

# FOOTBALL ANALYTICS

---

The Poli Football company tasked you with analyzing their data to find key insights and kpis that can help them improve their business. Their database is comprised of the following files:

## Players.csv

Contains information about football players. The file is composed of more than 2,000 rows and has 5 columns:

1. **Surname:** The surname of the football player.
2. **Jersey Number:** The jersey number of the player (i.e. the number associated to shirt that the player wears).
3. **Years:** The number of years the player has been playing football.
4. **Salary:** The salary of the player in euros.
5. **Position:** The position of the player (e.g. forward, midfielder, defender, goalkeeper).

A possible row in the file is:

```
Ronaldo,7,20,1000000,forward
```

## Matches.csv

Contains information about football matches. The file is composed of more than 200,000 rows and has 5 columns:

1. **Match ID:** The unique identifier of the match.
2. **Date:** The date of the match (Format: YYYY-MM-DD).
3. **Home Team:** The name of the team that played at home.
4. **Away Team:** The name of the team that played away.
5. **Result:** The result of the match, labelled as 1 if the home team won, 0 if it was a draw, and -1 if the away team won.

A possible row in the file is:

```
23,2023-10-01,Milan,Inter,1
```

## Goals.csv

Contains information about goals scored in matches. The file is composed of more than 500,000 rows and has 4 columns:

1. **Match ID:** The unique identifier of the match.
2. **Player surname:** The player who scored the goal.
3. **Number of Goals:** The number of goals scored by the player in the match.

4. **Home or Away:** Indicates whether the player scored the goal while playing at home or away (the value is 1 if the player was playing at home, 0 if away).

A possible row in the file is:

```
23,Ronaldo,2,1
```

**NOTE:** The Match ID and the player surname constitute a unique row identifier in the Goals.csv file. As such, is not possible to have two rows with the same Match ID and Player surname.

**NOTE:** Players are only included in the Goals.csv file if they scored at least one goal in the match. So is not possible to have a row where number of goals is 0.

---

## ASSUMPTIONS

---

Assume:

- to have already created both the Spark Context as `sc` and the Spark Session as `spark`.
- that the data type has already been inferred, so no need to convert the data types to integer, float, or string.
- that the data is already cleaned and does not contain any missing values or errors and the format is correct.
- the absence of the header in the CSV files if dealing with RDDs, but the presence of the header if dealing with DataFrames or Spark SQL.

---

## SPARK EXERCISES: Difficulty 'Easy'

---

This exercises are designed to teach and test the basic functionalities and patterns of Spark, such as RDDs, DataFrames, and Spark SQL. They are not meant to be complex or challenging, but rather to help you get familiar with the Spark API and its functionalities.

### Exercise 1:

For each different position of the players, find the average salary of the players that play in that position and earn at least 45,000 euros, store the information about the position and the average salary in the output folder "outputEx1\_1".

Expected output per row: "Position, Average Salary"

```
# Solving the exercise using RDDs
playersRdd = sc.textFile('Players.csv')
filteredPlayersRdd = playersRdd.filter(lambda x: x[3] >= 45000)
positionSalaryKVRdd = filteredPlayersRdd.map(lambda x: (x[4], (x[3], 1)))
positionAverageSalaryRdd = positionSalaryKVRdd.reduceByKey(lambda x, y: (x[0] +
y[0], x[1] + y[1]))
```

```

positionAverageSalaryRdd.mapValues(lambda x: x[0] /
x[1]).saveAsTextFile('outputEx1_1')

# Solving the exercise using DataFrames
playersDf = spark.read.csv('Players.csv', header=True, inferSchema=True)
playersDf = playersDf.filter(playersDf['Salary'] >= 45000)
positionAverageSalaryDf = playersDf.groupBy('Position').agg({'Salary': 'avg'})
positionAverageSalaryDf.write.csv('outputEx1_1', header=True)

# Solving the exercise using Spark SQL
playersDf = spark.read.csv('Players.csv', header=True, inferSchema=True)
playersDf.createOrReplaceTempView('players')
positionAverageSalaryDf = spark.sql("""
    SELECT Position, AVG(Salary) AS AverageSalary
    FROM players
    WHERE Salary >= 45000
    GROUP BY Position
""")
positionAverageSalaryDf.write.csv('outputEx1_1', header=True)

```

## Exercise 2:

For each team, regardless of whether they played at home or away, find the number of matches they played, store the information about the matches played and the team name in the output folder "outputEx1\_2".

