Data_102_Project_01

December 8, 2019

[1]:
```
#libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import re
```

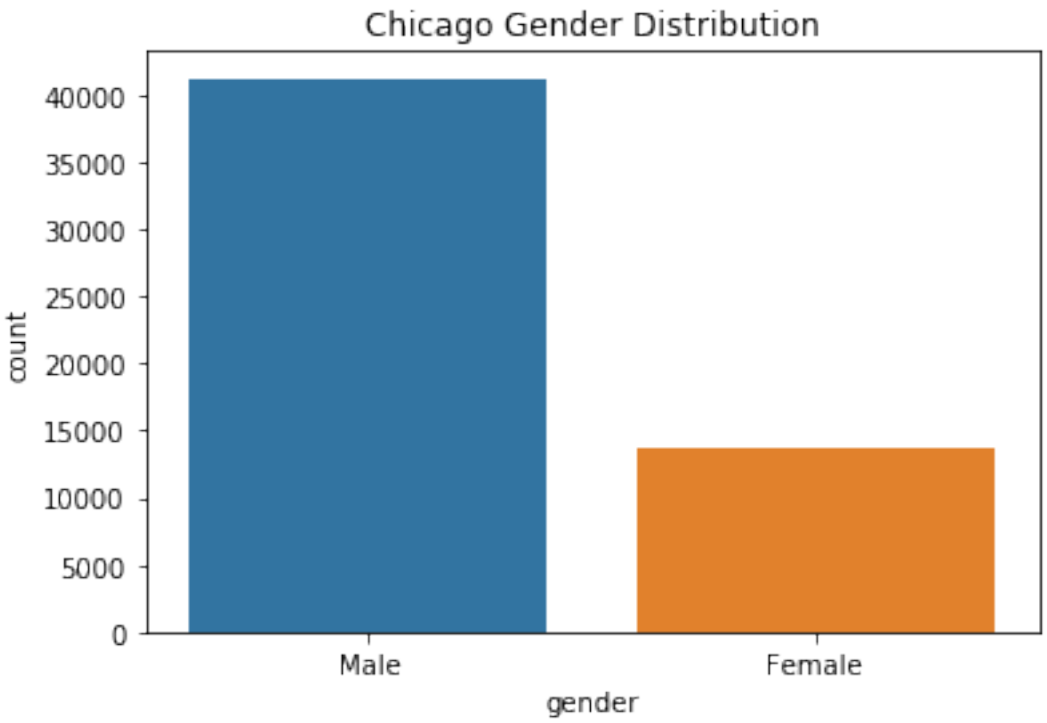
[2]:
```
#load datasets
ny = pd.read_csv('ny.csv')
dc = pd.read_csv('dc.csv')
chicago = pd.read_csv('chicago.csv')
day = pd.read_csv('day.csv')
```

0.0.1 1 Preliminary Data Analysis

1.1 Demographic Information

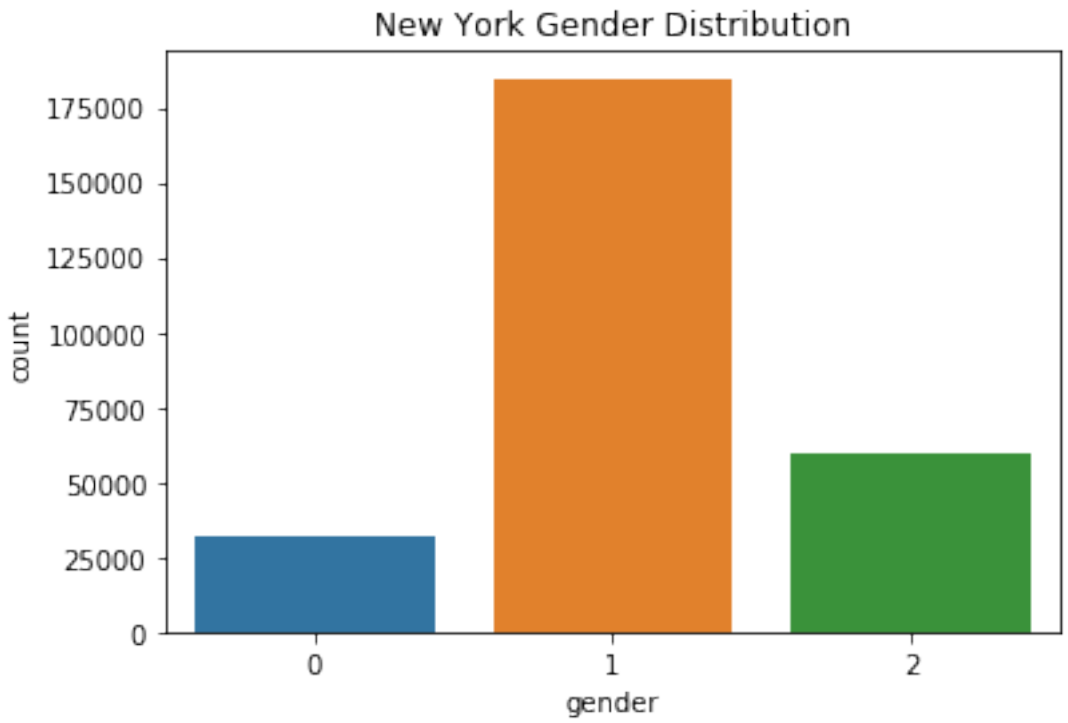
1. Plot the distribution of male to female riders for chicago.csv

[3]:
```
sns.countplot(chicago['gender'])
plt.title('Chicago Gender Distribution');
```



2. Plot the distribution of the gender column for ny.csv

[4]:
```
sns.countplot(ny['gender'])
plt.title('New York Gender Distribution');
```

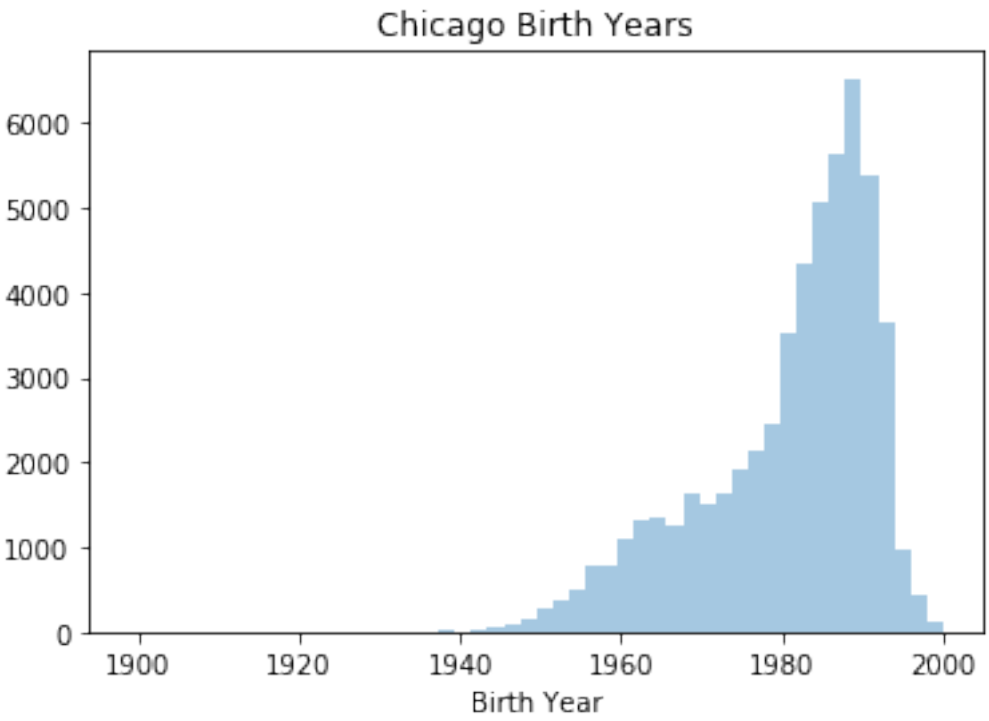


3. Given the results in Chicago, make an educated guess as to the mapping from numerical value to Male/Female/Unspecified within the ny.csv dataset.

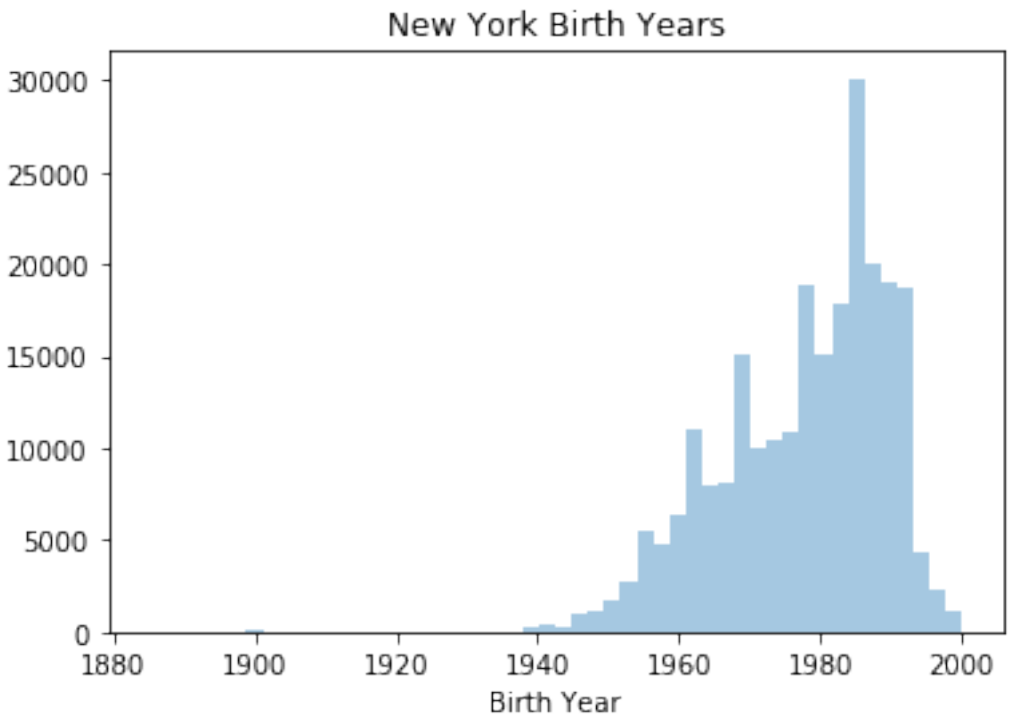
If I were to guess the mapping from numerical to male/female/unspecified in the ny.csv dataset from the chicago.csv dataset. I would say 1 is male, 2 is female, and 0 is unspecified.

4. Plot the distribution of the birth years of bike renters in Chicago and NY.

[5]:
```
sns.distplot(chicago['birthyear'][chicago['birthyear'].notnull()], hist = True,␣
 ↪kde = False)
plt.title('Chicago Birth Years')
plt.xlabel('Birth Year');
```



[6]:
```
sns.distplot(ny['birth year'][ny['birth year'].notnull()], hist = True, kde =␣
↪False)
plt.title('New York Birth Years')
plt.xlabel('Birth Year');
```



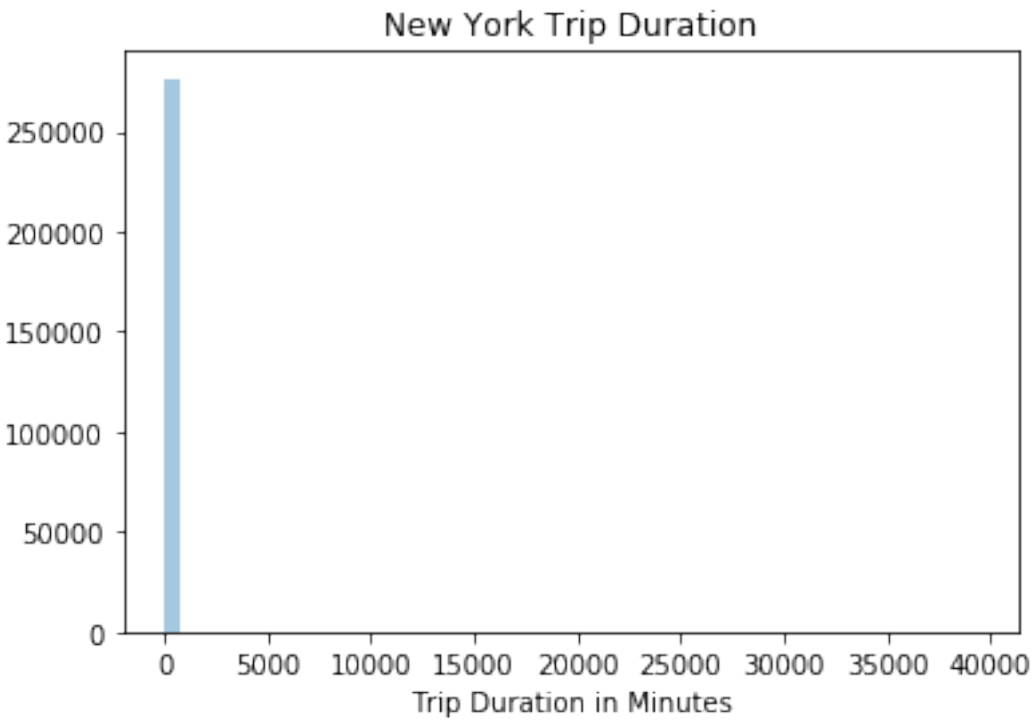
5. Discuss the results you observed in the age plots and whether this fits with your intuition. Would you remove any data? If so, why?

