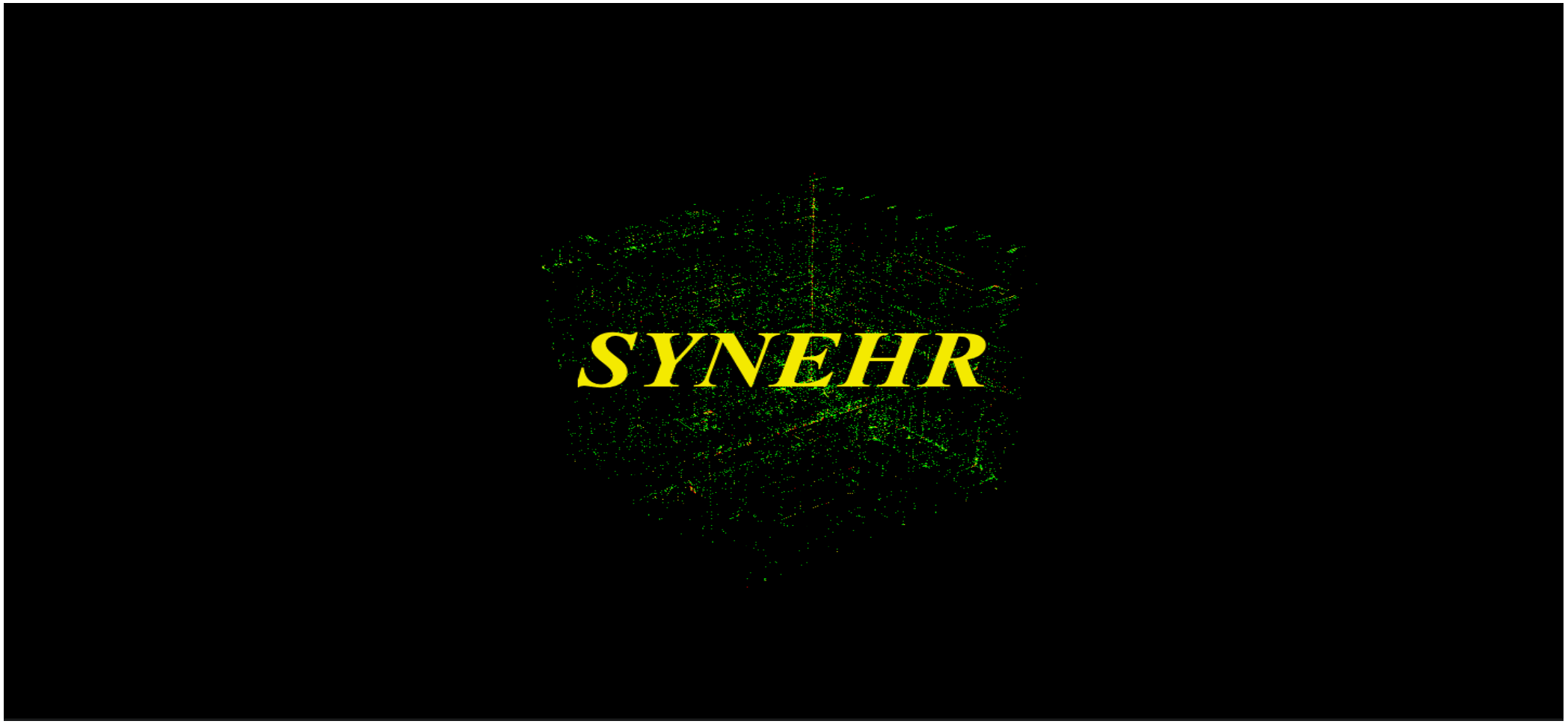
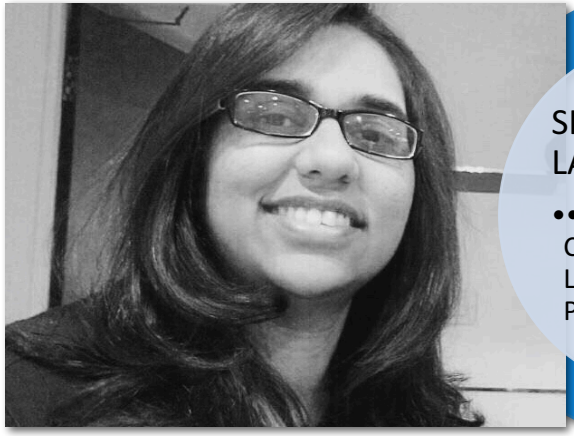


PYTHON PACKAGE TO GENERATE SYNTHETIC IDENTIFIED ELECTRONIC HEALTH RECORDS.



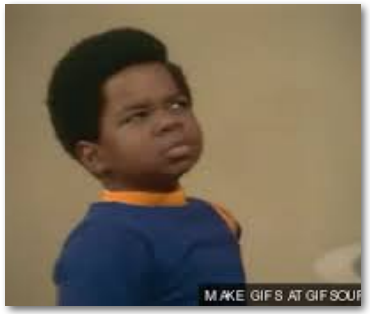
SHRADDHA LANKA

••Data Enthusiast, Comp-Fin Combo. Life Mantra? Inner Peace

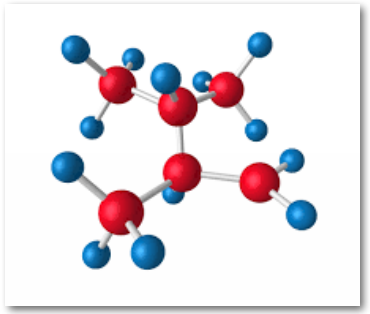
TENNYSON LEE

••Fitness Enthusiast, Life Mantra? Nothing is impossible with hardwork.

TODAY's AGENDA



What is SYNEHR?



Structure of SYNEHR

Functions

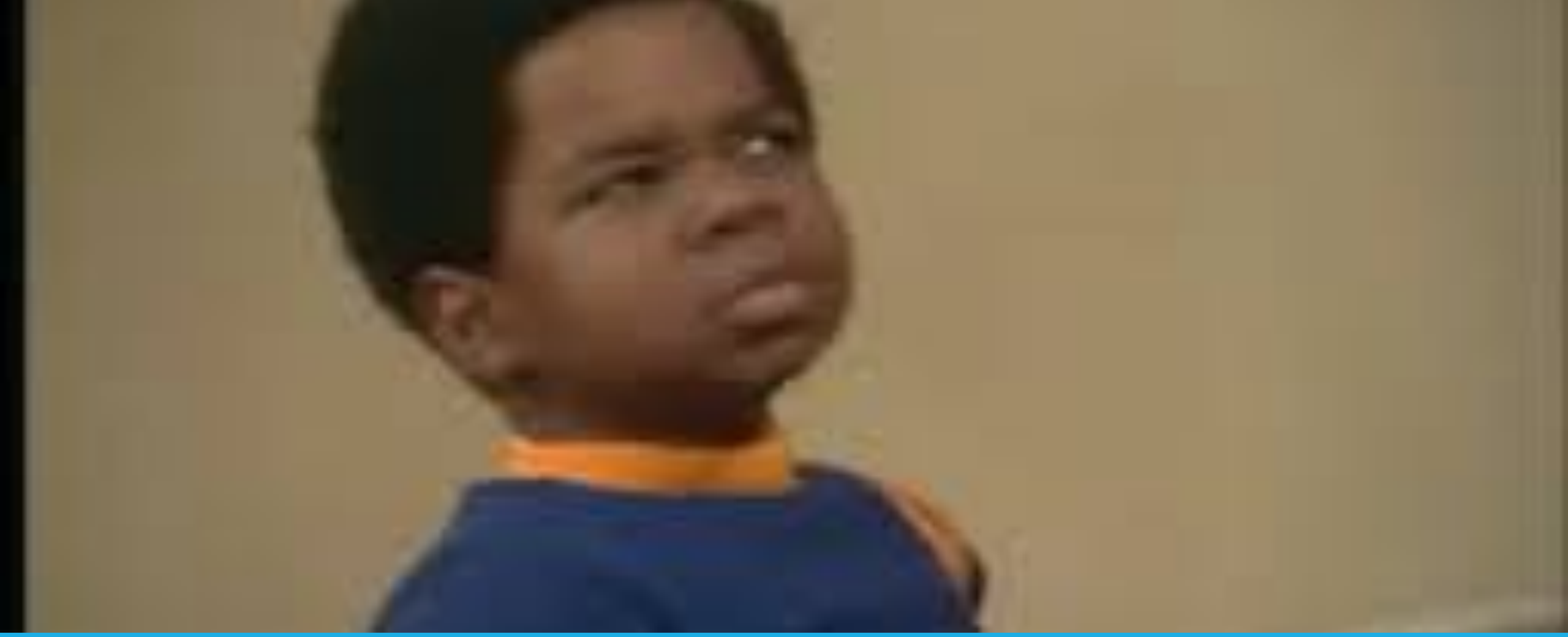
Testing

Challenges-Solutions

Using SYNEHR

Assumptions

Next Steps



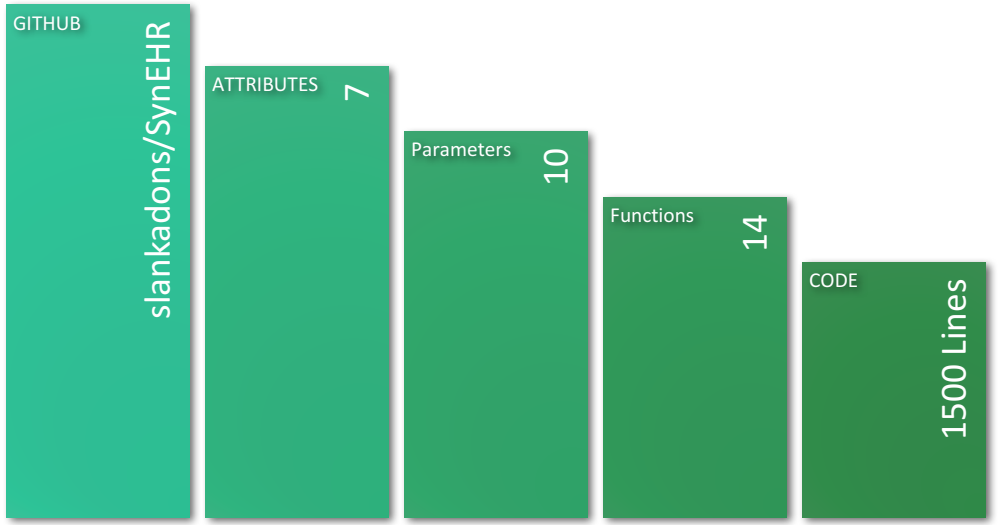
WHAT IS SYNEHR?

