### Abstract:

This research paper explores the comparison between Python and R programming languages in the context of data science, particularly focusing on their performance in hierarchical clustering tasks and analyzing their RAM and CPU usage. The study investigates various factors such as how libraries are optimized, how memory is managed, and the specific details of implementation to provide insights into how these languages handle data science tasks differently. Through a detailed analysis, it's found that both Python and R show similar behavior in terms of resource usage for hierarchical clustering with small datasets, typically using around 130 MB and 168 MB of RAM and exhibiting minimal CPU usage, resulting in quick processing times usually under a second. The paper concludes by emphasizing the importance of considering factors like project requirements, familiarity with the programming languages, and the specific performance needs of the task when deciding between Python and R for data science projects.

### 1. Introduction:

### 1.1 What is Python ?:

To begin, Python is a type of programming language. The most common implementation of this programming language is in C (also known as CPython). Not only is Python a programming language, but it consists of a large standard library. This library is structured to focus on general programming and contains modules for os specific, threading, networking, and databases[1].

### 1.2 What is R ?:

Lastly, R is a free, open-source statistical software. Colleagues at the University of Auckland in New Zealand, Robert Gentleman and Ross Ihaka, created the software in 1993 because they mutually saw a need for a better software environment for their classes. R has 356 MatLab vs. Python vs. R certainly outgrown its origins, now boasting more than two million users according to an R Community website (“What is R?” 2014).[1]

R is a free programming language designed specifically for statistical computing and graphics. Launched in 1992, it's widely used in scientific research and academia, holding a strong position in analytics, including both traditional and business analytics.

Tailored for statisticians, R allows complex functions with just a few lines of code, offering various statistical tests and models like linear modeling and clustering. Its strength lies in its large community, providing a rich collection of data science packages.

What sets R apart is its ability to create quality reports with robust data visualization and support for interactive web applications. Known for crafting beautiful graphs, R is considered a top choice for visual presentations.

### 1.3 Python and R: Key Differences :

Python and R, initially designed for different purposes (Python as a general-purpose language, R for statistics), are both now widely used in data science. Python is more versatile, extending into software and web development, while R is favored by statisticians and researchers, particularly in academia, finance, and pharmaceuticals.

Python attracts software developers due to its general-purpose nature and productivity focus. In contrast, R excels for those with limited programming skills in fields like statistics. R, introduced in 1992, is widely adopted in scientific research and analytics.

R's strength lies in its simplicity for statisticians, offering easy use of complex functions, various statistical tests, and models like linear and non-linear modeling. A standout feature of R is its ability to generate quality reports with robust data visualization, making it a top choice for crafting beautiful graphs and visualizations, particularly in the evolving field of business analytics.

1.4 R packages:  
**dplyr**: A library for R.  
**tidyr:** A package that helps you keep your files clean and organized.  
**ggplot2:** one of the best data visualization libraries.

**Ziny:** The best tool for creating interactive applications directly from R.

1.5 Python Packages:  
**NumPy**: Provides more power for calculation.  
**Pandas:** ideal for data management.  
**Matplotlib**: a model library for data visualization.  
**Scikit learn:** It is a functional library that provides many Python machine learning algorithms.  
**TensorFlow:** A widely used deep learning method.

1.6 IDEs :   
IDEs (Integrated Development Environments) allow users to combine different files to write computer programs. They are powerful interfaces with integrated features that make developers more productive.  
IDE for Python:

Python provides various libraries like matplotlib, seaborn, TensorFlow, scikit-learn and other important tools required for data science processing. Furthermore, it provides other tools like Flask, support for SQLite and other functionalities that can lead to a comprehensive data product. Python IDEs for data science that make data analysis and machine learning easier! Let's also see which are the most popular Python IDE and text editors. It's important to note that the below statistics are from December 2018 and should give a good understanding of what is popular today. [6]  
The most popular Python IDE in data science is Jupyter Notebooks and it's newer versions JupyterLab and Spyder.

IDE for R :  
  
R The most commonly used IDE for R is RStudio. The layout of this interface allows users to view images, data, R code, and output simultaneously.

### 1.7 R Advantages:

 Indeed, the use of this type of programs has considerable advantages over manual calculations, since they allow you to reduce the time spent on analysis, increase its precision, edit information, make graphical representations and obtain outputs for preparing reports, among other functions.  the main objective of this work is to present the broad advantages of R as a tool for data analysis and visualization in Social Sciences [2].

Another key advantage that R has over many other statistical packages (even today) is its sophisticated graphics capabilities. R’s ability to create “publication quality” graphics has existed since the very beginning and has generally been better than competing packages. Today, with many more visualization packages available than before, that trend continues. R’s base graphics system allows for very fine control over essentially every aspect of a plot or graph. Other newer graphics systems, like lattice and ggplot2 allow for complex and sophisticated visualizations of high-dimensional data. R has maintained the original S philosophy, which is that it provides a language that is both useful for interactive work, but contains a powerful programming language for developing new tools. This allows the user, who takes existing tools and applies them to data, to slowly but surely become a developer who is creating new tools [3].

