INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES CENTRE VAL DE LOIRE

Manuel d'utilisation

Projet Python 3A

Damien Briquet et Alexis Brunet

Contents

1 Objectifs

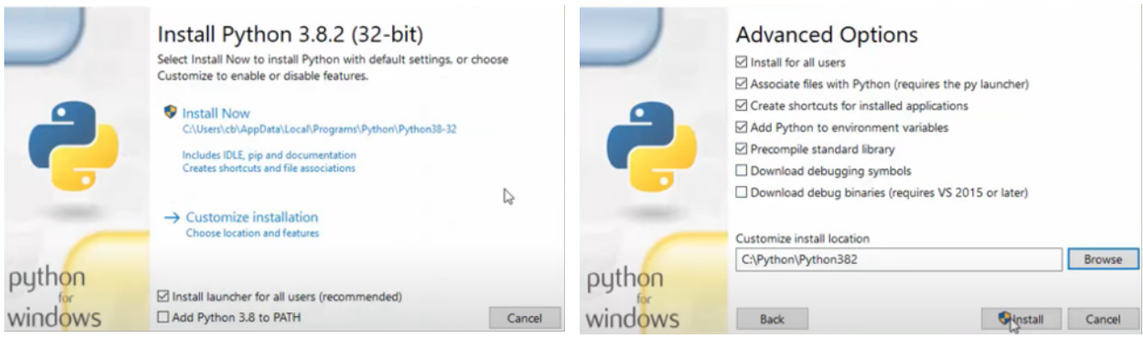
Python est un langage de programmation très populaire dans le monde informatique pour réaliser des tâches variées tel que l'automatisation, les sites internets, l'Intelligence Artificielle (IA), etc. Notre objectif est de comprendre en détail le code donné mais aussi le simplifier, en gardant uniquement 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 exe 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

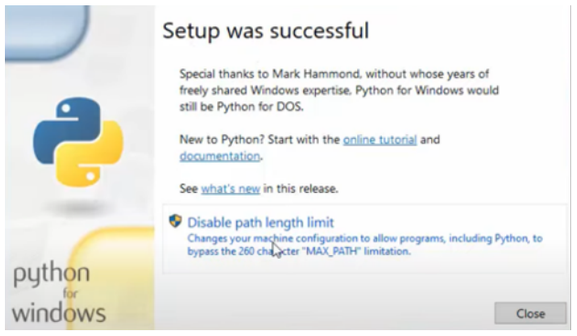


Figure 2: Fin de l'installation Python

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 :

```
python --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 ett tapez la commande suivante :

```
python --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 :

```
python --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 :

```
python --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 précèdemment. Suivez les instructions à l'écran pour l'installer.

Une fois installé vous pouvez le lancer et commencer à coder.

3.1.2 Linux - Ubunt, 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.

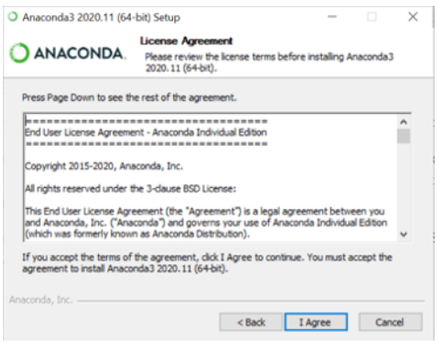
4.1 Windows

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

4.1.1 Méthode 1 : Anaconda

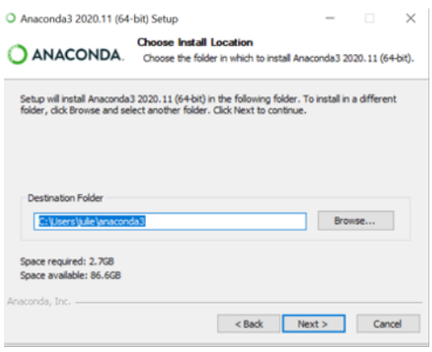
Cette méthode est plus simple, car vous allez installer Anaconda qui installera une partie des autres dépendances (Numpy, Pandas) nécessaires au projet. Téléchargez la dernière version de Anaconda pour Windows sur le site. Exécutez le fichier `.exe`, suivez les instructions qui apparaissent à l'écran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

(a) Anaconda - Étape 1

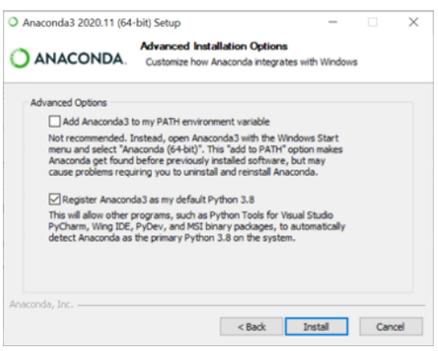


(b) Anaconda - Étape 2

Figure 3: Insatallation d'Anaconda

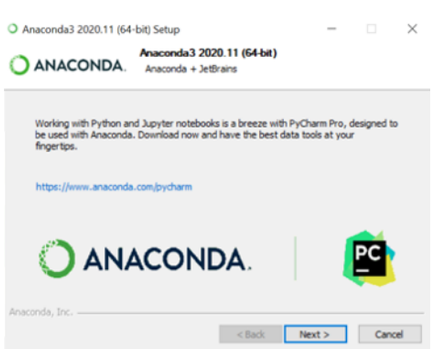


(a) Anaconda - Étape 3

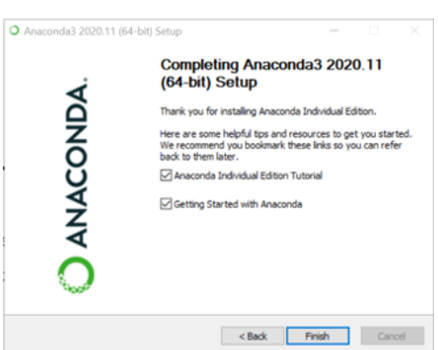


(b) Anaconda - Étape 4

Figure 4: Insatallation d'Anaconda



(a) Anaconda - Étape 5



(b) Anaconda - Étape 6

Figure 5: 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
```

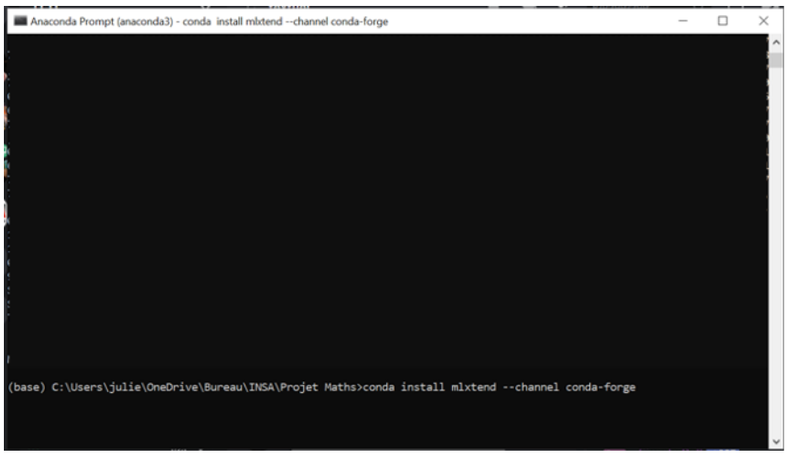


Figure 6: Anaconda - mlxtend - Étape 1

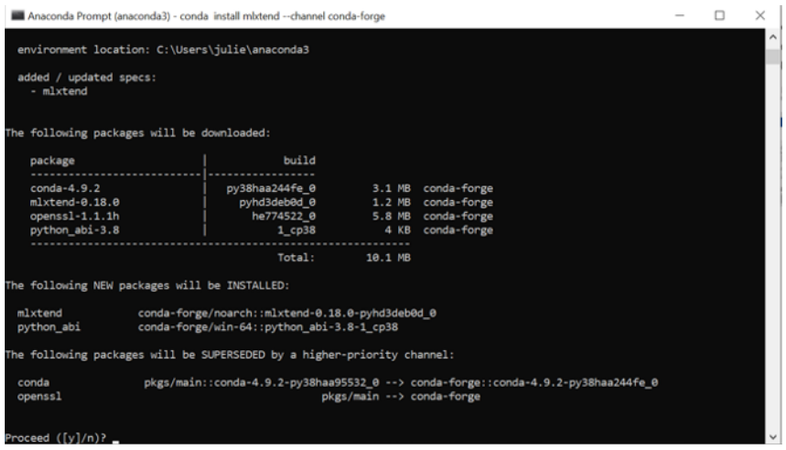


Figure 7: Anaconda - mlxtend - Étape 2

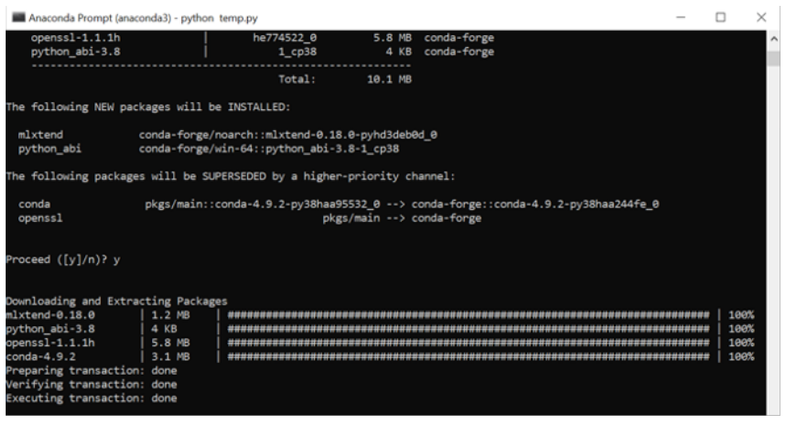


Figure 8: Anaconda - mxltend - Etape 3

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` Et voila ! 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 Python-Project-INSA-3A-v0.1.1.zip https://github.com/AlexTheGeek/Python-
    Project-INSA-3A/archive/v0.1.1.zip
```

Ouvrez un Terminal et tapez les commandes suivantes :

```
unzip Python-Project-INSA-3A-v0.1.1.zip
cd Project-Python-3A-MRI
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 `pip3`.

```
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 Python-Project-INSA-3A-v0.1.1.zip https://github.com/AlexTheGeek/Python-
    Project-INSA-3A/archive/v0.1.1.zip
```

Ouvrez un Terminal et tapez les commandes suivantes :

```
unzip Python-Project-INSA-3A-v0.1.1.zip
cd Project-Python-3A-MRI
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 programmes, par des blocs de lignes de code. Pour présenterons ainsi une sortie possible grâce à ces lignes. Vous pouvez aussi retrouver le code complet annoté à 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 utilisé dans main.py

5.1 Bloc 1 : initialisation d'un DataFram 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 cet DataFrame (dimension, statistique, utilisatioon mémoire, valeurs non nulle, colonnes, index dtype, ...).

```
dataset1 = pd.read_csv('weather.csv') #Lecture d'un fichier (csv) de valeurs separees par
    des virgules dans un DataFrame
dataset1.sample() #Generation d'un echantillon aleatoires de chaque groupe d'un objet du
    DataFrame
dataset=dataset1.sample(1000) #Un nouvel DataFrame (dataset) contenenant 1000 elements
    echantillonnes de facon aleatoire a partir de dataset1
print(dataset.shape) #Affichage d'un tuple representant la dimension du DataFrame dataset
print(dataset.describe()) #Affichage des statistiques pour chaque type de valeurs du
    DataFrame
print(dataset.info()) #Affichage des informations sur le DataFrame, notamment l'index
    dtype et les colonnnes et les valeurs non nulles et l'utilisation de la memoire
```

Sortie possible :

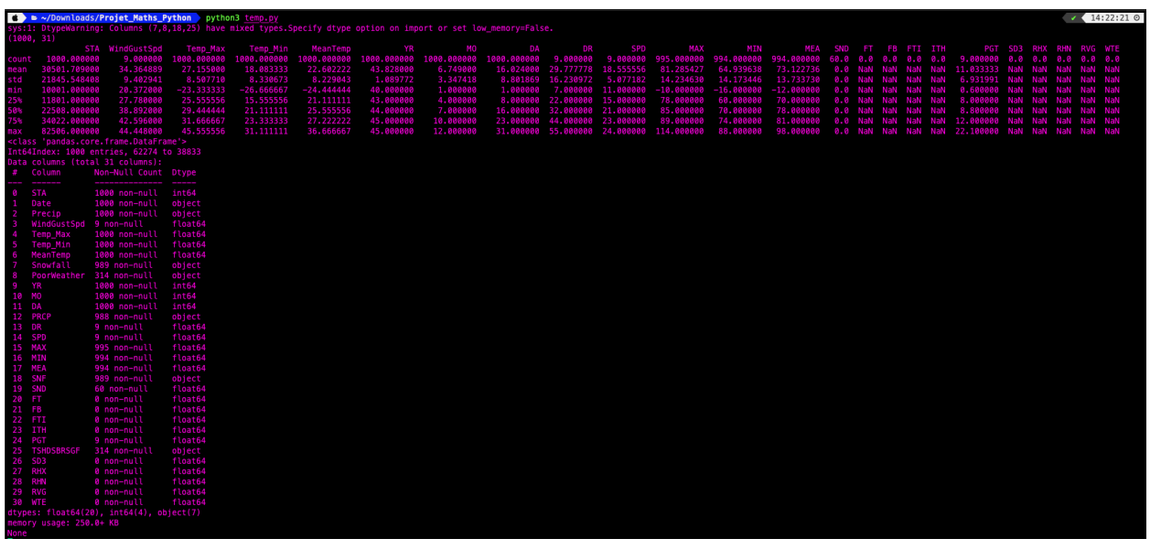


Figure 9: Sortie 1

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

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

```
dataset.plot(x='Temp_Min', y='Temp_Max', style='o') #Création d'un trace de DataFrame,
    avec l'axe x réprésentant les Temp min et l'axe y réprésentant les Temp Max, dans un
    style de nuge de points
plt.title('Temp_Min vs Temp_Max') #Parametrage du titre du graphique
plt.xlabel('Temp_Min') #Parametrage du titre de l'axe x
plt.ylabel('Temp_Max') #Parametrage du titre de l'axe y
plt.show() #Affichage du graphique/figure
```

Sortie possible :

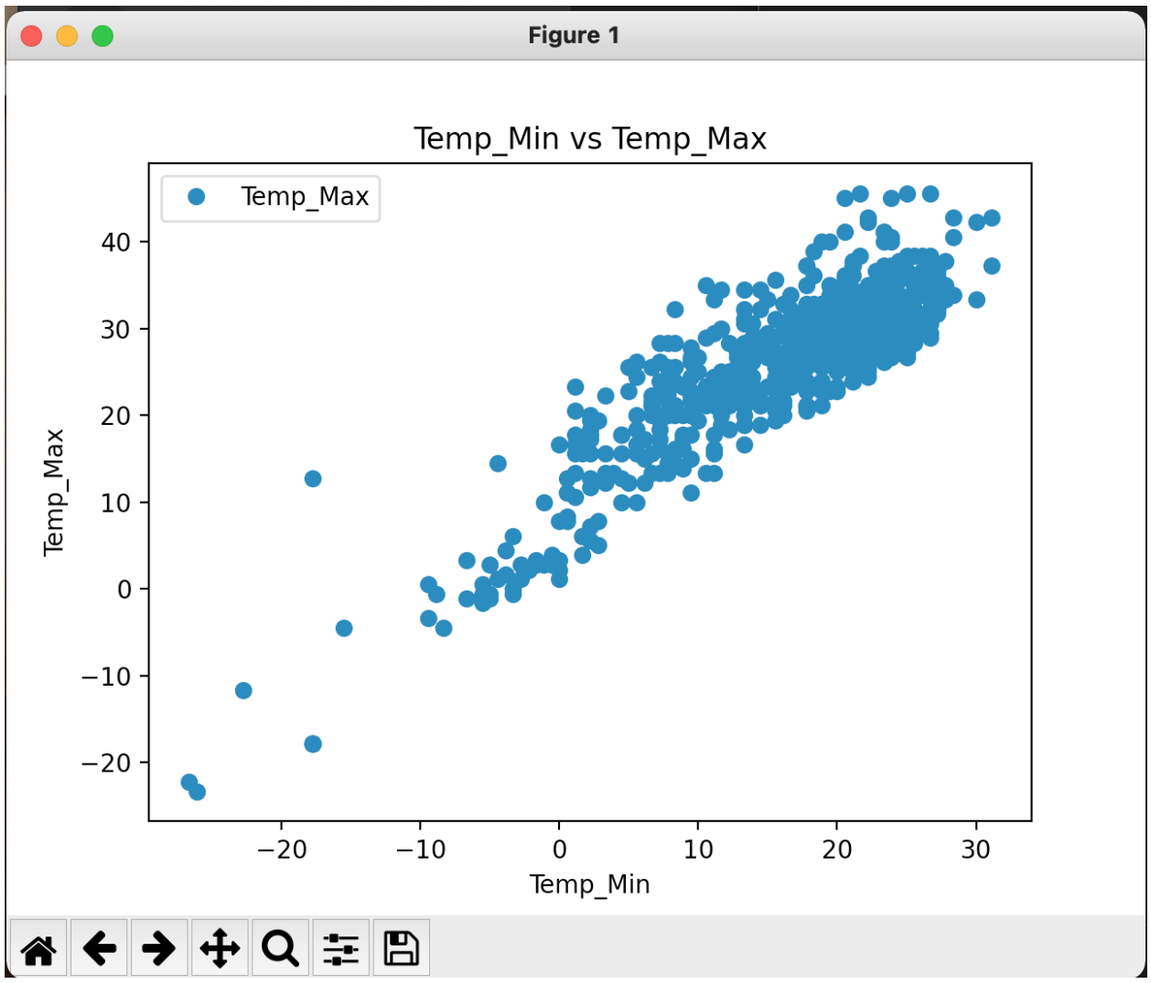


Figure 10: Sortie 2

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

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

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

Sortie possible :

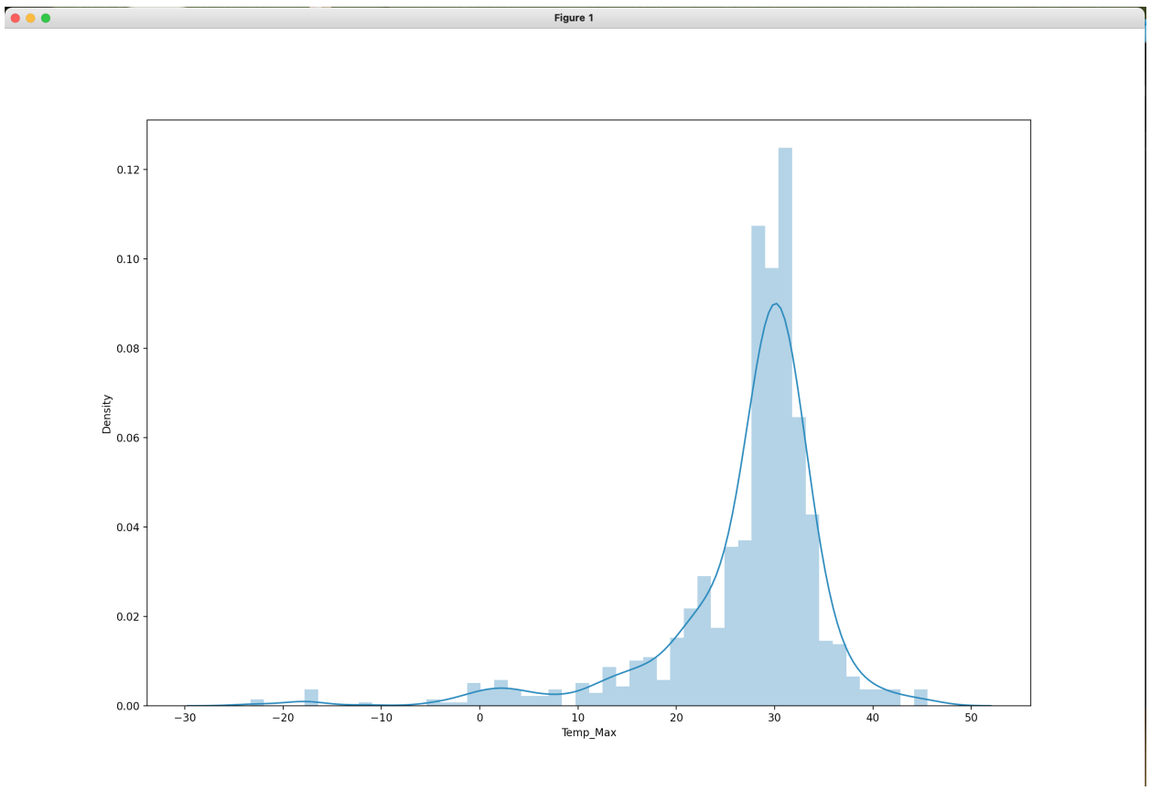


Figure 11: Sortie 3

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 includ 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

regressor = LinearRegression() #Creation un objet de regression lineaire

regressor.fit(X_train, y_train) #Entrainement du model en utilisant l'ensemble de
    formation
print(regressor.intercept_) #Affichage de l'intersection
print(regressor.coef_) #Affichage du coefficient directeur de la droite de regression
y_pred = regressor.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 prédictes et de
    test
print(df) #Affichage de DataFrame
```

Sortie possible :

```
  warnings.warn(msg, FutureWarning)
[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 12: 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 répartitions des températures mesurées et prédites.

```
df1 = df.head(25) #Récupération des 25 dernières 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 (makor) du graphique avec un style de trait plein, epaisseur 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 point, épaisseur de 0.5points, une
    couleur noire
plt.show() #Affichage du graphique/figure
```

Sortie possible :

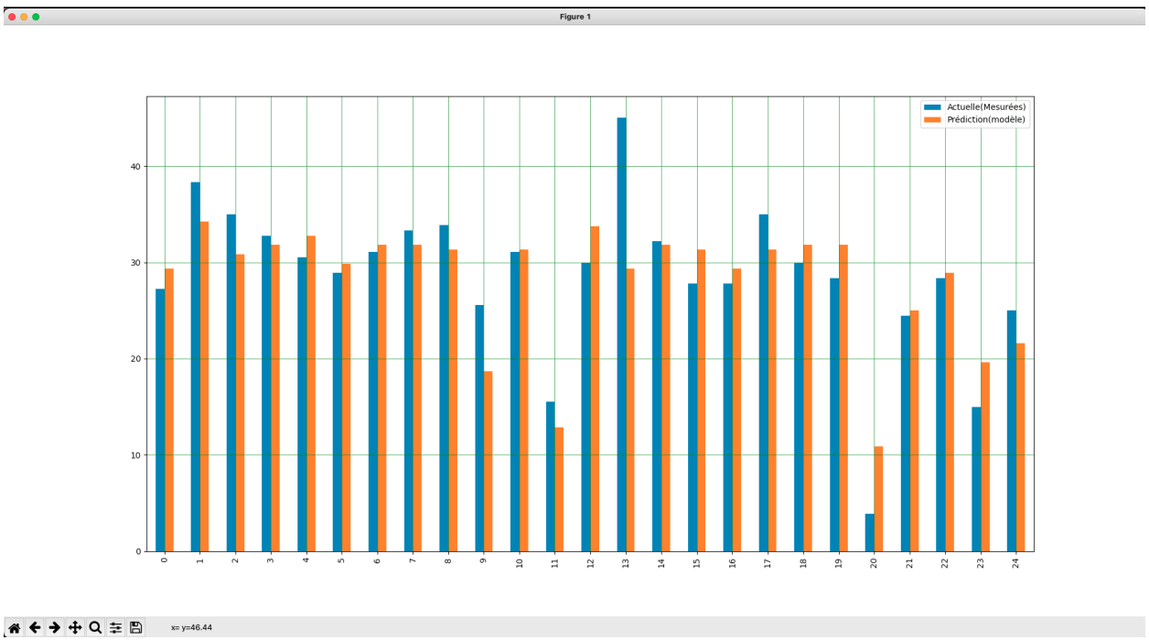


Figure 13: Sortie 5

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
    contre X test de couleur grise
plt.plot(X_test, y_pred, color='red', linewidth=2) #Création d'un trace (de la fonction
    ax+y) de DataFrame, avec l'axe x réprésentant les X Test et l'axe y réprésentant les
    y pred, dans la couleur rouge et de largeur 2points
plt.show() #Affichage du graphique/figure
```

Sortie possible :

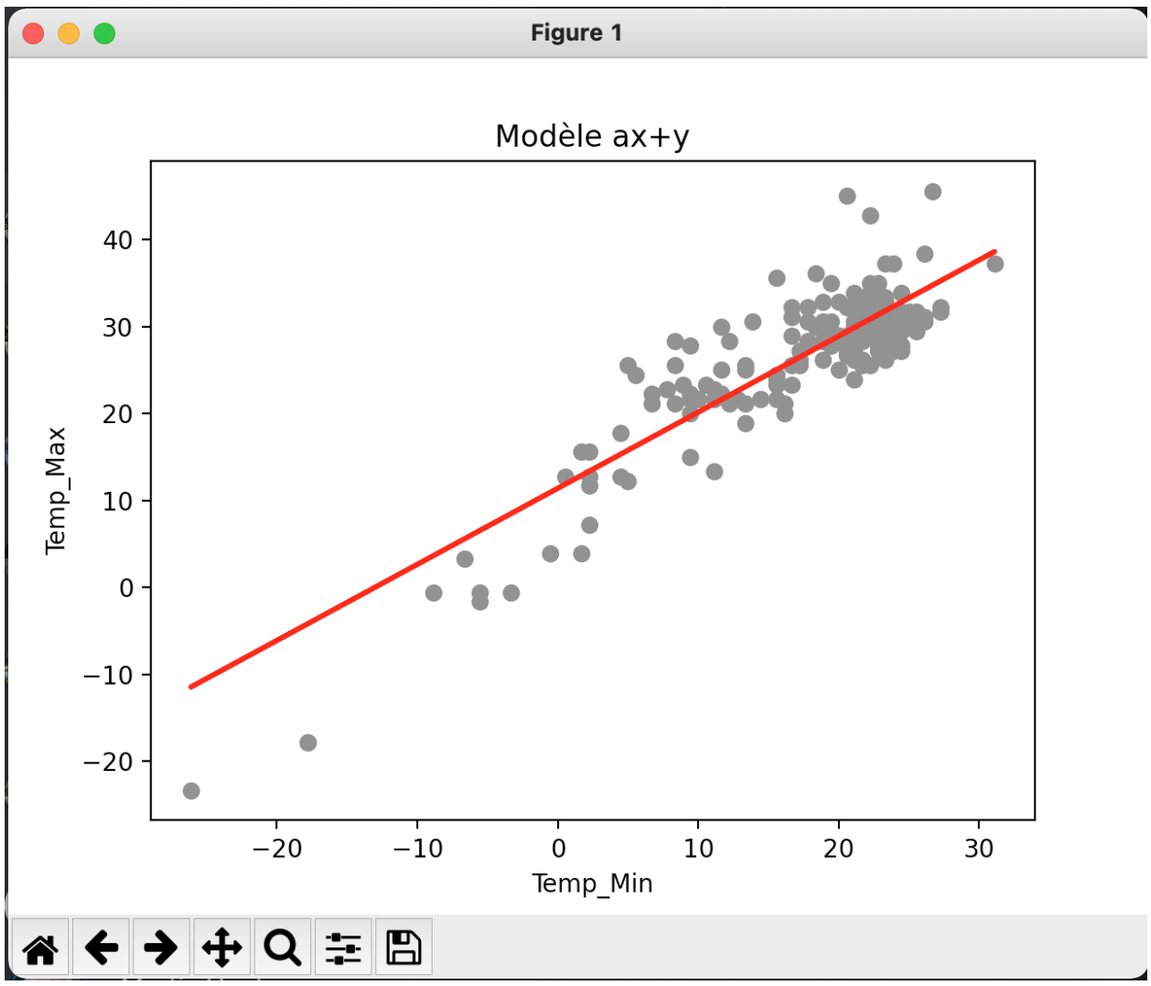


Figure 14: Sortie 6

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 15: 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 précèdemment
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 des 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
    des Temp Min
```

5.8.1 Bloc de la fonction associé

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 parametre 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 a numTrials-1 par pas
    de 1
        Y=data.sample(sampleSize) #Y récupère un échantillon aléatoire d'éléments à
    partir de SampleSize
        Means.append(sum(Y)/len(Y)) #Ajout dans le tabelau de la division de la somme de
    Y divise par la longuer de Y
    return Means  #On retourne le tableau Means
```

Sortie possible :

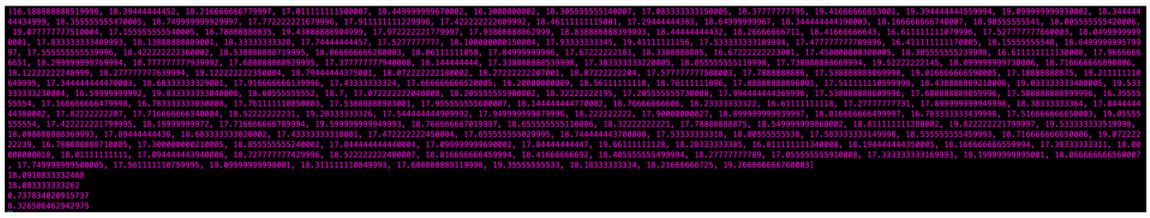


Figure 16: Sortie 8

5.9 Bloc 9 : diagramme en escalier

Création d'un histogramme en escalier.

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

Sortie possible :

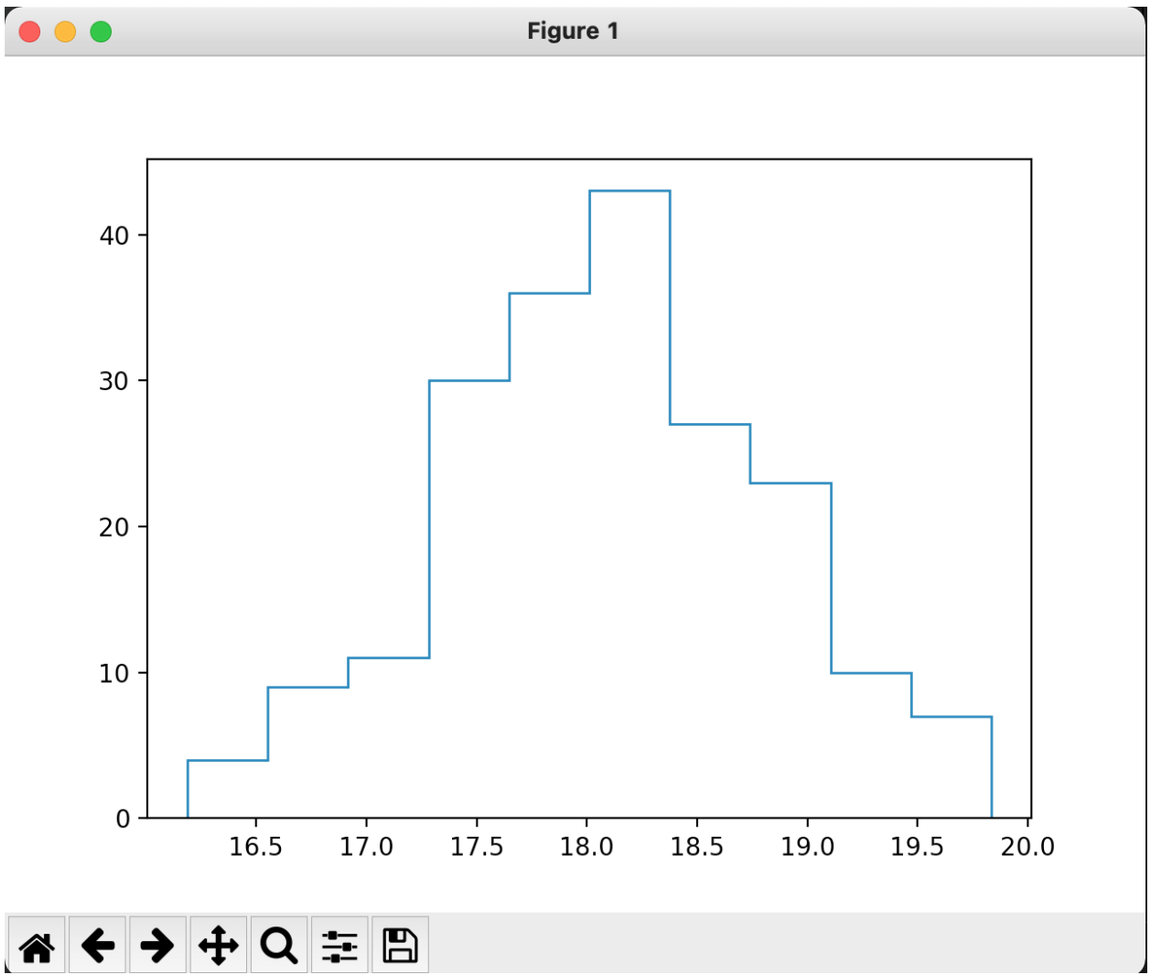


Figure 17: Sortie 9

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'hypothese
    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 #Iniatialisation 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 18: Sortie 10

5.11 Bloc 11 : boite à moustache

Diagramme en boites et en moutaches. 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 en moutaches de Means
plt.show() #Affichage du graphique/figure
```

Sortie possible :

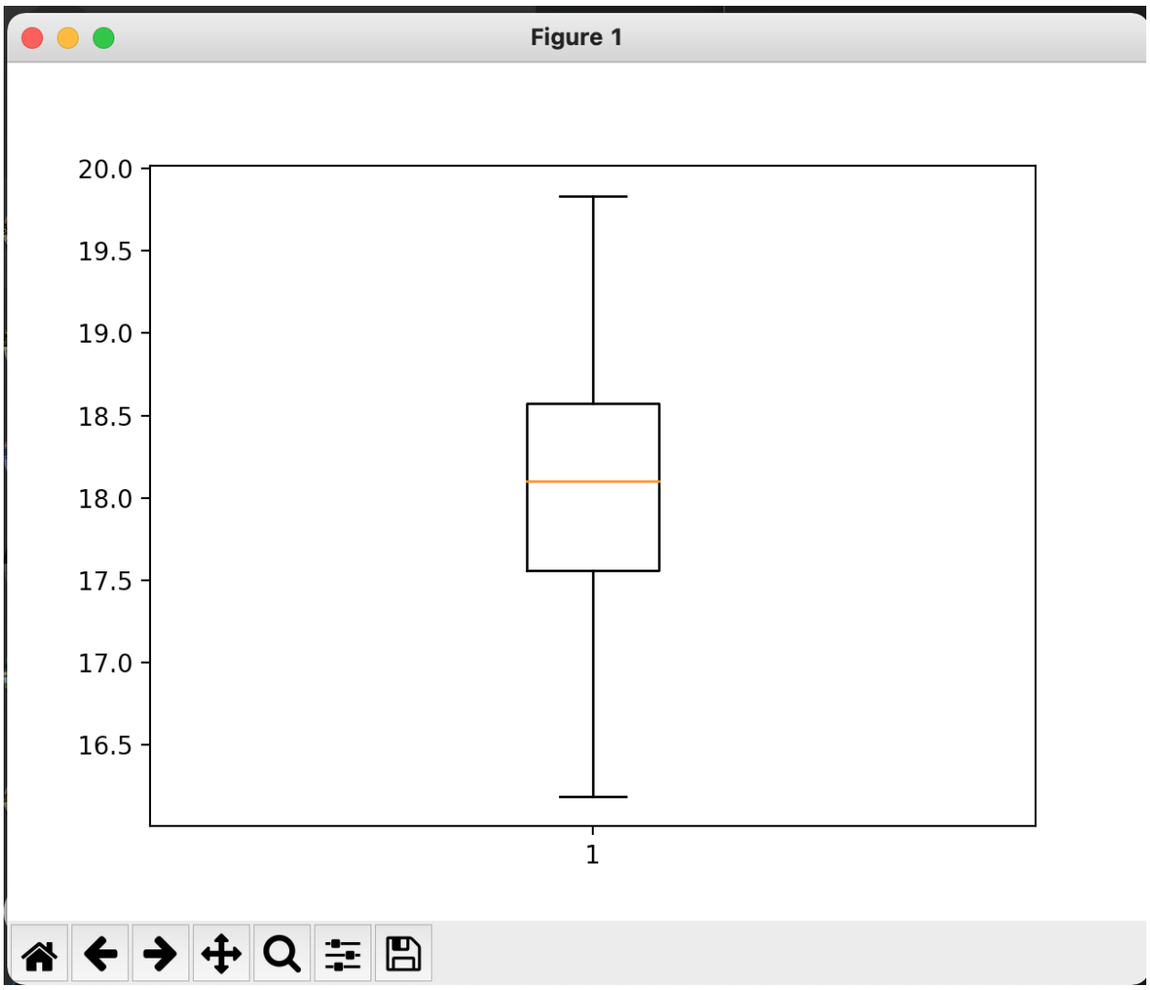


Figure 19: Sortie 11

5.12 Bloc 12 : test de Shapiro-Wilk

Création et affichage de la courbe de probabilité

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

Sortie possible :

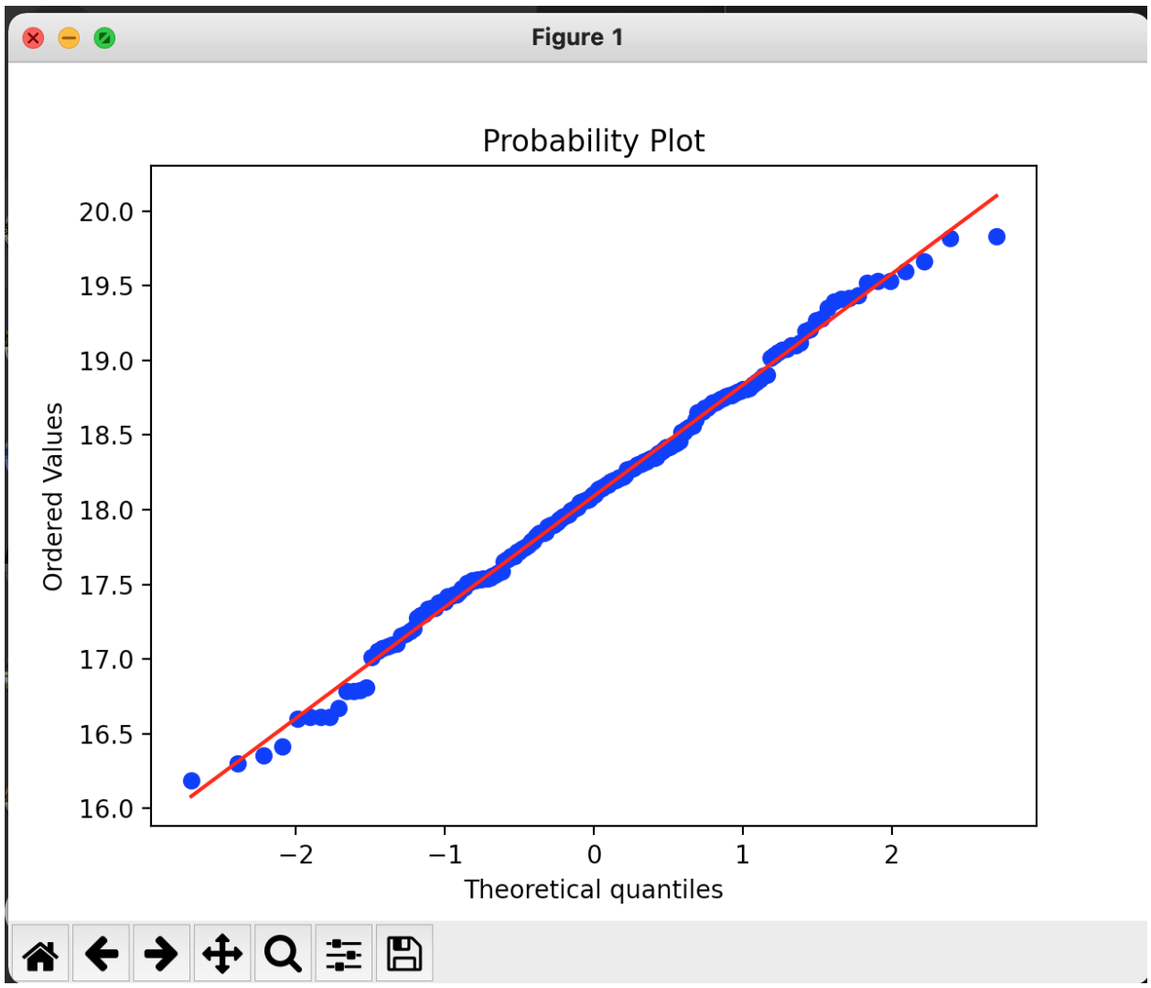


Figure 20: Sortie 12

5.13 Bloc 13 : coefficient de regression

Ecriture par deux manières différentes du calcul du coefficient de regression 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 regression 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 regression linéaire moyen pour le modèle
    2
```

Sortie possible :

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

Figure 21: 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) #Creation d'un
    diagramme boite à moustaches à partir de scores1 et scores2
plt.show() #Affichage du graphique/figure
```

Sortie possible :

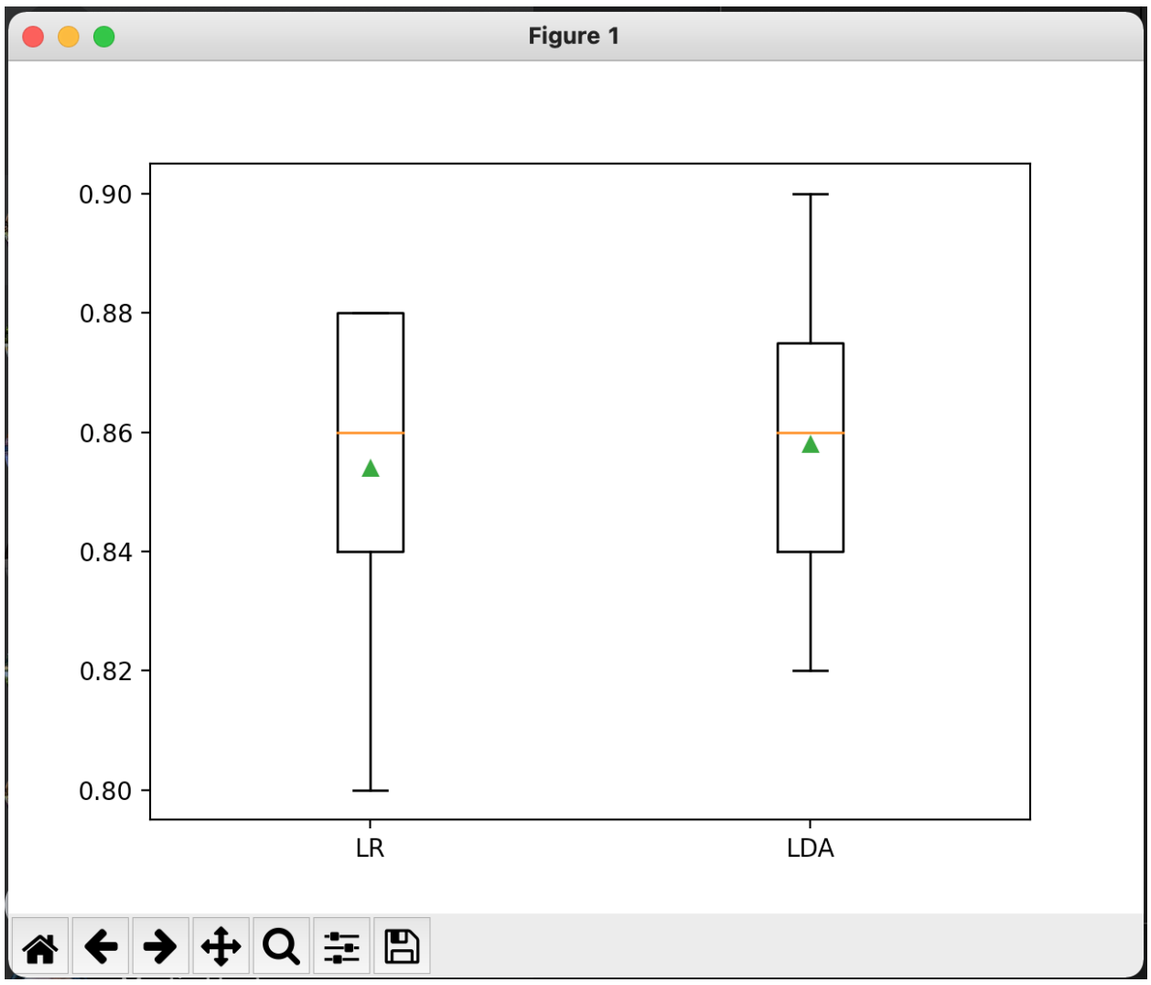


Figure 22: Sortie 14

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 valeur 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 23: Sortie 15

5.16 Bloc 16 : discriminant

Analyse du discriminant

```
X = standardize(X) #Lissage de la variable X
lda = lda(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 utilisable par les
    fonctions d'après
plt.figure(figsize=(6, 4)) #Creation d'un figure avec une certaine taille précisé en
    argument
for lab, col in zip((0, 1),('blue', 'red')): #boucle pour tracer le nuage de point 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) #tracage du nuage
    de point
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 les sous traces
plt.show() #Affichage du graphique/figure
```

Sortie possible :

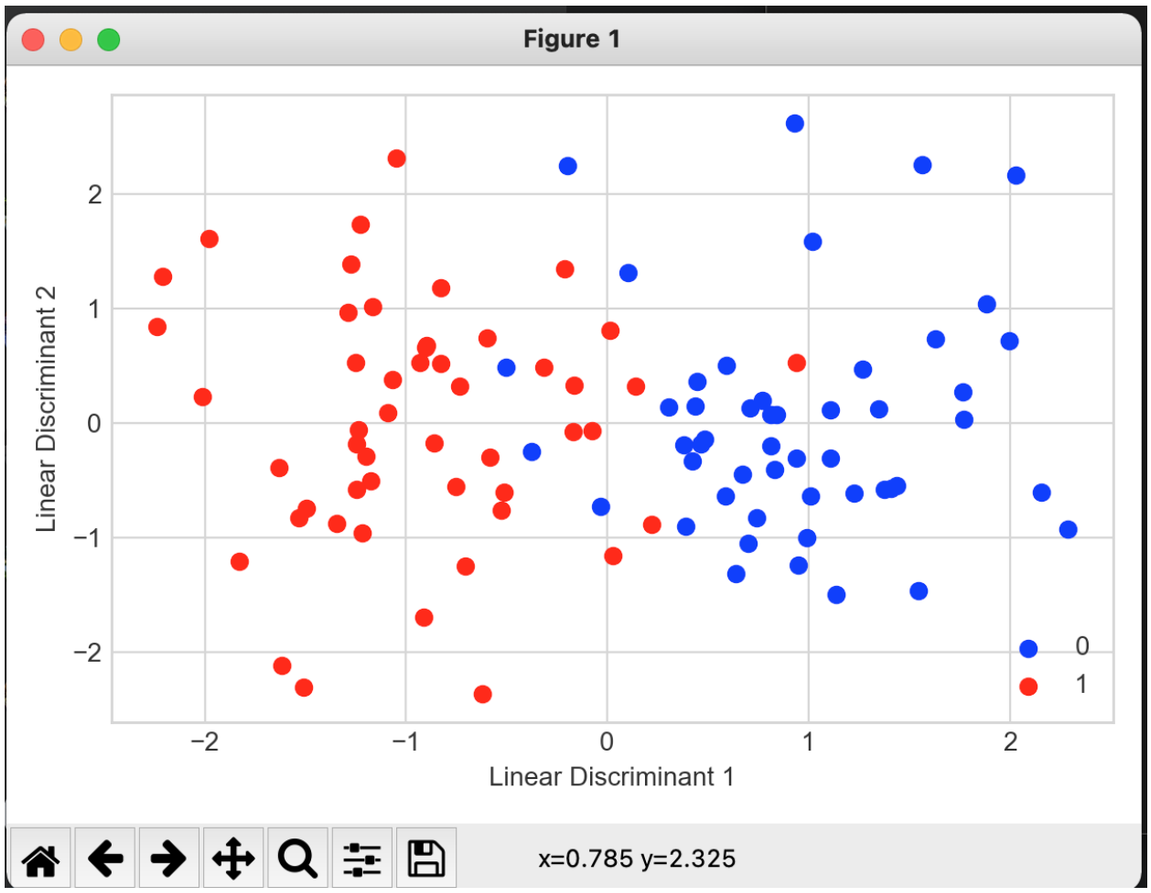


Figure 24: Sortie 16

6 Lancement du programme

6.1 Commande de lancement

Pour lancer le programme, il suffit de ce 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 créée par matplotlib. Pour avancer dans le programme, il suffit de fermer chaque fenêtre des représentations qui apparaissent à l'écran.

6.2 Exemples de lancement du programme

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

- MacOS
- Linux (Ubuntu)
- Windows 10

7 Sources

Retrouver les documentations officielles des librairies utilisées dans ce projet :

- pandas
- matplotlib
- searborn
- numpy

8 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