R Notebook

Install packages

R libraries are stored and managed in a repository called CRAN. You can download R packages with the install.packages() function

```r
install.packages("reticulate")
```

You only need to install packages once, but you need to mount those packages with the library() function each time you open R.

```r
library(reticulate)
```

Python libraries are stored and managed in a few different libraries and their dependencies are not regulated as strictly as R libraries are in CRAN. It's easier to publish a python package but it can also be more cumbersome for users because you need to manage dependencies yourself. You can download python packages using both R and Python code

```r
py_install("pandas")
```

```
## + '/Users/ty/opt/miniconda3/bin/conda' 'install' '--yes' '--prefix' '/Users/ty/opt/miniconda3/envs/ea
```

Load packages and change settings

```r
options(java.parameters = "-Xmx5G")

library(r5r)
```

```
## Please make sure you have already allocated some memory to Java by running:
##   options(java.parameters = '-Xmx2G').
## You should replace '2G' by the amount of memory you'll require. Currently, Java memory is set to -Xm
```

```r
library(sf)
```

```
## Linking to GEOS 3.10.2, GDAL 3.4.2, PROJ 8.2.1; sf_use_s2() is TRUE
```

```r
library(data.table)
library(ggplot2)
library(interp)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:data.table':
##
##     between, first, last

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(osmdata)
```

```
## Data (c) OpenStreetMap contributors, ODbL 1.0. https://www.openstreetmap.org/copyright
```

```
library(ggthemes)
library(sf)
library(data.table)
library(ggplot2)
library(akima)
```

```
##
## Attaching package: 'akima'

## The following objects are masked from 'package:interp':
##
##     aspline, bicubic, bicubic.grid, bilinear, bilinear.grid,
##     franke.data, franke.fn, interp, interp2xyz, interpp
```

```
library(dplyr)
library(raster)
```

```
## Loading required package: sp

##
## Attaching package: 'raster'

## The following object is masked from 'package:dplyr':
##
##     select
```

```
library(osmdata)
library(mapview)
library(cowplot)
```

```
##
## Attaching package: 'cowplot'

## The following object is masked from 'package:ggthemes':
##
##     theme_map
```

```
library(here)
```

```
## here() starts at /Users/ty/Documents/Github/pre-innovation-summit-training
```

```
library(testthat)
```

```
##
## Attaching package: 'testthat'

## The following object is masked from 'package:dplyr':
##
##     matches
```

```
import sys
sys.argv.append(["--max-memory", "5G"])

import pandas as pd
import geopandas
import matplotlib.pyplot as plt
import numpy as np
import plotnine
import contextily as cx
import r5py
import seaborn as sns
```

R and Python are two popular programming languages used for data analysis, statistics, and machine learning. Although they share some similarities, there are some fundamental differences between them. Here's an example code snippet in R and Python to illustrate some of the differences:

R Code:

```
# Create a vector of numbers from 1 to 10
x <- 1:10

# Compute the mean of the vector
mean_x <- mean(x)

# Print the result
print(mean_x)
```

```
## [1] 5.5
```

Python Code:

```
# Import the numpy library for numerical operations
import numpy as np

# Create a numpy array of numbers from 1 to 10
x = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

# Compute the mean of the array
mean_x = np.mean(x)

# Print the result
print(mean_x)
```

```
## 5.5
```

In this example, we can see that there are several differences between R and Python:

Syntax: R uses the assignment operator <- while Python uses the equals sign = for variable assignment.

Libraries: Python relies heavily on external libraries such as numpy, pandas, and matplotlib for data analysis, while R has built-in functions for many data analysis tasks.

Data types: R is designed to work with vectors and matrices, while Python uses lists and arrays. In the example above, we used the numpy library to create a numerical array in Python.

Function names: Function names in R and Python can differ significantly. In the example above, we used the mean() function in R and the np.mean() function in Python to calculate the mean of the vector/array.

These are just a few of the many differences between R and Python. Ultimately, the choice between the two languages will depend on your specific needs and preferences.