Expected output per row: "Team, Number of Matches Played"

```

matchesRdd = sc.textFile('Matches.csv')

# Solving the exercise using RDDs
homeTeamRdd = matchesRdd.map(lambda x: (x[2], 1))
awayTeamRdd = matchesRdd.map(lambda x: (x[3], 1))
teamMatchesRdd = homeTeamRdd.union(awayTeamRdd).reduceByKey(lambda x, y: x + y)
teamMatchesRdd.saveAsTextFile('outputEx1_2')

## More efficient way to do it -> Experience helps!
teamMatchesRdd = matchesRdd.flatMap(lambda x: [(x[2], 1), (x[3], 1)])
teamMatchesRdd.reduceByKey(lambda x, y: x + y).saveAsTextFile('outputEx1_2')

# Solving the exercise using DataFrames
matchesDf = spark.read.csv('Matches.csv', header=True, inferSchema=True)
homeTeamDf = matchesDf.groupBy('Home_Team').agg({'Match_ID':
'count'}).withColumnRenamed('count(Match_ID)', 'NumberOfMatches')
awayTeamDf = matchesDf.groupBy('Away_Team').agg({'Match_ID':
'count'}).withColumnRenamed('count(Match_ID)', 'NumberOfMatches')
teamMatchesDf = homeTeamDf.union(awayTeamDf)
teamMatchesDf.write.csv('outputEx1_2', header=True)

# Solving the exercise using Spark SQL
matchesDf = spark.read.csv('Matches.csv', header=True, inferSchema=True)

```

```

matchesDf.createOrReplaceTempView('matches')
teamMatchesDf = spark.sql("""
    SELECT Team, COUNT(*) AS NumberOfMatches
    FROM (
        SELECT Home_Team AS Team FROM matches
        UNION ALL
        SELECT Away_Team AS Team FROM matches
    ) AS teams
    GROUP BY Team
""")
teamMatchesDf.write.csv('outputEx1_2', header=True)

```

A small comment about the behavior of the `UNION ALL` operator in Spark SQL: it does not remove duplicates, so the correct count is returned. This is the expected behavior for this exercise. If we were to use `UNION` instead, it would remove duplicates, and we would not get the correct result.

## Exercise 3:

Compute the total number of goals per each match, store the information about the match ID and the total number of goals in the output folder "outputEx1\_3".

Expected output per row: "Match ID, Total Number of Goals"

```

# Solving the exercise using RDDs
goalsRdd = sc.textFile('Goals.csv')
matchGoalsRdd = goalsRdd.map(lambda x: (x[0], x[2])).reduceByKey(lambda x, y: x + y)
matchGoalsRdd.saveAsTextFile('outputEx1_3')

# Solving the exercise using DataFrames
goalsDf = spark.read.csv('Goals.csv', header=True, inferSchema=True)
matchGoalsDf = goalsDf.groupBy('Match ID').agg({'Number of Goals': 'sum'}).withColumnRenamed('sum(Number of Goals)', 'Total Number of Goals')
matchGoalsDf.write.csv('outputEx1_3', header=True)

# Solving the exercise using Spark SQL
goalsDf = spark.read.csv('Goals.csv', header=True, inferSchema=True)
goalsDf.createOrReplaceTempView('goals')
matchGoalsDf = spark.sql("""
    SELECT Match ID, SUM(Number of Goals) AS TotalNumberOfGoals
    FROM goals
    GROUP BY Match ID
""")
matchGoalsDf.write.csv('outputEx1_3', header=True)

```

# SPARK EXERCISES: Difficulty 'Medium'

This exercises are designed to teach and test the intermediate level functionalities and patterns of Spark, such as dealing with multiple data sources and formats. They are meant to be relatively straightforward and

focused on specific tasks.

## Exercise 1:

For players whose Position is 'forward', find the total number of goals scored by them in all matches, store the information about the player surname and the total number of goals in the output folder "outputEx2\_1".