There seems to be some very minimal amount of people in both Chicago and New York that are born before 1940 and rather small chunk of people born at about 1900 in New York. I would probably decide to remove any data prior to 1940 because it seems like an extremely small subset of the population. Another reason that that I would remove this data is because in 2016 if people were born before 1940 would be over 76 years old and in the US the typical life expectancy is only around there, so I wouldn't expect for the population generally to have people older than that.

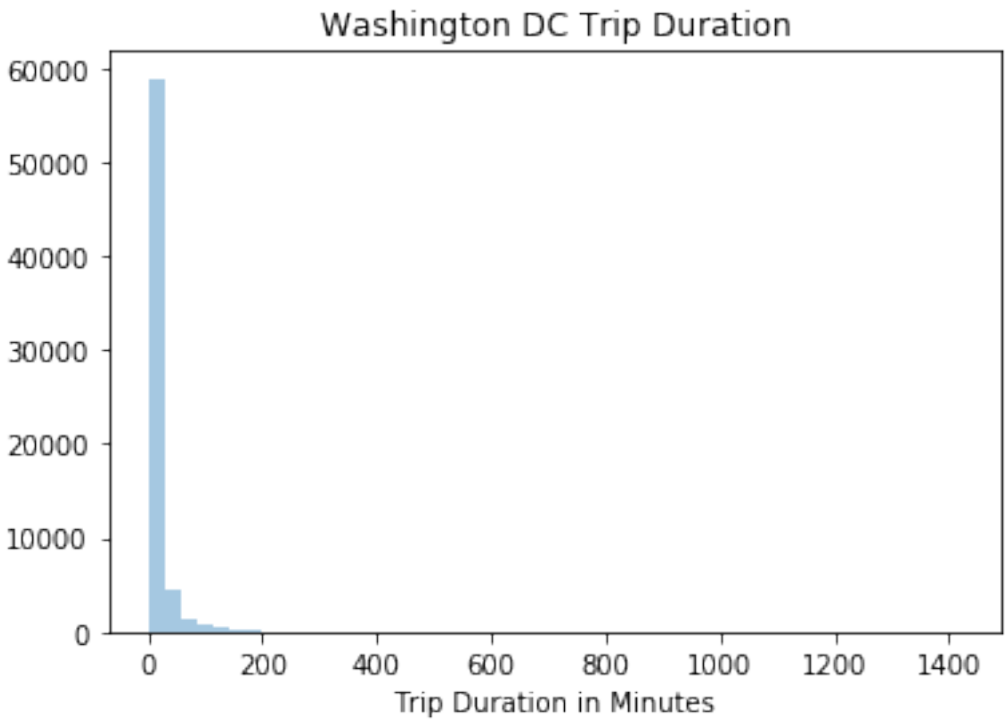
1.2 Rental Times

1. Plot the three distribution of trip duration in minutes across all three cities.

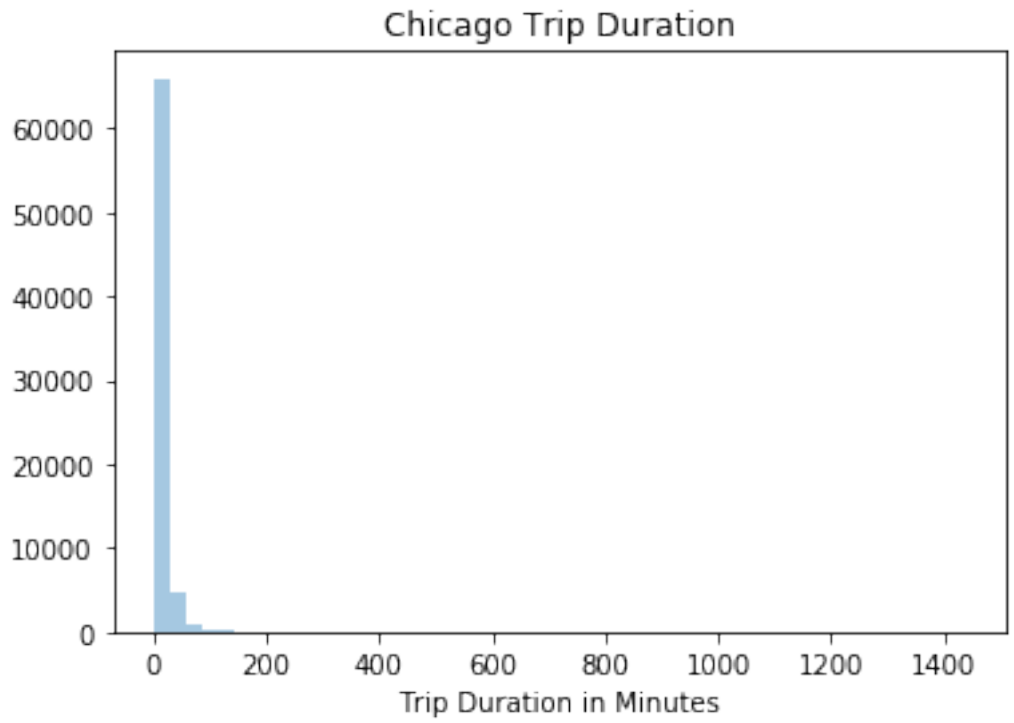
[7]:
```
# new york trip duration measured in seconds
sns.distplot(ny['tripduration'] / 60, kde = False, hist = True)
plt.title('New York Trip Duration')
plt.xlabel('Trip Duration in Minutes');
```



[8]:
```
# dc duration in miliseconds
sns.distplot(dc['Duration (ms)'] / 60000, kde = False, hist = True)
plt.title('Washington DC Trip Duration')
plt.xlabel('Trip Duration in Minutes');
```

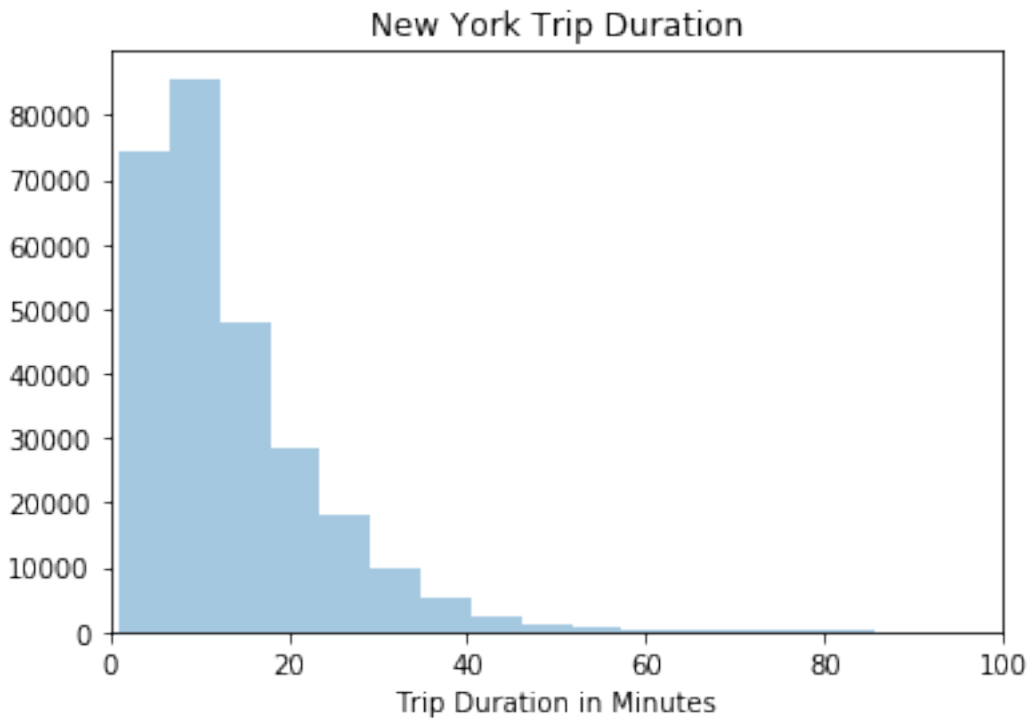


[9]:
```
# chicago trip duration in seconds
sns.distplot(chicago['tripduration'] / 60, kde = False, hist = True)
plt.title('Chicago Trip Duration')
plt.xlabel('Trip Duration in Minutes');
```

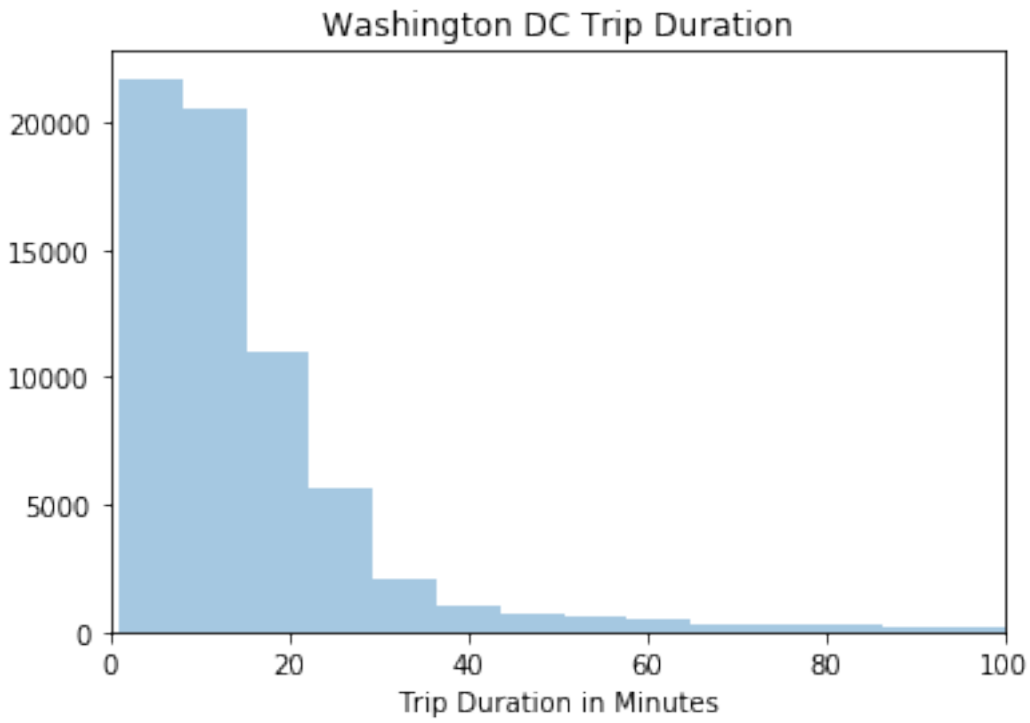


2. Are the plots you generated useful? If not, plot them again so that the visualisation is more useful.

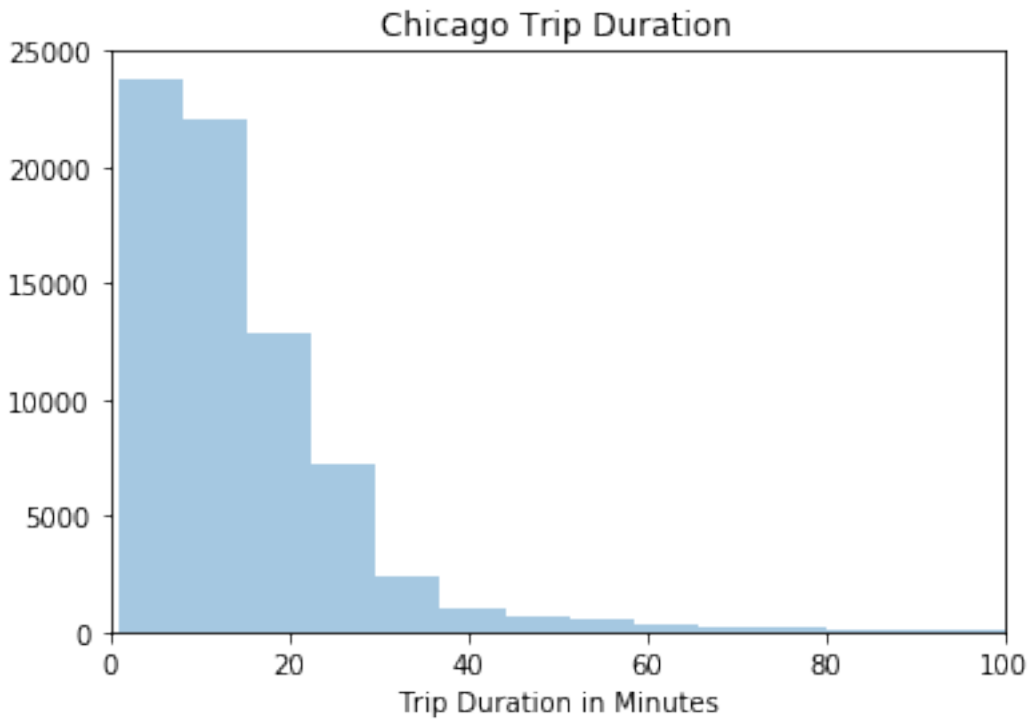
[10]:
```
sns.distplot(ny['tripduration'] / 60, kde = False, bins = 7000, hist = True)
plt.xlim((0, 100))
plt.title('New York Trip Duration')
plt.xlabel('Trip Duration in Minutes');
```