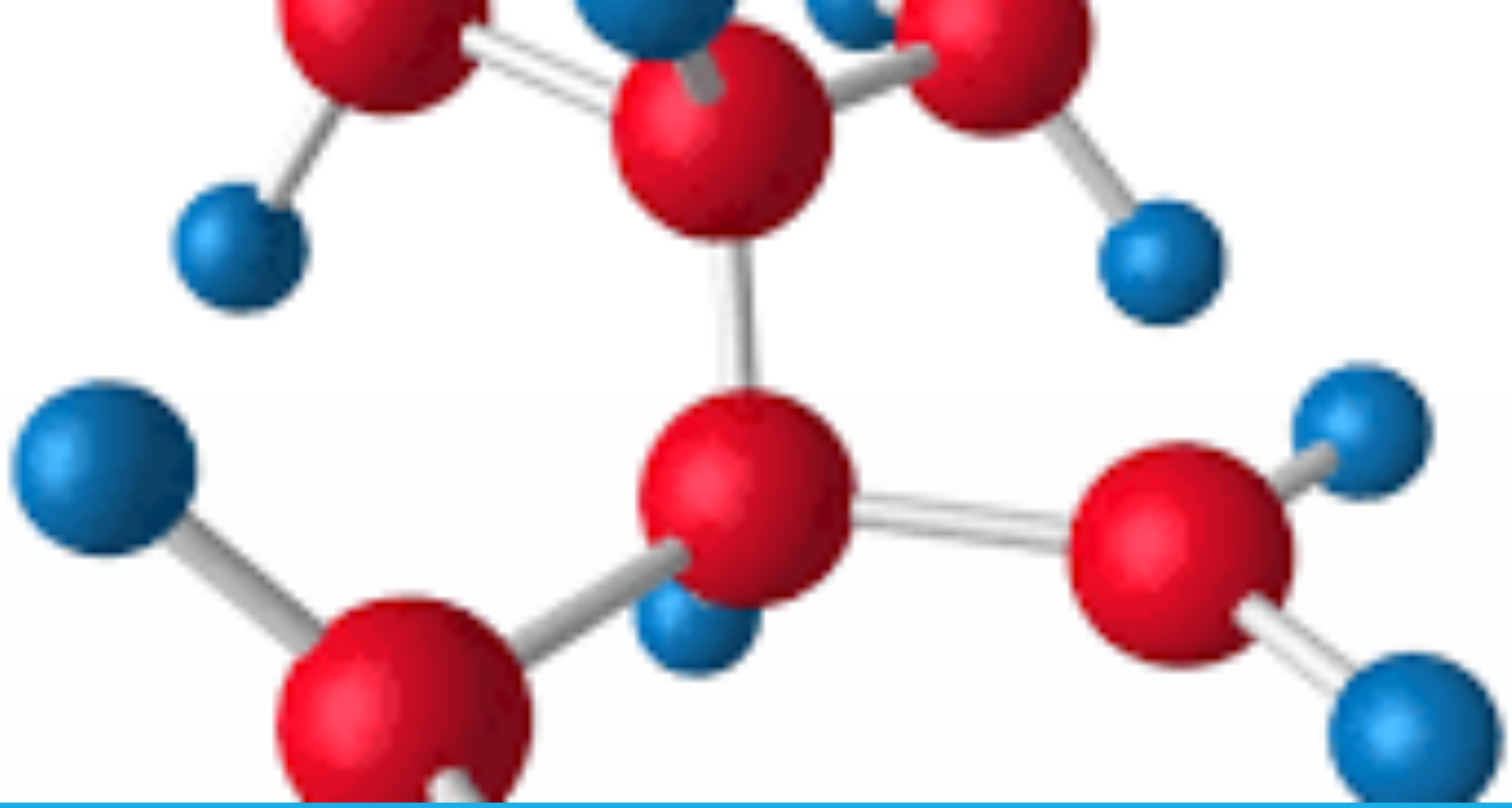
OVERVIEW

INTRODUCTION

USE

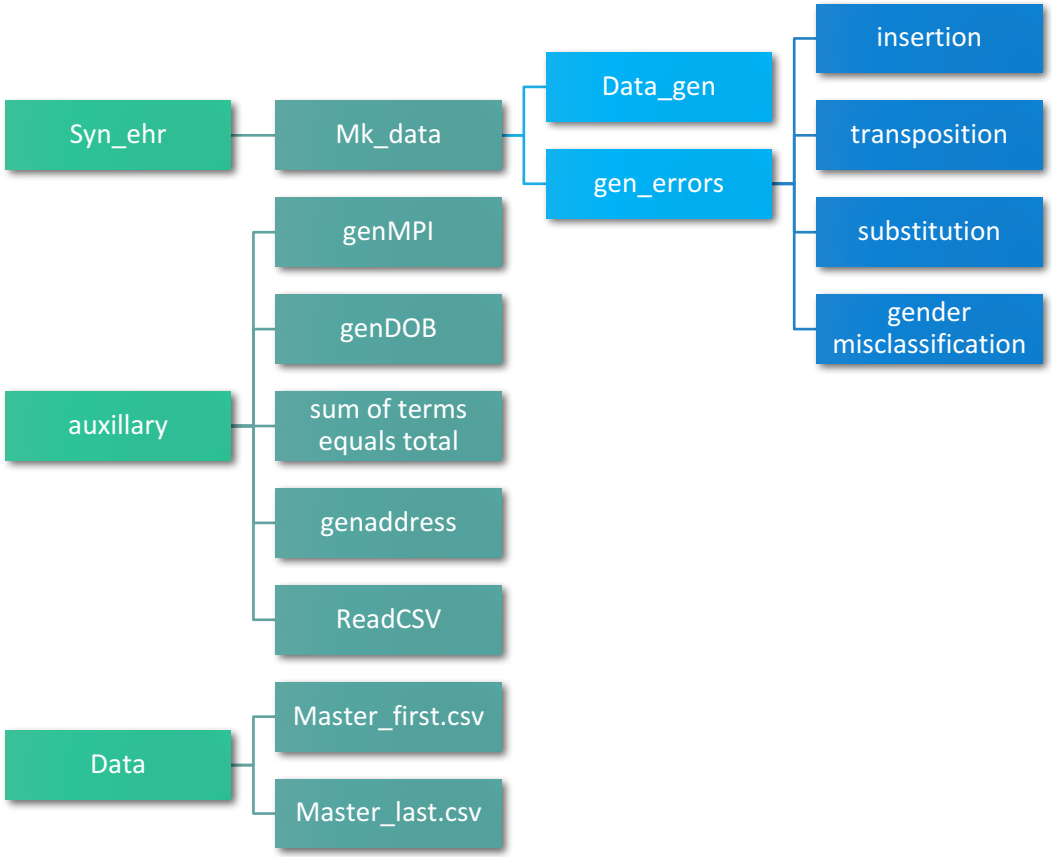
••This Package was created as a resource to test various master patient index algorithms and detect their efficiency.

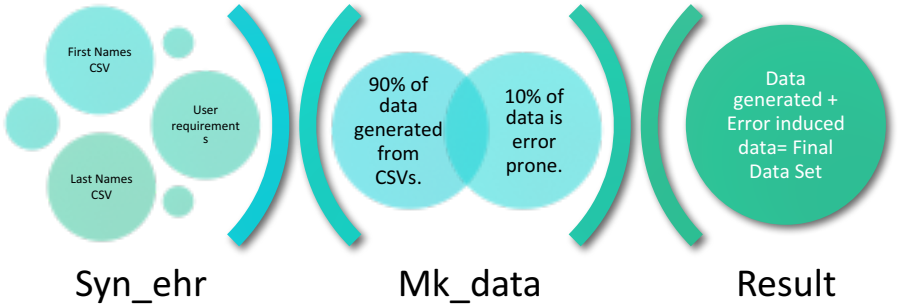




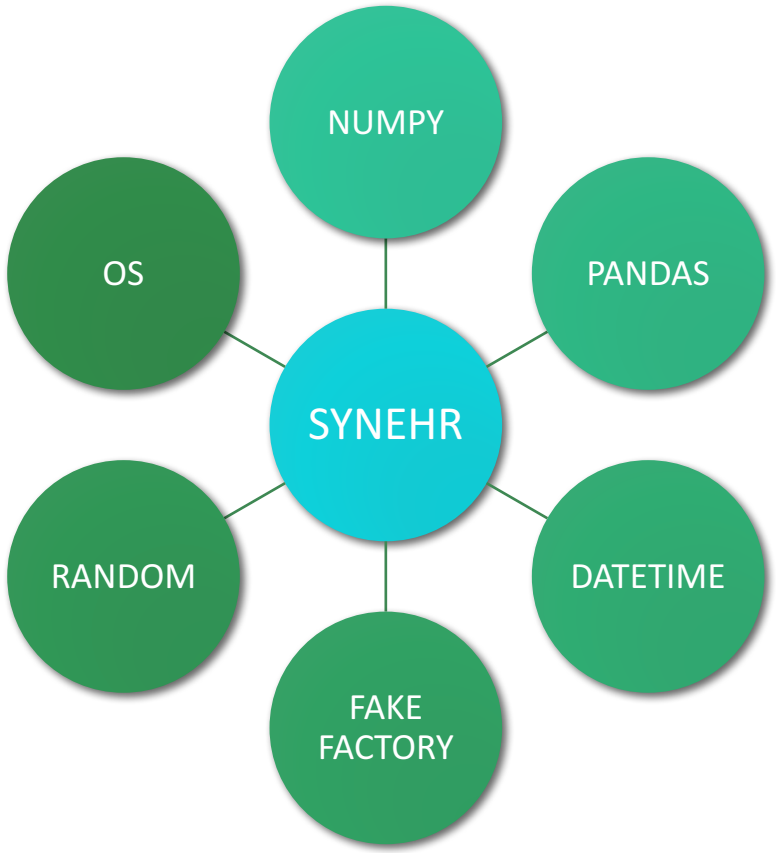
STRUCTURE OF SYNEHR

Functions and Data Generation





PACKAGES USED/ CUSTOM EXCEPTIONS



GENDER VALUE ERROR

••The Gender Value Error generates an Error if incorrect gender percentages are entered.

SIZE VALUE ERROR

••Generates Error if incorrect size ranges are entered.

THE FUNCTIONS

SPHINX DOCUMENTATION FOR FUNCTIONS

TESTING

TESTING

Error Induction

••Check if Errors are appropriately Induced

Exceptions Raised

••Check if appropriate Exceptions are raised when invalid values are assigned to parameters.

Auxiliary Functions Testing

••Check if all auxiliary functions work correctly.

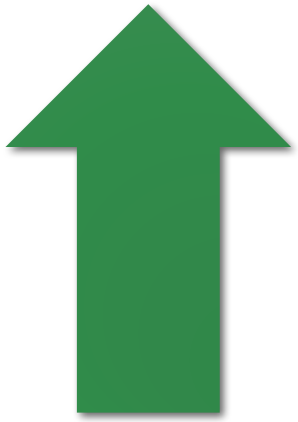
```
def test_gender_misclassification(self):
    data = pd.DataFrame({'gender': ["M","M","F","F"]})
    res = gender_misclassification(data)
    data_result = pd.DataFrame({'gender': ["F","F","M","M"]})
    self.assertTrue(data_result.equals(res))

def test_char_sub_str(self):
    string_test = ["Gaurika", "Shraddha", "Tennyson"]
    res = char_sub_str(string_test)
    for i in range(len(string_test)):
        # print string_test[i]
        # print res[i]
        self.assertFalse(string_test[i] is res[i])

def test_char_omission(self):
    string_test = ["Gaurika", "Shraddha", "Tennyson"]
    res = char_omission(string_test)
    for i in range(len(string_test)):
        # print string_test[i]
        # print res[i]
        self.assertFalse(string_test[i] is res[i])

def test_char_sub_date(self):
    date_test = [datetime.datetime.strptime("2010/2/10", "%Y/%m/%d"), datetime.datetime.strptime("2010/5/3", "%Y/%m/%d")]
    res = char_sub_date(date_test)
    for i in range(len(date_test)):
        # print date_test[i]
        # print res[i]
        self.assertFalse(date_test[i] is res[i])
```

CHALLENGES and SOLUTIONS

Because smart people learn from others mistakes…

Ensure that Chinese First Name Maps to a Chinese Last Name, Indian First Name to Indian Last Name.

Added an extra attribute: “Detailed Race” to further split the Asian Race to Indian and Chinese.



