

# Data Puzzle Generator

VRAJ SHAH<sup>1</sup>

May 10, 2021

<sup>1</sup><http://18.189.170.47:5000/>

# 1

## Introduction

### 1.1 What is Data Puzzle Generator?

Data Puzzle Generator is a tool which helps professors to create data puzzles as per the rules provided. All the rules can be customized. This project is developed under guidance by Prof. [Tomasz Imielinski](#). This tool will enable professors to save around 70% of time.

### 1.2 What one can do with this tool?

You can generate the schema of your choice for the dataset. Also you can provide the rules and distributions using which you want the data to be generated. One can also provide derived attribute(s). In that case, users will have to provide `.py` file which defines all the derived attributes. And at the last data will be generated of given length.

2

## Flowchart

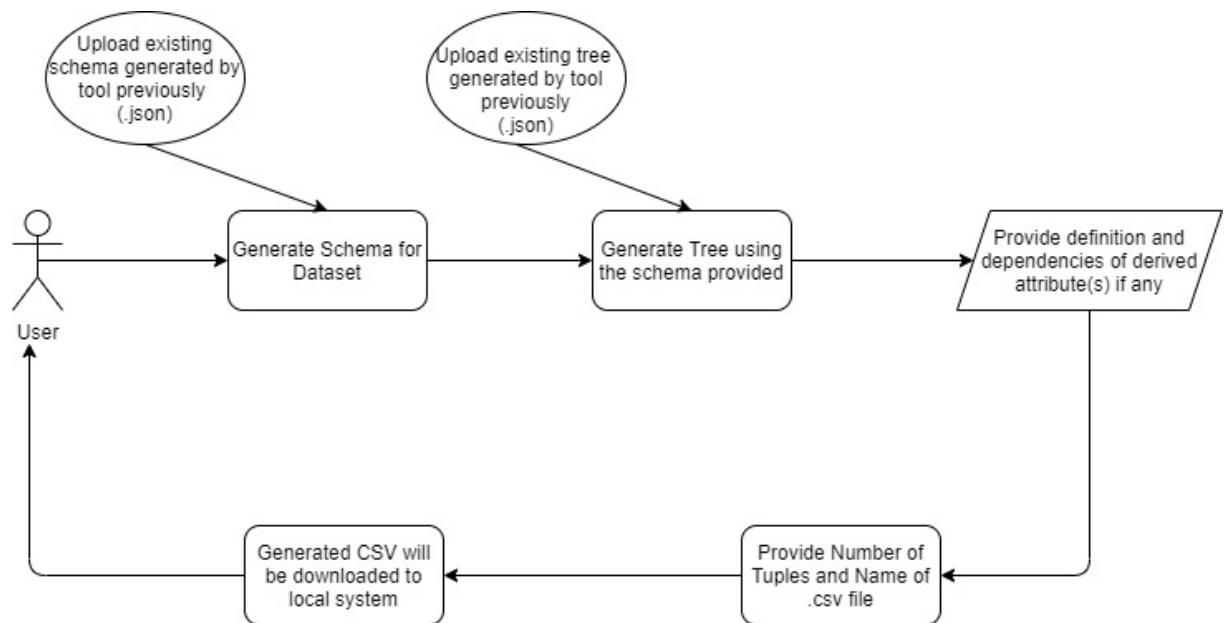


Figure 2.1: Flowchart of the system

## 3

# Generating Schema

The first step to generate dataset is to generate schema. If you are using the tool for the first time then you should consider to generate schema from scratch. You can add as many attributes as you want. Following fields will be asked for each attribute.

1. Name of the Attribute
2. Type of the Attribute (i.e. Categorical or Numerical)
3. Derived? (Is this attribute derived or base?)
4. Values of this Attribute.
  - (a) If Base Categorical - Domain of Attribute
  - (b) If Base Numerical - Lower and Upper value
  - (c) If Derived Categorical - Domain of Attribute
  - (d) If Derived Numerical - Nothing. It will be calculated at run time.

Once you have added all the attributes, you can download the entire schema by clicking on "Download Schema" button. It will save equivalent *".json"* file in your local system.

This feature increases the reusability. You can upload this downloaded *".json"* file, and system will render all the attributes you added previously. You can also modify this schema.

Screenshots can be found in subsequent part of this chapter.