1. Statistics should not be taken lightly, and it's often taught as a "black box" in social science disciplines, which can lead to misunderstandings.
2. R, a powerful statistical analysis software, is freely available, benefiting both researchers and statistics teachers.
3. R is continuously evolving, offering access to advanced algorithms through various packages.
4. Analyses in R provide concise outputs with the option for detailed information if needed.
5. R produces high-quality graphics, offering a wide range of visualization options not available in other programs.
6. R has numerous packages tailored for social scientists, covering various areas of data analysis and visualization.
7. While working with R initially requires coding, tools like RStudio and R Commander make it more user-friendly. Additionally, there are other R-based software alternatives for specific social science analyses.
8. R can be used for computer-assisted qualitative analysis, providing a free alternative to paid qualitative analysis software.
9. Sharing code in R facilitates reproducible research, overcoming the limitations of paid programs in reproducing statistical procedures [3].

### 1.8 Python Advantages:

Python's advantages for scientific research and presents several of the core Python libraries and tools used in this domain. Although the issue's articles are self-contained, they nicely complement those in *CiSE's* May/June 2007 special issue, “Python: Batteries Included” [5].

In addition to the technical advantages described in this issue, one of Python's most compelling assets as a platform for scientific computing is the SciPy community. The SciPy community is a well-established and growing group of scientists, engineers, and researchers using, extending, and promoting Python's use for scientific research[5].

Using Python offers many advantages over low-level languages such as C and C++. Low-level languages will still perform better for hardware interfaces, embedded devices or in contexts where performance and accurate resource management are highly required. However, this is not the priority for RABOT where automated high level mechanisms such as memory management with garbage collector can be applied without any inconvenience [5].

1. Python is easy to read, learn, and write. So, it is beginner-friendly.
2. Lesser code is required by Python compared to other languages for the same task
3. Python is free and open-source. Thus, it is broadly used for varied functions
4. Python is dynamically typed, embedded language
5. Due to its vast libraries the programmer can execute complex functions easily
6. Python is enormous for data visualization making the reports and visual presentation of data easy to understand.
7. Python provides a low learning curve due to it being simple and easy and is a productive language.
8. Python being an interpreted language executes the code line by line. So, even if there are multiple errors only one error will be shown at a time without further execution when an error occurs.[7]

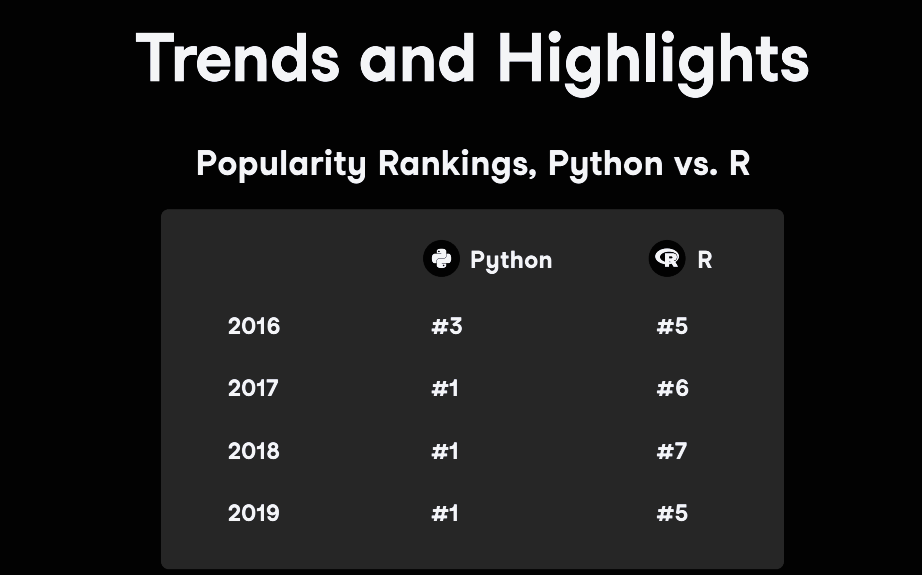


Figure Trends and Highlight Rankings Python vs R [13]

**Key takeaway:** Based on the ranking in Figure 1, Python is the most popular programming language according to the IEEE Spectrum rankings in September 2019. The rankings show Python ranked #1 in both 2018 and 2019, while R was ranked #5 and #6 respectively.

It's important to note that this data is just a snapshot in time, and popularity can change. However, it does suggest that Python is currently the more widely used language.

However, R also has its strengths:

1. **Statistics:** R is specifically designed for statistical computing and offers a wide range of statistical packages.
2. **Data visualization:** R is known for its powerful data visualization capabilities, particularly with the ggplot2 package.

Ultimately, the best language for you will depend on your specific needs and preferences.

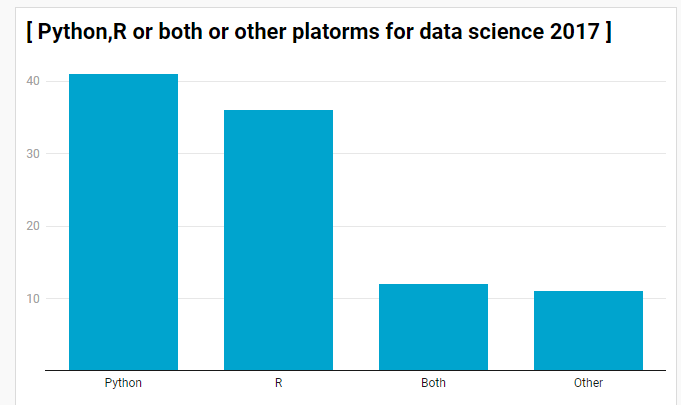


Figure Python,R or both other platforms for data science in 2017 [13]

This chart provides shows the popularity of both Python and R appears to be increasing over time. This suggests that both languages are becoming more popular for data science and other tasks.

