
    da = fer.decrypt(t)
    js1 = json.loads(da)
with open('mat.txt', 'rb') as (f):
    t = f.read()
    da = fer.decrypt(t)
    js2 = json.loads(da)
time.sleep(2)
s.listen(1)

print('Waiting for connection...')
conn, addr = s.accept()
print('Connected... Starting to push EPL data')
send_data_to_spark(conn, js1, js2)
```

2. try.py:

Importing libraries:

```
import time
from pyspark import SparkConf,SparkContext
from pyspark.streaming import StreamingContext
from pyspark.sql import Row,SQLContext
import sys
import json
import requests
#pyspark is the main tool , it helps to use Dstream object.

#spark-submit --master local[2] listen.py 2> logs.txt
#setting up the local host port , basically connecting to the socket and stream data
from the port(TCP)
conf=SparkConf()
conf.setAppName("BigData")
sc=SparkContext(conf=conf)

ssc=StreamingContext(sc,2)
ssc.checkpoint("checkpoint_BIGDATA")
#stream.py is doing the streaming and the data is coming, waiting for connection
is open , spark -submit is done, and then it gets the data.
dataStream=ssc.socketTextStream("localhost",6100)

lines = dataStream.map(lambda x:x.split('\n'))
#basically spllitting and giving us every line
```

```
records = lines.map(lambda x: json.loads(x[0]))
#gives us the records, we can filter any match from the records.

#duel effectiveness




def get_neutral(x):
   for i in x['tags']:
      if i['id'] == 702:
         return (x['playerId'],1)
      return (x['playerId'],0)
def get_won(x):
   for i in x['tags']:
      if i['id'] == 703:
         return (x['playerId'],1)
      return (x['playerId'],0)






#-------#

#every object here is Dstream object
events=records.filter(lambda x: len(x) == 12)#FILTERS the event records, we dont
want from match.txt
#eventmap = events.map(lambda x: (x['playerId'],x['matchId']))
trial = events.filter(lambda x: x['eventId'] ==10)#filters records with eventID 10
#freekicks

trial1 = events.filter(lambda x: x['eventId'] ==1)#filters records with eventID 1
#duel effeectiveness

passes = events.filter(lambda x: x['eventId'] ==8)#filters records with eventID 8,
ie.#passes
#eventid specifies what is happening , rather which event we are looking for, so
above it only uses only passes
```

```
#-------#
#in every event
#pass accuracy
def get_accpass(x):
    for i in x['tags']:
            if i['id'] == 1801:
                    return 1
    return 0

def get_inaccpass(x):
    for i in x['tags']:
            if i['id'] == 1802:
                    return 1
    return 0

def get_keypass(x):
    for i in x['tags']:
            if i['id'] == 302:
                    return 1
    return 0

def get_keypasses(x):
    for i in x['tags']:
            if i['id'] == 302:
                    return (x['playerId'],1)
    return (x['playerId'],0)

def get_notkeypass(x):
    for i in x['tags']:
            if i['id'] == 302:
                    return (x['playerId'],0)
    return (x['playerId'],1)

def get_acc_keypass(x):
    if get_keypass(x) and get_accpass(x):
            return (x['playerId'],1)
    else:
            return (x['playerId'],0)

def get_acc_notkeypass(x):
    if get_accpass(x):
            for i in x['tags']:
                    if i['id'] == 302:
                            return (x['playerId'],0)
            return (x['playerId'],1)
    else:
            return (x['playerId'],0)


#PASS ACCURACY
acc_normalpass = passes.map(get_acc_notkeypass)
acc_keypass = passes.map(get_acc_keypass)
all_normalpass = passes.map(get_notkeypass)
all_keypass = passes.map(get_keypasses)

#acc_normalpass_count=acc_normalpass.map(lambda x: (x['playerId'],1))
total_acc_normalpass=acc_normalpass.reduceByKey(lambda a,b:a +b)
#a +b is getting all the normal passes and adding to the previous accurate pass...
#reduce by key basically makes it small
#When a reduceByKey function is applied on a Spark RDD, it merges the values for
each key by using an associative reduce function.

