Project Based Internship

Programming Language Scripting

Programming for Data Science

Daftar Isi

Introduction

Explore the basics of these two open-source programming languages popular for Data Science, the key differences that set them apart and how to choose the right one for your situation. If you work in data science or analytics, you're probably well aware of the Python vs. R debate. Although both languages are bringing the future to life — through artificial intelligence, machine learning and data-driven innovation — there are strengths and weaknesses that come into play.

In many ways, the two open source languages are very similar. Free to download for everyone, both languages are well suited for data science tasks — from data manipulation and automation to business analysis and big data exploration. The main difference is that Python is a general-purpose programming language, while R has its roots in statistical analysis. Increasingly, the question isn't which to choose, but how to make the best use of both programming languages for your specific use cases.

What is Python?

Python is a general-purpose, object-oriented programming language that emphasizes code readability through its generous use of white space. Released in 1989, Python is easy to learn and a favorite of programmers and developers. In fact, Python is one of the most popular programming languages in the world, just behind Java and C.

Several Python libraries support data science tasks, including the following:

- Numpy for handling large dimensional arrays
- Pandas for data manipulation and analysis
- Matplotlib for building data visualizations

Plus, Python is particularly well suited for deploying machine learning at a large scale. Its suite of specialized deep learning and machine learning libraries includes tools like scikit-learn, Keras and TensorFlow, which enable data scientists to develop sophisticated data models that plug directly into a production system. Then, Jupyter Notebooks are an open source web application for easily sharing documents that contain your live Python code, equations, visualizations and data science explanations.

What is R?

R is an open source programming language that's optimized for statistical analysis and data visualization. Developed in 1992, R has a rich ecosystem with complex data models and elegant tools for data reporting. At last count, more than 13,000 R packages were available via the Comprehensive R Archive Network (CRAN) for deep analytics.

Popular among data science scholars and researchers, R provides a broad variety of libraries and tools for the following:

- Cleansing and prepping data
- Creating visualizations
- Training and evaluating machine learning and deep learning algorithms

R is commonly used within RStudio, an integrated development environment (IDE) for simplified statistical analysis, visualization and reporting. R applications can be used directly and interactively on the web via Shiny.

The main difference between R and Python: Data analysis goals

The main distinction between the two languages is in their approach to data science. Both open source programming languages are supported by large communities, continuously extending their libraries and tools. But while R is mainly used for statistical analysis, Python provides a more general approach to data wrangling.

Python is a multi-purpose language, much like C++ and Java, with a readable syntax that's easy to learn. Programmers use Python to delve into data analysis or use machine learning in scalable production environments. For example, you might use Python to build face recognition into your mobile API (Application Programming Interface) or for developing a machine learning application.

R, on the other hand, is built by statisticians and leans heavily into statistical models and specialized analytics. Data scientists use R for deep statistical analysis, supported by just a few lines of code and beautiful data visualizations. For example, you might use R for customer behavior analysis or genomics research.

Other key differences

- Data collection: Python supports all kinds of data formats, from comma-separated value (CSV) files to JSON (JavaScript Object Notation) sourced from the web. You can also import SQL tables directly into your Python code. For web development, the Python requests library lets you easily grab data from the web for building datasets. In contrast, R is designed for data analysts to import data from Excel, CSV and text files. Files built in Minitab or in SPSS (Statistical Package for the Social Sciences) format can also be turned into R data frames. While Python is more versatile for pulling data from the web, modern R packages like Rvest are designed for basic web scraping.

- Data exploration: In Python, you can explore data with Pandas, the data analysis library for Python. You're able to filter, sort and display data in a matter of seconds. R, on the other hand, is optimized for statistical analysis of large datasets, and it offers a number of different options for exploring data. With R, you're able to build probability distributions, apply different statistical tests, and use standard machine learning and data mining techniques.
- Data modeling: Python has standard libraries for data modeling, including Numpy for numerical modeling analysis, SciPy for scientific computing and calculations and scikit-learn for machine learning algorithms. For specific modeling analysis in R, you'll sometimes have to rely on packages outside of R's core functionality. But the specific set of packages known as the Tidyverse make it easy to import, manipulate, visualize and report on data.
- Data visualization: While visualization is not a strength in Python, you can use the Matplotlib library for generating basic graphs and charts. Plus, the Seaborn library allows you to draw more attractive and informative statistical graphics in Python. However, R was built to demonstrate the results of statistical analysis, with the base graphics module allowing you to easily create basic charts and plots. You can also use ggplot2 for more advanced plots, such as complex scatter plots with regression lines.

