Project 5: Communicate Data Findings

Bay Wheels is a regional public bicycle sharing system operated by Motivate in partnership with the Metropolitan Transportation Commission and the Bay Area Air Quality Management District, in the San Francisco Bay Area, California.

Bay Wheels is the first national and large-scale cycling sharing program to be deployed both in California and on the US West Coast. It was founded in August 2013 as Bay Area Bike Share. The Bay Wheels network had more than 2600 bicycles in 262 stations across San Francisco, East Bay and San Jose.

The system officially re-launched on 28 June 2017 in a partnership with Ford Motor Company as Ford GoBike. The system was subsequently renamed to Bay Wheels in June 2019 following Lyft 's acquisition of Motivate. The network is planned to extend to about 540 stations in San Francisco, Oakland, Berkeley, Emeryville and San Jose to 7,000 bicycles.

Preliminary Wrangling

In [82]:
```
# import all packages and set plots to be embedded inline
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
from datetime import datetime
import os
import glob
import zipfile
import datetime
import folium
from sklearn.cluster import KMeans

%matplotlib inline
```

Gathering and Assessing Data

In [83]:
```
# only ran once to append all monthly trip data together

folder_name = 'Data'
frames = [pd.read_csv(f) for f in glob.glob(os.path.join(folder_name, '*.csv'))]
result = pd.concat(frames, ignore_index=True)
print(result.shape)
result.sample(5)
```

```
(1863721, 16)
```

Out[83]:

	duration_sec	start_time	end_time	start_station_id	start_station_name	start_station_latitude	start_station_longitude
297528	137	2018-03-06 11:50:54.7170	2018-03-06 11:53:11.9610	312.0	San Jose Diridon Station	37.329732	-121.901782
1573381	735	2018-10-04 09:40:26.4190	2018-10-04 09:52:42.2550	181.0	Grand Ave at Webster St	37.811377	-122.265192
225259	224	2018-03-26 20:26:52.8100	2018-03-26 20:30:36.8630	36.0	Folsom St at 3rd St	37.783830	-122.398870
1136775	386	2018-08-13 06:24:45.4420	2018-08-13 06:31:12.4000	214.0	Market St at Brockhurst St	37.823321	-122.275732
1602919	282	2018-11-30 08:44:15.5700	2018-11-30 08:48:57.7990	244.0	Shattuck Ave at Hearst Ave	37.873676	-122.268487

In [84]:
```
# save the appended result to a .csv for further usage

result.to_csv('fordgobike_trips_2018.csv', index=False)
```

In [85]:
```
biketrips2018 = pd.read_csv('fordgobike_trips_2018.csv')
biketrips2018.head()
```

Out[85]:

	duration_sec	start_time	end_time	start_station_id	start_station_name	start_station_latitude	start_station_longitude
0	75284	2018-01-31 22:52:35.2390	2018-02-01 19:47:19.8240	120.0	Mission Dolores Park	37.761420	-122.426435
1	85422	2018-01-31 16:13:34.3510	2018-02-01 15:57:17.3100	15.0	San Francisco Ferry Building (Harry Bridges Pl...	37.795392	-122.394203
2	71576	2018-01-31 14:23:55.8890	2018-02-01 10:16:52.1160	304.0	Jackson St at 5th St	37.348759	-121.894798
3	61076	2018-01-31 14:53:23.5620	2018-02-01 07:51:20.5000	75.0	Market St at Franklin St	37.773793	-122.421239
4	39966	2018-01-31 19:52:24.6670	2018-02-01 06:58:31.0530	74.0	Laguna St at Hayes St	37.776435	-122.426244

In [86]:
```
biketrips2018.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1863721 entries, 0 to 1863720
Data columns (total 16 columns):
 #   Column                   Dtype  
---  ------                   -----  
 0   duration_sec             int64  
 1   start_time               object 
 2   end_time                 object 
 3   start_station_id         float64
 4   start_station_name       object 
 5   start_station_latitude   float64
 6   start_station_longitude  float64
 7   end_station_id           float64
 8   end_station_name         object 
 9   end_station_latitude     float64
 10  end_station_longitude    float64
 11  bike_id                  int64  
 12  user_type                object 
 13  member_birth_year        float64
 14  member_gender            object 
 15  bike_share_for_all_trip  object 
dtypes: float64(7), int64(2), object(7)
memory usage: 227.5+ MB
```

There are no missing values in the dataset.

Datatype of start_time, end_time are String but it should be DateTime.

We need to analyze bike_id, user_type, so it should be a categorical variable.

In [87]:
```
biketrips2018.isnull().sum()
```

Out[87]:
```
duration_sec                    0
start_time                      0
end_time                        0
start_station_id            11771
start_station_name          11771
start_station_latitude          0
start_station_longitude         0
end_station_id              11771
end_station_name            11771
end_station_latitude            0
end_station_longitude           0
bike_id                         0
user_type                       0
member_birth_year          110718
member_gender              110367
bike_share_for_all_trip         0
dtype: int64
```

In [88]:
```
biketrips2018.duplicated().sum()
```

Out[88]:
```
0
```

In [89]:
```
biketrips2018.member_gender.value_counts()
```

Out[89]:
```
Male      1288085
Female     438188
Other       27081
Name: member_gender, dtype: int64
```

In [90]:
```
biketrips2018.user_type.value_counts()
```

Out[90]:
```
Subscriber    1583554
Customer       280167
Name: user_type, dtype: int64
```

In [91]:
```
biketrips2018.bike_share_for_all_trip.value_counts()
```

Out[91]:
```
No     1701386
Yes     162335
Name: bike_share_for_all_trip, dtype: int64
```

In [92]:
```
biketrips2018.shape
```

Out[92]:
```
(1863721, 16)
```

In [93]:
```
# make a copy of the dataframe
trips18 = biketrips2018.copy()
```

In [94]:
```
trips18.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1863721 entries, 0 to 1863720
Data columns (total 16 columns):
 #   Column                   Dtype  
---  ------                   -----  
 0   duration_sec             int64  
 1   start_time               object 
 2   end_time                 object 
 3   start_station_id         float64
 4   start_station_name       object 
 5   start_station_latitude   float64
 6   start_station_longitude  float64
 7   end_station_id           float64
 8   end_station_name         object 
 9   end_station_latitude     float64
 10  end_station_longitude    float64
 11  bike_id                  int64  
 12  user_type                object 
 13  member_birth_year        float64
 14  member_gender            object 
 15  bike_share_for_all_trip  object 
dtypes: float64(7), int64(2), object(7)
memory usage: 227.5+ MB
```

Cleaning Data

In [ ]:

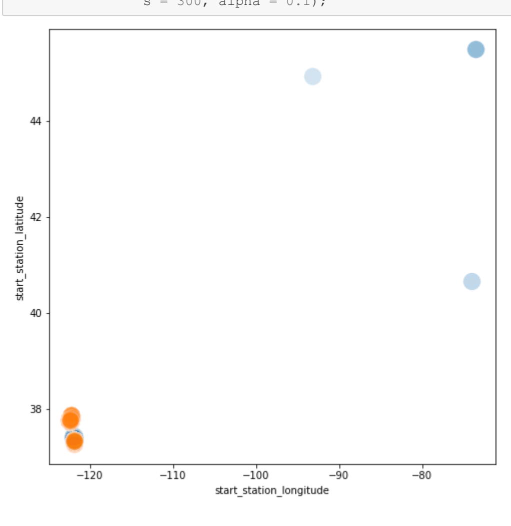
Define 1
start_station_id, start_station_name, end_station_id, end_station_name, member_birth_year and *member_gender* columns have missing values

In [95]:
```
# Check coordinates for rows with missing start/end station names and id
print(trips18[trips18.start_station_id.isnull()].start_station_latitude.value_counts())
print(trips18[trips18.start_station_id.isnull()].start_station_longitude.value_counts())
```

```
37.41    4809
37.40    4395
37.42    1204
37.39     822
37.38     405
37.34      37
37.43      23
37.37      17
37.35      17
37.33      17
37.36       8
45.51       4
40.66       3
37.44       3
37.32       2
44.95       2
45.50       2
37.45       1
Name: start_station_latitude, dtype: int64
-121.94    4047
-121.93    2790
-121.96    1948
-121.95    1537
-121.92    1250
-121.97      51
-121.91      40
-121.98      31
-121.90      25
-121.89      21
-121.88       8
-73.57        6
-121.87       4
-121.84       3
-74.01        3
-121.86       3
-93.22        2
-121.83       1
-121.99       1
Name: start_station_longitude, dtype: int64
```

> All those rows with missing values should be dropped. Most rows with missing start/end id and station name have similar coordinates. Also those rows have lower accuracy of coordinates with only 2 decimal points

In [96]:
```
# Plotting missing start/end id and station name
plt.figure(figsize=(8,8))
sb.scatterplot(data = trips18[trips18.start_station_id.isnull()],
               x = "start_station_longitude",
               y = "start_station_latitude",
               s = 300, alpha = 0.1)
sb.scatterplot(data = trips18.dropna(subset=["start_station_id"]).sample(100000),
               x = "start_station_longitude",
               y = "start_station_latitude",
               s = 300, alpha = 0.1);
```



> Only consider coordinates that are located in Bay Area. Outlier should be removed and not useful for the analysis.

