**KNOWLEDGE BASE**

**Steps to Install:**

1. Create a virtual environment and activate it - virtualenv env (python -m virtualenv env) source env/bin/activate (source env/Scripts/activate)
2. Install the required libraries (pip install -r requirements.txt)
3. Run the server (python patient\_normalization\_app.py)

To gain the domain knowledge please read this: KB\Documentations\Lab 100 Knowledge Base Write Up.docx

To know about the framework and the structure of the code, please go through this: KB\Documentations\Code Structure.docx.

To know about the techniques used while developing this framework, please go through this: KB\Documentations\Knowledgebase Final Wrap Up.docx

**Anthropometrics (Body Scan)**

The main targets in the body scan are:

1. Height
2. Waist Circumference
3. Trunk to Leg Volume Ratio
4. Weight
5. Waist to Hip Ratio

**Example URLs:**

<http://localhost:8085/normative_data/targets=height+50&variables=gender+2,age+24>

<http://localhost:8085/normative_data/targets=waist_circumference+36.1&variables=gender+2,age+24>

<http://localhost:8085/normative_data/targets=trunk_to_leg_volume_ratio+1.4&variables=gender+2,age+24>

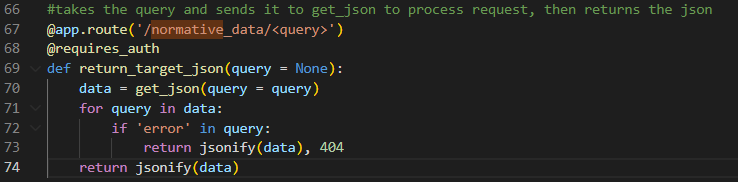
<http://localhost:8085/normative_data/targets=body_weight+67.545&variables=gender+2,age+24>

<http://localhost:8085/normative_data/targets=waist_to_hip_ratio_score+0.88&variables=gender+2,age+24>

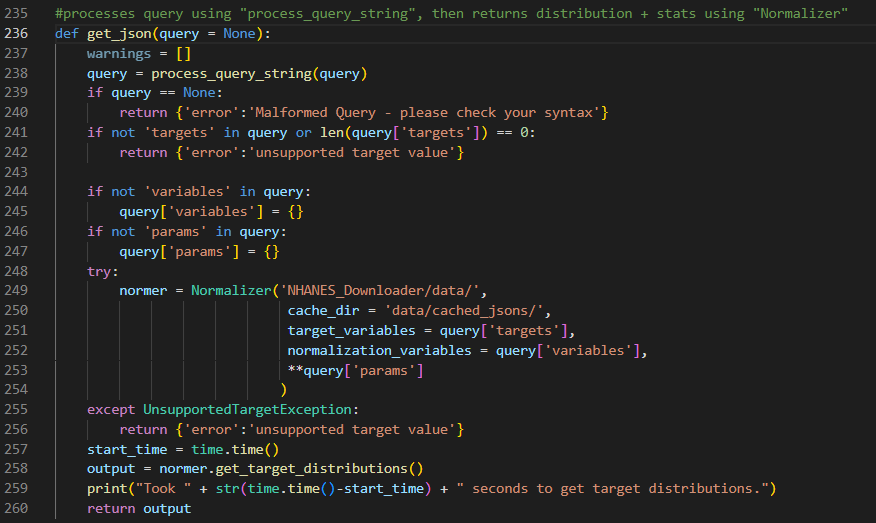
Response/output for each target in JSON can be seen here - [URL](https://quisitive-my.sharepoint.com/:f:/p/prudhvi_vuda/EjoAVd81Q61Bs4C1AyWg2BEB4GPZuH0LEhQYHDy19WntKw?e=2g5Ude)

**Request response cycle for Body Scan**

As soon as a call to KB from node server is made, return\_target\_json() in the app/flask\_index.py gets called.



The query is passed to the get\_json() function to get the response for the query.



The query is then passed to the process\_query\_string() function to process the query and separate the variables, targets and the other parameters.