Expected output per row: "Player Surname, Total Number of Goals"

```
# Solving the exercise using RDDs
playersRdd = sc.textFile('Players.csv')
goalsRdd = sc.textFile('Goals.csv')
forwardPlayersRdd = playersRdd.filter(lambda x: x[4] == 'forward').map(lambda x: (x[0], None))
forwardGoalsRdd = goalsRdd.map(lambda x: (x[1], int(x[2])))
forwardGoalsRdd = forwardGoalsRdd.join(forwardPlayersRdd).map(lambda x: (x[0], x[1][0]))
totalGoalsRdd = forwardGoalsRdd.reduceByKey(lambda x, y: x + y)
totalGoalsRdd.saveAsTextFile('outputEx2_1')

# Solving the exercise using DataFrames
playersDf = spark.read.csv('Players.csv', header=True, inferSchema=True)
goalsDf = spark.read.csv('Goals.csv', header=True, inferSchema=True)
forwardPlayersDf = playersDf.filter(col('Position') == 'forward')
forwardGoalsDf = goalsDf.join(forwardPlayersDf, goalsDf['Player surname'] == forwardPlayersDf['Surname'], 'inner')
totalGoalsDf = forwardGoalsDf.groupBy('Player surname').agg({'Number of Goals': 'sum'}).withColumnRenamed('sum(Number of Goals)', 'Total Number of Goals')
totalGoalsDf.write.csv('outputEx2_1', header=True)

# Solving the exercise using Spark SQL
playersDf = spark.read.csv('Players.csv', header=True, inferSchema=True)
goalsDf = spark.read.csv('Goals.csv', header=True, inferSchema=True)
playersDf.createOrReplaceTempView('players')
goalsDf.createOrReplaceTempView('goals')
totalGoalsDf = spark.sql("""
    SELECT g.Player surname, SUM(g.Number of Goals) AS Total Number of Goals
    FROM goals g
    JOIN players p
    WHERE p.Position = 'forward' and g.Player surname = p.Surname
    GROUP BY g.Player surname
""")
totalGoalsDf.write.csv('outputEx2_1', header=True)
```

## Exercise 2:

For each match where the home team was the team 'Milan' compute the total number of goals scored by the home team and the total number of goals scored by the away team for match that ended with at least a single goal made by either team, as well as the result of the match, store the information about the match ID, home team goals, away team goals, and result in the output folder "outputEx2\_2".

Expected output per row: "Match ID, Home Team Goals, Away Team Goals, Result"

```

# Solving the exercise using RDDs
matchesRdd = sc.textFile('Matches.csv')
goalsRdd = sc.textFile('Goals.csv')
milanMatchesRdd = matchesRdd.filter(lambda x: x[2] == 'Milan')
milanMatchesKVRdd = milanMatchesRdd.map(lambda x: (x[0], x[5]))
# NOTE: since the result is already provided in the Matches.csv file, we can
directly use it,
# another option would have been to compute the result based on the goals scored
by each team after evaluating them
# however, this would have been less efficient and less straightforward

# Copilot suggests a more complex solution, but we can do it in a more efficient
way,
# moreover, this solution is incorrect as it uses a natural join, thus the teams
are only shown
# if and only if, both teams scored at least one goal, which is incorrect,
# to be correct, we need to implement a full outer join as done in the dataframes
solution
goalsRdd = goalsRdd.map(lambda x: (x[0], (x[2], x[3])))
homeGoalsRdd = goalsRdd.filter(lambda x: x[1][0] == 'True').map(lambda x: (x[0],
x[1][1]))
awayGoalsRdd = goalsRdd.filter(lambda x: x[1][0] == 'False').map(lambda x: (x[0],
x[1][1]))
milanGoalsRdd = milanMatchesKVRdd.join(homeGoalsRdd).join(awayGoalsRdd)
milanGoalsRdd = milanGoalsRdd.map(lambda x: (x[0], x[1][0][1], x[1][1], x[1][0]
[0]))
milanGoalsRdd.saveAsTextFile('outputEx2_2')

# Our solution is more efficient than the one suggested by copilot, as it avoids
unnecessary joins and computations
goalsRdd = goalsRdd.map(lambda x: (x[0], (x[2] if x[3] == 1 else 0, x[2] if x[3]
== 0 else 0)))
milanGoalsRdd = milanMatchesKVRdd.join(goalsRdd).map(lambda x: (x[0], (x[1][1][0],
x[1][1][1], x[1][0])))
milanGoalsRdd.reduceByKey(lambda x, y: (x[0] + y[0], x[1] + y[1],
x[2])).saveAsTextFile('outputEx2_2')

# Solving the exercise using DataFrames
matchesDf = spark.read.csv('Matches.csv', header=True, inferSchema=True)
goalsDf = spark.read.csv('Goals.csv', header=True, inferSchema=True)

# Join with goals
joinedDf = matchesDf.filter(F.col('Home Team') == 'Milan')\
    .join(goalsDf, matchesDf['Match ID'] == goalsDf['Match ID'], 'inner')

# Aggregate home and away goals separately
homeGoalsDf = joinedDf.filter(F.col('Home or Away') == 1) \
    .groupBy('Match ID', 'Result') \
    .agg(F.sum('Number of Goals').alias('HomeTeamGoals'))

awayGoalsDf = joinedDf.filter(F.col('Home or Away') == 0) \