[11]:
```
sns.distplot(dc['Duration (ms)'] / 60000, kde = False, bins = 200, hist = True)
plt.xlim((0, 100))
plt.title('Washington DC Trip Duration')
plt.xlabel('Trip Duration in Minutes');
```

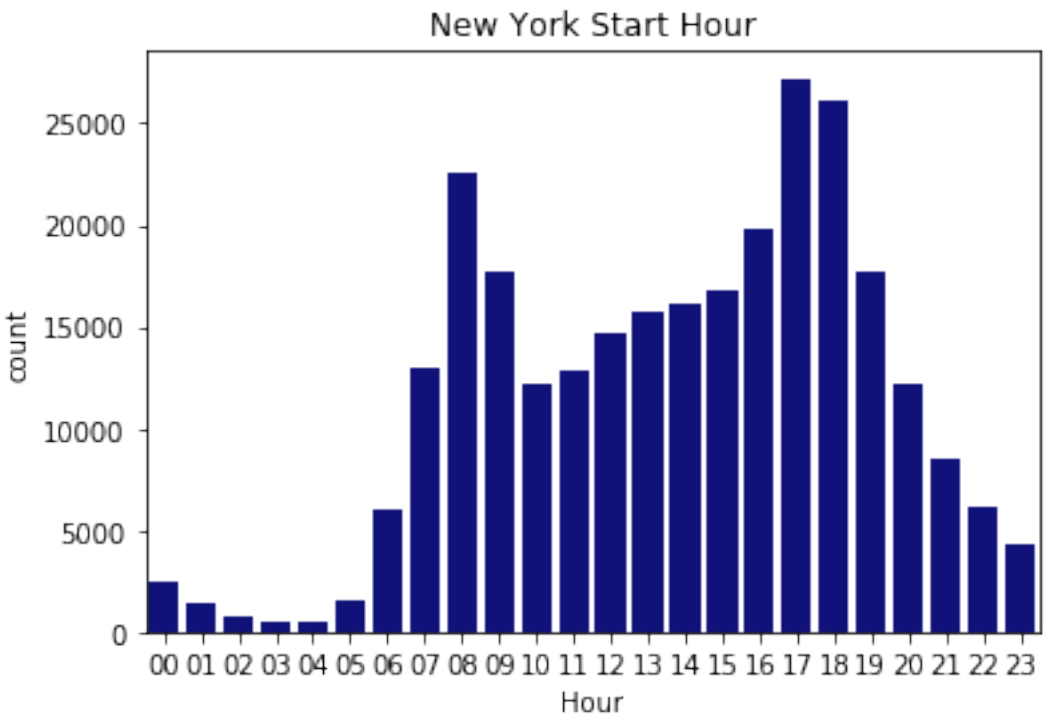


[12]:
```
sns.distplot(chicago['tripduration'] / 60, kde = False, bins = 200, hist = True)
plt.xlim((0, 100))
plt.title('Chicago Trip Duration')
plt.xlabel('Trip Duration in Minutes');
```

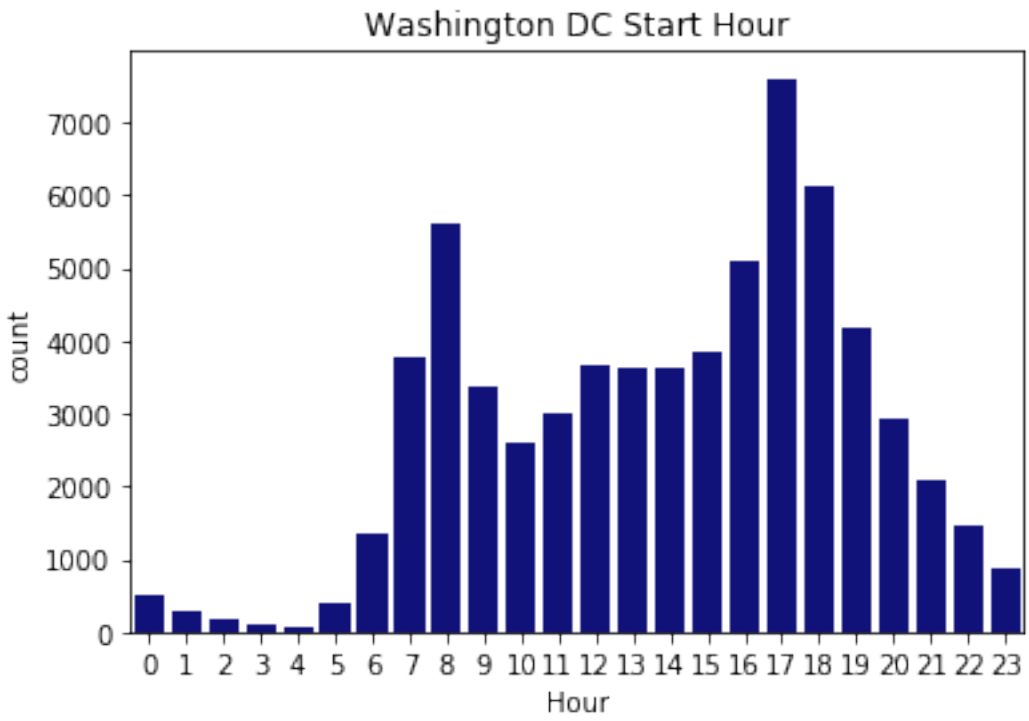


3. Plot the start time of trips split by hour for all three cities

[13]:
```
sns.countplot(ny['starttime'].apply(lambda x: re.findall(r'\s(\d{2}):', x)[0]),␣
↪color = 'darkblue')
plt.title('New York Start Hour')
plt.xlabel('Hour');
```

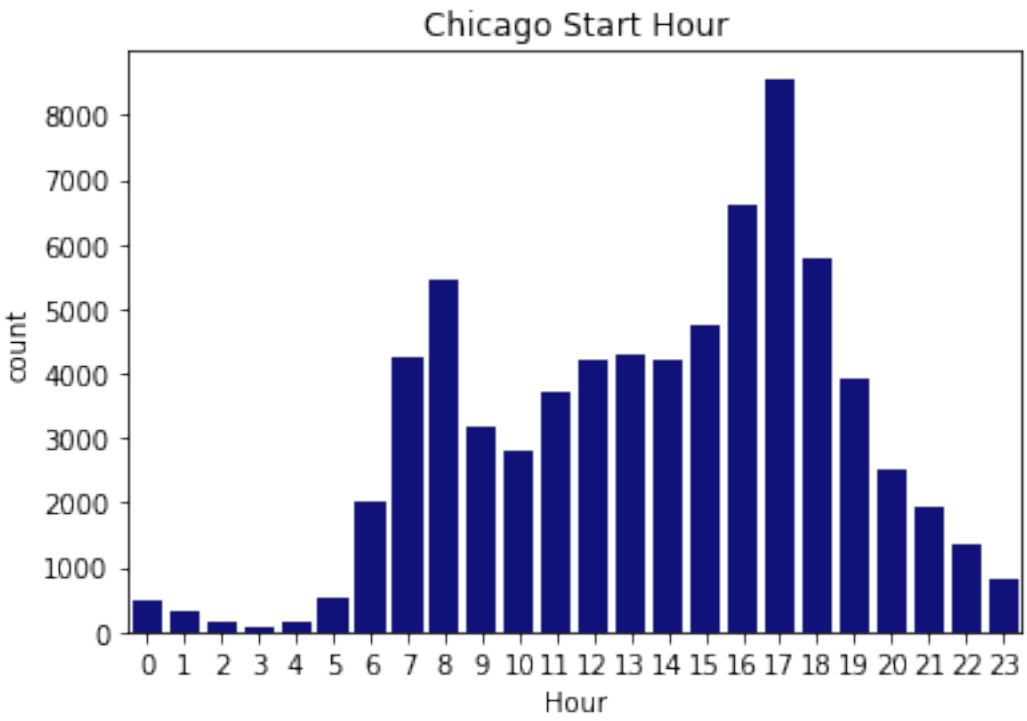


[14]:
```
sns.countplot(dc['Start date'].apply(lambda x: re.findall(r'\s(.+):', x)[0]),␣
 ↪order = [str(i) for i in np.arange(24)], color = 'darkblue')
plt.title('Washington DC Start Hour')
plt.xlabel('Hour');
```



[15]:
```
def strip_leading_0(st):
    if len(st) == 2 and st[0] == '0':
        return st[1]
    else:
        return st
```

[16]:
```
sns.countplot(chicago['starttime'].apply(lambda x: re.findall(r'\s(\d+):',␣
 ↪x)[0]).apply(strip_leading_0), order = [str(i) for i in np.arange(24)],␣
 ↪color = 'darkblue')
plt.title('Chicago Start Hour')
plt.xlabel('Hour');
```



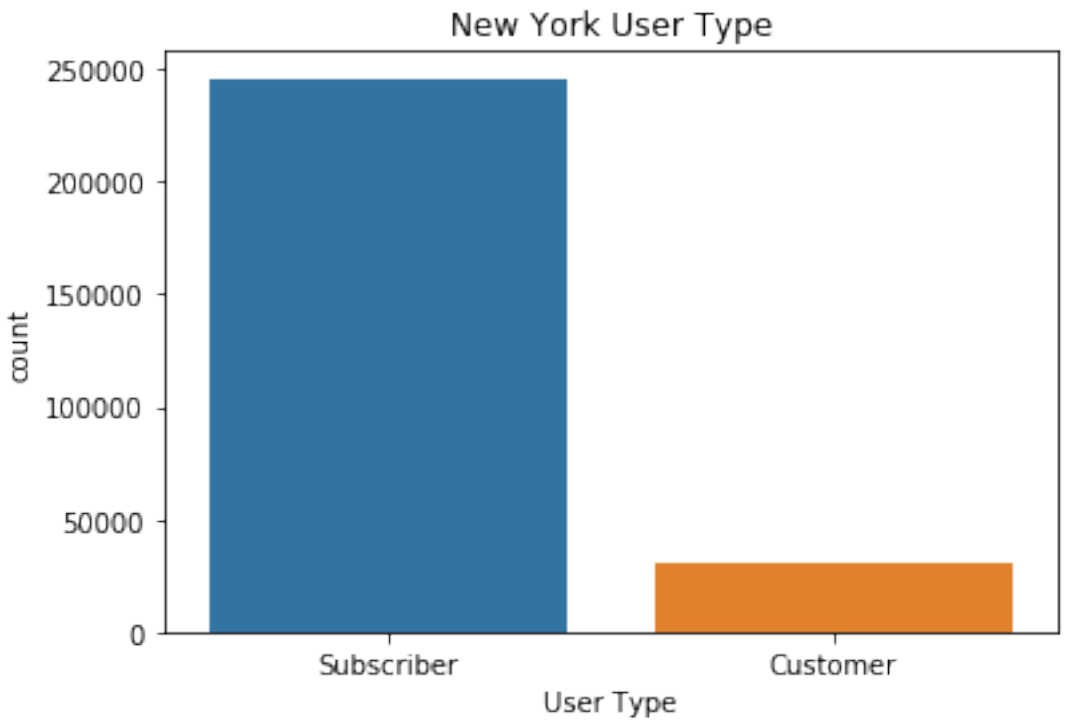
4. Discuss the results you observed in the start time plots. Do they fit your intuition?

In the results, the observed start times are what I expected. There were not very many rides from about midnight to 5 am. The number of rides started to increase from that time onwards and peaked around 8 am and the 17th hour which is 5pm which makes sense because those are very common commuting times for people to go to work.