The targets along with the variables and params are passed to the Normalizer class to initialize the variables which are needed in the return data.

For more information on Normalization, please go through this: KB\Documentations\Lab100Kb Normalization Method.docx.

Get\_target\_distribution() method of Normalizer class is called to get the density plots, interpretation, static\_distributions and the percentile values for the input target variable.

The static measures which are used in the calculations are mean, standard deviation, age\_min, age\_max, and gender. This data can be found in the directory – data/static\_distributions.csv

**Body Composition**

The targets in the body composition are:

1. Weight
2. Body fat mass
3. Skeletal muscle mass
4. Others

The request URL is: localhost:8085/derived\_stats

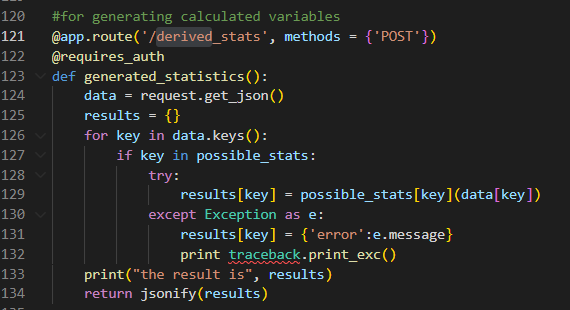
The request body is:

{  
    "body\_composition": {  
        "weight": 156.0258,  
        "body\_fat\_mass": 24.916500000000003,  
        "skeletal\_muscle\_mass": 75.41100000000002,  
        "other": 55.698299999999975,  
        "percent\_other": 35.69813453928772,  
        "percent\_body\_fat": 15.969474279253818,  
        "percent\_skeletal\_muscle": 48.33239118145846,  
        "visceral\_fat\_level": 4,  
        "age": 21,  
        "gender": 1,  
        "ethinicity":1  
    }  
}

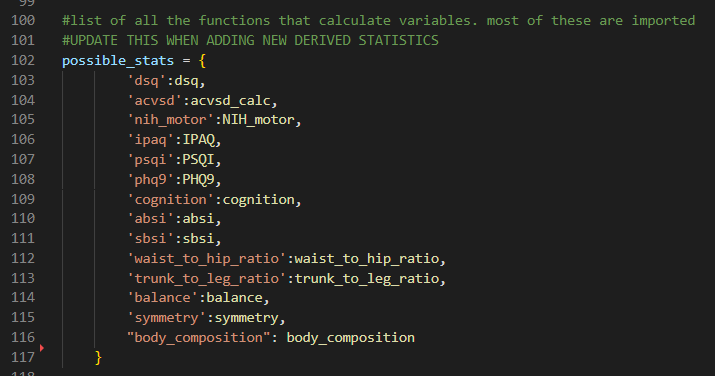
This is the response from the KB server: KB\Responses\Body Composition\body composition.txt

**Request Response Cycle for Body Composition**

When a request from Node is made to the Kb server, the below function in app/flask\_index.py is called and for each key in the response it checks if it has a match in the possible\_stats dictionary.

