REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
الجمهورية الجزائرية الديمقراطية الشعبية

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR
ET DE LA RECHERCHE SCIENTIFIQUE

ECOLE SUPERIEURE EN SCIENCES APPLIQUEES
-TLEMCEN-

وزارة التعليم العالي والبحث العلمي

المدرسة العليا في العلوم التطبيقية
-تلمسان-

HIGHER SCHOOL OF APPLIED SCIENCES OF TLEMCEN

SECOND-CYCLE DEPARTMENT

PRACTICAL WORK HANDOUT

Data Analysis

Prepared by:
Dr Imane NEDJAR

Preface

We are pleased to introduce this practical work handout, which brings together the practical exercises from the Data Analysis module, specifically designed for third-year Industrial Engineering students.

The main objective of these practical exercises is to empower students to implement and apply fundamental data analysis methods using the Python programming language, along with the SPSS (Statistical Package for the Social Sciences) analysis software.

We have crafted this practical work handout to provide an introduction to Python programming, aimed at strengthening students' proficiency in this language.
Throughout these practical exercises, students will have the opportunity to put statistical methods into action using Python, delve into data cleaning techniques, and explore the relationships between various variables. Additionally, we will utilize the SPSS software to conduct in-depth analyses of multidimensional relationships between variables.

We hope that this practical work handout will serve as a valuable resource for all participating students, equipping them with both practical programming skills and a profound understanding of data analysis methods.

Contents

List of Figures

List of Tables

INTRODUCTION

In the contemporary era, an unprecedented revolution is underway, characterized by the vast and diverse volume of data generated across various sectors. This data abundance presents an invaluable opportunity to extract meaningful insights, illuminate decision-making, uncover hidden trends, and provide valuable perspectives. However, the mere accumulation of data is insufficient; this is where the vital field of data analysis comes into play.

Data analysis comprises a range of methods and techniques designed to extract actionable knowledge from raw data. Whether in the realms of industry, commerce, scientific research, or other domains, data analysis plays a pivotal role in the transformation of data into actionable information. This equips stakeholders with the capacity to make informed decisions and tackle complex challenges.

This handbook, tailored for third-year industrial engineering students, aims to provide a comprehensive introduction to data analysis, with a focus on essential methods and pertinent tools. Through a blend of practical exercises and conceptual explanations, students will have the opportunity to explore the multifaceted world of data analysis.

The structure of this handbook is as follows:

Workshop 1: Introduction to Python Programming
This initial practical workshop introduces the fundamentals of Python programming.

Workshop 2: Exploring the NumPy Module
The second workshop delves into NumPy, a foundational Python library dedicated to scientific computing. Students will explore the diverse methods offered by NumPy and learn how to employ them for efficient numerical operations.

Workshop 3: Data Preprocessing with the Pandas Module
The third workshop focuses on data preprocessing using the Pandas module. Students will gain insights into how to manipulate, clean, and organize data in readiness for analysis.

Workshop 4: Bidimensional Analysis - Measuring the Relationships between Two Variables
In this fourth workshop, students will explore methods for analyzing the relationships between two variables. They will acquire the skills needed to assess and interpret relationships between different variables within a dataset.

Workshop 5: Multidimensional Analysis - PCA and MCA
The fifth and final workshop introduces Principal Component Analysis (PCA) and Multiple Correspondence Analysis Correspondence Analysis (MCA), powerful techniques for multidimensional analysis. Students will discover how these approaches can be employed to uncover the underlying structure of multidimensional data.

WORKSHOP 1 INTRODUCTION TO PYTHON PROGRAMMING

Python holds a pivotal role as a programming language in the realms of machine learning and data science. Moreover, its impact spans across various industries due to its user-friendly nature and compatibility.

As an open-source and object-oriented programming language, Python offers extensive capabilities. Through its specialized libraries, it finds applications in diverse areas, ranging from software development to data analysis.

The objective of this workshop is to introduce the fundamental principles of Python programming, and it comprises two main parts.

The first part primarily focuses on initiating Python programming and is divided into three sections. Section 1.1 introduces standard data types. Section 1.2 delves deeply into the structure of loops and conditions in the Python language. Finally, Section 1.3 provides an in-depth exploration of the concept of functions.

The second part of the workshop presents exercises designed for engineering students.

Part 1

1.1 Standard Data Types

Python offers a variety of variable types for storing different kinds of data. The five standard data types include:
Boolean, Numbers, Strings, Lists, Tuples, Dictionaries.

1.1.1 Boolean

The boolean type in Python represents a binary state, indicating either "true" or "false." It is used to express logical values and is an essential component in conditional statements, comparisons, and Boolean algebra operations.

1.1.2 Numbers

Python supports four different numeric types (Table 1.1):
int (signed integers)
long (long integers, they can also be represented in octal and hexadecimal)
float (floating-point real values)
complex (complex numbers)

Table 1.1: Numeric Types

int	long	float	complex
10	51924361L	0.0	3.14j
100	-0x19323L	15.2	45.j

Numeric data types store numeric values. Numeric objects are created when you assign them a value. For example, *x=10*, *y=120.5.*

In Python, you can display values using the "print()" function.
To display a variable, simply write its name. You can also print the results

of operations.
x=10
y=20
print("The value of x is",x)
print("x+y=",x+y)

Reading input is done using the "input()" function, which allows the user to enter data, and then storing that data in variables for later use in the program.
x=int(input('Please enter any positive number'))
y=float(input('Please enter a real number'))

1.1.3 Strings

In Python, strings are defined as contiguous sequences of characters enclosed within single or double quotation marks. Subsections of strings can be extracted using slicing operators ([ ] and [:]).

The plus sign (+) is the string concatenation operator, while the asterisk (*) is the repetition operator.
Example
str = 'Hello!'
print(str) # Displays the complete string
print(str[0]) # Displays the first character of the string
print(str[2:-1]) # Displays characters from the 3rd character to the second-to-last
print(str[2:]) # Displays the string starting from the 3rd character
*print(str * 2)* # Displays the string twice
firstName = input('Enter your first name: ') # Reading a string
print(str + "" + firstName) # String concatenation

Various methods are available for manipulating strings, and these are explained in detail in Appendix 1.

1.1.4 Lists

Lists are the most versatile composite data structures in Python. A list contains elements separated by commas and enclosed within square brackets ([ ]). In some ways, lists share similarities with arrays in the C language. One key distinction is that all elements in a list can be of different data types.

- List elements are indexed: the first element is at index [0], the second element at [1], and so on.
- Lists are ordered: when you add new elements to a list, they are placed at the end.
- Because lists are indexed, they can contain elements with the same value.
- List elements can be of any data type and can include different data types.

Example
myList = ["Monday", 2, "January"] # Initializing a list
print(myList[0]) # Display the first element of the list
print(myList[-3]) # Display the first element of the list
myList.append(2010) # Adding an element to the list
del myList[0] # Deleting an element from the list
print("Tuesday" in myList) # Checking the existence of an element in a list

Various methods are available for manipulating lists, and these are explained in detail in Appendix 1.

1.1.5 Tuple

A tuple is another type of sequential data structure in Python, similar to a list. The main distinctions between lists and tuples are as follows:
Lists are enclosed in square brackets ([ ]), and both their elements and size can be modified, whereas tuples are enclosed in parentheses (( )) and are immutable. Tuples can be thought of as read-only lists.

Example

```
tuple1 = (’abcd’, 786, 2.23, ’Yasmina’, 70.2)
tuple2 = (123, ’Yasmina’)
print(tuple1) # Displays the entire tuple
print(tuple1[0]) # Displays the first element of the tuple
print(tuple1[1:3]) # Displays elements of the tuple from the second to the third
print(tuple1[2:]) # Displays elements of the tuple starting from the third element
print(tuple2 * 2) # Displays the tuple contents twice
print(tuple1 + tuple2) # Displays the concatenated tuples
```

1.1.6 Dictionary

Dictionaries in Python are a kind of hash table data structure. They function as associative arrays and consist of key-value pairs. In Python, dictionary keys can be almost any data type, but they are usually numbers or strings. On the other hand, values can be any arbitrary Python object.

Dictionaries are enclosed in curly braces ( ), and values can be assigned and accessed using square brackets ([]).

Example

```
dic = {’computer’:’ordinateur’,’mouse’: ’souris’,’keyboard’:’clavier’}
print(dic["computer"])
print(dic.keys())
print(dic.values())
del dic[’mouse’]
dic["desktop"] = "bureau"
print(dic)
```

Various methods are available for manipulating dictionaries, and these are explained in detail in Appendix 1.

1.2 Loops and Conditions

In Python, unlike some other programming languages, there are no opening or closing curly braces to delineate a block of instructions. Blocks of code in Python are defined by the ':' symbol followed by indentation. All statements indented at the same level following a ':' belong to the same block of code.

1.2.1 for Loop

Example

```
for i in range(10):
        x = 2
        print(x * i)
for i in [0, 1, 2, 3]:
        print("i has the value", i)
```

1.2.2 while Loop

Example

```
a = 0
while (a < 12):
    a = a + 1
    print(a, a * 2)
```

1.2.3 if/then/else Condition

Example

```
a = 0
if a == 0:
    print('0')
   elif a == 1:
        print('1')
    else:
        print('2')
```

1.3 Functions

A function is a block of code that only runs when it is called. You can pass data, called parameters, into a function, and a function can return data as well.

1.3.1 Creating a Function

In Python, a function is defined using the **def** keyword:

```
def My_function():
        print("Hello from My function")
```

1.3.2 Calling a Function

To call a function, use the function name followed by parentheses:

```
def My_function():
        print("Hello from My function")
My_function()
```

1.3.3 Arguments

Information can be passed to functions as arguments. Arguments are specified after the function name within parentheses. You can add as many arguments as you like, separating them with commas. The following example demonstrates a function with one argument (first_name). When the function is called, we pass a first name, which is used inside the function to display the full name:

```
def My_function(first_name):
        print(first_name + " HADJ")
My_function("Hassiba")
My_function("Kamel")
My_function("Ikram")
```

1.3.4 Arbitrary Arguments, *args

If you're unsure how many arguments will be passed to your function, you can add an * before the parameter name in the function definition. This way, the function will receive a tuple of arguments and can access them accordingly.

```
def My_function(*children):
    print("The eldest is " + children[0])
My_function("Hanane", "Yasmina", "Mohamed")
```

1.3.5 Keyword Arguments

You can also send arguments using the key = value syntax. This way, the order of the arguments doesn't matter.

```
def My_function(child1, child2, child3):
    print("The eldest is " + child1)
My_function(child2 = "Hanane", child3 = "Yasmina", child1 = "Mohamed")
```

1.3.6 Default Parameter Value

The following example demonstrates how to use a default value for a parameter. If we call the function without providing an argument, it will use the default value.

```
def My_function(country = "Algeria"):
    print("I am from " + country)
My_function("Sweden")
My_function()
My_function("Brazil")
```

1.3.7 Return Values

To enable a function to return a value, use the return statement.

```
def My_function(x):
    return 5 * x
print(My_function(3))
print(My_function(5))
```

1.3.8 Global Variables vs. Local Variables

Variables defined within the body of a function have local scope, whereas those defined outside have global scope.

In essence, this means that local variables can only be accessed within the function in which they are declared, while global variables are accessible throughout the entire program, across all functions. When you call a function, any variables declared inside it have local scope within that function.

```
total = 0; # Here, total is a global variable
def My_function(arg1, arg2):
    total = arg1 + arg2; # Here, total is a local variable
    print("Inside the function, the local variable total=" , total)
    return total;
My_function(30, 20);
print("Outside the function, the global variable total= ", total)
```

The result of execution is as follows:

```
Inside the function, the local variable total= 50
Outside the function, the global variable total= 0
```

Part 2

Work requested

Exercise 1

Given the data production output of a manufacturing plant over several months:

- January: 500
- February: 600
- March: 700
- April: 550
- May: 800
- June: 750

-Calculate the average production.

Exercise 2

Given a list of product names: "Gold Watch", "Old Phone", "Old Laptop", "New Tablet".
Write a Python function to perform the following tasks:

-Find the longest product name in the list. Count the total number of characters in all the product names.

-Replace any occurrence of the word "old" with "new" in each product name.

-Concatenate all the product names into a single string separated by commas.

Exercise 3

To assess the efficiency of various machines on the production line, we've gathered data on their operating times (in hours) from the previous month:

- Machine A: 120 hours
- Machine B: 95 hours
- Machine C: 150 hours
- Machine D: 80 hours
- Machine E: 110 hours

These operating times fall into efficiency categories defined as follows:

- Low Efficiency: 0-99 hours
- Moderate Efficiency: 100-129 hours
- High Efficiency: 130-199 hours

-Write a Python function to categorize these machines based on their efficiency levels.

Solutions

Solution for Exercise 1

```
production_data = {
        'January': 500,
        'February': 600,
        'March': 700,
        'April': 550,
        'May': 800,
        'June': 750 }
total_production = 0
num_months = 0
for month, production in production_data.items():
        total_production += production
        num_months += 1
average_production = total_production / num_months
print("Average production :", average_production)
```

Solution for Exercise 2

```
def string_operations(product_names):
       longest_name = max(product_names, key=len)
       print("Longest product name:", longest_name)
       total_chars = sum(len(name) for name in product_names)
       print("Total number of characters:", total_chars)
       modified_names = [name.replace("old", "new") for name in product_names]
       print("Modified product names:", modified_names)
       concatenated_names = ",".join(product_names)
       print("Concatenated product names:", concatenated_names)
product_list = ["New Watch", "Old Phone", "Old Laptop", "New Tablet"]
string_operations(product_list)
```

Solution for Exercise 3

```
machine_data = {
    "Machine_A": 120,
    "Machine_B": 95,
    "Machine_C": 150,
    "Machine_D": 80,
    "Machine_E": 110 }
efficiency_categories = {
    "Low Efficiency": range(0, 100),
    "Moderate Efficiency": range(100, 130),
    "High Efficiency": range(130, 200) }

def categorize_efficiency(machine, operating_time):
    for category, hours_range in efficiency_categories.items():
        if operating_time in hours_range:
            return f"{machine} - {category}"
for machine, operating_time in machine_data.items():
    print(categorize_efficiency(machine, operating_time))
```

Conclusion

In this Workshop, we introduced fundamental concepts of Python programming. This included standard data types such as boolean, numbers, strings, lists, tuples, and dictionaries. We also covered control structures like loops (for and while) and conditional statements (if/else). Additionally, we discussed functions, addressing function creation, arguments, return values, and variable scope (global and local). The chapter concludes with exercises and their solutions.

WORKSHOP 2
EXPLORING THE NUMPY MODULE

In the domain of data analysis, especially when dealing with numerical datasets, comprehending data distribution and relationships between variables through unidimensional descriptive statistics like mean, variance, and standard deviation, or bidimensional measures such as covariance and correlation, is indispensable for informed decision-making. NumPy, with its efficient array handling capabilities, plays a pivotal role in this process. Whether we're dealing with simple 1-dimensional arrays like lists of values or complex multi-dimensional arrays representing time series or tabular data, NumPy equips us with the essential tools to conduct diverse data analysis tasks effectively.

The objective of this practical work is to introduce the NumPy module along with its various methods. The first part of the practical work is dedicated to definitions with some examples. The second part delves into exercises that utilize NumPy to explore the relationships between two variables.

Part 1

2.1 What is NumPy?

NumPy, short for "Numerical Python," was created in 2005 by Travis Oliphant as a Python library. Its primary aim is to streamline the manipulation of data arrays. While Python provides lists for array representation, their processing can be slow. This is where NumPy steps in, offering an array object that can be up to 50 times faster than traditional Python lists. Known as "ndarray" (multidimensional array) in NumPy, this object comes with a plethora of supporting functions that significantly simplify data manipulation. Arrays find extensive use in data analysis, where speed and efficient resource utilization are paramount. Unlike lists, NumPy arrays are stored continuously in memory, enabling processes to access and manipulate them with high efficiency.

2.2 Creating an Array

To create an ndarray, we can provide a list, a tuple, or any similar array-like object as an argument to the array() method, and it will be transformed into an ndarray :
Example

```
import numpy as np
my_list = [1, 2, 3, 4, 5]
my_tuple = (1, 2, 3, 4, 5)
arr1 = np.array(my_list)
arr2 = np.array(my_tuple)
print(arr1)
print(arr2)
print(type(arr1))
```

2.3 Dimensions in Arrays

A dimension in arrays corresponds to a level of array depth. 0-D, 1-D array is called a one-dimensional array.

2.3.1 0-D Arrays

0-D arrays, or scalars, are individual elements of an array.

```
import numpy as np
arr = np.array(42)
print(arr)
```

2.3.2 1-D Arrays

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
print(arr)
```

2.3.3 2-D Arrays

These are often used to represent matrices or 2nd-order tensors.

```
import numpy as np
arr = np.array([[1, 2, 3], [4, 5, 6]])
print(arr)
```

2.3.4 3-D Arrays

```
import numpy as np
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])
print(arr)
```

How to Check the Number of Dimensions?

NumPy Arrays provide the ‘ndim‘ attribute, which returns an integer indicating the number of dimensions of the array.

```
import numpy as np
a = np.array(42)
b = np.array([1, 2, 3, 4, 5])
c = np.array([[1, 2, 3], [4, 5, 6]])
d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])
print(a.ndim)
print(b.ndim)
print(c.ndim)
print(d.ndim)
```

2.4 Accessing Array Elements

2.4.1 Accessing 1D Arrays

```
import numpy as np
arr = np.array([1, 2, 3, 4])
print(arr[0])
```

=> result: 1

2.4.2 Accessing 2D Arrays

```
import numpy as np
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print("Second element of the first row: ", arr[0, 1])
```

=> result: 2

2.4.3 Accessing 3D Arrays

```
import numpy as np
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
print(arr[0, 1, 2])
```
=> result: 6

2.4.4 Negative Indexing

Use negative indexing to access an array from the end.

