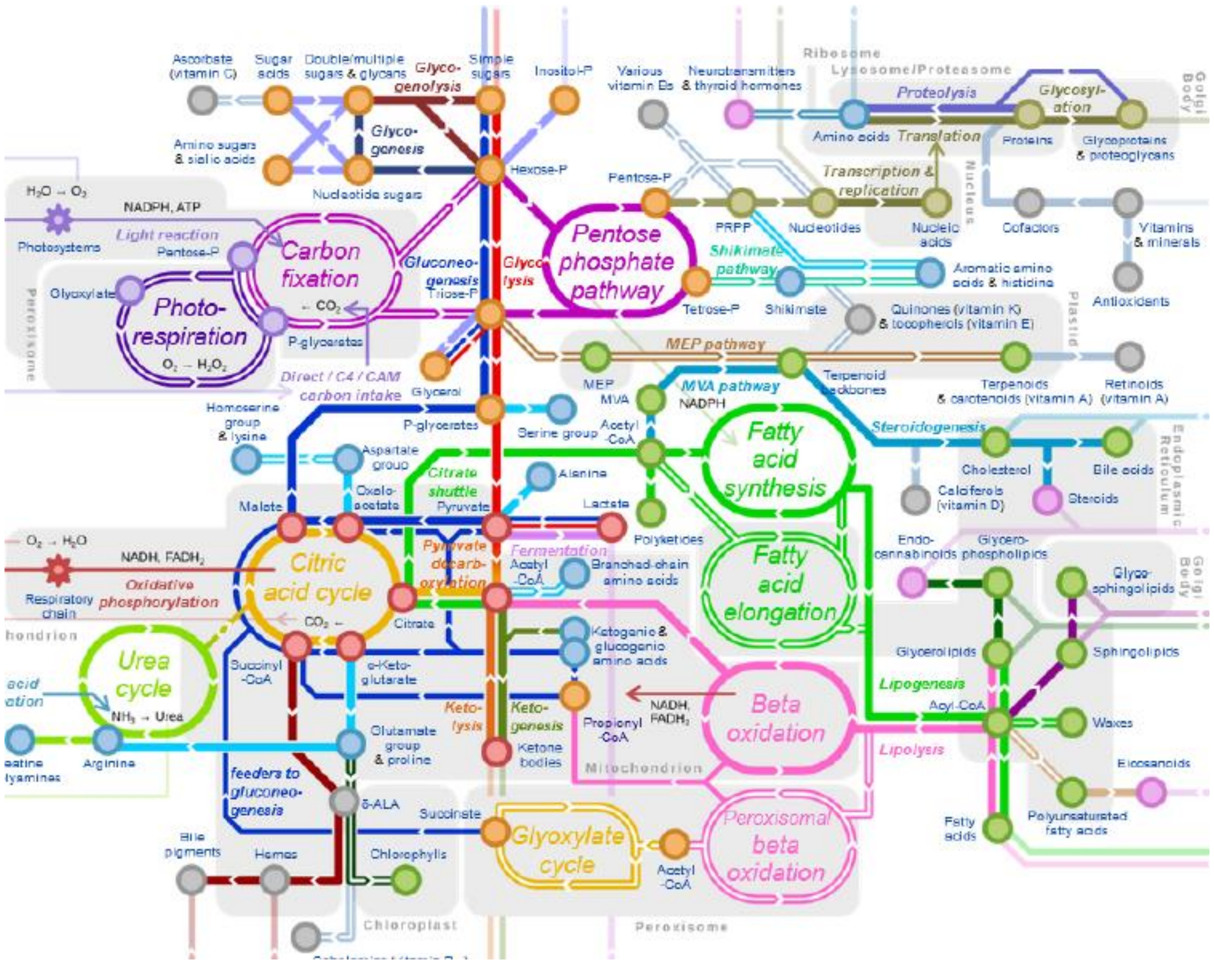
Business Intelligence Workspace Project 1

Classification of the trajectories obtained from mechanistic model of the genetic disease (DATA SET 2)



Submitted by: *Hammad Ullah*

Submitted to: *Piotr Nyczka*

Date: *7-April-2022*

Background:

Metabolism is one of the main requirements for a system in a living organism to be considered alive. The metabolic network itself is the center of the molecular layer of life. In other words, it drives the living cells by providing energy and essential materials needed to survive. Metabolites are intermediate or product of metabolism that are transformed by reactions and activated by genes. The metabolic network is a very complex network consisting of such metabolites and cyclic processes. However, it can be simplified into a toy model with conditional gates AND and OR as shown in figure 1 below.

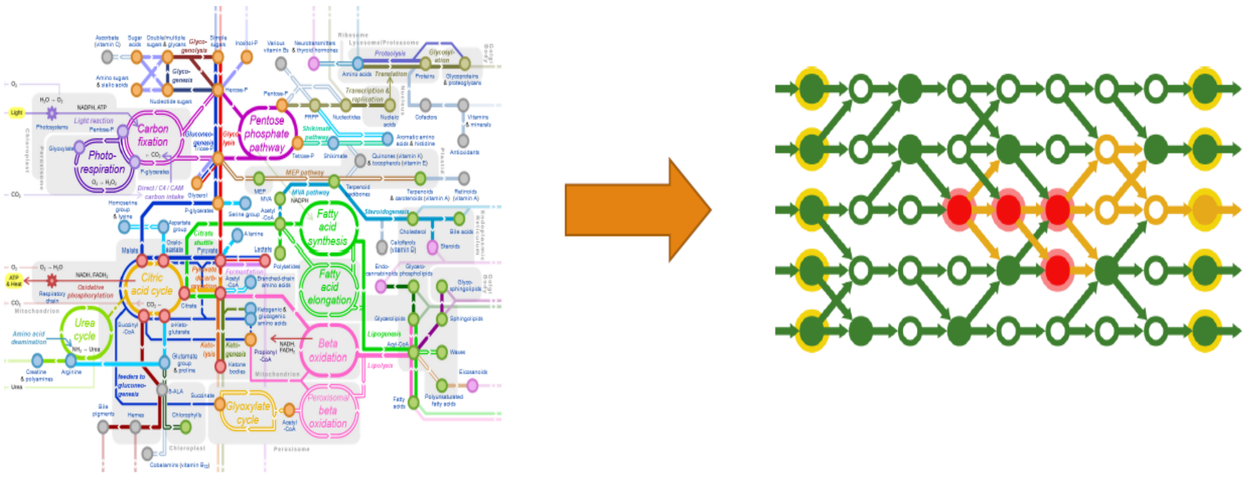


Figure 1 Simplification of Metabolic Network

Experimental Design Setup:

The experiment is set up in such a way where we have two copies of the same metabolic network: one is healthy whereas the other one is damaged. Both the networks are fed with the same output and the outputs are being read and classified into one of the four categories i.e. A, B, C and D. In order to obtain the timeseries of the experiment, the initial inputs are randomized a little and fed into the networks. The following figure shows us the experimental setup as well as what each category signifies.

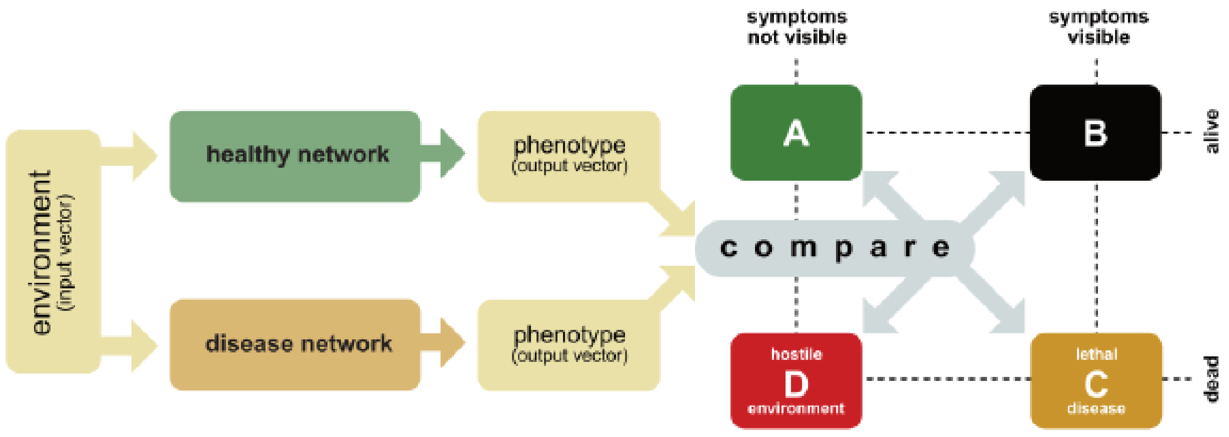


Figure 2 Experimental Setup

Task at hand:

We have been provided a total of 20,000 files with 1000 rows each. There are many parameters provided in the file itself as well as in the filename. The parameters and inputs of interests are:

- L – system's length
- H – system's height
- Bool – concentration of ANDs
- b – branching
- n – number of damaged genes
- p – clustering parameter of damage
- switch_upp – average input strength
- inputs_fraction – effective input strength
- outputs_fraction_hlt – effective output strength of the healthy system
- outputs_fraction_dis – effective output strength of the damaged system

Based on the parameters and inputs, each row in a file is assigned a state from one of the A, B, C or D states. The conditions for assigning these states are as such:

- If **outputs_fraction_hlt = outputs_fraction_dis** and both are nonzero then the state is **A**
- If **outputs_fraction_hlt =/= outputs_fraction_dis** and both are nonzero then the state is **B**
- If **outputs_fraction_hlt =/= outputs_fraction_dis** and only **outputs_fraction_hlt** is nonzero then the state is **C**
- If both are zero then the state is **D**

After assigning states to each row, the 1000 rows are compacted into 1 row where averages of the parameters and inputs are taken as well as one trajectory class is assigned to that row.

Trajectory classes are based on the presence or absence of a given state in the trajectory of interest. For example, trajectory containing all the states would be ABCD, while containing only A state would be just A, and oscillation between A and B will belong to class AB. There are a total of 15 trajectory classes possible, which are:

1. A
2. B
3. C
4. D
5. AB
6. AC

7. AD
8. BC
9. BD
10. CD
11. ABC
12. ACD
13. BCD
14. ABD
15. ABCD

Once, the trajectory class has been assigned to that row, we move to another file. This way, 20,000 rows are created, each being assigned to one of the 15 trajectory classes. These 20 thousand rows are then stored in a new compacted file which is used to visualize and analyze the results.

Code:

```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statistics as stat
```

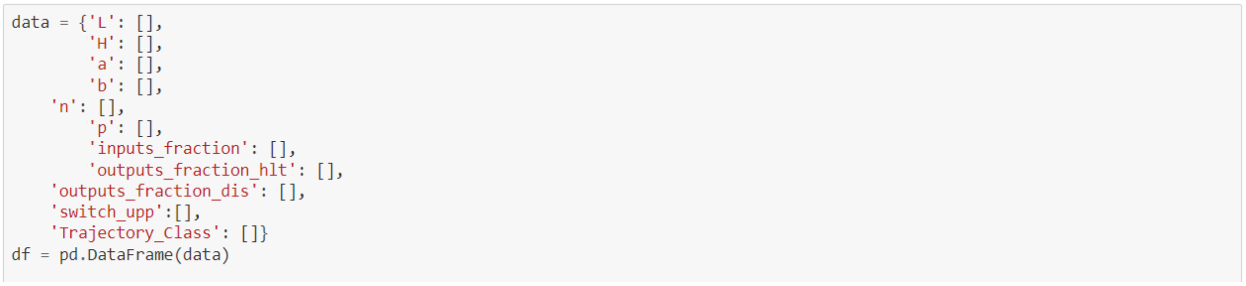
- First, the required libraries are imported. 'os' for changing directories and getting files from a certain directory. 'matplotlib.pyplot' for plotting in the later stages and 'statistics' for computing averages etc.

```
os.chdir("C:/Users/hamma/Desktop/WUST/Business Intelligence Workplace/Project/data")
```

- Using os library, the working directory has been changed to the folder with the data files.

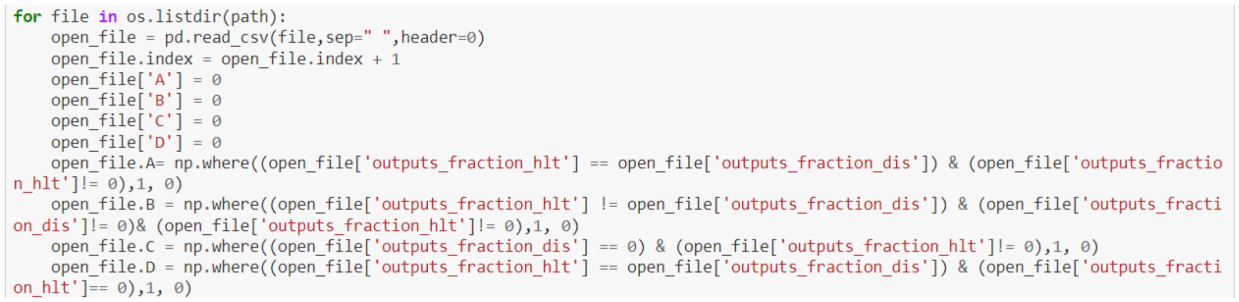
```
path ="C:/Users/hamma/Desktop/WUST/Business Intelligence Workplace/Project/data"
```

- A variable named 'path' has been assigned to the current working directory. This shall be used later to access files from using the os.listdir function.



```
data = {'L': [],
        'H': [],
        'a': [],
        'b': [],
    'n': [],
        'p': [],
        'inputs_fraction': [],
        'outputs_fraction_hlt': [],
    'outputs_fraction_dis': [],
    'switch_upp':[],
    'Trajectory_Class': []}
df = pd.DataFrame(data)
```

- An empty dictionary by the name of 'data' has been created with the column names. This dictionary is then turned into a dataframe using the pd.DataFrame function from the pandas library.

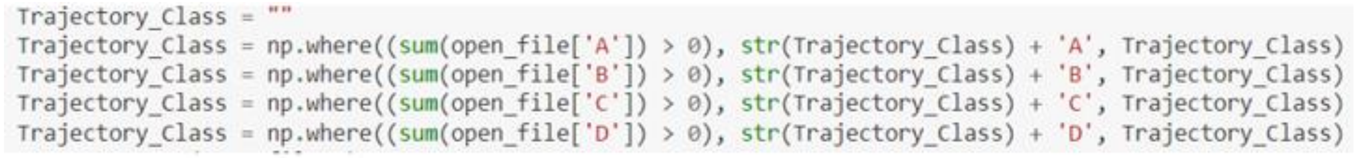


```
for file in os.listdir(path):
    open_file = pd.read_csv(file,sep=" ",header=0)
    open_file.index = open_file.index + 1
    open_file['A'] = 0
    open_file['B'] = 0
    open_file['C'] = 0
    open_file['D'] = 0
    open_file.A= np.where((open_file['outputs_fraction_hlt'] == open_file['outputs_fraction_dis']) & (open_file['outputs_fractio
n_hlt']!= 0),1, 0)
    open_file.B = np.where((open_file['outputs_fraction_hlt'] != open_file['outputs_fraction_dis']) & (open_file['outputs_fracti
on_dis']!= 0)& (open_file['outputs_fraction_hlt']!= 0),1, 0)
    open_file.C = np.where((open_file['outputs_fraction_dis'] == 0) & (open_file['outputs_fraction_hlt']!= 0),1, 0)
    open_file.D = np.where((open_file['outputs_fraction_hlt'] == open_file['outputs_fraction_dis']) & (open_file['outputs_fracti
on_hlt']== 0),1, 0)
```

- Here, a for loop has been used to access the files in the 'path'. The listdir() function from the os library only takes the name of the file so we must open it as well in the loop before we can perform any actions on the data in the file.

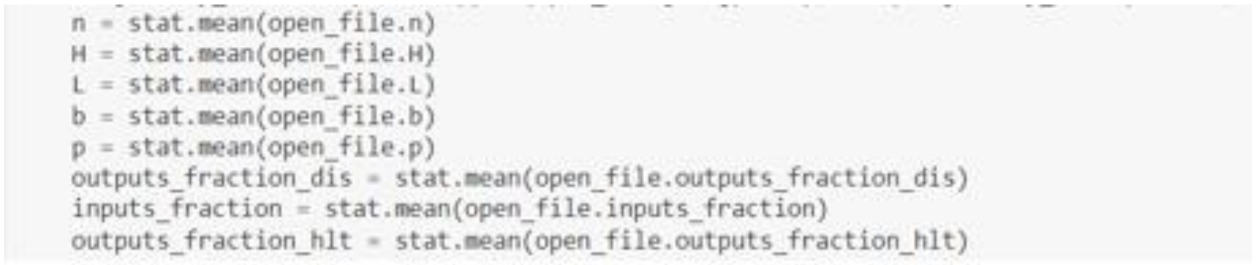
- After opening the file, I created 4 new columns (A, B, C and D) in the file and assigned values of zero to every entry of that column. This done for assigning trajectory states to each row.

- After creating these columns, the np.where() function is used to go through every row and assign a value of 1 in the corresponding column if it satisfies the condition stated in the function, and a value of 0 if the condition is not satisfied. Thus, a value of 1 appears in one of the 4 columns of the 4 states depending on which condition is satisfied and which is not. The rest of the values in the three remaining columns are zero.