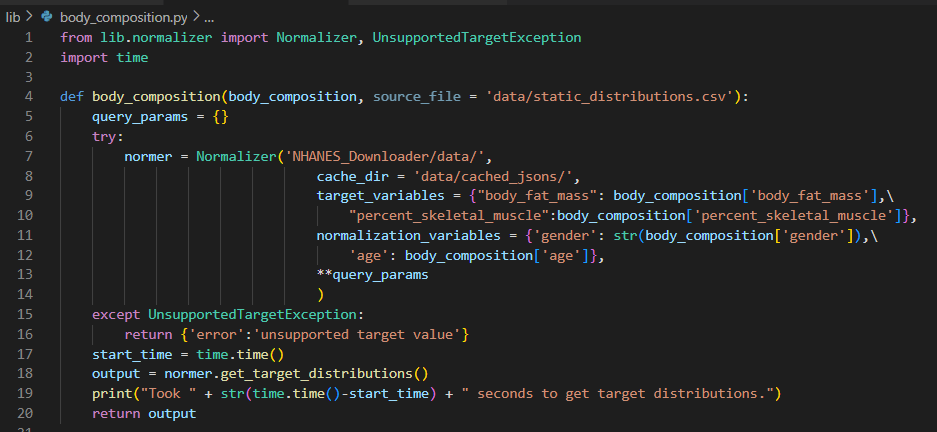


If it finds a match in the possible\_stats, it takes the value from this dictionary and calls it as a function with the request body as the argument. Here the called function is body\_composition which is in lib directory.



Normalizer class is initialized and the required paramters are passed into the constructor for initialization.

Get\_target\_distributions() method of Normalizer class is called to get the density plots, static\_distribution and the percentile vaues for the target\_variables.



**Dexterity**

Target Variables are

1. dominant\_hand\_dexterity
2. Non\_dominant\_hand\_dexterity

If we hit from node server, use the below url and request body

The request urls from node is: <http://localhost:3030/dexterity/data>

The request body is:

{  
    "right": 16.565537,  
    "left": 17.989352  
}

If we directly hit the KB server, use the below url as request GET call.

<http://localhost:8085/normative_data/targets=dominant_hand_dexterity+53.32192475940507&variables=gender+2,age+24,ethnicity+7>

<http://localhost:8085/normative_data/targets=non_dominant_hand_dexterity+54.36905270900573&variables=gender+2,age+24,ethnicity+7>

Normative data is called in the KB server like body scan and it excutes the code and returns the output.

Dexterity uses the data like mean and standard deviation for calculations from the directory data/static\_distributions.csv which are dominant\_hand\_dexterity and non\_dominant\_hand\_dexterity

The response from the KB server can be found at, KB\Responses\Dexterity

**Strength**

Target Variables are:

1. dominant\_hand\_grip\_strength
2. non\_dominant\_hand\_grip\_strength

Request from node is: <http://localhost:3030/strength/data>

Request body is:

{  
  "right": 131.334,  
  "left": 101.643  
}

Response is:

{  
    "status": "ok"  
}

If we hit the KB server directly

<http://localhost:8085/normative_data/targets=non_dominant_hand_grip_strength+66.45679920613193&variables=gender+2,age+24,ethnicity+7>

<http://localhost:8085/normative_data/targets=dominant_hand_grip_strength+83.8540994377095&variables=gender+2,age+24,ethnicity+7>

Normative data is called in the KB server like body scan and it excutes the code and returns the output.

Strength uses the data like mean and standard deviation for calculations from the directory data/static\_distributions.csv which are dominant\_hand\_grip\_strength and non\_dominant\_hand\_grip\_strength

To view the output responses, please go through KB\Responses\Strength

**Balance**

The input will be the balance score as below to the KB server

{  
    "balance": {  
        "balance\_score": 57  
    }  
}

The output response would be

{  
    "balance": {  
        "balance\_score": 57,  
        "percentile": 75.8036347776927  
    }  
}

**Blood Draw**

We have 2 types of Blood Draw:

1. Lipids
2. Metabolism

The URL is: http://localhost:3030/blood\_draw/data

The input request body for the node server to hit the APIs for Lipids is

{  
    "LDL": "171",  
    "TC/H": "4.2",  
    "VLDL": "17",  
    "TRIG": "83",  
    "nHDLc": "188",  
    "CHOL": "247",  
    "HDL": "59"  
}

The input request body to hit the node server for Metabolism is

{  
  "BUN": "18",  
  "CRE": "0.9",  
  "GLU": "99",  
  "NA+": "141",  
  "CL-": "105",  
  "ALP": "70",  
  "CA": "9.6",  
  "TP": "7.8",  
  "TBIL": "0.6",  
  "K+": "4.7",  
  "tCO2": "25",  
  "ALB": "4.0",  
  "ALT": "28",  
  "AST": "26"  
}