Code 1

In [97]:
```
# Make copy of original df and drop rows with missing values
trips18_clean = trips18.dropna()
```

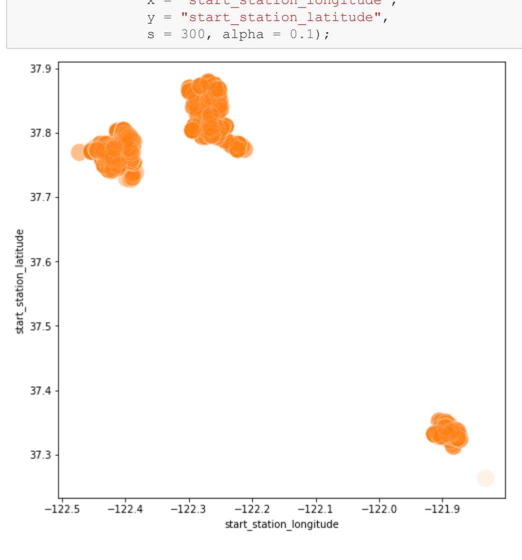
In [98]:
```
# Remove all the coordinates outside Bay Area
trips18_clean = trips18_clean.query('end_station_latitude > 37 and end_station_latitude < 38 and end_station_longitude > -123 and end_station_longitude < -121 and start_station_latitude > 37 and start_station_latitude < 38 and start_station_longitude > -123 and start_station_longitude < -121')
```

Test 1

In [99]:
```
# Check for missing values
trips18_clean.isna().sum().any()
```

Out[99]:
```
False
```

In [100]:
```
# Plotting no missing data df start/end id and station name
plt.figure(figsize=(8,8))
sb.scatterplot(data = trips18_clean[trips18_clean.start_station_id.isnull()],
               x = "start_station_longitude",
               y = "start_station_latitude",
               s = 300, alpha = 0.1)
sb.scatterplot(data = trips18_clean.dropna(subset=["start_station_id"]).sample(100000),
               x = "start_station_longitude",
               y = "start_station_latitude",
               s = 300, alpha = 0.1);
```



> Above graph shows the 3 main areas of GoBike: *SF, SJ* and *East Bay*.
> *start_metro_area* and *start_metro_area* columns should be added to label each station

Define 1.1
start_metro_area and *end_metro_area* columns should be added to label strat and end stations

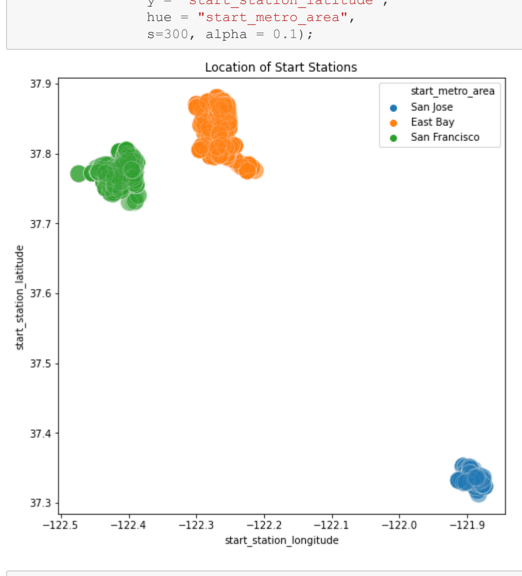
Code 1.1

In [101]:
```
# Using sklean module's K-Mean clustering algorithm to label each start/end stations
# Start staion
kmeans_start = KMeans(n_clusters=3, random_state=0).fit(trips18_clean[["start_station_latitude", "start_station_longitude"]])
trips18_clean['start_metro_area'] = kmeans_start.labels_
trips18_clean['start_metro_area'].replace({0:'East Bay', 1:'San Jose', 2:'San Francisco'}, inplace=True) # Hard coding the labels                          # set random_state = 0
```

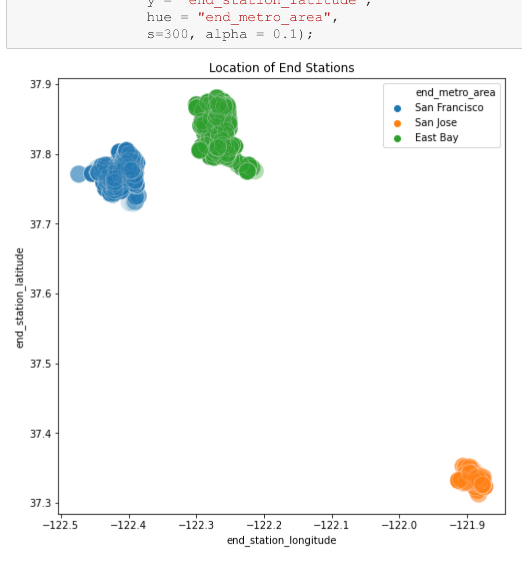
In [102]:
```
# End station
kmeans_end = KMeans(n_clusters=3, random_state=0).fit(trips18_clean[["end_station_latitude", "end_station_longitude"]])
trips18_clean['end_metro_area'] = kmeans_end.labels_
trips18_clean['end_metro_area'].replace({0:'East Bay', 1:'San Francisco', 2:'San Jose'}, inplace=True)   # Hard coding the labels                          # set random_state = 0
```

Test 1.1

In [103]:
```
# Plotting start station coordinates
plt.figure(figsize=(8,8))
plt.title('Location of Start Stations')
sb.scatterplot(data = trips18_clean.sample(100000),
               x = "start_station_longitude",
               y = "start_station_latitude",
               hue = "start_metro_area",
               s=300, alpha = 0.1);
```



In [104]:
```
# Plotting end station coordinates
plt.figure(figsize=(8,8))
plt.title('Location of End Stations')
sb.scatterplot(data = trips18_clean.sample(100000),
               x = "end_station_longitude",
               y = "end_station_latitude",
               hue = "end_metro_area",
               s=300, alpha = 0.1);
```



Define 2
`start_time` and `end_time` should be converted to datetime data type

Code 2

In [105]:
```
# Converted to datetime

# Change start_time and end_time to datetime format
trips18_clean.start_time = pd.to_datetime(trips18_clean.start_time)
trips18_clean.end_time = pd.to_datetime(trips18_clean.end_time)
```

Test 2

In [106]:
```
trips18_clean.dtypes
```

Out[106]:
```
duration_sec                        int64
start_time                 datetime64[ns]
end_time                   datetime64[ns]
start_station_id                  float64
start_station_name                 object
start_station_latitude            float64
start_station_longitude           float64
end_station_id                    float64
end_station_name                   object
end_station_latitude              float64
end_station_longitude             float64
bike_id                             int64
user_type                          object
member_birth_year                 float64
member_gender                      object
bike_share_for_all_trip            object
start_metro_area                   object
end_metro_area                     object
dtype: object
```

Define 3
`month`, `weekday` and `hours` column should be added for analysis

Code 3

In [107]:
```
# Create month column
trips18_clean['month'] = trips18_clean['start_time'].dt.month

# Create month column
trips18_clean['weekday'] = trips18_clean['start_time'].dt.weekday.astype('category')

# Create month column
trips18_clean['hour'] = trips18_clean['start_time'].dt.hour
```

Test 3

In [108]:
```
trips18_clean[['month','weekday','hour']].sample(5)
```

Out[108]:

	month	weekday	hour
720116	6	5	12
222068	3	1	16
367054	4	3	18
221785	3	1	17
1452073	10	1	13

Define 4
`start_station_id`, `end_station_id`, `member_birth_year` should be data type int

Code 4

In [109]:
```
trips18_clean['start_station_id']=trips18_clean['start_station_id'].astype('int')
trips18_clean['end_station_id']=trips18_clean['end_station_id'].astype('int')
trips18_clean['member_birth_year']=trips18_clean['member_birth_year'].astype('int')
```

Test 4

In [110]:
```
trips18_clean.dtypes
```

Out[110]:
```
duration_sec                        int64
start_time                 datetime64[ns]
end_time                   datetime64[ns]
start_station_id                    int32
start_station_name                 object
start_station_latitude            float64
start_station_longitude           float64
end_station_id                      int32
end_station_name                   object
end_station_latitude              float64
end_station_longitude             float64
bike_id                             int64
user_type                          object
member_birth_year                   int32
member_gender                      object
bike_share_for_all_trip            object
start_metro_area                   object
end_metro_area                     object
month                               int64
weekday                          category
hour                                int64
dtype: object
```

Define 5
`member_birth_year` should be converted to `member_age`

Code 5

In [111]:
```
# Get year when member use the bike
trips18_clean['start_time'].dt.year.value_counts()
```

Out[111]:
```
2018    1741556
Name: start_time, dtype: int64
```

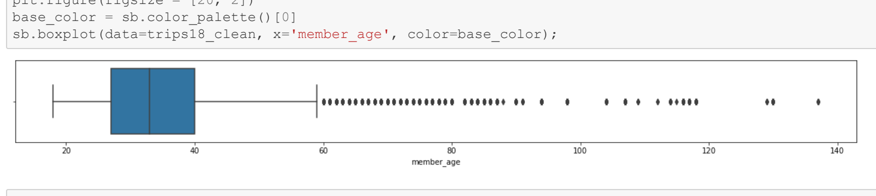
In [112]:
```
# Add member_age column
trips18_clean['member_age'] = trips18_clean['start_time'].dt.year - trips18_clean['member_birth_year']
```

Test 5

In [113]:
```
# Check for null
trips18_clean['member_age'].isna().sum()
```

Out[113]:
```
0
```

In [114]:
```
# Plot member age distibution
plt.figure(figsize = [20, 2])
base_color = sb.color_palette()[0]
sb.boxplot(data=trips18_clean, x='member_age', color=base_color);
```



In [115]:
```
trips18_clean.member_age.describe(percentiles = [0.01, 0.05, 0.95, 0.99])
```

Out[115]:
```
count    1.741556e+06
mean     3.493921e+01
std      1.045133e+01
min      1.800000e+01
1%       1.900000e+01
5%       2.200000e+01
50%      3.300000e+01
95%      5.600000e+01
99%      6.500000e+01
max      1.370000e+02
Name: member_age, dtype: float64
```

> From above, the youngest users are 18 which makes sense. However, 95% percentile combined with box plot shows older users above 55 year old seems to be outliers. It is logical that rows above 60 yrs old should be removed.

