

# RAPPORT DE STAGE

LOUISE VANNETZEL

## REMERCIEMENTS

Je tiens tout d'abord à remercier M. Modeste, professeur de Statistiques, sans qui ce stage n'aurait pas été possible. Son aide et ses conseils m'ont permis d'acquérir de nouvelles compétences qui me seront utiles à l'avenir.

Je souhaite également adresser mes remerciements à Mme Greff, responsable de la licence de Mathématiques et aux équipes administratives qui ont accepté de m'accueillir au sein de l'Université de Pau et des Pays de l'Adour.

## CONTENTS

Remerciements	1
Introduction	1
1. Premier jeu de données	2
2. Nouveau jeu de données	10
3. Tests de normalité	11
4. Clustering	13
Conclusion	16
References	17

## INTRODUCTION

Actuellement en troisième année de licence de Mathématiques à l'Université de Pau et des Pays de l'Adour et ayant l'intention de poursuivre mes études en master de statistiques, je souhaitais avoir une première expérience dans ce domaine afin de me conforter dans mon choix d'orientation. C'est ainsi que j'ai pris contact avec mon professeur de statistiques, qui m'a proposé d'effectuer un stage d'une durée d'un mois sous sa supervision.

L'objectif principal de ce stage était de mettre en pratique les connaissances que j'ai acquises en cours, en les appliquant à un domaine concret : les prévisions météorologiques.

Plusieurs missions m'ont été confiées telles que l'élaboration de modèles prédictifs grâce à l'estimation paramétrique, la validation de ces modèles à l'aide de données ou encore le traitement de données à l'aide du clustering.

Ce rapport de stage présente un compte-rendu détaillé des activités et des tâches accomplies pendant cette période, ainsi qu'une analyse des résultats obtenus et des compétences acquises.

## 1. PREMIER JEU DE DONNÉES

L'objectif de cette partie est de construire un modèle de prédiction simple pour prédire la température. On a à notre disposition le relevé de température de plusieurs stations en France.

### 1.1. Estimation d'une station en fonction d'une autre.

Dans un premier temps intéressons-nous à la notion d'espérance conditionnelle. On admet le fait suivant : il existe une fonction  $h^*$  tel que pour toute fonction mesurable  $f$ , on a :

$$\mathbb{E}[(Y - h^*(X_1, \dots, X_d))^2] \leq (\mathbb{E}[(Y - f(X_1, \dots, X_d))^2])$$

c'est-à-dire la variable aléatoire  $h^*(X_1, \dots, X_d)$  est la variable construite à partir uniquement de  $X_1, \dots, X_d$  qui est le plus proche de la variable aléatoire  $Y$ . Cette variable aléatoire est appelée espérance conditionnelle de  $Y$  sachant  $X_1, \dots, X_d$  et est notée :  $\mathbb{E}[Y|X_1, \dots, X_d]$ . C'est la meilleure manière d'approcher  $Y$  lorsque l'on connaît uniquement  $X_1, \dots, X_d$ .

On a le résultat classique suivant : pour  $a \in \mathbb{R}$

$$(1.1) \quad \mathbb{E}[(Y - \mathbb{E}[Y])^2] \leq (\mathbb{E}[(Y - a)^2])$$

Pour deux variables aléatoires  $X$  et  $Y$  trouvons  $a, b$  minimisant la quantité :  $\mathbb{E}[(Y - (a + bX))^2]$

*Preuve*

$$\mathbb{E}[(Y - (a + bX))^2] = \mathbb{E}[((Y - bX) - a)^2]$$

Or, d'après le résultat (1.1) en remplaçant  $Y$  par  $\tilde{Y} = Y - bX$ , on obtient :

$$\mathbb{E}[(Y - (a + bX))^2] \geq \mathbb{E}[((Y - bX) - \mathbb{E}[(Y - bX)])^2]$$

$$\text{Donc } a = \mathbb{E}[\tilde{Y}] = \mathbb{E}[Y - bX]$$

D'autre part, on a :

$$[(Y - bX) - \mathbb{E}[Y - bX]]^2 = b^2[X^2 - 2X\mathbb{E}[X] + (\mathbb{E}[X])^2] - 2b[XY - Y\mathbb{E}[X] - X\mathbb{E}[Y] + \mathbb{E}[X]\mathbb{E}[Y]]$$

D'où :

$$\mathbb{E}[(Y - bX) - \mathbb{E}[Y - bX]]^2 = -2b\text{Cov}(X, Y) + b^2\text{Var}(X) + \text{Var}(Y)$$

Le minimum est :

$$\frac{-(-2\text{Cov}(X, Y))}{2\text{Var}(X)} = \frac{\text{Cov}(X, Y)}{\text{Var}(X)} = b$$

Après avoir nettoyé la base de données, pour ne considérer que les jours avec uniquement des données observées, on construit un algorithme permettant de prédire la température d'une station Y grâce à une autre station X.

Soient  $\mathbf{Y} \in \mathbb{R}^n$  la variable aléatoire correspondant aux températures de la ville cible, et  $\mathbf{X}$  celle qui correspond aux températures de la ville utilisée.

