Problem 1

Sub Q1

Consider PT and its 4 generalized tables in Figure 2 and the hierarchies in Figure 1. Answer the following two questions:

1. **6 Points.** Do the 4 generalized tables satisfy the k-anonymity? What are the k? Which tables are the 2-minimal generalization? Why?

My answer:

Reference: 2023-01-18-Intro and K-Anonymity.pdf, P50-53 (K-Anonymity), P59 (k-minimal Generalization)

K-anonymous table: Instead of returning the original table: modify the data such that for each tuple in the new table, there are at least k-1 (*k-1 > 0*) other tuples with the same value for the quasi-identifier.

Quasi-identifier is {Race, ZIP} here. In the new generalized table, there are at least k-1 other tuples with the same value for the quasi-identifier. Each value is associated with an attribute of quasi-identifier at least k times. Hence: k-1 > 0. Thus:
GT[1, 0] satisfiesthe k-anonymity, k = 2.
GT[0, 1] satisfies the k-anonymity, k = 2
GT[1, 1] satisfies the k-anonymity, k = 2 or 3 or 4
GT[0, 2] satisfies the k-anonymity, k = 2 or 3 or 4

Each generalized table of PT satisfies k-anonymity for k = 2.
For GT[0, 1], it's generalized ZIP at one level.
For GT[1, 0], it's generalized Race at one level.
Compared to GT[0, 1] and GT[1, 0] which have already satisfy k=2, GT[1, 1] and GT[0, 2] did more generalization than necessary. Because k-minimal Generalization: A k-anonymization that is not a generalization of another k-anonymization, the 2-minimal generalization is GT[1, 0], GT[0, 1].

Sub Q2

2. **8 Points.** Using the hierarchies in Figure 1, what is the precision of the generalized tables $GT_{[1,0]}$, $GT_{[1,1]}$, $GT_{[0,2]}$, and $GT_{[0,1]}$ of PT shown in Figure 2? Please show the detailed intermediate steps, instead of the final value.

My answer:
Reference: 2023-01-18-Intro and K-Anonymity.pdf, P63 (Precision)

Here is the formula for precision:

Precision: *average height of generalized values*, normalized by Value Generalization Hierarchy (VGH) *depth per attribute* per record

$$Prec(\mathrm{RT}) = 1 - \frac{\sum_{i=1}^{N_A}\sum_{j=1}^{N}\frac{h}{|\mathrm{DGH}_{A_i}|}}{|\mathrm{PT}| \bullet |N_A|}$$

- N_A : number of attributes
- |PT|=N: private table size, i.e., number of rows/tuples
- h: height of generalized value
- $|\mathrm{DGH}_{A_i}|$: depth of the VGH for attribute A_i

For each generalized table, the values in common are:

N_A = the number of attributes = 2, attributes are Race, ZIP;

| PT | = N = private table size = number of tuples for each generalized table = 8

| DGH_{Race} | = depth of the VGH(Value Generalization Hierarchy) for 'Race' = 2

| DGH_{ZIP} | = depth of the VGH(Value Generalization Hierarchy) for 'ZIP' = 3

GT[1, 0] satisfies the k-anonymity, k = 2

h_{race} = height of generalized value of 'Race' = 1

h_{ZIP} = height of generalized value of 'ZIP' = 0

$\because N = 8,\ N_A = 2$

$$\sum_{i=1}^{N_A}\sum_{j=1}^{N}\frac{h}{|\mathrm{DGH}_{A_i}|} = \sum_{i=1}^{2}\sum_{j=1}^{8}\frac{h}{\mathrm{DGH}_{A_i}} \Rightarrow \sum_{i=1}^{2}\frac{h}{\mathrm{DGH}_{A_i}} = \frac{h_{Race}}{\mathrm{DGH}_{Race}} + \frac{h_{zip}}{\mathrm{DGH}_{zip}}$$

$$\sum_{j=1}^{8}\frac{h}{\mathrm{DGH}_{A_i}} \text{ as a constant } = \sum_{j=1}^{8} 1 = 8 \qquad = \frac{1}{2} + \frac{0}{3} = \frac{1}{2} \Rightarrow \therefore \sum_{i=1}^{N_A}\sum_{j=1}^{N}\frac{h}{|\mathrm{DGH}_{A_i}|} = \frac{1}{2}\times 8 = 4$$

$$|\mathrm{PT}| \bullet |N_A| = 8 * 2 = 16 \Rightarrow \frac{\sum_{i=1}^{N_A}\sum_{j=1}^{N}\frac{h}{|\mathrm{DGH}_{A_i}|}}{|\mathrm{PT}| \bullet |N_A|} = 4 / 16 = 0.25.$$

