STA 141B Assignment 1

Due **January 26, 2024** by **11:59pm**. Submit your work by uploading it to Gradescope through Canvas.

Instructions:

1. Provide your solutions in new cells following each exercise description. Create as many new cells as necessary. Use code cells for your Python scripts and Markdown cells for explanatory text or answers to non-coding questions. Answer all textual questions in complete sentences.
2. The use of assistive tools is permitted, but must be indicated. You will be graded on you proficiency in coding. Produce high quality code by adhering to proper programming principles.
3. Export the .jpynb as .pdf and submit it on Gradescope in time. To facilitate grading, indicate the area of the solution on the submission. Submissions without indication will be marked down. No late submissions accepted.
4. The total number of points is 10.

Exercise 1

This exercise will review basic concepts of programming. Only use pure python code and no methods (like `str.find`) that are optimized in, e.g., `C`. Likewise, do not use any packages except those suitable for parallelization in part (c).

(a) Write a recursive function `seq_count(x, ...)` that returns length of the longest subsequence of identical elements in the sequence object `x`. Run:

```
seq_count([[1], [1], [1], 1, 3, 3, 2, 2, 4, 0])
seq_count(('G', 'g', 'a', "a", "a", '''a''', 2, 's', 's'))
seq_count([3, 1, int(True), 1, 1, 1, 3, 3])
seq_count((1, 3, None, 3, 3, 1, 3, 3, 4, 0))
```

```
def seq_count(x, curr_count = 1, max_count = 1, curr_char = None):
    if not x:
        return max_count
    if x[0] == curr_char:
        curr_count += 1
    else:
        curr_count = 1
    max_count = max(curr_count, max_count)
    return seq_count(x[1:], curr_count, max_count, x[0])


seq_count([1, 3, 1, 1, 3, 3, 4, 4, 4])
```

```
3
```

```
print(seq_count([[1], [1], [1], 1, 3, 3, 2, 2, 4, 0]))
print(seq_count(('G', 'g', 'a', "a", "a", '''a''', 2, 's', 's')))
print(seq_count([3, 1, int(True), 1, 1, 1, 3, 3]))
print(seq_count((1, 3, None, 3, 3, 1, 3, 3, 4, 0)))
print(seq_count((1, 3, 1, 1, 1, '1', 1, [3, 3, 3, 3], 3, 4, 0)))
```

```
3
4
5
2
3
```

(b) Write a function `pattern_count(x, pattern, ...)` that takes the two iterable objects `x` and `pattern` and returns the length of the longest subsequence of `pattern`. Run:

```
pattern_count('CGGACTACTAGACT', 'ACT')
pattern_count((1, (1, 1, 1, 1), 2, 1, 1, 1), [1, 1])
pattern_count(['ab', 'ab', 'a', 'a', 'b'], ('ab',))
```

```
def pattern_count(sequence, pattern):
    # Initialize variables to keep track of current match and longest
match
    current_count = 0
    max_count = 0
    pattern_counter = 0
    for i in sequence:
        if i == pattern[pattern_counter]:
            pattern_counter += 1
            if pattern_counter == len(pattern):
                current_count += 1
                max_count = max(current_count, max_count)
                pattern_counter = 0
        else:
            current_count = 0
    return max_count

    # Iterate through the sequence
print(pattern_count('CGGACTACTAGACT', 'ACT'))
print(pattern_count((1, (1, 1, 1, 1), 2, 1, 1, 1), [1, 1]))
print(pattern_count(['ab', 'ab', 'a', 'a', 'b'], ('ab',)))
```

```
2
2
2
```

```
pattern_count([0, 1, 2, 1, 2, 3, 1, 2, 1, 2, 1, 2, 4, 1, 2], (1, 2))
#should be 3
```

```
3
```

```
pattern_count([], [2])

0

pattern_count(['ab', 'ab', 'a', 'a', 'b'], 'ab') # elements in pattern
must be identical to elements in x!

1

pattern_count((1, (1, 1, 1, 1), 2, 1, 1, 1), [1, 1])

2
```

(c) For a long string, write code that takes strings `x`, `pattern`, and an integer `n_splits`, and uses a suitable concurrency method to search for repeating patterns using `pattern_count` from (b). To this end, partition `x` into `n_splits` parts and search each of them individually. Make sure not to split where a pattern is present! Run:

```
from random import choices, seed

seed(2024)
x = "".join(choices('01', k = 5_000))
pattern = "01"
n_splits = 50

# here is your code
```

