from ucimlrepo import fetch\_ucirepo   
import pandas as pd   
import numpy as np

adult\_dataset = fetch\_ucirepo(id=2)   
adult\_df = adult\_dataset.data.features  
adult\_df['income'] = adult\_dataset.data.targets  
# print(adult\_dataset.metadata)   
# print(adult\_dataset.variables)   
adult\_df.head()

age workclass fnlwgt education education-num \  
0 39 State-gov 77516 Bachelors 13   
1 50 Self-emp-not-inc 83311 Bachelors 13   
2 38 Private 215646 HS-grad 9   
3 53 Private 234721 11th 7   
4 28 Private 338409 Bachelors 13   
  
 marital-status occupation relationship race sex \  
0 Never-married Adm-clerical Not-in-family White Male   
1 Married-civ-spouse Exec-managerial Husband White Male   
2 Divorced Handlers-cleaners Not-in-family White Male   
3 Married-civ-spouse Handlers-cleaners Husband Black Male   
4 Married-civ-spouse Prof-specialty Wife Black Female   
  
 capital-gain capital-loss hours-per-week native-country income   
0 2174 0 40 United-States <=50K   
1 0 0 13 United-States <=50K   
2 0 0 40 United-States <=50K   
3 0 0 40 United-States <=50K   
4 0 0 40 Cuba <=50K

## Dataset Analysis

categorical\_columns = adult\_df.select\_dtypes(include=['object']).columns.tolist()  
numerical\_columns = adult\_df.select\_dtypes(include=['int64']).columns.tolist()  
  
temp\_df = adult\_df.copy()  
temp\_df[categorical\_columns] = temp\_df[categorical\_columns].astype('string')  
  
data\_type\_df = temp\_df.dtypes  
  
print("List of Columns and Their Type:")  
data\_type\_df

List of Columns and Their Type:

age int64  
workclass string  
fnlwgt int64  
education string  
education-num int64  
marital-status string  
occupation string  
relationship string  
race string  
sex string  
capital-gain int64  
capital-loss int64  
hours-per-week int64  
native-country string  
income string  
dtype: object

unique\_values\_count = adult\_df[categorical\_columns].nunique()  
print('Number of unique values in categorical columns:')  
unique\_values\_count

Number of unique values in categorical columns:

workclass 9  
education 16  
marital-status 7  
occupation 15  
relationship 6  
race 5  
sex 2  
native-country 42  
income 4  
dtype: int64

def count\_outliers(series):  
 Q1 = series.quantile(0.25)  
 Q3 = series.quantile(0.75)  
 IQR = Q3 - Q1  
 lower\_bound = Q1 - 1.5 \* IQR  
 upper\_bound = Q3 + 1.5 \* IQR  
 return ((series < lower\_bound) | (series > upper\_bound)).sum()  
  
  
stats = {}  
  
for column in numerical\_columns:  
 stats[column] = {  
 'min': adult\_df[column].min(),  
 'max': adult\_df[column].max(),  
 'mean': adult\_df[column].mean(),  
 'std': adult\_df[column].std(),  
 'number\_of\_unique\_values': adult\_df[column].nunique(),  
 'number\_of\_outliers': count\_outliers(adult\_df[column])  
 }  
  
stats\_df = pd.DataFrame(stats).T  
  
stats\_df

min max mean std \  
age 17.0 90.0 38.643585 13.710510   
fnlwgt 12285.0 1490400.0 189664.134597 105604.025423   
education-num 1.0 16.0 10.078089 2.570973   
capital-gain 0.0 99999.0 1079.067626 7452.019058   
capital-loss 0.0 4356.0 87.502314 403.004552   
hours-per-week 1.0 99.0 40.422382 12.391444   
  
 number\_of\_unique\_values number\_of\_outliers   
age 74.0 216.0   
fnlwgt 28523.0 1453.0   
education-num 16.0 1794.0   
capital-gain 123.0 4035.0   
capital-loss 99.0 2282.0   
hours-per-week 96.0 13496.0

### Preserved Utilities

When anonymizing the Adult Income dataset, which contains information like age, job, education, and income, the goal is to keep individual identities private while ensuring the data remains useful for tasks like predicting income or studying how different factors affect earnings. This means keeping important details like age range, hours per week, and education levels accurate enough for analysis but altered slightly or grouped together to protect privacy. It’s a balancing act between making sure no one can be identified from the data and maintaining its value for understanding economic patterns, ensuring fairness, and educational purposes.