If we would like to hit directly from the KB server, we can the below URLs for the respective targets

**Lipids: Below are the targets which fall under Lipids category**

Cholestrol: <http://localhost:8085/normative_data/targets=cholesterol+247&variables=gender+2,age+24,ethnicity+7>

cholesterol\_hdl: <http://localhost:8085/normative_data/targets=cholesterol_hdl+4.2&variables=gender+2,age+24,ethnicity+7>

Hdl: <http://localhost:8085/normative_data/targets=hdl+59&variables=gender+2,age+24,ethnicity+7>

Ldl: <http://localhost:8085/normative_data/targets=ldl+171&variables=gender+2,age+24,ethnicity+7>

Triglycerides: <http://localhost:8085/normative_data/targets=triglycerides+83&variables=gender+2,age+24,ethnicity+7>

Vldl: <http://localhost:8085/normative_data/targets=vldl+17&variables=gender+2,age+24,ethnicity+7>

**Metabolism: Below are the targets which fall under metabolism category**

alanine\_aminotransferase: <http://localhost:8085/normative_data/targets=alanine_aminotransferase+28&variables=gender+2,age+24,ethnicity+7>

albumin: <http://localhost:8085/normative_data/targets=albumin+4&variables=gender+2,age+24,ethnicity+7>

alkaline\_phosphatase: <http://localhost:8085/normative_data/targets=alkaline_phosphatase+4&variables=gender+2,age+24,ethnicity+7>

aspartate\_aminotransferase: <http://localhost:8085/normative_data/targets=aspartate_aminotransferase+26&variables=gender+2,age+24,ethnicity+7>

bicarbonate: <http://localhost:8085/normative_data/targets=bicarbonate+25&variables=gender+2,age+24,ethnicity+7>

blood\_urea\_nitrogen: <http://localhost:8085/normative_data/targets=blood_urea_nitrogen+18&variables=gender+2,age+24,ethnicity+7>

calcium: <http://localhost:8085/normative_data/targets=calcium+9.6&variables=gender+2,age+24,ethnicity+7>

chloride: <http://localhost:8085/normative_data/targets=chloride+105&variables=gender+2,age+24,ethnicity+7>

creatinine: <http://localhost:8085/normative_data/targets=creatinine+0.9&variables=gender+2,age+24,ethnicity+7>

glucose: <http://localhost:8085/normative_data/targets=glucose+99&variables=gender+2,age+24,ethnicity+7>

Potassium: <http://localhost:8085/normative_data/targets=potassium+4.7&variables=gender+2,age+24,ethnicity+7>

sodium: <http://localhost:8085/normative_data/targets=sodium+141&variables=gender+2,age+24,ethnicity+7>

total\_bilirubin: <http://localhost:8085/normative_data/targets=total_bilirubin+0.6&variables=gender+2,age+24,ethnicity+7>

total\_protein: <http://localhost:8085/normative_data/targets=total_protein+7.8&variables=gender+2,age+24,ethnicity+7>

Alternatively, to reduce the number of GET calls made to the server we can use the below POST call for Lipids and Metabolism to get the response from the KB server

For Lipids

{  
    "blood\_draw": {  
        "targets": {  
            "ldl": "171",  
            "vldl": "17",  
            "triglycerides": "83",  
            "non\_hdl\_cholesterol": "188",  
            "cholesterol": "247",  
            "hdl": "59"  
        },  
        "variables": {  
            "age": 21,  
            "gender": 1,  
            "ethinicity": 1  
        }  
    }  
}

For Metabolism

{  
    "blood\_draw": {  
        "targets": {  
            "blood\_urea\_nitrogen": "18",  
            "creatinine": "0.9",  
            "glucose": "99",  
            "sodium": "141",  
            "chloride": "105",  
            "alkaline\_phosphatase": "70",  
            "calcium": "9.6",  
            "total\_protein": "7.8",  
            "total\_bilirubin": "0.6",  
            "potassium": "4.7",  
            "bicarbonate": "25",  
            "albumin": "4.0",  
            "alanine\_aminotransferase": "28",  
            "aspartate\_aminotransferase": "26"  
        },  
        "variables": {  
            "age": 21,  
            "gender": 1,  
            "ethinicity": 1  
        }  
    }  
}

