

## L'exploration de données: application sur Iris.csv en langage R et python

L'exploration de données consiste à analyser un jeu de données pour en comprendre la structure, identifier des tendances, des anomalies, des relations entre les variables, et ainsi préparer les données pour des analyses plus poussées ou des modèles prédictifs.

### Contexte du dataset Iris :

Le jeu de données Iris contient des mesures de caractéristiques de fleurs d'iris pour trois espèces différentes : **setosa**, **versicolor**, et **virginica**. Il y a 150 observations avec 5 variables :

1. **Sepal.Length** : longueur du sépale
2. **Sepal.Width** : largeur du sépale
3. **Petal.Length** : longueur du pétale
4. **Petal.Width** : largeur du pétale
5. **Species** : espèce de l'iris (setosa, versicolor, virginica)



### Étapes d'exploration des données en R et Python :

#### 1. Charger et examiner les données :

##### En R :

```
# Charger les bibliothèques nécessaires
library(ggplot2)

# Charger le jeu de données Iris
data(iris)

# Examiner les premières lignes du jeu de données
head(iris)

# Résumé statistique du dataset
summary(iris)

# Structure des données
str(iris)
```

```
dalilaiman@fedora:~ — /usr/lib64/R/bin/exec/R
```

R is a collaborative project with many contributors.  
Type 'contributors()' for more information and  
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or  
'help.start()' for an HTML browser interface to help.  
Type 'q()' to quit R.

```
> # Charger les bibliothèques nécessaires
library(ggplot2)

# Charger le jeu de données Iris
data(iris)

# Examiner les premières lignes du jeu de données
head(iris)

# Résumé statistique du dataset
summary(iris)

# Structure des données
str(iris)
```



dalilaiman@fedora:~ — /usr/lib64/R/bin/exec/R



Type 'demo()' for some demos, 'help()' for on-line help, or  
'help.start()' for an HTML browser interface to help.  
Type 'q()' to quit R.

```
> # Charger les bibliothèques nécessaires  
library(ggplot2)
```

```
# Charger le jeu de données Iris  
data(iris)
```

```
# Examiner les premières lignes du jeu de données  
head(iris)
```

```
# Résumé statistique du dataset  
summary(iris)
```

```
# Structure des données  
str(iris)
```

```
Error in library(ggplot2) : there is no package called 'ggplot2'
```

```
> install (ggplot2)
```

```
Error in install(ggplot2) : could not find function "install"
```

```
> install.packages("ggplot2")
```

```
dalilaiman@fedora:~ — /usr/lib64/R/bin/exec/R
5      5.0      3.6      1.4      0.2 setosa
6      5.4      3.9      1.7      0.4 setosa
  Sepal.Length Sepal.Width Petal.Length Petal.Width
Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
Median :5.800   Median :3.000   Median :4.350   Median :1.300
Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
  Species
setosa   :50
versicolor:50
virginica :50

'data.frame':  150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species      : Factor w/ 3 levels "setosa","versicolor",...: 1 1 1 1 1 1 1 1 1 1
1 ...
>
```

## En Python :

# Charger les bibliothèques nécessaires

```
import pandas as pd
```

```
import seaborn as sns
```

# Charger le dataset Iris

```
iris = sns.load_dataset("iris")
```

# Examiner les premières lignes du jeu de données

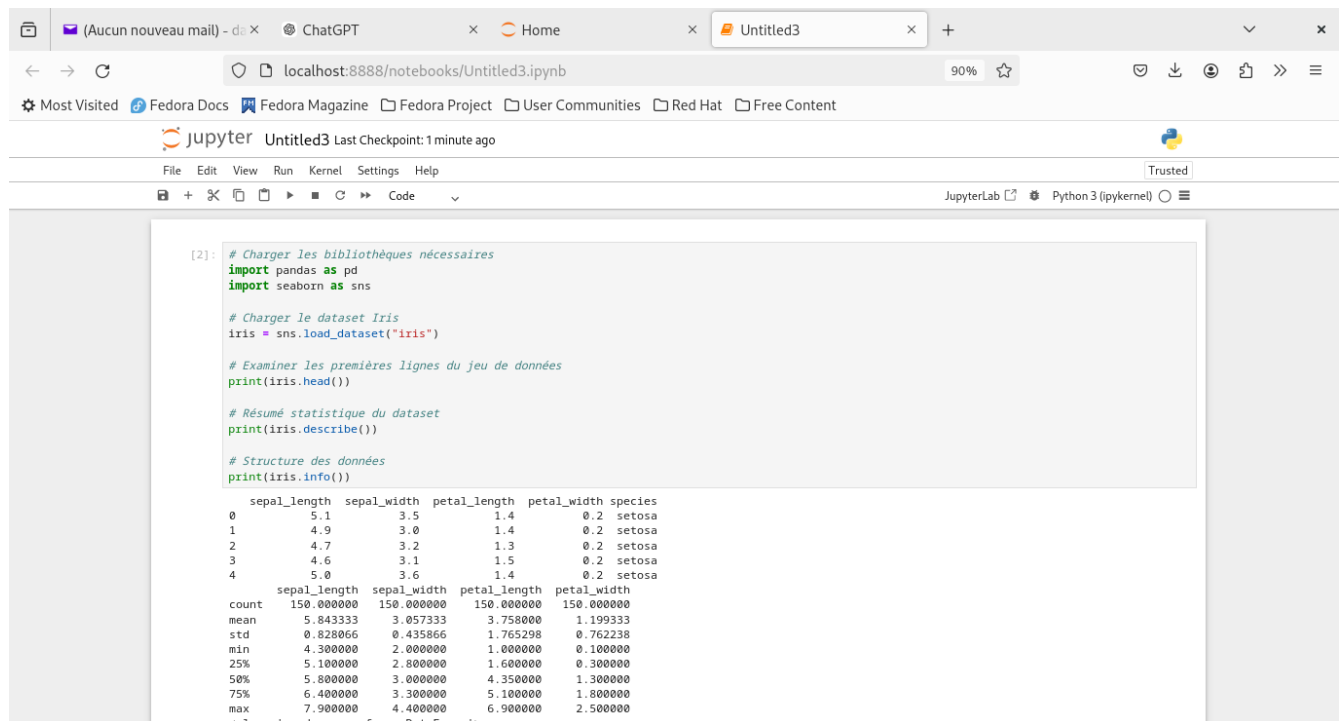
```
print(iris.head())
```

# Résumé statistique du dataset

```
print(iris.describe())
```

# Structure des données

```
print(iris.info())
```



The screenshot shows a JupyterLab interface with a code cell containing the following Python code:

```
[2]: # Charger les bibliothèques nécessaires
import pandas as pd
import seaborn as sns

# Charger le dataset Iris
iris = sns.load_dataset("Iris")

# Examiner les premières lignes du jeu de données
print(iris.head())

# Résumé statistique du dataset
print(iris.describe())

# Structure des données
print(iris.info())
```

The output of the code is displayed below the cell:

```
0      5.1      3.5      1.4      0.2  setosa
1      4.9      3.0      1.4      0.2  setosa
2      4.7      3.2      1.3      0.2  setosa
3      4.6      3.1      1.5      0.2  setosa
4      5.0      3.6      1.4      0.2  setosa
```

