k-Anonymity

k-Anonymity is the first formal privacy definition we have seen. The definition of k-Anonymity is designed to formalize our intuition that a piece of auxiliary information should not narrow down the set of possible records for an individual "too much." Stated another way, k-Anonymity is designed to ensure that each individual can "blend into the crowd."

Informally, we say that that a dataset is "k-Anonyized" for a particular k if each individual in the dataset is a member of a group of size at least k, such that each member of the group shares the same *quasi-identifiers* (a selected subset of all the dataset's columns) with all other members of the group. Thus, the individuals in each group "blend into" their group - it's possible to narrow down an individual to membership in a particular group, but not to determine which group member is the target.

More formally, we say that a dataset D satisfies k-Anonymity for a value of k if:

- For each row $r_1 \in D$, there exist at least $k-1$ other rows $r_2 \dots r_k \in D$ such that $\Pi_{qi(D)} r_1 = \Pi_{qi(D)} r_2, \dots, \Pi_{qi(D)} r_1 = \Pi_{qi(D)} r_k$

where $qi(D)$ is the quasi-identifiers of D, and $\Pi_{qi(D)} r$ represents the columns of r containing quasi-identifiers (i.e. the projection of the quasi-identifiers).

Checking for k-Anonymity

We'll start with a small dataset, so that we can immediately see by looking at the data whether it satisfies k-Anonymity or not. This dataset contains age plus two test scores; it clearly doesn't satisfy k-Anonymity for $k > 1$. Any dataset trivially satisfies k-Anonymity for $k = 1$, since each row can form its own group of size 1.

In [82]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')
```

```
In [50]: raw_data = {
             'first_name': ['Jason', 'Molly', 'Tina', 'Jake', 'Amy'],
             'last_name': ['Miller', 'Jacobson', 'Ali', 'Milner', 'Cooze'],
             'age': [42, 52, 36, 24, 73],
             'preTestScore': [4, 24, 31, 2, 3],
             'postTestScore': [25, 94, 57, 62, 70]}
         #df = pd.DataFrame(raw_data, columns = ['first_name', 'last_name', 'age', 'p
         df = pd.DataFrame(raw_data, columns = ['age', 'preTestScore', 'postTestScore
         df
```

Out[50]:

	age	preTestScore	postTestScore
0	42	4	25
1	52	24	94
2	36	31	57
3	24	2	62
4	73	3	70

```
In [51]: df
```

Out[51]:

	age	preTestScore	postTestScore
0	42	4	25
1	52	24	94
2	36	31	57
3	24	2	62
4	73	3	70

To implement a function to check whether a dataframe satisfies k-Anonymity, we loop over the rows; for each row, we query the dataframe to see how many rows match its values for the quasi-identifiers. If the number of rows in any group is less than k, the dataframe does not satisfy k-Anonymity for that value of k, and we return False. Note that in this simple definition, we consider *all* columns to contain quasi-identifiers; to limit our check to a subset of all columns, we would need to replace the `df.columns` expression with something else.

```
In [52]: def isKAnonymized(df, k):
             for index, row in df.iterrows():
                 query = ' & '.join([f'{col} == {row[col]}' for col in df.columns])
                 rows = df.query(query)
                 if (rows.shape[0] < k):
                     return False

             return True
```

As expected, our example dataframe does *not* satisfy k-Anonymity for $k = 2$, but it does satisfy the property for $k = 1$.

In [53]:
```
isKAnonymized(df, 1)
```
Out[53]:
```
True
```
In [54]:
```
isKAnonymized(df, 2)
```
Out[54]:
```
False
```

Generalizing Data to Satisfy k-Anonymity

The process of modifying a dataset so that it satisfies k-Anonymity for a desired k is generally accomplished by *generalizing* the data - modifying values to be less specific, and therefore more likely to match the values of other individuals in the dataset. For example, an age which is accurate to a year may be generalized by rounding to the nearest 10 years, or a ZIP code might have its rightmost digits replaced by zeros. For numeric values, this is easy to implement. We'll use the `apply` method of dataframes, and pass in a dictionary named `depths` which specifies how many digits to replace by zeros for each column. This gives us the flexibility to experiment with different levels of generalization for different columns.

In [55]:
```
def generalize(df, depths):
    return df.apply(lambda x: x.apply(lambda y: int(int(y/(10**depths[x.name
```

Now, we can generalize our example dataframe. First, we'll try generalizing each column by one "level" - i.e. rounding to the nearest 10.

In [56]:
```
depths = {
    'age': 1,
    'preTestScore': 1,
    'postTestScore': 1
}
df2 = generalize(df, depths)
df2
```
Out[56]:

	age	preTestScore	postTestScore
0	40	0	20
1	50	20	90
2	30	30	50
3	20	0	60
4	70	0	70

Notice that even after generalization, our example data *still* does not satisfy k-Anonymity for $k = 2$.

In [57]:
```python
isKAnonymized(df2, 2)
```

Out[57]:
```
False
```

We can try generalizing more - but then we'll end up removing *all* of the data!

In [58]:
```python
depths = {
    'age': 2,
    'preTestScore': 2,
    'postTestScore': 2
}
generalize(df, depths)
```

Out[58]:

	age	preTestScore	postTestScore
0	0	0	0
1	0	0	0
2	0	0	0
3	0	0	0
4	0	0	0

This example illustrates one of the key challenges of achieving k-Anonymity: often, achieving the property for mneaningful values of k requires removing quite a lot of information from the data.

Does More Data Improve Generalization?

Our example dataset is too small for k-Anonymity to work well. Because there are only 5 individuals in the dataset, building groups of 2 or more individuals who share the same properties is difficult. The solution to this problem is more data: in a dataset with more individuals, less generalization will typically be needed to satisfy k-Anonymity for a desired k.

Let's try the same census data we examined for de-identification. This dataset contains more than 32,000 rows, so it should be easier to achieve k-Anonymity.

In [59]:
```python
adult_data = pd.read_csv("adult_with_pii.csv")
adult_data.head()
```

Out[59]:

	Name	DOB	SSN	Zip	Age	Workclass	fnlwgt	Education	Education-Num	Martial Status	Oc
0	Karrie Trusslove	9/7/1967	732-14-6110	64152	39	State-gov	77516	Bachelors	13	Never-married	
1	Brandise Tripony	6/7/1988	150-19-2766	61523	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	m
2	Brenn McNeely	8/6/1991	725-59-9860	95668	38	Private	215646	HS-grad	9	Divorced	[illegible]
3	Dorry Poter	4/6/2009	659-57-4974	25503	53	Private	234721	11th	7	Married-civ-spouse	[illegible]
4	Dick Honnan	9/16/1951	220-93-3811	75387	28	Private	338409	Bachelors	13	Married-civ-spouse	

We'll consider the ZIP code, age, and educational achievement of each individual to be the quasi-identifiers. We'll project just those columns, and try to achieve k-Anonymity for $k = 2$. Of course, the data is k-Anonymous for $k = 1$; notice that we take just the first 100 rows from the dataset for this check - try running `isKAnonymized` on a larger subset of the data, and you'll find that it takes a very long time. For $k = 2$, our algorithm finds a failing row quickly and finishes fast.

In [69]:
```python
df = adult_data[['Age', 'Education-Num']]
df.columns = ['age', 'edu']
```

In [70]:
```python
isKAnonymized(df.head(100), 1)
```

Out[70]: True

In [71]:
```python
isKAnonymized(df.head(100), 2)
```

Out[71]: False

For example, running the $k = 1$ check on 5000 rows takes about 20 seconds on my laptop.

In [72]:
```python
isKAnonymized(df.head(5000), 1)
```

Out[72]: True

Now, we'll try to generalize to achieve k-Anonymity for $k = 2$. We'll start with generalizing both age and educational attainment to the nearest 10.

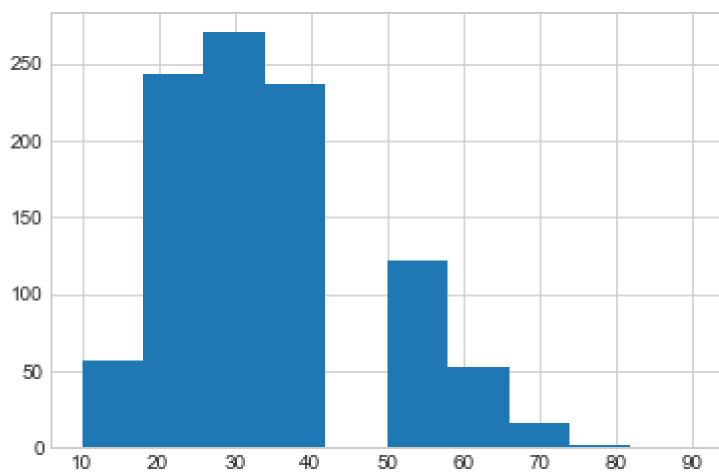
In [75]:
```
# outliers are a real problem!
depths = {
    'age': 1,
    'edu': 1
}
df2 = generalize(df.head(1000), depths)
isKAnonymized(df2, 2)
```

Out[75]: False

The generalized result still does not satisfy k-Anonymity for $k = 2$! In fact, we can perform this generalization on all 32,000 rows and still fail to satisfy k-Anonymity for $k = 2$ - so adding more data does not necessarily help as much as we expected.

The reason is that the dataset contains *outliers* - individuals who are very different from the rest of the population. These individuals do not fit easily into any group, even after generalization. Even considering *only* ages, we can see that adding more data is not likely to help, since very low and very high ages are poorly represented in the dataset.

In [77]:
```
df2['age'].hist();
```



Achieving the optimal generalization for k-Anonymity is very challenging in cases like this. Generalizing each row *more* would be overkill for the well-represented individuals with ages in the 20-40 range, and would hurt utility. However, more generalization is clearly needed for individuals at the upper and lower ends of the age range. This is the kind of challenge that occurs regularly in practice, and is difficult to solve automatically. In fact, optimal generalization for k-Anonymity has been shown to be NP-hard.

Removing Outliers

One solution to this problem is simply to clip the age of each individual in the dataset to lie within a specific range, eliminating outliers entirely. This can also hurt utility, since it replaces real ages with fake ones, but it can be better than generalizing each row more. We can use Numpy's `clip` method to perform this clipping. We clip ages to be 60 or below, and leave educational levels alone (by clipping them to a very large value).

In [81]:

```
# clipping away outliers
depths = {
    'age': 1,
    'edu': 1
}
dfp = df.clip(upper=np.array([60, 10000000000000]), axis='columns')
df2 = generalize(dfp.head(500), depths)
isKAnonymized(df2, 7)
```

Out[81]:

```
True
```

Now, the generalized dataset satisfies k-Anonymity for $k = 7$! In other words, our level of generalization was appropriate, but outliers prevented us from achieving k-Anonymity before, even for $k = 2$.

Summary

- k-Anonymity is a property of data, which ensures that each individual "blends in" with a group of at least k individuals.
- k-Anonymity is computationally expensive even to check: the naive algorithm is $O(n^2)$, and faster algorithms take considerable space.
- k-Anonymity can be achieved by modifying a dataset by *generalizing* it, so that particular values become more common and groups are easier to form.
- Optimal generalization is extremely difficult, and outliers can make it even more challenging. Solving this problem automatically is NP-hard.