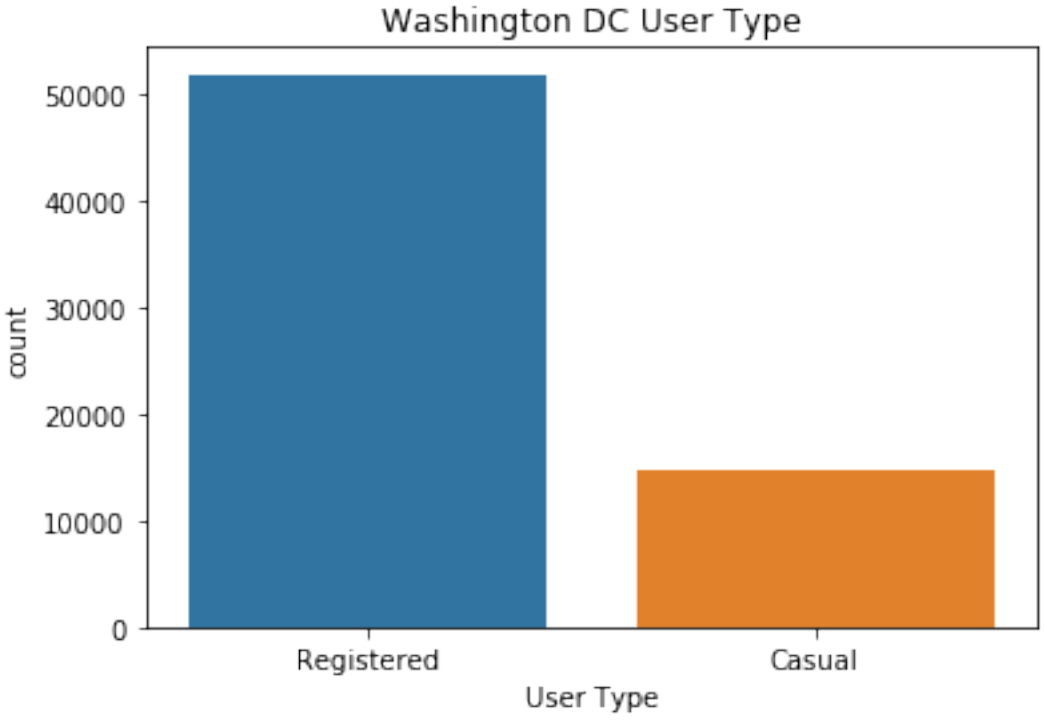
1.3 Further Exploration

1. Visualize three more attributes

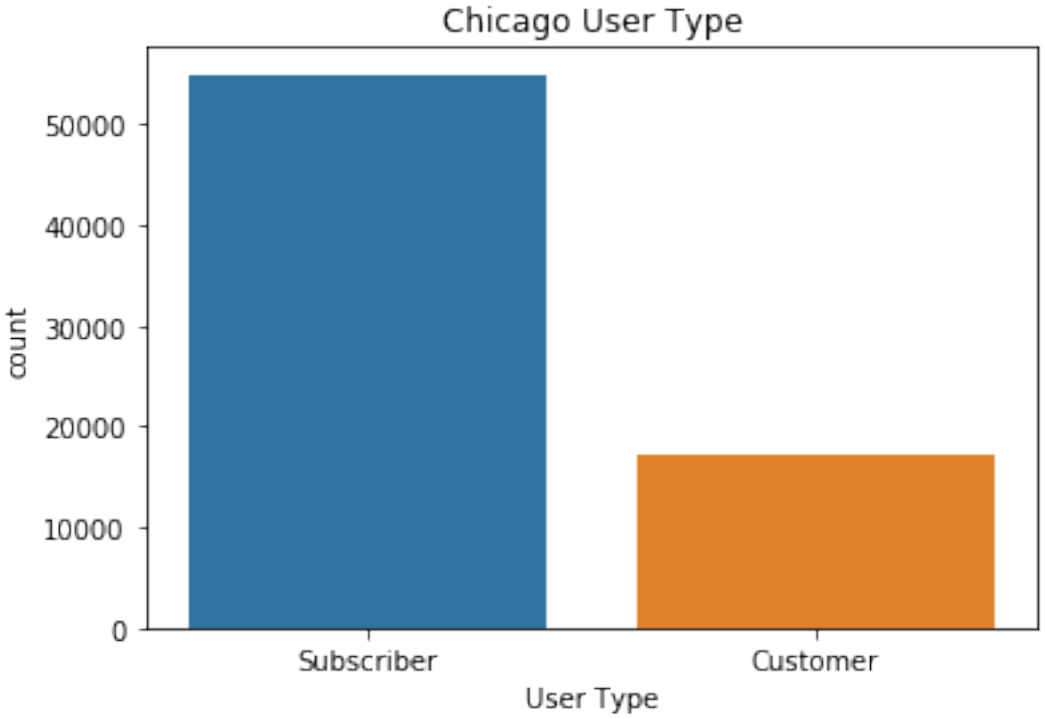
[17]:
```
sns.countplot(ny['usertype'], order = ['Subscriber', 'Customer'])
plt.xlabel("User Type")
plt.title('New York User Type');
```



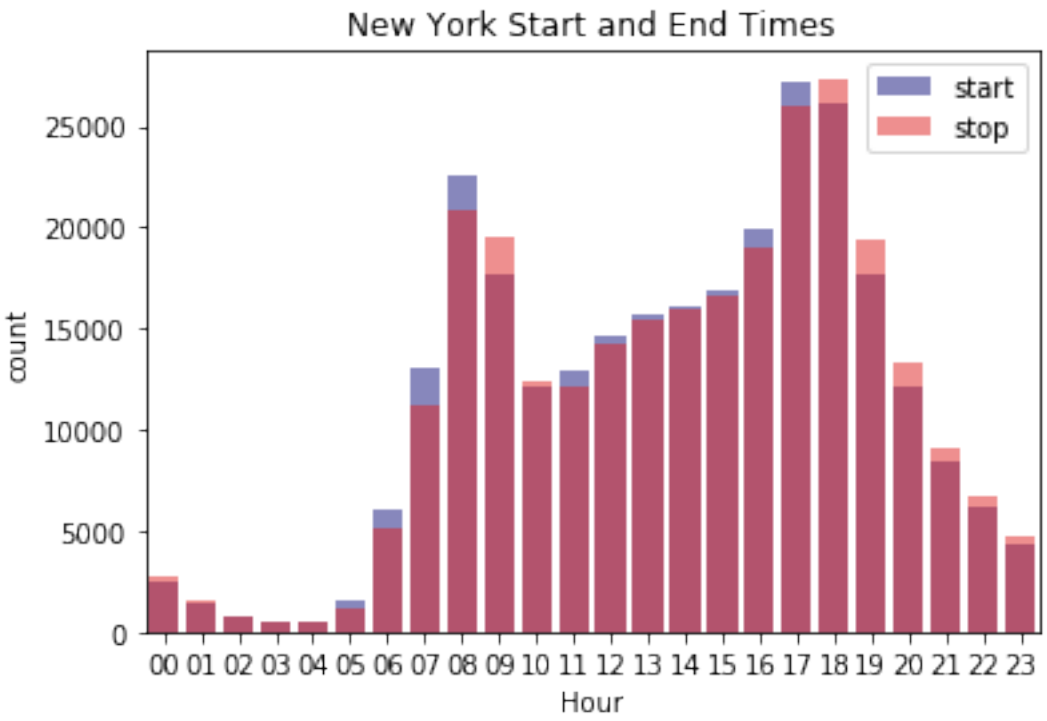
[18]:
```
sns.countplot(dc['Member Type'])
plt.xlabel('User Type')
plt.title('Washington DC User Type');
```



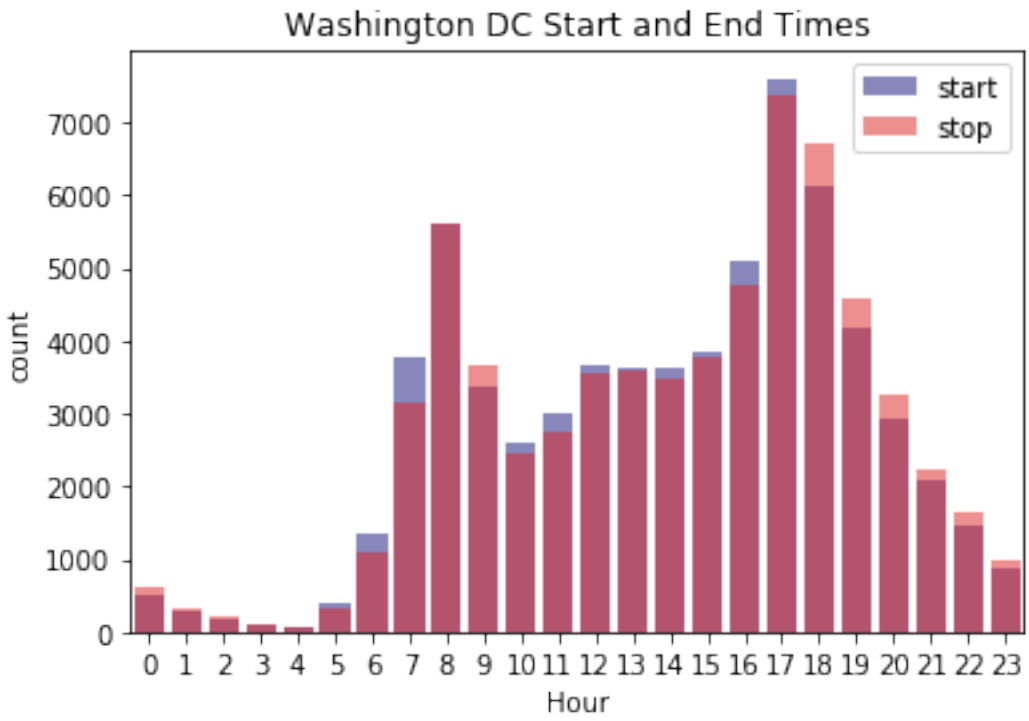
[19]:
```
sns.countplot(chicago['usertype'])
plt.xlabel('User Type')
plt.title('Chicago User Type');
```



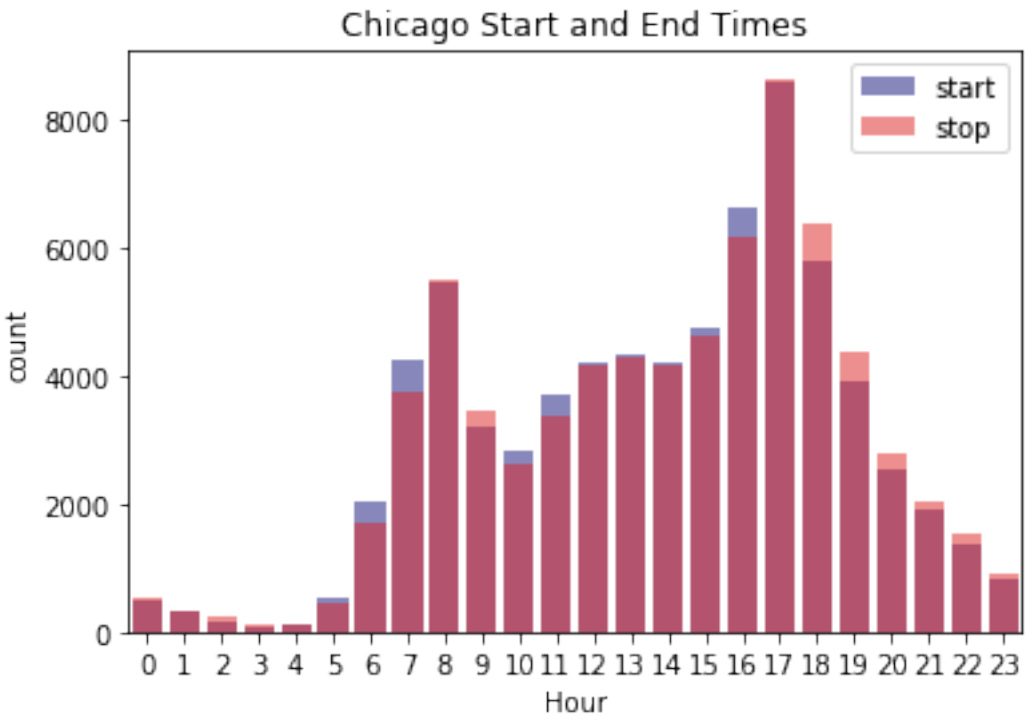
[20]:
```
fig, ax = plt.subplots()
sns.countplot(ny['starttime'].apply(lambda x: re.findall(r'\s(\d{2}):', x)[0]),␣
 ↪color = 'darkblue', ax = ax, label = 'start', alpha = 0.5)
sns.countplot(ny['stoptime'].apply(lambda x: re.findall(r'\s(\d{2}):', x)[0]),␣
 ↪color = 'red', ax = ax, label = 'stop', alpha = 0.5)
plt.title('New York Start and End Times')
plt.legend()
plt.xlabel('Hour');
```



