Learning Weather
Outil de prévision de la météo

Filière : Data Engineer

2021/2022

Réaliser par :

TAZZI Karim
JEHBALI Youssef

Encadrer par :

Mme EL ASRI IKRAME

I. Problématique

- Personne ne peut nier que le Maroc a connu des changements climatique importants durant ses dernières années. En effet l'année 2021 a connue des précipitations torrentielles due à des problèmes et même à des accidents grave surtout au niveau de la région de Casablanca, où les voitures était noyées et les rues sont plein d'eau comme montre **la figure 1**.

Figure 1 - Pluies diluviennes à Casablanca

- Et même au niveau de la région de Tanger qui a connu un accident très grave, où 25 personnes ont perdu la vie dans une usine clandestine à cause d'un court-circuit provoquer par les pluies torrentielles.

- Tout cela montre l'importance de la prévision de ses changements climatiques, et en particulier la prévision de la météo afin d'éviter plusieurs problèmes. Et aider les autorités à prendre des décisions pour être prêt le maximum de possible à met des actions contre tout situation climatique.

II. Sujet de Projet

- **Learning weather** est un projet consistent à développer un outil pour la prévision de la météo avec un certaine précision en utilisont machine learning et Deep learning (LSTM neural network) et en se basant seulement sur l'historique des données météorologiques de plus de 10 ans.

- Afin de réaliser ce projet nous avons cherché en première temp des sources météorologiques permettant d'extraire les données sur plus de 10 ans, plusieurs sites web ont été sélectionnés « estesparkweather », « infoclimat » …etc.

  Par la suite on va suivre l'enchainement suivante :

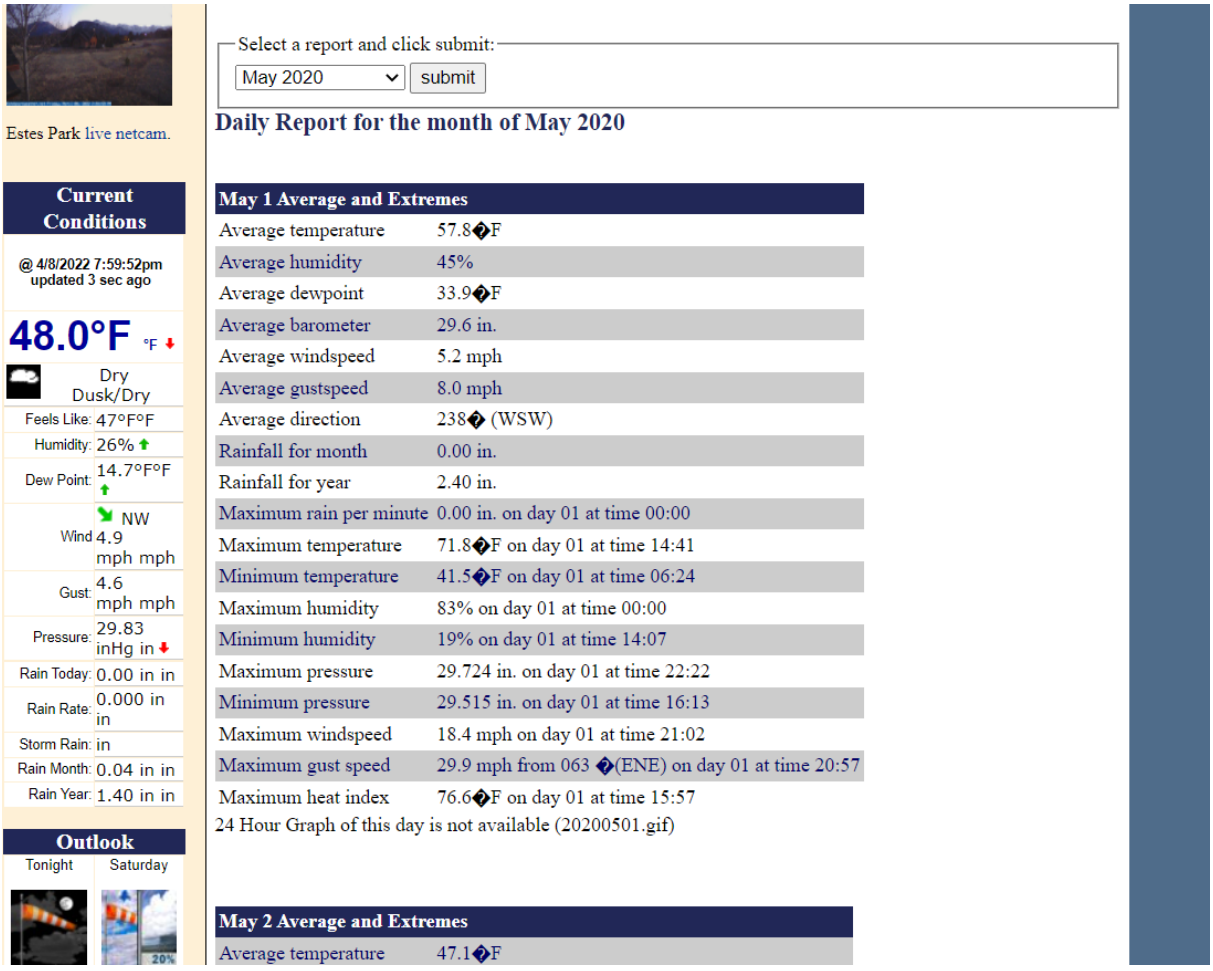
  - ✓ L'extraction des données.
  - ✓ La compréhension des données.
  - ✓ La Préparation des données.
  - ✓ L'analyse exploratoire des données.
  - ✓ La construction du modèle.
  - ✓ L'évaluation de modèle.
  - ✓ Discussion des résultats.
  - ✓ Observations.

- Concernant la région qu'on va traiter, nous avons décidé d'étudier la météo d'une petite ville située au United state nommer estes park Colorado. Du fait que nous avons trouvé ses données météorologiques de plus de 10 ans, des données qui sont bien structurés sous forme de texte.

- Le site web qu'on va utiliser contient pour une seule page les données quotidiennes de la météo de la mois toute entière. Ce qui facilite le grattage des données ainsi il contient des informations pertinentes sur le climat de la ville estes park.

III. Extraction des données – web scraping.

- Notre source de données sera le site web **estesparkweather.net** du fait qu'il contient des données bien structurés sous forme de texte comme on voie dans la figure 2.



Estes Park live netcam.

Current Conditions

@ 4/8/2022 7:59:52pm updated 3 sec ago

48.0°F °F

Dry
Dusk/Dry

Feels Like:	47°F°F
Humidity:	26%
Dew Point:	14.7°F°F
Wind	NW 4.9 mph mph
Gust:	4.6 mph mph
Pressure:	29.83 inHg in
Rain Today:	0.00 in in
Rain Rate:	0.000 in in
Storm Rain:	in
Rain Month:	0.04 in in
Rain Year:	1.40 in in

Outlook

Tonight | Saturday

Select a report and click submit:

May 2020 submit

Daily Report for the month of May 2020

May 1 Average and Extremes	
Average temperature	57.8°F
Average humidity	45%
Average dewpoint	33.9°F
Average barometer	29.6 in.
Average windspeed	5.2 mph
Average gustspeed	8.0 mph
Average direction	238° (WSW)
Rainfall for month	0.00 in.
Rainfall for year	2.40 in.
Maximum rain per minute	0.00 in. on day 01 at time 00:00
Maximum temperature	71.8°F on day 01 at time 14:41
Minimum temperature	41.5°F on day 01 at time 06:24
Maximum humidity	83% on day 01 at time 00:00
Minimum humidity	19% on day 01 at time 14:07
Maximum pressure	29.724 in. on day 01 at time 22:22
Minimum pressure	29.515 in. on day 01 at time 16:13
Maximum windspeed	18.4 mph on day 01 at time 21:02
Maximum gust speed	29.9 mph from 063 °(ENE) on day 01 at time 20:57
Maximum heat index	76.6°F on day 01 at time 15:57

24 Hour Graph of this day is not available (20200501.gif)

May 2 Average and Extremes	
Average temperature	47.1°F

Figure 2- weather data on estesparkweather.net of 1 May 2020.

- Le site web infoclimat contient aussi des données météorologiques de plus de 10 ans mais le problème c'est qu'il contient des données semi-structurés du fait qu'on trouve là-dessus des colonnes représenter par des images contenant de l'information sur la météo.

  D'où le choix de notre source de données **estesparkweather.net .**

- Pour extraire les données on va utiliser BeautifulSoup de python Et Requests librarys pour faire des requêtes http sous python.
- L'extraction des données se fait comme suit (la figure 3) :

```
Entrée [3]: import bs4
            from bs4 import BeautifulSoup
            import requests
            import pandas as pd
            from datetime import datetime
            import re
```

```
Entrée [5]: url = "https://www.estesparkweather.net/archive_reports.php?date=202201"
            page = requests.get(url)
            soup = BeautifulSoup(page.content, 'html.parser')
            table = soup.find_all('table')
            data_rows= [row.text.splitlines() for row in table]
```

Figure 3 : extraction des données avec requests & beautifulSoup

- Après l'inspection de la page on a constaté que les données de la météo, de la journée sont stockées dans des balises nommer table.

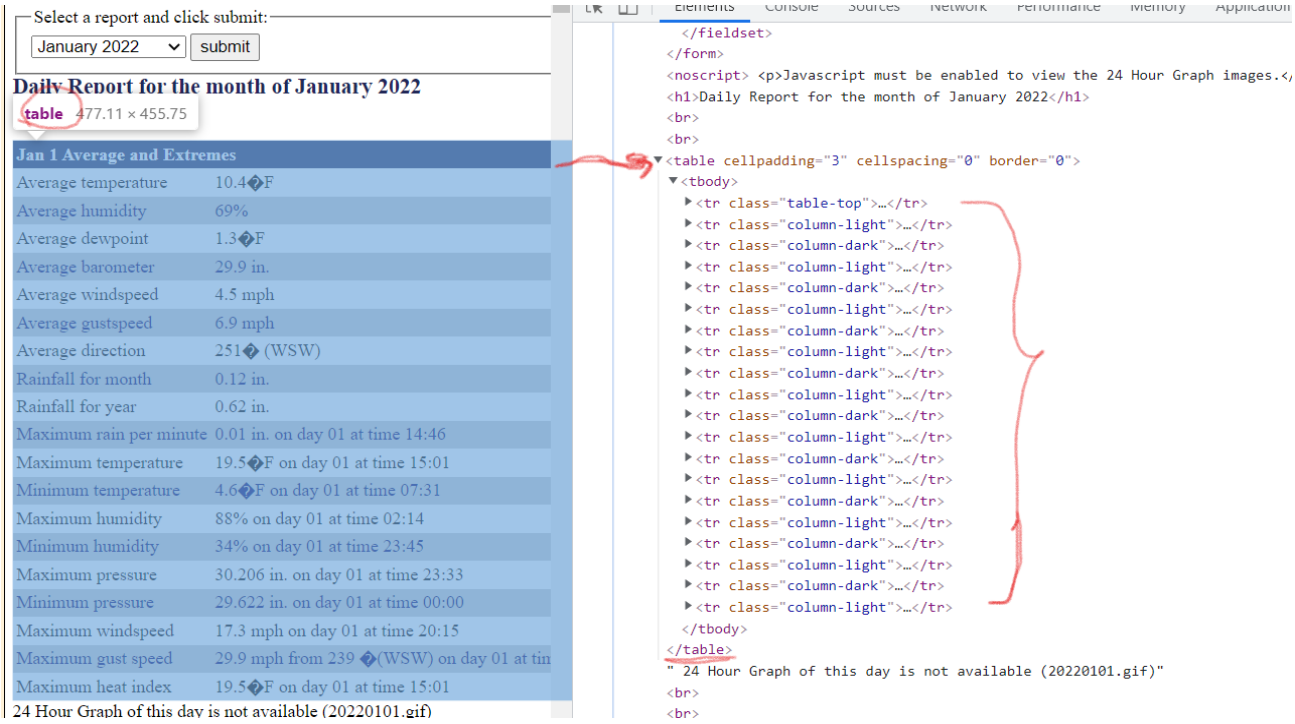


Figure 4 : les données d'une journée sous HTML

- Les « tr » représentes les lignes. Et les tables représente les journées. c'est-à-dire au niveau de chaque balise « table » sous html, il y a les données de la météo d'une journée de la mois. Comme montre la figure 5.

```
<br>
▶ <table cellpadding="3" cellspacing="0" border="0">…</table>
" 24 Hour Graph of this day is not available (20220101.gif)"
<br>
<br>
<br>
<br>
▶ <table cellpadding="3" cellspacing="0" border="0">…</table>
" 24 Hour Graph of this day is not available (20220102.gif)"
<br>
<br>
<br>
<br>
▶ <table cellpadding="3" cellspacing="0" border="0">…</table>
" 24 Hour Graph of this day is not available (20220103.gif)"
<br>
<br>
<br>
<br>
▶ <table cellpadding="3" cellspacing="0" border="0">…</table>
" 24 Hour Graph of this day is not available (20220104.gif)"
<br>
<br>
<br>
<br>
▶ <table cellpadding="3" cellspacing="0" border="0">…</table>
" 24 Hour Graph of this day is not available (20220105.gif)"
<br>
<br>
```

Figure 5 : les balises tables contenant les données chaque journée

- Nous voulons extraire les données de plus de 10ans et pas d'une seul mois, et on a déjà dit que pour une url, il y a une seule page , contenant les données de la météo d'une mois uniquement. Alors pour extraire les données des autres mois nous avons besoin d'autre urls.
  A cet égard nous avons constaté que la différence entre l url de moi de janvier de l'année 2022 qui est :
  https://www.estesparkweather.net/archive_reports.php?date=**202201**
  et les autres, seulement la date mentionné à la fin.
  +Construction d'une liste dates contenant les dates qu'on veut leurs data

```
Dates_r = pd.date_range(start = "01/01/2010",end = "01/01/2022",freq = "M")
dates = [str(i)[:4] + str(i)[5:7] for i in Dates_r]
print(dates)
```

```
['201001', '201002', '201003', '201004', '201005', '201006', '201007', '201008', '201009', '201010', '201011', '201012', '20110
1', '201102', '201103', '201104', '201105', '201106', '201107', '201108', '201109', '201110', '201111', '201112', '201201', '20
1202', '201203', '201204', '201205', '201206', '201207', '201208', '201209', '201210', '201211', '201212', '201301', '201302',
'201303', '201304', '201305', '201306', '201307', '201308', '201309', '201310', '201311', '201312', '201401', '201402', '20140
3', '201404', '201405', '201406', '201407', '201408', '201409', '201410', '201411', '201412', '201501', '201502', '201503', '20
1504', '201505', '201506', '201507', '201508', '201509', '201510', '201511', '201512', '201601', '201602', '201603', '201604',
'201605', '201606', '201607', '201608', '201609', '201610', '201611', '201612', '201701', '201702', '201703', '201704', '20170
5', '201706', '201707', '201708', '201709', '201710', '201711', '201712', '201801', '201802', '201803', '201804', '201805', '20
1806', '201807', '201808', '201809', '201810', '201811', '201812', '201901', '201902', '201903', '201904', '201905', '201906',
'201907', '201908', '201909', '201910', '201911', '201912', '202001', '202002', '202003', '202004', '202005', '202006', '20200
7', '202008', '202009', '202010', '202011', '202012', '202101', '202102', '202103', '202104', '202105', '202106', '202107', '20
2108', '202109', '202110', '202111', '202112']
```

Figure 6 : Liste des dates qu'on va utiliser pour former les urls

- Voici le code final afin d'extraire les données de la météo de estes-park entre 01/01/2010 et 01/01/2022.

```
Entrée [1]: import bs4
            from bs4 import BeautifulSoup
            import requests
            import pandas as pd
            from datetime import datetime
            import re

Entrée [9]: dates_r= pd.date_range(start="01/01/2010",end="01/01/2022",freq="M")
            dates = [str(i)[:4] + str(i)[5:7] for i in dates_r]
            lst = []
            index = []
            for j in range(len(dates)):
                url="https://www.estesparkweather.net/archive_reports.php?date="+ dates[j]
                page=requests.get(url)
                soup = BeautifulSoup(page.content, 'html.parser')
                table = soup.find_all('table')
                data_rows= [row.text.splitlines() for row in table]
                data_rows= data_rows[:-9]
                for k in range(len(data_rows)):
                    data_rows[k] = data_rows[k][2:len(data_rows[k]):3]
                for l in range(len(data_rows)):
                    c= [('.'.join(re.findall("\d+",str(data_rows[l][k].split()[:5])))) for k in range(len(data_rows[l]))]
                    lst.append(c)
                    index.append(dates[j] + c[0])
            d1_index = [index[i] for i in range(len(index)) if len(index[i]) > 6]
            data = [lst[i][1:] for i in range(len(lst)) if len(lst[i][1:]) == 19]
```

Figure 7 : processus d'extraction des données avec python

- Explication de chaque ligne de code de la figure 7:

  - dates_r contient les dates entre janvier 2010 et janvier 2022 mais ils sont de la forme « Ann-mois-jour » (2010-01-01 par exemple).
    On doit les rendre sous forme 201001.(annéemois) pour les utilisées.
    Par exemple pour i= '2010-01-01' on a str(i)[:4]+str(i)[5 :7]=201001
    Et c'est le résultat qu'on veut.
  - La première boucle c'est pour boucler sur les urls de tous les pages
    Après on stock les ligne de données des tables dans une liste date_rows
    Par la suite on élimine les 9 dernière lignes qui sont inutile pour nous (ne contient aucun information ,située à la fin de la page)
  - La deuxième boucle c'est pour éliminer les virgules
    Avant la boucle 2.

```
url = "https://www.estesparkweather.net/archive_reports.php?date=202201"
page = requests.get(url)
soup = BeautifulSoup(page.content, 'html.parser')
table = soup.find_all('table')
data_rows= [row.text.splitlines() for row in table]
data_rows= data_rows[:-9]
print(data_rows)
```

```
[['', '', 'Jan 1 Average and Extremes', '', '', 'Average temperature 10.4°F', '', '', 'Averag
e dewpoint 1.3°F', '', '', 'Average barometer 29.9 in.', '', '', 'Average windspeed 4.5 mph',
mph', '', '', 'Average direction 251° (WSW)', '', '', 'Rainfall for month 0.12 in.', '', '',
'', '', 'Maximum rain per minute 0.01 in. on day 01 at time 14:46', '', '', 'Maximum temperat
01', '', '', 'Minimum temperature 4.6°F on day 01 at time 07:31', '', '', 'Maximum humidity 8
'', 'Minimum humidity 34% on day 01 at time 23:45', '', '', 'Maximum pressure 30.206 in. on d
inimum pressure 29.622 in. on day 01 at time 00:00', '', '', 'Maximum windspeed 17.3 mph on d
aximum gust speed 29.9 mph  from 239 °(WSW) on day 01 at time 20:24', '', '', 'Maximum heat i
5:01', ''], ['', '', 'Jan 2 Average and Extremes', '', '', 'Average temperature 26.9°F', '',
'', 'Average dewpoint 4.2°F', '', '', 'Average barometer 30.2 in.', '', '', 'Average windspee
stspeed 15.4 mph', '', '', 'Average direction 238° (WSW)', '', '', 'Rainfall for month 0.14 i
0.64 in.', '', '', 'Maximum rain per minute 0.01 in. on day 02 at time 15:27', '', '', 'Maxim
at time 14:12', '', '', 'Minimum temperature 14.2°F on day 02 at time 00:18', '', '', 'Maximu
```

Après la boucle 2.

```
for i in range(len(data_rows)):
    data_rows[i]=data_rows[i][2:len(data_rows[i]):3]
print(data_rows)
```

```
[['Jan 1 Average and Extremes', 'Average temperature 10.4°F', 'Average humidity 69%', 'Average dewpoint 1.3°F', 'Average baro
meter 29.9 in.', 'Average windspeed 4.5 mph', 'Average gustspeed 6.9 mph', 'Average direction 251° (WSW)', 'Rainfall for mont
h 0.12 in.', 'Rainfall for year 0.62 in.', 'Maximum rain per minute 0.01 in. on day 01 at time 14:46', 'Maximum temperature 1
9.5°F on day 01 at time 15:01', 'Minimum temperature 4.6°F on day 01 at time 07:31', 'Maximum humidity 88% on day 01 at time
02:14', 'Minimum humidity 34% on day 01 at time 23:45', 'Maximum pressure 30.206 in. on day 01 at time 23:33', 'Minimum press
ure 29.622 in. on day 01 at time 00:00', 'Maximum windspeed 17.3 mph on day 01 at time 20:15', 'Maximum gust speed 29.9 mph
from 239 °(WSW) on day 01 at time 20:24', 'Maximum heat index 19.5°F on day 01 at time 15:01'], ['Jan 2 Average and Extreme
s', 'Average temperature 26.9°F', 'Average humidity 38%', 'Average dewpoint 4.2°F', 'Average barometer 30.2 in.', 'Average wi
ndspeed 10.5 mph', 'Average gustspeed 15.4 mph', 'Average direction 238° (WSW)', 'Rainfall for month 0.14 in.', 'Rainfall for
year 0.64 in.', 'Maximum rain per minute 0.01 in. on day 02 at time 15:27', 'Maximum temperature 36.7°F on day 02 at time 14:
12', 'Minimum temperature 14.2°F on day 02 at time 00:18', 'Maximum humidity 49% on day 02 at time 06:42', 'Minimum humidity
28% on day 02 at time 21:12', 'Maximum pressure 30.361 in. on day 02 at time 10:01', 'Minimum pressure 30.136 in. on day 02 a
t time 03:18', 'Maximum windspeed 19.6 mph on day 02 at time 22:04', 'Maximum gust speed 32.2 mph  from 232 °( SW) on day 02
at time 03:23', 'Maximum heat index 36.7°F on day 02 at time 14:12'], ['Jan 3 Average and Extremes', 'Average temperature 33.
2°F', 'Average humidity 33%', 'Average dewpoint 6.9°F', 'Average barometer 30.0 in.', 'Average windspeed 10.7 mph', 'Average
gustspeed 15.5 mph', 'Average direction 230° ( SW)', 'Rainfall for month 0.14 in.', 'Rainfall for year 0.64 in.', 'Maximum ra
in per minute 0.00 in. on day 03 at time 23:59', 'Maximum temperature 40.1°F on day 03 at time 14:03', 'Minimum temperature 2
```

- Finalement pour la boucle 3 c'est pour rassembler les nombres.

```
for i in range(len(data_rows)):
        c = [".".join(re.findall("\d+",str(data_rows[i][j].split()[:5]))) for j in range(len(data_rows[i]))]
        print(c)
```

```
['1', '10.4', '69', '1.3', '29.9', '4.5', '6.9', '251', '0.12', '0.62', '0.01', '19.5', '4.6', '88', '34', '30.206', '29.622',
'17.3', '29.9', '19.5']
['2', '26.9', '38', '4.2', '30.2', '10.5', '15.4', '238', '0.14', '0.64', '0.01', '36.7', '14.2', '49', '28', '30.361', '30.13
6', '19.6', '32.2', '36.7']
['3', '33.2', '33', '6.9', '30.0', '10.7', '15.5', '230', '0.14', '0.64', '0.00', '40.1', '24.7', '45', '25', '30.155', '29.74
```

- Après on construit une colonne pour les dates :

Entrée [20]:
```
final_index = [datetime.strptime(str(d1_index[i]), "%Y%m%d").strftime("%Y-%m-%d") for i in range(len(d1_index))]
final_index
```

Out[20]:
```
['2010-01-01',
 '2010-01-02',
 '2010-01-03',
 '2010-01-04',
 '2010-01-05',
 '2010-01-06',
 '2010-01-07'
```

On transforme notre liste de donnée en dataframe python et on insert la colonne des dates.

Entrée [21]:
```
weather_data=pd.DataFrame(data)
```

Entrée [22]:
```
weather_data.insert(0,"date",final_index,allow_duplicates=False)
```

Entrée [23]:
```
weather_data
```

Out[23]:

	date	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0	2010-01-01	32.1	49	15.2	30.2	14.6	19.7	297	0.00	0.00	0.00	40.0	22.1	61	41	30.330	30.034	34.5	51	40.0
1	2010-01-02	32.1	50	15.5	30.1	8.8	12.2	306	0.00	0.00	0.00	39.1	22.0	77	39	30.292	29.937	23.0	35	39.1
2	2010-01-03	23.1	64	12.1	30.4	2.9	4.4	21	0.00	0.00	0.00	33.4	9.2	85	34	30.549	30.237	17.3	24	33.4
3	2010-01-04	25.7	48	7.2	30.4	4.7	7.0	324	0.00	0.00	0.00	36.5	7.6	82	25	30.585	30.107	15.0	21	36.5
4	2010-01-05	34.3	51	17.8	30.0	14.2	18.5	300	0.00	0.00	0.00	40.1	28.4	60	43	30.173	29.897	34.5	43	40.1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4050	2021-12-27	22.7	76	15.9	29.6	2.8	4.5	242	0.50	1.04	0.00	29.2	17.0	90	53	29.751	29.325	18.4	31.1	29.2
4051	2021-12-28	21.0	54	6.5	29.4	9.8	14.4	208	0.50	1.04	0.00	25.2	14.5	79	42	29.569	29.315	26.5	41.4	25.2
4052	2021-12-29	23.5	57	10.4	29.7	8.3	12.1	234	0.50	1.04	0.00	24.4	14.5	75	50	29.758	29.568	17.3	31.1	24.4
4053	2021-12-30	26.3	59	13.4	29.5	16.0	23.1	241	0.50	1.04	0.00	30.6	21.4	87	42	29.682	29.427	34.5	52.9	30.6
4054	2021-12-31	20.7	87	17.6	29.5	2.6	4.1	352	0.50	1.04	0.00	26.3	12.0	92	77	29.645	29.367	11.5	17.3	26.3

4055 rows × 20 columns

- On renomme les colonnes de notre data set.

```
Entrée [24]: weather_data.columns=["date","Average temperature","Average humidity","Average dewpoint","Average barometer","Average windspeed"
```

```
Entrée [25]: weather_data
```

Out[25]:

	date	Average temperature	Average humidity	Average dewpoint	Average barometer	Average windspeed	Average gustspeed	Average direction	Rainfall for month	Rainfall for year	Maximum rain per minute	Maximum temperature	Minimum temperature	Maximum humidity	M
0	2010-01-01	32.1	49	15.2	30.2	14.6	19.7	297	0.00	0.00	0.00	40.0	22.1	61	
1	2010-01-02	32.1	50	15.5	30.1	8.8	12.2	306	0.00	0.00	0.00	39.1	22.0	77	
2	2010-01-03	23.1	64	12.1	30.4	2.9	4.4	21	0.00	0.00	0.00	33.4	9.2	85	
3	2010-01-04	25.7	48	7.2	30.4	4.7	7.0	324	0.00	0.00	0.00	36.5	7.6	82	
4	2010-01-05	34.3	51	17.8	30.0	14.2	18.5	300	0.00	0.00	0.00	40.1	28.4	60	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
4050	2021-12-27	22.7	76	15.9	29.6	2.8	4.5	242	0.50	1.04	0.00	29.2	17.0	90	
4051	2021-12-28	21.0	54	6.5	29.4	9.8	14.4	208	0.50	1.04	0.00	25.2	14.5	79	
4052	2021-12-29	23.5	57	10.4	29.7	8.3	12.1	234	0.50	1.04	0.00	24.4	14.5	75	
4053	2021-12-30	26.3	59	13.4	29.5	16.0	23.1	241	0.50	1.04	0.00	30.6	21.4	87	
4054	2021-12-31	20.7	87	17.6	29.5	2.6	4.1	352	0.50	1.04	0.00	26.3	12.0	92	

4055 rows x 20 columns

- Conservons notre dataframe dans un fichier weather_data.csv

```
ntrée [26]: weather_data.to_csv("Weather_Data.csv",index=False)
```

IV. Compréhension des données

Notre dataset « weather_data » représente les données de climat da la ville Estes-Park Colorado au US entre les années 2010 et 2022.
Constitué de 19 colonnes présenter comme suit :

date : de la forme « année-mois-jour » ,type : object (str)
Average temperature : la température moyenne de la journée en °F.
Average humidity : l' humidité moyenne de la journée en %.
Average dewpoint : Le **point de rosée** est la température à laquelle l'air doit être refroidi pour que la vapeur d'eau qu'il contient condense en rosée ou en givre.
Average barometre : La pression atmosphérique moyenne de la journée en inHg
Average Windspeed : La vitesse moyenne journalière du vent en mph.
Average gustspeed: la vitesse moyenne journalière de rafale en mph.

Average direction : moyenne de la direction du vent en wsw.
Rainfall for month : Précipitations du mois (unité : in | 1 in = 25.4milimiter).
Rainfall for year : Précipitations de l'année (unité : in).
Maximum rain per minute : Pluie maximale par minute (unité : in).
Maximum temperature : température maximale de la journée en °F.
Minimum temperature : température minimale de la journée en °F.
Maximum humidity : Humidité maximale en %.
Minimum humidity : Humidité minimale en %.
Maximum pressure : pression maximale de la journée.
Minimum pressure : pression minimale de la journée.
Maximum windspeed : Vitesse maximale du vent en mph.
Maximum gustspeed : Vitesse de rafale maximale en mph.
Maximum heat index : Indice de chaleur maximal en °F.

Notre but : la prédiction de la température moyenne quotidienne.

V. Préparation des données

- On va commencer par importer notre dataset et jeter un coup d'œil en appliquons les méthodes suivantes :

    - Data_weather.**head()**
    - Data_weather.**shape**
    - Data_weather.**dtypes**
    - Data_weather.**info()**
    - Data_weather.**isna().sum()**
    - Data_weather.**describe()**

- ## Démonstration et observations

    - On commence par importer the libraires nécessaire pour cette partie.

```
Entrée [365]: # Import libraries
              import pandas as pd
              import matplotlib.pyplot as plt
              import numpy as np
              import seaborn as sns
              import missingno as msno #Missing data visualization module for Python
```

- On observe qu'il y a des caractères au niveau de la dernière colonne ')' on va les éliminer après, afin de l'utiliser comme variable numérique.

```
Entrée [366]: data_weather=pd.read_csv(r"C:\Users\Hinnovis\Downloads\weather_data.csv")

Entrée [367]: data_weather.shape
Out[367]: (4055, 20)

Entrée [368]: # the header of the DataFrame
              data_weather.head()
Out[368]:
```

ıverage dspeed	Average gustspeed	Average direction	Rainfall for month	Rainfall for year	Maximum rain per minute	Maximum temperature	Minimum temperature	Maximum humidity	Minimum humidity	Maximum pressure	Minimum pressure	Maximum windspeed	Maximum gust speed	Maximum heat index
14.6	19.7	297.0	0.0	0.0	0.0	40.0	22.1	61.0	41.0	30.330	30.034	34.5	51.0	40.0
8.8	12.2	306.0	0.0	0.0	0.0	39.1	22.0	77.0	39.0	30.292	29.937	23.0	35.0	39.1
2.9	4.4	21.0	0.0	0.0	0.0	33.4	9.2	85.0	34.0	30.549	30.237	17.3	24.0	33.4
4.7	7.0	324.0	0.0	0.0	0.0	36.5	7.6	82.0	25.0	30.585	30.107	15.0	21.0	36.5)
14.2	18.5	300.0	0.0	0.0	0.0	40.1	28.4	60.0	43.0	30.173	29.897	34.5	43.0	40.1

- La dernière colonne a pour type object par contre ,c'est un variable qui doit représentent des valeurs numériques.

```
Entrée [196]: data_weather.dtypes
Out[196]: date                      object
          Average temperature      float64
          Average humidity           int64
          Average dewpoint         float64
          Average barometer        float64
          Average windspeed        float64
          Average gustspeed        float64
          Average direction        float64
          Rainfall for month       float64
          Rainfall for year        float64
          Maximum rain per minute  float64
          Maximum temperature      float64
          Minimum temperature      float64
          Maximum humidity         float64
          Minimum humidity         float64
          Maximum pressure         float64
          Minimum pressure         float64
          Maximum windspeed        float64
          Maximum gust speed       float64
          Maximum heat index        object
          dtype: object
```

- Informations sur nos données

```
Entrée [197]: # info of DataFrame
              data_weather.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4055 entries, 0 to 4054
Data columns (total 20 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   date                     4055 non-null   object
 1   Average temperature      4055 non-null   float64
 2   Average humidity         4055 non-null   int64
 3   Average dewpoint         4055 non-null   float64
 4   Average barometer        4055 non-null   float64
 5   Average windspeed        4054 non-null   float64
 6   Average gustspeed        4054 non-null   float64
 7   Average direction        4054 non-null   float64
 8   Rainfall for month       4054 non-null   float64
 9   Rainfall for year        4054 non-null   float64
 10  Maximum rain per minute  4054 non-null   float64
 11  Maximum temperature      4054 non-null   float64
 12  Minimum temperature      4054 non-null   float64
 13  Maximum humidity         4054 non-null   float64
 14  Minimum humidity         4054 non-null   float64
 15  Maximum pressure         4054 non-null   float64
 16  Minimum pressure         4054 non-null   float64
 17  Maximum windspeed        4054 non-null   float64
 18  Maximum gust speed       4053 non-null   float64
 19  Maximum heat index       4053 non-null   object
dtypes: float64(17), int64(1), object(2)
memory usage: 633.7+ KB
```

- Valeurs manquantes

```
Entrée [198]: data_weather.isna().sum()
```

```
Out[198]: date                     0
          Average temperature      0
          Average humidity         0
          Average dewpoint         0
          Average barometer        0
          Average windspeed        1
          Average gustspeed        1
          Average direction        1
          Rainfall for month       1
          Rainfall for year        1
          Maximum rain per minute  1
          Maximum temperature      1
          Minimum temperature      1
          Maximum humidity         1
          Minimum humidity         1
          Maximum pressure         1
          Minimum pressure         1
          Maximum windspeed        1
          Maximum gust speed       2
          Maximum heat index       2
          dtype: int64
```

- Description des données

```
Entrée [199]: # description of DataFrame
              data_weather.describe()
```

Out[199]:

	Average temperature	Average humidity	Average dewpoint	Average barometer	Average windspeed	Average gustspeed	Average direction	Rainfall for month	Rainfall for year	Maximum rain per minute	Maximum temperature	Mir tempe
count	4055.000000	4055.000000	4055.000000	4055.000000	4054.000000	4054.000000	4054.000000	4054.000000	4054.000000	4054.000000	4054.000000	4054.0
mean	44.690037	49.092232	23.704932	29.863428	5.583473	9.117341	215.879571	0.452881	5.597563	0.020069	57.687617	31.5
std	15.413125	17.534725	13.747731	0.520584	3.932204	11.212808	94.563260	0.613979	4.545367	0.992614	18.059098	13.1
min	0.400000	9.000000	0.000000	4.900000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0
25%	33.350000	36.000000	12.300000	29.700000	2.600000	4.300000	119.000000	0.050000	1.140000	0.000000	43.600000	22.7
50%	45.200000	48.000000	22.500000	29.800000	4.300000	6.800000	248.000000	0.220000	5.350000	0.000000	57.600000	32.6
75%	58.200000	61.000000	35.400000	30.000000	7.700000	11.600000	282.000000	0.660000	9.050000	0.010000	73.700000	41.8
max	76.300000	94.000000	55.100000	31.000000	22.300000	240.400000	360.000000	6.260000	16.410000	63.200000	92.800000	65.7

- Eliminons les caractères ‘)’ trouvé au niveau de la dernière colonne.
  Et remarquons qu’ils ont bien disparu.

```
Entrée [202]: data_weather["Maximum heat index"]=data_weather["Maximum heat index"].str.replace(")","")
              for i in data_weather["Maximum heat index"]:
                  print(i)
```

```
44.1
37.3
33.4
43.6
43.1
36.5
41.6
41.5
32.0
32.0
34.6
34.6
34.9
35.5
36.6
32.0
32.4
39.5
41.8
```

- On va convertir ce dernier colonne au float et remarquons qu'il est bien converti.

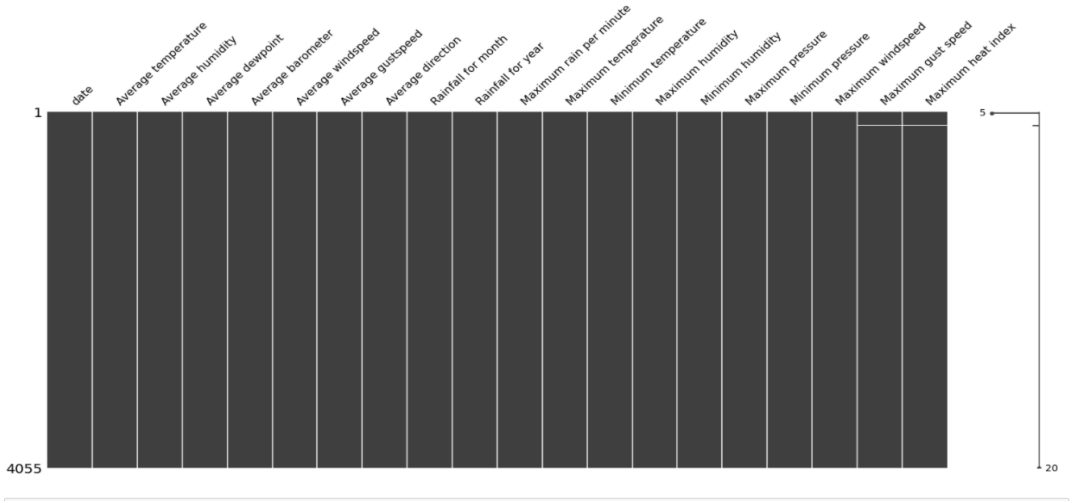
Entrée [204]:
```
#convert the maximum heat index into float
data_weather['Maximum heat index'] = data_weather['Maximum heat index'].astype('float')
```

Entrée [205]:
```
data_weather.dtypes
```

Out[205]:
```
date                        object
Average temperature        float64
Average humidity             int64
Average dewpoint           float64
Average barometer          float64
Average windspeed          float64
Average gustspeed          float64
Average direction          float64
Rainfall for month         float64
Rainfall for year          float64
Maximum rain per minute    float64
Maximum temperature        float64
Minimum temperature        float64
Maximum humidity           float64
Minimum humidity           float64
Maximum pressure           float64
Minimum pressure           float64
Maximum windspeed          float64
Maximum gust speed         float64
Maximum heat index         float64
dtype: object
```

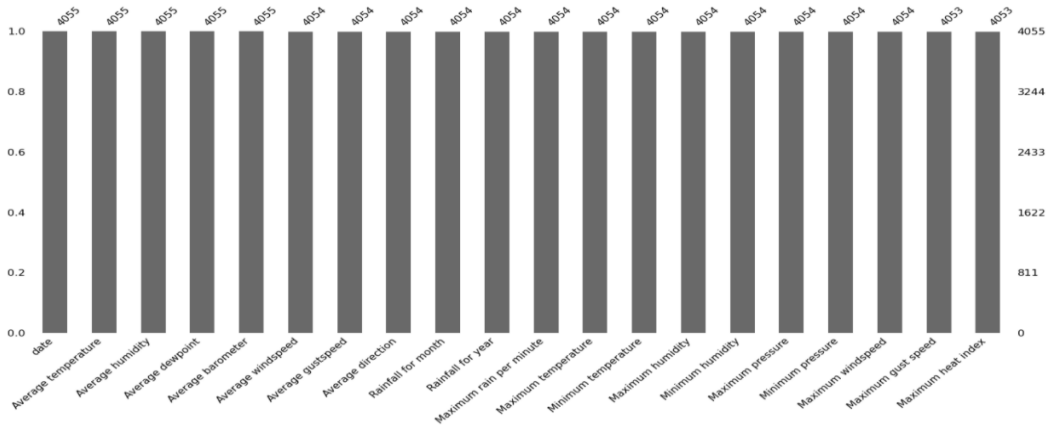
- Visualisation des valeurs manquantes

Entrée [206]:
```
#dealing with missing data
msno.matrix(data_weather)
plt.show()
```



Entrée [208]:
```
msno.bar(data_weather)
```

Out[208]:
```
<AxesSubplot:>
```



- Remplissons les valeurs manquantes de notre dataset. En effet les valeurs manquantes d'une colonne seront rempli par leur moyenne

```
Entrée [209]: data_weather['Average windspeed'].replace(np.nan,data_weather['Average windspeed'].mean(),inplace=True)
              data_weather['Average gustspeed'].replace(np.nan,data_weather['Average gustspeed'].mean(),inplace=True)
              data_weather['Average direction'].replace(np.nan,data_weather['Average direction'].mean(),inplace=True)
              data_weather['Rainfall for month'].replace(np.nan,data_weather['Rainfall for month'].mean(),inplace=True)
              data_weather['Rainfall for year'].replace(np.nan,data_weather['Rainfall for year'].mean(),inplace=True)
              data_weather['Maximum rain per minute'].replace(np.nan,data_weather['Maximum rain per minute'].mean(),inplace=True)
              data_weather['Maximum temperature'].replace(np.nan,data_weather['Maximum temperature'].mean(),inplace=True)
              data_weather['Minimum temperature'].replace(np.nan,data_weather['Minimum temperature'].mean(),inplace=True)
              data_weather['Maximum humidity'].replace(np.nan,data_weather['Maximum humidity'].mean(),inplace=True)
              data_weather['Minimum humidity'].replace(np.nan,data_weather['Minimum humidity'].mean(),inplace=True)
              data_weather['Maximum pressure'].replace(np.nan,data_weather['Maximum pressure'].mean(),inplace=True)
              data_weather['Minimum pressure'].replace(np.nan,data_weather['Minimum pressure'].mean(),inplace=True)
              data_weather['Maximum windspeed'].replace(np.nan,data_weather['Maximum windspeed'].mean(),inplace=True)
              data_weather['Maximum gust speed'].replace(np.nan,data_weather['Maximum gust speed'].mean(),inplace=True)
              data_weather['Maximum heat index'].replace(np.nan,data_weather['Maximum heat index'].mean(),inplace=True)
```

- Vérifions la disparition des valeurs manquantes.

```
Entrée [212]: data_weather.isna().sum()

Out[212]: date                    0
          Average temperature     0
          Average humidity        0
          Average dewpoint        0
          Average barometer       0
          Average windspeed       0
          Average gustspeed       0
          Average direction       0
          Rainfall for month      0
          Rainfall for year       0
          Maximum rain per minute 0
          Maximum temperature     0
          Minimum temperature     0
          Maximum humidity        0
          Minimum humidity        0
          Maximum pressure        0
          Minimum pressure        0
          Maximum windspeed       0
          Maximum gust speed      0
          Maximum heat index      0
          dtype: int64
```

- Ils nous restent deux tâches pour la partie préparation :

  - Standardisation des variables numérique
  - Encoder les variables catégorielles

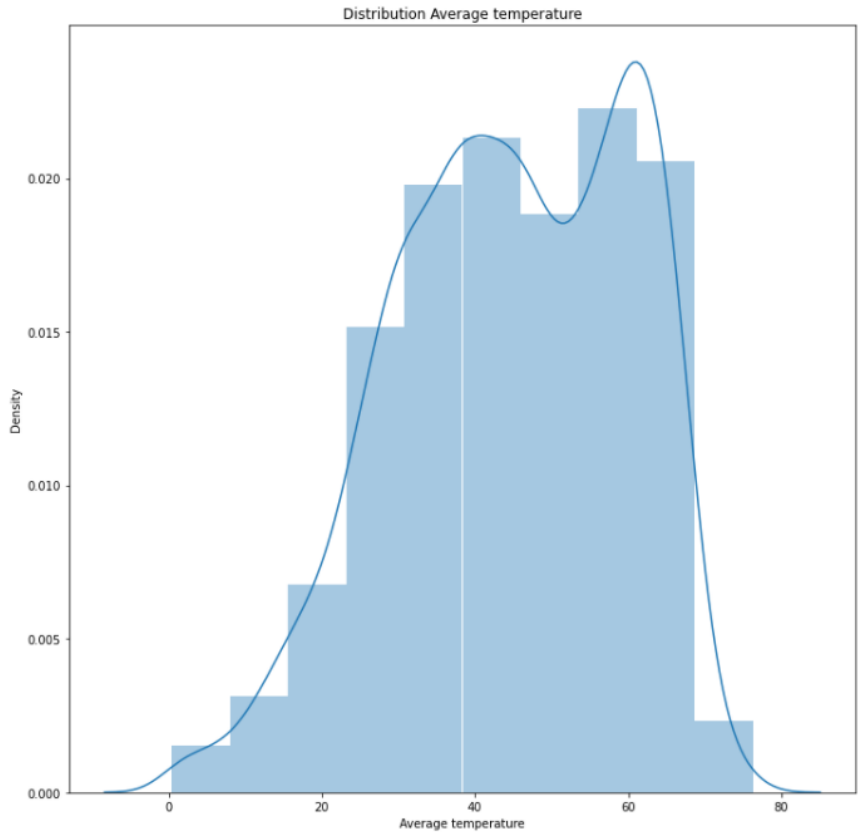
On va voire ses deux tâches au niveau de la partie construction de modèle. Dont laquelle on va crée un pipeline pour réaliser ses taches avec l'application de notre modèle.

IV. Analyse exploratoire des données

- Visualisation de la distribution de la variable target (température moyenne en °F) qu'on veut prédire.

```
Entrée [371]: sns.distplot(data_weather['Average temperature'], bins =10)
              plt.title('Distribution de la température moyene')
              plt.show()
```

```
C:\Users\Hinnovis\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function
and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar f
lexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

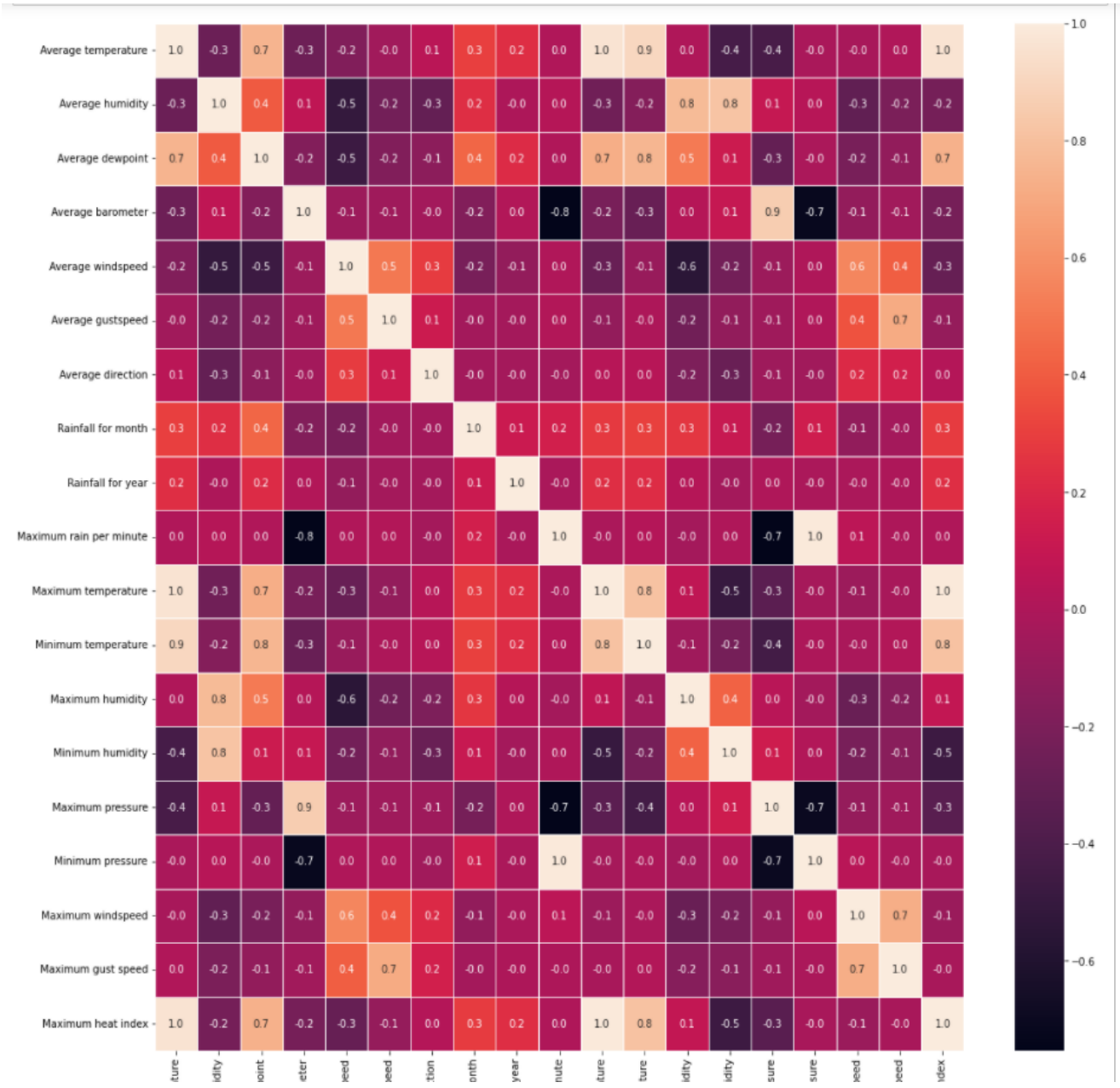


- Visualisons la matrice de corrélation entre les différentes variables

```
Entrée [214]: f,ax = plt.subplots(figsize=(18, 18))
              sns.heatmap(data_weather.corr(), annot=True, linewidths=.5, fmt= '.1f',ax=ax)
              plt.show()
```

- L'analyse de cette matrice de corrélation va nous permettre de réduire le nombre de variable existant. En effet on va conserver une seule variable entre deux variables, dont la corrélation est proche de 1.

- Analysons cette matrice de corrélation



Corrélation (Average temperature & Maximum temperature) = **1.0**
Corrélation (Average temperature & maximum heat index)= **1.0**
Corrélation (Average temperature & Minimum temperature) = **0.9**
Corrélation (Minimum pressure & Maximum rain per minute) = **1.0**
Corrélation (Maximum pressure & average barometerie) = **0.9**

- Donc on va supprimer les colonnes suivantes : **Maximum temperature , maximum heat index, Minimum temperature**, **Maximum rain per minute , average barometerie**.

- Suppression des colonnes ainsi sélectionnées.

```
Entrée [379]: data_weather_v2=data_weather.drop(columns=["Maximum temperature","Maximum heat index","Minimum temperature","Minimum pressure",'
```

```
Entrée [380]: data_weather_v2
```

Out[380]:

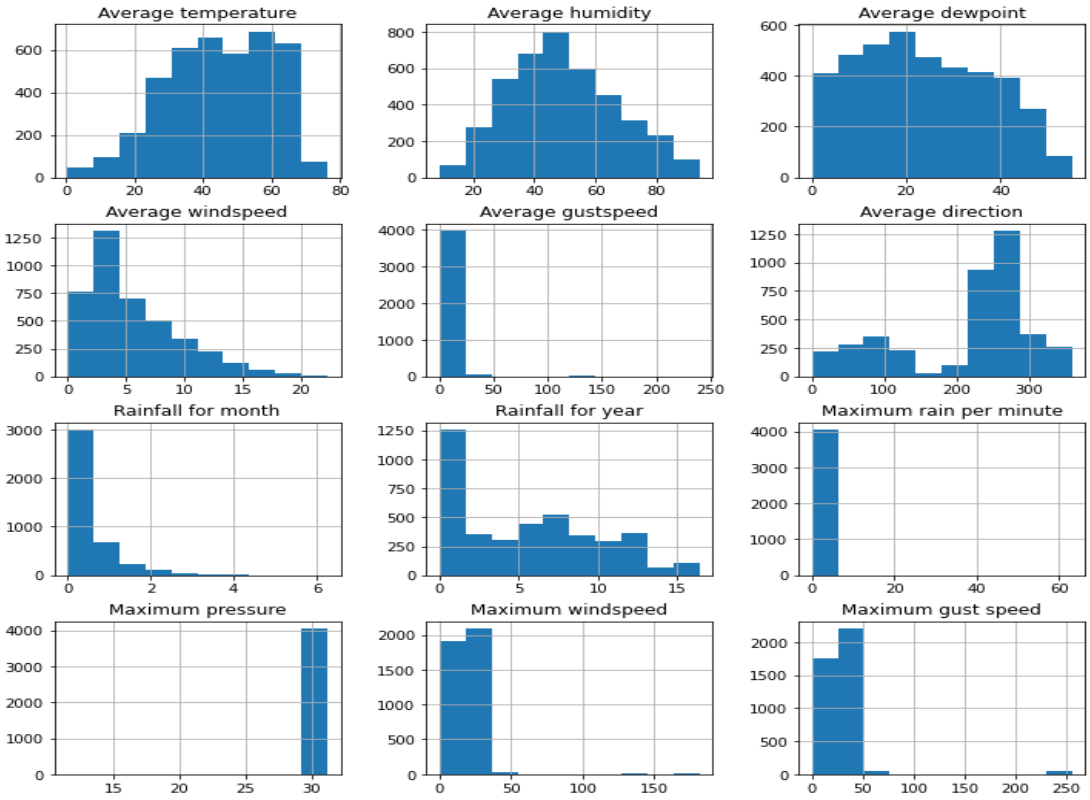
	date	Average temperature	Average humidity	Average dewpoint	Average windspeed	Average gustspeed	Average direction	Rainfall for month	Rainfall for year	Maximum rain per minute	Maximum humidity	Minimum humidity	Maximum pressure	Maximum windspeed	Maxim g sp
0	2010-01-01	32.1	49	15.2	14.6	19.7	297.0	0.0	0.00	0.0	61.0	41.0	30.330	34.5	
1	2010-01-02	32.1	50	15.5	8.8	12.2	306.0	0.0	0.00	0.0	77.0	39.0	30.292	23.0	
2	2010-01-03	23.1	64	12.1	2.9	4.4	21.0	0.0	0.00	0.0	85.0	34.0	30.549	17.3	
3	2010-01-04	25.7	48	7.2	4.7	7.0	324.0	0.0	0.00	0.0	82.0	25.0	30.585	15.0	
4	2010-01-05	34.3	51	17.8	14.2	18.5	300.0	0.0	0.00	0.0	60.0	43.0	30.173	34.5	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
4050	2021-12-27	22.7	76	15.9	2.8	4.5	242.0	0.5	1.04	0.0	90.0	53.0	29.751	18.4	
4051	2021-12-28	21.0	54	6.5	9.8	14.4	208.0	0.5	1.04	0.0	79.0	42.0	29.569	26.5	
4052	2021-12-29	23.5	57	10.4	8.3	12.1	234.0	0.5	1.04	0.0	75.0	50.0	29.758	17.3	
4053	2021-12-30	26.3	59	13.4	16.0	23.1	241.0	0.5	1.04	0.0	87.0	42.0	29.682	34.5	
4054	2021-12-31	20.7	87	17.6	2.6	4.1	352.0	0.5	1.04	0.0	92.0	77.0	29.645	11.5	

4055 rows × 15 columns

- Visualisation de la distribution de nos variables (diagrammes d'histogrammes

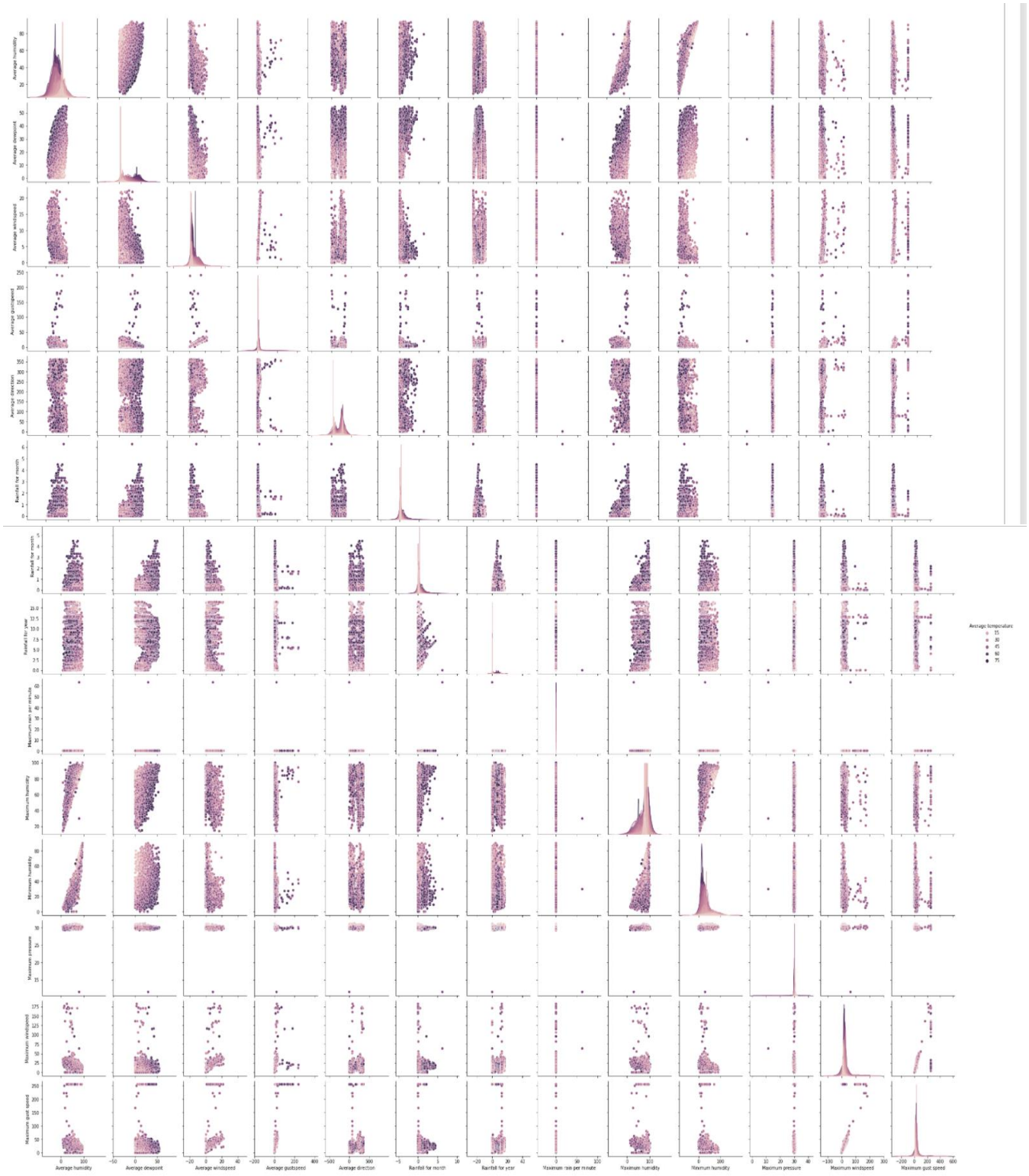
```
Entrée [329]: from matplotlib.pyplot import rcParams
              rcParams['figure.figsize'] = 12, 12
              data_weather_v2.hist()
```

```
Out[329]: array([[<AxesSubplot:title={'center':'Average temperature'}>,
                  <AxesSubplot:title={'center':'Average humidity'}>,
                  <AxesSubplot:title={'center':'Average dewpoint'}>],
                 [<AxesSubplot:title={'center':'Average windspeed'}>,
                  <AxesSubplot:title={'center':'Average gustspeed'}>,
                  <AxesSubplot:title={'center':'Average direction'}>],
                 [<AxesSubplot:title={'center':'Rainfall for month'}>,
                  <AxesSubplot:title={'center':'Rainfall for year'}>,
                  <AxesSubplot:title={'center':'Maximum rain per minute'}>],
                 [<AxesSubplot:title={'center':'Maximum pressure'}>,
                  <AxesSubplot:title={'center':'Maximum windspeed'}>,
                  <AxesSubplot:title={'center':'Maximum gust speed'}>]],
                dtype=object)
```

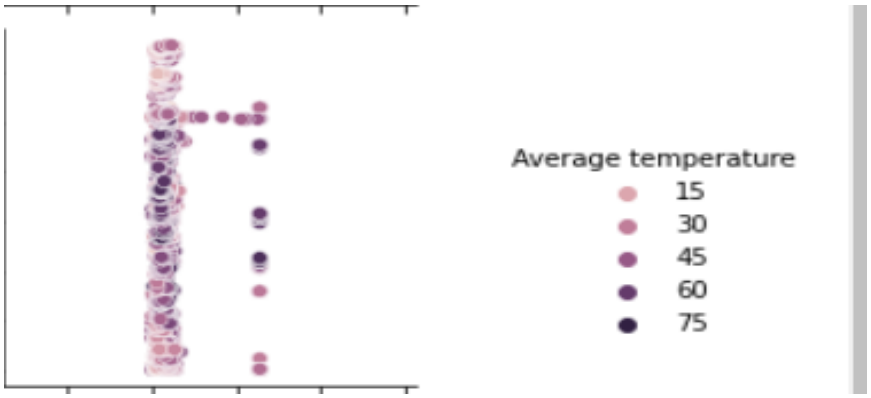


- Visualisation de la distribution de notre target par rapport au autre variables

```
Entrée [404]: plt.figure(figsize=(100,10))
              sns.pairplot(data_weather_v2,hue='Average temperature')
              plt.show()

              <Figure size 7200x720 with 0 Axes>
```



- Exemple rain-fall for month VS average gust speed.



IIV. Construction de modèle

- Fixons notre target (y) et la matrice X (data).

```
target=data_weather_v2["Average temperature"]
data=data_weather_v2.drop(columns="Average temperature")
```

- Sélectionnons les colonnes numériques pour les standardiser et les colonnes catégorielles pour les encoder .

Entrée [353]:
```
from sklearn.compose import make_column_selector as selector

numerical_columns_selector = selector(dtype_exclude=object)
categorical_columns_selector = selector(dtype_include=object)

numerical_columns = numerical_columns_selector(data)
categorical_columns = categorical_columns_selector(data)
```

- Initialisons nos transformateurs.

Entrée [354]:
```
from sklearn.preprocessing import OneHotEncoder, StandardScaler

categorical_preprocessor = OneHotEncoder(handle_unknown="ignore")
numerical_preprocessor = StandardScaler()
```

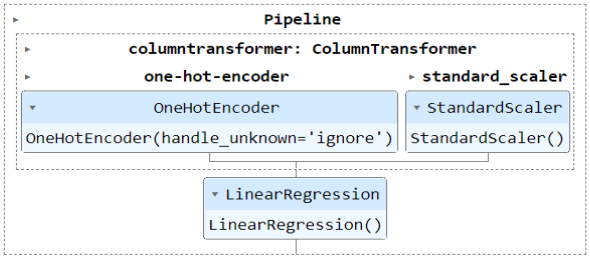
Entrée [355]:
```
from sklearn.compose import ColumnTransformer

preprocessor = ColumnTransformer([
    ('one-hot-encoder', categorical_preprocessor, categorical_columns),
    ('standard_scaler', numerical_preprocessor, numerical_columns)])
```

- Création d'un pipeline afin d'appliquer tous ses transformations au m temp

Entrée [423]:
```
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import make_pipeline
model = make_pipeline(preprocessor,LinearRegression())
```

Entrée [424]:
```
from sklearn import set_config
set_config(display='diagram')
model
```

Out[424]:



- Evaluons notre modèle avec un cross validation :

Entrée [429]:
```
from sklearn.model_selection import cross_validate
cv_results = cross_validate(model, data, target,cv=10)
scores = cv_results["test_score"]
print(f"The accuracy of our linear model is: {scores.mean():.3f} +/- {scores.std():.3f} (comme écart-type)")
```

```
The accuracy of our linear model is: 0.902 +/- 0.140 (comme écart-type)
```

Ps : nombre de split est initialiser à 10 (cross validation va diviser notre data set 10 fois en train_set and test_set et à chaque fois va générer un score. Ici 0.902 est la moyenne de tous ces scores.

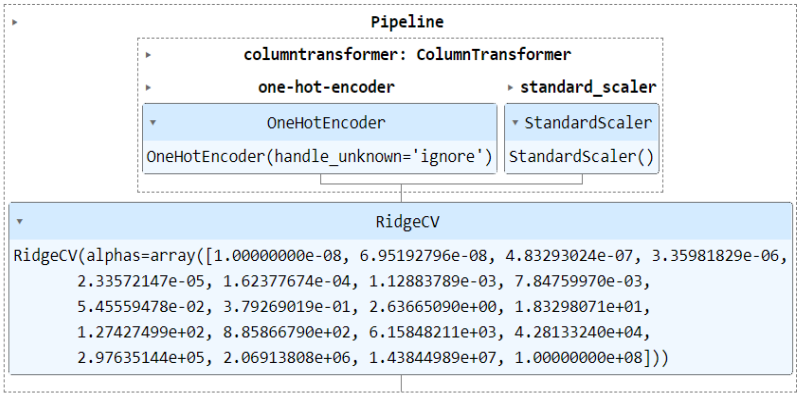
Hyperparamètres tuning

- On a décidé d'utiliser RidgeCV afin de régulariser notre modèle linéaire.
- Après on va comparer les deux model (model : linear et model2 :RidgeCV)
- On a commencé par initialiser une liste des alphas afin que notre modèle RidgeCV essayé avec chacune dan le but de trouver l'alpha optimale

Entrée [66]:
```
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import RidgeCV
model = make_pipeline(preprocessor,LinearRegression())
alpha=np.logspace(-8,8,20)
model2 = make_pipeline(preprocessor,RidgeCV(alphas=alpha))
```

Entrée [69]:
```
from sklearn import set_config
set_config(display='diagram')
model2
```

Out[69]:



- Résultats de RidgeCV :

Entrée [70]:
```
from sklearn.model_selection import cross_validate
cv_results = cross_validate(model2, data, target,cv=10)
scores = cv_results["test_score"]
print(f"The accuracy  of our RidgeCV model is: is: {scores.mean():.3f} +/- {scores.std():.3f} (comme écart-type)")
```

```
The accuracy  of our RidgeCV model is: is: 0.896 +/- 0.157 (comme écart-type)
```

- Résultats de modèle linear :

Entrée [71]:
```
from sklearn.model_selection import cross_validate
cv_results = cross_validate(model, data, target,cv=10)
scores = cv_results["test_score"]
print(f"The accuracy of oUR LINEAR model is: {scores.mean():.3f} +/- {scores.std():.3f} (comme écart-type)")
```

```
The accuracy of oUR LINEAR model is: 0.896 +/- 0.158 (comme écart-type)
```

PAS de grande changement presque les même (dans ce cas).

❖Etude de notre dataset comme série temporelle.

- On va commencer par copier notre dataset préparer dans une fichier csv, après en importent le fichier avec la colonne date en index.

```
Entrée [97]: data_weather.to_csv("data_w",index=False)

Entrée [111]: data_weather=pd.read_csv(r"C:\Users\Hinnovis\Downloads\data_w",index_col='date',parse_dates=True)

Entrée [112]: data_weather.head()
```

Out[112]:

date	Average temperature	Average humidity	Average dewpoint	Average barometer	Average windspeed	Average gustspeed	Average direction	Rainfall for month	Rainfall for year	Maximum rain per minute	Maximum temperature	Minimum temperature	Maximum humidity	Minimu humidit
2010-01-01	32.1	49	15.2	30.2	14.6	19.7	297.0	0.0	0.0	0.0	40.0	22.1	61.0	41.
2010-01-02	32.1	50	15.5	30.1	8.8	12.2	306.0	0.0	0.0	0.0	39.1	22.0	77.0	39.
2010-01-03	23.1	64	12.1	30.4	2.9	4.4	21.0	0.0	0.0	0.0	33.4	9.2	85.0	34.
2010-01-04	25.7	48	7.2	30.4	4.7	7.0	324.0	0.0	0.0	0.0	36.5	7.6	82.0	25.
2010-01-05	34.3	51	17.8	30.0	14.2	18.5	300.0	0.0	0.0	0.0	40.1	28.4	60.0	43.

✓ Visualisation

- Visualisons la distribution de « Average temperature » et « Rainfall for month » durant l'année 2020.

```
Entrée [113]: data_weather['2020']['Rainfall for month'].plot()
```

```
C:\PROGRA~1\KMSpico\temp/ipykernel_3388/1880299972.py:1: FutureWarning: Indexing a DataFrame with a datetimelike index using a single string to slice the rows, like `frame[string]`, is deprecated and will be removed in a future version. Use `frame.loc[string]` instead.
  data_weather['2020']['Rainfall for month'].plot()
```

Out[113]: <AxesSubplot:xlabel='date'>

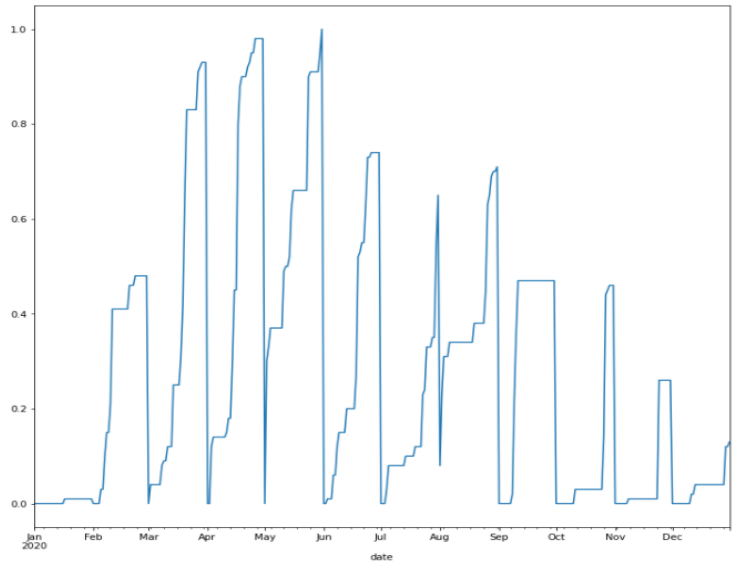


Figure : rainfall for month distribution in 2020

```
Entrée [114]: data_weather['2020']['Average temperature'].plot()
```

```
C:\PROGRA~1\KMSpico\temp/ipykernel_3388/1293131550.py:1: FutureWarning: Indexing a DataFrame with a datetimelike index using a
single string to slice the rows, like `frame[string]`, is deprecated and will be removed in a future version. Use `frame.loc[st
ring]` instead.
  data_weather['2020']['Average temperature'].plot()
```

```
Out[114]: <AxesSubplot:xlabel='date'>
```

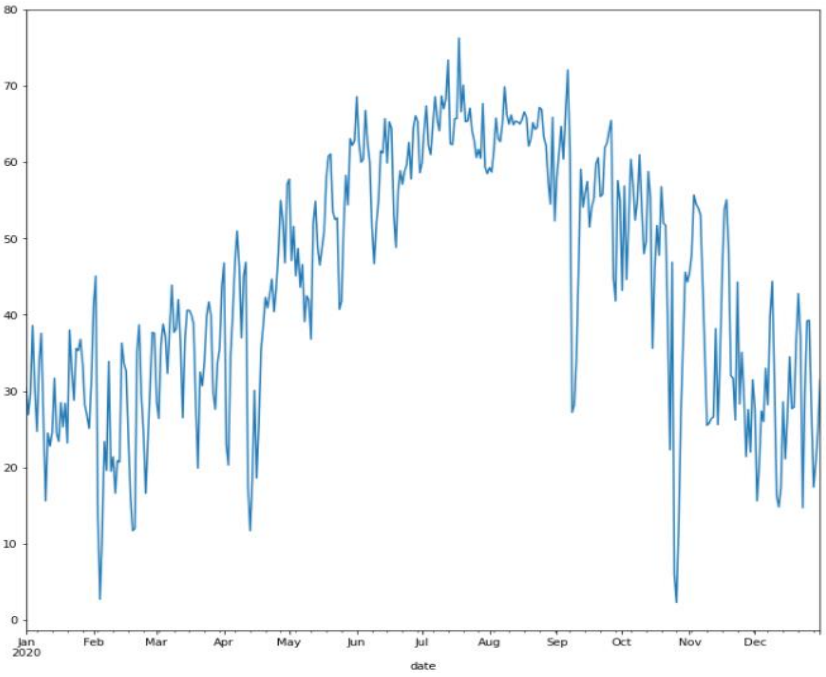
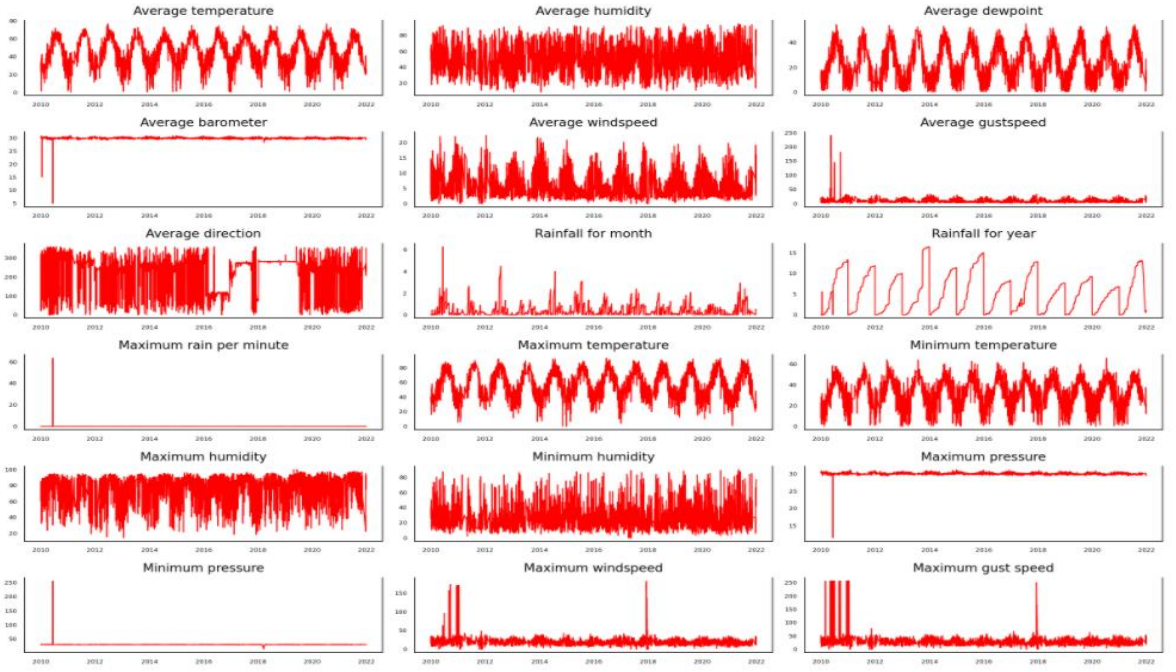


Figure : distribution of Average temperature in 2020

- La distribution de nos variables entre 2010 et 2022

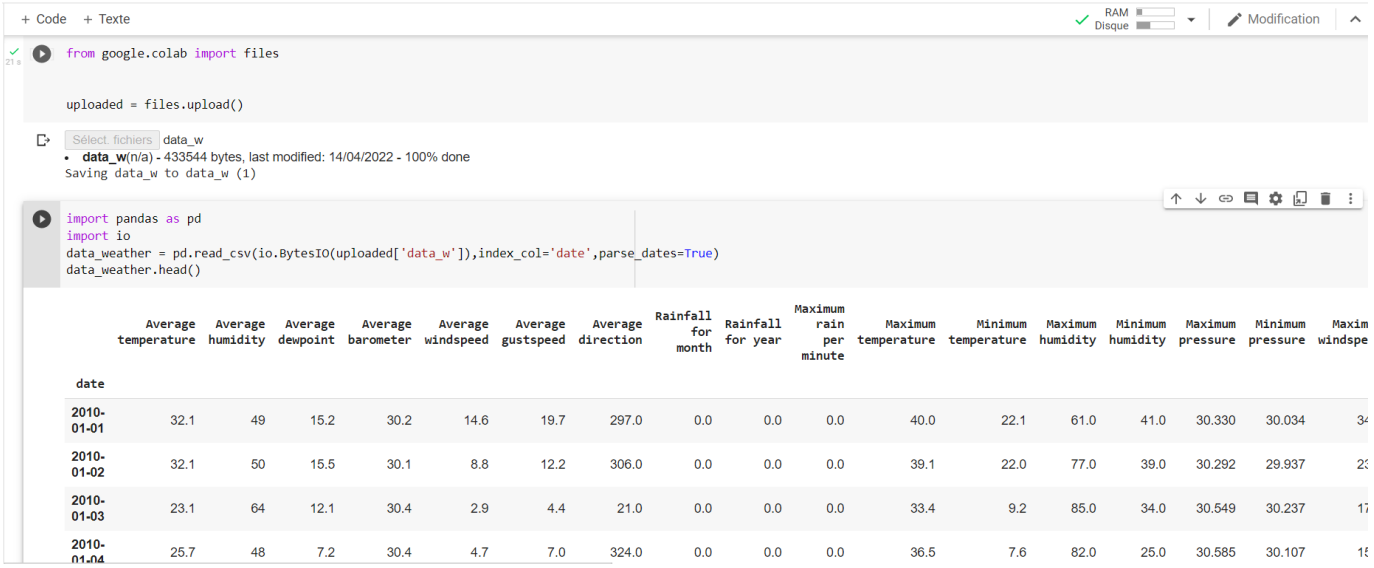
```
Entrée [115]: fig, axes = plt.subplots(nrows=6, ncols=3, dpi=160, figsize=(15,10))
for i, ax in enumerate(axes.flatten()):
    data = data_weather[data_weather.columns[i]]
    ax.plot(data, color='red', linewidth=1)
    ax.set_title(data_weather.columns[i])
    ax.xaxis.set_ticks_position('none')
    ax.yaxis.set_ticks_position('none')
    ax.spines["top"].set_alpha(0)
    ax.tick_params(labelsize=6)
plt.tight_layout();
```



❖ Construction d'un modèle LSTM neural network (deep learning)

On a switcher vers google collab car il contient toutes les bibliothèques nécessaires afin de réaliser cette partie.

- Importations de 'data_weather' sous google collab.



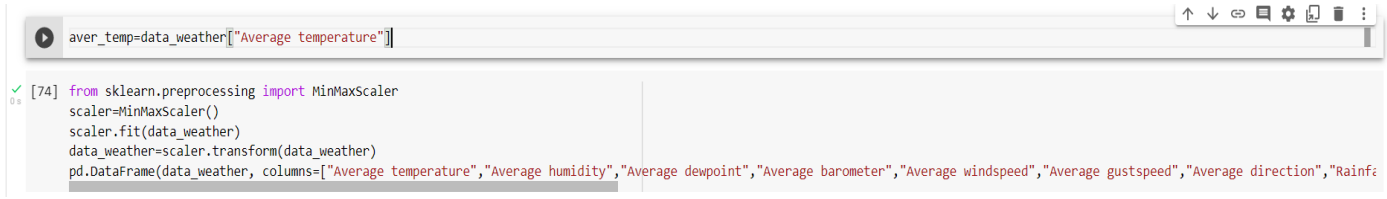
```
from google.colab import files

uploaded = files.upload()
```

```
Sélect. fichiers  data_w
• data_w(n/a) - 433544 bytes, last modified: 14/04/2022 - 100% done
Saving data_w to data_w (1)
```

```
import pandas as pd
import io
data_weather = pd.read_csv(io.BytesIO(uploaded['data_w']),index_col='date',parse_dates=True)
data_weather.head()
```

date	Average temperature	Average humidity	Average dewpoint	Average barometer	Average windspeed	Average gustspeed	Average direction	Rainfall for month	Rainfall for year	Maximum rain per minute	Maximum temperature	Minimum temperature	Maximum humidity	Minimum humidity	Maximum pressure	Minimum pressure	Maxim windspe
2010-01-01	32.1	49	15.2	30.2	14.6	19.7	297.0	0.0	0.0	0.0	40.0	22.1	61.0	41.0	30.330	30.034	34
2010-01-02	32.1	50	15.5	30.1	8.8	12.2	306.0	0.0	0.0	0.0	39.1	22.0	77.0	39.0	30.292	29.937	23
2010-01-03	23.1	64	12.1	30.4	2.9	4.4	21.0	0.0	0.0	0.0	33.4	9.2	85.0	34.0	30.549	30.237	17
2010-01-04	25.7	48	7.2	30.4	4.7	7.0	324.0	0.0	0.0	0.0	36.5	7.6	82.0	25.0	30.585	30.107	15

- On stocke notre target Average temperature dans un variable aver_temp, et par la suite on standardise notre dataset avec **MinMaxScaler** et enfin on la rendre sous forme de dataframe.



```
aver_temp=data_weather["Average temperature"]
```

```
[74] from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler()
scaler.fit(data_weather)
data_weather=scaler.transform(data_weather)
pd.DataFrame(data_weather, columns=["Average temperature","Average humidity","Average dewpoint","Average barometer","Average windspeed","Average gustspeed","Average direction","Rainfa
```

- Voici notre data set après scaling

	Average temperature	Average humidity	Average dewpoint	Average barometer	Average windspeed	Average gustspeed	Average direction	Rainfall for month	Rainfall for year	Maximum rain per minute	Maximum temperature	Minimum temperature	Maximum humidity	Minimum humidity	Maximum pressure	Minimum pressure	Maximu windspee
0	0.417655	0.470588	0.275862	0.969349	0.654709	0.081947	0.825000	0.000000	0.000000	0.0	0.431034	0.336377	0.546512	0.455556	0.955644	0.069366	0.18987
1	0.417655	0.482353	0.281307	0.965517	0.394619	0.050749	0.850000	0.000000	0.000000	0.0	0.421336	0.334855	0.732558	0.433333	0.953715	0.068964	0.12658
2	0.299078	0.647059	0.219601	0.977011	0.130045	0.018303	0.058333	0.000000	0.000000	0.0	0.359914	0.140030	0.825581	0.377778	0.966758	0.070205	0.09521
3	0.333333	0.458824	0.130672	0.977011	0.210762	0.029118	0.900000	0.000000	0.000000	0.0	0.393319	0.115677	0.790698	0.277778	0.968585	0.069667	0.08255
4	0.446640	0.494118	0.323049	0.961686	0.636771	0.076955	0.833333	0.000000	0.000000	0.0	0.432112	0.432268	0.534884	0.477778	0.947676	0.068799	0.18987
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4050	0.293808	0.788235	0.288566	0.946360	0.125561	0.018719	0.672222	0.079872	0.063376	0.0	0.314655	0.258752	0.883721	0.588889	0.926259	0.066433	0.10126
4051	0.271410	0.529412	0.117967	0.938697	0.439462	0.059900	0.577778	0.079872	0.063376	0.0	0.271552	0.220700	0.755814	0.466667	0.917022	0.066391	0.14584
4052	0.304348	0.564706	0.188748	0.950192	0.372197	0.050333	0.650000	0.079872	0.063376	0.0	0.262931	0.220700	0.709302	0.555556	0.926614	0.067438	0.09521
4053	0.341238	0.588235	0.243194	0.942529	0.717489	0.096090	0.669444	0.079872	0.063376	0.0	0.329741	0.325723	0.848837	0.466667	0.922757	0.066854	0.18987

- Voila on a arrivé à la fonction la plus important pour appliquer notre modèle LSTM, en effet le modèle LSTM apprendra une fonction qui mappe une séquence d'observations passées comme entrée avec une observation de sortie.
- C'est exactement ce que la fonction au-dessous fait. Elle prend deux arguments en entrée data_weather c'est notre dataset et window_size c'est le nombre d'observations, on a fixé le ici en 60. C'est-à-dire notre modèle va prendre en considération 60 observation (60 jours) pour prédire le 61.
  Par exemple : LSTM va prendre de 1 à 60 et prédire 61 par la suite de 2 à 61 et prédire 62 et à chaque fois va stock les 60 valeurs dans X en aura (4055-60=3995 valeur) et le résultat dans y. C'est la fonction au dessous.

```
[75] def data_weather_to_X_and_y(data_weather, window_size=60):
       data_weather_as_np = data_weather.to_numpy()
       X = []
       y = []
       for i in range(len(data_weather_as_np)-window_size):
         row = [[a] for a in data_weather_as_np[i:i+window_size]]
         X.append(row)
         label = data_weather_as_np[i+window_size]
         y.append(label)
       return np.array(X), np.array(y)
```

- Les dimensions de X et y retournées par la fonction ainsi expliquée.

```
[102] WINDOW_SIZE = 60
      X1, y1 = data_weather_to_X_and_y(aver_temp, WINDOW_SIZE)
      X1.shape, y1.shape

      ((3995, 60, 1), (3995,))
```

C'est exactement ce que j'ai expliqué avant.

- Maintenant on va diviser nos données en train| test | validation set.

```
[77] X_train1, y_train1 = X1[:3100], y1[:3100]
     X_val1, y_val1 = X1[3100:3600], y1[3100:3600]
     X_test1, y_test1 = X1[3600:], y1[3600:]
     X_train1.shape, y_train1.shape, X_val1.shape, y_val1.shape, X_test1.shape, y_test1.shape

     ((3100, 60, 1), (3100,), (500, 60, 1), (500,), (395, 60, 1), (395,))
```

- Construction de notre modèle LSTM

```
[78] from tensorflow.keras.models import Sequential
     from tensorflow.keras.layers import *
     from tensorflow.keras.callbacks import ModelCheckpoint
     from tensorflow.keras.losses import MeanSquaredError
     from tensorflow.keras.metrics import RootMeanSquaredError
     from tensorflow.keras.optimizers import Adam

     model1 = Sequential()
     model1.add(InputLayer((60, 1)))
     model1.add(LSTM(64))
     model1.add(Dense(10, 'relu'))
     model1.add(Dense(1, 'linear'))

     model1.summary()
```

```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_8 (LSTM)               (None, 64)                16896

 dense_16 (Dense)            (None, 10)                650

 dense_17 (Dense)            (None, 1)                 11

=================================================================
Total params: 17,557
Trainable params: 17,557
Non-trainable params: 0
```

- Compilation de notre model LSTM
- Cp1 pour conserver the best modèle ,et on a utilisé Adam comme optimizer avec un learning_rate =0.0001 pour éviter la divergence de modèle.

```
cp1 = ModelCheckpoint('model1/', save_best_only=True)
model1.compile(loss=MeanSquaredError(), optimizer=Adam(learning_rate=0.0001), metrics=[RootMeanSquaredError()])
```

- En fin en fit notre modèle avec un nombre d'epochs qui vaut 60 afin de déminuer les erreurs le plus maximum.

```
model1.fit(X_train1, y_train1, validation_data=(X_val1, y_val1), epochs=60,callbacks=[cp1])

Epoch 1/60
97/97 [==============================] - ETA: 0s - loss: 41.0119 - root_mean_squared_error: 6.4041WARNING:absl:Found untraced functions such as lstm_cell_14_layer_call_fn, lstm_c
INFO:tensorflow:Assets written to: model1/assets
INFO:tensorflow:Assets written to: model1/assets
WARNING:absl:<keras.layers.recurrent.LSTMCell object at 0x7f416edd7e50> has the same name 'LSTMCell' as a built-in Keras object. Consider renaming <class 'keras.layers.recurrent.
97/97 [==============================] - 12s 99ms/step - loss: 41.0119 - root_mean_squared_error: 6.4041 - val_loss: 49.4456 - val_root_mean_squared_error: 7.0318
Epoch 2/60
97/97 [==============================] - 3s 34ms/step - loss: 40.8865 - root_mean_squared_error: 6.3943 - val_loss: 49.4970 - val_root_mean_squared_error: 7.0354
Epoch 3/60
97/97 [==============================] - 3s 34ms/step - loss: 40.8094 - root_mean_squared_error: 6.3882 - val_loss: 49.5851 - val_root_mean_squared_error: 7.0417
Epoch 4/60
97/97 [==============================] - ETA: 0s - loss: 40.7418 - root_mean_squared_error: 6.3829WARNING:absl:Found untraced functions such as lstm_cell_14_layer_call_fn, lstm_c
INFO:tensorflow:Assets written to: model1/assets
INFO:tensorflow:Assets written to: model1/assets
WARNING:absl:<keras.layers.recurrent.LSTMCell object at 0x7f416edd7e50> has the same name 'LSTMCell' as a built-in Keras object. Consider renaming <class 'keras.layers.recurrent.
97/97 [==============================] - 8s 87ms/step - loss: 40.7418 - root_mean_squared_error: 6.3829 - val_loss: 49.2316 - val_root_mean_squared_error: 7.0165
Epoch 5/60
97/97 [==============================] - 3s 34ms/step - loss: 40.5945 - root_mean_squared_error: 6.3714 - val_loss: 49.5864 - val_root_mean_squared_error: 7.0418
Epoch 6/60
97/97 [==============================] - 3s 34ms/step - loss: 40.5459 - root_mean_squared_error: 6.3676 - val_loss: 49.3185 - val_root_mean_squared_error: 7.0227
Epoch 7/60
96/97 [============================>.] - ETA: 0s - loss: 40.5509 - root_mean_squared_error: 6.3680WARNING:absl:Found untraced functions such as lstm_cell_14_layer_call_fn, lstm_c
INFO:tensorflow:Assets written to: model1/assets
INFO:tensorflow:Assets written to: model1/assets
WARNING:absl:<keras.layers.recurrent.LSTMCell object at 0x7f416edd7e50> has the same name 'LSTMCell' as a built-in Keras object. Consider renaming <class 'keras.layers.recurrent.
97/97 [==============================] - 9s 88ms/step - loss: 40.5730 - root_mean_squared_error: 6.3697 - val_loss: 49.0022 - val_root_mean_squared_error: 7.0002
Epoch 8/60
97/97 [==============================] - ETA: 0s - loss: 40.4573 - root_mean_squared_error: 6.3606WARNING:absl:Found untraced functions such as lstm_cell_14_layer_call_fn, lstm_c
INFO:tensorflow:Assets written to: model1/assets
INFO:tensorflow:Assets written to: model1/assets
WARNING:absl:<keras.layers.recurrent.LSTMCell object at 0x7f416edd7e50> has the same name 'LSTMCell' as a built-in Keras object. Consider renaming <class 'keras.layers.recurrent.
```

- The loss function et les erreurs déminue après chaque époch (de fait les parmétres change à chaque fois

```
97/97 [==============================] - 8s 88ms/step - loss: 39.6943 - root_mean_squared_error: 6.3003 - val_loss: 47.9285 - val_root_mean_squared_error:
Epoch 54/60
97/97 [==============================] - ETA: 0s - loss: 39.7160 - root_mean_squared_error: 6.3021WARNING:absl:Found untraced functions such as lstm_cell_14_layer_call_fn, lstm_c
INFO:tensorflow:Assets written to: model1/assets
INFO:tensorflow:Assets written to: model1/assets
WARNING:absl:<keras.layers.recurrent.LSTMCell object at 0x7f416edd7e50> has the same name 'LSTMCell' as a built-in Keras object. Consider renaming <class 'keras.layers.recurrent.
97/97 [==============================] - 9s 93ms/step - loss: 39.7160 - root_mean_squared_error: 6.3021 - val_loss: 47.8678 - val_root_mean_squared_error: 6.9187
Epoch 55/60
97/97 [==============================] - 3s 34ms/step - loss: 39.7847 - root_mean_squared_error: 6.3075 - val_loss: 48.1611 - val_root_mean_squared_error: 6.9398
Epoch 56/60
97/97 [==============================] - 3s 33ms/step - loss: 39.8668 - root_mean_squared_error: 6.3140 - val_loss: 48.4229 - val_root_mean_squared_error: 6.9587
Epoch 57/60
97/97 [==============================] - ETA: 0s - loss: 39.6387 - root_mean_squared_error: 6.2959WARNING:absl:Found untraced functions such as lstm_cell_14_layer_call_fn, lstm_c
INFO:tensorflow:Assets written to: model1/assets
INFO:tensorflow:Assets written to: model1/assets
WARNING:absl:<keras.layers.recurrent.LSTMCell object at 0x7f416edd7e50> has the same name 'LSTMCell' as a built-in Keras object. Consider renaming <class 'keras.layers.recurrent.
97/97 [==============================] - 8s 86ms/step - loss: 39.6387 - root_mean_squared_error: 6.2959 - val_loss: 47.8433 - val_root_mean_squared_error: 6.9169
Epoch 58/60
97/97 [==============================] - 3s 33ms/step - loss: 39.6472 - root_mean_squared_error: 6.2966 - val_loss: 48.0625 - val_root_mean_squared_error: 6.9327
Epoch 59/60
97/97 [==============================] - 3s 34ms/step - loss: 39.5657 - root_mean_squared_error: 6.2901 - val_loss: 47.9886 - val_root_mean_squared_error: 6.9274
Epoch 60/60
97/97 [==============================] - 3s 33ms/step - loss: 39.6689 - root_mean_squared_error: 6.2983 - val_loss: 47.9728 - val_root_mean_squared_error: 6.9262
<keras.callbacks.History at 0x7f4174b62890>
```

- Rechargé le modèle dont la fonction loss est la plus faible

```
[ ] from tensorflow.keras.models import load_model
    model1 = load_model('model1/')
```

- Prédiction de la X_train et comparaison avec les valeurs de y_train.

```
[30] train_predictions = model1.predict(X_train1).flatten()
     train_results = pd.DataFrame(data={'Train Predictions':train_predictions, 'Actuals':y_train1})
     train_results
```

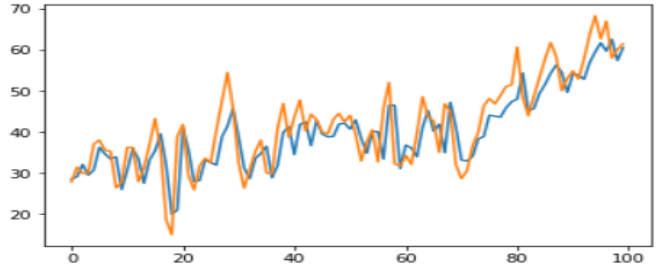
	Train Predictions	Actuals
0	28.511219	27.8
1	29.097788	31.4
2	32.083408	29.9
3	29.477680	29.9
4	30.717306	37.0
...	...	...
3095	55.347202	58.0
3096	55.092495	55.5
3097	54.059719	45.1
3098	46.567032	51.5
3099	54.447796	53.2

3100 rows × 2 columns

- Voici la visualisation des vrais valeurs y_train avec les prédictions X_train

```
import matplotlib.pyplot as plt
plt.plot(train_results['Train Predictions'][0:100])
plt.plot(train_results['Actuals'][0:100])
```

```
[<matplotlib.lines.Line2D at 0x7f1a7fa9b610>]
```



- Prédiction de la X_test et comparaison avec les valeurs de y_test.

```
[28] test_predictions = model1.predict(X_test1).flatten()
     test_results = pd.DataFrame(data={'Test Predictions':test_predictions, 'Actuals':y_test1})
     test_results
```

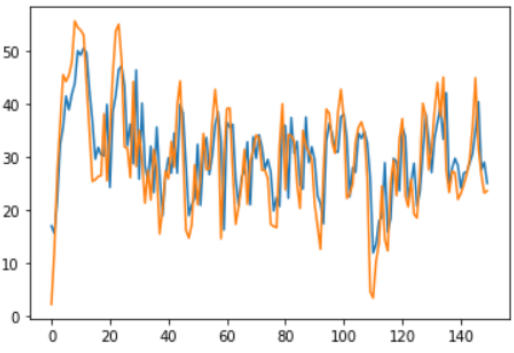
	Test Predictions	Actuals
0	17.072840	2.3
1	15.706940	12.4
2	21.839497	27.1
3	32.420582	36.5
4	35.666721	45.6
...	...	...
390	26.045780	22.7
391	26.631895	21.0
392	24.298073	23.5
393	27.513731	26.3
394	28.549788	20.7

395 rows × 2 columns

- Voici la visualisation des vrais valeurs y_test avec les prédictions X_test

```
import matplotlib.pyplot as plt
plt.plot(test_results['Test Predictions'][:150])
plt.plot(test_results['Actuals'][:150])
```

```
[<matplotlib.lines.Line2D at 0x7f1a7f5f6890>]
```



Observation :

On observe que notre modèle à réussie la prédiction de l'Average temperature avec un minimum d'erreur.comme on voie dans le graphe les deux courbes (bleu et orange) sont presque confondue.

On va refaire les mêmes procédures afin de prédire la colonne :

Rainfall for month

❖ Prédiction de la colonnes Rainfall for month en utilisant LSTM

En refait le même processus qu'on a fait précédement tout en remplaçons la colonne **'Average temperature'** par la colonne **'Rainfall for month'**

- Prédiction de la X_test et comparaison avec les valeurs de y_test.

```
[19] test_predictions = model1.predict(X_test1).flatten()
     test_results = pd.DataFrame(data={'Test Predictions of rainfall for month':test_predictions, 'Actuals':y_test1})
     test_results
```

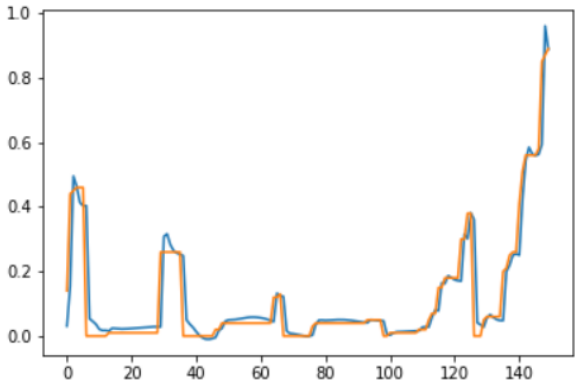
	Test Predictions of rainfall for month	Actuals
0	0.030737	0.14
1	0.154026	0.44
2	0.495242	0.45
3	0.464207	0.46
4	0.413789	0.46
...	...	...
390	0.534284	0.50
391	0.504884	0.50
392	0.497077	0.50
393	0.495200	0.50
394	0.492470	0.50

395 rows × 2 columns

- Voici la visualisation des vrais valeurs y_test avec les prédictions X_test

```
[21] import matplotlib.pyplot as plt
     plt.plot(test_results['Test Predictions of rainfall for month'][:150])
     plt.plot(test_results['Actuals'][:150])
```

```
[<matplotlib.lines.Line2D at 0x7f5fd9ba4610>]
```



Voila l'erreur est très minimale, donc on pourra dire que notre modèle LSTM à bien réussie la prédiction de la série de précipitation mensuelles.
De la même manière on fait les prédictions des autres colonnes.
Voilà on a atteint l'objective de notre projet !