INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES CENTRE VAL DE LOIRE

Manuel d'utilisation

Projet Python 3A

Damien Briquet et Alexis Brunet

Contents

1 Objectifs

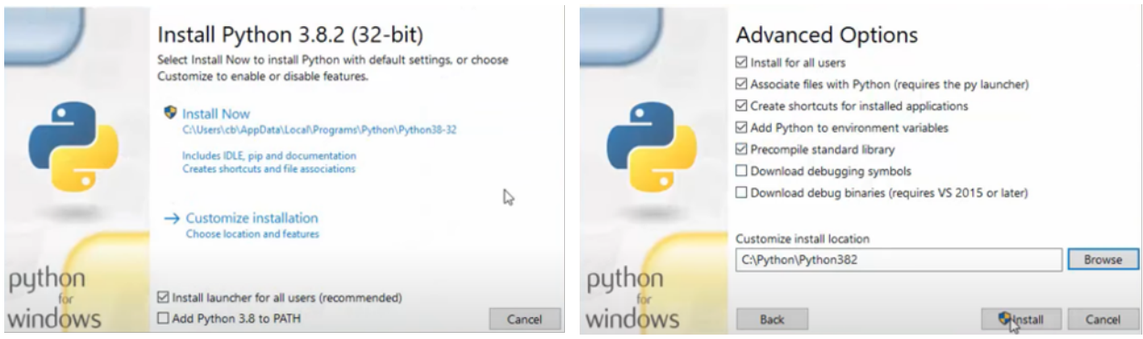
Python est un langage de programmation très populaire dans le monde informatique pour réaliser des tâches variées tel que l'automatisation, les sites internets, l'Intelligence Artificielle (IA), etc. Notre objectif est de comprendre en détail le code donné mais aussi le simplifier, en gardant uniquement les dépendances utiles au projet. Cela nous permettra de réaliser une documentation claire pour utiliser le script sur un ordinateur neuf et par une personne pas forcément formée pour son utilisation. Dans un second temps, nous pourrons élargir ce code aux bases de données disponibles sur le site kaggle.

2 Installation de Python

Ce projet fonctionne sur un environnement Python, et de préférence sur la version Python 3.8.0 (version vérifiée pour ce projet).

2.1 Windows

Téléchargez le fichier exécutable sur le site officiel Python de la version Python 3.8.0. Exécutez le fichier `.exe`. Commencez par cocher les deux cases en bas sur la première fenêtre de dialogue, comme sur l'image qui suit. Allez dans les `Advanced Options`, cochez toutes les cases, permettant ainsi d'installer pip (un gestionnaire de paquets utiles par la suite), puis revenez en arrière, et avancez jusqu'à l'installation complète.



(a) Fenêtre Python

(b) Options Python

Figure 1: Installation de python Windows 10

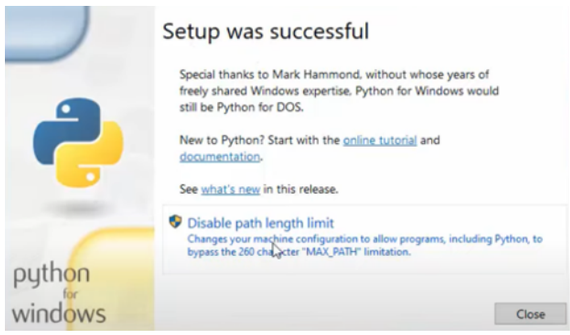


Figure 2: Fin de l'installation Python

À la fin de l'exécution, Python est complétement installé sur votre système. Pour vérifier que Python est bien installé, ouvrez un CMD et tapez la commande suivante :

```
python3 --version #Vous devez obtenir le résultat suivant Python 3.8.0
```

2.2 Mac

Deux méthodes s'offrent à vous, soit passant par le site officiel (méthode simple), soit en ligne de commandes avec Homebrew.

2.2.1 1ère méthode : Site Officiel

Téléchargez le package sur le site officiel Python de la version Python 3.8.0.
Exécutez le fichier `.pkg` et suivez les différentes étapes sur l'écran.
À la fin de l'exécution, Python est complétement installé sur votre système.
Pour vérifier que Python est bien installé, ouvrez un Terminal et tapez la commande suivante :

```
python3 --version #Vous devez obtenir le résultat suivant Python 3.8.0
```

2.2.2 2ème méthode : Homebrew

Pour installer Homebrew, ouvrez un Terminal et tapez la commande :

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/
    install.sh)"
```

Puis installez Python avec la commande suivante :

```
brew install python@3.8
```

À la fin de l'exécution, Python est complétement installé sur votre système. Pour vérifier que Python est bien installé, ouvrez un Terminal et tapez la commande suivante :

```
python3 --version #Vous devez obtenir le résultat suivant Python 3.8.0
```

2.3 Linux

Dans un Terminal, tapez la commande suivante :

```
sudo apt-get install python3.8
```

Pour vérifier que Python est bien installé, tapez la commande suivante :

```
python3 --version #Vous devez obtenir le résultat suivant Python 3.8.0
```

3 Installation d'un IDE

Vous pouvez installer un des deux IDE (Integrated Development Environment) suivants pour la suite du projet. Visual Studio Code est un IDE disponible sur Mac, Windows et Linux tandis que Spyder IDE n'est disponible que sur Mac et Windows.

3.1 Visual Studio Code

3.1.1 Windows - Mac

Téléchargez le fichier `.exe` (Windows) ou `.dmg` (Mac) disponible sur la page de Visual Studio Code. Ensuite il suffit d'exécuter le fichier `.exe` ou `.dmg` téléchargé précédemment. Suivez les instructions à l'écran pour l'installer. Une fois installé vous pouvez le lancer et commencer à coder.

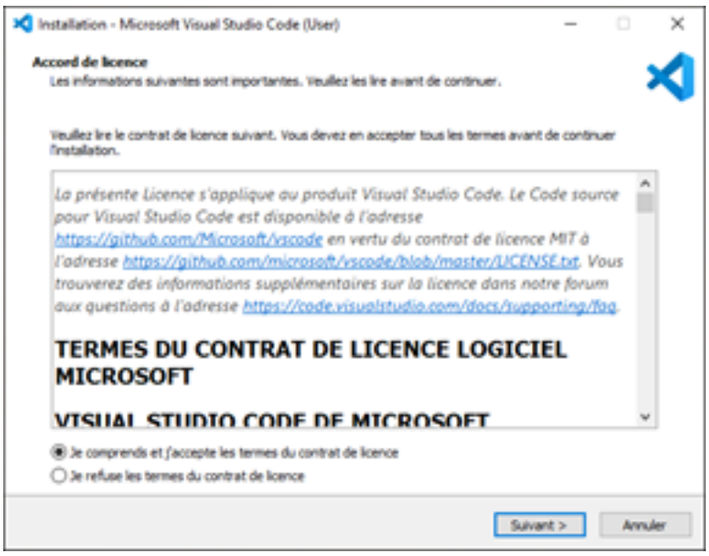


Figure 3: Installation VS Code - Windows 10

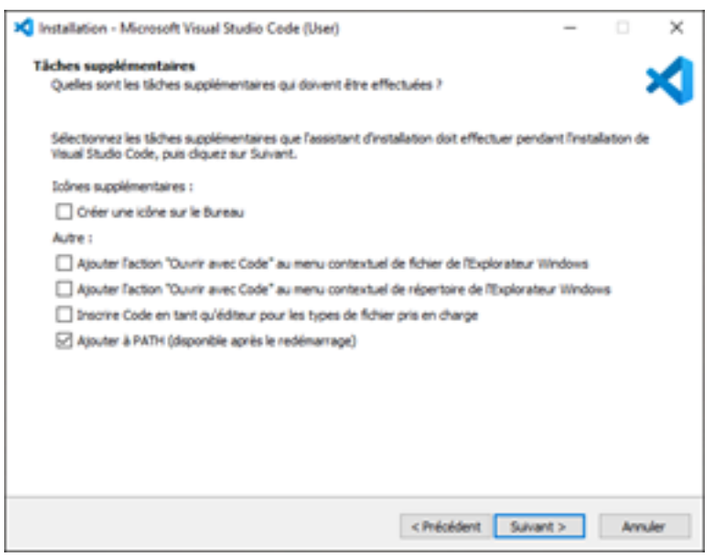


Figure 4: Installation VS Code - Windows 10

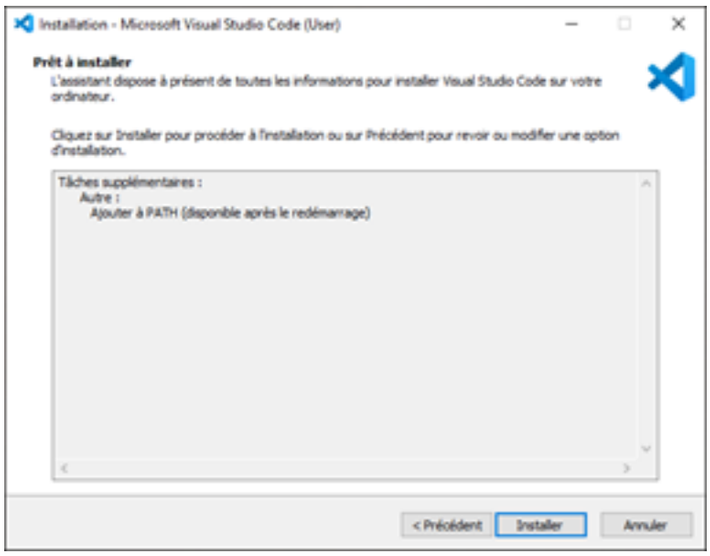


Figure 5: Installation VS Code - Windows 10

3.1.2 Linux - Ubuntu, Debian

Téléchargez le fichier `.deb` disponible sur la page de Visual Studio Code. Ensuite, ouvrez un Terminal, naviguez jusqu'au dossier où le fichier deb a été téléchargé (avec la commande `cd`) et tapez la commande suivante :

