Introduction to Epidemiology with Julia

This is a notebook written in Julia language and hosted on the free Juliahub platform. You can copy paste the codes and work on the codes to learn about various measurements used in Epidemiology, particularly Environmental Epidemiology.

We start with an exposition of Julia language. These are referred to here as Julia chops and introduces you to the basics of the language. You can learn more about these in the Julia manual. We then move to introduce you to the way Julia can be used to calculate the basic measurements in Epidemiology.

Work linearly through the workbook. We start with the definitions of prevalence and incidence, standardised mortality ratios, calculation of analysis of two by two tables, rates, odds ratios, attributable risks, and population attributable risks.

```
md"""
## Introduction to Epidemiology with Julia

This is a notebook written in Julia language and hosted on the free Juliahub platform. You can copy paste the codes and work on the codes to learn about various measurements used in Epidemiology, particularly Environmental Epidemiology.

We start with an exposition of Julia language. These are referred to here as Julia chops and introduces you to the basics of the language. You can learn more about these in the Julia manual. We then move to introduce you to the way Julia can be used to calculate the basic measurements in Epidemiology.

Work linearly through the workbook. We start with the definitions of prevalence and incidence, standardised mortality ratios, calculation of analysis of two by two tables, rates, odds ratios, attributable risks, and population attributable risks.


"""
```

```
using DataFrames, CSV, CairoMakie, GLM, PrettyTables,
CategoricalArrays,
Statistics,
Distributions,
HypothesisTests
```

Julia chops and foundations (try these first)

First learn some basic moves of Julia language. Julia language is an open source, fast, easy to learn language for data analysis and scientific computing.

Concept: Everything in Julia is an expression

- Variables and constants
- Vectors and Arrays and Linear Algebra
- Dot operations (broadcasting)
- Loops, conditionals, list comprehension
- Functions
- Iterations
- Data Frames and data manipulations
- Figures and tables

```
md"""

## Julia chops and foundations (try these first)

First learn some basic moves of Julia language. Julia language is an open source, fast, easy to learn language for data analysis and scientific computing.

### Concept: Everything in Julia is an expression

- Variables and constants
- Vectors and Arrays and Linear Algebra
- Dot operations (broadcasting)
- Loops, conditionals, list comprehension
- Functions
- Iterations
- Data Frames and data manipulations
- Figures and tables

"""
```

π = 3.1415926535897...

```
begin
    # x is a variable
    # x = 1 is an expression
    # Everything is an expression in Julia
    x = 1
    y = "Hello World" # is a string
    println(y) # prints the string
    z = [1,2,3] # this is a vector, vector has data that are of the
    same type
    typeof(z) # gives Vector of Int64 as these are integers
    z1 = [1,2.0, 3] # will convert all data inside this array to
    Floating point
    typeof(z1) # will result in Float64 Vector
    a = [1 2 3; 4 5 6] # is a 2 X 3 matrix where rows = 2, cols = 3
    ndims(a) # results in the number of dimension of the matrix here 2
    size(a) # results in the number of rows and columns (2,3)
    b = (1,2,3) # is a tuple, note the parentheses
    # you cannot edit the contents of a tuple, see
    # b[1] = 10  # will raise an error (don't run)
    # But you cadn change the contents of a vector or an array in
    square brackets
    z[1] = 10 # will work
    z # will result in 10,2,3
    pi # is a constant with value 3.14... (note the symbol, :-) )

end
```

```
Hello World
```

Arrays and matrices

```
md"""

## Arrays and matrices


"""
```

```
view(::Matrix{Int64}, 1, :): [66, 72, 78]
```

```
begin
 # take two matrices a1 and b1
 a1 = [1 2 3; 4 5 6] # 2 X 3 matrix
 b1 = [7 8 9; 10 11 12; 13 14 15] # 3 X 3 matrix
# c1 = a1 + b1 # will result in an error (do not run)
a2 = a1 + a1 # will work as the individual elements will add up
a3 = a1 * b1 # will result in  matrix dot product as a 2 X 3 matrix
a3[1] # will get the first element of the matrix as 66
axes(a3, 1) # will get you the number of rows
axes(a3, 2) # will get you the number of columns
ndims(a3)
first_col = [col for col in eachcol(a3)][1] # get the first column of
the matrix
first_row = [row for row in eachrow(a3)][1] # get the first row of the
matrix

end
```

