Privacy Concerns

2.1 Exploratory Analysis

[3]:
```
leaked = pd.read_csv('leaked.csv')
```

[4]:
```
leaked.iloc[range(10)]
```

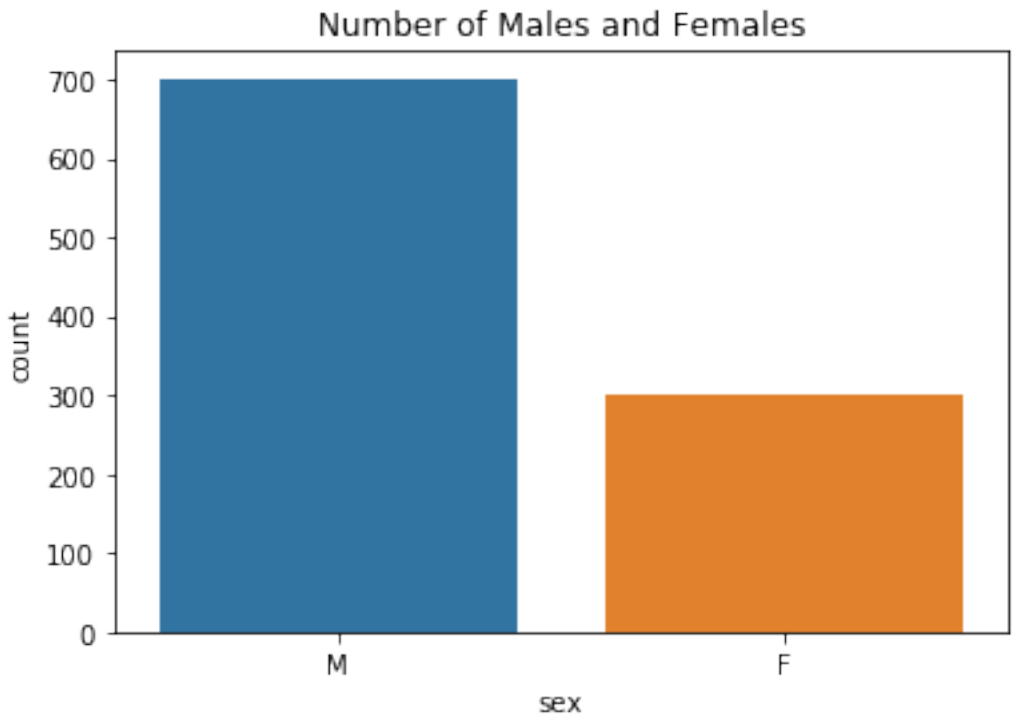
[4]:

	name	sex	zip	month	year
0	Avery Phillips	M	94709	3	1993
1	Grayson Rodriguez	M	94705	6	1998
2	Ethan Baker	M	94712	1	1998
3	Carter Wright	M	94720	7	1995
4	Elijah Young	M	94706	2	1996
5	Luke Parker	M	94720	8	1992
6	Daniel Phillips	M	94708	5	1997
7	Joseph Gonzalez	M	94708	3	2000
8	Andrew Baker	M	94709	4	1990
9	Carter Miller	M	94704	11	1997