Hence:

$$Prec(\mathrm{RT}) = 1 - \frac{\sum_{i=1}^{N_A}\sum_{j=1}^{N}\frac{h}{|\mathrm{DGH}_{A_i}|}}{|\mathrm{PT}| \bullet |N_A|} = 1 - 0.25 = 0.75.$$ The precision of GT[1, 0] =0.75.

GT[1, 1] satisfies the 4-anonymity, k = 4

h_{race} = height of generalized value of 'Race' = 1

h_{ZIP} = height of generalized value of 'ZIP' = 1

$\because N = 8,\ N_A = 2$

$$\sum_{i=1}^{N_A}\sum_{j=1}^{N}\frac{h}{|\mathrm{DGH}_{A_i}|} = \sum_{i=1}^{2}\sum_{j=1}^{8}\frac{h}{\mathrm{DGH}_{A_i}}$$

$$\sum_{j=1}^{8}\frac{h}{\mathrm{DGH}_{A_i}} \text{ as a constant } = \sum_{j=1}^{8} 1 = 8 \Rightarrow \sum_{i=1}^{2}\frac{h}{\mathrm{DGH}_{A_i}} = \frac{h_{Race}}{\mathrm{DGH}_{Race}} + \frac{h_{zip}}{\mathrm{DGH}_{zip}} = \frac{1}{2} + \frac{1}{3} = \frac{5}{6}$$

$$\sum_{i=1}^{N_A}\sum_{j=1}^{N}\frac{h}{|\mathrm{DGH}_{A_i}|} = 8 * \frac{5}{6} = \frac{20}{3} \Rightarrow |\mathrm{PT}| \bullet |N_A| = 8 * 2 = 16 \Rightarrow \frac{\sum_{i=1}^{N_A}\sum_{j=1}^{N}\frac{h}{|\mathrm{DGH}_{A_i}|}}{|\mathrm{PT}| \bullet |N_A|} = \frac{20}{3} * \frac{1}{16}.$$

Hence:

$$Prec(\mathrm{RT}) = 1 - \frac{\sum_{i=1}^{N_A}\sum_{j=1}^{N}\frac{h}{|\mathrm{DGH}_{A_i}|}}{|\mathrm{PT}| \bullet |N_A|} = 1 - \frac{20}{3} * \frac{1}{16} \approx 0.58.$$ The precision of GT[1, 1] = 0.58.

GT[0, 2] satisfies the 4-anonymity, k = 4

h_{race} = height of generalized value of 'Race' = 0

h_{ZIP} = height of generalized value of 'ZIP' = 2

$\because N = 8, N_A = 2$

$$\sum_{i=1}^{N_A}\sum_{j=1}^{N}\frac{h}{|DGH_{Ai}|} = \sum_{i=1}^{2}\sum_{j=1}^{8}\frac{h}{DGH_{Ai}}$$

$\sum_{j=1}^{8}\frac{h}{DGH_{Ai}}$ as a constant $= \sum_{j=1}^{8} 1 = 8 \Rightarrow \sum_{i=1}^{2}\frac{h}{DGH_{Ai}} = \frac{h_{Race}}{DGH_{Race}} + \frac{h_{zip}}{DGH_{zip}} = \frac{0}{2} + \frac{2}{3} = \frac{2}{3}$

$$\sum_{i=1}^{N_A}\sum_{j=1}^{N}\frac{h}{|DGH_{Ai}|} = 8*\frac{2}{3} = \frac{16}{3} \Rightarrow |PT|\bullet|N_A| = 8 * 2 = 16 \Rightarrow \frac{\sum_{i=1}^{N_A}\sum_{j=1}^{N}\frac{h}{|DGH_{Ai}|}}{|PT|\bullet|N_A|} = \frac{16}{3} * \frac{1}{16}.$$