Load saved data

```
data("iris")
here()
```

```
## [1] "/Users/ty/Documents/Github/pre-innovation-summit-training"
```

```
load(file=here("2_R_and_Py_bilingualism", "data", "iris_example_data.rdata"))
objects()
```

```
## [1] "iris"   "mean_x" "x"
```

Save data

```
save(iris, file=here("2_R_and_Py_bilingualism", "data", "iris_example_data.rdata"))

write.csv(iris, file=here("2_R_and_Py_bilingualism", "data", "iris_example_data.csv"))
```

Write a function

Both R and Python are powerful languages for writing functions that can take input, perform a specific task, and return output.

```
# Define a function that takes two arguments and returns their sum
sum_r <- function(a, b) {
  return(a + b)
}

# Call the function with two arguments and print the result
result_r <- sum_r(3, 5)
print(result_r)
```

```
## [1] 8
```

```
# Define a function that takes two arguments and returns their sum
def sum_py(a, b):
    return a + b

# Call the function with two arguments and print the result
result_py = sum_py(3, 5)
print(result_py)
```

```
## 8
```

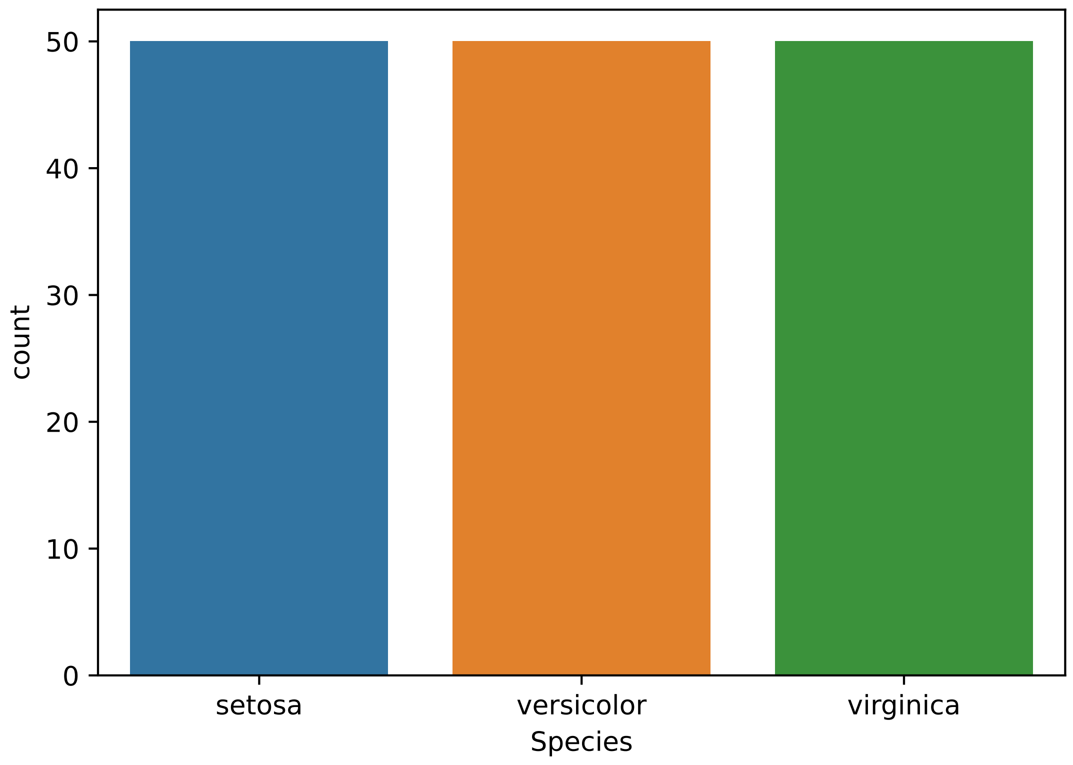
In both cases, we define a function that takes two arguments and returns their sum. In R, we use the function keyword to define a function, while in Python, we use the def keyword. The function body in R is enclosed in curly braces, while in Python it is indented.

There are a few differences in the syntax and functionality between the two approaches:

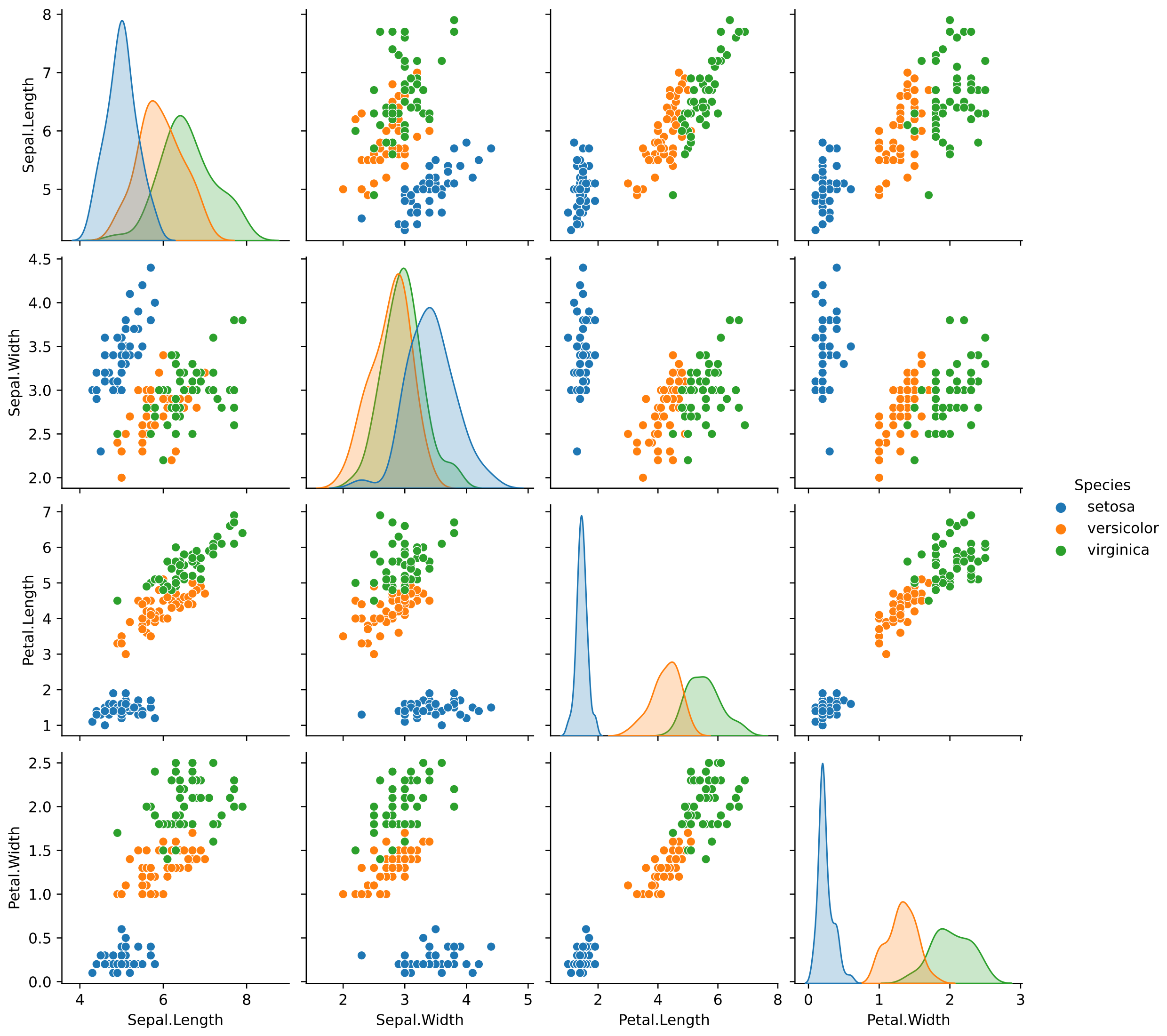
Function arguments: In R, function arguments are separated by commas, while in Python they are enclosed in parentheses. The syntax for specifying default arguments and variable-length argument lists can also differ between the two languages. Return statement: In R, we use the return keyword to specify the return value of a function, while in Python, we simply use the return statement. Function names: Function names in R and Python can differ significantly. In the example above, we used the sum_r() function in R and the sum_py() function in Python to calculate the sum of two numbers.

Data Plots

```
plt.clf()
sns.countplot(x='Species', data=r.iris)
plt.show()
```

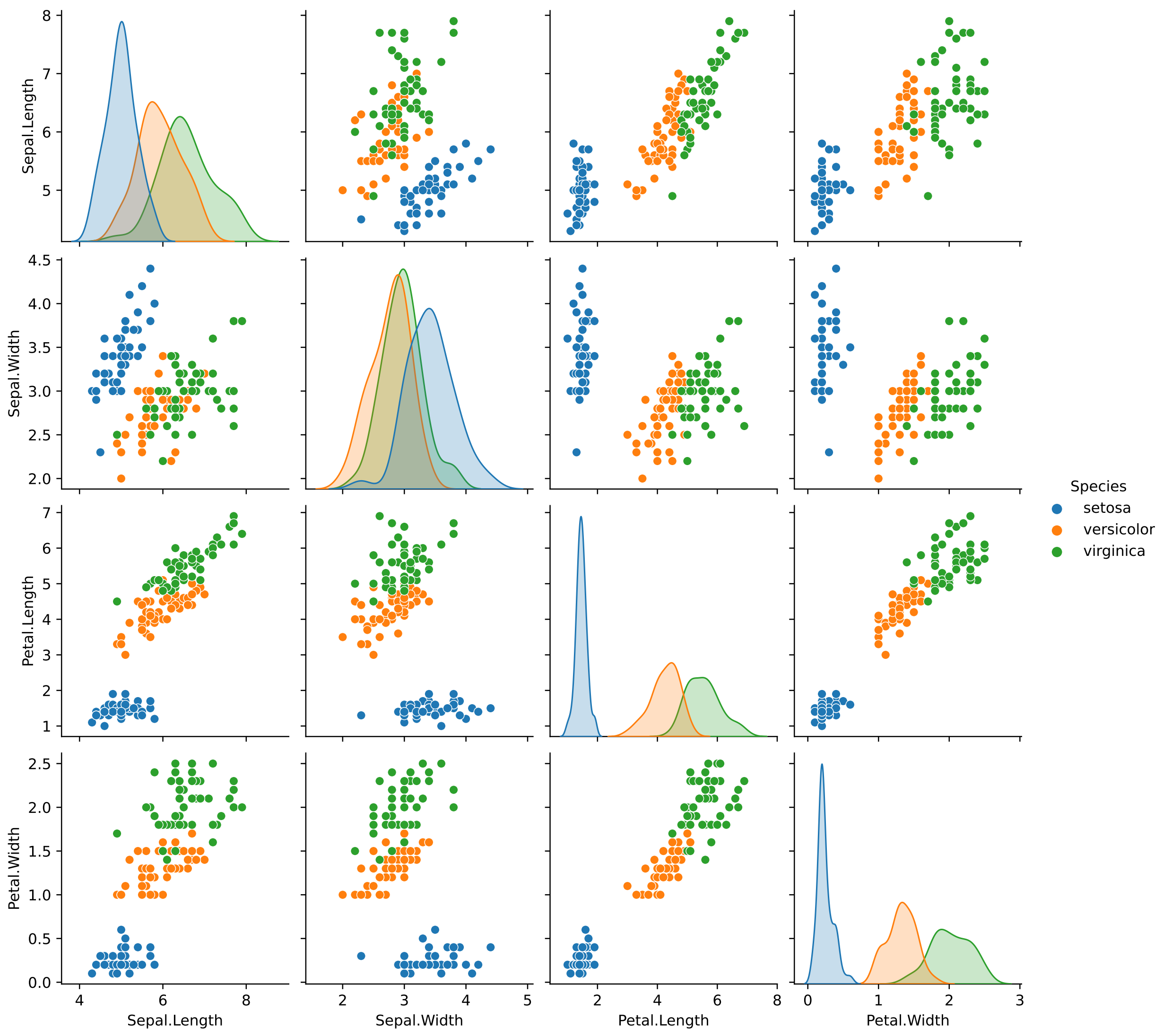


```
plt.clf()
sns.pairplot(r.iris, hue = 'Species')
```

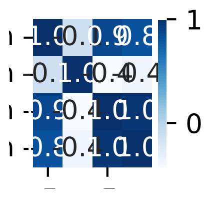


Sepal.Length
Sepal.Width
Petal.Length
Petal.Width
Species
setosa
versicolor
virginica

```
plt.show()
```



```
plt.clf()
correlation_matrix = r.iris.corr()
plt.figure(figsize=(1,1))
sns.heatmap(correlation_matrix, cbar=True, fmt='.1f', annot=True, cmap='Blues')
plt.show()
```



```
plt.savefig('Correlation Heat map')
```

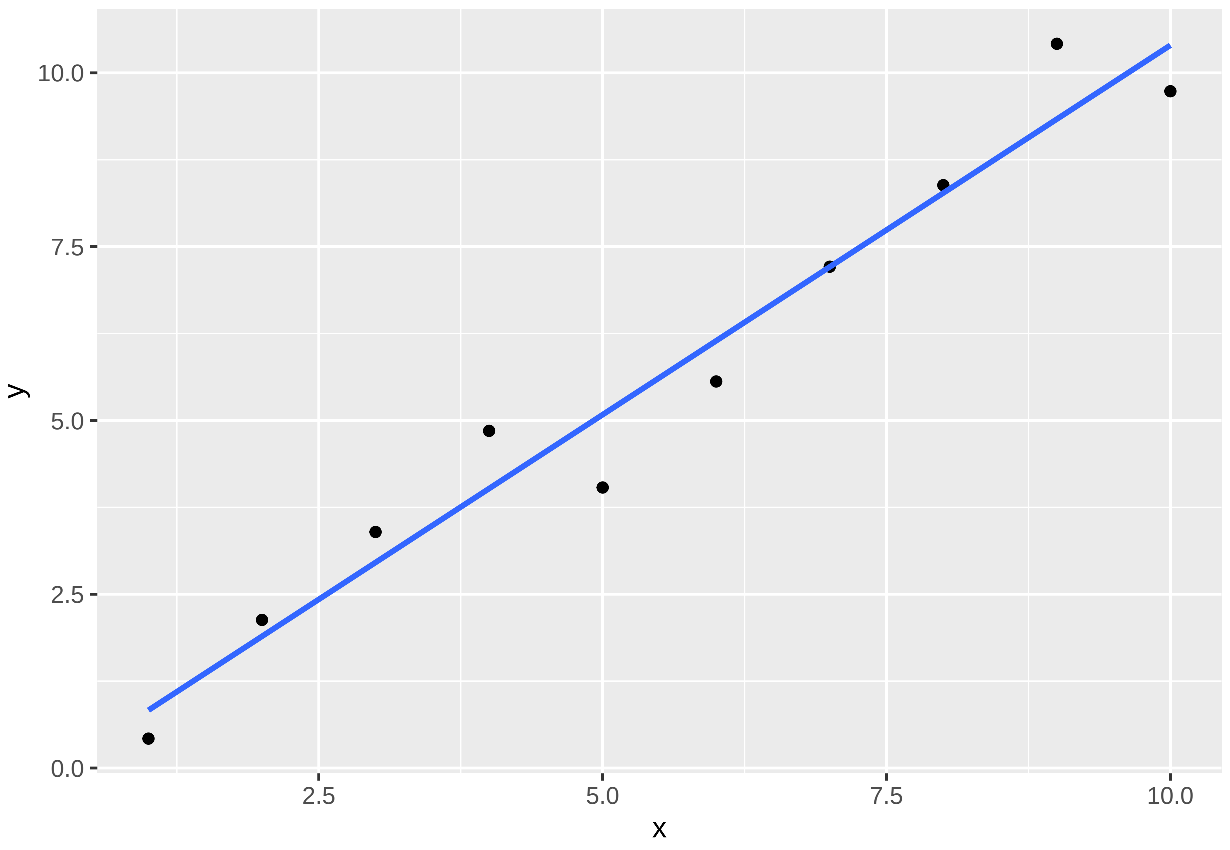
Linear regression

```
# Load the "ggplot2" package for plotting
library(ggplot2)

# Generate some sample data
x <- seq(1, 10, 1)
y <- x + rnorm(10)

# Perform linear regression
model_r <- lm(y ~ x)

# Print the model summary
summary(model_r)
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.04779 -0.54130  0.05758  0.38768  1.08397
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -0.2316     0.4933   -0.47    0.651
## x             1.0629     0.0795   13.37 9.37e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7221 on 8 degrees of freedom
## Multiple R-squared:  0.9572, Adjusted R-squared:  0.9518
## F-statistic: 178.8 on 1 and 8 DF,  p-value: 9.366e-07
```

```
# Plot the data and regression line
ggplot(data.frame(x, y), aes(x = x, y = y)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE)
```

```
## `geom_smooth()` using formula = 'y ~ x'
```



```
# Load the "matplotlib" and "scikit-learn" libraries
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Generate some sample data
import numpy as np
x = np.arange(1, 11)
y = x + np.random.normal(0, 1, 10)

# Perform linear regression
model_py = LinearRegression().fit(x.reshape(-1, 1), y)

# Print the model coefficients
print("Coefficients: ", model_py.coef_)
```

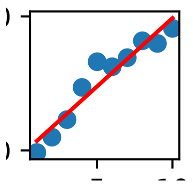
```
## Coefficients:  [1.01855091]
```

```
print("Intercept: ", model_py.intercept_)

#clear last plot
```

```
## Intercept:  -0.3142585996873386
```

```
plt.clf()
```

```
# Plot the data and regression line
plt.scatter(x, y)
plt.plot(x, model_py.predict(x.reshape(-1, 1)), color='red')
plt.show()
```



In both cases, we generate some sample data with a linear relationship between x and y, and then perform a simple linear regression to estimate the slope and intercept of the line. We then plot the data and regression line to visualize the fit.

There are a few differences in the syntax and functionality between the two approaches:

Library and package names: In R, we use the lm() function from the base package to perform linear regression, while in Python, we use the LinearRegression() class from the scikit-learn library. Additionally, we use the ggplot2 package in R for plotting, while we use the matplotlib library in Python. Data format: In R, we can specify the dependent and independent variables in the formula used for regression. In Python, we need to reshape the input data to a two-dimensional array before fitting the model. Model summary: In R, we can use the summary() function to print a summary of the model, including the estimated coefficients, standard errors, and p-values. In Python, we need to print the coefficients and intercept separately.

Retrieve data from an API