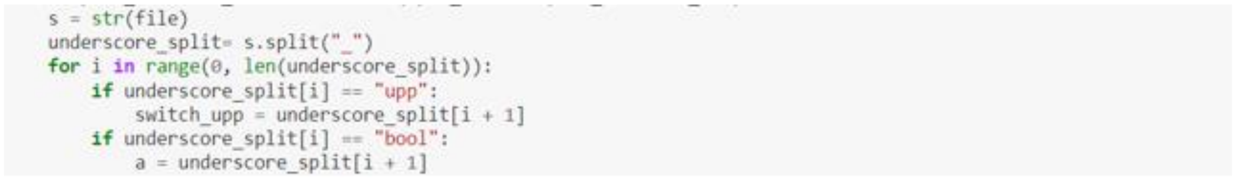
```python
Trajectory_Class = ""
Trajectory_Class = np.where((sum(open_file['A']) > 0), str(Trajectory_Class) + 'A', Trajectory_Class)
Trajectory_Class = np.where((sum(open_file['B']) > 0), str(Trajectory_Class) + 'B', Trajectory_Class)
Trajectory_Class = np.where((sum(open_file['C']) > 0), str(Trajectory_Class) + 'C', Trajectory_Class)
Trajectory_Class = np.where((sum(open_file['D']) > 0), str(Trajectory_Class) + 'D', Trajectory_Class)
```

- Here the np.where() function is used again to add strings to Trajectory_Class variable. Basically, as we said before trajectory classes are based on the presence or absence of a given state in the trajectory of interest, we sum every column (A, B, C and D) and see if the sum of the columns is greater than zero. It checks for the sum of all 4 columns one by one if the sum is greater than 0, then it adds the string of that trajectory ('A', 'B', 'C', 'D') to the empty string by the name: 'Trajectory_Class'. Thus, using all the 1000 rows in the file, we assigned one trajectory class to the trajectory file.



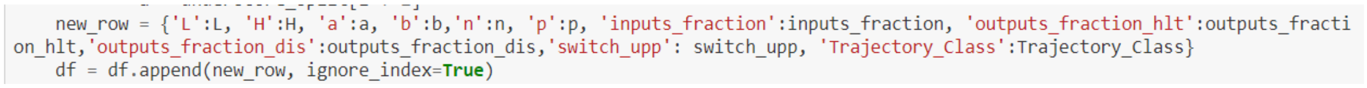
```python
    n = stat.mean(open_file.n)
    H = stat.mean(open_file.H)
    L = stat.mean(open_file.L)
    b = stat.mean(open_file.b)
    p = stat.mean(open_file.p)
    outputs_fraction_dis = stat.mean(open_file.outputs_fraction_dis)
    inputs_fraction = stat.mean(open_file.inputs_fraction)
    outputs_fraction_hlt = stat.mean(open_file.outputs_fraction_hlt)
```

- Here the averages of the results of interests and parameters have been taken. I realize that the parameters do not change within each trajectory file but getting the parameters this way was a more doable option at the time.



```python
    s = str(file)
    underscore_split= s.split("_")
    for i in range(0, len(underscore_split)):
        if underscore_split[i] == "upp":
            switch_upp = underscore_split[i + 1]
        if underscore_split[i] == "bool":
            a = underscore_split[i + 1]
```

- As mentioned before, some of the parameters needed to be taken from the file's name. Here the file's name has been split by underscores and using indexing, the relevant parameters (bool and upp) are extracted and assigned to variables.



```python
    new_row = {'L':L, 'H':H, 'a':a, 'b':b,'n':n, 'p':p, 'inputs_fraction':inputs_fraction, 'outputs_fraction_hlt':outputs_fracti
on_hlt,'outputs_fraction_dis':outputs_fraction_dis,'switch_upp': switch_upp, 'Trajectory_Class':Trajectory_Class}
    df = df.append(new_row, ignore_index=True)
```

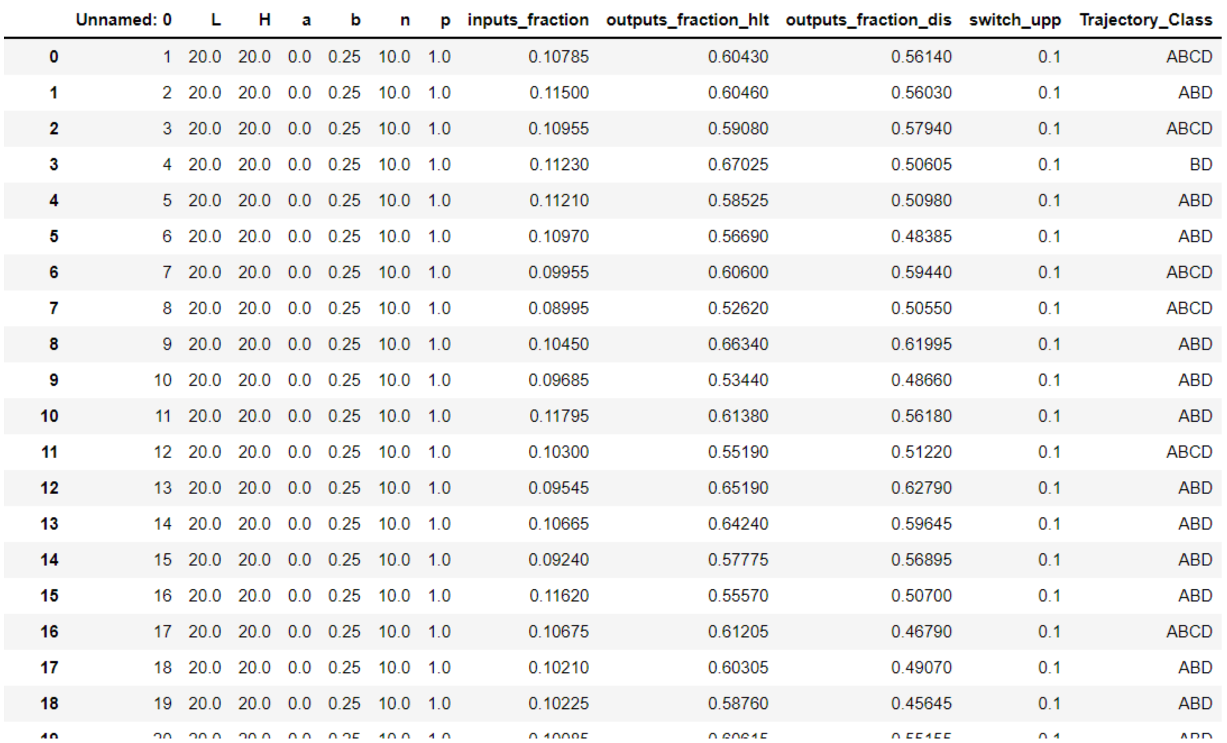
- Now, we have all the parameters so I added them all in a single row and added them to the dataframe created before the loop.
- The loop goes through all 20000 files and keeps adding rows to the dataframe

```
df.to_csv('PROJECT.csv')
```

```
proj_data = pd.read_csv('PROJECT.csv',header=0)
```

```
data_frame_from_csv = pd.DataFrame(proj_data)
```

- ➢ Once the loop finishes, the dataframe is converted to a csv file by the name of 'Project.csv'
- ➢ Since, we shall be using this data to make data visualization, the file 'Project.csv' is opened and turned into a new dataframe.
- ➢ The following is a preview of the dataframe:



	Unnamed: 0	L	H	a	b	n	p	inputs_fraction	outputs_fraction_hlt	outputs_fraction_dis	switch_upp	Trajectory_Class
0	1	20.0	20.0	0.0	0.25	10.0	1.0	0.10785	0.60430	0.56140	0.1	ABCD
1	2	20.0	20.0	0.0	0.25	10.0	1.0	0.11500	0.60460	0.56030	0.1	ABD
2	3	20.0	20.0	0.0	0.25	10.0	1.0	0.10955	0.59080	0.57940	0.1	ABCD
3	4	20.0	20.0	0.0	0.25	10.0	1.0	0.11230	0.67025	0.50605	0.1	BD
4	5	20.0	20.0	0.0	0.25	10.0	1.0	0.11210	0.58525	0.50980	0.1	ABD
5	6	20.0	20.0	0.0	0.25	10.0	1.0	0.10970	0.56690	0.48385	0.1	ABD
6	7	20.0	20.0	0.0	0.25	10.0	1.0	0.09955	0.60600	0.59440	0.1	ABCD
7	8	20.0	20.0	0.0	0.25	10.0	1.0	0.08995	0.52620	0.50550	0.1	ABCD
8	9	20.0	20.0	0.0	0.25	10.0	1.0	0.10450	0.66340	0.61995	0.1	ABD
9	10	20.0	20.0	0.0	0.25	10.0	1.0	0.09685	0.53440	0.48660	0.1	ABD
10	11	20.0	20.0	0.0	0.25	10.0	1.0	0.11795	0.61380	0.56180	0.1	ABD
11	12	20.0	20.0	0.0	0.25	10.0	1.0	0.10300	0.55190	0.51220	0.1	ABCD
12	13	20.0	20.0	0.0	0.25	10.0	1.0	0.09545	0.65190	0.62790	0.1	ABD
13	14	20.0	20.0	0.0	0.25	10.0	1.0	0.10665	0.64240	0.59645	0.1	ABD
14	15	20.0	20.0	0.0	0.25	10.0	1.0	0.09240	0.57775	0.56895	0.1	ABD
15	16	20.0	20.0	0.0	0.25	10.0	1.0	0.11620	0.55570	0.50700	0.1	ABD
16	17	20.0	20.0	0.0	0.25	10.0	1.0	0.10675	0.61205	0.46790	0.1	ABCD
17	18	20.0	20.0	0.0	0.25	10.0	1.0	0.10210	0.60305	0.49070	0.1	ABD
18	19	20.0	20.0	0.0	0.25	10.0	1.0	0.10225	0.58760	0.45645	0.1	ABD
19	20	20.0	20.0	0.0	0.25	10.0	1.0	0.10085	0.60615	0.55155	0.1	ABD

```
data_frame_from_csv['Trajectory_Class_plotting'] = data_frame_from_csv.loc[:, 'Trajectory_Class']
```

```
categ = pd.pivot_table(data_frame_from_csv, values = 'Trajectory_Class_plotting', index=['a','b','switch_upp'], columns = 'Traje
ctory_Class', aggfunc='count').reset_index()
categorization = categ.replace(np.nan, 0)
```

- ➢ A duplicate of the Trajectory_Clas column is created. It is created because in the next line, during pivoting, I had a hard time assigning columns and values to the same column.
- ➢ During pivoting, some parameters were fixed (a, b) and the trajectory classes are counted for each of the fixed parameters. And where there are no values, the entry is replaced by 0.

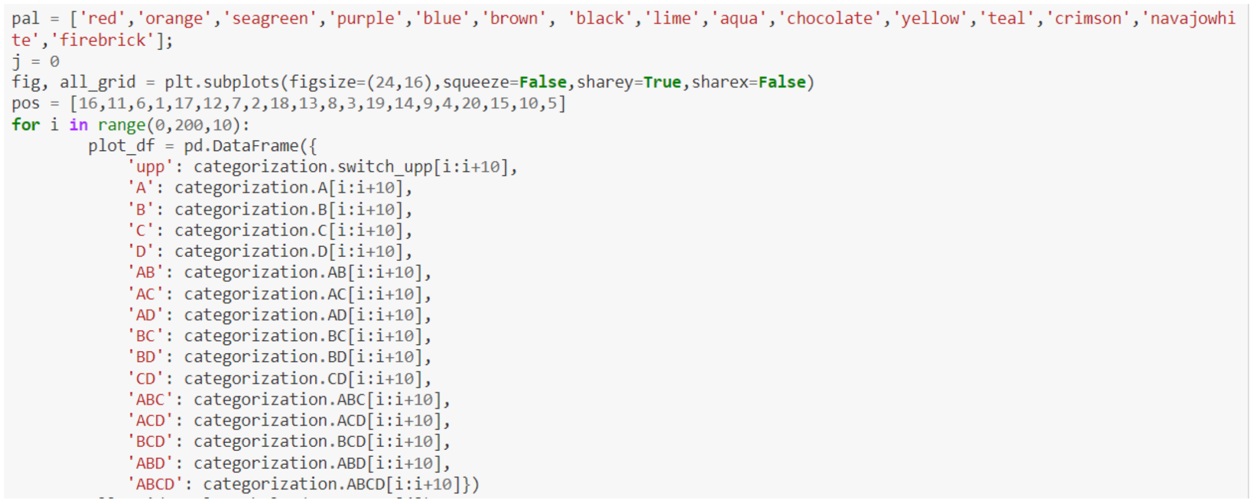
- The pivoted dataframe is of 200 rows with values of the fixed parameters changing every 10 rows.
- A preview of the pivoted dataframe is shown below:

Trajectory_Class	a	b	switch_upp	A	AB	ABC	ABCD	ABD	AC	ACD	AD	B	BC	BCD	BD	C	CD	D
0	0.0	0.25	0.1	0.0	0.0	0.0	40.0	57.0	0.0	0.0	2.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
1	0.0	0.25	0.2	0.0	0.0	0.0	31.0	67.0	0.0	0.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.25	0.3	0.0	49.0	15.0	6.0	29.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.25	0.4	2.0	90.0	4.0	0.0	3.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.25	0.5	2.0	86.0	0.0	0.0	0.0	0.0	0.0	0.0	12.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.25	0.6	6.0	63.0	0.0	0.0	0.0	0.0	0.0	0.0	31.0	0.0	0.0	0.0	0.0	0.0	0.0
6	0.0	0.25	0.7	17.0	42.0	0.0	0.0	0.0	0.0	0.0	0.0	41.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.25	0.8	35.0	24.0	0.0	0.0	0.0	0.0	0.0	0.0	41.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.25	0.9	45.0	16.0	0.0	0.0	0.0	0.0	0.0	0.0	39.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	0.25	1.0	51.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	49.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.50	0.1	0.0	0.0	0.0	26.0	63.0	0.0	0.0	9.0	0.0	0.0	0.0	2.0	0.0	0.0	0.0
11	0.0	0.50	0.2	0.0	0.0	0.0	24.0	57.0	0.0	0.0	16.0	0.0	0.0	0.0	3.0	0.0	0.0	0.0
12	0.0	0.50	0.3	6.0	44.0	3.0	6.0	27.0	0.0	0.0	3.0	8.0	0.0	0.0	3.0	0.0	0.0	0.0
13	0.0	0.50	0.4	14.0	65.0	0.0	0.0	2.0	0.0	0.0	0.0	18.0	0.0	0.0	1.0	0.0	0.0	0.0
14	0.0	0.50	0.5	30.0	38.0	0.0	0.0	0.0	0.0	0.0	0.0	32.0	0.0	0.0	0.0	0.0	0.0	0.0
15	0.0	0.50	0.6	42.0	28.0	0.0	0.0	0.0	0.0	0.0	0.0	30.0	0.0	0.0	0.0	0.0	0.0	0.0
16	0.0	0.50	0.7	60.0	12.0	0.0	0.0	0.0	0.0	0.0	0.0	28.0	0.0	0.0	0.0	0.0	0.0	0.0
17	0.0	0.50	0.8	66.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0	30.0	0.0	0.0	0.0	0.0	0.0	0.0
18	0.0	0.50	0.9	74.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	26.0	0.0	0.0	0.0	0.0	0.0	0.0
19	0.0	0.50	1.0	72.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	28.0	0.0	0.0	0.0	0.0	0.0	0.0

```
categorization[['A','B','C', 'D', 'AB', 'AC', 'AD', 'BC', 'BD','CD','ABC', 'ACD', 'BCD', 'ABD','ABCD']] = categorization
[['A','B','C', 'D', 'AB', 'AC', 'AD', 'BC', 'BD','CD','ABC', 'ACD', 'BCD', 'ABD','ABCD']].apply(lambda x: x/x.sum(), axis=1)
```

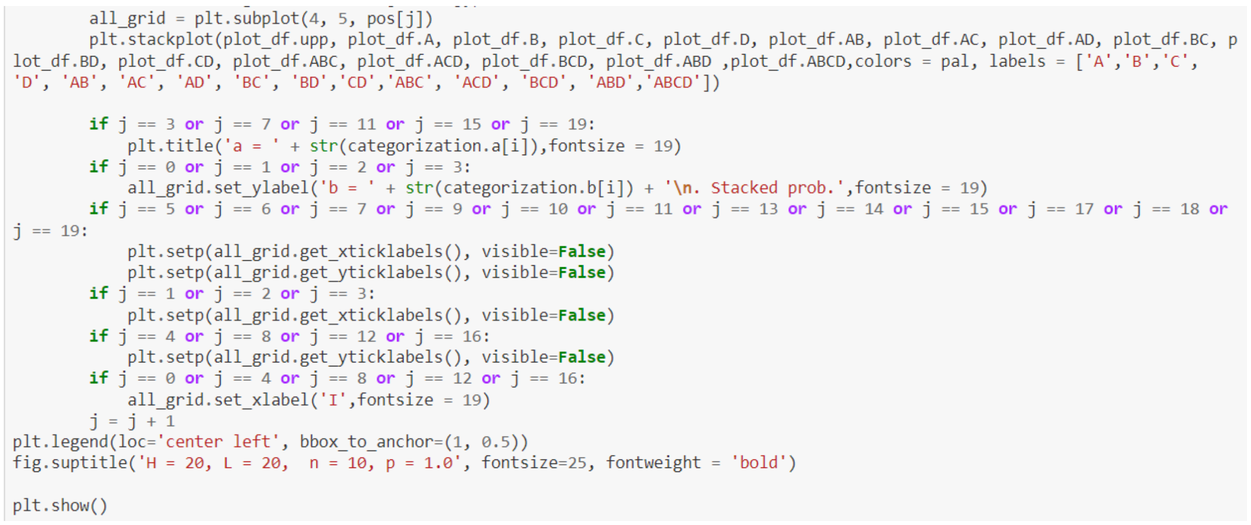
- Here, the counts of each trajectory class are converted into a percentage of the total counts of all the trajectory class. The data frame now looks like what is shown below:

Trajectory_Class	a	b	switch_upp	A	AB	ABC	ABCD	ABD	AC	ACD	AD	B	BC	BCD	BD	C	CD	D
0	0.0	0.25	0.1	0.00	0.00	0.00	0.40	0.57	0.0	0.00	0.02	0.00	0.0	0.00	0.01	0.00	0.00	0.00
1	0.0	0.25	0.2	0.00	0.00	0.00	0.31	0.67	0.0	0.00	0.02	0.00	0.0	0.00	0.00	0.00	0.00	0.00
2	0.0	0.25	0.3	0.00	0.49	0.15	0.06	0.29	0.0	0.00	0.00	0.01	0.0	0.00	0.00	0.00	0.00	0.00
3	0.0	0.25	0.4	0.02	0.90	0.04	0.00	0.03	0.0	0.00	0.00	0.01	0.0	0.00	0.00	0.00	0.00	0.00
4	0.0	0.25	0.5	0.02	0.86	0.00	0.00	0.00	0.0	0.00	0.00	0.12	0.0	0.00	0.00	0.00	0.00	0.00
5	0.0	0.25	0.6	0.06	0.63	0.00	0.00	0.00	0.0	0.00	0.00	0.31	0.0	0.00	0.00	0.00	0.00	0.00
6	0.0	0.25	0.7	0.17	0.42	0.00	0.00	0.00	0.0	0.00	0.00	0.41	0.0	0.00	0.00	0.00	0.00	0.00
7	0.0	0.25	0.8	0.35	0.24	0.00	0.00	0.00	0.0	0.00	0.00	0.41	0.0	0.00	0.00	0.00	0.00	0.00
8	0.0	0.25	0.9	0.45	0.16	0.00	0.00	0.00	0.0	0.00	0.00	0.39	0.0	0.00	0.00	0.00	0.00	0.00
9	0.0	0.25	1.0	0.51	0.00	0.00	0.00	0.00	0.0	0.00	0.00	0.49	0.0	0.00	0.00	0.00	0.00	0.00
10	0.0	0.50	0.1	0.00	0.00	0.00	0.26	0.63	0.0	0.00	0.09	0.00	0.0	0.00	0.02	0.00	0.00	0.00
11	0.0	0.50	0.2	0.00	0.00	0.00	0.24	0.57	0.0	0.00	0.16	0.00	0.0	0.00	0.03	0.00	0.00	0.00



```
pal = ['red','orange','seagreen','purple','blue','brown', 'black','lime','aqua','chocolate','yellow','teal','crimson','navajowhi
te','firebrick'];
j = 0
fig, all_grid = plt.subplots(figsize=(24,16),squeeze=False,sharey=True,sharex=False)
pos = [16,11,6,1,17,12,7,2,18,13,8,3,19,14,9,4,20,15,10,5]
for i in range(0,200,10):
        plot_df = pd.DataFrame({
            'upp': categorization.switch_upp[i:i+10],
            'A': categorization.A[i:i+10],
            'B': categorization.B[i:i+10],
            'C': categorization.C[i:i+10],
            'D': categorization.D[i:i+10],
            'AB': categorization.AB[i:i+10],
            'AC': categorization.AC[i:i+10],
            'AD': categorization.AD[i:i+10],
            'BC': categorization.BC[i:i+10],
            'BD': categorization.BD[i:i+10],
            'CD': categorization.CD[i:i+10],
            'ABC': categorization.ABC[i:i+10],
            'ACD': categorization.ACD[i:i+10],
            'BCD': categorization.BCD[i:i+10],
            'ABD': categorization.ABD[i:i+10],
            'ABCD': categorization.ABCD[i:i+10]})
```

- ➢ First of all, 15 colors have been chosen to accommodate the 15 different trajectory colors in a way where differentiation between the colors is easy.
- ➢ In the for loop, the value of i changes with a jump of 10 integers because as we mentioned before, the values of the fixed parameters change every 10 rows. In other words, one plot is being produced for every 10 rows.
- ➢ Inside the for loop, the first 10 values of each trajectory are taken and added to the dataframe by the name of 'plot_df'.
- ➢ The list by the name of 'pos' is made in order to plot each graph at the appropriate place.



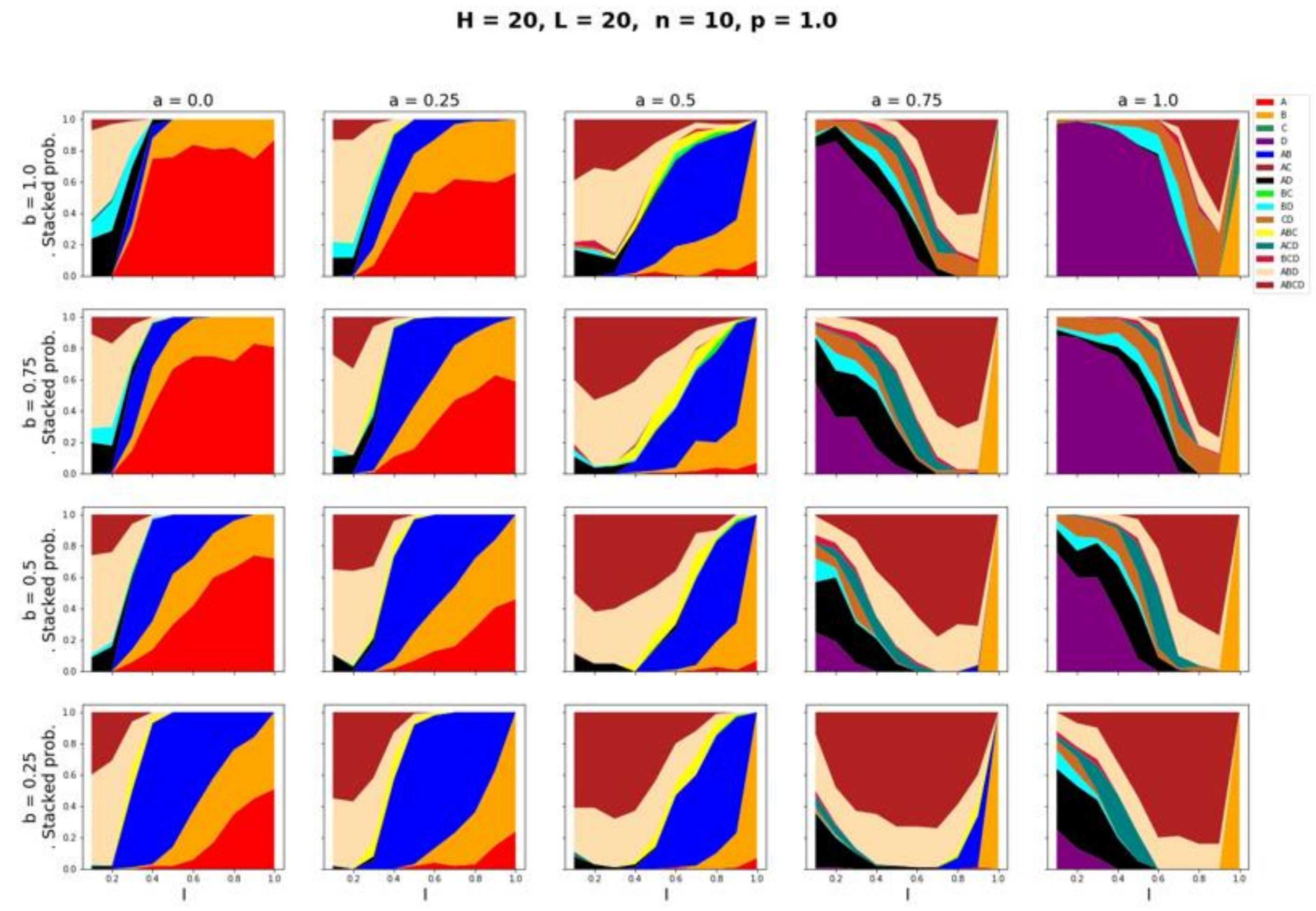
```
        all_grid = plt.subplot(4, 5, pos[j])
        plt.stackplot(plot_df.upp, plot_df.A, plot_df.B, plot_df.C, plot_df.D, plot_df.AB, plot_df.AC, plot_df.AD, plot_df.BC, p
lot_df.BD, plot_df.CD, plot_df.ABC, plot_df.ACD, plot_df.BCD, plot_df.ABD ,plot_df.ABCD,colors = pal, labels = ['A','B','C',
'D', 'AB', 'AC', 'AD', 'BC', 'BD','CD','ABC', 'ACD', 'BCD', 'ABD','ABCD'])

        if j == 3 or j == 7 or j == 11 or j == 15 or j == 19:
            plt.title('a = ' + str(categorization.a[i]),fontsize = 19)
        if j == 0 or j == 1 or j == 2 or j == 3:
            all_grid.set_ylabel('b = ' + str(categorization.b[i]) + '\n. Stacked prob.',fontsize = 19)
        if j == 5 or j == 6 or j == 7 or j == 9 or j == 10 or j == 11 or j == 13 or j == 14 or j == 15 or j == 17 or j == 18 or
j == 19:
            plt.setp(all_grid.get_xticklabels(), visible=False)
            plt.setp(all_grid.get_yticklabels(), visible=False)
        if j == 1 or j == 2 or j == 3:
            plt.setp(all_grid.get_xticklabels(), visible=False)
        if j == 4 or j == 8 or j == 12 or j == 16:
            plt.setp(all_grid.get_yticklabels(), visible=False)
        if j == 0 or j == 4 or j == 8 or j == 12 or j == 16:
            all_grid.set_xlabel('I',fontsize = 19)
        j = j + 1
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
fig.suptitle('H = 20, L = 20,  n = 10, p = 1.0', fontsize=25, fontweight = 'bold')

plt.show()
```

- ➢ After getting all the 10 values of each trajectory class, a stackplot is being plotted using pd.stackplot() function with x values of the 'upp' and y values of the trajectories.

- For some reason, the axis of the grid was not being shared amongst all the graphs, so I had to go over the respective graphs manually to remove their xtick labels and to add titles to each x and y axis.
- In the end the suptitle() is being added to the whole grid with specific font properties and the plot is being displayed.
- The plot being plotted is as follows:



Plot Analysis:

It is much more convenient to analyze the results when visualized as shown above. In order to understand the change in the plots as we go across or down the grid, we must see what each parameter represents. We are not going to take the fixed parameters (L, H, n and p) since they are constant in all graphs. The parameter ‘a’ and ‘b’ changes across and down the graph grid. Parameter ‘a’ represents the concentration of ANDs and parameter ‘b’ represents the extent of branching in the toy model. My analysis of the plots as much as I tried to understand them is that if the concentration of ANDs is constant and the branching value ‘b’ decreases, state A (alive and no symptoms), decreases as shown by the red stack in the stackplots. Moreover, if the concentration of ANDs is higher and constant, decreasing the branching parameter ‘b’, state D (dead and symptoms not visible), decreases as shown by the purple stack in the plot. Lastly, State B (alive and symptoms visible) decreases with the increase in the concentration of ANDs parameter which is ‘a’.