#acc_keypass_count=acc_keypass.map(lambda x: (x['playerId'],1))
total_acc_keypass=acc_keypass.reduceByKey(lambda a,b:a +b)

num_join=total_acc_normalpass.join(total_acc_keypass)
#num_join.pprint() #numerator of the formula, so you have to all join because
they are Dstream objects, so you just cant add them.

pass_acc_num = num_join.map(lambda x: (x[0],x[1][0] + 2*x[1][1]))
#A lambda function can take any number of arguments, but can only have one
expression.

#normalpass_count=all_normalpass.map(lambda x: (x['playerId'],1))
total_normalpass=all_normalpass.reduceByKey(lambda a,b:a +b) #denominator
```

```
#keypass_count=all_keypass.map(lambda x: (x['playerId'],1))
total_keypass=all_keypass.reduceByKey(lambda a,b:a +b)

den_join = total_normalpass.join(total_keypass)
pass_acc_den = den_join.map(lambda x: (x[0],x[1][0] + 2*x[1][1]))#denominator join

final_passjoin=pass_acc_num.join(pass_acc_den)#joining numerator and denominator

pass_accuracy = final_passjoin.map(lambda x:(x[0], round((x[1][0]/x[1][1]),5)))
#pass_accuracy.pprint()
#passes.pprint()#player information passes



#-------#



#DUEL EFFECTIVENESS
neutral_duel = trial1.map(get_neutral) #which records are nuetral
won_duel= trial1.map(get_won) #mappipng from trial1, becuase trial 1 is set of all events that are duel
#won_duel.pprint()
total_neutral=neutral_duel.reduceByKey(lambda a,b:a +b)

total_won=won_duel.reduceByKey(lambda a,b:a +b)

dj_1=total_won.join(total_neutral)
#dj_1.pprint() #gives something like this (7918, (5, 5)) (player,won,neutral)
a1=dj_1.map(lambda x: (x[0],x[1][0] + 0.5 *x[1][1]))
#a1.pprint()  # (3560, 6.5)
#count of duels
m= trial1.map(lambda x: (x['playerId'],1))
total_duels=m.reduceByKey(lambda a,b:a +b)
#total_duels.pprint() #(15054, 19)
```

```
a2=a1.join(total_duels)
#a2.pprint()  #(20450, (5.0, 19))
duel_eff=a2.map(lambda x:(x[0], round((x[1][0]/x[1][1]),5)))
#duel_eff.pprint() # (25804, 0.28846)


#-------#


#shot effectiveness
def get_target(x):
    for i in x['tags']:
            if i['id'] == 1801:
                    return x

def get_target1(x):
            for i in x['tags']:
                    if i['id'] == 1801:
                            return (x['playerId'],1)
            return (x['playerId'],0)

def get_nottarget(x):
    for i in x['tags']:
            if i['id'] == 1802:
                    return (x['playerId'],1)
    return (x['playerId'], 0)

def goals(x):
    for i in x['tags']:
            if i['id'] == 101:
                    return (x['playerId'],1)
    return (x['playerId'],0)


#SHOT EFFECTIVENESS
shots_on_target = trial.map(get_target1)#gets shots on target
total_targets=shots_on_target.reduceByKey(lambda a,b:a +b)#gets count of shots on target. total_targets = (eventId, count)
```

```
shots_non_target = trial.map(get_nottarget)#gets shots not on target

shots_on_targ = trial.filter(get_target)
shots_and_goals = shots_on_targ.map(goals)#gets goals from records which have
shots on targets

#shots_on_target_count=shots_on_target.map(lambda x: (x['playerId'],1))
#shots_and_goals_count = shots_and_goals.map(lambda x: (x['playerId'],1))

total_goals=shots_and_goals.reduceByKey(lambda a,b:a +b)#gets count of goals .
total_goal = (eventId, here 10, count of goals)

shots_non_goal = total_targets.join(total_goals)#DStream of the form (eventId,
(count of total_targets, count of total_goals))
shots_non_goals_count=shots_non_goal.map(lambda x: (x[0],x[1][0] -
x[1][1]))#gets total on target not goals records

joined_1= total_goals.join(shots_non_goals_count)# joining (eventID, (total goals,
shots on target and not goals count))
shots_numerator=joined_1.map(lambda x: (x[0],x[1][0] + 0.5 *x[1][1]))

#count of shots
x = trial.map(lambda x: (x['playerId'],1))
total_shots=x.reduceByKey(lambda a,b:a +b) #total_shots count

joined_2= shots_numerator.join(total_shots)#joining numerator and total shots
shot_effectiveness=joined_2.map(lambda x:(x[0], round((x[1][0]/x[1][1]),5)))#gets
the shot effectiveness.
#shot_effectiveness.pprint()
#new=total_shots.map(lambda x: x[1])


#-------#
```

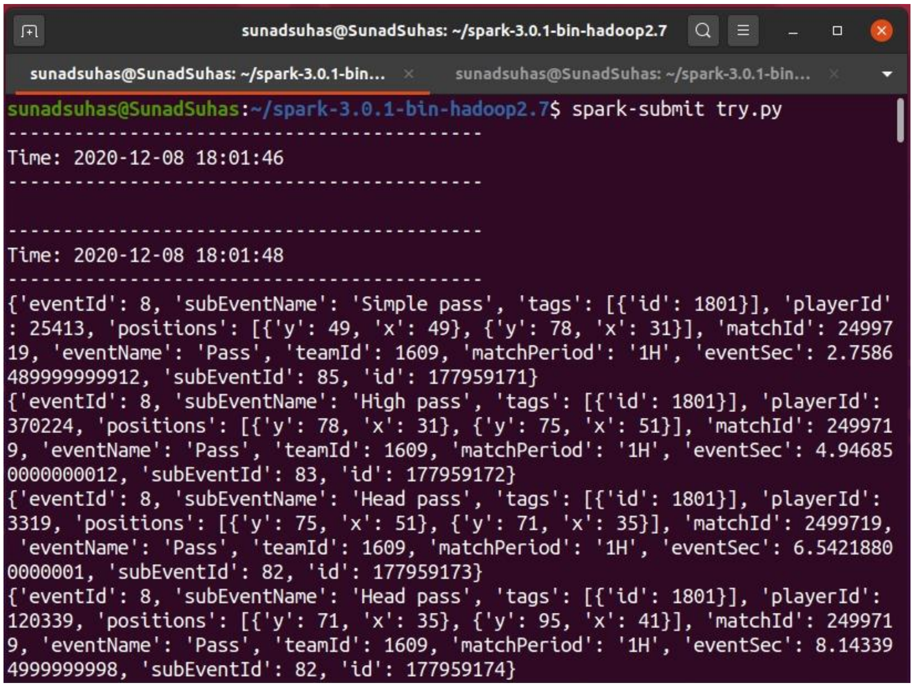
```
# free kick effectiveness

def func_3(x):
    if (len(x)==12):
            if (x['eventId']==3):
                    return x



def func_1802(x):
            for i in x['tags']:
                    if i['id'] == 1802:
                            return (x['playerId'],1)
            return (x['playerId'], 0)

def func_1801(x):
            for i in x['tags']:
                    if i['id'] == 1801:
                            return (x['playerId'],1)
            return (x['playerId'],0)



def func_1801_2(x):
            for i in x['tags']:
                    if i['id'] == 1801:
                            return x

def func_pengoal(x):
    if x['subEventId']==35:
                    for i in x['tags']:
                            if i['id']==101:
                                    return (x['playerId'],1)
    else:
            return (x['playerId'], 0)

free_kick = records.filter(func_3)

#Inaccurate passes
a_ = free_kick.map(func_1802)
```

```
a_inter1 = a_.reduceByKey(lambda x,y: x+y)

#accurate passes
b_ = free_kick.map(func_1801)
b_inter1 = b_.reduceByKey(lambda x,y: x+y)