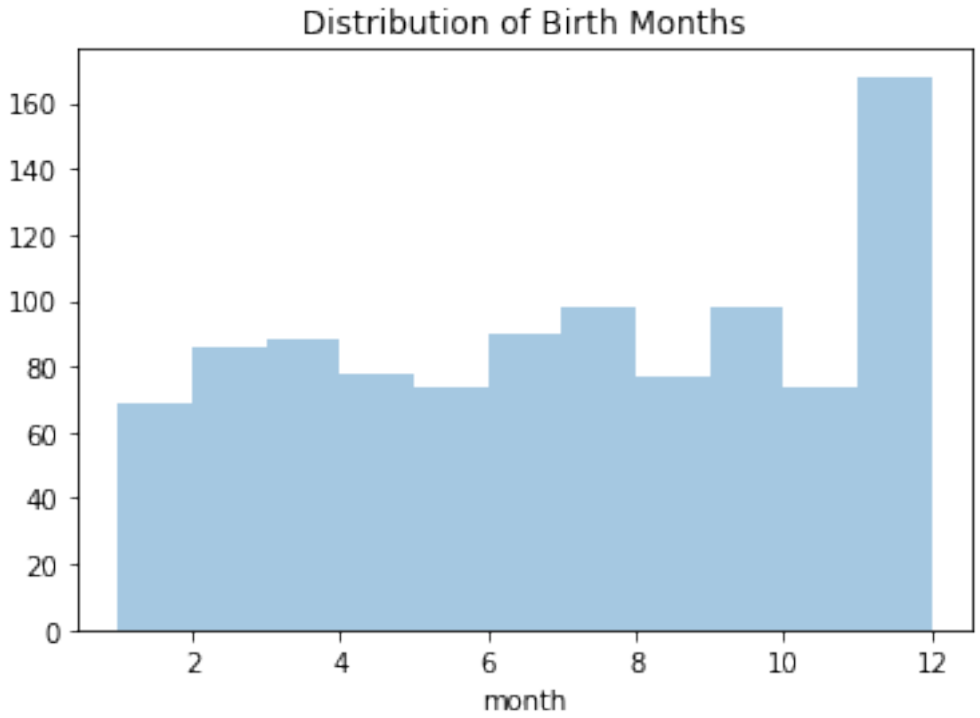
[5]:
```
leaked.columns
```

[5]:
```
Index(['name', 'sex', 'zip', 'month', 'year'], dtype='object')
```

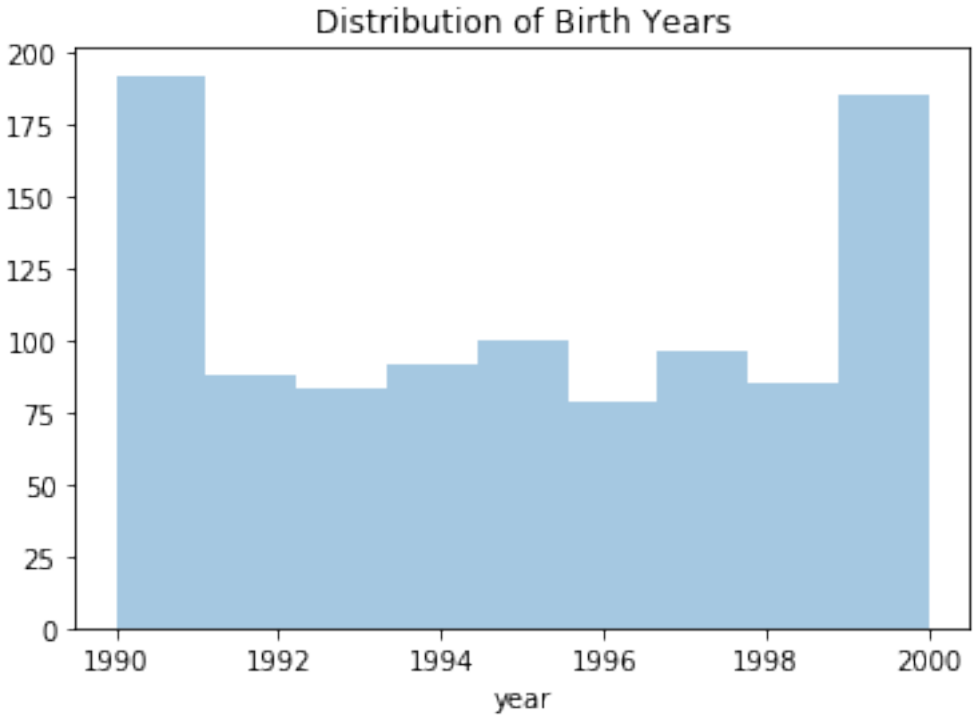
[6]:
```
sns.countplot(leaked['sex'])
plt.title('Number of Males and Females');
```



[7]:
```
sns.distplot(leaked['month'], kde = False)
plt.title('Distribution of Birth Months');
```



[8]:
```
sns.distplot(leaked['year'], kde = False)
plt.title('Distribution of Birth Years');
```



I wouldn't say that any of these distributions are that uniform because they seem to birth months seems to have a much larger proportion of values born in November and December despite the other months being seemingly uniform. Birth years also has a similar phenomenon with a majority of the years to be somewhat uniform in their distribution, there is the anomaly of more people born at the beginning and end of the decade of the nineties. The distribution of what was referred to as gender in part 1 and sex in part 2 are pretty similar. There are much more males than females in the population.