Broadcasting operations

- Use the dot operator
- Dot operator does operations element by element
- Start with the dot and then write the name of the function (see functions below)

```
md"""
## Broadcasting operations

- Use the dot operator
- Dot operator does operations element by element
- Start with the dot and then write the name of the function (see
functions below)
"""
```

```
2×3 Matrix{Int64}:
  4356   5184   6084
 24336  29241  34596
```

```
begin
    # Broadcasting operations
    a4 = a3 .* 2 # will multiply each element with 2
    a5 = a3 .* a3 # will  multiply each element with each element
    # but this is different from matrix cross product
    # a3 * a3 (don't run, won't work) as the dimensions will not match
end
```

Conditionals and Loops

- Conditionals (if, elseif, else)
- Ternary operator (test ? statement if true : statement if false)
- Loops (for ... end, and while...end)
- List comprehension

```
md"""
## Conditionals and Loops

- Conditionals (if, elseif, else)
- Ternary operator (test ? statement if true : statement if false)
- Loops (for ... end, and while...end)
- List comprehension
"""
```

```
# Conditional, if ...
# if ... end
begin
x2 = 7
if x < 1
    println("x is less than one")
else
    println("x is more than one")
end
# this will print x is more than one
end
```

```
x is more than one
```

```
# Conditional if elseif ... end
begin
    x3 = 14
    y3 = 15

    if x3 < 1 && y3 > 10
        println("x is less than one")
    elseif x3 > 10 && y3 > 10
        println("x and y are both greater than 10")
    else
        println("neither condition holds")
    end

    # remember to have evaluations in one line after if and elseif
    # remember to end the if ... end
end

```

```
x and y are both greater than 10
```

```
"x is greater than 10"
```

```
# ternary operator
begin
    x4 = 16
    x4 > 10 ? "x is greater than 10" : "x is less than 10"
    # Notes:
    # A single line expression
    # ternary is because there are two clauses, one with ? that tests
    if true
    # one with : that states if the test fails
end
```

For ... end loop

```
md"""
## For ... end loop
"""
```

```
begin
    # Loops for ... end
    x5 = 0
    for x5 = 0:6
        x5 += 1
        println("x is $x5")
    end
    # Note:
    # it continuously runs a loop starting with x = 0
    # then increases x by 1
    # then prints that value of x
# Alternatively, you could also write
    for x5 in 0:7
        x5 += 1
        println("x is $x5")
    end
end
```

```
x is 1
x is 2
x is 3
x is 4
x is 5
x is 6
x is 7
x is 1
x is 2
x is 3
x is 4
x is 5
x is 6
x is 7
x is 8
```

While...end loop

- Here while is evaluated as true
- As long as the while condition holds, the next statement will be evaluated

```
md"""

## While...end loop

- Here while is evaluated as true
- As long as the while condition holds, the next statement will be
evaluated

"""
```

```
begin
    i = 0
    while i < 7
        i += 1
        println("i is $i")
    end
    # this is another way to loop using the while...end statement
    # while indicates "as long as"
    # += number indicates increase the value of i by that number
    # the dollar sign signifies string interpolation
end
```

```
i is 1
i is 2
i is 3
i is 4
i is 5
i is 6
i is 7
```

List comprehension

- Is a useful way to concatenate over a series and print the outputs
- It works element by element and then returns a vector

```
md"""
## List comprehension

- Is a useful way to concatenate over a series and print the outputs
- It works element by element and then returns a vector


"""
```

```
[2, 4, 6]
```

```
begin
    # take a vector
    x6 = [1,2,3]
    # write an iterator that will return another vector y6
    # the y7 will contain two times the numbers in the x6
    # to have 2,4,6
    y6 = [i * 2 for i in x6]

end
```