# total free kick = accurate + inaccurate passes
a_b_inter = a_inter1.join(b_inter1)
a_b = a_b_inter.map(lambda x:(x[0], x[1][0] + x[1][1]))

# Penalties which are goal
b2 = free_kick.filter(func_1801_2)
b_1 = b2.map(func_pengoal)
b_1_inter1 = b_1.reduceByKey(lambda x,y: x+y)


# number of penalties + number of effective free kicks
b_b_1 = b_inter1.join(b_1_inter1)
b_b_1_fin = b_b_1.map(lambda x:(x[0], x[1][0] + x[1][1]))


#division
free_effect_inter = b_b_1_fin.join(a_b)
free_effect = free_effect_inter.map(lambda x:(x[0], round((x[1][0] / x[1][1]),5)))
#free_effect.pprint()


#-------#

#last part of the code
#joining everything (all 4 parameters)
pass_duel=pass_accuracy.join(duel_eff)
pass_duel1=pass_duel.map(lambda x:(x[0], round((x[1][0] + x[1][1]),5)))

pass_duel_shot = pass_duel1.join(shot_effectiveness)
pass_duel_shot1=pass_duel_shot.map(lambda x:(x[0], round((x[1][0] + x[1][1]),5)))
```

```
pass_duel_shot_free = pass_duel_shot1.join(free_effect)
contribution = pass_duel_shot.map(lambda x:(x[0], round((x[1][0] + x[1][1])/4,5)))
#eventmap.pprint()
contribution.pprint()
#entire contribution of that particular player to the match , which is average of all the 4 parameter, (overall effectiveness of the players)


ssc.start()
ssc.awaitTermination()
ssc.stop()
```

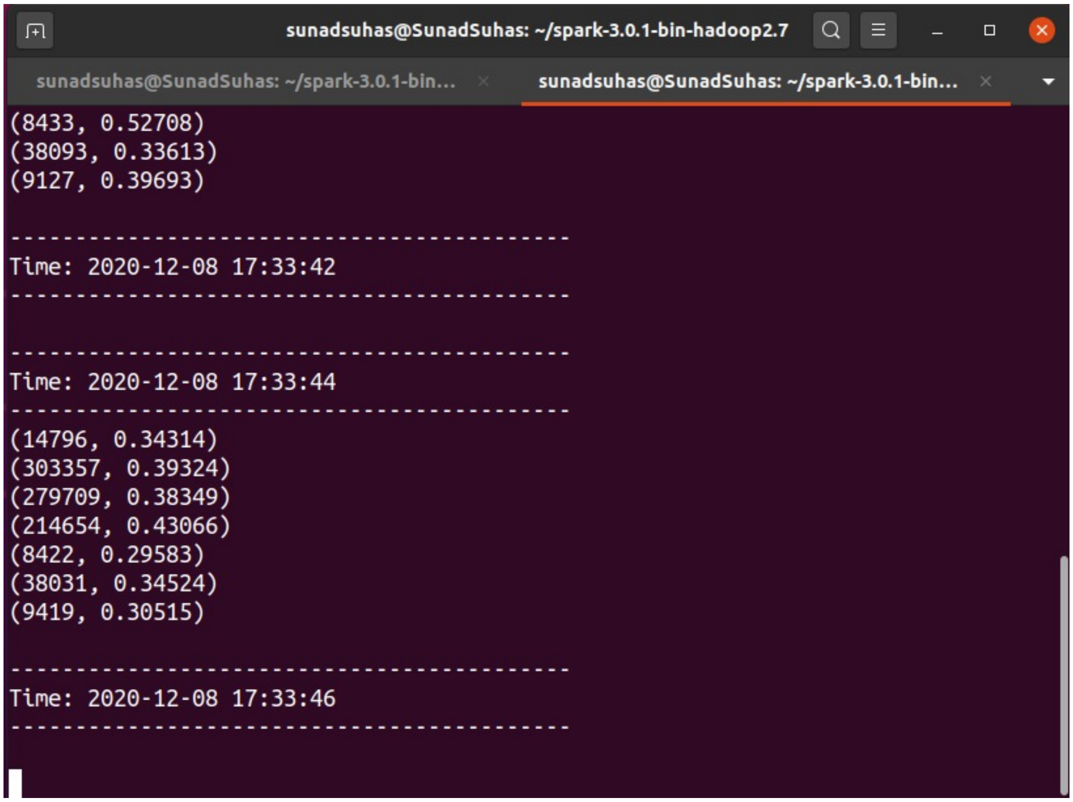
OUTPUT:

1. Pass accuracy of individual players:


sunadsuhas@SunadSuhas: ~/spark-3.0.1-bin-hadoop2.7
sunadsuhas@SunadSuhas: ~/spark-3.0.1-bin...
sunadsuhas@SunadSuhas: ~/spark-3.0.1-bin...