Hence: $Prec(RT) = 1 - \frac{\sum_{i=1}^{N_A}\sum_{j=1}^{N}\frac{h}{|DGH_{Ai}|}}{|PT|\bullet|N_A|} = 1 - \frac{16}{3} * \frac{1}{16} \approx 1 - \frac{1}{3} = \frac{2}{3} \approx 0.67$. The precision of GT[0, 2] = 0.67

GT[0, 1] satisfies the 2-anonymity, k = 2

h_{race} = height of generalized value of 'Race' = 0

h_{ZIP} = height of generalized value of 'ZIP' = 1

$\because N = 8, N_A = 2$

$$\sum_{i=1}^{N_A}\sum_{j=1}^{N}\frac{h}{|DGH_{Ai}|} = \sum_{i=1}^{2}\sum_{j=1}^{8}\frac{h}{DGH_{Ai}}$$

$\sum_{j=1}^{8}\frac{h}{DGH_{Ai}}$ as a constant $= \sum_{j=1}^{8} 1 = 8 \Rightarrow \sum_{i=1}^{2}\frac{h}{DGH_{Ai}} = \frac{h_{Race}}{DGH_{Race}} + \frac{h_{zip}}{DGH_{zip}} = \frac{0}{2} + \frac{1}{3} = \frac{1}{3}$

$$\sum_{i=1}^{N_A}\sum_{j=1}^{N}\frac{h}{|DGH_{Ai}|} = 8*\frac{1}{3} = \frac{8}{3} \Rightarrow |PT|\bullet|N_A| = 8 * 2 = 16 \Rightarrow \frac{\sum_{i=1}^{N_A}\sum_{j=1}^{N}\frac{h}{|DGH_{Ai}|}}{|PT|\bullet|N_A|} = \frac{8}{3} * \frac{1}{16}.$$

Hence:

$Prec(RT) = 1 - \frac{\sum_{i=1}^{N_A}\sum_{j=1}^{N}\frac{h}{|DGH_{Ai}|}}{|PT|\bullet|N_A|} = 1 - \frac{8}{3} * \frac{1}{16} \approx 1 - \frac{1}{6} = \frac{5}{6} \approx 0.83$. The precision of GT[0, 1] = 0.83

Sub Q3

Consider the health PT in Figure 3, answer the following question:

1. **6 Points.** Finding a 2-minimal distortion for the health PT table in Figure 3.

My answer:

The description of the question is not clear. I would like to answer this question from two different perspectives.

Way 1: Given we need to answer the question based on the above sub-questions, then Quasi-identifier is {Race, ZIP}.

Prec(GT[0, 1]) = 0.83 is greater than Prec(GT[1, 0]) = 0.75. It means: for the generalized table which satisfies k=2, GT[0, 1] has the least distortion. So we find GT[0, 1] is the 2-minimal generalization. Hence, a 2-minimal distortion for the health PT table in Figure 3 is GT[0, 1].

Way 2: Given we need to answer the question based on Quasi-identifier is { Race, Birthdate, Gender, Zip }. The attribute in the generalized table is GT [Race, Birthdate, Gender, Zip] and 2-minimal distortion: k = 2; Since k = 2 in the description, we only need to find the generalized tables that satisfy k-anonymity where k = 2 where 2-anonymous relation should have at least 2 tuples with the same values on Domain($Race_i$) * Domain(Birth $Date_j$) * Domain($Gender_m$) * Domain(ZIP_n).
Here are the hierarchies I would use:

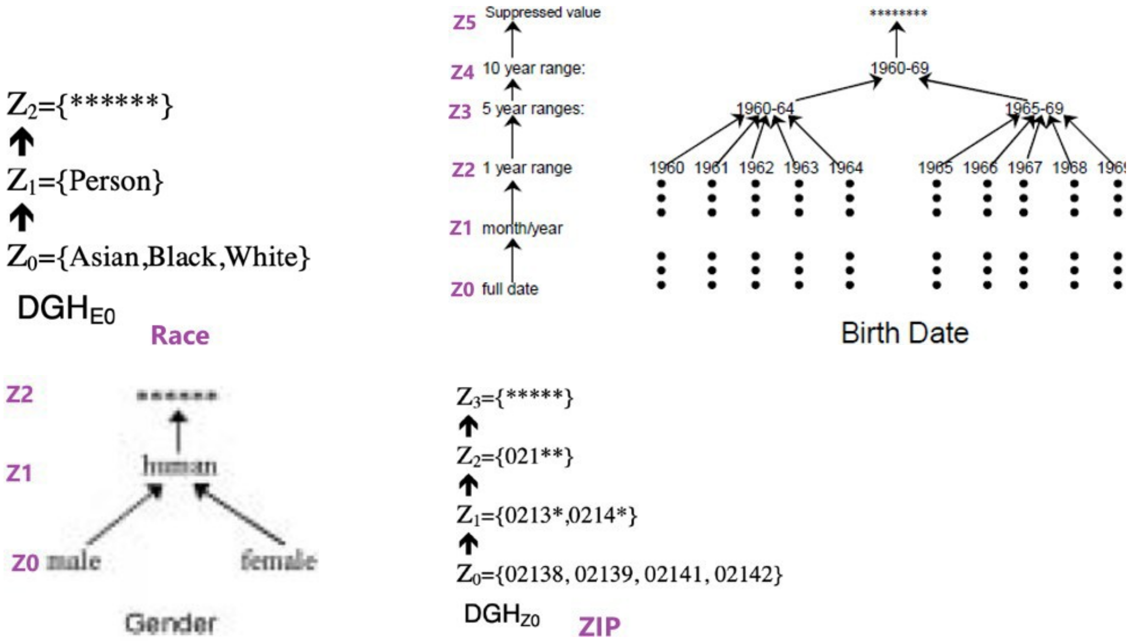


Figure of Quasi-identifier Hierarchies is partly from HW1 and partly from an International Journal on Uncertainty[1]

We get that $MaxLevel_{Race} = 2$, $MaxLevel_{BirthDate} = 5$, $MaxLevel_{Gender} = 2$, $MaxLevel_{ZIP} = 3$.
We can find that birthdate in the original table (the health PT table in Figure 3) is a unique identifier since it has no duplicated values. Because of the unique identifier (Birth Date), GT[0, 0, 0, 0], GT[0,1, 0, 0], GT[0, 0, 1, 0], GT[0, 0, 0, 1], GT[1,0, 0, 0] could not be k-anonymous. We can try to go up based on the unique identifier to make it easy to find minimal distortion. Hence, GT[Race, Birthdate, Gender, Zip] = GT[0, 2, 0, 0] satisfies k = 2 which is minimal.

k-minimal Distortion: A k-minimal generalization that has the least distortion

$$\text{Distortion D} = \frac{\sum_{\text{attrib } i} \frac{\text{Current level of generalization for attribute i}}{\text{Max level of generalization for attribute i}}}{\text{Number of attributes}} = \text{Distortion} = \frac{\frac{0}{2} + \frac{2}{5} + \frac{0}{2} + \frac{0}{3}}{4} = \frac{2}{5} \times \frac{1}{4} = 0.1$$

Problem 2

Source Code Info

The source code script of problem 2 is hw1-q2.ipynb

Algorithm Main Idea

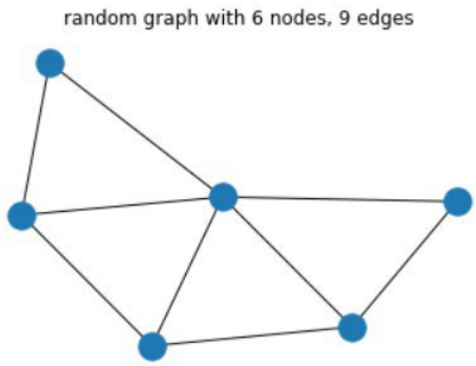
1. Sort the degree sequence d of a graph in descending order with networkZ.
2. Initialize a 2D array dp with dimensions (n+1) x (k+1), where dp[i][j] represents the minimum cost to achieve a k-anonymous degree sequence from the first i nodes with j being the maximum degree.
3. Initialize dp[0][0] = 0.

4. Loop through i=1 to n and j=1 to k, and update dp[i][j], for each x from j to d[i-1], calculate the cost as dp[i][j] = min(dp[i][j], dp[i-1][x] + abs(j - d[i-1])).

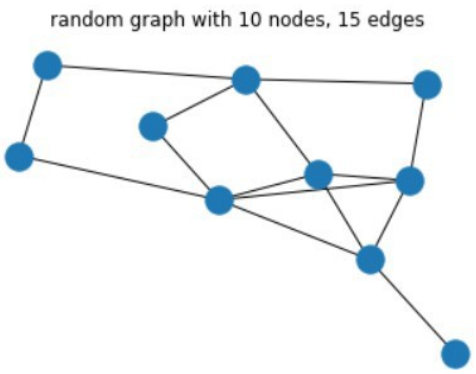
The detailed code design is below code and explanation:

In [91]:
```
from pylab import *
import networkx as nx
import numpy as np
from scipy.stats import pearsonr
# Check the installed version of NetworkX.
nx.__version__
```
Out[91]:
```
'3.0'
```

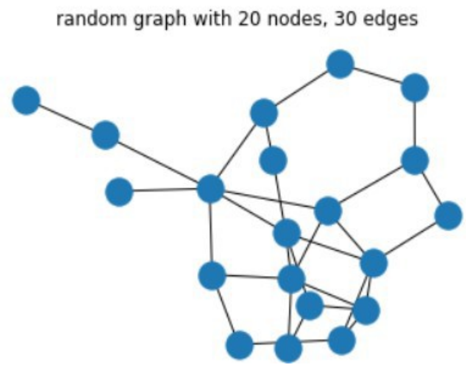
In [92]:
```
# ==================== Step 1 ==========================
# nodes
n_1 = 6
n_2 = 10
n_3 = 20

# plot 1: 6 nodes, 9 edges
Graph_1 = nx.gnm_random_graph(n_1, 1.5 * n_1)
title('random graph with 6 nodes, 9 edges')
nx.draw(Graph_1)
```



In [93]:
```
# plot 2: 10 ndoes, 15 edges
Graph_2 = nx.gnm_random_graph(n_2, 1.5 * n_2)
title('random graph with 10 nodes, 15 edges')
nx.draw(Graph_2)
```



In [94]:
```
# plot 3: 20 nodes, 30 edges
Graph_3 = nx.gnm_random_graph(n_3, 1.5 * n_3)
title('random graph with 20 nodes, 30 edges')
nx.draw(Graph_3)
```



In [95]:
```
# ==================== Step 2 ==========================
def func_degrees_descend(graph):
    degrees_descend = sorted([d for n, d in graph.degree()], reverse=True)
    # print(str(graph) + ": degrees sequence in a descending order is " + str(degrees_descend))
    return degrees_descend
```

In [96]:
```
# ==================== Step 3 ==========================
'''
Use a dynamic programming algorithm to achieve the anonymized degree sequences
'''

def degree_anonymization(d, n, k):
    # d: degree sequence
    # n: number of nodes
    # k: k-anonymity parameter

    # initialize a 2D dp array with minimum cost to make the first i nodes k-anonymous with total degree sum of j
    dp = np.zeros((n + 1, k * n + 1))

    # calculate the minimum cost for each node i
    for i in range(1, n + 1):
        for j in range(k, k * n + 1):
            # calculate the minimum cost of keeping the degree of node i as d[i-1]
            min_cost = dp[i - 1][j - d[i - 1]] + abs(d[i - 1] - j)

            # calculate the minimum cost of increasing the degree of node i to k
            increase_cost = dp[i - 1][j - (k - 1)] + abs(d[i - 1] - (k - 1))

            # store the minimum cost
            dp[i][j] = min(min_cost, increase_cost)

            # calculate the minimum cost of increasing the degree of node i to k + 1, k + 2, ... n - 1
            for x in range(k + 1, n):
                increase_cost = dp[i - 1][j - x] + abs(d[i - 1] - x)
                dp[i][j] = min(dp[i][j], increase_cost)
```

```
    # initialize the k-anonymous degree sequence seq with 0
    seq = [0 for i in range(n)]

    # find the k-anonymous degree sequence seq with minimal cost with the absolute values
    j = k * n
    for i in range(n - 1, -1, -1):
        if dp[i][j] == dp[i - 1][j - d[i - 1]] + abs(d[i - 1] - j):
            seq[i] = d[i - 1]
            j = j - d[i - 1]
        else:
            for x in range(k - 1, n - 1):
                if dp[i][j] == dp[i - 1][j - x] + abs(d[i - 1] - x):
                    seq[i] = x
                    j = j - x
                    break
    return seq
```