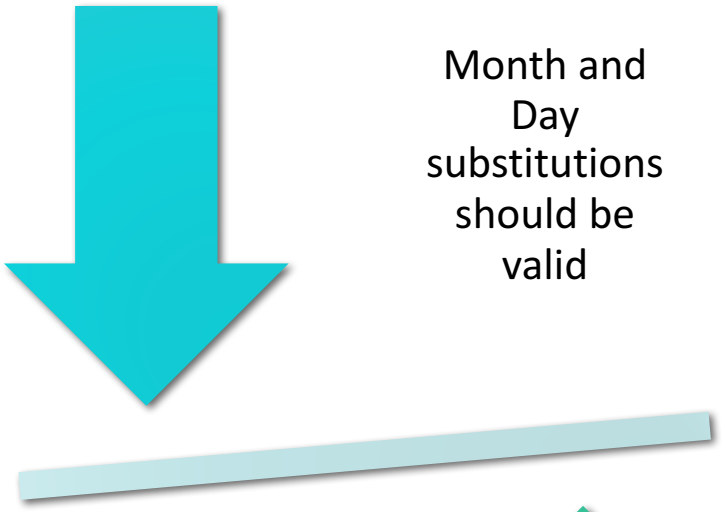
```
#Exract all chinese rows from data
asian_chinese_fn=asian_data_fn.loc[(asian_data_fn['detailed_race']=='Chinese')]
#Check how many records are there
chinese_rows = len(asian_chinese_fn)
#Extract Chinese Last Names
asian_chinese_ln=asian_dataset_ln.loc[(asian_dataset_ln['detailed_race']=='Chinese')]
#Select as many Last names randomly as you need
last_names=asian_chinese_ln.sample(n=chinese_rows,replace=True)

asian_chinese_fn.insert(1,'last',last_names['last'].tolist())

chinese_rows = len(asian_chinese_fn)
```

```
# Exract all Indian rows from data
asian_indian_fn = asian_data_fn.loc[(asian_data_fn['detailed_race'] == 'Indian')]
# Check how many records are there
indian_rows = len(asian_indian_fn)
#print "no of rows: ",indian_rows
# Extract Indian Last Names
asian_indian_ln = asian_dataset_ln.loc[(asian_dataset_ln['detailed_race'] == 'Indian')]
# Select as many Last names randomly as you need
last_names = asian_indian_ln.sample(n= indian_rows, replace=True)

asian_indian_fn.insert(1, 'last', last_names['last'].tolist())
```

Character Insertion cannot be random.

Created a dictionary where each letter could be replaced by a set of letters only.

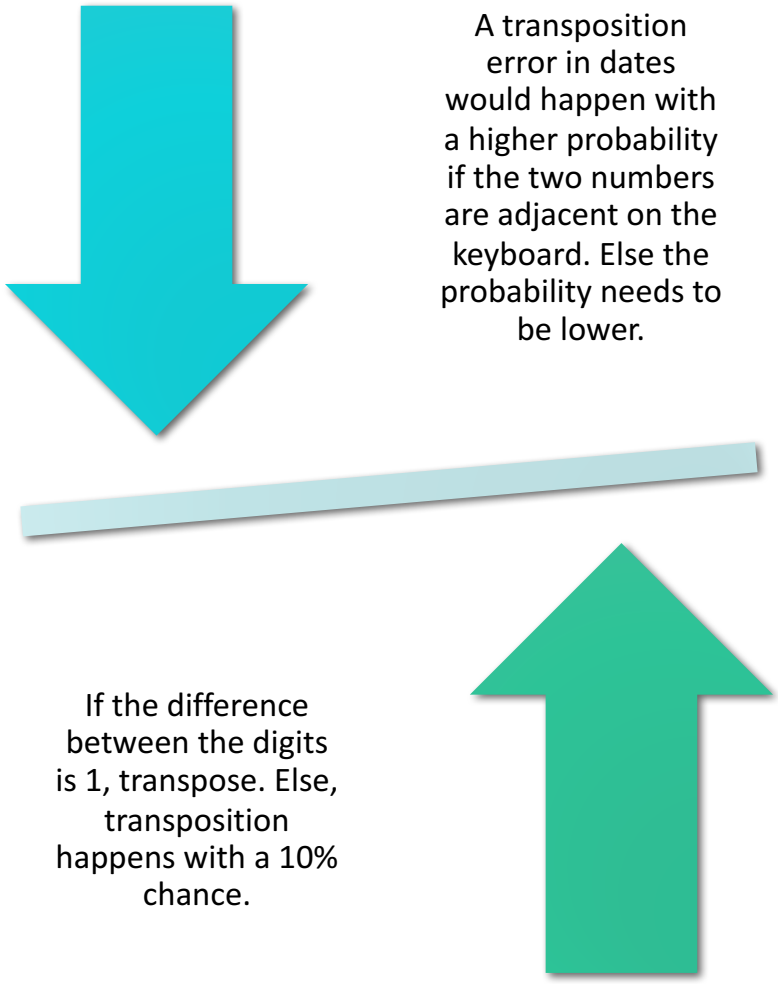
```
"""
characters = 'qwertyuioplkjhgfdsazxcvbnm'
close_keys = {"a": ["a","q","w","s","z","x"],
              "b": ["b","v","g","h","n"],
              "c": ["c","x","d","f","v"],
              "d": ["d","s","e","r","f","x","c"],
              "e": ["e","w","s","d","r"],
              "f": ["f","r","t","d","g","c","v"],
              "g": ["g","t","y","f","h","v","b"],
              "h": ["h","y","u","g","j","b","n"],
              "i": ["i","u","j","k","o"],
              "j": ["j","u","i","h","k","n","m"],
              "k": ["k","i","o","j","l","m"],
              "l": ["l","o","p","k"],
              "m": ["m","n","j","k"],
              "n": ["n","b","h","j","m"],
              "o": ["o","i","k","l","p"],
              "p": ["p","o","l"],
              "q": ["q","a","w"],
              "r": ["r","e","d","f","t"],
              "s": ["s","w","e","a","d","z","x"],
              " PEP 8: missing whitespace after ','
              "u": ["u","y","h","j","i"],
              "v": ["v","c","f","g","b"],
              "w": ["w","q","a","s","e"],
              "x": ["x","z","s","d","c"],
              "y": ["y","t","g","h","u"],
              "z": ["z","a","s","x"]}
print "insertion before", arr
arr_res = []
for i in range(0, len(arr)):
```