Hint: You can use the fast `x.find(pattern)` *to check your code.*

```
from random import choices, seed
import concurrent.futures, threading

seed(2024)
x = "".join(choices('01', k = 5_000))
pattern = "01"
n_splits = 50
# here is your code

# split_patterns = [x[i:i+split_size] for i in range(0, len(x),
split_size)]
split_patterns = []
def pattern_splitter(x, pattern, n_splits):
    split_size = len(x) // n_splits
    min_thresh = int((0.2 * split_size) // 1)
    indexing = 0
    for i in range(split_size - 1):
        start_index = indexing
        indexing += min_thresh
        end_index = indexing + len(pattern)
        while pattern_count(x[indexing:end_index], pattern) != 0:
```

```
            indexing += 1
            end_index += 1
        indexing = end_index
        split_patterns.append(x[start_index:indexing+1])
    split_patterns.append(x[indexing:-1])

pattern_splitter(x, pattern, n_splits)

def pattern_search(args):
    sequence, pattern = args
    return pattern_count(sequence, pattern)

def total_pattern_count(split_patterns: list, pattern):
    arg_list = [(curr_pattern, pattern) for curr_pattern in
split_patterns]
    with concurrent.futures.ThreadPoolExecutor(max_workers=8) as
executor:
        results = list(executor.map(pattern_search, arg_list))
    return max(results)
total_pattern_count(split_patterns, pattern)
```

```
7
```

Exercise 2

In this exercise, we will generate (pseudo-)random numbers using the inversion and accept-reject method. In order to generate the random numbers you are only allowed draw from the Uniform distribution and use

```
from random import uniform
from scipy.special import binom
from numpy import sqrt, pi, exp, tan, cumsum
from scipy.stats import probplot
import pandas as pd
import matplotlib.pyplot as plt
```

Inversion method: Let F *be a distribution function from which we want to draw. Define the quantile function* $F^{-1}(u) = inf\{x : F(x) \geq u, 0 \leq u \leq 1\}$*. Then, if* $U \sim Unif[0,1)$*,* $F^{-1}(U)$ *has distribution function* F*.*

Accept-reject: Let f *be a density function from which we want to draw and there exists a density* g *from which we can draw (e.g., via the inversion method) and for which there exists a constant* c *such that* $f(x) \leq c\, g(x)$ *for all* x*. The following algorithm generates a random variable* X *with density function* f*.*

1. Generate a random variable X from density g
2. Generate a random variable $U \sim Unif[0,1)$ (independent from X)
3. If $U\, c\, g(X) \leq f(X)$, return X, otherwise repeat 1.-3.

The number of iterations needed to successfully generate X is itself a random variable, which is geometrically distributed with the success (acceptance) probability $p = P\left(U \, c \, g\left(X\right) \leq f\left(X\right)\right)$. Hence, the expected number of iterations is $1/p$. Some calculations show that $p = 1/c$.

(a) Generate $10\,000$ samples from $Bin\left(10, 0.4\right)$ using **(i)** the inversion method directly and **(ii)** using the inversion method to draw corresponding Bernoulli distributed samples. **(iii)** Plot the resulting empirical distribution functions and add the theoretical distribution function in one figure.

```
from random import uniform
from scipy.special import binom
from numpy import sqrt, pi, exp, tan, cumsum
from scipy.stats import probplot, binom
import pandas as pd
import matplotlib.pyplot as plt
```

```
/var/folders/nv/5j9xydr90s30r_9rw5pwhf640000gn/T/
ipykernel_58215/4100087547.py:5: DeprecationWarning:
Pyarrow will become a required dependency of pandas in the next major
release of pandas (pandas 3.0),
(to allow more performant data types, such as the Arrow string type,
and better interoperability with other libraries)
but was not found to be installed on your system.
If this would cause problems for you,
please provide us feedback at
https://github.com/pandas-dev/pandas/issues/54466

  import pandas as pd
```

```
def direct_inverse_method(n, p, sample_size):
    values = []
    for i in range(sample_size):
        u = uniform(0,1)
        inversed_u = binom.ppf(u, n, p) #this gives the inverse cdf of
each value
        values.append(inversed_u)
    return values

direct_inverse_samples = direct_inverse_method(10, 0.4, 10000)
direct_inverse_samples.sort()
```