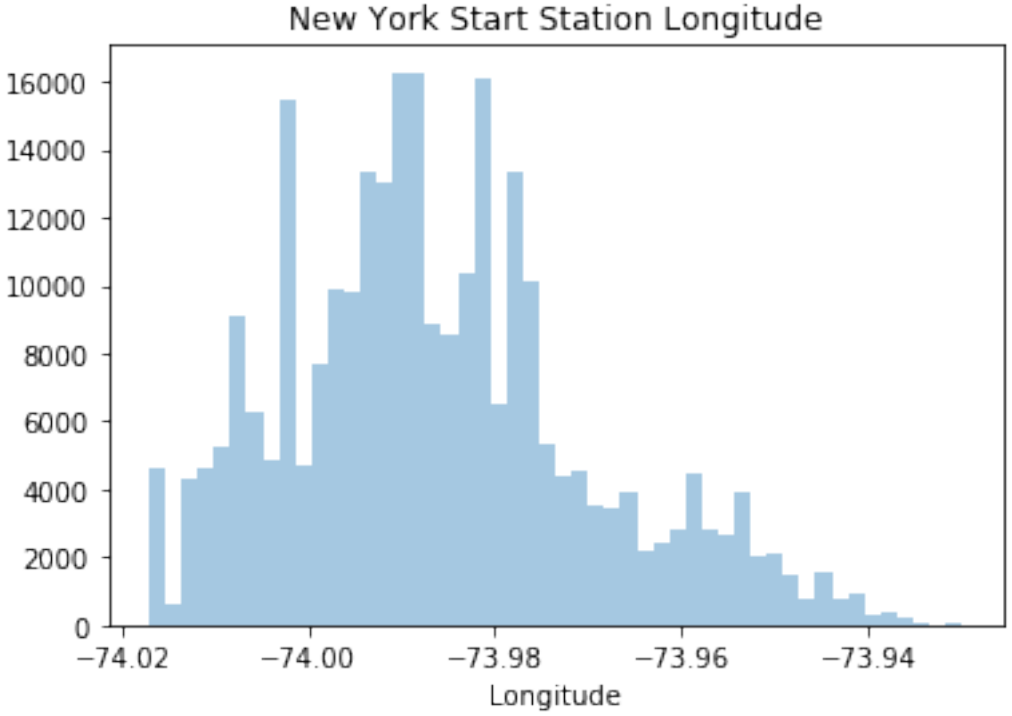
[21]:
```
fig, ax = plt.subplots()
sns.countplot(dc['Start date'].apply(lambda x: re.findall(r'\s(.+):', x)[0]),␣
 ↪order = [str(i) for i in np.arange(24)], color = 'darkblue', ax = ax, label␣
 ↪= 'start', alpha = 0.5)
sns.countplot(dc['End date'].apply(lambda x: re.findall(r'\s(.+):', x)[0]),␣
 ↪order = [str(i) for i in np.arange(24)], color = 'red', ax = ax, label =␣
 ↪'stop', alpha = 0.5)
plt.title('Washington DC Start and End Times')
plt.legend()
plt.xlabel('Hour');
```



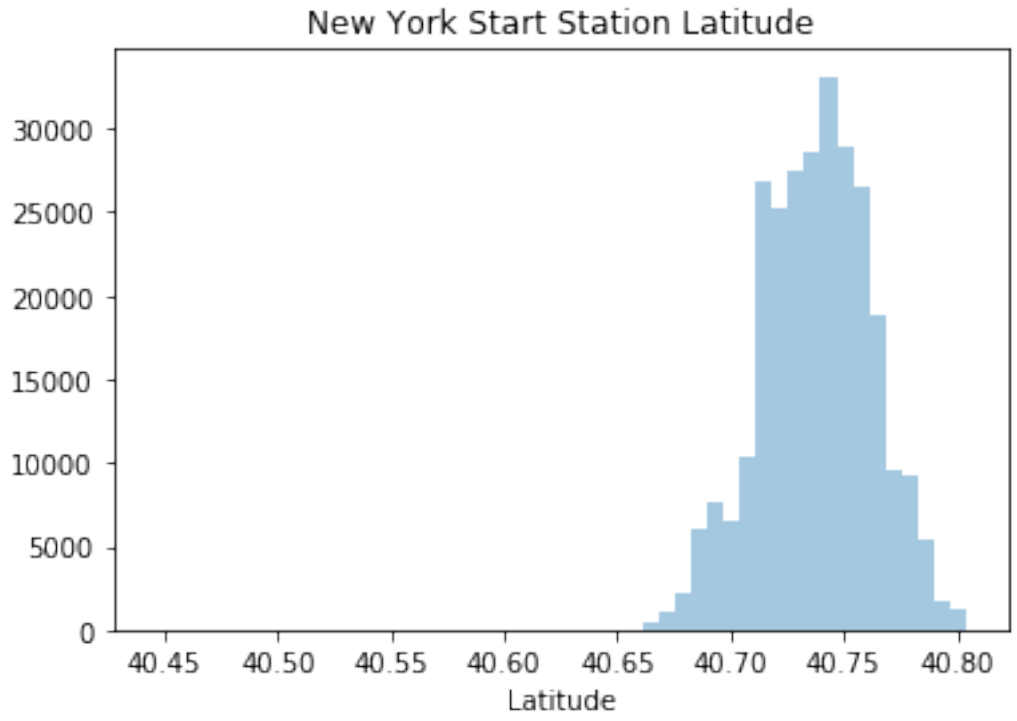
[22]:
```
fig, ax = plt.subplots()
sns.countplot(chicago['starttime'].apply(lambda x: re.findall(r'\s(\d+):',␣
 ↪x)[0]).apply(strip_leading_0), order = [str(i) for i in np.arange(24)],␣
 ↪color = 'darkblue', ax = ax, label = 'start', alpha = 0.5)
sns.countplot(chicago['stoptime'].apply(lambda x: re.findall(r'\s(\d+):',␣
 ↪x)[0]).apply(strip_leading_0), order = [str(i) for i in np.arange(24)],␣
 ↪color = 'red', ax = ax, label = 'stop', alpha = 0.5)
plt.title('Chicago Start and End Times')
plt.legend()
plt.xlabel('Hour');
```



[23]:
```
sns.distplot(ny['start station longitude'], hist = True, kde = False)
plt.title('New York Start Station Longitude')
plt.xlabel('Longitude');
```



[24]:
```
sns.distplot(ny['start station latitude'], hist = True, kde = False)
plt.title('New York Start Station Latitude')
plt.xlabel('Latitude');
```



2. Discuss any insight you obtained from these three new visualizations.

From these visualizations some of the insight I obtained was that the majority of riders in New York are subscribers with very few casual riders or customers compared to both Washington DC and Chicago. Washington DC and Chicago have a much larger proportion of riders that are not subscribed to the services. I also decided to visualize the end times of trips. They looked very similar to the start times as I expected, but I overlayed the end time graphs with the start time graph and this solidfied the idea I had of them being extremely similar which also makes logical sense since there should be some sort of relationship between start and end timres. The last thing I decided to visualize was the latitude and longitude of the stations in New York. The longitude had much more variance than latitude did. I'm not quite sure if this is correct, but this seems to infer that stations vary more in their distance from east to west than they do from north to south.

3. Pick one of the three attributes and plot its distribution if you haven't already. Explore the data further to get a plausible explanation for the shape of the distribution. For example you could explore whether the attribute is correlated with another attribute.

End hour times is an interesting attirbute to look at. As I have explained in the previous section the distribution is very similar to start hour times, so it peaks around 8 am and 5pm as these are popular times for work start times and end times. This attribute is probably fairly heavily correlated with start hour times or just start times in general since a ride has to start before it ends, so this would explain their very similar distributions.

4. Given the insights you obtained from the data, write down a hypothesis that you think is important and how you would go about testing it.

Given what I have observed from the data I think that Chicago and Washington DC distributions for start hour times are the same and the slight differences between them is just due to some sort of randomness in sampling. The cities would be on a very similar time schedules as a result and in order to test this I think running an A/B test could help us reach a conclusion. The test statistic in this case would be the absolute difference between the average of start hour times in Washington DC and the average of start hour times in Chicago. In order to simulate the null hypothesis, I would use a permutation test because if there is the distributions are the same than randomly shuffling the labels should produce a similar test statistic. After simulating this null situation about 1000 times and collecting all of the simulated test statistics I would calculate a p-value for our observed test statistic and allow for a 5% cutoff in order to reject the null hypothesis.

1.4 Optional Creating a new dataset

0.0.2 2 Hypothesis Testing

1.

[43]:
```
chicago_data = chicago[['tripduration', 'starttime', 'stoptime', 'usertype']]
chicago_data['starttime'] = chicago_data['starttime'].apply(lambda x: re.
 ↪findall(r'\s(\d+):', x)[0]).apply(strip_leading_0).apply(int)
chicago_data['stoptime'] = chicago_data['stoptime'].apply(lambda x: re.
 ↪findall(r'\s(\d+):', x)[0]).apply(strip_leading_0).apply(int)
shuffled_index_chicago = np.random.permutation(np.arange(chicago.shape[0]))
chicago_data = chicago_data.reindex(shuffled_index_chicago)
```

```
/srv/conda/envs/data102/lib/python3.7/site-packages/ipykernel_launcher.py:2:
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

/srv/conda/envs/data102/lib/python3.7/site-packages/ipykernel_launcher.py:3:
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
  This is separate from the ipykernel package so we can avoid doing imports
until
```

[44]:
```
sixty_ch = int(np.round(chicago_data.shape[0] * .6))
eighty_ch = int(np.round(chicago_data.shape[0] * .8))
end_ch = int(np.round(chicago_data.shape[0]))
```

[45]:
```
s_1_ch = chicago_data.iloc[np.arange(sixty_ch)]
s_2_ch = chicago_data.iloc[np.arange(sixty_ch, eighty_ch)]
s_3_ch = chicago_data.iloc[np.arange(eighty_ch, end_ch)]
```

[46]:
```
features = ['tripduration', 'starttime', 'stoptime']
target = 'usertype'
```

[47]:
```
from sklearn.linear_model import LogisticRegression
import math
```

[48]:
```
train = LogisticRegression(fit_intercept = False, solver = 'liblinear')
train.fit(s_1_ch[features].values, s_1_ch[target])
```

