assignment

March 17, 2023

1 CS5540, Group 1 - Apache Spark Assignment

This is the submission document for our programming assignment over Apache Spark.

The submission was written as a Jupyter notebook but will be exported to a PDF for submission. We can provide the GitHub repo or the original Jupyter notebook if requested.

[ ]:
```
%pip install pyspark
```

```
Requirement already satisfied: pyspark in ./env/lib/python3.10/site-packages
(3.3.2)
Requirement already satisfied: py4j==0.10.9.5 in ./env/lib/python3.10/site-
packages (from pyspark) (0.10.9.5)
Note: you may need to restart the kernel to use updated packages.
```

[ ]:
```
from pyspark.sql import SparkSession
from pyspark import SparkContext
from pyspark.sql.functions import col, split, explode, lower, trim, avg, expr,␣
 ↪regexp_replace, struct
from pyspark.sql.types import StructType, StructField, StringType
import re
```

[ ]:
```
spark = SparkSession.builder\
    .appName("my-spark-app")\
    .config("spark.sql.catalogImplementation", "hive")\
    .getOrCreate()
sc = SparkContext.getOrCreate()
```

1.1 Table of Contents

1.2 Team Members

This assignment was completed by the following team members (Group 1):

- Odai Athamneh
- Scott Brunton
- Ayushman (Jeet) Das
- Koti Paruchuri
- Varshith Thota

1.3 Question 1

Question 1 is as follows:

> Given file (`/data/shakespeare-1.txt`) contains the scenes from Shakespeare's plays. You may use this file as an input dataset to identify the following notes for a student of Classical Drama.

[ ]:
```
# tokenize input text file and clean the word column
df = spark.read.text("data/shakespeare-1.txt")

df = df.select(explode(split(col("value"), " ")).alias("word"))
df = df.select(lower(trim(col("word"))).alias("word"))

df.show(5, truncate=False)
```

```
+------+
|word  |
+------+
|this  |
|is    |
|the   |
|100th|
|etext|
+------+
only showing top 5 rows
```

1.3.1 Question 1.1

The question reads as follows:

> How many different countries are mentioned in the whole file? (Regardless of how many times a single country is mentioned, this country only contributes as a single entry).

To address this question, we need a dataset of country names. We are using the `country-list.csv` file provided by the professor. The file contains 211 entries.

The caveat to this approach is that the dataset may not contain all countries, such as: - Countries that no longer exist - Countries that are misspelled in the original Shakespearean text - Countries where the name or spelling has changed over time

Addressing this issue is beyond the scope of this assignment and would likely require some degree of manual curation.

[ ]:
```
# load countries dataframe and clean the country column
countries = spark.read.csv("data/country-list.csv", header=False)
countries = countries.select(lower(trim("_c0")).alias("country"))

countries.show(5, truncate=False)
countries.count()
```

```
+---------------+
|country        |
+---------------+
|afghanistan    |
|albania        |
|algeria        |
|american samoa|
|andorra        |
+---------------+
only showing top 5 rows
```

[ ]:
```
211
```

Now that we have our list of countries, we can use a simple `.join()` to find the number of countries mentioned in the Shakespearean text. We will use the `Country` column as our key and perform an inner join with the Shakespearean text. This will return a new DataFrame with only the rows that have a match in both DataFrames. We can then use `.count()` to get the number of rows in the resulting DataFrame.

[ ]:
```
# perform join after converting both columns to lowercase and trimming the␣
  ↪country column
unique_countries = df.join(countries, df.word == countries.country, "inner").
  ↪select("country").distinct()
unique_countries.show(5, truncate=False)

print("Number of unique countries in the text file: {}".format(unique_countries.
  ↪count()))
```

```
+--------+
|country|
+--------+
|greece |
|poland |
|austria|
|guinea |
|france |
+-------+
only showing top 5 rows

Number of unique countries in the text file: 22
```

1.3.2 Question 1.2

The question reads as follows:

> Compute the total number of times any country is mentioned. (This is different from the question1.1, since in this calculation, if a country is mentioned three times, then it contributes three times).

