ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA

Curso Académico 2022/2023

Trabajo Fin de Grado

INGENIERÍA DE DATOS CON EL FRAMEWORK DE BIG DATA SPARK Y SCALA

Autor: Álvaro Sánchez Pérez

Directores: Juan Manuel Serrano Hidalgo

Índice

1. Introducción

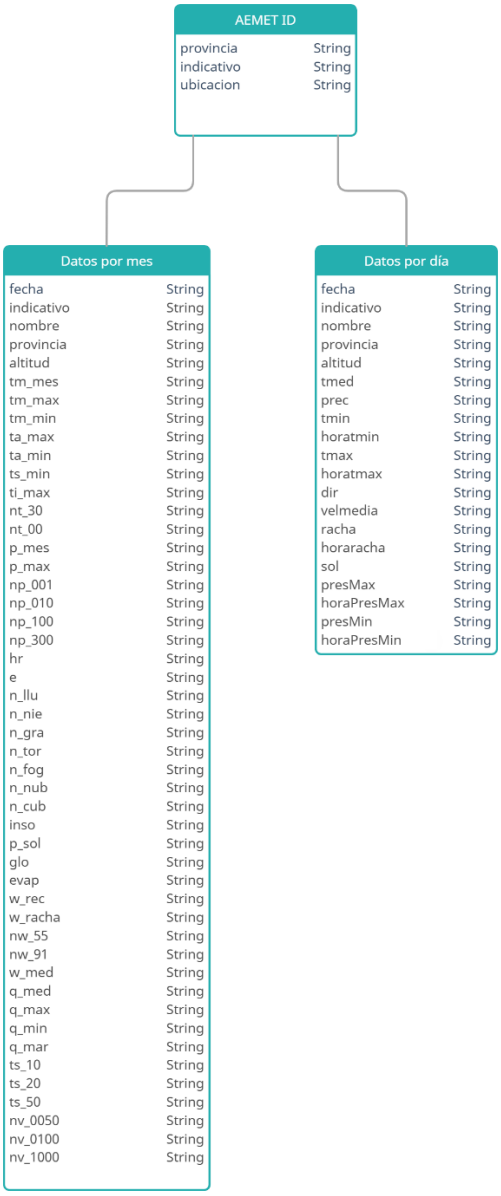
Prueba

2. Objetivos

3. Descripción informática

1. Fuentes de datos

Los datos meteorológicos, han sido obtenidos a través de la página de *AEMET OpenData*, se tratan tanto de datos que muestran la información de forma mensual como de forma diaria desde el año 2007 hasta el año actual 2022.



Resaltar que el diagrama mostrado, aparecen los id de las columnas y valores que deberían de encontrase en estas mediante los datos obtenidos, ya que así nos lo indican los metadatos que nos proporcionan. Pero en cambio tanto el nombre, como la provincia no aparecen en la mayoría de los archivos de datos que nos proporcionan, y por ello debemos de hacer uso de la tabla *AEMET ID*, que gracias a esta podremos relacionar mediante el indicativo de que provincia y ubicación se trata.

La estructura en la que vienen estos datos se puede observar en el anterior diagrama, todos los valores vienen en formato String, los cuales transformaremos mediante el casteo correspondiente gracias a Spark, para así poder trabajar posteriormente con ellos de una manera más cómoda. Deberemos de realizar lo siguiente para la transformación.

1. Primero creamos el esquema correspondiente a los datos, en este ejemplo lo realizaremos con los datos por día. En este esquema, le indicaremos tanto en nombre de la columna, como el tipo de dato que se trata, en nuestro caso todos String, y por último si la columna puede contener valores nulos.

```
val schema = StructType(
    Array(
        StructField("fecha", StringType, true),
        StructField("indicativo", StringType, true),
        StructField("nombre", StringType, true),
        StructField("provincia", StringType, true),
        StructField("altitud", StringType, true),
        StructField("tmed", StringType, true),
        StructField("prec", StringType, true),
        StructField("tmin", StringType, true),
        StructField("horatmin", StringType, true),
        StructField("tmax", StringType, true),
        StructField("horatmax", StringType, true),
        StructField("dir", StringType, true),
        StructField("velmedia", StringType, true),
        StructField("racha", StringType, true),
        StructField("horaracha", StringType, true),
        StructField("sol", StringType, true),
        StructField("presMax", StringType, true),
        StructField("horaPresMax", StringType, true),
        StructField("presMin", StringType, true),
        StructField("horaPresMin", StringType, true)
    )
)
```

2. A continuación, realizaremos la lectura de los datos y casteo de los datos.

```
val allData = spark.read.schema(schema).json("D:/TFGAlvaroSanchez/data/dayJSONLine/*.json")
    .withColumn("fecha", $"fecha".cast(DateType))
    .withColumn("altitud", $"altitud".cast(IntegerType))
    .withColumn("tmed", func.regexp_replace($"tmed", ",", ".").cast(DoubleType))
    .withColumn("prec", func.regexp_replace($"prec", ",", ".").cast(DoubleType))
    .withColumn("tmin", func.regexp_replace($"tmin", ",", ".").cast(DoubleType))
    .withColumn("tmax", func.regexp_replace($"tmax", ",", ".").cast(DoubleType))
    .withColumn("dir", $"dir".cast(IntegerType))
    .withColumn("velmedia", func.regexp_replace($"velmedia", ",", ".").cast(DoubleType))
    .withColumn("racha", func.regexp_replace($"racha", ",", ".").cast(DoubleType))
    .withColumn("sol", func.regexp_replace($"sol", ",", ".").cast(DoubleType))
    .withColumn("presMax", func.regexp_replace($"presMax", ",", ".").cast(DoubleType))
    .withColumn("horaPresMax", $"horaPresMax".cast(IntegerType))
    .withColumn("presMin", func.regexp_replace($"presMin", ",", ".").cast(DoubleType))
    .withColumn("horaPresMin", $"horaPresmin".cast(IntegerType))
```

2.1. Para la lectura le proporcionaremos el esquema creado anteriormente en la variable *schema*, y a su vez le indicamos en la ubicación donde se encuentran los datos, además del formato de fichero.

```
spark.read.schema(schema).json("D:/TFGAlvaroSanchez/data/dayJSONLine/*.json")
```

2.2. Finalmente, modificaremos el tipo de dato de las columnas que lo requieran. Usaremos *withColumn* donde primero le indicaremos el nuevo nombre de la columna que nos generará, en este caso usaremos el mismo nombre de la comuna que vamos a modificar para que así ocurra una sustitución de esta, y en la segunda parte será donde le indiquemos la columna con la que queremos trabajar y el cambio que deseamos.

```
.withColumn("fecha", $"fecha".cast(DateType))
```

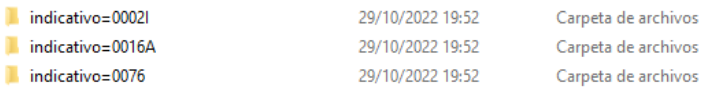
He de mencionar que, los datos que se tuvieron que transformar a tipo Double se debió realizar una modificación previa, sustituyendo las comas por puntos, ya que de otra manera no resultaba posible realizar el casteo debido a que para que se considere un numero decimal debe de venir separada la parte entera de la decimal mediante un punto.

```
.withColumn("tmed", func.regexp_replace($"tmed", ",", ".").cast(DoubleType))
```

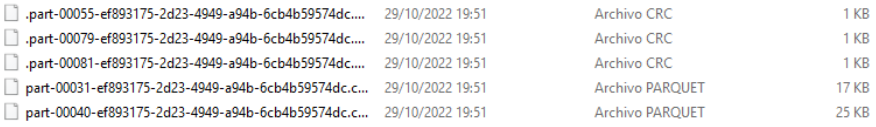
3. Por último, guardaremos los datos en formato Parquet, particionados mediante el campo *indicativo.*

```
allData.write.format("parquet").partitionBy("indicativo").mode("overwrite").save("D:/TFGAlvaroSanchez/data/dayParquet/")
```

Quedando particionado en memoria de la siguiente manera:



Y dentro de cada carpeta algo similar a lo siguiente:



Respecto a los datos mensuales nos quedaríamos únicamente con las siguientes columnas.

```
+----------+----------+-------+-------+----+-----+------+--------+------+-----+-----+-----------+------+-----+------+-------+--
---+-----+------+-----+-----+-----+------+------+------+------+
|     fecha|indicativo|  p_max|    glo|  hr|nw_55|tm_min|  ta_max|ts_min|nt_30|n_des|    w_racha|np_100|nw_91|np_001| ta_min|w_
rec|    e|np_300|p_mes|w_med|nt_00|ti_max|tm_mes|tm_max|np_010|
+----------+----------+-------+-------+----+-----+------+--------+------+-----+-----+-----------+------+-----+------+-------+--
---+-----+------+-----+-----+-----+------+------+------+------+
|2014-01-01|      B954|7.5(16)|26493.0|73.0|   12|   9.2|21.4(03)|  13.2|    0|    1|28/25.8(04)|     0|    1|     6|3.8(31)|
349|115.0|     0| 26.3| 15.0|    0|  14.6|  13.3|  17.4|     4|
|2014-02-01|      B954|4.1(09)|33644.0|72.0|   12|   8.3|20.4(13)|  13.0|    0|    4|26/21.7(10)|     0|    0|     9|3.0(03)|
457|110.0|     0| 10.8| 19.0|    0|  14.4|  13.0|  17.7|     3|
```

De las que podríamos destacar las siguientes: *tm_mes* la cual nos muestra una temperatura media mensual de la ubicación, *tm_max* y *tm_min* siendo las temperaturas medias máximas y mínimas respectivamente, *ta_max* y *ta_min* como las temperaturas máximas y mínimas absolutas del mes y *p_mes* la cual nos indica precipitación total en ese mes. En caso de querer obtener información acerca de alguna otra variable, se puede acceder a esta de las siguientes maneras: a través del fichero *metadataMonth.json* incluido en la carpeta *data* del proyecto o realizando cualquier consulta sobre climatologías mensuales/anuales en la página de *AEMET OpenData* donde obtendrá un enlace acerca de los metadatos donde encontrar este tipo de información.

En relación con los datos diarios, tendríamos lo siguiente:

```
+----------+----------+--------------------+---------+-------+----+----+----+--------+----+--------+---+--------+-----+---------
+---+-------+-----------+-------+-----------+
|     fecha|indicativo|              nombre|provincia|altitud|tmed|prec|tmin|horatmin|tmax|horatmax|dir|velmedia|racha|horaracha
|sol|presMax|horaPresMax|presMin|horaPresMin|
+----------+----------+--------------------+---------+-------+----+----+----+--------+----+--------+---+--------+-----+---------
+---+-------+-----------+-------+-----------+
|2010-01-01|      1387|           A CORUÑA| A CORUÑA|     58| 9.7| 3.3| 6.2|   05:30|13.2|   12:40| 23|     3.9| 10.8|   Varias
|3.3|   null|       null|   null|       null|
|2010-01-01|     1387E|A CORUÑA AEROPUERTO| A CORUÑA|     98| 7.8| 0.7| 4.0|   08:58|11.5|   13:17| 23|    null| 10.3|    18:03
|3.9| 1003.1|         23|  987.4|          4|
```

En este caso nos hemos quedado con todas las columnas, de las que resaltaríamos las siguientes: *tmed* la cual muestra la temperatura media diaria, *prec* que nos ofrece información acerca de la precipitación diaria, tmax *y tmin* las cuales nos muestran la temperatura máxima y mínima diaria respectivamente, y *horatmax* y *horatmin* que, de forma correspondiente, presentan la hora y minuto de la temperatura máxima y mínima. En caso de querer obtener información acerca de alguna otra variable, se puede acceder a esta de las siguientes maneras: a través del fichero *metadataDay.json* incluido en la carpeta *data* del proyecto o realizando cualquier consulta sobre climatologías diarias en la página de *AEMET OpenData* donde obtendrá un enlace acerca de los metadatos donde encontrar este tipo de información

He de mencionar que, a la hora de descargar los datos se encuentran en un formato JSON multilínea, el cual no llega a ser lo suficientemente óptimo para la lectura con Apache Spark. Por lo tanto, se realizaron cambios en todos los ficheros transformándolos a ficheros JSON que contuvieran toda la información en una única línea, esto se llevó a cabo mediante el siguiente comando utilizando la consola de Windows: **FOR %a IN (../data/*.json) DO jq . -c "%a" > "../JSONLine/%a”**. También cabe destacar que los datos una vez leídos en este formato JSON, la mayoría fueron transformados al formato Parquet, particionados por el indicativo de cada estación, ya que resulta más óptimo para la realización de consultas con Spark.

1.1. Obtención de los datos

Para la obtención de los datos, se crearon unos pequeños programas en el lenguaje Java. Estos programas, se basan en la simulación de los pasos que deberíamos de seguir para la descarga y los podemos encontrar en las carpetas *Descar*gaDatosPorDias y *DescargaDatosPorMeses*. Para la simulación de estos pasos, se hizo uso de la librería *Selenium*, a través de la cual se puede manejar un navegador web pudiendo realizar diversas acciones sobre la página mostrada. Por lo tanto, de manera resumida, los pasos que realizarían ambos programas para la descarga serían los siguientes.

```
WebDriver driver = new ChromeDriver();
driver.get(baseUrl);
```

Abriríamos un nuevo navegador donde accederemos al sitio web de *AEMET OpenData*. En nuestro caso le pasamos la URL del sitio web a través de la variable *baseURL*.

```
driver.findElement(By.id("apikey")).sendKeys(apiKey);
desplegable1 = new Select(driver.findElement(By.id("clim1")));
desplegable1.selectByIndex(provincia);
```

Una vez dentro de esta página, el primer paso sería introducir la clave API correspondiente a nuestro usuario, está la podremos solicitar también a través de la misma página. Posteriormente, dependiendo del tipo de información que quisiéramos obtener, ya fuera información por días o por meses, buscaríamos los desplegables y elementos necesarios mediante su *id* o *xpath*.

Después de seleccionar la información deseada, se nos abriría una nueva página donde nos proporciona diferentes tipos de información, además del enlace de la página donde se encontrarán los datos deseados.

En caso de que la consulta a los datos correspondientes se haya realizado con éxito, accederemos al nuevo enlace que nos muestra, donde obtendremos la información y la guardaremos en un nuevo fichero con el nombre correspondiente a la consulta.

```
String texto = driver.findElement(By.xpath("//pre[contains(@style,'word-wrap')]")).getText();
if(texto.contains("\"descripcion\" : \"exito\"")) {
    String urlDatos = texto.split( regex: "\"")[9];

    driver.get(urlDatos);
    texto = driver.findElement(By.xpath("//pre[contains(@style,'word-wrap')]")).getText();

    //Imprimimos la informacion en un fichero externo
    PrintWriter printWriter = null;
    String ubicacionGuardar = "D:\\TFGAlvaroSanchez\\data\\day\\";
    String nombreFichero = ubicacionGuardar.concat(estacionMeterologica).concat(" (").concat(ano).cor

    try {
        printWriter = new PrintWriter(nombreFichero);
    } catch (FileNotFoundException e) {
        System.out.println("Unable to locate the fileName: " + e.getMessage());
    }
    Objects.requireNonNull(printWriter).println(texto);
    printWriter.close();
}
```

Por último, cerramos todas las pestañas y navegadores que se hubieran abierto, y procederíamos a realizar los mismos pasos con otro rango de fechas o en otra estación meteorológica, el objetivo es obtener la información de todas las estaciones existentes entre el rango de fechas establecido.

He de destacar también, que al comienzo del código existen una serie de variables que pueden ser cambiadas por los usuarios. Desde introducir su correspondiente clave API, cambiar el driver de Chrome en caso de que se esté usando una versión diferente del navegador, o cambiar las fechas en caso de que se quisieran obtener datos en un rango diferente.

```
String apiKey = "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJhLnNhbmNoZXpwZS4yMDE5QGFsdW1ub3MudXJqYy5lcyIsImp0aSI6ImRhYTliMDk2LWI5OTgtNGY5Yi05
String baseUrl = "https://opendata.aemet.es/centrodedescargas/productosAEMET?";
System.setProperty("webdriver.chrome.driver", "D:\\TFGAlvaroSanchez\\DescargaDatosPorDias\\chromedriver_win32\\chromedriver.exe");
int anoInicio = 2009;
int anoFin = 2009;
```

2. Programación de queries

Año de la temperatura máxima promedio en cada mes

En esta consulta, realizaremos una comparación del mismo mes en diferentes años, para saber en qué año se registró la temperatura máxima promedia entre todas las estaciones en este mes. Por último, lo representaremos tanto mediante una tabla, como de manera gráfica para poder observar las diferencias de manera visual entre los distintos años.

Para realizar esta consulta utilizaremos los datos meteorológicos mensuales, ya que nos ofrecen la temperatura máxima registrada en cada mes de una manera bastante sencilla. Por lo tanto, el primer paso que deberíamos de realizar sería la lectura de estos datos.

A continuación, deberíamos de agrupar estos datos por fecha, ya que, al realizar la lectura, tenemos la información de muchas estaciones meteorológicas diferentes en el mismo mes y año, por lo tanto, los agruparemos haciendo que coincida la fecha, y realizando una media de la temperatura máxima en todas las estaciones. Por último, dividiremos los datos por meses y ordenándolos de manera descendente a través de nuestro objetivo, la temperatura máxima, ya que así estaríamos consiguiendo obtener el año, en el cual este mes fue el más caluroso.

3. Visualización de queries

Año de la temperatura máxima promedio en cada mes

```
import org.apache.spark.sql.expressions.Window

val data = spark.read.parquet("D:/TFGAlvaroSanchez/data/monthParquet/*").na.drop()

val window = Window.partitionBy("mes").orderBy($"ta_max".desc)

val dataWindow = data
    .withColumn("ta_max", func.split($"ta_max", "\\(")(0).cast(DoubleType))
    .groupBy($"fecha")
    .agg(func.avg($"ta_max").alias("ta_max"))
    .select(func.month($"fecha").alias("mes"), func.year($"fecha").alias("año"), $"ta_max")
    .withColumn("dense_rank", func.dense_rank().over(window))
    .filter($"dense_rank" === 1)
```

Como se puede observar, tal y como se describió anteriormente, el primer paso que realizamos es la lectura de los datos, los cuales los estamos leyendo en un formato Parquet para que resulte más óptimo y además estamos eliminando las filas que contienen valores nulos mediante la función *na.drop()*, la lectura de estos datos la almacenamos en la variable *data*. Al realizar la lectura de estos, se encontrarían de la siguiente manera.

```
+----------+--------+
|     fecha|  ta_max|
+----------+--------+
|2020-01-01|19.6(31)|
|2020-02-01|24.3(03)|
|2020-03-01|21.0(19)|
|2020-04-01|23.7(09)|
```

```
val window = Window.partitionBy("mes").orderBy($"ta_max".desc)
```

Por otro lado, también nos creamos la función ventana en la variable *window* que utilizaremos posteriormente. Le indicaremos la manera de la cual deseamos que se dividan los datos, en nuestro caso será a través de los meses y a su vez le señalamos la manera en la que se deberán de ordenar, teniendo en cuenta la temperatura máxima.

```
val dataWindow = data
    .withColumn("ta_max", func.split($"ta_max", "\\(")(0).cast(DoubleType))
```

A continuación, procedemos a realizar la consulta que almacenaremos en la variable *dataWindow*. El primer paso, que realizamos con los datos es la modificación de la columna de

temperatura máxima, ya que esta nos viene con el siguiente formato temperaturaMáximaAlcanzada(díaDelMesEnElQueSeAlcanzó) por lo que Spark estaría tratando esta columna como un String, cuando nuestro objetivo es manejarlo como un tipo Double. Para esto realizamos lo siguiente, separamos mediante la función *Split* la temperatura máxima del día en que se alcanzó, y nos quedamos únicamente con la temperatura máxima, por último, realizamos un casteo al tipo de datos que deseamos.

```
.groupBy($"fecha")
.agg(func.avg($"ta_max").alias("ta_max"))
```

El segundo paso, de la consulta seria agrupar los datos de las diferentes estaciones por fecha, para ello utilizamos la función *groupBy* indicándole la columna a través de la cual deseamos realizar la agrupación. Mediante la función *agg* y *avg*, obtenemos y calculamos la media de la temperatura máxima de las filas que hemos agrupado. Actualmente tendríamos el DataFrame que se muestra a continuación, que contendría únicamente las fechas y sus respectivas temperaturas.

```
+----------+------------------+
|     fecha|            ta_max|
+----------+------------------+
|2013-01-01| 20.84285714285715|
|2010-06-01| 32.37096774193548|
|2021-03-01|  25.4304347826087|
|2016-10-01|28.869696969696975|
|2019-01-01| 18.59714285714286|
```

```
.select(func.month($"fecha").alias("mes"), func.year($"fecha").alias("año"), $"ta_max")
```

En el siguiente paso, obtendremos tanto el mes y el año, a partir de la fecha. Para esto utilizamos las funciones *month* y *year* con las cuales obtenemos tanto el mes y el año respectivamente. Mediante la función *select* seleccionaremos únicamente las columnas que vamos a necesitar.

```
.withColumn("dense_rank", func.dense_rank().over(window))
```

Para ir finalizando, añadimos una columna con el valor que nos devuelve la función *dense_rank*, que hace uso de la ventana *window* anteriormente definida. Mediante esta función, obtenemos la posición en la que se encuentra un valor teniendo en cuenta la expresión *orderBy* establecida en la función ventana (*window*). Por lo tanto, en nuestro nos devolverá una clasificación de los años más calurosos dividido por meses, ya que era la partición que se estableció en la función *window*. Quedando la consulta de la siguiente manera después de aplicar la función ventana.

mes	año	ta_max	dense_rank
7	2022	39.46785714285714	1
7	2015	37.08	2
7	2016	37.00499999999999	3
7	2020	36.78799999999999	4
7	2017	36.67575757575758	5
7	2013	35.952777777777776	6
7	2021	35.90769230769231	7
7	2019	35.85454545454545	8
7	2012	35.5258064516129	9
7	2014	35.071875000000006	10
7	2010	34.84375	11
7	2018	34.155882352941184	12
7	2011	33.51379310344828	13
11	2015	25.378787878787882	1
11	2020	25.307407407407414	2

```
.filter($"dense_rank" === 1)
```

Por último, realizamos un filtrado quedándonos únicamente con las filas que poseen en la columna *dense_rank* un valor igual a 1, obteniendo así el año en el que se obtuvo la mayor temperatura máxima en un determinado mes.

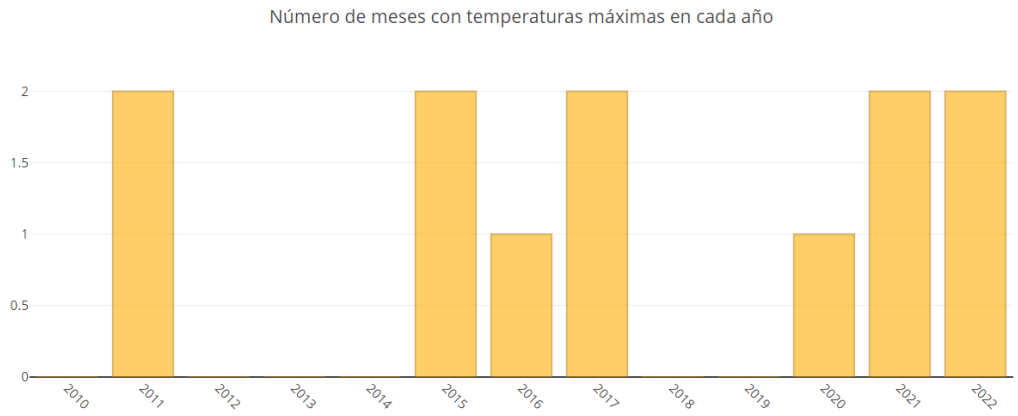
```
def countByYear(year : Int) : Long = {
    dataWindow
        .filter($"año" === year)
        .count()
}

val year2010 = countByYear(2010)
```

Se implementa también la función *countByYear*, la cual nos facilitará los datos que utilizaremos para la representación gráfica, ya que nos devuelve la cantidad de meses por cada año que se encuentran en la consulta realizada anteriormente.

Para finalizar ofrecemos los resultados tanto en forma de tabla como de una forma gráfica que sería para la cual habríamos hecho uso de la función de la *countByYear*.

mes	año	temperatura maxima
1	2021	21.84
2	2020	24.42
3	2017	27.82
4	2011	29.48
5	2015	33.73
6	2017	36.86
7	2022	39.47
8	2022	37.74
9	2016	36.39
10	2011	31.52
11	2015	25.38
12	2021	22.29


Número de meses con temperaturas máximas en cada año
2
1.5
1
0.5
0
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022

4. Despliegue en AWS EMR

2. Experimentos / validación

2.1. Consultas realizadas

2.2. Análisis de requisitos no funcionales

3. Conclusiones

4. Bibliografía

5. Apéndices