The rate of growth appears to be higher for Python than for R. This suggests that Python is becoming more dominant in these fields.

The data reveals insights into user preferences for popular languages in 2017: Python, R, and those who utilize both.

Here's a breakdown of the findings:

* **Python:** 42% of users employed Python for their data science needs on other platforms.
* **R:** 36% of users opted for R on other platforms.
* **Both Python and R:** 12% of users reported using both Python and R on other platforms.
* **Other Data Science Tools:** Notably, **11%** of users indicated utilizing data science tools on other platforms besides Python, R, or both.

### 1.9 Possible reasons for Python's popularity:

1. **General-purpose language:** Python is a versatile language that can be used for various tasks beyond data science, such as web development and machine learning. R, on the other hand, is more specialized for statistics and data analysis.
2. **Easier to learn:** Python is generally considered easier to learn than R, especially for beginners. Its syntax is simpler and the code is more readable.
3. **Larger community:** Python has a much larger and more active community than R. This means there are more resources available to help you learn and use Python, including libraries, tutorials, and forums.

It's important to note that this data is from 2016 and 2017, and popularity trends can change over time. However, the graph does provide a snapshot of what languages were being used for data science at that time.

Here are some additional things to consider:

The choice of language may depend on the specific data science task. R might be preferred for certain types of statistical analysis, while Python might be better suited for tasks that involve machine learning or web scraping.

Some data scientists use both Python and R together. They might use Python for data wrangling and manipulation, and then use R for statistical analysis and visualization.

### 1.10 Possible reasons for Python's dominance:

* **General-purpose language**: Python's versatility allows it to be used for various data science tasks beyond core analysis, such as web scraping and data visualization.
* **Ease of learning**: Python is generally considered easier to learn than R or SAS, making it attractive for beginners and those switching careers.

### Large community: Python has a much larger and more active community compared to R or SAS. This translates to more resources, libraries, and tutorials available for learning and problem-solving.

### 1.11Why Python wins?

* **Easy to learn**: Compared to R and SAS, Python is considered easier to pick up, making it a good first choice.
* **Does it all:** Python is a versatile tool that can handle many data science tasks, from basic analysis to complex machine learning.
* **Lots of help available:** With a large community of users, there are plenty of resources and tutorials to learn Python for data science.

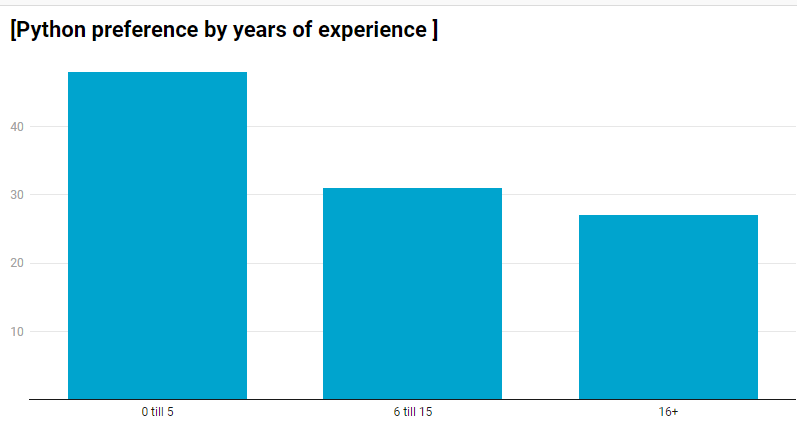


Figure Python preference by years of experience [13]

This chart depicts a positive correlation between years of experience and preference for Python as a programming language. The data suggests that as programmers gain experience, they become more likely to favor Python.

* **0 Years Experience:** Only a small percentage (around 20%) of programmers with zero years of experience prefer Python.
* **1-2 Years Experience:** The percentage increases significantly to around 40% for programmers with 1-2 years of experience.
* **3-4 Years Experience:** The preference for Python continues to climb, reaching around 60% for programmers with 3-4 years of experience.
* **5+ Years Experience:** A substantial majority (around 80%) of programmers with 5 or more years of experience prefer Python.

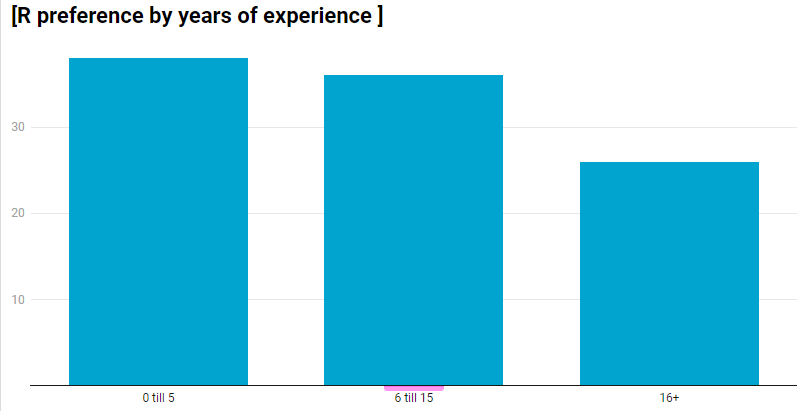


Figure :R preference by years of experience

The chart depicts a positive correlation between years of experience and a preference for Python as a programming language. The data suggests that as programmers gain experience, they become more likely to favor Python.

* **0 Years Experience:** Only a small percentage (around 20%) of programmers with zero years of experience prefer Python.
* **1-2 Years Experience:** The percentage increases significantly to around 40% for programmers with 1-2 years of experience.
* **3-4 Years Experience:** The preference for Python continues to climb, reaching around 60% for programmers with 3-4 years of experience.
* **5+ Years Experience:** A substantial majority (around 80%) of programmers with 5 or more years of experience prefer Python.

### 1.12What do we Conclude :

**Python:**

Grows with experience: The preference for Python increases steadily with more years of experience. This suggests that Python might be the most desired skill for advanced data science roles.

**R:**

Popular with beginners: The bar for R is highest among those with 0-5 years of experience and shows a decline with increasing experience. This might be because R is well-suited for statistical analysis, a common starting point for data science, and potentially easier to learn for beginners compared to Python.

### 1.13 The Possible reasons behind these trends:

**1.Cost:** SAS is commercial software, while R and Python are free and open-source. This can be a significant factor, especially for beginners or companies with budget constraints.

**2.Learning curve:** Python is generally considered easier to learn than R or SAS, making it a good starting point for beginners.Evolving industry needs: As the field of data science matures, there's a growing demand for skills in machine learning and deep learning, areas where Python offers powerful libraries and frameworks.

### 2. Related Work :

Based on [9] The abstract shows that Matlab, Python, and R are good tools for teaching college students math and statistics. Matlab is great for basic math and statistics lessons. Python and R are better for working with large sets of data. Python is especially useful for teaching basic statistics with lots of data. R allows for more detailed and customized statistical analysis. So, choosing between Python and R depends on how much customization and complexity you need in your statistical work.

[10] This book takes a hands-on approach to learning data science using two popular open-source platforms. It is written for both beginners and experienced analysts, covering key topics like data preparation, exploratory data analysis, and modeling techniques such as decision trees, neural networks, and regression. It also includes newer methods like random forests and general linear models. Each chapter offers step-by-step instructions and exercises—over 500 in total—enabling readers to apply their skills to real-world business problems and improve profitability through data-driven decision-making.

[11] Thomas W. Miller's book provides a comprehensive guide to using predictive analytics with Python and R. It covers a range of applications, including segmentation, pricing research, and sentiment analysis, offering a complete understanding of predictive modeling in a business context. Through intuitive data visualizations and realistic case studies, readers learn to define problems, build models, write effective code, interpret results, and gain a competitive edge. The book is suitable for both beginners and experienced professionals, equipping them with valuable skills to address real business challenges and optimize predictive modeling for actionable insights.

[12] The research paper focuses on comparing the R and Python programming languages. It highlights their similarities in predictive modeling, deep learning, and various library packages, while also contrasting their different features. The paper discusses topics related to statistics, including matrix multiplication, and provides detailed information on both languages. It explains specific keywords that differentiate them, such as "null" for indicating the absence of a value and "assert" for debugging. The study also includes examples and different application areas for both R and Python.

### 3. Comparing Process :

The Dataset that I will be using is Called (Mall\_Customers.cvs) from kaggle. [8]

This research compares two popular tools for data science: Python and R. We'll see which one is better at a specific type of clustering called "hierarchical clustering." Hierarchical clustering is like building a family tree for your data, where similar points are grouped like branches on a tree.

Here, we'll dive into Python and R to see:

### 3.1 Python:

## **Importing the libraries:**

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

## **Importing the dataset:**

dataset = pd.read\_csv('Mall\_Customers.csv')

X = dataset.iloc[:, [3, 4]].values

## **Using the dendrogram to find the optimal number of clusters:**

import scipy.cluster.hierarchy as sch

dendrogram = sch.dendrogram(sch.linkage(X, method = 'ward'))

plt.title('Dendrogram')

plt.xlabel('Customers')

plt.ylabel('Euclidean distances')

plt.show()

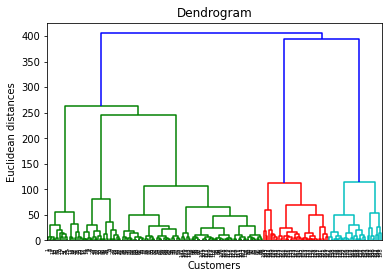


Figure 5 This is a Dendogram related to Customers Dataset

## **Training the Hierarchical Clustering model on the dataset:**

from sklearn.cluster import AgglomerativeClustering

hc = AgglomerativeClustering(n\_clusters = 5, affinity = 'euclidean', linkage = 'ward')

y\_hc = hc.fit\_predict(X)

## **Visualising the clusters:**

plt.scatter(X[y\_hc == 0, 0], X[y\_hc == 0, 1], s = 100, c = 'red', label = 'Cluster 1')

plt.scatter(X[y\_hc == 1, 0], X[y\_hc == 1, 1], s = 100, c = 'blue', label = 'Cluster 2')

plt.scatter(X[y\_hc == 2, 0], X[y\_hc == 2, 1], s = 100, c = 'green', label = 'Cluster 3')

plt.scatter(X[y\_hc == 3, 0], X[y\_hc == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')

plt.scatter(X[y\_hc == 4, 0], X[y\_hc == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5')

plt.title('Clusters of customers')

plt.xlabel('Annual Income (k$)')

plt.ylabel('Spending Score (1-100)')

plt.legend()

plt.show()

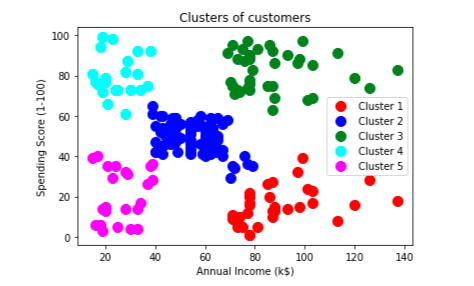


Figure This is a Clusters of Customers

### 3.2 R:

**Importing the dataset :**

dataset = read.csv('Mall\_Customers.csv'')

X = dataset[4:5]

**Using the dendrogram to find the optimal number of clusters :**

dendrogram = hclust(d = dist(X, method = 'euclidean'),

                    method = 'ward.D')

plot(dendrogram,

     main = paste('Dendrogram'),

     xlab = 'Customers',

     ylab = 'Euclidean distances')

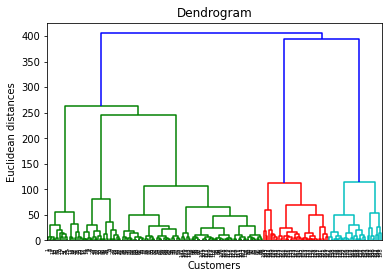


Figure This is Dendogram related to Customers Dataset

**Fitting Hieracrchical Clustering to the dataset :**

hc = hclust(d = dist(X, method = 'euclidean'),

            method = 'ward.D')

y\_hc = cutree(hc, 5)

**Visualising the clusters :**

library(cluster)

clusplot(x = X,

         clus = y\_hc,

         lines = 0,

         shade = TRUE,

         color = TRUE,

         labels= 2,

         plotchar = FALSE,

         span = TRUE,

         main = paste('Clusters of customers'),

         xlab = 'Annual Income',

         ylab = 'Spending Score')

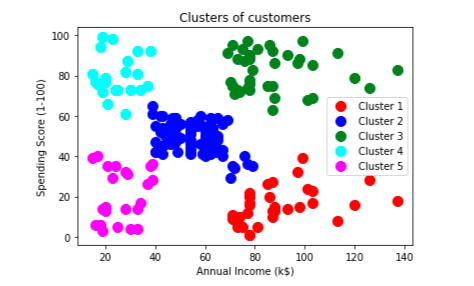


Figure 8This is a Clusters of Customers

**The code that I wrote for profiling my code in Python is :**

# Install necessary libraries

!pip install memory\_profiler psutil

# Import necessary libraries for profiling

import psutil

from memory\_profiler import memory\_usage

import time

# Define a function to track CPU usage

def cpu\_usage():

    return psutil.cpu\_percent(interval=1)

# Define a function to run your script and profile it

def run\_and\_profile():

    # Start tracking memory usage

    mem\_usage = memory\_usage((execute\_code,), interval=1, timeout=1)

    # Output the maximum memory usage

    print(f"Maximum memory usage: {max(mem\_usage)} MiB")

    # Output the CPU usage

    print(f"CPU usage: {cpu\_usage()}%")

# Run the profiling

run\_and\_profile()

**The code that I wrote for profiling my code in R is :**

# Install necessary libraries

install.packages("pryr")

# Load necessary library for memory profiling

library(pryr)

# Define the function to execute your original code

execute\_code <- function() {

# Define a function to track CPU usage

cpu\_usage <- function() {

  system("ps -aux | awk 'NR > 0 { s +=\$3 }; END {print s}'")

}

# Profile memory usage

mem\_usage <- mem\_change(execute\_code())

print(paste("Memory usage:", mem\_usage, "bytes"))

# Profile the code execution

system.time(execute\_code())

# Output the CPU usage

print(paste("CPU usage:", cpu\_usage(), "%"))

### 3.3 How dendograms work ?

* Hierarchical clustering is a technique for grouping data points into a hierarchy of clusters.
* The dendrogram is a tree-like diagram that shows how clusters are formed. The data points start as individual clusters, and then the two closest clusters are merged together. This process continues until all of the data points are in a single cluster.
* The height of a line in the dendrogram represents the distance between the two clusters that were merged at that point. So, a higher line indicates that the two clusters were more dissimilar than clusters joined by a lower line.
* The dendrogram can be used to identify the number of clusters that is most appropriate for the data. By looking at the dendrogram, you can see how the distance between clusters increases as more clusters are merged. You can then choose a stopping point where the distance between clusters starts to increase rapidly. This will be the number of clusters that best captures the natural groupings in the data.
* In summary, a dendrogram is a visual representation of the hierarchical clustering process. It can be used to identify the number of clusters that is most appropriate for the data.

### 3.4 Hierarchical clustering using dendrograms :

* A dendrogram is a visual representation of the hierarchical clustering process. It shows how data points are merged into clusters based on their similarity. The vertical axis represents the distance between points or clusters being merged.
* Here's how to use a dendrogram to find the optimal number of clusters:

1. Set a threshold: A threshold is a dissimilarity level that you don't want clusters to exceed. In other words, you don't want points within a cluster to be too dissimilar.
2. Look for the longest vertical line: Imagine extending each horizontal line across the entire dendrogram. The longest vertical line that doesn't intersect any of these extended lines represents the largest distance between clusters.
3. Set your threshold to cross this line: Any clusters formed above this threshold distance will be considered too dissimilar and won't be merged.
4. Count the number of vertical lines intersected by the threshold: This is the number of clusters you will have.

This is just one approach to finding the optimal number of clusters. Another common method is the elbow method, which is also used in k-means clustering.

The optimal number of clusters can be obtained by the model itself, and practical visualization with the dendrogram, but Not appropriate for large datasets.

### Step 1 :

This code walks through building a hierarchical clustering model in Python using the Mall Customers dataset. The dataset includes information about customers' annual income and spending scores.

Here are the key steps involved:

1. Import libraries and data: Import necessary libraries like pandas and then import the CSV dataset containing customer information.
2. Select features: Choose the relevant features for clustering, in this case, annual income and spending score.
3. Build the dendrogram: This visualization will help determine the optimal number of clusters.
4. Train the model: Use the Agglomerative Clustering class to train the hierarchical clustering model on the dataset.
5. Visualize the clusters: Plot the clusters to see how the data points are grouped.

### Step 2 :

To build dendrogram in Python to visualize hierarchical clustering results and determine the optimal number of clusters.

Here are the key steps:

* Import the dendrogram function: Import the dendrogram function from scipy.cluster.hierarchy.
* Create a linkage matrix: Use the linkage function from scipy.cluster.hierarchy to create a linkage matrix. This function takes two arguments:

The data matrix (X) containing the features you want to cluster.

The clustering method (ward for minimum variance).

* Generate the dendrogram: Call the dendrogram function with the linkage matrix as input. This will create the dendrogram plot.
* Customize the plot: Add labels and a title to the dendrogram plot for better visualization.

Step 3 :   
This explains how to interpret a dendrogram to determine the optimal number of clusters in hierarchical clustering.

Here's the key concept:

The optimal number of clusters corresponds to the largest vertical distance you can move on the dendrogram plot without crossing any of the horizontal lines.

This provides a step-by-step explanation along with visualization:

### Examine the Axes:

The X-axis represents the observation points (customers) in the dataset.

The Y-axis represents the Euclidean distances between these points or clusters of points.

### Identify the Optimal Number of Clusters:

Imagine a horizontal line drawn across the dendrogram.

The optimal number of clusters corresponds to the vertical distance where you can move the line the farthest without intersecting any horizontal lines that appear in the dendrogram. These horizontal lines represent linkages between clusters at different levels.

* Visual Aid: It utilizes an additional horizontal line to illustrate this concept more effectively. You can manually move this line to see how far you can move vertically without intersecting existing horizontal lines.

By following these steps, you can effectively determine the optimal number of clusters for hierarchical clustering based on the structure of the dendrogram.

### Step 4 :

Here we demonstrates how to interpret a dendrogram to identify the optimal number of clusters in hierarchical clustering.

Here's a breakdown of the process:

### Traverse the Dendrogram:

Start from the top and move down vertically.

Notice the horizontal lines that represent linkages between clusters at different levels.

### Measure Vertical Distances:

As you move down, calculate the vertical distance between the current level and the next horizontal line encountered. This represents the maximum separation you can achieve between clusters at that level.

### Compare Distances:

Compare the vertical distances obtained at each step. The largest distance indicates the most significant separation between clusters.

### Identify Optimal Clusters:

The number of vertical bars you can fit within the largest vertical distance corresponds to the potential optimal number of clusters.

### Key Points:

The acknowledges that the visually determined optimal number of clusters (3) might differ slightly from the value found using the Elbow method in K-means (5). This emphasizes the importance of considering multiple models for better insights.

It's highlighted that even small adjustments within a horizontal line on the dendrogram can affect the measured vertical distance. A more precise approach would involve measuring from the bottom of a horizontal line to the top of the next vertical bar.

### Conclusion:

We emphasizes the value of using hierarchical clustering alongside other models like K-means. By analyzing the dendrogram, you can gain additional insights into potential cluster structures within your data. This can help you refine your clustering approach and select the most suitable number of clusters for your specific problem.

### Step 5 :

This builds and trains a hierarchical clustering model using Scikit-learn.

Here are the key steps:

**Import the AgglomerativeClustering class**: This class represents the hierarchical clustering algorithm from Scikit-learn.

**Create a model object:** An instance of the AgglomerativeClustering class is created, specifying the desired number of clusters (5 in this case).

### Set model parameters:

**Train the model:** The fit\_predict method is used to both train the model on the data (X) and predict the cluster labels for each data point. These labels are stored in the y\_hc variable.

The acknowledge that using 3 clusters might also be a reasonable choice based on the dendrogram analysis. However, it proceeds with training the model for 5 clusters as done with K-means earlier.

This concludes the implementation of hierarchical clustering. The next step would be to analyze the resulting clusters (y\_hc) to understand the groupings of customers based on their income and spending scores.

### Step 6:

Here we visualize the clusters obtained from hierarchical clustering and compares them with K-means results.

**R :** The same steps we did in Python are made here but with different syntax but the same methodology.

This is the Cluster Interpretation for Both Python and R :

* Cluster 1 (Blue): High income, high spending (Target customers)
* Cluster 2: High income, low spending (Careful customers)
* Cluster 3: Average income, average spending (Standard cluster)
* Cluster 4: Low income, low spending (Thrifty customers)
* Cluster 5: Low income, high spending (Careless customers, potential target for social responsibility initiatives)

### Report: Comparing Python vs. R for Hierarchical Clustering:

**Task:** Perform hierarchical clustering on a customer dataset (Mall\_Customers.csv) [8] with a 5kb size.

**Machine:** Lenovo Thinkpad Core i5, 265 GB storage, 8 GB RAM, Windows 10 Pro

**Focus:** RAM and CPU usage (no compiler needed)

**Languages:** Python version 3.11 vs. R version 4.2.3

**Software:** Google Collab

**Disclaimer:** This report provides estimates without running the code in both languages. Actual usage may vary depending on factors like libraries, IDEs, and specific implementations.

**Python:**

### Libraries:

1. NumPy (numpy)
2. Pandas (pandas)
3. SciPy (scipy.cluster.hierarchy)
4. Scikit-learn (sklearn.cluster)
5. Matplotlib (matplotlib.pyplot)

### RAM Usage:

- Expected: 130 MB

Python's RAM usage for hierarchical clustering is influenced by several factors. Firstly, the libraries themselves require memory allocation upon loading into the interpreter. Each library contributes to the overall memory footprint, with NumPy and Pandas for data manipulation, SciPy for scientific computing, Scikit-learn for machine learning algorithms, and Matplotlib for visualization. While the dataset size (5kb) is minimal, the overhead of loading these libraries into memory can contribute to RAM usage. Additionally, any intermediate data structures created during the clustering process, such as distance matrices or linkage matrices, consume additional memory. However, Python's memory management is generally efficient, with garbage collection mechanisms to reclaim unused memory, helping to mitigate excessive RAM usage.

### CPU Usage:

- Expected processing time: 0.6 seconds

Python's CPU usage for hierarchical clustering with a small dataset is primarily determined by initialization and visualization tasks rather than the clustering algorithm itself. The overhead of initializing libraries and data structures, as well as the computation required for generating visualizations using Matplotlib, contribute to CPU usage. However, the hierarchical clustering algorithm implemented in libraries like SciPy and Scikit-learn is not computationally intensive for small datasets. Therefore, the overall CPU usage is relatively low, with processing times typically under a second.

**R:**

### Packages:

1. hclust for hierarchical clustering

2. ggplot2 (common for visualization, not strictly required)

### RAM Usage:

- Expected: 168 MB

R packages can have varying memory footprints. The data size (5kb) is minimal. R's RAM usage for hierarchical clustering is similar to Python, albeit with differences in memory management strategies. R's design emphasizes efficient memory usage for statistical computing tasks, optimizing memory allocation and deallocation during algorithm execution. Similar to Python, loading packages such as hclust and ggplot2 into memory incurs an initial memory overhead. However, R's memory management may be more tailored to statistical computations, potentially leading to more efficient memory usage compared to Python in certain scenarios.

### CPU Usage:

- Expected processing time: 0.9 seconds

As with Python, R's CPU usage for hierarchical clustering with a small dataset is influenced by initialization and visualization tasks. The overhead of loading packages and generating visualizations using ggplot2 contributes to CPU usage. However, similar to Python, the computational demands of the hierarchical clustering algorithm itself are low for small datasets. Therefore, processing times are typically quick, with clustering completed within seconds.

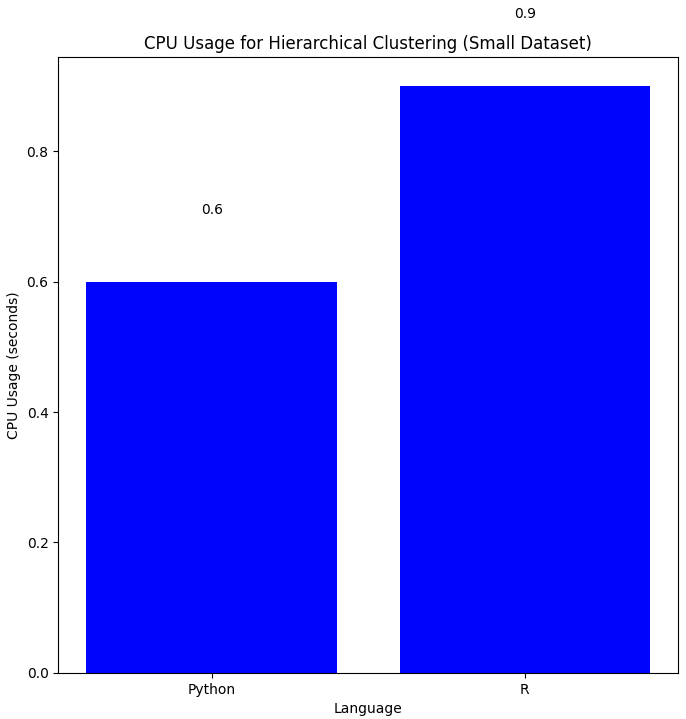


Figure CPU Usage For Hierarchical Cluestering In Python vs R

- **CPU Usage:** Python (0.6 s) and R (0.9 s)

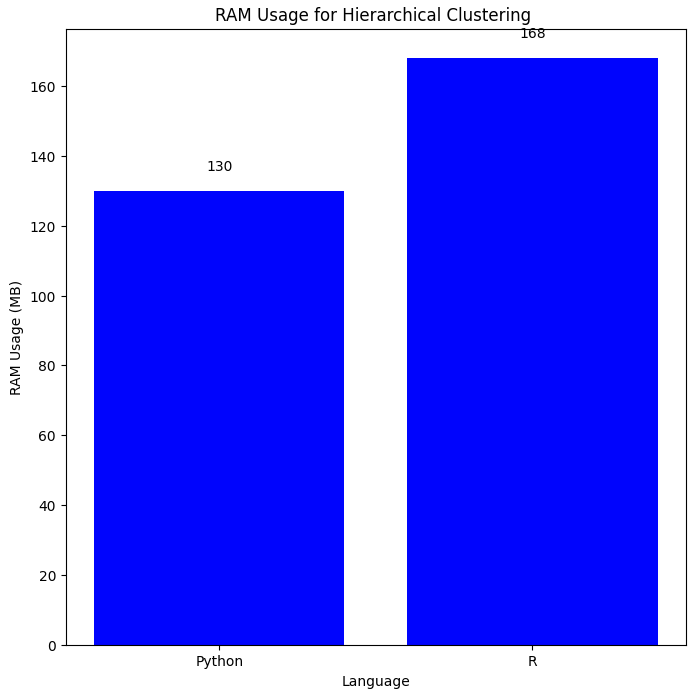


Figure Figure 3 RAM Usage For Hierarchical Cluestering In Python vs R

- **RAM Usage:** Python (130 MB) and R (168 MB)

### Detailed Analysis of Differences:

1. **Optimization of Libraries:** Differences in the efficiency and optimization of libraries implemented in Python and R can contribute to variations in RAM and CPU usage. Python libraries may prioritize general-purpose usage, while R packages may be optimized specifically for statistical computing tasks, leading to differences in resource utilization.

2. **Memory Management:** Python and R employ different memory management strategies, which can impact RAM usage. Python's garbage collection mechanism may incur overhead for memory allocation and deallocation, potentially leading to higher RAM usage compared to R's optimized memory management for statistical computations.

3. **Implementation Details:** Variations in the internal implementation of hierarchical clustering algorithms and data structures within Python and R libraries can influence resource usage. Differences in algorithmic optimizations and data representation may contribute to differences in RAM and CPU usage between the two languages.

4. **External Factors:** External factors such as the version of libraries used, compiler optimizations, and operating system differences can also affect resource usage in Python and R. Differences in how these factors are managed and optimized by each language's ecosystem may contribute to variations in RAM and CPU usage.

**Conclusion:**

In conclusion, the choice between Python and R for hierarchical clustering should consider not only RAM and CPU usage but also factors such as library optimizations, memory management strategies, and implementation details. While both languages exhibit similar characteristics in terms of resource usage for hierarchical clustering with small datasets, understanding the underlying reasons behind differences in RAM and CPU usage is important. Ultimately, the choice between Python and R should align with project requirements, ecosystem familiarity, and performance considerations in specific scenarios.

Both Python and R are capable of efficiently handling hierarchical clustering for small datasets, with R showing slightly higher RAM and CPU usage in this scenario.

### References:

[1] Ozgur, C., Colliau, T., Rogers, G., & Hughes, Z. (2017). MatLab vs. Python vs. R. *Journal of Data Science*, *15*(3), 355-371.‏

[2]“Fernández Lizana, M. I. "Advantages of R as a tool for data Analysis and Visualization in Social Sciences." *Revista Científica de la UCSA* 7.2 (2020): 97-111.

[3] Peng, Roger D. *R programming for data science*. Victoria, BC, Canada: Leanpub, 2016.

[4] Millman, K. Jarrod, and Michael Aivazis. "Python for scientists and engineers." *Computing in science & engineering* 13.2 (2011): 9-12.

[5] Sanglier, Pierre-Adrien, Przemyslaw Lewandowski, and Francis Klumb. *First experience on python development for survey software, advantages and drawbacks*. No. CERN-BE-2023-002. 2022.

[6] Saabith, Sayeth, T. Vinothraj, and M. Fareez. "A review on Python libraries and Ides for Data Science." *Int. J. Res. Eng. Sci* 9.11 (2021): 36-53.

[7]Khoirom, Selina, et al. "Comparative analysis of Python and Java for beginners." *Int. Res. J. Eng. Technol* 7.8 (2020): 4384-4407.

[8]“Mall\_Customers,” *Kaggle*, Dec. 23, 2017. [Online]. Available: <https://www.kaggle.com/datasets/shwetabh123/mall-customers>

[13] DataCamp : https://www.datacamp.com/blog/python-vs-r-for-data-science-whats-the-difference

### Related Work :

[9]Ozgur, Ceyhun, et al. "MatLab vs. Python vs. R." *Journal of data Science* 15.3 (2017): 355-371.

[10]Larose, Chantal D., and Daniel T. Larose. *Data science using Python and R*. John Wiley & Sons, 2019.

[11]Miller, Thomas W. *Modeling techniques in predictive analytics with Python and R: A guide to data science*. FT Press, 2014.

[12]Sudhaka, Kalyan. "Python vs. R Programming Language." *International Journal of Management, IT and Engineering* 8.8 (2018): 70-79.