2.2 Simple Proof of Concept

[9]:
```
years = leaked.year.unique()
months = leaked.month.unique()
sex = leaked.sex.unique()
```

[10]:
```
combos = []
for y in years:
    for m in months:
        for s in sex:
            filt = leaked[(leaked['year'] == y) & (leaked['month'] == m) &␣
 ↪(leaked['sex'] == s)]
```

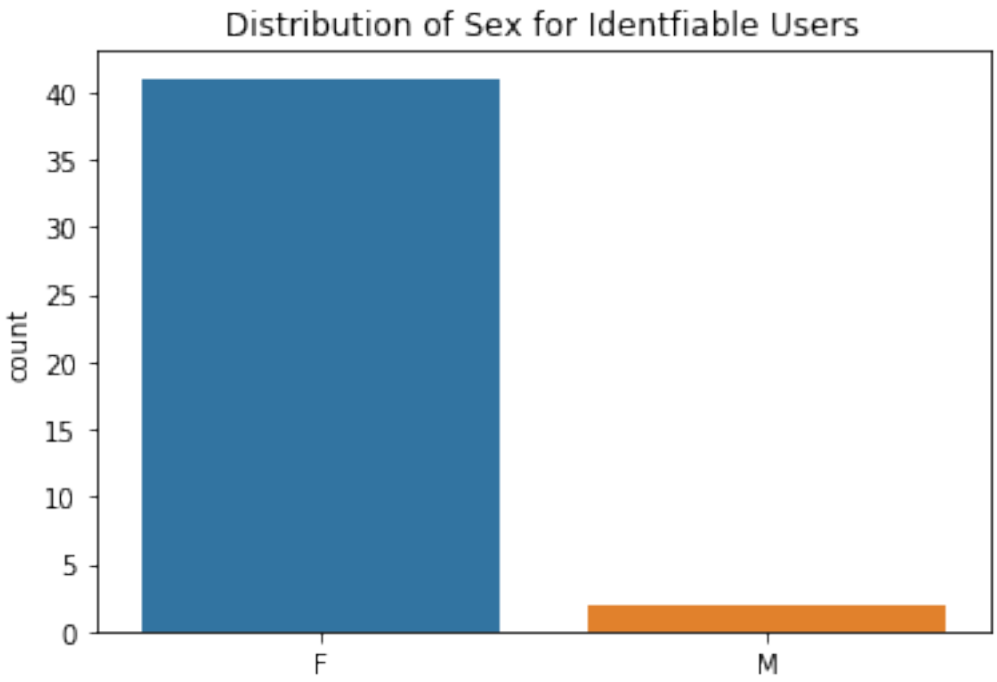
```
            if filt.shape[0] == 1:
                combos.append([s, m, y])
```

[11]:
```
len(combos)
```

[11]:
```
43
```

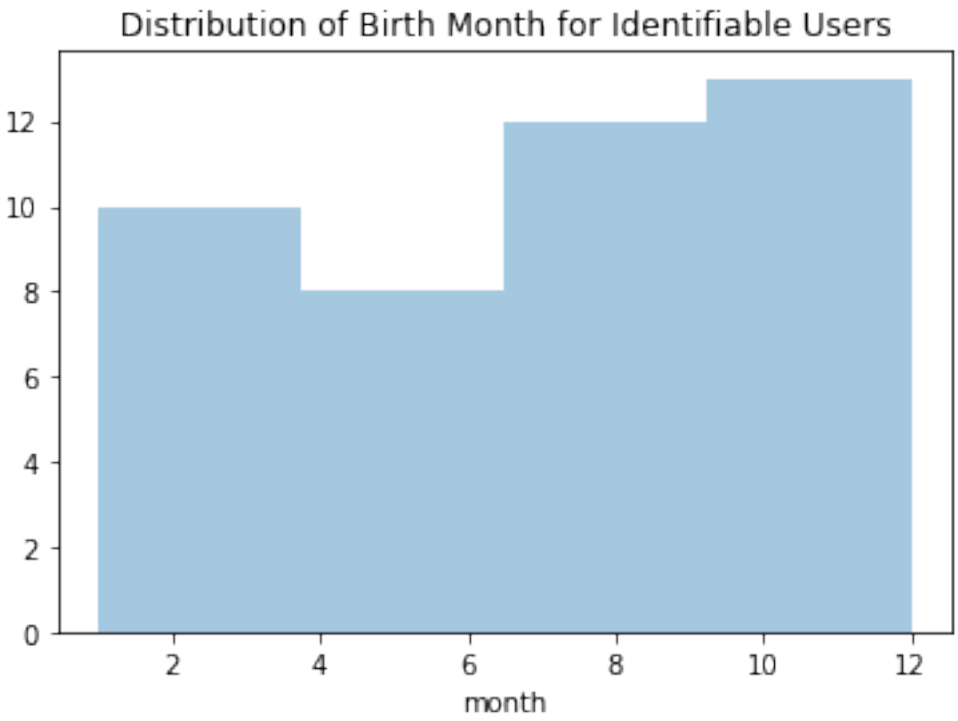
There are 43 identifiable users that can be isolated based on just those three attributes.

[12]:
```
identifiable_sex = [c[0] for c in combos]
sns.countplot(identifiable_sex)
plt.title('Distribution of Sex for Identfiable Users');
```



The distribution for the sex of the user in the set of identifiable users differs from the overall distribution that we say in section 2.1 because the users that we were able to identify from just their sex, birth month, and birth year probably have attributes that are not that common in the dataset. Males were much more common in the data, so it makes sense that the people that we were able to distinguish were mainly female not male.

[13]:
```
identifiable_birth_month = [c[1] for c in combos]
sns.distplot(identifiable_birth_month, kde = False)
plt.title('Distribution of Birth Month for Identifiable Users')
plt.xlabel('month');
```



The distribution plotted doesn't match the overall data's distribution of birth months for because these identifiable users are unique in some way, they probably do not have very common attributes to the data overall. If we were take a random sample of users, the sample's distribution would be similar to the population, but the set of these users is not random. Their attributes made them unique so the data is biased and not very representative of the population. This distribution is however closer to the original data's distribution of birth months than the distribution above for sex.

[14]:
```
berkeley = pd.read_csv('berkeley.csv')
berkeley.iloc[range(5)]
```

[14]:

	sex	month	year	start	end
0	M	12	1999	94704	94704
1	M	12	2000	94706	94706
2	M	4	1999	94703	94703
3	M	2	1999	94710	94710
4	M	12	1994	94707	94707

[15]:
```
def extract_scooter_rentals(attributes):
    return berkeley[(berkeley['sex'] == attributes[0]) & (berkeley['month'] ==␣
 ↪attributes[1]) & (berkeley['year'] == attributes[2])]
```

[16]:
```
# just displaying the first 10 rows
extract_scooter_rentals(combos[0]).iloc[range(10)]
```

[16]:

	sex	month	year	start	end
311	F	7	1993	94709	94709

```
1319   F      7  1993  94708  94709
2135   F      7  1993  94709  94707
2667   F      7  1993  94709  94709
3624   F      7  1993  94709  94709
4467   F      7  1993  94710  94709
4812   F      7  1993  94709  94709
5766   F      7  1993  94705  94708
7895   F      7  1993  94709  94709
8485   F      7  1993  94709  94709
```

2.3 A More Elaborate Attack

[17]:
```
# users will tend to rent scooters with start and end locations with same␣
 ↪zipcode as address
# users will rent a bike from another (uniformly sampled) zip code with␣
 ↪probability p1
# similarly users will end a rental from another zip code with probability p2
```

[18]:
```
def address_from_attributes(attributes):
    return leaked[(leaked['sex'] == attributes[0]) & (leaked['month'] ==␣
 ↪attributes[1]) & (leaked['year'] == attributes[2])].iloc[0]['zip']
```

[19]:
```
lst = []
for i in np.arange(len(combos)):
    c = combos[i]
    add = address_from_attributes(c)
    tab = extract_scooter_rentals(c)
    tab['address'] = add
    lst.append(tab)
```

```
/srv/conda/envs/data102/lib/python3.7/site-packages/ipykernel_launcher.py:6:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

[20]:
```
combined = pd.concat(lst, ignore_index = True)
```

[21]:
```
def find_percentages(tbl):
    start_dif = 0
    end_dif = 0
    rows = tbl.shape[0]
    for i in np.arange(tbl.shape[0]):
        rw = tbl.iloc[i]
        add = rw['address']
        if rw['start'] != add:
```

```
            start_dif += 1
        if rw['end'] != add:
            end_dif += 1
    return start_dif / rows , end_dif / rows
```

[22]:
```
p1, p2 = find_percentages(combined)
```

[23]:
```
p1
```

[23]:
```
0.09659863945578231
```

[24]:
```
p2
```

[24]:
```
0.2780045351473923
```

The way that I went about estimating the parameters for p_1 and p_2 was by extracting the address zip code of all of the identifiable users and appending it to the rows from the berkeley dataset that pertained to that user. I combined all of the rows of all of the identifiable users into one table and then estimated p_1 to be the proportion of rides where the starting zip code differed from the address zip code and estimated p_2 to be the proportion of rides where the ending zip code differed from the address zip code.

[25]:
```
p1s = []
p2s = []
numrows = combined.shape[0]
for i in np.arange(1000):
    boot = combined.sample(n = numrows, replace = True)
    p1_b, p2_b = find_percentages(boot)
    p1s.append(p1_b)
    p2s.append(p2_b)
```

[26]:
```
lower_p1 = np.percentile(p1s, 2.5)
upper_p1 = np.percentile(p1s, 97.5)
lower_p2 = np.percentile(p2s, 2.5)
upper_p2 = np.percentile(p2s, 97.5)
```

[27]:
```
(lower_p1, upper_p1)
```

[27]:
```
(0.08479591836734694, 0.10976190476190475)
```

[28]:
```
(lower_p2, upper_p2)
```

[28]:
```
(0.2598639455782313, 0.2970634920634921)
```

To generate 95% confidence intervals for p_1 and p_2, I used bootstrapping. I created 2 lists one to collect bootstrapped values of p_1 and another to collect values for p_2 and repeating the procedure of resampling from the original table containing all of the data with replacement and appending the estimates for p_1 and p_2 found from the resampled table to their respective collective lists one thousand times. I then used a numpy's percentile function to calculate the 2.5th and the 97.5th percentile for each list to find the lower and upper endpoint of the 95% confidence intervals for the estimates.

[29]:
```
theoretical_combos = []
for s in leaked.sex.unique():
    for z in leaked.zip.unique():
```

