Analysis of a public bike share network using Python

J. Michael Shockley

5 March 2019

For a public bike share service to function smoothly, bikes and docks at which to anchor them need to be available at stations throughout the network. When popular stations run out of bikes or fill up with them, the functionality of the entire bike share network is decreased. Redistribution of bikes between stations alleviates this issue, yet the redistribution process is expensive and labor intensive. An important logistics problem arises out of predicting which stations that will gain or lose bicycles. A multitude of factors may influence the usage of a public bike share service, and these are no doubt interconnected: geographic location of a station, weather, time of day, day of the week, etc.

The goal of this project was to take several well-known data sets of a San Francisco Bay Area bike share company and examine the relationships between trip-by-trip usage data, daily weather conditions, and information about the stations themselves. A year's worth of data from a time period of 09/2014 to 08/2015 helped to build a model predicting hourly rate of change per station.

Although these are common data sets, all work below (methodology, code, analysis, conclusions) is my own work. This project was written in Python making use of Pandas, Numpy, Matplotlib, Cartopy, and Plotnine packages.

1 Extract raw data

Three files were obtained: trip_data.csv, weather_data.csv, and station_data.csv. The data is imported below as data frames and the .head() method shows the contents of each data set. The trip data file includes a unique trip ID, the start and end date/time of each trip, the corresponding start and end station of each trip, and the subscriber type. The weather data contains various metrics of the weather including total precipitation, cloud cover, wind direction, maximum/mean/mininum temperature, dew point, relative humidity, visibility, and wind speed, and stated weather events for each zip code in which the bike share system operates. The station data contains the station ID, name, geographic location, and number of docks for each bike station.

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

#Trip data imported as dataframe TD
TD=pd.read_csv(
    "trip_data.csv",
    header=0,
    skip_blank_lines=False,
    parse_dates=['Start Date','End Date'],
    date_parser=lambda d: pd.to_datetime(
        d, format="%d/%m/%Y %H:%M", errors="coerce"
    )
)
TD.head()
```

Out[1]:

	Trip ID	Start Date	Start Station	End Date	End Station	Subscriber Type
0	913460	2015-08-31 23:26:00	50	2015-08-31 23:39:00	70	Subscriber
1	913459	2015-08-31 23:11:00	31	2015-08-31 23:28:00	27	Subscriber
2	913455	2015-08-31 23:13:00	47	2015-08-31 23:18:00	64	Subscriber
3	913454	2015-08-31 23:10:00	10	2015-08-31 23:17:00	8	Subscriber
4	913453	2015-08-31 23:09:00	51	2015-08-31 23:22:00	60	Customer

In [2]:
```
#Weather data imported as dataframe WD
WD=pd.read_csv(
    "weather_data.csv",
    header=0,
    skip_blank_lines=False,
    parse_dates=['Date'],
    date_parser=lambda d: pd.to_datetime(
        d, format="%d/%m/%Y", errors="coerce"
    )
)
WD.head()
```

Out[2]:

	Date	Max TemperatureF	Mean TemperatureF	Min TemperatureF	Max Dew PointF	MeanDew PointF	Min DewpointF	Max Humidity	H
0	2014-09-01	83.0	70.0	57.0	58.0	56.0	52.0	86.0	
1	2014-09-02	72.0	66.0	60.0	58.0	57.0	55.0	84.0	
2	2014-09-03	76.0	69.0	61.0	57.0	56.0	55.0	84.0	
3	2014-09-04	74.0	68.0	61.0	57.0	57.0	56.0	84.0	
4	2014-09-05	72.0	66.0	60.0	57.0	56.0	54.0	84.0	

5 rows × 24 columns

In [3]:
```
#Station data imported as dataframe SD
SD=pd.read_csv(
    "station_data.csv",
    header=0,
    skip_blank_lines=False
)
SD.head()
```

Out[3]:

	Id	Name	Lat	Long	Dock Count	City
0	2	San Jose Diridon Caltrain Station	37.329732	-121.901782	27	San Jose
1	3	San Jose Civic Center	37.330698	-121.888979	15	San Jose
2	4	Santa Clara at Almaden	37.333988	-121.894902	11	San Jose
3	5	Adobe on Almaden	37.331415	-121.893200	19	San Jose
4	6	San Pedro Square	37.336721	-121.894074	15	San Jose

2 Create a single data set of bikes entering and leaving each station, per hour

The first step to solving the logistics problem of interest requires transforming the raw trip data into an hourly change in bike count per station. This is calculated by counting the bikes entering and leaving each station for each hour during the time period. The next step is to merge the data sets using commmonalities between the data sets as join keys. This will serve as a template for later incorporating weather data (section 3).

First, the trip data (TD) is modified by correcting for stations that changed station ID over the time period. The .apply() method is used for this purpose.

In [4]:
```
#Modify rows in trip data (TD) for stations that changed station ID over
the time period
REPLACE_MAP = {23:85, 25:86, 49:87, 69:88, 72:90, 89:90}
def replace_if_changed(station_id):
    # returns the value of the replacement, or inputted station ID if no
t present in the dict
    return REPLACE_MAP.get(station_id, station_id)

TD['Start Station'] = TD['Start Station'].apply(replace_if_changed)
TD['End Station'] = TD['End Station'].apply(replace_if_changed)
```

Next, new columns are added to the trip data (TD) containing date/time data truncated to the hour, creating date/hour combinations.

In [5]:
```
#Use .dt.floor method to truncate trip data to the hour.
TD['Start Date Truncated']=TD['Start Date'].dt.floor('h')
TD['End Date Truncated']=TD['End Date'].dt.floor('h')
TD.head()
```

Out[5]:

	Trip ID	Start Date	Start Station	End Date	End Station	Subscriber Type	Start Date Truncated	End Date Truncated
0	913460	2015-08-31 23:26:00	50	2015-08-31 23:39:00	70	Subscriber	2015-08-31 23:00:00	2015-08-31 23:00:00
1	913459	2015-08-31 23:11:00	31	2015-08-31 23:28:00	27	Subscriber	2015-08-31 23:00:00	2015-08-31 23:00:00
2	913455	2015-08-31 23:13:00	47	2015-08-31 23:18:00	64	Subscriber	2015-08-31 23:00:00	2015-08-31 23:00:00
3	913454	2015-08-31 23:10:00	10	2015-08-31 23:17:00	8	Subscriber	2015-08-31 23:00:00	2015-08-31 23:00:00
4	913453	2015-08-31 23:09:00	51	2015-08-31 23:22:00	60	Customer	2015-08-31 23:00:00	2015-08-31 23:00:00

Now new data frames are created to collect the number of bikes entering and leaving each station by counting each date/hour combination for each station. The pandas .groupby() and .count() methods work similarly to the SQL statements GROUP BY and COUNT.

In [6]:
```
TD_bystation_bystartdate = (pd.DataFrame(TD.groupby(['Start Station','St
art Date Truncated'])
                                ['Start Date Truncated']
                                .count()
                                .fillna(0)))
TD_bystation_bystartdate.columns=['Start Count']
TD_bystation_bystartdate.reset_index(inplace=True)