### 3.1 Generating Schema from Scratch

---

**Design your schema from scratch or upload the schema that you have!**

Upload existing schema file:  No file chosen

---

Name of the Attribute:

Type of the Attribute: ☒ Numerical ☐ Categorical

Derived?: ☒ No ☐ Yes

Values:

---

Figure 3.1: Adding New Attribute to Schema

You can fill out all the details for the attribute. By clicking on "Add Attribute", you can add a new attribute for the schema. Once you have added all the attributes, you would like to save it for later use by clicking on "Download Schema". It will save ".json" file in your local machine.

The format of .json file would look something like following.

```

1 { "base": [{
2   "attributeName": "Bride age",
3   "attributeType": "numerical",
4   "values": ["18", "60"]
5 }],
6 "derived": [{
7   "attributeName": "Age diff",
8   "attributeType": "numerical",
9   "values": ["", ""]
10 }]
11 }
```

This is example of schema where only one base attribute and one derived attribute is added.

## 3.2 Generating Schema from JSON

**Design your schema from scratch or upload the schema that you have!**

Upload existing schema file:  DerivedAttributes.json

---

Name of the Attribute:

Type of the Attribute: ☒ Numerical ☐ Categorical

Derived?: ☒ No ☐ Yes

Values:

---

Name of the Attribute:

Type of the Attribute: ☒ Numerical ☐ Categorical

Derived?: ☒ No ☐ Yes

Values:

Figure 3.2: Uploading Previously Generated Schema

As we can see in 3.2, once you upload .json file, it will generate all the attributes you added previously. You can further modify it as per requirements and also save the updated version.

This feature is very useful if you just want to use previously generate schema with some minor changes.

## 3.3 Proceed to Generate Rules

Now that we have generated our Schema, we would like to add rules using which our data will be generated. You can click on "Submit" to navigate to the next page. On clicking submit, it will perform all the checks. Should there be any errors, user will have to correct it before proceeding further.

### 3.3.1 API

- End Point: /getData
- Method: POST
- Documentation: Follow link → [7](#)

## 4

# Generating Tree

Once we have generated schema, we have everything handy to generate rules. Throughout the chapter, we will refer rules to tree because we will be displaying these rules in tree manner with indentation. If you are using the tool for the first time then you should consider to generate tree from scratch. Following are the steps to create valid tree.

1. Select the target attribute. (Currently on categorical variables are supported.)
2. Provide the default distribution.
3. Along the root node, you can see the dropdown which contains all the attributes you have added in your schema. You can select any attribute and it will be added as child node to the current node.
4. If selected attribute is
  - (a) Categorical: Number of children nodes will be size of the domain of that attribute.
  - (b) Numerical: You will be asked to provide the split value.
5. If you are done with a branch, you can provide distribution. If no distribution provided to leaf node then default distribution will be considered.

Once you have generated the tree, you can download the entire tree by clicking on "Download Tree" button. Just like the Attribute Schema, it will save equivalent ".json" file in your local system. You can upload this downloaded ".json" file, and system will render the tree you had generated previously. You can also modify this tree.

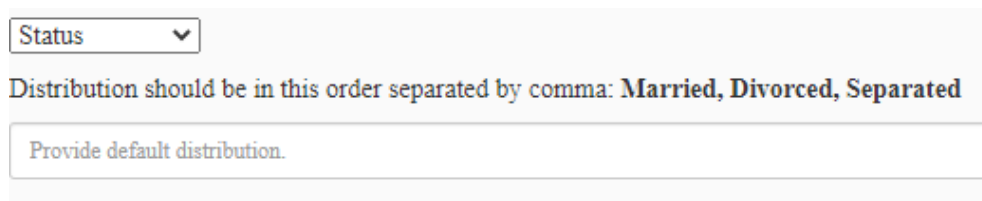
Screenshots can be found in subsequent part of this chapter.

## 4.1 Generating Tree From Scratch

In order to generate tree from scratch, following steps need to be taken.

1. Select the target variable
2. Provide default distribution.
3. Add all rules as per requirement.

### 4.1.1 Target Variable Selection



Status ▼

Distribution should be in this order separated by comma: Married, Divorced, Separated

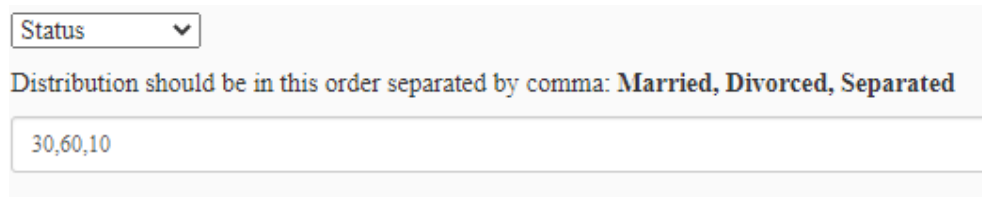
Provide default distribution.