Month and Day substitutions should be valid

Check for leap years, months and days.

```
    elif choice == 1:
        if int(day) == 31:
            months_with_31_days = ["1", "3", "5", "7", "8", "10", "12"]
            month = str(random.choice(months_with_31_days))
        elif int(day) == 30 or int(day) == 29:
            months_with_30_days = ["1", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12"]
            month = str(random.choice(months_with_30_days))
        else:
            month = str(random.randint(1, 12))
        # print "error induced month: ",month
    elif choice == 2:
        if int(month) in [1, 3, 5, 7, 8, 10, 12]:
            day = str(random.randint(1, 31))
        elif int(month) in [4, 6, 9, 11]:
            day = str(random.randint(1, 30))
        else:
            day = str(random.randint(1, 28))
    date_err = '/'.join([year, month, day])
    date = datetime.datetime.strptime(date_err, "%Y/%m/%d")
    new_dates += [date]
return new_dates
```




```
if choice == 0:

    year = list(year)
    if abs(int(year[2]) - int(year[3])) == 1 or random.randint(0, 10) == 1:
        buf = year[3]
        year[3] = year[2]
        year[2] = buf
    year = ''.join(year)
    # print "Error induced year: ",year
```

If user does not define race percentages, generate randomly such that sum of all equals 1.

Function to generate random numbers such that sum always equals a specified total.

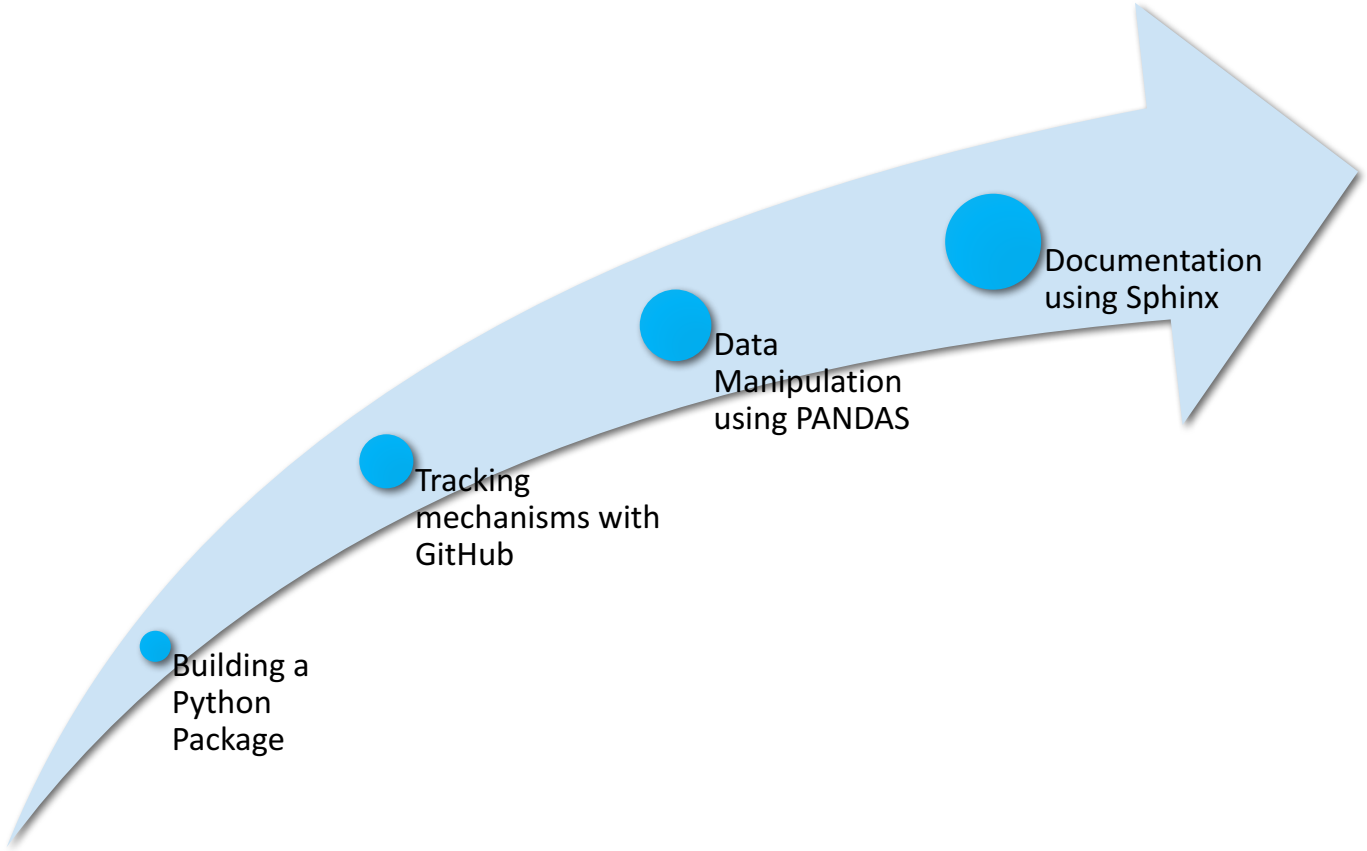
```
def sum_num_terms_equals_total(num_terms, total):
    """
blank lines, found 1 ndomly chosen list of n positive integers summing to total.
        Each such list is equally likely to occur.

        Args:
            num_terms (int): Number of terms to generate
            total (int/float): The sum total of the number of terms.

        Returns:
            num (list): A list of numbers generated.
    """

    num = []
    if type(total) == float:
        while (total >= 0):
            term = round(random.uniform(0, total), 2)
            # print term
            num += [term]
            total -= term
            num_terms -= 1
            # print "num_terms: ", num_terms
            if (num_terms == 1):
                num += [round(total, 2)]
                break;
        # print "num: ", num
```

And what if the random number generated is zero?

Used flags to check if the random function picked up a zero. Induce an error if flag is set to true.

```
if(flag_insert):
    data_err=insertion_error
    if(flag_omission):
        data_err=pd.concat([data_err,omission_error])
    if(flag_sub==True):
        data_err=pd.concat([data_err,sub_error])
    if(flag_gendermis):
        data_err=pd.concat([data_err,gender_mis_error])
    if(flag_transpo):
        data_err=pd.concat([data_err,transpo_error])
elif (flag_omission):
    data_err = omission_error
    if (flag_sub):
        data_err = pd.concat([data_err, sub_error])
    if (flag_gendermis):
        data_err = pd.concat([data_err, gender_mis_error])
    if (flag_transpo):
        data_err = pd.concat([data_err, transpo_error])
elif (flag_sub):
    data_err = sub_error
    if (flag_gendermis):
        data_err = pd.concat([data_err, gender_mis_error])
    if (flag_transpo):
        data_err = pd.concat([data_err, transpo_error])
elif (flag_gendermis):
    data_err = gender_mis_error
    if (flag_transpo):
        data_err = pd.concat([data_err, transpo_error])
elif (flag_transpo):
        data_err = transpo_error
```

The LEARNING curve



USING SYN EHR

ANALYSIS OF OUTPUT

ASSUMPTIONS AND SHORTCOMINGS

Random date generation in month of February only between 1 and 28.

A record may have more than one type of error

Race percentages is on the number of unique records generated, not the entire dataset.

Native Americans and Alaskans have similar surnames.

The database barely has Muslim Names.

Error generation is not based on phonetics

The percentage of error induced records is 10%.

NEXT STEPS

- Publish Package on PyPi.
- Make a detailed documentation.
- Induce errors based on phonetics.
- Write Scripts to crawl for first names and last names to auto update master database.
- Simulate Clinical Texts.
- Option for user to control error percentage.
- Detailed race for Native American and Alaskan Races as well and other races.
- Better representation of other country names (Poland, Germany, France)

QUESTIONS?