[ ]:
```
country_mentions = df.join(countries, df.word == countries.country, "inner").
 ↪select("country").groupBy("country").count().orderBy("count",␣
 ↪ascending=False)

# show the sum of the count column
country_mentions.agg({"count": "sum"}).show(5)
```

```
+----------+
|sum(count)|
+----------+
|       392|
+----------+
```

1.3.3 Question 1.3

The question reads as follows:

> Determine the most popular countries. (It can be done by finding the three countries mentioned the most).

This is fairly straightforward, and we can reuse the `country_mentions` variable from the last question.

[ ]:
```
country_mentions.show(3)
```

```
+--------+-----+
| country|count|
+--------+-----+
|  france|  149|
| england|  128|
|scotland|   24|
+--------+-----+
only showing top 3 rows
```

1.3.4 Question 1.4

The question reads as follows:

> After exploring the dataset, now calculate how many times specific countries are mentioned. (For example, how many times was France mentioned?)

The code to do this is below and reuses the `country_mentions` variable again. Note that, by default, a Jupyter notebook will only show the first 20 rows of the resulting DataFrame. We use `.show(1000)` to ensure all rows are listed.

```
[ ]: country_mentions.show(1000)
```

```
+---------+-----+
|  country|count|
+---------+-----+
|   france|  149|
|  england|  128|
| scotland|   24|
|    egypt|   15|
|    wales|   15|
|    italy|   12|
|   cyprus|   10|
|  denmark|   10|
|   greece|    6|
|     oman|    4|
|   norway|    3|
|  austria|    3|
|    syria|    3|
|    spain|    2|
|   poland|    1|
|   guinea|    1|
|  iceland|    1|
|  germany|    1|
|palestine|    1|
|   turkey|    1|
|   russia|    1|
|  armenia|    1|
+---------+-----+
```

1.3.5 Question 1.5

The question reads as follows:

> Finally, what is the average number of times a country is mentioned?

```
[ ]: total_countries = countries.count()
     total_mentions = country_mentions.agg({"count": "sum"}).collect()[0][0]

     avg_mentions = (total_countries/total_mentions) * 100
     avg_words = (total_mentions/df.count()) * 100

     print("Percentage of countries mentioned in the text file: {}%".
      ↪format(round(avg_mentions, 2)))
     print("Percentage of words in the text file that are countries: {}%".
      ↪format(round(avg_words, 2)))
```

```
Percentage of countries mentioned in the text file: 53.83%
Percentage of words in the text file that are countries: 0.03%
```