On effectue deux approximations :

- $\mathbf{Y} = a + b\mathbf{X}$
- $\mathbf{Y} = \mathbb{E}[\mathbf{X}]$

avec  $a = \mathbb{E}[Y - bX]$  et  $b = \frac{\text{Cov}(X,Y)}{\text{Var}(X)}$

Et on introduit les estimateurs suivants :

- $\bar{X}_d = \frac{1}{d} \sum_{i=1}^d X_i^p$
- $\bar{Y}_d = \frac{1}{d} \sum_{i=1}^d Y_i^p$
- $\bar{S}_d^2 = \frac{1}{d-1} \sum_{i=1}^d (X_i - \bar{X}_d)^2$
- $\frac{1}{d-1} \sum_{i=1}^d (X_i - \bar{X}_d) \times (Y_i - \bar{Y}_d)$

Puis on construit le programme ci-dessous :

```
def prediction1(s,t) : # calibration du modèle sur les 80 %
    mean_s = np.mean(temp[:1363,s]) #moyenne ville utilisée sur les 1364 premiers jours
    mean_t = np.mean(temp[:1363,t]) #moyenne ville cible
    cov = cov_emp(temp[:1363,s],temp[:1363,t])
    var_s = var_emp(temp[:1363,s])
    var_t = var_emp(temp[:1363,t])

    b = cov / var_s
    a = mean_t - b * mean_s
    return(a+b*temp[1363:,s]) # test sur les 20%

def prediction2(t) :
    moy_t = np.mean(temp[:1363,t]) #moyenne toulouse sur les 1364 premiers j
    y = np.ones(342)
    return moy_t * y
```

On crée la fonction prediction1 pour qu'elle corresponde à l'approximation 1 de notre Y, de même pour la fonction prediction2, puis on teste ce programme en estimant les températures de Toulouse grâce à celles de Carcassonne.

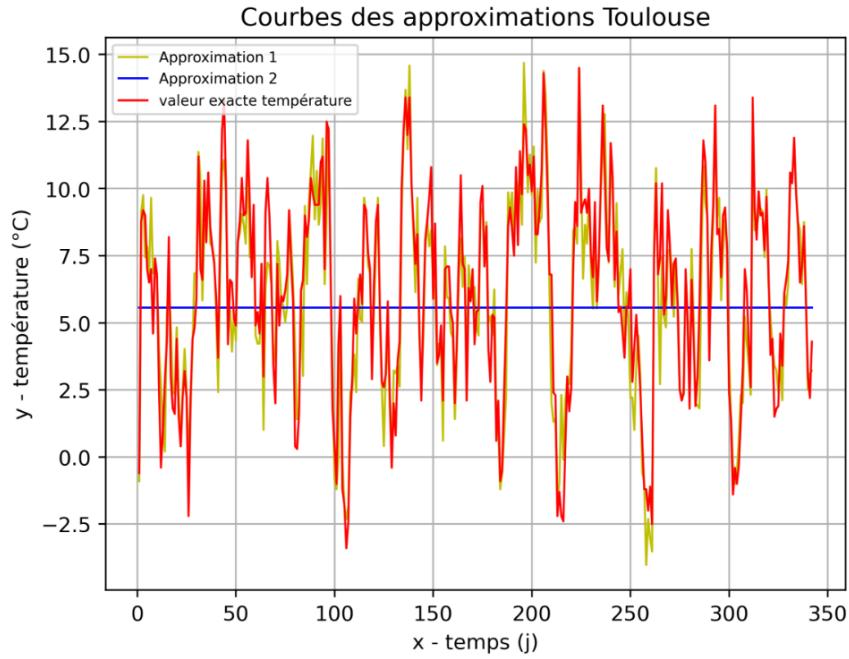


FIGURE 1. Estimation de la température de Toulouse grâce à la station de Carcassonne

On observe la différence entre la température prédite par notre modèle et celle de notre base de données. On remarque que l'approximation 1 est la plus proche de la réalité. On pouvait s'y attendre puisqu'elle évolue en fonction des jours, contrairement à l'approximation 2 qui est constante.

On cherche à vérifier l'hypothèse suivante : plus la station de la ville utilisée est loin de celle de la ville cible, plus l'erreur des approximations est grande. Pour cela, on teste ce programme en utilisant les températures de Biarritz, afin d'estimer celles du Cap de la Hève.

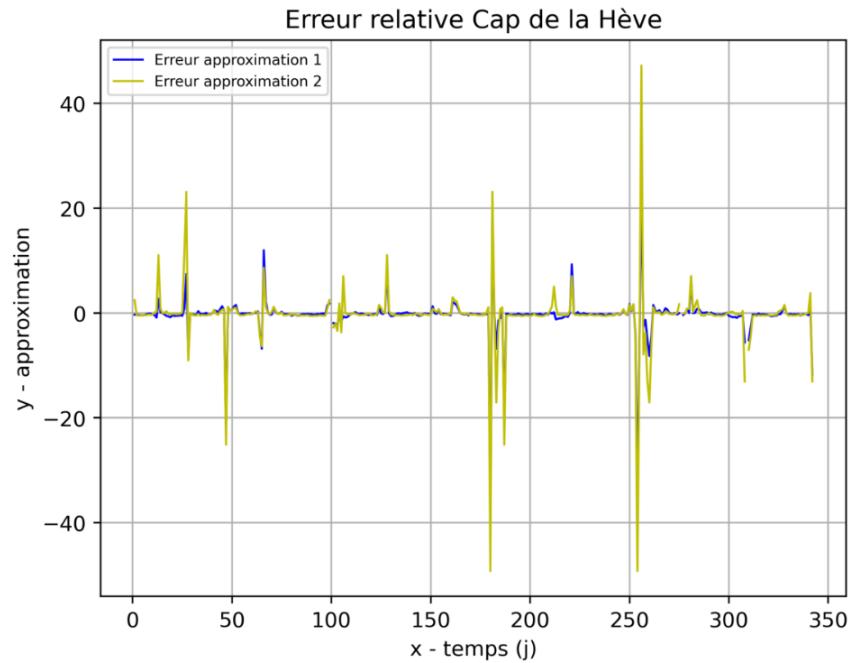


FIGURE 2. Estimation de la température de Cap de la Hève grâce à la station de Biarritz

A titre de comparaison, l'erreur relative que nous avions trouvée pour Toulouse est présentée ci-dessous.

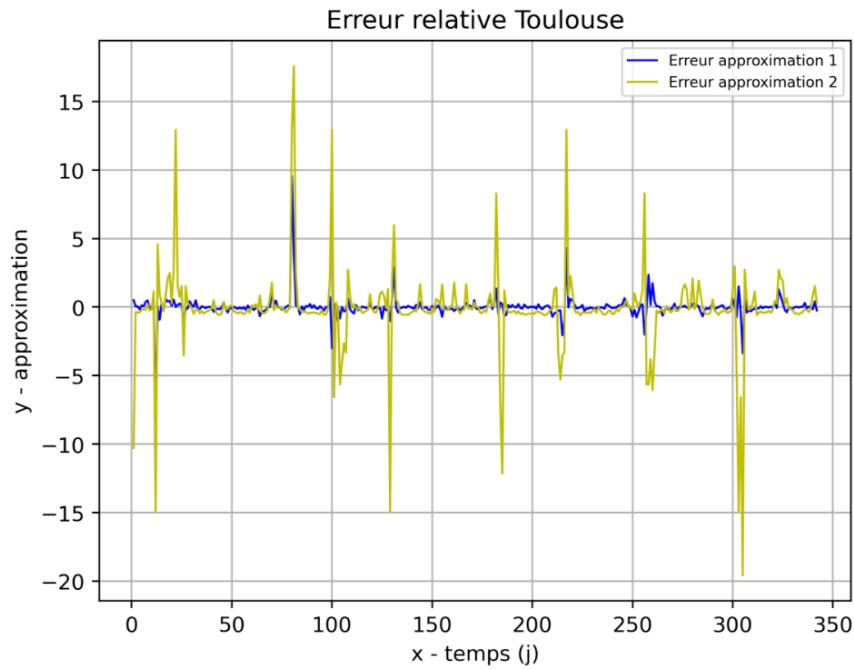


FIGURE 3

On constate qu'en effet, plus les stations sont éloignées plus il est difficile de prédire correctement les températures en ne prenant en compte qu'une seule station.

De plus, comme l'indique la *figure 4*, bien que les stations du Mont Aigoual et de Nîmes soient proches l'une de l'autre, l'estimation réalisée est moins précise que pour les autres villes.

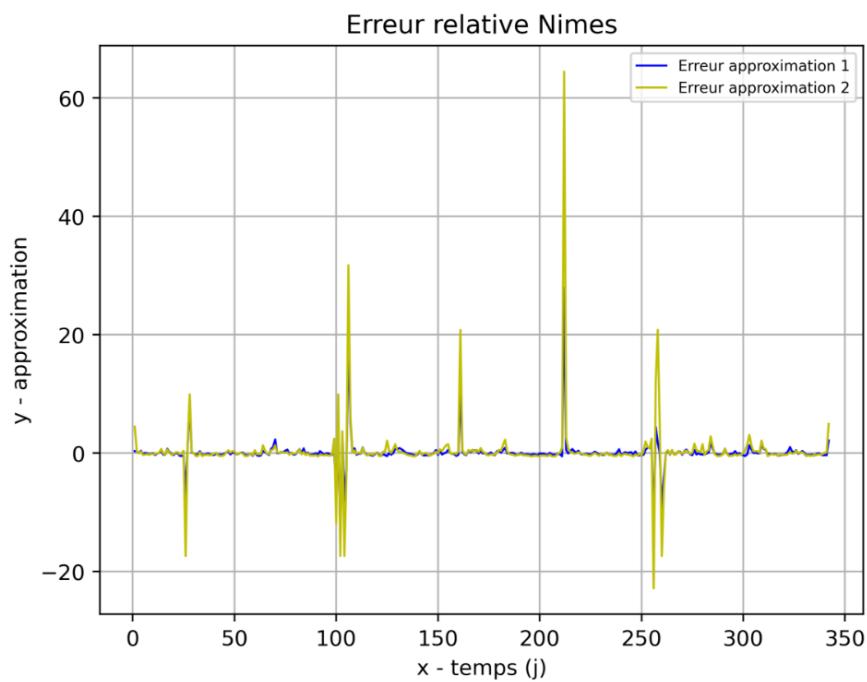


FIGURE 4. Estimation de la température de Nîmes grâce à la station du Mont Aigoual