```

```

.groupBy('Match ID') \
.agg(F.sum('Number of Goals').alias('AwayTeamGoals'))

# Join home and away goals
homeGoalsDf.join(awayGoalsDf, 'Match ID', 'full') \
.select('Match ID', 'HomeTeamGoals', 'AwayTeamGoals', 'Result')\
.write.csv('outputEx2_2', header=True)

# Solving the exercise using Spark SQL
matchesDf = spark.read.csv('Matches.csv', header=True, inferSchema=True)
goalsDf = spark.read.csv('Goals.csv', header=True, inferSchema=True)
matchesDf.createOrReplaceTempView('matches')
goalsDf.createOrReplaceTempView('goals')
milanGoalsDf = spark.sql("""
    SELECT m.Match ID,
           SUM(CASE WHEN g.Home or Away = 1 THEN g.Number of Goals ELSE 0 END) AS
HomeTeamGoals,
           SUM(CASE WHEN g.Home or Away = 0 THEN g.Number of Goals ELSE 0 END) AS
AwayTeamGoals,
           m.Result
    FROM matches m
    JOIN goals g
    WHERE m.Home Team = 'Milan' and m.Match ID = g.Match ID
    GROUP BY m.Match ID, m.Result
""")
milanGoalsDf.write.csv('outputEx2_2', header=True)

```

**NOTE:** The solution using Spark SQL is more complex than the one using RDDs, as it requires a more complex query to compute the total number of goals scored by each team and requires the knowledge of the **CASE** mechanic for SQL queries. However, it is still a valid solution and can be used if you prefer to use Spark SQL instead of RDDs.

## Exercise 3:

For every match disputed in the year 2023 where there was a draw different from 0-0, so at least a single goal was made by both teams, compute the total number of goals scored by the home team and the away team, store the information about the match ID, home team goals, away team goals in the output folder "outputEx2\_3".

Expected output per row: "Match ID, Home Team Goals, Away Team Goals"

```

# Solving the exercise using RDDs
matchesRdd = sc.textFile('Matches.csv')
goalsRdd = sc.textFile('Goals.csv')
matchesRdd = matchesRdd.filter(lambda x: x[1].startswith('2023') and x[4] == 0)
matchesKVRdd = matchesRdd.map(lambda x: (x[0], None))
goalsRdd = goalsRdd.map(lambda x: (x[0], (x[2] if x[3] == 1 else 0, x[2] if x[3] == 0 else 0)))
matchesGoalsRdd = matchesKVRdd.join(goalsRdd).map(lambda x: (x[0], (x[1][1][0], x[1][1][1])))
matchesGoalsRdd.reduceByKey(lambda x, y: (x[0] + y[0], x[1] +

```

```

y[1])).saveAsTextFile('outputEx2_3')

# Solving the exercise using DataFrames
matchesDf = spark.read.csv('Matches.csv', header=True, inferSchema=True)
goalsDf = spark.read.csv('Goals.csv', header=True, inferSchema=True)
matchesDf = matchesDf.filter((col('Date').startswith('2023')) & (col('Result') == 0))
matchesGoalsDf = matchesDf.join(goalsDf, matchesDf['Match ID'] == goalsDf['Match ID'], 'inner')\
    .groupBy('Match ID')\
    .agg({'Number of Goals': 'sum', 'Home or Away': 'max'})\
    .withColumnRenamed('sum(Number of Goals)', 'HomeTeamGoals')

spark.udf.register('evaluateGoals', lambda x: x//2, IntegerType())

matchesGoalsDf = matchesGoalsDf.selectExpr('evaluateGoals(HomeTeamGoals) as AwayTeamGoals', 'Match ID', 'evaluateGoals(HomeTeamGoals) as HomeTeamGoals')\
    .select('Match ID', 'HomeTeamGoals', 'AwayTeamGoals')
matchesGoalsDf.write.csv('outputEx2_3', header=True)

# Solving the exercise using Spark SQL
matchesDf = spark.read.csv('Matches.csv', header=True, inferSchema=True)
goalsDf = spark.read.csv('Goals.csv', header=True, inferSchema=True)
matchesDf.createOrReplaceTempView('matches')
goalsDf.createOrReplaceTempView('goals')
drawGoalsDf = spark.sql("""
    SELECT m.Match ID,
           SUM(CASE WHEN g.Home or Away = 1 THEN g.Number of Goals ELSE 0 END) AS HomeTeamGoals,
           SUM(CASE WHEN g.Home or Away = 0 THEN g.Number of Goals ELSE 0 END) AS AwayTeamGoals
    FROM matches m
    JOIN goals g
    WHERE m.Date LIKE '2023%' AND m.Result = 0 AND m.Match ID = g.Match ID
    GROUP BY m.Match ID
""")
drawGoalsDf.write.csv('outputEx2_3', header=True)

```

**NOTE:** The solution using Spark SQL is more complex than the one using RDDs, as it requires a more complex query to compute the total number of goals scored by each team and requires the knowledge of the **CASE** mechanic for SQL queries and the **LIKE** operator and syntax. However, it is still a valid solution and can be used if you prefer to use Spark SQL instead of RDDs.

## SPARK EXERCISES: Difficulty 'Hard'

This exercises are designed to teach and test the hardest level functionalities and patterns of Spark, such as dealing with multiple data sources and formats, computing more complex information, optimizing for efficiency (ex cache, broadcasting). They are meant to be challenging.

### Exercise 1:



Return the name, or the names if more than one, of the home teams that have scored the overall maximum number of goals in a match scored by teams in the home position, store the information about the team name in the output folder "outputEx3\_1".

Expected output per row: "Team Name"

```
# Solving the exercise using RDDs
goalsRdd = sc.textFile('Goals.csv')
homeGoalsRdd = goalsRdd.filter(lambda x: x[3] == 1).map(lambda x: (x[0],
int(x[2])))
# Cache for performance
homeGoalsRdd = homeGoalsRdd.reduceByKey(lambda x, y: x + y).cache()
maxGoals = homeGoalsRdd.values().max()
# broadcast the maximum goals for performance
maxGoals = sc.broadcast(maxGoals)
homeTeamsMaxGoalsRdd = homeGoalsRdd.filter(lambda x: x[1] == maxGoals.value)
# broadcast the matchId associated with home team having the maximum number of
goals
homeTeamsMatchId = homeTeamsMaxGoalsRdd.keys().collect()
homeTeamsMatchId = sc.broadcast(homeTeamsMatchId)

matchesRdd = sc.textFile('Matches.csv')
# broadcast join on the matchId with maximum goals
matchesRdd = matchesRdd.filter(lambda x: x[0] in homeTeamsMatchId.value)
matchesRdd.map(lambda x: x[2]).distinct().saveAsTextFile('outputEx3_1')

# Solving the exercise using DataFrames
goalsDf = spark.read.csv('Goals.csv', header=True, inferSchema=True)
homeGoalsDf = goalsDf.filter(col('Home or Away') == 1).groupBy('Match
ID').agg({'Number of Goals': 'sum'}).withColumnRenamed('sum(Number of Goals)',
'Total Home Goals')
maxGoals = homeGoalsDf.agg({'Total Home Goals': 'max'}).collect()[0]['max(Total
Home Goals)']
homeTeamsMaxGoalsDf = homeGoalsDf.filter(col('Total Home Goals') ==
maxGoals).select('Match ID')

matchesDf = spark.read.csv('Matches.csv', header=True, inferSchema=True)
homeTeamsMaxGoalsDf = homeTeamsMaxGoalsDf.join(matchesDf,
homeTeamsMaxGoalsDf['Match ID'] == matchesDf['Match ID'], 'inner').select('Home
Team')
homeTeamsMaxGoalsDf.distinct().write.csv('outputEx3_1', header=True)

# Solving the exercise using Spark SQL
goalsDf = spark.read.csv('Goals.csv', header=True, inferSchema=True)
matchesDf = spark.read.csv('Matches.csv', header=True, inferSchema=True)
goalsDf.createOrReplaceTempView('goals')
matchesDf.createOrReplaceTempView('matches')

# 2 levels nested query to find the maximum goals scored by home teams
maxGoalsDf = spark.sql("""
    SELECT m.Home Team
    FROM matches m
    WHERE m.match ID IN (
```

```

        SELECT g.Match ID
        FROM goals g
        WHERE g.Home or Away = 1
        GROUP BY g.Match ID
        HAVING SUM(g.Number of Goals) = (
            SELECT MAX(SUM(g.Number of Goals))
            FROM goals g
            WHERE g.Home or Away = 1
            GROUP BY g.Match ID
        )
    )
    """
)

# Alternative solution
maxGoalsDf= spark.sql(
    """
    SELECT m.Home Team
    FROM matches m
    WHERE m.match ID IN (
        SELECT g.Match ID
        FROM goals g
        WHERE g.Home or Away = 1
        GROUP BY g.Match ID
        HAVING SUM(g.Number of Goals) =
            (SELECT MAX(TotGoals)
             FROM (
                 SELECT SUM(g.Number of Goals) as TotGoals
                 FROM goals g
                 WHERE g.Home or Away = 1
                 GROUP BY g.Match ID))
            )
    )
    """

    maxGoalsDf.write.csv('outputEx3_1', header=True)

```

## Exercise 2:

For each player that has a Jersey Number greater than 10 and is a goalkeeper and only for players that have stored at least a single goal in 2023 or 2024, compute the total number of goals scored by them in all matches of 2023 and 2024 in matches where is squad either won or drawn, store the information about the player surname and the total number of goals in the output folder "outputEx3\_2".

Expected output per row: "Player Surname, Total Number of Goals"

```

# Solving the exercise using RDDs
playersRdd = sc.textFile('Players.csv')
goalsRdd = sc.textFile('Goals.csv')
matchesRdd = sc.textFile('Matches.csv')

# Filter players who are goalkeepers and have a Jersey Number greater than 10
goalkeepersRdd = playersRdd.filter(lambda x: x[1] > 10 and x[4] ==
'goalkeeper').map(lambda x: (x[0], None))

```

```
# Filter matches for 2023 and 2024 where the home team or away team won or drew
matchesRdd = matchesRdd.filter(lambda x: (x[1].startswith('2023') or
x[1].startswith('2024')) and (x[4] == 1 or x[4] == 0))

# Note, now there are two possibilities:
# 1. Join goals with matches and then with players
# 2. Join goals with players and then with matches
# The second option is, logically speaking, more efficient as the reduction in
number of rows much more significant
joinedRdd = goalsRdd.map(lambda x: (x[1], (x[0], x[2],
x[3]))).join(goalkeepersRdd) # format is (Player Surname, ((Match ID, Number of
Goals, Home or Away), (None)))
joinedRdd = joinedRdd.map(lambda x: (x[1][0][0], (x[0], x[1][0][1], x[1][0][2])))
# format is (Match ID, (player surname, Number of Goals, Home or Away))

# Now we join with matches
matchesRdd = matchesRdd.map(lambda x: (x[0], None)) # format is (Match ID, None)
joinedRdd = joinedRdd.join(matchesRdd) # format is (Match ID, ((player surname,
Number of Goals, Home or Away), None))

# Now we can just compute the total number of goals scored by each player
totalGoalsRdd = joinedRdd.map(lambda x: (x[1][0][0], x[1][0][1])) # format is
(player surname, Number of Goals)
totalGoalsRdd = totalGoalsRdd.reduceByKey(lambda x, y: x + y) # reduce by key to
get the total number of goals scored by each player
totalGoalsRdd.saveAsTextFile('outputEx3_2')
```

## SPARK EXERCISES: Difficulty 'Very Hard'

This exercises are designed to teach and test the most complex functionalities and patterns of Spark. They are meant to be challenging and similar to the second part of the spark exam.

For the following exercises, assume to have already created both the Spark Context as `sc` and the Spark Session as `spark`.

### Exercise 1: Improving Teams

We want to find the teams that, regardless of them being home or away, have a higher number of wins in 2024 than in 2023. We want to store the information about the team name and the win rate in 2023 and in 2024 in the output folder "outputEx4\_1".

The win rate is defined as the number of wins divided by the number of matches played  $\text{WinRate} = \frac{\text{Wins}}{\text{MatchesPlayed}} = \frac{\text{Wins}}{\text{Wins} + \text{Losses} + \text{Draws}}$ . A match is considered played if the team has either played at home or away. The expected output per row is: "Team Name, Win Rate 2023, Win Rate 2024"

```
# Load the data
matchesRdd = sc.textFile('Matches.csv').filter(lambda x: x[1].startswith('2023')
or x[1].startswith('2024')).cache()
goalsRdd = sc.textFile('Goals.csv')
```

```

# compute the teams having a higher win rate in 2024 than in 2023
def computeWins(squad:str, status:int, home:bool, year2024:bool) ->tuple[str,
tuple[int, int, int]]:
    """
    Compute the number of wins, losses and draws for a team in a given year.

    Args:
        squad (str): The name of the team.
        status (int): 1 if the team played at home, 0 if the team played away, -1
if the team lost.
        home (bool): True if the team played at home, False if the team played
away.
        year2024 (bool): True if the match is in 2024, False if the match is in
2023.

    Returns:
        results (tuple[str, tuple[int, int, int]]): A tuple containing the team
name and a tuple with the number of wins, losses and draws.
    """
    if home:
        if year2024:
            if status == 1: return (squad, [[0, 0], [1,0]]) # Win
            else: return (squad, ([0, 0], [0,1])) # Draw or Loss
        else:
            if status == 1: return (squad, [[1, 0], [0,0]]) # Win
            else: return (squad, ([0, 1], [0,0])) # Draw
    else:
        if year2024:
            if status == -1: return (squad, [[0, 0], [1,0]]) # Win
            else: return (squad, ([0, 0], [0,1])) # Draw or Loss
        else:
            if status == -1: return (squad, [[1, 0], [0,0]]) # Win
            else: return (squad, ([0, 1], [0,0])) # Draw or Loss

# filtering the matches for 2023 and 2024, and computing the wins for each team
teamsRdd = matchesRdd.flatMap(lambda x: [computeWins(x[2], x[5], True,
x[1].startswith('2024')), computeWins(x[3], x[5], False,
x[1].startswith('2024'))])\
    .reduceByKey(lambda x, y: ([x[0][0] + y[0][0], x[0][1] + y[0][1]], [x[1][0] +
y[1][0], x[1][1] + y[1][1]]))\
    .mapValues(lambda x: (x[0][0] / (x[0][1] + x[0][0]), x[1][0] / (x[1][1] + x[1]
[0])))\
    .filter(lambda x: x[1][1] > x[1][0]) # filter the teams that have a higher
win rate in 2024 than in 2023

# We can not assume that the number of teams is small enough to collect (going
with the hardest difficulty)

# Now we want to select the teams that have a higher number of goals in 2024 than
in 2023
matchesRdd2 = matchesRdd.flatMap(lambda x: [(x[2], (x[0], x[1].startswith('2024'),
True)), (x[3], (x[0], x[1].startswith('2024'), False))]).join(teamsRdd)\
    .map(lambda x: ((x[1][0][0], x[1][1][2]), (x[0], x[1][0][1], x[1][1][0], x[1]

```

```
[1][1]))))
# we have created a KV rdd in the form ((Match ID, Home Or Away), (Team Name, Year
2024?, Win Rate 2023, Win Rate 2024))

# Now we want to compute the total number of goals scored by each team in 2023 and
2024
goalsRdd = goalsRdd.map(lambda x: ((x[0], x[3] == 1), x[2]))

# when we join we have ((matchId, HomeOrAway), ((matchId, numberOfGoals),
(teamName, year2024?, winRate2023, winRate2024)))
JoinedRdd = goalsRdd.join(matchesRdd2).map(lambda x: (x[1][1][0], (x[1][0][1] if
x[1][1][1] else -x[1][0][1], x[1][1][2], x[1][1][3])))\
# creating a joined RDD with the (teamName, (numberOfGoalsDiff, Win Rate 2023, Win
Rate 2024)))
FinalRdd = JoinedRdd.reduceByKey(lambda x, y: (x[0] + y[0], x[1], x[2]))\
    .filter(lambda x: x[1][0] > 0) # filter the teams that have a higher number
of goals in 2024 than in 2023

FinalRdd.map(lambda x: (x[0], x[1][1], x[1][2])).saveAsTextFile('outputEx4_1')
```

## Exercise 2: Consecutive Matches

Output to a file the teams, that regardless of the them being home or away, have won two matches in the span of three consecutive days after the 2020, the start of the three days and the total number of goals scored in those matches, the output file is "outputEx4\_2".

The expected output per row is: "Team Name, Starting Date, Total Number of Goals Scored in the Matches"

**NOTE:** For this exercise, you can assume to have the function `prev_date(date:str, days:int) -> str` that returns the date of the previous day, given a date in the format 'YYYY-MM-DD' and the number of days to go back. As example `prev_date('2024-01-01', 1)` would return '2023-12-31'

```
# Load the data
matchesRdd = sc.textFile('Matches.csv').filter(lambda x:x[1] >= '2021/01/01')
goalsRdd = sc.textFile('Goals.csv')

# lets compute the teams --> output will be in the form (teamName, (matchId, date,
Home or Away))
teamsRdd = matchesRdd.flatMap(lambda x: [(x[2], (x[0], x[1], x[5] == 1, True)),
(x[3], (x[0], x[1], x[5]==-1, False))])
# filtering for teams that have won
teamsRdd = teamsRdd.filter(lambda x:x[1][2]).cache() # equivalent to x[1][2] ==
True

# creating the windows
windowTeamRdd = teamsRdd.flatMap(lambda x: [(x[0], prev_date(x[1][1], 2)), (x[1]
[0]+ '-' + str(x[1][2]), 1)), (x[0], prev_date(x[1][1], 1), (x[1][0] + '-' +
str(x[1][2])), 1),
((x[0], x[1][1]), (x[1][0] + '-' + str(x[1][2]), 1))])
# we have created the windows in the form ((teamName, date), (str(matchId - Home
or Away), 1)) where 1 is the counter
```

```

# Now we want to join the windows and check if the team has won two matches in the
span of three consecutive days
# To do so, we know that if the date same team and the same date appears twice, it
means that the team has won two matches in the span of three consecutive days
windowTeamRdd = windowTeamRdd.reduceByKey(lambda x, y: (x[0]+' '+y[0], x[1] +
y[1]))
# Now we have something like ((teamName, date), (matchId - Home or Away,
numberOfWins))

windowTeamRdd = windowTeamRdd.filter(lambda x: x[1][1] >= 2) # filter the teams
that have won two matches in the span of three consecutive days

# We can assume that the number of teams is small enough to collect
# We want to collect as map to do a broadcast join in the form
# matchId : (teamName, date, Home or Away)
def createTeamsMap(row:tuple)->list[tuple]:
    """
    Create a map of teams from the row.
    Args:
        row (tuple): The row to create the map from.
    Returns:
        rows (list[tuple]): A list of tuples in the form (teamName, (matchId,
date, Home or Away))
    """

    match1 = row[1][0].split(",")[0]
    match2 = row[1][0].split(",")[1]
    return [(match1.split("-")[0], (row[0][0], row[0][1], match1.split("-")[1])),
            (match2.split("-")[0], (row[0][0], row[0][1], match2.split("-")[1]))]

teams = windowTeamRdd.flatMap(createTeamsMap).collectAsMap()
teams = sc.broadcast(teams)

# Now we can select the total number of goals scored in those matches
goalsRdd = goalsRdd.filter(lambda x: x[0] in teams.value.keys()) # filter only for
the cared matches

# Now we can create the Rdd in the form (teamName, (date, numGoals))
goalsRdd = goalsRdd.map(lambda x: (teams.value[x[0]][0], (teams.value[x[0]][1],
x[2] if teams.value[x[0]][2] == x[3] else 0)))

# Finally we can reduce the whole thing and store it
goalsRdd.reduceByKey(lambda x,y: (x[0], x[1]+y[1])).saveAsTextFile('outputEx4_2')

```