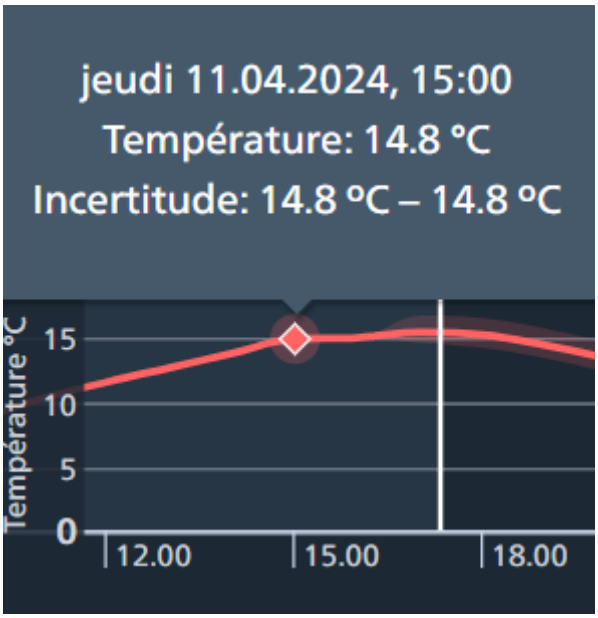
Paul Gillet & Kevin Auberson
Groupe : L05GrL
Date : 11.04.2024

TASK 1: EXPLORE METEOSWISS DATA

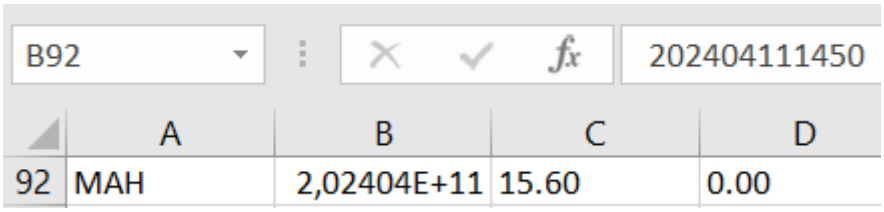
> Does it correspond to what your thermometer or MeteoSwiss' website or mobile app shows?

Taking the last updated data (14h50) from the meteo station of Mathod which is the closest to Yverdon. We have 15.6°C from the CSV file and from the website we get 14.8°C.

Web page:



CSV file:



> In the measurement table examine the Date column (you may have to change its format to see it properly). What does it contain exactly? Precision?

It contains the date and hour of the last update, the format is YYYY-MM-DD HH-MM, in our exemple 2024-04-11 14-50

> For the two data products copy the URLs where the data can be downloaded in the report.

https://data.geo.admin.ch/ch.meteoschweiz.messnetz-automatisch/ch.meteoschweiz.messnetz-automatisch_en.json
https://data.geo.admin.ch/ch.meteoschweiz.messwerte-aktuell/VQHA80.csv

> Document your exploration of the measurement values.

Datas contains differents specific column names (for exemple, tre200s0, dkl010z0). Those represent differents meteorological measures. The traduction of those name is given to us in a text file 'VQHA80_fr.txt', it also contains the different unit of measurement used.

Some data can be missing, those missing data are then replaced by (-)

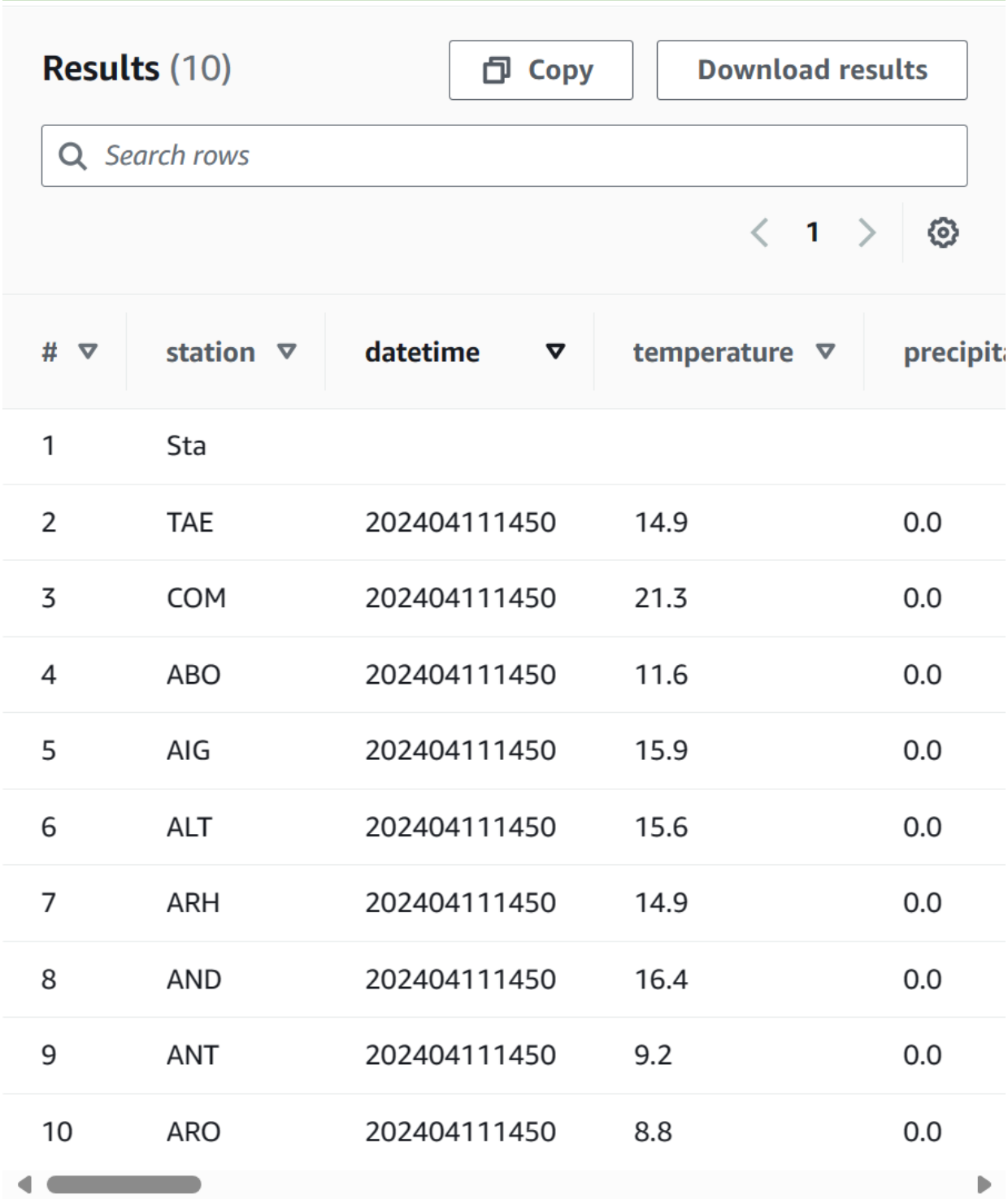
> What is your impression of the the opendata.swiss portal and of MeteoSwiss' data products?

The question is about the portal and products, not about the data you downlaoded (0/1)

Well for some reason the CSV file we downloaded didn't have the same values even tho we got it at the same time. For the Mathod meteo station Kevin's last update was from 15:00 and mine was from 14:50, we even got different temperature mesurements from the CSV and website.

TASK 2: UPLOAD THE CURRENT MEASUREMENT DATA TO S3 AND RUN SQL QUERIES ON IT

6. Verify that you see the first 10 rows of the table.



#	station	datetime	temperature	precipit
1	Sta			
2	TAE	202404111450	14.9	0.0
3	COM	202404111450	21.3	0.0
4	ABO	202404111450	11.6	0.0
5	AIG	202404111450	15.9	0.0
6	ALT	202404111450	15.6	0.0
7	ARH	202404111450	14.9	0.0
8	AND	202404111450	16.4	0.0
9	ANT	202404111450	9.2	0.0
10	ARO	202404111450	8.8	0.0

TASK 3: WRITE A PYTHON SCRIPT TO DOWNLOAD THE CURRENT MEASUREMENT VALUES FROM METEOSWISS AND UPLOAD THEM TO S3

Copy the script into the report.

```
import requests
import logging
import boto3
from botocore.exceptions import ClientError
from io import BytesIO

def download_data(url):
    """
    Download data from a given URL and return it as BytesIO object.

    :param url: URL to download data from
    :return: BytesIO object containing downloaded data
    """
    response = requests.get(url)
    return BytesIO(response.content)

def upload_file(data, bucket, object_name):
    """
    Upload a file to an S3 bucket.

    :param data: Data to upload (BytesIO object)
    :param bucket: Bucket to upload to
    :param object_name: S3 object name
    :return: True if file was uploaded successfully, else False
    """
    s3_client = boto3.client('s3')
    try:
        # Upload the file to the specified bucket with the given object name
        s3_client.upload_fileobj(data, bucket, object_name)
    except ClientError as e:
        # Log any errors that occur during the upload process
        logging.error(e)
        return False  # Return False to indicate upload failure
    return True  # Return True to indicate upload success

def main():
    # Download data from the specified URL
    meteoswiss_url = 'https://data.geo.admin.ch/ch.meteoschweiz.messwerte-
aktuell/VQHA80.csv'
    data = download_data(meteoswiss_url)

    # Define the S3 bucket name and object name
    bucket_name = 'ist-meteo-grl-auberson-gillet'
    object_name = 'current/VQHA80.csv'
    # Upload the downloaded data to S3
    upload_file(data, bucket_name, object_name)
```

```
if __name__ == '__main__':
    main()
```

TASK 4: CONVERT YOUR SCRIPT INTO AN AWS LAMBDA FUNCTION FOR DATA INGESTION

> Copy the data ingestion function and the IAM policy into the lab report.

```
import requests
import boto3
from io import BytesIO
import datetime


def download_data(url):
    """
    Download data from a given URL and return it as BytesIO object.

    :param url: URL to download data from
    :return: BytesIO object containing downloaded data
    """
    response = requests.get(url)
    return BytesIO(response.content)

def upload_file(data, bucket, object_name):
    """
    Upload a file to an S3 bucket.

    :param data: Data to upload (BytesIO object)
    :param bucket: Bucket to upload to
    :param object_name: S3 object name
    :return: True if file was uploaded successfully, else False
    """
    s3_client = boto3.client('s3')
    try:
        # Upload the file to the specified bucket with the given object name
        s3_client.upload_fileobj(data, bucket, object_name)
    except Exception:
        return False  # Return False to indicate upload failure
    return True  # Return True to indicate upload success

def lambda_handler(event, context):
    # Download data from the specified URL
```

```
    meteoswiss_url = 'https://data.geo.admin.ch/ch.meteoschweiz.messwerte-
aktuell/VQHA80.csv'
    data = download_data(meteoswiss_url)

    # Define the S3 bucket name and object name
    bucket_name = 'ist-meteo-grl-auberson-gillet'
    current_time = datetime.datetime.now().replace(microsecond=0)
    formatted_time = current_time.strftime("%Y-%m-%dT%H:%M")
    object_name = 'current/VQHA80-' + formatted_time + '.csv'
    # Upload the downloaded data to S3
    if upload_file(data, bucket_name, object_name):
        return {
            'statusCode': 200,
            'body': 'File uploaded successfully'
        }
    else:
        return {
            'statusCode': 500,
            'body': 'Failed to upload file'
        }
```

> IAM policy

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::ist-meteo-grl-auberson-gillet/current/"
        }
    ]
}
```

TASK 6: TRANSFORM THE WEATHER STATIONS FILE INTO A CSV FILE

> Examine the YAML. There are two top-level keys, what are their names?

`crs` and `features`

One of the keys has an array as value. Which one?

`features`

> What key contains the station name?

The key that contains the stations names is `.properties.station_name`

> Copy the final jq command into the report.

```
jq -j '["id", "station_name", "altitude", "coord_lng", "coord_lat"],
(.features[] | [.id, (.properties.station_name | @json), (.properties.altitude
| tostring), (.geometry.coordinates[0] | tostring), (.geometry.coordinates[1]
| tostring)]) | join(",") + "\n"' ch.meteoschweiz.messnetz-automatisch_en.json
> stations.csv
```

TASK 7: QUERY THE ACCUMULATED DATA

> Make a query that returns all measurements for the Payerne station (PAY), sorted by ascending datetime.

```
SELECT * FROM "meteoswiss_grl"."current" WHERE station = 'PAY' ORDER BY
datetime;
```

#	station	datetime	temperature	precipitation	sunshine	radiation	humidity	despoint
1	PAY	202404251540	11.7	0.0	10.0	498.0	34.8	-3.3
2	PAY	202405021000	10.2	0.0	0.0	249.0	87.1	8.2
3	PAY	202405021010	10.3	0.1	0.0	262.0	87.1	8.3
4	PAY	202405021020	10.5	0.0	0.0	293.0	85.7	8.2
5	PAY	202405021030	10.3	0.0	0.0	396.0	83.9	7.7
6	PAY	202405021040	10.2	0.1	0.0	327.0	85.8	7.9
7	PAY	202405021050	10.3	0.0	0.0	238.0	80.7	7.1
8	PAY	202405021100	10.3	0.0	0.0	212.0	79.8	7.0
9	PAY	202405021110	10.5	0.0	0.0	287.0	79.6	7.1
10	PAY	202405021120	10.7	0.0	0.0	339.0	80.9	7.6

> For Payerne, make a query that returns the maximum temperature for each hour, sorted by increasing hour.

```
SELECT (datetime/100)%100 AS hour, MAX(temperature) AS max_temperature FROM
"meteoswiss_grl"."current" WHERE station = 'PAY' GROUP BY (datetime/100)%100
ORDER BY (datetime/100)%100;
```

Results (7)

Search rows

#	hour	max_temperature
1	10	10.5
2	11	10.7
3	12	12.3
4	13	12.1
5	14	11.9
6	15	11.7
7	16	11.2

> Create a table for the stations folder. Find all stations whose altitude is similar to Yverdon, i.e. 400 m >= altitude < 500 m, sorted by altitude.

```
CREATE EXTERNAL TABLE IF NOT EXISTS `meteoswiss_grl`.`stations` (
  `id` varchar(5),
  `station_name` varchar(100),
  `altitude` int,
  `coord_lng` int,
  `coord_lat` int
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe'
WITH SERDEPROPERTIES ('field.delim' = ',')
STORED AS INPUTFORMAT 'org.apache.hadoop.mapred.TextInputFormat' OUTPUTFORMAT
'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION 's3://ist-meteo-grl-auberson-gillet/stations_csv/'
TBLPROPERTIES ('classification' = 'csv');

SELECT * FROM "meteoswiss_grl"."stations" limit 10;
```

Results (10)

Copy | Download res

Search rows

1

#	id	station_name	altitude	coord_lng	coord_lat
1	id	station_name			
2	ARO	"Arosa"	1878	2771031	1184830
3	RAG	"Bad Ragaz"	497	2756911	1209351
4	HAI	"Salen-Reutenen"	719	2719100	1279047
5	HLL	"Hallau"	419	2677457	1283472
6	DEM	"Delémont"	439	2593270	1244543
7	EBK	"Ebnat-Kappel"	623	2726348	1237176
8	ELM	"Elm"	958	2732266	1198424
9	EIN	"Einsiedeln"	911	2699984	1221068
10	ANT	"Andermatt"	1435	2687445	1165044

```
SELECT * FROM stations WHERE altitude ≥ 400 AND altitude < 500 ORDER BY altitude;
```

Results (57)

Copy | Download resu

Search rows

1

#	id	station_name	altitude	coord_lng	coord_lat
1	BEX	"Bex"	402	2565805	1121511
2	BUE	"Bülach"	403	2682029	1263775
3	VEV	"Vevey / Corseaux"	405	2552106	1146847
4	DOB	"Benken / Doggen"	408	2715388	1227540
5	SCM	"Schmerikon"	408	2713726	1231533
6	OBR	"Oberriet / Kriessern"	409	2764171	1249582
7	JON	"Jona"	410	2706761	1231290
8	GVE	"Genève / Cointrin"	411	2498904	1122632
9	ESZ	"Eschenz"	414	2707844	1278214
10	CEV	"Cevio"	417	2689688	1130564
11	QUI	"Quinten"	419	2734849	1221278

> Find the maximum temperature of all stations at an altitude similar to Yverdon, sorted by altitude.

```
SELECT id, MAX(temperature) as "temperature max" FROM
"meteoswiss_grl"."stations" inner join "meteoswiss_grl"."current" on
"meteoswiss_grl"."current".station = stations.id WHERE altitude ≥ 400 AND
altitude < 500 group by id, altitude ORDER BY altitude;
```

Results (35)

Search rows

#	id	temperature max
1	VEV	12.3
2	SCM	
3	OBR	22.9
4	GVE	13.1
5	CEV	13.0
6	HLL	18.3
7	QUI	
8	WYN	16.8
9	PRE	
10	KLO	19.1

TASK 8: WRITE AN S3 OBJECT LAMBDA FUNCTION TO TRANSFORM DATA

> Copy the code of your function into the report and document your tests.

```
import boto3
import pandas as pd
import requests
from io import StringIO

def lambda_handler(event, context):
    # Extracting required information from the event
    object_get_context = event["getObjectContext"]
    request_route = object_get_context["outputRoute"]
    request_token = object_get_context["outputToken"]
    s3_url = object_get_context["inputS3Url"]

    # Reading CSV data from the provided S3 URL
    response = requests.get(s3_url)
    csv_content = StringIO(response.content.decode('utf-8'))
    data_csv = pd.read_csv(csv_content, sep=";", header=None)

    # Transforming the CSV data
    transformed_csv = transform_csv(data_csv)

    # Writing the transformed data back to S3 Object Lambda
    s3_client = boto3.client('s3')
    s3_client.write_get_object_response(
        Body=transformed_csv,
        RequestRoute=request_route,
        RequestToken=request_token
    )

    return {'status_code': 200}

def transform_csv(csv_data):
    # Renaming columns for clarity
    csv_data.columns = ["station", "date", "temperature", "precipitation",
"sunshine", "radiation", "humidity", "despoint",
                        "wind_dir", "wind_speed", "gust_peak", "pressure",
"press_sea", "press_sea_qnh", "height_850_hpa",
                        "height_700_hpa", "wind_dir_vec", "wind_speed_tower",
"gust_peak_tower", "temp_tool1",
                        "humidity_tower", "dew_point_tower"]

    # Extracting date components
    csv_data["year"] = csv_data["date"].astype(str).str[:4]
    csv_data["month"] = csv_data["date"].astype(str).str[4:6]
    csv_data["day"] = csv_data["date"].astype(str).str[6:8]
    csv_data["hour"] = csv_data["date"].astype(str).str[8:10]
    csv_data["minute"] = csv_data["date"].astype(str).str[10:12]
```

```
    # Dropping unnecessary columns and first row
    csv_data.drop(columns=['date'], inplace=True)
    csv_data.drop(index=0, inplace=True)

    # Converting DataFrame back to CSV string
    transformed_csv = csv_data.to_csv(index=False)
    return transformed_csv
```

To test the lambda function from AWS, we configured the JSON event as follows:

```
{
  "getObjectContext": {
    "outputRoute": "arn:aws:s3-object-lambda:us-east-
1:851725581851:accesspoint/meteoswiss-olap-grl",
    "outputToken": "...",
    "inputS3Url": "https://ist-meteo-grl-auberson-
gillet.s3.amazonaws.com/current/VQHA80.csv"
  }
}
```

The test result is inconclusive - an error has occurred.

```
{
  "errorMessage": "2024-05-05T12:57:49.248Z 3fe31ef3-aa20-4fd5-bd90-
cff89a82ba67 Task timed out after 3.09 seconds"
}
```

We tried a number of alternatives and code modifications to modify the CSV file, but none of them worked.

Ok! J'ai eu un timeout similaire avec Python 3.12, je ne sais pas pourquoi vous avez également un timeout. (je vous mets tous les points)

TASK 9: SCENARIO

> As an engineer working with data products, you might face unexpected issues with your application due to sudden modification of the source data. How could MeteoSwiss change the data product, and which modifications would you make to ensure that your code still functions correctly? How would you improve your code robustness for such changes? What can be done to detect schema changes in the source data?

Since data structure wouldn't be changed everyday. We could store a hash of the expected data structure from precedent files. Each time before processing the new incoming file we can

compare the structures hash to see if any changes were made. This would trigger an alert and so we can make change in our scripts