The output JSONs for all the Lipids and Metabolism can be found here: KB\Responses\Blood Draw

**Patient Surveys**

The targets for the patient surveys are: -

1. Sleep - PSQI
2. Mental health - PHQ9
3. Physical activity - IPAQ
4. Nutrition – DSQ

To check the usage of the APIs, URL (common for all the APIs): localhost:8085/derived\_stats

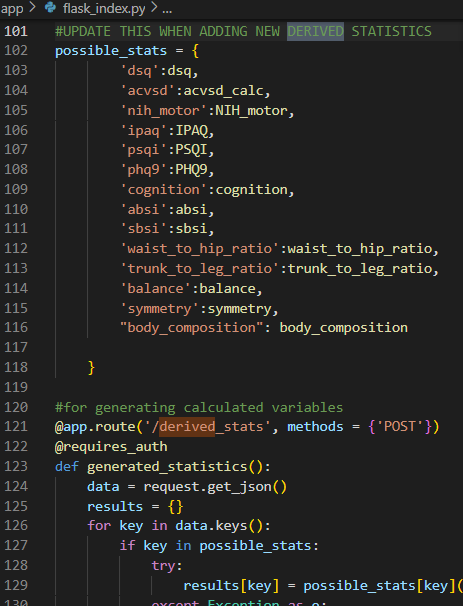
**PSQI:**

Use the below body to hit from the KB server

{  
  "psqi": {  
    "Question\_1": "12:00:00",  
    "Question\_2": "5",  
    "Question\_3": "6:30:00",  
    "Question\_4": "6",  
    "Question\_5A": "Not during the past month",  
    "Question\_5B": "Three or more times a week",  
    "Question\_5C": "Not during the past month",  
    "Question\_5D": "Not during the past month",  
    "Question\_5E": "Not during the past month",  
    "Question\_5F": "Not during the past month",  
    "Question\_5G": "Less than once a week",  
    "Question\_5H": "Not during the past month",  
    "Question\_5I": "Not during the past month",  
    "Question\_5J": "Three or more times a week",  
    "Question\_6": "Fairly good",  
    "Question\_7": "Not during the past month",  
    "Question\_8": "Once or twice a week",  
    "Question\_9": "Less than once a week"  
  }  
}

When a request is made with the above body, a call to generated\_statistics() is made which is in app/flask\_index.py file.

It checks for the key PSQI in the possible\_stats dictionary and calls the function PSQI in lib/PSQI.py to excutes the core logic.



**PHQ9:**

{  
    "phq9": {  
        "doing\_things": "Several days",  
        "feeling\_down": "Not at all",  
        "falling\_asleep": "More than half the days",  
        "feeling\_tired": "More than half the days",  
        "poor\_appetite": "Several days",  
        "feeling\_bad": "Not at all",  
        "trouble\_concentrating": "Several days",  
        "moving\_slowly": "Not at all",  
        "thoughts\_of\_death": "Not at all",  
        "things\_at\_home": "Not difficult at all"  
    }  
}

**IPAQ:**

{  
    "ipaq": {  
        "vigorous\_days": "3",  
        "vigorous\_hours": "0",  
        "vigorous\_mins": "30",  
        "moderate\_days": "2",  
        "moderate\_hours": "0",  
        "moderate\_mins": "22",  
        "walking\_days": "5",  
        "walking\_hours": "1",  
        "walking\_mins": "0",  
        "sit\_min": 10  
    }  
}