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

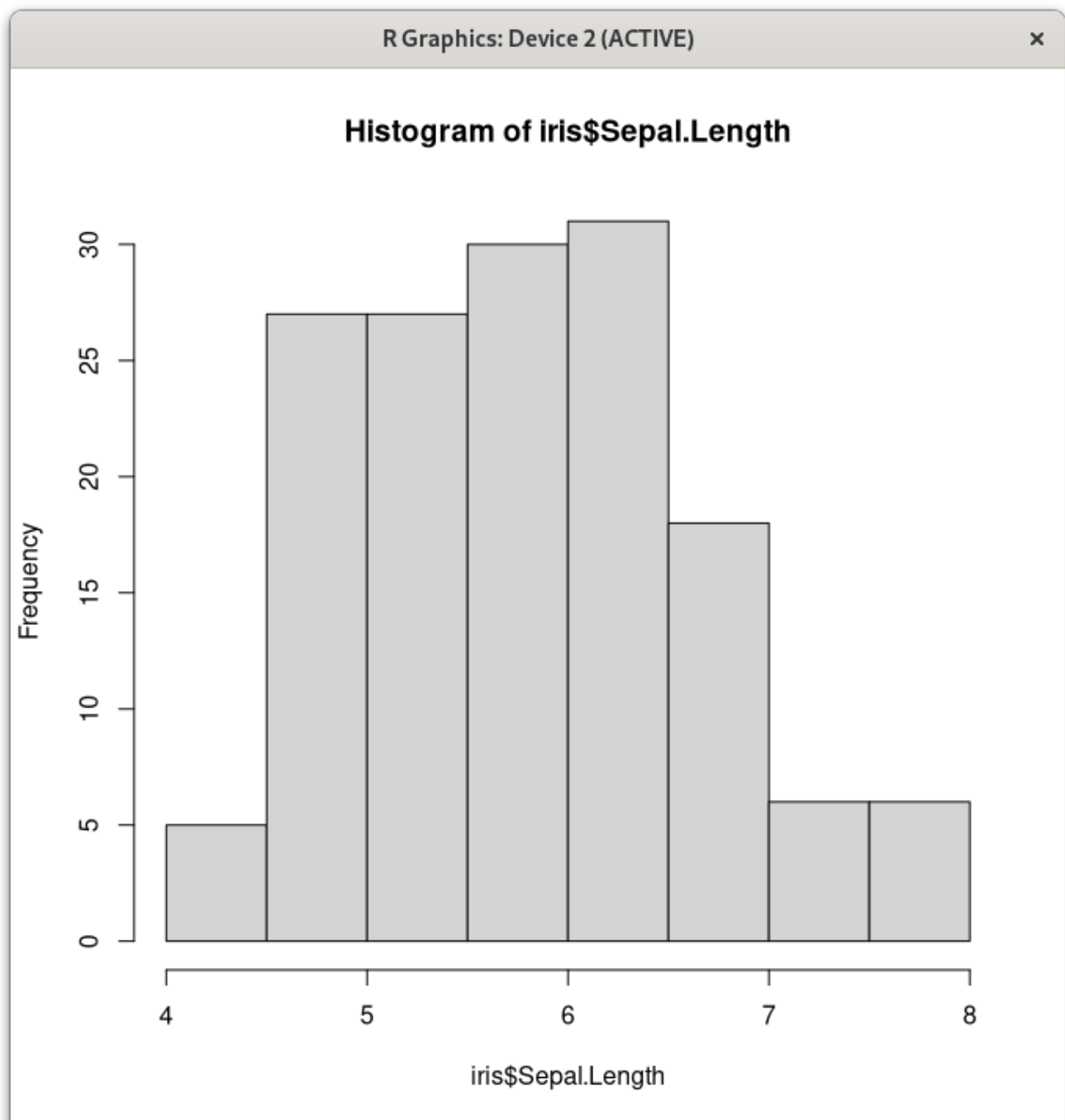
<class 'pandas.core.frame.DataFrame'>

## 2. Visualisation des données :

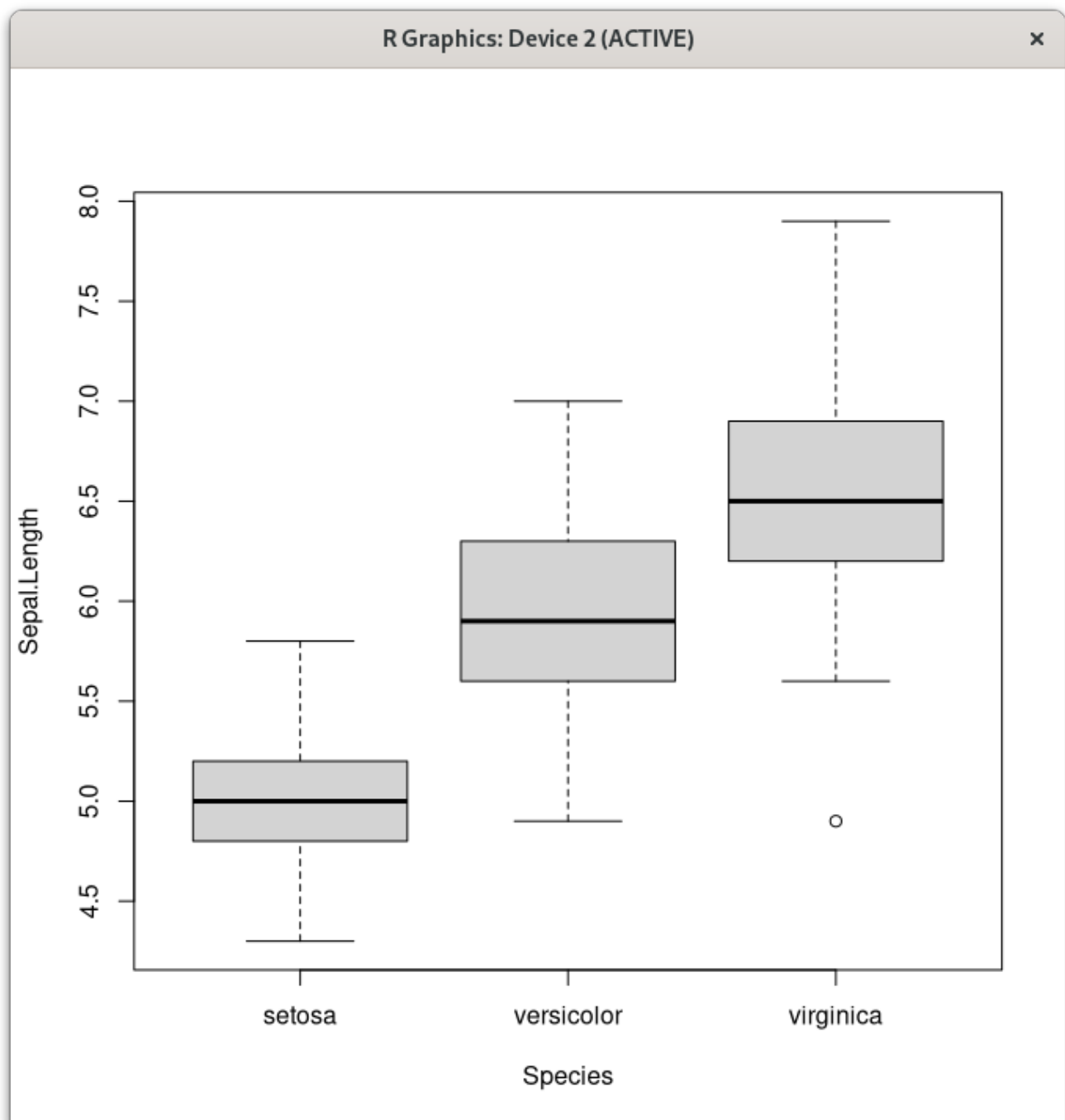
La visualisation est une étape cruciale pour explorer les relations entre les différentes variables.

**En R :**

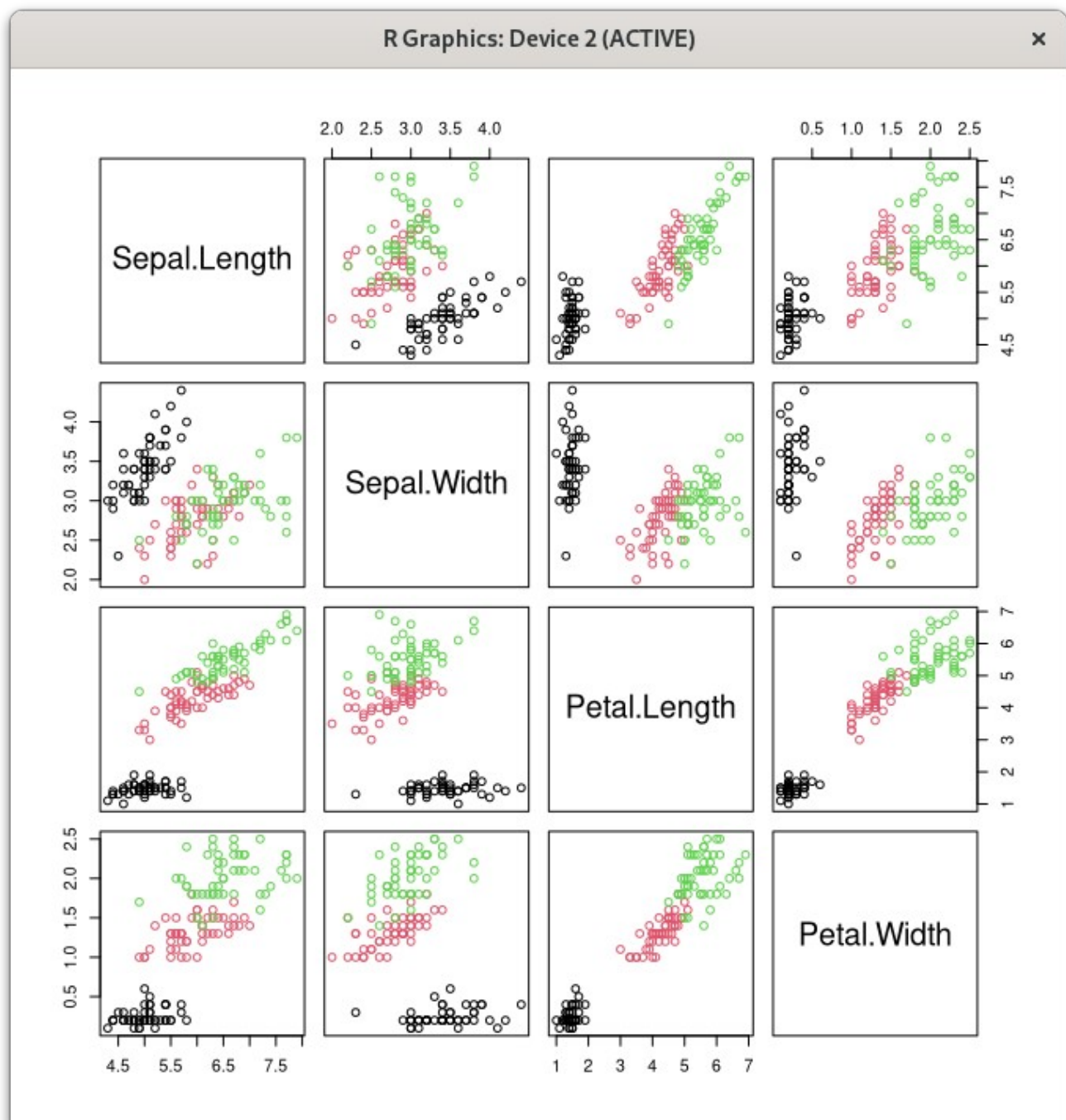
```
# Histogramme pour la distribution des différentes variables
hist(iris$Sepal.Length)
```



```
# Boxplot pour visualiser la distribution des longueurs de  
sépalés par espèce  
boxplot(Sepal.Length ~ Species, data = iris)
```



```
# Pair plot (ou graphique de paires) pour visualiser les  
relations entre toutes les variables  
pairs(iris[,1:4], col = iris$Species)
```



### En Python :

```
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# Histogramme pour la distribution des différentes
variables
sns.histplot(iris['sepal_length'], kde=True)
```



