UNIVERSITY OF ABOMEY-CALAVI (UAC)

****

DOCTORAL SCHOOL OF AGRONOMIC AND WATER

****

LABORATOIRE DE BIOMATHEMATIQUES ET D'ESTIMATIONS FORESTIERES

****

MASTER1-BIOSTATISTICS

Project Title:

Development of Nonparametric Kruskal-Wallis Test in MATLAB, R, and Python

Prepared by:

Mannondé D. GBAGUIDI

Lecturer:

Romain L. Glèlè Kakaï

Contents

List of Tables

1 Introduction

The information age has generated a massive amount of data in various domains such as manufacturing, medicine, security, and biology. This valuable data holds crucial information to inform decisions, manage risks, and optimize processes. Statistical analysis and comparison of these data, often organized in large datasets, are essential for extracting useful knowledge.

A major challenge lies in the fact that these datasets can contain multiple distinct groups. Understanding the differences and similarities between these groups is a fundamental step in statistical analysis. However, traditional parametric methods such as analysis of variance (ANOVA) rely on strict assumptions about data distribution, such as normality and homogeneity of variances (D.S.J.C Gbemavo, 2021). When these assumptions are not met, the results of parametric tests can be biased and unreliable. The Kruskal-Wallis test, a non-parametric extension of ANOVA, provides a robust alternative for comparing medians of multiple groups, even when data does not meet the assumptions of parametric tests (Kruskal and Wallis 1952). This non-parametric statistical test makes no assumptions about data distribution, making it applicable to a wide range of situations (D.S.J.C Gbemavo, 2021).

This project aims to develop and implement the Kruskal-Wallis test in two popular programming environments: MATLAB and Python. The main objective is to provide a comprehensive understanding of this non-parametric statistical test and demonstrate its practical implementation in these two programming languages. This involves gaining an in-depth understanding of the Kruskal-Wallis test, including its assumptions, test statistic, interpretation, and applications. Importantly, it involves developing comprehensive functions to perform the Kruskal-Wallis test in MATLAB and Python. These functions should include data handling, rank calculation, calculation of the H statistic, determination of degrees of freedom, and calculation of the p-value.

2 Theoretical framework: Kruskal-Wallis test

2.1 Description

The Kruskal-Wallis test, developed by William Kruskal and Wilson Allen Wallis, is a nonparametric statistical method used to compare distributions from multiple independent samples. It is an alternative to the one-factor ANOVA test, which assumes a normal distribution of data (Hollander, Wolfe, et Chicken 2013).

Objective of the test:

The Kruskal-Wallis test aims to determine whether the samples come from the same distribution. In other words, it allows us to test the null hypothesis that the medians of the samples are equal, against the alternative hypothesis that at least one median differs significantly from the others.

H_0**:** Null Hypothesis

- This is the default assumption based on knowledge or logic.
- We fail to reject or accept H_0 if the significance level (α) is greater than 0.05.

H_1**:** Alternative Hypothesis

- This represents the opposite of the default assumption stated in H_0.

Globally, we can express our hypotheses as:

$$H_0 : \text{Null Hypothesis} \quad \text{vs.} \quad H_1 : \text{Alternative Hypothesis}$$

Applicability:

The Kruskal-Wallis test is particularly useful when the data do not follow a normal distribution or when the variances of the samples are not homogeneous. It can be used to compare two or more independent samples, of similar or different sizes.

Interpretation of results:

A significant Kruskal-Wallis test (p-value < 0.05) indicates that at least one sample statistically dominates another sample. However, the test does not accurately identify dominant groups or pairs of groups with significant differences.

Post-hoc testing: To identify specific groups that have significant differences, it is necessary to use post-hoc tests, such as the Dunn test, the Mann-Whitney pairwise tests without Bonferroni correction, or the Conover-Iman test.

Advantages of the Kruskal-Wallis test:

- Does not assume normal data distribution
- Robust to variance homogeneity violations

- Applicable to different size samples
- Simple to interpret

Limitations of the Kruskal-Wallis test:

- Does not provide information on the nature of the differences between groups
- Requires post-hoc testing to identify specific groups that differ

In summary, the Kruskal-Wallis test is a valuable statistical tool for comparing distributions from multiple independent samples, especially when assumptions of normal distribution and homogeneity of variances are not met.

2.2 The H Test: Statistic of Kruskal-Wallis

The test of Kriskal-Wallis used the rank of all the observation, and the sum of the ranks obtained for each sample (Kruskal and Wallis 1952). The test statistic to be computed if there are no ties (that is, if no two observations are equal) is

$$H = \frac{12}{N(N+1)} \sum_{i=1}^{p} \frac{R_i^2}{n_i} - 3(N+1) \tag{2.1}$$

Where
$p =$ the number of samples,
$n_i =$ the number of observations in the ith sample,
$N = \sum n_i$, the number of observations in all samples combined,
$R_i =$ the sum of the ranks in the ith sample.
Large values of H lead to rejection of the null hypothesis.
If the samples come from identical continuous populations and the n_i are not too small, H is distributed as $\chi^2(p-1)$, because on the null hypothesis the ranks follow the uniform distribution. If there are ties, each observation is given the mean of the ranks for which it is tied. H as computed from (2.1) is then divided by

$$1 - \frac{\sum T}{N^3 - N} \tag{2.2}$$

Where the summation is over all groups of ties and $T = (t-1)t(t+1) = t^3 - t$ for each group of ties, t being the number of tied observations in the group.

Thus, (2.1) divided by (2.2) gives a general expression which holds whether or not there are ties, assuming that such ties as occur are give mean ranks:

$$H = \frac{\frac{12}{N(N+1)} \sum_{i=1}^{p} \frac{R_i^2}{n_i} - 3(N+1)}{1 - \sum T/(N^3 - N)} \tag{2.3}$$

In many situations the difference between(2.1) and (2.3) is negligible. H for large samples is still distributed as $\chi^2(p-1)$ when ties are handled by mean ranks; but the tables for small samples, while still useful, are no longer exact (Kruskal and Wallis 1952).
For understanding the nature of H, a better formulation of (2.1) is

$$H = \frac{N-1}{N} \sum_{i=1}^{p} \frac{n_i[\bar{R}_i - \frac{1}{2}(N+1)]^2}{(N^2-1)/12} \tag{2.4}$$

where$\bar{R}_i$ is the mean of the n_i ranks in the ith sample.

The mean of all ranks is give by $\bar{R} = \frac{1}{N}\sum_{i}^{N} r_i = \frac{N+1}{2}$ and $\sigma^2 = \frac{N^2-1}{12}$ the variance.