1.4 Question 2

We must first define the schema. The schema must be defined and created in this case (rather then let Spark infer as is generally preferred) because there are additonal columns needed for a handful of rows. If we don't manually define the schema here, these rows would be rejected when Spark attempts to infer the schema.

```
[ ]: schema = StructType([
         StructField("_c0",StringType(),True), \
         StructField("_c1",StringType(),True), \
         StructField("_c2",StringType(),True), \
         StructField("_c3",StringType(),True), \
         StructField("_c4",StringType(),True), \
         StructField("_c5",StringType(),True), \
         StructField("_c6",StringType(),True), \
         StructField("_c7",StringType(),True), \
         StructField("_c8",StringType(),True), \
         StructField("_c9",StringType(),True), \
         StructField("_c10",StringType(),True), \
         StructField("_c11",StringType(),True), \
         StructField("_c12",StringType(),True), \
         StructField("_c13",StringType(),True), \
         StructField("_c14",StringType(),True), \
         StructField("_c15",StringType(),True), \
         StructField("_c16",StringType(),True), \
         StructField("_c17",StringType(),True), \
       ])

     df = spark.read.schema(schema).csv('data/wx-data-1.txt', header=False)
     df.show(5)
```

```
+----------------+-------+-------+-------+-------+-------+--------+----+----+----
+----+----+----+----+----+----+----+----+
|             _c0|    _c1|    _c2|    _c3|    _c4|    _c5|     _c6| _c7| _c8|
_c9|_c10|_c11|_c12|_c13|_c14|_c15|_c16|_c17|
+---------------+-------+-------+-------+-------+-------+--------+----+----+----
+----+----+----+----+----+----+----+----+
|1419408000\t0R1|Dn=038D|Dm=079D|Dx=120D|Sn=2.8M|Sm=6.0M|
Sx=8.1M|null|null|null|null|null|null|null|null|null|null|null|
|1419408001\t0R1|Dn=038D|Dm=074D|Dx=120D|Sn=2.8M|Sm=6.2M|
Sx=8.8M|null|null|null|null|null|null|null|null|null|null|null|
|1419408002\t0R1|Dn=038D|Dm=071D|Dx=120D|Sn=2.8M|Sm=6.5M|
Sx=9.2M|null|null|null|null|null|null|null|null|null|null|null|
|1419408003\t0R1|Dn=038D|Dm=067D|Dx=120D|Sn=2.8M|Sm=6.8M|
Sx=9.5M|null|null|null|null|null|null|null|null|null|null|null|
|1419408004\t0R1|Dn=038D|Dm=062D|Dx=081D|Sn=2.8M|Sm=7.2M|Sx=10.0M|null|null|null
|null|null|null|null|null|null|null|null|
+---------------+-------+-------+-------+-------+-------+--------+----+----+----
+----+----+----+----+----+----+----+----+
only showing top 5 rows
```

1.4.1 Question 2.1

The question reads as follows:

> Find out how many days, weather was in ideal conditions? (Where Ideal condition means 8.5   Sm   9.0 M and 060D Dm   065D)

Create a staging df. In staging df split the ID from the category reading type(Readtype). Drop unsplit column before providing results to new df.

[ ]:
```
dfStaging = df.withColumn('Id', split(df['_c0'], "\t").getItem(0)).
↪withColumn('ReadType', split(df['_c0'], '\t').getItem(1)).drop("_c0")
```

Clean up the data and create a df of only 0R1 type readings. Split the Measurement type from the value of the measurement. Measurement type becomes column header. Measurement value and measurement unit symbol are added to temp df. Remove measurement unit symbol, leaving only measurement value.

[ ]:
```
dfTemp1 = dfStaging.where(dfStaging.ReadType == '0R1').withColumn(
    'Dn', split(dfStaging['_c1'], "=").getItem(1)).withColumn(
        'Dm', split(dfStaging['_c2'], "=").getItem(1)).withColumn(
            'Dx', split(dfStaging['_c3'], "=").getItem(1)).withColumn(
                'Sn', split(dfStaging['_c4'], "=").getItem(1)).withColumn(
                    'Sm', split(dfStaging['_c5'], "=").getItem(1)).withColumn(
                        'Sx', split(dfStaging['_c6'], "=").getItem(1))

df0R1 = dfTemp1.select(dfTemp1.Id, regexp_replace(dfTemp1.Dn, "D", "").
↪alias('Dn'),regexp_replace(
    dfTemp1.Dm, "D", "").alias('Dm'), regexp_replace(
        dfTemp1.Dx, "D", "").alias('Dx'), regexp_replace(
            dfTemp1.Sn, "M", "").alias('Sn'), regexp_replace(
                    dfTemp1.Sm, "M", "").alias('Sm'), regexp_replace(
                         dfTemp1.Sx, "M", "").alias('Sx'))

df0R1.show(5)
```

```
+----------+---+---+---+---+---+----+
|        Id| Dn| Dm| Dx| Sn| Sm|  Sx|
+----------+---+---+---+---+---+----+
|1419408000|038|079|120|2.8|6.0| 8.1|
|1419408001|038|074|120|2.8|6.2| 8.8|
|1419408002|038|071|120|2.8|6.5| 9.2|
|1419408003|038|067|120|2.8|6.8| 9.5|
|1419408004|038|062|081|2.8|7.2|10.0|
+----------+---+---+---+---+---+----+
only showing top 5 rows
```

Clean up the data and create a df of only 0R2 type readings. Split the Measurement type from the value of the measurement. Measurement type becomes column header. Measurement value and measurement unit symbol are added to temp df. Remove measurement unit symbol, leaving only measurement value.