[48]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=False,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='liblinear', tol=0.0001, verbose=0,
                   warm_start=False)
```

[49]:
```
theta_star = train.coef_
theta_star
```

[49]:
```
array([[-0.00098121,  0.08706371,  0.05368791]])
```

2.

[50]:
```
def logistic_func(xi, theta):
    xi_vec = np.ndarray(shape = (1, len(xi)))
    xi_vec[0] = np.array(xi)
    exp = (theta @ xi_vec.T)[0][0]
    return 1 / (1 + math.exp(-1 * exp))
```

[51]:
```
s_2_ch_probs = [logistic_func(s_2_ch[features].iloc[i], theta_star) for i in␣
 ↪range(s_2_ch.shape[0])]
s_3_ch_probs = [logistic_func(s_3_ch[features].iloc[j], theta_star) for j in␣
 ↪range(s_3_ch.shape[0])]
```

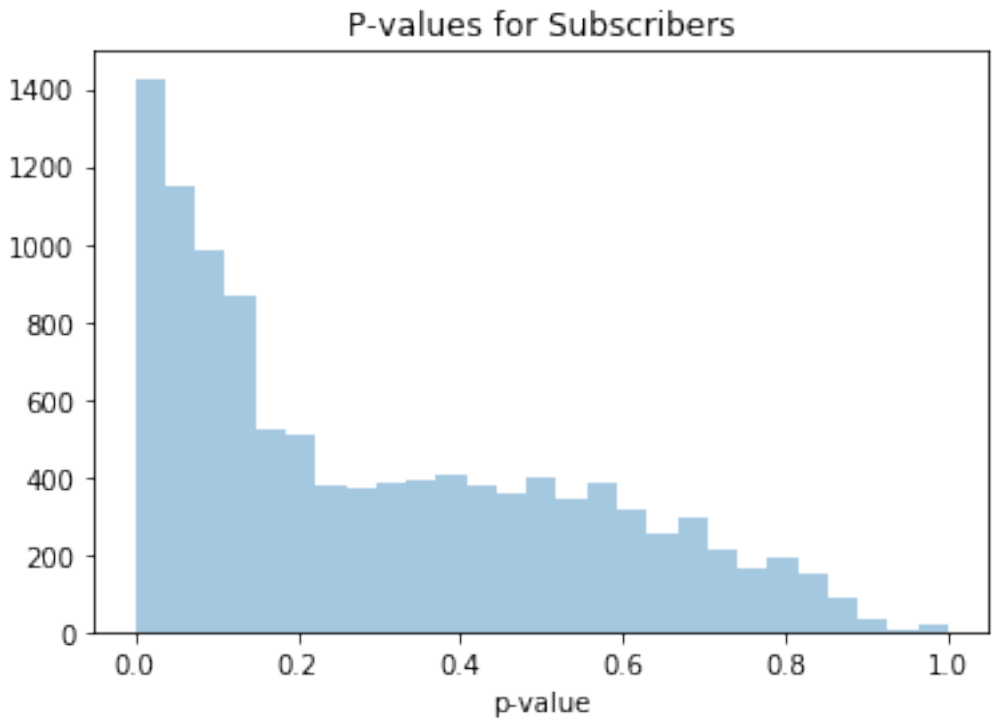
[52]:
```
s_2_0_ch = s_2_ch[s_2_ch['usertype'] == 'Customer']
s_2_0_ch_cardinality = s_2_0_ch.shape[0]
s_2_0_ch_probs = np.array([logistic_func(s_2_0_ch[features].iloc[k],␣
 ↪theta_star) for k in range(s_2_0_ch.shape[0])])
```

[53]:
```
s_3_ch_p_values = []
for s3 in range(len(s_3_ch_probs)):
    s_3_ch_p_values.append((1 / s_2_0_ch_cardinality) * np.sum(s_2_0_ch_probs >␣
 ↪s_3_ch_probs[s3]))
```

[54]:
```
s_3_ch_p_values_df = pd.DataFrame([s_3_ch_p_values, s_3_ch[target]]).T.
 ↪rename({0:'p-value', 1:'usertype'}, axis = 1)
s_3_ch_p_value_subscriber = s_3_ch_p_values_df[s_3_ch_p_values_df['usertype']␣
 ↪== 'Subscriber']
s_3_ch_p_value_customer = s_3_ch_p_values_df[s_3_ch_p_values_df['usertype'] ==␣
 ↪'Customer']
```

[55]:
```
sns.distplot(s_3_ch_p_value_customer['p-value'], kde = False)
plt.title('P-values for Customers');
```

[56]:
```
sns.distplot(s_3_ch_p_value_subscriber['p-value'], kde = False)
plt.title('P-values for Subscribers');
```



For casual riders, the distribution of p-values is fairly uniform and then for registered riders the distribution has a right tail with the majority of values being lower.

3.

[57]:
```
def benjamini_hochberg(p_values, alpha):
    sorted_p = np.sort(p_values)
    significance = np.max(sorted_p[(sorted_p <= (np.arange(len(p_values)) + 1)␣
 ↪* (alpha / len(p_values)))])
    decisions = np.array([int(d <= significance) for d in p_values])
    return decisions
```

[58]:
```
decisions = benjamini_hochberg(s_3_ch_p_values, 0.2)
```

[59]:
```
def fdp_and_sensitivity(decisions, target, disc, not_disc):
    num_true_disc = np.sum((decisions == 1) & (target == disc))
    num_not_true_disc = np.sum((decisions == 1) & (target == not_disc))
    total_found = np.sum(decisions == 1)
    total_disc = np.sum(target == disc)
    return [num_not_true_disc / total_found, num_true_disc / total_disc]
```

[60]:
```
fdp_and_sensitivity(decisions, s_3_ch[target], 'Subscriber', 'Customer')
```

[60]:
```
[0.12121212121212122, 0.007930720145852324]
```

[61]:
```
20 % 10
```

[61]:
```
0
```

[62]:
```
np.random.permutation(np.arange(5))
```

[62]:
```
array([1, 4, 3, 2, 0])
```

[63]:
```
s_3_p_values_200 = []
s_3_fdp_200 = []
s_3_sens_200 = []
for i in np.arange(200):
    shuffled_index_chicago_i = np.random.permutation(np.arange(chicago.
 ↪shape[0]))
    chicago_data_i = chicago_data.reindex(shuffled_index_chicago_i)
    s_1_i = chicago_data_i.iloc[np.arange(sixty_ch)]
    s_2_i = chicago_data_i.iloc[np.arange(sixty_ch, eighty_ch)]
    s_3_i = chicago_data_i.iloc[np.arange(eighty_ch, end_ch)]
    train_i = LogisticRegression(fit_intercept = False, solver = 'liblinear')
    train_i.fit(s_1_i[features].values, s_1_i[target])
    theta_star_i = train_i.coef_
    s_3_i_probs = [logistic_func(s_3_i[features].iloc[j], theta_star_i) for j␣
 ↪in range(s_3_i.shape[0])]
    s_2_0_i = s_2_ch[s_2_ch['usertype'] == 'Customer']
    s_2_0_i_cardinality = s_2_0_i.shape[0]
    s_2_0_i_probs = np.array([logistic_func(s_2_0_i[features].iloc[k],␣
 ↪theta_star_i) for k in range(s_2_0_i.shape[0])])
    s_3_i_p_values = []
    for s3 in range(len(s_3_i_probs)):
        s_3_i_p_values.append((1 / s_2_0_i_cardinality) * np.sum(s_2_0_i_probs␣
 ↪> s_3_i_probs[s3]))
    decisions_i = benjamini_hochberg(s_3_i_p_values, 0.2)
    calc = fdp_and_sensitivity(decisions_i, s_3_i[target], 'Subscriber',␣
 ↪'Customer')
    s_3_p_values_200.append(s_3_i_p_values)
    s_3_fdp_200.append(calc[0])
    s_3_sens_200.append(calc[1])
    if i % 10 == 0:
        print(i)
```

```
0
10
20
30
40
50
60
70
80
90
100
110
```

```
120
130
140
150
160
170
180
190
```

[64]:
```
# average FDP
np.average(s_3_fdp_200)
```

[64]:
```
0.0819529579278128
```

[65]:
```
# average sensitivity
np.average(s_3_sens_200)
```

[65]:
```
0.006958765206649672
```

The average FDP is below 0.2, because the Benjamin Hochberg algorithm guarantees that the approximate fdr under the alpha value provided is met across all of the given n number of tests. Bonferroni is a very conservative bound to control fdr with using the bound of alpha / n, but if we were to choose the least conservative bound, it works out that that bound would be the one that we find using the Benjamin Hochberg algorithm. To go into more depth, the distribution of p-values for the null is uniform so the probability of a false positive (or a false discovery in this case) would be p(k), so the overall bound that we want which is probability of false positive / probability of positives to be less than or equal to alpha. Leaving would then be p(k) / (k / n) to be less than or equal to alpha and isolating p(k) to one side we get the bound p(k) <= alpha * k / n.

2.1 Optional Improving the Average Sensitivity

0.0.3 3 Gaussian Mixture Models of Trip Durations

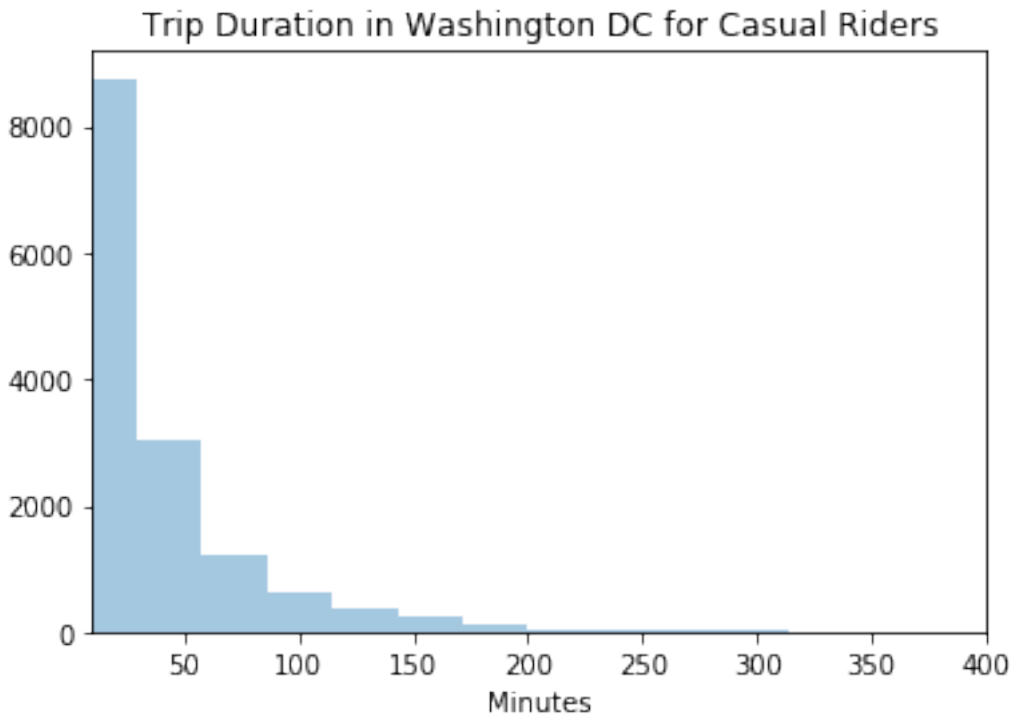
1. Describe why it is reasonable to believe that should a difference in distributions of subscribers and non-subscribed customers. Use figures from the data to back up your argument.

It is reasonable to believe that there should be a difference in the distributions of trup durations of subscribers and non-subscribed customers because subscribed members probably use the bike sharing service as a way to commute to work or something similar while casual members would probably be using the bikes for more leisurely activities. I would expect casual riders to have some longer rides than subscribed members because they could be biking around the city and stuff rather than just using it purely as transportation. In the plots below, this guess is solidified as the distribution of trip durations for casual riders has a right tail and most subscriber trip durations are very low.

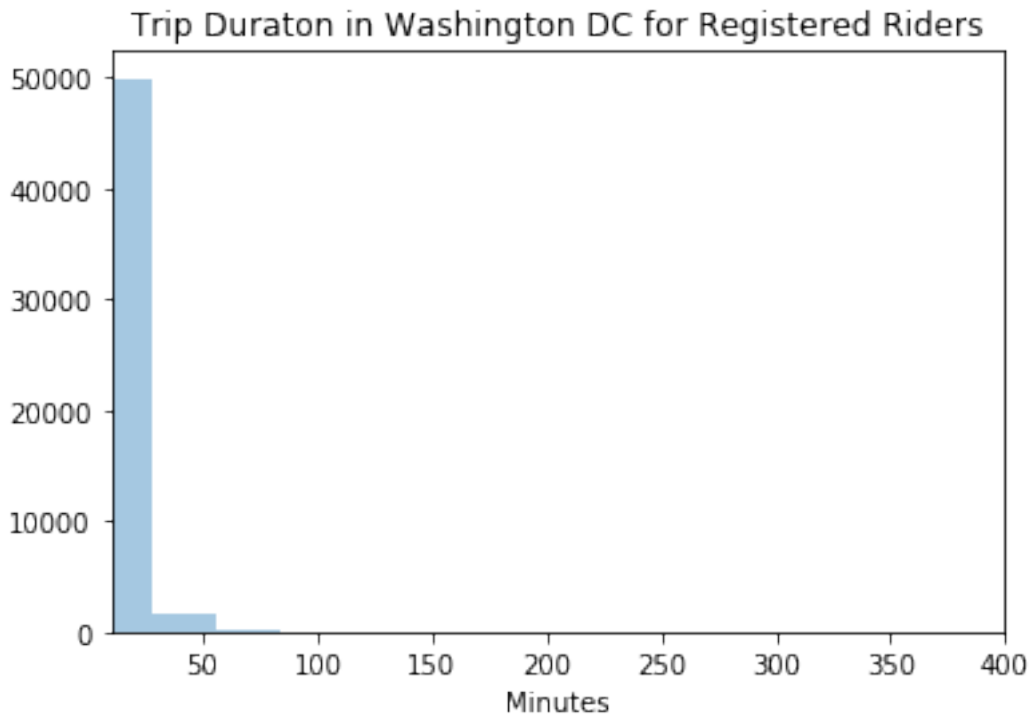
[3]:
```
dc_data_casual = dc[dc['Member Type'] == 'Casual']
dc_data_registered = dc[dc['Member Type'] == 'Registered']
```

[4]:
```
sns.distplot(dc_data_casual['Duration (ms)'] / 60000, kde = False)
plt.xlim((10, 400))
```

```
plt.xlabel('Minutes')
plt.title('Trip Duration in Washington DC for Casual Riders');
```



[5]:
```
sns.distplot(dc_data_registered['Duration (ms)'] / 60000, kde = False)
plt.xlim((10, 400))
plt.xlabel('Minutes')
plt.title('Trip Duraton in Washington DC for Registered Riders');
```



2. Use the Expectation-Maximization (E-M) Algorithm to learn a mixture of two Gaussians that describes the distribution trip-durations of length **less than one hour** in the chicago.csv dataset. Run this multiple times from different initializations. Do your results change drastically depending on the initialization? What are the means and variances of the fitted Gaussians?

[5]:
```
ch_one_hour = chicago[(chicago['tripduration'] / 60) < 60][['tripduration',␣
↪'starttime', 'stoptime', 'usertype']]
```

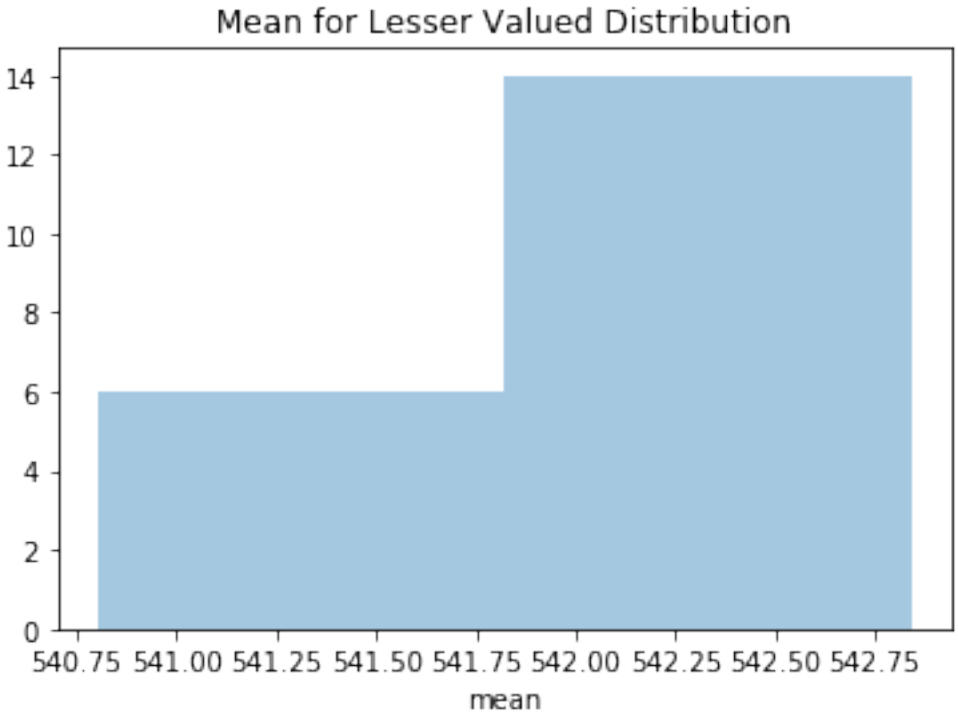
[6]:
```
ch_one_hour_td = ch_one_hour[['tripduration']]
```

[7]:
```
from sklearn.mixture import GaussianMixture
```

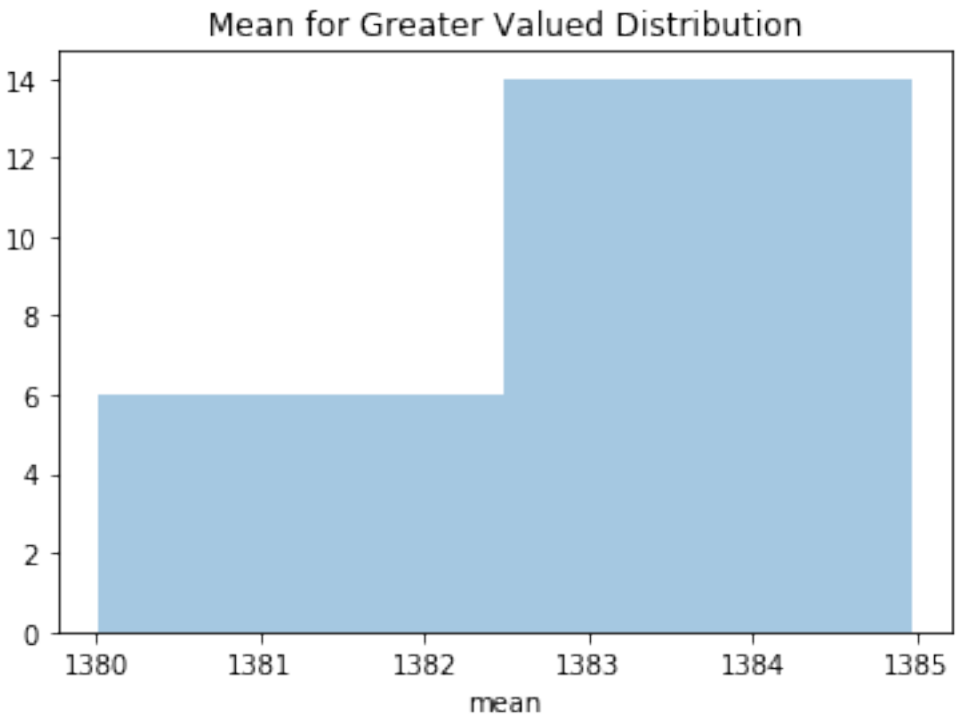
[8]:
```
means_1 = []
means_2 = []
var_1 = []
var_2 = []
for i in np.arange(20):
    mix = GaussianMixture(n_components = 2)
    mix.fit(ch_one_hour_td)
    means = mix.means_
    var = mix.covariances_
    mean_1 = means[0][0]
    mean_2 = means[1][0]
    v_1 = var[0][0][0]
```