Python vs. R: Which is right for you?

Choosing the right language depends on your situation. Here are some things to consider:

- Do you have programming experience? Thanks to its easy-to-read syntax, Python has a learning curve that's linear and smooth. It's considered a good language for beginning programmers. With R, novices can be running data analysis tasks within minutes. But the complexity of advanced functionality in R makes it more difficult to develop expertise.
- What do your colleagues use? R is a statistical tool used by academics, engineers and scientists without any programming skills. Python is a production-ready language used in a wide range of industry, research and engineering workflows.
- What problems are you trying to solve? R programming is better suited for statistical learning, with unmatched libraries for data exploration and experimentation. Python is a better choice for machine learning and large-scale applications, especially for data analysis within web applications.
- How important are charts and graphs? R applications are ideal for visualizing your data in beautiful graphics. In contrast, Python applications are easier to integrate in an engineering environment.
- Note that many tools, such as Microsoft Machine Learning Server, support both R and Python. That's why most organizations use a combination of both languages, and the R vs. Python debate is all for naught. In fact, you might conduct early-stage data analysis and exploration in R and then switch to Python when it's time to ship some data products.

Scripting Languages for Data Science

In addition to programming languages, data scientists also use scripting languages. Scripting languages are interpreted languages that are typically used to automate tasks. The most popular scripting languages for data science are Python, Bash, and SQL. Python is a general-purpose scripting language that is easy to learn and use. It is also very powerful, making it suitable for a wide variety of data science tasks. Bash is a Unix shell scripting language that is used to automate tasks on Unix and Linux systems. SQL is a database query language that is used to interact with databases. Scripting languages empower Data Scientists to extract insights from raw data through effective data manipulation and analysis. With Python's Pandas library, for instance, data can be cleansed, transformed, and aggregated effortlessly. This ability to reshape data is essential for creating a consistent and structured foundation for analysis. By leveraging Pandas' intuitive syntax, Data Scientists can filter, sort, and group data with ease, enabling them to uncover patterns and trends.

In addition to data manipulation, scripting languages offer a rich ecosystem of statistical and machine learning tools. Libraries such as scikit-learn in Python and caret in R provide pre-built algorithms for classification, regression, clustering, and more. Through scripting, Data Scientists can experiment with various models, fine-tuning parameters and evaluating performance metrics to select the best-fit solution for their data.

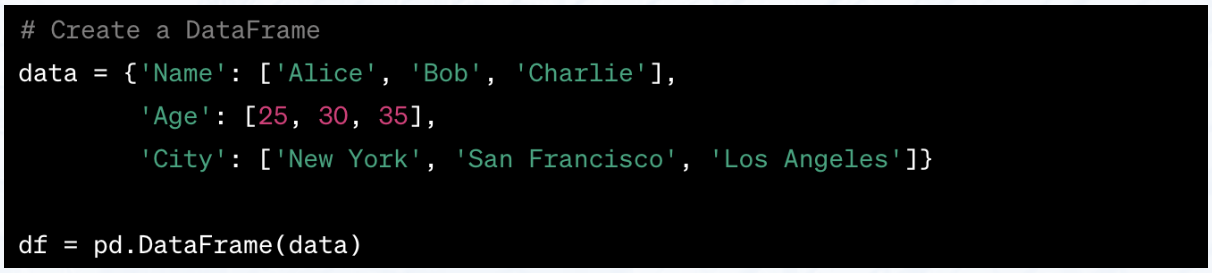
Why Use Programming and Scripting Languages for Data Science?

There are several reasons why data scientists use programming and scripting languages. First, these languages allow data scientists to automate tasks. This can save a lot of time and effort, especially when working with large datasets. Second, these languages allow data scientists to write code that is reusable and portable. This means that the code can be used on different computers and with different datasets. Third, these languages allow data scientists to create custom tools and applications. This can be useful for solving specific problems or for conducting research.