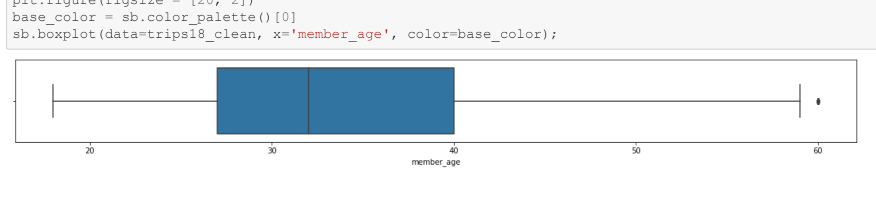
Define 5.1
`member_age` outlier should be removed

Code 5.1

In [116]:
```
# Remove all rows with age <= 60
trips18_clean = trips18_clean.query('member_age <= 60')
```

Test 5.1

In [117]:
```
# Plot member age <= 60 yrs distibution
plt.figure(figsize = [20, 2])
base_color = sb.color_palette()[0]
sb.boxplot(data=trips18_clean, x='member_age', color=base_color);
```



Define 5.2
Add `age_group` label column

Code 5.2

In [118]:
```
# Add age_group label column
labels = ["{}s".format(i) for i in range(10,51,10)]
trips18_clean['age_group'] = pd.cut(trips18_clean['member_age'], range(10, 61, 10), right=True, labels=labels)
```

Test 5.2

In [119]:
```
# Test member age and their age_group
trips18_clean[['age_group','member_age']].sample(5)
```

Out[119]:

	age_group	member_age
133194	30s	38
509995	20s	28
544051	20s	24
728821	30s	31
301696	30s	34

In [120]:
```
trips18_clean['age_group'].isna().any()
```

Out[120]:
```
False
```

Define 6
`bike_share_for_all_trip` column should be boolean

In [121]:
```
trips18_clean.bike_share_for_all_trip.replace({'Yes':True, 'No':False}, inplace=True)
```

Test 6

In [122]:
```
trips18_clean.bike_share_for_all_trip.dtypes
```

Out[122]:
```
dtype('bool')
```

Define 7
`user_type`, `member_gender`, `age_group`, `month`, `weekday` and `hour` should be categorical variables

Code 7

In [123]:
```
# Converted to categorical
trips18_clean['user_type']=trips18_clean['user_type'].astype('category')
trips18_clean['member_gender']=trips18_clean['member_gender'].astype('category')
trips18_clean['age_group']=trips18_clean['age_group'].astype('category')
trips18_clean['month']=trips18_clean['month'].astype('category')
trips18_clean['weekday']=trips18_clean['weekday'].astype('category')
trips18_clean['hour']=trips18_clean['hour'].astype('category')
trips18_clean['start_metro_area']=trips18_clean['start_metro_area'].astype('category')
trips18_clean['end_metro_area']=trips18_clean['end_metro_area'].astype('category')
```

Test 7

In [124]:
```
trips18_clean.dtypes
```

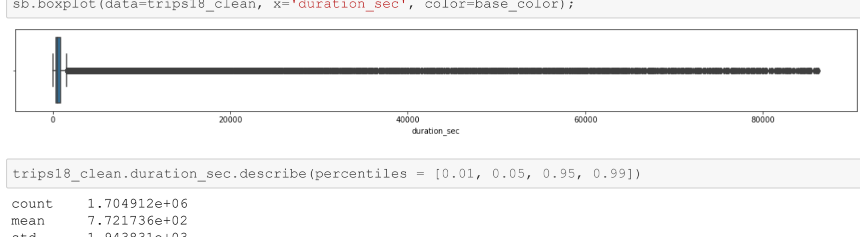
Out[124]:
```
duration_sec                        int64
start_time                 datetime64[ns]
end_time                   datetime64[ns]
start_station_id                    int32
start_station_name                 object
start_station_latitude            float64
start_station_longitude           float64
end_station_id                      int32
end_station_name                   object
end_station_latitude              float64
end_station_longitude             float64
bike_id                             int64
user_type                        category
member_birth_year                   int32
member_gender                    category
bike_share_for_all_trip              bool
start_metro_area                 category
end_metro_area                   category
month                            category
weekday                          category
hour                             category
member_age                          int64
age_group                        category
dtype: object
```

In [125]:
```
trips18_clean.member_gender.value_counts()
```

Out[125]:
```
Male      1249454
Female     429039
Other       26419
Name: member_gender, dtype: int64
```

Define 8
`duration_sec` has many large values maybe outlier due to customers forgot to log off after using

In [126]:
```
# Plot duration distibution
plt.figure(figsize = [20, 2])
base_color = sb.color_palette()[0]
sb.boxplot(data=trips18_clean, x='duration_sec', color=base_color);
```



In [127]:
```
trips18_clean.duration_sec.describe(percentiles = [0.01, 0.05, 0.95, 0.99])
```

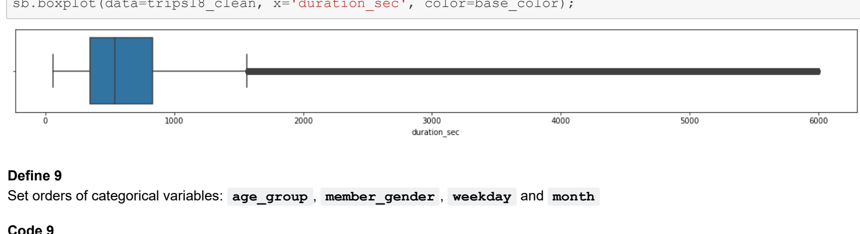
Out[127]:
```
count    1.704912e+06
mean     7.721736e+02
std      1.943831e+03
min      6.100000e+01
1%       1.060000e+02
5%       1.760000e+02
50%      5.420000e+02
95%      1.642000e+03
99%      3.858000e+03
max      8.628100e+04
Name: duration_sec, dtype: float64
```

Code 8

In [128]:
```
# Remove duration outliers
trips18_clean = trips18_clean.query('duration_sec <= 6000')
```

Test 8

In [129]:
```
# Plot duration < 6000 sec distibution
plt.figure(figsize = [20, 2])
base_color = sb.color_palette()[0]
sb.boxplot(data=trips18_clean, x='duration_sec', color=base_color);
```



Define 9
Set orders of categorical variables: `age_group`, `member_gender`, `weekday` and `month`

Code 9

In [130]:
```
# Set order of each categorical variable

# age_group
trips18_clean['age_group'] = pd.Categorical(trips18_clean['age_group'], ['10s', '20s', '30s', '40s', '50s'])
# member_gender
trips18_clean['member_gender'] = pd.Categorical(trips18_clean['member_gender'], ['Male','Female','Other'])
# weekday
trips18_clean['weekday'] = pd.Categorical(trips18_clean['weekday'], ['Monday','Tuesday','Wednesday','Thursday','Friday','Saturday','Sunday'])
# month
trips18_clean['month'] = pd.Categorical(trips18_clean['month'], [6,7,8,9,10,11,12,1,2,3,4,5])
```

Test 9

In [131]:
```
trips18_clean.weekday.value_counts().index.categories
```

Out[131]:
```
Index(['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday',
       'Sunday'],
      dtype='object')
```

In [132]:
```
# Final df_clean
trips18_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1695316 entries, 6 to 1863719
Data columns (total 23 columns):
 #   Column                   Dtype
---  ------                   -----
 0   duration_sec             int64
 1   start_time               datetime64[ns]
 2   end_time                 datetime64[ns]
 3   start_station_id         int32
 4   start_station_name       object
 5   start_station_latitude   float64
 6   start_station_longitude  float64
 7   end_station_id           int32
 8   end_station_name         object
 9   end_station_latitude     float64
 10  end_station_longitude    float64
 11  bike_id                  int64
 12  user_type                category
 13  member_birth_year        int32
 14  member_gender            category
 15  bike_share_for_all_trip  bool
 16  start_metro_area         category
 17  end_metro_area           category
 18  month                    category
 19  weekday                  category
 20  hour                     category
 21  member_age               int64
 22  age_group                category
dtypes: bool(1), category(8), datetime64[ns](2), float64(4), int32(3), int64(3), object(2)
memory usage: 189.2+ MB
```

In [133]:
```
# trips18.info()
```

In [134]:
```
# # issue 2: add new columns for trip duration in minute, trip start date in yyyy-mm-dd format, trip s
tart hour of the day, day of week and month

trips18_clean['duration_minute'] = trips18_clean['duration_sec']/60

trips18_clean['start_date'] = trips18_clean.start_time.dt.strftime('%Y-%m-%d')
trips18_clean['start_hourofday'] = trips18_clean.start_time.dt.strftime('%H')
trips18_clean['start_dayofweek'] = trips18_clean.start_time.dt.strftime('%A')
trips18_clean['start_month'] = trips18_clean.start_time.dt.strftime('%B')

trips18_clean.head()
```

Out[134]:

	duration_sec	start_time	end_time	start_station_id	start_station_name	start_station_latitude	start_station_longitude	end_station_
6	453	2018-01-31 23:53:53.632	2018-02-01 00:01:26.805	110	17th & Folsom Street Park (17th St at Folsom St)	37.763708	-122.415204	1
7	180	2018-01-31 23:52:09.903	2018-01-31 23:55:10.807	81	Berry St at 4th St	37.775880	-122.393170	
8	996	2018-01-31 23:34:56.004	2018-01-31 23:51:32.674	134	Valencia St at 24th St	37.752428	-122.420628	
9	825	2018-01-31 23:34:14.027	2018-01-31 23:47:59.809	305	Ryland Park	37.342725	-121.895617	3
11	432	2018-01-31 23:34:26.484	2018-01-31 23:41:39.297	89	Division St at Potrero Ave	37.769218	-122.407646	

5 rows × 28 columns

In [135]:
```
# issue 3: add a new column calculating riders' age from 'member_birth_year'

trips18_clean['member_age'] = 2019 - trips18_clean['member_birth_year']
trips18_clean.describe()
```