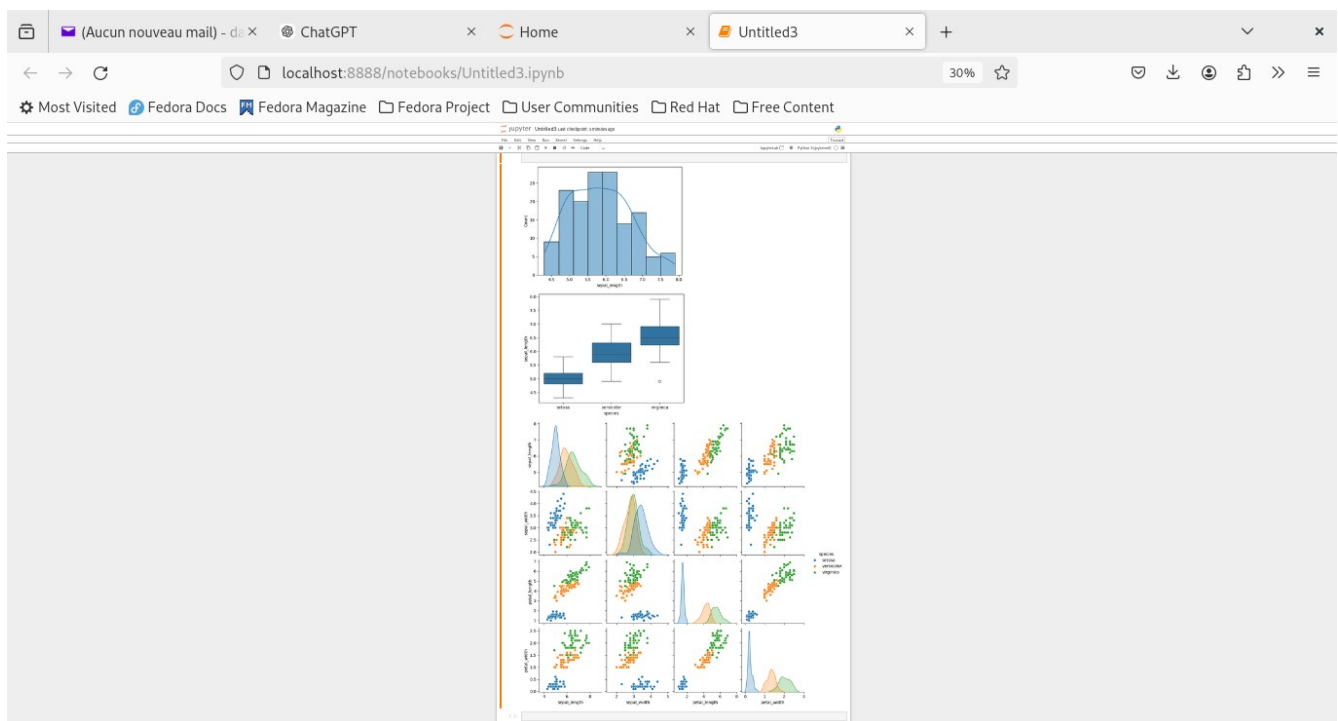
```
plt.show()
```

```
# Boxplot pour visualiser la distribution des longueurs de  
sépalles par espèce
```

```
sns.boxplot(x='species', y='sepal_length', data=iris)  
plt.show()
```

```
# Pairplot pour voir les relations entre les variables
```

```
sns.pairplot(iris, hue='species')  
plt.show()
```



### 3. Analyse des corrélations :

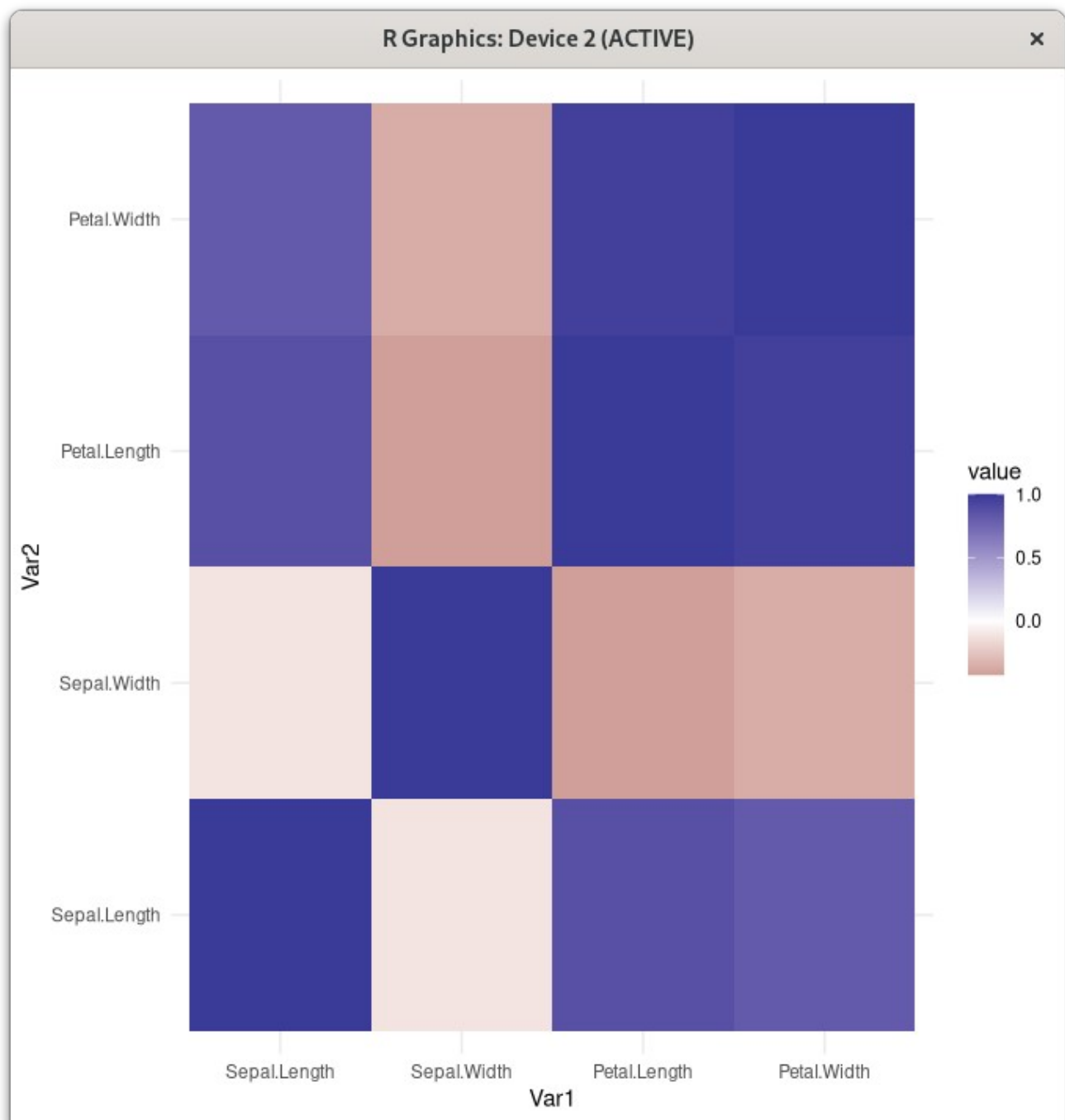
**En R :**

```
# Matrice de corrélation  
cor(iris[, 1:4])
```

```
dalilaiman@fedora:~ — /usr/lib64/R/bin/exec/R
pairs(iris[, 1:4], col = iris$Species) ~ X
> # Histogramme pour la distribution des différentes variables
hist(iris$Sepal.Length)
> # Pair plot (ou graphique de paires) pour visualiser les relations entre toutes les variables
pairs(iris[,1:4], col = iris$Species)
> ^[[200~# Boxplot pour visualiser la distribution des longueurs de sépales par espèce
Error: unexpected invalid token in "
> boxplot(Sepal.Length ~ Species, data = iris)~
+
+
+ X
boxplot(Sepal.Length ~ Species, data = iris) ~ X
> # Boxplot pour visualiser la distribution des longueurs de sépales par espèce
boxplot(Sepal.Length ~ Species, data = iris)
> # Matrice de corrélation
cor(iris[, 1:4])
      Sepal.Length Sepal.Width Petal.Length Petal.Width
Sepal.Length    1.0000000   -0.1175698    0.8717538    0.8179411
Sepal.Width     -0.1175698    1.0000000   -0.4284401   -0.3661259
Petal.Length     0.8717538   -0.4284401    1.0000000    0.9628654
Petal.Width      0.8179411   -0.3661259    0.9628654    1.0000000
>
```

Ce code crée une **heatmap** (carte thermique) pour visualiser la matrice de corrélation entre les variables du jeu de données iris.

```
# Heatmap pour visualiser la corrélation
library(ggplot2)
library(reshape2)
corr_matrix <- cor(iris[,1:4])
melted_corr <- melt(corr_matrix)
ggplot(data = melted_corr, aes(Var1, Var2, fill = value)) +
  geom_tile() +
  scale_fill_gradient2() +
  theme_minimal()
```



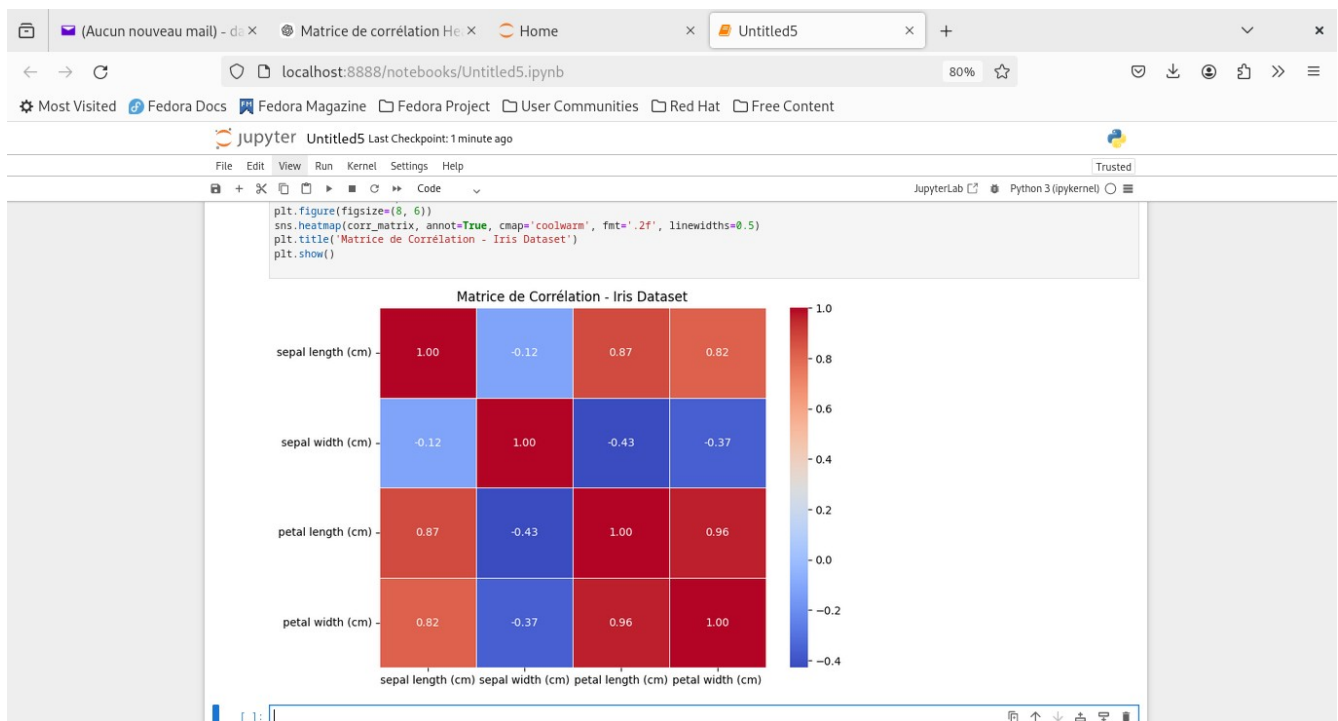
### En Python :

```
# Matrice de corrélation
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
```

```
# Charger le dataset Iris
iris = load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)

# Calculer la matrice de corrélation
corr_matrix = df.corr()

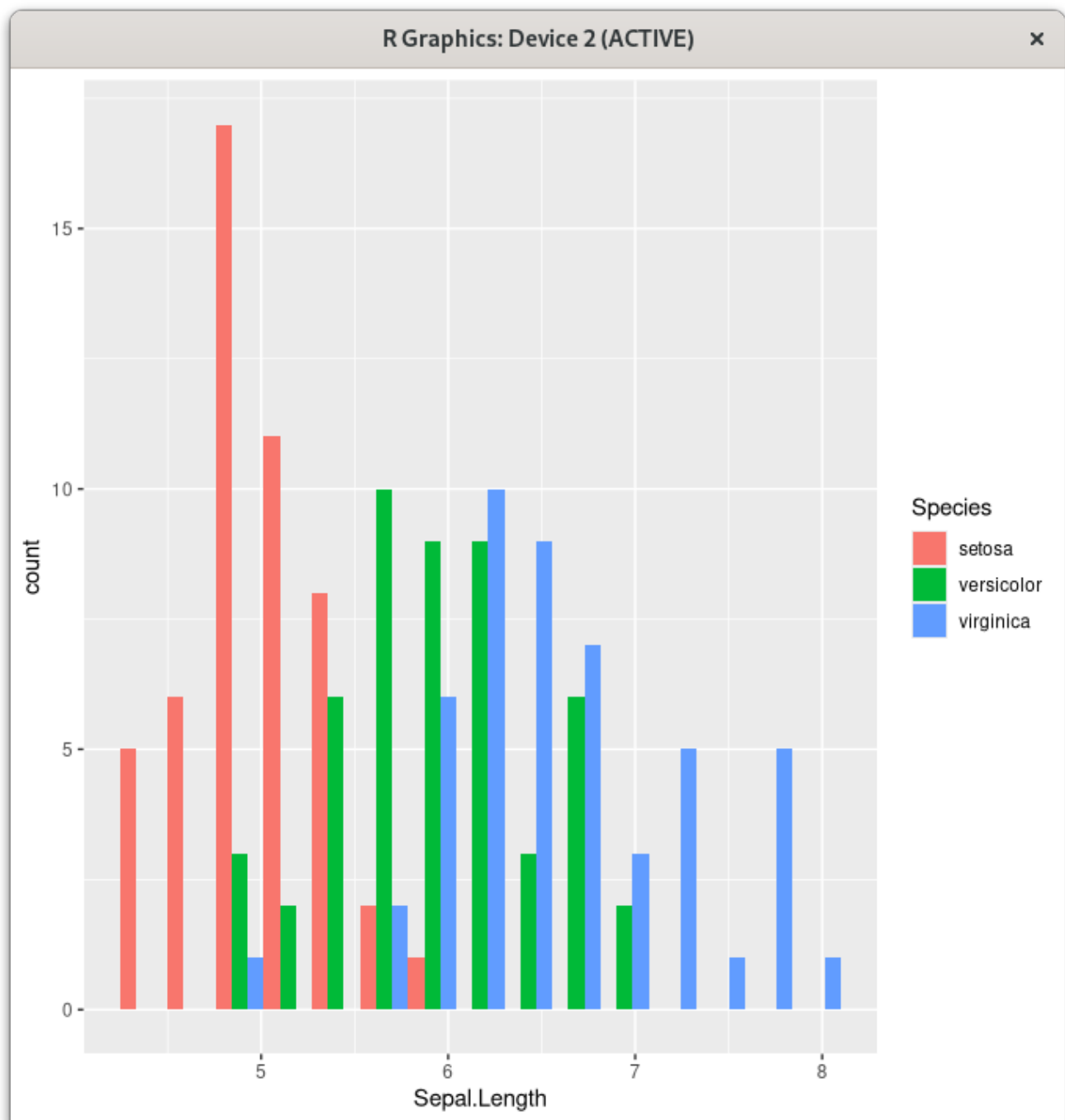
# Afficher la heatmap de la matrice de corrélation
plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm',
            fmt='.2f', linewidths=0.5)
plt.title('Matrice de Corrélation - Iris Dataset')
plt.show()
```



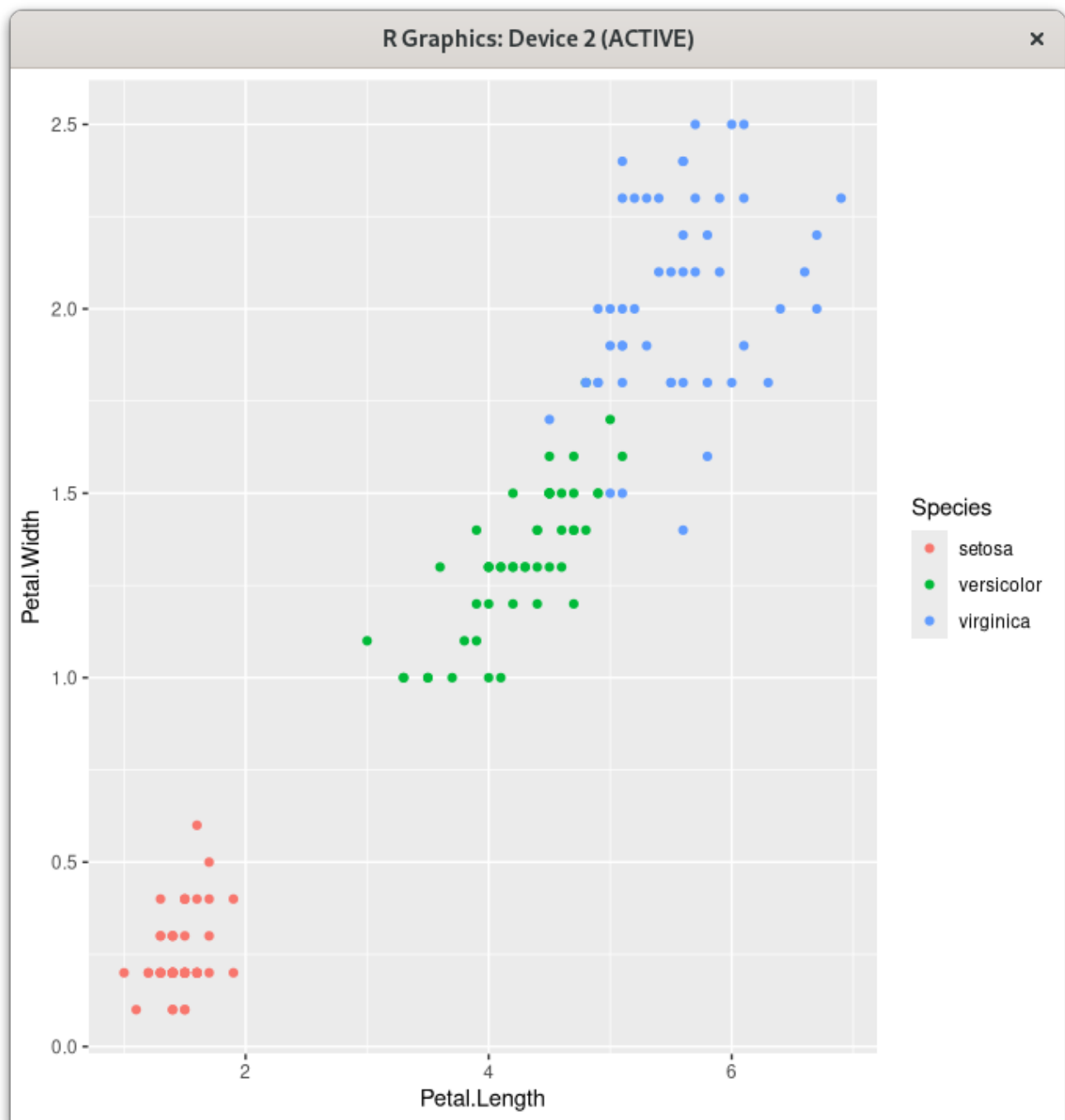
#### 4. Analyse des tendances et distribution par espèce :

**En R :**

```
# Visualiser la distribution des variables par espèce
ggplot(iris, aes(x = Sepal.Length, fill = Species)) +
  geom_histogram(position = "dodge", bins = 15)
```



```
# Scatter plot pour explorer la relation entre Petal.Length  
et Petal.Width  
ggplot(iris, aes(x = Petal.Length, y = Petal.Width, color =  
Species)) +  
  geom_point()
```



```
sns.scatterplot(data=iris, x="petal_length",  
y="petal_width", hue="species")  
plt.show()
```

```
# Charger les bibliothèques nécessaires
```

```
import pandas as pd
```

```
import seaborn as sns
```

```
# Charger le dataset Iris
```

```
iris = sns.load_dataset("iris")
```

```
# Visualiser la distribution des variables par espèce
```

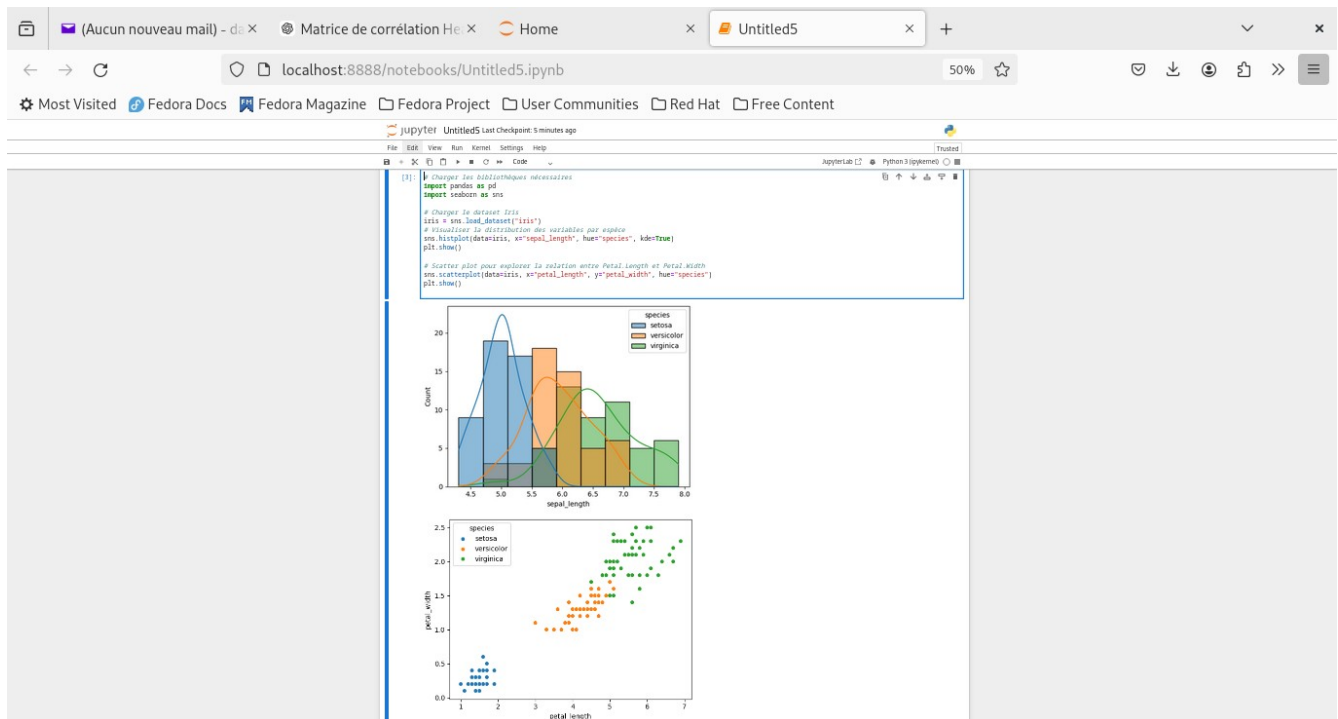
```
sns.histplot(data=iris, x="sepal_length", hue="species",  
kde=True)
```

```
plt.show()
```

```
# Scatter plot pour explorer la relation entre Petal.Length  
et Petal.Width
```

```
sns.scatterplot(data=iris, x="petal_length",  
y="petal_width", hue="species")
```

```
plt.show()
```



## Conclusion :

L'exploration des données sur **Iris.csv** permet de mieux comprendre la distribution et les relations entre les différentes variables, et de préparer les données pour des analyses ou des modèles prédictifs. La visualisation et la matrice de corrélation aident à déceler des tendances, des clusters et des relations importantes. Que ce soit en **R** ou en **Python**, les deux langages offrent des outils puissants pour cette exploration, avec des bibliothèques telles que **ggplot2** et **seaborn** qui facilitent la visualisation et l'analyse.

### 1. Valeurs manquantes

Les valeurs manquantes (ou "missing values") sont des observations absentes ou non renseignées dans un jeu de données. Pour le dataset Iris, vérifions si des valeurs sont manquantes.

#### En R :

```
# Vérifier les valeurs manquantes
sum(is.na(iris))
```



## En Python :

```
# Vérifier les valeurs manquantes  
print(iris.isna().sum())
```

**Résultat attendu :** Le dataset Iris ne contient généralement pas de valeurs manquantes, donc le résultat devrait être nul (0).

---

## 2. Valeurs aberrantes (outliers)

Les valeurs aberrantes (outliers) sont des observations qui s'écartent de manière significative des autres observations dans un jeu de données. Pour identifier les valeurs aberrantes, nous utiliserons l'**IQR** et le **Z-score**.

### a. IQR (Intervalle interquartile)

L'IQR est la différence entre le 75ème percentile (Q3) et le 25ème percentile (Q1). Les valeurs qui se trouvent en dehors de cet intervalle peuvent être considérées comme aberrantes.

**\*\*En R :**

```
# Calculer l'IQR pour chaque variable quantitative  
IQR_values <- apply(iris[,1:4], 2, IQR)  
  
# Définir les limites pour chaque variable (Q1 - 1.5*IQR et  
# Q3 + 1.5*IQR)  
outlier_limits <- apply(iris[,1:4], 2, function(x) {  
  Q1 <- quantile(x, 0.25)  
  Q3 <- quantile(x, 0.75)  
  IQR <- Q3 - Q1  
  c(Q1 - 1.5*IQR, Q3 + 1.5*IQR)  
})  
  
# Identifier les valeurs aberrantes  
outliers <- apply(iris[,1:4], 2, function(x) {  
  any(x < outlier_limits[1] | x > outlier_limits[2])  
})  
print (outliers)
```

```
dalilaiman@fedora:~ — /usr/lib64/R/bin/exec/R
ggplot(iris, aes(x = Petal.Length, y = Petal.Width, color = Species)) +
  geom_point()
> # Vérifier les valeurs manquantes
sum(is.na(iris))
[1] 0
> # Calculer l'IQR pour chaque variable quantitative
IQR_values <- apply(iris[,1:4], 2, IQR)

# Définir les limites pour chaque variable (Q1 - 1.5*IQR et Q3 + 1.5*IQR)
outlier_limits <- apply(iris[,1:4], 2, function(x) {
  Q1 <- quantile(x, 0.25)
  Q3 <- quantile(x, 0.75)
  IQR <- Q3 - Q1
  c(Q1 - 1.5*IQR, Q3 + 1.5*IQR)
})

# Identifier les valeurs aberrantes
outliers <- apply(iris[,1:4], 2, function(x) {
  any(x < outlier_limits[1] | x > outlier_limits[2])
})
> print (outliers)
Sepal.Length Sepal.Width Petal.Length Petal.Width
          FALSE          TRUE          TRUE          TRUE
>
```

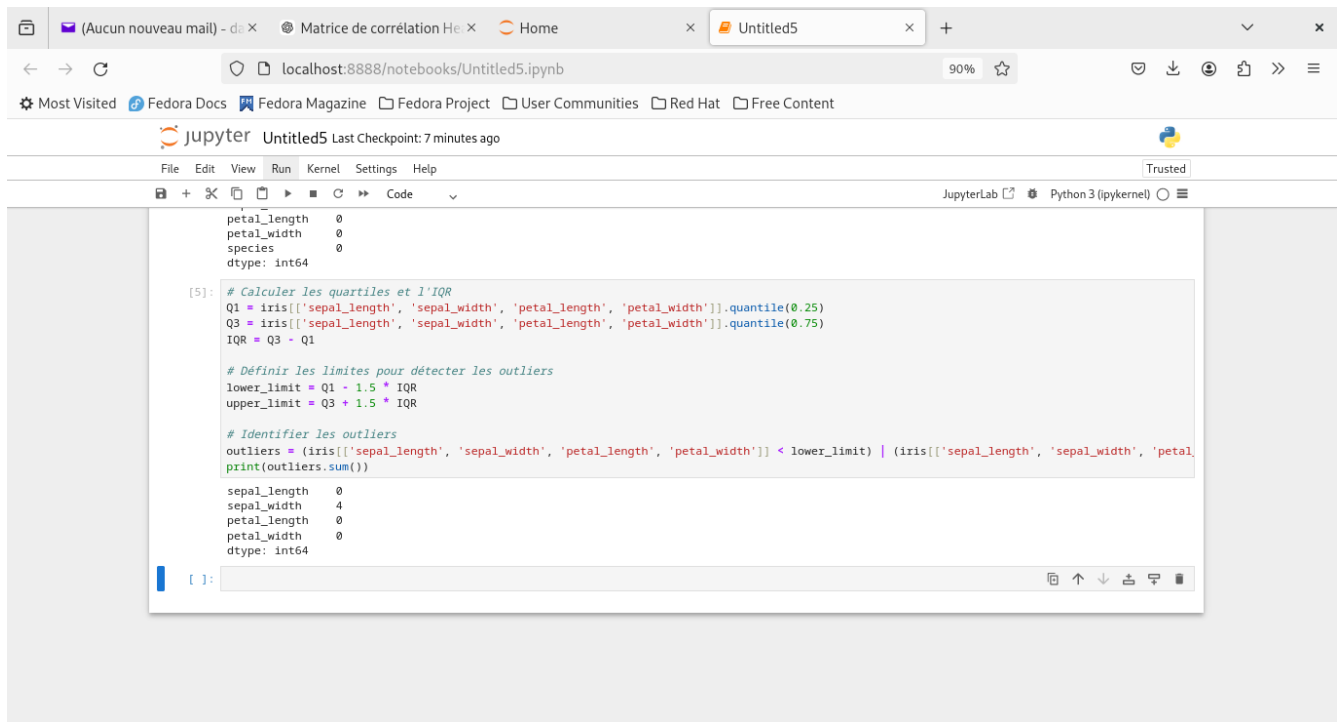
**\*\*En Python :**

```
# Calculer les quartiles et l'IQR
Q1 = iris[['sepal_length', 'sepal_width', 'petal_length',
'petal_width']].quantile(0.25)
Q3 = iris[['sepal_length', 'sepal_width', 'petal_length',
'petal_width']].quantile(0.75)
IQR = Q3 - Q1

# Définir les limites pour détecter les outliers
lower_limit = Q1 - 1.5 * IQR
upper_limit = Q3 + 1.5 * IQR

# Identifier les outliers
outliers = (iris[['sepal_length', 'sepal_width',
'petal_length', 'petal_width']] < lower_limit) |
```

```
(iris[['sepal_length', 'sepal_width', 'petal_length',
'petal_width']] > upper_limit)
print(outliers.sum())
```



The screenshot shows a JupyterLab window titled 'Untitled5' running on a local host. The code in the notebook calculates the IQR for the 'sepal\_length', 'sepal\_width', and 'petal\_length' variables, defines lower and upper limits based on 1.5 times the IQR, and then identifies outliers as values below the lower limit or above the upper limit. The output shows that there are 4 outliers in the 'sepal\_width' variable.

```
petal_length    0
petal_width     0
species         0
dtype: int64

[5]: # Calculer les quartiles et l'IQR
Q1 = iris[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']].quantile(0.25)
Q3 = iris[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']].quantile(0.75)
IQR = Q3 - Q1

# Définir les limites pour détecter les outliers
lower_limit = Q1 - 1.5 * IQR
upper_limit = Q3 + 1.5 * IQR

# Identifier les outliers
outliers = (iris[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']] < lower_limit) | (iris[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']] > upper_limit)
print(outliers.sum())

sepal_length    0
sepal_width     4
petal_length     0
petal_width     0
dtype: int64
```

## b. Z-Score

Le **Z-score** mesure combien d'écart-types une observation se trouve par rapport à la moyenne. Si le Z-score est supérieur à 3 ou inférieur à -3, cela pourrait indiquer une valeur aberrante.

**\*\*En R :**

```
# Calculer les Z-scores
z_scores <- scale(iris[,1:4])
```

```
# Identifier les valeurs aberrantes (z-score > 3 ou < -3)
outliers_zscore <- apply(z_scores, 2, function(x)
sum(abs(x) > 3))
print(outliers_zscore)
```

```
dalilaiman@fedora:~ — /usr/lib64/R/bin/exec/R
# Définir les limites pour chaque variable (Q1 - 1.5*IQR et Q3 + 1.5*IQR)
outlier_limits <- apply(iris[,1:4], 2, function(x) {
  Q1 <- quantile(x, 0.25)
  Q3 <- quantile(x, 0.75)
  IQR <- Q3 - Q1
  c(Q1 - 1.5*IQR, Q3 + 1.5*IQR)
})

# Identifier les valeurs aberrantes
outliers <- apply(iris[,1:4], 2, function(x) {
  any(x < outlier_limits[1] | x > outlier_limits[2])
})
> print (outliers)
Sepal.Length Sepal.Width Petal.Length Petal.Width
          FALSE          TRUE          TRUE          TRUE
> # Calculer les Z-scores
z_scores <- scale(iris[,1:4])

# Identifier les valeurs aberrantes (z-score > 3 ou < -3)
outliers_zscore <- apply(z_scores, 2, function(x) sum(abs(x) > 3))
print(outliers_zscore)
Sepal.Length Sepal.Width Petal.Length Petal.Width
           0           1           0           0
> 
```

**\*\*En Python :**

```
from scipy.stats import zscore
```

```
# Calculer les Z-scores
```

```
z_scores = zscore(iris[['sepal_length', 'sepal_width',  
'petal_length', 'petal_width']])
```

```
# Identifier les valeurs aberrantes (z-score > 3 ou < -3)
```

```
outliers_zscore = (abs(z_scores) > 3).sum(axis=0)
```

```
print(outliers_zscore)
```

```
# Identifier les outliers
outliers = (iris[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']] < lower_limit) | (iris[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']] > upper_limit)
print(outliers.sum())

sepal_length    0
sepal_width     4
petal_length    0
petal_width     0
dtype: int64

[6]: from scipy.stats import zscore

# Calculer les Z-scores
z_scores = zscore(iris[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']])

# Identifier les valeurs aberrantes (z-score > 3 ou < -3)
outliers_zscore = (abs(z_scores) > 3).sum(axis=0)
print(outliers_zscore)

sepal_length    0
sepal_width     1
petal_length    0
petal_width     0
dtype: int64
```

## b. Introduction de Valeurs Manquantes

Imaginons que nous introduisons des valeurs manquantes pour simuler des situations réelles.

- **En R :**

- # Introduire des valeurs manquantes dans une colonne (par exemple Petal.Length)

```
set.seed(123) # Pour la reproductibilité
```

```
iris[sample(1:nrow(iris), 20), "Petal.Length"] <- NA
```

```
iris[10, "Petal.Length"] <- NA
```

- **En Python :**

```
import numpy as np
```

```
# Introduire des valeurs manquantes aléatoires
```

```
np.random.seed(123)
```

```
missing_indices = np.random.choice(iris.index, size=20, replace=False)
```

```
iris.loc[missing_indices, 'petal_length'] = np.nan
```

### c. Traitement des Valeurs Manquantes

Une fois les valeurs manquantes identifiées, nous devons choisir une méthode pour les gérer. L'une des méthodes les plus courantes est **l'interpolation**, qui consiste à estimer les valeurs manquantes en fonction des autres valeurs disponibles dans les données.

- **En R** (en utilisant l'interpolation linéaire) :

```
# Imputation par interpolation linéaire
library(zoo)
iris$Petal.Length <- na.approx(iris$Petal.Length)
```

- **En Python** (en utilisant l'interpolation linéaire) :

```
# Imputation par interpolation linéaire
iris['petal_length'] =
iris['petal_length'].interpolate(method='linear')
```

L'interpolation permet de remplir les valeurs manquantes sans introduire de biais significatifs, surtout dans les données numériques continues comme celles du dataset Iris.

---

## 2. Valeurs Manquantes

Les valeurs manquantes sont fréquentes dans de nombreux jeux de données réels. Dans le cas du dataset Iris, il n'y en a généralement pas, mais nous allons simuler leur présence pour comprendre comment les traiter.

### a. Visualisation des valeurs manquantes

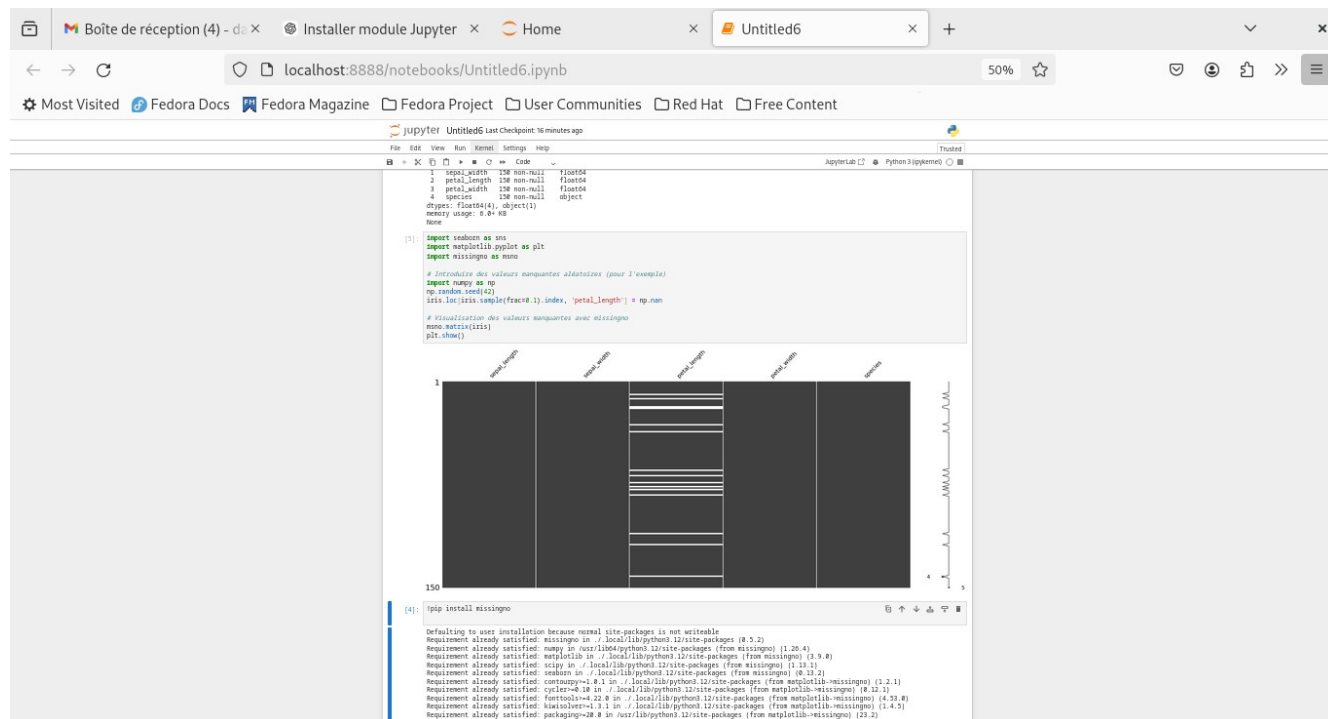
**En Python** (avec **seaborn** et **missingno** pour visualiser les valeurs manquantes) :

```
import seaborn as sns
import matplotlib.pyplot as plt
import missingno as msno
```

```
# Introduire des valeurs manquantes aléatoires (pour
l'exemple)
import numpy as np
```

```
np.random.seed(42)
iris.loc[iris.sample(frac=0.1).index, 'petal_length'] =
np.nan
```

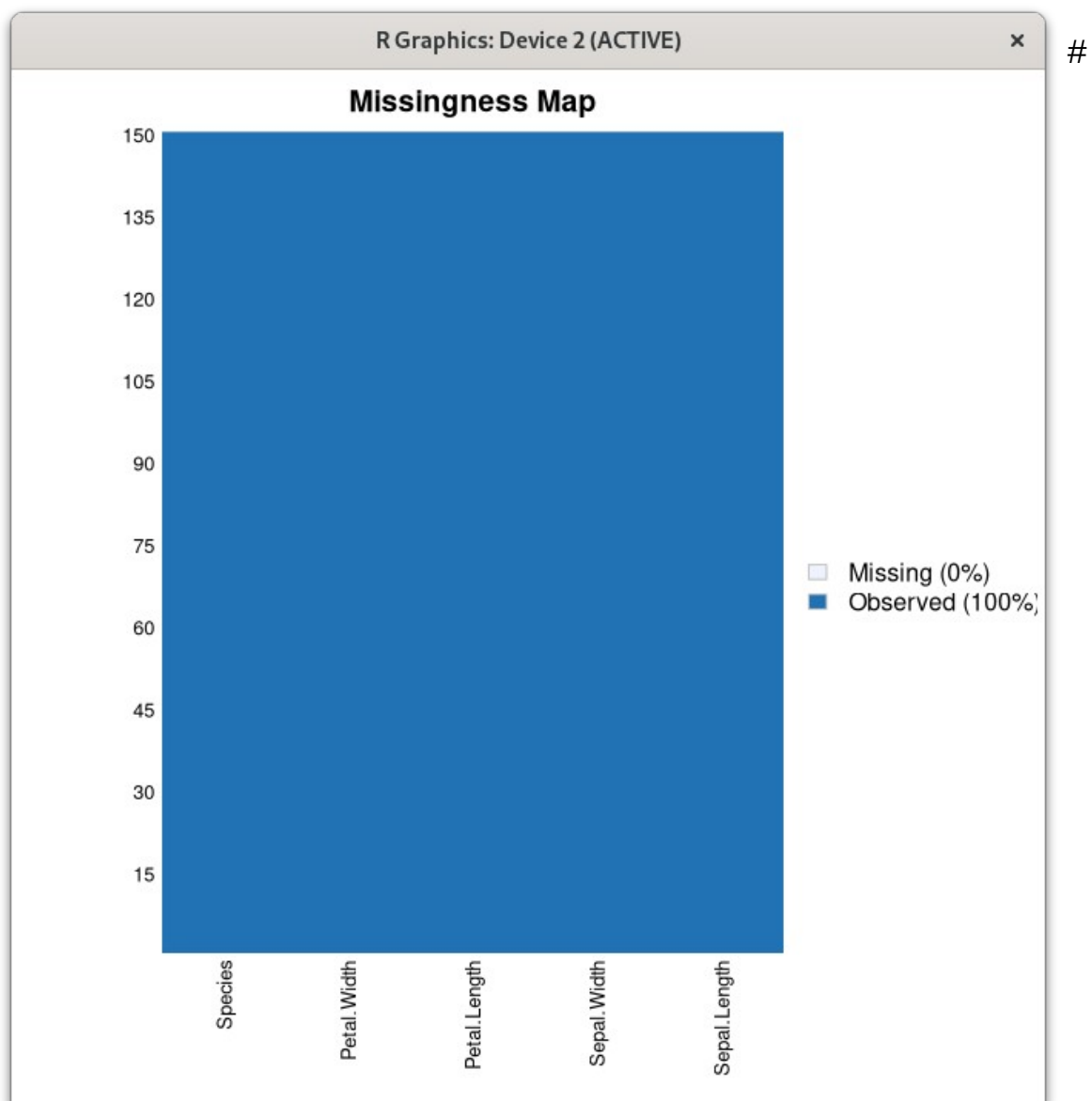
```
# Visualisation des valeurs manquantes avec missingno
msno.matrix(iris)
plt.show()
```



## En R :

```
# Introduire des valeurs manquantes
set.seed(42)
iris[sample(1:nrow(iris), 15), "Petal.Length"] <- NA
```

```
# Visualisation des valeurs manquantes
library(ggplot2)
library(Amelia)
missmap(iris)
```



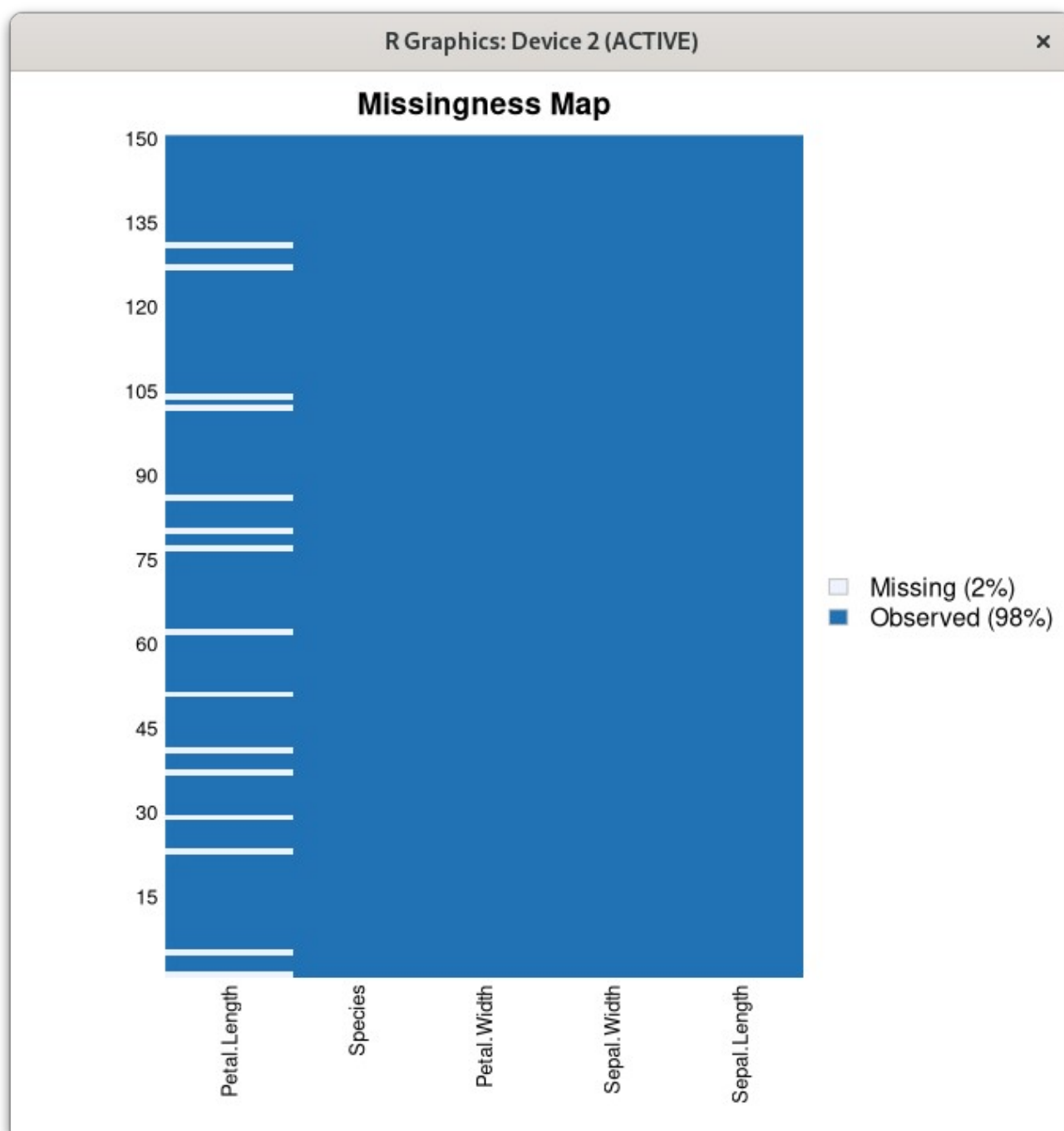
Introduire des valeurs manquantes

```
set.seed(42)
```

```
iris[sample(1:nrow(iris), 15), "Petal.Length"] <- NA
```



```
# Visualisation des valeurs manquantes  
library(ggplot2)  
library(Amelia)  
missmap(iris)
```



### 3. Variables quantitatives et qualitatives

- **Variables quantitatives** : Ce sont des variables numériques qui mesurent des quantités et peuvent être continues ou discrètes.
  - Dans le dataset Iris, les variables quantitatives sont :
    - Sepal.Length
    - Sepal.Width
    - Petal.Length
    - Petal.Width
- **Variables qualitatives** : Ce sont des variables catégorielles qui indiquent des qualités ou des groupes.
  - Dans le dataset Iris, la variable qualitative est :
    - Species (setosa, versicolor, virginica)

### 4. Résumé statistique et distributions

Voici un résumé statistique des variables quantitatives dans le dataset Iris (moyenne, écart-type, min, max, quartiles) :

**\*\*En R :**

```
# Résumé statistique des variables quantitatives  
summary(iris[,1:4])
```

```
dalilaiman@fedora:~ — /usr/lib64/R/bin/exec/R
any(x < outlier_limits[1] | x > outlier_limits[2])
})
> print (outliers)
Sepal.Length Sepal.Width Petal.Length Petal.Width
          FALSE          TRUE          TRUE          TRUE
> # Calculer les Z-scores
z_scores <- scale(iris[,1:4])

# Identifier les valeurs aberrantes (z-score > 3 ou < -3)
outliers_zscore <- apply(z_scores, 2, function(x) sum(abs(x) > 3))
print(outliers_zscore)
Sepal.Length Sepal.Width Petal.Length Petal.Width
           0           1           0           0
> ^[[200~# Résumé statistique des variables quantitatives
Error: unexpected invalid token in "
> summary(iris[,1:4])
  Sepal.Length    Sepal.Width    Petal.Length    Petal.Width
Min.   :4.300    Min.   :2.000    Min.   :1.000    Min.   :0.100
1st Qu.:5.100    1st Qu.:2.800    1st Qu.:1.600    1st Qu.:0.300
Median :5.800    Median :3.000    Median :4.350    Median :1.300
Mean   :5.843    Mean   :3.057    Mean   :3.758    Mean   :1.199
3rd Qu.:6.400    3rd Qu.:3.300    3rd Qu.:5.100    3rd Qu.:1.800
Max.   :7.900    Max.   :4.400    Max.   :6.900    Max.   :2.500
> ~
```

**\*\*En Python :**

```
# Résumé statistique des variables quantitatives
print(iris.describe())
```

**Résultat attendu :** Ce résumé fournit des informations sur les caractéristiques principales des variables quantitatives, comme la moyenne, l'écart-type, les quartiles, etc.

---

### Conclusion sur l'exploration des données Iris :

- **Valeurs manquantes :** Le dataset Iris ne présente pas de valeurs manquantes.
- **Valeurs aberrantes :** En utilisant l'IQR et les Z-scores, nous pouvons identifier des valeurs aberrantes, bien que le dataset Iris soit généralement bien comporté.

Quelques points peuvent être considérés comme extrêmes mais sont généralement acceptables.

- **Variables quantitatives** : Les variables numériques telles que la longueur et la largeur des sépales et pétales sont des données continues.
- **Variables qualitatives** : La variable `Species` est une variable catégorielle.

L'exploration des données à travers ces différentes étapes nous aide à préparer le jeu de données pour des analyses plus poussées ou la construction de modèles prédictifs.

L'exploration des données (ou **Data Exploration**) est une étape clé dans le processus de science des données. Elle permet d'examiner un jeu de données pour mieux comprendre sa structure, ses caractéristiques et sa qualité. Dans ce contexte, nous allons détailler une **exploration des données appliquée au dataset Iris**, en incluant la gestion des valeurs manquantes, l'interpolation et l'analyse des variables quantitatives et qualitatives. Voici un résumé détaillé :

---

## ii. Comparaison des Variables Quantitatives par Espèce

Un **boxplot** permet de comparer les distributions des variables quantitatives entre les différentes espèces.

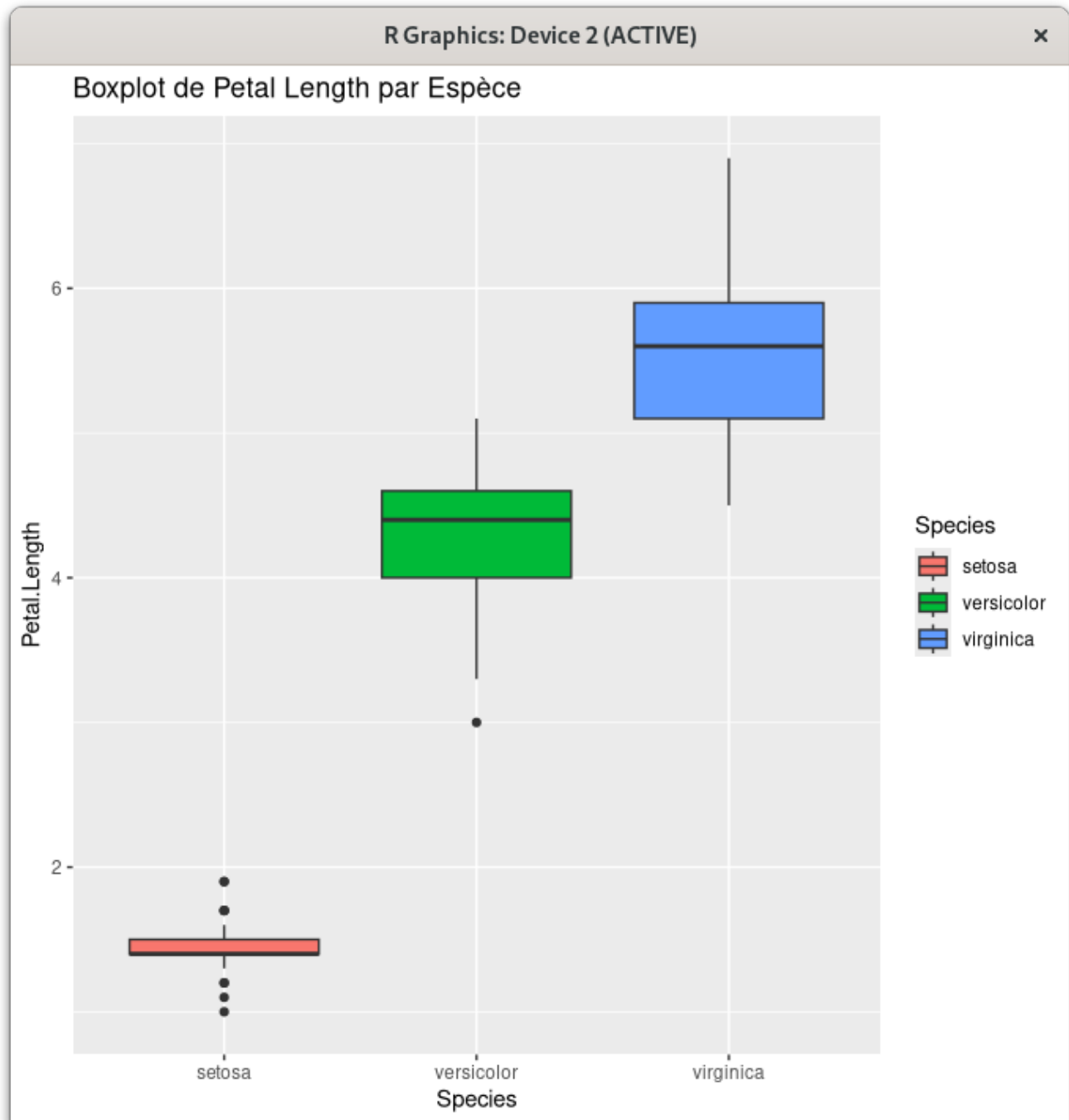
**En Python :**

```
# Comparaison de Petal Length par espèce
sns.boxplot(x='species', y='petal length (cm)', data=iris)
plt.title("Boxplot de Petal Length par Espèce")
plt.show()
```

**En R :**

```
# Boxplot de Petal Length par espèce
ggplot(iris, aes(x = Species, y = Petal.Length, fill =
Species)) +
  geom_boxplot() + ggtitle("Boxplot de Petal Length par
Espèce")
```

---



### b. Diagramme de Dispersion 3D

Pour explorer des relations multidimensionnelles, un graphique en 3D peut être utile.

**En Python :**

```
from mpl_toolkits.mplot3d import Axes3D

# Visualisation 3D des variables Petal Length, Petal Width,
et Sepal Length
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(iris['sepal length (cm)'], iris['petal length
(cm)'], iris['petal width (cm)'],
c=iris['species'].astype('category').cat.codes)
ax.set_xlabel('Sepal Length')
ax.set_ylabel('Petal Length')
ax.set_zlabel('Petal Width')
plt.title("Visualisation 3D des Variables")
plt.show()
```

#### **En R :**

```
# Installation de plotly pour visualisation 3D
install.packages("plotly")
library(plotly)

# Visualisation 3D de Petal.Length, Petal.Width,
Sepal.Length
plot_ly(iris, x = ~Sepal.Length, y = ~Petal.Length, z =
~Petal.Width, color = ~Species, type = 'scatter3d', mode =
'markers')
```

---