```
import numpy as np
arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
print("Last element of the second dimension: ", arr[1, -1])
```

2.4.5 Shape of an Array

```
import numpy as np
arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
print(arr.shape)
```

2.4.6 Reshaping Arrays

Reshaping means changing the shape of an array. The shape of an array represents the number of elements in each dimension. By performing reshaping, we have the ability to add or remove dimensions and modify the number of elements in each of these dimensions.

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
```

```
newarr = arr.reshape(4, 3)
print(newarr)
```

2.5 Slicing Arrays

2.5.1 Slicing a 1D Array

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[1:5])
print(arr[4:])
print(arr[:4])
print(arr[-3:-1])
print(arr[0:5:2])
print(arr[::2])
```

2.5.2 Slicing a 2D Array

Starting from the second element, slice elements from index 1 to index 4 (exclusive):

```
import numpy as np
arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
print(arr[1, 1:4])
```

Starting from the second element, slice from index 1 to index 4 (exclusive). This will return a 2D array:

```
import numpy as np
arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
print(arr[0:2, 1:4])
```

2.6 Data Types Supported by NumPy

Here are some data types supported by NumPy:

i - integer
b - boolean
f - float
S - string
c - complex number
m - timedelta
M - datetime
O - object

Example

```
import numpy as np
arr1 = np.array([15, 10, 3, 40], dtype='i')
arr2 = np.array([1.1, 2.1, 3.1], dtype='f')
print(arr1)
print(arr1.dtype)
print(arr2)
print(arr2.dtype)
```

2.7 Iterating Through Arrays

2.7.1 1D Arrays

```
import numpy as np
arr = np.array([1, 2, 3])
for x in arr:
        print(x)
```

2.7.2 2D Arrays

```
import numpy as np
arr = np.array([[1, 2, 3], [4, 5, 6]])
print("Displaying rows")
for x in arr:
    print(x)
print("Displaying elements")
for x in arr:
    for y in x:
        print(y)
```

2.7.3 3D Arrays

```
import numpy as np
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
print("Displaying rows")
for x in arr:
    print(x)
print("Displaying elements")
for x in arr:
    for y in x:
        for z in y:
            print(z)
```

2.8 Searching in Arrays

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 4, 4])
x = np.where(arr == 4)
```

```
print(x)
```

2.9 Sorting Arrays

```
import numpy as np
arr = np.array([3, 2, 0, 1])
print(np.sort(arr))
```

2.10 Concatenating Arrays

```
import numpy as np
arr1 = np.array([[1, 2], [3, 4]])
arr2 = np.array([[5, 6], [7, 8]])
arr = np.concatenate((arr1, arr2), axis=1)
print(arr)
```

2.11 Splitting Arrays

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6])
newarr = np.array_split(arr, 3)
print(newarr)
```

Part 2

Work requested

In a factory, pieces are being manufactured for a machine. Each piece is linked to its production cost (Y), measured in Dinars, as well as the time needed for its creation (X), measured in minutes. The data regarding this distribution is presented in the following Table 2.1.

piece	1	2	3	4	5
X	2	3	5	2	4
Y	10	16	24	10	20

Table 2.1: Pieces and their associated cost and time of realization

Referring to equation (2.1) for the correlation coefficient and equation (2.1) for the covariance

$$R = \frac{Cov(X,Y)}{\sigma_x \sigma_y} \tag{2.1}$$

$$Cov(X,Y) = \sum_{i=1}^{N} \frac{1}{N}(x_i - \overline{X})(y_i - \overline{Y}) \tag{2.2}$$

- Calculate the mean, variance, and standard deviation of the statistical variables X and Y.
- Calculate the correlation coefficient.

If a linear correlation between X and Y is present:

- Determine the regression line equation.
- Estimate the production cost of the parts as a function of their production time.

Solution

```
from numpy.lib.function_ base import cov
import numpy as np
arr = np.array([[2,3,5,2,4],[10,16,24,10,20]])
print(arr)
meanX=np.mean(arr[0])
meanY=np.mean(arr[1])
varX=np.var(arr[0])
varY=np.var(arr[1])
varX= round(varX,2)
varY= round(varY,2)
stdX=np.std(arr[0])
stdY=np.std(arr[1])
stdX= round(stdX,2)
stdY= round(stdX,2)
print("meanX =",meanX)
print("meanY =",meanY)
print("varX",varX)
print("varY",varY)
print("stdX =",stdX)
print("stdY =",stdX)
#Correlation coefficient= covXY\stdX*stdY
# Covariance matrix (x,y):
#                           cov(xx) cov(xy)
#                           cov(yx) cov(yy)
covariance=np.cov(arr[0,:],arr[1,:])
covXY=covariance[0,1]
coefcorr=covXY/stdX*stdY
coefcorr=round(coefcorr,2)
```

```
if coefcorr>0.6:
        print("There is a strong correlation equal to ",coefcorr)
        # The correlation line : Y=aX+b
        # a=covXY/varX
        # b=mean(Y)-a*mean(X)
        a=covXY/varX
        b=meanY-(a*meanX)
        a=round(a,2)
        b=round(b,2)
        print("The correlation line:","y=",a,"x+",b)
        x=int(input("Enter the production time of the piece:"))
        y=a*x+b
        print("The estimated cost of producing this piece using the correlation
        line is", y, "DA")
else:
        print("There is a weak correlation equal to", coefcorr)
```

Conclusion

NumPy is integral in multidimensional data analysis, offering indispensable tools and capabilities for effectively handling numerical data.

Throughout this workshop, we delved into the fundamental concepts of the NumPy module. This encompassed array creation across various dimensions (0-D, 1-D, 2-D, 3-D) and accessing array elements. Additionally, we explored advanced functionalities such as negative indexing, reshaping arrays, slicing, iteration, and diverse array operations including searching, sorting, concatenation, and splitting. The chapter concluded with a series of exercises, each accompanied by its solution.

WORKSHOP 3 DATA PREPROCESSING WITH THE PANDAS MODULE

Data preprocessing plays a critical role in ensuring the accuracy and reliability of subsequent data analysis. It involves various steps and techniques to address issues like noise, bias, and inconsistencies, thereby transforming raw data into clean, high-quality data suitable for analysis. This practical session aims to utilize the Pandas library, a widely used tool for data preprocessing and analysis. The first part will cover definitions and methods of Pandas for data cleansing, exploration, and manipulation, supported by examples. In the second part, students will work on preprocessing and analyzing students' grade report using the Pandas module.

Part 1

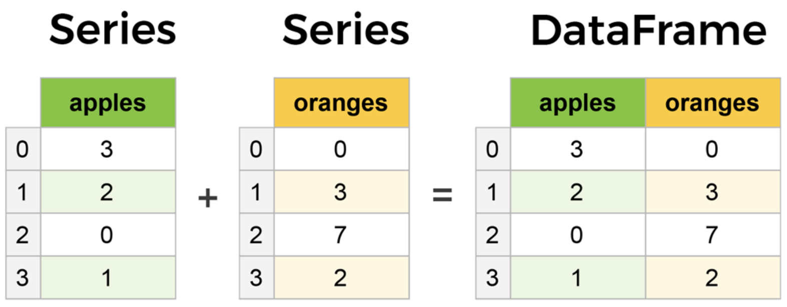
3.1 What is Pandas?

"Pandas" is a blend of "Panel Data" and "Python Data Analysis," coined by its creator Wes McKinney in 2008. It provides a comprehensive set of tools for data analysis, cleansing, exploration, and manipulation. In the realm of data analysis, Pandas facilitates the examination of large datasets and the derivation of insights using statistical methods. It excels in organizing disorderly datasets, enhancing readability, and enabling actionable insights, all of which are crucial aspects of data science.

3.2 Main Components of Pandas

The two main components of Pandas are Series and DataFrames (see Figure 3.1).

A Series is essentially a column, and a DataFrame is a multi-dimensional table composed of a collection of Series.



	apples
0	3
1	2
2	0
3	1

	oranges
0	0
1	3
2	7
3	2

	apples	oranges
0	3	0
1	2	3
2	0	7
3	1	2

Figure 3.1: The Series and DataFrame

3.3 Pandas Series

A Series is a one-dimensional array containing data of various types. By default, values are labeled (indexed) with their index number.

The first value has index 0, the second value has index 1, and so on. These labels can be used to access specific values.

```
import pandas as pd
a = [1, 7, 2]
x = pd.Series(a)
print(x)
print(x[0])
```

3.3.1 Creating Labels

With the index argument, you have the option to set your own labels.

```
import pandas as pd
a = [1, 7, 2]
x = pd.Series(a, index = ["x", "y", "z"])
print(x)
print(x["y"])
```

3.3.2 Key/Value Objects in a Series

You can also use a key/value object, such as a dictionary, when creating a series.

```
import pandas as pd
calories = {"day1": 20, "day2": 360, "day3": 30}
x = pd.Series(calories)
print(x)
```

3.4 DataFrames

Data sets in Pandas are typically multi-dimensional tables called DataFrames.

Example

```
import pandas as pd
data = {
        "calories": [420, 380, 390],
        "duration": [50, 40, 45]
        }
x = pd.DataFrame(data)
print(x)
```

3.4.1 Locating a Row

Pandas uses the loc attribute to return one or more specified rows.
print(x.loc[0])

3.4.2 Named Indexes

```
import pandas as pd
data = {
        "calories": [420, 380, 390],
        "duration": [50, 40, 45]
      }
x = pd.DataFrame(data, index = ["day1", "day2", "day3"])
print(x)
print(x.loc["day2"])
```

3.5 Reading Data with Pandas

Pandas provides the ability to extract and load data from a variety of file formats, including CSV, Excel, JSON, HDF5, SQL, and many others. This functionality allows for creating data structures like DataFrames within Pandas, greatly simplifying the analysis, manipulation, and transformation of this data within Python.

Example:
# Reading data from an Excel file
df = pd.read_ excel('file.xls')
# Reading data from a CSV file
df = pd.read_ csv('file.csv')
# Reading data from a JSON file
df = pd.read_json('file.json')
# Reading data from a SQL database
import sqlite3
con = sqlite3.connect("database.db")

3.6 Displaying Data with Pandas

Using the **print()** method allows for displaying data with Pandas.
import pandas as pd
df = pd.read_ excel('file.xls')
print(df)

With Pandas, the number of returned rows can be determined by adjusting the library's option parameters.

import pandas as pd
print(pd.options.display.max_ rows)
# Increase the maximum number of rows to display the entire DataFrame
pd.options.display.max_ rows = 9999

3.7 Data Preprocessing

Data preprocessing encompasses all the steps and techniques applied to raw data to clean and prepare it for further analysis. Typically, data cleaning involves correcting incorrect or inconsistent information in your dataset. Incorrect data can include empty cells, data in the wrong format, erroneous data, and duplicates.

To apply preprocessing methods, we will work with an Excel file (saved in the 97-2003 format) named 'Dirtydata.xls,' which is available via this link Workshop-Pandas. This file contains the number of pulses, maximum values recorded during a duration in minutes over several days, as well as calories.

Duration	**Date**	**Pulse**	**Maxpulse**	**Calories**
60	'2020/12/01'	110	130	400
60	'2020/12/02'	117	145	470
:	:	:	:	:

3.7.1 Empty Cells

Empty cells can potentially lead to incorrect results when analyzing data.

3.7.1.1 Deletion

One way to handle empty cells is to delete rows containing empty cells.

```
import pandas as pd
df1 = pd.read_excel('Dirtydata.xls')
df2 = df1.dropna()
print(df2)
```

If you want to modify the original DataFrame, use the **inplace=True** argument:

```
import pandas as pd
df = pd.read_ excel('Dirtydata.xls')
df.dropna(inplace=True)
print(df)
```

3.7.1.2 Replacing Empty Cells

Another way to handle empty cells is to insert a new value in their place. This way, you don't have to delete entire rows just because of some empty cells. The **fillna()** method allows us to replace empty cells with a value.

Replace with any value

```
import pandas as pd
df = pd.read_ excel('Dirtydata.xls')
df.fillna(1000, inplace=True)
print(df)
```

Replace with Mean, Median, or Mode

A common way to replace empty cells is by calculating the mean, median, or mode value of the column.

Pandas uses the **mean()**, **median()**, and **mode()** methods to calculate the respective values of a specified column:

Mean

```
import pandas as pd
df = pd.read_ excel('Dirtydata.xls')
x = df["Calories"].mean()
df["Calories"].fillna(x, inplace=True)
print(df)
```

Median

```
import pandas as pd
```

```
df = pd.read_excel('Dirtydata.xls')
x = df["Calories"].median()
df["Calories"].fillna(x, inplace=True)
print(df)
```

Mode

```
import pandas as pd
df = pd.read_excel('Dirtydata.xls')
x = df["Calories"].mode()[0]
df["Calories"].fillna(x, inplace=True)
print(df)
```

3.7.2 Data in the Wrong Format

Cells containing data in the wrong format can make data analysis difficult or even impossible.
To address this issue, you have two options: convert all cells in the columns to the correct format or delete the affected rows.
In our DataFrame, there are two cells with incorrect formats. Please check rows 22 and 26; the 'Date' column should be a string representing a date.

Perform Conversion to Correct Format

```
import pandas as pd
df = pd.read_excel('Dirtydata.xls')
df['Date'] = pd.to_datetime(df['Date']).dt.strftime('%d-%m-%Y ')
print(df)
```

As you can see in the result, the date in row 26 was set as a random date, while the missing date in row 22 was represented as NaN. One approach to handling these missing values is simply to delete the entire row.

Deletion of Rows

We can delete the row using the **dropna()** method.

```
import pandas as pd
df = pd.read_excel('Dirtydata.xls')
df['Date'] = pd.to_datetime(df['Date']).dt.strftime('%d-%m-%Y ')
df.dropna(subset=['Date'], inplace=True)
```

3.7.3 Erroneous Data

"Erroneous data" doesn’t necessarily have to be "empty cells" or "wrong format"; it can simply be incorrect, like someone recording "199" instead of "1.99".
One way to correct erroneous values is to replace them with something else.

Example 1:

```
df.loc[7, 'Duration'] = 45
```

Example 2:

```
for x in df.index:
if df.loc[x, "Duration"] > 60:
    df.loc[x, "Duration"] = 60
```

3.7.4 Discovering Duplicates

To discover duplicates, we can use the **duplicated()** method. This method returns a boolean value for each row: *print(df.duplicated())*

3.7.5 Removing Duplicates

To remove duplicates, use the **drop_duplicates()** method:

```
df.drop_duplicates(inplace=True)
```

3.8 Plotting

Pandas uses the **plot()** method to create charts. We can use **Pyplot**, a submodule of the Matplotlib library, to visualize the chart.

```
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_ excel('Dirtydata.xls')
df.plot()
plt.show()
```

3.8.1 Scatter Plot

```
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_ excel('Dirtydata.xls')
df.plot(kind='scatter', x='Duration', y='Calories')
plt.show()
```

3.8.2 Histogram

The histogram shows the frequency of each interval and requires only a single column.

```
df["Duration"].plot(kind='hist')
```

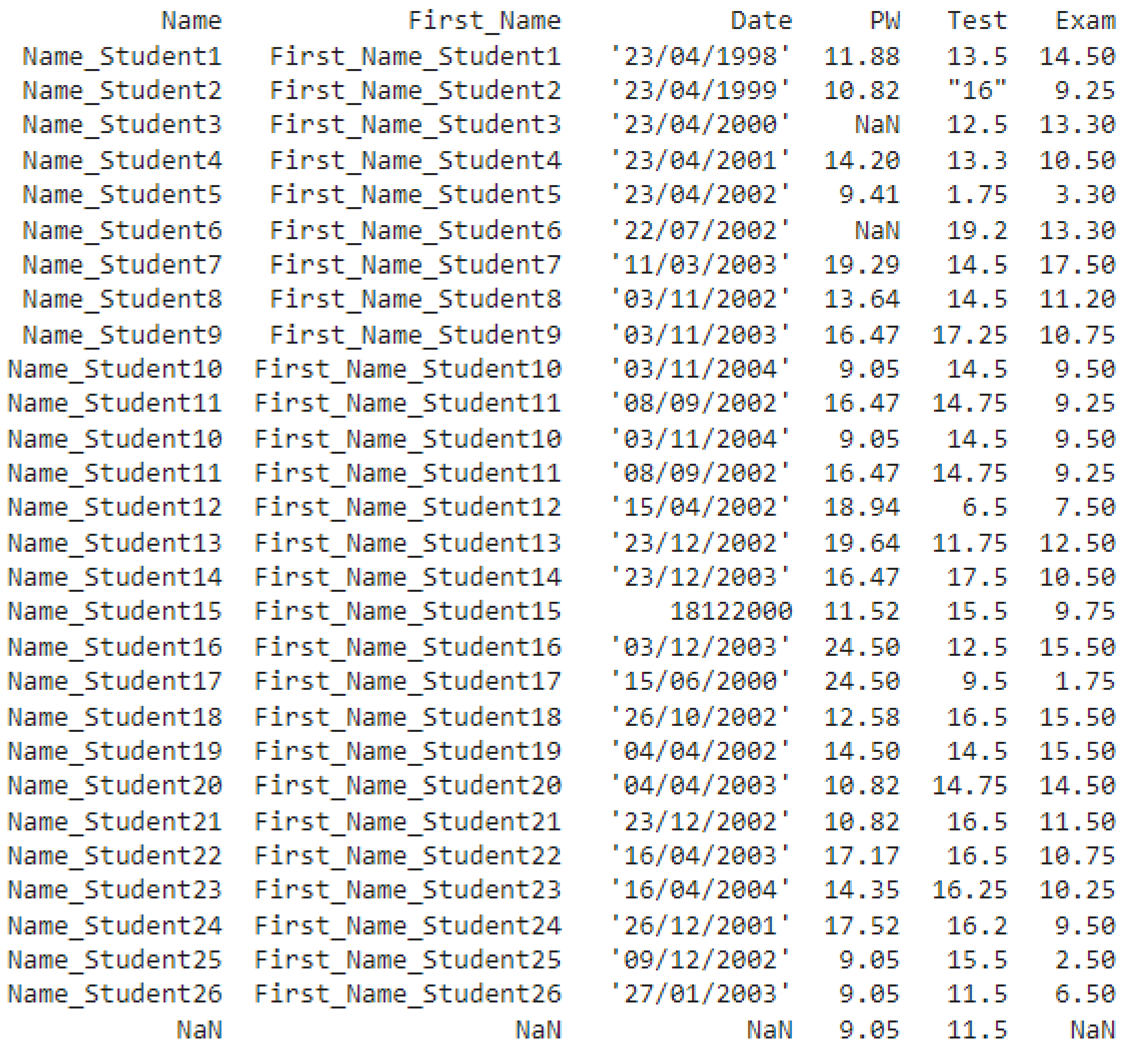
Part 2

Work requested

You have the student grade report file, including Practical Work (PW) evaluations, Tests, and Exams scores, as illustrated in Figure 3.2. Using the Pandas module,

- Perform a complete data preprocessing of the dataset.
- Calculate the correlation between Test scores and Exam scores. Interpret the result.
- Display the scatter plot of exam scores and test scores.
- Display the frequency distribution for Exam scores.

Note: You can find the 'Score.xls' file by following this link:
Workshop3-Pandas



```
            Name             First_Name          Date     PW   Test   Exam
   Name_Student1   First_Name_Student1  '23/04/1998'  11.88   13.5  14.50
   Name_Student2   First_Name_Student2  '23/04/1999'  10.82   "16"   9.25
   Name_Student3   First_Name_Student3  '23/04/2000'    NaN   12.5  13.30
   Name_Student4   First_Name_Student4  '23/04/2001'  14.20   13.3  10.50
   Name_Student5   First_Name_Student5  '23/04/2002'   9.41   1.75   3.30
   Name_Student6   First_Name_Student6  '22/07/2002'    NaN   19.2  13.30
   Name_Student7   First_Name_Student7  '11/03/2003'  19.29   14.5  17.50
   Name_Student8   First_Name_Student8  '03/11/2002'  13.64   14.5  11.20
   Name_Student9   First_Name_Student9  '03/11/2003'  16.47  17.25  10.75
  Name_Student10  First_Name_Student10  '03/11/2004'   9.05   14.5   9.50
  Name_Student11  First_Name_Student11  '08/09/2002'  16.47  14.75   9.25
  Name_Student10  First_Name_Student10  '03/11/2004'   9.05   14.5   9.50
  Name_Student11  First_Name_Student11  '08/09/2002'  16.47  14.75   9.25
  Name_Student12  First_Name_Student12  '15/04/2002'  18.94    6.5   7.50
  Name_Student13  First_Name_Student13  '23/12/2002'  19.64  11.75  12.50
  Name_Student14  First_Name_Student14  '23/12/2003'  16.47   17.5  10.50
  Name_Student15  First_Name_Student15      18122000  11.52   15.5   9.75
  Name_Student16  First_Name_Student16  '03/12/2003'  24.50   12.5  15.50
  Name_Student17  First_Name_Student17  '15/06/2000'  24.50    9.5   1.75
  Name_Student18  First_Name_Student18  '26/10/2002'  12.58   16.5  15.50
  Name_Student19  First_Name_Student19  '04/04/2002'  14.50   14.5  15.50
  Name_Student20  First_Name_Student20  '04/04/2003'  10.82  14.75  14.50
  Name_Student21  First_Name_Student21  '23/12/2002'  10.82   16.5  11.50
  Name_Student22  First_Name_Student22  '16/04/2003'  17.17   16.5  10.75
  Name_Student23  First_Name_Student23  '16/04/2004'  14.35  16.25  10.25
  Name_Student24  First_Name_Student24  '26/12/2001'  17.52   16.2   9.50
  Name_Student25  First_Name_Student25  '09/12/2002'   9.05   15.5   2.50
  Name_Student26  First_Name_Student26  '27/01/2003'   9.05   11.5   6.50
             NaN                   NaN           NaN   9.05   11.5    NaN
```

Figure 3.2: Student grade report

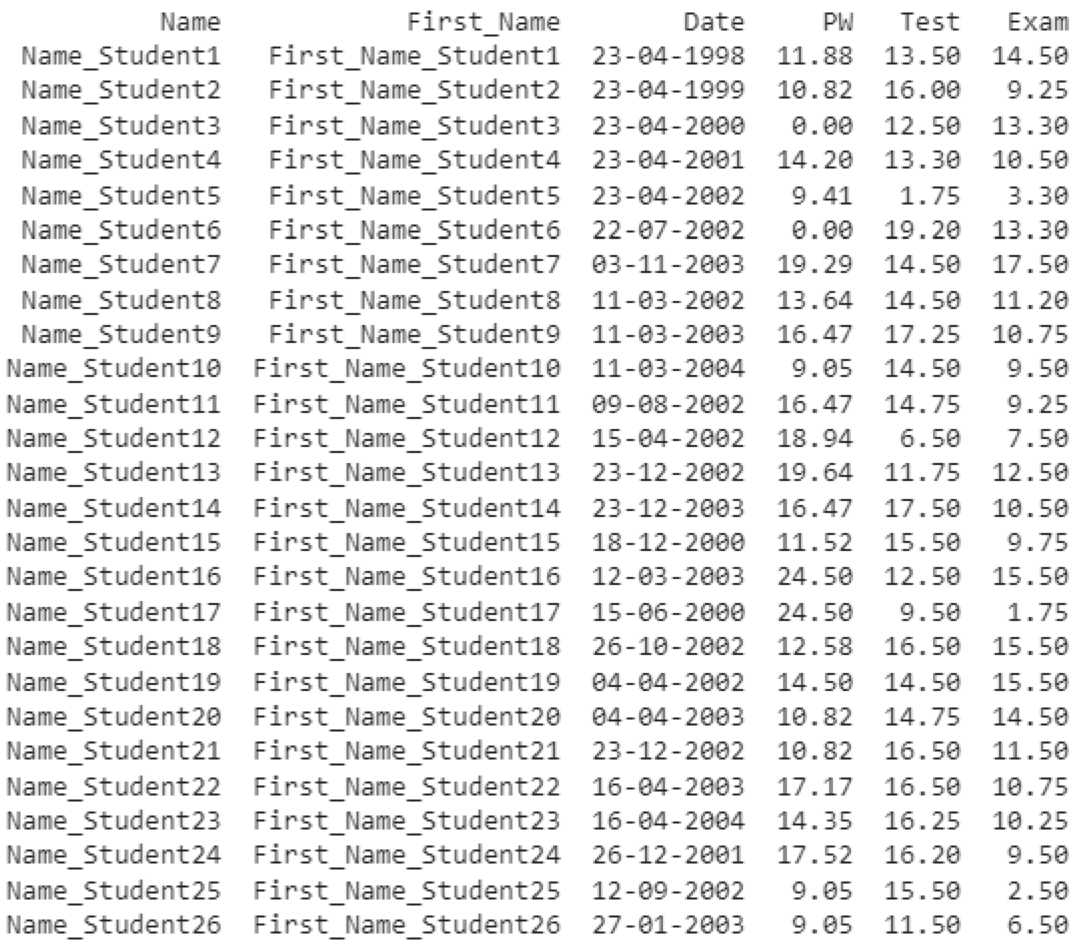
Solution

```
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_ excel('Score.xls')
df.Test[1] = 16
df.fillna(0, inplace = True)
df.drop_ duplicates(inplace = True)
df['Date'] = pd.to_ datetime(df['Date']).dt.strftime('%d-%m-%Y')
df.Date[16] = '18-12-2000'
df = df.drop(labels=28, axis=0)
print(df)
col1, col2 = "Test", "Exam"
corr = df[col1].corr(df[col2])
print("Correlation between ", col1, " and ", col2, " is: ", round(corr, 2))
df.plot(kind = 'scatter', x = 'Test', y = 'Exam')
plt.show()
df["Exam"].plot(kind = 'hist')
```

- The processed final data is presented in Figure 3.3.
- The correlation coefficient between Test and Exam is 0.47.
-the scatter plot of exam scores and test scores is shown in Figure 3.4.
-The frequency distribution for Exam scores is shown in Figure 3.5.

Analysis of results:

A low correlation between test and exam scores suggests potential differences between the two assessment methods or variations in students' efforts across them. Referring to the scatter plot of exam scores and test scores.(Figure 3.4), we note a score range between 11 and 17 for that class. Moreover, the frequency distribution for exam scores (Figure 3.5) indicates a relatively uniform level among students.



```
           Name            First_Name        Date     PW   Test   Exam
  Name_Student1   First_Name_Student1  23-04-1998  11.88  13.50  14.50
  Name_Student2   First_Name_Student2  23-04-1999  10.82  16.00   9.25
  Name_Student3   First_Name_Student3  23-04-2000   0.00  12.50  13.30
  Name_Student4   First_Name_Student4  23-04-2001  14.20  13.30  10.50
  Name_Student5   First_Name_Student5  23-04-2002   9.41   1.75   3.30
  Name_Student6   First_Name_Student6  22-07-2002   0.00  19.20  13.30
  Name_Student7   First_Name_Student7  03-11-2003  19.29  14.50  17.50
  Name_Student8   First_Name_Student8  11-03-2002  13.64  14.50  11.20
  Name_Student9   First_Name_Student9  11-03-2003  16.47  17.25  10.75
 Name_Student10  First_Name_Student10  11-03-2004   9.05  14.50   9.50
 Name_Student11  First_Name_Student11  09-08-2002  16.47  14.75   9.25
 Name_Student12  First_Name_Student12  15-04-2002  18.94   6.50   7.50
 Name_Student13  First_Name_Student13  23-12-2002  19.64  11.75  12.50
 Name_Student14  First_Name_Student14  23-12-2003  16.47  17.50  10.50
 Name_Student15  First_Name_Student15  18-12-2000  11.52  15.50   9.75
 Name_Student16  First_Name_Student16  12-03-2003  24.50  12.50  15.50
 Name_Student17  First_Name_Student17  15-06-2000  24.50   9.50   1.75
 Name_Student18  First_Name_Student18  26-10-2002  12.58  16.50  15.50
 Name_Student19  First_Name_Student19  04-04-2002  14.50  14.50  15.50
 Name_Student20  First_Name_Student20  04-04-2003  10.82  14.75  14.50
 Name_Student21  First_Name_Student21  23-12-2002  10.82  16.50  11.50
 Name_Student22  First_Name_Student22  16-04-2003  17.17  16.50  10.75
 Name_Student23  First_Name_Student23  16-04-2004  14.35  16.25  10.25
 Name_Student24  First_Name_Student24  26-12-2001  17.52  16.20   9.50
 Name_Student25  First_Name_Student25  12-09-2002   9.05  15.50   2.50
 Name_Student26  First_Name_Student26  27-01-2003   9.05  11.50   6.50
```

Figure 3.3: Student grade report after data preprocessing

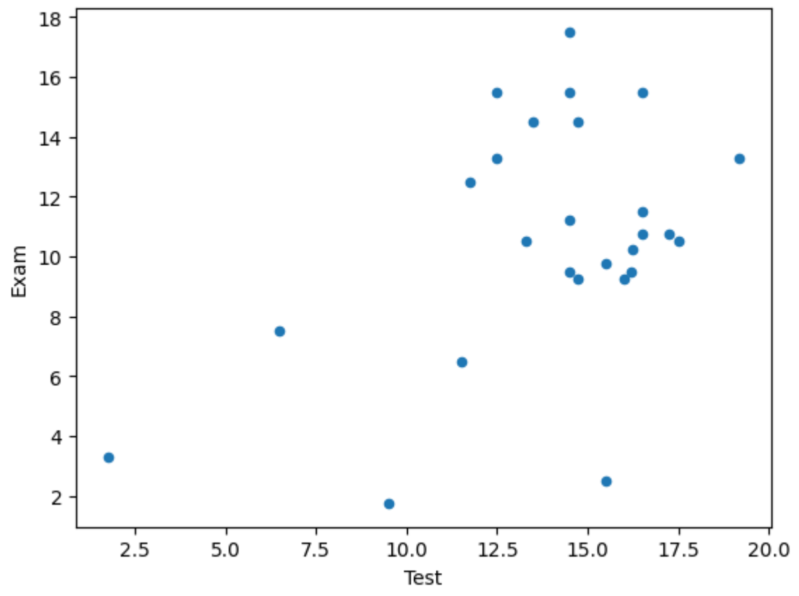


Figure 3.4: Cross-Tabulation of Exam Scores and Test Scores

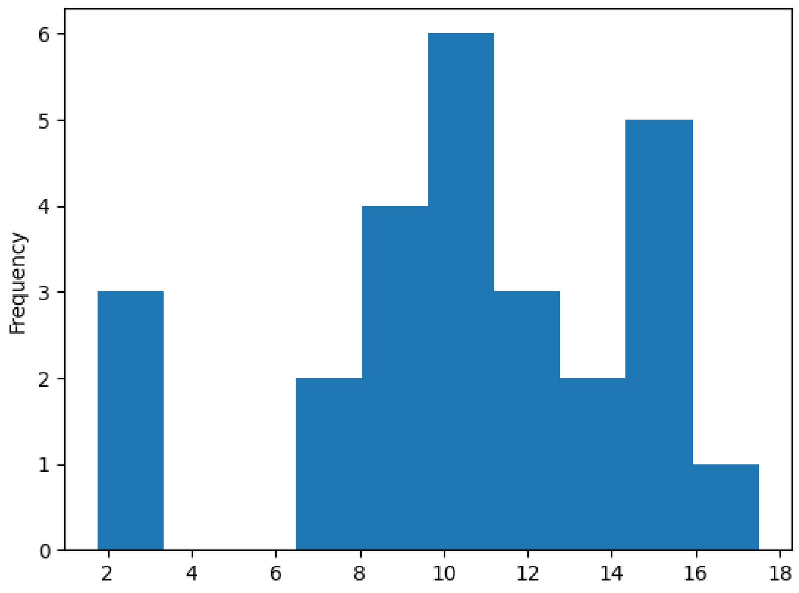


Figure 3.5: The Frequencies for Exam Scores

Conclusion

The Pandas module is extensively utilized for data preprocessing tasks, encompassing data cleansing, exploration, and manipulation. In this workshop, we explored the core components of Pandas, including Pandas Series and Data Frame.

Subsequently, we delved into data handling with Pandas, covering data reading, display, and preprocessing methods. Specifically, within data preprocessing, we addressed strategies for managing empty cells by either deletion or replacement. Additionally, we discussed techniques for handling erroneous data, such as identifying and removing duplicates.

Moreover, we introduced data visualization techniques using scatter plots and histograms. This workshop concludes with a practical exercise along with its solution.

WORKSHOP 4 BIDIMENSIONAL ANALYSIS: MEASURING THE RELATIONSHIP BETWEEN TWO VARIABLES

Bidimensional Description and the Measurement of Association between Variables refer to the analysis of relationships between two variables within a dataset. This analysis seeks to comprehend the nature and magnitude of the connections between these two variables.

The measurement of association between variables refers to how the two variables are interconnected or correlated. Correlation is a statistical concept that allows us to evaluate the direction and strength of linear relationships between two variables. Various types of correlation coefficients can be employed to quantify these relationships, including Pearson's correlation coefficient for continuous variables, Spearman's rank correlation coefficient for ordinal variables, and the Chi-square test.

The aim of this practical exercise is to apply these different measures to grasp the intensity of the links between two variables.

The first part of this work includes an overview of the concepts of Pearson's correlation coefficient (see Section 4.1), Spearman's rank correlation coefficient (see Section 4.2), and the Chi-square test (see Section 4.3).

The second part of this practical exercise outlines the tasks assigned to the engineering students.

Part 1

4.1 Pearson Correlation Coefficient

The Pearson correlation coefficient serves as a statistical measure to evaluate the strength and direction of a linear relationship between two quantitative variables. It helps determine to what extent variations in one variable are associated with variations in another variable.
The formula (4.1) for computing the Pearson correlation coefficient between two variables X and Y is as follows:

$$r = \frac{n\sum XY - (\sum X)(\sum Y)}{\sqrt{n\sum X^2 - (\sum X)^2} \times \sqrt{n\sum Y^2 - (\sum Y)^2} \tag{4.1}$$

where,
n is the sample size.

∗ Pearson returns a value between -1 and 1.

∗ +1 indicates a perfect positive correlation.

∗ -1 indicates a perfect negative correlation between the ranks.

∗ 0 = no correlation between the ranks.

4.2 Spearman Correlation Coefficient (Rho)

The Spearman correlation coefficient is a statistical measure used to assess the strength and direction of a relationship between two ordinal variables, even when this relationship is not strongly linear (denoted by equation (4.2)). Unlike the Pearson correlation coefficient, which is applied to quantitative variables and specifically measures linear relationships, the Spearman correlation coefficient is intended for variables with ordered values, but it does not require a linear relationship.

Spearman's methodology relies on ranking the data rather than considering their exact values. The original values are transformed into ranks, which represent their relative positions within the distribution. Subsequently, the Spearman correlation coefficient is calculated by comparing the ranks of the two variables to determine whether variations in one variable are associated with variations in the other.

$$\rho = 1 - \frac{6\sum_{i=1}^{n} D_i^2}{n(n^2-1)} \tag{4.2}$$

where:
D_i: corresponds to differences between the ranks of corresponding data points for X and Y (see equation 4.3).

$$D_i = Ri - Si \tag{4.3}$$

where:
$Ri = \text{Rank}(x_i)$: corresponds to the rank of observation x_i in the X column.
$Si = \text{Rank}(y_i)$: corresponds to the rank of observation y_i in the Y column.

$\ast$ Spearman returns a value between -1 and 1.

$\ast$ $+1$ indicates a perfect positive correlation.

$\ast$ -1 indicates a perfect negative correlation between the ranks.

$\ast$ $0 =$ no correlation between the ranks.

4.3 Chi-Square Test (χ^2)

The test of independence between two variables, also known as the chi-square test, is a statistical method used to determine whether there is a significant association between two nominal or categorical variables. It helps assess whether the observed frequencies in different categories of the variables differ from what would be expected if the variables were independent.

The chi-square test is based on the calculation of a statistic that compares the observed frequencies in a contingency table (cross-tabulation) with the expected frequencies, calculated assuming that the variables are independent.

More specifically, the process involves the following steps:

1. Construction of a contingency table (T): A two-dimensional table is created where the categories of the two variables are crossed to obtain the observed frequencies in each cell of the table.

2. Calculation of expected frequencies: Expected frequencies are calculated using the total margins of the table and the assumption of independence between variables using equation (4.4).

$$T_{ij} = \frac{L_i \times C_j}{Toral} \tag{4.4}$$

where:
L_i: is the total of Row i
L_i: is the total of Column j
$Total$: is Total Number of Observations

3. Calculation of the chi-square statistic (χ^2): The chi-square statistic is calculated by comparing the observed and expected frequencies for each cell of the table (equation 4.5). This measures the deviation between the observed data and what would be expected under independence.

$$\chi^2 = \sum_{i=1}^{l}\sum_{j=1}^{c} \frac{(O_{ij} - T_{ij})^2}{T_{ij}} \tag{4.5}$$

Where:
O_{ij} is the observed value in each cell of observed data table.
T_{ij} is the expected value in each cell of expected data table.

4. Interpretation of the result: The chi-square statistic is compared to a

critical value in a distribution table. If the calculated statistic exceeds this critical value, the null hypothesis of independence is rejected, and it is concluded that there is a significant association between the variables.

Part 2

Work requested

Exercise 1

Table 4.1 represents the daily reports (over a period of 10 days) of the sales quantities of two products.

Write a function that takes this table as a parameter and calculates the Pearson correlation coefficient.

Assume a critical value for the Pearson correlation coefficient of 5%: 0.576.

Determine if there is a correlation, whether linear or not, between the sales of the two products.

Days	**Product 1 (X)**	**Product 2 (Y)**
1	31	50
2	31	55
3	32	52
4	33	56
5	33	63
6	34	65
7	35	69
8	36	90
9	37	110
10	38	150

Table 4.1: Daily Reports

Exercise 2

During a recruitment process, a company aims to assess whether there is a correlation between the success of new candidates (individuals) in the first interview and their success in the second interview (see Table 4.2).

- Write a function that calculates the ranks (Ri, Si). This function takes as parameters the codes of the individuals and their respective rankings in interviews 1 and 2.

- Write a function that takes the ranks (Ri, Si) as parameters and calculates the Spearman correlation coefficient.

Assume a critical value for the Spearman correlation coefficient of 5%: 0.503.

Determine if there is a linear correlation.

Individual	**Interview 1 (X)**	**Interview 2 (Y)**
s1	7	10
s2	10	12
s3	1	4
s4	6	7
s5	9	11
s6	13	9
s7	3	2
s8	5	4
s9	11	5
s10	9	11
s11	6	6
s12	4	1

Table 4.2: Recruitment Rankings

Exercise 3

Given Table 4.3, is there independence between age and the number of accidents at a 5% significance level? (We are given χ^2 (critical) = 21.03)

		Driver's Age					
		21-30	**31-40**	**41-50**	**51-60**	**61-70**	**Total**
Number of Accidents	**0**	748	821	786	720	672	**3747**
	1	74	60	51	66	50	**301**
	2	31	25	22	16	15	**109**
	More than 2	9	10	6	5	7	**37**
	Total	**862**	**916**	**865**	**807**	**744**	**4194**

Table 4.3: Relationship Between Driver's Age and Number of Accidents

Note

You can find the Critical Values tables for Pearson, Spearman, and Chi-Square by following this link: Workshop-Pearson-Spearman-Chi-2
Additional theoretical explanations can be found in the linked lecture handout.

Solutions

Solution for Exercise 1

Method 1

```
import numpy as np
from math import sqrt
def Calculate_Pearson(arr):
    X = arr[0]
    Y = arr[1]
    SX = sum(X)
    SY = sum(Y)
    productXY = X * Y
    sproductXY = sum(productXY)
    n = arr.shape[1]
    term1 = n * sproductXY - SX * SY
    spowerX = sum(pow(X, 2))
    spowerY = sum(pow(Y, 2))
    powerSX = pow(SX, 2)
    powerSY = pow(SY, 2)
    term2 = sqrt(n * spowerX - powerSX) * sqrt(n * spowerY - powerSY)
    pearson = term1 / term2
    return pearson
arr = np.array([[31, 31, 32, 33, 33, 34, 35, 36, 37, 38], [50, 55, 52, 56, 63,
65, 69, 90, 110, 150]])
threshold = 0.576
pearson = Calculate_Pearson(arr)
print("Pearson correlation coefficient = ", round(pearson, 2))
if pearson >= threshold:
    print("Positive linear correlation")
elif pearson <= threshold:
    print("Negative linear correlation")
```

```
else:
    print("No linear correlation")
```

Method 2

```
import numpy as np
from scipy.stats import pearsonr
def Calculate_Pearson(arr):
    X = arr[0]
    Y = arr[1]
    correlation, _ = pearsonr(X, Y)
    return correlation
arr = np.array([[31, 31, 32, 33, 33, 34, 35, 36, 37, 38], [50, 55, 52, 56, 63,
65, 69, 90, 110, 150]])
threshold = 0.576
pearson = Calculate_Pearson(arr)
print("Pearson correlation coefficient = ", round(pearson, 2))
if pearson >= threshold:
    print("Positive linear correlation")
elif pearson <= threshold:
    print("Negative linear correlation")
else:
    print("No linear correlation")
```

Solution for Exercise 2

Method 1

```
import numpy as np
def Calculate_Ranks(R):
    # Get indices sorted by column 1
    sorted_indices = np.argsort(R[:, 1])
    R = R[sorted_indices]
    n = R.shape[0]
    raw_ranks = np.arange(1, n+1).T
    ranks = np.arange(1, n+1).T
    R = np.column_stack([R, raw_ranks, ranks])
    index = []
    i = -1
    while (i < n-2):
        i = i + 1
        index = []
        p = -1
        while(R[i,1] == R[i+1,1]):
            index.append(i)
            i = i + 1
            p = 1
        if p == 1:
        index.append(i)
        if len(index) != 0:
            s = 0
            for j in range(len(index)):
                s = s + R[index[j], 2]
            m = s / len(index)
            R[index, 3] = m
    R = np.column_stack([R[:, 0], R[:, 3]])
    # Get indices sorted by column 1
    sorted_indices = np.argsort(R[:, 0])
    R = R[sorted_indices]
    return R


def Calculate_Spearman(Rx, Ry):
    final_array = np.column_stack([Rx, Ry[:, 1]])
    SD = sum(pow((final_array[:, 1] - final_array[:, 2]), 2))
    n = Rx.shape[0]
    a = float(n * (pow(n, 2) - 1))
    spearman = 1 - ((6 * SD) / a)
    return spearman


arr = np.array([[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12], [7, 10, 1, 6, 9, 13, 3,
5, 11, 9, 6, 4], [10, 12, 4, 7, 11, 9, 2, 4, 5, 11, 6, 1]], dtype='f')
x = arr[0:2, :]
x = x.T
y = arr[0:3:2, :]
y = y.T
Rx = Calculate_Ranks(x)
Ry = Calculate_Ranks(y)
spearman = Calculate_Spearman(Rx, Ry)
print("Spearman correlation coefficient = ", round(spearman, 2))
threshold = 0.503
if spearman >= threshold:
        print("Positive linear correlation")
elif spearman <= threshold:
        print("Negative linear correlation")
else:
        print("No linear correlation")
```

Method 2

```
from scipy import stats
import numpy as np

def Calculate_Spearman(x, y):
        res = stats.spearmanr(x, y)
        spearman = res.statistic
        return spearman


arr = np.array([[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12], [7, 10, 1, 6, 9, 13, 3,
5, 11, 9, 6, 4], [10, 12, 4, 7, 11, 9, 2, 4, 5, 11, 6, 1]], dtype='f')
x = arr[1, :]
y = arr[2, :]
spearman = Calculate_Spearman(x, y)
print("Spearman correlation coefficient = ", round(spearman, 2))
threshold = 0.503
if spearman >= threshold:
            print("Positive linear correlation")
elif spearman <-threshold:
            print("Negative linear correlation")
else:
            print("No linear correlation")
```

Solution for Exercise 3

```
import numpy as np
def Calculate_ChiSquared(OriginalTable):
    SumColumn = np.sum(OriginalTable, axis=1)
    SumRow = np.sum(OriginalTable, axis=0)
    Total = sum(SumColumn)
    TheoreticalTable = np.zeros((OriginalTable.shape[0], OriginalTable.shape[1]))
    for x in range(OriginalTable.shape[0]):
        for y in range(OriginalTable.shape[1]):
            TheoreticalTable[x, y] = round(((SumRow[y] * SumColumn[x]) /
Total), 2)
    chi_squared = round(sum(sum(pow((OriginalTable - TheoreticalTable),
2) / TheoreticalTable)), 3)
    return chi_squared

chi_squared_critical = 21.03
OriginalTable = np.array([[748, 821, 786, 720, 672], [74, 60, 51, 66, 50],
[31, 25, 22, 16, 15], [9, 10, 6, 5, 7]], dtype='f')
print("Null Hypothesis (H0): Age and number of accidents are independent"
)
print("Alternative Hypothesis (H1): Age and number of accidents are depen-
dent" )
chi_squared = Calculate_ChiSquared(OriginalTable)
print("Chi-squared =", chi_squared)
if chi_squared < chi_squared_critical:
    print("We accept H0 and reject H1" )
else:
    print ("We accept H1 and reject H0" )
```

Conclusion

In bivariate analysis, correlation serves as a crucial measure for examining the relationship between two variables. It's a statistical concept that helps evaluate both the direction and strength of linear relationships between these variables. In this workshop, we have chosen the most common correlations: Pearson's correlation coefficient for continuous variables and Spearman's rank correlation coefficient for ordinal variables to quantify the relationship between two variables using Python language. Additionally, the Chi-square test is utilized to assess the dependence between two variables.

WORKSHOP 5 MULTIDIMENSIONAL ANALYSIS: PCA AND MCA

Principal Component Analysis (PCA) and Multiple Correspondence Analysis (MCA) are two multidimensional analysis approaches used to explore and visualize relationships between quantitative variables (in the case of PCA) or categorical variables (in the case of MCA).

PCA aims to simplify the complexity of a dataset by transforming the original variables into a new coordinate system called principal components.

On the other hand, MCA is employed to examine the relationships between categorical variables.

The goal of this practical exercise is to apply two methods for assessing the relationships among multiple variables. The initial section provides an introduction to PCA, illustrated with a sample example. In the latter part of this work, we introduce the Correspondence Analysis method (CA) and subsequently the Multiple Correspondence Analysis (MCA) method, accompanied by illustrations and practical examples.

Part 1

5.1 Principal Component Analysis

PCA, or Principal Component Analysis, is one of the most commonly used approaches in multivariate data analysis. It is useful for exploring multidimensional datasets containing quantitative variables.

It aims to transform a dataset with potentially correlated variables into a set of linearly uncorrelated variables called principal components. These principal components capture the maximum variance present in the data, allowing for simplification and interpretation of complex datasets.
Theoretical explanations can be found in the lecture handout.

5.1.1 Objectives of Principal Component Analysis

The fundamental objective of PCA is to represent the data as effectively as possible in a reduced-dimensional space compared to the original observations, which can have higher dimensions (with Xj variables). This approach aims to achieve the following objectives:

- Simplification of Reality: PCA seeks to simplify the complexity inherent in multidimensional data by expressing it through a more limited set of dimensions while preserving the essence of the information.
- Concentration of Initial Information: It aims to concentrate and focus the information dispersed in the initial multidimensional space by projecting it into a reduced-dimensional space, highlighting underlying trends and structures.
- Description of Maximum Variability in a Reduced Space: PCA aims to capture as much variability as possible from the original data using a limited number of principal components. These components help describe the most significant variations among observations.

5.1.2 Performing Principal Component Analysis in SPSS

SPSS, short for Statistical Package for the Social Sciences, is widely utilized for statistical analysis, offering an array of tools for data management, manipulation, and analysis. Within SPSS, Principal Component Analysis is employed for dimensionality reduction and data visualization.

Performing PCA in SPSS entails several steps:

1. Importing the dataset
2. Executing PCA: Utilize the PCA function in SPSS to conduct the analysis. Specify the variables to be included and any additional parameters.
3. Result interpretation: Analyze the output generated by SPSS, which includes eigenvalues, eigenvectors, and variance explained by each principal component. Interpret the principal components and their significance in elucidating data variation
4. Result visualization: Employ graphs to visually represent the principal components and their associations with the original variables

5.1.3 Practical Application of PCA with SPSS: An Illustrative Example

In this section, we will apply PCA to analyze the distribution of a country's expenditures covering the period from 1872 to 1971 (see Table 5.1). This distribution is represented in the table below.

These expenditures are expressed in percentages (%) for 11 categories:
- Public Authorities (PAU),
- Agriculture (AGR),
- Commerce and Industry (CAI),
- Transportation (TRA),
- Housing and Urban Planning (HAU),
- Education and Culture (EAC),

- Social Action (SAC),
- Veterans (VET),
- Defense (DEF),
- Debt Repayment (DET),
- Miscellaneous (MIS).

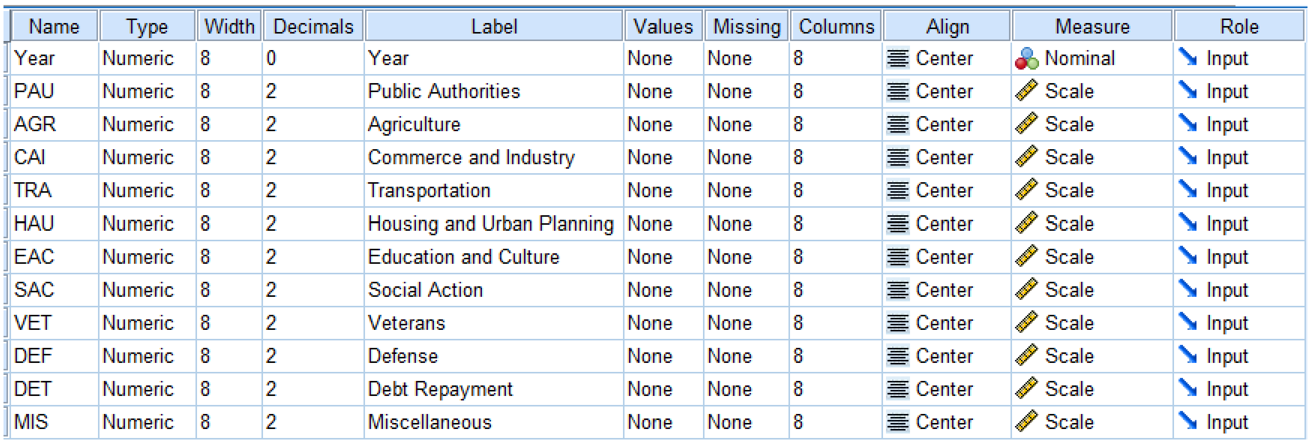
Year	**PAU**	**AGR**	**CAI**	**TRA**	**HAU**	**EAC**	**SAC**	**VET**	**DEF**	**DET**	**MIS**
1872	18	0.5	0.1	6.7	0.5	2.1	2	0	26.4	41.5	2.1
1880	14.1	0.8	0.1	15.3	1.9	3.7	0.5	0	29.8	31.3	2.5
1890	13.6	0.7	0.7	6.8	0.6	7.1	0.7	0	33.8	34.4	1.7
1900	14.3	1.7	1.7	6.9	1.2	7.4	0.8	0	37.7	26.2	2.2
1903	10.3	1.5	1.4	9.3	0.6	8.5	0.9	0	38.4	27.2	3
1906	13.4	1.4	0.5	8.1	0.7	8.6	1.8	0	38.5	25.3	1.9
1909	13.5	1.1	0.5	9	1.6	9	3.4	0	36.8	23.5	2.6
1912	12.9	1.4	0.3	9.4	0.6	9.3	4.3	0	41.1	19.4	1.3
1920	12.3	0.3	0.1	11.9	2.4	3.7	1.7	1.9	42.4	23.1	0.2
1923	7.6	1.2	3.2	5.1	0.6	5.6	1.8	10	29	35	0.9
1926	10.5	0.3	0.4	4.5	1.8	6.6	2.1	10.1	19.9	41.6	2.3
1939	10	0.6	0.6	9	1	8.1	3.2	11.8	28	25.8	2
1932	10.6	0.8	0.3	8.9	3	10	6.4	13.4	27.4	19.2	0
1936	8.8	2.6	1.4	7.8	1.4	12.4	6.2	11.3	29.3	18.5	0.4
1938	10.1	1.1	1.2	5.9	1.4	9.5	6	5.9	40.7	18.2	0
1947	15.6	1.6	10	11.4	7.6	8.8	4.8	3.4	32.2	4.6	0

1950	11.2	1.3	16.5	12.4	15.8	8.1	4.9	3.4	20.7	4.2	1.5
1953	12.9	1.5	7	7.9	12.1	8.1	5.3	3.9	36.1	5.2	0
1956	10.9	5.3	9.7	7.6	9.6	9.4	8.5	4.6	28.2	6.2	0
1959	13.1	4.4	7.3	5.7	9.8	12.5	8	5	26.7	7.5	0
1962	12.8	4.7	7.5	6.6	6.8	15.7	9.7	5.3	24.5	6.4	0.1
1965	12.4	4.3	8.4	9.1	6	19.5	10.6	4.7	19.8	3.5	1.8
1968	11.4	6	9.5	5.9	5	21.1	10.7	4.2	20	4.4	1.9
1971	12.8	2.8	7.1	8.5	4	23.8	11.3	3.7	18.8	7.2	0

Table 5.1: Expenditure Distribution for 11 Categories

Step 1

Define variable types in the **Variable View** window (see Figure 5.1).

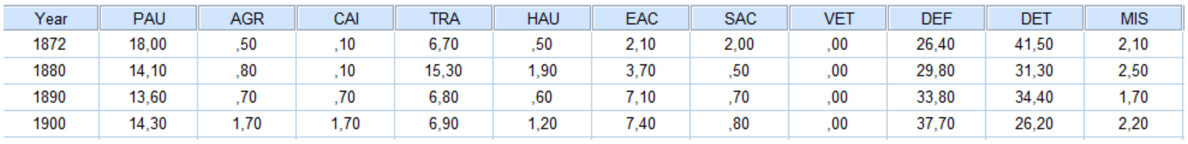


Name	Type	Width	Decimals	Label	Values	Missing	Columns	Align	Measure	Role
Year	Numeric	8	0	Year	None	None	8	Center	Nominal	Input
PAU	Numeric	8	2	Public Authorities	None	None	8	Center	Scale	Input
AGR	Numeric	8	2	Agriculture	None	None	8	Center	Scale	Input
CAI	Numeric	8	2	Commerce and Industry	None	None	8	Center	Scale	Input
TRA	Numeric	8	2	Transportation	None	None	8	Center	Scale	Input
HAU	Numeric	8	2	Housing and Urban Planning	None	None	8	Center	Scale	Input
EAC	Numeric	8	2	Education and Culture	None	None	8	Center	Scale	Input
SAC	Numeric	8	2	Social Action	None	None	8	Center	Scale	Input
VET	Numeric	8	2	Veterans	None	None	8	Center	Scale	Input
DEF	Numeric	8	2	Defense	None	None	8	Center	Scale	Input
DET	Numeric	8	2	Debt Repayment	None	None	8	Center	Scale	Input
MIS	Numeric	8	2	Miscellaneous	None	None	8	Center	Scale	Input

Figure 5.1: PCA: Step 1

Step 2

Copy the data into the **Data View** window (see Figure 5.2).



Year	PAU	AGR	CAI	TRA	HAU	EAC	SAC	VET	DEF	DET	MIS
1872	18,00	,50	,10	6,70	,50	2,10	2,00	,00	26,40	41,50	2,10
1880	14,10	,80	,10	15,30	1,90	3,70	,50	,00	29,80	31,30	2,50
1890	13,60	,70	,70	6,80	,60	7,10	,70	,00	33,80	34,40	1,70
1900	14,30	1,70	1,70	6,90	1,20	7,40	,80	,00	37,70	26,20	2,20

Figure 5.2: PCA: Step 2

Step 3

Go to
Analyze -> Dimension Reduction -> Factor.
Select the desired variables by checking them, then click the right-pointing arrow.

In this window, you have five buttons: **Descriptives**, **Extraction**, **Rotation**, **Scores**, **Options** (see Figure 5.3).

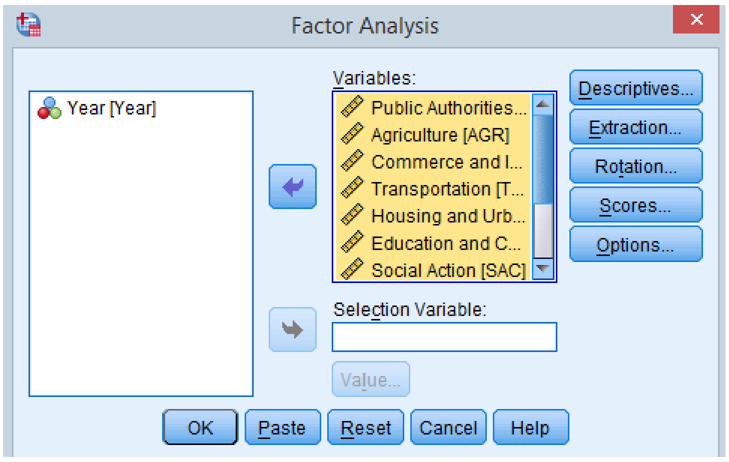
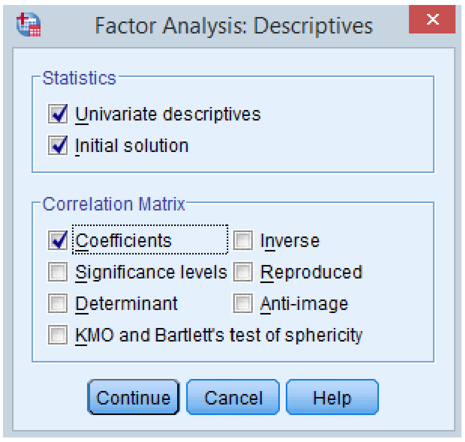


Figure 5.3: PCA: Step 3

Step 4

In **Descriptives**, check "Univariate descriptives" and "Initial solution". In the "Correlation Matrix," click "Coefficients" -> Then click "Continue" (see Figure 5.4).


Factor Analysis: Descriptives

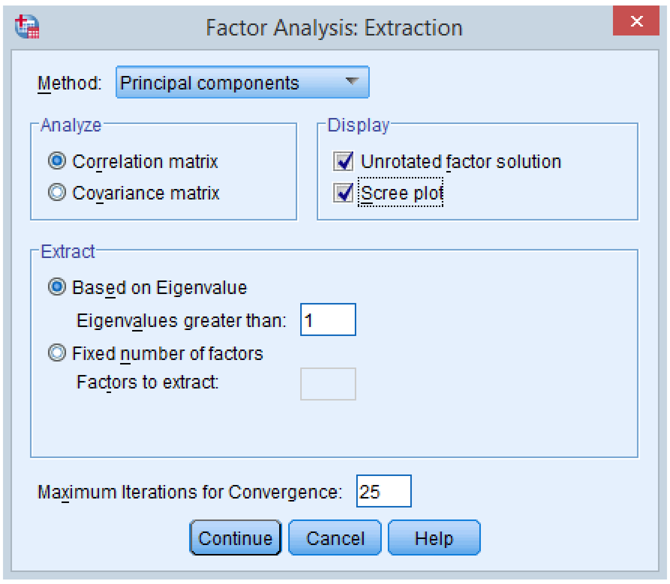
Statistics
- [✓] Univariate descriptives
- [✓] Initial solution

Correlation Matrix
- [✓] Coefficients
- [ ] Inverse
- [ ] Significance levels
- [ ] Reproduced
- [ ] Determinant
- [ ] Anti-image
- [ ] KMO and Bartlett's test of sphericity

Figure 5.4: PCA: Step 4

Step 5

In **Extraction**, check "Scree plot" and "Correlation matrix" -> Then click "Continue" (see Figure 5.5).


Factor Analysis: Extraction

Method: Principal components

Analyze
- [✓ Correlation matrix]
- Covariance matrix

Display
- [✓ Unrotated factor solution]
- [✓ Scree plot]

Extract
- [✓ Based on Eigenvalue]
  - Eigenvalues greater than: 1
- Fixed number of factors
  - Factors to extract:

Maximum Iterations for Convergence: 25

Figure 5.5: PCA: Step 5

Step 6

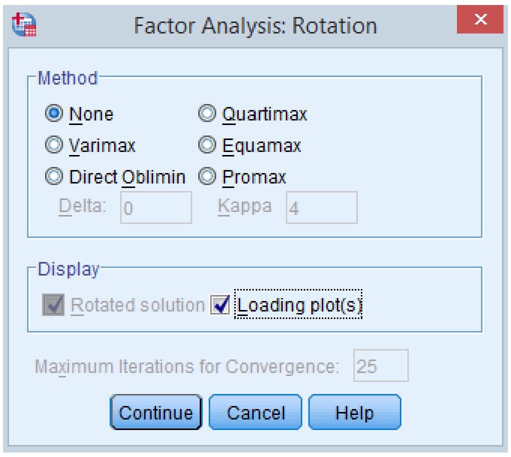
In **Rotation**, check "Loading plot(s)". This option allows obtaining a representation on different axes (see Figure 5.6).

Step 7

In **Scores**, check "Save as variables" (Regression method) and "Display factor score matrix" (see Figure 5.7).

Step 8

In **Options**, choose the "Sorted by size" option -> Then click "Continue" (see Figure 5.8).

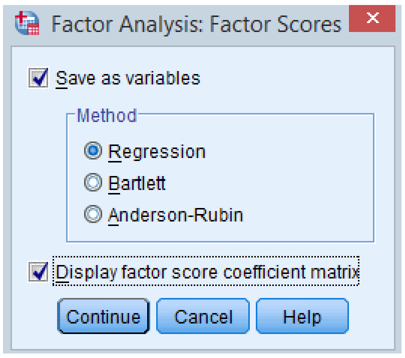

Factor Analysis: Rotation

Method
[✓ None] Quartimax
Varimax Equamax
Direct Oblimin Promax
Delta: 0 Kappa 4

Display
[✓ Rotated solution] [✓ Loading plot(s)]

Maximum Iterations for Convergence: 25

Figure 5.6: PCA: Step 6


Factor Analysis: Factor Scores

[✓ Save as variables]

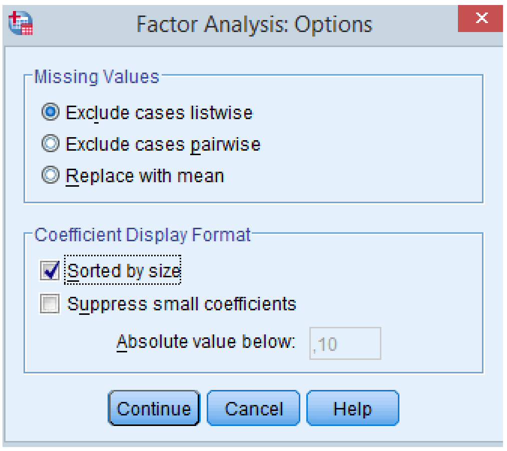
Method
[✓ Regression]
Bartlett
Anderson-Rubin

[✓ Display factor score coefficient matrix]

Figure 5.7: PCA: Step 7

Step 9

Finally, click OK. The "IBM SPSS Statistics" file will be generated.



Factor Analysis: Options

Missing Values

- [✓ Exclude cases listwise]
- Exclude cases pairwise
- Replace with mean

Coefficient Display Format

- [✓ Sorted by size]
- Suppress small coefficients

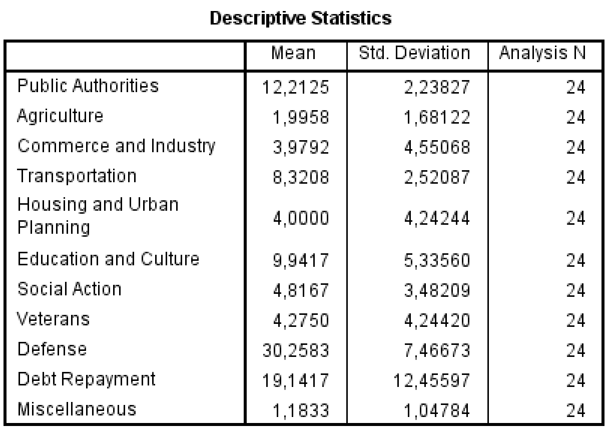
Absolute value below: ,10

Continue Cancel Help

Figure 5.8: PCA: Step 8

Analysis of Results

The first table, named **Descriptive Statistics**, presents the means and standard deviations of all variables (see Table 5.2).



Descriptive Statistics

	Mean	Std. Deviation	Analysis N
Public Authorities	12,2125	2,23827	24
Agriculture	1,9958	1,68122	24
Commerce and Industry	3,9792	4,55068	24
Transportation	8,3208	2,52087	24
Housing and Urban Planning	4,0000	4,24244	24
Education and Culture	9,9417	5,33560	24
Social Action	4,8167	3,48209	24
Veterans	4,2750	4,24420	24
Defense	30,2583	7,46673	24
Debt Repayment	19,1417	12,45597	24
Miscellaneous	1,1833	1,04784	24

Table 5.2: Descriptive Statistics

Correlation Matrix

Displays the correlations between all variables (see Figure 5.9).

Correlation Matrix

		Public Authorities	Agriculture	Commerce and Industry	Transportation	Housing and Urban Planning	Education and Culture	Social Action	Veterans	Defense	Debt Repayment	Miscellaneous
Correlation	Public Authorities	1,000	-,085	-,010	,233	,042	-,150	-,131	-,687	,101	,034	,149
	Agriculture	-,085	1,000	,602	-,276	,433	,731	,806	,044	-,448	-,695	-,277
	Commerce and Industry	-,010	,602	1,000	,096	,888	,468	,615	,013	-,530	-,803	-,333
	Transportation	,233	-,276	,096	1,000	,170	-,213	-,203	-,313	,158	-,148	,114
	Housing and Urban Planning	,042	,433	,888	,170	1,000	,232	,487	,035	-,372	-,760	-,427
	Education and Culture	-,150	,731	,468	-,213	,232	1,000	,875	,157	-,524	-,670	-,249
	Social Action	-,131	,806	,615	-,203	,487	,875	1,000	,288	-,567	-,808	-,530
	Veterans	-,687	,044	,013	-,313	,035	,157	,288	1,000	-,417	-,049	-,377
	Defense	,101	-,448	-,530	,158	-,372	-,524	-,567	-,417	1,000	,262	,020
	Debt Repayment	,034	-,695	-,803	-,148	-,760	-,670	-,808	-,049	,262	1,000	,554
	Miscellaneous	,149	-,277	-,333	,114	-,427	-,249	-,530	-,377	,020	,554	1,000

Figure 5.9: Correlation Matrix

Total Explained Variance

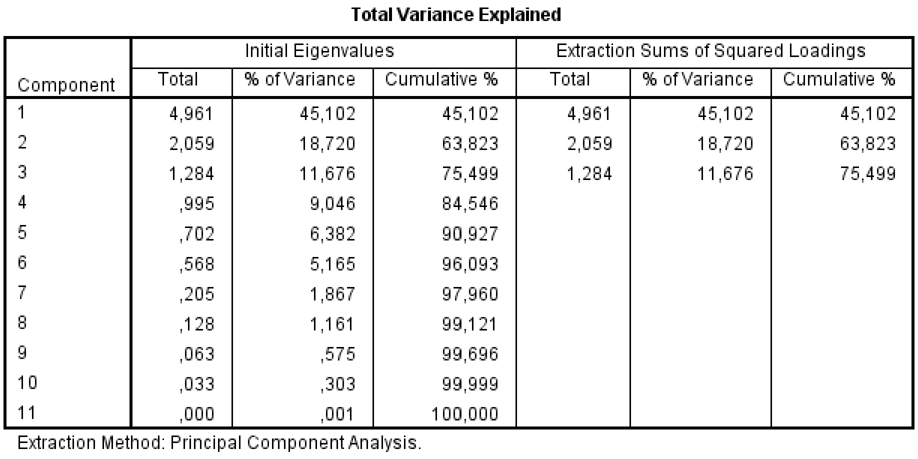
The eigenvalues are listed in the "Total" column and reflect the inertia carried by the principal axes (see Table 5.3).

Each principal axis contributes to the total inertia, which is expressed as a percentage of inertia relative to the eigenvalue and total inertia.

The highest eigenvalue in the correlation matrix is 4.96, associated with the first principal axis, explaining 45.10% of the variability. Similarly, the eigenvalue of 2.059 is associated with the principal axis D2, explaining 18.72% of the variance.

We choose to retain the first three principal axes, which explain 75.49% of the variance. This selection is clearly illustrated in the scree plot.

To decide on the number of principal axes to retain, two rules are generally applied:
First rule: Choose the number of axes based on the minimum level of information to be retained. For example, if you aim to retain at least 80% of the information.
Second rule: Examine the scree plot of eigenvalues and keep only the values to the left of the inflection point. Graphically, start from the components providing the least information (those on the right), connect the almost aligned points with a line, and retain only the axes above this line (see Figure 5.10).



Total Variance Explained

Component	Initial Eigenvalues			Extraction Sums of Squared Loadings		
	Total	% of Variance	Cumulative %	Total	% of Variance	Cumulative %
1	4,961	45,102	45,102	4,961	45,102	45,102
2	2,059	18,720	63,823	2,059	18,720	63,823
3	1,284	11,676	75,499	1,284	11,676	75,499
4	,995	9,046	84,546			
5	,702	6,382	90,927			
6	,568	5,165	96,093			
7	,205	1,867	97,960			
8	,128	1,161	99,121			
9	,063	,575	99,696			
10	,033	,303	99,999			
11	,000	,001	100,000			

Extraction Method: Principal Component Analysis.

Table 5.3: Total Variance Explained

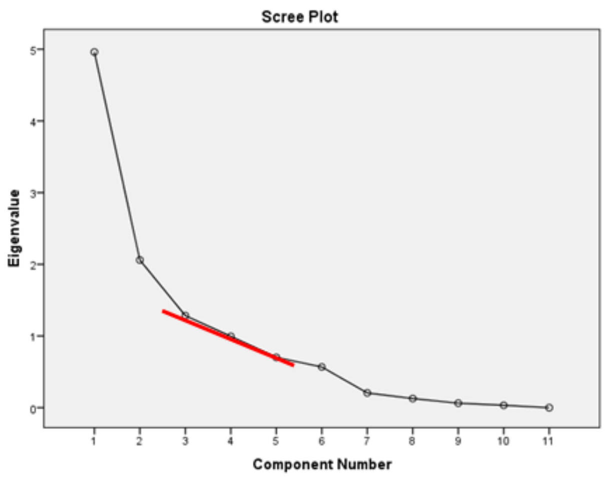


Figure 5.10: Collapse diagram

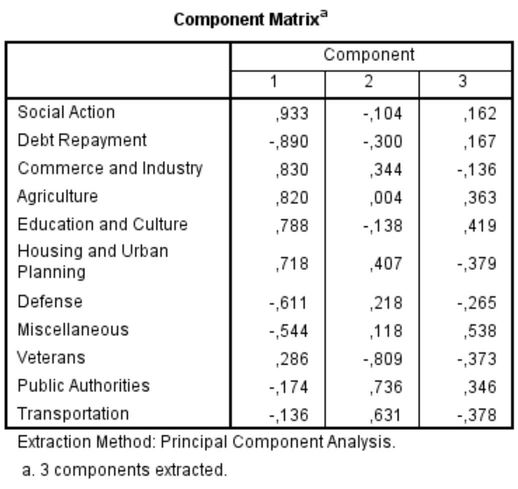
Component Matrix

The correlation coefficients between the original variables and the principal components are provided in the Component Matrix (see Figure 5.11).

The first principal component (PC1) is positively correlated with the variables Social Action, Commerce-Industry, and Agriculture. In contrast, it has a negative correlation with the variables Defense and Debt Repayment.

Similarly, the second principal component (PC2) shows a negative correlation with the variable Veterans and a positive correlation with the variables Public Authorities and Transportation. The other correlations are less pronounced.

As for the third component (PC3), it exhibits a relatively significant correlation (compared to other values) with the variable Miscellaneous.



Component Matrix[a]

	Component		
	1	2	3
Social Action	,933	-,104	,162
Debt Repayment	-,890	-,300	,167
Commerce and Industry	,830	,344	-,136
Agriculture	,820	,004	,363
Education and Culture	,788	-,138	,419
Housing and Urban Planning	,718	,407	-,379
Defense	-,611	,218	-,265
Miscellaneous	-,544	,118	,538
Veterans	,286	-,809	-,373
Public Authorities	-,174	,736	,346
Transportation	-,136	,631	-,378

Extraction Method: Principal Component Analysis.

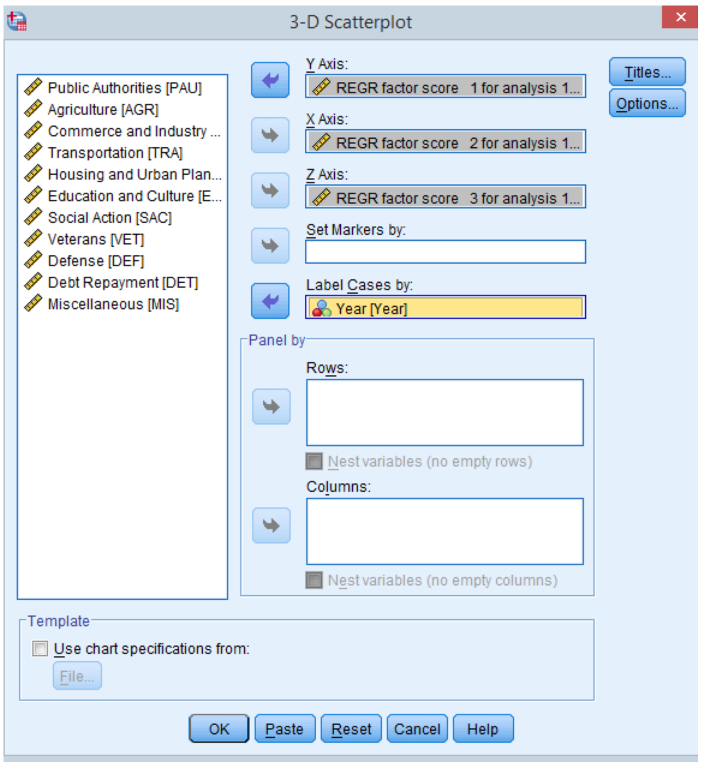
a. 3 components extracted.

Figure 5.11: Component Matrix

Graphical Representation

For the graphical representation (Figure 5.12), click on the "Graphs + Scatter/Dot + 3D Scatter" for 3D representation (Figure 5.13) and on "Overlay Scatter" for 2D representation (Figure 5.14). In this window, click on "Options" and check "Display chart with case labels".

In the representation on the first two principal axes, we observe that the years are divided into three groups: before World War I, between the two World Wars, and after World War II. However, only the year 1920, the first year when a dedicated expenditure category for veterans appears, is positioned with the first group, even though it belongs to the second group.



Y Axis: REGR factor score 1 for analysis 1...

X Axis: REGR factor score 2 for analysis 1...

Z Axis: REGR factor score 3 for analysis 1...

Set Markers by:

Label Cases by: Year [Year]

Figure 5.12: Graphical Configuration

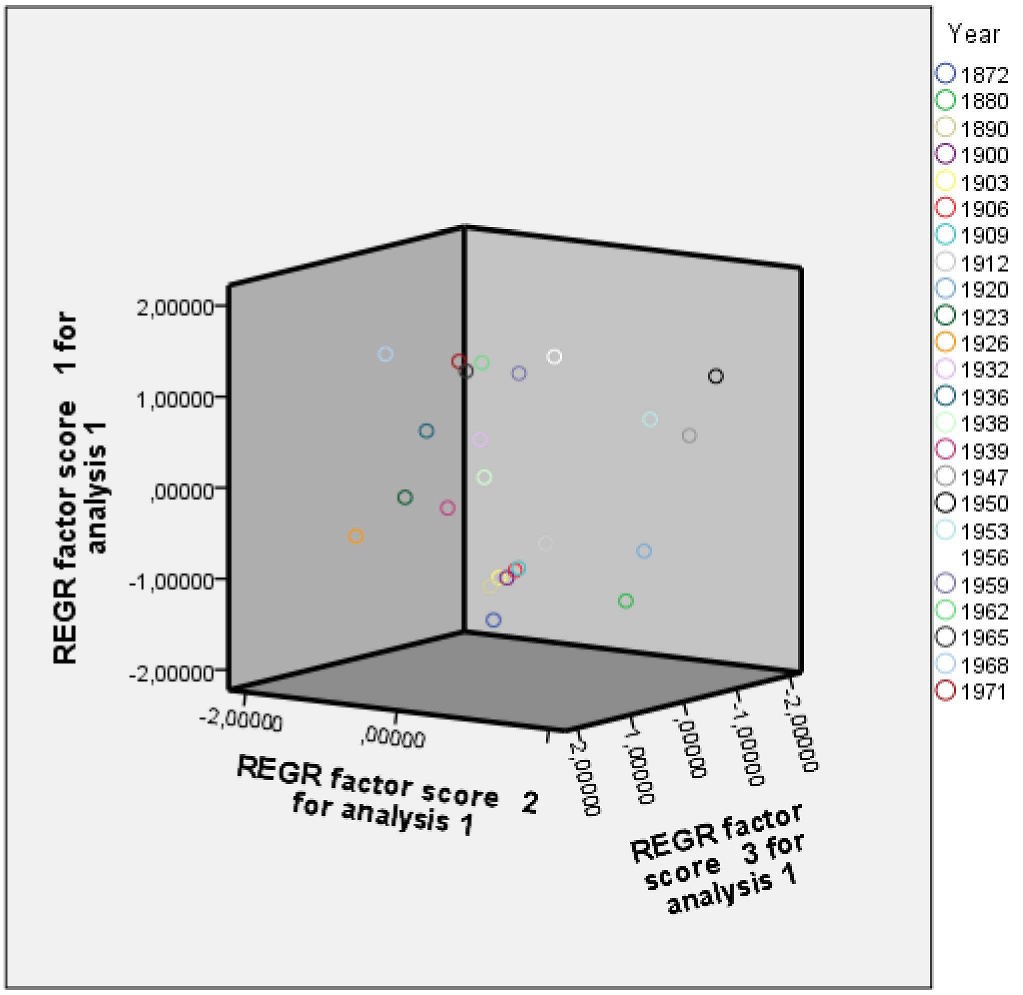


Figure 5.13: 3D Graphical Representation

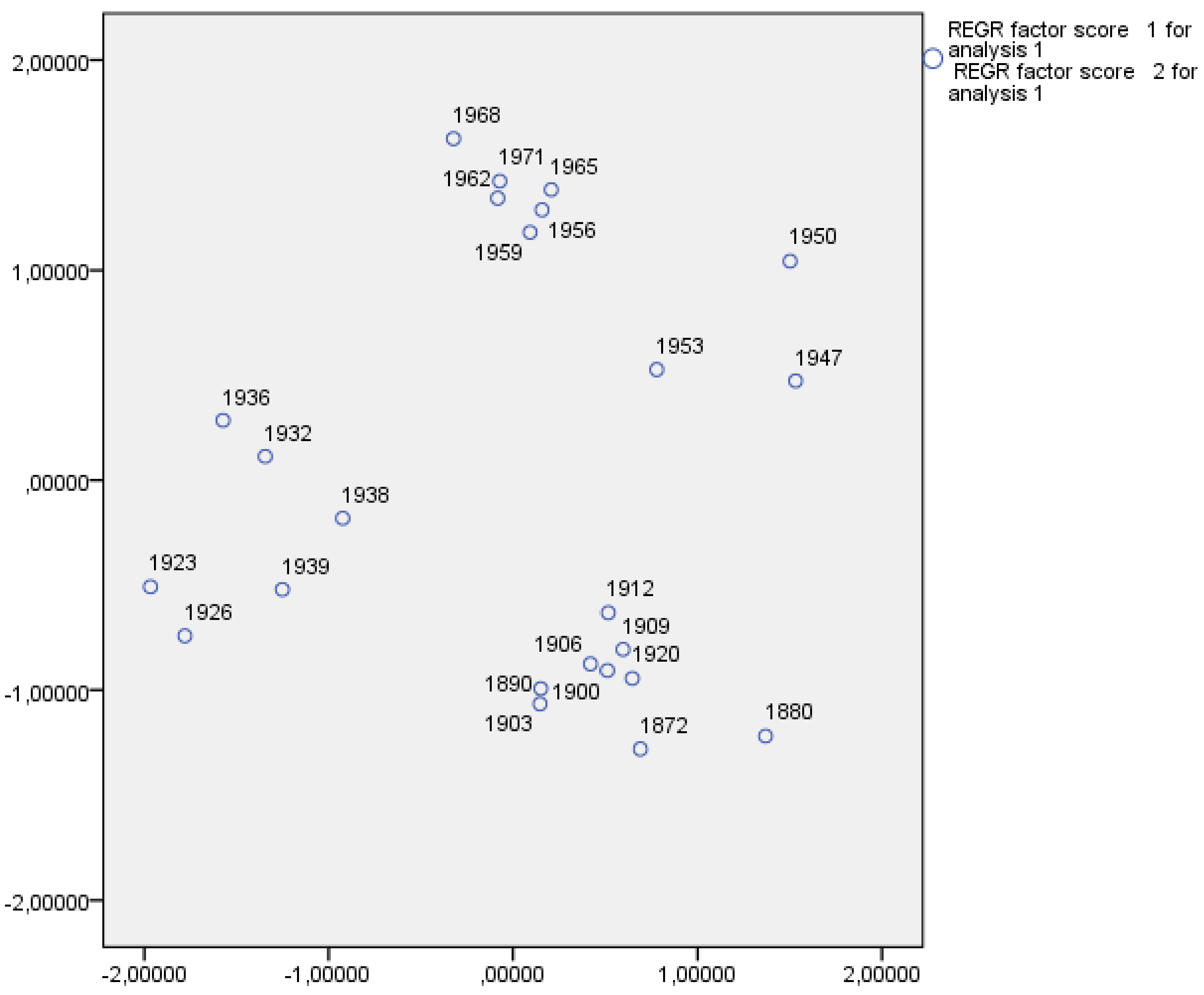


Figure 5.14: 2D Graphical Representation

Part 2

5.2 Correspondence Analysis (CA)

Correspondence Analysis (CA) is a bi-variate statistical technique used to study relationships between categorical variables in a contingency table. It allows for the visualization and interpretation of associations between variable categories.
Theoretical explanations can be found in the lecture handout.

5.2.1 Objectives of Correspondence Analysis

The objective is to construct a configuration where objects sharing the same category are placed close to each other, while objects of different categories are positioned farther apart. Each object is positioned as closely as possible to the corresponding category points. This approach leads to the formation of homogeneous subgroups of objects based on their categories. Similarly, variables are considered homogeneous when they group objects from the same categories into the same subgroups.

5.2.2 Performing Correspondence Analysis in SPSS

SPSS stands out for its intuitive and user-friendly interface, which makes conducting correspondence analysis (CA) easy.

Performing CA in SPSS entails several steps:

1. Importing the dataset
2. Executing CA
3. Interpret Results: Examine the cross-tabulation table to understand the relationship between the variables
4. Visualize Results

5.2.3 Practical Application of CA with SPSS: An Illustrative Example

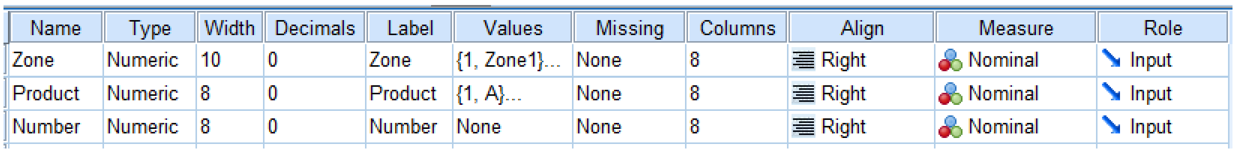
In this section, we will implement Correspondence Analysis to analyze the distribution of products A, B, C, and D in zones 1, 2, and 3.

		Product			
		A	**B**	**C**	**D**
Zone	**1**	5	5	15	15
	2	5	25	5	5
	3	30	5	5	0

Table 5.4: Relationship between Products and Zones

Step 1

Define variable types in the **Variable View** window (see Figure 5.15).

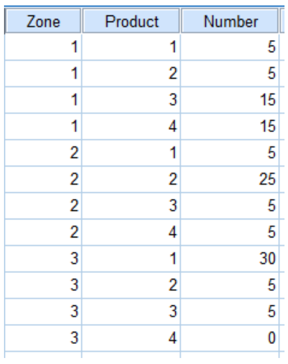


Name	Type	Width	Decimals	Label	Values	Missing	Columns	Align	Measure	Role
Zone	Numeric	10	0	Zone	{1, Zone1}...	None	8	Right	Nominal	Input
Product	Numeric	8	0	Product	{1, A}...	None	8	Right	Nominal	Input
Number	Numeric	8	0	Number	None	None	8	Right	Nominal	Input

Figure 5.15: CA: Step 1

Step 2

Enter the data into the **Data View** window (see Figure 5.16).



Zone	Product	Number
1	1	5
1	2	5
1	3	15
1	4	15
2	1	5
2	2	25
2	3	5
2	4	5
3	1	30
3	2	5
3	3	5
3	4	0

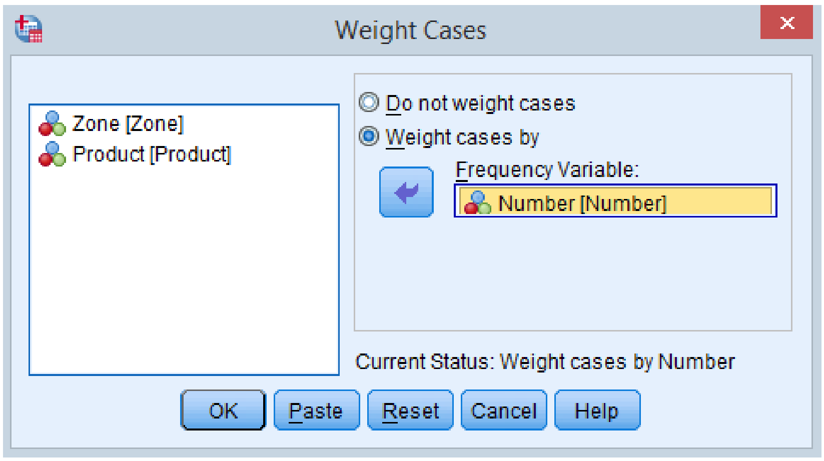
Figure 5.16: CA: Step 2

Step 3

Before proceeding with the analysis, it is necessary to weight the observations. To do this, from the menus, select: **Data -> Weight Cases** (see Figure 5.17).

Step 4

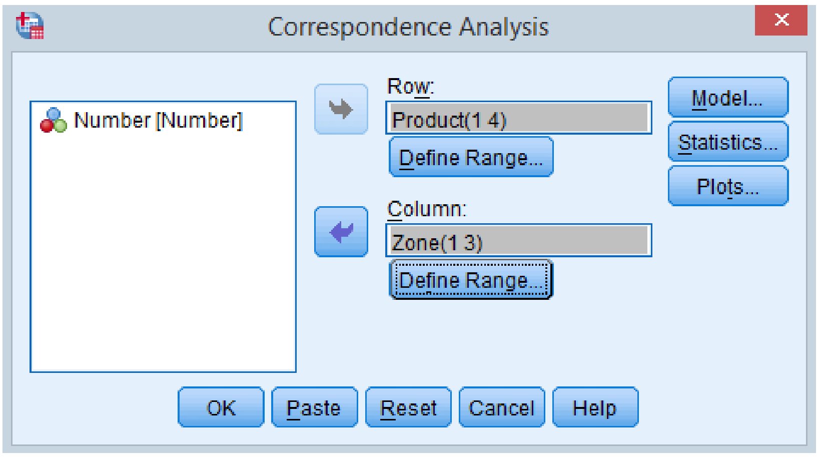
To perform a correspondence analysis, follow these steps from the menus: **Analyze -> Dimension Reduction -> Correspondence Analysis** (see



Do not weight cases
[✓ Weight cases by]
Frequency Variable: Number [Number]
Current Status: Weight cases by Number

Figure 5.17: CA: Step 3

Figure 5.18).
In the row, put "Product," and in the column, put "Zone."
To define the range, click on **Define Range**. For rows, set the maximum value to 4 and the minimum value to 1. Then click **Define Range** again for columns, setting the maximum value to 3 and the minimum value to 1.

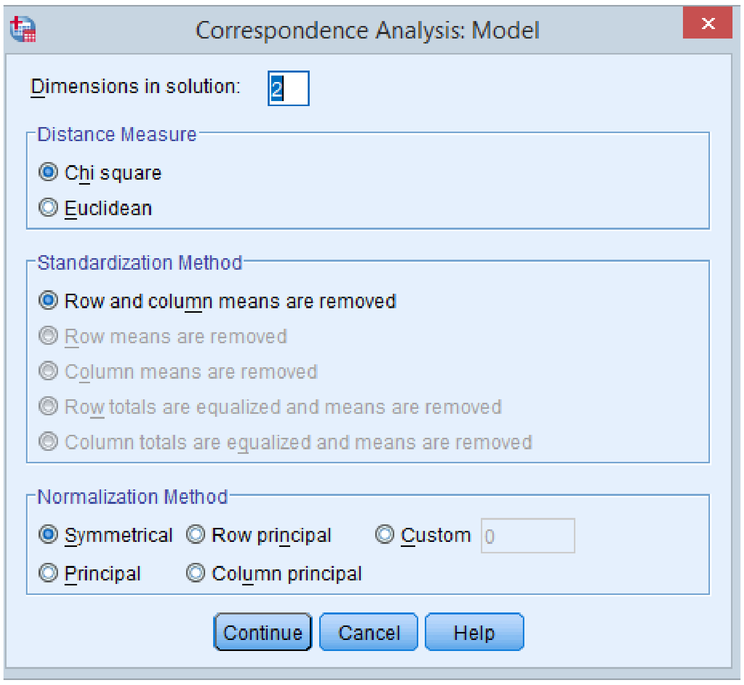


Row: Product(1 4)
Column: Zone(1 3)

Figure 5.18: CA: Step 4

Step 5

Click on **Model** and select the option for the Chi square (see Figure 5.19).



Correspondence Analysis: Model

Dimensions in solution: 2

Distance Measure
[✓ Chi square]
Euclidean

Standardization Method
[✓ Row and column means are removed]
Row means are removed
Column means are removed
Row totals are equalized and means are removed
Column totals are equalized and means are removed

Normalization Method
[✓ Symmetrical] Row principal Custom 0
Principal Column principal

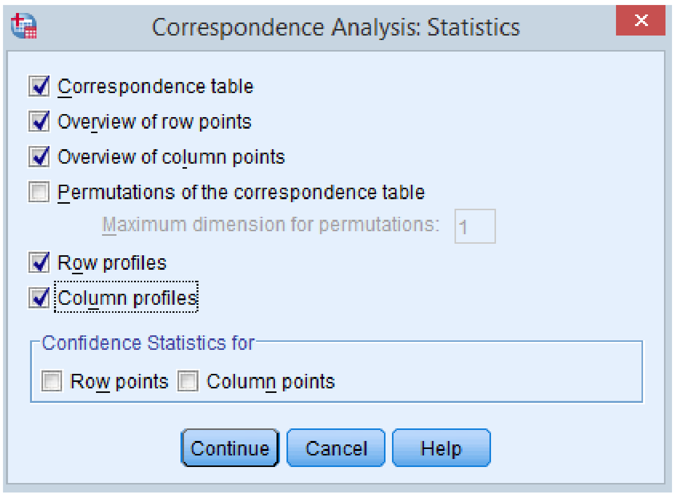
Figure 5.19: CA: Step 5

Step 6

Under the **Statistics** tab, select Row profiles and Column profiles (see Figure 5.20).

Step 7

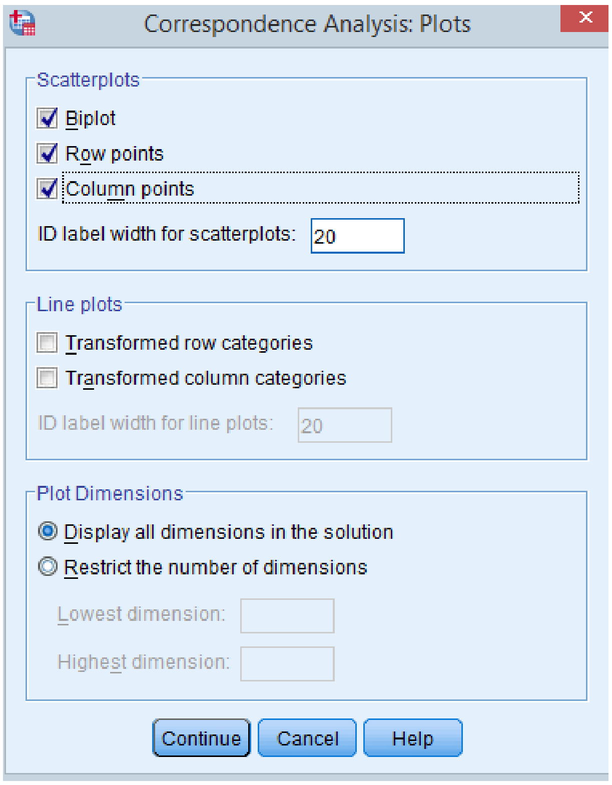
Under the **Plots** tab, select Row points and Column points (see Figure 5.21).



- [✓] Correspondence table
- [✓] Overview of row points
- [✓] Overview of column points
- [ ] Permutations of the correspondence table
  - Maximum dimension for permutations: 1
- [✓] Row profiles
- [✓] Column profiles

Confidence Statistics for
- [ ] Row points [ ] Column points

Figure 5.20: CA: Step 6



Scatterplots
- [✓] Biplot
- [✓] Row points
- [✓] Column points
- ID label width for scatterplots: 20

Line plots
- [ ] Transformed row categories
- [ ] Transformed column categories
- ID label width for line plots: 20

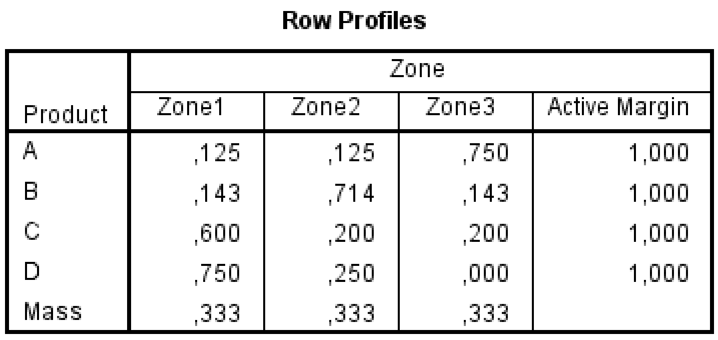
Plot Dimensions
- [✓ Display all dimensions in the solution]
- Restrict the number of dimensions
  - Lowest dimension:
  - Highest dimension:

Figure 5.21: CA: Step 7

Analysis of Results

Row Profiles (Table 5.5)

The rows correspond to different products. We can observe that Product A is present in Zone 1 with a rate of 12.5%, in Zone 2 with the same rate, and at 75% in Zone 3. This distribution explains the availability of Product A in the last zone.
Product B is mainly accessible in Zone 2.
Regarding Product C, it is available to the extent of 60% in Zone 1, while Product D has an availability rate of 75%.



Row Profiles

Product	Zone			
	Zone1	Zone2	Zone3	Active Margin
A	,125	,125	,750	1,000
B	,143	,714	,143	1,000
C	,600	,200	,200	1,000
D	,750	,250	,000	1,000
Mass	,333	,333	,333	

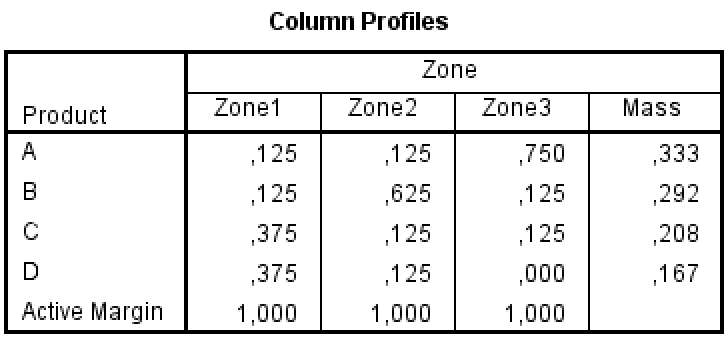
Table 5.5: Row Profiles

Column Profiles (Table 5.6)

In Zone 1, Products A and B show similar availability, both at a rate of 12.5%. Similarly, Products C and D have the same availability, each at 37.5%.
For Zone 2, the most widely available product is B, and the other three products are also present.
In Zone 3, Products A, B, and C are present. Among them, Product A is the most abundant, with an availability rate of 75%.

Summary Table (Table 5.7)

The initial dimension (Axis 1) accounts for 61.8% of the variability in the entire dataset, while the second dimension (Axis 2) explains 38.2% of this variability.
When considering both axes cumulatively, their contribution reaches 100% of the variation.

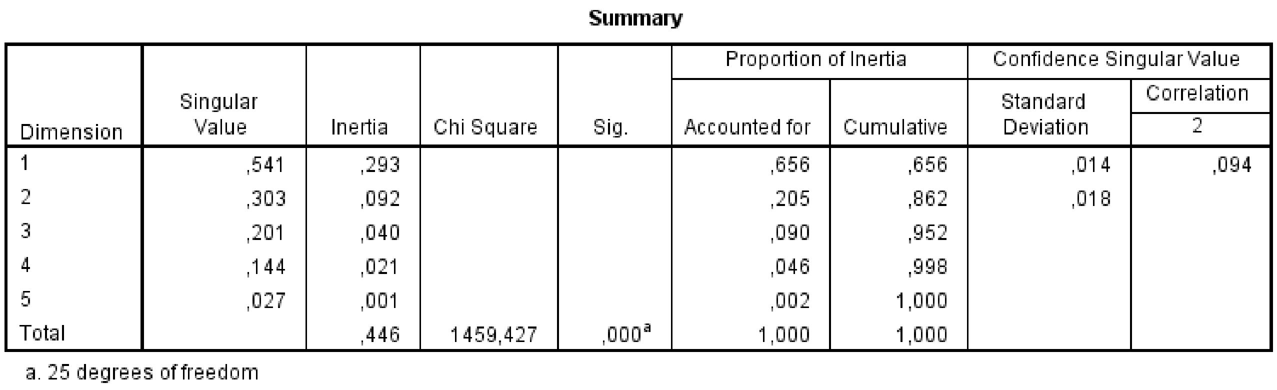


Column Profiles

Product	Zone			
	Zone1	Zone2	Zone3	Mass
A	,125	,125	,750	,333
B	,125	,625	,125	,292
C	,375	,125	,125	,208
D	,375	,125	,000	,167
Active Margin	1,000	1,000	1,000	

Table 5.6: Column Profiles

The Chi-squared test yields a result of 79.607 with a significance level (sig) less than 5%. This indicates a correlation between geographical zones and products, demonstrating a dependency between them.



Summary

Dimension	Singular Value	Inertia	Chi Square	Sig.	Proportion of Inertia		Confidence Singular Value	
					Accounted for	Cumulative	Standard Deviation	Correlation
								2
1	,541	,293			,656	,656	,014	,094
2	,303	,092			,205	,862	,018	
3	,201	,040			,090	,952		
4	,144	,021			,046	,998		
5	,027	,001			,002	1,000		
Total		,446	1459,427	,000[a]	1,000	1,000		

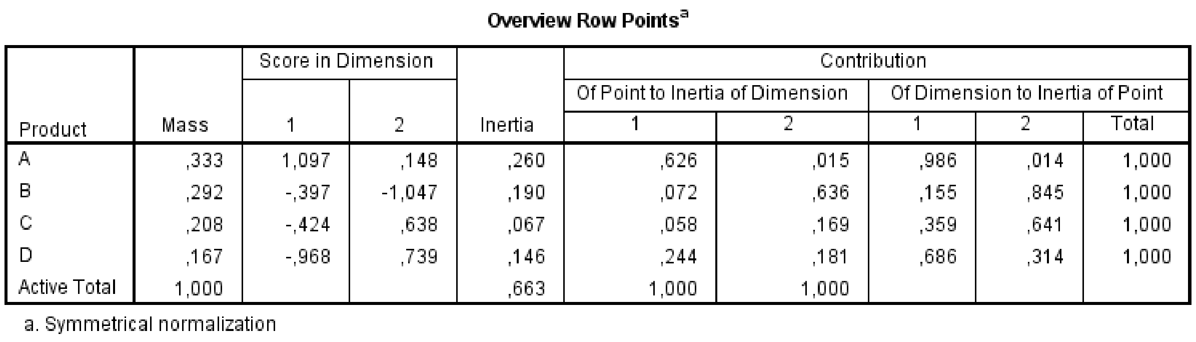
a. 25 degrees of freedom

Table 5.7: Summary Table

Presentation of Row Points (Table 5.8)

62.6% of the variation explained by Axis 1 relates to Product A, 7.2% for Product B, 5.8% for Product C, and 24.4% for Product D.

Axis 1 also explains 98.6% of the variation for Product A, while Axis 2 explains only 1.4%.



Overview Row Points[a]

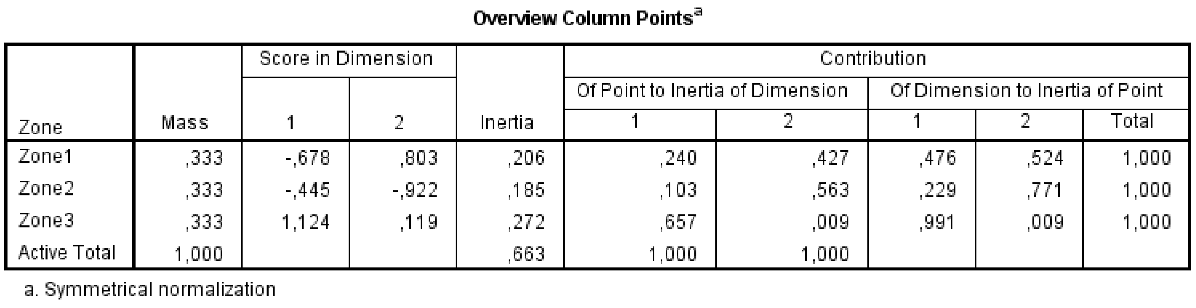
		Score in Dimension			Contribution				
					Of Point to Inertia of Dimension		Of Dimension to Inertia of Point		
Product	Mass	1	2	Inertia	1	2	1	2	Total
A	,333	1,097	,148	,260	,626	,015	,986	,014	1,000
B	,292	-,397	-1,047	,190	,072	,636	,155	,845	1,000
C	,208	-,424	,638	,067	,058	,169	,359	,641	1,000
D	,167	-,968	,739	,146	,244	,181	,686	,314	1,000
Active Total	1,000			,663	1,000	1,000			

a. Symmetrical normalization

Table 5.8: Presentation of Row Points

Presentation of Column Points (Table 5.9)
We also have 24% of the variation explained by Axis 1 concerning Zone 1, 10.3% for Zone 2, and 65.7% for Zone 3.
Axis 1 also explains 99.1% of the variation for Zone 3, while Axis 2 explains only 0.9%.

From the two tables, we can conclude that there is a correlation between Product A and Zone 3. Additionally, we can deduce a relationship between Zone 1 with Products C and D, as well as between Zone 2 and Product B.



Overview Column Points[a]

		Score in Dimension			Contribution				
					Of Point to Inertia of Dimension		Of Dimension to Inertia of Point		
Zone	Mass	1	2	Inertia	1	2	1	2	Total
Zone1	,333	-,678	,803	,206	,240	,427	,476	,524	1,000
Zone2	,333	-,445	-,922	,185	,103	,563	,229	,771	1,000
Zone3	,333	1,124	,119	,272	,657	,009	,991	,009	1,000
Active Total	1,000			,663	1,000	1,000			

a. Symmetrical normalization

Table 5.9: Presentation of Column Points

Figure 5.22 shows the relationships between zones and products.

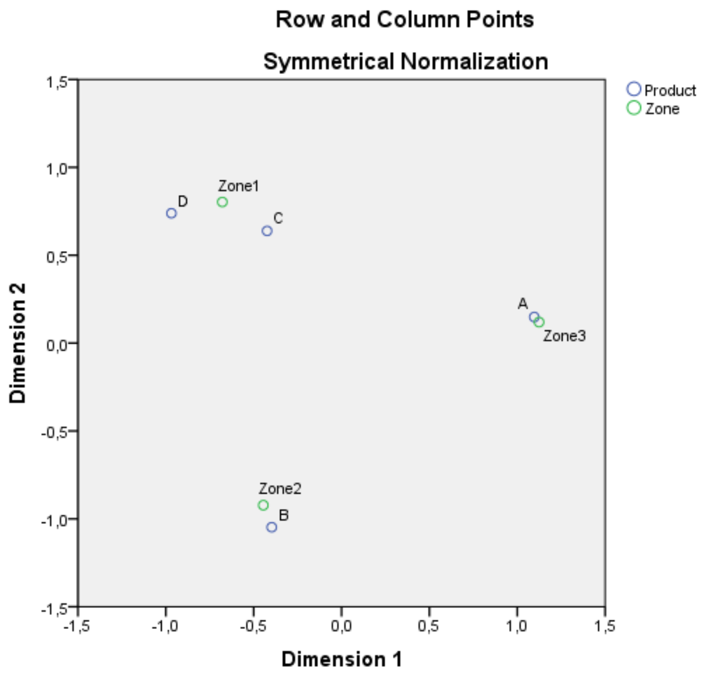


Figure 5.22: Row and Column Points

5.3 Multiple Correspondence Analysis (MCA)

Multiple Correspondence Analysis (MCA) is an advanced statistical technique that extends the capabilities of Correspondence Analysis (CA). It is specifically designed to analyze the complex patterns of relationships that exist among several categorical dependent variables. MCA can also be used for data reduction and dimensionality reduction, which can be helpful in simplifying complex data sets. While CA is suitable for analyzing two-way contingency tables, MCA can analyze multi-dimensional contingency tables involving three or more categorical variables.
Theoretical explanations can be found in the lecture handout.

5.3.1 Performing Multiple Correspondence Analysis in SPSS

SPSS is commonly used for Multiple correspondence analysis (MCA) due to its versatility in handling various statistical analyses, including those related to categorical data.

Performing MCA in SPSS entails several steps:

1. Importing the dataset
2. Executing MCA
3. Review Results: After running the analysis, SPSS will provide output tables and charts summarizing the results. The tables show eigenvalues, factor loadings
4. Interpret Results: For CA, we focus on interpreting the correspondence plot, visually representing associations between categories of the two variables. For MCA, we interpret similar plots for each pair of variables and explore additional output tables to understand relationships between multiple categorical variables.
5. Result visualization

5.3.2 Objectives of Multiple Correspondence Analysis

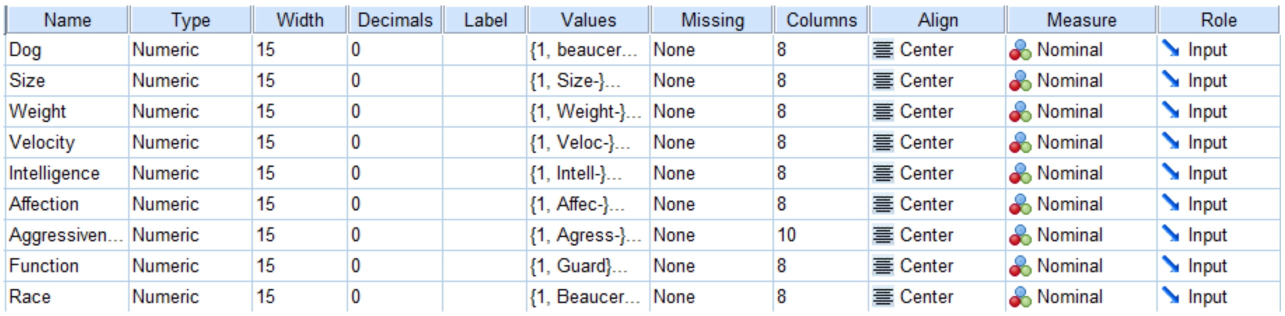
MCA serves several key objectives in data analysis. It offers a powerful means to explore complex datasets with categorical variables, effectively reducing dimensionality while preserving crucial information. It aids in hypothesis testing, variable selection, and segmentation tasks.

5.3.3 Practical Application of MCA with SPSS: An Illustrative Example

In the manufacturing of dog products, a holistic approach is employed, taking into account their functions, behavior, and morphology. This comprehensive consideration allows for a robust analysis and interpretation of the interplay between these varied parameters. The utilization of Multiple Correspondence Analysis (MCA) will enhance our ability to delve deeper into these relationships.
In the following example, there are 27 types of dogs categorized based on the variables of size, weight, velocity, intelligence, affection, aggressiveness, function, and race.
The variable 'Size' has three modalities: 'Size-', 'Size+', 'Size++'.
The variable 'Weight' has three modalities: 'Weight-', 'Weight+', 'Weight++'.
The variable 'Velocity' has three modalities: 'Veloc-', 'Veloc+', 'Veloc++'.
The variable 'Intelligence' has three modalities: 'Intell-', 'Intell+', 'Intell+'.
The variable 'Affection' has two modalities: 'Affec-', 'Affec+'.
The variable 'Aggressiveness' has two modalities: 'Agress-', 'Agress+'.
The variable 'Function' has three modalities: 'Guard', 'Hunt', 'Companion'.
The variable 'Race' has 27 modalities: 'Beauceron', 'Basset', 'Berger-Allemand', 'Boxer', 'Bull-Dog', 'Bull-Mastiff', 'Poodle', 'Chihuahua', 'Cocker', 'Collie', 'Dalmatian', 'Doberman', 'Dogue-Allemand', 'Brittany', 'Spaniel-French', 'Fox-hound', 'Fox-Terrier', 'Grand Bleu de Gascogne', 'Labrador', 'Greyhound', 'Mastiff', 'Pekingese', 'Pointer', 'St-Bernard', 'Setter', 'Dachshund', and 'Newfoundland.

Step 1

Define variable types in the **Variable View** window (see Figure 5.23).

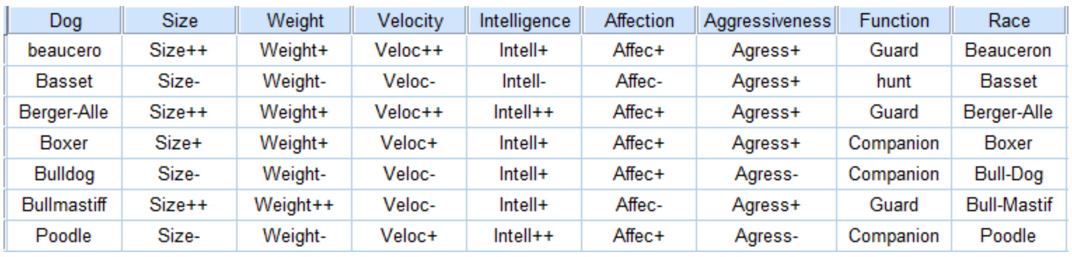


Name	Type	Width	Decimals	Label	Values	Missing	Columns	Align	Measure	Role
Dog	Numeric	15	0		{1, beaucer...	None	8	Center	Nominal	Input
Size	Numeric	15	0		{1, Size-}...	None	8	Center	Nominal	Input
Weight	Numeric	15	0		{1, Weight-}...	None	8	Center	Nominal	Input
Velocity	Numeric	15	0		{1, Veloc-}...	None	8	Center	Nominal	Input
Intelligence	Numeric	15	0		{1, Intell-}...	None	8	Center	Nominal	Input
Affection	Numeric	15	0		{1, Affec-}...	None	8	Center	Nominal	Input
Aggressiven...	Numeric	15	0		{1, Agress-}...	None	10	Center	Nominal	Input
Function	Numeric	15	0		{1, Guard}...	None	8	Center	Nominal	Input
Race	Numeric	15	0		{1, Beaucer...	None	8	Center	Nominal	Input

Figure 5.23: MCA: Step 1

Step 2

Enter the data into the **Data View** window (see Figure 5.24).



Dog	Size	Weight	Velocity	Intelligence	Affection	Aggressiveness	Function	Race
beaucero	Size++	Weight+	Veloc++	Intell+	Affec+	Agress+	Guard	Beauceron
Basset	Size-	Weight-	Veloc-	Intell-	Affec-	Agress+	hunt	Basset
Berger-Alle	Size++	Weight+	Veloc++	Intell++	Affec+	Agress+	Guard	Berger-Alle
Boxer	Size+	Weight+	Veloc+	Intell+	Affec+	Agress+	Companion	Boxer
Bulldog	Size-	Weight-	Veloc-	Intell+	Affec+	Agress-	Companion	Bull-Dog
Bullmastiff	Size++	Weight++	Veloc-	Intell+	Affec-	Agress+	Guard	Bull-Mastif
Poodle	Size-	Weight-	Veloc+	Intell++	Affec+	Agress-	Companion	Poodle

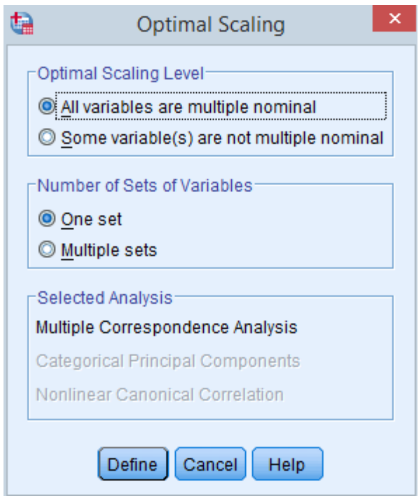
Figure 5.24: MCA: Step 2

Step 3

To perform a correspondence analysis, follow these steps from the menus: **Analyze -> Dimension Reduction -> Optimal Scaling**.

Step 4

In the **Optimal Scaling** window, select the option 'All variables are multiple nominal' and choose 'One set' (see Figure 5.25).


Optimal Scaling

Optimal Scaling Level

[✓ All variables are multiple nominal] Some variable(s) are not multiple nominal

Number of Sets of Variables

[✓ One set] Multiple sets

Selected Analysis

Multiple Correspondence Analysis

Categorical Principal Components

Nonlinear Canonical Correlation

Figure 5.25: MCA: Step 4

Step 5

In the **Multiple Correspondence Analysis** window, select the following variables for analysis: size, weight, velocity, intelligence, affection, and aggressiveness. For supplementary variables, choose 'function', and for the labeling variable, select 'race' (see Figure 5.26).

Step 6

In the **MCA: Objet Plots** window, select 'Object points' and 'Objects and Centroids'. In Biplot variables select the following variables : size, weight, velocity, intelligence, affection, aggressiveness and function. For the labeling variable, select 'race' (see Figure 5.27).

Step 7

In the **MCA: Variable Plots** window, select the following variables: size, weight, velocity, intelligence, affection, aggressiveness, and function for the 'Joint Category Plots' (see Figure 5.28).

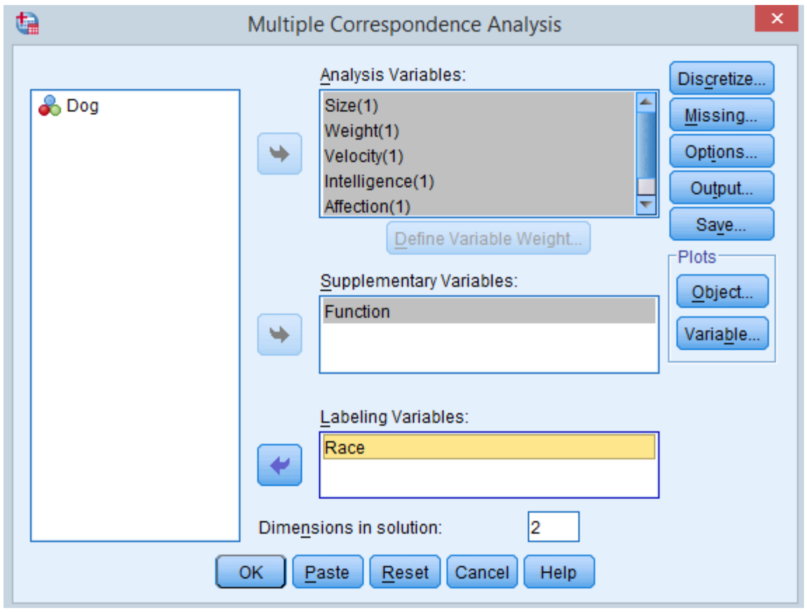


Figure 5.26: MCA: Step 5

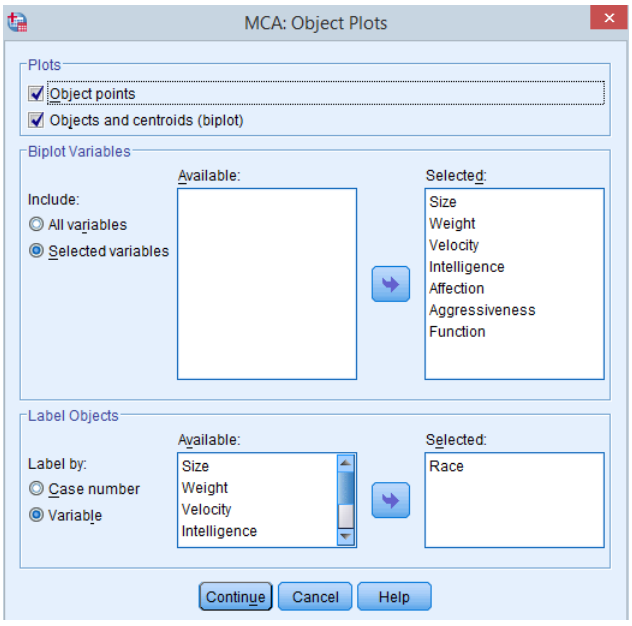
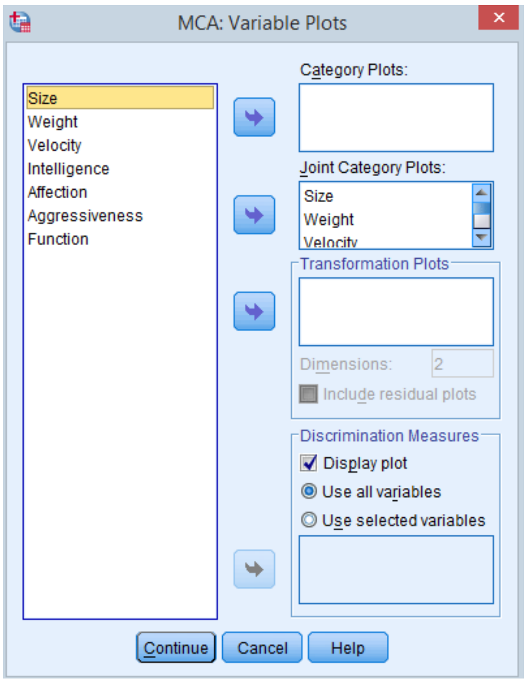


Figure 5.27: MCA: Step 6


MCA: Variable Plots

Size
Weight
Velocity
Intelligence
Affection
Aggressiveness
Function

Category Plots:

Joint Category Plots:
Size
Weight
Velocity

Transformation Plots
Dimensions: 2
Include residual plots

Discrimination Measures
[✓] Display plot
(●) Use all variables
( ) Use selected variables

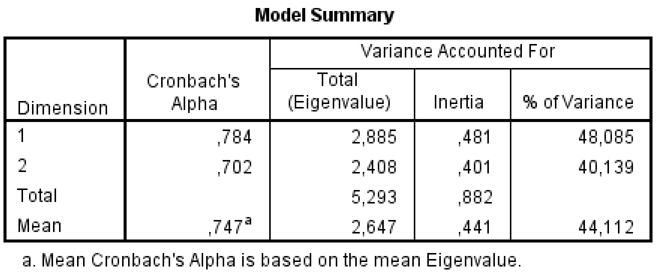
Figure 5.28: MCA: Step 7

Step 8

Finally, click OK. The "IBM SPSS Statistics" file will be generated.

Analysis of Results

Model Summary (Table 5.10) contains Total (Eigenvalue), Inertia, and % of Variance.
In the **Model Summary**, it is evident that we have chosen two dimensions, and these two dimensions account for 44.112% of the variance.



Model Summary

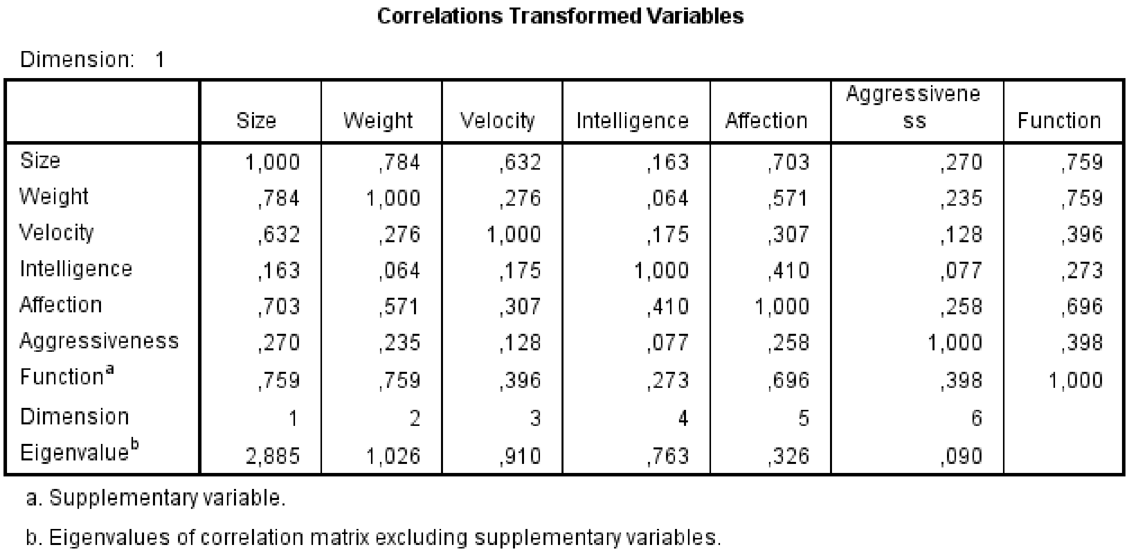
Dimension	Cronbach's Alpha	Variance Accounted For		
		Total (Eigenvalue)	Inertia	% of Variance
1	,784	2,885	,481	48,085
2	,702	2,408	,401	40,139
Total		5,293	,882	
Mean	,747[a]	2,647	,441	44,112

a. Mean Cronbach's Alpha is based on the mean Eigenvalue.

Table 5.10: Model Summary

Correlation Matrix (Table 5.11)

Displays the correlations between all variables. In this case, we have taken the variable 'function' as a supplementary (or demonstrative) variable.



Correlations Transformed Variables

Dimension: 1

	Size	Weight	Velocity	Intelligence	Affection	Aggressiveness	Function
Size	1,000	,784	,632	,163	,703	,270	,759
Weight	,784	1,000	,276	,064	,571	,235	,759
Velocity	,632	,276	1,000	,175	,307	,128	,396
Intelligence	,163	,064	,175	1,000	,410	,077	,273
Affection	,703	,571	,307	,410	1,000	,258	,696
Aggressiveness	,270	,235	,128	,077	,258	1,000	,398
Function[a]	,759	,759	,396	,273	,696	,398	1,000
Dimension	1	2	3	4	5	6	
Eigenvalue[b]	2,885	1,026	,910	,763	,326	,090	

a. Supplementary variable.

b. Eigenvalues of correlation matrix excluding supplementary variables.

Table 5.11: Correlations Transformed Variables

Discrimination Measures

Figure 5.29 display the discrimination measures of all variables based on the two dimensions. As we can see, Velocity, Weight, and Size have the highest discrimination measures.

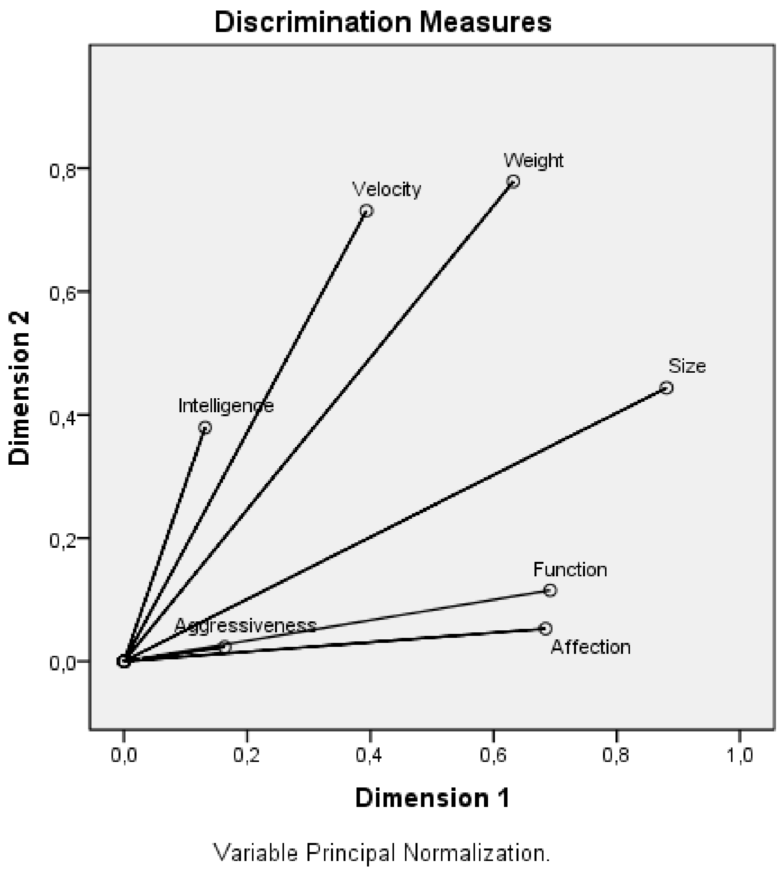


Figure 5.29: Discrimination Measures

Category Points

Figure 5.30 displays all the modalities of the various variables.
When examining the results in relation to the function, we observe that companion dogs tend to be less aggressive, more affectionate, and more intelligent. Conversely, guard dogs exhibit lower levels of affection and higher aggressiveness. Hunt dogs, on the other hand, display greater velocity, weight, and size.

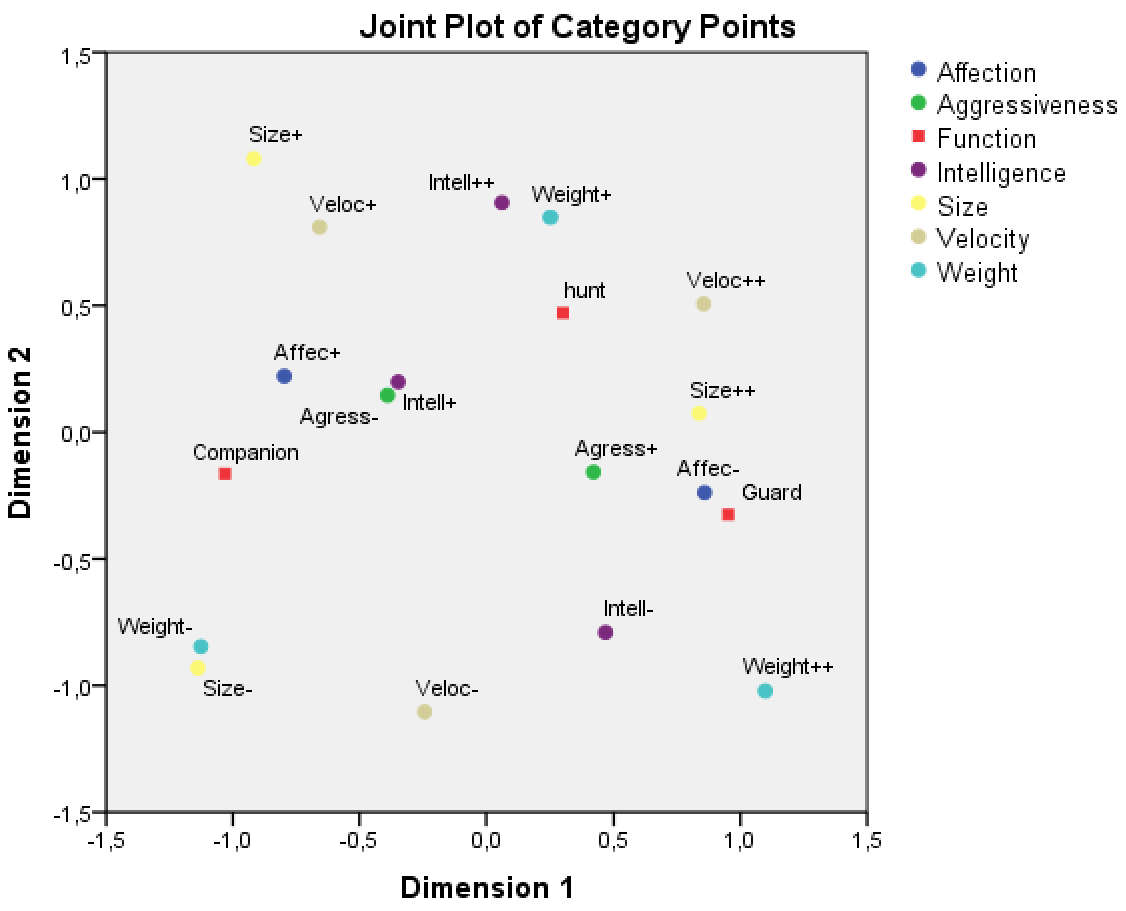


Figure 5.30: Category Points

Object Points Labeled by Race

In Figure 5.31, we can observe all the modalities of the 'Race' variable, which we have employed as a label. This figure also allows us to categorize the dogs into three distinct clusters. The first cluster includes breeds like Brittany, Dalmatian, Boxer, Labrador, Berger Allemand, Collie, Doberman, Pointer, Foxhound, Greyhound, Grand Bleu de Gascogne, Beauceron, Setter, and Spaniel-French. The second cluster contains breeds such as Dogue Allemand, St. Bernard, Bullmastiff, Newfoundland, and Mastiff. The third cluster consists of Poodle, Cocker, Fox-Terrier, Dachshund, Bull-Dog, Chihuahua, Pekingese, and Basset.

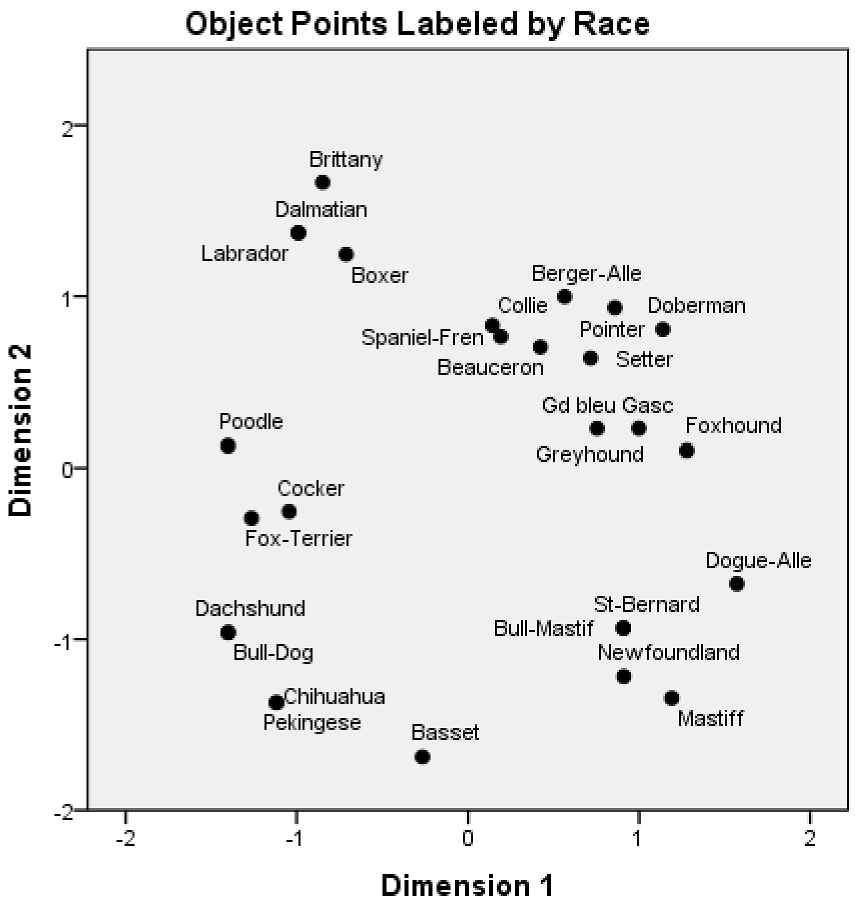


Figure 5.31: Object Points Labeled by Race

Biplot

In the final Figure 5.32, we observe the plot of all variables. The three clusters of dog breeds identified in Figure 5.31 all serve distinct functions. The first cluster represents hunting dogs, the second cluster comprises guard dogs, and the last cluster consists of companion dogs. Each function is associated with specific characteristics, as illustrated in Figure 5.30. For instance, guard dogs tend to exhibit higher levels of aggressiveness and size but lower levels of affection. The Biplot offers us a comprehensive view of the entire dataset.

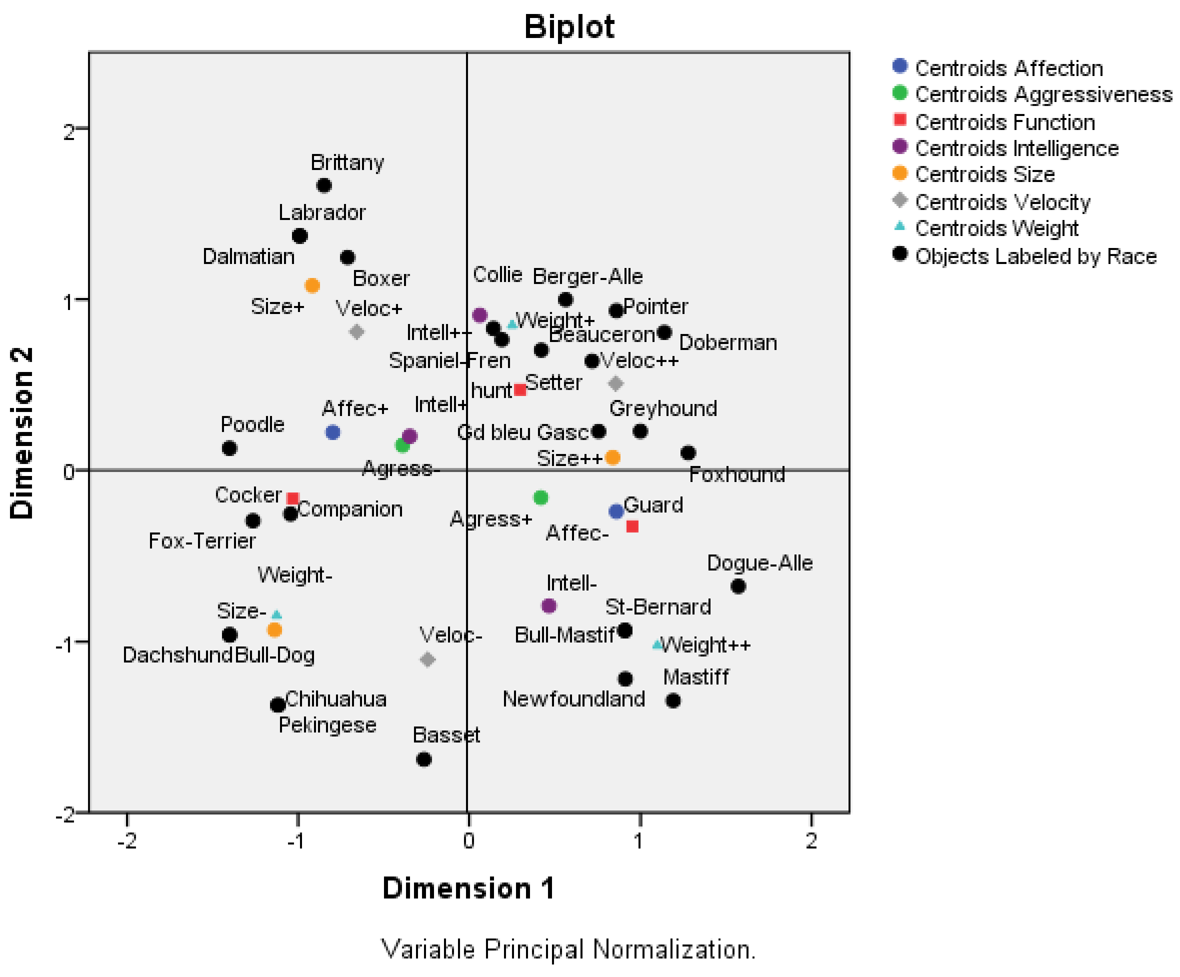


Figure 5.32: Biplot

Note

You can access the files containing the tables relevant to this Workshop through the following link: Workshop-PCA-MCA

Conclusion

In the realm of multidimensional analysis approaches, Principal Component Analysis (PCA) stands out as one of the most renowned unsupervised dimensionality reduction methods. It works by transforming a large set of variables into a smaller one that retains most of the information from the original set. On the other hand, there is Multiple Correspondence Analysis (MCA). It is essentially a generalized version of principal component analysis, specifically designed for qualitative data analysis. In this workshop, we present the use of these two methods in multidimensional analysis using SPSS. We provide detailed steps for both methods using an illustrative example, followed by an interpretation of the obtained results.

GENERAL CONCLUSION

The purpose of this manual is to lead students through a step-by-step learning process of data analysis using Python and the SPSS software.

Each practical workshop has been meticulously designed to provide engineering students with both a deep theoretical understanding and practical skills.

This comprehensive approach aims to empower engineering students with a solid grasp of the fundamental concepts required for effective data analysis in diverse settings.

By blending theory with hands-on practice, this manual aims to prepare engineering students to leverage these data analysis skills for making well-informed decisions and tackling complex challenges within their academic and professional domains.

ANNEXE

Python provides a range of built-in methods designed for performing operations on strings, lists, tuples, and dictionaries.

1 String Methods

Méthode	**Description**
capitalize()	Converts the first character to upper case
casefold()	Converts string into lower case
center()	Returns a centered string
count()	Returns the number of times a specified value occurs in a string
encode()	Returns an encoded version of the string
endswith()	Returns true if the string ends with the specified value
expandtabs()	Sets the tab size of the string
find()	Searches the string for a specified value and returns the position of where it was found

format()	Formats specified values in a string
format_map()	Formats specified values in a string
index()	Searches the string for a specified value and returns the position of where it was found
isalnum()	Returns True if all characters in the string are alphanumeric
isalpha()	Returns True if all characters in the string are in the alphabet
isascii()	Returns True if all characters in the string are ascii characters
isdecimal()	Returns True if all characters in the string are decimals
isdigit()	Returns True if all characters in the string are digits
isidentifier()	Returns True if the string is an identifier
islower()	Returns True if all characters in the string are lower case
isnumeric()	Returns True if all characters in the string are numeric
isprintable()	Returns True if all characters in the string are printable
isspace()	Returns True if all characters in the string are whitespaces
istitle()	Returns True if the string follows the rules of a title
isupper()	Returns True if all characters in the string are upper case
join()	Joins the elements of an iterable to the end of the string
ljust()	Returns a left justified version of the string
lower()	Converts a string into lower case
lstrip()	Returns a left trim version of the string
maketrans()	Returns a translation table to be used in translations
partition()	Returns a tuple where the string is parted into three parts

replace()	Returns a string where a specified value is replaced with a specified value
rfind()	Searches the string for a specified value and returns the last position of where it was found
rindex()	Searches the string for a specified value and returns the last position of where it was found
rjust()	Returns a right justified version of the string
rpartition()	Returns a tuple where the string is parted into three parts
rsplit()	Splits the string at the specified separator, and returns a list
rstrip()	Returns a right trim version of the string
split()	Splits the string at the specified separator, and returns a list
splitlines()	Splits the string at line breaks and returns a list
startswith()	Returns true if the string starts with the specified value
strip()	Returns a trimmed version of the string
swapcase()	Swaps cases, lower case becomes upper case and vice versa
title()	Converts the first character of each word to upper case
translate()	Returns a translated string
upper()	Converts a string into upper case
zfill()	Fills the string with a specified number of 0 values at the beginning

Table 1: String Methods

2 List Methods

Method	Description
append()	Adds an element to the end of the list
clear()	Removes all elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Adds the elements of a list (or any iterable) to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the element with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

Table 2: List Methods

3 Tuple Methods

Method	Description
count()	Returns the number of times a specified value appears in a tuple
index()	Searches for a specified value in the tuple and returns it position

Table 3: Tuple Methods

4 Dictionary Methods

Method	Description
clear()	Removes all elements from the dictionary
copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and values
get()	Returns the value of the specified key
items()	Returns a list containing a tuple for each key-value pair
keys()	Returns a list containing the keys of the dictionary
pop()	Removes the item with the specified key
popitem()	Removes the last inserted key-value pair
setdefault()	Returns the value of the specified key. If the key does not exist, inserts the key with the specified value
update()	Updates the dictionary with the specified key-value pairs
values()	Returns a list of all values in the dictionary

Table 4: Dictionary Methods

BIBLIOGRAPHY

[1] Hervé Abdi and Michel Béra. Correspondence analysis., 2014.

[2] R Artusi, P Verderio, and EJTIjobm Marubini. Bravais-pearson and spearman correlation coefficients: meaning, test of hypothesis and confidence interval. *The International journal of biological markers*, 17(2):148–151, 2002.

[3] Sorana-Daniela Bolboaca and Lorentz Jäntschi. Pearson versus spearman, kendall's tau correlation analysis on structure-activity relationships of biologic active compounds. *Leonardo Journal of Sciences*, 5(9):179–200, 2006.

[4] Giovanni Di Franco. Multiple correspondence analysis: one only or several techniques? *Quality & Quantity*, 50(3):1299–1315, 2016.

[5] Lauren Washington George McIntire, Brendan Martin. Python pandas tutorial: A complete introduction for beginners. `<https://www.learndatasci.com/tutorials/python-pandas-tutorial-complete-introduction-for-beginners/>`, 2023. [Online; accessed August 2023].

[6] Michael Greenacre, Patrick JF Groenen, Trevor Hastie, Alfonso Iodice d'Enza, Angelos Markos, and Elena Tuzhilina. Principal component analysis. *Nature Reviews Methods Primers*, 2(1):100, 2022.

[7] Antoine Massé. Analyses en composantes principales et une de ses alternatives, le tsne. <https://sites.google.com/view/aide-python/statistiques/machine-learning-en-python/analyses-en-composantes-principales>, 2023. [Online; accessed August 2023].

[8] Leann Myers and Maria J Sirois. Spearman correlation coefficients, differences between. *Encyclopedia of statistical sciences*, 12, 2004.

[9] NumPy.org. Numpy reference. <https://numpy.org/doc/stable/reference/>, 2023. [Online; accessed August 2023].

[10] Site officiel de Python. Python for programmers. <https://www.python.org>, 2023. [Online; accessed August 2023].

[11] Pydata.org. Pandas documentation. <http://pandas.pydata.org/docs/>, 2023. [Online; accessed August 2023].

[12] Alaa Tharwat. Principal component analysis-a tutorial. *International Journal of Applied Pattern Recognition*, 3(3):197–240, 2016.

[13] John W Tukey. The future of data analysis. *The annals of mathematical statistics*, 33(1):1–67, 1962.

[14] TutorialsPoint. Python tutorial. <https://www.tutorialspoint.com/python/index.htm>, 2023. [Online; accessed August 2023].

[15] w3schools. Python tutorial. <https://www.w3schools.com/python/>, 2023. [Online; accessed August 2023].

[16] webin R. Analyse des correspondances multiples (acm). <https://larmarange.github.io/analyse-R/analyse-des-correspondances-multiples.html>, 2023. [Online; accessed August 2023].

[17] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987.