**DSQ:**

{  
    "dsq": {  
        "age": 30,  
        "sex": 1,  
        "cereal": {  
            "cereals": [  
                {  
                    "freq": 0.10,  
                    "name": "Cheerios",  
                    "type": "all"  
                },  
                {  
                    "freq": 0.25,  
                    "name": "Branola",  
                    "type": "all"  
                },  
                {  
                    "freq": 0.5,  
                    "name": "Fruit Harvest",  
                    "type": "all"  
                },  
                {  
                    "freq": 0.5,  
                    "name": "Oat bran cereal, cooked, fat not added in cooking",  
                    "type": "all"  
                }  
            ]  
        },  
        "milk": {  
            "freq": "1.0"  
        },  
        "soda": {  
            "freq": "0.133"  
        },  
        "fruit\_juice": {  
            "freq": "0.1"  
        },  
        "coffee": {  
            "freq": 0  
        },  
        "sweet\_drinks": {  
            "freq": 0  
        },  
        "fruit": {  
            "freq": "0.133"  
        },  
        "salad": {  
            "freq": "0.714"  
        },  
        "potatoes\_fried": {  
            "freq": "0.266"  
        },  
        "dry\_beans": {  
            "freq": "0.133"  
        },  
        "potatoes\_other": {  
            "freq": "0.2"  
        },  
        "grains": {  
            "freq": "0.133"  
        },  
        "vegetables": {  
            "freq": "1.0"  
        },  
        "salsa": {  
            "freq": "0.166"  
        },  
        "pizza": {  
            "freq": "0.0"  
        },  
        "tomato\_sauce": {  
            "freq": "0.166"  
        },  
        "cheese": {  
            "freq": "0.286"  
        },  
        "red\_meat": {  
            "freq": "0.286"  
        },  
        "proc\_meat": {  
            "freq": "0.1"  
        },  
        "bread": {  
            "freq": "0"  
        },  
        "candy": {  
            "freq": 0  
        },  
        "doughnuts": {  
            "freq": "0.0333"  
        },  
        "cookies": {  
            "freq": "0.133"  
        },  
        "desserts": {  
            "freq": "0.0"  
        },  
        "popcorn": {  
            "freq": "0.0666"  
        }  
    }  
}

PHQ9, IPAQ, DSQ also follows the similar approach to get the results for the given inputs.

The responses to the above requests can be found here. KB\Responses\Patient Surveys

**Script to prepare or configure the data when new data comes in**

# run this file when new dataset comes to convert the .csv files into required JSONs and Dictionary.  
# this script works assuming the given dataset is in sorted order and need to be run only once.  
  
  
**from** copy **import** deepcopy  
**import** csv, json  
**from** collections **import** OrderedDict  
  
  
**def** **read\_csv\_universal\_newline** (filename, delimiter = ","):  
    **try**:  
        csvreader = csv.reader(open(filename, 'r'))  
        data = []  
        **for** row **in** csvreader:  
            data.append(row)  
        **return** data  
    **except** Exception **as** e:  
        log.error(e)  
        **return** False  
      
**def** **tryFloat**(var):  
    **try**:  
        **return** float(var)  
    **except**:  
        **return** var  
  
  
**def** **tryFloatBool**(var):  
    **try**:  
        float(var)  
        **return** True  
    **except**:  
        **return** False  
  
  
**def** **ordertest**(A):  
    **for** i **in** range(len(A) - 1):  
        **if** isinstance(A[i], float) **and** isinstance(A[i+1], float) **and** A[i]<A[i+1]:  
            **return** False  
    **return** True  
  
  
# give the path of the new dataset for which we need to configure or convert the data into existing file formats  
source = '/Work/Quisitive/Weight.csv'  
  
  
template = OrderedDict()  
'''  
    template can be created based on the number of age groups and the gender.  
    here, there are 8 age groups in each gender, so there would be 16 keys and   
    values depends on the data for each age group.  
'''  
template["gender\_2\_age\_0"] = []  
template["gender\_2\_age\_1"] = []  
template["gender\_2\_age\_2"] = []  
template["gender\_2\_age\_3"] = []  
template["gender\_2\_age\_4"] = []  
template["gender\_2\_age\_5"] = []  
template["gender\_2\_age\_6"] = []  
template["gender\_2\_age\_7"] = []  
template["gender\_1\_age\_0"] = []  
template["gender\_1\_age\_1"] = []  
template["gender\_1\_age\_2"] = []  
template["gender\_1\_age\_3"] = []  
template["gender\_1\_age\_4"] = []  
template["gender\_1\_age\_5"] = []  
template["gender\_1\_age\_6"] = []  
template["gender\_1\_age\_7"] = []  
  
  
**def** **static\_distribution\_list**():  
    raw\_data = read\_csv\_universal\_newline(source)  
    keys = raw\_data[0]  
    keys = dict(map(**lambda** x: (keys[x], x), range(len(keys))))    
    proc\_data = {}  
    **for** row **in** raw\_data[1:]:  
        var\_name = row[keys['variable']]  
        **if** **not** var\_name **in** proc\_data:  
            proc\_data[var\_name] = []  
        proc\_data[var\_name].append({})  
        **for** key, index **in** keys.items():  
            proc\_data[var\_name][-1][key] = row[index]  
    results = {}  
    **for** key **in** proc\_data.keys():  
        var\_data = deepcopy(proc\_data[key])  
        age\_table = sorted(list(set(map(**lambda** x: tryFloat(x['age\_min']), var\_data))))  
        print(age\_table)  
        gender\_table = sorted(list(set(map(**lambda** x: tryFloat(x['gender']), var\_data))))  
        print(gender\_table)  
        value\_table = list(map(**lambda** x: tryFloat(x['value']), var\_data))  
        percentile\_table = list(map(**lambda** x: tryFloat(x['percentile']), var\_data))  
          
    '''  
        code to get the variable(weight/height/waistcircumferece) values   
        for the keys in the above defined template in the form of a dictionary  
    '''  
    i = 0  
    **while** i < len(value\_table):  
        **for** key **in** template:  
            template[key].append(value\_table[i])  
            i += 1  
    result = json.dumps(template)  
      
  
  
    # code to get the percentiles for a variable in the form of a dictionary  
    i = 0  
    p = {}  
    **while** i < len(percentile\_table):  
        p[i] = percentile\_table[i]  
        i += 1  
    result = json.dumps(p)  
    print(len(p))  
      
    **with** open("demo.json", "w") **as** outfile:  
        outfile.write(result)

The above script file can be found at thornelabs-kb\lib\script.py

Below script is used to get the percentile for given input

age\_diff = sorted(list(set(map(lambda x: tryFloat(x['age\_min']), var\_data))))

def get\_diff(ls, val):  
    **for** i **in** range(1, len(ls)):  
        **if** ls[i-1] <= int(val) < ls[i]:  
            **return** i-1  
    **return** len(age\_diff)-1  
  
def get\_percentile(age, gender, value):  
    age\_index = get\_diff(age\_diff, age)  
    res\_key = result\_key[key]  
    pos1 = bisect.bisect\_right(res\_key, float(value))-1  
    pos2 = bisect.bisect(res\_key, float(value))  
    **if** abs(float(value)-res\_key[pos1]) < abs(float(value)- res\_key[pos2]):  
        pos = pos1      
    **else**:  
        pos = pos2  
    print(res\_key[pos1], res\_key[pos2], res\_key[pos])  
    **if** pos == 1001: **return** 99.9  
    **if** pos == 0: **return** 0.1  
    percentile\_index = pos\*16 + (age\_index+1)  
    **if** gender == 1:  
        percentile\_index += 8  
    percentile = weight\_percentiles[str(percentile\_index)]  
    print("percentile is", percentile)  
    **if** percentile == 100: **return** 99.9  
    **if** percentile == 0: **return** 0.1  
      
    **return** percentile

Python3 KB Response Times and Validations: KB\KB Python3 Validation.xlsx