TD_bystation_byenddate = (pd.DataFrame(TD.groupby(['End Station','End Da
te Truncated'])
                                ['End Date Truncated']
                                .count()
                                .fillna(0)))
TD_bystation_byenddate.columns=['End Count']
TD_bystation_byenddate.reset_index(inplace=True)
TD_bystation_byenddate.head()
```

Out[6]:

	End Station	End Date Truncated	End Count
0	2	2014-09-01 14:00:00	1
1	2	2014-09-02 06:00:00	3
2	2	2014-09-02 07:00:00	6
3	2	2014-09-02 08:00:00	1
4	2	2014-09-02 09:00:00	2

Finally, the hourly start/end trip counts for each station are consolidated into a single data set. First an empty data set is created containing every station ID and every date/hour combination in the one year period. In Pandas the MultiIndex.from_product() method is used, accomplishing a similar result to the CROSS JOIN statement in SQL. This creates an empty multi-index array DFMerge where the indices are station ID and the unique date/hour combinations.

In [7]:
```
#Create a data frame containing the station IDs
DFstation=SD['Id']
#Create a data frame containing every date/hour combination from 2014-09
-01 to 2015-08-31
DFtime=pd.date_range(start=TD_bystation_byenddate['End Date Truncated'].
min(), end=TD_bystation_byenddate['End Date Truncated'].max(), freq='h')
#Create a multi-index array where the indices are the station ID and uni
que date/hour combinations
DFMerge=pd.MultiIndex.from_product([DFstation,DFtime],names=['Station',
'Date/Hour'])
#Convert the multi-level indices to column values
DFMerge=pd.DataFrame(index=DFMerge).reset_index()
DFMerge.head()
```

Out[7]:

	Station	Date/Hour
0	2	2014-09-01 00:00:00
1	2	2014-09-01 01:00:00
2	2	2014-09-01 02:00:00
3	2	2014-09-01 03:00:00
4	2	2014-09-01 04:00:00

Then, the pd.merge function is used to incorporate the TD_bystation_bystartdate and TD_bystation_byenddate into DFMerge using station ID and date/hour combinations as join keys. This is functionally similar to the LEFT JOIN in SQL. The DFall data frame is created in the process.

In [8]:
```
DFall=pd.merge(left=DFMerge, right=TD_bystation_bystartdate, how='left',
on=None, left_on=['Station','Date/Hour'], right_on=['Start Station','Sta
rt Date Truncated'])
DFall=pd.merge(left=DFall, right=TD_bystation_byenddate, how='left', on=
None, left_on=['Station','Date/Hour'], right_on=['End Station','End Date
Truncated'])
#Remove extraneous columns from the merge and fill in zero values as na
DFall=DFall.drop(columns=['Start Station','Start Date Truncated','End St
ation','End Date Truncated'])
DFall=DFall.fillna(0)

DFall.head()
```

Out[8]:

	Station	Date/Hour	Start Count	End Count
0	2	2014-09-01 00:00:00	0.0	0.0
1	2	2014-09-01 01:00:00	0.0	0.0
2	2	2014-09-01 02:00:00	0.0	0.0
3	2	2014-09-01 03:00:00	0.0	0.0
4	2	2014-09-01 04:00:00	0.0	0.0

A new column, Net Bikes Per Hour, is calculated as the difference between incoming and outgoing bike counts.

In [9]:
```
DFall['Net Bikes Per Hour']=DFall['End Count']-DFall['Start Count']
DFall['Net Bikes Per Hour'] = DFall['Net Bikes Per Hour'].astype(int)
DFall.head()
```

Out[9]:

	Station	Date/Hour	Start Count	End Count	Net Bikes Per Hour
0	2	2014-09-01 00:00:00	0.0	0.0	0
1	2	2014-09-01 01:00:00	0.0	0.0	0
2	2	2014-09-01 02:00:00	0.0	0.0	0
3	2	2014-09-01 03:00:00	0.0	0.0	0
4	2	2014-09-01 04:00:00	0.0	0.0	0

3 Incorporate station information and weather data into the hourly trip data set

The DFall data frame contains a count of the bikes entering and leaving each station for each hour in the period between 2014-09-01 to 2015-08-31. Now, station data (SD) and weather data (WD) are merged into the data set.

First, the replace_if_changed function is used to add the zip code to the station data (SD) data set.

In [10]:
```
SD.head()
```

Out[10]:

	Id	Name	Lat	Long	Dock Count	City
0	2	San Jose Diridon Caltrain Station	37.329732	-121.901782	27	San Jose
1	3	San Jose Civic Center	37.330698	-121.888979	15	San Jose
2	4	Santa Clara at Almaden	37.333988	-121.894902	11	San Jose
3	5	Adobe on Almaden	37.331415	-121.893200	19	San Jose
4	6	San Pedro Square	37.336721	-121.894074	15	San Jose

In [11]:
```
#Add zip code column to station data (SD)
#Modify rows in trip data (TD) for stations that changed station ID over
the time period
REPLACE_MAP = {'San Jose':95113, 'Mountain View':94041, 'Palo Alto':9430
1, 'San Francisco':94107, 'Redwood City':94063}
SD['Zip'] = SD['City'].apply(replace_if_changed)
```

Then pd.merge is used to join the station information SD into the hourly trip data set, DFall.

In [12]:
```
DFall=pd.merge(left=DFall, right=SD, how='left', left_on=['Station'], ri
ght_on=['Id'])
#Remove extraneous columns
DFall=DFall.drop(columns=['Id','Dock Count'])
DFall.head()
```

Out[12]:

	Station	Date/Hour	Start Count	End Count	Net Bikes Per Hour	Name	Lat	Long	City	Zip
0	2	2014-09-01 00:00:00	0.0	0.0	0	San Jose Diridon Caltrain Station	37.329732	-121.901782	San Jose	95113
1	2	2014-09-01 01:00:00	0.0	0.0	0	San Jose Diridon Caltrain Station	37.329732	-121.901782	San Jose	95113
2	2	2014-09-01 02:00:00	0.0	0.0	0	San Jose Diridon Caltrain Station	37.329732	-121.901782	San Jose	95113
3	2	2014-09-01 03:00:00	0.0	0.0	0	San Jose Diridon Caltrain Station	37.329732	-121.901782	San Jose	95113
4	2	2014-09-01 04:00:00	0.0	0.0	0	San Jose Diridon Caltrain Station	37.329732	-121.901782	San Jose	95113

To join the daily weather data (WD), a column is created to truncate the date/hour data to the date only. This permits the date to be used as a join key. The same is done to the weather data (WD) to avoid any risk of formatting errors. Then the two data sets are merged into a final data frame, DFfinal, using pd.merge(). The column Zip is used as an additional join key to match each station to the weather of its corresponding zip code.

In [13]:
```
DFall['Date']=DFall['Date/Hour'].dt.floor('D')
WD['DateTrunc']=WD['Date'].dt.floor('D')
DFfinal=pd.merge(left=DFall, right=WD, how='left', on=['Zip','Date'])
DFfinal.head()
```

Out[13]:

	Station	Date/Hour	Start Count	End Count	Net Bikes Per Hour	Name	Lat	Long	City	Zip	...	V
0	2	2014-09-01 00:00:00	0.0	0.0	0	San Jose Diridon Caltrain Station	37.329732	-121.901782	San Jose	95113	...	
1	2	2014-09-01 01:00:00	0.0	0.0	0	San Jose Diridon Caltrain Station	37.329732	-121.901782	San Jose	95113	...	
2	2	2014-09-01 02:00:00	0.0	0.0	0	San Jose Diridon Caltrain Station	37.329732	-121.901782	San Jose	95113	...	
3	2	2014-09-01 03:00:00	0.0	0.0	0	San Jose Diridon Caltrain Station	37.329732	-121.901782	San Jose	95113	...	
4	2	2014-09-01 04:00:00	0.0	0.0	0	San Jose Diridon Caltrain Station	37.329732	-121.901782	San Jose	95113	...	

5 rows × 34 columns

Now that all of the trip, station, and weather data have been collected into a single dataframe, the relationships between them can be closely studied. A few extra columns are added for the ease of plotting: hour, weekday, and month are extracted from the Date/Hour column.

In [14]:
```
#Extract and create new columns hour, weekday, and month from Date/Hour
 column
DFfinal['hour']=DFfinal['Date/Hour'].dt.hour
DFfinal['weekday']=DFfinal['Date/Hour'].dt.weekday+1
DFfinal['month']=DFfinal['Date/Hour'].dt.month
#Calculate absolute value of Net Bikes Per Hour column
DFfinal['Abs Net Bikes Per Hour']=DFfinal['Net Bikes Per Hour'].abs()
#Convert hour of day and weekday to numeric string
DFfinal['weekday_str']=DFfinal['weekday'].apply(str)
DFfinal['hour_str']=DFfinal['hour'].apply(str)
#Convert weekday to string
conditions = [
    (DFfinal['weekday_str']=='1'),
    (DFfinal['weekday_str']=='2'),
    (DFfinal['weekday_str']=='3'),
    (DFfinal['weekday_str']=='4'),
    (DFfinal['weekday_str']=='5'),
    (DFfinal['weekday_str']=='6'),
    (DFfinal['weekday_str']=='7')]

choices = ['Mon', 'Tues', 'Wed','Thur','Fri','Sat','Sun']
DFfinal['WeekdayName'] = np.select(conditions, choices)

DFfinal['Mean VisibilityMiles']=DFfinal['Mean VisibilityMiles'].apply(st
r)
```

In [15]:
```
DFfinal.head()
```

Out[15]:

	Station	Date/Hour	Start Count	End Count	Net Bikes Per Hour	Name	Lat	Long	City	Zip	...	E
0	2	2014-09-01 00:00:00	0.0	0.0	0	San Jose Diridon Caltrain Station	37.329732	-121.901782	San Jose	95113	...	
1	2	2014-09-01 01:00:00	0.0	0.0	0	San Jose Diridon Caltrain Station	37.329732	-121.901782	San Jose	95113	...	
2	2	2014-09-01 02:00:00	0.0	0.0	0	San Jose Diridon Caltrain Station	37.329732	-121.901782	San Jose	95113	...	
3	2	2014-09-01 03:00:00	0.0	0.0	0	San Jose Diridon Caltrain Station	37.329732	-121.901782	San Jose	95113	...	
4	2	2014-09-01 04:00:00	0.0	0.0	0	San Jose Diridon Caltrain Station	37.329732	-121.901782	San Jose	95113	...	

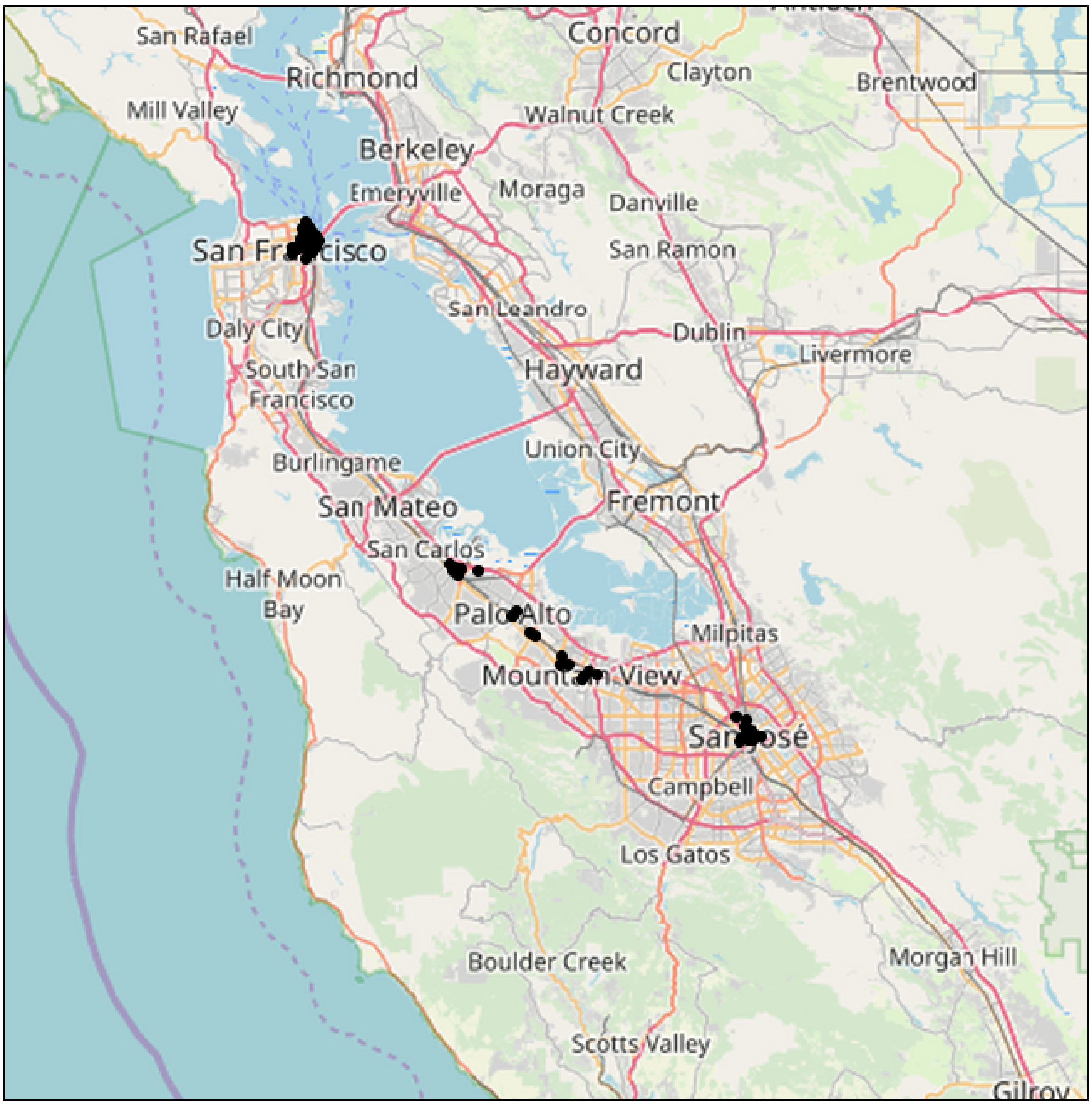
5 rows × 41 columns

4 Relationships

Geographic locations of bike stations

The station data (SD) data frame includes the longitude and latitude of each station. Using the Cartopy package, each station can be plotted on a map of the local area.

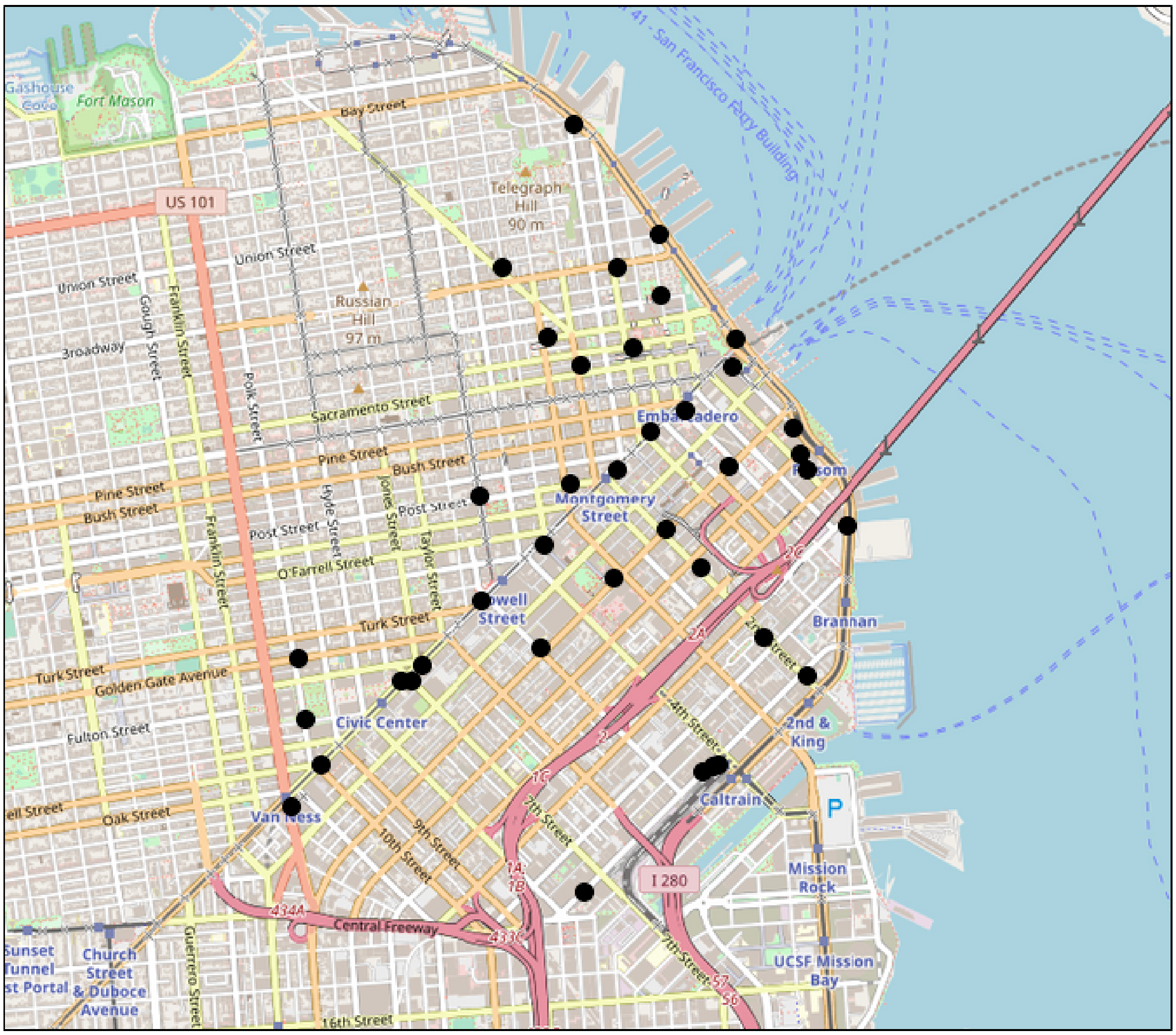
In [16]:
```
import cartopy.crs as ccrs
from cartopy.io.img_tiles import OSM

# Initialize figure and specify Open Street Map as map tile source
plt.figure(figsize=(12, 12))
imagery = OSM()
ax = plt.axes(projection=imagery.crs)
# Define axes in terms of longitude and latitude and add map tile imager
y
ax.set_extent([-122.75, -121.5,37, 38])
ax.add_image(imagery, 9,interpolation='spline36')
# Add scatter plot of station locations based longitude and latitude
plt.scatter(SD['Long'],SD['Lat'],transform=ccrs.Geodetic(),color='k')
plt.show()
```



Clusters of bike share stations exist in San Francisco, Redwood City, Palo Alto, Mountain View, and San Jose. This map confirms the 'City', 'Lat', and 'Long' columns conform to one another.

San Francisco bike stations

In [17]:
```
plt.figure(figsize=(12, 12))
imagery = OSM()
ax = plt.axes(projection=imagery.crs)
ax.set_extent([-122.435, -122.37,37.765, 37.81])
ax.add_image(imagery, 14,interpolation='spline36')
plt.scatter(SD['Long'],SD['Lat'],s=100,transform=ccrs.Geodetic(),color=
'k')
plt.show()
```

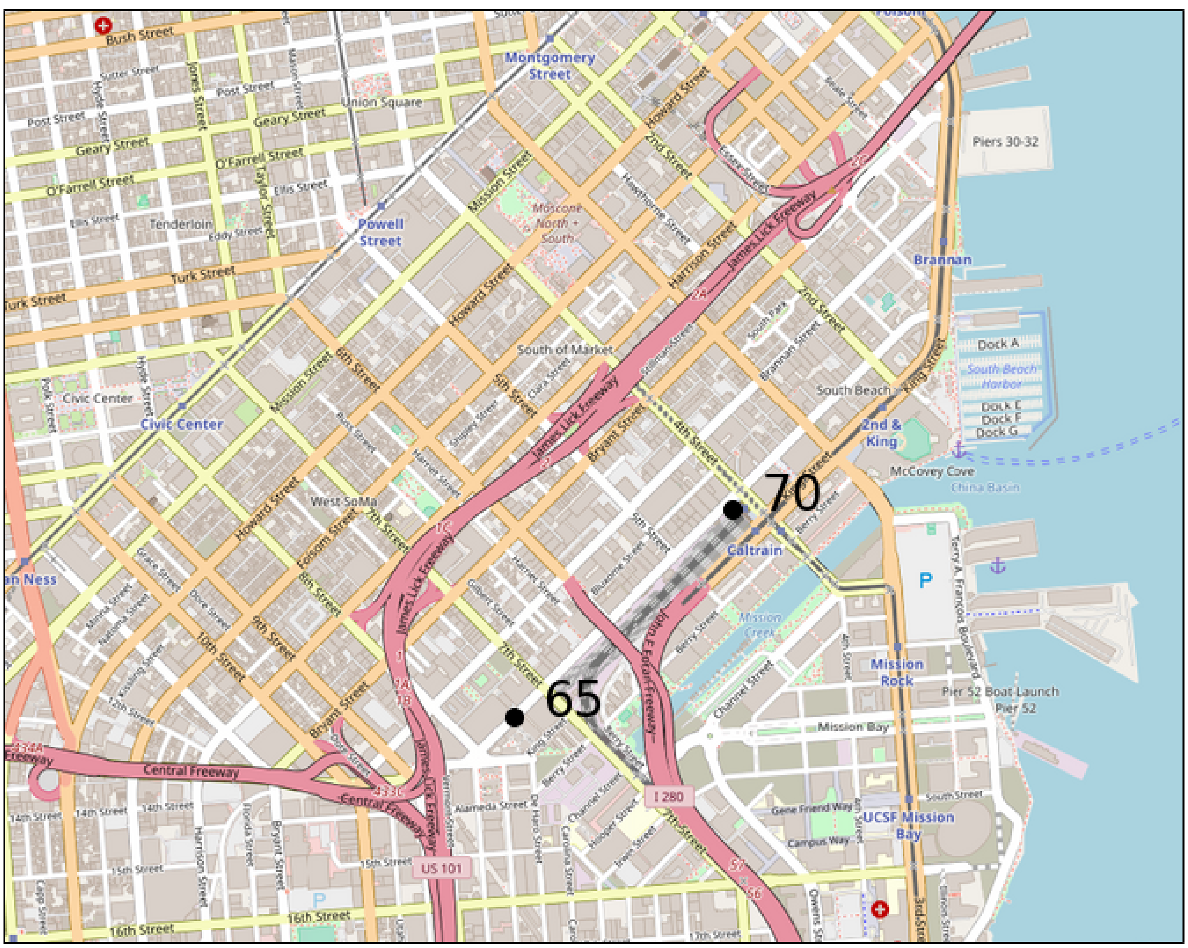


The stations in San Francisco are located in the downtown area, with some stations closer to public transport options and landmarks than others.

Bike share use by day of the week: two stations

Two stations are selected for analysis: station 65, located in close proximity to a Caltrain commuter rail station, and station 70, located a few blocks away.

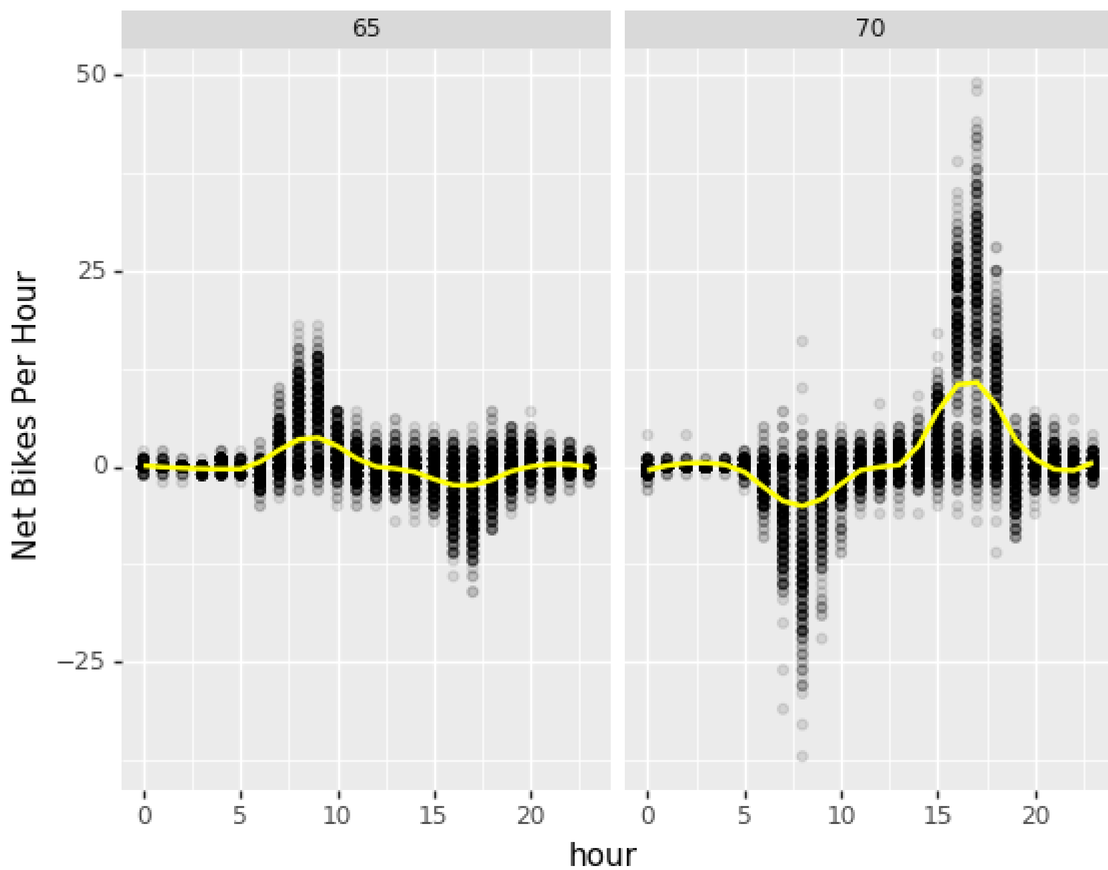
In [18]:
```
#Subset DFfinal to stations 65 and 70
DFfinal_65_70 = DFfinal[DFfinal["Station"].isin([65, 70])]
#Subset DFfinal to stations 65 and 70 between 5am and 9pm
DFfinal_65_70_day = DFfinal[DFfinal["Station"].isin([65, 70]) & DFfinal[
'hour'].isin(list(range(5,21)))]
#Subset DFfinal to stations 65 and 70 between 5am and 9pm
DFfinal_day=DFfinal[DFfinal['hour'].isin(list(range(5,21)))]
#Subset SD to stations 65 and 70
SD_65_70=SD[SD['Id'].isin([65,70])]

plt.figure(figsize=(12, 12))
imagery = OSM()
ax = plt.axes(projection=imagery.crs)
ax.set_extent([-122.42, -122.38,37.765, 37.79])
ax.add_image(imagery, 15,interpolation='spline36')
plt.scatter(SD_65_70['Long'],SD_65_70['Lat'],s=100,transform=ccrs.Geodet
ic(),color='k')
#Label stations
plt.text(-122.401717,37.771058,"65",fontsize=28,transform=ccrs.Geodetic
())
plt.text(-122.394260,37.776617,"70",fontsize=28,transform=ccrs.Geodetic
())
plt.show()
```



For the two selected stations (Station IDs 65 and 70), the hourly change in bikes per station is plotted below as a scatter plot as a function of time of day (24-hour scale) between 5am and 9pm. The line of best fit is plotted as well in yellow.

In [19]:
```
import plotnine as p9
import skmisc
```

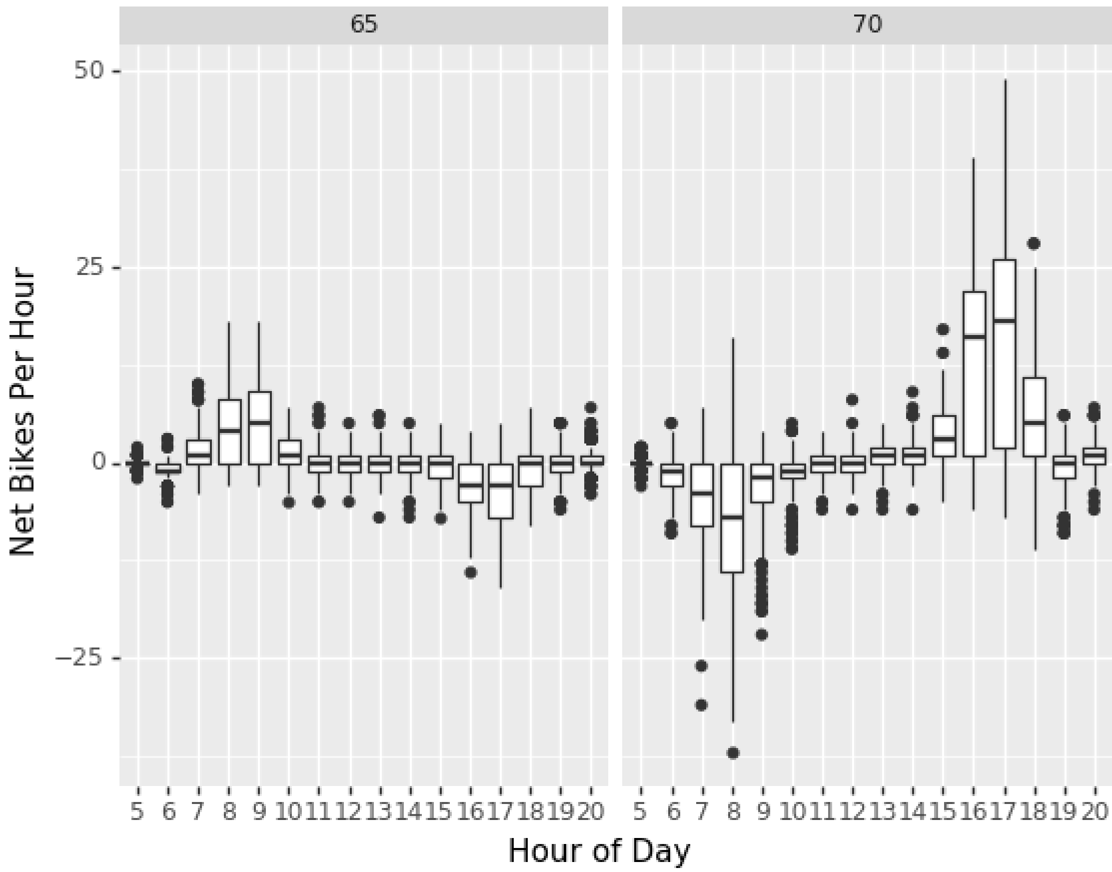
In [20]:
```
#Scatter plot with line of best fit shown in yellow
(p9.ggplot(data=DFfinal_65_70,
           mapping=p9.aes(x="hour", y="Net Bikes Per Hour"))
    + p9.geom_point(alpha=0.1)
    + p9.facet_wrap("Station")
    + p9.geom_smooth(method="loess",size=1, color="yellow",alpha=1,se=Tr
ue,span=0.4,level=0.9)
)
```



Out[20]: <ggplot: (307157894)>

For stations 65 and 70, the highest rate of change of bikes per station occurs during the morning and evening rush hours. The least amount of usage is at night. Interestingly, the stations have differing trends in morning/evening usage due to their location: stations 65 is located near the financial district, implying riders are commuting to these stations in the morning and vice/versa. Station 70 is located next to a commuter rail station, implying riders are commuting from this station in the morning. Importantly, this shows that each station has unique ridership behavior based on its geographic location.

In [21]:
```
(p9.ggplot(data=DFfinal_65_70_day,
          mapping=p9.aes(x="hour_str", y="Net Bikes Per Hour"))
    #+ p9.geom_point(alpha=0.1)
    + p9.geom_boxplot()
    + p9.facet_wrap("Station")
    +p9.xlab("Hour of Day")
    +p9.scale_x_discrete(limits=[str(i) for i in list(range(5,21))])
)
```



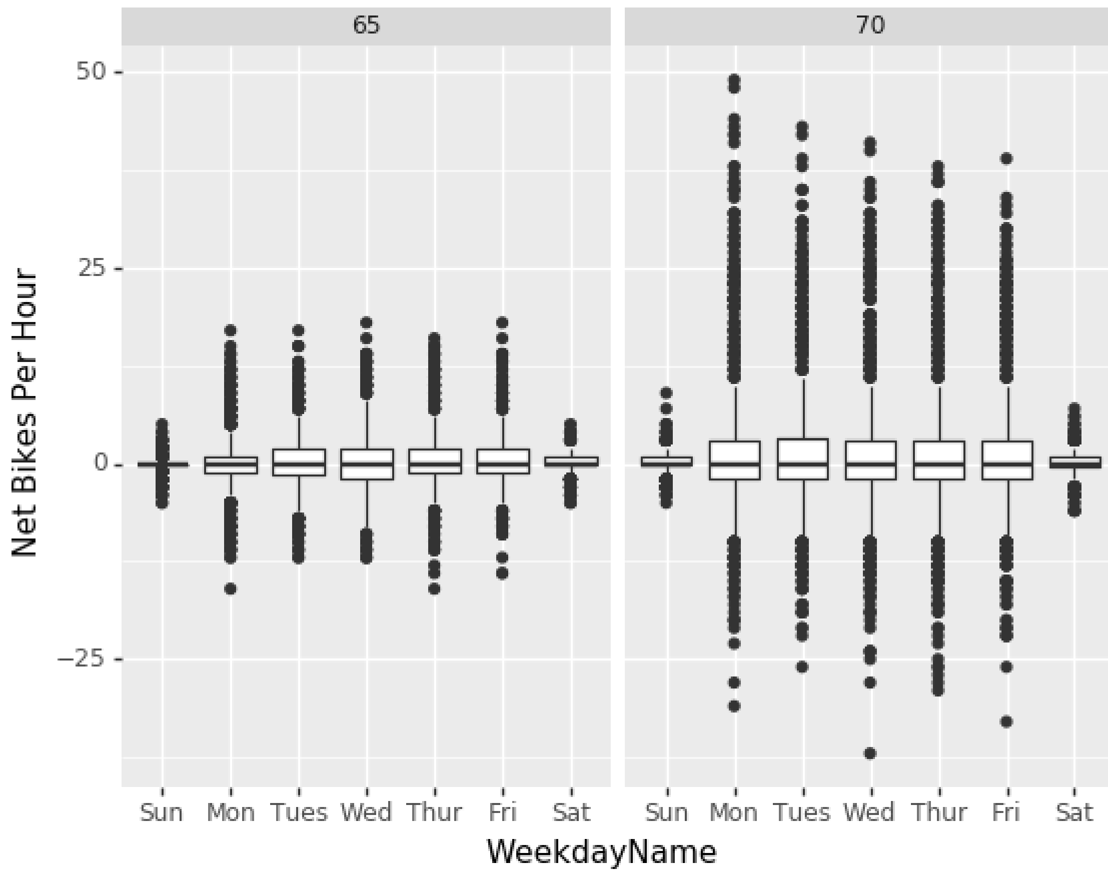
Out[21]: <ggplot: (289752317)>

Box plots reveal richer statistical detail than previous scatter plot, which has many overlapping data points. Each box plot shows the outliers, first quartile, median, and third quartile, quantitatively displaying the pattern of ridership over the course of a day.

Bike share usage by day of the week: two stations

A similar plot per day of the week shows that most ridership occurs on weekdays:

In [22]:
```
(p9.ggplot(data=DFfinal_65_70_day,
          mapping=p9.aes(x="WeekdayName", y="Net Bikes Per Hour"))
    #+ p9.geom_point(alpha=0.1)
    + p9.geom_boxplot()
    + p9.facet_wrap("Station")
    +p9.scale_x_discrete(limits=['Sun','Mon','Tues','Wed','Thur','Fri',
'Sat'])
)
```

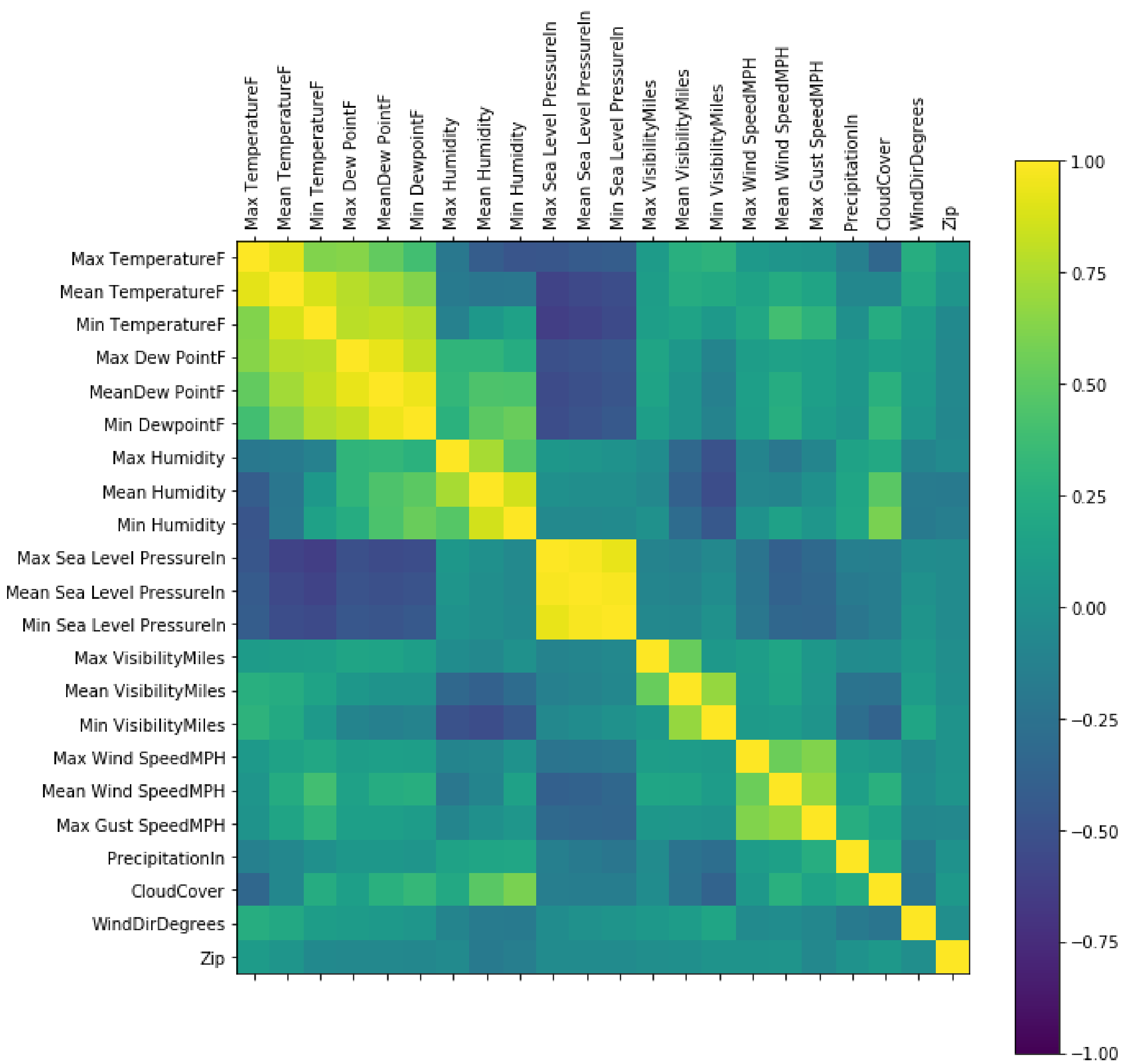


Out[22]: <ggplot: (-9223372036565034962)>

Effect of weather on ridership

As 24 columns exist in the original weather data (WD) set, there is a lot of information to consider. Certain columns, however, may be redundant. For example, several sets of weather data is in triplicate min/mean/max form, and related variables may already correlate well to one another. A correlation matrix can show these relationships and be used to avoid redundancies in the analysis.

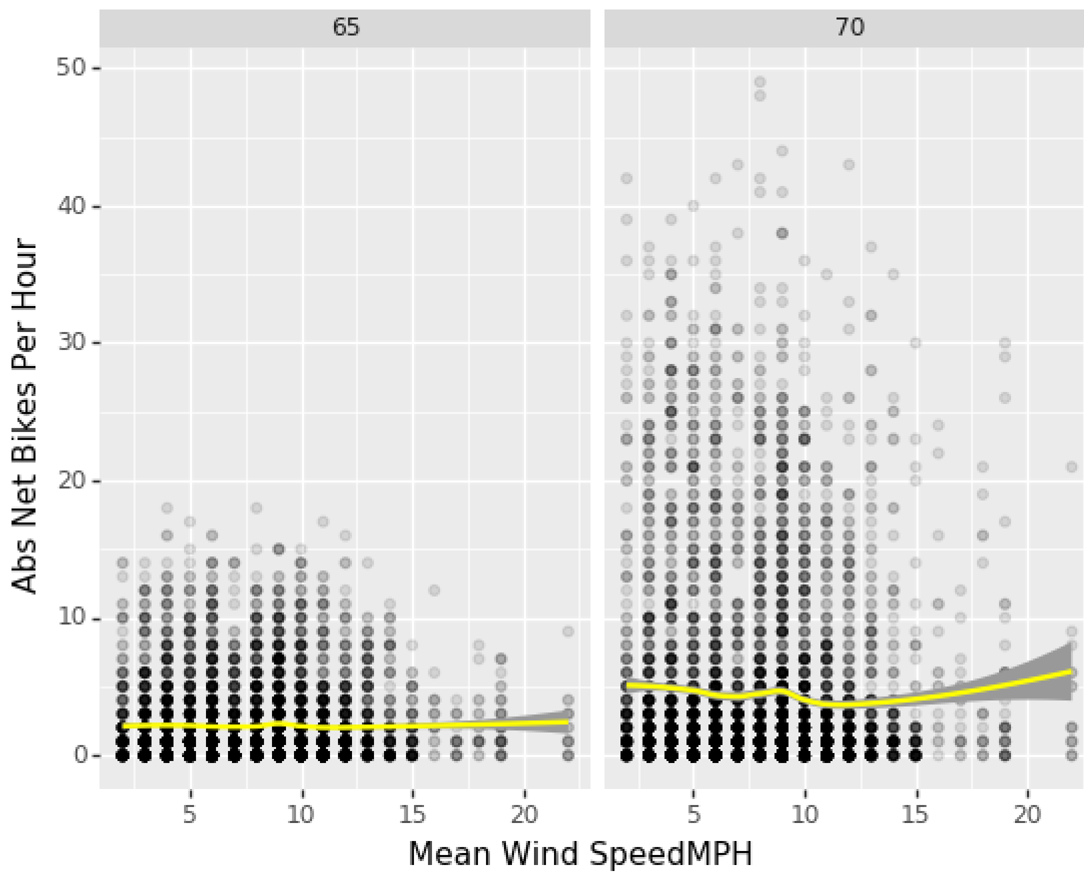
In [23]:
```
def plot_corr(df,size=10):
    '''Function plots a graphical correlation matrix for each pair of co
lumns in the dataframe.

    Input:
        df: pandas DataFrame
        size: vertical and horizontal size of the plot'''

    corr = df.corr()
    fig, ax = plt.subplots(figsize=(size, size))
    cax = ax.matshow(corr, vmin=-1, vmax=1)
    fig.colorbar(cax)
    #ax.matshow(corr)
    plt.xticks(range(len(corr.columns)), corr.columns);
    plt.xticks(rotation=90)
    plt.yticks(range(len(corr.columns)), corr.columns);

plot_corr(WD)
```



The correlation matrix shows that clusters of weather data correlate well to one another. Min/mean/max temperature and dew point correlate well to another another, as do humidity, sea level pressure, and wind speed. For this reason, only the mean of each of these values will be analyzed.

Wind Speed

In [24]:
```
(p9.ggplot(data=DFfinal_65_70_day,
           mapping=p9.aes(x="Mean Wind SpeedMPH", y="Abs Net Bikes Per Ho
ur"))
    #+ p9.geom_point(alpha=0.1)
    + p9.geom_point(alpha=0.1)
    + p9.facet_wrap("Station")
    + p9.geom_smooth(method="loess",size=1, color="yellow",alpha=1,se=Tr
ue,span=0.6,level=0.9)
)
```

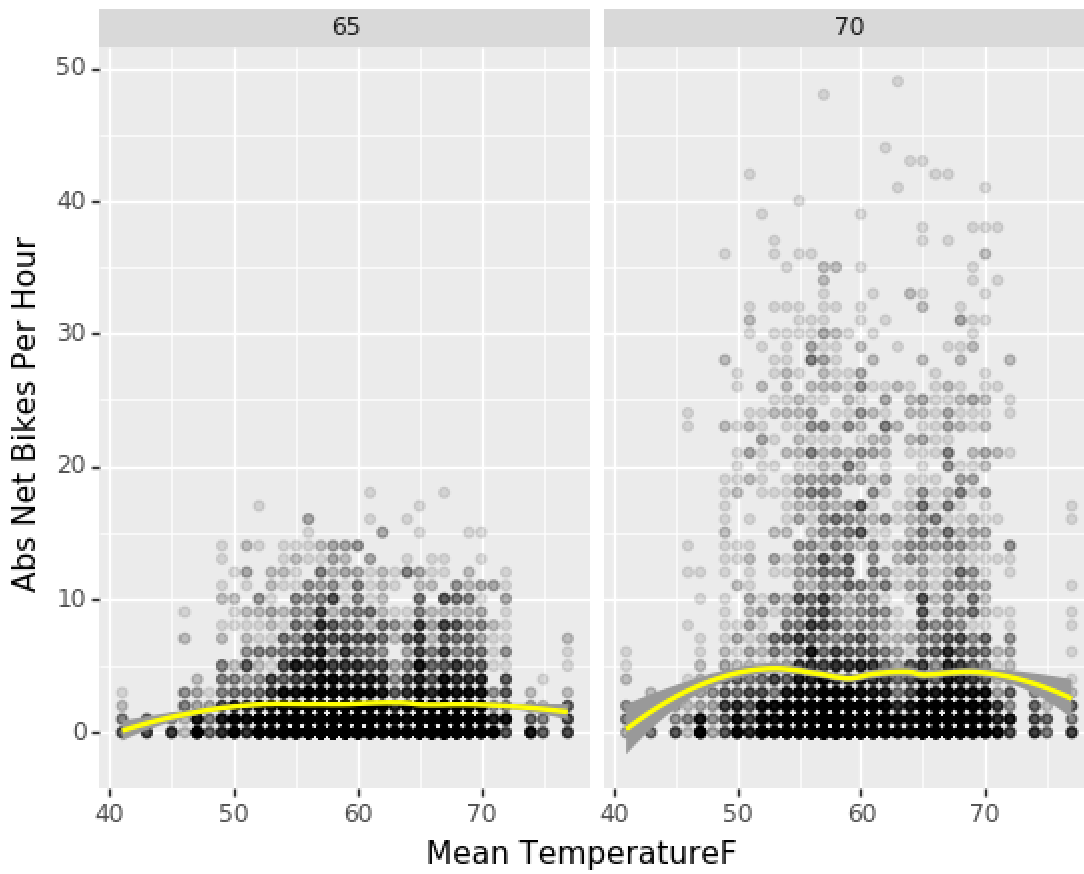


Out[24]: <ggplot: (289557028)>

Mean wind speed, plotted with a best fit line and 90% confidence interval, does not have a strong effect on ridership for stations 65 and 70. Above about 15 MPH, the confidence interval expands -- this is an artefact of having fewer data points above this wind speed.

Mean Temperature

In [25]:
```
(p9.ggplot(data=DFfinal_65_70_day,
           mapping=p9.aes(x="Mean TemperatureF", y="Abs Net Bikes Per Hou
r"))
    #+ p9.geom_point(alpha=0.1)
    + p9.geom_point(alpha=0.1)
    + p9.facet_wrap("Station")
    + p9.geom_smooth(method="loess",size=1, color="yellow",alpha=1,se=Tr
ue,span=0.6,level=0.9)
)
```

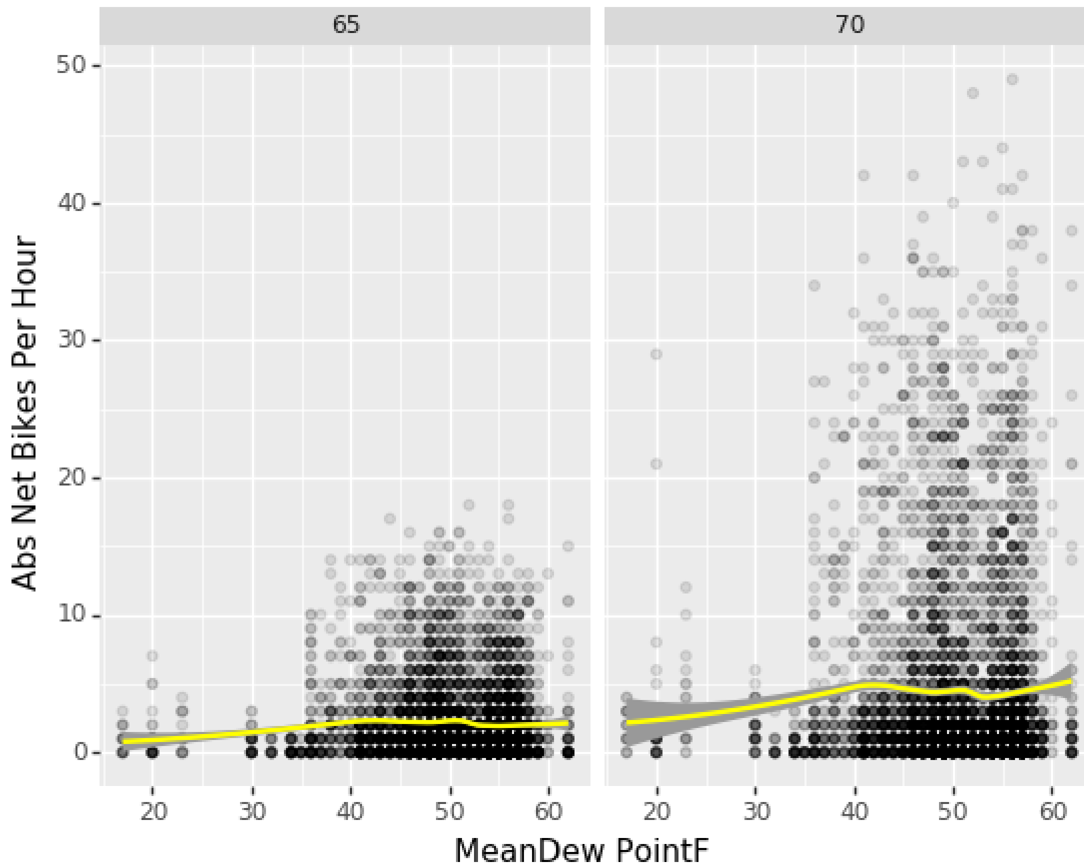


Out[25]: `<ggplot: (-9223372036559268436)>`

For both stations 65 and 70, the ridership occurs most often between 50 and 70 degrees F. This is likely a function of comfort and of the stable temperatures in this part of California, with fewer data points below 50 degrees and above 70 degrees.

Dew Point

In [26]:
```
(p9.ggplot(data=DFfinal_65_70_day,
           mapping=p9.aes(x="MeanDew PointF", y="Abs Net Bikes Per Hour"
))
    #+ p9.geom_point(alpha=0.1)
    + p9.geom_point(alpha=0.1)
    + p9.facet_wrap("Station")
    + p9.geom_smooth(method="loess",size=1, color="yellow",alpha=1,se=True,span=0.6,level=0.9)
)
```



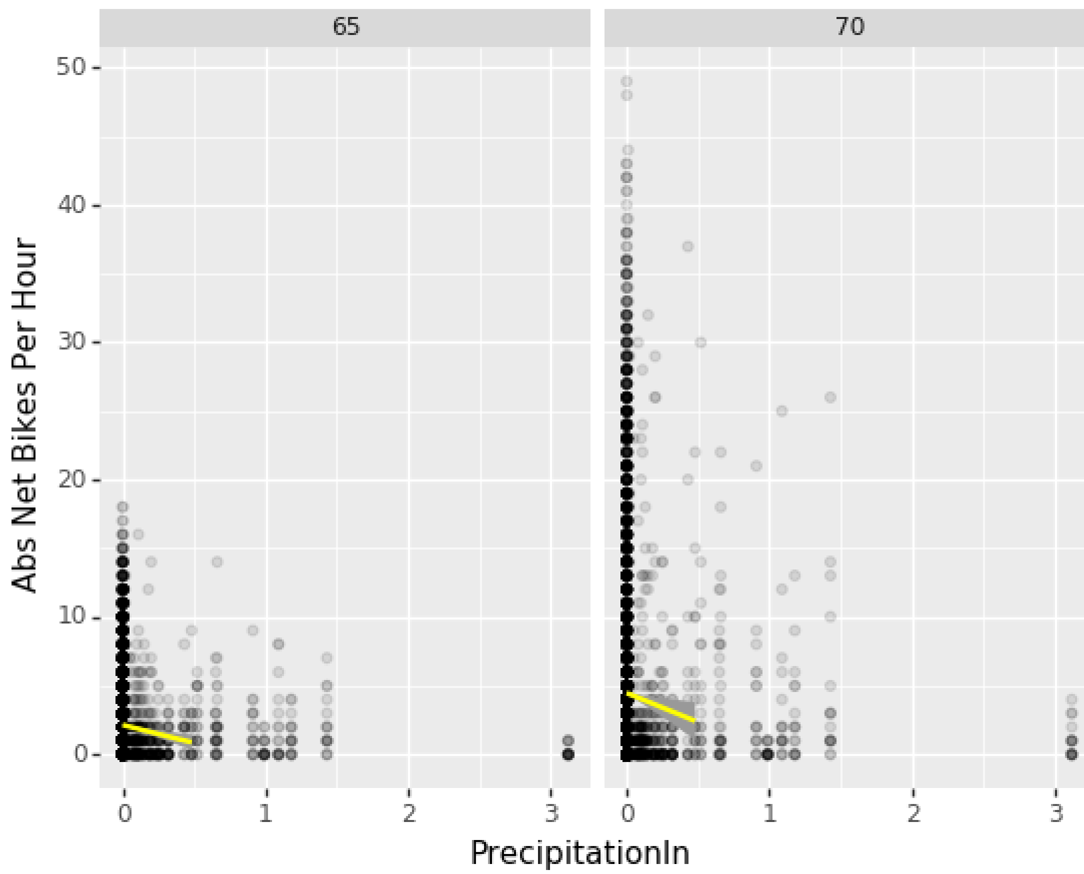
Out[26]: <ggplot: (289554913)>

Similar to the temperature data above, the dew point largely falls between 35 and 60 degrees F, all of which are comfortable. With few data points outside this range, it is unclear if dew point has an effect on ridership. If this project analyzed bike share data in Washington DC, this relationship would be very different.

Precipitation

In [27]:
```
(p9.ggplot(data=DFfinal_65_70_day,
          mapping=p9.aes(x="PrecipitationIn", y="Abs Net Bikes Per Hour"
))
    + p9.geom_point(alpha=0.1)
    + p9.facet_wrap("Station")
+ p9.geom_smooth(data=DFfinal_65_70_day[DFfinal_65_70_day['Precipitation
In']<0.5], method="lm",size=1, color="yellow",alpha=1,se=True,span=0.9,l
evel=0.9)
)
```

```
/anaconda3/lib/python3.6/site-packages/numpy/core/fromnumeric.py:52: Fu
tureWarning: Method .ptp is deprecated and will be removed in a future
version. Use numpy.ptp instead.
  return getattr(obj, method)(*args, **kwds)
```



Out[27]: <ggplot: (316279644)>

Precipitation has a strong effect on ridership at stations 65 and 70. Above 0.5 inches of rain, ridership drops off considerably. The line of best fit is to daily precipitation values below 0.5.

5 Summary and recommendations for future work

The present project has revealed relationships within a San Francisco bay area bike share network during the period of 09/2014 to 08/2015. The geographic location of stations, time of day, day of the week, and weather parameters all affect the ridership numbers to greater or lesser extents.

To fully leverage this data for the logistics problem stated in the introduction, a regression model may be used to fit and predict the Net Bikes Per Hour data to the many features that have been shown to influence it. Such a model would almost certainly need to be performed an a station-by-station basis, given the high variability between stations based on geographic location. This is left as a consideration for future work.