Meteo_Martin

Martin J

November 11, 2021

1 Projet DATA : Prédiction de la météo

Martin J

Projet de prédiction de la météo à Paris en utilisant les données de Meteonet.

1.1 Analyse du problème

Objectif : Prédire la météo à Paris en ayant les données météorologiques des stations du quart nord-ouest et du quart sud-est de la France. Nous avons en donnée d'entrée une date, des coordonnées de latitude, longitude et la hauteur, et nous voulons que le modèle prévoit la météo de cette date.

Pour simplifier nous allons essayer de déterminer la température à Paris sur 24h connaissant toutes les données sur les 7 derniers jours sur un certains nombre de station

Modules nécessaires : * pandas pour la manipulation de données * seaborn pour l'affichage des données * matplotlib pour les graphes * cartopy pour afficher la carte de France * datetime pour la gestion des dates * numpy pour le calcul vectoriel et la manipulation des `numpy.ndarray`

1.2 Importation des bibliothèques nécessaires

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
import datetime as dt
from time import time
from datetime import timedelta, datetime
import seaborn as sb

mpl.style.use('ggplot')
```

1.3 Import des données

Nous allons d'abord importer et visualiser l'ensemble des données pour l'année 2016.

Telecharger `NW_Ground_Stations_2016.csv` sur le site de Meteonet ou sur leur page Kaggle. Renseigner le chemin de ce fichier dans `fname`

[2]:
```
year = '2016'
fname = 'data/NW_Ground_Stations_'+year+".csv"
df = pd.read_csv(fname,parse_dates=[4],infer_datetime_format=True)
df.head(25)
```

[2]:

	number_sta	lat	lon	height_sta	date	dd	ff	precip	hu	td	t	psl
0	14066001	49.330	-0.430	2.0	2016-01-01	210.0	4.4	0.0	91.0	278.45	279.85	NaN
1	14126001	49.150	0.040	125.0	2016-01-01	NaN	NaN	0.0	99.0	278.35	278.45	NaN
2	14137001	49.180	-0.460	67.0	2016-01-01	220.0	0.6	0.0	92.0	276.45	277.65	102360.0
3	14216001	48.930	-0.150	155.0	2016-01-01	220.0	1.9	0.0	95.0	278.25	278.95	NaN
4	14296001	48.800	-1.030	339.0	2016-01-01	NaN	NaN	0.0	NaN	NaN	278.35	NaN
5	14357002	48.930	-0.690	223.0	2016-01-01	NaN	NaN	0.0	NaN	NaN	277.65	NaN
6	14366002	49.170	0.230	62.0	2016-01-01	NaN	NaN	0.0	NaN	NaN	279.55	NaN
7	14372001	49.102	-0.765	184.0	2016-01-01	230.0	4.1	0.0	92.0	278.05	279.25	NaN
8	14501002	48.890	-0.390	185.0	2016-01-01	NaN	NaN	0.0	NaN	NaN	278.35	NaN
9	14515001	49.350	-0.770	68.0	2016-01-01	220.0	5.1	NaN	NaN	NaN	NaN	NaN
10	14577003	49.280	-0.560	15.0	2016-01-01	NaN	NaN	0.0	99.0	279.65	279.75	NaN
11	14578001	49.360	0.170	143.0	2016-01-01	230.0	2.5	0.0				
12	14624001	48.990	-0.010	52.0	2016-01-01	NaN	NaN	0.0				
13	14659001	49.060	-0.230	62.0	2016-01-01	NaN	NaN	0.0				
14	14762004	48.850	-0.900	100.0	2016-01-01	170.0	1.0	0.0				
15	17218001	46.315	-1.000	2.0	2016-01-01	NaN	NaN	0.0				
16	22016001	48.857	-3.005	25.0	2016-01-01	NaN	NaN	0.0				
17	22092001	48.400	-3.150	281.0	2016-01-01	160.0	2.3	0.0				
18	22113006	48.760	-3.470	87.0	2016-01-01	150.0	2.7	0.0				
19	22135001	48.550	-3.380	148.0	2016-01-01	170.0	1.2	0.0				
20	22147006	48.180	-2.410	131.0	2016-01-01	NaN	NaN	0.0				
21	22168001	48.830	-3.470	58.0	2016-01-01	200.0	3.7	0.0				
22	22219003	48.270	-2.750	235.0	2016-01-01	180.0	0.9	0.0				
23	22247002	48.740	-3.250	55.0	2016-01-01	0.0	0.0	0.0				
24	22261002	48.520	-2.420	71.0	2016-01-01	0.0	0.0	0.0				

```
11  96.0  277.45  278.05  102380.0
12   NaN     NaN  278.05       NaN
13   NaN     NaN  279.15       NaN
14  91.0  276.55  277.85       NaN
15   NaN     NaN  280.05       NaN
16   NaN     NaN  278.55       NaN
17  97.0  277.05  277.45       NaN
18  97.0  275.95  276.35  102220.0
19  97.0  275.05  275.45       NaN
20   NaN     NaN  275.65       NaN
21  79.0  275.75  279.15  102180.0
22  99.0  276.05  276.15       NaN
23   NaN     NaN  276.35       NaN
24  96.0  274.75  275.35       NaN
```

2 1. Exploration de la base de données

A quoi correspondent les différents paramètres ? `number_sta` : numéro de la station

`lat` : latitude

`lon` : longitude

`height_sta` : la hauteur de la station en mètre

`date` : la date

`dd` : la direction du vent

`ff` : la vitesse du vent en mètre par seconde

`precip` : la précipitation pendant la durée d'enregistrement en kg.mˆ2

`hu` : l'humidité en %

`td` : la température de condensation en Kelvin

`t` : la température en Kelvin

`psl` : la pression amené au niveau de la mer

2.1 Traitement des données

On remarque que certains paramètres sont souvent NaN, voyons la proportion pour chacun d'entre eux.

[3]:
```
df.isnull().sum()/df.shape[0]
```

[3]:
```
number_sta    0.000000
lat           0.000000
```

```
lon            0.000000
height_sta     0.000000
date           0.000000
dd             0.401370
ff             0.400872
precip         0.059082
hu             0.415354
td             0.415693
t              0.162504
psl            0.799442
dtype: float64
```

On constate que la pression a 80% de valeur “non indiquée” et que le numéro de la station n’est qu’un identifiant.

On décidera donc de ne pas prendre en compte la pression dans certains cas, car elle a une trop grande proportion NaN, ni le numbe_sta, car il s’agit uniquement d’un identifiant.

[4]:
```
df = df.iloc[:,0:-1]
df.head(15)
```

[4]:

	number_sta	lat	lon	height_sta	date	dd	ff	precip
0	14066001	49.330	-0.430	2.0	2016-01-01	210.0	4.4	0.0
1	14126001	49.150	0.040	125.0	2016-01-01	NaN	NaN	0.0
2	14137001	49.180	-0.460	67.0	2016-01-01	220.0	0.6	0.0
3	14216001	48.930	-0.150	155.0	2016-01-01	220.0	1.9	0.0
4	14296001	48.800	-1.030	339.0	2016-01-01	NaN	NaN	0.0
5	14357002	48.930	-0.690	223.0	2016-01-01	NaN	NaN	0.0
6	14366002	49.170	0.230	62.0	2016-01-01	NaN	NaN	0.0
7	14372001	49.102	-0.765	184.0	2016-01-01	230.0	4.1	0.0
8	14501002	48.890	-0.390	185.0	2016-01-01	NaN	NaN	0.0
9	14515001	49.350	-0.770	68.0	2016-01-01	220.0	5.1	NaN
10	14577003	49.280	-0.560	15.0	2016-01-01	NaN	NaN	0.0
11	14578001	49.360	0.170	143.0	2016-01-01	230.0	2.5	0.0
12	14624001	48.990	-0.010	52.0	2016-01-01	NaN	NaN	0.0
13	14659001	49.060	-0.230	62.0	2016-01-01	NaN	NaN	0.0
14	14762004	48.850	-0.900	100.0	2016-01-01	170.0	1.0	0.0

	hu	td	t
0	91.0	278.45	279.85
1	99.0	278.35	278.45
2	92.0	276.45	277.65
3	95.0	278.25	278.95
4	NaN	NaN	278.35
5	NaN	NaN	277.65
6	NaN	NaN	279.55
7	92.0	278.05	279.25
8	NaN	NaN	278.35