```
sudo dpkg -i nom_du_paquet_deb.deb #Remplacer nom_du_paquet_deb.deb par votre nom de
    paquet
```

3.2 Spyde IDE Python (Mac - Windows)

Téléchargez le fichier `.exe` (Windows) ou `.dmg` (Mac) disponible sur la page de Spyder IDE. Ensuite il suffit d'exécuter le fichier `.exe` ou `.dmg` téléchargé précèdemment. Suivez les instructions à l'écran pour l'installer. Une fois installé vous pouvez le lancer et commencer à coder.

4 Installation des dépendances

Pour le bon fonctionnement du projet, vous avez besoin d'installer plusieurs dépendances. Vous avez besoin des dépendances suivantes :

- **numpy** sert à créer et manipuler des tableaux multidimensionnels et permet l'utilisation de fonctions mathématiques dans les dits tableaux.
- **pandas** nous permet la manipulation et l'analyse de données numériques et de séries temporelles.
- **seaborn** nous sert d'interface permettant d'afficher les résultats sous forme de graphiques statistiques.
- **statsmodels** regroupe les modèles mathématiques statistiques existants afin de faire des prévisions.
- **mlxtend** est un module de machine learning analysant les données pour pouvoir ensuite se développer et affiner les prévisions.
- **matplotlib** est destinée à tracer des graphiques à partir de données.

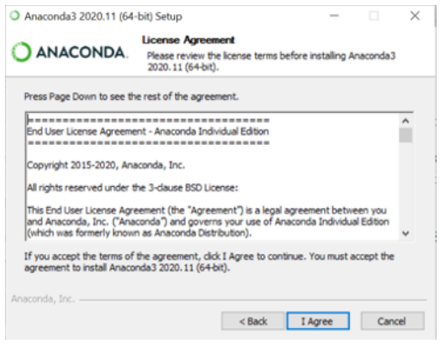
4.1 Windows

Vous pouvez installer les dépendances soit avec Anaconda (méthode 1), soit avec pip (méthode 2).

4.1.1 Méthode 1 : Anaconda

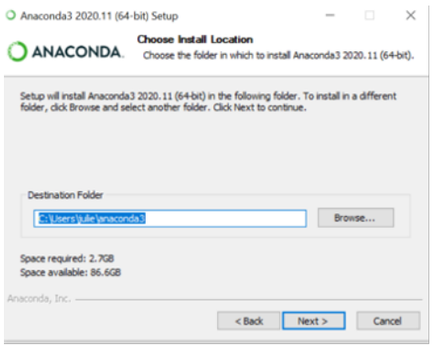
Cette méthode est plus simple, car vous allez installer Anaconda qui installera une partie des autres dépendances (Numpy, Pandas) nécessaires au projet. Téléchargez la dernière version de Anaconda pour Windows sur le site. Exécutez le fichier `.exe`, suivez les instructions qui apparaissent à l'écran jusqu'à terminer l'installation.

(a) Anaconda - Étape 1

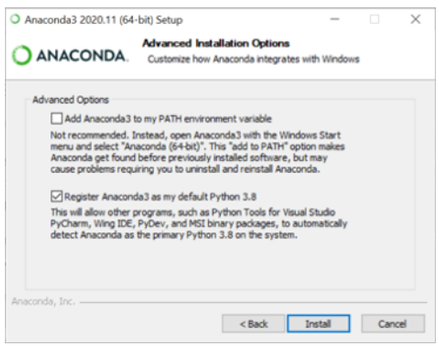


(b) Anaconda - Étape 2

Figure 6: Insatallation d'Anaconda

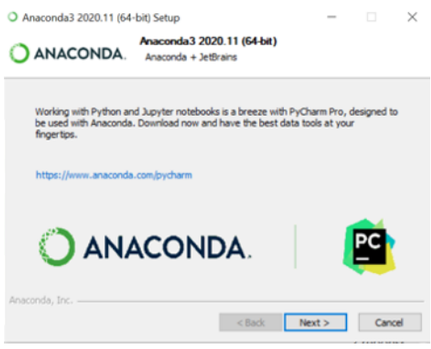


(a) Anaconda - Étape 3



(b) Anaconda - Étape 4

Figure 7: Insatallation d'Anaconda



(a) Anaconda - Étape 5

(b) Anaconda - Étape 6

Figure 8: Insatallation d'Anaconda

Ensuite vous allez installer la dernière dépendance nécessaire mlxtend, donc ouvrir un Terminal d'Anaconda par l'outil de recherche Windows en tapant "Anaconda Powershell Prompt", et copiez la commande suivante :

```
conda install mlxtend --channel conda-forge
```

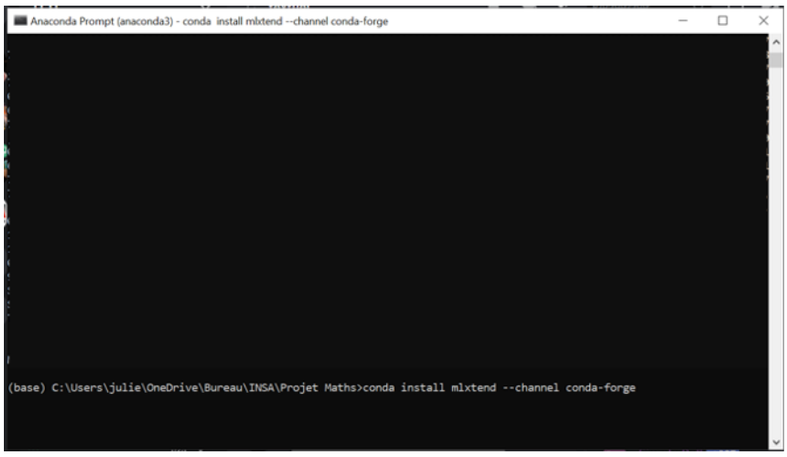


Figure 9: Anaconda - mlxtend - Étape 1

Figure 10: Anaconda - mlxtend - Étape 2

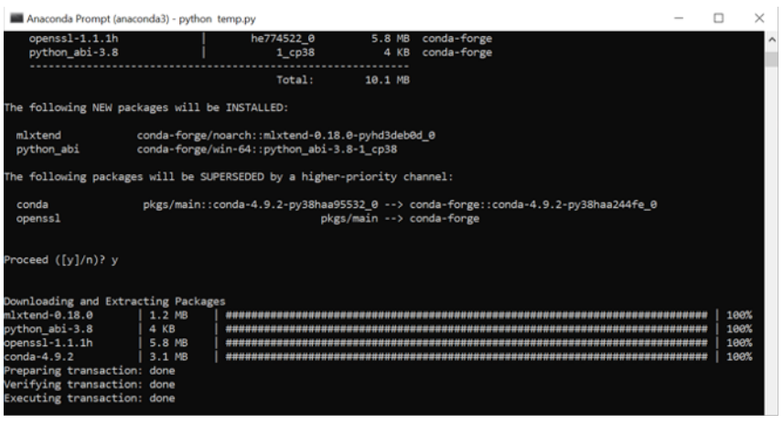


Figure 11: Anaconda - mxltend - Étape 3

Maintenant toutes les dépendances nécessaires sont installées.

4.1.2 Méthode 2 : pip

Vérifiez que pip est installé avec la commande :

```
pip -V
```

S'il n'est pas installé, téléchargez le fichier python get-pip.py. Placez ce fichier python dans le dossier d'installation, le chemin d'accès à ce dossier ressemble à `C:\Users\XXX\AppData\Local\Programs\Python\Python-38\`
Python-38 correspond à la version de votre Python, que vous pouvez connaitre en ouvrant un cmd et en tapant la commande `python --version`.
XXX correspond à votre nom d'utilisateur Windows.

Ensuite ouvrez un cmd, et naviguez jusqu'à ce dossier Python avec la commande `cd C:\Users\XXX\AppData\Local\Programs\Python\Python-38\`
Puis ensuite exécutez le script : `python get-pip.py`
Pip est maintenant installé sur votre système. Vous pouvez vérifier de nouveau avec la commande :