0R2,Ta=14.4C,Ua=26.6P,Pa=889.6H

[ ]:
```
dfTemp2 = dfStaging.where(dfStaging.ReadType == '0R2').withColumn(
     'Ta', split(dfStaging['_c1'], "=").getItem(1)).withColumn(
          'Ua', split(dfStaging['_c2'], "=").getItem(1)).withColumn(
               'Pa', split(dfStaging['_c3'], "=").getItem(1))

df0R2 = dfTemp2.select(dfTemp2.Id, regexp_replace(dfTemp2.Ta, "C", "").
 ↪alias('Ta'),regexp_replace(
     dfTemp2.Ua, "P", "").alias('Ua'), regexp_replace(
          dfTemp2.Pa, "H", "").alias('Pa') )


df0R2.show(5)
```

```
+----------+----+----+-----+
|        Id|  Ta|  Ua|   Pa|
+----------+----+----+-----+
|1419408006|13.9|28.5|889.9|
|1419408016|13.9|28.5|889.9|
|1419408026|13.9|28.4|889.9|
|1419408036|13.9|28.3|889.7|
|1419408046|13.9|28.3|889.9|
+----------+----+----+-----+
only showing top 5 rows
```

Clean up the data and create a df of only 0R5 type readings. Split the Measurement type from the value of the measurement. Measurement type becomes column header. Measurement value and measurement unit symbol are added to temp df. Remove measurement unit symbol, leaving only measurement value.

0R5,Th=13.3C,Vh=0.0#,Vs=25.6V,Vr=3.5

```
[ ]: dfTemp3 = dfStaging.where(dfStaging.ReadType == '0R5').withColumn(
         'Th', split(dfStaging['_c1'], "=").getItem(1)).withColumn(
             'Vh', split(dfStaging['_c2'], "=").getItem(1)).withColumn(
                 'Vs', split(dfStaging['_c3'], "=").getItem(1)).withColumn(
                     'Vr', split(dfStaging['_c4'], "=").getItem(1))

     df0R5 = dfTemp3.select(dfTemp3.Id, regexp_replace(dfTemp3.Th, "C", "").
      ↪alias('Th'),regexp_replace(
         dfTemp3.Vh, "#", "").alias('Vh'), regexp_replace(
             dfTemp3.Vs, "V", "").alias('Vs'), regexp_replace(
                 dfTemp3.Vr, "V", "").alias('Vr'))

     df0R5.show(5)
```

```
+----------+----+---+----+-----+
|        Id|  Th| Vh|  Vs|   Vr|
+----------+----+---+----+-----+
|1419408023|13.1|0.0|25.6|3.516|
|1419408083|13.1|0.0|25.5|3.518|
|1419408143|12.9|0.0|25.6|3.516|
|1419408203|13.3|0.0|25.6|3.516|
|1419408263|13.1|0.0|25.6|3.518|
+----------+----+---+----+-----+
only showing top 5 rows
```

Clean up the data and create a df of only 0R0 type readings. Split the Measurement type from the value of the measurement. Measurement type becomes column header. Measurement value and measurement unit symbol are added to temp df. Remove measurement unit symbol, leaving only measurement value.