# Identification of Sensitive Information

print('Numerical comuns:')  
print(numerical\_columns)  
print('Categorical Columns:')  
print(categorical\_columns)

Numerical comuns:  
['age', 'fnlwgt', 'education-num', 'capital-gain', 'capital-loss', 'hours-per-week']  
Categorical Columns:  
['workclass', 'education', 'marital-status', 'occupation', 'relationship', 'race', 'sex', 'native-country', 'income']

| Category | Attributes | Reason for Classification |
| --- | --- | --- |
| **Explicit Identifiers** | None provided directly in the dataset. | Explicit identifiers (like names, SSN) that can directly identify an individual are not included in the dataset for privacy reasons. |
| **Quasi-Identifiers** | Age, Workclass, Education, Marital Status, Occupation, Relationship, Race, Sex, Native-country, Hours-per-week | These attributes, in combination, can potentially be used to re-identify individuals indirectly, especially when linked with external information. |
| **Sensitive Attributes** | Income | Income is considered sensitive as it can reveal personal economic status and potentially be used for discriminatory purposes if linked to an individual. |
| **Non-sensitive Attributes** | Education-num, Capital-gain, Capital-loss | These attributes, while informative about an individual's socioeconomic status, are less likely to identify an individual directly or be used discriminatively on their own. |

# K-Anonymization Technique (Mondiran)

Reference:

LeFevre, Kristen, David J. DeWitt, and Raghu Ramakrishnan. "Mondrian multidimensional k-anonymity." ​22nd International conference on data engineering (ICDE'06).​ IEEE,2006.

<https://github.com/Nuclearstar/K-Anonymity/>

Mondrian k-anonymization is a partitioning algorithm used to achieve k-anonymity in datasets, ensuring that each record is indistinguishable from at least (k-1) other records regarding certain identifying attributes. This process helps protect privacy by making re-identification more difficult. The basic steps for Mondrian k-anonymization are as follows:

### Preparation:

* Identify the quasi-identifiers in the dataset. These are attributes that, in combination, could be linked with external information to re-identify individuals.
* Determine the value of (k) for (k)-anonymity, reflecting the privacy level desired.

### Recursiveness:

* Mondrian is a recursive algorithm. Start by considering the entire dataset as a single group.

### Partitioning:

* Choose one quasi-identifier to partition the data. This choice is typically based on the attribute that has the highest range of values (i.e., the attribute that can be divided most evenly).
* Split the dataset into two partitions based on the median value of the chosen attribute, aiming to distribute records as evenly as possible while ensuring that each partition has at least (k) records.

### Recursion:

* Recursively apply the partitioning step to each new partition created until further partitioning would violate the (k)-anonymity constraint (i.e., a partition cannot be split without resulting in a subgroup with fewer than (k) records).

### Generalization:

* For each partition, generalize the quasi-identifier values to the smallest range that includes all values in the partition. This step ensures that all records in a partition are indistinguishable based on the quasi-identifiers.

### Stop Condition:

* The recursion stops when it's no longer possible to split a partition into two groups that both satisfy the (k)-anonymity requirement.

### Post-processing:

* After the partitioning and generalization steps are complete, review the anonymized data to ensure it meets the desired (k)-anonymity level. Adjustments can be made if necessary.

Mondrian k-anonymization is particularly useful for numerical and categorical data that can be ordered. The algorithm's simplicity and effectiveness make it a popular choice for anonymizing datasets while preserving their utility for analysis. However, it's important to note that k-anonymity by itself does not protect against all types of re-identification attacks, especially when attackers have access to additional background information.