```
pip -V
```

Maintenant vous pouvez installer les dépendances. Tout d'abord, téléchargez le zip du projet. Dézippez-le, ouvez un CMD et tapez les commandes suivantes :

```
cd C:\Users\XXX\Documents\Projet-Python-INSA-3A #Par exemple, dépend de là où vous l'avez
    mis, ici c'est dans les Documents
pip install -r requirements.txt
```

Maintenant toutes les dépendances nécessaires sont installées.

4.2 Mac

Pour installer ces dépendances, vous utiliserez pip un gestionnaire de paquets Python. Installez `pip` avec les commandes suivantes :

```
curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
python3 get-pip.py
```

pip est normalement installé. Pour vérifier, tapez :

```
pip --version
```

Téléchargez le zip du projet à cette adresse ou avec la commande suivante :

```
curl -LJ -o weather.zip https://github.com/AlexTheGeek/Python-Project-INSA-3A/archive/v1.
    zip
```

Ouvrez un Terminal et tapez les commandes suivantes :

```
unzip weather.zip
cd Project-Python-3A-MRI-1
pip install -r requirements.txt
```

Maintenant toutes les dépendances nécessaires sont installées.

4.3 Linux

Pour installer les dépendances, il faut que pip soit installé sur votre système : Sur Ubuntu pour installer pip, vous pouvez taper la commande suivante : `sudo apt-get install python3-pip`. Dans la suite vous utiliserez la commande `pip` mais en passant par le gestionnaire de paquets `apt`, pour cela il faudra taper la commande `pip`.

```
curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
python3 get-pip.py
```

Pip est normalement installé. Pour vérifier sa version, tapez :

```
pip --version
```

Maintenant vous pouvez installer les dépendances.
Téléchargez le zip du projet à cette adresse ou avec la commande suivante :

```
curl -LJ -o weather.zip https://github.com/AlexTheGeek/Python-Project-INSA-3A/archive/v1.
    zip
```

Ouvrez un Terminal et tapez les commandes suivantes :

```
unzip weather.zip
cd Project-Python-3A-MRI-1
pip install -r requirements.txt
```

Maintenant toutes les dépendances nécessaires sont installées.

5 Explication du code

Dans cette partie nous allons expliquer les différentes actions faites par le programme, par des blocs de lignes de code. Nous présenterons ainsi une sortie possible grâce à ces lignes. Vous pouvez aussi retrouver le code complet annoté à cette adresse pour le main et la fonction. Tout d'abord notre programme est composé de deux scripts Python :

- main.py : contenant le corps du programme (script python à exécuter pour lancer le programme)
- function.py : contenant la fonction utilisée dans main.py

5.1 Bloc 1 : initialisation d'un DataFrame à partir un fichier csv

Récupération des données séparées par des virgules à partir d'un fichier csv pour générer un échantillon de manière aléatoire, et ainsi afficher des informations sur ce DataFrame (dimension, statistique, utilisatioon mémoire, valeurs non nulles, colonnes, index dtype, ...).

```
dataset1 = pd.read_csv('weather.csv') #Lecture d'un fichier (csv) de valeurs séparées par
    des virgules dans un DataFrame
dataset1.sample() #Génération d'un échantillon aléatoire de chaque groupe d'un objet du
    DataFrame
dataset=dataset1.sample(1000) #Un nouvel DataFrame (dataset) contenant 1000 élèments
    échantillonnés de façon aléatoire à partir de dataset1
print(dataset.shape) #Affichage d'un tuple représentant la dimension du DataFrame dataset
print(dataset.describe()) #Affichage des statistiques pour chaque type de valeurs du
    DataFrame
dataset.info() #Affichage des informations sur le DataFrame, notamment l'index dtype, les
    colonnnes, les valeurs non nulles et l'utilisation de la mémoire
```

Sortie possible :

```
~/Downloads/Projet_Maths_Python python3 temp.py                                    14:22:21
sys:1: DtypeWarning: Columns (7,8,18,25) have mixed types.Specify dtype option on import or set low_memory=False.
(1000, 31)
                STA  WindGustSpd     Temp_Max     Temp_Min    MeanTemp           YR           MO           DA           DR          SPD          MAX          MIN          MEA   SND   FT   FB  FTI  ITH          PGT  SD3  RHX  RHN  RVG  WTE
count   1000.000000    9.000000  1000.000000  1000.000000  1000.000000  1000.000000  1000.000000  1000.000000    9.000000    9.000000   995.000000   994.000000   994.000000  60.0  0.0  0.0  0.0  0.0    9.000000  0.0  0.0  0.0  0.0  0.0
mean   30501.709000   34.364889    27.155000    18.083333    22.602222    43.828000     6.749000    16.024000   29.777778   18.555556    81.285427    64.939638    73.122736   0.0  NaN  NaN  NaN  NaN   11.033333  NaN  NaN  NaN  NaN  NaN
std    21845.548408    9.402941     8.507710     8.330673     8.229043     1.089772     3.347418     8.801869   16.230972    5.077182    14.234630    14.173446    13.733730   0.0  NaN  NaN  NaN  NaN    6.931991  NaN  NaN  NaN  NaN  NaN
min    10001.000000   20.372000   -23.333333   -26.666667   -24.444444    40.000000     1.000000     1.000000    7.000000   11.000000   -10.000000   -16.000000   -12.000000   0.0  NaN  NaN  NaN  NaN    0.600000  NaN  NaN  NaN  NaN  NaN
25%    11801.000000   27.780000    25.555556    15.555556    21.111111    43.000000     4.000000     8.000000   22.000000   15.000000    76.000000    60.000000    70.000000   0.0  NaN  NaN  NaN  NaN    8.000000  NaN  NaN  NaN  NaN  NaN
50%    22508.000000   38.892000    29.444444    21.111111    25.555556    44.000000     7.000000    16.000000   32.000000   21.000000    85.000000    70.000000    78.000000   0.0  NaN  NaN  NaN  NaN    8.800000  NaN  NaN  NaN  NaN  NaN
75%    34022.000000   42.596000    31.666667    23.333333    27.222222    45.000000    10.000000    23.000000   44.000000   23.000000    89.000000    74.000000    81.000000   0.0  NaN  NaN  NaN  NaN   12.000000  NaN  NaN  NaN  NaN  NaN
max    82506.000000   44.448000    45.555556    31.111111    36.666667    45.000000    12.000000    31.000000   55.000000   24.000000   114.000000    88.000000    98.000000   0.0  NaN  NaN  NaN  NaN   22.100000  NaN  NaN  NaN  NaN  NaN
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1000 entries, 62274 to 38833
Data columns (total 31 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   STA          1000 non-null   int64
 1   Date         1000 non-null   object
 2   PrecIp       1000 non-null   object
 3   WindGustSpd  9 non-null      float64
 4   Temp_Max     1000 non-null   float64
 5   Temp_Min     1000 non-null   float64
 6   MeanTemp     1000 non-null   float64
 7   Snowfall     989 non-null    object
 8   PoorWeather  314 non-null    object
 9   YR           1000 non-null   int64
 10  MO           1000 non-null   int64
 11  DA           1000 non-null   int64
 12  PRCP         988 non-null    object
 13  DR           9 non-null      float64
 14  SPD          9 non-null      float64
 15  MAX          995 non-null    float64
 16  MIN          994 non-null    float64
 17  MEA          994 non-null    float64
 18  SNF          989 non-null    object
 19  SND          60 non-null     float64
 20  FT           0 non-null      float64
 21  FB           0 non-null      float64
 22  FTI          0 non-null      float64
 23  ITH          0 non-null      float64
 24  PGT          9 non-null      float64
 25  TSHDSBRSGF   314 non-null    object
 26  SD3          0 non-null      float64
 27  RHX          0 non-null      float64
 28  RHN          0 non-null      float64
 29  RVG          0 non-null      float64
 30  WTE          0 non-null      float64
dtypes: float64(20), int64(4), object(7)
memory usage: 250.0+ KB
None
```

Figure 12: Sortie 1

5.2 Bloc 2 : graphique en nuage de points de la répartition des températures

Affichage de la répartition des températures en fonction des températures minimales et maximales, dans un graphique en nuage de points.

```
dataset.plot(x='Temp_Min', y='Temp_Max', style='o') #Création d'un tracé de DataFrame,
    avec l'axe x représentant les Temp_min et l'axe y représentant les Temp_Max, dans un
    style de nuage de points
plt.title('Temp_Min vs Temp_Max') #Paramétrage du titre du graphique
plt.xlabel('Temp_Min') #Paramétrage du titre de l'axe x
plt.ylabel('Temp_Max') #Paramétrage du titre de l'axe y
plt.show() #Affichage du graphique/figure
```

Sortie possible :

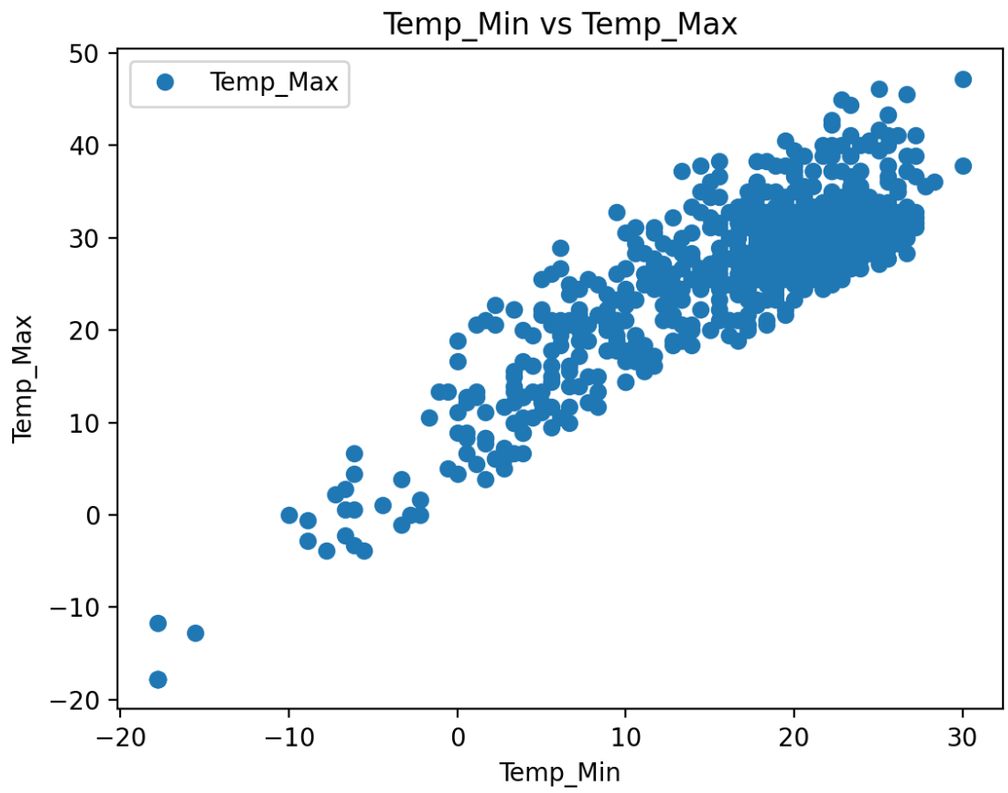


Figure 13: Sortie 2

5.3 Bloc 3 : graphique en histogramme de la répartition des températures Max

Création d'un graphique histogramme pour visualiser la répartion des valeurs de `Temp_Max` du DataFrame.

```
plt.figure(figsize=(15,10)) #Création d'une figure de taille définie par figsize en inch,
    15 inch de largeur et 10 de hauteur
plt.tight_layout() #Ajustement des bordures entre et autour des sous traces
seabornInstance.distplot(dataset['Temp_Max']) #Permet de dessiner un tracé de
    distribution sur une FacetGrid, permettant de visualiser les données de DataFrame des
    Temp_Max dans un format d'histogramme (par défaut)
plt.show() #Affichage du graphique/figure
```

Sortie possible :

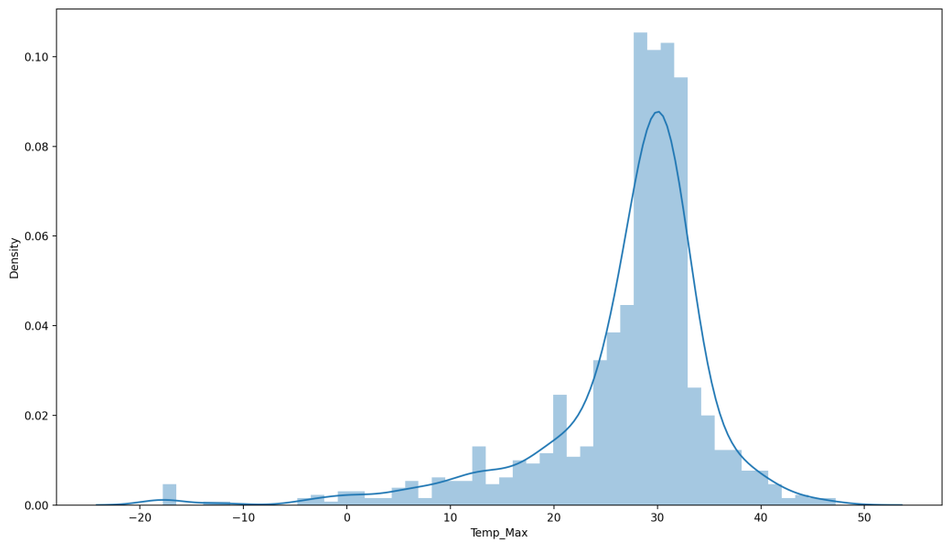


Figure 14: Sortie 3

5.4 Bloc 4 : initialisation de la prédiction

Prédiction des `Temp_Max` à partir du fichier précèdent.

```
X = dataset['Temp_Min'].values.reshape(-1,1) #La valeur X inclut l'attribut Temp\_Min
y = dataset['Temp_Max'].values.reshape(-1,1) #La valeur y inclut l'attribut Temp\_Max
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
    #Attribution de 80% des donnees a l'ensemble de formation et le reste a l'ensemble de
     test
reg = LinearRegression().fit(X_train, y_train) #Entrainement du modele en utilisant l'
    ensemble de formation

print(reg.intercept_) #Affichage de l'intersection
print(reg.coef_) #Affichage du coefficient directeur de la droite de régression
y_pred = reg.predict(X_test) #Utilisation des données de test pour faire des prédictions
    sur le Temp_Max
df = pd.DataFrame({'Actuelle(Mesurees)': y_test.flatten(), 'Prediction(modele)': y_pred.
    flatten()}) #Construction d'un DataFrame à partir des données prédites et de
    tests
print(df) #Affichage de DataFrame
```

Sortie possible :

```
[11.39385154]
[[0.8749467]]
     Actuelle(Mesurées)  Prédiction(modèle)
0             27.222222           29.378867
1             38.333333           34.239682
2             35.000000           30.837112
3             32.777778           31.809275
4             30.555556           32.781438
..                  ...                 ...
195           28.888889           31.809275
196           30.000000           21.601563
197           15.555556           13.338178
198           25.000000           23.059808
199           30.000000           29.864949

[200 rows x 2 columns]
```

Figure 15: Sortie 4

5.5 Bloc 5 : graphique à barre verticales de la répartition des valeur prédites et mesurées

Création du graphique à barre verticale montrant la répartition des températures mesurées et prédites.

```
df1 = df.head(25)  #Récupération des 25 premières lignes de df pour les mettre dans df1
df1.plot(kind='bar',figsize=(16,10)) #Création de DataFrame, de taille 16inch de largeur
    et 10inch de hauteur, et le style du graphique sera des barres verticales
plt.grid(which='major', linestyle='-', linewidth='0.5', color='green') #Création de la
    grille interne (major) du graphique avec un style de trait plein, épaisseur de
    0.5points, une couleur verte
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black') #Création du cadre
    (minor) du graphique avec un style de trait à points, épaisseur de 0.5points, une
    couleur noire
plt.show() #Affichage du graphique/figure
```

Sortie possible :

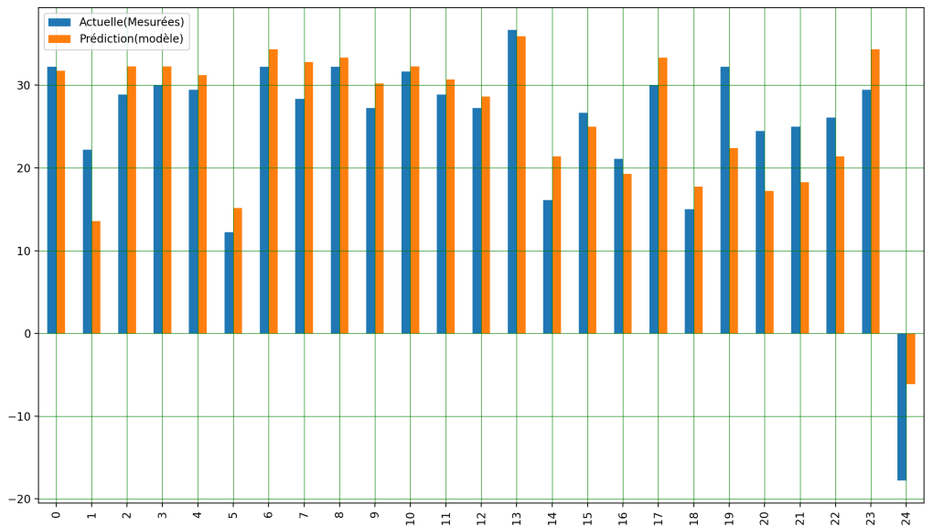


Figure 16: Sortie 5

5.6 Bloc 6 : graphique de la droite de régression

Création d'un graphique de dispersion en gris avec la droite de régréssion en rouge calculée précèdemment

```
plt.title('Modele ax+y') #Paramétrage du titre du graphique
plt.xlabel('Temp_Min') #Paramétrage du titre de l'axe x
plt.ylabel('Temp_Max') #Paramétrage du titre de l'axe y
plt.scatter(X_test, y_test,  color='gray') #Création d'un diagramme de dispersion y_test
    par rapport à X_test de couleur grise
plt.plot(X_test, y_pred, color='red', linewidth=2) #Création d'un tracé (de la fonction
    ax+y) de DataFrame, avec l'axe x représentant les X_Test et l'axe y représentant les
    y_pred, dans la couleur rouge et de largeur 2points
plt.show() #Affichage du graphique/figure
```

Sortie possible :

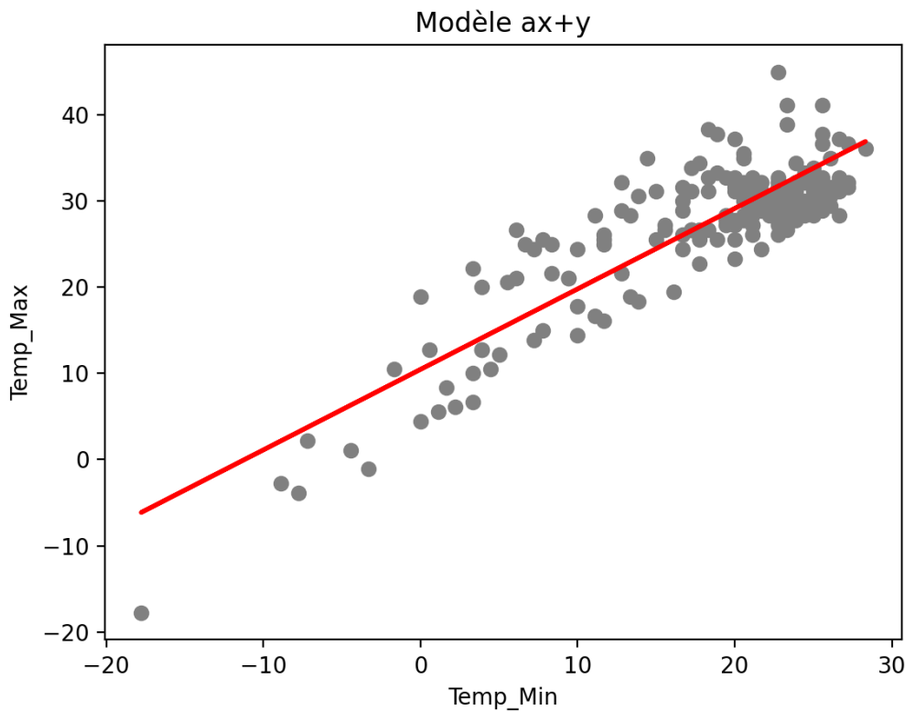


Figure 17: Sortie 6

5.7 Bloc 7 : Quelques valeurs de la prédiction

Affiche des valeurs en fonction des prédictions

```
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred)) #Affichage du
    calcul des valeurs absolues moyennes des erreurs
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred)) #Affichage du
    calcul de la moyenne des erreurs au carré
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred))) #
    Affichage du calcul de la racine carrée de la moyenne des erreurs
    quadratiques
```

Sortie possible :

```
Mean Absolute Error: 3.159714093142225
Mean Squared Error: 17.851588986133976
Root Mean Squared Error: 4.225114079659149
```

Figure 18: Sortie 7

5.8 Bloc 8 : Statistique d'un tableau de moyenne

Ajout des valeurs moyenne dans un tableau et affichage de ce tableau et des valeurs moyennes et écart-type.

```
Means = [] #Création d'un tableau vide des valeurs moyennes
Means=testSamples(200, 100,dataset['Temp_Min']) #Ajout des valeurs moyenne dans le
    tableau avec la fonction testSamples créée précédemment
print(Means) #Affichage du tableau des valeurs moyennes
print(np.mean(Means)) #Affichage de la moyenne du tableau Means
print(np.mean(dataset['Temp_Min'])) #Affichage de la moyenne du tableau
    dataset['Temp_Min']
print(np.std(Means)) #Affiche l'écart-type des valeurs du tableau Means
print(np.std(dataset['Temp_Min'])) #Affiche l'écart-type des valeurs du tableau
    dataset['Temp_Min']
```

5.8.1 Bloc de la fonction associée

La fonction `testSamples` permet de créer des valeurs à partir d'un échantillon et ainsi de les insérer dans un tableau.

```
class fonction: #Création de la classe fonction pour toutes les fonctions utilisables
    pour temp.py
def  testSamples(numTrials, sampleSize, data): #Définition de la fonction testsamples
    prenant en paramètre numTrials (int), sampleSize (int), data (tableau de float)
    Means = [] #Création d'un tableau vide
    for t in range(numTrials): #Boucle for commençant de 0 allant à numTrials-1 par pas
    de 1
        Y=data.sample(sampleSize) #Y récupère un échantillon aléatoire d'éléments de
    taille SampleSize
        Means.append(sum(Y)/len(Y)) #Ajout dans le tableau de la division de la somme de
    Y divise par la longuer de Y
    return Means  #On retourne le tableau Means
```

Sortie possible :

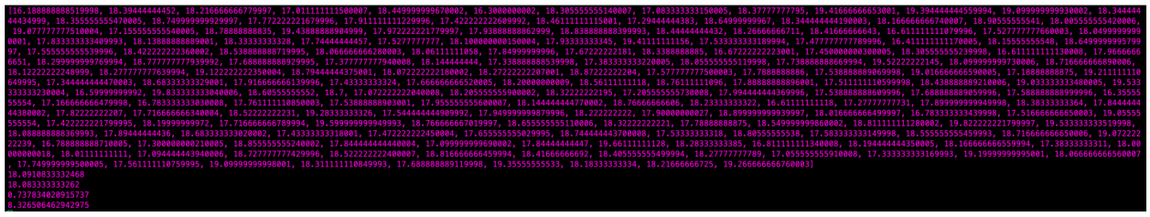


Figure 19: Sortie 8

5.9 Bloc 9 : diagramme en escalier

Création d'un histogramme en escalier.

```
plt.figure(1) #Création d'une figure avec un unique identifiant égal à 1
plt.hist(Means, bins=10, histtype='step') #Création d'un histogramme en escalier avec un
    seul trait et sans remplissage, avec 10 marches ayant la même largeur
plt.show() #Affichage du graphique/figure
```

Sortie possible :

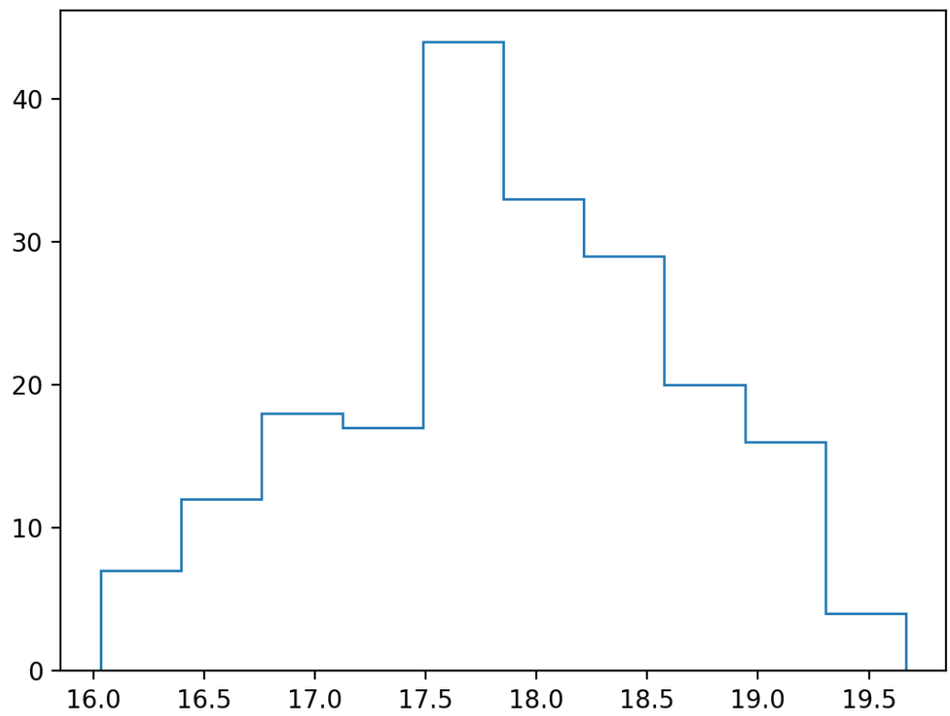


Figure 20: Sortie 9

5.10 Bloc 10 : test de Shapiro-Wilk

Test de Shapiro-Wilk sur une population distribuée normalement.

```
stat, p = stats.shapiro(Means) #On fait le test de Shapiro-Wilk qui vérifie l'hypothèse
    nulle selon les données de Means, et retourne la valeur de la statistique du test et
    la p-value pour l'hypothése du test
print('Statistics={}, p={}'.format(stat, p)) #Affichage de la statistique et de la
    p-value
alpha = 0.05 #Initialisation de alpha
if p > alpha: #Test entre p-value et alpha
    print('Sample looks Normal (do not reject H0)') #Affichage si p-value > alpha
else:
    print('Sample does not look Normal (reject H0)') #Affichage si p-value < alpha
```

Sortie possible :

```
Statistics=0.9941580295562744, p=0.624136209487915
Sample looks Normal (do not reject H0)
```

Figure 21: Sortie 10

5.11 Bloc 11 : boite à moustaches

Diagramme en boites et à moustaches. La boîte s'étend des valeurs du quartile inférieur au quartile supérieur des données, avec une ligne à la médiane. Les moustaches s'étendent à partir de la boîte pour montrer l'étendue des données. Les points de vol sont ceux qui se trouvent après l'extrémité des moustaches.

```
plt.boxplot(Means) #Création d'un diagramme en boites et à moutaches de Means
plt.show() #Affichage du graphique/figure
```

Sortie possible :

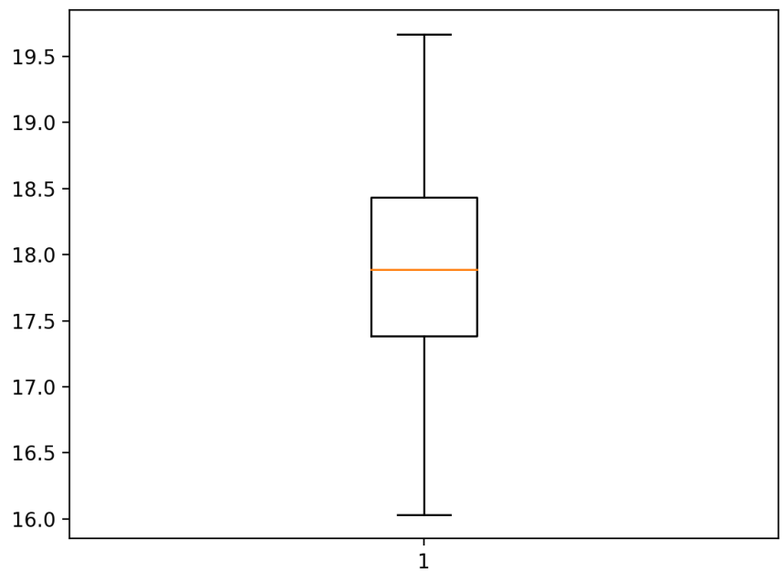


Figure 22: Sortie 11

5.12 Bloc 12 : test de Shapiro-Wilk

Création et affichage de la courbe de probabilité

```
stats.probplot(Means, plot=plt) #Calcule les quantiles de la courbe de probabilité
    normale de Means et la trace avec matplotlib
plt.show() #Affichage du graphique/figure
```

Sortie possible :

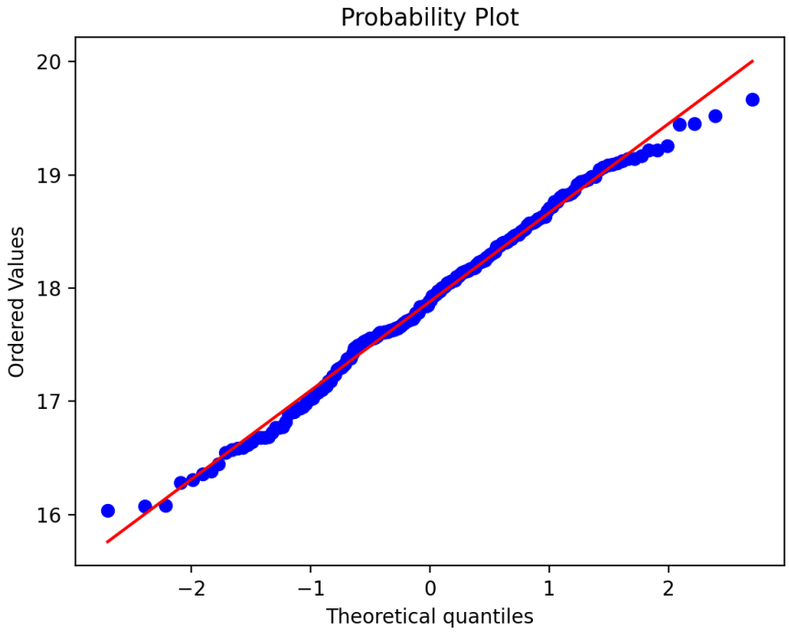


Figure 23: Sortie 12

5.13 Bloc 13 : coefficient de régression

Ecriture par deux manières différentes du calcul du coefficient de régression linéaire.

```
X, y = make_classification(n_samples=100, n_features=10, n_informative=10, n_redundant=0,
     random_state=1) #Paramètrage du nuage de
    points
model1 = LogisticRegression() #Initialisation de l'évaluation du modèle 1 par la fonction
    LogisticRegression()
cv1 = RepeatedStratifiedKFold(n_splits=2, n_repeats=5, random_state=1) #Paramétrage de la
    variable cv1
scores1 = cross_val_score(model1, X, y, scoring='accuracy', cv=cv1, n_jobs=-1) #
    Paramétrage de la variable
    scores1
print('LogisticRegression Mean Accuracy: %.3f (%.3f)' % (np.mean(scores1), np.std(scores1
    ))) #Affichage du coefficient de régression linéaire moyen pour le modèle
    1

model2 = LinearDiscriminantAnalysis() #Initialisation de l'évaluation du modèle 2 par la
    fonction
cv2 = RepeatedStratifiedKFold(n_splits=2, n_repeats=5, random_state=1) #Paramétrage de la
    variable cv2
scores2 = cross_val_score(model2, X, y, scoring='accuracy', cv=cv2, n_jobs=-1) #
    Paramétrage de la variable
    scores2
print('LinearDiscriminantAnalysis Mean Accuracy: %.3f (%.3f)' % (np.mean(scores2), np.std
    (scores2))) #Affichage du coefficient de régression linéaire moyen pour le modèle
    2
```

Sortie possible :

```
LogisticRegression Mean Accuracy: 0.854 (0.027)
LinearDiscriminantAnalysis Mean Accuracy: 0.858 (0.023)
```

Figure 24: Sortie 13

5.14 Bloc 14 : diagramme boite à moustaches

Création du diagramme boite à moustaches

```
plt.boxplot([scores1, scores2], labels=['LR', 'LDA'], showmeans=True) #Création d'un
    diagramme boite à moustaches à partir de scores1 et scores2
plt.show() #Affichage du graphique/figure
```

Sortie possible :

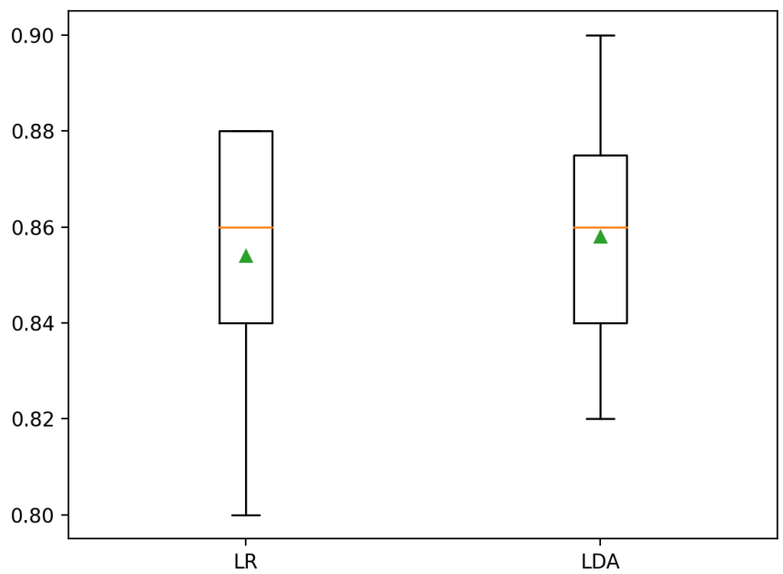


Figure 25: Sortie 14

5.15 Bloc 15 : test entre les deux modèles

Détermination de la P-value et de T-stastitic pour faire un test entre les deux modèles

```
t, p = paired_ttest_5x2cv(estimator1=model1, estimator2=model2, X=X, y=y, scoring='
    accuracy', random_seed=1) #Initialisation du couple de valeurs t et
    p
print('P-value: %.3f, t-Statistic: %.3f' % (p, t)) #Affichage de la p-Value et de
    t-Statistic initialisé ci-dessus
if p <= 0.05: #test de la valeur de la variable p
    print('Difference between mean performance is probably real') #Affichage si p<=0.05
else:
    print('Algorithms probably have the same performance') #Affichage si p>0.05
```

Sortie possible :

```
P-value: 0.384, t-Statistic: -0.953
Algorithms probably have the same performance
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-packages/numpy/core/_asarray.py:136: ComplexWarning: Casting complex values to real discards the imaginary part
  return array(a, dtype, copy=False, order=order, subok=True)
```

Figure 26: Sortie 15

5.16 Bloc 16 : discriminant

Analyse du discriminant

```
X = standardize(X) #Lissage de la variable X
lda = ldaf(n_discriminants=2) #Initialisation du discriminant
lda.fit(X, y) #Entrainement du modèle en utilisant l'ensemble de formation
X_lda = lda.transform(X) #Transforme les valeurs pour qu'elles soient utilisables par les
    fonctions d'après
plt.figure(figsize=(6, 4)) #Création d'une figure avec une certaine taille précisée en
    argument
for lab, col in zip((0, 1),('blue', 'red')): #Boucle pour tracer le nuage de points en
    fonction de la ligne et de la colonne
    plt.scatter(X_lda[y == lab, 0],X_lda[y == lab, 1],label=lab,c=col) #Traçage du nuage
    de points
plt.xlabel('Linear Discriminant 1') #Paramètrage du titre de l'axe X
plt.ylabel('Linear Discriminant 2') #Paramètrage du titre de l'axe Y
plt.legend(loc='lower right') #Paramètrage de la légende située en bas à droite
plt.tight_layout() #Ajustement des bordures entre et autour des sous traces
plt.show() #Affichage du graphique/figure
```

Sortie possible :

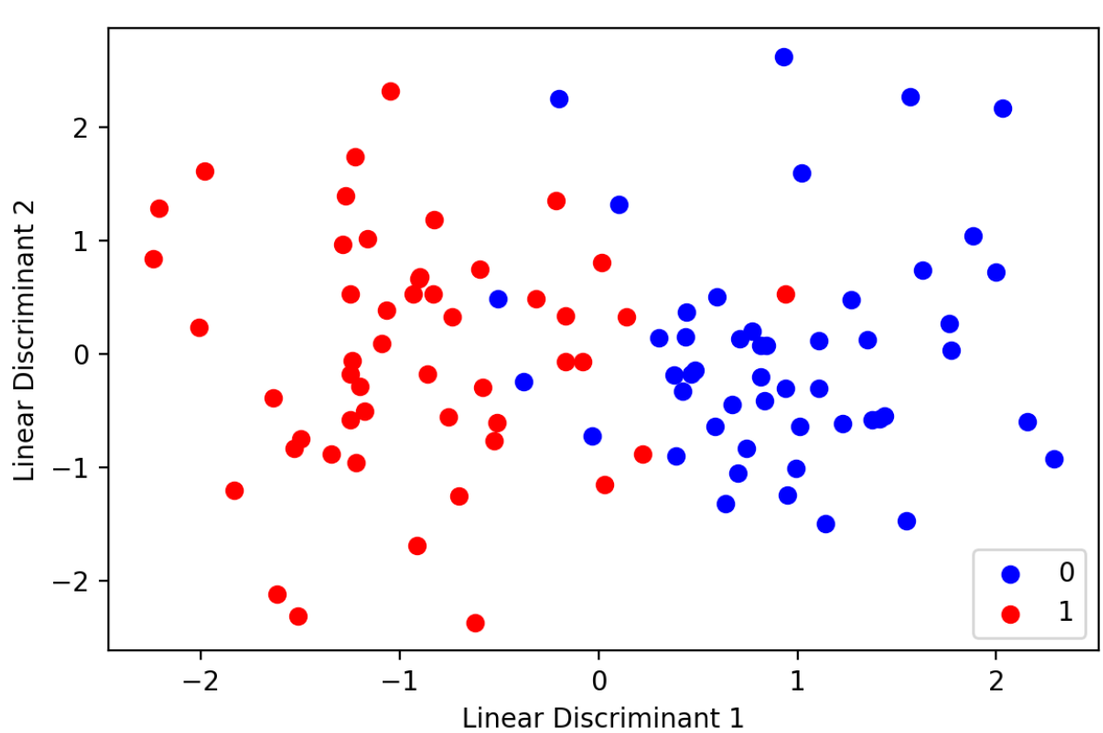


Figure 27: Sortie 16

6 Les fonctions

Dans cette partie, nous allons détailler les paramètres pris par chacune des fonctions de chacune des bibliothèques.

6.1 pandas

6.1.1 pandas.read_csv('filepath')

Lecture d'un fichier csv contenant des valeurs séparées par des virgules dans un DataFrame.

Paramètre

filepath : str, path object
N'importe quel chemin d'accés (chemin absolu ou chemin relatif (répertoire courant)) ou même URL est accepté.

6.1.2 pandas.DataFrame.sample(n=None)

Retourne un échantillon aléatoire d'éléments à partir d'un DataFrame (axe d'objets).

Paramètre

n : int
Nombre d'éléments à retourner. Valeur par défaut est 1.

6.1.3 pandas.DataFrame.shape

Retourne un tuple représentant la dimension du DataFrame.

6.1.4 pandas.DataFrame.describe(percentiles=None)

Retourne la valeur de chaque quartile.

Paramètre

percentiles : liste de nombres
Les quartiles à inclure dans la sortie. Les valeurs doivent être comprises entre 0 et 1. La valeur par défaut `[0.25, 0.5, 0.75]` représentant anssi le premier quartile, la médiane et le troisième quartile.

6.1.5 pandas.DataFrame.info()

Affichage des informations sur le DataFrame.

Paramètre

data : DataFrame
Affichage des informations à propos de ce DataFrame.

6.1.6 pandas.DataFrame.plot(x=None, y=None, style=None)

Création d'un tracé de DataFrame.

Paramètres

data : DataFrame
L'objet pour laquel la méthode est appelée.
x : str, étiquette
Étiquette de l'axe des abscisses. Utilisé seulement si c'est un DataFrame en data.
y : str, étiquette
Étiquette de l'axe des ordonnées. Utilisé seulemnt si c'est un DataFrame en data.
style : str
Style de l'affichage des points sur le graphique, par défaut tous les points sont reliés.
'.' : point, marqueur point.
'o' : point cercle, marqueur circulaire.
'v' : point triangle bas, marqueur triangulaire bas.
'<' : point triangle gauche, marqueur triangulaire gauche.
'>' : point triangle droit, marqueur triangulaire droit.
's' : point carré, marqueur carré.
kind : str, étiquette
Le type de graphique à afficher.
'line' : graphique à tracé de ligne (default).
'bar' : graphique à barre verticale.
figsize : (int, int)
Taille de la figure.
largeur, hauteur en inch.

6.1.7 pandas.DataFrame.values()

Retourne une représentation Numpy de la DataFrame.

6.1.8 pandas.DataFrame.sample(n=None)

Retourne un échantillon de n éléments.

Paramètre

n : int
Nombre d'items à retourner.

6.1.9 pandas.DataFrame.head(n=5)

Retourne les n premiers éléments.

Paramètre

n : int
Nombre d'éléments à retourner.

6.1.10 pandas.DataFrame(data=None)

Retourne un tableau à deux dimensions, de taille variable (DataFrame).

Paramètre

data : ndarray, DataFrame, dict
Les données pour former le tableau (DataFrame).

6.2 matplotlib

6.2.1 matplotlib.pyplot.title(label)

Paramètre le titre du graphique.

Paramètre

label : str
Texte utilisé pour le titre.

6.2.2 matplotlib.pyplot.xlabel(label)

Paramètre le titre de l'axe des abscisses du graphique.

Paramètre

label : str
Texte utilisé pour le titre de l'axe.

6.2.3 matplotlib.pyplot.ylabel(label)

Paramètre le titre de l'axe des ordonnées du graphique.

Paramètre

label : str
Texte utilisé pour le titre de l'axe.

6.2.4 matplotlib.pyplot.show(*, block=None)

Affichage de toutes les figures.

Paramètre

block : bool
par défaut à True.
Si c'est True, la boucle prinicipale est stoppée jusqu'à ce que toutes les fenêtres soient fermées.
Si c'est False, toutes les fenêtres sont ouvertes immédiatement.

6.2.5 matplotlib.pyplot.figure(num=None,figsize=None)

Création d'une nouvelle figure.

Paramètres

figsize : (float,float)
par défaut (6.4, 4.8).
largeur, hauteur en inch.
figsize : int
identifiant unique de la figure.

6.2.6 matplotlib.pyplot.tight_layout(pad=1.08)

Ajustement des bordures entre et autour des sous tracés.

Paramètre

pad : float
par défaut 1.08.
Espace entre le bord de la figure et le bord du tracé.

6.2.7 matplotlib.pyplot.grid(which='major',**kwargs)

Définir les propriétés des lignes du graphique.

Paramètres

label : 'major', 'minor', 'both'
Les lignes de la grille sur lesquelles appliquer les changements.
**kwargs : Toutes les propriétés des Line2D de matplotlib
linestyle : '-', '–', '-.', ':', ''
linewidth : float
color : la couleur

6.2.8 matplotlib.pyplot.scatter(x,y,c=None,**kwargs)

Diagramme de dispersion de y en fonction de x avec des couleurs différentes.

Paramètres

x : float ou forme de tableau
Position des données en x.
y : float ou forme de tableau
Position des données en y.
c : forme de tableau ou liste de couleur
La couleur des points sur le graphique.
**kargs : Propriétés supplémentaires
color : la couleur.
label : list ou str, les étiquettes pour les élèments légendés.

6.2.9 matplotlib.pyplot.plot(x,y, **kwargs)

Graphique de y en fonction de x.

Paramètres

x : forme de tableau
Coordonnées horizontales.
y : forme de tableau
Coordonnées verticales.
**kwargs : Toutes les propriétés des Line2D de matplotlib
color : la couleur.

6.2.10 matplotlib.pyplot.hist(x, bins=None, histtype='bar')

Graphique en histogramme.

Paramètres

x : forme de tableau
Les valeurs d'entrée.
bins : int
Définition du nombre de bandes de même largeur sur le graphique.
histtype : 'bar', 'barstacked', 'step', 'stepfilled', default: 'bar'
Type d'histogramme à tracer.

6.2.11 matplotlib.pyplot.box(x, labels=None, showmeans=None)

Graphique en boîte à moustaches.

Paramètres

x : forme de tableau
Les valeurs d'entrée.
labels : sequence
Étiquette pour chaque ensemble de données.
showmeans : bool, default False
Moyenne arthimétique.

6.3 seaborn

6.3.1 searborn.displot(data=None)

Ajustement des bordures entre et autour des sous tracés.

Paramètre

data : pandas.DataFrame, numpy.ndarray
Structure des données d'entrée.

6.3.2 matplotlib.pyplot.grid(*args,**kwargs)

Placer une légende sur les axes.

Paramètre

**kwargs : Toutes les propriétés sur les légendes
loc : str, position de la légende.

6.4 sklearn

6.4.1 sklearn.model_selection.train_test_split(*arrays, test_size=None, random_state=None)

Diviser la matrice ou le tableau en sous-ensemble d'entrainement et de test.

Paramètres

*arrays : séquence d'indexables avec la même longueur.
Les entrées autorisées sont les listes, les tableaux numérisés, les matrices scipy-sparse ou les cadres de données pandas.
test_size : float ou int
S'il est float, il doit être compris entre 0,0 et 1,0 et représenter la proportion de l'ensemble de données à inclure dans la répartition de l'essai.
Si int, représente le nombre absolu d'échantillons à tester. Si none, la valeur est fixée au complément de la taille du train.
random_state : float ou int
Contrôle le brassage appliqué aux données avant d'appliquer le fractionnement.
Passe un int pour une sortie reproductible sur plusieurs appels de fonction.

6.4.2 sklearn.model.linear_model.LinearRegression().fit(X,Y)

Entrainement du modèle linéaire.

Paramètres

X : forme de tableau
Les données de formation.
Y : forme de tableau
Les valeurs cibles.

6.4.3 sklearn.model.linear_model.LinearRegression().fit().intercept_

Retourne l'intersection du modèle.

6.4.4 sklearn.model.linear_model.LinearRegression().fit().coef_

Retourne le coefficient directement de la droite de régression.

6.4.5 sklearn.model.linear_model.LinearRegression().fit().predict(X)

Prévoir en fonction du modéle linéaire.

Paramètre

X : forme de tableau
Un échantillon de valeurs.

6.4.6 sklearn.metrics.mean_absolute_error(y_true, y_pred)

Perte moyenne par régression d'erreur absolue.

Paramètres

y_true : forme de tableau
Valeur cible de base.
y_pred : forme de tableau
Valeurs cibles estimées.

6.4.7 sklearn.metrics.mean_squared_error(y_true, y_pred)

Perte de régression de l'erreur au carré moyenne.

Paramètres

y_true : forme de tableau
Valeur cible de base.
y_pred : forme de tableau
Valeurs cibles estimées.

6.4.8 sklearn.datasets.make_classification(n_samples=100, n_features=20, n_informative=2, n_redundant=2, random_state=None)

Générer un système de classification aléatoire de classe n.

Paramètres

n_samples : int
Nombre d'échantillons.
n_features : int
Le nombre total d'éléments.
n_informative : int
Nombre d'éléments d'information.
n_redundant : int
Nombre d'éléments redondants.
random_state : int
Détermine la génération de nombres aléatoires pour la création d'un ensemble de données.

6.4.9 sklearn.model_selection.RepeatedStratifiedKFold(n_splits=5, n_repeats=10, random_state=None)

Paramètres

n_splits : int
Nombre de plis.
n_repeats : int
Nombre de fois que le validateur croisé doit être répété.
random_state : int
Contrôle la génération des états aléatoires pour chaque répétition.

6.4.10 sklearn.model_selection.cross_val_score(estimator, X, y=None, scoring=None, cv=None, n_jobs=None)

Paramètres

estimator : objet estimateur mettant en œuvre l'ajustement
L'objet à utiliser pour adapter les données.
X : forme de tableau
Les données à adapter.
y : forme de tableau
La variable cible à essayer de prévoir dans le cas d'un apprentissage supervisé.
scoring : str
Pointeur objet.
cv : int
Détermine la stratégie de fractionnement de la validation croisée.
Spécifie le nombre de plis dans le `(Stratified)KFold`.
n_jobs : int
Nombre de tâches à gérer en arrière plan/en parallèle.

6.5 numpy

6.5.1 numpy.sqrt(x)

Retourne racine carré non négative d'un tableau par élément.

Paramètre

x : forme de tableau, float, int
Valeur sous la racine carré qui est requise.

6.5.2 numpy.mean(a)

Retourne la moyenne du tableau.

Paramètre

a : forme de tableau
Tableau contenant les valeurs dont la moyenne est souhaitée.

6.5.3 numpy.std(a)

Retourne l'écart-type d'un tableau.

Paramètre

a : forme de tableau
Tableau contenant les valeurs dont l'écart-type est souhaité.

6.6 scipy

6.6.1 scipy.stats.shapiro(x)

Réalisation du test de Shapiro-Wilk sur un tableau de données. Retourne la statistique du test (float) et la p-value (float).

Paramètre

x : forme de tableau, float, int
Tableau de données pour faire le test.

6.6.2 scipy.stats.probplot(x, plot=None)

Calculer des quartiles pour une courbe de probabilité (par défaut de probabilité normale), et éventuellement tracer la courbe.

Paramètres

x : forme de tableau, float, int
Tableau de données pour faire le test.
plot : objet
Tableau de données pour faire le test.

6.7 mlxtend

6.7.1 mlxtend.evaluate.paired_ttest_5x2cv(estimator1, estimator2, X, y, scoring=None, random_seed=None)

Met en œuvre le test t jumelé 5x2cv proposé par Dieterrich (1998) pour comparer les performances de deux modèles. Retourne la statistique t et la valeur p.

Paramètres

estimator1 : régresseur
estimator2 : régresseur
X : forme de tableau
Les valeurs d'entrainement.
y : forme de tableau
Les valeurs cibles.
scoring : str, accuracy, f1, precision, recall, roc_auc
sklearn scoring identificateur de chaîne de caractères métriques.
random_seed : int
Nombre aléatoire pour la création des séquences d'essai/train.

6.7.2 mlxtend.preprocessing.standardize(array)

Normaliser les colonnes dans les Pandas DataFrame (Tableau).

Paramètre

array : forme de tableau
Pandas DataFrame.

6.7.3 mlxtend.feature_extraction.LinearDiscriminantAnalysis(n_discriminants=None)

Classe d'analyse du discriminant linéaire.

Paramètre

n_discriminants : int
Le nombre de discriminant pour la transformation.

6.7.4 mlxtend.feature_extraction.LinearDiscriminantAnalysis().fit(X,y)

Classe d'analyse du discriminant linéaire.

Paramètres

X : forme de tableau
Les valeurs d'entrainement.
y : forme de tableau
Les valeurs cibles.

6.7.5 mlxtend.feature_extraction.LinearDiscriminantAnalysis().transform(X)

Application de la transformation linéaire sur X.

Paramètre

X : forme de tableau
Les valeurs d'entrainement.

7 Le programme

7.1 Lancement

Pour lancer le programme, il suffit de se placer dans le dossier où se trouve le fichier `main.py`, ainsi vous pourrez lancer le programme avec la commande suivante :

```
python3 main.py
```

Le programme affichera des informations dans le Terminal et aussi grâce à des fenêtres créées par matplotlib. Pour avancer dans le programme, il suffit de fermer chaque fenêtre des représentations qui apparaissent à l'écran.

7.2 Exemples de lancement du programme

Vous trouverez pour chaque système d'exploitation une vidéo de présentation du programme lors de son fonctionnement :

- MacOS
- Linux (Ubuntu)
- Windows 10

8 Sources

Retrouver les documentations officielles des librairies utilisées dans ce projet :

- pandas
- matplotlib
- searborn
- numpy
- mlxtend

9 Crédit

Membres de l'équipe :

- Damien Briquet
- Alexis Brunet

Veuillez retouver aussi toute la documentation sur notre site et sur notre GitHub.
INSA CVL | 3A MRI - TD3 - TP5

© 2020-2021