Analyzing soccer players

In [1]:

```
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .appName("Analyzing soccer players") \
    .getOrCreate()
```

In [2]:

```
players = spark.read\
               .format("csv")\
               .option("header", "true")\
               .load("../datasets/player.csv")
```

In [3]:

```
players.printSchema()
```

```
root
 |-- id: string (nullable = true)
 |-- player_api_id: string (nullable = true)
 |-- player_name: string (nullable = true)
 |-- player_fifa_api_id: string (nullable = true)
 |-- birthday: string (nullable = true)
 |-- height: string (nullable = true)
 |-- weight: string (nullable = true)
```

In [4]:

```
players.show(5)
```

```
+---+-------------+------------------+------------------+-------------------+------+---
---+
| id|player_api_id|       player_name|player_fifa_api_id|           birthday|height|weigh
t|
+---+-------------+------------------+------------------+-------------------+------+---
---+
|  1|       505942|Aaron Appindangoye|            218353|1992-02-29 00:00:00|182.88|   18
7|
|  2|       155782|   Aaron Cresswell|            189615|1989-12-15 00:00:00|170.18|   14
6|
|  3|       162549|       Aaron Doran|            186170|1991-05-13 00:00:00|170.18|   1
63|
|  4|        30572|     Aaron Galindo|            140161|1982-05-08 00:00:00|182.88|   1
98|
|  5|        23780|      Aaron Hughes|             17725|1979-11-08 00:00:00|182.88|   1
54|
+---+-------------+------------------+------------------+-------------------+------+---
---+
only showing top 5 rows
```

In [5]:

```
player_attributes = spark.read\
                         .format("csv")\
                         .option("header", "true")\
                         .load("../datasets/Player_Attributes.csv")
```

In [6]:

```
player_attributes.printSchema()
```

```
root
 |-- id: string (nullable = true)
 |-- player_fifa_api_id: string (nullable = true)
 |-- player_api_id: string (nullable = true)
 |-- date: string (nullable = true)
 |-- overall_rating: string (nullable = true)
 |-- potential: string (nullable = true)
 |-- preferred_foot: string (nullable = true)
 |-- attacking_work_rate: string (nullable = true)
 |-- defensive_work_rate: string (nullable = true)
 |-- crossing: string (nullable = true)
 |-- finishing: string (nullable = true)
 |-- heading_accuracy: string (nullable = true)
 |-- short_passing: string (nullable = true)
 |-- volleys: string (nullable = true)
 |-- dribbling: string (nullable = true)
 |-- curve: string (nullable = true)
 |-- free_kick_accuracy: string (nullable = true)
 |-- long_passing: string (nullable = true)
 |-- ball_control: string (nullable = true)
 |-- acceleration: string (nullable = true)
 |-- sprint_speed: string (nullable = true)
 |-- agility: string (nullable = true)
 |-- reactions: string (nullable = true)
 |-- balance: string (nullable = true)
 |-- shot_power: string (nullable = true)
 |-- jumping: string (nullable = true)
 |-- stamina: string (nullable = true)
 |-- strength: string (nullable = true)
 |-- long_shots: string (nullable = true)
 |-- aggression: string (nullable = true)
 |-- interceptions: string (nullable = true)
 |-- positioning: string (nullable = true)
 |-- vision: string (nullable = true)
 |-- penalties: string (nullable = true)
 |-- marking: string (nullable = true)
 |-- standing_tackle: string (nullable = true)
 |-- sliding_tackle: string (nullable = true)
 |-- gk_diving: string (nullable = true)
 |-- gk_handling: string (nullable = true)
 |-- gk_kicking: string (nullable = true)
 |-- gk_positioning: string (nullable = true)
 |-- gk_reflexes: string (nullable = true)
```

Player attributes

- **Have values across multiple years**
- **Can be associated with a particular player using the player_api_id column**
- **Different attributes are valuable for different kinds of players i.e strikers, midfields, goalkeepers**

In [7]:

```
players.count() , player_attributes.count()
```

Out[7]:

```
(11060, 183978)
```

In [8]:

```
player_attributes.select('player_api_id')\
                 .distinct()\
                 .count()
```

Out[8]:

```
11060
```

Cleaning Data

In [9]:

```
players = players.drop('id', 'player_fifa_api_id')
players.columns
```

Out[9]:

```
['player_api_id', 'player_name', 'birthday', 'height', 'weight']
```

According to our requirement there are certain traits which we are not at all going to use in this entire program So its better to remove those traits to make our dataset less bulky

In [10]:

```
player_attributes = player_attributes.drop(
    'id',
    'player_fifa_api_id',
    'preferred_foot',
    'attacking_work_rate',
    'defensive_work_rate',
    'crossing',
    'jumping',
    'sprint_speed',
    'balance',
    'aggression',
    'short_passing',
    'potential'
)
player_attributes.columns
```

Out[10]:

```
['player_api_id',
 'date',
 'overall_rating',
 'finishing',
 'heading_accuracy',
 'volleys',
 'dribbling',
 'curve',
 'free_kick_accuracy',
 'long_passing',
 'ball_control',
 'acceleration',
 'agility',
 'reactions',
 'shot_power',
 'stamina',
 'strength',
 'long_shots',
 'interceptions',
 'positioning',
 'vision',
 'penalties',
 'marking',
 'standing_tackle',
 'sliding_tackle',
 'gk_diving',
 'gk_handling',
 'gk_kicking',
 'gk_positioning',
 'gk_reflexes']
```

In [11]:

```
player_attributes = player_attributes.dropna()
players = players.dropna()
```

In [12]:

```
players.count() , player_attributes.count()
```

Out[12]:

```
(11060, 181265)
```

Extract year information into a separate column

In [13]:

```
from pyspark.sql.functions import udf
```

In [14]:

```
year_extract_udf = udf(lambda date: date.split('-')[0])

player_attributes = player_attributes.withColumn(
    "year",
    year_extract_udf(player_attributes.date)
)
```

In [15]:

```
player_attributes = player_attributes.drop('date')
```

In [16]:

```
player_attributes.columns
```

Out[16]:

```
['player_api_id',
 'overall_rating',
 'finishing',
 'heading_accuracy',
 'volleys',
 'dribbling',
 'curve',
 'free_kick_accuracy',
 'long_passing',
 'ball_control',
 'acceleration',
 'agility',
 'reactions',
 'shot_power',
 'stamina',
 'strength',
 'long_shots',
 'interceptions',
 'positioning',
 'vision',
 'penalties',
 'marking',
 'standing_tackle',
 'sliding_tackle',
 'gk_diving',
 'gk_handling',
 'gk_kicking',
 'gk_positioning',
 'gk_reflexes',
 'year']
```

Filter to get all players who were active in the year 2016

In [17]:

```
pa_2016 = player_attributes.filter(player_attributes.year == 2016)
```

In [18]:

```
pa_2016.count()
```

Out[18]:

```
14098
```

In [19]:

```
pa_2016.select(pa_2016.player_api_id)\
       .distinct()\
       .count()
```

Out[19]:

```
5586
```

Find the best striker in the year 2016

- **Consider the scores for finishing, shot_power and acceleration to determine this**
- **There can be more than one entry for a player in the year (multiple seasons, some teams make entries per quarter)**
- **Find the average scores across the multiple records**

In [20]:

```
pa_striker_2016 = pa_2016.groupBy('player_api_id')\
                       .agg({
                           'finishing':"avg",
                           "shot_power":"avg",
                           "acceleration":"avg"
                       })
```

In [21]:

```
pa_striker_2016.count()
```

Out[21]:

```
5586
```

In [22]:

```
pa_striker_2016.show(5)
```

```
+-------------+-----------------+-----------------+---------------+
|player_api_id|   avg(finishing)|avg(acceleration)|avg(shot_power)|
+-------------+-----------------+-----------------+---------------+
|       309726|75.44444444444444|74.11111111111111|           76.0|
|        26112|             53.0|             51.0|           76.0|
|        38433|            68.25|             74.0|           74.0|
|       295060|             25.0|             62.0|           40.0|
|       161396|             29.0|             72.0|           69.0|
+-------------+-----------------+-----------------+---------------+
only showing top 5 rows
```

In [23]:

```
pa_striker_2016 = pa_striker_2016.withColumnRenamed("avg(finishing)","finishing")\
                                 .withColumnRenamed("avg(shot_power)","shot_power")\
                                 .withColumnRenamed("avg(acceleration)","acceleration")
```

Find an aggregate score to represent how good a particular player is

- **Each attribute has a weighing factor**
- **Find a total score for each striker**

In [24]:

```
weight_finishing = 1
weight_shot_power = 2
weight_acceleration = 1

total_weight = weight_finishing + weight_shot_power + weight_acceleration
```

In [27]:

```
strikers = pa_striker_2016.withColumn("striker_grade",
                                      (pa_striker_2016.finishing * weight_finishing + \
                                       pa_striker_2016.shot_power * weight_shot_power+ \
                                       pa_striker_2016.acceleration * weight_acceleratio
n) / total_weight)
```

In [28]:

```
strikers = strikers.drop('finishing',
                        'acceleration',
                        'shot_power'
)
```

In [31]:

```
strikers = strikers.filter(strikers.striker_grade > 70)\
                  .sort(strikers.striker_grade.desc())

strikers.show(10)
```

```
+-------------+-----------------+
|player_api_id|    striker_grade|
+-------------+-----------------+
|        20276|            89.25|
|        37412|             89.0|
|        38817|            88.75|
|        32118|            88.25|
|        31921|             87.0|
|        30834|            86.75|
|       303824|85.10714285714286|
|       129944|             85.0|
|       150565|            84.75|
|       158263|            84.75|
+-------------+-----------------+
only showing top 10 rows
```

Find name and other details of the best strikers

- **The information is present in the *players* dataframe**
- **Will involve a join operation between *players* and *strikers***

In [33]:

```
strikers.count(), players.count()
```

Out[33]:

```
(1609, 11060)
```

Joining dataframes

In [35]:

```
striker_details = players.join(strikers, players.player_api_id == strikers.player_api_id)
```

In [36]:

```
striker_details.columns
striker_details.columns
```

Out[36]:

```
['player_api_id',
 'player_name',
 'birthday',
 'height',
 'weight',
 'player_api_id',
 'striker_grade']
```

In [37]:

```
striker_details.count()
```

Out[37]:

```
1609
```

In [38]:

```
striker_details = players.join(strikers, ['player_api_id'])
```

In [39]:

```
striker_details.show(5)
```

```
+-------------+--------------+-------------------+------+------+-------------+
|player_api_id|   player_name|           birthday|height|weight|striker_grade|
+-------------+--------------+-------------------+------+------+-------------+
|        20276|          Hulk|1986-07-25 00:00:00|180.34|   187|        89.25|
|        37412| Sergio Aguero|1988-06-02 00:00:00|172.72|   163|         89.0|
|        38817|  Carlos Tevez|1984-02-05 00:00:00|172.72|   157|        88.75|
|        32118|Lukas Podolski|1985-06-04 00:00:00|182.88|   183|        88.25|
|        31921|   Gareth Bale|1989-07-16 00:00:00|182.88|   163|         87.0|
+-------------+--------------+-------------------+------+------+-------------+
only showing top 5 rows
```

Broadcast & Join

- **Broadcast the smaller dataframe so it is available on all cluster machines**
- **The data should be small enough so it is held in memory**
- **All nodes in the cluster distribute the data as fast as they can so overall computation is faster**

In [34]:

```
from pyspark.sql.functions import broadcast
```

In [32]:

```
striker_details = players.select(
                                "player_api_id",
                                "player_name"
                                 )\
                  .join(
                        broadcast(strikers),
                        ['player_api_id'],
                        'inner'
                  )
```

In [40]:

```
striker_details = striker_details.sort(striker_details.striker_grade.desc())
```

In [41]:

```
striker_details.show(5)
```

```
+-------------+--------------+-------------------+------+------+-------------+
|player_api_id|   player_name|           birthday|height|weight|striker_grade|
+-------------+--------------+-------------------+------+------+-------------+
|        20276|          Hulk|1986-07-25 00:00:00|180.34|   187|        89.25|
|        37412| Sergio Aguero|1988-06-02 00:00:00|172.72|   163|         89.0|
|        38817|  Carlos Tevez|1984-02-05 00:00:00|172.72|   157|        88.75|
|        32118|Lukas Podolski|1985-06-04 00:00:00|182.88|   183|        88.25|
|        31921|   Gareth Bale|1989-07-16 00:00:00|182.88|   163|         87.0|
+-------------+--------------+-------------------+------+------+-------------+
only showing top 5 rows
```

Accumulators

- **Shared variables which are updated by processes running across multiple nodes**

In [42]:

```
players.count(), player_attributes.count()
```

Out[42]:

```
(11060, 181265)
```

In [44]:

```
players_heading_acc = player_attributes.select('player_api_id',
                                               'heading_accuracy')\
                                       .join(broadcast(players),
                                             player_attributes.player_api_id == players
.player_api_id)
```

In [78]:

```
players_heading_acc.columns
```

Out[78]:

```
['player_api_id',
 'heading_accuracy',
 'player_api_id',
 'player_name',
 'birthday',
 'height',
 'weight']
```

Get player counts by height

In [82]:

```
short_count = spark.sparkContext.accumulator(0)
medium_low_count = spark.sparkContext.accumulator(0)
medium_high_count = spark.sparkContext.accumulator(0)
tall_count = spark.sparkContext.accumulator(0)
```

In [83]:

```
def count_players_by_height(row):
    height = float(row.height)

    if (height <= 175 ):
        short_count.add(1)
    elif (height <= 183 and height > 175 ):
        medium_low_count.add(1)
    elif (height <= 195 and height > 183 ):
        medium_high_count.add(1)
    elif (height > 195) :
        tall_count.add(1)
```

In [84]:

```
players_heading_acc.foreach(lambda x: count_players_by_height(x))
```

In [85]:

```
all_players = [short_count.value,
               medium_low_count.value,
               medium_high_count.value,
               tall_count.value]

all_players
```

Out[85]:

```
[18977, 97399, 61518, 3371]
```

Find the players who have the best heading accuracy

- **Count players who have a heading accuracy above the threshold**
- **Bucket them by height**

In [86]:

```
short_ha_count = spark.sparkContext.accumulator(0)
medium_low_ha_count = spark.sparkContext.accumulator(0)
medium_high_ha_count = spark.sparkContext.accumulator(0)
tall_ha_count = spark.sparkContext.accumulator(0)
```

In [87]:

```
def count_players_by_height_and_heading_accuracy(row, threshold_score):

    height = float(row.height)
    ha = float(row.heading_accuracy)

    if ha <= threshold_score:
        return

    if (height <= 175 ):
        short_ha_count.add(1)
    elif (height <= 183 and height > 175):
        medium_low_ha_count.add(1)
    elif (height <= 195 and height > 183):
        medium_high_ha_count.add(1)
    elif (height > 195) :
        tall_ha_count.add(1)
```

In [88]:

```
players_heading_acc.foreach(lambda x: count_players_by_height_and_heading_accuracy(x, 60
))
```

In [89]:

```
all_players_above_threshold = [short_ha_count.value,
                               medium_low_ha_count.value,
                               medium_high_ha_count.value,
                               tall_ha_count.value]

all_players_above_threshold
```

Out[89]:

```
[3653, 41448, 40270, 1573]
```

Convert to percentages

- **% of players above the threshold heading accuracy for each height bucket**

In [90]:

```
percentage_values = [short_ha_count.value / short_count.value *100,
                     medium_low_ha_count.value / medium_low_count.value *100,
                     medium_high_ha_count.value / medium_high_count.value *100,
                     tall_ha_count.value / tall_count.value *100
                    ]

percentage_values
```

Out[90]:

```
[19.249617958581442, 42.55485169252251, 65.46051562144413, 46.66271136161376]
```

Custom accumulator

In [116]:

```
from pyspark.accumulators import AccumulatorParam

class VectorAccumulatorParam(AccumulatorParam):

    def zero(self, value):
        return [0.0] * len(value)

    def addInPlace(self, v1, v2):
        for i in range(len(v1)):
            v1[i] += v2[i]

        return v1
```

In [117]:

```
vector_accum = sc.accumulator([10.0, 20.0, 30.0], VectorAccumulatorParam())

vector_accum.value
```

Out[117]:

```
[10.0, 20.0, 30.0]
```

In [118]:

```
vector_accum += [1, 2, 3]

vector_accum.value
```

Out[118]:

```
[11.0, 22.0, 33.0]
```

Save data to file

In [96]:

```
pa_2016.columns
```

Out[96]:

```
['player_api_id',
 'overall_rating',
 'finishing',
 'heading_accuracy',
 'volleys',
 'dribbling',
 'curve',
 'free_kick_accuracy',
 'long_passing',
 'ball_control',
 'acceleration',
 'agility',
 'reactions',
 'shot_power',
 'stamina',
 'strength',
 'long_shots',
 'interceptions',
 'positioning',
 'vision',
 'penalties',
 'marking',
 'standing_tackle',
 'sliding_tackle',
 'gk_diving',
 'gk_handling',
 'gk_kicking',
 'gk_positioning',
 'gk_reflexes',
 'year']
```

Save the dataframe to a file

In [101]:

```
pa_2016.select("player_api_id", "overall_rating")\
    .coalesce(1)\
    .write\
    .option("header", "true")\
    .csv("players_overall.csv")
```

In [102]:

```
pa_2016.select("player_api_id", "overall_rating")\
    .write\
    .json("players_overall.json")
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

```
```