Figure 4.1: Select Target Variable

As shown in 4.1, a target variable needs to be selected before starting to build tree. You can change the target variable any number of time you want before building tree. Domain values will be displayed as a hint below that.

### 4.1.2 Default Distribution

Once target variable is selected, user should provide default distribution. And all the values will be considered as %. Hence, it should add upto 100. Percentage share should be separated by comma (.). Order does matter here. Consider following example.



Status ▼

Distribution should be in this order separated by comma: Married, Divorced, Separated

30,60,10

Figure 4.2: Default Distribution Example

As shown in 4.2, the selected target variable is Status which has three values in its domain namely Married, Divorced, Separated. Thus, in default distribution as well, there should be exactly three values. Where,

- Married : 30%
- Divorced : 60%
- Separated : 10%

These values should add up to 100. Otherwise system will not accept it.



### 4.1.3 Adding Rules to Tree

In this step, you can add rules to the tree. For that you can select any attribute from dropdown menu. Consider following example.

Figure 4.3: Adding Child(ren) To The Tree

As shown in 4.3, by selecting any attribute; you can generate children to the current node.

Figure 4.4: Selecting Categorical Attribute

As shown in 4.4, If categorical attribute is selected then according to domain, children will be created. Here, selected attribute is *Groom Pers* which is categorical and has two unique values in domain *A,B*. Upon selection, it will create two children to current node i.e. Root.

Figure 4.5: Selecting Numerical Attribute

If you select numerical attribute instead, it will open the dialog box as shown in 4.5. Here, *Groom age* is selected which is numerical. You can provide the value for split based on which children will be created. Consider 4.6.

+ Root	Groom age ▼
+ Groom age $\geq 50$	Bride age ▼
+ Groom age $< 50$	Bride age ▼

Figure 4.6: Created Children Upon Selection Of Numerical Attribute

Once if you are done with one branch, you can provide custom distribution if you want. Rules for adding distribution is same as before. (4.1.2) If no distribution is provided to leaf node, then default distribution will be applied.

**Note:** Only leaf node can have distribution. Non-leaf node cannot have distribution. System will not allow to add distribution for non-leaf nodes.

Attribute	Available Attributes	Distribution OR Operation
+ Root	Distribution ▼	
+ Groom age $\geq 50$	Distribution ▼	10,80,10 ✖
+ Groom age $< 50$	Bride age ▼	

Figure 4.7: Custom Distribution

As shown in 4.7, For the branch  $Root \rightarrow Groom\ age \geq 50$ , distribution will be 10,80,10. Once you have added all the attributes, you would like to save it for later use by clicking on "Download Schema". It will save ".json" file in your local machine.

The format of .json file would look something like following.

```

1 {"1": {
2   "id": 1,
3   "name": "Root",
4   "pid": 0},
5  "2": {
6   "id": 2,
7   "name": "Groom age  $\geq 50$ ",
8   "pid": 1,
9   "distribution": "10,80,10"},
10 "3": {
11  "id": 3,
12  "name": "Groom age  $< 50$ ",
13  "pid": 1}
14 }
```

## 4.2 Generating Tree From JSON

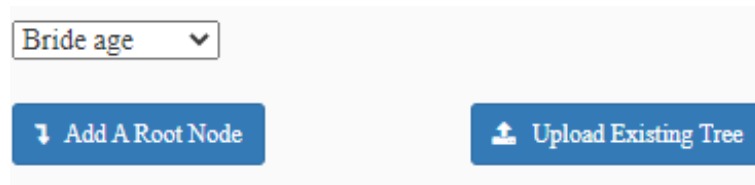


Figure 4.8: Uploading Previously Generated Tree

Generating tree from JSON is similar to Generating Schema from JSON. You need to click on "Upload Existing Tree". It will not only render the tree, but also the target variable will also be selected.