```
sunadsuhas@SunadSuhas:~/spark-3.0.1-bin-hadoop2.7$ spark-submit try.py
-------------------------------------------
Time: 2020-12-08 18:01:46
-------------------------------------------

-------------------------------------------
Time: 2020-12-08 18:01:48
-------------------------------------------
{'eventId': 8, 'subEventName': 'Simple pass', 'tags': [{'id': 1801}], 'playerId'
: 25413, 'positions': [{'y': 49, 'x': 49}, {'y': 78, 'x': 31}], 'matchId': 24997
19, 'eventName': 'Pass', 'teamId': 1609, 'matchPeriod': '1H', 'eventSec': 2.7586
489999999912, 'subEventId': 85, 'id': 177959171}
{'eventId': 8, 'subEventName': 'High pass', 'tags': [{'id': 1801}], 'playerId':
370224, 'positions': [{'y': 78, 'x': 31}, {'y': 75, 'x': 51}], 'matchId': 249971
9, 'eventName': 'Pass', 'teamId': 1609, 'matchPeriod': '1H', 'eventSec': 4.94685
0000000012, 'subEventId': 83, 'id': 177959172}
{'eventId': 8, 'subEventName': 'Head pass', 'tags': [{'id': 1801}], 'playerId':
3319, 'positions': [{'y': 75, 'x': 51}, {'y': 71, 'x': 35}], 'matchId': 2499719,
 'eventName': 'Pass', 'teamId': 1609, 'matchPeriod': '1H', 'eventSec': 6.5421880
0000001, 'subEventId': 82, 'id': 177959173}
{'eventId': 8, 'subEventName': 'Head pass', 'tags': [{'id': 1801}], 'playerId':
120339, 'positions': [{'y': 71, 'x': 35}, {'y': 95, 'x': 41}], 'matchId': 249971
9, 'eventName': 'Pass', 'teamId': 1609, 'matchPeriod': '1H', 'eventSec': 8.14339
4999999998, 'subEventId': 82, 'id': 177959174}
```

2. playerID with their final score:
   The final score is the average of all the 4 parameters considered.



sunadsuhas@SunadSuhas: ~/spark-3.0.1-bin-hadoop2.7
sunadsuhas@SunadSuhas: ~/spark-3.0.1-bin...
sunadsuhas@SunadSuhas: ~/spark-3.0.1-bin...

```
(8433, 0.52708)
(38093, 0.33613)
(9127, 0.39693)

-------------------------------------------
Time: 2020-12-08 17:33:42
-------------------------------------------


-------------------------------------------
Time: 2020-12-08 17:33:44
-------------------------------------------
(14796, 0.34314)
(303357, 0.39324)
(279709, 0.38349)
(214654, 0.43066)
(8422, 0.29583)
(38031, 0.34524)
(9419, 0.30515)

-------------------------------------------
Time: 2020-12-08 17:33:46
-------------------------------------------
```

CONCLUSION:

Analysis of performance of all European soccer players is a dynamic concept, and hence the amount of data generated is absolutely humongous. Thus, big data approach in this topic would be the most apt way of approaching it. We have implemented just one of the numerous perspectives one can have related to this.