In [1]:
```
%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')
import pandas as pd
import numpy as np

adult = pd.read_csv("adult_with_pii.csv")
def laplace_mech(v, sensitivity, epsilon):
    return v + np.random.laplace(loc=0, scale=sensitivity / epsilon)
def pct_error(orig, priv):
    return np.abs(orig - priv)/orig * 100.0
```

The Exponential Mechanism

The fundamental mechanisms we have seen so far (Laplace and Gaussian) are focused on numerical answers, and add noise directly to the answer itself. What if we want to return a precise answer (i.e. no added noise), but still preserve differential privacy? One solution is the exponential mechanism, which allows selecting the "best" element from a set while preserving differential privacy. The analyst defines which element is the "best" by specifying a *scoring function* that outputs a score for each element in the set, and also defines the set of things to pick from. The mechanism provides differential privacy by *approximately* maximizing the score of the element it returns - in other words, to satisfy differential privacy, the exponential mechanism sometimes returns an element from the set which does *not* have the highest score.

The exponential mechanism satisfies ϵ-differential privacy:

1. The analyst selects a set $\mathcal{R}$ of possible outputs
2. The analyst specifies a scoring function $u : \mathcal{D} \times \mathcal{R} \rightarrow \mathbb{R}$ with global sensitivity Δu
3. The exponential mechanism outputs $r \in \mathcal{R}$ with probability proportional to:

$$\exp\left(\frac{\epsilon u(x, r)}{2\Delta u}\right)$$

The biggest practical difference between the exponential mechanism and the previous mechanisms we've seen (e.g. the Laplace mechanism) is that the output of the exponential mechanism is *always* a member of the set $\mathcal{R}$. This is extremely useful when selecting an item from a finite set, when a noisy answer would not make sense. For example, we might want to pick a date for a big meeting, which uses each participant's personal calendar to maximize the number of participants without a conflict, while providing differential privacy for the calendars. Adding noise to a date doesn't make much sense: it might turn a Friday into a Saturday, and *increase* the number of conflicts significantly. The exponential mechanism is perfect for problems like this one: it selects a date *without noise*.

The exponential mechanism is interesting for several reasons:

- The privacy cost of the mechanism is just ϵ, regardless of the size of $\mathcal{R}$ - more on this next.
- It works for both finite and infinite sets $\mathcal{R}$, but it can be really challenging to build a practical implementation which samples from the appropriate probability distribution when $\mathcal{R}$ is infinite.
- It represents a "fundamental mechanism" of ϵ-differential privacy: all other ϵ-differentially private mechanisms can be defined in terms of the exponential mechanism with the appropriate

definition of the scoring function u.

The Exponential Mechanism for Finite Sets

In [28]:
```
options = adult['Martial Status'].unique()

def score(data, option):
    return data.value_counts()[option]/1000

score(adult['Martial Status'], 'Never-married')
```

Out[28]:
```
10.683
```

In [29]:
```
def exponential(x, R, u, sensitivity, epsilon):
    # Calculate the score for each element of R
    scores = [u(x, r) for r in R]

    # Calculate the probability for each element, based on its score
    probabilities = [np.exp(epsilon * score / (2 * sensitivity)) for score i

    # Normalize the probabilties so they sum to 1
    probabilities = probabilities / np.linalg.norm(probabilities, ord=1)

    # Choose an element from R based on the probabilities
    return np.random.choice(R, 1, p=probabilities)[0]

exponential(adult['Martial Status'], options, score, 1, 1)
```

Out[29]:
```
'Married-civ-spouse'
```

In [30]:
```
r = [exponential(adult['Martial Status'], options, score, 1, 1) for i in ran
pd.Series(r).value_counts()
```

Out[30]:
```
Married-civ-spouse    174
Never-married          25
Divorced                1
dtype: int64
```

Report Noisy Max

Can we recover the exponential mechanism using the Laplace mechanism? In the case of a finite set $\mathcal{R}$, the basic idea of the exponential mechanism - to select from a set with differential privacy - suggests a naive implementation in terms of the Laplace mechanism:

1. For each $r \in \mathcal{R}$, calculate a *noisy score* $u(x, r) + \text{Lap}(\frac{\Delta u}{\epsilon})$
2. Output the element $r \in \mathcal{R}$ with the maximum noisy score

Since the scoring function u is Δu sensitive in x, each "query" in step 1 satisfies ϵ-differential privacy. Thus if $\mathcal{R}$ contains n elements, the above algorithm satisfies $n\epsilon$-differential privacy by sequential composition.

However, if we used the exponential mechanism, the total cost would be just ϵ instead! Why is the exponential mechanism so much better? Because *it releases less information*.

Our analysis of the Laplace-based approach defined above is very pessimistic. The whole set of noisy scores computed in step 1 actually satisfies $n\epsilon$-differential privacy, and we could release the whole thing. That the output in step 2 satisfies $n\epsilon$-differential privacy follows from the post-processing property.

But the exponential mechanism releases *only* the identity of the element with the maximum noisy score - *not* the score itself, or the scores of any other element.

The algorithm defined above is often called the *report noisy max* algorithm, and it actually satisfies ϵ-differential privacy, no matter how large the set $\mathcal{R}$ is - specifically because it releases *only* the identity of the element with the largest noisy count. The proof can be found in Dwork and Roth (https://www.cis.upenn.edu/~aaroth/Papers/privacybook.pdf), Claim 3.9.

Report noisy max is easy to implement, and it's easy to see that it produces very similar results to our earlier implementation of the exponential mechanism for finite sets.

In [31]:
```
def report_noisy_max(x, R, u, sensitivity, epsilon):
    # Calculate the score for each element of R
    scores = [u(x, r) for r in R]

    # Add noise to each score
    noisy_scores = [laplace_mech(score, sensitivity, epsilon) for score in s

    # Find the index of the maximum score
    max_idx = np.argmax(noisy_scores)

    # Return the element corresponding to that index
    return R[max_idx]

report_noisy_max(adult['Martial Status'], options, score, 1, 1)
```

Out[31]:
```
'Married-civ-spouse'
```

In [34]:
```
r = [report_noisy_max(adult['Martial Status'], options, score, 1, 1) for i i
pd.Series(r).value_counts()
```

Out[34]:
```
Married-civ-spouse    195
Never-married           5
dtype: int64
```

So the exponential mechanism can be replaced with report noisy max when the set $\mathcal{R}$ is finite, but what about when it's infinite? We can't easily add Laplace noise to an infinite set of scores. In this context, we have to use the actual exponential mechanism.

In practice, however, using the exponential mechanism for infinite sets is often challenging or impossible. While it's easy to write down the probability density function defined by the mechanism, it's often the case that no efficient algorithm exists for sampling from it. As a result, numerous theoretical papers appeal to the exponential mechanism to show that a differentially private algorithm "exists" with certain desirable properties, but many of these algorithms are impossible to use in practice.

The Exponential Mechanism as Fundamental Mechanism for ϵ-Differential Privacy

We've seen that it's not possible to recover the exponential mechanism using the Laplace mechanism plus sequential composition, because we can't capture the fact that the algorithm we designed doesn't release all of the noisy scores. What about the reverse - can we recover the Laplace mechanism from the exponential mechanism? It turns out that we can!

Consider a query $q(x) : \mathcal{D} \rightarrow \mathbb{R}$ with sensitivity Δq. We can release an ϵ-differentially private answer by adding Laplace noise: $F(x) = q(x) + \text{Lap}(\Delta q/\epsilon)$. The probability density function for this differentially private version of q is:

$$\begin{aligned}\Pr[F(x) = r] &= \frac{1}{2b}\exp\left(-\frac{|r-\mu|}{b}\right) \\ &= \frac{\epsilon}{2\Delta q}\exp\left(-\frac{\epsilon|r-q(x)|}{\Delta q}\right)\end{aligned}$$

Consider what happens when we set the scoring function for the exponential mechanism to $u(x, r) = -2|q(x) - r|$. The exponential mechanism says that we should sample from the probability distribution proportional to:

$$\begin{aligned}\Pr[F(x) = r] &= \exp\left(\frac{\epsilon u(x,r)}{2\Delta u}\right) \\ &= \exp\left(\frac{\epsilon(-2|q(x)-r|)}{2\Delta q}\right) \\ &= \exp\left(-\frac{\epsilon|r-q(x)|}{\Delta q}\right)\end{aligned}$$

So it's possible to recover the Laplace mechanism from the exponential mechanism, and we get the same results (up to constant factors - the general analysis for the exponential mechanism is not tight in all cases).

The exponential mechanism is extremely general - it's generally possible to re-define any ϵ-differentially private mechanism in terms of a carefully chosen definition of the scoring function u. If we can analyze the sensitivity of this scoring function, then the proof of differential privacy comes for free.

On the other hand, applying the general analysis of the exponential mechanism sometimes comes at the cost of looser bounds (as in the Laplace example above), and mechanisms defined in terms of the exponential mechanism are often very difficult to implement. The exponential mechanism is often used to prove theoretical lower bounds (by showing that a differentially private algorithm *exists*), but practical algorithms often replicate the same behavior using some other approach (as in the case of report noisy max above).