



ΠΑΝΕΠΙΣΤΗΜΙΟ  
ΠΑΤΡΩΝ  
UNIVERSITY OF PATRAS

Πολυτεχνική Σχολή

Τμήμα Μηχανικών Η/Υ & Πληροφορικής

---

# ΕΡΓΑΣΙΑ 2 ΣΤΙΣ ΤΕΧΝΟΛΟΓΙΕΣ ΑΠΟΚΕΝΤΡΩΜΕΝΩΝ ΣΥΣΤΗΜΑΤΩΝ - ARACHE SPARK

---

<Σεϊτανίδης Νικόλαος >

A.M. <1072553>

<Οικονομόπουλος Ιωάννης >

A.M. <1072582>

Πάτρα, <2024>

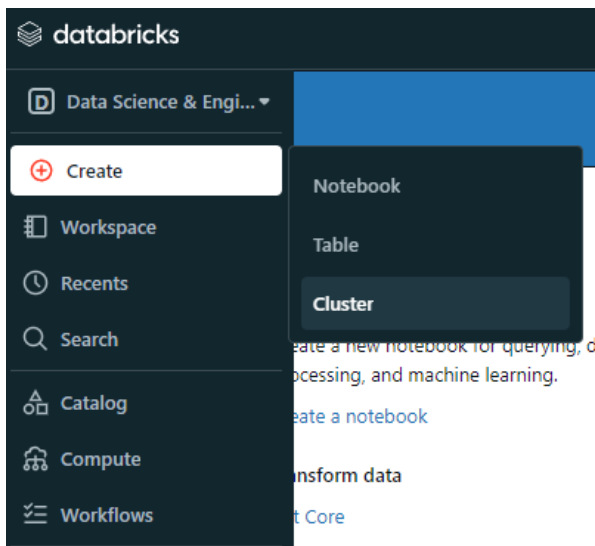
## Περιεχόμενα

Σύνδεση στο Databricks .....	3
Θέμα 1 .....	7
1.1 .....	9
1.2 .....	13
1.3 .....	15
1.4 .....	17
Θέμα 2 .....	20
2.1 .....	20
2.2 .....	24
2.3 .....	26
2.4 .....	29
Θέμα 3 .....	31
3.1 .....	32
3.2 .....	38
3.3 .....	42
3.4 .....	45

## Σύνδεση στο Databricks

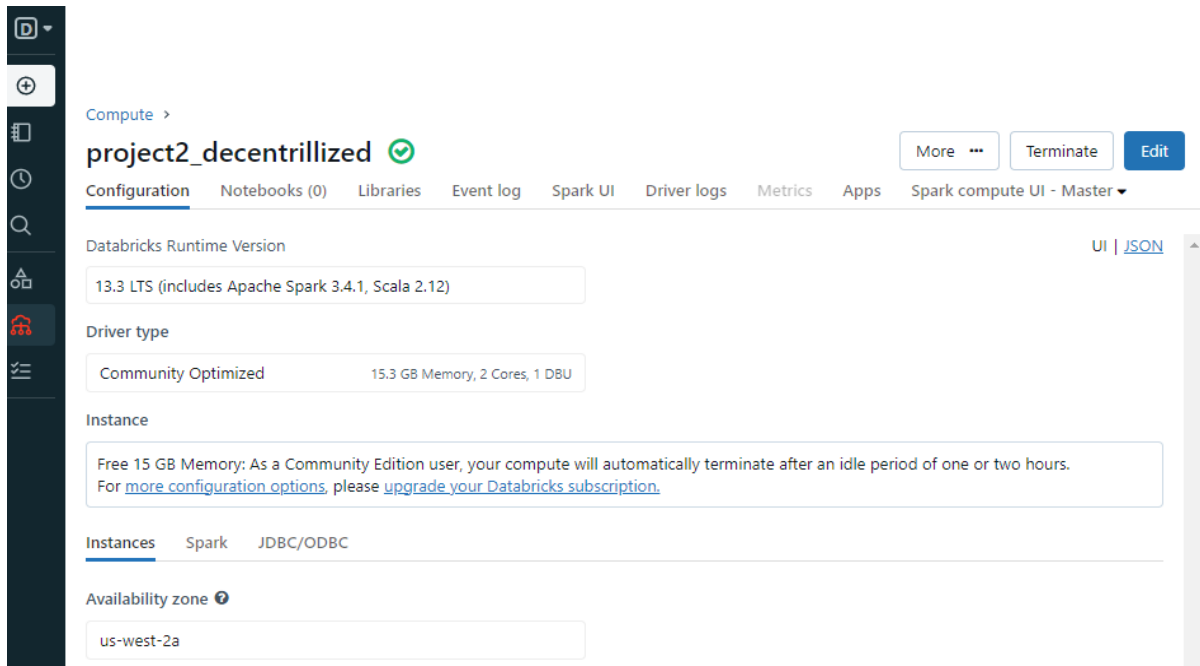
Αρχικά επιλέγουμε να υλοποιήσουμε το project σε PySpark στο περιβάλλον του Databricks. Δημιουργούμε έναν λογαριασμό στο Databricks Community Edition ώστε να είναι δωρεάν και χωρίς καμία μελλοντική συνδρομή. Αφού συνδεθούμε στο προφίλ μας, για να τρέξουμε κώδικα στο Databricks χρειάζεται να γίνει η δημιουργία ενός cluster όπως απεικονίζεται παρακάτω:

- Βήμα 1: Επιλέγουμε την δημιουργία Cluster στο Databricks.



**Εικόνα 1:** Επιλογή δημιουργίας Cluster

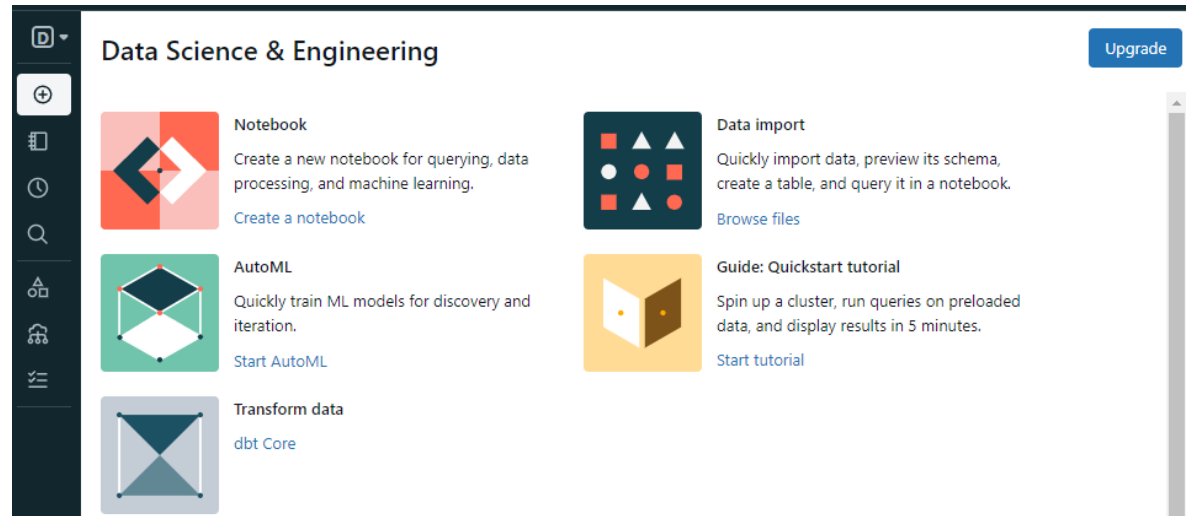
- Βήμα 2: Δίνουμε όνομα και επιλέγουμε την έκδοση 13.3 LTS.



Εικόνα 2: Δημιουργημένο cluster

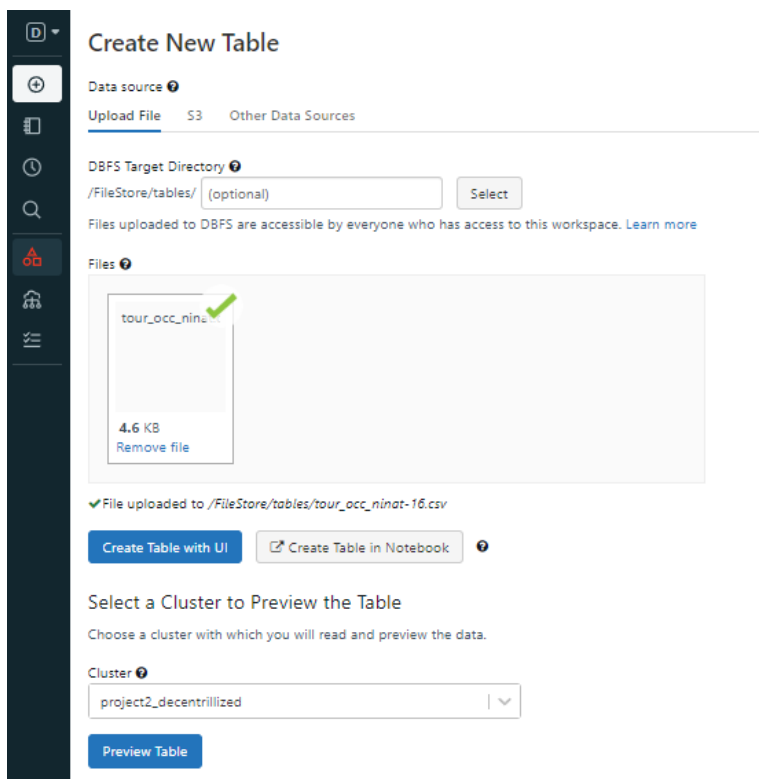
Να σημειωθεί πως ο χρόνος ζωής ενός cluster στην αδράνεια είναι 1 ώρα, οπότε στην περίπτωση που δεν επεξεργαστούμε κάποιο αρχείο ή κώδικα για 1 ώρα, τότε ο cluster γίνεται terminated. Παρόλα αυτά, μπορούμε να κάνουμε restart έναν cluster που είχαμε δημιουργήσει. Ο λόγος για την δημιουργία του cluster στο Databricks είναι γιατί με αυτόν τον τρόπο το ίδιο το Databricks συνδέει το συγκεκριμένο cluster με VMs ώστε να μπορούμε να τρέξουμε κώδικα και να επεξεργαστούμε δεδομένα. Όσον αφορά τα δεδομένα που θέλουμε να επεξεργαστούμε, πρέπει πρώτα να κάνουμε import τα αρχεία που θέλουμε να επεξεργαστούμε. Για παράδειγμα, στο Θέμα 3<sup>ο</sup>, θέλουμε να επεξεργαστούμε ένα αρχείο csv (αναλύουμε παρακάτω την επεξεργασία του αρχείου για το συγκεκριμένο ερώτημα). Οπότε μεταβαίνουμε στο home page και ακολουθούμε τις παρακάτω ενέργειες:

- Βήμα 1º: Data import->Browse files και επιλέγουμε το αρχείο που θέλουμε να επεξεργαστούμε.



Εικόνα 3: Αρχική οθόνη Databricks από την οποία επιλέγουμε το Data Import

- Βήμα 2º: Choose file-> to Create Table With UI-> Select Cluster (τον cluster που δημιουργήσαμε παραπάνω)->Preview Table-> Choose First row is header(στην περίπτωση μας τουλάχιστον)->Create table.



Εικόνα 4: Δημιουργία νέου table από ανεβασμένο αρχείο

### Specify Table Attributes

Specify the Table Name, Database and Schema to add this to the data UI for other users to access

Table Name	Table Preview				
<input type="text" value="tour_occ_ninat_16_csv"/>	<input type="text" value="GEO/TIME"/>	<input type="text" value="Belgium"/>	<input type="text" value="Bulgaria"/>	<input type="text" value="Czech Republic"/>	<input type="text" value="Denmark"/>
Create in Database	<input type="text" value="STRING"/>	<input type="text" value="STRING"/>	<input type="text" value="STRING"/>	<input type="text" value="STRING"/>	<input type="text" value="STRING"/>
<input type="text" value="default"/>					
File Type					
<input type="text" value="CSV"/>					
Column Delimiter					
<input type="text" value="."/>					
<input checked="" type="checkbox"/> First row is header					
<input type="checkbox"/> Infer schema					
<input type="checkbox"/> Multi-line					
<input type="button" value="Create Table"/>					
<input type="button" value="Create Table in Notebook"/>					

2006	16,039,090	11,944,694	20,090,348	9,453,026
2007	16,271,311	12,006,786	20,610,186	9,327,579
2008	16,360,702	11,791,454	19,987,022	8,918,197
2009	15,451,017	9,460,922	17,746,893	8,299,403
2010	16,169,676	10,547,112	18,365,947	8,981,992
2011	16,723,867	12,426,723	19,424,839	9,491,137

Εικόνα 5: Ενδεικτική μορφή του δημιουργημένου πίνακα

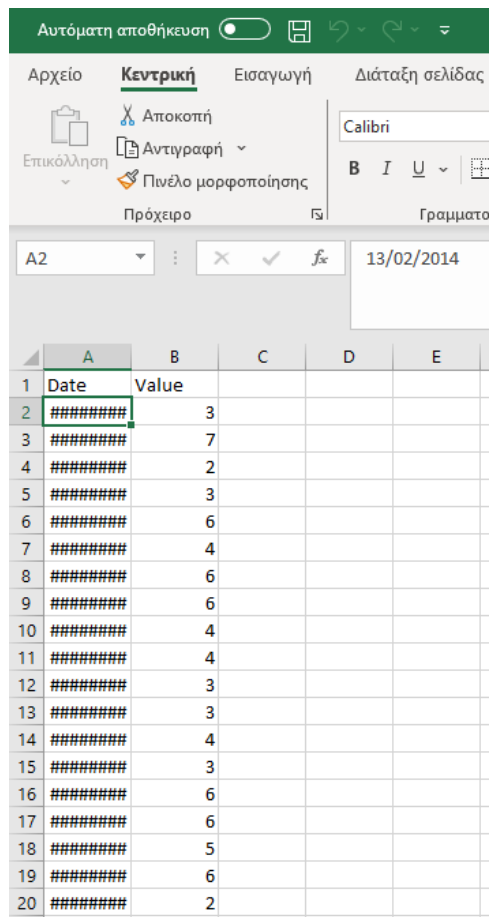
Σε επίπεδο κώδικα, προκειμένου να εισάγουμε τον πίνακα που δημιουργήσαμε, γράφουμε την εξής εντολή:

```
Όνομα_dataframe =  
spark.read.option("header", True).csv("/FileStore/tables/όνομα_αρχείου_που_εισαγάγαμε.csv")
```

Σε αυτό το σημείο να σημειωθεί πως η τοποθεσία του αρχείου πλέον είναι το `‘/FileStore/tables/’` μετά την δημιουργία του πίνακα. Πλέον είμαστε έτοιμοι για επεξεργασία του αρχείου.

## Θέμα 1

Αρχικά για την υλοποίηση του θέματος χρησιμοποιήθηκαν δύο python scripts τα οποία μετέτρεψαν τα αρχεία tempm.txt, hum.txt σε .csv αρχεία ενώ ταυτόχρονα αλλάζουν τη μορφή τους από αυτή που υπάρχει στα .txt. Συγκεκριμένα στα .txt κάθε γραμμή είναι μια μέρα με το «,» να διαχωρίζει κάθε διαφορετική μέτρηση για την ίδια μέρα σε διαφορετικές ώρες. Ενώ η μετατροπή οδηγεί στην μορφή όπου διατηρείται η ημερομηνία κάθε μέτρησης, η ίδια η μέτρηση και εισάγονται στο .csv με κάθε μέτρηση και την αντίστοιχη ημερομηνία καταγραφής της να βρίσκονται ανά γραμμή. Τα python scripts βρίσκονται στον αντίστοιχο φάκελο με τα ερωτήματα του θέματος και περιέχουν σχόλια για κατανόηση του κώδικα για να αποφευχθεί η χρήση του κώδικα στο παρόν κείμενο. Για τον ίδιο λόγο δεν παραθέτουμε ούτε ενδεικτικά στιγμιότυπα καθώς θα χρειαζόταν εξήγηση για όλο τον κώδικα των scripts. Παρακάτω βρίσκεται ενδεικτικό στιγμιότυπο από το tempm.csv όπου φαίνεται η μορφή των αρχείων:



	A	B	C	D	E
1	Date	Value			
2	13/02/2014	3			
3	13/02/2014	7			
4	13/02/2014	2			
5	13/02/2014	3			
6	13/02/2014	6			
7	13/02/2014	4			
8	13/02/2014	6			
9	13/02/2014	6			
10	13/02/2014	4			
11	13/02/2014	4			
12	13/02/2014	3			
13	13/02/2014	3			
14	13/02/2014	4			
15	13/02/2014	3			
16	13/02/2014	6			
17	13/02/2014	6			
18	13/02/2014	5			
19	13/02/2014	6			
20	13/02/2014	2			

Εικόνα 6: Παράδειγμα εγγραφών στο αρχείο tempm.csv

**ΣΗΜΕΙΩΣΗ:** Τα παραγόμενα .csv στο πρόγραμμα Excel εμφανίζουν στα πεδία Date τη μορφή «#####» ωστόσο όπως φαίνεται και στην γραμμή τύπων η τιμή του κάθε πεδίο είναι σε κανονική μορφή ημερομηνίας «dd/mm/YYYY»

**ΣΗΜΕΙΩΣΗ:** Στα στιγμιότυπα κώδικα παρακάτω υπάρχουν και ενδεικτικά αποτελέσματα κάθε εντολής τα οποία δεν υπάρχουν στην τελική μορφή των .py αρχείων και χρησιμοποιούνται μόνο για την σύνταξη της αναφοράς και για debugging.



## 1.1

Αρχείο που περιέχει το κώδικα του ερωτήματος 1 για το θέμα 1: Thema\_1\_Query\_1.py

Η μέθοδος επίλυσης που ακολουθήσαμε αφορούσε την εύρεση των ελάχιστων και μέγιστων τιμών θερμοκρασίας για κάθε ημέρα/εγγραφή στο .csv αρχείο καθώς το ερώτημα είναι να βρεθούν οι ημέρες με τη θερμοκρασία να κυμαίνεται από 18 έως 22 βαθμούς οπότε η ελάχιστη τιμή είναι 18 και η μέγιστη 22.

Ξεκινώντας δημιουργούμε το dataframe χρησιμοποιώντας την spark.read.option για να διαβάσουμε τον πίνακα που δημιουργήθηκε στο databricks από το tempm.csv αρχείο:

```
Cmd 2

1  #Making the dataframe from the table that was created from the file
2
3  df_temp = spark.read.option("header",True).csv("/FileStore/tables/tempm-5.csv")
4  df_temp.printSchema()
5  df_temp.show(5)

▶ (2) Spark Jobs
▶ df_temp: pyspark.sql.dataframe.DataFrame = [Date: string, Value: string]

root
|-- Date: string (nullable = true)
|-- Value: string (nullable = true)

+-----+-----+
|      Date|Value|
+-----+-----+
|2014-02-13|  3.0|
|2014-02-13|  7.0|
|2014-02-13|   2|
|2014-02-13|   3|
|2014-02-13|   6|
+-----+-----+
only showing top 5 rows
```

Εικόνα 7: Δημιουργία dataframe από το tempm.csv

Στη συνέχεια μετατρέπουμε τα πεδία Date, Value από string type σε Date type και double type αντίστοιχα χρησιμοποιώντας την εντολή withColumn της spark.

```
Cmd 3

1 #Transform column date from String type to Date type
2
3 df_temp = df_temp.withColumn('Date',to_date('Date'))

▸ df_temp: pyspark.sql.dataframe.DataFrame = [Date: date, Value: string]
Command took 0.18 seconds -- by johnekonon@gmail.com at 2/1/2024, 4:42:45 μ.μ. on Project

Cmd 4

1 #Transform column value from String type to Double type
2
3 df_temp = df_temp.withColumn("Value",col("Value").cast('double'))
4 df_temp.printSchema()
5 df_temp.show(5)

▸ (1) Spark Jobs
▸ df_temp: pyspark.sql.dataframe.DataFrame = [Date: date, Value: double]

root
|-- Date: date (nullable = true)
|-- Value: double (nullable = true)

+-----+-----+
|      Date|Value|
+-----+-----+
|2014-02-13|  3.0|
|2014-02-13|  7.0|
|2014-02-13|  2.0|
|2014-02-13|  3.0|
|2014-02-13|  6.0|
+-----+-----+
only showing top 5 rows
```

Εικόνα 8: Μετατροπή των τύπων των πεδίων Date,Value

Έπειτα εντοπίζουμε τις ελάχιστες και μέγιστες τιμές για κάθε ημερομηνία και κατασκευάζουμε ένα dataframe το οποίο έχει τα πεδία ημερομηνίας, ελάχιστη τιμή και μέγιστη τιμή.

```
Cmd 5

1  #Extract minimum and maximum values for each day and produce a dataframe that contains them
2
3  df_min = df_temp.groupBy("Date").min("Value")
4  df_min = df_min.withColumn("Minimum",col("min(Value)"))
5  df_max = df_temp.groupBy("Date").max("Value")
6  df_max = df_max.withColumn("Maximum",col("max(Value)"))
7  df = (df_min.join(df_max, "Date").orderBy("Date")).drop("min(Value)","max(Value)")
8
9  df.printSchema()
10 df.show(5)

▶ (4) Spark Jobs
▶ df_min: pyspark.sql.dataframe.DataFrame = [Date: date, min(Value): double ... 1 more field]
▶ df_max: pyspark.sql.dataframe.DataFrame = [Date: date, max(Value): double ... 1 more field]
▶ df: pyspark.sql.dataframe.DataFrame = [Date: date, Minimum: double ... 1 more field]

root
|-- Date: date (nullable = true)
|-- Minimum: double (nullable = true)
|-- Maximum: double (nullable = true)

+-----+-----+-----+
|      Date|Minimum|Maximum|
+-----+-----+-----+
|2014-02-13|    2.0|    7.0|
|2014-02-14|    1.0|    6.0|
|2014-02-15|    3.0|    9.0|
|2014-02-16|    3.0|    7.0|
|2014-02-17|    1.0|    8.0|
+-----+-----+-----+
only showing top 5 rows
```

**Εικόνα 9:** Δημιουργία dataframe με τις ελάχιστες και μέγιστες τιμές για κάθε ημέρα

Από το dataframe που δημιουργήθηκε παραπάνω διατηρούμε τις εγγραφές/ημερομηνίες που έχουν ελάχιστη τιμή μεγαλύτερη από 18 βαθμούς και μέγιστη τιμή μικρότερη από 22 βαθμούς. Τέλος δημιουργούμε την μεταβλητή q\_count η οποία διατηρεί την τιμή των εγγραφών που βρίσκονται στο τελικό dataframe και εμφανίζουμε τα αποτελέσματα.

```
Cmd 6

1 #Find the dates with minimum value over 18 and maximum below 22 and count how many there are
2
3 df_q = df.filter(df["Minimum"] >= 18).filter(df["Maximum"] <= 22)
4 q_count = df_q.count()

▶ (5) Spark Jobs
▶ df_q: pyspark.sql.dataframe.DataFrame = [Date: date, Minimum: double ... 1 more field]
Command took 1.71 seconds -- by johnekonon@gmail.com at 2/1/2024, 4:50:32 μ.μ. on Project

Cmd 7

1 #Print the results and the count
2
3 print("Dataframe for dates with temperature between 18 and 22 celcius: ")
4 df_q.printSchema()
5 df_q.show()
6
7 print("Number of days with temperature between 18 and 22 celcius: ",q_count)

▶ (3) Spark Jobs

Dataframe for dates with temperature between 18 and 22 celcius:
root
|-- Date: date (nullable = true)
|-- Minimum: double (nullable = true)
|-- Maximum: double (nullable = true)

+----+-----+-----+
|Date|Minimum|Maximum|
+----+-----+-----+

Number of days with temperature between 18 and 22 celcius: 0
```

**Εικόνα 10:** Τελικό dataframe που περιέχει τα αποτελέσματα και εμφάνιση αποτελεσμάτων

Όπως φαίνεται και στα αποτελέσματα δεν υπάρχουν ημέρες/ημερομηνίες με θερμοκρασία από 18 έως 22 βαθμούς. Αυτό οφείλεται κυρίως στις ελάχιστες τιμές όπου δεν υπάρχουν μέρες που να έχουν ελάχιστη τιμή ανώτερη από 18 βαθμούς παρότι υπάρχουν ημέρες με μέγιστη θερμοκρασία έως και μεγαλύτερη των 22 βαθμών.

## 1.2

Αρχείο που περιέχει το κώδικα του ερωτήματος 2 για το θέμα 1: Thema\_1\_Query\_2.py

Η μεθοδολογία που ακολουθήσαμε στο ερώτημα 2 είναι παρόμοια με του ερωτήματος 1 έως το σημείο που δημιουργείται το dataframe που περιέχει τις ελάχιστες και μέγιστες τιμές θερμοκρασίας για κάθε ημέρα. Από αυτό το dataframe δημιουργούνται δύο νέα τα όπου το ένα διατηρεί τις πιο κρύες ημέρες (δηλαδή τις ελάχιστες τιμές με την μικρότερη τιμή πρώτη) και το άλλο τις πιο ζεστές (δηλαδή τις μέγιστες τιμές με την μεγαλύτερη πρώτη).

Τα παρακάτω στιγμιότυπα αφορούν το σημείο του κώδικα που είναι διαφορετικό από τον κώδικα του ερωτήματος 1. Συγκεκριμένα οι δύο κώδικες διαφοροποιούνται μετά τη δημιουργία του dataframe που περιέχει ελάχιστες και μέγιστες τιμές.

Για τη δημιουργία κάθε dataframe διατηρούμε είτε την στήλη με τις ελάχιστες τιμές είτε την λίστα με τις μέγιστες τιμές και περιορίζουμε στις πρώτες 10 εγγραφές. Ανάλογα με το dataframe που δημιουργούμε γίνεται και ταξινόμηση βάσει των ελάχιστων ή μέγιστων τιμών αντίστοιχα ώστε να αντληθούν οι 10 πιο κρύες ή πιο ζεστές μέρες.

Cmd 6

```
1 #Make two new dataframes one for the coldest days and one for the hottest
2 #The two new dfs are generated through the dataframe that contains the minimum and maximum for each date
3 #The coldest are ordered by minimum and the hottest are ordered by maximum in descending order
4 #By limiting to 10 the df contain only the 10 coldest and 10 hottest dates
5
6 df_cold = df.orderBy("Minimum").drop("Maximum").limit(10)
7 df_hot = df.orderBy(desc("Maximum")).drop("Minimum").limit(10)
```

**Εικόνα 11:** Κώδικας που παράγει τα dataframes για τις πιο ζεστές και πιο κρύες ημέρες

Τέλος εμφανίζουμε τα δύο dataframe που περιέχουν τα αποτελέσματα.

```
Cmd 7

1  #Print the results and the count
2
3  print("Dataframe that contains the 10 coldest dates: ")
4  df_cold.printSchema()
5  df_cold.show()
6
7  print("Dataframe that contains the 10 hottest dates: ")
8  df_hot.printSchema()
9  df_hot.show()
10
```

► (8) Spark Jobs

Dataframe that contains the 10 coldest dates:

root

```
-- Date: date (nullable = true)
-- Minimum: double (nullable = true)
```

Date	Minimum
2014-03-27	-3.0
2014-03-25	-3.0
2014-03-11	-3.0
2014-03-24	-2.0
2014-04-16	-1.0
2014-03-30	-1.0
2014-03-12	-1.0
2014-03-13	0.0
2014-03-26	0.0
2014-03-31	0.0

**Εικόνα 12:** Κώδικας για εμφάνιση αποτελεσμάτων και τα αποτελέσματα για τις πιο κρύες μέρες

```
Dataframe that contains the 10 hottest dates:
```

root

```
-- Date: date (nullable = true)
-- Maximum: double (nullable = true)
```

Date	Maximum
2014-05-22	25.0
2014-05-21	24.0
2014-05-30	22.0
2014-06-02	22.0
2014-06-08	22.0
2014-05-25	21.0
2014-04-29	20.0
2014-05-24	20.0
2014-05-26	20.0
2014-05-17	20.0

**Εικόνα 13:** Αποτελέσματα για τις πιο ζεστές μέρες

### 1.3

Αρχείο που περιέχει το κώδικα του ερωτήματος 3 για το θέμα 1: Thema\_1\_Query\_3.py

Για την επίλυση του ερωτήματος 3 αρχικά δημιουργήσαμε το dataframe από το αρχείο hum.csv. Έπειτα αντλήσαμε από το πεδίο κάθε ημερομηνίας τον μήνα κάθε εγγραφής. Τέλος χρησιμοποιήσαμε την .agg συνάρτηση της spark για να υπολογίσουμε την τυπική απόκλιση των τιμών που αφορούν ίδιους μήνες και διατηρήσαμε σε ένα νέο dataframe την εγγραφή με την μεγαλύτερη τιμή τυπικής απόκλισης.

Όπως και στα προηγούμενα ερωτήματα με τη δημιουργία του dataframe μετατρέψαμε τα πεδία date, value από string type σε Date type και Double type. Η μόνη διαφορά είναι ότι χρησιμοποιείται το αρχείο hum.csv. Λόγω της ομοιότητας αυτής δεν παραθέτουμε στιγμιότυπα από αυτό το κομμάτι του κώδικα.

Συνεχίζοντας μετά την μετατροπή των πεδίων στους σωστούς τύπους, προσθέτουμε στο dataframe τη στήλη "Month" η οποία περιέχει το μήνα κάθε ημερομηνίας.

```
Cmd 5

1 #Create a new column where the values are the months extracted from the Date column and drop the Date column
2
3 df_hum = df_hum.withColumn("Month",month("Date"))
4 df_hum.printSchema()
5 df_hum.show(5)

▶ (1) Spark Jobs
▶ df_hum: pyspark.sql.dataframe.DataFrame = [Date: date, Value: double ... 1 more field]

root
 |-- Date: date (nullable = true)
 |-- Value: double (nullable = true)
 |-- Month: integer (nullable = true)

+-----+-----+-----+
|   Date|Value|Month|
+-----+-----+-----+
|2014-02-13| 93.0|    2|
|2014-02-13| 66.0|    2|
|2014-02-13| 91.0|    2|
|2014-02-13| 84.0|    2|
|2014-02-13| 62.0|    2|
+-----+-----+-----+
only showing top 5 rows
```

**Εικόνα 14:** Κώδικας που προσθέτει την στήλη "Month" στο dataframe

Ολοκληρώνοντας υπολογίζουμε την τυπική απόκλιση κάθε μήνα, κρατάμε σε ένα νέο dataframe τη μεγαλύτερη και εμφανίζουμε τα αποτελέσματα.

```
Cnd 6
1 #Calculate the standard deviation for each month, order them in descending order and keep the first row which is the one with the largest standard deviation value
2
3 q_hum = df_hum.groupBy("Month").agg({'Value': 'stddev'}).withColumnRenamed('stddev(Value)', 'Standard deviation').orderBy(desc('Standard deviation')).limit(1)

▶ q_hum: pyspark.sql.dataframe.DataFrame = [Month: integer, Standard deviation: double]
Command took 0.30 seconds -- by johnekonon@gmail.com at 2/1/2024, 5:22:26 μ.μ. on Project

Cnd 7

1 #Print the results and the count
2
3 print("The month with the largest standard deviation is: ")
4 q_hum.printSchema()
5 q_hum.show()
6
7
8

▶ (2) Spark Jobs
The month with the largest standard deviation is:
root
|-- Month: integer (nullable = true)
|-- Standard deviation: double (nullable = true)

+-----+-----+
|Month|Standard deviation|
+-----+-----+
| 4|17.733347100625892|
+-----+-----+
```

**Εικόνα 15:** Κώδικας που δημιουργεί το dataframe με τη μέγιστη απόκλιση και εμφανίζει τα αποτελέσματα



## 1.4

Αρχείο που περιέχει το κώδικα του ερωτήματος 4 για το θέμα 1: Thema\_1\_Query\_4.py

Για την επίλυση του ερωτήματος 4 χρησιμοποιούμε και τα δύο αρχεία .csv κάνοντας τις ίδιες μετατροπές με τα προηγούμενα ερωτήματα έχοντας από ένα dataframe και για τα δύο. Όσον αφορά τον υπολογισμό του δείκτη δυσφορίας χρησιμοποιούμε τις ελάχιστες τιμές για τον ελάχιστο δείκτη και τις μέγιστες για τον μέγιστο δείκτη.

Στη συνέχεια υπολογίζουμε τις ελάχιστες και μέγιστες τιμές θερμοκρασίας και υγρασίας για κάθε εγγραφή στα δύο dataframe και τα ενώνουμε σε ένα ενιαίο.

```
Cmd 5
1 #Extract minimum and maximum values for each day and produce a dataframe that contains them
2
3 temp_min = df_temp.groupBy("Date").min("Value")
4 temp_min = temp_min.withColumn("Minimum temp",col("min(Value)"))
5 temp_max = df_temp.groupBy("Date").max("Value")
6 temp_max = temp_max.withColumn("Maximum temp",col("max(Value)"))
7 temp_val = (temp_min.join(temp_max, "Date").orderBy("Date")).drop("min(Value)","max(Value)")
8
9 hum_min = df_hum.groupBy("Date").min("Value")
10 hum_min = hum_min.withColumn("Minimum hum",col("min(Value)"))
11 hum_max = df_hum.groupBy("Date").max("Value")
12 hum_max = hum_max.withColumn("Maximum hum",col("max(Value)"))
13 hum_val = (hum_min.join(hum_max, "Date").orderBy("Date")).drop("min(Value)","max(Value)")
14
15 df = hum_val.join(temp_val, "Date").orderBy("Date")
16
17 df.printSchema()
18 df.show(5)
```

**Εικόνα 16:** Κώδικας που δημιουργεί το ενιαίο dataframe

```

16
17 df.printSchema()
18 df.show(5)

```

► (8) Spark Jobs

► temp\_min: pyspark.sql.dataframe.DataFrame = [Date: date, min(Value): double ... 1 more field]

► temp\_max: pyspark.sql.dataframe.DataFrame = [Date: date, max(Value): double ... 1 more field]

► temp\_val: pyspark.sql.dataframe.DataFrame = [Date: date, Minimum temp: double ... 1 more field]

► hum\_min: pyspark.sql.dataframe.DataFrame = [Date: date, min(Value): double ... 1 more field]

► hum\_max: pyspark.sql.dataframe.DataFrame = [Date: date, max(Value): double ... 1 more field]

► hum\_val: pyspark.sql.dataframe.DataFrame = [Date: date, Minimum hum: double ... 1 more field]

► df: pyspark.sql.dataframe.DataFrame = [Date: date, Minimum hum: double ... 3 more fields]

root

```

|-- Date: date (nullable = true)
|-- Minimum hum: double (nullable = true)
|-- Maximum hum: double (nullable = true)
|-- Minimum temp: double (nullable = true)
|-- Maximum temp: double (nullable = true)

```

Date	Minimum hum	Maximum hum	Minimum temp	Maximum temp
2014-02-13	55.0	93.0	2.0	7.0
2014-02-14	67.0	93.0	1.0	6.0
2014-02-15	47.0	87.0	3.0	9.0
2014-02-16	60.0	87.0	3.0	7.0
2014-02-17	40.0	93.0	1.0	8.0

only showing top 5 rows

**Εικόνα 17:** Ενδεικτικά αποτελέσματα από το ενιαίο dataframe

Έπειτα υπολογίζουμε τις ελάχιστες και μέγιστες τιμές του δείκτη δυσφορίας και δημιουργούμε ένα dataframe που περιέχει της ημερομηνίες και τιμές αυτές.

```

Cmd 6

1 #Calculate the minimum and maximum DI for each day and make two new columns that contain each value
2
3 df = df.withColumn("DI Minimum", col('Minimum temp') - 0.55*(1-0.01*col('Minimum hum'))*(col('Minimum temp') - 14.5))
4 df = df.withColumn("DI Maximum", col('Maximum temp') - 0.55*(1-0.01*col('Maximum hum'))*(col('Maximum temp') - 14.5))
5 df = df.drop('Minimum temp', 'Maximum temp', 'Minimum hum', 'Maximum hum')
6
7 df.printSchema()
8 df.show(5)

```

► (8) Spark Jobs

► df: pyspark.sql.dataframe.DataFrame = [Date: date, DI Minimum: double ... 1 more field]

root

```

|-- Date: date (nullable = true)
|-- DI Minimum: double (nullable = true)
|-- DI Maximum: double (nullable = true)

```

Date	DI Minimum	DI Maximum
2014-02-13	5.09375	7.28875
2014-02-14	3.45025	6.327249999999999
2014-02-15	6.352250000000001	9.39325
2014-02-16	5.53	7.53625
2014-02-17	5.455	8.25025

only showing top 5 rows

**Εικόνα 18:** Το dataframe με τις τιμές του δείκτη δυσφορίας

Τέλος υπολογίζουμε την ελάχιστη τιμή δείκτη και την μέγιστη από όλες και εμφανίζουμε τα αποτελέσματα.

```
Cmd 7
1 #Create a dataframe that contains the minimum DI values and a dataframe that contains the maximum DI values to calculate min and max for each by limiting to the first entry
2
3 max_val = df.drop('DI Minimum').orderBy(desc('DI Maximum')).limit(1)
4 min_val = df.drop('DI Maximum').orderBy('DI Minimum').limit(1)

> max_val: pyspark.sql.dataframe.DataFrame = [Date: date, DI Maximum: double]
> min_val: pyspark.sql.dataframe.DataFrame = [Date: date, DI Minimum: double]

Command took 0.19 seconds -- by johnekonon@gmail.com at 2/1/2024, 5:45:35 μ.μ. on Project

Cmd 8
1 #Print dataframes that contain the results
2
3 print('The first 5 rows of the dataframe that contains the minimum and maximum DI values for each date are: ')
4 df.printSchema()
5 df.show(5)
6
7 print('The dataframe that contains the minimum DI value is: ')
8 min_val.printSchema()
9 min_val.show()
10
11 print('The dataframe that contains the maximum DI value is: ')
12 max_val.printSchema()
13 max_val.show()
```

Εικόνα 19: Κώδικας που υπολογίζει και εμφανίζει τα τελικά αποτελέσματα

```
The dataframe that contains the minimum DI value is:
root
 |-- Date: date (nullable = true)
 |-- DI Minimum: double (nullable = true)

+-----+-----+
|      Date|      DI Minimum|
+-----+-----+
|2014-03-27|0.07999999999999963|
+-----+-----+

The dataframe that contains the maximum DI value is:
root
 |-- Date: date (nullable = true)
 |-- DI Maximum: double (nullable = true)

+-----+-----+
|      Date|DI Maximum|
+-----+-----+
|2014-05-22|    23.9605|
+-----+-----+
```

Εικόνα 20: Τα τελικά αποτελέσματα

## Θέμα 2

Αρχικά για το θέμα 2 η μορφή των αρχείων .txt ήταν κατάλληλη για την επίλυση των ερωτημάτων επομένως δεν έχουν γίνει αλλαγές στα αρχεία.

**ΣΗΜΕΙΩΣΗ:** Στα στιγμιότυπα κώδικα παρακάτω υπάρχουν και ενδεικτικά αποτελέσματα κάθε εντολής τα οποία δεν υπάρχουν στην τελική μορφή των .py αρχείων και χρησιμοποιούνται μόνο για την σύνταξη της αναφοράς και για debugging.

### 2.1

Αρχείο που περιέχει το κώδικα του ερωτήματος 1 για το θέμα 2: Thema\_2\_Query\_1.py

Αρχικά δημιουργούνται τα τρία dataframes που αντιστοιχούν στα 3 αρχεία .txt και μετατρέπονται τα πεδία από String type στους σωστούς τύπους με Date type για το πεδίο Date, double type για τα πεδία Open, Close, High, Low και int type για τα πεδία Volume, Open int.

Ενδεικτική μορφή αρχείων μετά την μετατροπή:

```
Edit View Run Help Last edit was 3 minutes ago Provide feedback

1 #Transform each column that has integer type values from String type to Int type
2
3 columns = ['Volume', 'OpenInt']
4 for x in columns:
5     df_agn = df_agn.withColumn(x, col(x).cast('int'))
6     df_ainv = df_ainv.withColumn(x, col(x).cast('int'))
7     df_ale = df_ale.withColumn(x, col(x).cast('int'))
8
9 df_agn.printSchema()
10 df_agn.show(5)
```

▶ (1) Spark Jobs

- ▶ df\_agn: pyspark.sql.dataframe.DataFrame = [Date: date, Open: double ... 5 more fields]
- ▶ df\_ainv: pyspark.sql.dataframe.DataFrame = [Date: date, Open: double ... 5 more fields]
- ▶ df\_ale: pyspark.sql.dataframe.DataFrame = [Date: date, Open: double ... 5 more fields]

root

```
-- Date: date (nullable = true)
-- Open: double (nullable = true)
-- High: double (nullable = true)
-- Low: double (nullable = true)
-- Close: double (nullable = true)
-- Volume: integer (nullable = true)
-- OpenInt: integer (nullable = true)
```

Date	Open	High	Low	Close	Volume	OpenInt
2005-01-03	32.31	32.31	31.527	31.616	1027044	0
2005-01-04	31.527	31.616	31.22	31.338	1927762	0
2005-01-05	30.971	31.051	30.714	30.843	943399	0
2005-01-06	30.843	31.398	30.764	31.26	662398	0
2005-01-07	31.26	31.26	30.456	30.566	1087886	0

only showing top 5 rows

**Εικόνα 21:** Κώδικας που μετατρέπει τα πεδία Volume, OpenInt και τελική μορφή του ενός αρχείου

Στη συνέχεια προσθέτουμε στο dataframe τη στήλη “Month” που περιέχει το μήνα κάθε ημερομηνίας.

```
Cmd 6

1 #Create a new column where the values are the months extracted from the Date column
2
3 df_agn = df_agn.withColumn("Month",month("Date"))
4 df_ainv = df_ainv.withColumn("Month",month("Date"))
5 df_ale = df_ale.withColumn("Month",month("Date"))
6
7 df_agn.printSchema()
8 df_agn.show(5)

▶ (1) Spark Jobs
▶ df_agn: pyspark.sql.dataframe.DataFrame = [Date: date, Open: double ... 6 more fields]
▶ df_ainv: pyspark.sql.dataframe.DataFrame = [Date: date, Open: double ... 6 more fields]
▶ df_ale: pyspark.sql.dataframe.DataFrame = [Date: date, Open: double ... 6 more fields]

root
|-- Date: date (nullable = true)
|-- Open: double (nullable = true)
|-- High: double (nullable = true)
|-- Low: double (nullable = true)
|-- Close: double (nullable = true)
|-- Volume: integer (nullable = true)
|-- OpenInt: integer (nullable = true)
|-- Month: integer (nullable = true)

+-----+-----+-----+-----+-----+-----+-----+
| Date | Open | High | Low | Close | Volume | OpenInt | Month |
+-----+-----+-----+-----+-----+-----+-----+
| 2005-01-03 | 32.31 | 32.31 | 31.527 | 31.616 | 1027044 | 0 | 1 |
| 2005-01-04 | 31.527 | 31.616 | 31.22 | 31.338 | 1927762 | 0 | 1 |
| 2005-01-05 | 30.971 | 31.051 | 30.714 | 30.843 | 943399 | 0 | 1 |
| 2005-01-06 | 30.843 | 31.398 | 30.764 | 31.26 | 662398 | 0 | 1 |
| 2005-01-07 | 31.26 | 31.26 | 30.456 | 30.566 | 1087886 | 0 | 1 |
+-----+-----+-----+-----+-----+-----+-----+

only showing top 5 rows
```

Εικόνα 22: Κώδικας που προσθέτει τη στήλη Month

Τέλος υπολογίζουμε τους μέσους όρους με την συνάρτηση avg για τα πεδία Open, Close, Volume και εμφανίζουμε τα αποτελέσματα.

```
Cmd 7

1 #Query that returns a dataframe where the average values of columns open, close and volume are calculated and ordered by its month
2
3 q_agn = df_agn.groupBy("Month").avg('Open','Close','Volume').orderBy('Month')
4 q_ainv = df_ainv.groupBy("Month").avg('Open','Close','Volume').orderBy('Month')
5 q_ale = df_ale.groupBy("Month").avg('Open','Close','Volume').orderBy('Month')

▶ q_agn: pyspark.sql.dataframe.DataFrame = [Month: integer, avg(Open): double ... 2 more fields]
▶ q_ainv: pyspark.sql.dataframe.DataFrame = [Month: integer, avg(Open): double ... 2 more fields]
▶ q_ale: pyspark.sql.dataframe.DataFrame = [Month: integer, avg(Open): double ... 2 more fields]

Command took 0.29 seconds -- by johnekonon@gmail.com at 2/1/2024, 5:57:34 μ.μ. on Project

Cmd 8

1 #Print the schema for each dataframe and show the results
2
3 print("Values for AGN:")
4 q_agn.printSchema()
5 q_agn.show()
6
7 print("Values for AINV:")
8 q_ainv.printSchema()
9 q_ainv.show()
10
11 print("Values for ALE:")
12 q_ale.printSchema()
13 q_ale.show()
```

Εικόνα 23: Κώδικας που υπολογίζει τους μέσους όρους και εμφανίζει τα αποτελέσματα

```

Values for AGN:
root
|-- Month: integer (nullable = true)
|-- avg(Open): double (nullable = true)
|-- avg(Close): double (nullable = true)
|-- avg(Volume): double (nullable = true)

+-----+-----+-----+-----+
|Month|      avg(Open)|      avg(Close)|      avg(Volume)|
+-----+-----+-----+-----+
| 1|101.83225287356325|101.80168582375484|      1796786.0|
| 2|106.10592400000006|106.37493199999994|      2018339.112|
| 3| 108.9747754385965|108.91760000000008|1954400.0771929824|
| 4|104.96801119402988|104.85799626865673|2048475.3208955224|
| 5|106.58948727272725|106.51974545454549| 2020206.469090909|
| 6|110.05557857142851|110.13075357142851|1611504.9321428572|
| 7|113.98697802197806|114.04540659340655|1486405.2857142857|
| 8|112.94372664359865|112.76380276816606|1592136.8719723183|
| 9|113.42711654135339| 113.2947368421053|1720100.3007518798|
|10|109.94360701754385|109.83696140350877| 2045412.403508772|
|11|105.93957707509878| 105.7247312252964| 2363413.675889328|

Command took 1.70 seconds -- by johnekonom@gmail.com at 2/1/2024, 5:57:34 μ.μ.

```

**Εικόνα 24:** Ενδεικτικά αποτελέσματα για τη μετοχή AGN

```

Values for AINV:
root
|-- Month: integer (nullable = true)
|-- avg(Open): double (nullable = true)
|-- avg(Close): double (nullable = true)
|-- avg(Volume): double (nullable = true)

+-----+-----+-----+-----+
|Month|      avg(Open)|      avg(Close)|      avg(Volume)|
+-----+-----+-----+-----+
| 1| 5.960922821576766| 5.957890041493774|2664324.3029045644|
| 2|5.9107146551724075| 5.914387068965516| 3277507.879310345|
| 3| 6.044150877192985| 6.036955789473684|3266939.4280701755|
| 4| 6.208691417910444| 6.210719029850749|2431237.5970149254|
| 5|6.2512025454545475| 6.252906909090904|2611925.7163636363|
| 6| 6.227297857142855| 6.216372857142856|2798179.3964285715|
| 7| 6.111186080586078|6.1186201465201435|2189656.0036630034|
| 8| 6.28397370242215| 6.29405778546713| 2754273.910034602|
| 9| 6.352289097744363| 6.345983082706774|2706609.6917293235|
|10| 6.212461403508779| 6.203042456140353|2141400.1192982458|
|11| 6.114790118577075|6.1033130434782565|2523151.9881422925|

```

**Εικόνα 25:** Ενδεικτικά αποτελέσματα για τη μετοχή AINV

```

Values for ALE:
root
|-- Month: integer (nullable = true)
|-- avg(Open): double (nullable = true)
|-- avg(Close): double (nullable = true)
|-- avg(Volume): double (nullable = true)

```

Month	avg(Open)	avg(Close)	avg(Volume)
1	37.59970539419089	37.633917012448144	223401.15352697094
2	38.356318965517254	38.34859051724136	240881.875
3	38.04251228070175	38.07297192982455	253724.5649122807
4	38.59285447761192	38.61462686567163	240209.06343283583
5	39.37950181818183	39.3823163636364	244737.14909090908
6	39.890178571428585	39.918649999999999	257443.88214285715
7	40.26568864468865	40.257758241758246	209357.9010989011
8	40.1072802768166	40.08710726643595	245987.4429065744
9	39.87034210526317	39.86111654135338	226992.70676691728
10	40.06615789473683	40.06423157894736	235927.1754385965
11	38.162494071146256	38.12544268774704	225893.2648221344

Εικόνα 26: Ενδεικτικά αποτελέσματα για τη μετοχή ALE

## 2.2

Αρχείο που περιέχει το κώδικα του ερωτήματος 2 για το θέμα 2: Thema\_2\_Query\_2.py

Η μετατροπή των αρχείων όσον αφορά τους τύπους των πεδίων είναι ίδια με το ερώτημα 1.

Για τον υπολογισμό των αποτελεσμάτων δημιουργούμε νέα dataframe τα οποία περιέχουν τις εγγραφές με τιμή Open μεγαλύτερη ή ίση με 35.

```
Cmd 6  
  
1  #Query that returns a dataframe where the Open value for each entry is over 35 dollars and then counts how many entries are in each dataframe  
2  
3  q_agn = df_agn.filter(df_agn["Open"] >= 35)  
4  agn_count = q_agn.count()  
5  
6  q_ainv = df_ainv.filter(df_ainv["Open"] >= 35)  
7  ainv_count = q_ainv.count()  
8  
9  q_ale = df_ale.filter(df_ale["Open"] >= 35)  
10 ale_count = q_ale.count()
```

**Εικόνα 27:** Κώδικας που υλοποιεί το ερώτημα



Τέλος εμφανίζουμε τα αποτελέσματα.

```
Cmd 7

1  #Print the schema for each dataframe, show the first 5 entries and print how many days the open price was over 35 dollars
2
3  print("Values for AGN:")
4  q_agn.printSchema()
5  q_agn.show(5)
6
7  print("Values for AINV:")
8  q_ainv.printSchema()
9  q_ainv.show(5)
10
11 print("Values for ALE:")
12 q_ale.printSchema()
13 q_ale.show(5)
14
15 print("Number of days that AGN stock opened at a price over 35 dollars: ",agn_count,"\n")
16 print("Number of days that AINV stock opened at a price over 35 dollars: ",ainv_count,"\n")
17 print("Number of days that ALE stock opened at a price over 35 dollars: ",ale_count,"\n")

▶ (3) Spark Jobs

root
|-- Date: date (nullable = true)
|-- Open: double (nullable = true)
|-- High: double (nullable = true)
|-- Low: double (nullable = true)
|-- Close: double (nullable = true)
|-- Volume: integer (nullable = true)
|-- OpenInt: integer (nullable = true)

+-----+-----+-----+-----+-----+-----+-----+
|   Date|  Open|  High|  Low| Close| Volume|OpenInt|
+-----+-----+-----+-----+-----+-----+-----+
|2005-08-22|35.183|35.452|34.926|35.383|1391488|    0|
|2005-08-23|35.293|35.323|34.529|34.569|1014834|    0|
|2005-09-12| 35.56|35.808|34.827|34.946|1238627|    0|
|2005-09-16|35.085|35.283|34.896|35.243|1010899|    0|
|2005-09-19|35.283|35.333| 34.49|34.619| 434770|    0|
+-----+-----+-----+-----+-----+-----+-----+
only showing top 5 rows
```

**Εικόνα 28:** Κώδικας για εμφάνιση αποτελεσμάτων και ενδεικτικά αποτελέσματα για την μετοχή AGN

```
Number of days that AGN stock opened at a price over 35 dollars:  2071

Number of days that AINV stock opened at a price over 35 dollars:  0

Number of days that ALE stock opened at a price over 35 dollars: 1667
```

**Εικόνα 29:** Τελικά αποτελέσματα ερωτήματος

## 2.3

Αρχείο που περιέχει το κώδικα του ερωτήματος 3 για το θέμα 2: Thema\_2\_Query\_3.py

Η μετατροπή των αρχείων όσον αφορά τους τύπους των πεδίων είναι ίδια με το ερώτημα 1.

Για την υλοποίηση του ερωτήματος προσθέτουμε στο dataframe μια στήλη που περιέχει struct των τιμών Open ή Volume (Ανάλογα με το πεδίο που επιθυμούμε να βρούμε) και την ημερομηνία. Έπειτα βρίσκουμε την μέγιστη τιμή από το Open ή το Volume και μαζί την ημέρα που καταγράφηκαν οι τιμές αυτές.

```
Cmd 6

1 #Query that create structs with Open,Date values for each entry in the original dataframe
2 #Then finds the maximum struct by Open
3 #And then returns a dataframe that has the maximum Open value and the Day that it was achieved
4
5 df_agn_open = df_agn.withColumn('Open_Date_struct', struct(df_agn.Open, df_agn.Date))
6 max_df_open = df_agn_open.agg(max('Open_Date_struct').alias('Max_Open'))
7 q_agn_open = max_df_open.withColumn('Date', max_df_open.Max_Open.Date).withColumn('Open', max_df_open.Max_Open.Open).drop('Max_Open')
8
9 df_ainv_open = df_ainv.withColumn('Open_Date_struct', struct(df_ainv.Open, df_ainv.Date))
10 max_df_open = df_ainv_open.agg(max('Open_Date_struct').alias('Max_Open'))
11 q_ainv_open = max_df_open.withColumn('Date', max_df_open.Max_Open.Date).withColumn('Open', max_df_open.Max_Open.Open).drop('Max_Open')
12
13 df_ale_open = df_ale.withColumn('Open_Date_struct', struct(df_ale.Open, df_ale.Date))
14 max_df_open = df_ale_open.agg(max('Open_Date_struct').alias('Max_Open'))
15 q_ale_open = max_df_open.withColumn('Date', max_df_open.Max_Open.Date).withColumn('Open', max_df_open.Max_Open.Open).drop('Max_Open')
```

**Εικόνα 30:** Κώδικας που βρίσκει τη μέγιστη τιμή για το Open και την ημέρα

Τέλος εμφανίζει τα αποτελέσματα.

```
Cmd 8

1 #Print the schema for each dataframe and show it, where each dataframe contains the maximum value of Open or Volume for each stock
2
3 print("Open values for AGN: ")
4 q_agn_open.printSchema()
5 q_agn_open.show()
6
7 print("Volume Values for AGN: ")
8 q_agn_volume.printSchema()
9 q_agn_volume.show()
10
11 print("Open values for AINV: ")
12 q_ainv_open.printSchema()
13 q_ainv_open.show()
14
15 print("Volume Values for AINV: ")
16 q_ainv_volume.printSchema()
17 q_ainv_volume.show()
18
19 print("Open values for ALE: ")
20 q_ale_open.printSchema()
21 q_ale_open.show()
22
23 print("Volume Values for AINV: ")
24 q_ale_volume.printSchema()
25 q_ale_volume.show()
26
```

**Εικόνα 31:** Κώδικας που εμφανίζει τα αποτελέσματα

```

Open values for AGN:
root
|-- Date: date (nullable = true)
|-- Open: double (nullable = true)

+-----+-----+
|      Date|  Open|
+-----+-----+
|2015-07-30|334.08|
+-----+-----+

Volume Values for AGN:
root
|-- Date: date (nullable = true)
|-- Volume: integer (nullable = true)

+-----+-----+
|      Date| Volume|
+-----+-----+
|2016-04-05|36807460|
+-----+-----+

Command took 3.42 seconds -- by johneconom@gmail.com at 2/1/2024, 6:08:26 μ.μ. on Project

```

**Εικόνα 32:** Ενδεικτικά αποτελέσματα για τη μετοχή AGN

```

Open values for AINV:
root
|-- Date: date (nullable = true)
|-- Open: double (nullable = true)

+-----+-----+
|      Date|  Open|
+-----+-----+
|2007-02-20|11.474|
+-----+-----+

Volume Values for AINV:
root
|-- Date: date (nullable = true)
|-- Volume: integer (nullable = true)

+-----+-----+
|      Date| Volume|
+-----+-----+
|2014-02-28|57522365|
+-----+-----+

```

**Εικόνα 33:** Ενδεικτικά αποτελέσματα για τη μετοχή AINV

```

Open values for ALE:
root
|-- Date: date (nullable = true)
|-- Open: double (nullable = true)

+-----+-----+
|      Date|Open|
+-----+-----+
|2017-11-01|80.0|
+-----+-----+

Volume Values for ALE:
root
|-- Date: date (nullable = true)
|-- Volume: integer (nullable = true)

+-----+-----+
|      Date| Volume|
+-----+-----+
|2014-02-27|2118295|

```

**Εικόνα 34:** Ενδεικτικά αποτελέσματα για τη μετοχή ALE

## 2.4

Αρχείο που περιέχει το κώδικα του ερωτήματος 4 για το θέμα 2: Thema\_2\_Query\_4.py

Η μετατροπή των αρχείων όσον αφορά τους τύπους των πεδίων είναι ίδια με το ερώτημα 1. Επίσης χρησιμοποιούμε την ίδια λογική με το ερώτημα 3 για την εύρεση της χρονιάς με επιθυμητές τιμές.

Αρχικά προσθέτουμε τη στήλη “Year” που περιέχει τη χρονιά για κάθε τιμή.

```
Cmd 6

1 #Create a new column where the values are the years extracted from the Date column
2
3 df_agm = df_agm.withColumn("Year",year("Date"))
4 df_ainv = df_ainv.withColumn("Year",year("Date"))
5 df_ale = df_ale.withColumn("Year",year("Date"))
```

Εικόνα 35: Κώδικας που προσθέτει την στήλη Year

Στη συνέχεια υπολογίζουμε με τα struct Open\_Year και Close\_Year τις χρονιές με τη μέγιστη τιμή σε Open και την ελάχιστη σε Close και ενώνουμε τα δύο αυτά dataframe σε ένα.

```
Cmd 7

1 #Query that create structs with Open,Date values for each entry in the original dataframe
2 #Then finds the maximum or minimum struct by Open and Close
3 #And then through join returns a dataframe that has the maximum Open, minimum Close and Year values
4
5 df_agm_open = df_agm.withColumn('Open_Year_struct', struct(df_agm.Open, df_agm.Year))
6 df_agm_close = df_agm.withColumn('Close_Year_struct', struct(df_agm.Close, df_agm.Year))
7 max_df_open = df_agm_open.agg(max('Open_Year_struct').alias('Max_Open'))
8 min_df_close = df_agm_close.agg(min('Close_Year_struct').alias('Min_Close'))
9 q_agm = (max_df_open.withColumn('Max_Year', max_df_open.Max_Open.Year).withColumn('Max_Open', max_df_open.Max_Open.Open).drop('Max_Open')).join(min_df_close.withColumn('Min_Year', min_df_close.Min_Close.Year).withColumn('Min_Close', min_df_close.Min_Close.Close).drop('Min_Close'))
10
11 df_ainv_open = df_ainv.withColumn('Open_Year_struct', struct(df_ainv.Open, df_ainv.Year))
12 df_ainv_close = df_ainv.withColumn('Close_Year_struct', struct(df_ainv.Close, df_ainv.Year))
13 max_df_open = df_ainv_open.agg(max('Open_Year_struct').alias('Max_Open'))
14 min_df_close = df_ainv_close.agg(min('Close_Year_struct').alias('Min_Close'))
15 q_ainv = (max_df_open.withColumn('Max_Year', max_df_open.Max_Open.Year).withColumn('Max_Open', max_df_open.Max_Open.Open).drop('Max_Open')).join(min_df_close.withColumn('Min_Year', min_df_close.Min_Close.Year).withColumn('Min_Close', min_df_close.Min_Close.Close).drop('Min_Close'))
16
17 df_ale_open = df_ale.withColumn('Open_Year_struct', struct(df_ale.Open, df_ale.Year))
18 df_ale_close = df_ale.withColumn('Close_Year_struct', struct(df_ale.Close, df_ale.Year))
19 max_df_open = df_ale_open.agg(max('Open_Year_struct').alias('Max_Open'))
20 min_df_close = df_ale_close.agg(min('Close_Year_struct').alias('Min_Close'))
21 q_ale = (max_df_open.withColumn('Max_Year', max_df_open.Max_Open.Year).withColumn('Max_Open', max_df_open.Max_Open.Open).drop('Max_Open')).join(min_df_close.withColumn('Min_Year', min_df_close.Min_Close.Year).withColumn('Min_Close', min_df_close.Min_Close.Close).drop('Min_Close'))
```

Notebook detached  
cluster not found

Εικόνα 36: Κώδικας που δημιουργεί το ενιαίο dataframe και υλοποιεί το ερώτημα

Τέλος εμφανίζει τα αποτελέσματα για κάθε μετοχή.

```
Cmd 8

1  #Print the schema for each dataframe and show it, where each dataframe contains the maximum and minimum values for Open and the Year each one was achieved
2
3  print("Max and min Value for AGN: ")
4  q_agn.printSchema()
5  q_agn.show()
6
7  print("Max and min Value for AINV: ")
8  q_ainv.printSchema()
9  q_ainv.show()
10
11 print("Max and min Value for ALE: ")
12 q_ale.printSchema()
13 q_ale.show()
```

► (12) Spark Jobs

Max and min Value for AGN:

```
root
|-- Max Year: integer (nullable = true)
|-- Max Open: double (nullable = true)
|-- Min Year: integer (nullable = true)
|-- Min Close: double (nullable = true)
```

Max Year	Max Open	Min Year	Min Close
2015	334.08	2008	20.575

**Εικόνα 37:** Κώδικας που εμφανίζει τα αποτελέσματα και ενδεικτικά τα αποτελέσματα για τη μετοχή AGN.

```
Max and min Value for AINV:
root
|-- Max Year: integer (nullable = true)
|-- Max Open: double (nullable = true)
|-- Min Year: integer (nullable = true)
|-- Min Close: double (nullable = true)
```

Max Year	Max Open	Min Year	Min Close
2007	11.474	2009	0.9797

**Εικόνα 38:** Ενδεικτικά αποτελέσματα για την μετοχή AINV

```
Max and min Value for ALE:
root
|-- Max Year: integer (nullable = true)
|-- Max Open: double (nullable = true)
|-- Min Year: integer (nullable = true)
|-- Min Close: double (nullable = true)
```

Max Year	Max Open	Min Year	Min Close
2017	80.0	2009	18.29

**Εικόνα 39:** Ενδεικτικά αποτελέσματα για την μετοχή ALE

### Θέμα 3

Στην αναφορά δεν παραθέτουμε αυτούσιο τον κώδικα. Παρόλα αυτά, μέσω των παρακάτω screenshots φαίνεται ξεκάθαρα ο κώδικας, τα σχόλια που αναφέρουν την λειτουργία του κώδικα καθώς και τα αποτελέσματα του.

Αρχικά να σημειωθεί πως προσαρμόζουμε το αρχείο με τρόπο ώστε να μπορούν να ολοκληρωθούν τα ζητούμενα του θέματος και να γίνουν οι απαραίτητοι υπολογισμοί. Συγκεκριμένα, ορισμένοι χαρακτήρες ':' παίρνουν τιμή 0. Αυτό γίνεται επειδή στην συνέχεια μετατρέπουμε όλες τις τιμές στο αρχείο από strings σε integers ώστε να μπορούν να γίνουν οι απαραίτητοι υπολογισμοί βάσει αυτών των τιμών (πχ. δεν μπορεί να γίνει υπολογισμός μικρότερης τιμής ανάμεσα σε strings). Άλλη μια μορφοποίηση του αρχείου csv που έχει γίνει για διευκόλυνση στην επεξεργασία των δεδομένων είναι η αναστροφή των columns και των rows αντίστοιχα, δηλαδή στο ανανεωμένο csv αρχείο οι στήλες αναπαριστούν τις χώρες και οι γραμμές τις χρονολογίες. Αναλυτικά μέρος του πίνακα φαίνεται παρακάτω:

GEO/TIME	Belgium	Bulgaria	Czech Rep	Denmark	Germany	Estonia	Ireland
2006	#####	#####	#####	9,453,026	#####	3,020,367	#####
2007	#####	#####	#####	9,327,579	#####	2,915,456	:
2008	#####	#####	#####	8,918,197	#####	2,932,662	:
2009	#####	9,460,922	#####	8,299,403	#####	2,740,696	:
2010	#####	#####	#####	8,981,992	#####	3,203,721	:
2011	#####	#####	#####	9,491,137	#####	3,748,865	:
2012	#####	#####	#####	9,608,124	#####	3,823,039	#####
2013	#####	#####	#####	9,914,273	#####	3,909,326	#####
2014	#####	#####	#####	#####	#####	3,919,299	#####
2015	#####	#####	#####	#####	#####	3,770,207	#####

**Εικόνα 40:** <Ενδεικτικό κομμάτι από το αρχικό csv αρχείο >

Σε αυτό το σημείο να σημειωθεί πως παρόλο που δεν φαίνονται όλες οι τιμές σαν αριθμοί αλλά ως '##' (με μια μικρή επεξεργασία του csv αρχείου μπορούμε να δούμε αναλυτικά όλες τις τιμές), όταν εκτυπώνεται το αρχείο στο Databricks οι τιμές έχουν την κανονική μορφή τους.

Για το 1<sup>ο</sup> ερώτημα εξηγούμε αναλυτικά την διαδικασία της επεξεργασίας του αρχείου και της δημιουργίας του τελικού πίνακα που είναι έτοιμος για επεξεργασία, το οποίο γίνεται σε αρκετά blocks κώδικα, όμως για τα υπόλοιπα ερωτήματα αυτή η διαδικασία γίνεται στο 1<sup>ο</sup> block κώδικα.

**Σημείωση:** Παρακάτω παραθέτουμε ενδεικτικά screenshots από τα αποτελέσματα. Παρόλα αυτά, επειδή οι στήλες είναι αρκετές παραπάνω από τις γραμμές, το περιβάλλον του Databricks δεν εκτυπώνει στοιχισμένους τους πίνακες με αρκετές τιμές.

Σε αυτό το σημείο παραθέτουμε την εικόνα με ολόκληρο το csv:

2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047	2048	2049	2050	2051	2052	2053	2054	2055	2056	2057	2058	2059	2060	2061	2062	2063	2064	2065	2066	2067	2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095	2096	2097	2098	2099	2100	2101	2102	2103	2104	2105	2106	2107	2108	2109	2110	2111	2112	2113	2114	2115	2116	2117	2118	2119	2120	2121	2122	2123	2124	2125	2126	2127	2128	2129	2130	2131	2132	2133	2134	2135	2136	2137	2138	2139	2140	2141	2142	2143	2144	2145	2146	2147	2148	2149	2150	2151	2152	2153	2154	2155	2156	2157	2158	2159	2160	2161	2162	2163	2164	2165	2166	2167	2168	2169	2170	2171	2172	2173	2174	2175	2176	2177	2178	2179	2180	2181	2182	2183	2184	2185	2186	2187	2188	2189	2190	2191	2192	2193	2194	2195	2196	2197	2198	2199	2200	2201	2202	2203	2204	2205	2206	2207	2208	2209	2210	2211	2212	2213	2214	2215	2216	2217	2218	2219	2220	2221	2222	2223	2224	2225	2226	2227	2228	2229	2230	2231	2232	2233	2234	2235	2236	2237	2238	2239	2240	2241	2242	2243	2244	2245	2246	2247	2248	2249	2250	2251	2252	2253	2254	2255	2256	2257	2258	2259	2260	2261	2262	2263	2264	2265	2266	2267	2268	2269	2270	2271	2272	2273	2274	2275	2276	2277	2278	2279	2280	2281	2282	2283	2284	2285	2286	2287	2288	2289	2290	2291	2292	2293	2294	2295	2296	2297	2298	2299	2300	2301	2302	2303	2304	2305	2306	2307	2308	2309	2310	2311	2312	2313	2314	2315	2316	2317	2318	2319	2320	2321	2322	2323	2324	2325	2326	2327	2328	2329	2330	2331	2332	2333	2334	2335	2336	2337	2338	2339	2340	2341	2342	2343	2344	2345	2346	2347	2348	2349	2350	2351	2352	2353	2354	2355	2356	2357	2358	2359	2360	2361	2362	2363	2364	2365	2366	2367	2368	2369	2370	2371	2372	2373	2374	2375	2376	2377	2378	2379	2380	2381	2382	2383	2384	2385	2386	2387	2388	2389	2390	2391	2392	2393	2394	2395	2396	2397	2398	2399	2400	2401	2402	2403	2404	2405	2406	2407	2408	2409	2410	2411	2412	2413	2414	2415	2416	2417	2418	2419	2420	2421	2422	2423	2424	2425	2426	2427	2428	2429	2430	2431	2432	2433	2434	2435	2436	2437	2438	2439	2440	2441	2442	2443	2444	2445	2446	2447	2448	2449	2450	2451	2452	2453	2454	2455	2456	2457	2458	2459	2460	2461	2462	2463	2464	2465	2466	2467	2468	2469	2470	2471	2472	2473	2474	2475	2476	2477	2478	2479	2480	2481	2482	2483	2484	2485	2486	2487	2488	2489	2490	2491	2492	2493	2494	2495	2496	2497	2498	2499	2500	2501	2502	2503	2504	2505	2506	2507	2508	2509	2510	2511	2512	2513	2514	2515	2516	2517	2518	2519	2520	2521	2522	2523	2524	2525	2526	2527	2528	2529	2530	2531	2532	2533	2534	2535	2536	2537	2538	2539	2540	2541	2542	2543	2544	2545	2546	2547	2548	2549	2550	2551	2552	2553	2554	2555	2556	2557	2558	2559	2560	2561	2562	2563	2564	2565	2566	2567	2568	2569	2570	2571	2572	2573	2574	2575	2576	2577	2578	2579	2580	2581	2582	2583	2584	2585	2586	2587	2588	2589	2590	2591	2592	2593	2594	2595	2596	2597	2598	2599	2600	2601	2602	2603	2604	2605	2606	2607	2608	2609	2610	2611	2612	2613	2614	2615	2616	2617	2618	2619	2620	2621	2622	2623	2624	2625	2626	2627	2628	2629	2630	2631	2632	2633	2634	2635	2636	2637	2638	2639	2640	2641	2642	2643	2644	2645	2646	2647	2648	2649	2650	2651	2652	2653	2654	2655	2656	2657	2658	2659	2660	2661	2662	2663	2664	2665	2666	2667	2668	2669	2670	2671	2672	2673	2674	2675	2676	2677	2678	2679	2680	2681	2682	2683	2684	2685	2686	2687	2688	2689	2690	2691	2692	2693	2694	2695	2696	2697	2698	2699	2700	2701	2702	2703	2704	2705	2706	2707	2708	2709	2710	2711	2712	2713	2714	2715	2716	2717	2718	2719	2720	2721	2722	2723	2724	2725	2726	2727	2728	2729	2730	2731	2732	2733	2734	2735	2736	2737	2738	2739	2740	2741	2742	2743	2744	2745	2746	2747	2748	2749	2750	2751	2752	2753	2754	2755	2756	2757	2758	2759	2760	2761	2762	2763	2764	2765	2766	2767	2768	2769	2770	2771	2772	2773	2774	2775	2776	2777	2778	2779	2780	2781	2782	2783	2784	2785	2786	2787	2788	2789	2790	2791	2792	2793	2794	2795	2796	2797	2798	2799	2800	2801	2802	2803	2804	2805	2806	2807	2808	2809	2810	2811	2812	2813	2814	2815	2816	2817	2818	2819	2820	2821	2822	2823	2824	2825	2826	2827	2828	2829	2830	2831	2832	2833	2834	2835	2836	2837	2838	2839	2840	2841	2842	2843	2844	2845	2846	2847	2848	2849	2850	2851	2852	2853	2854	2855	2856	2857	2858	2859	2860	2861	2862	2863	2864	2865	2866	2867	2868	2869	2870	2871	2872	2873	2874	2875	2876	2877	2878	2879	2880	2881	2882	2883	2884	2885	2886	2887	2888	2889	2890	2891	2892	2893	2894	2895	2896	2897	2898	2899	2900	2901	2902	2903	2904	2905	2906	2907	2908	2909	2910	2911	2912	2913	2914	2915	2916	2917	2918	2919	2920	2921	2922	2923	2924	2925	2926	2927	2928	2929	2930	2931	2932	2933	2934	2935	2936	2937	2938	2939	2940	2941	2942	2943	2944	2945	2946	2947	2948	2949	2950	2951	2952	2953	2954	2955	2956	2957	2958	2959	2960	2961	2962	2963	2964	2965	2966	2967	2968	2969	2970	2971	2972	2973	2974	2975	2976	2977	2978	2979	2980	2981	2982	2983	2984	2985	2986	2987	2988	2989	2990	2991	2992	2993	2994	2995	2996	2997	2998	2999	3000	3001	3002	3003	3004	3005	3006	3007	3008	3009	3010	3011	3012	3013	3014	3015	3016	3017	3018	3019	3020	3021	3022	3023	3024	3025	3026	3027	3028	3029	3030	3031	3032	3033	3034	3035	3036	3037	3038	3039	3040	3041	3042	3043	3044	3045	3046	3047	3048	3049	3050	3051	3052	3053	3054	3055	3056	3057	3058	3059	3060	3061	3062	3063	3064	3065	3066	3067	3068	3069	3070	3071	3072	3073	3074	3075	3076	3077	3078	3079	3080	3081	3082	3083	3084	3085	3086	3087	3088	3089	3090	3091	3092	3093	3094	3095	3096	3097	3098	3099	3100	3101	3102	3103	3104	3105	3106	3107	3108	3109	3110	3111	3112	3113	3114	3115	3116	3117	3118	3119	3120	3121	3122	3123	3124	3125	3126	3127	3128	3129	3130	3131	3132	3133	3134	3135	3136	3137	3138	3139	3140	3141	3142	3143	3144	3145	3146	3147	3148	3149	3150	3151	3152	3153	3154	3155	3156	3157	3158	3159	3160	3161	3162	3163	3164	3165	3166	3167	3168	3169	3170	3171	3172	3173	3174	3175	3176	3177	3178	3179	3180	3181	3182	3183	3184	3185	3186	3187	3188	3189	3190	3191	3192	3193	3194	3195	3196	3197	3198	3199	3200	3201	3202	3203	3204	3205	3206	3207	3208	3209	3210	3211	3212	3213	3214	3215	3216	3217	3218	3219	3220	3221	3222	3223	3224	3225	3226	3227	3228	3229	3230	3231	3232	3233	3234	3235	3236	3237	3238	3239	3240	3241	3242	3243	3244	3245	3246	3247	3248	3249	3250	3251	3252	3253	3254	3255	3256	3257	3258	3259	3260	3261	3262	3263	3264	3265	3266	3267	3268	3269	3270	3271	3272	3273	3274	3275	3276	3277	3278	3279	3280	3281	3282	3283	3284	3285	3286	3287	3288	3289	3290	3291	3292	3293	3294	3295	3296	3297	3298	3299	3300	3301	3302	3303	3304	3305	3306	3307	3308	3309	3310	3311	3312	3313	3314	3315	3316	3317	3318	3319	3320	3321	3322	3323	3324	3325	3326	3327	3328	3329	3330	3331	3332	3333	3334	3335	3336	3337	3338	3339	3340	3341	3342	3343	3344	3345	3346	3347	3348	3349	3350	3351	3352	3353	3354	3355	3356	3357	3358	3359	3360	3
------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	---



```

df_agm: pyspark.sql.dataframe.DataFrame
GEO/TIME: string
Belgium: string
Bulgaria: string
Czech Republic: string
Denmark: string
Germany (until 1990 former territory of the FRG): string
Estonia: string
Ireland: string
Greece: string
Spain: string
France: string
Croatia: string
Italy: string
Cyprus: string
Latvia: string
Lithuania: string
Luxembourg: string
Hungary: string
Malta: string
Netherlands: string
Austria: string
Poland: string

```

Εικόνα 43: <Το είδος των δεδομένων του αρχικού dataframe>

Παρακάτω φαίνεται ένα κομμάτι του πίνακα με τις τιμές:

GEO/TIME	Belgium	Bulgaria	Czech Republic	Denmark	Germany (until 1990 former territory of t	Italy	Malta	Netherlands	Austria	Poland	Portugal	Romania	Slovenia	Slovakia	Finland	Turkey
2006	16,039,090	11,944,694	20,090,348	9,453,026		52,917,094,031	26,886,500	70,017,089	10,555,119	26,842,277	3,242,105	4,332,049	5,058,147	5,003,750	:	:
2007	16,271,311	12,006,786	20,610,186	9,327,579		54,087,749,831	27,952,200	71,518,637	10,918,100	28,669,799	3,586,439	4,707,220	5,115,168	5,328,215	:	:
2008	16,360,702	11,791,454	19,987,022	8,918,197		56,317,581,137	25,267,600	74,730,956	10,173,237	28,126,716	3,359,244	5,085,308	5,144,831	5,502,554	:	:
2009	15,451,017	9,460,922	17,746,893	8,299,403		54,486,550,794	25,013,700	72,224,605	9,609,447	25,024,678	2,667,666	4,656,825	3,707,327	4,890,006	:	:

Εικόνα 44: <Ενδεικτικό κομμάτι του αρχικού dataframe ως πίνακα με τις τιμές του>

Αυτή η μορφή στο αποτέλεσμα οφείλεται στο ότι τα columns είναι πολλά περισσότερα από τα rows και στο output του databricks ο πίνακας έχει αυτή την ‘αποκομμένη’

μορφή. Παρόλα αυτά, περνώντας το αποτέλεσμα στο Notepad έχει την σωστή στοιχισμένη μορφή. Παραθέτω απόσπασμα του πίνακα αφού τα δεδομένα είναι πολλά για να φαίνονται καθαρά σε ένα screenshot:

GEO/TIME	Belgium	Bulgaria	Czech Republic	Denmark	Germany (until 1990 former territory of the FRG)	Estonia	Ireland	Greece	Spain	France	Croatia	Italy	Cyprus	Latvia	Lithuania	Luxembourg	Hungar	
2006	16,039,090	11,944,694	20,090,348	9,453,026		52,947,373	3,020,367	21,652,000	43,055,381	224,518,083	105,865,432	32,858,014	156,861,341	13,310,257	1,872,393	1,514,197	2,399,913	10,045,89
2007	16,271,311	12,006,786	20,610,186	9,327,579		54,485,379	2,915,456		48,081,473	225,450,241	108,567,043	33,701,925	163,465,680	13,197,004	1,935,984	1,609,998	2,328,688	10,170,80
2008	16,360,702	11,791,454	19,987,022	8,918,197		56,239,679	2,932,662		47,973,949	223,756,216	106,993,811	33,902,735	161,797,434	13,208,954	2,115,618	1,626,829	2,249,545	10,009,53
2009	15,451,017	9,460,922	17,746,893	8,299,403		54,096,574	2,740,696		57,413,304	200,551,728	98,705,212	33,357,844	159,493,866	11,666,663	1,699,562	1,395,899	2,075,831	9,220,14
2010	16,169,676	10,547,112	18,365,947	8,981,992		59,658,760	3,203,721		59,184,397	213,349,649	120,390,105	33,234,882	165,202,498	12,448,158	1,912,336	1,571,325	1,717,130	9,358,37
2011	16,723,867	12,426,723	19,424,839	9,491,137		63,081,467	3,748,865		65,514,230	239,369,167	123,227,704	35,389,002	176,474,062	13,112,596	2,257,021	1,883,003	2,057,883	9,920,33
2012	16,432,646	13,451,440	21,793,985	9,608,124		68,161,503	3,823,039	11,839,245	61,054,383	243,389,006	125,038,453	57,079,967	180,594,988	13,488,127	2,429,093	2,680,048	2,298,068	11,392,18
2013	16,511,721	14,370,426	22,144,896	9,914,273		71,191,942	3,909,326	10,871,006	68,992,640	252,447,766	132,251,136	59,378,896	184,793,382	13,152,589	2,639,434	2,906,201	2,313,124	11,982,88
2014	17,068,872	14,077,798	22,110,112	10,608,119		74,805,253	3,919,299	11,276,424	74,675,156	259,635,794	130,908,700	61,072,661	186,792,507	12,884,399	2,875,934	3,033,826	2,513,585	12,351,33
2015	18,852,087	13,352,281	23,286,515	11,171,416		78,827,773	3,770,207	13,462,348	78,254,524	269,418,103	130,464,997	65,683,010	192,607,930	12,550,320	2,873,885	3,010,727	2,655,733	12,962,39

Εικόνα 45: <Ενδεικτικό κομμάτι του αρχικού dataframe ως πίνακα με στοιχισμένες τιμές>

Για όλες τις στήλες που αναφέρονται(στήλες στο csv αρχείο), αντικαθιστούμε τους χαρακτήρες ‘:’ με 0 και παραθέτουμε κομμάτι του ανανεωμένου πίνακα:

```
1 #Specify the columns and replace ':' with '0' in the specified columns
2 #This conversion will be needed after to compare the values so that the minimum and maximum value will be found
3 columns = ['Belgium','Bulgaria','Czech Republic','Denmark','Germany (until 1990 former territory of the FRG)',
4           'Estonia','Ireland','Greece','Spain','France','Croatia','Italy','Cyprus','Latvia','Lithuania',
5           'Luxembourg','Hungary','Malta','Netherlands','Austria','Poland','Portugal','Romania','Slovenia',
6           'Slovakia','Finland','Sweden','United Kingdom','Iceland','Liechtenstein','Norway','Switzerland',
7           'Montenegro','Former Yugoslav Republic of Macedonia, the','Serbia','Turkey']
8 for column in columns:
9     df_agn= df_agn.withColumn(column, when(df_agn[column] == ":", "0") .otherwise(df_agn[column]))
10
11 # Show the modified DataFrame
12 df_agn.show()
```

Εικόνα 46: <Κώδικας για ορισμό των στηλών που θα επεξεργαστούμε, αντικαθιστώντας τους χαρακτήρες ‘:’ με ‘0’>

France	Croatia	Italy	Cyprus	Norway	Switzerland	Montenegro	Former Yugoslav Re
105,865,432	32,858,014	156,861,341	13,310,257	1,921,169	0	0	
108,567,043	33,701,925	163,465,680	13,197,004	1,305,287	0	0	
106,993,811	33,902,735	161,797,434	13,208,954	2,119,045	0	0	
98,705,212	33,357,844	159,493,866	11,666,663	1,500,431	0	0	

**Εικόνα 47:** <Εκτύπωση αποτελέσματος για το αποτέλεσμα της Εικόνας 5>

Αφαιρούμε όπου υπάρχει ‘,’ στην αναπαράσταση μεγάλων αριθμών μετατρέποντας τον πίνακα σε πίνακα ακεραίων και παραθέτουμε την δομή που δείχνει πως τα στοιχεία είναι ακέραιοι με κομμάτι του ανανεωμένου πίνακα:

```

1 # Remove commas and convert to integers so that the comparison can take place after
2 for column in columns:
3     df_agn = df_agn.withColumn(column, regexp_replace(col(column), ",", "").cast(IntegerType()))
4
5 # Show the final clean DataFrame with integer columns
6 df_agn.show()
```

**Εικόνα 48:** <Κώδικας για αφαίρεση του ‘,’ και μετατροπή των τιμών σε ακέραιους>

▼ df\_agn: pyspark.sql.dataframe.DataFrame

```
GEO/TIME: string
Belgium: integer
Bulgaria: integer
Czech Republic: integer
Denmark: integer
Germany (until 1990 former territory of the FRG): integer
Estonia: integer
Ireland: integer
Greece: integer
Spain: integer
France: integer
Croatia: integer
Italy: integer
Cyprus: integer
Latvia: integer
Lithuania: integer
Luxembourg: integer
Hungary: integer
Malta: integer
Netherlands: integer
Austria: integer
Poland: integer
```

Εικόνα 49: <Ο τύπος των δεδομένων ως ακέραιοι>

```
+-----+-----+-----+-----+
+-----+-----+-----+-----+
|GEO/TIME| Belgium|Bulgaria|Czech
|      | Poland|Portugal|Romania|:
+-----+-----+-----+-----+
+-----+-----+-----+-----+
|      | 2006|16039090|11944694|
17089|10555119|26842277|3242105|
|      | 2007|16271311|12006786|
18637|10918100|28669799|3586439|
|      | 2008|16360702|11791454|
30956|10173237|28126716|3359244|
|      | 2009|15451017| 9460922|
24605| 9609447|25024678|2667666|
|      | 2010|16169676|10547112|
85976|10064628|25386084|2766581|
|      | 2011|16723867|12426723|
47325|10620264|27860103|3066882|
|      | 2012|16432646|13451440|
58431|11876599|29033970|3291504|
|      | 2013|16511721|14370426|
```

Εικόνα 50: <Ενδεικτικό κομμάτι του πίνακα με τις επεξεργασμένες 'καθαρές' τιμές>

Επειδή για το 1<sup>ο</sup> ερώτημα μας ενδιαφέρουν μόνο οι τιμές για τα έτη 2007-2014, αφαιρούμε τα columns 2006 και 2015:

```
1 # Remove the 2004 and 2015 so that after that the average of 2005-2014 is calculated
2 df_clean = df_agm.filter((F.col("GEO/TIME") != "2006") & (F.col("GEO/TIME") != "2015"))
3 df_clean.show()
```

**Εικόνα 51:** <Κώδικας για το φιλτράρισμα/περιορισμό ορισμένων στηλών>

Για τον πίνακα που έχουμε δημιουργήσει, υπολογίζουμε την μέση τιμή των columns, δηλαδή τις τιμές στις στήλες που αναπαριστούν τον αριθμό των διανυκτερεύσεων για κάθε χώρα στο διάστημα από 2006-2015(rows) και παραθέτουμε το αποτέλεσμα:

```
1 #Calculate the average for the all the countries
2 avrg = df_clean.agg({col_name: 'avg' for col_name in columns})
3
4 #Set the average result at double precision
5 avrg_columns = [format_number(col(country), 2).alias(country) for country in avrg.columns]
6
7 #Select the average values and print it
8 avrg = avrg.select(avrg_columns)
9 avrg.show(truncate=False) #if truncate=True not all of the avg for all the countries will be printed
```

**Εικόνα 52 :** <Κώδικας για υπολογισμό της μέσης τιμής του πίνακα με ακρίβεια 2 δεκαδικών ψηφίων και εκτύπωση του αποτελέσματος>

Ενδεικτικό μέρος του πίνακα με τα αποτελέσματα:

```
+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+
|avg(Poland) |avg(Liechtenstein)|avg(Germany (until 1990 former territory of the FRG))|avg(Croatia) |avg(Luxembourg)|
lovakia)|avg(United Kingdom)|avg(Czech Republic)|avg(Finland)|avg(Lithuania)|avg(Sweden) |avg(Austria) |avg(Latvia) |
+-----+-----+-----+-----+-----+-----+-----+-----+
|11,090,723.00|149,480.88 |62,715,069.62 |43,389,739.00|2,194,231.75 |
4,543.38 |65,383,005.38 |20,272,985.00 |5,450,927.38|2,088,391.12 |11,371,281.75|74,799,821.50|2,233,122.75|
+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+
```

**Εικόνα 53:** <Εκτύπωση του αποτελέσματος του πίνακα με τον μέσο όρο>

Ο πίνακας με τα τελικά αποτελέσματα:

Country	Average
Poland	11090723.00
Liechtenstein	149480.88
Germany (until 1990 former territory of the FRG)	62715069.62
Croatia	43389739.00
Luxembourg	2194231.75
lovakia	4543.38
United Kingdom	65383005.38
Czech Republic	20272985.00
Finland	5450927.38
Lithuania	2088391.12
Sweden	11371281.75
Austria	74799821.50
Latvia	2233122.75

**Εικόνα 54:** <Ο τελικός πίνακας με τον μέσο όρο διανυκτερεύσεων για όλες τις χώρες στο διάστημα 2007-2014

## 3.2

Αρχείο που περιέχει το κώδικα του ερωτήματος 4 για το θέμα 1: Thema\_3\_Query\_2.txt

Σε αυτό το ερώτημα ζητείται πόσες και ποιες χρονιές ο αριθμός των διανυκτερεύσεων της Ελλάδας ήταν μεγαλύτερος από 5 άλλες ευρωπαϊκές χώρες. Δηλαδή να επιλέξουμε τον τίτλο της κάθε γραμμής για την οποία η τιμή της είναι μικρότερη από αυτή της Ελλάδας και να τις αθροίσουμε.

Ο κώδικας για την επεξεργασία του αρχείου με τις απαραίτητες βιβλιοθήκες:

```
1 # Databricks notebook source
2 #Importing everything required for spark sql
3 import pyspark
4 from pyspark.sql.functions import *
5 from pyspark.sql.functions import when
6 from pyspark.sql.functions import col
7 from pyspark.sql.types import IntegerType
8 from pyspark.sql.functions import col, avg, format_number
9 from pyspark.sql import functions as F
10
11 #Making the dataframes from the tables that were created from each file
12 df_agm = spark.read.option("header",True).csv("/FileStore/tables/tour_occ_ninat-14.csv")
13
14 #Print the schema for each dataframe and show the results
15 df_agm.printSchema()
16
17 #Specify the columns and replace ':' with '0' in the specified columns
18 #This conversion will be needed after to compare the values so that the minimum and maximum value will be found
19 columns = ['Belgium', 'Bulgaria', 'Czech Republic', 'Denmark', 'Germany (until 1990 former territory of the FRG)',
20            'Estonia', 'Ireland', 'Greece', 'Spain', 'France', 'Croatia', 'Italy', 'Cyprus', 'Latvia', 'Lithuania',
21            'Luxembourg', 'Hungary', 'Malta', 'Netherlands', 'Austria', 'Poland', 'Portugal', 'Romania', 'Slovenia',
22            'Slovakia', 'Finland', 'Sweden', 'United Kingdom', 'Iceland', 'Liechtenstein', 'Norway', 'Switzerland',
23            'Montenegro', 'Former Yugoslav Republic of Macedonia, the', 'Serbia', 'Turkey']
24 for column in columns:
25     df_agm = df_agm.withColumn(column, when(df_agm[column] == ":", "0").otherwise(df_agm[column]))
26
27 # Remove commas and convert to integers
28 for column in columns:
29     df_agm = df_agm.withColumn(column, regexp_replace(col(column), ",", "").cast(IntegerType()))
30 # Show the final clean dataframe
31 df_agm.show()
```

**Εικόνα 55:** <Κώδικας με τις απαραίτητες βιβλιοθήκες, τις στήλες και το τελικό dataframe που θα χρησιμοποιηθεί στο ερώτημα>

Δημιουργούμε μια λίστα με 5 τυχαίες χώρες ώστε να βρεθεί για πόσες και ποιες ακριβώς χρονιές (γραμμές/rows) οι τιμές ήταν μικρότερες από αυτές της Ελλάδας σε κάθε χρονιά. Στην συγκεκριμένη περίπτωση, η επιλογή δεν είναι τυχαία ώστε να εμφανίσει χώρες με μικρότερες και μεγαλύτερες τιμές. Ύστερα, απομονώνουμε την στήλη για την Ελλάδα και κάνουμε σύγκριση με κάθε τιμή σε κάθε στήλη. Στην περίπτωση που η τιμή της χώρας που συγκρίνουμε είναι μεγαλύτερη από της Ελλάδας,

τότε θα εμφανίζεται στον πίνακα που δημιουργούμε ως 'Lower', αλλιώς ως 'Higher'. Σε αυτό το σημείο να σημειωθεί πως παρόλο που συγκρίνουμε πρώτα τις υπόλοιπες στήλες και μετά την στήλη της Ελλάδας, η τιμή 'Higher' σημαίνει πως η Ελλάδα έχει υψηλότερη τιμή από την εκάστοτε χώρα, παρόλο που στην στήλη της χώρας αναφέρεται ως 'Higher'. Με την ίδια λογική λειτουργεί και η τιμή 'Lower'.

Στην συνέχεια, μετράμε από αυτό τον πίνακα τις φορές που κάθε χώρα έχει ψηλότερες ή χαμηλότερες τιμές από την Ελλάδα μέσω ενός μετρητή που αυξάνεται κάθε φορά που υπάρχει η τιμή 'Higher' σε κάθε στήλη του πίνακα:

```
1  # List of 5 random countries for comparison with Greece
2  column_of_choice = ['Belgium', 'Austria', 'Czech Republic', 'Turkey', 'Germany (until 1990
   former territory of the FRG)']
3
4  # Select the columns for comparison
5  comparison_df = df_agr.select(['GEO/TIME', 'Greece'] + column_of_choice)
6
7  # Do the comparison, when the value in the table is Higher, it means that Greece's value is
   higher than the country's
8  # If the value in the table is Lower, it means that Greece's value is lower than the country's
9  #By value i mean number of overnight stays
10 comparison_df_higher_lower = comparison_df.select( col('GEO/TIME'),*[when(col(country) > col
   ('Greece'), 'Lower').otherwise('Higher').alias(f'{country}') for country in column_of_choice])
11
12 # Show the table with the comparison
13 comparison_df_higher_lower.show(truncate=False)
14
15 # Calculate the number of times Greece had more overnight stays than each of the 5 random
   countries
16 num_times_higher_list = []
17 for country in column_of_choice:
18     num_times_higher = comparison_df_higher_lower.agg(sum(when(col(country) == 'Higher', 1).
   otherwise(0)).alias(f'NumTimesLower_{country}')).first()[f'NumTimesLower_{country}']
19     num_times_higher_list.append(num_times_higher)
20
21 # Add a row for the total count
22 total_counts = ["Total"] + num_times_higher_list
23
24 # Print the table for the total count
25 total_counts_df = spark.createDataFrame([total_counts], schema=['FinalTotal'] +
   [f'TimesHigherThan_{country}' for country in column_of_choice])
26 total_counts_df.show(truncate=False)
```

**Εικόνα 56:** <Κώδικας τον ορισμό 5 στηλών προς σύγκριση με την Ελλάδα, δημιουργία λίστας με τα αποτελέσματα της σύγκρισης και υπολογισμός του αθροίσματος των τιμών όταν η Ελλάδα έχει μεγαλύτερες τιμές>

Παραθέτω τον πίνακα με τις τιμές ‘Higher’ , ‘Lower’:

GEO/TIME	Belgium	Austria	Czech Republic	Turkey	Germany (until 1990 former territory of the FRG)
2006	Higher	Lower	Higher	Higher	Lower
2007	Higher	Lower	Higher	Higher	Lower
2008	Higher	Lower	Higher	Higher	Lower
2009	Higher	Lower	Higher	Higher	Higher
2010	Higher	Lower	Higher	Higher	Lower
2011	Higher	Lower	Higher	Higher	Higher
2012	Higher	Lower	Higher	Higher	Lower
2013	Higher	Lower	Higher	Lower	Lower
2014	Higher	Lower	Higher	Lower	Lower
2015	Higher	Lower	Higher	Lower	Lower

Εικόνα 57: <Ο πίνακας με τις τιμές σύγκρισης των 5 χωρών με την Ελλάδα>

Παραθέτω τον πίνακα με το άθροισμα των τιμών όταν η Ελλάδα είχε μεγαλύτερες τιμές από την κάθε χώρα:

FinalTotal	TimesHigherThan_Belgium	TimesHigherThan_Austria	TimesHigherThan_Czech Republic	TimesHigherThan_Turkey	TimesHigherThan_Germany (until 1990
Total	10	0	10	7	2

Εικόνα 58: <Ο πίνακας με το άθροισμα των τιμών όταν η Ελλάδα είχε μεγαλύτερες τιμές από την κάθε χώρα >

Τέλος, αφού στο ερώτημα ζητείται όχι μόνο ποιες αλλά και πόσες χρονιές, μετατρέπουμε τον παραπάνω πίνακα ‘FinalTotal’ σε λίστα ώστε να προσθέσουμε τις τιμές του. Για να γίνει αυτό, πρώτα μετατρέπουμε τις τιμές σε ακεραίους και ύστερα τις προσθέτουμε. Το αποτέλεσμα:



```

1  # Extract the values from the DataFrame and convert to a list
2  total_values = total_counts_df.head(1)[0].asDict().values()
3
4  # Convert the values to integers (excluding the first string value)
5  int_values = [int(value) if isinstance(value, int) else 0 for value in total_values]
6
7  # Initialize the sum
8  total_sum = 0
9
10 #The final result must be a number (not a table), so the values of the FinalTotal table from
    above will be added to make a sum as below
11 # Iterate over the columns, excluding the first string value
12 for i, col_name in enumerate(total_counts_df.columns):
13     if i == 0:
14         continue # Skip the first string value
15     # Add the value to the sum
16     total_sum += int_values[i]
17     # Print the value
18     print(f"{col_name}: {int_values[i]}")
19
20 # Print the total sum of years that the number of overnight stays was higher than the 5 random
    European countries
21 print("Sum of overnight stays:", total_sum)

```

**Εικόνα 59:** <Κώδικας που μετατρέπει τον πίνακα με τις αθροίσεις σε λίστα, προσθέτει τις τιμές της λίστας και υπολογίζει το τελικό ολικό άθροισμα >

```

TimesHigherThan_Belgium: 10
TimesHigherThan_Austria: 0
TimesHigherThan_Czech Republic: 10
TimesHigherThan_Turkey: 7
TimesHigherThan_Germany (until 1990 former territory of the FRG): 2
Sum of overnight stays: 29

```

**Εικόνα 60:** <Το αποτέλεσμα με τα αθροίσματα και το τελικό ολικό άθροισμα >

Άρα, συνολικά 29 χρονιές η Ελλάδα είχε περισσότερες διανυκτερεύσεις ανάμεσα σε 5 τυχαίες Ευρωπαϊκές χώρες στο διάστημα 2006-2015.

### 3.3

Αρχείο που περιέχει το κώδικα του ερωτήματος 4 για το θέμα 1: Thema\_3\_Query\_3.txt

Σε αυτό το ερώτημα μας ζητούνται οι χώρες με τον μεγαλύτερο αριθμό διανυκτερεύσεων, δηλαδή ο τίτλος της στήλης που περιέχει τις μεγαλύτερες τιμές για κάθε γραμμή.

Ο κώδικας για την επεξεργασία του αρχείου με τις απαραίτητες βιβλιοθήκες:

```
1 # Databricks notebook source
2 #Importing everything required for spark sql
3 import pyspark
4 from pyspark.sql.functions import *
5 from pyspark.sql.functions import when
6 from pyspark.sql.functions import col
7 from pyspark.sql.types import IntegerType
8 from pyspark.sql.functions import col, avg, format_number
9 from pyspark.sql import functions as F
10
11 #Making the dataframes from the tables that were created from each file
12 df_agm = spark.read.option("header",True).csv("/FileStore/tables/tour_occ_ninat-14.csv")
13
14 #Print the schema for each dataframe and show the results
15 df_agm.printSchema()
16
17 #Specify the columns and replace ':' with '0' in the specified columns
18 #This conversion will be needed after to compare the values so that the minimum and maximum value will be found
19 columns = ['Belgium','Bulgaria','Czech Republic','Denmark','Germany (until 1990 former territory of the FRG)',
20            'Estonia','Ireland','Greece','Spain','France','Croatia','Italy','Cyprus','Latvia','Lithuania',
21            'Luxembourg','Hungary','Malta','Netherlands','Austria','Poland','Portugal','Romania','Slovenia',
22            'Slovakia','Finland','Sweden','United Kingdom','Iceland','Liechtenstein','Norway','Switzerland',
23            'Montenegro','Former Yugoslav Republic of Macedonia, the','Serbia','Turkey']
24 for column in columns:
25     df_agm = df_agm.withColumn(column, when(df_agm[column] == ":", "0") .otherwise(df_agm[column]))
26
27 # Remove commas and convert to integers
28 for column in columns:
29     df_agm = df_agm.withColumn(column, regexp_replace(col(column), ",", "").cast(IntegerType()))
30 # Show the final clean dataframe
31 df_agm.show()
```

**Εικόνα 61:** <Κώδικας με τις απαραίτητες βιβλιοθήκες, τις στήλες και το τελικό dataframe που θα χρησιμοποιηθεί στο ερώτημα>

Σε αυτήν την περίπτωση καλούμαστε να υπολογίσουμε την κάθε μέγιστη τιμή από κάθε στήλη του πίνακα. Δημιουργούμε μια νέα στήλη που περιέχει την μέγιστη τιμή από κάθε στήλη του πίνακα και ύστερα ελέγχουμε μέσω των conditions πως έχουν επιλεγθεί οι μέγιστες τιμές. Δημιουργούμε έναν πίνακα που περιέχει το κάθε έτος, τον τίτλο της κάθε στήλης στην οποία βρέθηκε η μέγιστη τιμή (δηλαδή το όνομα της χώρας) και την μέγιστη τιμή:

```

1 # Define a new column that contains the title of the column with the max value of overnight stays
2 max_column_expr = greatest(*[col(column) for column in columns[1:]])
3
4 # Use a loop to create conditions for each column that checks that the value of the column has the max value
5 conditions = [col(column) == max_column_expr for column in columns[1:]]
6
7 # Use the when and otherwise functions to assign the title of the max column
8 #The final result will be a table with each year(GEO/TIME), each year's max number of overnight stays(MaxValue),
9 #and the European country with the max number of overnight stays(MaxColumnName)
10 df_result = df_aggr.withColumn("MaxColumnName", col("GEO/TIME"))
11 for condition, column in zip(conditions, columns[1:]):
12     df_result = df_result.withColumn("MaxColumnName", when(condition, column).otherwise(col("MaxColumnName")))
13
14 # Select the 'GEO/TIME', 'MaxColumnName', and the column with the maximum value
15 max_values_df = df_result.select('GEO/TIME', 'MaxColumnName', greatest(*[col(country) for country in columns[1:]]).alias('MaxValue'))
16
17 # Show the DataFrame with 'GEO/TIME', 'MaxColumnName', and the corresponding column with the maximum value
18 max_values_df.show(truncate=False)

```

**Εικόνα 62:** <Κώδικας που δημιουργεί νέα στήλη με την μέγιστη τιμή της κάθε στήλης και την δημιουργία ενός πίνακα με το κάθε έτος, την μέγιστη τιμή και μια στήλη με τον τίτλο της στήλης με την μέγιστη τιμή >

```

+-----+-----+-----+
|GEO/TIME|MaxColumnName|MaxValue |
+-----+-----+-----+
|2006    |Spain        |224518083|
|2007    |Spain        |225450241|
|2008    |Spain        |223756216|
|2009    |Spain        |200551728|
|2010    |Spain        |213349649|
|2011    |Spain        |239369167|
|2012    |Spain        |243389006|
|2013    |Spain        |252447766|
|2014    |Spain        |259635794|
|2015    |Spain        |269418103|
+-----+-----+-----+

```

**Εικόνα 63:** <Ο πίνακας με το άθροισμα των τιμών όταν η Ελλάδα είχε μεγαλύτερες τιμές από την κάθε χώρα >

Επειδή για το συγκεκριμένο dataset είναι προφανής η πρώτη θέση της Ισπανίας συγκεκριμένα, αλλάζω την λίστα με τις στήλες(δηλαδή χώρες) για τις οποίες θα βρεθεί η μέγιστη τιμή. Συγκεκριμένα αφαιρώ την Ισπανία, την Γαλλία και την Ιταλία:

```

1 #Changing the Countries list so that it is calculated for other countries than Spain, France
  and Italy, that had the second max number of overnight stays
2 new_columns = ['Belgium','Bulgaria','Czech Republic','Denmark','Germany (until 1990 former
  territory of the FRG)','Estonia','Ireland','Greece','Croatia','Cyprus','Latvia','Lithuania',
  'Luxembourg','Hungary','Malta','Netherlands','Austria','Poland','Portugal','Romania',
  'Slovenia','Slovakia','Finland','Sweden','United Kingdom','Iceland','Liechtenstein','Norway',
  'Switzerland','Montenegro','Former Yugoslav Republic of Macedonia, the','Serbia','Turkey']
3 # Define a new column that contains the title of the column with the max value of overnight
  stays
4 max_column_expr = greatest(*[col(column) for column in new_columns[1:]])
5
6 # Use a loop to create conditions for each column that checks that the value of the column has
  the max value
7 conditions = [col(column) == max_column_expr for column in new_columns[1:]]
8
9 # Use the when and otherwise functions to assign the title of the max column
10 #The final result will be a table with each year(GEO/TIME), each year's max number of
  overnight stays(MaxValue),
11 #and the European country with the max number of overnight stays(MaxColumnName)
12 df_result = df_agn.withColumn("MaxColumnName", col("GEO/TIME"))
13 for condition, column in zip(conditions, new_columns[1:]):
14     df_result = df_result.withColumn("MaxColumnName", when(condition, column).otherwise(col
      ("MaxColumnName")))
15
16 # Select the 'GEO/TIME', 'MaxColumnName', and the column with the maximum value
17 max_values_df = df_result.select('GEO/TIME', 'MaxColumnName', greatest(*[col(country) for
  country in new_columns[1:]]).alias('MaxValue'))
18
19 # Show the DataFrame with 'GEO/TIME', 'MaxColumnName', and the corresponding column with the
  maximum value
20 max_values_df.show(truncate=False)

```

**Εικόνα 64:** <Αφαίρεση ορισμένων στηλών και και υλοποίηση του ίδιου task με τον κώδικα της Εικόνας 2 >

```

+-----+-----+-----+
|GEO/TIME|MaxColumnName|MaxValue |
+-----+-----+-----+
|2006    |United Kingdom|90740612 |
|2007    |United Kingdom|84690331 |
|2008    |United Kingdom|80393221 |
|2009    |United Kingdom|79920519 |
|2010    |United Kingdom|84610527 |
|2011    |United Kingdom|87993917 |
|2012    |United Kingdom|105455528|
|2013    |Turkey        |89594261 |
|2014    |Turkey        |97581075 |
|2015    |Turkey        |96400316 |
+-----+-----+-----+

```

**Εικόνα 65:** <Ο πίνακας με την ανανεωμένη λίστα από στήλες με το άθροισμα των τιμών όταν η Ελλάδα είχε μεγαλύτερες τιμές από την κάθε χώρα >

### 3.4

Αρχείο που περιέχει το κώδικα του ερωτήματος 4 για το θέμα 1: Thema\_3\_Query\_4.py

Σε αυτό το ερώτημα μας ζητείται η χρονιά με τον μικρότερο αριθμό διανυκτερεύσεων για κάθε χώρα, δηλαδή ο τίτλος της γραμμής με την μικρότερη τιμή για κάθε στήλη.

Ο κώδικας για την επεξεργασία του αρχείου:

```
1  # Databricks notebook source
2  #Importing everything required for spark sql
3
4  import pyspark
5  from pyspark.sql.functions import *
6  from pyspark.sql.types import IntegerType
7  from pyspark.sql.functions import col, avg, format_number,min, when, least
8
9
10 #Making the dataframes from the tables that were created from each file
11 df_agn = spark.read.option("header",True).csv("/FileStore/tables/tour_occ_ninat-14.csv")
12
13 #Print the schema for each dataframe and show the results
14 df_agn.printSchema()
15
16 # Replace ':' with '0' in the specified columns
17 columns = ['Belgium','Bulgaria','Czech Republic','Denmark','Germany (until 1990 former territory of the FRG)',
18            'Estonia','Ireland','Greece','Spain','France','Croatia','Italy','Cyprus','Latvia','Lithuania',
19            'Luxembourg','Hungary','Malta','Netherlands','Austria','Poland','Portugal','Romania','Slovenia',
20            'Slovakia','Finland','Sweden','United Kingdom','Iceland','Liechtenstein','Norway','Switzerland',
21            'Montenegro','Former Yugoslav Republic of Macedonia, the','Serbia','Turkey']
22 for column in columns:
23     df_agn= df_agn.withColumn(column, when(df_agn[column] == ":", "0") .otherwise(df_agn[column]))
24
25 # Remove commas and convert to integers
26 for column in columns:
27     df_agn = df_agn.withColumn(column, regexp_replace(col(column), ",", "").cast(IntegerType()))
28 # Show the final clean dataframe
29 df_agn.show()
30
```

**Εικόνα 66:** <Κώδικας με τις απαραίτητες βιβλιοθήκες, τις στήλες και το τελικό dataframe που θα χρησιμοποιηθεί στο ερώτημα>

Το dataframe που έχει διαμορφωθεί έχει ορισμένες τιμές ίσες με 0, αφού θεωρούμε βάσει του αρχείου πως δεν έχουν γίνει μετρήσεις εκείνη την χρονική περίοδο. Για αυτόν τον λόγο, επειδή δεν είναι αντικειμενική η θεώρηση εκείνης της περιόδου ως περίοδο με την ελάχιστη επισκεψιμότητα, στο πρώτο βήμα επιλέγουμε τις τιμές από κάθε στήλη που δεν περιέχουν τιμή 0. Έτσι, υπολογίζουμε την ελάχιστη τιμή για κάθε

στήλη, δηλαδή για κάθε χώρα και ελέγχουμε εάν κάποια χώρα/στήλη δεν είχε καμία τιμή, ώστε να την αποκλείσουμε σε αυτή την περίπτωση (συγκεκριμένα στο αρχείο υπάρχει μια χώρα που όλες οι τιμές της ήταν ίση με 0). Τέλος εκτυπώνεται η στήλη και ο τίτλος της κάθε στήλης από τον πίνακα με τις ελάχιστες τιμές ανά χώρα:

```
1 # Define a new column that contains the title of the column with the min value
2 #We exclude zero as a value because some of the values are converted from NULL to zero, so this is not representative
3 min_column_expr = least(*[when(col(column) != 0, col(column)) for column in columns[1:]])
4
5 #Calculate the overall min value of each column(Country) and store it in a dictionary (MinColumnTitle)
6 overall_min_values = df_agr.select([min(when(col(column) != 0, col(column))).alias(column) for column in columns[1:]])
7 min_column_titles = {}
8
9 # Iterate through each column to find each MinColumnTitle
10 for column in columns[1:]:
11     min_value = overall_min_values.select(column).first()[0]
12
13     # Check if min_value is not None before accessing each value
14     if min_value is not None:
15         min_column_title = df_agr.filter((col(column) == min_value) & (col(column) != 0)).select("GEO/TIME").first()
16
17         # Check if the result is not None before accessing the element
18         #We do it for the case that there are no values for a specific country(eg Switzerland)
19         if min_column_title is not None:
20             min_column_titles[column] = min_column_title[0]
21
22 #Print the dictionary with column names and their corresponding MinColumnTitle
23 print("MinColumnTitles:")
24 for column, title in min_column_titles.items():
25     print(f"{column}: {title}")
```

**Εικόνα 67:** <Κώδικας που υπολογίζει τις ελάχιστες τιμές των στηλών του πίνακα και επιλέγει τον τίτλο της κάθε γραμμής στην οποία βρίσκεται η ελάχιστη τιμή>

Ενδεικτικό κομμάτι από το αποτέλεσμα:

```
MinColumnTitles:
Bulgaria: 2009
Czech Republic: 2009
Denmark: 2009
Germany (until 1990 former territory of the FRG): 2006
Estonia: 2009
Ireland: 2013
Greece: 2006
Spain: 2009
France: 2009
Croatia: 2006
Italy: 2006
Cyprus: 2009
Latvia: 2009
Lithuania: 2009
Luxembourg: 2010
Hungary: 2009
Malta: 2009
Netherlands: 2009
```

**Εικόνα 68:** <Ορισμένες χώρες με την χρονολογία με την μικρότερη τιμή >

Για να σιγουρευτούμε πως όντως υπολογίζει σωστά τις τιμές παρακάτω παραθέτω έναν πίνακα με τις τιμές κάθε χώρας την εκάστοτε χρονιά, η οποία χρονιά φαίνεται στον δεύτερο πίνακα που παραθέτω παρακάτω:

```
1 # Define a new column that contains the title of the column with the min value
2 min_values_expr = [min(when(col(column) != 0, col(column))).alias(column) for column in columns[1:]]
3
4 # Select the minimum value for each column
5 min_values = df_agr.select(min_values_expr)
6
7 # Display the DataFrame with the minimum values
8 print("MinValues:")
9 min_values.show(truncate=False)
```

**Εικόνα 69:** Κώδικας που εμφανίζει τις ελάχιστες τιμές σε πίνακα >

```
MinValues:
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|Bulgaria|Czech Republic|Denmark|Germany (until 1990 former territory of the FRG)|Estonia|Ireland |
|omania|Slovenia|Slovakia|Finland|Sweden |United Kingdom|Iceland|Liechtenstein|Norway |Switzerland|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|9460922 |17746893      |8299403|52947373      |2740696|10871806|
667666|4332049 |3707327 |4890006|10920106|79920519      |1687744|110465      |7500431|NULL      |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
```

**Εικόνα 70:** <Το αποτέλεσμα με τον πίνακα με τις ελάχιστες τιμές κάθε στήλης >

Από το παραπάνω αποτέλεσμα μπορούμε να απαντήσουμε στο ερώτημα. Παρόλα αυτά, μετατρέπουμε το παραπάνω dictionary σε έναν πίνακα (απλά αλλάζουμε την δομή):

```
from pyspark.sql import Row

# Convert MinColumnTitles dictionary into a dataframe that is created with a single row
min_titles_row = Row(*min_column_titles)
min_titles_df = spark.createDataFrame([min_titles_row])

# Display the DataFrame with MinColumnTitles
print("MinColumnTitles:")
min_titles_df.show(truncate=False)
```

**Εικόνα 71:** <Κώδικας που δημιουργεί πίνακα με τον τίτλο της γραμμής (χρονολογίες) οι οποίες αντιστοιχούν στις ελάχιστες τιμές της κάθε στήλης του αρχικού πίνακα >

	Bulgaria	Czech Republic	Denmark	Germany (until 1990 former territory of the FRG)	Estonia	Ireland	Greece	Finland	Sweden	United Kingdom	Iceland	Liechtenstein	Norway	Montenegro	Former Yugoslav Republic of Macedonia
2009	2009	2009	2006												
2009	2008	2009		2006	2015		2009	2011	2010						

Ο πίνακας με τις τελικές τιμές:

Bulgaria	Czech Republic	Dominican Republic	Germany (until 1900 former territory of the FRG)	Honduras	Ireland	Greece	Spain	France	Croatia	Italy	Cyprus	Latvia	Lithuania	Luxembourg	Hungary	Malta	Netherlands	Austria	Poland	Portugal	Romania	Slovenia	Slovakia	Finland	Sweden	United Kingdom	Iceland	Liechtenstein	Norway	Montenegro	Former Yugoslav Republic of Macedonia	Serbia	Turkey			
2000	2005		2000	2006			2000	2003	2006	2009	2000	2006	2000	2000	2000	2000	2000	2000	2000	2000	2000	2000	2000	2000	2000	2000	2000	2000	2000	2000	2001	2000	2011	2010	2012	2013

**Εικόνα 73:** <Ο τελικός πίνακας με τις χρονιές με τον μικρότερο αριθμό διανυκτερεύσεων για κάθε χώρα >