```
        for m in leaked.month.unique():
            for y in leaked.year.unique():
                filt = leaked[(leaked['sex'] == s) & (leaked['zip'] == z) &␣
↪(leaked['month'] == m) & (leaked['year'] == y)]
                if filt.shape[0] == 1:
                    theoretical_combos.append([s, z, m, y])
```

[30]:
```
len(theoretical_combos)
```

[30]:
```
720
```

There are 720 "theoretical identifiable users".

[45]:
```
tst = berkeley.iloc[1]
```

[46]:
```
leaked_tst_1 = leaked[(leaked['sex'] == tst['sex']) & (leaked['zip'] ==␣
↪tst['start']) & (leaked['month'] == tst['month']) & (leaked['year'] ==␣
↪tst['year'])]
```

[47]:
```
leaked_tst_1
```

[47]:

	name	sex	zip	month	year
585	James Lopez	M	94706	12	2000
694	Wyatt Smith	M	94706	12	2000

[48]:
```
leaked_tst_2 = leaked[(leaked['sex'] == tst['sex']) & (leaked['zip'] ==␣
↪tst['end']) & (leaked['month'] == tst['month']) & (leaked['year'] ==␣
↪tst['year'])]
```

[49]:
```
leaked_tst_2
```

[49]:

	name	sex	zip	month	year
585	James Lopez	M	94706	12	2000
694	Wyatt Smith	M	94706	12	2000

[50]:
```
# possibilities most likely trip starts and ends in same zipcode as address
# next most likely trip starts in same zipcode but ends in different zipcode
# next after that is starts in dif zipcode but ends in same zipcode
# less likely is that it starts and ends in different zipcode
```

[63]:
```
def guess_user(trip):
    start_same = leaked[(leaked['sex'] == trip['sex']) & (leaked['zip'] ==␣
↪trip['start']) & (leaked['month'] == trip['month']) & (leaked['year'] ==␣
↪trip['year'])]
    end_same = leaked[(leaked['sex'] == trip['sex']) & (leaked['zip'] ==␣
↪trip['end']) & (leaked['month'] == trip['month']) & (leaked['year'] ==␣
↪trip['year'])]
    # most likely start and end same place and if more than one randomly choose␣
↪one
    if sum(np.array(start_same['name']) == np.array(end_same['name'])) ==␣
↪start_same.shape[0]:
        return start_same.sample().iloc[0]['name']
    # starts same zipcode but ends in different zipcode
```

```
    elif start_same.shape[0] >= 1:
        return start_same.sample().iloc[0]['name']
    # ends same zipcode but starts in different zipcode
    elif end_same.shape[0] >= 1:
        return end_same.sample().iloc[0]['name']
    # starts and ends in different zipcode
    # first filter just by sex, month, and year then randomly sample
    else:
        filt = leaked[(leaked['sex'] == trip['sex']) & (leaked['month'] ==␣
↪trip['month']) & (leaked['year'] == trip['year'])]
        return filt.sample().iloc[0]['name']
```

2.4 Takeaways

[64]:
```
leaked.shape[0]
```

[64]:
```
1000
```

My takeaway from this section is like what we learned in lecture is that the combination of seemly unimportant or very basic info can actually be very identifying. Just using the info of zipcodes, sex, birth month, and birth year we were able to make some pretty good guesses for about 720 users and actually identify 43 people out of 1000 users data that were leaked. I think one of the really important things I noticed was with just a little inference with finding out the percentage of people that start or end trips outside of their zipcode allowed for so many more identifications of people (the jump from 43 identifiable users to possibly 720 theoretically identifiable users).

Assuming that the already released data can not be changed since the original set of it is already out in the world and can be found, I think that releasing the rest of the data could actually be helpful in decreasing the odds of people being identified since the algorithm that we developed randomly chooses from people who are equally likely. It might even be beneficial to augment the released data with created data if that's allowed because if we were to add more people to the mix filtering just based on sex, zipcode, birth month, and birth year would yield more results and the chance of randomly picking the true real person in the mix of other "people" would reduce the likelihood of their identification.

If we were to release future datasets, I would say that if we were to remove any info it would firt be removing the sex column. The data is mainly male, so females were much more vulnerable in this attack and this would make it more fair. To remove or change other data from future datasets, removing birth months followed by birth years would be good choices. If we wanted to keep some sort of indicator for their birthday in the dataset, I think that we could make the data more general. Someone born in the months if January, February, and March would be considered born in the first quarter of months and the other groups of months would follow. We could also do something similarly to birth years by splitting the years into groups and labeling them that way instead of having the exact year. However, it is really scary that with just a few inferences we can make relatively good predictions for things if we were to get a hold of some released information, so obviously the best choice would to not release the data, but if we had to I think making some of the modifications above could help with our privacy.