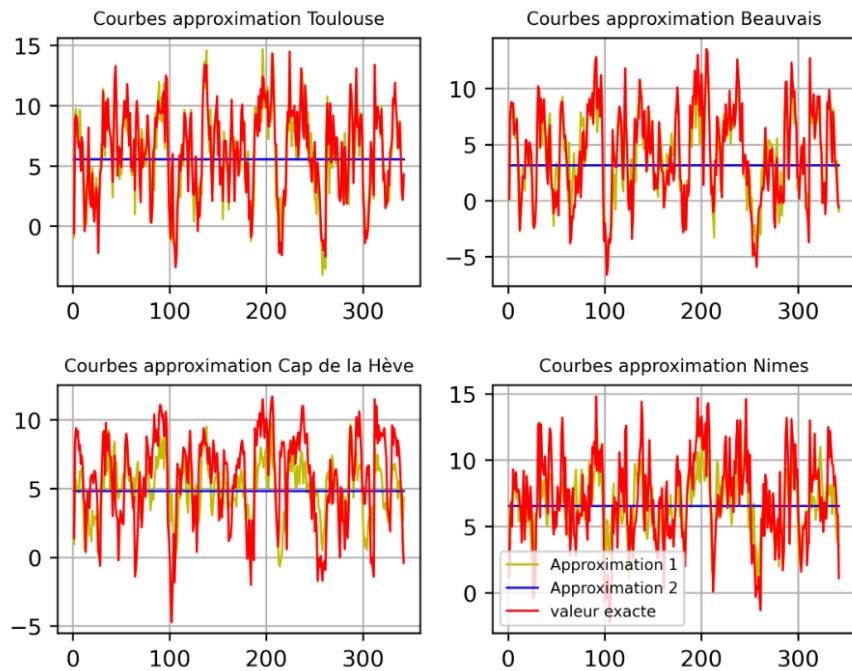


FIGURE 5

- (1) graphe 1 : estimation de Toulouse grâce à Carcassonne
- (2) graphe 2 : estimation de Beauvais grâce au Mans
- (3) graphe 3 : estimation du Cap de la Hève grâce à Biarritz
- (4) graphe 4 : estimation de Nîmes grâce au Mont Aigoual

## 1.2. Estimation d'une station grâce aux 11 autres.

Dans cette section, on construit un algorithme qui connaissant la température de 11 stations permet d'estimer la température sur la dernière station. On suppose que  $P^*$  est un vecteur gaussien qui régit notre phénomène.

On construit une approximation grâce à l'espérance conditionnelle. On a :  $\mathbf{Y} = \mathbb{E}[Y|X]$  avec  $X$  le vecteur correspondant aux températures des 11 stations. On modélise l'approximation 3 grâce à la formule suivante :

$$\mathbf{Y} = \mathbb{E}[Y] + C^t K^{-1}(X - \mathbb{E}[X]), \text{ avec } K = \text{Cov}[X_i, X_j]_{i,j} \text{ et } C = \text{Cov}[X_i, Y]_i$$

Puis on construit le programme ci-dessous :

```

from statistics import covariance , variance

def prediction3( t ) :
    M = np.mean(new_temp[:1363 ,] , axis=0)
    mean_X = np.delete(M, t)      # esperance X
    mean_t = M[t] # esperance Y

    # Initialisation des matrices
    K = np.zeros((11 , 11))
    C = np.zeros(12)
    cov_matrix = np.cov(np.transpose(new_temp[:1363 ,]))

    # Calcul des matrices K et C
    K = np.delete(cov_matrix ,t , axis=0)
    K = np.delete(K,t , axis=1)
    C = cov_matrix[t ,:]
    C = np.delete(C,[t ,t])

    # Calcul de l'inverse de la matrice K
    K_inv = np.linalg.inv(K)

    # On transpose C pour obtenir Ct
    Ct = C.transpose()

    # Calcul du coefficient a
    a = np.dot(Ct , K_inv) # a = Ct*(K-1)

    # Calcul de X
    X = np.delete(new_temp[1363: ,] ,t , axis = 1) # test sur les 20 %
    for i in range(11):
        X[:, i] = X[:, i] - mean_X[i]

    return mean_t + np.dot(X, a) # E(Y) + a*(X-E(X))

```

Comme précédemment, les courbes d'erreur obtenues après l'exécution du programme sont affichées. Cette fois, seules les approximations 1 et 3 sont présentées, car l'approximation 2, jugée moins précise, est exclue.

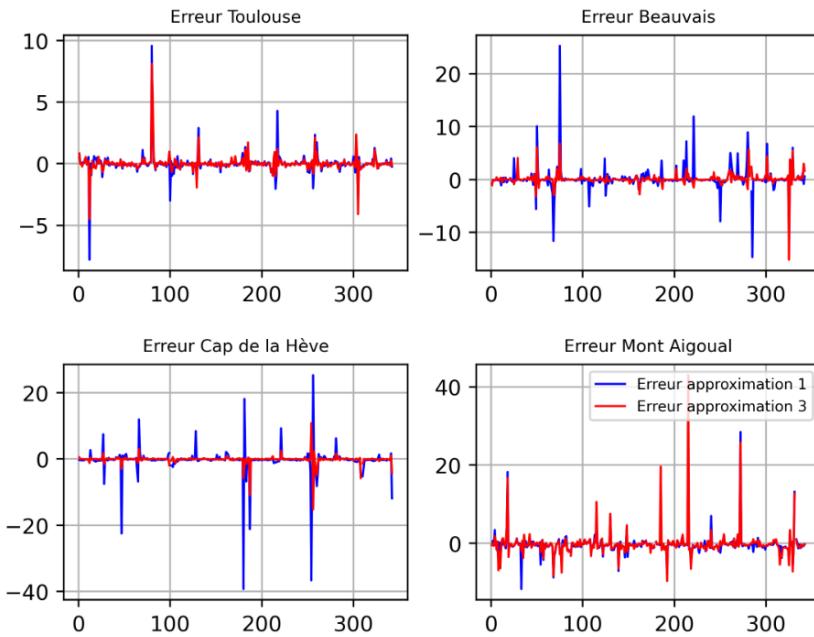


FIGURE 6

On remarque que l'approximation 3 est meilleure que la première. Ce constat n'est pas étonnant car l'approximation 3 prend en compte initialement un ensemble de données plus important.

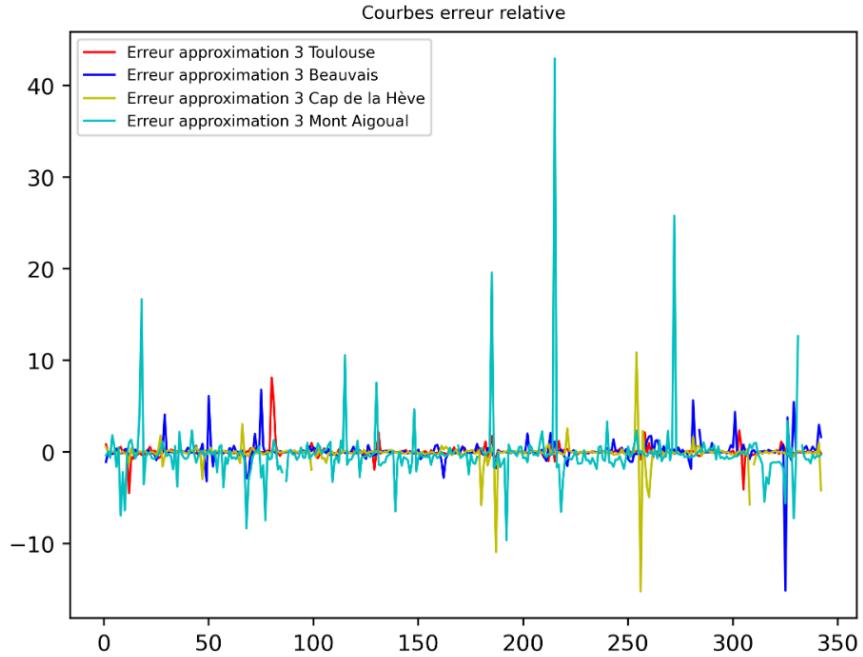


FIGURE 7

D'autre part, l'approximation 3 donne une erreur importante pour le Mont Aigoual *cf figure 7*, alors que celle-ci a été faite grâce aux températures de 11 autres stations. Ces stations ne se trouvant pas à la même altitude que le Mont Aigoual, on en déduit que l'erreur est due à l'altitude.

## 2. NOUVEAU JEU DE DONNÉES

On se munit à présent d'un nouveau jeu de données contenant la localisation, la température, le niveau de pluie et de vent de 104 stations sur une période de deux ans. On s'intéresse dans un premier temps au dataset contenant la localisation des 104 stations et on crée un programme permettant d'afficher ces stations sur une carte de la France *cf figure 8*.

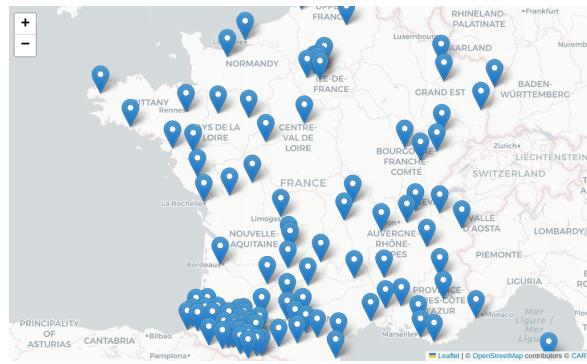


FIGURE 8

Puis on se sert des données de température collectées dans les stations afin de créer un nouveau modèle d'approximation. Étant donné que les températures ont été prises tout au long de l'année, il nous faut prendre en compte les changements de saisons dans notre estimation, c'est pourquoi on programme de manière dynamique.

De plus, la prédition est basée sur les données recueillies au cours des 30 derniers jours et utilise les températures des dix stations les plus proches de celle que l'on souhaite estimer.

Pour cela, on crée une fonction permettant de déterminer les dix villes les plus proches d'une ville donnée. Puis on se sert de cette fonction pour créer notre prédition 4, dont le code est similaire à la prédition 3. Et on teste ce programme en estimant la température pour la station d'Amberieu.

```

import math
def dist(i,j): # calcul distance euclidienne entre x et y
    x = [loc[i,1], loc[i,2]]
    y = [loc[j,1], loc[j,2]]
    return math.sqrt(sum((np.array(x)-np.array(y))**2))

def dist_from(i): # calcul des distances entre x et chaque ligne de data
    res = [0] * len(loc)
    for j in range(104):
        res[i] += dist(i,j)
    return res
    
```

```

    res[j] = dist(i, j)
    return res

def voisins(i):
    a = dist_from(i)
    b = np.argsort(a)
    c = np.argwhere(b<11).tolist()
    voisin = []
    for j in c:
        if j[0]!=i:
            voisin.append(j[0])
    return voisin

```

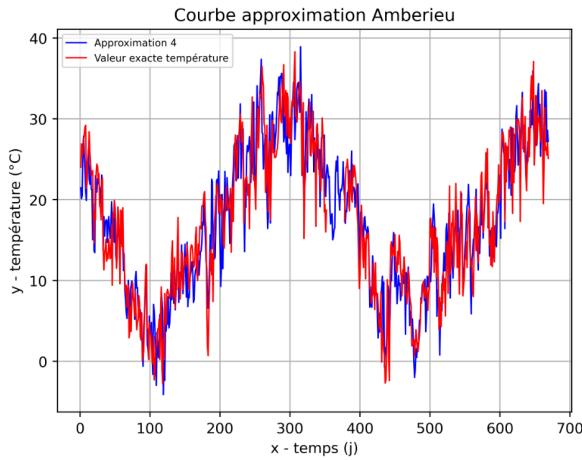


FIGURE 9

Comme on peut le voir *cf figure 9*, les courbes d'approximation de la prédiction 4 et de la température exacte suivent la même tendance.

### 3. TESTS DE NORMALITÉ

On se place dans le cas à un échantillon où  $X$  est une variable continue. On cherche à vérifier si les données sont normalement distribuées. Vu qu'on ne connaît pas les paramètres  $(\mu, \sigma^2)$ , on les estime directement grâce à la moyenne et la variance empiriques. On souhaite effectuer un test de Lilliefors.

De plus, on utilisera le premier jeu de données car les températures sont celles du mois de janvier, ainsi on supposera que les données suivent une loi gaussienne.

#### 3.1. Test de Kolmogorov-Smirnov.

Soient  $F_n$  et  $F$ , respectivement les fonctions de répartition observée et théorique de  $X$ . Notons  $n$  le nombre d'observations de  $X$ . La statistique du test de Kolmogorov-Smirnov est alors :

$$D = \max_{x \in [1, n]} |F_n(x) - F(x)| = \max_{x \in [1, n]} \left( |F(x) - \frac{i-1}{n}|, \left| \frac{i}{n} - F(x) \right| \right)$$

L'hypothèse nulle est :  $(H_0)$  : " $F_n = F$ ", c'est-à-dire X suit la loi que l'on a fixée, ici la loi normale. En notant  $D_{n,1-\alpha}$  la valeur seuil de la distribution de la statistique de test pour une confiance  $\alpha$  et pour le paramètre  $n$ , on a l'hypothèse alternative suivante :

$$(H_1) : "F_n \neq F" \text{ soit } D = \max_{x \in [1,n]} (|F(X_i) - \frac{i-1}{n}|, |\frac{i}{n} - F(X_i)|) > D_{n,1-\alpha}$$

### 3.2. Test de Lilliefors.

Après avoir estimé nos paramètres  $\mu$  et  $\sigma$  et calculé la statistique de test D de Kolmogorov-Smirnov, on ajuste éventuellement D en fonction de la taille de notre échantillon.

Si  $n \leq 100$  alors on conserve  $D_{obs} = D$  et on fixe  $\sigma = n$  ; Sinon, on lui applique l'ajustement suivant :  $D_{obs} = D(\frac{n}{100})^{0.49}$  et on fixe  $\sigma = 100$ .

Déterminons à présent la p-valeur. Pour cela on utilise la table du test de Lilliefors, ou on la calcule à l'aide de la formule suivante :

$$p = e^{-7.01256 \cdot D_{obs}^2 \cdot (\delta + 2.78019) + 2.99587 \cdot D_{obs} \cdot \sqrt{\delta + 2.78019} - 0.122119 + \frac{0.974598}{\sqrt{\delta}} + \frac{1.67997}{\delta}}$$

Interprétation : On se fixe un seuil  $\alpha \in ]0, 1[$  alors si  $p(x) < \alpha$ , alors on rejette et si  $p(x) > \alpha$ , alors on ne rejette pas.

Effectuons d'abord un test de Lilliefors sur 30 jours, suivi par un test sur 60 jours avec les nouvelles données, puis récapitulons nos résultats dans un dataset.

	Lilliefors	Statistic	p-value	Non Rejet
0		0.206261	0.002395	False
1		0.127883	0.236040	True
2		0.127145	0.241621	True
3		0.120407	0.324018	True
4		0.138965	0.152185	True
5		0.182791	0.012230	False
6		0.108810	0.476450	True
7		0.183229	0.011770	False
8		0.106219	0.512626	True
9		0.120030	0.328969	True
10		0.116252	0.378631	True
11		0.139555	0.147720	True
12		NaN	NaN	False
13		NaN	NaN	False
14		0.093005	0.721203	True
15		0.103699	0.552394	True
16		0.174490	0.020917	False
17		0.115647	0.386577	True
18		0.082896	0.855742	True
19		0.091157	0.750301	True
20		0.108678	0.478179	True
21		0.217485	0.001000	False
22		0.093745	0.709519	True
23		0.226304	0.214349	True

FIGURE 10. Test de Lilliefors sur 30j

	Lilliefors Statistic	p-value	Non Rejet
0	0.086239	0.374162	True
1	0.072829	0.636582	True
2	0.106828	0.109951	True
3	0.089320	0.319152	True
4	0.115372	0.059046	True
5	0.095162	0.229780	True
6	NaN	NaN	False
7	0.065022	0.793851	True
8	0.089935	0.308178	True
9	0.063098	0.827270	True
10	0.127275	0.023383	False
11	0.078683	0.780217	True

FIGURE 11. Test de Lilliefors sur 60j

On observe que l'hypothèse nulle est rarement rejetée. En effet, les seuls mois où  $H_0$  est rejetée après calcul de la valeur-p sont : 2021 – 08, 2022 – 01, 2022 – 03, 2022 – 12, 2023 – 05. Ainsi, l'hypothèse initiale selon laquelle  $P^*$  est un vecteur gaussien régissant notre phénomène n'est pas absurde.

#### 4. CLUSTERING

Le clustering, une branche spécifique du Machine Learning, permet d'organiser les données étudiées en groupes homogènes, en fonction de leur similarité. Nous allons nous intéresser dans cette partie à l'algorithme des k-means.

##### 4.1. Méthode des k-means.

Initialisation : On choisit au hasard k centroïdes (points du jeu de données) qui seront les centres des clusters de départ.

Boucle :

- On construit k clusters : chaque point est dans le cluster du centroïde le plus proche. (distance formule)
- On calcule les nouveaux centroïdes : pour chacun des clusters que l'on vient de former on calcule la moyenne arithmétique, de tous les points de données appartenant à un cluster, celle-ci devient le nouveau centroïde.
- On recommence jusqu'à ce qu'il y ait convergence : la convergence correspond au fait que les centroïdes ne change pas après une mise à jour.

La notion de distance intervient dans la méthode des k-means, en général on utilise la distance euclidienne définie de la manière suivante :

Soient deux groupes d'éléments  $p = (p_1, \dots, p_n)$  et  $q = (q_1, \dots, q_n)$  alors la distance d entre les points p et q se calcule avec cette formule :

$$d(p, q) = \sqrt{\sum_{i=1}^n (p_i - q_i)^2}$$

On a à présent à notre disposition des nouveaux jeux de données. Après les avoir nettoyés et ajuster leur taille, on crée un code afin d'afficher la localisation de nos nouvelles stations. On obtient alors la carte de la *figure 12*.



FIGURE 12

On remarque trois groupes de stations bien distincts. On souhaite ajouter un quatrième groupe dans le sud-ouest. Pour cela, on utilise le jeu de données précédent *cf figure 8* afin de sélectionner uniquement les villes qui nous intéressent, puis on les rajoute au nouveau dataset.

On délimite le sud-ouest de la façon suivante : latitude maximale de 43.69, correspondant à celle de Auch, et longitude maximale de 1.38, correspondant à celle de Toulouse. Après avoir créé et exécuté le programme permettant d'afficher les quatre groupes de stations, on obtient la carte suivante :



FIGURE 13

On utilise alors l'algorithme des k-means sur la position et on affiche le résultat sur une carte.

```
loc = df3.to_numpy()

# Ajout des labels de cluster au DataFrame
df3['cluster'] = kmeans.labels_

# Initialisation de la carte Folium centrée sur la France
m = folium.Map(location=[46.603354, 1.888334], zoom_start=6)

# Définition des couleurs pour les clusters
colormap = np.array(["red", "green", "blue", "yellow"])

# Ajout des points clusterisés à la carte
```

```

for i in range(143):
    folium.CircleMarker(
        location = [loc[i,0], loc[i,1]],
        radius=5,
        color=colormap[df3['cluster'].iloc[i]],
        fill=True,
        fill_color=colormap[df3['cluster'].iloc[i]],
        fill_opacity=0.7
    ).add_to(m)

# Affichage de la carte
m.save('kmeans-clusters-france.html')
m

```

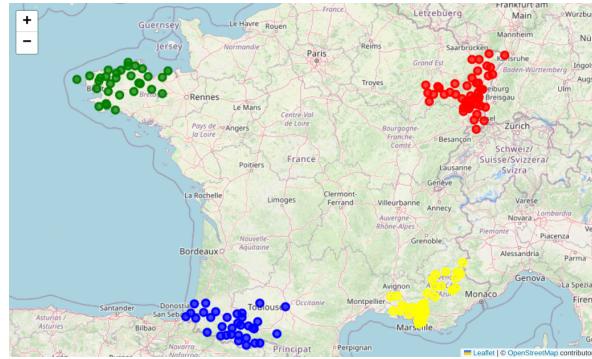


FIGURE 14. Méthode des k-means sur la position

Ici la méthode des k-means a mis en évidence les quatre groupes que nous avions nous-mêmes identifiés sur la *figure 13*. Répétons à présent cette démarche, mais appliquée aux températures. On crée un code qui calcule la moyenne des températures de chaque mois de l'année et stocke les résultats dans un tableau.

```

# Obtenir les indices d'un mois spécifique
def indices_mois(i):
    new_date2['a'] = pd.to_datetime(new_date2['a'], format='%Y-%m-%d')
    indices_mois = new_date2[new_date2['a'].dt.month == i].index.tolist()
    return indices_mois

# Calcul des moyennes de chaque mois
def mean_temp(i):
    indices = indices_mois(i)
    tem_mois = new_temp4[:, indices]
    mean_temp = np.nanmean(tem_mois, axis=1)
    return mean_temp

# Tout regrouper dans un tableau

```

```
moyennes_mensuelles = np.zeros((141, 12))
```

```
for i in range(12):
```

```
    moyennes_mensuelles[:, i] = mean_temp(i+1)
```

Puis grâce à la méthode d'Elbow on s'assure que le nombre de clusters optimal est bien 4 et on utilise l'algorithme des k-means.

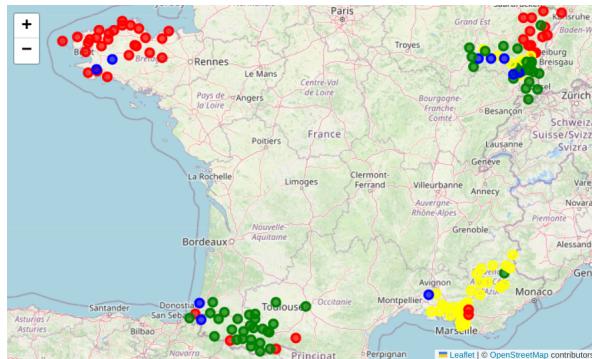


FIGURE 15. Méthode des k-means sur la température

La *figure 15* met en évidence trois groupes distincts au nord-ouest, sud-ouest et sud-est, mais le groupe du nord-est n'est pas clairement identifiable. On peut émettre des hypothèses qui expliqueraient la raison pour laquelle la méthode des k-means n'a pas mis en évidence quatre groupes distincts.

Ce résultat peut venir des données de température de la région du nord-est qui pourraient être assez similaires à celles des autres régions. Dans ce cas, le modèle peut avoir du mal à définir des clusters clairs.

Sans compter que la géographie de la région peut aussi influencer le résultat. En effet, si la région nord-est partage des caractéristiques géographiques similaires avec une autre région (par exemple : des plaines agricoles ou des montagnes), cela peut conduire à une similarité dans les données de température.

## CONCLUSION

En conclusion, ce stage au sein de l'Université m'a offert l'occasion d'approfondir mes connaissances en traitement et analyse des données. J'ai pu interpréter les résultats obtenus et en tirer des conclusions éclairées.

De plus, ce stage m'a permis de perfectionner mes compétences en Python tout en me formant à l'utilisation de LaTeX.

Cela m'a conforté dans mon choix d'orientation en me permettant de me rendre compte que grâce aux statistiques il est possible de préciser de nombreuses incertitudes de la vie courante.

REFERENCES

1. <https://ledatascientist.com/faire-du-clustering-avec-lalgorithme-k-means/>
2. <https://lemakistatheux.wordpress.com/2013/05/09/le-test-de-kolmogorov-smirnov/>
3. [https://eric.univ-lyon2.fr/ricco/cours/cours/Test\\_Normalite.pdf](https://eric.univ-lyon2.fr/ricco/cours/cours/Test_Normalite.pdf)