Out[135]:

	duration_sec	start_station_id	start_station_latitude	start_station_longitude	end_station_id	end_station_latitude	end_station_longit
count	1.695316e+06	1.695316e+06	1.695316e+06	1.695316e+06	1.695316e+06	1.695316e+06	1.695316e
mean	6.733463e+02	1.204515e+02	3.776863e+01	-1.223512e+02	1.188763e+02	3.776873e+01	-1.223507e
std	5.418813e+02	1.001461e+02	1.016470e-01	1.194266e-01	1.001396e+02	1.014922e-01	1.188235e
min	6.100000e+01	3.000000e+00	3.726331e+01	-1.224737e+02	3.000000e+00	3.726331e+01	-1.224737e
25%	3.430000e+02	3.600000e+01	3.777106e+01	-1.224116e+02	3.000000e+01	3.777106e+01	-1.224094e
50%	5.400000e+02	8.900000e+01	3.778107e+01	-1.223974e+02	8.900000e+01	3.778107e+01	-1.223971e
75%	8.310000e+02	1.860000e+02	3.779625e+01	-1.222876e+02	1.860000e+02	3.779728e+01	-1.222894e
max	6.000000e+03	3.810000e+02	3.788022e+01	-1.218333e+02	3.810000e+02	3.788022e+01	-1.218333e

In [ ]:

In [136]:
```
# # issue 4: filter out outlier ages from visually examination of the distribution above
# # issue 5: cast 'member_birth_year' and 'member_age' to integer instead of float type

trips18_clean = trips18_clean.query('member_age <= 70')
trips18_clean['member_birth_year'] = trips18_clean['member_birth_year'].astype('int')
trips18_clean['member_age'] = trips18_clean['member_age'].astype('int')
trips18_clean.info(null_counts=True)
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1695316 entries, 6 to 1863719
Data columns (total 28 columns):
 #   Column                   Non-Null Count    Dtype
---  ------                   --------------    -----
 0   duration_sec             1695316 non-null  int64
 1   start_time               1695316 non-null  datetime64[ns]
 2   end_time                 1695316 non-null  datetime64[ns]
 3   start_station_id         1695316 non-null  int32
 4   start_station_name       1695316 non-null  object
 5   start_station_latitude   1695316 non-null  float64
 6   start_station_longitude  1695316 non-null  float64
 7   end_station_id           1695316 non-null  int32
 8   end_station_name         1695316 non-null  object
 9   end_station_latitude     1695316 non-null  float64
 10  end_station_longitude    1695316 non-null  float64
 11  bike_id                  1695316 non-null  int64
 12  user_type                1695316 non-null  category
 13  member_birth_year        1695316 non-null  int32
 14  member_gender            1695316 non-null  category
 15  bike_share_for_all_trip  1695316 non-null  bool
 16  start_metro_area         1695316 non-null  category
 17  end_metro_area           1695316 non-null  category
 18  month                    1695316 non-null  category
 19  weekday                  0 non-null        category
 20  hour                     1695316 non-null  category
 21  member_age               1695316 non-null  int32
 22  age_group                1695316 non-null  category
 23  duration_minute          1695316 non-null  float64
 24  start_date               1695316 non-null  object
 25  start_hourofday          1695316 non-null  object
 26  start_dayofweek          1695316 non-null  object
 27  start_month              1695316 non-null  object
dtypes: bool(1), category(8), datetime64[ns](2), float64(5), int32(4), int64(2), object(6)
memory usage: 247.4+ MB
```

What is the structure of your dataset?

The original combined data contains approximately 1,863,719 individual trip records with 16 variables collected. These trips depend on various factors that is Duration of trip, Start Date, Time and End Date time, of the trip, User Type etc.

Main features of my interest in the Dataset :

I'm most interested in figuring out what features are best for predicting the more number of bike trips in the dataset.

Features in the dataset which will help me support our investigation :

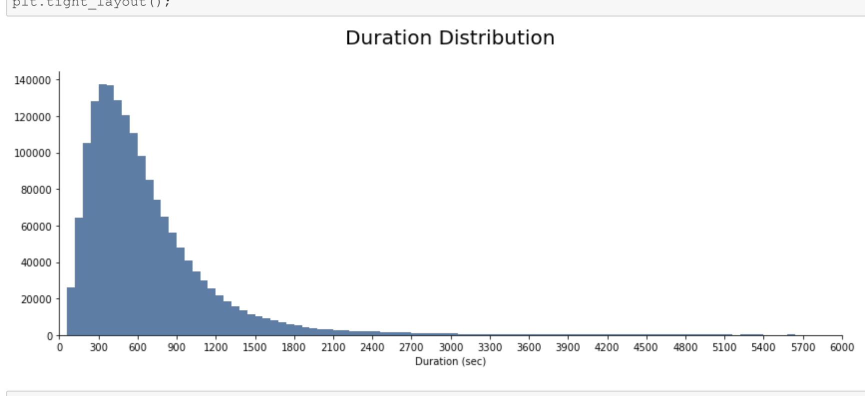
According to the dataset, the features like Start Date, Time, Month, Duration of Trip and User Type will have the strongest effect on Bike Trips.

Univariate Exploration

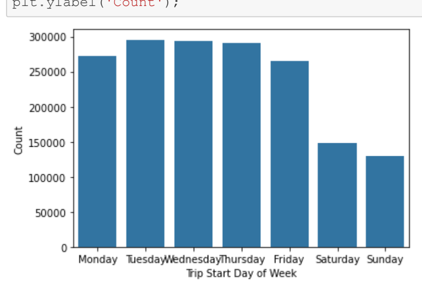
A series of plots to first explore the trips distribution over hour-of-day, day-of-week and month.

Duration Distribution

In [137]:
```
# Set bin size and color
bin_size = 60
bins = np.arange(0, trips18_clean.duration_sec.max()+bin_size, bin_size)
color = sb.color_palette('viridis')[1]

# Plotting
fig, axes = plt.subplots(figsize = (12,5))
plt.hist(trips18_clean.duration_sec, bins = bins, color= color, alpha=0.8);

# Aesthetic wrangling
plt.xticks(ticks = [x for x in range(0,6001,300)])
plt.title('Duration Distribution\n', size=20)
plt.xlabel('Duration (sec)')
plt.xlim(0,6000)
sb.despine(fig)
plt.tight_layout();
```

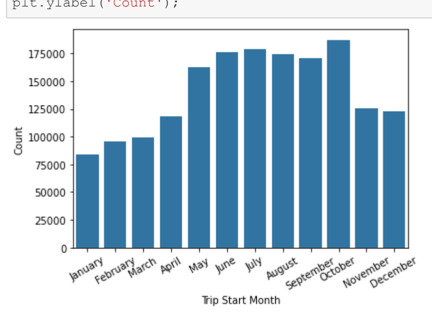


In [138]:
```
# trip distribution over weekdays

weekday = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
weekdaycat = pd.api.types.CategoricalDtype(ordered=True, categories=weekday)
trips18_clean['start_dayofweek'] = trips18_clean['start_dayofweek'].astype(weekdaycat)

sb.countplot(data=trips18_clean, x='start_dayofweek', color=base_color);
plt.xlabel('Trip Start Day of Week');
plt.ylabel('Count');
```



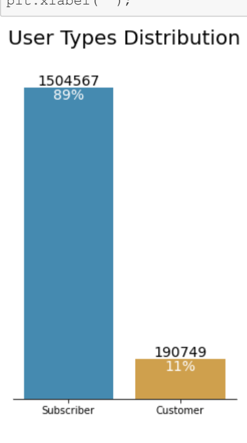
> Less trips are taken on the weekends as compared to weekdays.

In [139]:
```
# trip distribution over months

month = ['January', 'February', 'March', 'April', 'May', 'June', 'July', 'August', 'September', 'Octob
er', 'November', 'December']
monthcat = pd.api.types.CategoricalDtype(ordered=True, categories=month)
trips18_clean['start_month'] = trips18_clean['start_month'].astype(monthcat)

sb.countplot(data=trips18_clean, x='start_month', color=base_color);
plt.xticks(rotation=30);
plt.xlabel('Trip Start Month');
plt.ylabel('Count');
```



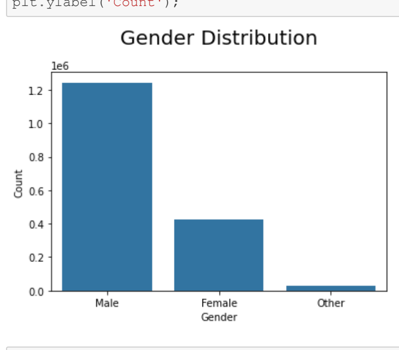
Looking in combined with the trip distribution over day of week plot, it is quite obvious that the majority of rides happened on work days (Mon-Fri) and primary usage is probably for commuting. In the 12 months in 2018, October had the most ride trips compared to the others, but overall it was the most popular during summer time (May-Sept), probably due to the weather in the area.

In [140]:
```
# Plotting
fig, ax = plt.subplots(figsize = (4,6))
#color = sns.color_palette('colorblind')[10]
sb.countplot(x = "user_type", data = trips18_clean,
             order = trips18_clean['user_type'].value_counts().index,
             palette = sb.color_palette('colorblind'), alpha=0.8)
# Percentage for each age-group
perc_list = []

# Calculate % for 2 user types for each age_group
type_sum = trips18_clean['user_type'].value_counts().to_list()
total_sum = trips18_clean.shape[0]
# arrange the % list in same as annotate loop order
for i in range(0,len(type_sum)):
    percent = int(round(100 * type_sum[i] / total_sum))
    perc_list.append(percent)
# Annotate each bar
i=0
for p in ax.patches:
    ax.annotate('{:.0f}'.format(p.get_height()),
                (p.get_x()+p.get_width()/2, p.get_height()),
                ha="center", va="bottom", size=14)
    ax.annotate('{:.0f}%'.format(perc_list[i]),
                (p.get_x()+p.get_width()/2, p.get_height()),
                ha="center", va="top", color='white', size=14)
    i+=1

# Aesthetic wrangling
plt.title('User Types Distribution\n', size=20)
cur_axes = plt.gca()
cur_axes.axes.get_yaxis().set_visible(False)
sb.despine(fig, left = True)
plt.xlabel("");
```

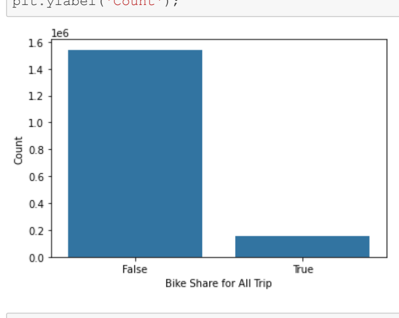


> From graph above, **most of the users are members which is about 7.6 times as many as casual users**.

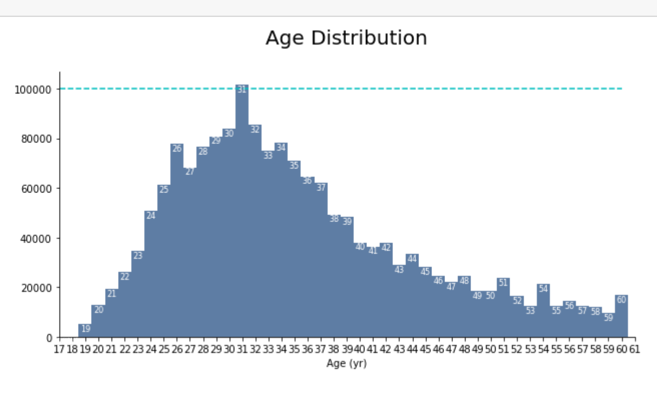
In [141]:
```
sb.countplot(data=trips18_clean, x='member_gender', color=base_color);
plt.title('Gender Distribution\n', size=20)
plt.xlabel('Gender');
plt.ylabel('Count');
```



In [142]:
```
sb.countplot(data=trips18_clean, x='bike_share_for_all_trip', color=base_color);
plt.xlabel('Bike Share for All Trip');
plt.ylabel('Count');
```



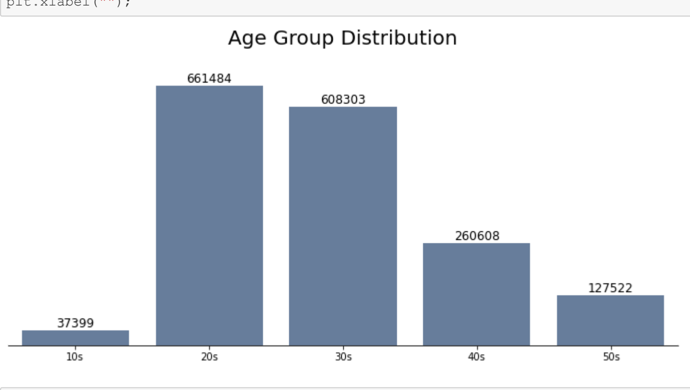
In [143]:
```
# Set bin size equal to 1 yr and color
bin_size = 1
bins = np.arange(0, trips18_clean.member_age.max()+bin_size, bin_size)
color = sb.color_palette('viridis')[1]

# Plotting
fig, ax = plt.subplots(figsize = (12,5))
plt.hist(trips18_clean.member_age, bins = bins, color= color, align="left", alpha=0.8)
for p in ax.patches:
    ax.annotate('{:.0f}'.format(p.get_x()+0.5),
                (p.get_x()+p.get_width()/2, p.get_height()),
                ha='center', va='top', color = "white", size=8)

# Aesthetic wrangling
plt.xticks(ticks = [x for x in range(0,70,1)])
plt.xlim((17,61))
plt.title('Age Distribution\n', size=20)
plt.xlabel('Age (yr)')
plt.hlines(100000, 0, 60, colors = "c", linestyles = "dashed")
sb.despine(fig)
plt.tight_layout();
```



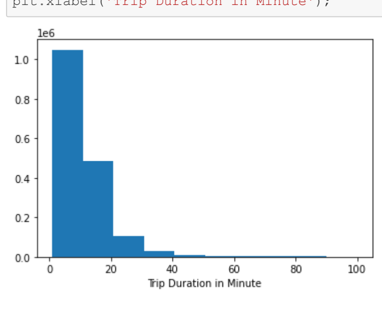
Most riders were male subscribers who did not use bike share for all trips. Most members were around 25 to 40 years old, corresponding to the trip distribution plots that most trips were on Mon-Fri and maxed during rush hours of a day. As the age gets older, bike usage dropped significantly.

Trip duration distribution to plot next.

In [144]:
```
# Plotting
fig, ax = plt.subplots(figsize = (12,5))
sb.countplot(x = "age_group", data = trips18_clean,
             color = sb.color_palette('viridis')[1],
             order = ['10s','20s','30s','40s','50s'],
             # order by age_group
             alpha=0.8)

# Aesthetic wrangling
for p in ax.patches:
    ax.annotate('{:.0f}'.format(p.get_height()),
                (p.get_x()+p.get_width()/2, p.get_height()),
                ha='center', va='bottom', color = "black", size=12)
plt.title('Age Group Distribution\n', size=20)
cur_axes = plt.gca()
cur_axes.axes.get_yaxis().set_visible(False)
sb.despine(fig, left = True)
plt.xlabel("");
```



In [145]:
```
plt.hist(data=trips18_clean, x='duration_minute');
plt.xlabel('Trip Duration in Minute');
```

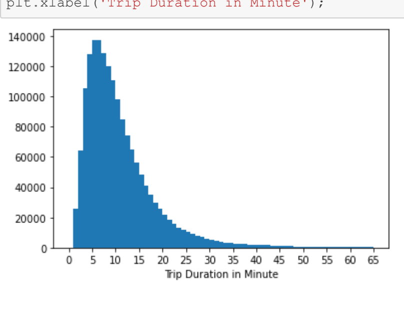


It was a little surprising to see the duration distribution spans pretty far to the right (indicates a long time rental, almost 24 hrs) while the majority of the rides were quite short (within 200 mins). Going to zoom in to the most cases for a closer picture of what the distribution is like.

In [146]:
```
trips18_clean['duration_minute'].describe(percentiles=[.99])
```

Out[146]:
```
count    1.695316e+06
mean     1.122244e+01
std      9.031355e+00
min      1.016667e+00
50%      9.000000e+00
99%      4.725000e+01
max      1.000000e+02
Name: duration_minute, dtype: float64
```

In [147]:
```
bins = np.arange(0, 66, 1)
ticks = np.arange(0, 66, 5)
plt.hist(data=trips18_clean, x='duration_minute', bins=bins);
plt.xticks(ticks, ticks);
plt.xlabel('Trip Duration in Minute');
```



It looks like 99% of the trips were less than 1 hour long with most of them fell between 5 to 10 minutes range. I'll remove the outlier long trips for this study to allow focusing on the most common cases to explore the pattern.

In [148]:
```
# issue 8: filter out outlier trip records where the duration was very long

trips18_clean = trips18_clean.query('duration_minute <= 66')
trips18_clean.info(null_counts=True)
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1688410 entries, 6 to 1863719
Data columns (total 28 columns):
 #   Column                   Non-Null Count    Dtype
---  ------                   --------------    -----
 0   duration_sec             1688410 non-null  int64
 1   start_time               1688410 non-null  datetime64[ns]
 2   end_time                 1688410 non-null  datetime64[ns]
 3   start_station_id         1688410 non-null  int32
 4   start_station_name       1688410 non-null  object
 5   start_station_latitude   1688410 non-null  float64
 6   start_station_longitude  1688410 non-null  float64
 7   end_station_id           1688410 non-null  int32
 8   end_station_name         1688410 non-null  object
 9   end_station_latitude     1688410 non-null  float64
 10  end_station_longitude    1688410 non-null  float64
 11  bike_id                  1688410 non-null  int64
 12  user_type                1688410 non-null  category
 13  member_birth_year        1688410 non-null  int32
 14  member_gender            1688410 non-null  category
 15  bike_share_for_all_trip  1688410 non-null  bool
 16  start_metro_area         1688410 non-null  category
 17  end_metro_area           1688410 non-null  category
 18  month                    1688410 non-null  category
 19  weekday                  0 non-null        category
 20  hour                     1688410 non-null  category
 21  member_age               1688410 non-null  int32
 22  age_group                1688410 non-null  category
 23  duration_minute          1688410 non-null  float64
 24  start_date               1688410 non-null  object
 25  start_hourofday          1688410 non-null  object
 26  start_dayofweek          1688410 non-null  category
 27  start_month              1688410 non-null  category
dtypes: bool(1), category(10), datetime64[ns](2), float64(5), int32(4), int64(2), object(4)
memory usage: 223.8+ MB
```

In [149]:
```
# Remove column name 'month' and weekday
trips18_clean = trips18_clean.drop(['month'], axis = 1)
trips18_clean = trips18_clean.drop(['weekday'], axis = 1)
```

In [150]:
```
trips18_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1688410 entries, 6 to 1863719
Data columns (total 26 columns):
 #   Column                   Non-Null Count    Dtype
---  ------                   --------------    -----
 0   duration_sec             1688410 non-null  int64
 1   start_time               1688410 non-null  datetime64[ns]
 2   end_time                 1688410 non-null  datetime64[ns]
 3   start_station_id         1688410 non-null  int32
 4   start_station_name       1688410 non-null  object
 5   start_station_latitude   1688410 non-null  float64
 6   start_station_longitude  1688410 non-null  float64
 7   end_station_id           1688410 non-null  int32
 8   end_station_name         1688410 non-null  object
 9   end_station_latitude     1688410 non-null  float64
 10  end_station_longitude    1688410 non-null  float64
 11  bike_id                  1688410 non-null  int64
 12  user_type                1688410 non-null  category
 13  member_birth_year        1688410 non-null  int32
 14  member_gender            1688410 non-null  category
 15  bike_share_for_all_trip  1688410 non-null  bool
 16  start_metro_area         1688410 non-null  category
 17  end_metro_area           1688410 non-null  category
 18  hour                     1688410 non-null  category
 19  member_age               1688410 non-null  int32
 20  age_group                1688410 non-null  category
 21  duration_minute          1688410 non-null  float64
 22  start_date               1688410 non-null  object
 23  start_hourofday          1688410 non-null  object
 24  start_dayofweek          1688410 non-null  category
 25  start_month              1688410 non-null  category
dtypes: bool(1), category(8), datetime64[ns](2), float64(5), int32(4), int64(2), object(4)
memory usage: 220.6+ MB
```

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

There were more trips on work days (Mon-Fri) compared to weekends. Summar time was the most popular season of a year, likely due to the weather.

User-wise, there were more male riders than female, and most members were subscribers compared to casual riders. The majority of the members did not use bike share for all of their trips, and most were around 25 to 40 years old.

Most rides were quick and short, lasted between 5 to 10 minutes, though there were some very long outliers like 24hrs.

No transformation was needed luckily due to the straightforwardness of the data.

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?

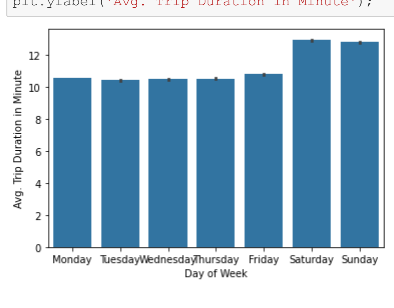
> 1. From duration graph, we can see the most rides takes within 5-10 minutes (300-600 secs). Durations dramaticlly higher are outliers probably due to user forgot to log out after finish their ride.
> 2. From age graph, we can see people from 27 to 31 years old use the GoBike over 100,000 times. Customers could have given the wrong birth year information that cause the super old age data outliers.
> 3. From gender graph, male users are about 3 times as many as female users.
> 4. From user_type graph, most of the users are members which is about 7.6 times as many as casual users.
> 5. From month graph, March and April have high usage over 200,000 and November and December's usage all below 130,000. This probably due to weather reasons. Hot summer season and cold winter season prevent users from riding. I also suspect holiday season could be reason for the dramatic dropNovember and December, and school summer vacations could be reason for the drop in May.
> 6. From weekday graph, weekday usage is higher than weekend usage by almost 2 times. From hour graph, there are two peaks of traffic during the day around 8-9am and 5-6pm. The times correspond to the morning and evening rush hours so I believe people go to and leave from work are using the bike to commute between home, office or Bart(connect to other transportations).
> 7. From Most Popular Starting/Ending Stations graph, most popular starting stations are in SF area. And the fact that popular GoBike stations are near Bart or Caltrain stations confirms the idea people use bike to commute between public transportation and work/home.
> 8. From Area Distribution for Bike Activity graph, most bike riding activities happened in San Francisco area. This is make sense because SF is the first location of GoBike. I am curious about if there is any ride that starting from one area but end in another area.
> 9. Inter-area Bike Activity graph, among the 2114592 bike activity in df_clean, only 65 inter-area rides. All of the rides are between SF and oakland across the Bay Bridge.

In [151]:
```
# save the clean data to a .csv file

trips18_clean.to_csv('fordgobike_trips_2018_clean.csv', index=False)
```

Bivariate Exploration

Average Trip Duration on Weekdays

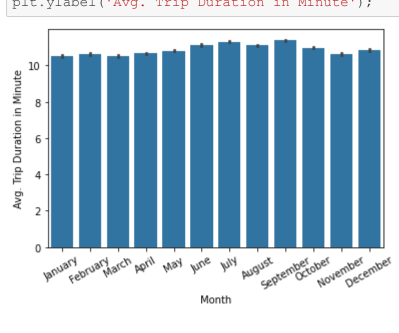
In [152]:
```
sb.barplot(data=trips18_clean, x='start_dayofweek', y='duration_minute', color=base_color);
plt.xlabel('Day of Week');
plt.ylabel('Avg. Trip Duration in Minute');
```



The riding trips are much shorter on Monday through Friday compared to weekends. It indicates a pretty stable and efficient usage of the sharing system on normal work days, while more casual flexible use on weekends.

Average trip duration by month

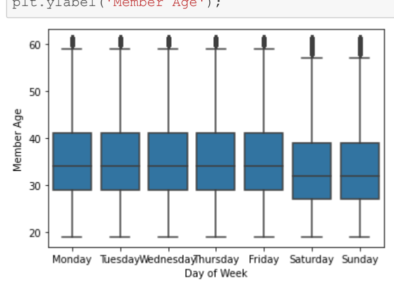
In [153]:
```
sb.barplot(data=trips18_clean, x='start_month', y='duration_minute', color=base_color);
plt.xticks(rotation=30);
plt.xlabel('Month');
plt.ylabel('Avg. Trip Duration in Minute');
```



Starting from March, the average usage time shown an increasing trend almost all the way til October before it down turned. The average longest trips happened during the summer months, around June, July and September, which probably has a lot to do with the weather in the area.

Member age by weekdays

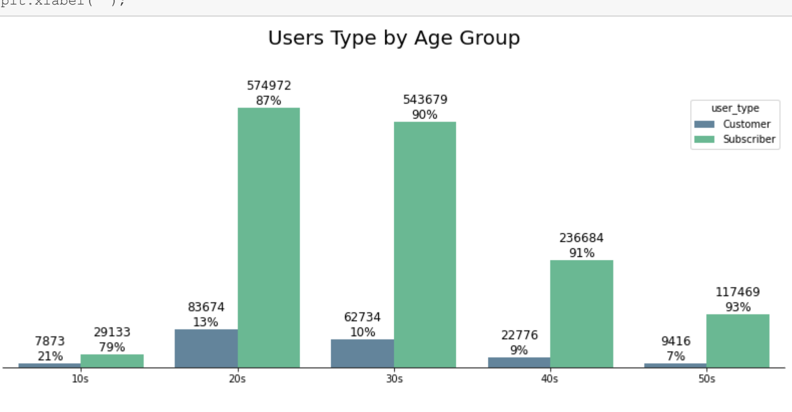
In [154]:
```
sb.boxplot(data=trips18_clean, x='start_dayofweek', y='member_age', color=base_color);
plt.xlabel('Day of Week');
plt.ylabel('Member Age');
```



Riders who rented the bikes Monday through Friday are slightly older than those who ride on weekends, which corresponds to the commute to work usage that was observed from the above univariable exploration plots.

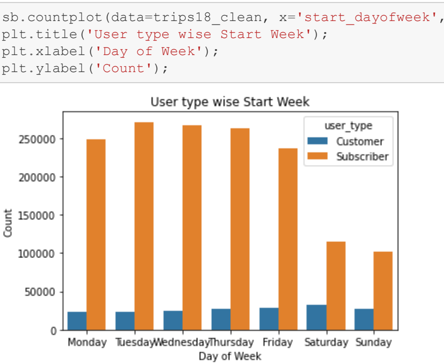
Weekly usage between customers and subscribers

In [155]:
```
fig, ax = plt.subplots(figsize = (14,5))

# Plotting
sb.countplot(x = "age_group", data = trips18_clean,
             palette = "viridis", hue = "user_type", alpha = 0.8)

# Percentage for each age-group
perc_list_customer, perc_list_subscriber, perc_list = [], [], []

# Calculate % for 2 user types for each age_group
type_sum = trips18_clean.groupby('age_group')['user_type'].value_counts().sort_index().to_list()
total_sum = trips18_clean['age_group'].value_counts().sort_index().to_list()

# arrange the % list in same as annotate loop order
for i in range(0,len(total_sum)):
    perc_customer = int(round(100 * type_sum[2*i] / total_sum[i]))
    perc_list_customer.append(perc_customer)
for i in range(0,len(total_sum)):
    perc_subscriber = int(round(100 * type_sum[2*i+1]/ total_sum[i]))
    perc_list_subscriber.append(perc_subscriber)
perc_list = perc_list_customer + perc_list_subscriber

# Annotate each bar
i=0
for p in ax.patches:
    ax.annotate('{:.0f}\n{:.0f}%'.format(p.get_height(), perc_list[i]),
                (p.get_x()+p.get_width()/2, p.get_height()),
                ha="center", va="bottom", size=12)
    i+=1

# Aesthetic wrangling
cur_axes = plt.gca()
cur_axes.axes.get_yaxis().set_visible(False)
sb.despine(fig, left = True)
plt.title("Users Type by Age Group\n\n", fontsize= 20)
plt.xlabel("");
```



> From graph above, user's subscription ratio increases as the age increase. Younger user tend to use the service but do not subscribe.

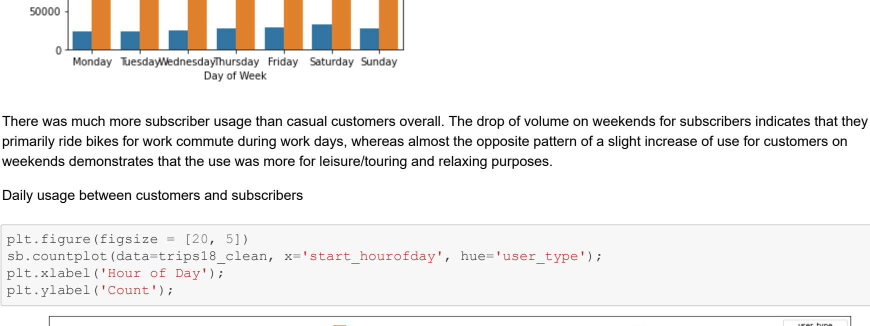
In [156]:
```
sb.countplot(data=trips18_clean, x='start_dayofweek', hue='user_type');
plt.title('User type wise Start Week');
plt.xlabel('Day of Week');
plt.ylabel('Count');
```



There was much more subscriber usage than casual customers overall. The drop of volume on weekends for subscribers indicates that they primarily ride bikes for work commute during work days, whereas almost the opposite pattern of a slight increase of use for customers on weekends demonstrates that the use was more for leisure/touring and relaxing purposes.

Daily usage between customers and subscribers

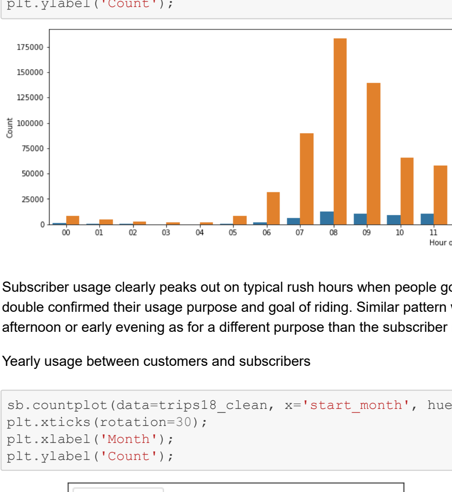
In [157]:
```
plt.figure(figsize = [20, 5])
sb.countplot(data=trips18_clean, x='start_hourofday', hue='user_type');
plt.xlabel('Hour of Day');
plt.ylabel('Count');
```



Subscriber usage clearly peaks out on typical rush hours when people go to work in the morning and getting off work in the afternoon, double confirmed their usage purpose and goal of riding. Similar pattern was not observed among customers who tend to ride most in the afternoon or early evening as for a different purpose than the subscriber riders.

Yearly usage between customers and subscribers

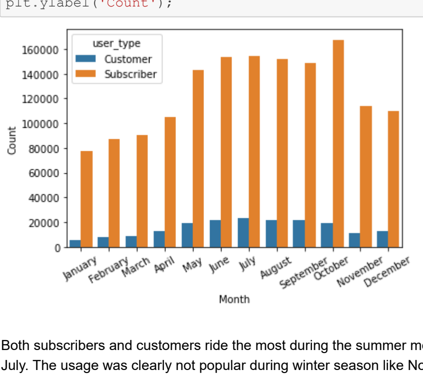
In [158]:
```
sb.countplot(data=trips18_clean, x='start_month', hue='user_type');
plt.xticks(rotation=30);
plt.xlabel('Month');
plt.ylabel('Count');
```



Both subscribers and customers ride the most during the summer months with subscribers maxed out in October and customers peaked in July. The usage was clearly not popular during winter season like November, December and January likely due to the weather.

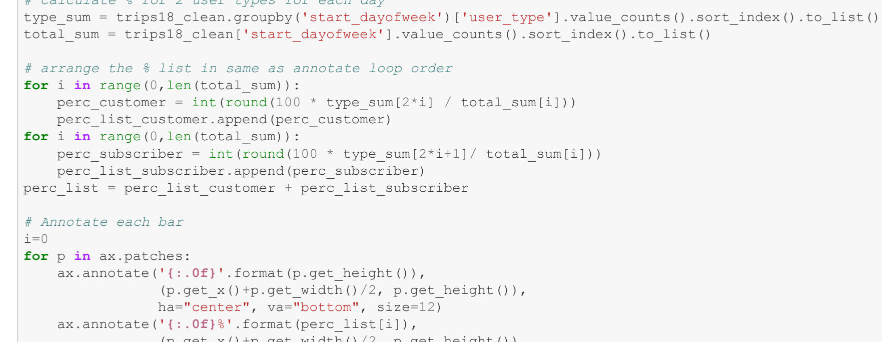
Member age between customers and subscribers

In [159]:
```
sb.boxplot(data=trips18_clean, x='user_type', y='member_age', color=base_color);
plt.xlabel('User Type');
plt.ylabel('Member Age');
```



Similar to the Member age by weekdays plot, subscribers who ride most often Monday through Friday are slightly older than customers, with a wider range of ages as well.
The trip duration distribution is much narrower for subscribers compared to casual riders on the shorter/quicker trip end overall. It seems like subscribers have a more specific usage or targeted goal riding the bikes compared to customers who vary more and generally rented the bikes for longer.

In [160]:
```
fig, ax = plt.subplots(figsize = (14,5))
# Plotting
sb.countplot(x = "start_dayofweek", data = trips18_clean,
             palette = "viridis", hue = "user_type", alpha = 0.8)

# Percentage for each day
perc_list_customer, perc_list_subscriber, perc_list = [], [], []

# Calculate % for 2 user types for each day
type_sum = trips18_clean.groupby('start_dayofweek')['user_type'].value_counts().sort_index().to_list()
total_sum = trips18_clean['start_dayofweek'].value_counts().sort_index().to_list()

# arrange the % list in same as annotate loop order
for i in range(0,len(total_sum)):
    perc_customer = int(round(100 * type_sum[2*i] / total_sum[i]))
    perc_list_customer.append(perc_customer)
for i in range(0,len(total_sum)):
    perc_subscriber = int(round(100 * type_sum[2*i+1]/ total_sum[i]))
    perc_list_subscriber.append(perc_subscriber)
perc_list = perc_list_customer + perc_list_subscriber

# Annotate each bar
i=0
for p in ax.patches:
    ax.annotate('{:.0f}'.format(p.get_height()),
                (p.get_x()+p.get_width()/2, p.get_height()),
                ha="center", va="bottom", size=12)
    ax.annotate('{:.0f}%'.format(perc_list[i]),
                (p.get_x()+p.get_width()/2, p.get_height()),
                ha="center", va="top", color='white', size=12)
    i+=1

# Aesthetic wrangling
cur_axes = plt.gca()
cur_axes.axes.get_yaxis().set_visible(False)
sb.despine(fig, left = True)
plt.title("Users Type by Weekday\n\n", fontsize= 20)
plt.xlabel("")
plt.tight_layout();
```



> From graph above, Saturday and Sunday user populations have unusual low subscription ratio and total rides. This is probably due to people use the bike for work mostly.

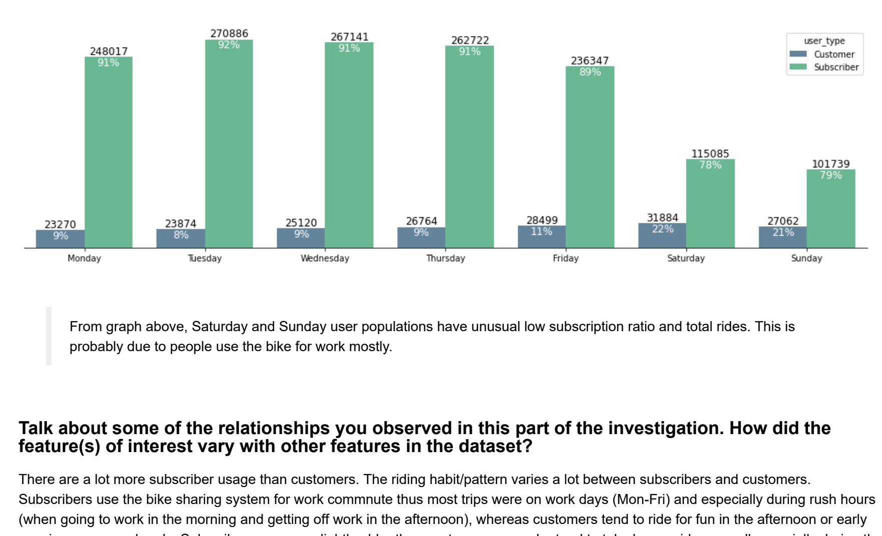
Talk about some of the relationships you observed in this part of the investigation. How did the feature(s) of interest vary with other features in the dataset?

There are a lot more subscriber usage than customers. The riding habit/pattern varies a lot between subscribers and customers. Subscribers use the bike sharing system for work commnute thus most trips were on work days (Mon-Fri) and especially during rush hours (when going to work in the morning and getting off work in the afternoon), whereas customers tend to ride for fun in the afternoon or early evenings over weekends. Subscriber users are slightly older than customer users who tend to take longer rides overall especially during the summer.

Did you observe any interesting relationships between the other features (not the main feature(s) of interest)?

It is interesting to see that subscribers are slightly older than customers on average but take much shorter/quicker rides. And October had the most bike riding usage for subscribers throughout the year.

In [161]:
```
user_type_count = trips18_clean.groupby(["start_month", "user_type"]).size().reset_index()
plt.figure(figsize=(15,6))
color = {'Subscriber':'orange', 'Customer':'black'}
axis = sb.pointplot(x='start_month', y=0, hue='user_type', palette=color, scale=.7, data=user_type_cou
nt)
plt.title('The monthly bike rides per user type')
plt.xlabel('Date')
plt.ylabel('Count')
plt.legend();
```

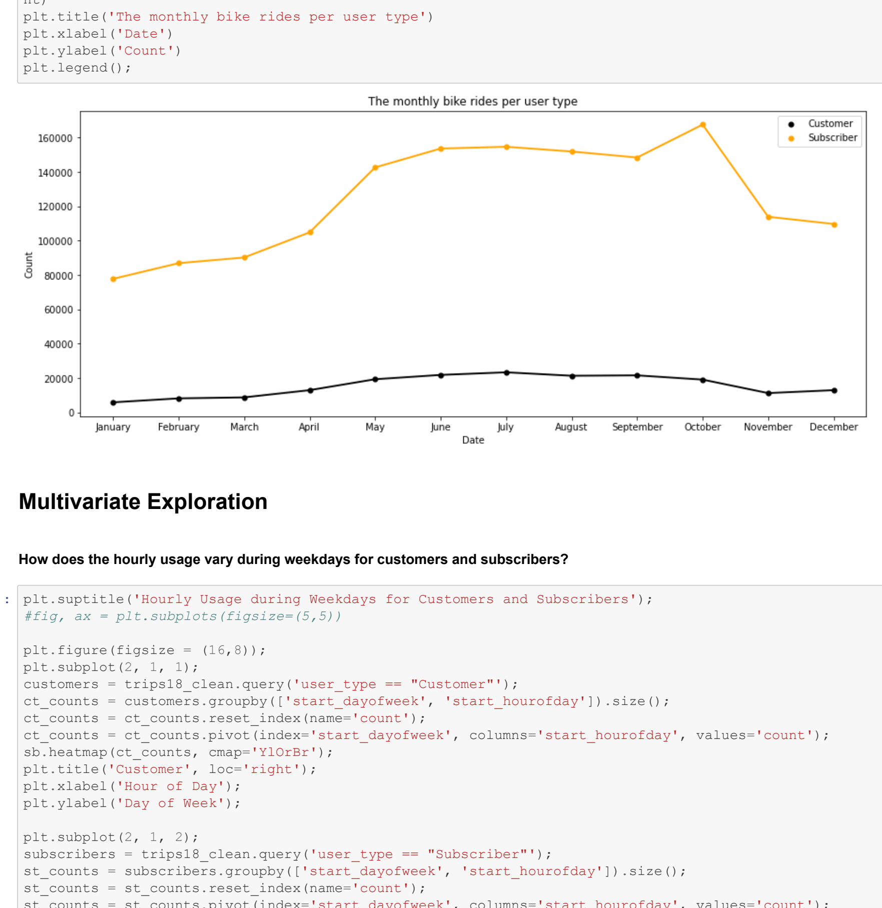


Multivariate Exploration

How does the hourly usage vary during weekdays for customers and subscribers?

In [162]:
```
plt.suptitle('Hourly Usage during Weekdays for Customers and Subscribers');
#fig, ax = plt.subplots(figsize=(5,5))

plt.figure(figsize = (16,8));
plt.subplot(2, 1, 1);
customers = trips18_clean.query('user_type == "Customer"');
ct_counts = customers.groupby(['start_dayofweek', 'start_hourofday']).size();
ct_counts = ct_counts.reset_index(name='count');
ct_counts = ct_counts.pivot(index='start_dayofweek', columns='start_hourofday', values='count');
sb.heatmap(ct_counts, cmap='YlOrBr');
plt.title('Customer', loc='right');
plt.xlabel('Hour of Day');
plt.ylabel('Day of Week');

plt.subplot(2, 1, 2);
subscribers = trips18_clean.query('user_type == "Subscriber"');
st_counts = subscribers.groupby(['start_dayofweek', 'start_hourofday']).size();
st_counts = st_counts.reset_index(name='count');
st_counts = st_counts.pivot(index='start_dayofweek', columns='start_hourofday', values='count');
sb.heatmap(st_counts, cmap='YlOrRd');
plt.title('Subscriber', loc='right');
plt.xlabel('Hour of Day');
plt.ylabel('Day of Week');
```

<Figure size 432x288 with 0 Axes>



By the above heat map, we can see clearly the trips taken by customers and subscribers on weekdays and weekends.

user_type for age_group by metro_area

In [163]:
```
# Since there's only three subplots to create, using the full data should be fine.
fig = plt.figure(figsize = [16, 12])

# Subplot 1: San Francisco, ride count for age_group by user_type
###############################################################
ax1 = plt.subplot(3, 1, 1)
ax1.set_title("San Francisco\n", size=18)
sb.countplot(data = trips18_clean.query('start_metro_area == "San Francisco"'),
             x = 'age_group', hue = 'user_type', palette = 'Set2', alpha=0.8)
plt.xlabel('')
plt.ylabel('')
cur_axes = plt.gca()
cur_axes.axes.get_yaxis().set_visible(False)
ax1.legend(loc = 1, ncol = 2)

# Percentage for each age_group
perc_list_customer, perc_list_subscriber, perc_list = [], [], []

# Calculate % for 2 user types for each age_group
type_sum = trips18_clean.query('start_metro_area == "San Francisco"').groupby('age_group')['user_type'
].value_counts().sort_index().to_list()
total_sum = trips18_clean.query('start_metro_area == "San Francisco"')['age_group'].value_counts().sor
t_index().to_list()

# Arrange the % list in same as annotate loop order
for i in range(0,len(total_sum)):
    perc_customer = int(round(100 * type_sum[2*i] / total_sum[i]))
    perc_list_customer.append(perc_customer)
for i in range(0,len(total_sum)):
    perc_subscriber = int(round(100 * type_sum[2*i+1]/ total_sum[i]))
    perc_list_subscriber.append(perc_subscriber)
perc_list = perc_list_customer + perc_list_subscriber

# Add annotations
i=0
for p in ax1.patches:
    ax1.annotate('{:.0f}\n{:.0f}%'.format(p.get_height(), perc_list[i]),
                (p.get_x()+p.get_width()/2, p.get_height()),
                ha="center", va="bottom", size=12)
    i+=1

# Subplot 2: East Bay, ride count for age_group by user_type
###############################################################
ax2 = plt.subplot(3, 1, 2)
ax2.set_title("East Bay", size=18)
sb.countplot(data = trips18_clean.query('start_metro_area == "East Bay"'),
             x = 'age_group', hue = 'user_type', palette = "Set2", alpha=0.8)
plt.xlabel('')
plt.ylabel('')
cur_axes = plt.gca()
cur_axes.axes.get_yaxis().set_visible(False)
ax2.legend(loc = 1, ncol = 2)

# Percentage for each age_group
perc_list_customer, perc_list_subscriber, perc_list = [], [], []

# Calculate % for 2 user types for each age_group
type_sum = trips18_clean.query('start_metro_area == "East Bay"').groupby('age_group')['user_type'].val
ue_counts().sort_index().to_list()
total_sum = trips18_clean.query('start_metro_area == "East Bay"')['age_group'].value_counts().sort_ind
ex().to_list()

# Arrange the % list in same as annotate loop order
for i in range(0,len(total_sum)):
    perc_customer = int(round(100 * type_sum[2*i] / total_sum[i]))
    perc_list_customer.append(perc_customer)
for i in range(0,len(total_sum)):
    perc_subscriber = int(round(100 * type_sum[2*i+1]/ total_sum[i]))
    perc_list_subscriber.append(perc_subscriber)
perc_list = perc_list_customer + perc_list_subscriber

# Add annotations
i=0
for p in ax2.patches:
    ax2.annotate('{:.0f}\n{:.0f}%'.format(p.get_height(), perc_list[i]),
                (p.get_x()+p.get_width()/2, p.get_height()),
                ha="center", va="bottom", size=12)
    i+=1

# Subplot 3: East Bay, ride count for age_group by user_type
###############################################################
ax3 = plt.subplot(3, 1, 3)
ax3.set_title("San Jose", size=18)
sb.countplot(data = trips18_clean.query('start_metro_area == "San Jose"'),
             x = 'age_group', hue = 'user_type', palette = 'Set2', alpha=0.8)
plt.xlabel('')
plt.ylabel('')
cur_axes = plt.gca()
cur_axes.axes.get_yaxis().set_visible(False)
ax3.legend(loc = 1, ncol = 2)
# Percentage for each age_group
perc_list_customer, perc_list_subscriber, perc_list = [], [], []

# Calculate % for 2 user types for each age_group
type_sum = trips18_clean.query('start_metro_area == "San Jose"').groupby('age_group')['user_type'].val
ue_counts().sort_index().to_list()
total_sum = trips18_clean.query('start_metro_area == "San Jose"')['age_group'].value_counts().sort_ind
ex().to_list()

# Arrange the % list in same as annotate loop order
for i in range(0,len(total_sum)):
    perc_customer = int(round(100 * type_sum[2*i] / total_sum[i]))
    perc_list_customer.append(perc_customer)
for i in range(0,len(total_sum)):
    perc_subscriber = int(round(100 * type_sum[2*i+1]/ total_sum[i]))
    perc_list_subscriber.append(perc_subscriber)
perc_list = perc_list_customer + perc_list_subscriber

# Add annotations
i=0
for p in ax3.patches:
    ax3.annotate('{:.0f}\n{:.0f}%'.format(p.get_height(), perc_list[i]),
                (p.get_x()+p.get_width()/2, p.get_height()),
                ha="center", va="bottom", size=12)
    i+=1

# Aesthetic Wrangling
fig.suptitle('User Type for Age Group by Metro Area', size=20, y=1.02)
sb.despine(fig, left = True)
plt.tight_layout();
```

> From graph above, majority of users is from age groups 20s and 30s. Although San Francisco metro area has most rides but the two largest user populations 20s and 30s age groups have the lowest subscription ratio.

Conclusion

The relationship between the multiple variables plotted are visualized altogether and information are presented. The short period of usage for subscribers corresponds to their high concentration on rush hours Monday through Friday,which is for for work. The customer use shows that they're taking advantage of the bike sharing system quite differently from the subscribers, heavily over weekends and in the afternoon, for city tour or leisure purpose mostly.

The interactions between features are all supplementing each other and quite make sense when looked at combined, there's no big surprise observed.

- We observed both users prefer to use a bike to go to work and to leave work and also in the afternoon they use bikes to go to restaurants or cafes.
- It has observed that on weekdays the usage level is higher than at weekends for subscribers but customer usage shows that approximately the same level all week.
- It observed that the average trip time is approximately 15 minutes.
- It has been observed customer usage trend increases towards the end of the year. On the other hand, subscriber usage trend gets the least level in the whole year.
- It can be seen that subscribers use bikes for a short time period than customers.
- It does not seem that any regular trends in a year. Subscriber usage seems that they are the least ride at the end of the year this situation maybe because of the Christmas holiday, they have a holiday and do not have to go to work. On the other hand, customer usage shows that the highest level in December for them.
- It can be seen that subscribers ride much shorter/quicker trips compared to customers on each day of the week. Both user types have an obvious increase in trip duration on Saturdays and Sundays over weekends. Subscribers usage seems to maintain a very consistent average duration Monday through Friday.

In [ ]: