Q3.

1. Select two columns - games and seasons - and add a column with total goals (sum of home and away goals). Suggestion: use df.withColumn() function –

```
df_game = spark.read.option("header",True).csv("/FileStore/tables/game.csv")
df_game_teams_stats = spark.read.option("header",True).csv("/FileStore/tables/game_teams_stats.csv")
df_team_info = spark.read.option("header",True).csv("/FileStore/tables/team_info.csv")
```

▸ (3) Spark Jobs
▸ df_game: pyspark.sql.dataframe.DataFrame = [game_id: string, season: string ... 14 more fields]
▸ df_game_teams_stats: pyspark.sql.dataframe.DataFrame = [game_id: string, team_id: string ... 13 more fields]
▸ df_team_info: pyspark.sql.dataframe.DataFrame = [team_id: string, franchiseId: string ... 4 more fields]

Command took 12.89 seconds -- by yuchao.wu@mail.utoronto.ca at 7/23/2020, 3:41:05 PM on My Cluster

Cmd 4

```
df_game_with_total_goals = df_game.select(['game_id', 'season', 'away_goals', 'home_goals'])
df_game_with_total_goals = df_game_with_total_goals\
.withColumn('total_goals', df_game_with_total_goals.away_goals+df_game_with_total_goals.home_goals)\
.select('game_id', 'season', 'total_goals')
df_game_with_total_goals.show()
```

▸ (1) Spark Jobs
▸ df_game_with_total_goals: pyspark.sql.dataframe.DataFrame = [game_id: string, season: string ... 1 more fields]

```
+----------+--------+-----------+
|   game_id|  season|total_goals|
+----------+--------+-----------+
|2011030221|20112012|        7.0|
|2011030222|20112012|        5.0|
|2011030223|20112012|        7.0|
|2011030224|20112012|        6.0|
|2011030225|20112012|        4.0|
|2011030411|20112012|        3.0|
|2011030412|20112012|        3.0|
|2011030413|20112012|        4.0|
|2011030414|20112012|        4.0|
|2011030415|20112012|        3.0|
|2011030416|20112012|        7.0|
|2010030311|20102011|        7.0|
|2010030312|20102011|       11.0|
|2010030313|20102011|        2.0|
|2010030314|20102011|        8.0|
|2010030315|20102011|        4.0|
|2010030316|20102011|        9.0|
|2010030317|20102011|        1.0|
|2012030221|20122013|        5.0|
|2012030222|20122013|        7.0|
+----------+--------+-----------+
only showing top 20 rows
```

Command took 1.03 seconds -- by yuchao.wu@mail.utoronto.ca at 7/23/2020, 3:41:26 PM on My Cluster

2. Organize records in ascending order (by season)

```
1 df_game_with_total_goals = df_game_with_total_goals.orderBy(["season"], ascending=True)
2 df_game_with_total_goals.show()
```

▸ (1) Spark Jobs

▸ df_game_with_total_goals: pyspark.sql.dataframe.DataFrame = [game_id: string, season: string ... 1 more fields]

```
+----------+--------+-----------+
|   game_id|  season|total_goals|
+----------+--------+-----------+
|2010030241|20102011|        3.0|
|2010030224|20102011|        6.0|
|2010030242|20102011|        3.0|
|2010030231|20102011|        1.0|
|2010030232|20102011|        3.0|
|2010030311|20102011|        7.0|
|2010030233|20102011|        5.0|
|2010030313|20102011|        2.0|
|2010030234|20102011|        6.0|
|2010030315|20102011|        4.0|
|2010030235|20102011|        7.0|
|2010030317|20102011|        1.0|
|2010030236|20102011|        3.0|
|2010030244|20102011|        7.0|
|2010030211|20102011|        6.0|
|2010030246|20102011|        4.0|
|2010030212|20102011|        5.0|
|2010030223|20102011|        6.0|
|2010030213|20102011|        7.0|
|2010030314|20102011|        8.0|
+----------+--------+-----------+
only showing top 20 rows
```

3. Add a column with an average, min and max total score for each season. Suggestion: use Window function

```
import pyspark.sql.functions as F
from pyspark.sql.types import StringType as string
from pyspark.sql import Window

#Define a window
window = Window.partitionBy("season")

df_game_with_total_goals = df_game_with_total_goals\
.withColumn('avg_season_goals', F.bround(F.avg('total_goals').over(window),2))\
.withColumn('min_season_goals', F.min('total_goals').over(window))\
.withColumn('max_season_goals', F.max('total_goals').over(window))\
.orderBy(["season", "total_goals"], ascending=True)

df_game_with_total_goals.show()
```

▸ (2) Spark Jobs

▸ df_game_with_total_goals: pyspark.sql.dataframe.DataFrame = [game_id: string, season: string ... 4 more fields]

game_id	season	total_goals	avg_season_goals	min_season_goals	max_season_goals
2010020649	20102011	1.0	5.59	1.0	15.0
2010020106	20102011	1.0	5.59	1.0	15.0
2010020571	20102011	1.0	5.59	1.0	15.0
2010030124	20102011	1.0	5.59	1.0	15.0
2010020870	20102011	1.0	5.59	1.0	15.0
2010020843	20102011	1.0	5.59	1.0	15.0
2010020878	20102011	1.0	5.59	1.0	15.0
2010030411	20102011	1.0	5.59	1.0	15.0
2010030121	20102011	1.0	5.59	1.0	15.0
2010020761	20102011	1.0	5.59	1.0	15.0
2010020966	20102011	1.0	5.59	1.0	15.0
2010020937	20102011	1.0	5.59	1.0	15.0
2010021152	20102011	1.0	5.59	1.0	15.0
2010020946	20102011	1.0	5.59	1.0	15.0
2010020736	20102011	1.0	5.59	1.0	15.0
2010030317	20102011	1.0	5.59	1.0	15.0
2010030231	20102011	1.0	5.59	1.0	15.0
2010030415	20102011	1.0	5.59	1.0	15.0
2010030147	20102011	1.0	5.59	1.0	15.0
2010020791	20102011	1.0	5.59	1.0	15.0

only showing top 20 rows

Command took 12.50 seconds -- by yuchao.wu@mail.utoronto.ca at 7/23/2020, 3:41:35 PM on My Cluster

4. Add a column that finds a difference between each game's total score and average for that season. Suggestion: use Window function

```
df_game_with_total_goals = df_game_with_total_goals\
.withColumn('diff_goals_to_avg_season_goals',F.bround(df_game_with_total_goals.total_goals - F.avg('total_goals').over(window),2))\
.orderBy(["season", "total_goals"], ascending=True)

df_game_with_total_goals.show()
```

▸ (4) Spark Jobs
▸ df_game_with_total_goals: pyspark.sql.dataframe.DataFrame = [game_id: string, season: string ... 5 more fields]

game_id	season	total_goals	avg_season_goals	min_season_goals	max_season_goals	diff_goals_to_avg_season_goals
2010020649	20102011	1.0	5.59	1.0	15.0	-4.59
2010020106	20102011	1.0	5.59	1.0	15.0	-4.59
2010020571	20102011	1.0	5.59	1.0	15.0	-4.59
2010030124	20102011	1.0	5.59	1.0	15.0	-4.59
2010020870	20102011	1.0	5.59	1.0	15.0	-4.59
2010020843	20102011	1.0	5.59	1.0	15.0	-4.59
2010020878	20102011	1.0	5.59	1.0	15.0	-4.59
2010030411	20102011	1.0	5.59	1.0	15.0	-4.59
2010030121	20102011	1.0	5.59	1.0	15.0	-4.59
2010020761	20102011	1.0	5.59	1.0	15.0	-4.59
2010020966	20102011	1.0	5.59	1.0	15.0	-4.59
2010020937	20102011	1.0	5.59	1.0	15.0	-4.59
2010021152	20102011	1.0	5.59	1.0	15.0	-4.59
2010020946	20102011	1.0	5.59	1.0	15.0	-4.59
2010020736	20102011	1.0	5.59	1.0	15.0	-4.59
2010030317	20102011	1.0	5.59	1.0	15.0	-4.59
2010030231	20102011	1.0	5.59	1.0	15.0	-4.59
2010030415	20102011	1.0	5.59	1.0	15.0	-4.59
2010030147	20102011	1.0	5.59	1.0	15.0	-4.59
2010020791	20102011	1.0	5.59	1.0	15.0	-4.59

only showing top 20 rows

5. Print top 10 records
Output: Your code and a snapshot of the top 10 records

```
df_game_with_total_goals.show(10)
```

▸ (4) Spark Jobs

game_id	season	total_goals	avg_season_goals	min_season_goals	max_season_goals	diff_goals_to_avg_season_goals
2010020966	20102011	1.0	5.59	1.0	15.0	-4.59
2010030121	20102011	1.0	5.59	1.0	15.0	-4.59
2010020761	20102011	1.0	5.59	1.0	15.0	-4.59
2010030411	20102011	1.0	5.59	1.0	15.0	-4.59
2010030147	20102011	1.0	5.59	1.0	15.0	-4.59
2010030124	20102011	1.0	5.59	1.0	15.0	-4.59
2010020791	20102011	1.0	5.59	1.0	15.0	-4.59
2010030317	20102011	1.0	5.59	1.0	15.0	-4.59
2010030231	20102011	1.0	5.59	1.0	15.0	-4.59
2010030415	20102011	1.0	5.59	1.0	15.0	-4.59

only showing top 10 rows

Q4
1. List all team names (teamName) for teams that played as away team at TD Garden during seasons 2012-2013 and 2013-2014 – 5 points

```
df_away_teams_season_12_and_13_at_td_garden = df_game.select('season', 'away_team_id', 'venue')\
.join(df_team_info.select('team_id','teamName'),df_game.away_team_id == df_team_info.team_id, how='left')\
.drop('team_id')\
.filter((F.col("venue") == 'TD Garden') & ((F.col("season") == 20122013) | (F.col("season") == 20132014)))

df_away_teams_season_12_and_13_at_td_garden.select('teamName').rdd.flatMap(lambda x: x).collect()
```

▸ (1) Spark Jobs

▸ df_away_teams_season_12_and_13_at_td_garden: pyspark.sql.dataframe.DataFrame = [season: string, away_team_id: string ... 2 more fields]

```
Out[15]: ['Rangers',
 'Rangers',
 'Rangers',
 'Penguins',
 'Penguins',
 'Maple Leafs',
 'Maple Leafs',
 'Maple Leafs',
 'Maple Leafs',
 'Blackhawks',
 'Blackhawks',
 'Blackhawks',
 'Red Wings',
 'Red Wings',
 'Red Wings',
 'Canadiens',
 'Canadiens',
 'Canadiens',
 'Canadiens',
 'Hurricanes',
 'Red Wings',
 'Panthers',
```

Command took 1.14 seconds -- by yuchao.wu@mail.utoronto.ca at 7/23/2020, 3:45:37 PM on My Cluster

2. How many unique teams are on the list? - 5 points

```
len(df_away_teams_season_12_and_13_at_td_garden.select('teamName').drop_duplicates().collect())
```

▸ (1) Spark Jobs

```
Out[74]: 29
```

Command took 3.23 seconds -- by yuchao.wu@mail.utoronto.ca at 7/22/2020, 2:25:45 PM on My Cluster

Q.5 Create a data frame that have three columns: Student, Subject, Score. Student name includes: Demar and Kawhi. Subject includes math, history, science. Score can be any number between 0 and 100. The result should be a dataframe with 6 rows and 3 columns.

```
df_student_subject_score = sqlContext.createDataFrame(
    [('Demar', 'math'),\
     ('Demar', 'history'),\
     ('Demar', 'science'),\
     ('Kawhi', 'math'),\
     ('Kawhi', 'history'),\
     ('Kawhi', 'science')], ("Student", "Subject"))
```

▸ df_student_subject_score: pyspark.sql.dataframe.DataFrame = [Student: string, Subject: string]

```
# Add the score column and generate the random number
df_student_subject_score = df_student_subject_score.withColumn('Score', F.bround(F.rand()*100, 0))
```

▸ df_student_subject_score: pyspark.sql.dataframe.DataFrame = [Student: string, Subject: string ... 1 more fields]

Command took 0.06 seconds -- by yuchao.wu@mail.utoronto.ca at 7/23/2020, 10:42:35 AM on My Cluster

Use user define function to add a category column to the data frame, if score higher than 70, category show 'Good', if score higher than 50 but lower than 70, category should show 'Ok', if score lower than 50 then show 'Not good'. – 20 points

```
from pyspark.sql.types import StringType
# Create a category function
def create_score_category(score):
  if score >70:
    return "Good"
  elif score < 50:
    return "Not Good"
  else:
    return "OK"

# Define a UDF function
spark_udf = F.udf(create_score_category,StringType())

df_student_subject_score = df_student_subject_score.withColumn('Score_Category', spark_udf('Score'))
#Show the result
display(df_student_subject_score)
```

▸ (3) Spark Jobs

▸ df_student_subject_score: pyspark.sql.dataframe.DataFrame = [Student: string, Subject: string ... 2 more fields]

	Student	Subject	Score	Score_Category
1	Demar	math	27	Not Good
2	Demar	history	95	Good
3	Demar	science	52	OK
4	Kawhi	math	14	Not Good
5	Kawhi	history	50	OK
6	Kawhi	science	6	Not Good

Showing all 6 rows.

Command took 0.58 seconds -- by yuchao.wu@mail.utoronto.ca at 7/23/2020, 10:42:36 AM on My Cluster

Q6. Create a function that when input a number n returns a list of prime numbers between 1 and n. Test your function with number 17. – 10 points

```
def check_prime(number):
  # Prime number should be greater 1
  if number <=1:
    return False
  # 2 is a prime number
  if number ==2:
    return True
  # Other even numbers aren't
  if not number & 1:
    return False
  # check whether number is divisible into it's square root
  for x in range(3,int(number**0.5)+1,2):
    if number % x ==0:
      return False
    # if we get this far we are good
  return True

from six.moves import xrange
def get_prime_numbers_from_1_to_n(n):
    # create a set of numbers to 100,000
    numbers = sc.parallelize(xrange(n+1))
    print(numbers.filter(check_prime).collect())

```

```
Command took 0.03 seconds -- by yuchao.wu@mail.utoronto.ca at 7/23/2020, 5:38:58 PM on My Cluster
```

```
Cmd 25
```

```
get_prime_numbers_from_1_to_n(17)
```

```
▸ (1) Spark Jobs
[2, 3, 5, 7, 11, 13, 17]
Command took 0.18 seconds -- by yuchao.wu@mail.utoronto.ca at 7/23/2020, 5:39:53 PM on My Cluster
```

Bibliography

[1] A. Anthony, "How Apache Spark's Transformations And Action works…," 12 July 2017. [Online]. Available: https://medium.com/@aristo_alex/how-apache-sparks-transformations-and-action-works-ceb0d03b00d0.