We will first get values using the inverse method function defined in part 1. Then, divide each value by 10 to get a probability value, and if those values are greater than 0.4 which is the threshold earlier defined, it will take a value of 1 (success) or else a 0 (fail)

```
bernoulli_samples = []
for i in range(10000):
    total = 0
    for j in range(10):
```

```
        total+= binom.ppf(uniform(0,1), 1, 0.4)
    bernoulli_samples.append(total)


direct_empirical = []
for i in range(11):
    direct_empirical.append((direct_inverse_samples.count(i))/10000)
direct_empirical = cumsum(direct_empirical)
direct_empirical
```

```
array([0.0078, 0.048 , 0.169 , 0.3788, 0.635 , 0.8357, 0.9483, 0.9908,
       0.9987, 0.9999, 1.    ])
```

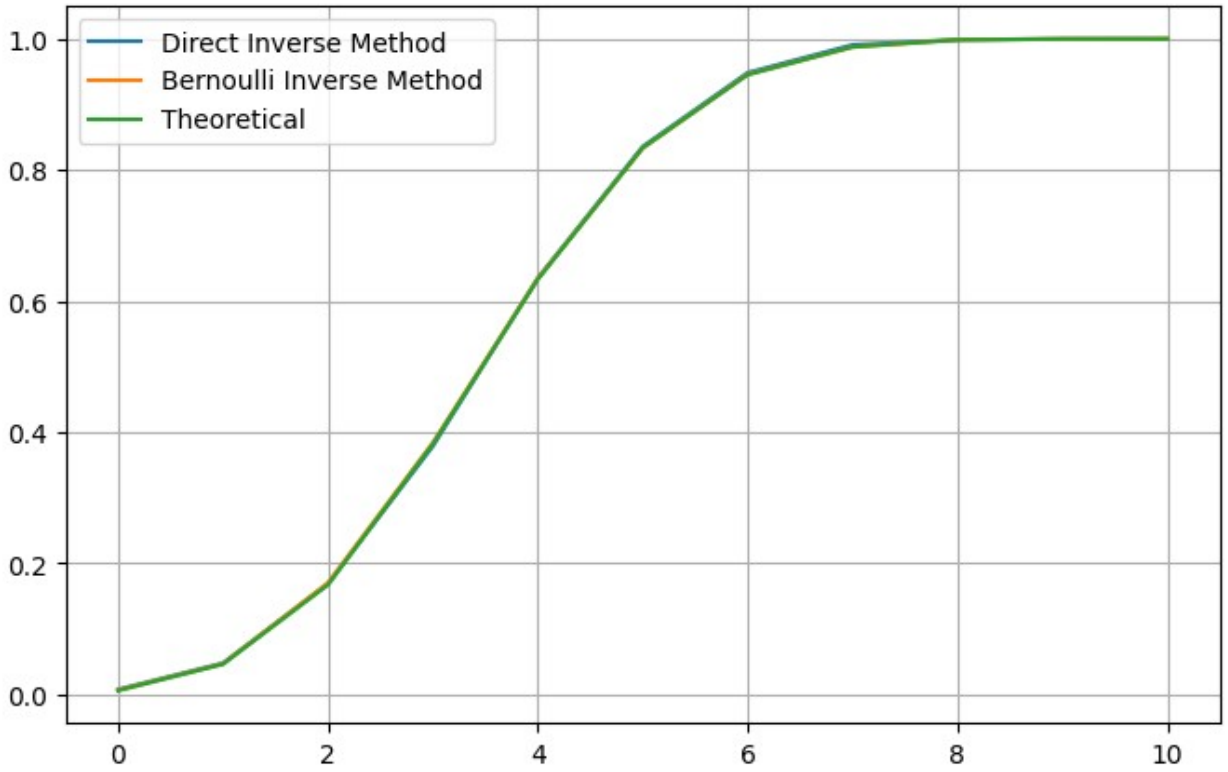
```
bernoulli_samples.sort()
bernoulli_empirical = []
for i in range(11):
    bernoulli_empirical.append((bernoulli_samples.count(i))/10000)

bernoulli_empirical = cumsum(bernoulli_empirical)
bernoulli_empirical
```

```
array([0.0069, 0.0469, 0.1704, 0.3841, 0.6344, 0.8333, 0.9459, 0.9875,
       0.9988, 0.9999, 1.    ])
```

```
x_val = list(range(11))
cdf_theoretical = cumsum([binom.pmf(k, 10, 0.4) for k in x_val])
cdf_theoretical
```

```
array([0.00604662, 0.0463574 , 0.16728975, 0.3822806 , 0.63310326,
       0.83376138, 0.94523812, 0.98770545, 0.99832228, 0.99989514,
       1.        ])
```

```
plt.figure(figsize=(8, 5))
plt.plot(x_val,direct_empirical, label = 'Direct Inverse Method')
plt.plot(x_val,bernoulli_empirical, label = 'Bernoulli Inverse
Method')
plt.plot(x_val,cdf_theoretical, label = 'Theoretical')
plt.legend()
plt.grid(True)
plt.show()
```



(b) Generate 10000 samples from the standard normal distribution using the accept-reject method with candidate density $g(x)=\left(\pi\left(1+x^2\right)\right)^{-1}$ with distribution funciton $G(x)=\tan^{-1}(x)/\pi$ from the standard Cauchy distribution. To this end, **(i)** determine (mathematically or via simulation) the value of $c \geq 1$ closest to one so that $f(x) \leq c\, g(x)$ for all x. **(ii)** Obtain 10000 standard normal random variables using the accept-reject method, generating Cauchy distributed random variables using inversion method. **(iii)** Compare estimated and theoretical acceptance probabilities. **(iv)** Generate a QQ-plot of the generated sample.

```
#To find c, simulate 10,000 x values and find the max ratio where c >=
f(x) / g(x)
sim_x  = [-5 + (i/(1000)) for i in range(10000)]
sim_x = pd.Series(sim_x)
def ratio_calc(x):
    f_x = exp(-x**2 / 2) / sqrt(2 * pi)
    g_x = 1 / (pi * (1 + x**2))
    return f_x/g_x
result_x = sim_x.apply(ratio_calc)
c = max(result_x)
print(c)

1.520346901066281
```

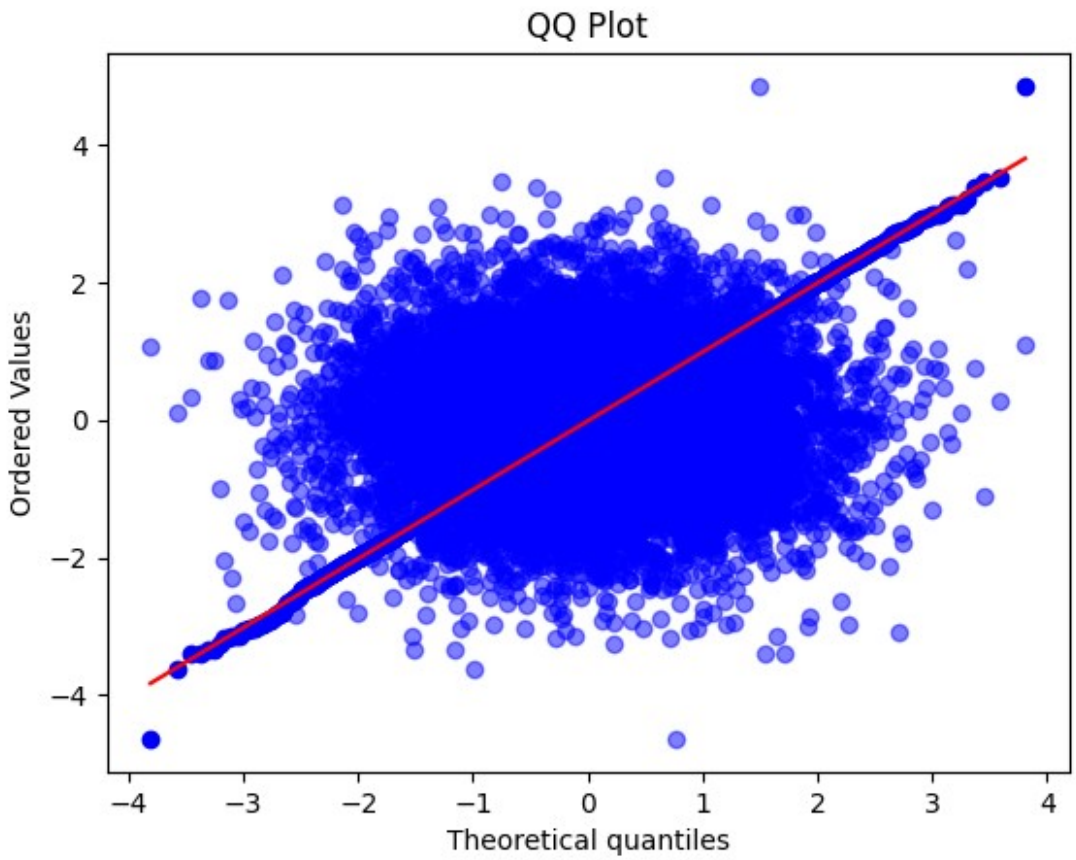
Using a simulation where we generating equally spaced values from -5 to 5, we put each value into the f(x) and g(x), took the ratio of the two for each x value and got the maximum ratio from all 10000 values to get a c value of approximately 1.5203

```
#part 2 (ii)
#inverse of G(x) is tan(pi * u)
nums = []
tot_generated = 0
c = 1.52034
while len(nums) < 10000:
    init_val = uniform(0,1)
    X = tan(pi * init_val)
    U = uniform(0,1)
    Ucg = U * c * (pi*(1 + (X**2)))**(-1)
    fx = (1/sqrt(2*pi))* exp(-(X**2)/2)
    if Ucg <= fx:
        nums.append(X)
    tot_generated += 1
print(tot_generated)
```

```
15196
```

iii) As can be seen previously, in order to generate 10000 values, 15196 total values were generated. 10000/15196 = 0.65806 which means about a 66% acceptance rate. The theoretical acceptance rate is approximately 1/c, which is 1/1.52034 = 0.65775 which is also about a 66% acceptance rate, so they are pretty similar.

```
theor_quant = probplot(nums, plot=plt)[0][0]
plt.scatter(theor_quant, nums, color='blue', alpha=0.5)
plt.title('QQ Plot')
plt.show()
```



Exercise 3

The demographic makeup of regions can offer crucial insights into various socio-economic factors. For policymakers, understanding age distributions can be particularly useful, as it can provide direction for initiatives ranging from educational policy to elderly care. In this section, we will work with a dataset detailing the age distribution across United States counties, broken down into specific age bins.

The files `county_age_dist.csv`, `fips_state.csv` and `fips_county.csv` contain information about the age distribution of counties in selected brackets as well as names and FIPS codes and additional information.

(a, i) Merge all three data frames into one `pandas.DataFrame` object names `data` with appropriate column names. **(ii)** Remove the `info` column. Standardize column names and entries to be capitalized according to spelling rules. Remove any preceding whitespace if present for any entries. Run:

```
data.head(4)
```

```
import pandas as pd
county_age = pd.read_csv('county_age_dist.csv') #3220 lines
fips_county = pd.read_csv('fips_county.csv') #3199 lines
fips_state = pd.read_csv('fips_state.csv') #51 lines
print(county_age.head(), fips_county.head(), fips_state.head())
```

```
   fips   0-17  18-24  25-34  35-44  45-54  55-64  65-74  75-84   85+
0  1001  25941  11422  12315  13828  14000  12697   9594   5430  1945
1  1003  86587  37568  44133  46730  49675  52405  43252  23262  8854
2  1005  11057   6162   6603   5907   6490   6377   5255   2795  1074
3  1007   9671   5241   5788   5472   6707   5563   4270   2555   638
4  1009  25671  11360  12635  13570  14737  14123  12106   6560  2022
fips             name  info
0  01000          Alabama   NaN
1  01001   Autauga County   NaN
2  01003   Baldwin County   NaN
3  01005   Barbour County   NaN
4  01007      Bibb County   NaN       FIPS; STATE
0     01; ALABAMA
1      02; ALASKA
2     04; ARIZONA
3    05; ARKANSAS
4  06; CALIFORNIA
```

```
print(county_age.columns, fips_county.columns, fips_state.columns)
```

```
Index(['fips', '0-17', '18-24', '25-34', '35-44', '45-54', '55-64',
'65-74',
       '75-84', '85+'],
      dtype='object') Index(['fips', ' name', ' info'],
dtype='object') Index(['FIPS; STATE'], dtype='object')
```

```
county_age['fips'] = county_age['fips'].astype(str).str.zfill(5)
data = pd.merge(county_age, fips_county, how='outer', on='fips')

fips_value_placeholder = data['fips']
data['fips'] = data['fips'].astype(str).str[:2]

fips_state[['fips', 'state']] = fips_state['FIPS; STATE'].str.split(';
', expand=True)
fips_state = fips_state.drop('FIPS; STATE', axis=1)
fips_state.head()
```

```
  fips       state
0   01     ALABAMA
1   02      ALASKA
2   04     ARIZONA
3   05    ARKANSAS
4   06  CALIFORNIA
```

```
data = pd.merge(data, fips_state, how = 'outer', on='fips')
data.head()
```

```
  fips     0-17    18-24    25-34    35-44    45-54    55-64    65-74
\
0   01  25941.0  11422.0  12315.0  13828.0  14000.0  12697.0   9594.0
```

```
1   01  86587.0  37568.0  44133.0  46730.0  49675.0  52405.0  43252.0

2   01  11057.0   6162.0   6603.0   5907.0   6490.0   6377.0   5255.0

3   01   9671.0   5241.0   5788.0   5472.0   6707.0   5563.0   4270.0

4   01  25671.0  11360.0  12635.0  13570.0  14737.0  14123.0  12106.0
```

```
     75-84     85+             name  info    state
0   5430.0  1945.0   Autauga County   NaN  ALABAMA
1  23262.0  8854.0   Baldwin County   NaN  ALABAMA
2   2795.0  1074.0   Barbour County   NaN  ALABAMA
3   2555.0   638.0      Bibb County   NaN  ALABAMA
4   6560.0  2022.0    Blount County   NaN  ALABAMA
```

```
data['fips'] = fips_value_placeholder
data = data.drop(' info', axis = 1)
data['state'] = data['state'].str.lower().str.capitalize()
data.rename(columns={'fips': 'FIPS'}, inplace=True)
data.rename(columns={' name': 'Name'}, inplace=True)
data.rename(columns={'state': 'State'}, inplace=True)
data.head()
```

```
    FIPS     0-17    18-24    25-34    35-44    45-54    55-64    65-
74  \
0  01001  25941.0  11422.0  12315.0  13828.0  14000.0  12697.0
9594.0
1  01003  86587.0  37568.0  44133.0  46730.0  49675.0  52405.0
43252.0
2  01005  11057.0   6162.0   6603.0   5907.0   6490.0   6377.0
5255.0
3  01007   9671.0   5241.0   5788.0   5472.0   6707.0   5563.0
4270.0
4  01009  25671.0  11360.0  12635.0  13570.0  14737.0  14123.0
12106.0
```

```
     75-84     85+             Name    State
0   5430.0  1945.0   Autauga County  Alabama
1  23262.0  8854.0   Baldwin County  Alabama
2   2795.0  1074.0   Barbour County  Alabama
3   2555.0   638.0      Bibb County  Alabama
4   6560.0  2022.0    Blount County  Alabama
```

(b) For each county and state, compute the proportion of elderly `CPE` and `SPE` (65 and older) to the total population as well as the proportion of young people `CPY` and `SPY` (24 or younger). Add those values to the data frame. You may ignore all FIPS regions that are not in states. Run:

```
data.head(4)
```

```
data.iloc[:, 1:10] = data.iloc[:, 1:10].apply(pd.to_numeric,
errors='coerce')
data.head
```

```
<bound method NDFrame.head of        FIPS     0-17    18-24    25-34
35-44    45-54    55-64    65-74  \
0     01001  25941.0  11422.0  12315.0  13828.0  14000.0  12697.0
9594.0
1     01003  86587.0  37568.0  44133.0  46730.0  49675.0  52405.0
43252.0
2     01005  11057.0   6162.0   6603.0   5907.0   6490.0   6377.0
5255.0
3     01007   9671.0   5241.0   5788.0   5472.0   6707.0   5563.0
4270.0
4     01009  25671.0  11360.0  12635.0  13570.0  14737.0  14123.0
12106.0
...     ...      ...      ...      ...      ...      ...      ...
...
3281   1990  27016.0  14455.0  14882.0  14168.0  15026.0  14450.0
11928.0
3282  53000   4724.0   2727.0   2092.0   2356.0   2496.0   2972.0
2364.0
3283  54000  11353.0   6431.0   5521.0   5319.0   5788.0   6228.0
4631.0
3284  55000  16068.0   9025.0   8465.0   9199.0   9548.0   9805.0
7926.0
3285  56000  17375.0   8974.0   9422.0   9457.0  10028.0  10672.0
8571.0

        75-84     85+             Name    State
0      5430.0  1945.0   Autauga County  Alabama
1     23262.0  8854.0   Baldwin County  Alabama
2      2795.0  1074.0   Barbour County  Alabama
3      2555.0   638.0      Bibb County  Alabama
4      6560.0  2022.0    Blount County  Alabama
...       ...     ...              ...      ...
3281   6139.0  3159.0              NaN      NaN
3282   1498.0   479.0              NaN      NaN
3283   2218.0   859.0              NaN      NaN
3284   3784.0  2103.0              NaN      NaN
3285   4620.0  2346.0              NaN      NaN

[3286 rows x 12 columns]>
```

```
data['CPY'] = (data.iloc[:, 1:3].sum(axis = 1)) /
(data.iloc[:,1:10].sum(axis = 1))
data['CPE'] = (data.iloc[:, 7:10].sum(axis = 1)) /
(data.iloc[:,1:10].sum(axis = 1))
data.head()
```

```
    FIPS     0-17    18-24    25-34    35-44    45-54    55-64    65-
74  \
0  01001  25941.0  11422.0  12315.0  13828.0  14000.0  12697.0
9594.0
1  01003  86587.0  37568.0  44133.0  46730.0  49675.0  52405.0
43252.0
2  01005  11057.0   6162.0   6603.0   5907.0   6490.0   6377.0
5255.0
3  01007   9671.0   5241.0   5788.0   5472.0   6707.0   5563.0
4270.0
4  01009  25671.0  11360.0  12635.0  13570.0  14737.0  14123.0
12106.0

     75-84     85+             Name    State       CPY       CPE
0   5430.0  1945.0   Autauga County  Alabama  0.348627  0.158334
1  23262.0  8854.0   Baldwin County  Alabama  0.316346  0.192037
2   2795.0  1074.0   Barbour County  Alabama  0.332927  0.176411
3   2555.0   638.0      Bibb County  Alabama  0.324845  0.162575
4   6560.0  2022.0    Blount County  Alabama  0.328336  0.183430
```

```
pop_by_state = data.groupby('State')[['0-17', '18-24', '25-34', '35-
44', '45-54', '55-64', '65-74', '75-84', '85+']].sum()
pop_by_state['Total Pop'] = pop_by_state.iloc[:,:].sum(axis=1)
pop_by_state = pop_by_state.drop(pop_by_state.columns[:9], axis=1)

pop_by_state
```

```
                      Total Pop
State
Alabama                9670608.0
Alaska                 1383156.0
Arizona               13149489.0
Arkansas               5947364.0
California            74746038.0
Colorado              10366853.0
Connecticut            7212616.0
Delaware               1782091.0
District of columbia   1312611.0
Florida               39432216.0
Georgia               19729216.0
Hawaii                 2678974.0
Idaho                  3113537.0
Illinois              25354026.0
Indiana               12982214.0
Iowa                   5921751.0
Kansas                 5658103.0
Kentucky               8757532.0
Louisiana              9597495.0
Maine                  2600418.0
Maryland              11601841.0
```

```
Massachusetts          13685494.0
Michigan               19257846.0
Minnesota               9942928.0
Mississippi             6177970.0
Missouri               11958891.0
Montana                 2006991.0
Nebraska                3611268.0
Nevada                  5626036.0
New hampshire           2590979.0
New jersey             17802603.0
New mexico              4187275.0
New york               40095320.0
North carolina         19389348.0
North dakota            1350668.0
Ohio                   23225879.0
Oklahoma                7155000.0
Oregon                  7876967.0
Pennsylvania           24744520.0
Rhode island            2182415.0
South carolina          9528410.0
South dakota            1582886.0
Tennessee              12857371.0
Texas                  53200892.0
Utah                    5429984.0
Vermont                 1169004.0
Virginia               16112775.0
Washington             14044143.0
West virginia           3842001.0
Wisconsin              10684137.0
Wyoming                 1158930.0
```

```
data = data.merge(pop_by_state, on='State', how='left' )
data.head()
```

```
    FIPS     0-17    18-24    25-34    35-44    45-54    55-64    65-
74  \
0  01001  25941.0  11422.0  12315.0  13828.0  14000.0  12697.0
9594.0
1  01003  86587.0  37568.0  44133.0  46730.0  49675.0  52405.0
43252.0
2  01005  11057.0   6162.0   6603.0   5907.0   6490.0   6377.0
5255.0
3  01007   9671.0   5241.0   5788.0   5472.0   6707.0   5563.0
4270.0
4  01009  25671.0  11360.0  12635.0  13570.0  14737.0  14123.0
12106.0

     75-84     85+             Name    State       CPY       CPE
Total Pop
0   5430.0  1945.0   Autauga County  Alabama  0.348627  0.158334
```

```
9670608.0
1  23262.0  8854.0   Baldwin County  Alabama  0.316346  0.192037
9670608.0
2   2795.0  1074.0   Barbour County  Alabama  0.332927  0.176411
9670608.0
3   2555.0   638.0      Bibb County  Alabama  0.324845  0.162575
9670608.0
4   6560.0  2022.0    Blount County  Alabama  0.328336  0.183430
9670608.0

data.head()

    FIPS     0-17    18-24    25-34    35-44    45-54    55-64    65-
74  \
0  01001  25941.0  11422.0  12315.0  13828.0  14000.0  12697.0
9594.0
1  01003  86587.0  37568.0  44133.0  46730.0  49675.0  52405.0
43252.0
2  01005  11057.0   6162.0   6603.0   5907.0   6490.0   6377.0
5255.0
3  01007   9671.0   5241.0   5788.0   5472.0   6707.0   5563.0
4270.0
4  01009  25671.0  11360.0  12635.0  13570.0  14737.0  14123.0
12106.0

     75-84     85+              Name    State       CPY       CPE
Total Pop
0   5430.0  1945.0   Autauga County  Alabama  0.348627  0.158334
9670608.0
1  23262.0  8854.0   Baldwin County  Alabama  0.316346  0.192037
9670608.0
2   2795.0  1074.0   Barbour County  Alabama  0.332927  0.176411
9670608.0
3   2555.0   638.0      Bibb County  Alabama  0.324845  0.162575
9670608.0
4   6560.0  2022.0    Blount County  Alabama  0.328336  0.183430
9670608.0

data['SPY'] = (data.iloc[:,1:3].sum(axis=1)) / (data.iloc[:,-1])
data['SPE'] = (data.iloc[:,7:10].sum(axis=1)) / (data.iloc[:,-1])
data.head()

    FIPS     0-17    18-24    25-34    35-44    45-54    55-64    65-
74  \
0  01001  25941.0  11422.0  12315.0  13828.0  14000.0  12697.0
9594.0
1  01003  86587.0  37568.0  44133.0  46730.0  49675.0  52405.0
43252.0
2  01005  11057.0   6162.0   6603.0   5907.0   6490.0   6377.0
5255.0
```

```
3  01007   9671.0   5241.0   5788.0   5472.0   6707.0   5563.0
4270.0
4  01009  25671.0  11360.0  12635.0  13570.0  14737.0  14123.0
12106.0

     75-84     85+             Name    State       CPY       CPE
Total Pop  \
0   5430.0  1945.0   Autauga County  Alabama  0.348627  0.158334
9670608.0
1  23262.0  8854.0   Baldwin County  Alabama  0.316346  0.192037
9670608.0
2   2795.0  1074.0   Barbour County  Alabama  0.332927  0.176411
9670608.0
3   2555.0   638.0      Bibb County  Alabama  0.324845  0.162575
9670608.0
4   6560.0  2022.0    Blount County  Alabama  0.328336  0.183430
9670608.0

          SPY       SPE
0  9670608.0  0.001755
1  9670608.0  0.007794
2  9670608.0  0.000943
3  9670608.0  0.000772
4  9670608.0  0.002139

data = data.drop('Total Pop', axis = 1)

data.iloc[1000:1004,:] #different from the sample because i believe
the sample did not account for total population

       FIPS      0-17     18-24     25-34     35-44     45-54     55-
64  \
1000  21015  254825.0  116058.0  134015.0  115104.0  117175.0
121444.0
1001  21017   13931.0    6183.0    6546.0    5392.0    5194.0
4655.0
1002  21019   83441.0   37898.0   43565.0   38611.0   42499.0
46549.0
1003  21021    1042.0     393.0     552.0     474.0     541.0
691.0

        65-74    75-84      85+              Name   State       CPY
CPE  \
1000  80369.0  45953.0  21417.0   Sedgwick County  Kansas  0.368539
0.146805
1001   2959.0   1777.0    679.0     Seward County  Kansas  0.425099
0.114443
1002  32866.0  18560.0   9490.0    Shawnee County  Kansas  0.343271
0.172333
1003    480.0    392.0    176.0   Sheridan County  Kansas  0.302679
```

```
0.221050

             SPY       SPE
1000  5658103.0  0.026111
1001  5658103.0  0.000957
1002  5658103.0  0.010766
1003  5658103.0  0.000185
```