# Anonymizer Script

Analysis of the Adult dataset is available in the notebooks/k\_anonymization.ipynb file.

The anonymizer.py script applies the Mondrian k-anonymization algorithm to achieve k-anonymity in datasets, such as the Adult dataset, ensuring privacy protection and preventing individual re-identification. This script is designed to handle both numerical and categorical data, making it versatile for various anonymization needs.

## Getting Started

### Prerequisites

Ensure you have Python 3 installed on your system. You can check your Python version by running:

python3 --version

### Usage

The anonymizer.py script takes several arguments to specify the level of anonymity (k-value), the quasi-identifiers, the sensitive attribute, and the output file path. You can learn about all available arguments and the script's functionality by using the --help option:

python3 anonymizer.py --help

### Arguments

* -k, --k\_value: The k-value for k-anonymity, determining the level of privacy.
* -q, --quasi\_identifiers: A list of column names to be treated as quasi-identifiers, separated by commas.
* -s, --sensitive\_attribute: The name of the sensitive attribute in the dataset.
* -o, --output: Path to save the anonymized dataset.

### Running Example

To anonymize a dataset with a k-value of 3, treating 'age' and 'hours-per-week' as quasi-identifiers, 'income' as a sensitive attribute, and specifying input and output files, you would run:

python3 anonymizer.py --k 3 --quasi\_identifiers age,hours-per-week --sensitive\_attribute income --output path/to/anonymized\_dataset.csv

Replace path/to/anonymized\_dataset.csv with the desired path for the anonymized output.

anonymized\_df = pd.read\_csv('../anonymized\_data.csv')

anonymized\_df

Unnamed: 0 age education-num hours-per-week income Count  
0 0 17 7-8 20-36 <=50K 93  
1 1 17 7-8 20-36 <=50K. 52  
2 2 31-32 9 40-99 <=50K 407  
3 3 31-32 9 40-99 <=50K. 218  
4 4 31-32 9 40-99 >50K 75  
... ... ... ... ... ... ...  
5328 5328 90 14-15 50-60 <=50K. 1  
5329 5329 90 14-15 50-60 >50K 2  
5330 5330 84-90 14-15 66-99 <=50K 1  
5331 5331 84-90 14-15 66-99 <=50K. 1  
5332 5332 84-90 14-15 66-99 >50K 1  
  
[5333 rows x 6 columns]

# Anonymized Dataset Overview

The anonymized dataset has been processed to include ranges for age, education-num, and hours-per-week to ensure privacy through k-anonymity. This approach helps protect individual identities by making each record indistinguishable from at least (k-1) other records in terms of the specified quasi-identifiers. Here is how the anonymized data is structured:

| Unnamed: 0 | age | education-num | hours-per-week | income | Count |
| --- | --- | --- | --- | --- | --- |
| 0 | 17 | 7-8 | 20-36 | <=50K | 93 |
| 1 | 17 | 7-8 | 20-36 | <=50K. | 52 |
| 2 | 31-32 | 9 | 40-99 | <=50K | 407 |
| 3 | 31-32 | 9 | 40-99 | <=50K. | 218 |
| 4 | 31-32 | 9 | 40-99 | >50K | 75 |
| ... | ... | ... | ... | ... | ... |
| 5328 | 90 | 14-15 | 50-60 | <=50K. | 1 |
| 5329 | 90 | 14-15 | 50-60 | >50K | 2 |
| 5330 | 84-90 | 14-15 | 66-99 | <=50K | 1 |
| 5331 | 84-90 | 14-15 | 66-99 | <=50K. | 1 |
| 5332 | 84-90 | 14-15 | 66-99 | >50K | 1 |

## Utility Differences: Before vs. After Anonymization

* **Precision**: Anonymization reduces data precision by grouping exact values into ranges, potentially impacting detailed analyses and insights.
* **Statistical Measures**: Measures like mean or median may shift or become less precise due to the generalization of values, affecting statistical validity.

### Before anonymization

stats\_df

min max mean std \  
age 17.0 90.0 38.643585 13.710510   
fnlwgt 12285.0 1490400.0 189664.134597 105604.025423   
education-num 1.0 16.0 10.078089 2.570973   
capital-gain 0.0 99999.0 1079.067626 7452.019058   
capital-loss 0.0 4356.0 87.502314 403.004552   
hours-per-week 1.0 99.0 40.422382 12.391444   
  
 number\_of\_unique\_values number\_of\_outliers   
age 74.0 216.0   
fnlwgt 28523.0 1453.0   
education-num 16.0 1794.0   
capital-gain 123.0 4035.0   
capital-loss 99.0 2282.0   
hours-per-week 96.0 13496.0

mean\_hours = adult\_df['hours-per-week'].mean()  
print(f"Mean Hours-Per-Week for Income: {mean\_hours}")  
mean\_hours = adult\_df[adult\_df['income'] == '>50K']['hours-per-week'].mean()  
print(f"Mean Hours-Per-Week for Income >50K: {mean\_hours}")  
mean\_hours = adult\_df[adult\_df['income'] == '<=50K']['hours-per-week'].mean()  
print(f"Mean Hours-Per-Week for Income <=50K: {mean\_hours}")

Mean Hours-Per-Week for Income: 40.422382375824085  
Mean Hours-Per-Week for Income >50K: 45.473026399693914  
Mean Hours-Per-Week for Income <=50K: 38.840210355987054

mean\_hours = adult\_df['age'].mean()  
print(f"Mean Age for Income: {mean\_hours}")  
mean\_hours = adult\_df[adult\_df['income'] == '>50K']['age'].mean()  
print(f"Mean Age for Income >50K: {mean\_hours}")  
mean\_hours = adult\_df[adult\_df['income'] == '<=50K']['age'].mean()  
print(f"Mean Age for Income <=50K: {mean\_hours}")

Mean Age for Income: 38.64358543876172  
Mean Age for Income >50K: 44.24984058155847  
Mean Age for Income <=50K: 36.78373786407767

### After Anonymization

anonymized\_df['hours\_mid'] = anonymized\_df['hours-per-week'].apply(lambda x: sum(int(y) for y in x.split('-')) / 2)  
  
  
mean\_hours = anonymized\_df['hours\_mid'].mean()  
print(f"Mean Hours-Per-Week for Income: {mean\_hours}")  
mean\_hours = anonymized\_df[anonymized\_df['income'] == '>50K']['hours\_mid'].mean()  
print(f"Mean Hours-Per-Week for Income >50K: {mean\_hours}")  
mean\_hours = anonymized\_df[anonymized\_df['income'] == '<=50K']['hours\_mid'].mean()  
print(f"Mean Hours-Per-Week for Income <=50K: {mean\_hours}")

Mean Hours-Per-Week for Income: 32.855803487717985  
Mean Hours-Per-Week for Income >50K: 37.44700214132762  
Mean Hours-Per-Week for Income <=50K: 29.894182547642927

anonymized\_df['age\_mid'] = anonymized\_df['age'].apply(lambda x: sum(int(y) for y in x.split('-')) / 2)  
  
  
mean\_hours = anonymized\_df['age\_mid'].mean()  
print(f"Mean Age for Income: {mean\_hours}")  
mean\_hours = anonymized\_df[anonymized\_df['income'] == '>50K']['age\_mid'].mean()  
print(f"Mean Age for Income >50K: {mean\_hours}")  
mean\_hours = anonymized\_df[anonymized\_df['income'] == '<=50K']['age\_mid'].mean()  
print(f"Mean Age for Income <=50K: {mean\_hours}")

Mean Age for Income: 34.88186761672605  
Mean Age for Income >50K: 35.51284796573876  
Mean Age for Income <=50K: 34.99699097291876