Dn=058D,Dm=062D,Dx=064D,Sn=9.7M,Sm=10.3M,Sx=10.8M,Ta=14.4C,Ua=26.7P,Pa=889.6H, Rc=76.74M,Rd=34084s,Ri=0.0M,Hc=0.0M,Hd=0s,Hi=0.0M,Vs=25.5V,Vr=3.516V

```
[ ]: dfTemp4 = dfStaging.where(dfStaging.ReadType == '0R0').withColumn(
         'Dn', split(dfStaging['_c1'], "=").getItem(1)).withColumn(
             'Dm', split(dfStaging['_c2'], "=").getItem(1)).withColumn(
                 'Dx', split(dfStaging['_c3'], "=").getItem(1)).withColumn(
                     'Sn', split(dfStaging['_c4'], "=").getItem(1)).withColumn(
                         'Sm', split(dfStaging['_c5'], "=").getItem(1)).withColumn(
                             'Sx', split(dfStaging['_c6'], "=").getItem(1)).
      ↪withColumn(
         'Ta', split(dfStaging['_c7'], "=").getItem(1)).withColumn(
             'Ua', split(dfStaging['_c8'], "=").getItem(1)).withColumn(
                 'Pa', split(dfStaging['_c9'], "=").getItem(1)).withColumn(
                     'Rc', split(dfStaging['_c10'], "=").getItem(1)).withColumn(
                         'Rd', split(dfStaging['_c11'], "=").getItem(1)).withColumn(
                             'Ri', split(dfStaging['_c12'], "=").getItem(1)).
      ↪withColumn(
         'Hc', split(dfStaging['_c13'], "=").getItem(1)).withColumn(
             'Hd', split(dfStaging['_c14'], "=").getItem(1)).withColumn(
                 'Hi', split(dfStaging['_c15'], "=").getItem(1)).withColumn(
                     'Vs', split(dfStaging['_c16'], "=").getItem(1)).withColumn(
                         'Vr', split(dfStaging['_c17'], "=").getItem(1))

     df0R0 = dfTemp4.select(dfTemp4.Id, regexp_replace(dfTemp4.Dn, "D", "").
      ↪alias('Dn'),regexp_replace(
         dfTemp4.Dm, "D", "").alias('Dm'), regexp_replace(
             dfTemp4.Dx, "D", "").alias('Dx'), regexp_replace(
                 dfTemp4.Sn, "M", "").alias('Sn'), regexp_replace(
                     dfTemp4.Sm, "M", "").alias('Sm'), regexp_replace(
                         dfTemp4.Sx, "M", "").alias('Sx'), regexp_replace(
         dfTemp4.Ta, "C", "").alias('Ta'),regexp_replace(
             dfTemp4.Ua, "P", "").alias('Ua'), regexp_replace(
                 dfTemp4.Pa, "H", "").alias('Pa'),  regexp_replace(
                     dfTemp4.Rc, "M", "").alias('Rc'),  regexp_replace(
                         dfTemp4.Rd, "s", "").alias('Rd'),  regexp_replace(
                             dfTemp4.Ri, "M", "").alias('Ri'),  regexp_replace(
         dfTemp4.Hc, "M", "").alias('Hc'),  regexp_replace(
             dfTemp4.Hd, "s", "").alias('Hd'),  regexp_replace(
                 dfTemp4.Hi, "M", "").alias('Hi'),  regexp_replace(
                     dfTemp4.Vs, "V", "").alias('Vs'),  regexp_replace(
                         dfTemp4.Vr, "V", "").alias('Vr'))

     df0R0.show(5)
```

```
+----------+---+---+---+---+---+----+----+-----+-----+-----+---+---+---
+----+------+
|        Id| Dn| Dm| Dx| Sn| Sm|  Sx|  Ta|   Ua|   Pa|   Rc|   Rd| Ri| Hc| Hd|
Hi|  Vs|   Vr|
+----------+---+---+---+---+---+----+----+-----+-----+-----+---+---+---
+----+------+
|1419408024|057|064|069|8.8|9.6|10.3|13.9|28.5|889.9|76.74|34084|0.0|0.0|
0|0.0|25.6|3.516|
|1419408084|056|062|066|8.4|8.8| 9.3|13.9|28.0|889.8|76.74|34084|0.0|0.0|
0|0.0|25.5|3.518|
|1419408144|058|060|065|8.4|9.1|10.2|14.0|27.6|889.8|76.74|34084|0.0|0.0|
0|0.0|25.6|3.516|
|1419408204|060|064|068|7.4|8.5| 8.9|14.2|27.8|889.9|76.74|34084|0.0|0.0|
0|0.0|25.6|3.516|
|1419408264|056|057|060|7.5|8.1| 8.7|14.1|27.0|889.9|76.74|34084|0.0|0.0|
0|0.0|25.6|3.518|
+----------+---+---+---+---+---+----+----+----+-----+-----+-----+---+---+---
+----+-----+
only showing top 5 rows
```

Will need to use distinct count of id for count of days. Id will be used to join all read df as it is duplicated accross all readtypes.

(Where Ideal condition means 8.5   Sm   9.0 M and 060D Dm   065D)

[ ]:
```
dfOR0.createOrReplaceTempView('OR0')
dfOR1.createOrReplaceTempView('OR1')
dfOR2.createOrReplaceTempView('OR2')
dfOR5.createOrReplaceTempView('OR5')

spark.sql('select count(*) as IdealDays from OR1 as a left join OR0 as b on a.
 ↪Id = b.Id ' +
             ' left join OR2 as c on a.Id = c.Id ' +
             ' left join OR5 as d on a.Id = d.Id ' +
             ' where (a.Sm < 8.5 or b.Sm > 9.0) and ' +
             ' (b.Sm < 8.5 or a.Sm > 9.0) and ' +
             ' (a.Dm <= 065 or b.Dm <= 065)').show(5)
```

```
23/03/17 11:12:37 WARN package: Truncated the string representation of a plan
since it was too large. This behavior can be adjusted by setting
'spark.sql.debug.maxToStringFields'.
+----------+
|IdealDays|
+----------+
|       182|
+----------+
```

1.4.2 Question 2.2

The question reads as follows:

> Find out what the are minimum values for Sn and Dn?

Fortunately, our cleanup work from the previous question makes this question very easy to answer. We can simply use `.min()` on the `Sn` and `Dn` columns.

[ ]:
```
spark.sql('select min(a.Dn) as MinDn from OR1 as a union all select min(b.Dn)␣
 ↪from OR0 as b').show(10)
spark.sql('select min(a.Sn) as MinSn from OR1 as a union all select min(b.Sn)␣
 ↪from OR0 as b').show(10)
```

```
+-----+
|MinDn|
+-----+
|  000|
|  000|
+-----+

+-----+
|MinSn|
+-----+
|  0.0|
|  0.0|
+-----+
```

1.4.3 Question 2.3

The question reads as follows:

> Find out what the are maximum values for Sx are and Dx?

As above, so below. We can use `.max()` on the `Sx` and `Dx` columns.

```
[ ]: spark.sql('select max(a.Sx) as MaxSx from OR1 as a union all select max(b.Sx)␣
     ↪from OR0 as b').show(10)
     spark.sql('select max(a.Dx) as MaxDx from OR1 as a union all select max(b.Dx)␣
     ↪from OR0 as b').show(10)
```

```
+-----+
|MaxSx|
+-----+
|  9.9|
|  9.9|
+-----+

+-----+
|MaxDx|
+-----+
|  359|
|  359|
+-----+
```

1.5 Question 3

```
[ ]: df = spark.read.csv('data/daily_weather-2.csv', header=True)
     df.show(5)
```

```
+------+-------------+-----------+------------------+---------------+----------
---------+---------------+------------------+
|number|air_pressure.|  air_temp.|avg_wind_direction.|avg_wind_speed.|max_wind_d
irection.|max_wind_speed.|relative_humidity.|
+------+-------------+-----------+------------------+---------------+----------
---------+---------------+------------------+
|     0|       918.06|     74.822|             271.1|      2.0803542|
295.4|      2.8632832|             42.42|
|     1|  917.3476881|71.40384263|       101.9351794|    2.443009216|
140.4715485|    3.533323602|       24.32869729|
|     2|       923.04|     60.638|                51|     17.0678522|
63.7|     22.1009672|              8.9|
|     3|  920.5027512|70.13889487|       198.8321327|    4.337363056|
211.2033412|     5.19004536|       12.18910187|
|     4|       921.16|     44.294|             277.8|      1.8566602|
136.5|      2.8632832|             92.41|
+------+-------------+-----------+------------------+---------------+----------
---------+---------------+------------------+
only showing top 5 rows
```

```
[ ]: new_cols=(column.replace('.', '') for column in df.columns)
     df = df.toDF(*new_cols)
     df = df.drop("number")
     df.show(5)
```

```
+------------+-----------+------------------+--------------+------------------+-
-------------+-----------------+
|air_pressure|   air_temp|avg_wind_direction|avg_wind_speed|max_wind_direction|m
ax_wind_speed|relative_humidity|
+------------+-----------+------------------+--------------+------------------+-
-------------+-----------------+
|      918.06|     74.822|             271.1|     2.0803542|             295.4|
2.8632832|             42.42|
| 917.3476881|71.40384263|       101.9351794|   2.443009216|       140.4715485|
3.533323602|      24.32869729|
|      923.04|     60.638|                51|    17.0678522|              63.7|
22.1009672|              8.9|
| 920.5027512|70.13889487|       198.8321327|   4.337363056|       211.2033412|
5.19004536|      12.18910187|
|      921.16|     44.294|             277.8|     1.8566602|             136.5|
2.8632832|             92.41|
+------------+-----------+------------------+--------------+------------------+-
```

```
-------------+-----------------+
only showing top 5 rows
```

1.5.1 Question 3.1

The question reads as follows:

> Count the number of days where all parameters have difference of ± 2

Informally pivoting the column headers to the column typeMeasurement, these will be used later to create a relation to the DataTable. Column name, avgs, lower and upper ranges will be collected to create a new dataframe.

```
[ ]: dfAvg = df.select( avg('air_pressure'), avg('air_temp'),␣
      ↪avg('avg_wind_direction'), avg('avg_wind_speed'),
                                  avg('max_wind_direction'), avg('max_wind_speed'),␣
      ↪avg('relative_humidity'))
     #set to one to offset row used for intilization of list var data
     x = 1
     data = [(0,"ab",0.0,0.0,0.0)]

     for col in df.columns:
       typOfMeasurment = x
       avrg = dfAvg.collect()[0][x -1]
       upprRng = avrg + 2
       lowrRng = avrg -2

       data.append([x,col,avrg,upprRng,lowrRng])
       x += 1

     dfAvgSummary = spark.createDataFrame(schema =␣
      ↪['index','typeMeasurement','avg','upprRng','lowrRng'], data = data)
     dfAvgSummary.show(5)
```

```
+-----+------------------+------------------+------------------+----------------
--+
|index|   typeMeasurement|               avg|           upprRng|
lowrRng|
+-----+------------------+------------------+------------------+----------------
--+
|    0|                ab|               0.0|               0.0|
0.0|
|    1|      air_pressure| 918.8825513141026| 920.8825513141026|
916.8825513141026|
|    2|          air_temp| 64.93300141293575| 66.93300141293575|
62.93300141293575|
|
3|avg_wind_direction|142.23551070020164|144.23551070020164|140.23551070020164|
|    4|    avg_wind_speed| 5.508284242259157|
7.508284242259157|3.5082842422591574|
+-----+------------------+------------------+------------------+----------------
--+
only showing top 5 rows
```

Removes record that was only used to intilize the column data types.

```
[ ]: # clean DF and remove values that were used to initilize df columen types
     dfAvgSummaryC = dfAvgSummary.filter(dfAvgSummary.index != 0)
     dfAvgSummaryC.show(5)
```

```
+-----+------------------+------------------+------------------+----------------
--+
|index|   typeMeasurement|               avg|           upprRng|
lowrRng|
+-----+------------------+------------------+------------------+----------------
--+
|    1|      air_pressure| 918.8825513141026| 920.8825513141026|
916.8825513141026|
|    2|          air_temp| 64.93300141293575| 66.93300141293575|
62.93300141293575|
|
3|avg_wind_direction|142.23551070020164|144.23551070020164|140.23551070020164|
|    4|    avg_wind_speed| 5.508284242259157|
7.508284242259157|3.5082842422591574|
|
5|max_wind_direction|148.95351796495402|150.95351796495402|146.95351796495402|
+-----+------------------+------------------+------------------+----------------
--+
only showing top 5 rows
```

For Loop to iterate through each column in the datatable. Once the column is selected the correlating lower and upper range are also retrived. The lower and upper range are type cast to perform regular expression to only retreive the ranges(numeric values). Column, lower and upper range are appended to query to display number of records outside of the +-2 range.

```
[ ]: df.createOrReplaceTempView('dataTable')
     dfAvgSummaryC.createOrReplaceTempView('avgTable')

     exp = r"[^0-9.]"

     dataTableCol = df.columns
     avgCols = dfAvg.columns

     for dCol in dataTableCol:
```

```
 upprRng = dfAvgSummaryC.select(dfAvgSummaryC.upprRng).filter(dfAvgSummaryC.
↪typeMeasurement.contains(dCol)).collect()
 lowrRng = dfAvgSummaryC.select(dfAvgSummaryC.lowrRng).filter(dfAvgSummaryC.
↪typeMeasurement.contains(dCol)).collect()

 x =''.join(map(str, upprRng))
 upr = re.sub(exp, '', x)
 y =''.join(map(str, lowrRng))
 lowr = re.sub(exp, '', y)

 spark.sql("select count(dt." + dCol + ") from dataTable as dt" +
                     " where dt." + dCol + " > " + upr +
                     " or dt." + dCol + " < " + lowr ).show(5)
```

```
+--------------------+
|count(air_pressure)|
+--------------------+
|                 606|
+--------------------+

+----------------+
|count(air_temp)|
+----------------+
|             951|
+----------------+

+--------------------------+
|count(avg_wind_direction)|
+--------------------------+
|                      1084|
+--------------------------+

+----------------------+
|count(avg_wind_speed)|
+----------------------+
|                   754|
+----------------------+

+--------------------------+
|count(max_wind_direction)|
+--------------------------+
|                      1082|
+--------------------------+

+----------------------+
|count(max_wind_speed)|
+----------------------+
|                   826|
+----------------------+

+-------------------------+
|count(relative_humidity)|
+-------------------------+
|                     1041|
+-------------------------+
```

1.5.2 Question 3.2

The question reads as follows:

> Count number of days where max_wind_speed and ang_wind_speed has difference more than 5.

```
[ ]: df = spark.read.csv('data/daily_weather-2.csv', header=True)
     df.show(5)
```

```
+------+-------------+-----------+-------------------+---------------+----------
---------+---------------+------------------+
|number|air_pressure.|  air_temp.|avg_wind_direction.|avg_wind_speed.|max_wind_d
irection.|max_wind_speed.|relative_humidity.|
+------+-------------+-----------+-------------------+---------------+----------
---------+---------------+------------------+
|     0|       918.06|     74.822|              271.1|      2.0803542|
295.4|      2.8632832|             42.42|
|     1|  917.3476881|71.40384263|        101.9351794|    2.443009216|
140.4715485|    3.533323602|       24.32869729|
|     2|       923.04|     60.638|                 51|     17.0678522|
63.7|     22.1009672|               8.9|
|     3|  920.5027512|70.13889487|        198.8321327|    4.337363056|
211.2033412|     5.19004536|       12.18910187|
|     4|       921.16|     44.294|              277.8|      1.8566602|
136.5|      2.8632832|             92.41|
+------+-------------+-----------+-------------------+---------------+----------
---------+---------------+------------------+
only showing top 5 rows
```

```
[ ]: new_cols=(column.replace('.', '') for column in df.columns)
     df = df.toDF(*new_cols)

     df.createOrReplaceTempView("dataT")

     spark.sql("select  count(*) as 5DaysGreater " +
               " from dataT dt " +
               " where dt.max_wind_speed - dt.avg_wind_speed  > 5").show(5)
```

```
+------------+
|5DaysGreater|
+------------+
|          20|
+------------+
```