CRUD Dataframe and Table, Merge, Export, Import

Performing CRUD (Create, Read, Update, Delete) operations on a DataFrame in Python is a common task, especially when working with data analysis and manipulation. Additionally, merging, exporting, and importing data are also essential operations. Below is a guide on how to perform these operations using the Pandas library in Python.

Create Dataframe



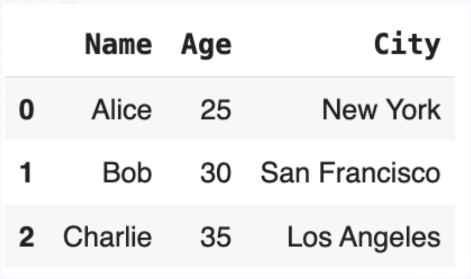
```python
# Create a DataFrame
data = {'Name': ['Alice', 'Bob', 'Charlie'],
        'Age': [25, 30, 35],
        'City': ['New York', 'San Francisco', 'Los Angeles']}

df = pd.DataFrame(data)
```

Figure 1. Create Dataframe

Read Dataframe

```
# Read data from DataFrame
print(df.head())  # Display the first 5 rows
```

Output:



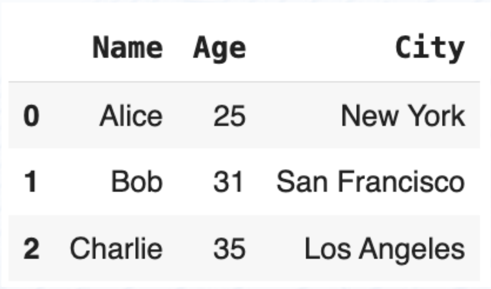
	Name	Age	City
0	Alice	25	New York
1	Bob	30	San Francisco
2	Charlie	35	Los Angeles

Figure 2. Read Dataframe

Update Dataframe

```
# Update data in DataFrame
df.loc[1, 'Age'] = 31  # Update Bob's age
print(df)
```

Output:



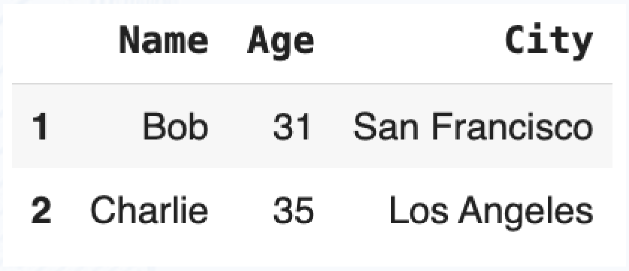
	Name	Age	City
0	Alice	25	New York
1	Bob	31	San Francisco
2	Charlie	35	Los Angeles

Figure 3. Update Dataframe

Delete Dataframe

```
# Delete a row from DataFrame
df = df.drop(0)  # Delete the first row
print(df)
```

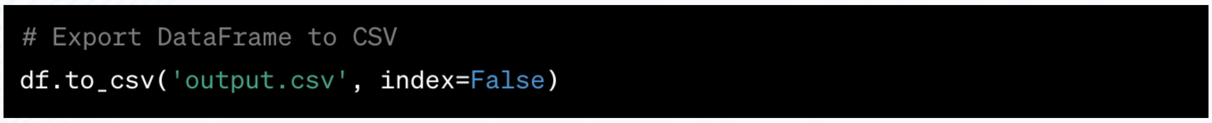
Output:



	Name	Age	City
1	Bob	31	San Francisco
2	Charlie	35	Los Angeles

Figure 4. Delete Dataframe

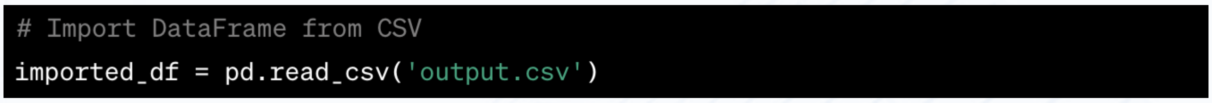
Export Dataframe



```
# Export DataFrame to CSV
df.to_csv('output.csv', index=False)
```

Figure 5. Export Dataframe

Import Dataframe



```
# Import DataFrame from CSV
imported_df = pd.read_csv('output.csv')
```

Figure 6. Import Dataframe

Function and Class

Object-Oriented Programming (OOP)

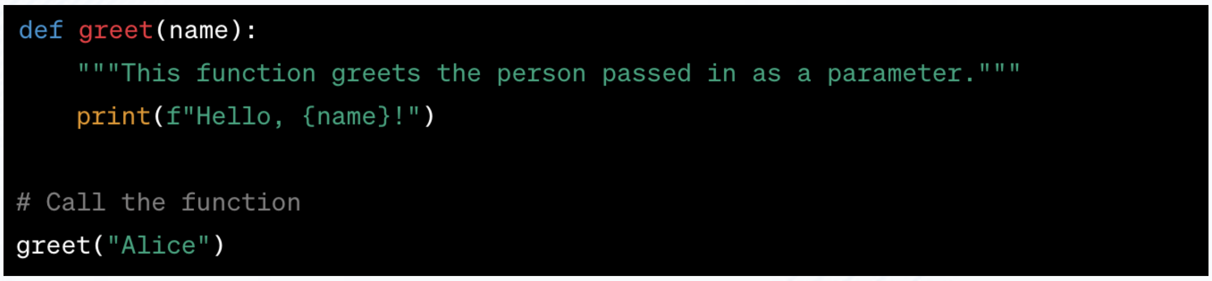
Python is an object-oriented programming (OOP) language, and both functions and classes play a crucial role in OOP. Functions are used for procedural programming, while classes support the creation of objects and encapsulation of data and behavior.

Understanding and using functions and classes are essential for writing clean, modular, and maintainable code. They provide a way to organize and structure code logically, making it easier to manage and extend.

Functions

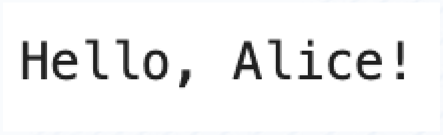
A function in Python is a block of reusable code that performs a specific task. Functions are defined using the def keyword, followed by the function name, parameters in parentheses, and a colon. The function body is indented.

Example Functions:



```python
def greet(name):
    """This function greets the person passed in as a parameter."""
    print(f"Hello, {name}!")

# Call the function
greet("Alice")
```

Output:



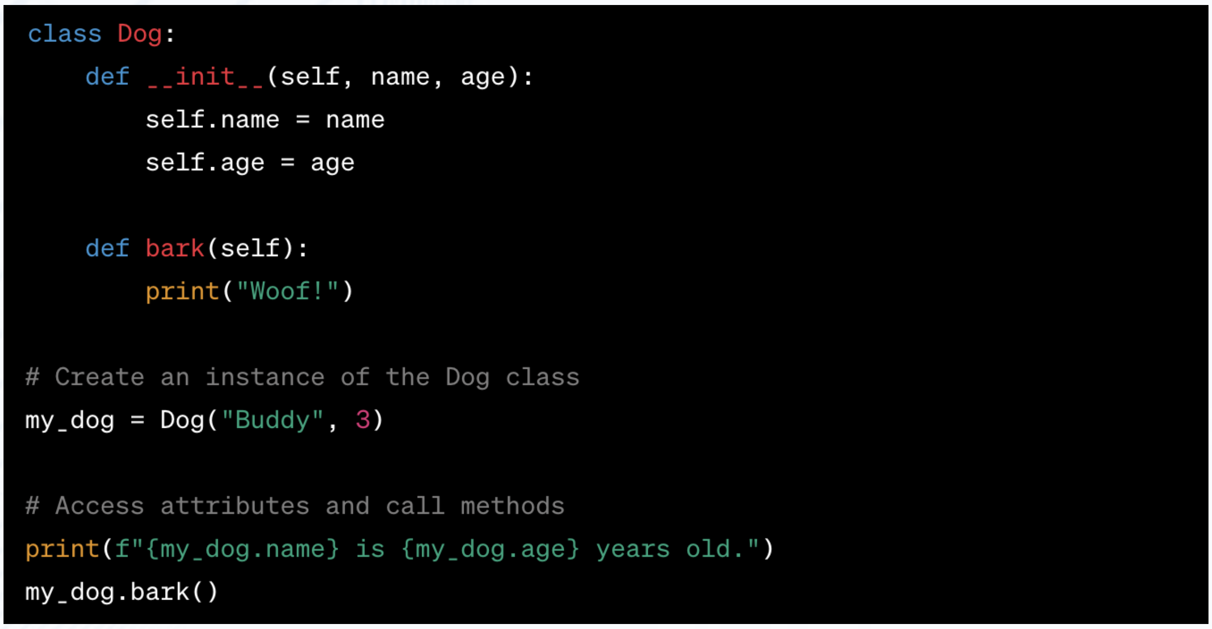
```
Hello, Alice!
```

Figure 7. Example Functions

In the example above, greet is a simple function that takes a name parameter and prints a greeting. Functions can also return values using the return statement.

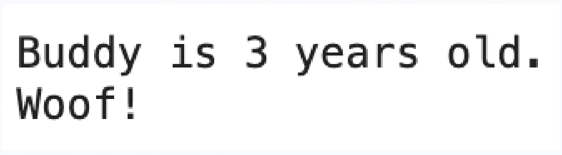
Classes

A class is a blueprint for creating objects. Objects are instances of a class, and classes define attributes (characteristics) and methods (functions) that operate on those attributes. Classes are defined using the class keyword.



```python
class Dog:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def bark(self):
        print("Woof!")

# Create an instance of the Dog class
my_dog = Dog("Buddy", 3)

# Access attributes and call methods
print(f"{my_dog.name} is {my_dog.age} years old.")
my_dog.bark()
```

Output:



```
Buddy is 3 years old.
Woof!
```

Figure 8. Example Class

In this example, Dog is a class with an __init__ method (constructor) that initializes the attributes name and age. The bark method is a behavior associated with the Dog class.

Use Case

Background and Problem Statement

You are a data scientist at ID/X Partners doing a data science project. The project you are working on is related to creating machine learning to predict whether customers will churn or not.

Create a function to evaluate the machine learning model that has been created. Functions have 2 parameters input (prediction result and actual result) and must provide the output for accuracy and precision metrics!

Solution

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay

def classification_eval (actual, predicted, name):
        cm = confusion_matrix(actual, predicted)
        tp = cm[1][1]
        tn = cm[0][0]
        fp = cm[0][1]
        fn = cm[1][0]

        accuracy = round((tp+tn) / (tp+tn+fp+fn) * 100, 2)
        precision = round((tp) / (tp+fp) * 100, 2)

        print('Evaluation Model:', name)
        print(cm)
        print('Accuracy :', accuracy, '%')
        print('Precision :', precision, '%')
```

This function evaluates the classification based on the predictions and actual values (ground truth) provided. This syntax has several steps:

Calculating Confusion Matrix: Using predicted and actual values, this function calculates the confusion matrix (cm) using the confusion_matrix function.

Calculating Evaluation Metrics: After getting the confusion matrix, this function calculates several evaluation metrics such as:

Accuracy: The accuracy of a classification model, calculated as the proportion of correct predictions overall.
Precision: The proportion of correct positive predictions out of all positive predictions made.

Displaying Evaluation Results: After calculating the metrics above, this function prints the model evaluation results, including confusion matrix, accuracy, precision, recall, and F1 score.

References

https://www.ibm.com/cloud/blog/python-vs-r

https://www.python.org/about/gettingstarted/

http://www.sthda.com/english/wiki/r-basics-quick-and-easy

https://towardsdatascience.com/python-basics-for-data-science-6a6c987f2755

https://medium.com/datactw/a-complete-introduction-to-r-for-data-science-1858c69f76b0

https://towardsdatascience.com/getting-started-with-r-programming-2f15e9256c9

https://r4ds.had.co.nz/index.html

https://dplyr.tidyverse.org/

https://datacarpentry.org/R-ecology-lesson/03-dplyr.html

https://medium.com/analytics-vidhya/python-data-manipulation-fb86d0cdd028