```
    v_2 = var[1][0][0]
    if mean_1 < mean_2:
        means_1.append(mean_1)
        means_2.append(mean_2)
        var_1.append(v_1)
        var_2.append(v_2)
    else:
        means_1.append(mean_2)
        means_2.append(mean_1)
        var_1.append(v_2)
        var_2.append(v_1)
```

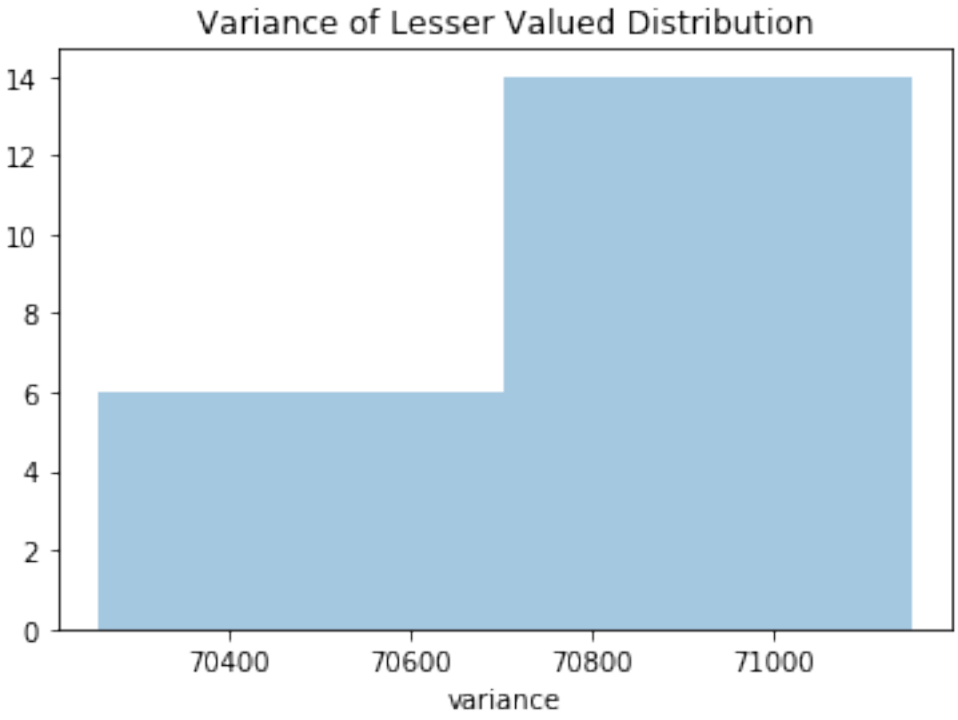
[9]:
```
sns.distplot(means_1, kde = False)
plt.title('Mean for Lesser Valued Distribution')
plt.xlabel('mean');
```



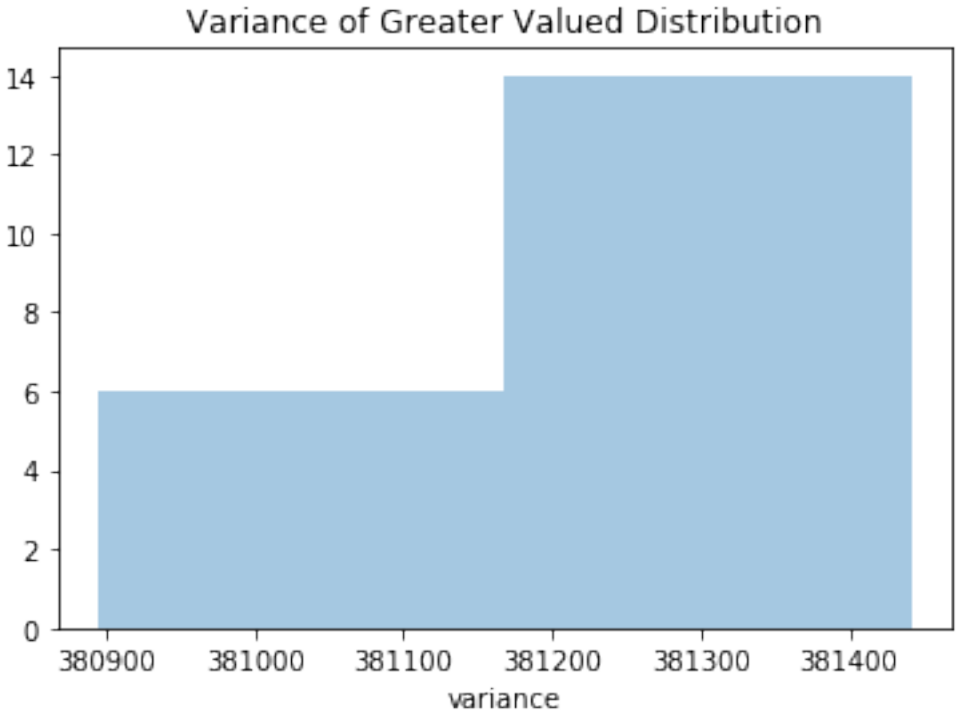
[10]:
```
sns.distplot(means_2, kde = False)
plt.title('Mean for Greater Valued Distribution')
plt.xlabel('mean');
```



[11]:
```
sns.distplot(var_1, kde = False)
plt.title('Variance of Lesser Valued Distribution')
plt.xlabel('variance');
```



[12]:
```
sns.distplot(var_2, kde = False)
plt.title('Variance of Greater Valued Distribution')
plt.xlabel('variance');
```



After trying a couple of different initializations, the results of the EM algorithm for finding the mean and variances of the two mixtures do not change drastically as shown in the plots above. The mean of one of the mixtures is around 542 and the mean of the other is around 1383 while the variance of one is around 70700 and of the other mixture's variance is around 381100. These numbers imply that there is distribtion that on average has lower values for trip duration with also a smaller variance as compared to an averagely greater valued distribution with a mean of around 1383 seconds and a much higher variance of around 381100 seconds as compared to a variance of around 70700 of the other distribution. So one of the distributions I'm assuming the subscriber distribution is less on average with a smaller variance since there might not be a huge difference in times for people who use the bikes as transportation as compared to the other distribution that the algorithm found with higher mean and variance which seems characteristic of casual riders that might use the bikes for various uses and for usually more time.

3. Given the output of the E-M Algorithm, which of the distributions captures the behavior of the subscribed customers? For each customer, in the dataset, calculate the posterior probability that the customer is from this distribution

Given the output of the E-M algorithm, I think the lesser valued distribution captures the behavior of the subscribed customers while the greater valued distribution captures casual customers.

[13]:
```
# first element or group is what we expect to be the casual group
mix.means_
```

[13]:
```
array([[1384.39735291],
       [ 542.60831978]])
```

[14]:
```
posterior_probs = mix.predict_proba(ch_one_hour_td)
```

[15]:
```
posterior_probs_casual = [probs[0] for probs in posterior_probs]
posterior_probs_subscriber = [probs[1] for probs in posterior_probs]
```

4. If you design a classifier which classifies a customer as a Subscriber if their posterior probability is greater than 0.5, what is the error of this classifier given the true User Types in the chicago.csv dataset?

[16]:
```
subscriber_predictions = np.array(posterior_probs_subscriber) > 0.5
subscribers_true = np.array(ch_one_hour['usertype'] == 'Subscriber')
```

[17]:
```
check = subscriber_predictions == subscribers_true
```

[18]:
```
# error of the classifier on the true user types in chicago.csv
np.sum(check) / len(check)
```

[18]:
```
0.772890764394347
```

5. Use the classifier derived from the chicago.csv dataset on the ny.csv and dc.csv datasets. How does this classifier perform? How does the performance compare to that in the previous part? (Make sure that the data for each city is in comparable units.)

[19]:
```
ny_one_hour = ny[(ny['tripduration'] / 60) < 60][['tripduration', 'starttime',␣
 ↪'stoptime', 'usertype']]
ny_one_hour_td = ny_one_hour[['tripduration']]
dc_one_hour = dc[(dc['Duration (ms)'] / 60000) < 60][['Duration (ms)', 'Start␣
 ↪date', 'End date', 'Member Type']]
dc_one_hour['tripduration'] = dc['Duration (ms)'].apply(lambda x: x / 1000)
dc_one_hour_td = dc_one_hour[['tripduration']]
```

[30]:
```
posterior_probs_ny = mix.predict_proba(ny_one_hour_td)
posterior_probs_dc = mix.predict_proba(dc_one_hour_td)
posterior_probs_ny_subscriber = [probs[1] for probs in posterior_probs_ny]
posterior_probs_dc_subscriber = [probs[1] for probs in posterior_probs_dc]
subscribers_ny_predictions = np.array(posterior_probs_ny_subscriber) > 0.5
subscribers_ny_true = np.array(ny_one_hour['usertype'] == 'Subscriber')
subscribers_dc_predictions = np.array(posterior_probs_dc_subscriber) > 0.5
subscribers_dc_true = np.array(dc_one_hour['Member Type'] == 'Registered')
check_ny = subscribers_ny_predictions == subscribers_ny_true
check_dc = subscribers_dc_predictions == subscribers_dc_true
```

[31]:
```
# error on the classifier on user types in ny.csv with points with trip␣
 ↪durations under 60 minutes
np.sum(check_ny) / len(check_ny)
```

[31]:
```
0.7649095539356261
```

[32]:
```
# error on the classifier on user types in dc.csv with points with trip␣
 ↪durations under 60 minutes
np.sum(check_dc) / len(check_dc)
```

[32]:
```
0.7760664627802125
```

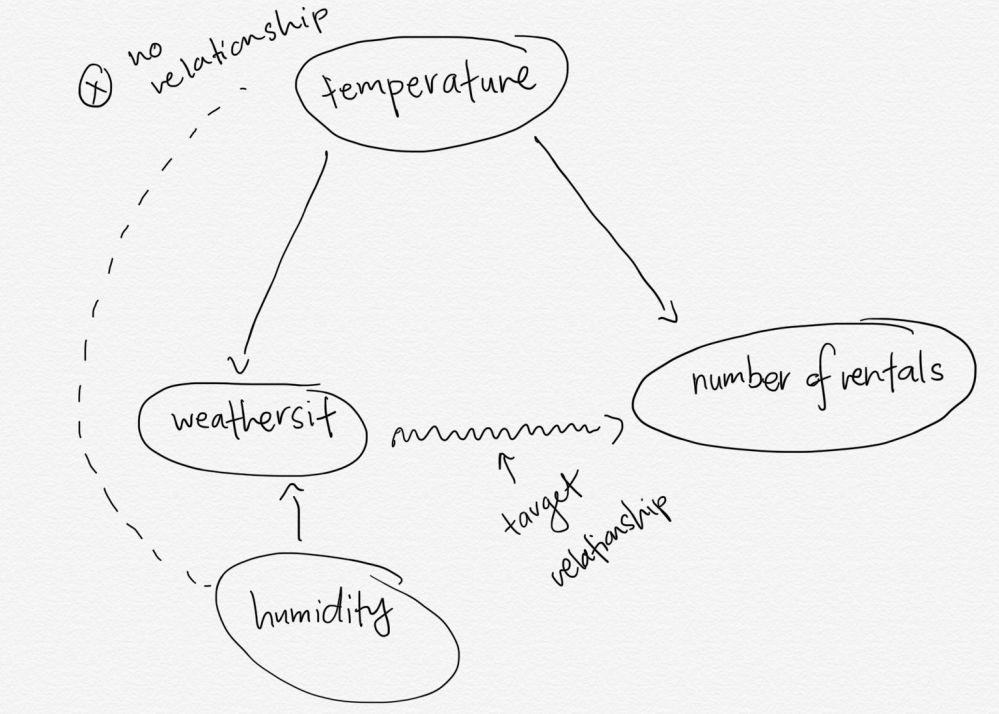
The performance of the classifier on the New York and Washington DC is actually very similar to the performance of the classifier on the data that it was trained on from Chicago. If we look back to part 1, we can see that from minute 0 to about 60, the distribution of trip durations for each city are fairly similar and although the proportion of casual riders to subscribers differed slightly from each city, there was still a much larger proportion of subscribers for all cities. The classifier did fairly well or just about as well as it performed on the training data because the data we used to test accuracy on were fairly similar to the data it was trained on.

3.1 Optional Learning Mixtures of Bivariate Gaussians

0.0.4 4 Causality and Experiment Design

4.1 2SLS to estimate the effect of precipitation on #bike rentals 4.1.1. The causal model

1. Draw the graph of the causal model, with arrows between variables to denote causality. Your graph should include the following variables: temperature, weathersit, humidity, and number of rentals.



title

2. Clearly describe the assumptions necessary for 2SLS analysis. For any assumptions you can check with the data set, do so. For the remainder of the assumptions, give your best guess as to whether they hold given the problem setting, and discuss how you would test them (this could mean collecting more data.)

Assumptions are that: 1. Temperature is correlated with weathersit 2. Tempearature is correlated with the number of rentals 3. Humidity is correlated with weathersit 4. Humidity is not correlated with temperature 5. Humidity is not correlated with number of rentals

[4]:
```
from scipy.stats import pearsonr
```

[9]:
```
# correlation between temperature and weathersit
pearsonr(day['temp'], day['weathersit'])[0]
```

[9]:
```
-0.120602236499715
```

[12]:
```
# correlation between temperature and number of rentals
pearsonr(day['temp'], day['cnt'])
```

[12]:
```
(0.627494009033492, 2.8106223975907754e-81)
```

[13]:
```
# correlation between humidity and weathersit
pearsonr(day['hum'], day['weathersit'])
```

[13]:
```
(0.5910445992972727, 4.643103930673548e-70)
```

[15]:
```
# correlation between humdity and temperature
pearsonr(day['hum'], day['temp'])[0]
```

[15]:
```
0.1269629390271887
```

[17]:
```
# correlation between humidity and number of rentals
pearsonr(day['hum'], day['cnt'])[0]
```

[17]:
```
-0.10065856213715527
```

[18]:
```
day.shape[0]
```

[18]:
```
731
```

Some of the assumptions necessary for 2SLS analysis I guess generally would be that the data is representative of a good amount of days with varying weather with bike rental counts, humidity is randomized which probably hold since the data has over 2 years worth of data that probably has enough variation in humidity.

4.1.2. 2 Stage Least Squares

1. Describe the 2SLS procedure, as it applies to the variables above. Describe the procedure at a level such that someone who has taken DS100, but not DS102, would be able to reproduce the analysis (without looking at your code).

To apply the 2SLS procedure the the variables above we will first regress weathersit onto temperature to get a model that predicts weathersit from humidity. Use this model to find predictions for the weathersit values of the data using temperature and store those values. We will now regress number of bike rentals a day onto these predicted values and the coefficient of the predicted values of weathersit should be able to tell us the impact on number of bike rentals by the weather.

2. Report the resulting treatment affect of weathersit on the total number of bike rentals. Interpret this treatment effect estimate in terms of the variables of this problem.

[19]:
```
from sklearn.linear_model import LinearRegression
```

[51]:
```
lm_mod_hum_weather = LinearRegression()
lm_mod_pred_weather_rentals = LinearRegression()
```

[52]:
```
humidity_data = day[['hum']].values
weathersit_data = day['weathersit']
num_rentals = day['cnt']
```

[53]:
```
lm_mod_hum_weather.fit(humidity_data, weathersit_data)
predicted_weathersit = pd.DataFrame(lm_mod_hum_weather.predict(humidity_data),␣
↪columns = ['pred weathersit']).values
```

[54]:
```
lm_mod_pred_weather_rentals.fit(predicted_weathersit, num_rentals)
lm_mod_pred_weather_rentals.coef_
```

[54]:
```
array([-605.47358154])
```

The coefficient that we have for our *predicted* weathersit is -605, this means that there is a negative relationship between the variable weathersit and the number of bike rentals that they are. Approximately an increase of the magnitude of 1 in the variable weathersit decreases the number of rentals of the day by about 605.

3. Report the resulting treatment affect of weathersit on the number of causal bike rentals, and on the number of registered bike rentals.

[56]:
```
num_rentals_casual = day['casual']
num_rentals_registered = day['registered']
lm_mod_pred_weather_casual = LinearRegression()
lm_mod_pred_weather_casual.fit(predicted_weathersit, num_rentals_casual)
lm_mod_pred_weather_registered = LinearRegression()
lm_mod_pred_weather_registered.fit(predicted_weathersit, num_rentals_registered)
```

[56]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

[57]:
```
lm_mod_pred_weather_casual.coef_
```

[57]:
```
array([-164.18015256])
```

The coefficient for our relationship between weathersit and causal number of rentals is still negative, but with a smaller magnitude of only about 164.

[60]:
```
lm_mod_pred_weather_registered.coef_
```

[60]:
```
array([-441.29342898])
```

Like the coefficient of our relationships between weathersit and casual number of rentals, the coefficient is negative with a larger magnitude of about 441.

4.1.3. Discussion

1. Give a three sentence summary of the question, how and why we tested it, and the results of the analysis.

The question that we are trying to answer is what is the relationship between the weather and the number of bike rentals or more generally does weather impact the number of rentals? We could not directly regress the number of rentals on weathersit because temperature may be a confounding factor that impacts both weather and the number of rentals. In order to remedy this we made the assumption that there is no relationship between humidity and temperature as well as no relationship between humidity and the number of rental days in order to run 2SLS and we came out with the results of a negative relationship with approximately each increase value of 1 of weathersit, the number of rentals decreases by about 605.

2. Interpret the treatment effect estimates that resulted for the number of causal rentals and for the number of registered rentals. Is the magnitude of the treatment effect higher for one group than another? Give at least one possible reason for the difference/similarity you find.

The treatment effect estimates that resulted for the number of causal rentals and the number of registered rentals are different. The magnitude of the effect on registered rentals is larger than casual renters which means that the bad weather causes a larger reduction in the number of rentals for registered riders than casual riders. I think something to take into account though is that there are more registered riders than casual riders, so although the number of rentals decreases more for registered riders than casual riders. I would assume that the difference in percentages may be different because casual riders probably won't use the service when it's rainy or cold, but registered riders who use it as a commute option may still more often.

3. Discuss the applicability of the chosen model (causal graph from above) to this problem. Are there any variables that might be missing form the causal graph? Are there any arrows that are missing?

I think the chosen model is relatively good. I can definitely agree with temperature impacting both the weather and the number of rentals. We sought to remedy this with 2SLS by using humidity as an instrumental variable. In the data it showed that humidity and temperature were not highly correlated, but I'm not really sure if that is the case because logically I would believe that hotter days typically have more humidity. Another thing that might be questioned is that there is no relationship between humidity and the number of rentals. I would say that at least for me I think higher humidity would decrease the chance of me using a bike service since the air is already really moist and extra physical activity would make it worse for me. I also think there are probably a lot more other factors that are probably confounding with weather and the number of rentals such as the season? Rentals are probably higher in the summer since students are off of school and such.

4. If you had the oppportunity to design your own study (and collect new data) to test the effect of adverse weather on the number of rentals, what would that study look like? Explain at a high level the design decisions you would make and why (keep your repsonse to no more than 5 sentences).

If I had the opportunity to design my own study and collect new data in order to test the effect of adverse weather on the number of rentals, I think that I would probably decide on collecting other variables as well such and the season since I think that that factor could influence both the weather as well as the number of rentals fairly much. Another thing that I would consider is possibly splitting up the days that are weekends vs weekdays to distinguish a difference in the way that weather may impact more casual riders who are riding to places or exploring on the

weekends vs the impact that weather affects commuters who use the service to get to work. I think the use of the 2SLS is a good idea and something I would continue because there are confounding factors.

4.2 Optional Simulating Experiment Design Approaches via Sub-sampling

[ ]: