exploration_gobike

December 24, 2019

GoBike 2017-19 dataset exploration

by Konstantin Leonenko

1 Preliminary Wrangling

Briefly introduce your dataset here.

[1]:
```
# import all packages and set plots to be embedded inline
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure
import seaborn as sb
import random

%matplotlib inline
```

1.1 Load all the data from .csv files

[2]:
```
gobike_2017 = pd.read_csv('data/gobike/2017-fordgobike-tripdata.csv')
gobike_2018_01 = pd.read_csv('data/gobike/201801-fordgobike-tripdata.csv')
gobike_2018_02 = pd.read_csv('data/gobike/201802-fordgobike-tripdata.csv')
gobike_2018_03 = pd.read_csv('data/gobike/201803-fordgobike-tripdata.csv')
gobike_2018_04 = pd.read_csv('data/gobike/201804-fordgobike-tripdata.csv')
gobike_2018_05 = pd.read_csv('data/gobike/201805-fordgobike-tripdata.csv')
gobike_2018_06 = pd.read_csv('data/gobike/201806-fordgobike-tripdata.csv')
gobike_2018_07 = pd.read_csv('data/gobike/201807-fordgobike-tripdata.csv')
gobike_2018_08 = pd.read_csv('data/gobike/201808-fordgobike-tripdata.csv')
gobike_2018_09 = pd.read_csv('data/gobike/201809-fordgobike-tripdata.csv')
gobike_2018_10 = pd.read_csv('data/gobike/201810-fordgobike-tripdata.csv')
gobike_2018_11 = pd.read_csv('data/gobike/201811-fordgobike-tripdata.csv')
gobike_2018_12 = pd.read_csv('data/gobike/201812-fordgobike-tripdata.csv')
gobike_2019_01 = pd.read_csv('data/gobike/201901-fordgobike-tripdata.csv')
gobike_2019_02 = pd.read_csv('data/gobike/201902-fordgobike-tripdata.csv')
gobike_2019_03 = pd.read_csv('data/gobike/201903-fordgobike-tripdata.csv')
gobike_2019_04 = pd.read_csv('data/gobike/201904-fordgobike-tripdata.csv')
gobike_2019_05 = pd.read_csv('data/gobike/201905-baywheels-tripdata.csv')
```

```
gobike_2019_06 = pd.read_csv('data/gobike/201906-baywheels-tripdata.csv')
# in the july 2019 dataset values are separated by semicolons rather than commas
gobike_2019_07 = pd.read_csv('data/gobike/201907-baywheels-tripdata.csv',␣
 ↪delimiter = ';')
gobike_2019_08 = pd.read_csv('data/gobike/201908-baywheels-tripdata.csv')
gobike_2019_09 = pd.read_csv('data/gobike/201909-baywheels-tripdata.csv')
gobike_2019_10 = pd.read_csv('data/gobike/201910-baywheels-tripdata.csv')
```

```
/Users/s8/anaconda3/lib/python3.7/site-
packages/IPython/core/interactiveshell.py:3051: DtypeWarning: Columns (16) have
mixed types. Specify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
/Users/s8/anaconda3/lib/python3.7/site-
packages/IPython/core/interactiveshell.py:3051: DtypeWarning: Columns (15,16)
have mixed types. Specify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

1.2 Merge all data into a single dataframe

[3]:
```
df_gobike = pd.DataFrame(data = gobike_2017)
df_gobike = df_gobike.append(gobike_2018_01)
df_gobike = df_gobike.append(gobike_2018_02)
df_gobike = df_gobike.append(gobike_2018_03)
df_gobike = df_gobike.append(gobike_2018_04)
df_gobike = df_gobike.append(gobike_2018_05)
df_gobike = df_gobike.append(gobike_2018_06)
df_gobike = df_gobike.append(gobike_2018_07)
df_gobike = df_gobike.append(gobike_2018_08)
df_gobike = df_gobike.append(gobike_2018_09)
df_gobike = df_gobike.append(gobike_2018_10)
df_gobike = df_gobike.append(gobike_2018_11)
df_gobike = df_gobike.append(gobike_2018_12)
df_gobike = df_gobike.append(gobike_2019_01)
df_gobike = df_gobike.append(gobike_2019_02)
df_gobike = df_gobike.append(gobike_2019_03)
df_gobike = df_gobike.append(gobike_2019_04)
df_gobike = df_gobike.append(gobike_2019_05)
df_gobike = df_gobike.append(gobike_2019_06)
df_gobike = df_gobike.append(gobike_2019_07)
df_gobike = df_gobike.append(gobike_2019_08)
df_gobike = df_gobike.append(gobike_2019_09)
df_gobike = df_gobike.append(gobike_2019_10)
```

```
/Users/s8/anaconda3/lib/python3.7/site-packages/pandas/core/frame.py:7138:
FutureWarning: Sorting because non-concatenation axis is not aligned. A future
version
of pandas will change to not sort by default.
```

```
To accept the future behavior, pass 'sort=False'.

To retain the current behavior and silence the warning, pass 'sort=True'.

  sort=sort,
```

[4]:
```
df_gobike.columns
```

[4]:
```
Index(['bike_id', 'bike_share_for_all_trip', 'duration_sec', 'end_station_id',
       'end_station_latitude', 'end_station_longitude', 'end_station_name',
       'end_time', 'member_birth_year', 'member_gender',
       'rental_access_method', 'start_station_id', 'start_station_latitude',
       'start_station_longitude', 'start_station_name', 'start_time',
       'user_type'],
      dtype='object')
```

[5]:
```
df_gobike.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4554806 entries, 0 to 239894
Data columns (total 17 columns):
bike_id                    int64
bike_share_for_all_trip    object
duration_sec               int64
end_station_id             float64
end_station_latitude       float64
end_station_longitude      float64
end_station_name           object
end_time                   object
member_birth_year          float64
member_gender              object
rental_access_method       object
start_station_id           float64
start_station_latitude     float64
start_station_longitude    float64
start_station_name         object
start_time                 object
user_type                  object
dtypes: float64(7), int64(2), object(8)
memory usage: 625.5+ MB
```

[6]:
```
df_gobike.head()
```

[6]:
```
   bike_id bike_share_for_all_trip  duration_sec  end_station_id  \
0       96                     NaN         80110            43.0
1       88                     NaN         78800            96.0
2     1094                     NaN         45768           245.0
```

```
3     2831                     NaN         62172              5.0
4     3167                     NaN         43603            247.0

   end_station_latitude  end_station_longitude  \
0             37.778768            -122.415929
1             37.766210            -122.426614
2             37.870348            -122.267764
3             37.783899            -122.408445
4             37.867789            -122.265896

                                  end_station_name  \
0  San Francisco Public Library (Grove St at Hyde…
1                             Dolores St at 15th St
2                            Downtown Berkeley BART
3       Powell St BART Station (Market St at 5th St)
4                         Fulton St at Bancroft Way

                   end_time  member_birth_year member_gender  \
0  2018-01-01 15:12:50.2450             1987.0          Male
1  2018-01-01 13:49:55.6170             1965.0        Female
2  2018-01-01 11:28:36.8830                NaN           NaN
3  2018-01-01 10:47:23.5310                NaN           NaN
4  2018-01-01 02:29:57.5710             1997.0        Female

  rental_access_method  start_station_id  start_station_latitude  \
0                  NaN              74.0               37.776435
1                  NaN             284.0               37.784872
2                  NaN             245.0               37.870348
3                  NaN              60.0               37.774520
4                  NaN             239.0               37.868813

   start_station_longitude                               start_station_name  \
0              -122.426244                            Laguna St at Hayes St
1              -122.400876  Yerba Buena Center for the Arts (Howard St at …
2              -122.267764                           Downtown Berkeley BART
3              -122.409449                             8th St at Ringold St
4              -122.258764                     Bancroft Way at Telegraph Ave

                 start_time   user_type
0  2017-12-31 16:57:39.6540    Customer
1  2017-12-31 15:56:34.8420    Customer
2  2017-12-31 22:45:48.4110    Customer
3  2017-12-31 17:31:10.6360    Customer
4  2017-12-31 14:23:14.0010  Subscriber
```

[7]:
```
# let's quickly see how much data is missing from the dataset
for i in df_gobike.columns:
```

```
    print (i, ' : ', len(df_gobike[df_gobike[i].isnull()]))
```

```
bike_id  :  0
bike_share_for_all_trip  :  611447
duration_sec  :  0
end_station_id  :  72354
end_station_latitude  :  0
end_station_longitude  :  0
end_station_name  :  71804
end_time  :  0
member_birth_year  :  432245
member_gender  :  419016
rental_access_method  :  4463059
start_station_id  :  70563
start_station_latitude  :  0
start_station_longitude  :  0
start_station_name  :  69967
start_time  :  0
user_type  :  0
```

mainly it's data about station id's and names. According to the rules of tidy data we should move the information about the stations, such as location, name, etc. into a separate table, and only keep trip start and end station id's in the dataset.

1.3 Extract station geographical information into a separate dataframe

[8]:
```
# let's extract start and end station information
df_start_stations =␣
↪df_gobike[['start_station_id','start_station_latitude','start_station_longitude','start_sta
df_end_stations = ␣
↪df_gobike[['end_station_id','end_station_latitude','end_station_longitude','end_station_nam
# start and end information belongs in the rides table, not in the station␣
↪table.
# let's just merge these
df_start_stations = df_start_stations.rename(columns={'start_station_id':
↪'station_id', 'start_station_latitude':'latitude','start_station_longitude':
↪'longitude','start_station_name':'name'})
df_end_stations = df_end_stations.rename(columns={'end_station_id':
↪'station_id', 'end_station_latitude':'latitude','end_station_longitude':
↪'longitude','end_station_name':'name'})
df_stations = df_start_stations.append(df_end_stations)
```

[9]:
```
df_stations.head()
```

[9]:
```
   station_id   latitude   longitude  \
0        74.0  37.776435 -122.426244
1       284.0  37.784872 -122.400876
```

```
2       245.0  37.870348 -122.267764
3        60.0  37.774520 -122.409449
4       239.0  37.868813 -122.258764

                                                 name
0                                 Laguna St at Hayes St
1  Yerba Buena Center for the Arts (Howard St at …
2                               Downtown Berkeley BART
3                                 8th St at Ringold St
4                        Bancroft Way at Telegraph Ave
```

[10]:
```
df_stations.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9109612 entries, 0 to 239894
Data columns (total 4 columns):
station_id    float64
latitude      float64
longitude     float64
name          object
dtypes: float64(3), object(1)
memory usage: 347.5+ MB
```

[11]:
```
# how many unique station id's are there?
print ('unique station ids: ', df_stations.station_id.nunique())
print ('unique station names: ', df_stations.name.nunique())
print ('total entries: ', len(df_stations))
```

```
unique station ids:  427
unique station names:  459
total entries:  9109612
```

[12]:
```
# make a copy of the stations dataframe before cleaning
df_stations_clean = df_stations.copy()
```

[13]:
```
# save stations with no id into a separate dataframe and remove them from the␣
↪main station dataframe
df_null_stations = df_stations_clean[df_stations_clean.station_id.isnull()]
df_stations_clean.drop(df_null_stations.index, inplace=True)
```

[14]:
```
# check whether the data was removed
len(df_stations_clean[df_stations_clean.station_id.isnull()])
```

[14]: 0

[15]:
```
# cast station id's from floats into ints
df_stations_clean.station_id = df_stations_clean.station_id.astype(int)
```

[16]:
```
# aggregate stations around station_id field
df_stations_clean = df_stations_clean.groupby('station_id', as_index=False).
↪agg({
    'latitude':'mean', 'longitude':'mean', 'name':set})
```

[17]:
```
df_stations_clean.columns
```

[17]:
```
Index(['station_id', 'latitude', 'longitude', 'name'], dtype='object')
```

To get a better understanding of station usage - let's put total amount of ride starts and ends per station into the df_stations dataset.

[18]:
```
df_stations_clean['starts'] = df_gobike.groupby('start_station_id',␣
↪as_index=False).agg( 'count')['bike_id']
df_stations_clean['ends'] = df_gobike.groupby('end_station_id', as_index=False).
↪agg('count')['bike_id']

df_stations_clean['starts'] = df_stations_clean['starts'].fillna(0).
↪astype(float)
df_stations_clean['ends'] = df_stations_clean['ends'].fillna(0).astype(float)
```

[19]:
```
df_stations_clean.head()
```

[19]:
```
   station_id   latitude   longitude  \
0           3  37.786375 -122.404904
1           4  37.785881 -122.408915
2           5  37.783899 -122.408445
3           6  37.804770 -122.403234
4           7  37.804562 -122.271738

                                         name   starts     ends
0  {Powell St BART Station (Market St at 4th St)}  72173.0  76349.0
1                     {Cyril Magnin St at Ellis St}  14926.0  14967.0
2  {Powell St BART Station (Market St at 5th St)}  59997.0  62238.0
3                   {The Embarcadero at Sansome St}  72832.0  85649.0
4                             {Frank H Ogawa Plaza}  20518.0  20278.0
```

1.4 Clean the rides dataframe

Now that we have station info stored in an efficient dictionary - let's remove station specific data from the main dataset

[20]:
```
# before cleaning - make a copy of the dataset
df_rides = df_gobike.copy()
```

Now that all station-specific information is stored separately - we don't need it in out dataframe.

[21]:
```
drop_columns = ['start_station_name', 'end_station_name',
                 'start_station_latitude', 'end_station_latitude',
                'start_station_longitude', 'end_station_longitude']
df_rides.drop(drop_columns, axis=1, inplace=True)
```

[22]:
```
df_rides.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4554806 entries, 0 to 239894
Data columns (total 11 columns):
bike_id                    int64
bike_share_for_all_trip    object
duration_sec               int64
end_station_id             float64
end_time                   object
member_birth_year          float64
member_gender              object
rental_access_method       object
start_station_id           float64
start_time                 object
user_type                  object
dtypes: float64(3), int64(2), object(6)
memory usage: 417.0+ MB
```

[23]:
```
# how many trips don't have either start or end station id's?
null_start_station = df_rides[df_rides.start_station_id.isnull()]
null_end_station = df_rides[df_rides.end_station_id.isnull()]

print ('no start station: ', len(null_start_station))
print ('no end station: ', len(null_end_station))
```

```
no start station:  70563
no end station:  72354
```

[24]:
```
drop_stations = null_start_station.merge(null_end_station, how='outer')
len(drop_stations)
```

[24]: 85954

[25]:
```
# for the overview purpose of our analysis these records are not worth repairing
df_rides.dropna(subset=['start_station_id', 'end_station_id'],inplace=True)
```

[26]:
```
df_rides.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4468852 entries, 0 to 239894
Data columns (total 11 columns):
bike_id                    int64
```

```
bike_share_for_all_trip    object
duration_sec               int64
end_station_id             float64
end_time                   object
member_birth_year          float64
member_gender              object
rental_access_method       object
start_station_id           float64
start_time                 object
user_type                  object
dtypes: float64(3), int64(2), object(6)
memory usage: 409.1+ MB
```

Station ID's in the dataframe are floats, whereas they're better represented as ints.

[27]:
```
df_rides.start_station_id = df_rides.start_station_id.astype(int)
df_rides.end_station_id = df_rides.end_station_id.astype(int)
```

Convert time columns into proper datetime format

[28]:
```
df_rides.start_time = pd.to_datetime(df_rides.start_time)
df_rides.end_time = pd.to_datetime(df_rides.end_time)
```

Personal / social information in the dataset. Information regarding gender, age and customer type contained in the dataset is very limited and can lead to drawing very biased conclusions about social tendencies of users. Therefore, for the purposes of this analysis I would like to remove this data to avoid jumping to potentially very divisive demographic observations. In this analysis I would L to focus entirely on the patterns of transportation rather than any social or human factors, like access method or bike share. Therefore I will remove irrelevat columns from the dataset.

[29]:
```
df_rides.drop (['member_gender', 'member_birth_year', 'user_type',
                'rental_access_method','bike_share_for_all_trip'], axis=1,␣
 ↪inplace=True)
```

[30]:
```
df_rides.head()
```

[30]:
```
   bike_id  duration_sec  end_station_id                end_time  \
0       96         80110              43 2018-01-01 15:12:50.245
1       88         78800              96 2018-01-01 13:49:55.617
2     1094         45768             245 2018-01-01 11:28:36.883
3     2831         62172               5 2018-01-01 10:47:23.531
4     3167         43603             247 2018-01-01 02:29:57.571

   start_station_id              start_time
0                74 2017-12-31 16:57:39.654
1               284 2017-12-31 15:56:34.842
2               245 2017-12-31 22:45:48.411
3                60 2017-12-31 17:31:10.636
```

```
4                239 2017-12-31 14:23:14.001
```

1.4.1 What is the structure of your dataset?

Dataset contains information about individual bicycle rides together with station geographic information.

1.4.2 What is/are the main feature(s) of interest in your dataset?

I'm interested in seeing bike and station usage patterns over times of the day and days of the week.

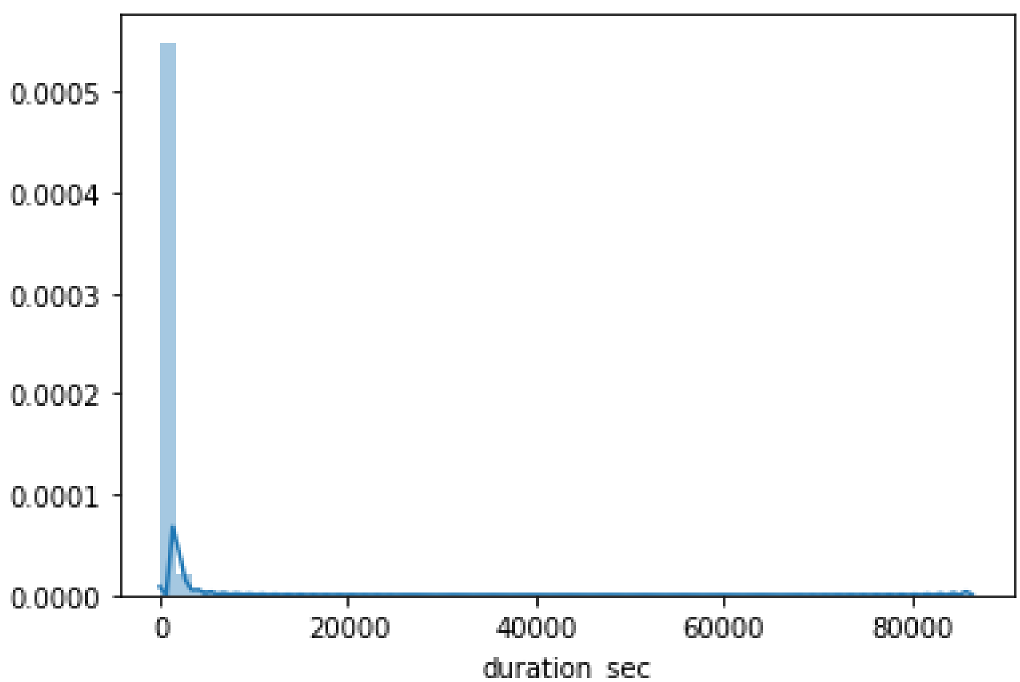
1.4.3 What features in the dataset do you think will help support your investigation into your feature(s) of interest?

Most important fields for my analysis are: - start station - end station - start time - end time - ride duration

1.5 Univariate Exploration

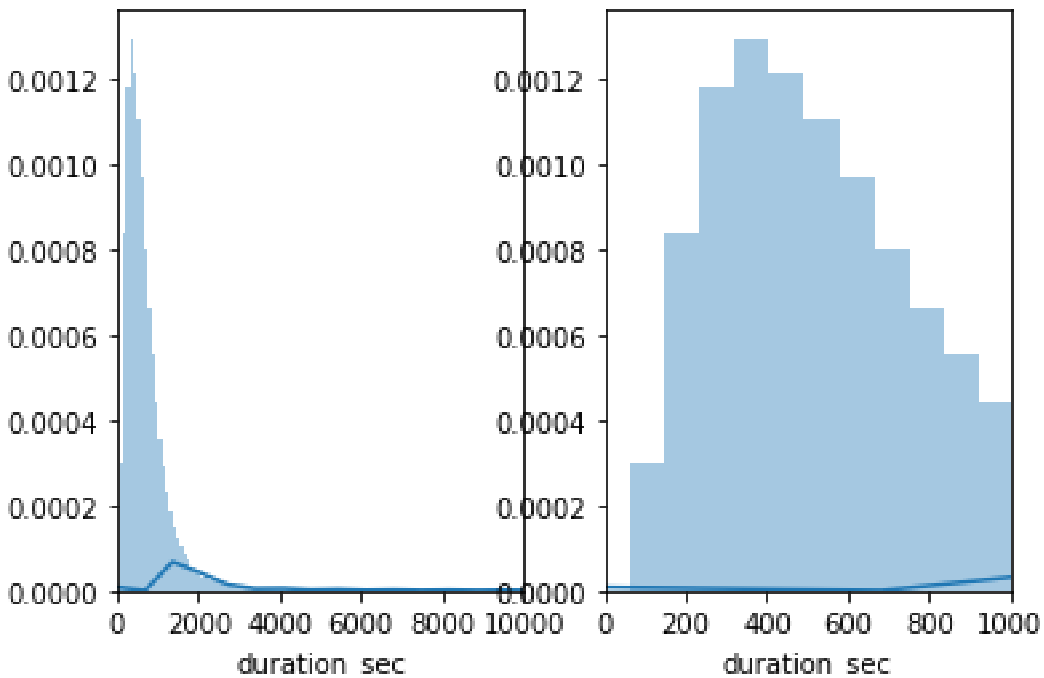
[31]:
```
# set the base color for the plots
base_color = sb.color_palette()[0]
secondary_color = sb.color_palette()[1]
tertiary_color = sb.color_palette()[2]
```

[32]:
```
# what do trip durations look like in our dataset
sb.distplot(df_rides.duration_sec);
```



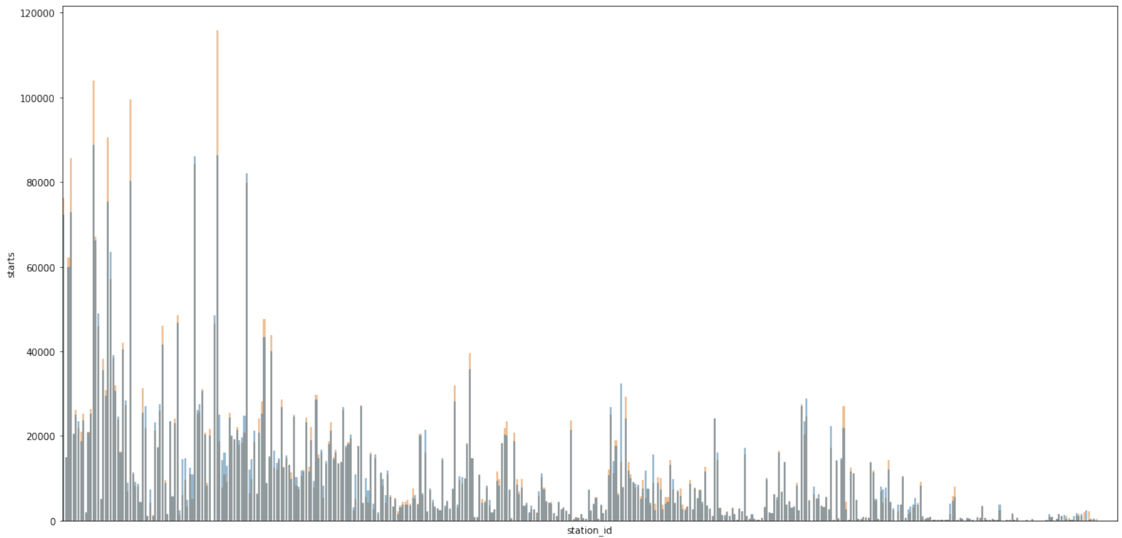
The distribution is clearly unimodal and heavily right-skewed. There are some heavy outliers - probably forgotten returns. Let's replot the data zooming into the peak

[33]:
```
plt.subplot(1,2,1);
sb.distplot(df_rides.duration_sec,1000);
plt.xlim(0,10000);
plt.subplot(1,2,2);
sb.distplot(df_rides.duration_sec,1000);
plt.xlim(0,1000);
```



The peak is around 400 seconds, which makes sense for the bike ride. Now let's look at the overall station popularity.

[34]:
```
plt.figure(figsize=(20,10))
sb.barplot(data=df_stations_clean, x='station_id', y='ends', color =␣
 ↪secondary_color, alpha=0.5);
sb.barplot(data=df_stations_clean, x='station_id', y='starts', color =␣
 ↪base_color, alpha=0.5);
plt.xticks([]);
```



Judging by the grey color of the plot - on average the start and end stations popularity is well balanced, however colored bar tips tell us that some stations have more starts and some - more ends.

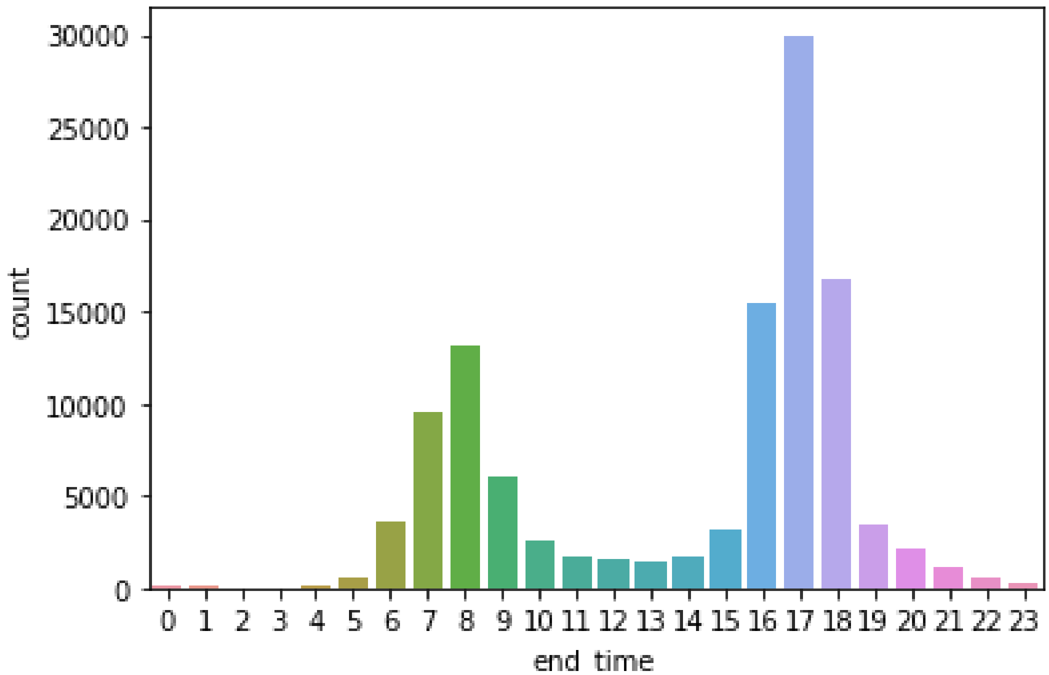
Let's try to identify single most popular destination.

[35]:
```
df_stations_clean.query('ends > 110000')
```

[35]:
```
    station_id   latitude   longitude  \
62          67  37.776639 -122.395526

                                                    name   starts      ends
62  {San Francisco Caltrain Station 2  (Townsend S…  86248.0  115804.0
```

It's the San Francisco Caltrain Station. Let's see what's the time pattern for arriving at this location.

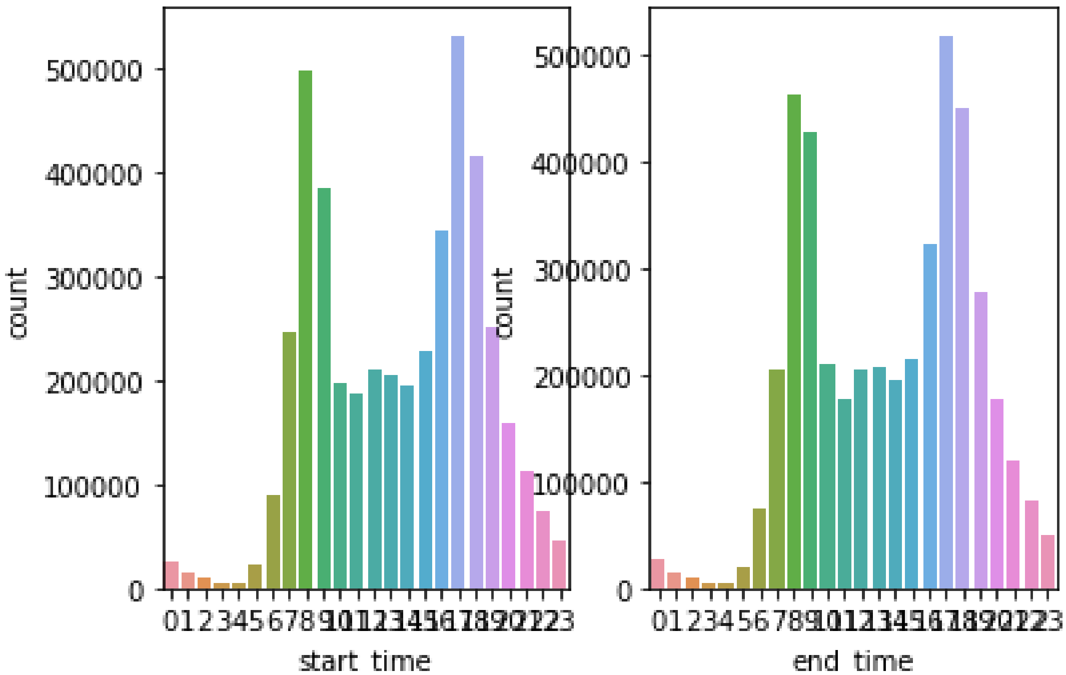
[36]:
```
train_station_arrival_hours = df_rides.query('end_station_id == 67').end_time.
 ↪apply(lambda x: x.hour)
sb.countplot(x=train_station_arrival_hours, data=df_rides);
```



It seems that most rides arrive here at the end of the day. Looks like bike+train is a popular end-of-the-day commute option.

Further - let's look at the overall distributions of ride start and end times.

[37]:
```
plt.subplot(1,2,1)
sb.countplot(x=df_rides.start_time.apply(lambda x: x.hour), data=df_rides);
plt.subplot(1,2,2)
sb.countplot(x=df_rides.end_time.apply(lambda x: x.hour), data=df_rides);
```



Start times and end times seem generally well-aligned. It's even possible to see end times peaks being slighly later than start times.

1.5.1 Discuss the distribution(s) of your variable(s) of interest. Were there any unusual points? Did you need to perform any transformations?

Distributions observed in this exploration, were ride duration, station popularity (start and end) and ride start times.

Ride duration was a unimodal, heavily right-skewed distribution with peak at around 400 seconds, which shows that a 15-minute ride is the most popular kind of use for this bike sharing service.

Station popularity was mapped with station id's mapped along the x-axis. This resulted in a rather noisy, but well-defined right-skewed distribution. This is somewhat surprising to see how much popular lower station id's are. This goes to show that, if id's reflect historical pattern of installing stations - that either the prior market research was very successful that resulted in covering most popular locations first, or that service adoption grew steadily with the availability of the service at the location. Historic data on station installation would be required to draw further conclusions.

Ride times is a bi-modal distribution with peaks at 8am and 17pm aligned perfectly with the start and end of a typical workday. The tail tapering off in an exponential curve after 17pm shows how service is used primarily to reach home destinations from work and probably party locations later.

1.5.2 Of the features you investigated, were there any unusual distributions? Did you perform any operations on the data to tidy, adjust, or change the form of the data? If so, why did you do this?

The unusual thing about distributions observed was how perfectly they reflected common sense assumptions about the use of such service in a busy city - how well they mapped on central stations being busier and overall use mapping perfectly onto a busy work day schedule.

To tidy the data I separated the dataset in two: one containing information about the stations and the other one - information about the rides. This follows the rule of tidy data make is easier to work with geographic station information later on.

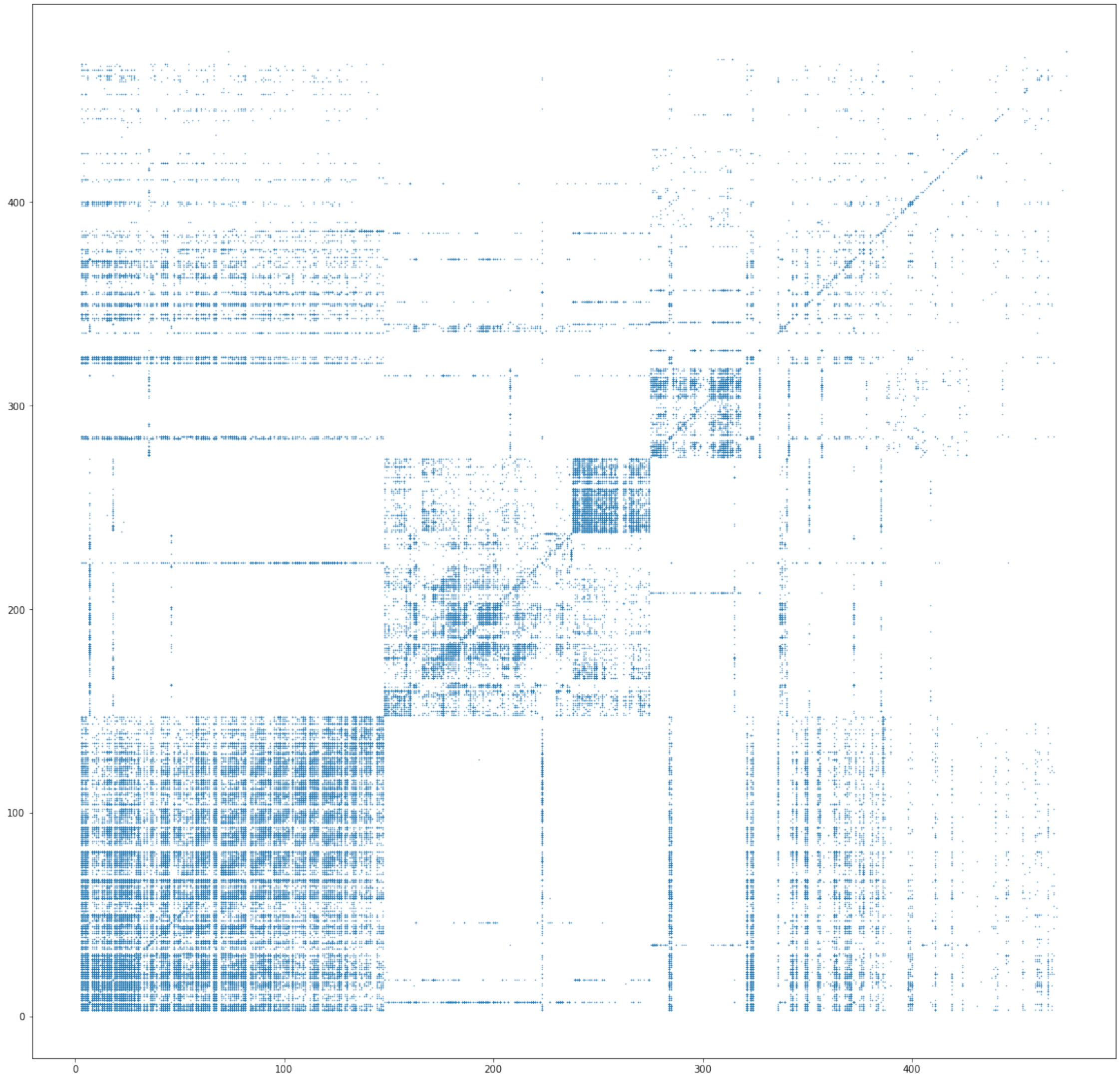
1.6 Bivariate Exploration

It would be interesting to see if there are any patterns behind the use of stations to pick up or return the bicycles.

[42]:

```
sample = np.random.choice(df_rides.shape[0],200000, replace = False);
sample = df_rides.loc[sample];
plt.figure(figsize=(20,20))
plt.scatter(data = sample, x = 'start_station_id', y = 'end_station_id',␣
↪alpha=1, s=0.1);
```

```
/Users/s8/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:2:
FutureWarning:
Passing list-likes to .loc or [] with any missing label will raise
KeyError in the future, you can use .reindex() as an alternative.

See the documentation here:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#deprecate-
loc-reindex-listlike
```



From this visualization it is clear that: * Travel patterns are clustered around very specific groups of stations. * Roundtrips, where start and end stations are the same are quite popular. * Plot is very symmetrical along the diagonal, which means a lot of bikes go back to the same stations they were picked up from

For now let's look at roundtrips, where bikes are picked up and dropped off at the same station and compare their durations with the random sample of the same size from the whole dataset.

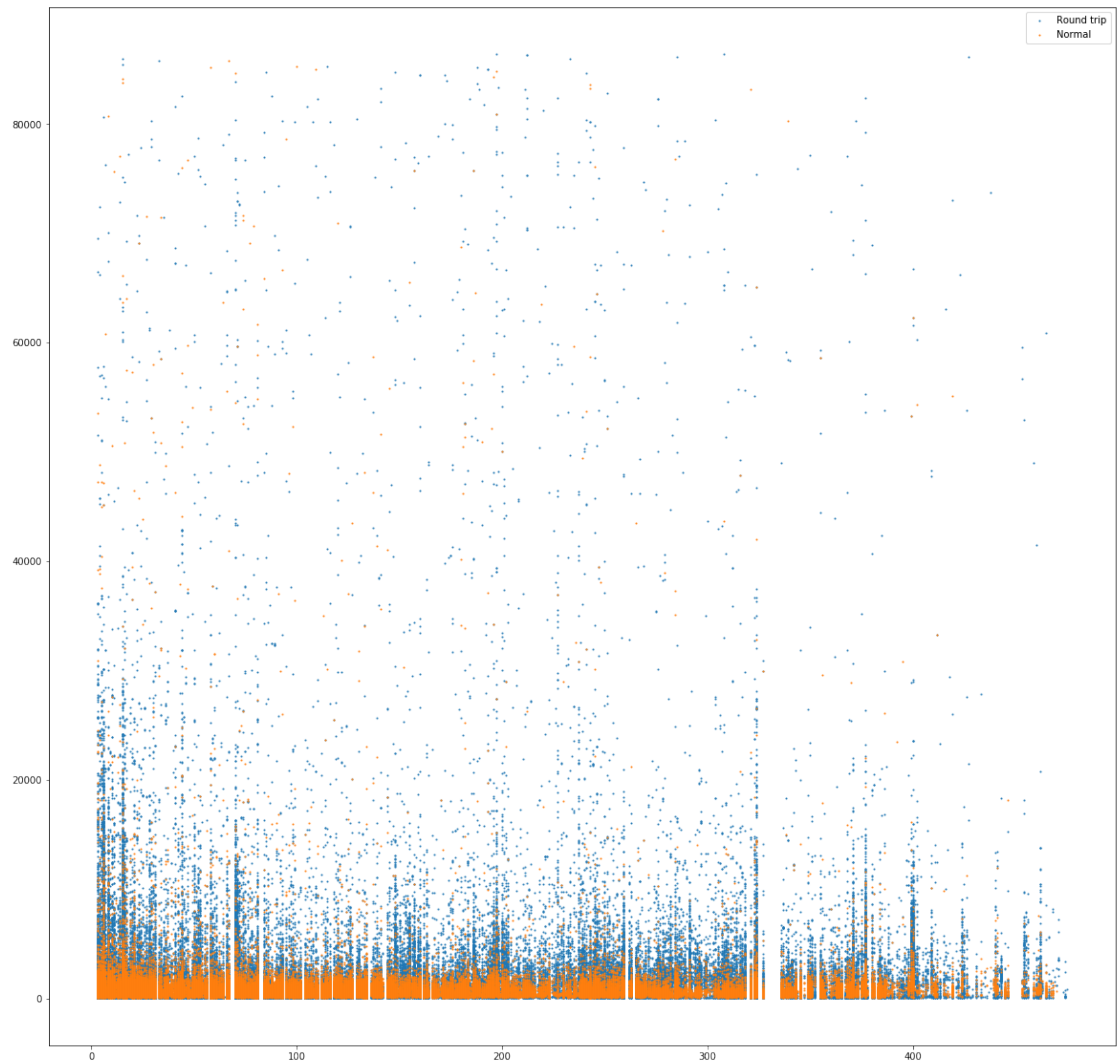
[39]:
```
# make a dataframe with roundtrips only
df_roundtrips = df_rides.query('start_station_id == end_station_id')
```

[ ]:
```
df_roundtrips.shape
```

[40]:

```
rides_sample = np.random.choice(df_rides.shape[0],122230, replace = False);
rides_sample = df_rides.loc[rides_sample];

plt.figure(figsize=(20,20))

plt.scatter(data = df_roundtrips, x = 'start_station_id', y = 'duration_sec',␣
 ↪alpha=1, s=1);
plt.scatter(data = rides_sample, x = 'start_station_id', y = 'duration_sec',␣
 ↪alpha=1, s=1, color=secondary_color);

plt.legend(['Round trip','Normal']);
```

```
/Users/s8/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:2:
FutureWarning:
Passing list-likes to .loc or [] with any missing label will raise
KeyError in the future, you can use .reindex() as an alternative.

See the documentation here:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#deprecate-
loc-reindex-listlike
```



[43]:
```
print ('overall ride duration mean:   ', df_rides.duration_sec.mean())
print ('roundtrip ride duration mean: ', df_roundtrips.duration_sec.mean())
```

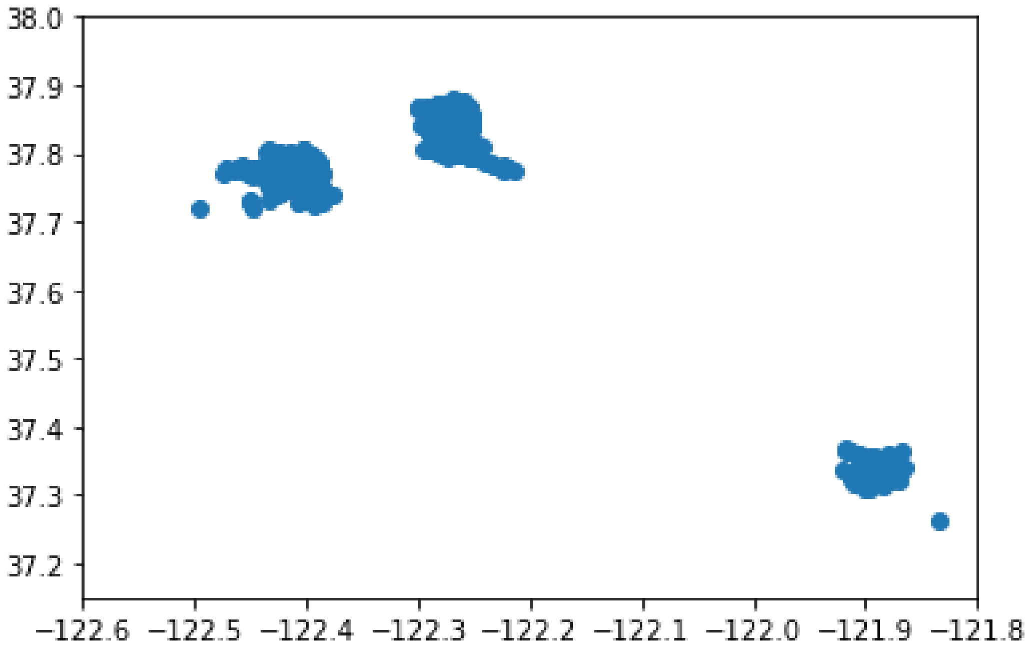
```
overall ride duration mean:    859.7962501331438
roundtrip ride duration mean:  2529.719487850773
```

Plot above as well as mean calculation clearly shows that roundtrips are usually around 3 times longer than A-B rides.

Now let's look at the geographical spread of the stations.

[52]:
```
plt.scatter(data = df_stations_clean, x = 'longitude', y = 'latitude', )
plt.xlim(-122.6,-121.8)
plt.ylim(37.15,38)
```

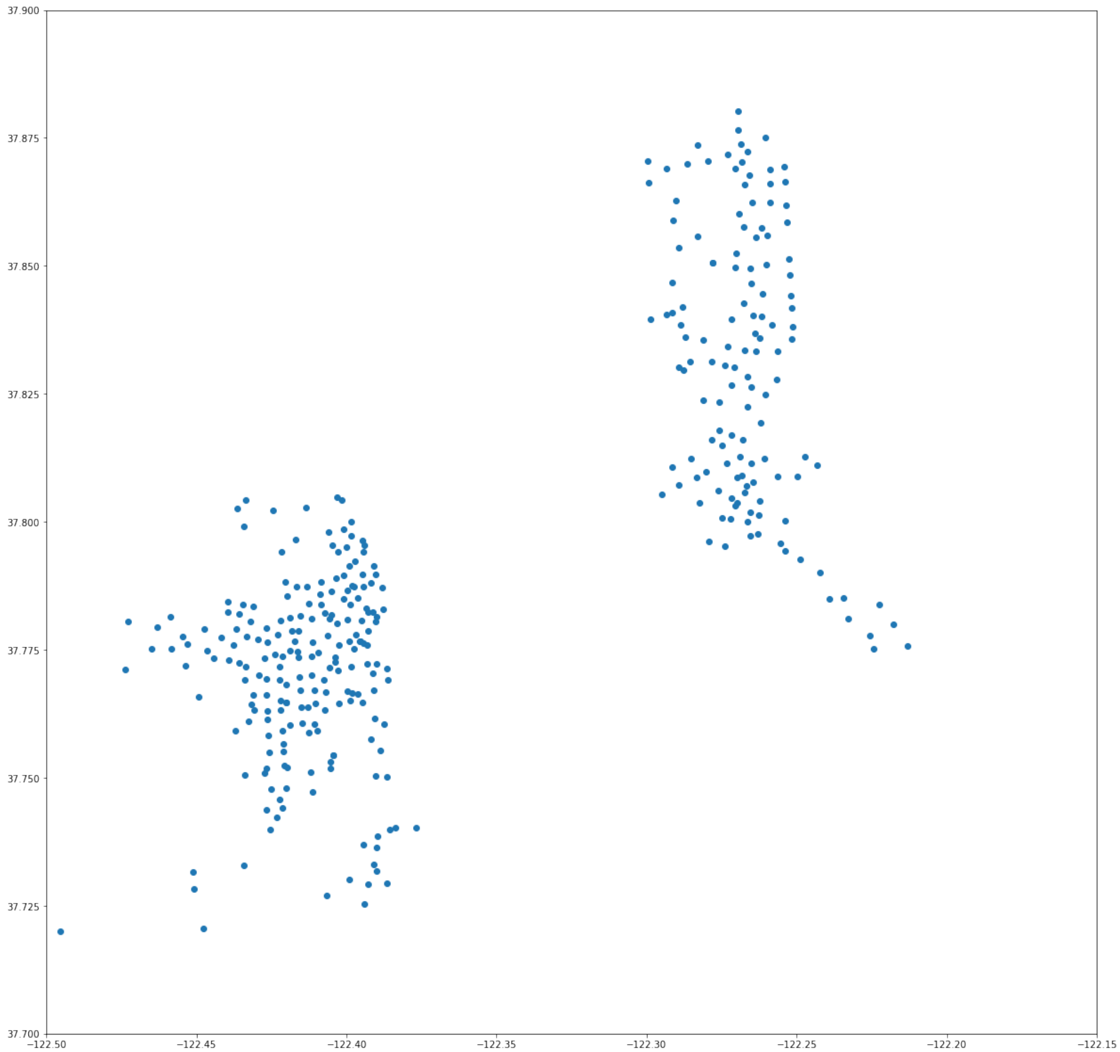
[52]: (37.15, 38)



there seems to be four distinct clusters of stations. Let's look at them separately.

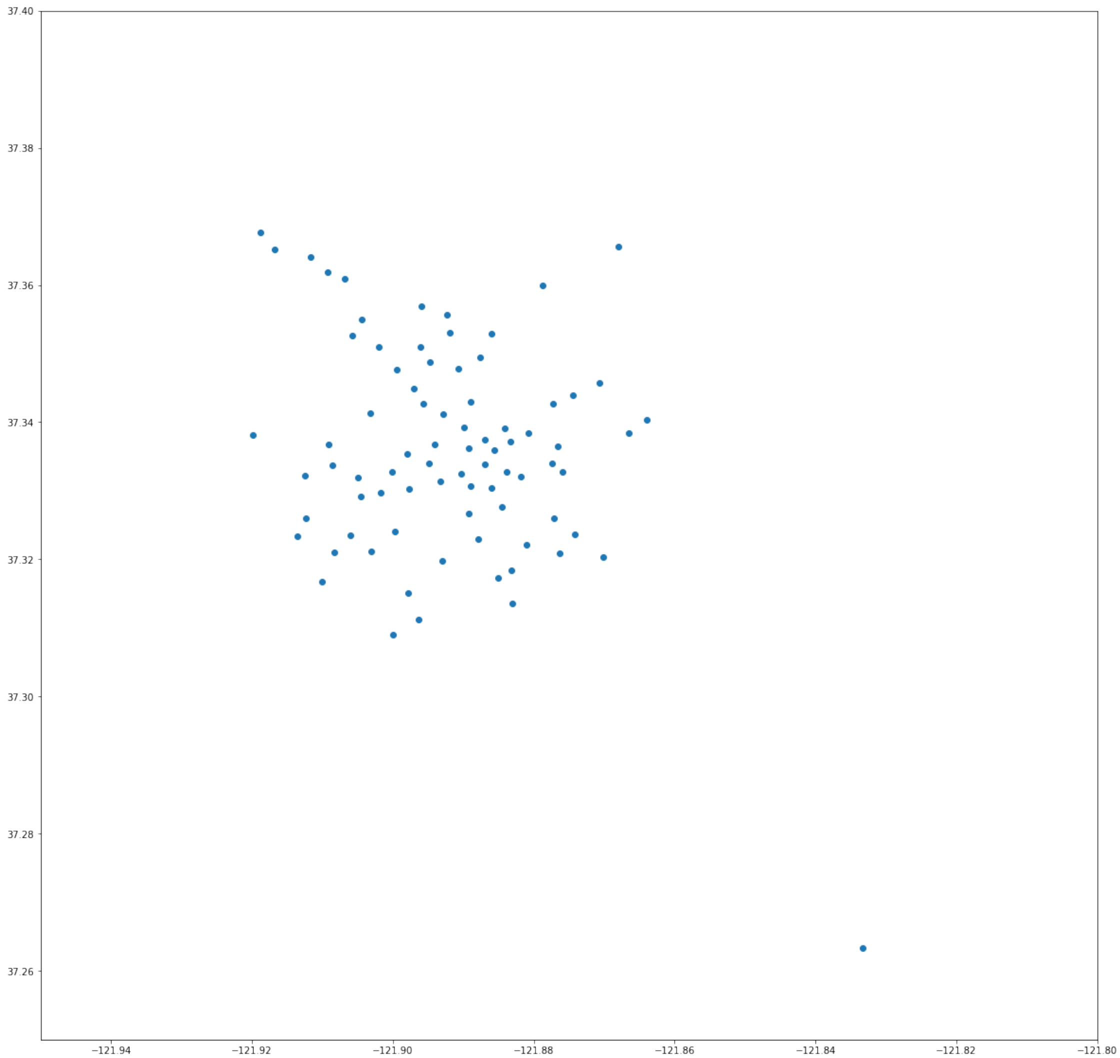
[45]:
```
# San Francisco, Berkeley and Oakland
plt.figure(figsize=(20,20))
plt.scatter(data = df_stations_clean, x = 'longitude', y = 'latitude', )
plt.xlim(-122.5,-122.15)
plt.ylim(37.7,37.9)
```

[45]: (37.7, 37.9)



[46]:
```
# San Jose and Palo Alto
plt.figure(figsize=(20,20))
plt.scatter(data = df_stations_clean, x = 'longitude', y = 'latitude', )
plt.xlim(-121.95,-121.8)
plt.ylim(37.25,37.4)
```

[46]: (37.25, 37.4)



There seems to be one station isolated from the cluster. Let's see if that's a data quality issue.

[47]:
```
df_stations_clean.query('latitude < 37.28').query('longitude > -121.84')
```

[47]:
```
     station_id  latitude   longitude                           name  \
335         374  37.26331 -121.833332                 {Viva Calle SJ}
378         420   0.00000    0.000000               {SF Test Station}
403         449   0.00000    0.000000  {16th Depot Bike Fleet Station}

     starts    ends
335  2795.0  2516.0
378  3859.0  2374.0
403  1115.0  1124.0
```

Apart from the test depot - there is only one station. Quick search on the map revelas that this is a large park on the southern end of San Jose. Let's see use pattern of this station.

[48]:
```
print ('ride starts: ', len (df_rides.query('end_station_id == 374')))
print ('ride ends: ', len (df_rides.query('start_station_id == 374')))
```

```
ride starts:  15
ride ends:  25
```

This doesn't seem to be a very popular station, which is likely to be due to it being isolated from the rest of the infrastructure.

1.6.1 Talk about some of the relationships you observed in this part of the investigation. How did the feature(s) of interest vary with other features in the dataset?

About 2.7% of all rides are roudntrips that start and finish at the same station. Their duration is on average 300% that of a normal a-b ride.

Mapping start and finish stations to a scatterplot it became clear that most of the rides are concentrated in distinct clusters that seem to map onto the way the station infrastructure has been deployed.

1.7 Multivariate Exploration

Let's identify the most popular stations in the network.

[49]:
```
df_rides.head()
```

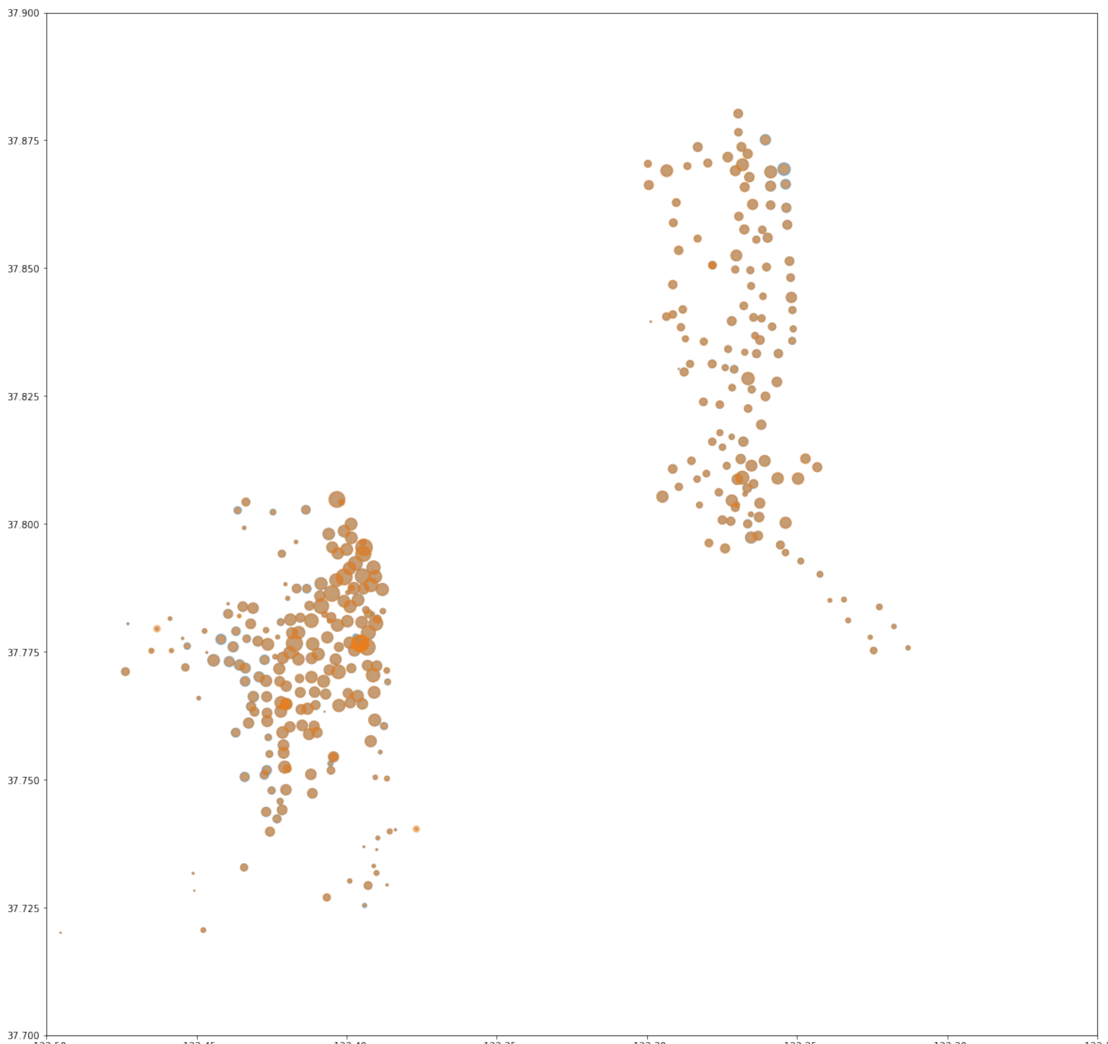
[49]:
```
   bike_id  duration_sec  end_station_id                end_time  \
0       96         80110              43 2018-01-01 15:12:50.245
1       88         78800              96 2018-01-01 13:49:55.617
2     1094         45768             245 2018-01-01 11:28:36.883
3     2831         62172               5 2018-01-01 10:47:23.531
4     3167         43603             247 2018-01-01 02:29:57.571

   start_station_id              start_time
0                74 2017-12-31 16:57:39.654
1               284 2017-12-31 15:56:34.842
2               245 2017-12-31 22:45:48.411
3                60 2017-12-31 17:31:10.636
4               239 2017-12-31 14:23:14.001
```

[50]:
```
starts = df_rides.groupby('start_station_id', as_index=False).agg('count')
ends = df_rides.groupby('end_station_id', as_index=False).agg('count')
```

[51]:
```
# San Francisco and Berkeley
plt.figure(figsize=(20,20))
plt.scatter(data = df_stations_clean, x = 'longitude', y = 'latitude',␣
 ↪s=df_stations_clean['starts']**0.5, alpha=0.5)
```

```
plt.scatter(data = df_stations_clean, x = 'longitude', y = 'latitude', s=df_stations_clean['ends']**0.5, color=secondary_color, alpha=0.5)
plt.xlim(-122.5,-122.15)
plt.ylim(37.7,37.9)
```

[51]: (37.7, 37.9)



It seems that in San Francisco and Berkeley there are two particular stations - one with lots of starts and the other one with lots of ends. Let's take a closer look at them.

[53]:
```
df_stations_clean.query('starts-ends > 10000')
```

[53]:
```
     station_id  latitude   longitude                          name   starts  \
225         243  37.86936 -122.254337  {Bancroft Way at College Ave}  32476.0
```

```
        ends
225  13776.0
```

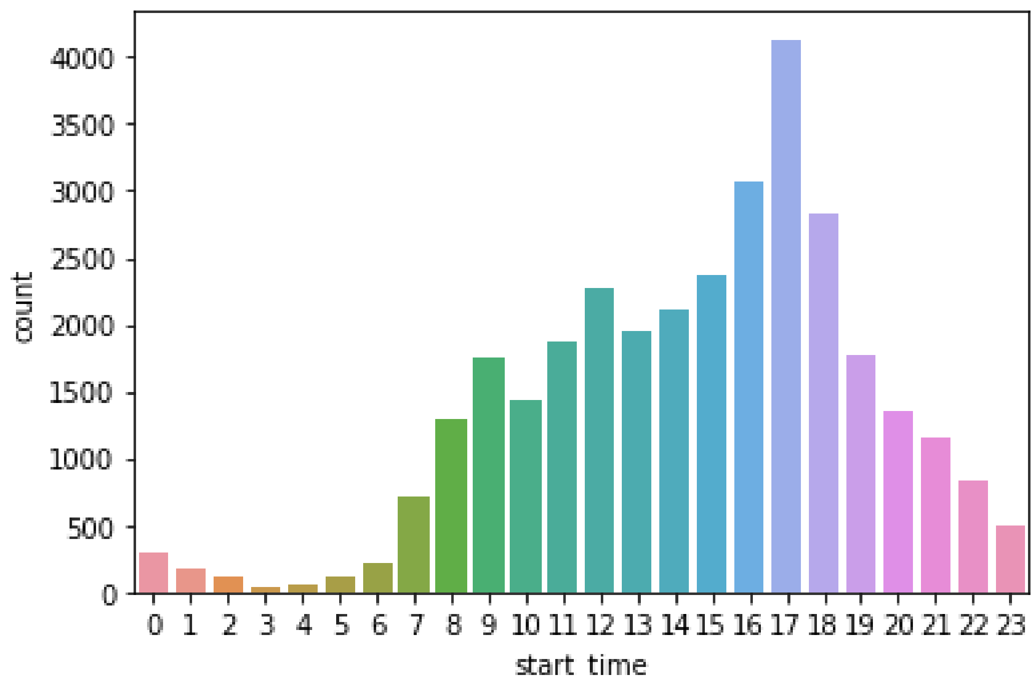
[54]:
```
df_stations_clean.query('ends-starts > 20000')
```

[54]:
```
    station_id   latitude   longitude  \
62          67  37.776639 -122.395526

                                                name   starts      ends
62  {San Francisco Caltrain Station 2  (Townsend S…  86248.0  115804.0
```

Again - we can see that the train statio is the most popular bike 'sink'.

Station with more starts rather than ends is located next to Berkley University as well as the stadium in northern part of Berkley. Judging by the time use histogram - the bikes are mainly used to leave the place at the end of a work day.

[64]:
```
berkley_departure_hours = df_rides.query('start_station_id == 243').start_time.
 ↪apply(lambda x: x.hour)
sb.countplot(x=berkley_departure_hours, data=df_rides);
```



[56]:
```
bikes = df_rides.groupby('bike_id', as_index=False).agg({'start_station_id':
 ↪list, 'end_station_id':list,'start_time':list,'end_time':list,'duration_sec':
 ↪list})
```

[57]:
```python
bikes['count'] = df_rides.groupby('bike_id', as_index=False).
 ↪agg('count')['start_station_id']
```

[58]:
```python
bikes.head()
```

[58]:
```
   bike_id                                   start_station_id  \
0       10  [150, 150, 149, 149, 149, 149, 149, 149, 149, …
1       11  [163, 182, 194, 181, 160, 213, 160, 235, 160, …
2       12  [189, 246, 189, 190, 189, 176, 160, 160, 230, …
3       13  [44, 15, 14, 11, 211, 176, 154, 176, 189, 162,…
4       14  [212, 176, 215, 176, 173, 241, 243, 249, 247, …

                                      end_station_id  \
0  [150, 150, 149, 149, 149, 149, 149, 149, 149, …
1  [233, 196, 182, 194, 213, 160, 213, 160, 235, …
2  [190, 189, 246, 189, 190, 189, 213, 160, 160, …
3  [59, 6, 15, 14, 7, 211, 176, 154, 176, 189, 16…
4  [176, 212, 176, 215, 176, 173, 241, 243, 249, …

                                          start_time  \
0  [2017-11-25 16:48:48.689000, 2017-11-25 16:30:…
1  [2017-12-19 14:58:36.866000, 2017-12-18 18:10:…
2  [2017-12-23 11:47:37.482000, 2017-12-22 19:21:…
3  [2017-07-03 08:32:29.946000, 2017-06-29 08:30:…
4  [2017-12-26 12:40:01.358000, 2017-12-19 18:15:…

                                            end_time  \
0  [2017-11-25 17:08:58.128000, 2017-11-25 16:38:…
1  [2017-12-19 15:03:29.717000, 2017-12-18 18:16:…
2  [2017-12-23 15:52:55.630000, 2017-12-22 19:33:…
3  [2017-07-03 08:39:10.562000, 2017-06-29 08:34:…
4  [2017-12-26 12:45:32.090000, 2017-12-19 18:20:…

                                        duration_sec  count
0  [1209, 472, 98, 343, 66, 121, 168, 187, 234, 104]     10
1  [292, 343, 662, 1160, 1029, 630, 565, 153, 195…    696
2  [14718, 719, 873, 348, 314, 527, 547, 1879, 18…    903
3  [400, 261, 217, 147, 762, 247, 1077, 1813, 420…   1024
4  [330, 251, 3146, 237, 247, 360, 467, 472, 834,…    503
```

[59]:
```python
bikes.query('count > 1800')
```

[59]:
```
      bike_id                                   start_station_id  \
116       126  [285, 55, 70, 43, 141, 98, 93, 19, 70, 56, 33,…
222       232  [15, 66, 133, 115, 10, 33, 78, 147, 108, 123, …
736       746  [72, 125, 107, 22, 79, 30, 25, 19, 133, 121, 1…
1141     1161  [23, 8, 30, 15, 60, 21, 14, 30, 17, 30, 323, 1…
```

```
1160     1181  [89, 106, 139, 106, 106, 22, 45, 61, 45, 49, 2…
1375     1396  [144, 41, 321, 84, 97, 72, 127, 97, 59, 97, 43…
1522     1543  [85, 36, 58, 75, 6, 16, 6, 15, 15, 15, 81, 63,…
2153     2174  [324, 323, 141, 109, 120, 25, 81, 27, 17, 27, …
2366     2387  [67, 99, 97, 324, 47, 86, 88, 85, 223, 96, 101…
2476     2497  [34, 74, 58, 81, 36, 97, 85, 61, 67, 78, 30, 5…
2524     2545  [58, 71, 60, 31, 6, 8, 22, 5, 81, 15, 6, 93, 8…
2671     2692  [129, 112, 122, 147, 17, 144, 34, 114, 81, 47,…
2732     2753  [95, 34, 4, 95, 4, 47, 5, 58, 86, 141, 129, 76…
3081     3105  [98, 134, 42, 133, 47, 321, 144, 45, 20, 42, 1…
3122     3146  [6, 6, 9, 20, 28, 66, 127, 120, 108, 108, 101,…
3351     3379  [52, 10, 25, 15, 16, 28, 11, 28, 22, 81, 126, …

                                          end_station_id  \
116   [121, 285, 55, 90, 42, 141, 98, 93, 19, 70, 56…
222   [6, 15, 66, 133, 115, 10, 33, 78, 124, 108, 12…
736   [72, 72, 125, 102, 25, 79, 30, 67, 19, 133, 12…
1141  [30, 23, 8, 30, 15, 60, 21, 14, 30, 17, 30, 32…
1160  [106, 89, 106, 139, 106, 106, 22, 45, 61, 45, …
1375  [77, 144, 41, 321, 84, 97, 134, 127, 97, 59, 9…
1522  [78, 85, 36, 58, 75, 6, 16, 6, 6, 6, 15, 81, 6…
2153  [30, 324, 323, 141, 109, 120, 20, 81, 27, 17, …
2366  [45, 67, 99, 97, 324, 49, 86, 88, 85, 223, 96,…
2476  [86, 34, 74, 58, 67, 36, 97, 120, 61, 67, 78, …
2524  [5, 58, 116, 60, 31, 6, 8, 6, 5, 81, 15, 6, 47…
2671  [139, 129, 112, 122, 147, 17, 144, 34, 114, 81…
2732  [122, 95, 34, 4, 95, 4, 47, 5, 58, 86, 141, 12…
3081  [133, 98, 134, 42, 133, 47, 321, 144, 45, 20, …
3122  [26, 6, 6, 9, 20, 28, 66, 127, 120, 108, 108, …
3351  [52, 4, 10, 25, 6, 16, 28, 11, 8, 22, 81, 126,…

                                              start_time  \
116   [2017-12-26 20:48:44.817000, 2017-12-26 15:18:…
222   [2017-12-29 12:34:49.312000, 2017-12-29 10:51:…
736   [2017-12-31 14:38:46.875000, 2017-12-30 17:02:…
1141  [2017-12-29 15:17:50.202000, 2017-12-28 20:32:…
1160  [2017-12-31 11:49:44.955000, 2017-12-31 11:07:…
1375  [2017-12-31 16:12:40.417000, 2017-12-31 15:15:…
1522  [2017-12-31 11:49:38.212000, 2017-12-29 17:21:…
2153  [2017-12-29 15:50:58.257000, 2017-12-28 15:54:…
2366  [2017-12-11 08:29:57.662000, 2017-12-10 10:49:…
2476  [2017-12-24 13:08:51.655000, 2017-12-24 10:32:…
2524  [2017-12-31 11:14:40.962000, 2017-12-29 16:00:…
2671  [2017-12-31 20:47:49.756000, 2017-12-31 14:24:…
2732  [2017-12-27 18:05:55.445000, 2017-12-27 10:04:…
3081  [2017-12-31 13:11:15.157000, 2017-12-31 10:45:…
3122  [2017-12-30 20:38:34.373000, 2017-12-30 15:07:…
```

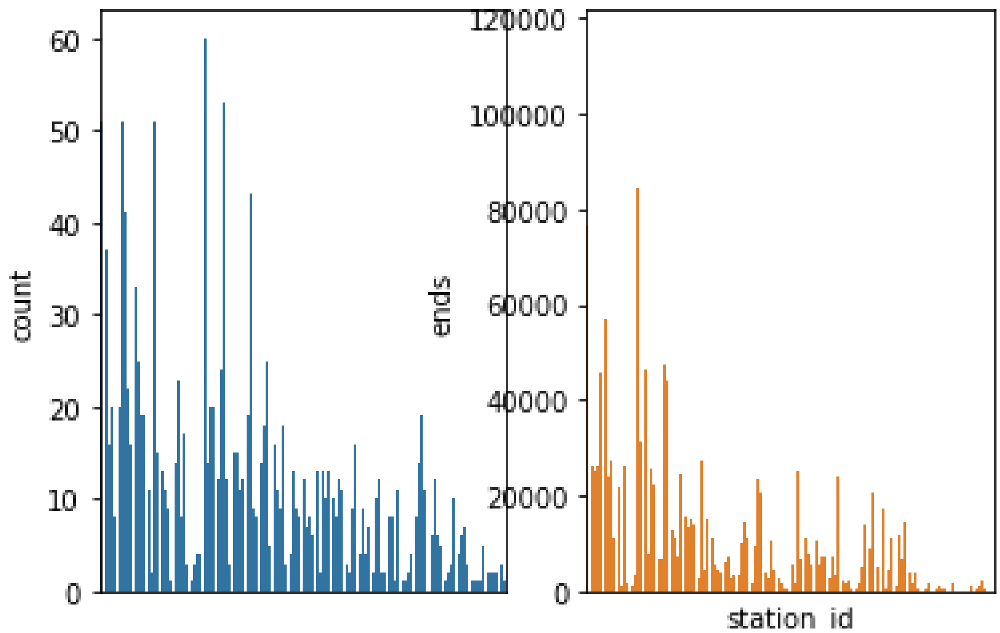
```
3351  [2017-12-30 10:43:47.721000, 2017-12-23 13:23:…

                                           end_time  \
116   [2017-12-26 21:00:25.041000, 2017-12-26 15:22:…
222   [2017-12-29 13:08:29.972000, 2017-12-29 11:04:…
736   [2017-12-31 17:33:56.394000, 2017-12-30 17:24:…
1141  [2017-12-29 15:26:30.277000, 2017-12-28 20:52:…
1160  [2017-12-31 12:03:33.312000, 2017-12-31 11:19:…
1375  [2017-12-31 16:27:20.332000, 2017-12-31 15:33:…
1522  [2017-12-31 12:02:24.265000, 2017-12-29 17:45:…
2153  [2017-12-29 16:02:29.323000, 2017-12-28 16:11:…
2366  [2017-12-11 08:36:45.499000, 2017-12-10 10:59:…
2476  [2017-12-24 13:21:01.236000, 2017-12-24 10:43:…
2524  [2017-12-31 11:19:55.155000, 2017-12-29 16:15:…
2671  [2017-12-31 20:51:23.284000, 2017-12-31 14:26:…
2732  [2017-12-27 18:12:53.762000, 2017-12-27 10:24:…
3081  [2017-12-31 13:16:37.987000, 2017-12-31 10:50:…
3122  [2017-12-30 20:50:10.841000, 2017-12-30 15:25:…
3351  [2017-12-30 11:16:44.065000, 2017-12-23 13:40:…

                                       duration_sec  count
116   [700, 253, 1310, 406, 936, 570, 690, 1123, 139…   1817
222   [2020, 789, 1009, 1226, 2320, 1512, 904, 995, …   1835
736   [10509, 1280, 552, 1110, 461, 237, 370, 666, 1…   1812
1141  [520, 1171, 1147, 638, 2356, 573, 302, 870, 59…   1882
1160  [828, 751, 1181, 1061, 2732, 1361, 431, 189, 2…   1808
1375  [879, 1044, 753, 902, 514, 473, 3860, 352, 340…   1927
1522  [766, 1443, 585, 195, 8269, 763, 280, 301, 374…   1804
2153  [691, 1023, 1608, 656, 394, 1120, 720, 513, 18…   1967
2366  [407, 604, 152, 866, 1102, 1249, 434, 469, 546…   1843
2476  [729, 634, 407, 904, 502, 715, 218, 1360, 465,…   1832
2524  [314, 922, 455, 919, 3636, 403, 393, 5379, 654…   1823
2671  [213, 135, 277, 455, 1605, 1587, 1578, 1231, 5…   1964
2732  [418, 1228, 316, 1360, 1238, 487, 335, 271, 30…   1818
3081  [322, 267, 662, 766, 1282, 172, 1517, 1434, 38…   1831
3122  [696, 1104, 1759, 1722, 369, 380, 1061, 484, 1…   1864
3351  [1976, 1048, 434, 366, 353, 340, 486, 491, 508…   1883
```

[60]:

```
bike_2174 = bikes.loc[2153]
```

[62]:

```
plt.subplot(1,2,1)
sb.countplot(x='start_station_id', data=bike_2174, color=base_color);
plt.xticks([]);
plt.subplot(1,2,2)
sb.barplot(data=df_stations_clean, x='station_id', y='ends', color =␣
 ↪secondary_color);
plt.xticks([]);
```



Interesting that in the broad general tendendcies, use pattern of the single most used bike is resembling use patterns of all the bikes acrosst the network. Also interesting to see that it has travelled across the whole network rather than staying in a particular area.

1.7.1 Talk about some of the relationships you observed in this part of the investigation. Were there features that strengthened each other in terms of looking at your feature(s) of interest?

It was interesting to see that the bicycle network is mostly well balanced, i.e. most of the stations act as both - starts and ends for the ride. There is a only small amount of mainly 'sink' stations.

1.7.2 Were there any interesting or surprising interactions between features?

It was interesting to see how round trips are almost a different kind of entity from the rest of the dataset.

> At the end of your report, make sure that you export the notebook as an html file from the `File > Download as... > HTML` menu. Make sure you keep track of where the exported file goes, so you can put it in the same folder as this notebook for project submission. Also, make sure you remove all of the quote-formatted guide notes like this one before you finish your report!

```
[ ]:
```