## 4.3 Proceed to Generate Data

Now that we have generated our Tree, we would like to generate synthetic data using these rules. You can click on "Submit" to navigate to the next page. On clicking submit, it will perform all the checks. Should there be any errors, user will have to correct it before proceeding further.

### 4.3.1 API

- End Point: /getTree
- Method: POST
- Documentation: Follow link → [7](#)

## 5

# Generating Data

Now that Schema and Tree has been generated, we are all set to generate synthetic data. Last step depends upon whether you have any derived attribute in your schema.

- If your schema contains any derived attribute, then you will be asked to provide definition of these attributes. You will also have to provide base attributes on which derived attribute depends.
- But if it doesn't contain any derived attribute, then you will not be asked for these inputs.

And then you will have to provide number of tuples to be generated and also the name of the file. It will download the file in your local system.

### 5.1 When Schema Has Derived Attributes

If your schema contains derived attributes then you will have to provide following.

- Definition of each derived attribute (.py file):
- Dependency:

#### 5.1.1 Definition:

Users shall provide **valid** python file containing all the functions to calculate the derived attribute. There should be one main function for each derived attribute. *e.g.* if your schema has 2 derived attributes, then .py script should also have 2 main functions. It can have helper functions. Correctness of the python code will be user's responsibility. However, system will generate appropriate exception message for user's readability.

Example of such python script is provided here.

```
def getAgeDiff(**args):
    return abs(args["Bride age"] - args["Groom age"])

def getPersonComp(**args):
    if(args["Bride pers"] == args["Groom pers"]):
        return "Yes"

    return "No"

def getAdjustedInc(**args):
    num = args["Bride salary"] + args["Groom salary"]
    denom = args["Bride age"] + args["Groom age"]

    return num / denom
```

Please consider following summary table for above example.

Attribute	Type	Domain	Function	Base Attribute
Age diff	Num	-	getAgeDiff	Bride age, Groom age
Person comp	Cat	Yes, No	getPersonComp	Bride pers, Groom pers
Adjusted inc	Num	-	getAdjustedInc	Bride age, Groom age, Bride salary, Groom salary

Rules for python scripts are as following:

1. Arguments to the function should exactly be **\*\*args**.
2. Value from **\*\*args** can be accessed by passing key value. Here key is case sensitive.  
**Example:** "Bride age" can be accessed like this → args["Bride age"]
3. It should only access members on which its value depends upon.  
**Example:** as we can see *getAgeDiff* is defined for "Age diff" which depends on "Bride age" & "Groom age". Then this function should only attempt to access these two values from **\*\*args**.
4. In case of categorical derived attribute's case; it should only return the value which belongs to its domain. This value is also case sensitive.  
**Example:** getPersonComp should only return either "Yes" or "No". If out of domain value is returned, exception will be generated and users will be notified accordingly.

### 5.1.2 Dependency:

Once user has provided python code which follows the instruction provided, they should provide base attributes on which each derived attribute depends and also they should provide respective function for each of the derived attribute.

You can multi select attributes if derived attribute depends upon more than one attributes. Order does not matter. You can only use attributes in your python code which you are selecting here. Function name should match with the name provided in python file. It is case sensitive.

Upload Derived Attribute's Definition Code:  DerivedDefCode.py

---

Name of the Attribute: Age diff  
 Corresponding Method in .py file:

Arguments: 

Bride age  
 Groom salary  
 Groom pers  
 Bride pers

Unmasked?: ☒ Yes ☐ No

---

Name of the Attribute: Adjusted inc  
 Corresponding Method in .py file:

Arguments: 

Bride age  
 Groom salary  
 Groom pers  
 Bride pers

Unmasked?: ☐ Yes ☒ No

---

Figure 5.1: Generate Data With Derived Attribute

### 5.1.3 Unmasking Of Attributes:

By default all the derived attributes will be **masked**. In other words, they will not be included in final generated data. However, if you want some of the derived attributes to be unmasked, you can check "Yes" for those attributes. And they will be included in the final generated synthetic data.

As we can see in 5.1, we have unmasked "Age diff". Which means, in the final data we will have all the base attributes and Age diff as columns. And other two derived attributes will be masked.

## 5.2 Final Step

If schema doesn't contain any derived attributes, then users will not be asked for any of the above inputs.

Now final step is to provide  $N$  number of tuples to be generated and file name. By Clicking on "Download Data", system will save file with provided file name to local system containing  $N$  tuples.

---

Number of Tuples:

Name of the File:

Figure 5.2: Provide  $N$  and File Name

Upon clicking "Download Data", system will perform checks onto the inputs provided. If there are any derived attributes, definition script and dependency of the derived attributes are mandatory. If not provided, system will not allow to submit the form.

There should be any errors in python script, system will raise exception and provide appropriate message to user.

Snippet of the data is as following.

Age diff	Bride age	Bride pers	Bride sala	Groom ag	Groom pe	Groom sal	Status
36	57	A	304	21	A	729	Married
4	40	A	921	44	A	389	Married
16	27	B	775	43	B	356	Separated
5	41	A	610	46	A	188	Married
19	41	A	611	60	A	649	Separated
2	29	A	711	31	B	423	Married
2	43	A	380	41	A	784	Married
11	48	B	844	37	A	465	Married
35	21	B	614	56	B	742	Married
7	33	A	374	40	A	947	Separated
37	58	A	560	21	B	930	Married
9	36	B	326	45	A	560	Married
30	18	B	630	48	A	848	Married
27	23	B	977	50	A	931	Married

Figure 5.3: Data Snippet

**Recall:** We unmasked "Age diff" and not any other derived attributes. Hence, along with base attributes, we have Age diff.

### 5.2.1 API

- End Point: /generateData
- Method: POST
- Documentation: Follow link → [7](#)

## 6

# Data Generation Scheme

Data is getting generated in sequential manner. Following steps are performed to generate one tuple.

1. First and foremost all the values will be generated for both numerical and categorical attributes using following scheme.
  - (a) Categorical  $\rightarrow$  Randomly any value will be picked from the domain.
  - (b) Numerical  $\rightarrow$  Using the range provided (lower and upper values), system will pick random value from this range with equal distribution. [Refer this](#)
2. If the schema contains derived attributes, system will use the functions provided by user through python script to generate the value of derived attributes. These attributes will also be added in tuple.
3. Once the tuple is generated, system will perform DFS to find the appropriate distribution.
4. If current tuple yields any distribution then system will consider it otherwise if no distribution provided then it will use default distribution. Using this distribution, system will generate the target attribute.
5. This process is repeated until  $N$  tuples have been generated.

Consider the following tree and couple of example tuples generated by the system. Output screenshot also contains the distribution it got.



40,20,40

Add A Root Node
Upload Existing Tree
Clear Tree

Attribute	Available Attributes	Distribution OR Operation	
+ Root	Bride age		×
+ Bride age $\geq 25$	Groom age		×
+ Groom age $\geq 30$	Age diff		×
+ Age diff $\geq 5$	Person comp		×
+ Person comp = Yes	Adjusted inc		×
+ Adjusted inc $\geq 1$	Distribution	10,10,80	×
+ Adjusted inc $< 1$	Bride age		×
+ Person comp = No	Distribution	90,0,10	×
+ Age diff $< 5$	Bride age		×
+ Groom age $< 30$	Distribution	70,5,25	×
+ Bride age $< 25$	Distribution	99,1,0	×

Figure 6.1: Example Tree With Default Distribution

```

1 {
2   'Groom age': 42,
3   'Groom salary': 479,
4   'Groom pers': 'A',
5   'Bride pers': 'B',
6   'Bride age': 44,
7   'Bride salary': 885
8 }

```

Above tuple was generated randomly by the system and if we follow the path of the tree we will get following:

**Root  $\rightarrow$  Bride age  $\geq 25 \rightarrow$  Groom age  $\geq 30 \rightarrow$  Age diff  $< 5$**

As we can see, Age diff  $< 5$  is leaf node where distribution is not provided, hence system will use default distribution to generate target attribute.

Consider another example:-

```

1 {
2   'Groom age': 54,
3   'Groom salary': 337,
4   'Bride age': 49,
5   'Bride pers': 'B',
6   'Bride salary': 73,
7   'Groom pers': 'B'
8 }

```

If we follow the path along the tree we will get following:

**Root  $\rightarrow$  Bride age  $\geq 25 \rightarrow$  Groom age  $\geq 30 \rightarrow$  Age diff  $\geq 5 \rightarrow$  Person comp = Yes  $\rightarrow$  Adjusted inc  $\geq 1$**

As we can see, Adjusted inc  $\geq 1$  is leaf node where distribution is provided, hence system will use that distribution to generate target attribute. Refer [6.2](#).

```
127.0.0.1 - - [09/May/2021 02:56:57] "POST /generateData HTTP/1.1"
200 -
TRYING ON TUPLE: {'Groom age': 42, 'Groom salary': 479, 'Groom
pers': 'A', 'Bride pers': 'B', 'Bride age': 44, 'Bride salary':
885}
DISTRIBUTION WE GOT: 40,20,40
TRYING ON TUPLE: {'Groom age': 54, 'Groom salary': 337, 'Bride
age': 49, 'Bride pers': 'B', 'Bride salary': 73, 'Groom pers':
'B'}
DISTRIBUTION WE GOT: 10,10,80
```

Figure 6.2: Example output

# 7

## Backend

Backend APIs have been written in Flask. (Python) It's been currently hosted on AWS. → [Here](#). Different endpoints served by application are as following.

- /:
  - Whenever this [URL](#) is hit, this endpoint will be called. It will navigate you to the home page.
  - Request Type: **GET**
- /getData:
  - This will be called when user submits the schema. This endpoint will receive the data provided by the user, it will process it and navigate user to the page where he can generate tree.
  - Request Type: **POST**
  - Sample Request:

```
1 {
2   "base": [{
3     "attributeName": "Bride age",
4     "attributeType": "numerical",
5     "values": ["18", "60"]},
6     {
7       "attributeName": "Bride pers",
8       "attributeType": "categorical",
9       "values": ["A", "B"]}],
10  "derived": [{
11    "attributeName": "Age diff",
12    "attributeType": "numerical",
13    "values": ["", "" ]},
14    {
15      "attributeName": "Person comp",
16      "attributeType": "categorical",
17      "values": ["Yes", "No"]}]
18 }
```

- /getTree:
  - This will be called when user submits the tree. It will receive the tree generated by the user. It will create the object of custom class *Tree* and initialize it with the tree provided. This will redirect users to the page where they can download the data.
  - Request Type: **POST**
  - Sample Request:

```

1 {
2   "1": {
3     "id": 1,
4     "name": "Root",
5     "pid": 0
6   },
7   "2": {
8     "id": 2,
9     "name": "Bride age >= 25",
10    "pid": 1
11  },
12  "3": {
13    "id": 3,
14    "name": "Bride age < 25",
15    "pid": 1,
16    "distribution": "99,1,0"
17  },
18  "target_variable": {
19    "name": "Status",
20    "distribution": "40,20,40",
21    "type": "categorical",
22    "values": ["Married", "Divorce", "Separate"],
23    "derived": "no"
24  },
25  "attr_info": {
26    "Bride age": {
27      "type": "numerical",
28      "values": ["18", "60"],
29      "derived": "no"
30    },
31    "Groom age": {
32      "type": "numerical",
33      "values": ["18", "60"],
34      "derived": "no"
35    },
36  }

```

- Request will contain information regarding all the attributes added in schema, information regarding target variable (including default distribution) and all the rules added in tree and distribution if provided.

- /generateData:
  - This will be called when user clicks on generate data. This endpoint will receive the number of tuples to be generated and definition of derived attribute(s) if there are any. It will generate the data which will be downloaded to the local system.
  - Request Type: **POST**
  - Sample Request:

```
1 {
2   "derived_atts": {
3     "Adjusted inc": {
4       "parameters": ["Bride age", "Bride sal"],
5       "method_name": "getAdjustedInc",
6       "visible": true
7     },
8     "Person comp": {
9       "parameters": ["Bride pers", "Groom pers"],
10      ,
11      "method_name": "getPersonComp",
12      "visible": false
13    },
14    "Age diff": {
15      "parameters": ["Bride age", "Groom age"],
16      "method_name": "getAgeDiff",
17      "visible": false
18    }
19  },
20  "module_name": "DerivedDefCode",
21  "data_points": "1500"
22 }
```

- Along with this JSON data, python script will also be sent to server. Server will temporary make a copy of that file and store it.

## 8

# Credits and Code Base

- Tree Generation UI is borrowed from : [GitHub Repo](#)
- GitHub Repository: [Quiz Generator](#)
- URL: [Live Version](#)
- Server Info: AWS - Windows Server OS 2016
- Technologies Used: Python3.6, JavaScript, AJAX, HTML, jQuery.
  - Framework Used: Flask (v1.1.2)
  - Libraries Used: Numpy (v1.19.4)