Functions

Functions are essential when you find yourself repeating a task. Write a function to accomplish a given task. You can include conditionals, loops, and other expressions inside a function. Function returns an expression with the return statement. Functions can be named functions or they can be anonymous functions (see the examples)

```
md"""

## Functions
Functions are essential when you find yourself repeating a task. Write a function to accomplish a given task. You can include conditionals, loops, and other expressions inside a function. Function returns an expression with the return statement. Functions can be named functions or they can be anonymous functions (see the examples)

"""
```

```
fahrenheit2celsius (generic function with 2 methods)
```

```
begin
    function fahrenheit2celsius(f)
        cel = ((f - 32) / 9 ) * 5
        return cel
    end

    function fahrenheit2celsius(f::Float64)
        cel = ((f - 32) / 9 ) * 5
        return "In celsius it is $cel"
    end

end
```

```
37.77777777777778
```

```
# test the function
fahrenheit2celsius(100) # returns 37.78 degrees celsius
```

```
"In celsius it is 39.05555555555556"
```

```
# test the function again this time with decimal point
fahrenheit2celsius(102.3) # returns 39.05
```

Multiple dispatch

The above two examples are those of multiple dispatch. You can see that our function can take either integer values or decimal point values (referred to as floats). So depending on whether you will input integer (as we did with 100) or with floating point number (as we did with 102.3), you will or you may see different outputs. You can of course produce the same style of output regardless of what format the user inputs the data, but you have this flexibility. This is referred to as multiple dispatch in Julia.

```
md"""
## Multiple dispatch

The above two examples are those of multiple dispatch. You can see that our function can take either integer values or decimal point values (referred to as floats). So depending on whether you will input integer (as we did with 100) or with floating point number (as we did with 102.3), you will or you may see different outputs. You can of course produce the same style of output regardless of what format the user inputs the data, but you have this flexibility. This is referred to as multiple dispatch in Julia.

"""
```

Iterate functions over a vector

- Concept of anonymous function
- Map() function
- do...end statement

```
md"""

## Iterate functions over a vector

- Concept of anonymous function
- Map() function
- do...end statement

"""
```

```
[2, 4, 6]
```

```
begin
    # take a vector x8
    x8 = [1,2,3]
    # and define a function that doubles a value
    function doublevalue(x)
        return x * 2
    end
    # you can map doublevalue() over x8 thus to return another vector x9
    x9 = map(doublevalue, x8) # will produce a vector 2,4,6
    # So mapping of a function is you define a function, then
    # use map() where the function is the first argument,
    # and the vector is a second argument

end
```

Concept of an anonymous function and map()

In an anonymous function, you do not have to name the function as such, but write the function statements and then proceed to apply as before.

```
md"""
## Concept of an anonymous function and map()

In an anonymous function, you do not have to name the function as such, but write the function statements and then proceed to apply as before.
"""
```

```
[2, 4, 6]
```

```
begin
    # let's take a similar vector x10
    x10 = [1,2,3]
    # then apply an anonymous function to double the elements
    # and return another vector x11 to contain 2,4,6
    # so we do
    x11 = map(x -> x * 2, x10)
end

```

Note that in the above statement, we did not name the function at all. Instead, we had only one argument that of x, and the right arrow pointed to the logic or statement or expression of the function and we did not have a return statement either. We then applied this function to the array or vector x10 in this case, and it returned another vector. You will meet this function named `map()` and its various forms when we will be manipulating data frames and doing data transformations.

```
md"""
Note that in the above statement, we did not name the function at all. Instead, we had only one argument that of x, and the right arrow pointed to the logic or statement or expression of the function and we did not have a return statement either. We then applied this function to the array or vector x10 in this case, and it returned another vector. You will meet this function named `map()` and its various forms when we will be manipulating data frames and doing data transformations.
"""
```

Concept of do...end

This time, we can map a longer function that we define to a vector or an array and return an expression. We start with the array, then use the expression do...end

```
md"""
## Concept of do...end
This time, we can map a longer function that we define to a vector or an array and return an expression. We start with the array, then use the expression do...end
"""
```

```
[37.7778, 38.3333, 38.8889]
```

```
begin
    # let's take an array
    x12 = [100, 101, 102] # like fahrenheit temperature
    # and apply fahrenheit to celsius by writing it out
    x13 = map(x12) do x
        cel = ((x - 32) / 9 ) * 5
        return cel

    end
end

```

In this case, start with `map()` function. Place an array nside the map function. Then start an anonymous function starting with do and write the function as you'd normally write. End the 'do' function with an 'end' statement. As you can see, with this statement, we start with x12 as an array of 100, 101, and 102 fahreneheit temperature and then wrote a converter function from fahrenheit to celsius and it returned the temperatures in celsius.

```
md"""

In this case, start with `map()` function. Place an array nside the map function. Then start an anonymous function starting with do and write the function as you'd normally write. End the 'do' function with an 'end' statement. As you can see, with this statement, we start with x12 as an array of 100, 101, and 102 fahreneheit temperature and then wrote a converter function from fahrenheit to celsius and it returned the temperatures in celsius.

"""
```

Data manipulation and simple data analyses

The concepts we have covered in the previous "chops" will help you to generally work in Julia well. There are many other concepts, some of which we will cover as we go along. In this section, we will cover the following:

- Read data into data frames from csv files
- Visualise data frames and create tables
- Transform data tables
- Visualise data tables and create visualisations
- Model relationships

```
md"""

## Data manipulation and simple data analyses

The concepts we have covered in the previous "chops" will help you to generally work in Julia well. There are many other concepts, some of which we will cover as we go along. In this section, we will cover the following:

- Read data into data frames from csv files
- Visualise data frames and create tables
- Transform data tables
- Visualise data tables and create visualisations
- Model relationships

"""
```

	age	sex	hemoglobin
1	12	"F"	5.5
2	13	"M"	6.1
3	14	"M"	4.5
4	15	"M"	6.2
5	16	"M"	5.1
6	17	"F"	5.4
7	18	"F"	5.5

```
begin
    # using DataFrames package, you can create data frames from arrays
    # given three arrays
    age = [12,13,14,15,16,17,18]
    sex = ["F","M","M","M","M","F", "F"]
    hba1c = [5.5, 6.1, 4.5, 6.2, 5.1, 5.4,5.5]
    # you can create a data frame as follows
    df = DataFrame("age" => age,
    "sex" => sex,
    "hemoglobin" => hba1c)
    # visualise the data set
    df
end
```

	sex	Count
1	"F"	3
2	"M"	4

```
begin
    # find the mean and sd for age
    mean_age = mean(df.age) # use the Statistics package to do
    statistics
    mean_hemo = mean(df.hemoglobin)
    gender = combine(groupby(df, :sex), nrow => :Count)

end
```

Most statistical functions are straightforward. When you want to find out the tables of data, such as sex count, you:

- First group the data set with your preferred variable
- In this case, our data set is df, and our variable is sex
- We use a "symbol" that points to the variable and prefix with :
- If we do groupby(dataset, :variablename) then the data set is divided into
- as many segments as we have with the variable levels
- then we put them back with the combine() function and indicate what we want
- in this case, we have asked for how many row counts there are nrow()

```
md"""
Most statistical functions are straightforward.
When you want to find out the tables of data, such as sex count, you:

- First group the data set with your preferred variable
- In this case, our data set is df, and our variable is sex
- We use a "symbol" that points to the variable and prefix with :
- If we do groupby(dataset, :variablename) then the data set is divided
into
- as many segments as we have with the variable levels
- then we put them back with the combine() function and indicate what
we want
- in this case, we have asked for how many row counts there are nrow()
"""
```

	date	location_key	cumulative_confirmed	cumulative_deceased
1	2022-09-11	"ZW"	256888	5596
2	2022-09-12	"ZW"	256939	5596
3	2022-09-13	"ZW"	256939	5596

```
begin
    # if you can locate a CSV file you can read that file and
    manipulate with
    # DataFrames as well as in
    # file: https://storage.googleapis.com/covid19-open-
    data/v3/epidemiology.csv
    url = "https://storage.googleapis.com/covid19-open-
    data/v3/epidemiology.csv"
    covid = CSV.read(download(url), DataFrame)
    # download the url
    # read the data file using CSV
    # store the data in DataFrame
    first(covid, 3) # read the first three rows
    last(covid, 3) # read the last three rows
    # select date, location, cumulative_confirmed and
    cumulative_deceased
    covid2 = select(covid, :date, :location_key, :cumulative_confirmed,
    :cumulative_deceased)
    last(covid2, 3) # see the last three rows
end
```

	date	location_key	cumulative_confirmed	cumulative_deceased	log
1	2020-02-26	"NZ"	1	0	0.0
2	2020-02-27	"NZ"	1	0	0.0
3	2020-02-28	"NZ"	1	0	0.0

```
begin
    # select date, location, cumulative_confirmed and
    cumulative_deceased
    covid3 = select(covid, :date, :location_key, :cumulative_confirmed,
    :cumulative_deceased)
    last(covid3, 3) # see the last three rows
    # filter "NZ" from the location key
    covidnz = filter(:location_key => (x -> x == "NZ"), covid3)
     # Now filter cumulative_confirmed > 0
    covidnz2 = filter(:cumulative_confirmed => (x -> x > 0), covidnz)
    first(covidnz2, 3)
    # Now convert the cumulative confirmed cases to log values
    covidnz3 = transform(covidnz2, :cumulative_confirmed => ByRow(x ->
    log(x)) => :logconfirmed)
    first(covidnz3, 3) # log transformed values of covid cases

end
```

Visualise data

```
md"""
## Visualise data
"""
```

```
begin
    # Visualise covid cases in NZ
    fig2, ax2, plt2 = lines(covidnz3.logconfirmed)
    ax2.xlabel = "Days since first case"
    ax2.ylabel = "Cumulative cases"
    ax2.title = "Total number of Covid cases in NZ since the beginning"
    fig2
end
```

We will now apply these principles to learn Epidemiology

- Prevalence
- Incidence
- Rates
- Ratios
- Standardised mortality or morbidity ratios
- Analysis of two by two tables
- Odds Ratios and Rate ratios
- Attributable risks
- Population attributable risks
- Survival analysis
- Cox Proportional hazards
- Logistic regression of Case control studies
- Time series analysis of ecological studies
- Field epidemiology investigations
- Sample size and power calculations

```
md"""

## We will now apply these principles to learn Epidemiology
- Prevalence
- Incidence
- Rates
- Ratios
- Standardised mortality or morbidity ratios
- Analysis of two by two tables
- Odds Ratios and Rate ratios
- Attributable risks
- Population attributable risks
- Survival analysis
- Cox Proportional hazards
- Logistic regression of Case control studies
- Time series analysis of ecological studies
- Field epidemiology investigations
- Sample size and power calculations

```

```
1 Enter cell code...
```

```
1 Enter cell code...
```

```
1 Enter cell code...
```

```
1 Enter cell code...
```

```
1 Enter cell code...
```

```
1 Enter cell code...
```

```
1 Enter cell code...
```

```
1 Enter cell code...
```

```
1 Enter cell code...
```

```
1 Enter cell code...
```

```
1 Enter cell code...
```

```
1 Enter cell code...
```

```
1 Enter cell code...
```

```
1 Enter cell code...
```

```
1 Enter cell code...
```

```
1 Enter cell code...
```

```
1 Enter cell code...
```

```
1 Enter cell code...
```

```
1 Enter cell code...
```

```
1 Enter cell code...
```

This instance of Pluto runs on JuliaHub!

🚀 A static version of this notebook is **automagically** deployed every time you save & run!

You can extend the time limit of the job serving this pluto notebook by clicking below:

1hour 53 minutes left
extend by 1 hour

Click on the JuliaHub logo to toggle the visibility of this control panel!