In [97]:
```
# ==================== Step 4 ==========================
def anonymized_deg_seq_graph1():
    print("====================== Try k = 2, 3 for Graph_1 with n = 6 =============================")
    # Degree sequence of Graph 1 where n = 6
    graph1_degs = func_degrees_descend(Graph_1)

    # Degree sequence of Graph 1 with 2-anonymity
    graph1_degs_2anonymity = degree_anonymization(graph1_degs, 6, 2)

    print("Graph_1 (6 nodes, 9 edges) : degree sequence is", graph1_degs)
    print("When n = 6, k = 2: ", end="")
    print("2-anonymous degree sequence:", graph1_degs_2anonymity)
    k2_n6_cor, k2_n6_pvalue = pearsonr(graph1_degs, graph1_degs_2anonymity)
    print("                           Pearson's correlation coefficient:", k2_n6_cor)
    print("                           p-value:", k2_n6_pvalue)

    print("When n = 6, k = 3: ", end="")
    # Degree sequence of Graph 1 with 3-anonymity
    graph1_degs_3anonymity = degree_anonymization(graph1_degs, 6, 3)
    print("3-anonymous degree sequence:", graph1_degs_3anonymity)
    k3_n6_cor, k3_n6_pvalue = pearsonr(graph1_degs, graph1_degs_3anonymity)
    print("                           Pearson's correlation coefficient:", k3_n6_cor)
    print("                           p-value:", k3_n6_pvalue)

    # Draw the anonymized graph based on n = 6, seq = 2 anonymity seq
    G2 = nx.configuration_model(graph1_degs_2anonymity)
    nx.draw(G2, with_labels=True)
    title('The graph with 6 nodes, 2-Anonmity')
    plt.show()

     # Draw the anonymized graph based on n = 6, seq = 3 anonymity seq
    G3 = nx.configuration_model(graph1_degs_3anonymity)
    nx.draw(G3, with_labels=True)
    title('The graph with 6 nodes, 3-Anonmity')
    plt.show()
```

In [98]:
```
def anonymized_deg_seq_graph2():
    print("====================== Try k = 3, 4 for the Graph_2 with n = 10 ============================")
    # Degree sequence of Graph 2 where n = 10
    graph2_degs = func_degrees_descend(Graph_2)

    # Degree sequence of Graph 2 with 3-anonymity
    graph2_degs_3anonymity = degree_anonymization(graph2_degs, 10, 3)

    print("Graph_2 (10 nodes, 15 edges): degree sequence is", graph2_degs)
    print("When n = 10, k = 3: ", end ="")
    print("3-anonymous degree sequence:", graph2_degs_3anonymity)

    print("When n = 10, k = 4: ", end="")
    # Degree sequence of Graph 1 with 3-anonymity
    graph2_degs_4anonymity = degree_anonymization(graph2_degs, 10, 4)
    print("3-anonymous degree sequence:", graph2_degs_4anonymity)
```

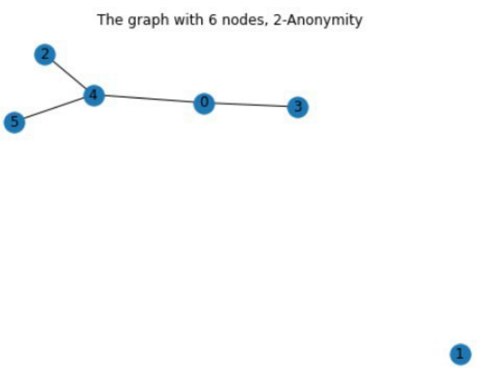
In [99]:
```
def anonymized_deg_seq_graph3():
    print("====================== Try k = 3, 4 for the Graph_3 with n = 20 ============================")
    # Degree sequence of Graph 3 where n = 20
    graph3_degs = func_degrees_descend(Graph_3)

    # Degree sequence of Graph 2 with 3-anonymity
    graph3_degs_3anonymity = degree_anonymization(graph3_degs, 20, 3)

    print("Graph_3 (20 nodes, 30 edges): degree sequence is", graph3_degs)
    print("When n = 20, k = 3: ", end ="")
    print("3-anonymous degree sequence:", graph3_degs_3anonymity)

    print("When n = 20, k = 4: ", end="")
    # Degree sequence of Graph 1 with 3-anonymity
    graph3_degs_4anonymity = degree_anonymization(graph3_degs, 20, 4)
    print("3-anonymous degree sequence:", graph3_degs_4anonymity)
```

```
In [100]: if __name__ == "__main__":
              anonymized_deg_seq_graph1()
              print()
              anonymized_deg_seq_graph2()
              print()
              anonymized_deg_seq_graph3()
```

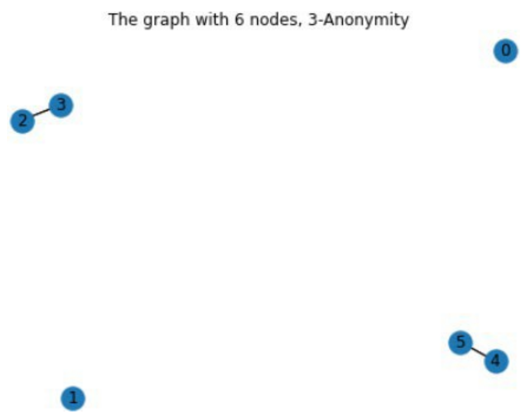
Here is the output:

```
======================= Try k = 2, 3 for Graph_1 with n = 6 =============================
Graph_1 (6 nodes, 9 edges) : degree sequence is [5, 3, 3, 3, 2, 2]
When n = 6, k = 2: 2-anonymous degree sequence: [2, 0, 3, 3, 3, 1]
                   Pearson's correlation coefficient: 0.0
                   p-value: 1.0
When n = 6, k = 3: 3-anonymous degree sequence: [0, 0, 2, 2, 2, 2]
                   Pearson's correlation coefficient: -0.7071067811865477
                   p-value: 0.11611652351681546

====================== Try k = 3, 4 for the Graph_2 with n = 10 ============================
Graph_2 (10 nodes, 15 edges): degree sequence is [5, 4, 4, 4, 4, 2, 2, 2, 2, 1]
When n = 10, k = 3: 3-anonymous degree sequence: [1, 5, 4, 4, 4, 4, 2, 2, 2, 2]
When n = 10, k = 4: 3-anonymous degree sequence: [0, 5, 3, 3, 3, 3, 3, 3, 3, 3]

====================== Try k = 3, 4 for the Graph_3 with n = 20 ============================
Graph_3 (20 nodes, 30 edges): degree sequence is [6, 5, 5, 5, 4, 4, 3, 3, 3, 3, 3, 2, 2, 2, 2, 2, 2, 2, 1, 1]
When n = 20, k = 3: 3-anonymous degree sequence: [0, 6, 5, 5, 5, 4, 4, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]
When n = 20, k = 4: 3-anonymous degree sequence: [0, 6, 5, 5, 5, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3]
```

Here are the new graphs based on 2-Anonymity, 3-Anonymity when n = 6



Graph of n=6, k=2



Graph of n=6, k=3

Comparing the shape and sequences of new graphs with the original graph's, we find the shapes and new degree sequences of new graphs are very different from the original graph. It means the new degree sequences are not reliable. I also calculated some stat metrics as a reference for me to check my conclusion.

Problem 3

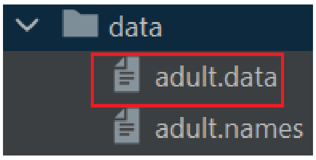
Step 1. Data Source & Input Data & Output Data

Download source data from UCI Repository:
https://archive.ics.uci.edu/ml/machine-learning-databases/adult/

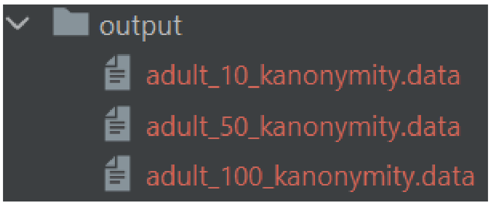
Index of /ml/machine-learning-databases/adult

- Parent Directory
- Index
- adult.data
- adult.names
- adult.test
- old.adult.names

Here are the input data in my code project:

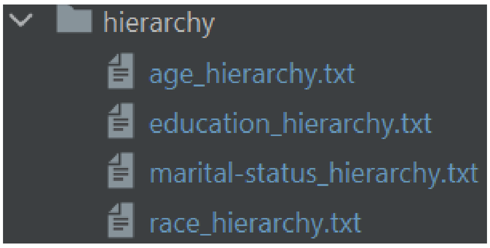


Here are the output data in my code project:



Step 2. Define a generalization hierarchy for each of the 4 attributes

I randomly define the hierarchy in four text files based on Quasi-identifiers (QIs) : age, education, marital-status, race



Step 3 Write a program for the heuristic algorithm (which generalizes/suppresses the data while minimizing the utility loss).

Source code in the script of problem3_solution.py
The core code is shown below:

```python
def anonymize_func(self, quasi=['age', 'education', 'marital-status', 'race'], k=5):
    """
    k-anonymity: Given k = 5
    Arguments: {['age', 'education', 'marital-status', 'race']}
    """

    identifiers, generalized_levels = dict(), dict()
    qi_frequency = dict()

    if not len(self.defined_hierarchy) == len(quasi):
        raise ValueError('number of files is not equal to number of quasi!!!!')

    hierarchy = dict()
    for idx, name in enumerate(quasi):
        hierarchy[name] = DGHTree(self.defined_hierarchy[idx])
```

```python
for idx, record in enumerate(self.datasets):
    qi_sequence = self._get_qi_values(record[:], quasi, hierarchy)

    if qi_sequence in qi_frequency:
        qi_frequency[qi_sequence].add(idx)
    else:
        qi_frequency[qi_sequence] = {idx}
        for j, value in enumerate(qi_sequence):
            identifiers[quasi[j]].add(value)
```

```python
while True:
    # number of records which is not satisfied k-anonymity
    invalid_count = 0
    for qi_sequence, idxset in qi_frequency.items():
        if len(idxset) < k:
            invalid_count += len(idxset)

    if invalid_count > k:
        # keep generalizing
        most_freq_att_num, most_freq_att_name = -1, None
        for identifier in quasi:
            if len(identifiers[identifier]) > most_freq_att_num:
                most_freq_att_num = len(identifiers[identifier])
                most_freq_att_name = identifier

        generalize_att = most_freq_att_name
        qi_index = quasi.index(generalize_att)
        identifiers[generalize_att] = set()
```

```
for qi_sequence in list(qi_frequency.keys()):
    new_qi_sequence = list(qi_sequence)
    new_qi_sequence[qi_index] = hierarchy[generalize_att].root[qi_sequence[qi_index]][0]
    new_qi_sequence = tuple(new_qi_sequence)

    if new_qi_sequence in qi_frequency:
        qi_frequency[new_qi_sequence].update(
            qi_frequency[qi_sequence])
        qi_frequency.pop(qi_sequence, 0)
    else:
        qi_frequency[new_qi_sequence] = qi_frequency.pop(qi_sequence)

    identifiers[generalize_att].add(new_qi_sequence[qi_index])

generalized_levels[generalize_att] += 1
```

Step 4 The distortion and precision

Source code in the script of problem3_solution.py

```
def calc_precision(self, generalized_levels, max_depth, data_size, attribute_num=4):
    precision = 1 - sum([generalized_levels[i] / max_depth[i] for i in range(attribute_num)]) / (
            data_size * attribute_num)
    return precision
```

```
def calc_distoration(self, generalized_levels, max_depth, attribute_num):
    print('Generalized level of attribute:', generalized_levels)
    print('Depth of the hierarchy:', max_depth)
    distoration = sum([generalized_levels[i] / max_depth[i] for i in range(attribute_num)]) / attribute_num
    return distoration
```

The output of calculation:

Test 01: Given k = 100

```
quasi identifiers:  ['age', 'education', 'marital-status', 'race']
Generalized level of attribute: [2, 3, 2, 1]
Depth of the hierarchy: [3, 4, 3, 2]
precision: 0.35575343201990106, distoration: 0.6458333333333333
```

Test 02: Given k = 10

```
quasi identifiers:  ['age', 'education', 'marital-status', 'race']
Generalized level of attribute: [2, 2, 1, 1]
Depth of the hierarchy: [3, 4, 3, 2]
precision: 0.5001228463499279, distoration: 0.49999999999999994
```

Test 02: Given k = 50

```
quasi identifiers:  ['age', 'education', 'marital-status', 'race']
Generalized level of attribute: [2, 3, 2, 1]
Depth of the hierarchy: [3, 4, 3, 2]
precision: 0.3544245160365672, distoration: 0.6458333333333333
```

Reference

L. Sweeney. Achieving k-anonymity privacy protection using generalization and suppression. International Journal on Uncertainty, Fuzziness and Knowledge-based Systems, 10 (5), 2002; 571-588.