```
9    NaN     NaN     NaN
10  99.0  279.65  279.75
11  96.0  277.45  278.05
12   NaN     NaN  278.05
13   NaN     NaN  279.15
14  91.0  276.55  277.85
```

2.2 Visualisation des données pour une seule date

Pour visualiser les données on affiche la température en fonction de la latitude et la longitude.

[5]:
```
date =  '20160101'
d_sub = df[df['date'] == date]
d_sub.head(10)
```

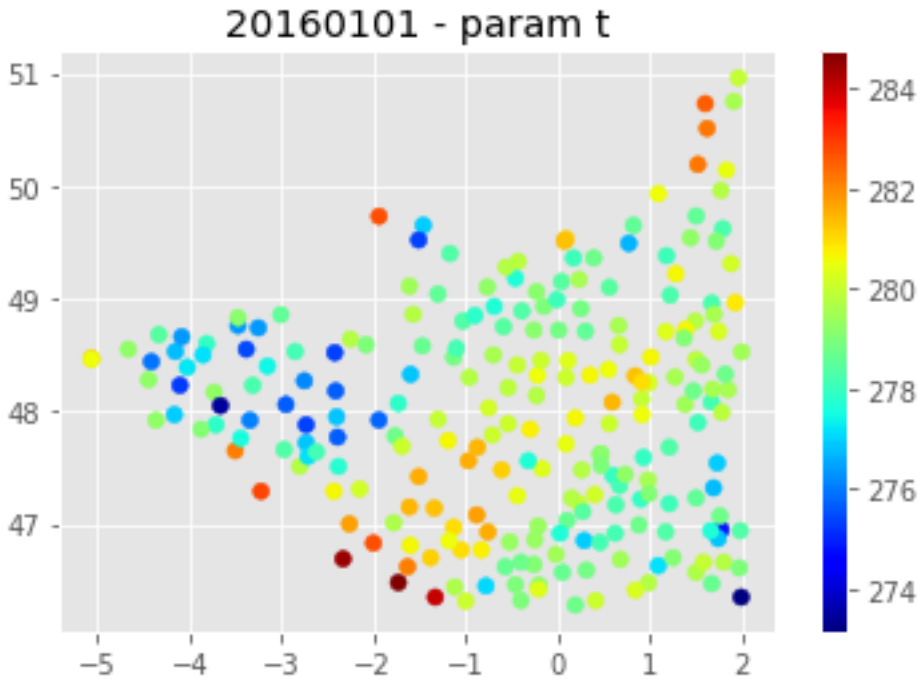
[5]:
```
   number_sta     lat    lon  height_sta       date     dd   ff  precip    hu  \
0    14066001  49.330 -0.430         2.0 2016-01-01  210.0  4.4     0.0  91.0
1    14126001  49.150  0.040       125.0 2016-01-01    NaN  NaN     0.0  99.0
2    14137001  49.180 -0.460        67.0 2016-01-01  220.0  0.6     0.0  92.0
3    14216001  48.930 -0.150       155.0 2016-01-01  220.0  1.9     0.0  95.0
4    14296001  48.800 -1.030       339.0 2016-01-01    NaN  NaN     0.0   NaN
5    14357002  48.930 -0.690       223.0 2016-01-01    NaN  NaN     0.0   NaN
6    14366002  49.170  0.230        62.0 2016-01-01    NaN  NaN     0.0   NaN
7    14372001  49.102 -0.765       184.0 2016-01-01  230.0  4.1     0.0  92.0
8    14501002  48.890 -0.390       185.0 2016-01-01    NaN  NaN     0.0   NaN
9    14515001  49.350 -0.770        68.0 2016-01-01  220.0  5.1     NaN   NaN

       td       t
0  278.45  279.85
1  278.35  278.45
2  276.45  277.65
3  278.25  278.95
4     NaN  278.35
5     NaN  277.65
6     NaN  279.55
7  278.05  279.25
8     NaN  278.35
9     NaN     NaN
```

[6]:
```
d_sub.iloc[1]
```

[6]:
```
number_sta               14126001
lat                         49.15
lon                          0.04
height_sta                  125.0
date          2016-01-01 00:00:00
```

```
dd                            NaN
ff                            NaN
precip                        0.0
hu                           99.0
td                         278.35
t                          278.45
Name: 1, dtype: object
```

[7]:
```
param = 't'
plt.scatter(d_sub['lon'], d_sub['lat'], c=d_sub[param], cmap='jet')
plt.colorbar()
plt.title(date+' - param '+param)

plt.show()
```



On voit donc que la température le premier janvier à minuit, dans le quart nord-est de la France est autour de 7 °C.

Pour réalisation du projet, nous allons tout d'abord nous concentré sur une seule station.

3 Etude des données pour une seule station

Nous allons tout d'abord étudier statisquement l'évolution des conditions météorologique dans une seule station (la première de la base de données) tous les jours à midi.

Création d'un dataframe avec une seule station et des données tous les jours à midi.

[8]:
```
number_station = 14066001
hour = "12:00:00"
date0 = dt.datetime.fromisoformat('2016-01-01T'+hour)
data0 = df[df['date'] == date0]
delta = timedelta(days=1)
time_range = [date0+i*delta for i in range(365)]
df1 = df.set_index('date').loc[time_range].reset_index(inplace=False)
print(df1.shape)
df1.head()
```

```
(90735, 11)
```

[8]:

	date	number_sta	lat	lon	height_sta	dd	ff	\
0	2016-01-01 12:00:00	14066001	49.33	-0.43	2.0	140.0	3.3	
1	2016-01-01 12:00:00	14126001	49.15	0.04	125.0	NaN	NaN	
2	2016-01-01 12:00:00	14137001	49.18	-0.46	67.0	140.0	5.1	
3	2016-01-01 12:00:00	14216001	48.93	-0.15	155.0	150.0	3.8	
4	2016-01-01 12:00:00	14296001	48.80	-1.03	339.0	NaN	NaN	

	precip	hu	td	t
0	0.0	92.0	280.65	281.85
1	0.0	99.0	281.95	282.05
2	0.0	94.0	280.85	281.75
3	0.0	91.0	279.65	281.05
4	0.0	NaN	NaN	279.85

Sélection d'un paramètre pertinent

[9]:
```
params = ["date", "precip", "hu", "td", "t","ff"]

df_week = df1[df1["number_sta"] == number_station]
df_week = df_week[params].dropna()
print(df_week.shape)
df_week.reset_index(inplace=True)
df_week.drop(columns=["index"], inplace=True)
df_week.head()
```

```
(355, 6)
```

[9]:

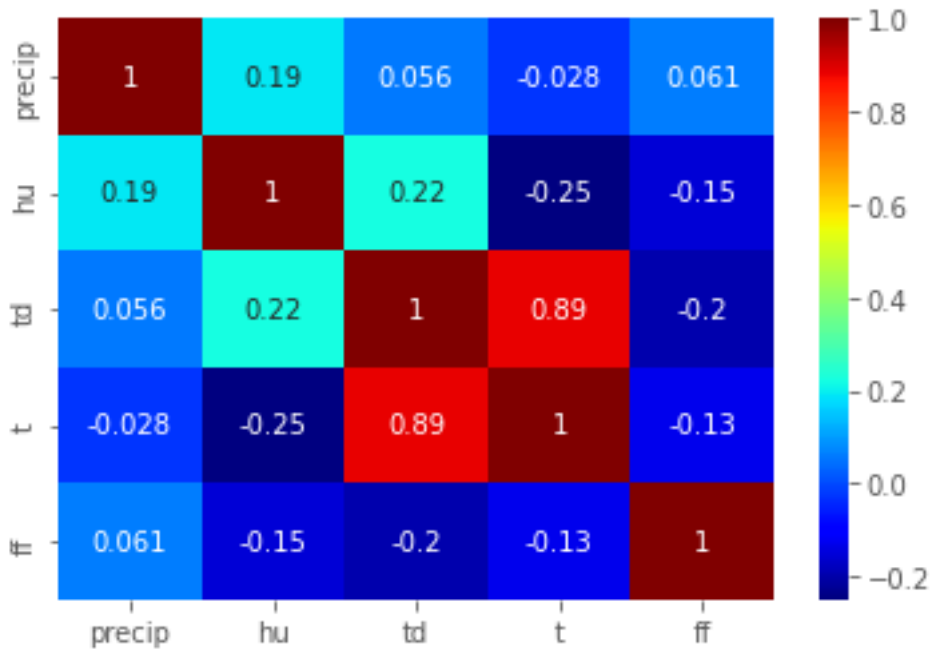
	date	precip	hu	td	t	ff
0	2016-01-01 12:00:00	0.0	92.0	280.65	281.85	3.3
1	2016-01-02 12:00:00	0.0	88.0	280.95	282.85	11.8
2	2016-01-03 12:00:00	0.2	84.0	280.15	282.75	9.2
3	2016-01-04 12:00:00	0.0	72.0	278.55	283.35	9.2
4	2016-01-05 12:00:00	0.0	77.0	279.15	282.95	8.4

[10]:
```
sb.pairplot(df_week)
```

[10]: <seaborn.axisgrid.PairGrid at 0x1249e5ca0>

[11]:
```
sb.heatmap(df_week.corr(),cmap='jet',annot=True)
```

[11]: <AxesSubplot:>



3.1 Analyse

On obtient premièrement une analyse statistique graphique (avec pairplot) qui lie chacun des paramètres entre eux.

On peut voir que la température t et la température de condensation td sont très liée, car les nuages de points ont dans les dexu cas l'allure d'une droite.

Aussi, on remarque tous les paramètres sauf la précipitation et la vitesse du vent sont réparties sous forme de gaussienne.

Enfin on affiche la matrice contenant les coefficients de corrélation entre chacun des paramètres. Comme dit plus tôt, on voit un lien très fort entre `t` et `td`.

Il existe également des corrélations notables entre `t` et `hu`, `td` et `hu`, et enfin `hu` et `precip`

3.2 2. Implémentation du modèle

3.2.1 2.1 Traitement des données

On présente ci-dessous une fonction qui prépare les données :

Elle prend en entrée la dataframe brute et : * Selectionne les données pour l'entrainement (1 semaine) * Selectionne les données pour la prédiction (24h apres) * Selectionne les `n_stations` les plus proches de Paris

Retourne un dataframe pour les données d'entrainement et un dataframe pour la prédiction

[12]:
```
N_STA = 6  # Paramètre concernant le nombre de station utilisées par la suite
LAT_PARIS = 48.8
LON_PARIS = 2.35
```

[13]:
```
def data_prep(df,t0=None, t1=None, n_stations=3, verbose=False):
    """
    Prepare les données pour l'analyse

    args
    - t0 : temps initial des données d'apprentissage
    - t1 : temps final des données d'apprentissage
        -> t2 = t1 + 24h => on cherche a prédire entre t1 et t2
    - n_stations : nombre de stations plus proche de Paris
    """
    # Args
    if t0 is None :
        t0 = datetime.fromisoformat('2016-09-10T00:00:00')
    if t1 is None:
        t1 = datetime.fromisoformat('2016-09-18T00:00:00')

    # Select stations in df_temp
    lat_Paris = LAT_PARIS
    long_Paris = LON_PARIS

    all_stations = df[['number_sta','lat','lon']].
↪drop_duplicates(subset=['number_sta']).reset_index(drop=True)
    all_stations['dist'] = (df["lat"].sub(lat_Paris).pow(2) + df["lon"].
↪sub(long_Paris).pow(2)).pow(0.5)

    assert n_stations < all_stations.shape[0], "Le nombre de stations␣
↪sélectionnées doit etre inferieur au nombre de station total"
    id_stations = all_stations.nsmallest(n_stations,'dist')['number_sta']
    dist_sta = all_stations.nsmallest(n_stations,'dist')
    if verbose:
        print(list(id_stations))

    df_temp = df[df['number_sta'].isin(id_stations)]

    # Split entrainement prédiction
    # Select times
    t2 = t1 + timedelta(days=1)

    mask1 = (df['date'] >= t0) & (df['date'] < t1)
    mask2 = (df['date'] >= t1) & (df['date'] < t2)
```

```
    df_train = df_temp.loc[mask1]
    df_test = df_temp.loc[mask2]

    return df_train, df_test, id_stations, dist_sta
```

[14]:
```
# Data prepation
df_sta, df_pred, id_stations, dist_sta =␣
 ↪data_prep(df,verbose=True,n_stations=N_STA)
```

```
[91200002, 78140001, 78562001, 78465001, 60322001, 78354001]
```

On présente ensuite une fonction qui va couper les dataframes en un dictionnaire de dataframes par station:

Ce dictionnaire est de forme `{id_station : dataframe_de_cette_station}`

[15]:
```
def split_df_by_station(df):
    """
    renvoie une dataframe pour chaque station

    return : tuple of array-like of dataframes
    """
    dfs = {}
    stations = list(df['number_sta'].drop_duplicates())
    #print(stations)

    for id_sta in stations:
        df_sta = df[df["number_sta"] == id_sta]
        df_sta = df_sta.reset_index(drop=True)
        dfs[id_sta] = df_sta
    return dfs
```

[16]:
```
# Splitting :
dfs_train = split_df_by_station(df_sta)
dfs_test  = split_df_by_station(df_pred)
```

On a ensuite une fonction qui permet de nettoyer les dataframes : * Drop la colonne `number_sta` * Dates -> int * Drop des colonnes qui contiennent plus de `dropping_limit` de valeurs `NaN` -> pose problème dans certains cas

[17]:
```
def clean_df(df,dropping_limit=0.5):
    number_sta = str(df.loc[0,"number_sta"])

    # Drop number sta
    df = df.drop(columns=['number_sta'])

    # converting date to timestamp
    #df['date'] = df['date'].astype(int)
    df['date'] = df['date'].apply(lambda x:x.value)
```

```
    #calcul du nombre de nan values
    df_null = df.isnull().sum()/df.shape[0] # dataframe des pourcentges de␣
↪valeurs nulles

    columns_to_drop = [id_ for id_ in df_null.index if df_null[id_] >=␣
↪dropping_limit]

    # suppr les colonnes ou les nan sont trop nombreuses
    assert 't' not in columns_to_drop
    df = df.drop(columns=columns_to_drop)

    df = df.dropna()

    return df.reset_index(drop=True)

def clean(dfs):
    d = {}
    for index,df in dfs.items():
        d[index] = clean_df(df)
    return d
```

[18]:
```
# Cleaning
dfs_train = clean(dfs_train)
dfs_test = clean(dfs_test)
```

[19]:
```
dfs_train[id_stations.iloc[0]]
```

[19]:

	lat	lon	height_sta	date	dd	ff	precip	\
0	48.527	1.995	116.0	1473465600000000000	0.0	0.0	0.0	
1	48.527	1.995	116.0	1473465960000000000	260.0	0.5	0.0	
2	48.527	1.995	116.0	1473466320000000000	260.0	0.8	0.0	
3	48.527	1.995	116.0	1473466680000000000	260.0	0.9	0.0	
4	48.527	1.995	116.0	1473467040000000000	270.0	1.0	0.0	
...	...	...	...	...	...	...	...	
1915	48.527	1.995	116.0	1474155000000000000	120.0	0.6	0.0	
1916	48.527	1.995	116.0	1474155360000000000	110.0	1.0	0.0	
1917	48.527	1.995	116.0	1474155720000000000	110.0	1.2	0.0	
1918	48.527	1.995	116.0	1474156080000000000	100.0	1.1	0.0	
1919	48.527	1.995	116.0	1474156440000000000	110.0	1.1	0.0	

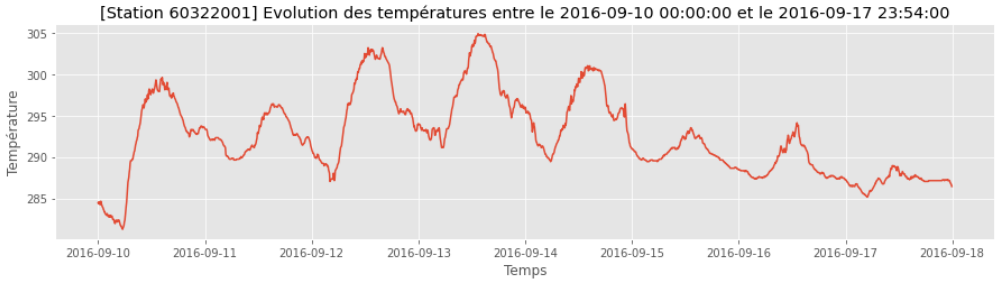
	hu	td	t
0	93.0	282.95	284.05
1	94.0	283.25	284.15
2	94.0	283.25	284.15
3	95.0	283.35	284.15
4	94.0	283.05	283.95

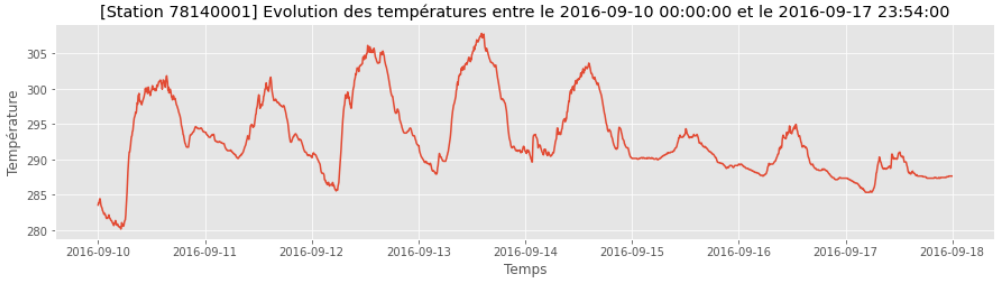
```
...    ...     ...     ...
1915  96.0  286.35  286.95
1916  97.0  286.45  286.95
1917  97.0  286.45  286.95
1918  97.0  286.45  286.95
1919  97.0  286.45  286.95

[1920 rows x 10 columns]
```

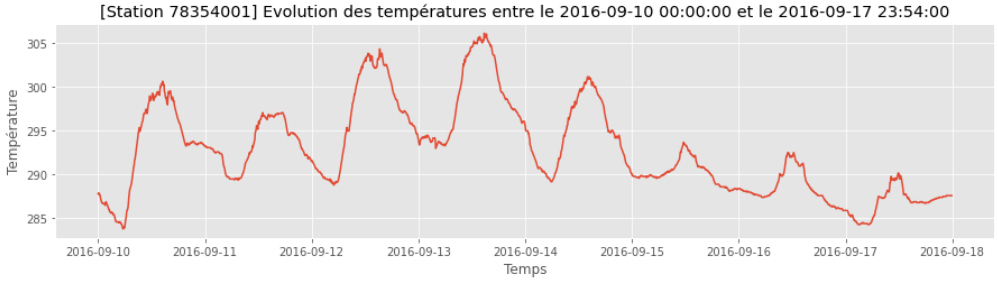
Ensuite on implemente deux fonction pour afficher les températures en fonction des stations sélectionnées :

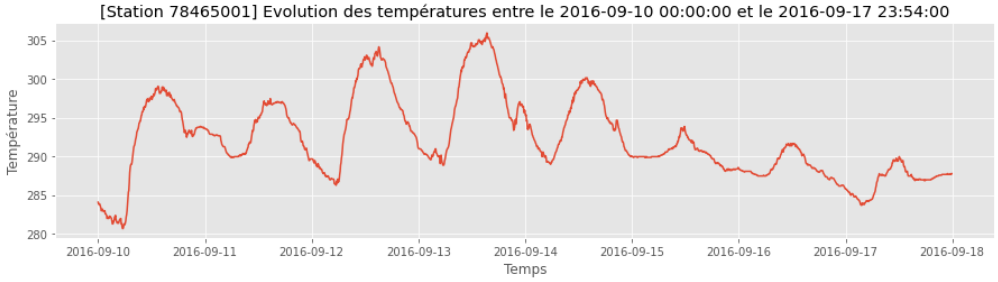
[20]:
```python
def display_temp(dfs):
    n = len(dfs)
    fig,ax = plt.subplots(n,1,figsize = (15,n*5))
    fig.subplots_adjust(hspace=0.5)
    i = 0
    for index,df in dfs.items():
        dates = pd.to_datetime(df['date'])
        ax[i].plot(dates,df['t'])
        ax[i].set_title(f'[Station {index}] Evolution des températures entre le␣
 ↪{dates[0]} et le {dates.iloc[-1]}')
        ax[i].set_xlabel('Temps')
        ax[i].set_ylabel('Température')
        i+=1

def display_all_in_one(dfs):
    fig = plt.figure(figsize=(15,7))
    i = 0
    for index,df in dfs.items():
        dates = pd.to_datetime(df['date'])
        plt.plot(dates,df['t'],label=f'station{index}')
        i+=1
    plt.legend()
    plt.title(f'Evolution des températures entre le {dates[0]} et le {dates.
 ↪iloc[-1]} pour les {len(dfs)} stations')
    plt.xlabel('Temps')
    plt.ylabel('Température')
```

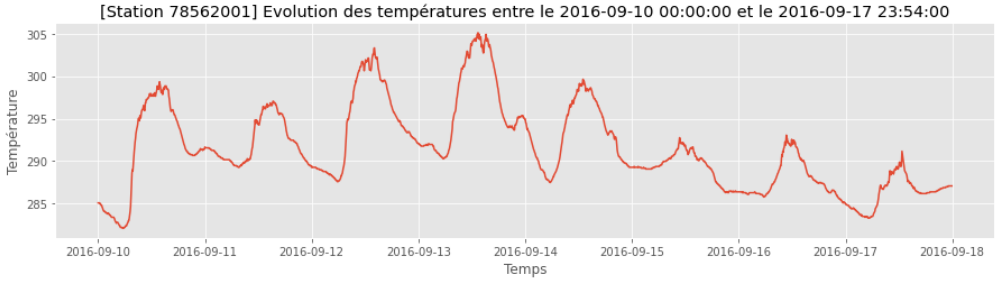
[21]:
```python
# Plotting
display_temp(dfs_train)
```

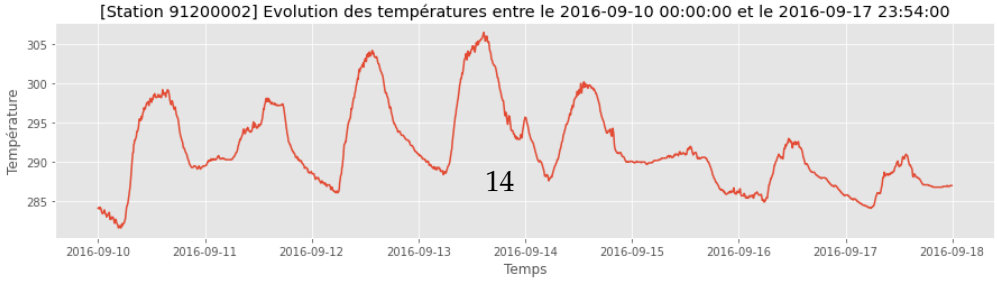

[Station 60322001] Evolution des températures entre le 2016-09-10 00:00:00 et le 2016-09-17 23:54:00
Température
Temps
305
300
295
290
285
2016-09-10
2016-09-11
2016-09-12
2016-09-13
2016-09-14
2016-09-15
2016-09-16
2016-09-17
2016-09-18


[Station 78140001] Evolution des températures entre le 2016-09-10 00:00:00 et le 2016-09-17 23:54:00
Température
Temps
305
300
295
290
285
280
2016-09-10
2016-09-11
2016-09-12
2016-09-13
2016-09-14
2016-09-15
2016-09-16
2016-09-17
2016-09-18

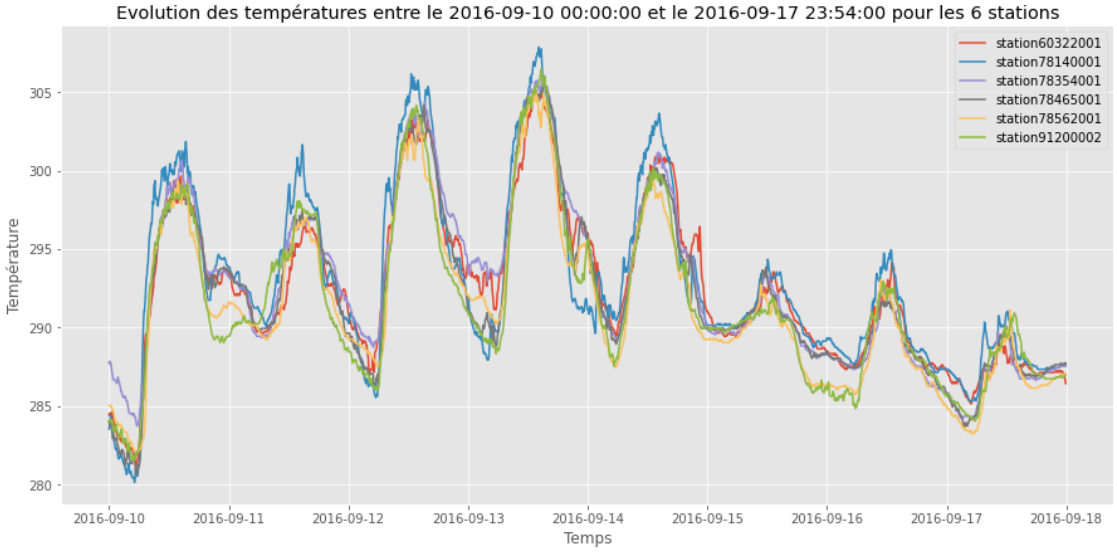

[Station 78354001] Evolution des températures entre le 2016-09-10 00:00:00 et le 2016-09-17 23:54:00
Température
Temps
305
300
295
290
285
2016-09-10
2016-09-11
2016-09-12
2016-09-13
2016-09-14
2016-09-15
2016-09-16
2016-09-17
2016-09-18


[Station 78465001] Evolution des températures entre le 2016-09-10 00:00:00 et le 2016-09-17 23:54:00
Température
Temps
305
300
295
290
285
280
2016-09-10
2016-09-11
2016-09-12
2016-09-13
2016-09-14
2016-09-15
2016-09-16
2016-09-17
2016-09-18


[Station 78562001] Evolution des températures entre le 2016-09-10 00:00:00 et le 2016-09-17 23:54:00
Température
Temps
305
300
295
290
285
2016-09-10
2016-09-11
2016-09-12
2016-09-13
2016-09-14
2016-09-15
2016-09-16
2016-09-17
2016-09-18


[Station 91200002] Evolution des températures entre le 2016-09-10 00:00:00 et le 2016-09-17 23:54:00
Température
Temps
305
300
295
290
285
2016-09-10
2016-09-11
2016-09-12
2016-09-13
2016-09-14
2016-09-15
2016-09-16
2016-09-17
2016-09-18

[22]:
```
display_all_in_one(dfs_train)
```



3.3 Preparation des données pour le modèle

On va séparer les données pour l'entrainement et le test. On va ensuite normaliser les données pour augmenter les performances du modèle

[23]:
```
from sklearn.neural_network import MLPClassifier, MLPRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

On va préparer les données en vue du modèle à l'aide de la fonction `prep`. Pour cela on va la séparer en deux dataframes qui représenteront X et y par la suite. On passe d'un dictionnaire de forme `{id_station : dataframe_station}` à `{id_station : (X_train, X_test, y_train, y_test, X_pred, y_pred)}`

- `X_train` : contient toutes les valeurs sauf la température pour des dates aléatoires (sur les 7 premiers jours) pour l'entrainement (proportion :`1-test_size`)
- `y_train`: contient les température associées aux dates de X_train (proportion :`1-test_size`)
- `X_test` : idem que X_train mais pour le test du modèle (proportion : `test_size`)
- `y_test` : idem que y_train mais pour le test du modèle (proportion : `test_size`)
- `X_pred` : contient toutes les valeurs sauf la température pour le huitième jour
- `y_pred`: contient les températures du huitième jour

La séparation en données d'entrainement et de test est réalisée par la fonction `sklearn.model_selection.train_test_split`

[24]:
```
def prep_to_fit(df, test_size):
    """
    Crée la matrice X et le vecteur y en fonction de la dataframe pour le modèle␣
 ↪(7 premiers jours)
    """
    X = df.loc[:,df.columns != 't']
    y = df['t']
    return list(train_test_split(X,y,test_size=test_size))

def prep_to_predict(df):
    """
    idem que prep_to_fit mais pour le prédict (8e jour)
    """
    return [df.drop(columns=['t']),df['t']]

def prep(dfs_train, dfs_test,test_size=0.25):
    """
    renvoie un array-like de tuple : (X_train, X_test, y_train, y_test, X_pred,␣
 ↪y_pred)
    """
    d = {}
    for index,df in dfs_train.items():
        d[index] = tuple(prep_to_fit(dfs_train[index], test_size) +␣
 ↪prep_to_predict(dfs_test[index]))
    return d
```

[25]:
```
prep_data = prep(dfs_train, dfs_test)
```

On va ensuite normaliser les données avec `sklearn.preprocessing.StandardScaler`. Cela permet d'augementer les performances du réseau de neurones.

On crée un `StandardScaler` par station

[26]:
```
def scaling_data(prep_data):
    """
    normalise les données X selon  la normalisation de X_train
    """
    scalers = {}
    scaled_data = {}

    for index,(X_train0, X_test0, y_train, y_test, X_pred0, y_pred) in prep_data.
 ↪items():
        scaler = StandardScaler()
        scaler.fit(X_train0)
        X_train = scaler.transform(X_train0)
        X_test = scaler.transform(X_test0)
        X_pred = scaler.transform(X_pred0)
```

```
        scalers[index] = scaler
        scaled_data[index] = (X_train, X_test, y_train, y_test, X_pred, y_pred )

    return scaled_data, scalers
```

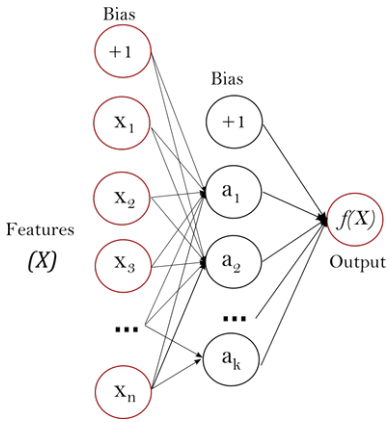
[27]:
```
scaled_data, scalers = scaling_data(prep_data)
```

3.4 Création des modèles

Nous allons essayer de prédire la température sur 24h pour chaque station. Il s'agit donc d'un problème de régression.

Pour ce problème de régression nous allons utiliser un `sklearn.neural_network.MLPRegressor`.

MLP siginfie Multi Layer Perceptron (voir image ci_dessous)



On va alors générer un modèle par station :

[28]:
```
def generate_models(scaled_data, verbose=False, compute_time=True):
    kwargs = {"hidden_layer_sizes":100,
              "activation":'logistic',
              "alpha":0.0001,
              "batch_size":'auto',
              "learning_rate":'constant',
              "learning_rate_init":0.01,
              "max_iter":1000,
              "verbose":verbose}
    models = {}

    time0 = time()
    for index,(X_train, X_test, y_train, y_test, X_pred, y_pred) in scaled_data.
 ↪items():
        models[index] = MLPRegressor(**kwargs).fit(X_train,y_train)
    if compute_time:
```

```
        print(f"Time to train {len(models)} models : {time()-time0:.3f}")
    return models
```

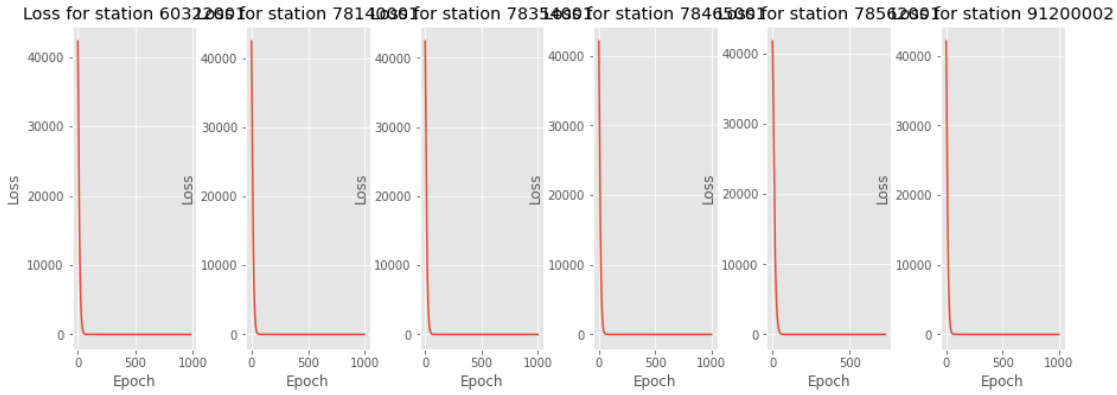
[29]:
```
models = generate_models(scaled_data, verbose=True, compute_time=True)
```

```
Iteration 1, loss = 42278.98865852
Iteration 2, loss = 40919.92050831
Iteration 3, loss = 39412.45284511
Iteration 4, loss = 37669.27326441
Iteration 5, loss = 35664.58753708
Iteration 6, loss = 33448.25969260
Iteration 7, loss = 31101.22609097
Iteration 8, loss = 28716.09222571
...

Time to train 6 models : 51.826
```

```
/usr/local/Cellar/jupyterlab/3.0.0_1/libexec/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (1000) reached and
the optimization hasn't converged yet.
  warnings.warn(
```

[30]:
```
def display_loss(models):
    n = len(models)
    fig,ax = plt.subplots(1,n,figsize = (15,5))
    fig.subplots_adjust(wspace=0.4)
    i=0
    for index,model in models.items():
        ax[i].plot([i+1 for i in range(len(model.loss_curve_))],model.
 ↪loss_curve_)
        ax[i].set_title(f'Loss for station {index}')
        ax[i].set_xlabel('Epoch')
        ax[i].set_ylabel('Loss')
        i+=1
```

[31]:
```
display_loss(models)
```



On ve ensuite créer une fonction qui score les modèles en fonction de `X_test` et `y_test`

[32]:
```
def score_models(models, scaled_data):
    scores = {}
    for i,regr in models.items():
        X_test,y_test = scaled_data[i][1], scaled_data[i][3]
        scores[i] = regr.score(X_test,y_test)
    return scores
```

[33]:
```
scores = score_models(models, scaled_data)
scores
```

[33]:
```
{60322001: 0.9894172272965374,
 78140001: 0.9903722676497436,
 78354001: 0.9944289807081473,
 78465001: 0.8333349522056572,
 78562001: 0.4001550445381262,
 91200002: 0.988189223574093}
```

3.5 Prédictions sur 24h

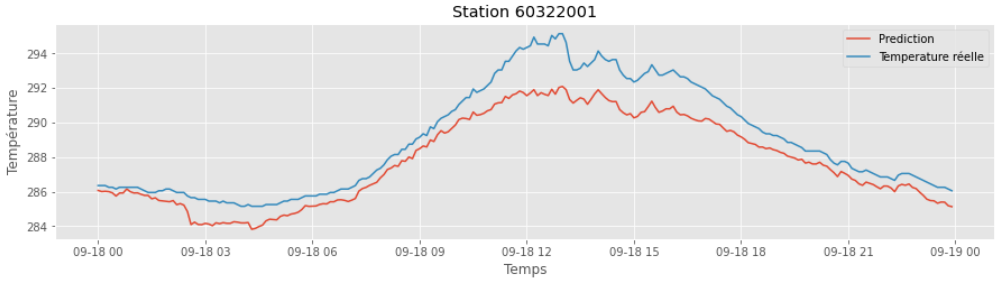
On va enfin pouvoir prédire la température pour chaque station via la fonction suivante :

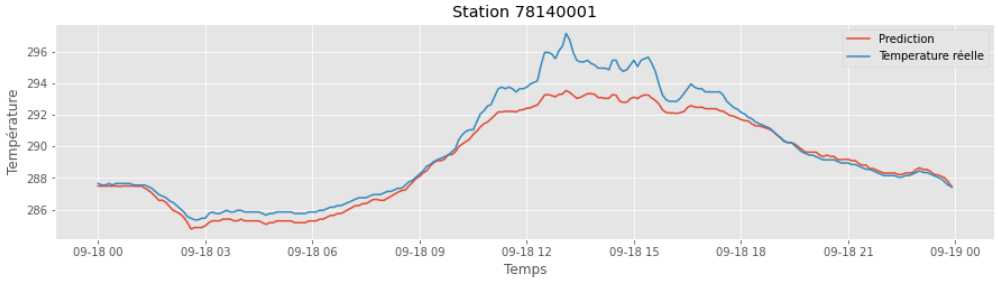
Cette fonction se charge aussi d'afficher les résultats de prédiction

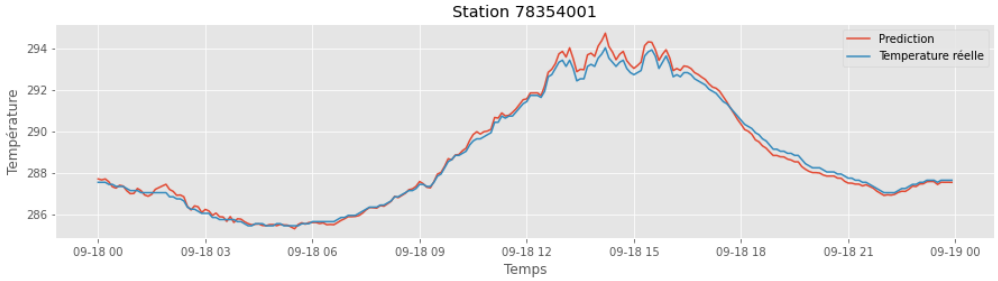
[34]:
```
def pred(models,scaled_data,display=True):
    l = {}
    for i,model in models.items():
        pred = model.predict(scaled_data[i][4])
        l[i] = pred

    if display:
        n = len(l)
```

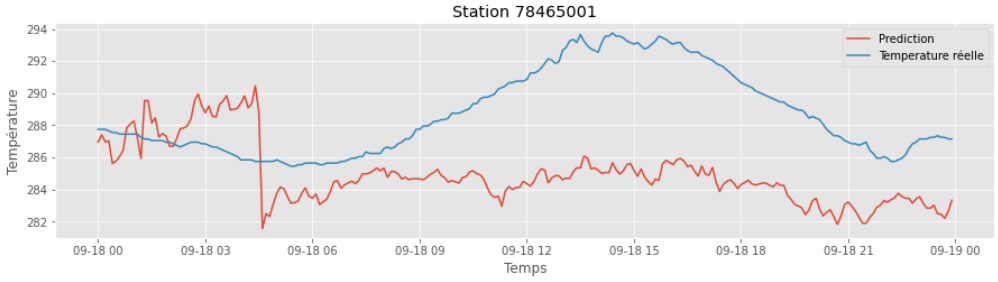
```
        dates = pd.to_datetime(prep_data[id_stations.iloc[0]][4]['date'])
        fig,ax = plt.subplots(n,1,figsize = (15,n*5))
        fig.subplots_adjust(hspace=0.5)
        i=0
        for index,pred in l.items():
            ax[i].plot(dates,pred,label='Prediction')
            ax[i].plot(dates,scaled_data[index][5],label='Temperature réelle')
            ax[i].set_title(f'Station {index}')
            ax[i].set_xlabel('Temps')
            ax[i].set_ylabel('Température')
            ax[i].legend()
            i+=1

    return l
```

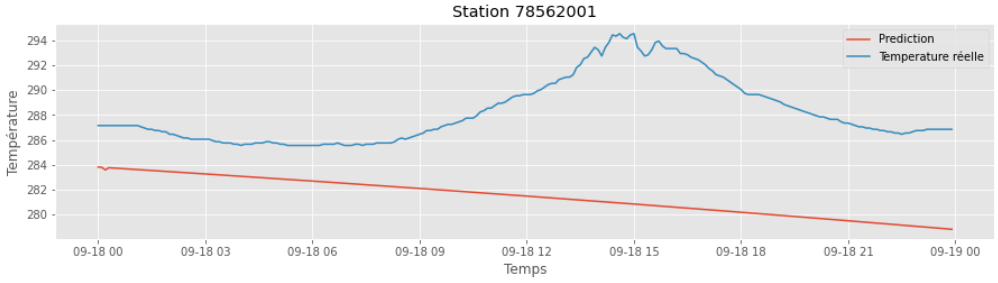
[35]:
```
preds24h = pred(models,scaled_data)
```

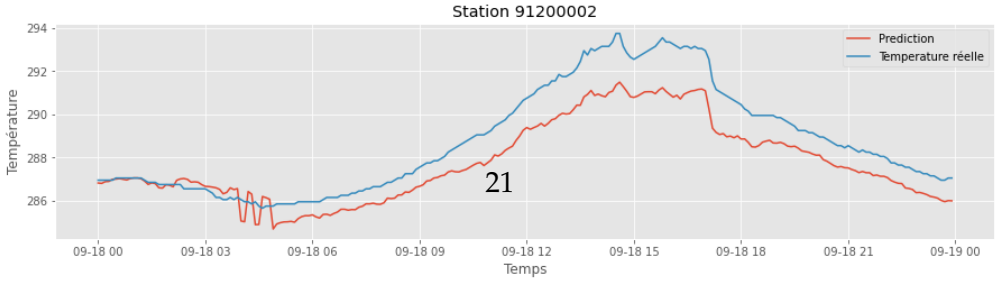

Station 60322001
Prediction
Temperature réelle
Température
Temps
09-18 00
09-18 03
09-18 06
09-18 09
09-18 12
09-18 15
09-18 18
09-18 21
09-19 00


Station 78140001
Prediction
Temperature réelle
Température
Temps
09-18 00
09-18 03
09-18 06
09-18 09
09-18 12
09-18 15
09-18 18
09-18 21
09-19 00


Station 78354001
Prediction
Temperature réelle
Température
Temps
09-18 00
09-18 03
09-18 06
09-18 09
09-18 12
09-18 15
09-18 18
09-18 21
09-19 00


Station 78465001
Prediction
Temperature réelle
Température
Temps
09-18 00
09-18 03
09-18 06
09-18 09
09-18 12
09-18 15
09-18 18
09-18 21
09-19 00


Station 78562001
Prediction
Temperature réelle
Température
Temps
09-18 00
09-18 03
09-18 06
09-18 09
09-18 12
09-18 15
09-18 18
09-18 21
09-19 00


Station 91200002
Prediction
Temperature réelle
Température
Temps
09-18 00
09-18 03
09-18 06
09-18 09
09-18 12
09-18 15
09-18 18
09-18 21
09-19 00

On remarque que pour certaines stations la prédiction se fait très bien alors que pour d'autre cela est bien moins bon. Cela est dû au fait que les stations où l'apprentissage se fait mal étaient constituées de beaucoup de `NaN`. Si on affiche la dataframe de la station 1 et la station 2 on peut remarquer ce phénoméne :

[36]:
```
dfs_train[id_stations.iloc[0]]
```

[36]:

	lat	lon	height_sta	date	dd	ff	precip	hu	td	t
0	48.527	1.995	116.0	1473465600000000000	0.0	0.0	0.0	93.0	282.95	284.05
1	48.527	1.995	116.0	1473465960000000000	260.0	0.5	0.0	94.0	283.25	284.15
2	48.527	1.995	116.0	1473466320000000000	260.0	0.8	0.0	94.0	283.25	284.15
3	48.527	1.995	116.0	1473466680000000000	260.0	0.9	0.0	95.0	283.35	284.15
4	48.527	1.995	116.0	1473467040000000000	270.0	1.0	0.0	94.0	283.05	283.95
...	...	...	...	...	...	...	...	...	...	...
1915	48.527	1.995	116.0	1474155000000000000	120.0	0.6	0.0	96.0	286.35	286.95
1916	48.527	1.995	116.0	1474155360000000000	110.0	1.0	0.0	97.0	286.45	286.95
1917	48.527	1.995	116.0	1474155720000000000	110.0	1.2	0.0	97.0	286.45	286.95
1918	48.527	1.995	116.0	1474156080000000000	100.0	1.1	0.0	97.0	286.45	286.95
1919	48.527	1.995	116.0	1474156440000000000	110.0	1.1	0.0	97.0	286.45	286.95

[1920 rows x 10 columns]

On a donc une fonction qui nous permet de créer un modèle de `sklearn.neural_network.MLPRegressor` qui permet de prédire la température sur 24h dans des configurations où les données le permettent. On peut alors passer à l'interpolation de ces températures pour prédire la température à Paris

---

3.6 3. Interpolation des données

Nous essaierons dans cette partie de déterminer la température à Paris en fonction des prévisions faites précédemment

[37]:
```
dist_sta
```

[37]:

	number_sta	lat	lon	dist
248	91200002	48.527	1.995	0.447833
203	78140001	48.964	1.925	0.455545
206	78562001	48.707	1.746	0.611118
205	78465001	48.860	1.694	0.658738
169	60322001	49.310	1.880	0.693542
204	78354001	48.960	1.670	0.698570

On va ajouter à la dataframe précédente une colonne avec le score du modèle associé à la station et indexer la dataframe par le `number_sta`

[38]:
```
def get_infos_stations():
    df_temp = dist_sta.set_index('number_sta')
    score_column = pd.DataFrame(scores,index=['score']).transpose()

    df_temp = pd.concat([df_temp,score_column],axis=1)
    return df_temp
```

La fonction `select_sta_from_score` (ci-dessous) selectionne les stations dont le modèle à donné un score superieur à `min_score`.

[39]:
```
def select_sta_from_score(min_score):
    infos_sta = get_infos_stations()
    return list(infos_sta[infos_sta['score']>min_score].index)
```

[40]:
```
select_sta_from_score(0.9)
```

[40]:
```
[60322001, 78140001, 78354001, 91200002]
```

3.7 Interpolation à Paris

On va donc conserver seulement les stations qui ont un score superieur à `MIN_SCORE`

[41]:
```
MIN_SCORE = 0.9
```

[42]:
```
infos_sta = get_infos_stations().loc[select_sta_from_score(0.9)]
infos_sta
```

[42]:

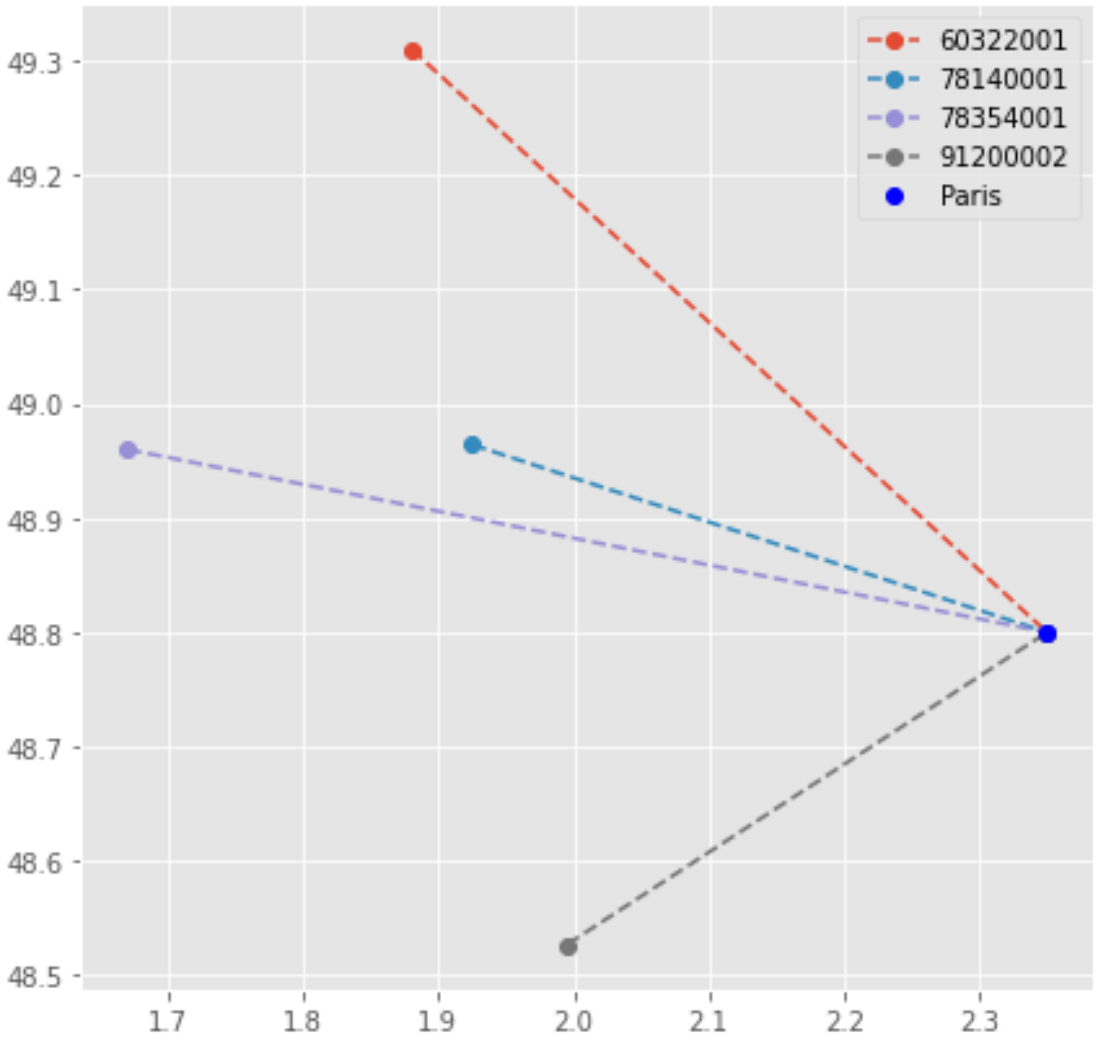
	lat	lon	dist	score
60322001	49.310	1.880	0.693542	0.989417

```
78140001  48.964  1.925  0.455545  0.990372
78354001  48.960  1.670  0.698570  0.994429
91200002  48.527  1.995  0.447833  0.988189
```

Pour représenter la positions des stations restantes par rapport à Paris on implémente la fonction `show_stations`. * En bleu : Paris * En couleur : La station concernée * En pointillée : la distance entre Paris et une station

[43]:
```
def show_stations(min_score):
    # Selection of stations :
    infos_sta = get_infos_stations().loc[select_sta_from_score(min_score)]

    plt.figure(figsize=(7,7))
    for station in infos_sta.index:
        plt.plot([infos_sta.loc[station,'lon'],LON_PARIS], [infos_sta.
↪loc[station,'lat'],LAT_PARIS], '--o', label=station)
    plt.plot([LON_PARIS],[LAT_PARIS], 'bo', label='Paris')
    plt.legend()
```

[44]:
```
show_stations(MIN_SCORE)
```



On définit la fonction `get_temps_in_Paris` qui interpole les résultat des modèles en fonctino de la distance qui les sépare à Paris

On prend ici la moyenne des températures des stations les plus proches de Paris pondérée par leur proximité

[45]:
```
def get_temps_in_Paris(output_model,min_score=0.9,verbose=True):
    """
    Interpolate model data at Paris with a mean from distance
    return : array-like of temperatures in Paris
    """

    # Selection of stations :
    infos_sta = get_infos_stations().loc[select_sta_from_score(min_score)]
    stations_left = list(infos_sta.index)
```

```
    if verbose:
        print(f"Now working with {len(stations_left)} stations : ")
        print(f"\t{stations_left}")
    if len(stations_left) == 0:
        return np.array()

    # Selection of outputs for selected stations
    output_from_selected_stations = {index : x for index, x in output_model.
↪items() if index in infos_sta.index}

    # Scaling distance with distance
    max_dist = infos_sta['dist'].max()
    infos_sta['dist'] = infos_sta['dist']/max_dist

    # Mean
    array_size = len(list(output_from_selected_stations.values())[0])
    mean = np.zeros(array_size)
    for station, scaled_dist in infos_sta['dist'].items():
        mean += scaled_dist * output_from_selected_stations[station]
    return mean/sum(infos_sta['dist'])
```

[46]:
```
temps_in_Paris = get_temps_in_Paris(preds24h, min_score=MIN_SCORE)
```

```
Now working with 4 stations :
        [60322001, 78140001, 78354001, 91200002]
```

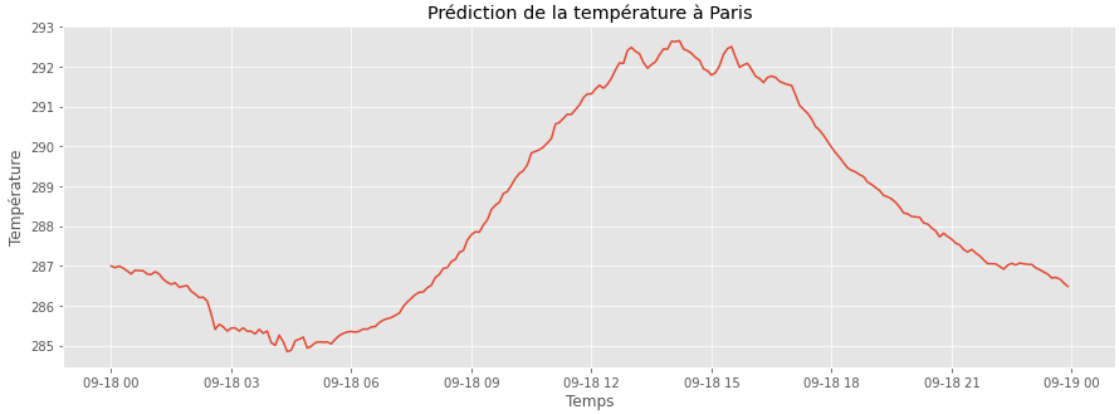
On implémente ensuite une fonction de visualisation : `display_temps` avec pour arguments : * `output_from_models` : les courbes de températuresen sortie des modèles * `min_score` : le score minimum des modèles que l'on considère comme acceptable (ici = `MIN_SCORE`) * `show_stations` : afficher les autres stations en pointillées

[52]:
```
def display_temps(output_from_models, min_score, show_stations=True):

    stations_left = list(get_infos_stations().
↪loc[select_sta_from_score(min_score)].index)

    plt.figure(figsize=(15, 5))
    plt.xlabel('Temps')
    plt.ylabel('Température')
    plt.title('Prédiction de la température à Paris')

    dates = pd.to_datetime(prep_data[id_stations.iloc[0]][4]['date'])
    plt.plot(dates, temps_in_Paris, label='Temperature à Paris')
    if not show_stations:
        return
    for station in stations_left:
```

```
        plt.plot(dates, output_from_models[station],'--',label=station)
    plt.legend()
```

Température seule à Paris

[53]:
```
display_temps(preds24h,MIN_SCORE,show_stations=False)
```



Température à Paris avec les stations selectionnées

[54]:
```
display_temps(preds24h,MIN_SCORE)
```