Therefore, we have:

$$\begin{aligned}
H &= \frac{1}{(N^2-1)/12}\left(\sum_{i=1}^{p} n_i[\bar{R}_i - \bar{R}]^2 - \frac{1}{N}\sum_{i=1}^{p} n_i[\bar{R}_i - \bar{R}]^2\right) \\
&= \frac{1}{(N^2-1)/12}\left((1-\frac{1}{N})\sum_{i=1}^{p} n_i[\bar{R}_i - \bar{R}]^2\right) \\
&= \frac{1}{(N^2-1)/12}\left(\frac{N-1}{N}\sum_{i=1}^{p} n_i[\bar{R}_i - \bar{R}]^2\right) \qquad so \\
&= \frac{N-1}{N}\sum_{i=1}^{p}\frac{n_i[\bar{R}_i - \frac{1}{2}(N+1)]^2}{(N^2-1)/12} \\
H &= \frac{12}{N(N+1)}\sum_{i=1}^{p}\frac{R_i^2}{n_i} - 3(N+1)
\end{aligned}$$

In summary, the H statistic of Kruskal-Wallis is given by the weighted difference between the mean ranks of the overall centred reduced ranks, which follows the chi-square distribution and in general give by Equation (2.1).

For the rest of the project, it will be a matter of developing the Kruskal-Wallis test with Matlab and Python. To do this, we will use the formula for the statistic of H obtained in Equation (2.1)and (2.4) to have the best value of the H statistic.[1]

[1]In many situations the difference between (2.1) and (2.4) is negligible but this is not for all situations.

3 Design and implementation of the Kruskal-Wallis test

3.1 Steps and explanation of the algorithm

$$H = \frac{12}{N(N+1)} \sum_{i=1}^{p} \frac{R_i^2}{n_i} - 3(N+1)$$

We will make an example that can illustrate each of the steps in calculating the value of the H-statistic. For this example, we will use data from Snedecor [46], Table 10.12.

Table 3.1: BIRTH WEIGHTS (lbs.) OF EIGHT LITTERS OF PIGS.
source: Snedecor [46], Table 10.12

Litter1	Litter2	Litter3	Litter4	Litter5	Litter6	Litter7	Litter8
2.0	3.5	3.3	3.2	2.6	3.1	2.6	2.5
2.8	2.8	3.6	3.3	2.6	2.9	2.2	2.4
3.3	3.2	2.6	3.2	2.9	3.1	2.2	3.0
3.2	3.5	3.1	2.9	2.0	2.5	2.5	1.4
4.4	2.3	3.2	3.3	2.0		1.2	
3.6	2.4	3.3	2.5	2.1		1.2	
1.9	2.0	2.9	2.6				
3.3	1.6	3.4	2.8				
2.8		3.2					
1.1		3.2					

The first step is to determine the rank of all the data together. For this step, the rank is the average rank. It is not necessary to arrange the data in a given order before determining the ranks of the data. After this step, the next step (step two) is to group the rank of the data by group and sum the ranks from each group (Compute the sum of ranks of the observations for each sample, let R_1, R_2, R_3,...).

Table 3.2: BIRTH WEIGHTS (lbs.) OF EIGHT LITTERS OF PIGS (Ranks).
source: Snedecor [46], Table 10.12

	Litter1	Litter2	Litter3	Litter4	Litter5	Litter6	Litter7	Litter8
	Ranks	Ranks	Ranks	Ranks	Ranks	Ranks	Ranks	Ranks
	8.5	52.5	47.5	41.	23.	36.	23.	18.5
	27.5	27.5	54.5	47.5	23.	31.5	12.5	15.5
	47.5	41.	23.	41.	31.5	36.	12.5	34.
	41.	52.5	36.	31.5	8.5	18.5	18.5	4.
	56.	14.	41.	47.5	8.5		2.5	
	54.5	15.5	47.5	18.5	11.		2.5	
	6.	8.5	31.5	23.				
	47.5	5.	51.	27.5				
	27.5		41.					
	1.		41.					
n_i	10	8	10	8	6	4	6	4
R_i	317.0	216.5	414.0	277.5	105.5	122.0	71.5	72.0
$\frac{R_i^2}{n_i}$	10048.9	5859.031	17139.6	9625.781	1855.042	3721.0	852.042	1296.0

The next step is to calculate the H-statistic. To do this, we'll first calculate the square of the sum of the rows per group divided by the size of each group.
$N = \sum n = 10 + 8 + 10 + 8 + 6 + 4 + 6 + 4 = 56$ and
$\sum \frac{R_i^2}{n_i} = 10048.9 + 5859.031 + 17139.6 + 9625.782 + 1855.042 + 3721.0 + 852.042 + 1296.0$
$\sum \frac{R_i^2}{n_i} = 50397.396$
So we have $H = \frac{12}{56(56+1)} \times 50397.396 - 3 \times (56 + 1) = 18.464$
In the case of tied values, H is divided by (2.2). In this case we have:

t :	2	4	2	2	4	5	4	4	3	7	6	2	2
T :	6	60	6	6	60	120	60	60	24	336	210	6	6

$\sum T = 960$ so $1 - \frac{\sum T}{N(N+1)} = 0.9945$ and we have $H = 18.566$

So to compute H we have four step.

Step1. Data from all group samples are combined and observations are assigned rank R. The data is sommetimes put in ascending order.

Step2. Determine the rank of the observations of each sample.

Step3. Compute the sum of ranks of the observations for each sample, let R1, R2, R3, ...

Step4. Compute the Kruskal-Wallis statistic, H.

$$H = \frac{12}{N(N+1)} \sum_{i=1}^{p} \frac{R_i^2}{n_i} - 3(N+1)$$

k = number of samples; nj=size of sample j; N = n1 + n2 + n3 + ... In the case of tied values, H is divided by (2.2)(D.S.J.C Gbemavo, 2021).

3.2 Analysis of Environments: Python and MATLAB for Kruskal-Wallis Test Development

We will conduct an analysis comparing two programming environments used for the development of the Kruskal-Wallis test: Python and MATLAB. We will evaluate each tool based on its ease of use, flexibility, performance, and user community.

Python is a general-purpose programming language known for its simple and readable syntax. It offers a wide range of statistical libraries, including scipy.stats which implements the Kruskal-Wallis test. Many tutorials and examples are available online to help get started with Python. Additionally, Python is a general-purpose language that can be used for various tasks beyond the Kruskal-Wallis test, such as web development, machine learning, and data science. Python is an interpreted programming language, which means it is generally less performant than compiled languages like MATLAB. However, the performance of the Kruskal-Wallis test in Python is typically sufficient for most applications. It has a large and active user community, with plenty of resources available online. This makes it easy to find help and guidance if you encounter issues.

MATLAB is a programming language specialized in numerical computation and data analysis. It offers built-in functions for the Kruskal-Wallis test, as well as powerful data visualization and management tools. However, MATLAB syntax may be less intuitive for beginners compared to Python. MATLAB is a compiled programming language, which generally provides better performance than Python. This can be important for applications requiring processing of large datasets. MATLAB has a smaller user community compared to Python, but it is still active and supportive. Numerous resources are also available online for MATLAB.

Therefore, the choice between Python and MATLAB for Kruskal-Wallis test development depends on specific needs and preferences.

Python: If you are looking for an easy-to-use, flexible language with a large user community, Python is a good choice.

MATLAB: If you require optimal performance, deal with very large datasets, or already work with MATLAB for other data analysis tasks, MATLAB is a good choice.

Ultimately, the best way to choose is to try both tools and see which one suits you best. However, for this project, focusing solely on developing the test in both environments, our goal will be to expose and explore the potential of each environment for this task with the source codes.

3.3 Implementation in Python and Matlab: Python Script

The input dataset in the codes are presented like the table 3.1 **A-Importation of libraries:**

```
import pandas as pd
import numpy as np
from scipy.stats import rankdata, chi2
from collections import Counter
import math
```

This code importing necessary libraries:
pandas: for data manipulation
numpy: for mathematical operations
rankdata: for assigning ranks to data
chi2: for conducting the chi-squared test
Counter: for counting occurrences of elements
math: for mathematical functions and handling NaN values

B- Function of correction: *rank_correction*
This function is very important to have a very accurate value of the H statistic.

```
def rank_correction(rm):
  sv = sorted(rm)
  ct = Counter(sv)
  n = len(sv)
  corr = 0.0
```

This code defines a function named *rank_correction* that calculates a correction factor for tied ranks in a list of values. Here's a step-by-step explanation:

1. Sorting, Counting and Initialization:
- $sv = sorted(rm)$: This line sorts the input list rm in ascending order. The sorted values are stored in the variable sv.
- $ct = Counter(sv)$: This line creates a dictionary named ct using $Counter$ from the $collections$ library (likely imported). $Counter$ is used to count the occurrences of each element in sv. So, ct will map each unique value in sv to its frequency (number of times it appears).
- $n = len(sv)$: This line calculates the total number of elements (n) in the original list rm (which is the same length as the sorted list sv).
- $corr = 0.0$: This line initializes a variable $corr$ to 0.0, which will accumulate the correction value for tied ranks.

```
  for value, count in ct.items():
    if count > 1:
      corr += (count**3 - count)
  if n < 2:
    return 1.0
  else:
    return 1.0 - corr / (n**3 - n)
```

2. Looping through Unique Values and Counts:
- $for\ value,\ count\ in\ ct.items()$: This line starts a loop that iterates over each key-value pair in the dictionary ct.
- $value$: This variable represents the unique value from the original list (sv).
- $count$: This variable represents the count (frequency) of that value in sv.

3. Correction for Tied Ranks:

- $if\ count\ >\ 1$: This condition checks if the current *count* (frequency) is greater than 1. This indicates that there are ties in the ranks (multiple elements have the same value).
- $corr += (count**3 - count)$: If there are ties, this line calculates a correction factor based on the current *count*. It subtracts 1 from the cube of *count*. This formula is used to adjust the sum of ranks for tied elements. The correction value is added to the *corr* variable.

4. Handling Short Lists:

- $if\ n\ <\ 2$: This condition checks if there are less than two elements (n) in the original list.
- $return\ 1.0$: If there's less than two elements, there can't be ties, so the correction factor is simply 1.0 (no correction needed), and the function returns this value.

5. Calculating Final Correction Factor:
- *else*: This block executes if there are two or more elements ($n\ >=\ 2$).
- $return\ 1.0 - corr\ /(n**3-n)$: This line calculates the final rank correction factor. It subtracts the accumulated correction value ($corr$) from 1.0 and then divides it by another correction term based on the total number of elements (n). This final formula considers the total number of elements and the extent of ties to determine the appropriate correction factor. The function then returns this calculated correction factor.

This function calculates a correction factor to account for ties in ranks. It considers the frequency of each value and applies a correction based on the number of ties. The correction factor is used to adjust the sum of ranks in statistical tests like the Kruskal-Wallis test, where tied ranks can affect the results.

C- Function of Test : *kruskalwallis_test*
The function *kruskalwallis_test* compute the Kruskal-Wallis H-test for independent samples.
1. Definition of a function *kruskalwallis_test*.

```
def kruskalwallis_test(data, interpretation = True, alpha = 0.05):
```

Arguments:
*data, A pandas DataFrame containing the data for the test.This data will have 2 or more columns.
*alpha, (Optional) To give a interpretation, this parameter is use like the value of the error. The value of error depend of the domaine of application.The following options are available (default is : 0.05)
*interpretation,(Optional) Give a intrepretion of the result. The following options are available (default is 'True'):
True: print in the terminal a interpretation of the result
False: this option dont give the interpretation of the result

return:
This fonction return The p-value of the Kruskal-Wallis test, a float between 0 and 1.

2. Validation of input parameters and size of the data.
This code performs input validation for the *interpretation* and *alpha* parameters of a potential Kruskal-Wallis test function. It then validates the data (*data*) by ensuring there are at least two groups and that each group has at least one valid data point (numeric and not missing). It cleans the data by removing non-numeric or missing values before potentially proceeding with the Kruskal-Wallis test.

```
    if interpretation not in (True, False):
        raise ValueError("interpretation must be True or False")
     if alpha>0 and alpha<1:
        pass
     else:
        raise ValueError("alpha is a float number betwen 0 and 1, like: 0.05
            ")
    p = len(data.columns.tolist())
    n = []
    N = 0
    if p < 2:
        raise ValueError("Need at least two groups in stats.kruskal()")
    for i in range(1, p+1) :
        n = n + [len(list(filter(lambda x: not isinstance(x,float) or not
            math.isnan(x),data[data.columns.tolist()[i-1]])))]
        N += len(list(filter(lambda x: not isinstance(x,float) or not math.
            isnan(x),data[data.columns.tolist()[i-1]])))
    for i in range(1, p+1):
        if n[i-1]==0:
            raise ValueError("Need at least one data in each groups")
```

. Input Validation:
- *if interpretation not in (True, False)*: This line checks if the *interpretation* parameter is not either *True* or *False*. If it's not, it raises a *ValueError* exception with a message indicating that *interpretation* must be *True* or *False*. This enforces valid input for the interpretation parameter.
- *if alpha > 0 and alpha < 1*: This line checks if the *alpha* parameter is a float between 0 and 1 (exclusive). If it's not, it raises a *ValueError* exception with a message explaining the expected format for *alpha*. This ensures a valid significance level for the test.

. Data Preparation:
- $p = len(data.columns.tolist())$: This line gets the number of columns (groups) in the data (*data*) and stores it in p.
- $n = []$: This line initializes an empty list n to store the number of valid data points in each group.
- $N = 0$: This line initializes a variable N to 0, which will accumulate the total number of valid data points across all groups.

. Group Validation:
- *if* $p < 2$: This line checks if there are less than two groups (p). If so, it raises a *ValueError* exception with a message indicating that the Kruskal-Wallis test needs at least two groups.
. Looping through Groups and Data Cleaning:
- $n = n + [len(list(filter(lambda\ x:\ not\ isinstance(x, float)\ or\ not$
$math.isnan(x), data[data.columns.tolist()[i-1]])))]$: This line uses a list comprehension to achieve two things:
- It filters out non-numeric values (not *float*) or missing values ($math.isnan(x)$) from the current group's data ($data[data.columns.tolist()[i-1]]$).
- It then counts the number of remaining valid data points in the group and appends that count to the n list. This ensures only valid data is considered for the test.
- $N += len(list(filter(...)))$: This line is functionally similar to the previous one. It accumulates the total number of valid data points across all groups into the N variable.

. Validating Data Points per Group:
- *for i in range*$(1, p+1)$: This loop iterates through each group again.
- *if* $n[i-1] == 0$: This line checks if the number of valid data points ($n[i-1]$) in the current group is zero. If it is, it raises a *ValueError* exception with a message indicating that each group needs at least one data point. This ensures there's enough data in each group for the test to function.

3. Concatenated the data into a list:
This block of code iterates through each group, extracts valid data points using filtering, and appends them all to the *Alldata* list. After the loop completes, *Alldata* will contain all the valid data points from all groups in the dataset.

```
    Alldata = []
    for i in range(1, p+1) :
        Alldata  = Alldata  + list(filter(lambda x: not isinstance(x,float)
            or not math.isnan(x),data[data.columns.tolist()[i-1]]))
```

. Initialization:
- $Alldata\ =\ []$: This line initializes an empty list called *Alldata*. This list will eventually store all the collected valid data points.
. Looping through Groups:
- $for\ i\ in\ range(1,\ p+1)$: This loop iterates through each group (column) from 1 to p (assuming p represents the number of groups/columns).
. Data Filtering and Collection:
- $list(filter(lambda\ x:\ not\ isinstance(x, float)\ or\ not\ math.isnan(x), data[data.columns.tolist()[i-1]]))$: This part uses a list comprehension to achieve two tasks:
* Filtering: It filters the data for the current group ($data[data.columns.tolist()[i-1]]$). The $filter$ function applies a custom logic ($lambda\ x:\ ...$) to each element (x) in the data.
* Logic: The *lambda* function checks two conditions:
* $not\ isinstance(x, float)$: This condition excludes elements that are not of type $float$. This removes non-numeric values from the data.
* $not\ math.isnan(x)$: This condition excludes elements where $math.isnan(x)$ is True. The *math.isnan* function checks for missing numerical values (Not a Number). So, this condition removes missing values from the data.
- Extracting Valid Data: After filtering, the remaining elements are valid data points (numeric and not missing). This filtered list is converted to a list using $list()$.
- $Alldata\ =\ Alldata\ +...$: This line concatenates the filtered list of valid data points from the current group to the *Alldata* list. The + operator performs list concatenation.

4. Sum of rank by group

The code iterates through each group, extracts the corresponding ranked data points, calculates the sum of their ranks, and stores that sum in the R list. After the loop completes, R will contain a list of rank sums, one for each group in the data. These rank sums can then be used for further analysis in the Kruskal-Wallis test.

```
    rm = rankdata(Alldata, method = 'average')
    R = []
    start_index = 0
    for taille in n :
        R.append(np.sum(rm[start_index:start_index+taille]))
        start_index += taille
```

. Ranking Data:
- $rm\ =\ rankdata(Alldata, method\ ='\ average')$: This line uses the *rankdata* function (likely from *scipy.stats*) to rank all the data points in *Alldata*. The $method\ ='\ average'$ argument specifies the averaging method for ties (in case multiple data points have the same value). The ranked data is stored in the variable *rm*.

. Initialization:
- $R = []$: This line initializes an empty list R to store the sum of ranks for each group.
- $start_index = 0$: This line initializes a variable *start_index* to 0. This variable will be used to track the starting index of each group's data points within the ranked data (rm).
. Looping through Groups and Calculating Rank Sums:
- $fortailleinn$: This loop iterates through each group size (*taille*) from the n list (which likely stores the number of data points in each group).
- $R.append(np.sum(rm[start_index : start_index + taille]))$: This line calculates the sum of ranks for the current group. Here's what it does:

- $rm[start_index : start_index + taille]$: This selects a slice of the ranked data (rm) from $start\_index$ (inclusive) to $start\_index + taille$ (exclusive). This captures the ranked data points belonging to the current group based on its size ($taille$).
- $np.sum(...)$: This calculates the sum of the elements in the selected slice using $numpy.sum$ (denoted by $np.sum$). This effectively sums the ranks of all data points in the current group.
- $R.append(...)$: The calculated sum of ranks for the group is then appended to the R list.
- $start\_index+ = taille$: This line updates the $start\_index$ by adding the current group size ($taille$). This ensures that the next iteration considers data points from the next group onwards in the ranked data (rm).

This code creates a dictionary to store information about each group (rank sum and group size). It then calculates the Kruskal-Wallis correction factor and the weighted sum of squares of ranks, which are both essential components for calculating the final Kruskal-Wallis H statistic.

```
    dic = {}
    for i in range(1 , p+1) :
        dic[f"V{i}"] = [R[i-1], n[i-1]]
    factor = 12/(N*(N+1))
    S = 0
    for j in range(1, p+1) :
        V = dic[f'V{j}']
        S = S + (V[0]*V[0])/V[1]
```

- $dic =$: This line (commented out in some explanations) initializes an empty dictionary dic. Its purpose might be for temporary storage or future calculations, but it's not used in the provided code. This code snippet builds upon the previous parts and seems to be related to calculating the Kruskal-Wallis H statistic. Here's a breakdown of what it does:

. Storing Rank Sums and Group Sizes (Dictionary): - $foriinrange(1, p + 1)$: This loop iterates through each group (from 1 to p).
- $dic[f"Vi"] = [R[i - 1], n[i - 1]]$: This line does two things:
- Key Creation: It creates a key for the current group in the dictionary dic. The key is constructed as $f"Vi"$ using f-strings (formatted string literals). This creates keys like "V1", "V2", etc., for each group.
- Value Assignment: It assigns a list as the value for the corresponding key. This list contains two elements:
- $R[i - 1]$: This is the sum of ranks for the current group, retrieved from the R list calculated earlier.
- $n[i - 1]$: This is the number of data points in the current group, retrieved from the n list.

. Kruskal-Wallis Correction Factor:
- $factor = 12/(N * (N + 1))$: This line calculates the Kruskal-Wallis correction factor ($factor$). The formula $12/(N * (N + 1))$ is used, where N is the total number of data points (previously calculated and stored in the N variable). This factor is used to adjust the H statistic for ties in ranks.

. Weighted Sum of Squares of Ranks:
- $S = 0$: This line initializes a variable S to 0. It will accumulate the weighted sum of squares of ranks.
- $forjinrange(1, p + 1)$: This loop iterates through each group (from 1 to p).
- $V = dic[f'Vj']$: This line retrieves the value (list) associated with the current group's key ($f"Vj"$) from the dictionary dic. Recall that this list contains the sum of ranks ($V[0]$) and the number of data points ($V[1]$) for the current group.
- $S = S + (V[0] * V[0])/V[1]$: This line calculates the contribution of the current group to the weighted sum of squares of ranks and adds it to the S variable. Here's how it works:
* $V[0] * V[0]$: This squares the sum of ranks for the current group.
* $/V[1]$: This divides the squared sum of ranks by the number of data points in the group. This

effectively weights the contribution of each group based on its size.
* $S = S + ...$: The calculated weighted contribution is then added to the S variable, which accumulates the total weighted sum of squares of ranks.

5. Final steps of Kruskal-Wallis test calculation
This part of the code calculates the Kruskal-Wallis statistic H and the associated p-value. The H indicates the strength of the effect (differences between group medians), and the p-value evaluates the statistical significance of this effect.

```
H = factor*S-3*(N+1)
H /= rank_correction(rm)

df = p-1
p_value = 1 - chi2.cdf(H, df)
```

The preceding code has prepared the data and calculated intermediate values necessary for the Kruskal-Wallis test. Let's now look at the final steps:
. Calculation of Kruskal-Wallis statistic H:
- $H = factor * S - 3 * (N + 1)$: This line calculates the Kruskal-Wallis statistic H.
* $factor$: Kruskal-Wallis correction factor calculated earlier.
* S: Weighted sum of squares of ranks calculated earlier.
* N: Total number of valid data points.
- $H\ /=\ rank\_correction(rm)$: This line divides the calculated H by the rank correction obtained by calling the $rank\_correction(rm)$ function. This function (not provided) takes into account ties in ranks and applies a correction for a more accurate test.
. Calculation of p-value:
- $df\ =\ p - 1$: This line calculates the degrees of freedom (df) of the test. In the Kruskal-Wallis test, df is equal to the number of groups minus one.
- $p\_value\ =\ 1\ -\ chi2.cdf(H,\ df)$: This line calculates the p-value of the test.
- $chi2.cdf(H,\ df)$: Uses the cumulative distribution function (*cdf*) of the chi-square distribution to calculate the cumulative probability of obtaining a H value as extreme or more extreme, taking into account the degrees of freedom (df).
- $1\ -\ chi2.cdf(H,\ df)$: We subtract this cumulative probability from 1 to obtain the p-value. A low p-value (close to 0) indicates that the observations from the groups are likely from different populations, rejecting the null hypothesis of the test (null hypothesis of equality of group medians).
6. Interpretation of results
This code snippet provides a clear and informative way to interpret and report the results of a Kruskal-Wallis test. It conditionally prints the results based on the *interpretation* flag, presents the test statistic, degrees of freedom, and p-value, and offers basic interpretation of the results based on the p-value and significance level.

```
if interpretation:

    print('                Kruskal-Wallis    ')
    print('              ____________________');
    print('              ____________________');
    print('\n \n')
    print(f'H = {H}, df = {df}, p-value = {p_value} \n')

    if (p_value <= alpha):
        print(' Alternative hypothesis: True \n')
        print(' So the medians are not all equal. \n')
    else:
        print(' Null hypothesis: True \n')
        print(' So the medians are all equal. \n')

return [p_value ,H]
```

. Conditional Printing Based on Interpretation Flag:
- $if\ interpretation$: This line checks if the $interpretation$ parameter passed to the function is $True$. If it is, the code proceeds to print the test results. This allows you to control whether the results are printed or not based on your needs.
. Printing Test Statistic, Degrees of Freedom, and p-value:
- Assuming $interpretation$ is $True$, the code performs the following actions:
- Prints a formatted header for the Kruskal-Wallis test results.
- Prints the calculated Kruskal-Wallis statistic (H), degrees of freedom (df), and p-value (p_value) in a user-friendly format.
. Hypothesis Testing Interpretation:
- It compares the p-value (p_value) to the significance level ($alpha$) (assumed to be set elsewhere in the function).
- $if(p\_value <= alpha)$: If the p-value is less than or equal to the significance level (usually 0.05), it indicates evidence against the null hypothesis.
- Prints a message stating that the alternative hypothesis is likely true, suggesting the medians of the groups are not all equal.
- $else$: If the p-value is greater than the significance level, it fails to reject the null hypothesis.
- Prints a message stating that the null hypothesis is likely true, suggesting the medians of the groups might be equal.
. Returning the p-value and the statistic H:
- The function returns the p_value and the statistic H calculated earlier. This allows you to use the p-value in your program for further analysis or decision-making if needed.

3.4 Implementation in Python and Matlab: MATLAB Script

• This function *rank_correction* is defined but not called in the main script. It calculates a correction factor for tied ranks, which can be useful for more accurate p-value calculation in some scenarios. The function have the same structure within the code in python.

```
function correction_factor = rank_correction(rm)
    sv = sort(rm);
    count_values = unique(sv);
    count = histc(sv, count_values);
    n = numel(sv);
    corr = 0.0;
    for i = 1:numel(count_values)
        if count(i) > 1
            corr = corr + (count(i)^3 - count(i));
        end
    end
    if n < 2
        correction_factor = 1.0;
    else
        correction_factor = 1.0 - corr / (n^3 - n);
    end
```

kruskalwallis_test **Function**:
The code defines the beginning of a function named *kruskalwallis_test*. It appears to be implementing the Kruskal-Wallis test, a non-parametric statistical test used to compare the medians of several groups.

```
function [p_value, H] = KW_test(data, interpretation, alpha)

    if nargin < 2
        interpretation = true; %  by defaut
    end
    if nargin < 3
        alpha = 0.05; %  by defaut
    end
```

Function Definition and Arguments:
The function definition starts with the *function* keyword followed by the function name *kruskalwallis_test* and a pair of square brackets enclosing the function's input arguments. Inside the brackets, we see the argument names separated by commas:
. *data*: This is a DataFrame containing the data to be analyzed.
. *interpretation*: This is an optional Boolean argument that determines whether to display the interpretation of the test results. The default value is *true*.
. *alpha*: This is another optional argument representing the significance level for the test. The default value is 0.05.

Input Validation:

The function includes input validation checks to ensure the arguments are provided correctly. It uses the *nargin* function to check the number of input arguments and assigns default values if they are not provided.
. Checking for *interpretation*: The $if nargin < 2$ statement checks if there are fewer than two arguments. If so, it sets the *interpretation* variable to *true*, indicating that the interpretation should be displayed.
. Checking for *alpha*: The $if nargin < 3$ statement checks if there are fewer than three arguments. If so, it sets the *alpha* variable to 0.05, representing the default significance level.

Validation of input

These lines of code perform additional input validation for the *interpretation* and *alpha* arguments in the *kruskalwallis_test* function.

```
    if ~(islogical(interpretation) || isscalar(interpretation))
         error('interpretation must be a logical scalar (true or false)');
    end
    if alpha > 0 && alpha < 1
    else
        error('alpha must be a float number between 0 and 1, e.g., 0.05');
    end
```

Validation for *interpretation*:
* $if\ (islogical(interpretation)||isscalar(interpretation))$: This line checks if the *interpretation* argument is not a logical scalar (True or False) using the (NOT) operator.
* $islogical(interpretation)$: This part checks if *interpretation* is a logical data type (True or False).
* $isscalar(interpretation)$: This part checks if *interpretation* is a single value (scalar).
* $error('interpretationmustbealogicalscalar(trueorfalse)');$: If the condition is not met (interpretation is not logical or scalar), the code throws an error message indicating the expected format (logical scalar - True or False).

Validation for *alpha*:
* $if alpha > 0\&\&alpha < 1$: This line checks if *alpha* is greater than 0 and less than 1 using the logical AND (&&) operator. This ensures *alpha* is within the valid range for significance levels (between 0 and 1).
* The *else* block after this *if* statement isn't shown, but it likely throws an error message if *alpha* is outside the valid range. The error message would likely specify the expected range (e.g., 0 to 1) and suggest a common value like 0.05.
These lines ensure that *interpretation* is a clear True or False value and *alpha* is a valid number between 0 and 1 for the Kruskal-Wallis test function.

This code prepares for the Kruskal-Wallis test by extracting group information from the DataFrame and performing basic checks to ensure there are enough groups for the test.

```
    nc = width(data);
    if nc < 2
    error('Need at least two groups in stats.kruskal()');
    end
```

The code focuses on checking if there are at least two groups in the data for the Kruskal-Wallis test. Here's a breakdown:
1. $nc = width(data);$: This line calculates the number of columns in the *data* variable and stores it in *nc*. Since the Kruskal-Wallis test compares multiple groups, it requires at least two columns (groups) for analysis.
2. $ifnc < 2$: This line checks if *nc* (number of columns) is less than 2.
3. $error('Needatleasttwogroupsinstats.kruskal()');$: If the condition $nc < 2$ is true, it means there's less than two groups in the data. The code throws an error message indicating that at least two groups are required for the Kruskal-Wallis test ($stats.kruskal()$ might be a reference function name used within the code).
In essence, this code ensures the Kruskal-Wallis test is applied only when there are sufficient groups (at least two) for meaningful comparison.

```
    if all(isnumeric(data{:,:}))
         M = table2array(data);
    else
        M = table2array(data);
        M = str2double(M);
    end
```

This code aims to convert a data structure, potentially a table, into a numerical array suitable for further analysis. It handles both cases where the data is already numeric and when it needs conversion from strings or other non-numeric formats.

Breakdown:
1. Checking Data Type:
- $ifall(isnumeric(data:,:))$: This line checks if all elements in the *data* variable are numeric using the $isnumeric()$ function. The $all()$ function ensures that all elements are checked, and the $:,'$ notation indicates iterating over all rows and columns in the data structure.

2. Converting Numeric Data:
- $M = table2array(data);$: If all elements are numeric, this line directly converts the *data* table to a NumPy array using the $table2array()$ function. This creates a multidimensional array representing the data.

3. Handling Non-Numeric Data:
- $else$ :: If not all elements are numeric, this block executes.
- $M = table2array(data);$: Similar to the previous step, it converts the *data* table to a NumPy array.
- $M = str2double(M);$: This line converts each element in the NumPy array M to a numeric value using the $str2double()$ function. This assumes the non-numeric elements are strings that can be converted to numbers.

Alternative Approach:
If the data structure is not a standard MATLAB table, it might be more appropriate to use more specific data handling functions depending on the actual data format. For instance, if the data is in a text file, *csvread* or *importdata* functions could be used.

```
    list = M(:,1);
    % Initialize the list with the first column
    for i = 2:nc
        list = vertcat(list,M(:,i));
        % Concatenate each column to the list
    end

    clist = list(~isnan(list));
    % Remove NaN values from the list
    rank = tiedrank(clist, 'average');
    % Compute ranks, handling ties with the average method
```

The code demonstrates two steps in processing data for the Kruskal-Wallis test:

1. Combining Data and Removing NaN values:
- $list = M(:,1);$: This line initializes a list named *list* with the first column of the data matrix M.
- $fori = 2 : nc$: This loop iterates through the remaining columns (2nd to $n$cth) of the data matrix M.
- $list = vertcat(list, M(:,i));$: Inside the loop, each iteration concatenates the current column $(M(:,i))$ to the existing *list* using *vertcat*. This effectively combines all data points from different columns into a single list.
- $clist = list(isnan(list));$: This line removes Not-a-Number (NaN) values from the *list*. The *isnan* part creates a logical index that selects elements that are not NaN, and these elements are assigned to the new list *clist*.

2. Calculating Ranks:
- $rank = tiedrank(clist,'average');$: This line calculates the ranks for each data point in the *clist* (NaN-removed list). The *tiedrank* function (potentially from a statistics toolbox) is used and set to

handle ties with the 'average' method. This assigns a rank to each data point, considering ties and averaging the rank if multiple data points share the same value.

```
n = [];
 for i = 1:nc
     g = M(:,i);
     n = [n, length(g(~isnan(g)))];
     % Count the number of non-NaN values in each group
 end

 % Check for at least one data point in each group
 for i = 1:nc
 if n(i) == 0
     error('Need at least one data in each group');
 end
 end
```

1. Counting Non-NaN Values per Group:
- $n = [];$: Initializes an empty array n to store the count of non-NaN values in each group.
- $fori = 1 : nc$: This loop iterates through each column (group) from 1 to nc (number of columns).
- $g = M(:, i);$: This line extracts the current column (group) i from the data matrix M and assigns it to a temporary variable g.
- $n = [n, length(g(\ isnan(g)))]$: This line counts the number of elements in g that are not NaN using $\ isnan$ for logical indexing. The resulting count is appended to the n array using square brackets [].

2. Checking for Empty Groups:
- $fori = 1 : nc$: This loop iterates through the elements in the n array (representing group counts).
- $ifn(i) == 0$: This conditional statement checks if the count for the current group ($n(i)$) is zero.
- $error('Needatleastonedataineachgroup');$: If the count is zero, this line throws an error message indicating that the Kruskal-Wallis test cannot proceed because there's no data in a particular group.

```
SR = [];
 j = 0;
 for i = 1:nc
     t = rank(j+1:j+n(i),1);
     % Extract ranks for the current group
     sommeT = sum(t);
     SR = [SR, sommeT];
     % Sum of ranks for the current group
     j = j + n(i);
     % Update the index
 end
```

This code calculates the sum of ranks for each group in the Kruskal-Wallis test. Here's a breakdown:

1. Initialization:
- $SR = [];$: Initializes an empty array SR to store the sum of ranks for each group.
- $j = 0;$: Sets a variable j to 0, which will be used as an index to keep track of the starting position of each group in the rank list.

2. Looping Through Groups:
- $fori = 1 : nc$: This loop iterates through each group (column) from 1 to nc (number of columns).

3. Extracting Ranks for a Group:
- $t = rank(j + 1 : j + n(i), 1);$: This line extracts the ranks for the current group i.
- $j + 1 : j + n(i)$: This defines the starting and ending indices for extracting ranks. It starts at $j + 1$

(because j is initially 0) and goes up to $j + n(i)$. The $n(i)$ part ensures we extract the correct number of ranks based on the number of data points in the current group ($n(i)$).
- $rank(...)$: This function (potentially from a statistics toolbox) retrieves the ranks for the specified elements ($j + 1$ to $j + n(i)$). The 1 argument specifies returning ranks as a column vector.

4. Calculating and Storing Sum of Ranks:
- $sommeT = sum(t);$: This line calculates the sum of the ranks extracted for the current group (t).
- $SR = [SR, sommeT];$: This line appends the calculated sum of ranks ($sommeT$) to the SR array, effectively storing the sum of ranks for each group.

5. Updating Index:
- $j = j + n(i);$: This line updates the index j by adding the number of data points in the current group ($n(i)$) to it. This ensures j points to the starting position of the next group in the rank list for the next iteration of the loop.
Overall, this code iterates through each group, extracts the corresponding ranks, calculates their sum, and stores the sum of ranks for each group in the SR array. This is a crucial step in the Kruskal-Wallis test for comparing rank distributions across groups.

```
    N = sum(n);
    % Total number of observations
    f = 12 / (N * (N + 1));
    % Pre-factor for H statistic calculation
    S = 0;
    for i = 1:nc
        S = S + (SR(i) * SR(i)) / n(i);
        % Sum of squared ranks divided by group size
    end
```

The code calculates parts of the Kruskal-Wallis H statistic in MATLAB. Here's a breakdown:

1. Total Number of Observations:
- $N = sum(n);$: This line calculates the total number of observations (N) by summing the elements in the n array. The n array stores the count of non-NaN values in each group, so summing these counts provides the total number of data points across all groups.

2. Pre-factor for H Statistic:
- $f = 12/(N * (N + 1));$: This line calculates a constant factor f used in the Kruskal-Wallis H statistic formula. It involves the total number of observations (N) calculated earlier. This factor helps normalize the H statistic based on the sample size.

3. Sum of Squared Ranks Divided by Group Size:
- $S = 0;$: Initializes a variable S to 0. This variable will accumulate the sum of squared ranks divided by group size for all groups.
- $fori = 1 : nc$: This loop iterates through each group (column) from 1 to nc (number of columns).
- $S = S + (SR(i) * SR(i))/n(i);$: This line calculates the contribution of the current group i to the S variable.
- $SR(i) * SR(i)$: Squares the sum of ranks ($SR(i)$) for the current group. Squaring emphasizes the contribution of groups with larger overall ranks.
- $/n(i)$: Divides the squared sum of ranks by the number of data points ($n(i)$) in the current group. This normalization accounts for groups with different sizes.
- The entire expression is added to the S variable, accumulating the contributions from each group.
Overall, this code calculates the total number of observations and a pre-factor for the H statistic. It also iterates through groups, squares the sum of ranks for each group, divides by the group size, and accumulates these values to prepare for the final calculation of the Kruskal-Wallis H statistic.

```
    H = f * S - 3 * (N + 1);
    H = H/rank_correction(rank);
    df = nc - 1;
    % Calculate the Kruskal-Wallis H statistic
    p_value  = 1 - chi2cdf(H, df);
    % Calculate the p-value from the chi-squared distribution
```

This code snippet finalizes the calculation of the Kruskal-Wallis H statistic, calculates the p-value, and sets the degrees of freedom in MATLAB. Here's a breakdown:

1. Kruskal-Wallis H Statistic:

- $H = f * S - 3 * (N + 1);$: This line calculates the Kruskal-Wallis H statistic (H).
- f: The pre-factor for H statistic calculated earlier.
- S: The sum of squared ranks divided by group size, accumulated for all groups.
- $3 * (N + 1)$: A constant term based on the total number of observations (N).
- This formula essentially calculates the overall deviation of rank distributions across groups, considering the sample size and normalization by group size.

2. Rank Correction:

- $H = H/rank\_correction(rank);$: This line applies a correction to the H statistic based on the $rank\_correction$ function (assumed to be defined elsewhere). The corrected H statistic is stored back in H.
- The $rank\_correction$ function likely accounts for ties in the data by adjusting the H statistic to reflect the reduced information content due to ties.
3. Degrees of Freedom:
- $df = nc - 1;$: This line calculates the degrees of freedom (df) for the chi-squared test used to determine the p-value
. - nc: The number of columns (groups) in the data.
- Subtracting 1 accounts for one degree of freedom lost due to estimating the population median with the sample medians.

4. P-value Calculation:

- $p_value = 1 - chi2cdf(H, df);$: This line calculates the p-value associated with the H statistic using the chi-squared cumulative distribution function ($chi2cdf$).
- H: The corrected Kruskal-Wallis H statistic.
- df: The degrees of freedom calculated earlier.
- $1 - chi2cdf(H, df)$: This expression calculates the probability of observing a chi-squared statistic greater than or equal to H with df degrees of freedom. This is the p-value, representing the probability of obtaining such an extreme H statistic by chance if the null hypothesis (all groups have equal medians) is true.

This code snippet completes the Kruskal-Wallis test by calculating the corrected H statistic, determining the degrees of freedom, and calculating the p-value from the chi-squared distribution. These values provide insights into the potential differences in medians between the groups in the data. **Interpretation**

```
    if interpretation
     disp(['                        Kruskal-Wallis test']);
     disp(['                        ____________________']);
     disp(['                        ____________________']);
     disp([' H = ' + string(H) + ',          df = ' + string(df) + ',        p-
        value = ' + string(p_value) ]);
     if (p_value <= alpha)
        disp(' Alternative hypothesis: True')
        disp(' So the medians are not all equal. ')
     else
        disp(' Null hypothesis: True ')
        disp(' So the medians are all equal.')
     end
   end
```

The Interpretation have the same structure within the code in python.

3.5 Comparative Study of the script

Similarities:
- Operation: Both codes implement the Kruskal-Wallis test to compare the medians of multiple groups within a dataset. They follow the same overall structure and logic of the Kruskal-Wallis test.
- Handling of Missing Values (NaN): Both codes handle missing values to ensure they do not affect the calculations.
- Calculation of Degrees of Freedom (df) and p-value: Both codes calculate degrees of freedom (df) and the p-value using the chi-squared cumulative distribution function (chi2cdf).

Differences:
- Python: Requires external libraries like pandas, numpy, scipy.stats for data manipulation and statistical calculations.
- MATLAB: Uses built-in functions for data manipulation and statistical calculations, eliminating the need for additional libraries.
- Data Loading:
- MATLAB uses *readtable* to load data from a CSV file, whereas Python uses *pd.read_excel* to load data from an Excel file.
- Data Processing:
- MATLAB uses *table2cell* and *filter* functions for data processing, while Python uses *filter* and direct indexing within loops.
- MATLAB creates a list *Alldata* to store all observations, while Python filters observations directly within loops.
- Calculation of Sum of Ranks:
- MATLAB initializes an empty list *R* and accumulates sums for each group, while Python uses a dictionary *dic* to store sums and group information.
- Calculation of Weighted Sum of Squares of Ranks:
- MATLAB uses matrix multiplication and element-wise division, whereas Python uses a loop to calculate sums of squares and divides by group sizes.
- Error Handling:
- MATLAB appears to have more detailed error handling using filtering steps and *isnan*.
- Conciseness:
- The Python code appears more concise and makes efficient use of built-in functions.

Both the MATLAB and Python implementations efficiently perform the Kruskal-Wallis test, with slight differences in data processing and calculation methods due to the specific features of each language. The choice between MATLAB and Python depends on factors such as familiarity with the language, data format, and desired level of control over data processing steps.
- The *rank_correction* function is optional in both implementations and allows for adjusting the p-value for tied ranks in certain cases.
- The Kruskal-Wallis test is non-parametric, making it suitable for data that does not necessarily follow a normal distribution.
- Assumptions of the Kruskal-Wallis test include independence of observations within groups and homogeneity of variances between groups.

4 Verification and Validation

Verification and Validation with Open-Source Implementation: SciPy Stats (scipy.stats)

To validate the Python and MATLAB codes, we will apply them to a database.
DATA: BIRTH WEIGHTS (lbs.) OF EIGHT LITTERS OF PIGS. Source: Snedecor [46], (Table 10.12)3.1
Python:
To apply the Kruskal-Wallis test function to the data in python, we will first put the data into an Excel, CSV, or other file format to import it as a pandas database. Here we will use an Excel file for the importation. The file will be named Snedecor and imported as 'Snedecor.xlsx'. The developpement environment used here is : Spyder

```
import pandas as pd
import numpy as np
from scipy.stats import rankdata, chi2
from collections import Counter
import math
## The data is loaded from a CSV, excel or any which fille to have a dataset
    .
data = pd.read_excel("Snedecor.xlsx")
## Calling the function
p = kruskalwallis_test(data)
```

The result obtained in the terminal is:

```
Python 3.11.7 | packaged by Anaconda, Inc. | (main, Dec 15 2023, 18:05:47) [
    MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 8.20.0 -- An enhanced Interactive Python.

runfile('C:/Users/GBAGUIDI/Documents/Projet_StatComp/Python/Kruskal-Wallis.
    py', wdir='C:/Users/GBAGUIDI/Documents/Projet_StatComp/Python')
                  Kruskal-Wallis
                -------------------
                -------------------

H = 18.56541380297822, df = 7, p-value = 0.009663431264406164

 Alternative hypothesis: True

 So the medians are not all equal.
```

MATLAB:

To apply the Kruskal-Wallis test function to the data in Matlab, we will first put the data into an Excel, CSV, or other file format to import it as a table of data. Here we will use a csv file for the importation. The file will be named Snedecor and imported as 'Snedecor.csv'. The development environment used here is : MATLAB R2023a

```
% Load data from a file.
data = readtable("Snedecor.csv");

% Call the kruskalwallis_test function to perform the Kruskal-Wallis test on
    the loaded data.
p=kruskalwallis_test(data)
```

The result obtained in the command window is:

```
>> Kruskal-Wallis
                      Kruskal-Wallis test
                      -------------------
                      -------------------
 H = 18.5654,        df = 7,       p-value = 0.0096634
 Alternative hypothesis: True
 So the medians are not all equal.

p =

   0.009663431264406
```

scipy.stats:
When using the integrate function for the Kruskal-Wallis test in python, on the scipy.stats library, we have this:

```
from scipy.stats import kruskal

Litter1 = [2.0,2.8,3.3,3.2,4.4,3.6,1.9,3.3,2.8,1.1]
Litter2 = [3.5,2.8,3.2,3.5,2.3,2.4,2.0,1.6]
Litter3 = [3.3,3.6,2.6,3.1,3.2,3.3,2.9,3.4,3.2,3.2]
Litter4 = [3.2,3.3,3.2,2.9,3.3,2.5,2.6,2.8]
Litter5 = [2.6,2.6,2.9,2.0,2.0,2.1]
Litter6 = [3.1,2.9,3.1,2.5]
Litter7 = [2.6,2.2,2.2,2.5,1.2,1.2]
Litter8 = [2.5,2.4,3.0,1.4]
p=kruskalwallis_test(data)
H , pv = kruskal(Litter1,Litter2,Litter3,Litter4,Litter5,Litter6,Litter7,
   Litter8)
print(H ,pv)
```

The result obtained in the terminal is:

```
Python 3.11.7 | packaged by Anaconda, Inc. | (main, Dec 15 2023, 18:05:47) [
   MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 8.20.0 -- An enhanced Interactive Python.

runfile('C:/Users/GBAGUIDI/Documents/Projet_StatComp/Python/Kruskal-Wallis.
   py', wdir='C:/Users/GBAGUIDI/Documents/Projet_StatComp/Python')
18.56541380297822 0.009663431264406185
```

The value of the Kruskal-Wallis statistic given by the python code in this report and that of the scipy.stats library is exactly the same $H = 18.56541380297822$. The p-value is almost the same with a difference of $21 \times 10^{-18} \simeq 0.000000000000000021 \simeq 0$.
For MATLAB we have $H = 18.565413802978220$, this is the same H given by the scipy.stats library. The p-value is equal to 0.009663431264406, the difference beteween this and the p-value of the scipy.stats library is $1.8388068845354155e - 16 = 0.00000000000000018388 \simeq 0$.
These results indicate that the implementations of the Kruskal-Wallis test in Python and MATLAB are consistent and reliable, producing comparable results to those obtained with a well-established statistical library. The small differences observed in the p-values can be attributed to numerical variations and do not question the validity of the methods used. Overall, the results obtained by the Python and MATLAB codes demonstrate strong agreement in their ability to perform the Kruskal-Wallis test and interpret the results. This reinforces confidence in the validity and robustness of both implementations.

5 CONCLUSION

This project aimed to develop a deep understanding of the Kruskal-Wallis test, a non-parametric statistical test used to compare the medians of several groups in a dataset. The goal was to implement this test in two different programming environments: Python and MATLAB. We conducted a comparative analysis of the codes. From this analysis, it was noted that the implementations of the Kruskal-Wallis test in Python and MATLAB followed similar approaches to perform the statistical test, both languages used strategies to handle missing values (NaN) to ensure the integrity of the calculations. Similarly, MATLAB leveraged built-in functions for data processing and statistical calculations, while Python used external libraries such as pandas, numpy, and scipy.stats. Differences also existed in the specific methods of data processing, the calculation of rank sums, and the approach to calculating the Kruskal-Wallis statistic. For validation and verification, the results obtained by the Python and MATLAB implementations were validated by comparing the H statistic values, degrees of freedom (df), and p-values with the *scipy.stats.kruskal* function from SciPy. Minor numerical differences observed in the p-values could be attributed to variations in implementation or calculation methods but did not undermine the validity of the results.
The results demonstrated that the implementations of the Kruskal-Wallis test in Python and MATLAB were consistent and reliable, producing results comparable to those obtained with a well-established statistical library such as SciPy. Although minor differences were observed in the p-values, these were acceptable given the numerical variations.
In conclusion, the Python and MATLAB codes demonstrated a strong agreement in their ability to perform the Kruskal-Wallis test and interpret the results. These findings reinforce confidence in the validity and robustness of both implementations, while also highlighting the specific features and relative advantages of each language for this type of statistical analysis.

Bibliography

Univariate statistical methods, D.S.J.C Gbemavo.
Course for Biostatistics Master Program (Labef, Benin),Faculty of Agronomic Science, University of Abomey-Calavi.

Kruskal, William H., et W. Allen Wallis. 1952. « Use of Ranks in One-Criterion Variance Analysis ». Journal of the American Statistical Association 47(260): 583-621. doi:10.1080/01621459.1952.10483441.

Hollander, Myles, Douglas A. Wolfe, et Eric Chicken. 2013. Nonparametric Statistical Methods. John Wiley and Sons.

Annexes

- Complete source code for the Kruskal-Wallis test in MATLAB and Python
* https://github.com/GBAGUIDI-M/kruskal-wallis

- Examples of datasets used for testing
* https://github.com/GBAGUIDI-M/kruskal-wallis