Data Science Programming In Python

Anita Raichand

Data Science Programming in Python

Table of Contents

Introduction - Data Science Programming in Python

The aim of this book is to show how to apply data analysis principles to a practical use case scenario using Python as the data analysis language. We'll go on this journey by looking at the the data workflow from munging to grouping data to visualizing and also include some time-series analysis as well. The format includes asking questions of the data and showing the programming steps needed to answer the question. By the end of reading this book, you will be able to apply these techniques to your own data.

About the book

This book is written in a literate programming style where text, code, and output are presented together . This will maximize your learning and understanding of code and the data analysis workflow. The book teaches the type of interactive coding and iterative analysis that is essential to be successful in data science programming.

Coding Tips

In the code snippets, a backslash character (\) means that the same line of code is wrapped to the next line in the book. You do not need to type this character into an interpreter.

Use a REPL (en.wikipedia.org/wiki/Read–eval–print_loop) to have an interactive environment where you can write code and see the resulting output.

Try the methods you learn in this book on your own data to reinforce learning. Use a Python interpreter to code and your favorite editor to take notes.

Data Science Programming in Python - Data Munging

Background

Bay Area Bike Share commenced it's pilot phase of operation in the San Francisco bay area in August 2013 with plans to expand. It is the first bike sharing scheme in California. As it is meant for short trips, the bikes should be returned to a dock in thirty minutes or less or an additional fee would be incurred according to the website. There are two types of memberships: customer and subscriber. A subscriber is an annual membership while a customer is defined as someone using either the twenty-four hour or three day passes. Currently(Sept 2014), it costs nine dollars for twenty-four hours, twenty-two dollars for three days, and eight-eight dollars for the year. Overtime fees are four dollars for an extra thirty minutes and seven dollars for each thirty minutes after that. Data on the first six months of operations were released as part of a data challenge. The data included three files for trip history, weather information, and dock availability. The merged data was used for the following analysis.

Data Munging and Carpentry

First, we'll read in the data and inspect the data columns and datatypes and think about what questions we want to ask our data and what things are we interested in learning about the data. Be curious and empathetic in thinking about what the various stakeholders including the City, the customers, and other interested people would be interested in gleaning by keeping civic fiscal, civic, and social goals in mind. In addition to that, there will be quite a bit of cleaning and data carpentry needed to get the data into a format useful for analysis.

The dataset comes from three csv files from the Bay Area Bikeshare data challenge. We merged the data in R as we started a similar analysis there but really wanted to use IPython and the superb time series functionality in Pandas.

Optionally, one can cache or log the code in IPython with the following two commands.

```
%load_ext ipycache
%logstart
```

```
Activating auto-logging. Current session state plus future input saved. Filename :
ipython_log.py
Mode : rotate
Output logging : False
Raw input log : False
Timestamping : False
State : active
```

Import the libraries that are needed.

```
import numpy as np
import pandas as pd
```

```
from datetime import datetime, time

%matplotlib inline
import seaborn as sns
from ggplot import *

print pd.__version__
print np.__version__
```

```
0.14.1
1.9.0
```

Read the data.

```
dmerge4 = pd.read_csv('finalmerge.csv', parse_dates=['Start.Date'])
dmerge4.head(3)
```

	lubstartdate	Start.Station	Trip.ID	Duration	Start.Date	Start.Terminal	End.Date	End.Station	End.Terminal	Bike..	...	Events	Wind_Dir_Degrees	zip	landmark	station_id	name	lat	long	dockc
0	2013-08-29	2nd at Folsom	4636	186	2013-08-29 15:11:00	62	8/29/2013 15:14	2nd at Townsend	61	366	...	NaN	286	94107	San Francisco	62	2nd at Folsom	37.785299	-122.396236	19
1	2013-08-29	2nd at Folsom	4820	813	2013-08-29 17:35:00	62	8/29/2013 17:48	Townsend at 7th	65	409	...	NaN	286	94107	San Francisco	62	2nd at Folsom	37.785299	-122.396236	19
2	2013-08-29	2nd at Folsom	5001	236	2013-08-29 20:00:00	62	8/29/2013 20:04	Market at Sansome	77	567	...	NaN	286	94107	San Francisco	62	2nd at Folsom	37.785299	-122.396236	19

3 rows × 43 columns

Format column names.

```
dmerge4.columns = dmerge4.columns.map(lambda x: x.replace('.','').lower())

dmerge4['zip_name'] = dmerge4.zip.replace({94107: 'San Francisco', 95113:\
 'San Jose', 94301:'Palo Alto',94041:'Mountain View',94063:'Redwood City'})
```

Inspect datatypes.

Inspecting the datatypes and variables are the first things that should be done in data munging. And, we'll do it again after doing our data cleaning and formatting operations. Notice that the end date and start date are not currently in datetime format so let's remedy that and take care of some other things for the questions related to time-series.

```
dmerge4.dtypes
```

```
> lubstartdate:  object
> startstation:  object
> tripid:  int64
> duration:  int64
> startdate:  datetime64[ns]
> startterminal:  int64
> enddate:  object
> endstation:  object
> endterminal:  int64
> bike:  int64
> subscriptiontype:  object
> zipcode:  object
> date:  object
> max_temperature_f:  int64
> mean_temperature_f:  int64
> min_temperaturef:  int64
> max_dew_point_f:  int64
>meandew_point_f:  int64
>min_dewpoint_f:  int64
>max_humidity:  int64
>mean_humidity:  int64
>min_humidity:  int64
>max_sea_level_pressure_in:  float64
>mean_sea_level_pressure_in:  float64
>min_sea_level_pressure_in:  float64
>max_visibility_miles:  int64
>mean_visibility_miles:  int64
>min_visibility_miles:  int64
>max_wind_speed_mph:  int64
>mean_wind_speed_mph:  int64
```

```
>max_gust_speed_mph:  float64
>precipitation_in:  object
>cloud_cover: int64
>events:  object
>wind_dir_degrees:  int64
>zip:  int64
>landmark:  object
>station_id:  int64
>name:  object
>lat:  float64
>long:  float64
>dockcount:  int64
>installation:  object
>zip_name:  object
>dtype: object
```

Working with dates and time

Parse date columns into datetime format.

```
dmerge4['enddate'] = pd.to_datetime(dmerge4['enddate'])
dmerge4['lubstartdate'] = pd.to_datetime(dmerge4['lubstartdate'])
```

Set datetime index column.

```
dmerge4.set_index('startdate', inplace=True, drop=False, append=False)
```

Extract hour,day,and month from datetime object.

```
dmerge4['day'] = dmerge4.index.day
dmerge4['hour'] = dmerge4.index.hour
dmerge4['month'] = dmerge4.index.month
```

Having the date and time is great. However for the analysis, it would be really useful to group time into subjective useful categories that would be useful for understanding usage a little bit better. We think we would be really interested to learn about the differences and similarities in usage and duration patterns between the morning commute and evening rush period. It would also be interesting to know the profile and usage of users in the middle of the day as well as at night and in what we refer to as the wee hours. So, We've decided to split the time into the following segments.

morning: 5,6,7,8,9,10
evening: 15,16,17,18,19
midday: 11,12,13,14
night: 20,21,22,23,0
wee hours: 1,2,3,4

```
dmerge4['timeofday'] = dmerge4.hour.replace({15: 'evening', 17: 'evening',20:' \
night',19:'evening',12:'mid_day',14:'mid_day', 13:'mid_day', 9: 'morning', 22:'n \
ight',11:'mid_day', 18:'evening', 16:'evening', 10:'morning', 21:'night', 23:'night',
6:'morning',8:'morning',7:'morning',  1:'wee_hours',  2:'wee_hours',  3:'wee_ \
hours', 0:'night',5:'morning', 4:'wee_hours'})
```

Difference between end date and start date.

```
dmerge4['diff'] = dmerge4.apply(lambda x: x['enddate'] - x['startdate'], axis=1)
```

Difference between end hour and start hour.

```
dmerge4['hourdiff'] = dmerge4.apply(lambda x: x['enddate'].hour - x['startdate']\
.hour, axis=1)
```

Now, one of the most important variables in the dataset is duration. We are very interested in understanding how long a customer or subscriber has a bicycle. Duration is defined as

the number of seconds from taking a bicycle from a dock and returning it to a dock. Many of the interesting questions such as breakdown of duration between customer and subscriber by landmark or by time of day are relevant to both customer end users and civic authorities as it affects how many bikes will be available and where will they be needed. Let’s transform duration into minutes and then also define a binary variable to compare trips under thirty minutes and less to trips over thirty minutes.

Duration is in seconds so let’s convert it to minutes.

```
dmerge4['durationminutes'] = dmerge4['duration']/60
```

Convert duration to an integer type.

```
#round and convert to integer
dmerge4["duration_i"] = dmerge4['durationminutes'].round(0).astype('int64')
```

Duration as float type to two decimal places.

```
#round to two decimal spaces and keeep as float
dmerge4["duration_f"] = dmerge4['durationminutes'].round(2)
```

We’ll write a function that maps ‘under thirty’ to all values under 30 (minutes) and ‘over thirty’ to all values over 30 (minutes)

```
def isitthirty(x):
if x <= 30: return 'in_thirty'
elif 31 <= x : return 'over_thirty'
else: return 'None'
```

```
dmerge4["thirtymin"] = dmerge4['duration_i'].map(isitthirty)
```

```
dmerge4.dtypes
```

```
lubstartdate                    datetime64[ns]
startstation                            object
tripid                                   int64
duration                                 int64
startdate                       datetime64[ns]
startterminal                            int64
enddate                         datetime64[ns]
endstation                              object
endterminal                              int64
bike                                     int64
subscriptiontype                        object
zipcode                                 object
date                                    object
max_temperature_f                        int64
mean_temperature_f                       int64
min_temperaturef                         int64
max_dew_point_f                          int64
meandew_point_f                          int64
min_dewpoint_f                           int64
max_humidity                             int64
mean_humidity                            int64
min_humidity                             int64
max_sea_level_pressure_in              float64
mean_sea_level_pressure_in             float64
min_sea_level_pressure_in              float64
max_visibility_miles                     int64
mean_visibility_miles                    int64
```

```
min_visibility_miles                    int64
max_wind_speed_mph                      int64
mean_wind_speed_mph                     int64
max_gust_speed_mph                    float64
precipitation_in                       object
cloud_cover                             int64
events                                 object
wind_dir_degrees                        int64
zip                                     int64
landmark                               object
station_id                              int64
name                                   object
lat                                   float64
long                                  float64
dockcount                               int64
installation                           object
zip_name                               object
day                                     int64
hour                                    int64
month                                   int64
timeofday                              object
diff                          timedelta64[ns]
hourdiff                                int64
durationminutes                       float64
duration_i                              int64
duration_f                            float64
thirtymin                              object
Length: 54, dtype: object
```

```
%logstop
```

Data Science Programming in Python - Grouping and Aggregating Data

Grouping and querying data

Now we will slice and dice our dataset using grouping, aggregation functions, summary statistics, and various querying and indexing operations. To really understand the bike sharing data, we need to look at natural groupings such as starting landmark, subscription type, and the time of day of the start of a bicycle trip. The work done in this section will set up some beautiful and informative plots in the next section of the analysis. Most of the tables below have plotting representations in the visualization section.

Grouping and aggregating or reducing data and obtaining summary measures and statistics is the heart of data analysis and work together with the visualization process including those of an exploratory nature. The iterative process of data discovery at best leads to new questions and insights being discovered which inform and refine the data carpentry process. This is where the interactive repl excels. Think of it as a circular inquisitive process of data munging, grouping and aggregating, and visualizing. We will explore using SQL inspired methods of grouping operations in Pandas and the use of mult-indexes and hierarchical indexing.

```
import numpy as np
import pandas as pd
from datetime import datetime, time
```

```
%matplotlib inline
import seaborn as sns
from ggplot import *
```

```
print pd.__version__
print np.__version__
```

```
0.14.1
1.9.0
```

```
%load_ext ipycache
```

Grouping and aggregating

It’s time to explore various ways of grouping and aggregating data. Let’s review some terminology. Dockcount refers to the number of docks installed, subscription type is either customer or subscriber, and landmark refers to an area covered by San Francisco, San Jose, Redwood City, or Palo Alto.

In the data munging section, we created a column indicating whether the duration of a segment of a bicycle trip was under thirty minutes or over thirty minutes as this is the allotted time before being charged overtime fees. So, what percentage of total trips were over thirty minutes?

What percentage of total trips were over thirty minutes?

```
dmerge4.thirtymin.value_counts()
```

```
in_thirty      134912
over_thirty      9103
dtype: int64
```

Only 6% of total rides had a duration over thirty minutes so it looks like people are good about returning bikes on time. Or, were they? When we look at the statistics based on landmark and subscriber type, we will get a better picture of who and where have longer duration times.

How does average duration vary by time of day for trips both under thirty minutes and over thirty minutes?

```
dmerge4.groupby(['thirtymin','timeofday'])[['duration_f']].mean()
```

		duration_f
thirtymin	timeofday	
in_thirty	evening	9.736020
	mid_day	10.387775
	morning	8.914918
	night	9.641814
	wee_hours	10.150168
over_thirty	evening	161.593282
	mid_day	164.873060
	morning	218.527759
	night	271.402983
	wee_hours	345.829388

Here is another way to check on how many observations out of the total had trip durations over thirty minutes.

```
len(dmerge4[dmerge4['thirtymin']=='over_thirty'])
```

```
9103
```

How many trips began and ended at the same station?

Only 6878 out 144015 observations or 4.8% were round trips so it would seem that people are really using the bicycles to get from point A to point B

```
len(dmerge4[dmerge4['startstation']== dmerge4['endstation']])
```

```
6878
```

Let’s look at average duration by subscription type. Customers are defined has having either a twenty-four hour or three day pass while Subscribers are defined as having an annual pass. Average durations for subscribers is almost ten minutes while it is sixty minutes for customers. Perhaps customers are recreational riders who take longer trips while subscribers are shorter distance commuters. We’ll learn more when we slice and dice the data later.

What is average duration by subscription type?

```
dmerge4.groupby(['subscriptiontype']).agg({'duration_f' : np.mean})
```

	duration_f
subscriptiontype	
Customer	60.491974
Subscriber	9.832834

How many observations in the dataset belong to each subscription type?

```
dmerge4.groupby('subscriptiontype').size()
```

```
subscriptiontype
Customer             30368
Subscriber          113647
dtype: int64
```

What landmark occurs the most frequently among subscription type of subscribers?

```
dmerge4[dmerge4['subscriptiontype'] =="Subscriber"].landmark.value_counts()
```

```
San Francisco    102735
San Jose           7219
Mountain View      2157
Palo Alto           905
Redwood City        631
dtype: int64
```

So, we've learned that the majority of total subscribers are in San Francisco and take very short trips.

What are the top five start stations?

```
dmerge4.groupby('startstation').duration.sum().order(ascending=False)[:5]
```

```
startstation
Harry Bridges Plaza (Ferry Building)        12579544
Embarcadero at Sansome                      10294998
San Francisco Caltrain (Townsend at 4th)     8457338
Market at 4th                                7706843
Powell Street BART                           6100023
Name: duration, dtype: int64
```

What landmark has the highest total duration?

The table below indicates that San Francisco had the highest total duration but keep in mind that the majority of total observations are also in San Francisco so of course the total would be higher. Using summary statistics will give a better idea of bicycling behavior than looking at total values. Although, for some purposes, it is good to know the total as well. For example, bicycle wear and tear based on total durations.

```
dmerge4.groupby("landmark")['duration_i','subscriptiontype'].aggregate(np.sum)
```

	duration_i
landmark	
Mountain View	121895
Palo Alto	131810
Redwood City	38924
San Francisco	2454177
San Jose	207658

```
%%cache station.pkl stationdata
stationdata
```

What percentage of total bicycle docks area located in each landmark?

54% of all docks are located in San Francisco so this may partially explain why total duration and bicycle trips taken in San Francisco are higher and why durations are higher in Palo Alto where only 6% of the docks are located.

```
stationdata.groupby('landmark')[['dockcount']].sum()/stationdata[['dockcount']].\
sum()
```

	dockcount
landmark	
Mountain View	0.095823
Palo Alto	0.061425
Redwood City	0.094185
San Francisco	0.544636
San Jose	0.203931

What are the dockcounts by landmark and station name?

```
stationdata.groupby(['landmark','name'])[['dockcount']].sum()
```

		dockcount
landmark	**name**	
Mountain View	**Castro Street and El Camino Real**	11
	Evelyn Park and Ride	15
	Mountain View Caltrain Station	23
	Mountain View City Hall	15
	Rengstorff Avenue / California Street	15
	San Antonio Caltrain Station	23
	San Antonio Shopping Center	15
Palo Alto	**California Ave Caltrain Station**	15
	Cowper at University	11
	Palo Alto Caltrain Station	23
	Park at Olive	15
	University and Emerson	11
Redwood City	**Broadway at Main**	15
	Franklin at Maple	15
	Mezes Park	15
	Redwood City Caltrain Station	25
	Redwood City Medical Center	15
	Redwood City Public Library	15
	San Mateo County Center	15
	2nd at Folsom	19
	2nd at South Park	15

What are the total number of bicycle trips by landmark?

When looking at number of bicycle trips based on landmark, we see that 129,853 or 90% of total observations in the dataset were in San Francisco. 6% of total observations were in San Jose while only 1% were in Palo Alto.

```
dmerge4.groupby("landmark")[['duration_i']].aggregate(np.size)
```

	duration_i
landmark	
Mountain View	2728
Palo Alto	1706
Redwood City	793
San Francisco	129853
San Jose	8935

What are average durations by landmark?

Looking at average duration by landmark, San Francisco has the lowest duration followed by San Jose while Palo Alto had the highest duration. We can hypothesize that this may be partly due to fewer docks in Palo Alto or greater distances to travel as well as the subscription profile of Palo Alto bicycle users.

```
dmerge4.groupby("landmark")[['duration_i']].aggregate(np.mean)
```

	duration_i
landmark	
Mountain View	44.682918
Palo Alto	77.262603
Redwood City	49.084489
San Francisco	18.899656
San Jose	23.240963

What are average duration and dockcount by landmark and time of day for first six months of the bike share system?

```
timelandmark = dmerge4.groupby(['timeofday','landmark']).agg({'dockcount' : np.m\
ean, 'duration_f': np.mean,  }).reset_index()
timelandmark

timelandmark.dockcount.max()
```

	timeofday	landmark	duration_f	dockcount
0	evening	Mountain View	34.096655	18.079393
1	evening	Palo Alto	66.580671	14.565495
2	evening	Redwood City	50.191700	18.198381
3	evening	San Francisco	16.224655	19.349130
4	evening	San Jose	18.737437	17.385234
5	mid_day	Mountain View	82.030353	18.465995
6	mid_day	Palo Alto	109.684100	14.439331
7	mid_day	Redwood City	76.925088	17.982456
8	mid_day	San Francisco	27.520457	19.436506
9	mid_day	San Jose	23.002496	16.776488
10	morning	Mountain View	29.786291	19.715105
11	morning	Palo Alto	52.318829	16.978308
12	morning	Redwood City	22.378051	22.647059
13	morning	San Francisco	15.172821	19.948991
14	morning	San Jose	18.945759	19.782161
15	night	Mountain View	126.402177	20.061224
16	night	Palo Alto	83.482982	14.017544
17	night	Redwood City	65.530455	20.000000
18	night	San Francisco	19.055272	19.432370
19	night	San Jose	35.876195	18.286119

Where and at when was the highest average dockcount?

The highest dockcount was in Redwood City in the morning.

```
timelandmark.dockcount.max()
```

```
22.647058823529413
```

What are the top five longest average duration grouped by time of day and landmark?

The highest average duration was in Palo Alto in the wee hours.

```
timelandmark.duration_f.order(ascending=False)[:5]
```

```
21    150.501852
24    127.363475
15    126.402177
6     109.684100
16     83.482982
Name: duration_f, dtype: float64
```

What are the average dockcount and durations by time of day and landmark for the first six months of the bike share system?

Pandas produces nice readable multi-index tables.

```
np.round(dmerge4.groupby(['timeofday','landmark']).agg({'dockcount' : np.mean, '\
duration_i': np.mean,  }))
```

		dockcount	duration_i
timeofday	landmark		
evening	Mountain View	18	34
	Palo Alto	15	67
	Redwood City	18	50
	San Francisco	19	16
	San Jose	17	19
mid_day	Mountain View	18	82
	Palo Alto	14	110
	Redwood City	18	77
	San Francisco	19	28
	San Jose	17	23
morning	Mountain View	20	30
	Palo Alto	17	52
	Redwood City	23	22
	San Francisco	20	15
	San Jose	20	19
night	Mountain View	20	126
	Palo Alto	14	83
	Redwood City	20	66
	San Francisco	19	19
	San Jose	18	36
	Mountain View	21	82

Grouping and aggregating time-series data

Let’s obtain mean values by day of the month by creating a grouped object from the day component of the timestamp and applying an aggregation function. We want to know if there is higher demand or trip duration at certain days of the month.

```
day_means = dmerge4.groupby('day').aggregate(np.mean)
day_means
```

	tripid	duration	startterminal	endterminal	bike	max_temperature_f	mean_temperature_f	min_temperaturef	max_dew_point_f	meandew_point_f
day										
1	73878.288738	1816.236080	56.575208	56.547493	432.526833	69.336105	60.620559	51.125976	51.413706	46.602923
2	80351.443674	1273.880168	57.369151	57.382025	433.812825	66.287447	58.723942	50.482298	51.430800	47.488240
3	98238.235528	1135.100522	57.273354	57.356113	436.655799	63.523302	56.361338	49.060397	47.898433	42.093626
4	94721.030403	1101.818939	57.161696	57.186871	437.853021	64.680093	56.829783	48.094694	44.157242	39.132843
5	95326.050161	1173.435091	56.902045	56.801292	432.594833	65.471259	56.496878	47.178902	44.220883	37.509580
6	94048.015109	1202.085258	57.029570	57.127131	439.644507	68.017052	58.334556	48.195985	49.679258	42.862724
7	89738.111635	1146.163488	57.201200	57.143561	438.932719	71.251125	62.068138	52.430684	52.893936	45.455967
8	84325.793095	1386.051549	57.085836	57.125325	436.400804	67.203358	59.201939	50.980137	51.354694	47.276425
9	87579.505991	1144.179954	57.035023	57.111521	439.260829	64.475576	56.658986	48.722811	47.882949	43.535714
10	100164.634270	941.555230	57.384660	57.602459	434.971507	63.406518	56.257026	48.694184	49.707650	46.169789
11	96293.107869	982.990743	57.493862	57.524452	436.820487	62.358221	56.467498	50.252767	48.957939	46.354397
12	101958.670921	1180.773218	57.565482	57.644021	441.782250	63.251522	56.464559	49.364422	50.882388	46.859415
13	108374.918449	1113.472481	57.505760	57.708173	439.695922	65.535564	57.216310	48.298775	50.977144	46.897605
14	107914.239321	1331.676583	57.442601	57.464912	435.837338	68.113272	59.215294	50.194889	49.682113	45.583715
15	100907.198866	1433.743385	57.403612	57.506510	440.304704	71.000420	61.042839	50.341033	47.715456	41.582108
16	101172.588106	1202.279385	58.006985	57.971463	437.107763	71.047895	60.637597	49.693474	49.862303	42.463381
17	99680.420043	1112.376430	57.745106	57.859081	438.638884	68.343090	59.417523	49.817794	49.477418	44.306842
18	103648.507761	1175.414987	57.896343	57.921677	439.719536	65.938805	57.212310	48.155219	50.161285	45.867440

We can also create a grouped object using Pandas grouper. We can actually specify a resampling and a grouping using the Grouper. So, in this case I want the frequency to be monthly and group by landmark and subscription type.

```
three_grouper = dmerge4.groupby([pd.Grouper(freq='M',key='startdate'),"landmark"\
,"subscriptiontype"])[['duration_f']].mean().reset_index()
three_grouper
```

	startdate	landmark	subscriptiontype	duration_f
0	2013-08-31	Mountain View	Customer	39.107407
1	2013-08-31	Mountain View	Subscriber	108.082692
2	2013-08-31	Palo Alto	Customer	43.618750
3	2013-08-31	Palo Alto	Subscriber	16.942083
4	2013-08-31	Redwood City	Customer	101.255000
5	2013-08-31	Redwood City	Subscriber	9.770526
6	2013-08-31	San Francisco	Customer	67.455091
7	2013-08-31	San Francisco	Subscriber	12.192543
8	2013-08-31	San Jose	Customer	56.080820
9	2013-08-31	San Jose	Subscriber	11.547216
10	2013-09-30	Mountain View	Customer	182.846638
11	2013-09-30	Mountain View	Subscriber	8.582500
12	2013-09-30	Palo Alto	Customer	160.404739
13	2013-09-30	Palo Alto	Subscriber	9.989910
14	2013-09-30	Redwood City	Customer	70.426667
15	2013-09-30	Redwood City	Subscriber	6.197643
16	2013-09-30	San Francisco	Customer	46.922383
17	2013-09-30	San Francisco	Subscriber	10.614401
18	2013-09-30	San Jose	Customer	72.983308

By unstacking the data created in the grouper object, we now have duration by customer and subscriber by landmark. This is a nice way to present the data in tabular format.

```
three_grouper_unstack = dmerge4.groupby([pd.Grouper(freq='M',key='startdate'),\
"landmark","subscriptiontype[['duration']].mean().unstack().\
reset_index()three_grouper_unstack.head()
```

	startdate	landmark	duration	
subscriptiontype			Customer	Subscriber
0	2013-08-31	Mountain View	2346.444444	6485.000000
1	2013-08-31	Palo Alto	2617.075000	1016.541667
2	2013-08-31	Redwood City	6075.250000	586.157895
3	2013-08-31	San Francisco	4047.301633	731.557012
4	2013-08-31	San Jose	3364.852459	692.814433

```
three_grouper_unstack = dmerge4.groupby([pd.Grouper(freq='M',key='startdate'),"l\
andmark","subscriptiontype"])[['duration']].mean().unstack()
three_grouper_unstack.head()
```

		duration	
	subscriptiontype	Customer	Subscriber
startdate	landmark		
2013-08-31	Mountain View	2346.444444	6485.000000
	Palo Alto	2617.075000	1016.541667
	Redwood City	6075.250000	586.157895
	San Francisco	4047.301633	731.557012
	San Jose	3364.852459	692.814433

Average duration by month faceted by landmark

```
grouper1 = dmerge4.groupby([pd.Grouper(freq='M',key='startdate'),'landmark']).me\
an()[['duration_f']]
grouper1.head(10)
```

		duration_f
startdate	landmark	
2013-08-31	Mountain View	72.944340
	Palo Alto	33.615000
	Redwood City	25.680870
	San Francisco	44.081851
	San Jose	28.740570
2013-09-30	Mountain View	67.345988
	Palo Alto	108.553665
	Redwood City	17.353211
	San Francisco	24.351433
	San Jose	29.502793

Grouping on a variable with many values is also possible. In this case, we can see summary statistics on every single duration as an example.

```
dmerge4.groupby([pd.Grouper(freq='M',key='startdate'),'duration_f']).sum()[:5]
```

		tripid	duration	startterminal	endterminal	bike	max_temperature_f	mean_temperature_f	min_temperaturef	max_dew_point_f	meandew_point_f
startdate	duration_f										
2013-08-31	1.03	7416	62	60	60	511	71	64	57	57	56
	1.05	4576	63	66	66	520	74	68	61	61	58
	1.13	7250	68	35	35	100	76	69	62	61	59
	1.15	13518	138	122	122	965	149	133	117	118	114
	1.17	4607	70	10	10	661	81	72	63	62	61

5 rows × 36 columns

We can be very specific and the level of granularity can be very detailed. We can pinpoint the average duration for a very specific profile or demographic which can be very powerful for a data-driven decision making including marketing or civic infrastructure goals for the bike sharing system.

This next table gives an indication of monthly durations based on starting landmark and endstation. In other words, we get an idea of how long it takes to reach by bicycle and the differences between customers and subscribers all in one table. We can identify which customers are keeping the bicycles for longer periods of time.

```
np.round(dmerge4.groupby([pd.Grouper(freq='M',key='startdate'),"landmark","subsc\
riptiontype","endstation"])[['duration_f']].mean())
```

```
np.round(dmerge4.groupby([pd.Grouper(freq='M',key='startdate')
,"landmark","subscriptiontype","endstation"])[['duration_f']].mean())
```

				duration_f
startdate	landmark	subscriptiontype	endstation	
2013-08-31	Mountain View	Customer	Evelyn Park and Ride	26
			Mountain View Caltrain Station	61
			Mountain View City Hall	11
			Park at Olive	34
			Rengstorff Avenue / California Street	16
			San Antonio Caltrain Station	52
		Subscriber	Evelyn Park and Ride	22
			Mountain View Caltrain Station	8
			Mountain View City Hall	215
			Park at Olive	453
			Rengstorff Avenue / California Street	10
			San Antonio Caltrain Station	15
	Palo Alto	Customer	California Ave Caltrain Station	19
			Cowper at University	24
			Palo Alto Caltrain Station	105
			Park at Olive	17
			Rengstorff Avenue / California Street	25
			San Antonio Caltrain Station	17
			University and Emerson	62

Let’s look at the breakdown of average duration by both landmark and subscription type.

We can see some clear differences between customers and subscribers. In most cases, subscribers are taking short trips.

```
np.round(dmerge4.groupby([pd.Grouper(freq='M',key='startdate'),"landmark","subsc\
riptiontype"])[['duration_f']].mean()).unstack()
```

		duration_f	
	subscriptiontype	Customer	Subscriber
startdate	landmark		
2013-08-31	Mountain View	39	108
	Palo Alto	44	17
	Redwood City	101	10
	San Francisco	67	12
	San Jose	56	12
2013-09-30	Mountain View	183	9
	Palo Alto	160	10
	Redwood City	70	6
	San Francisco	47	11
	San Jose	73	10
2013-10-31	Mountain View	113	12
	Palo Alto	84	12
	Redwood City	131	29
	San Francisco	53	10
	San Jose	104	9
2013-11-30	Mountain View	135	6
	Palo Alto	249	10
	Redwood City	175	6
	San Francisco	59	10
	San Jose	81	8

Let's look at each level of duration on an hourly basis.

```
dmerge4.groupby([pd.Grouper(freq='H',key='startdate'),'duration_f']).mean().head()
```

		tripid	duration	startterminal	endterminal	bike	max_temperature_f	mean_temperature_f	min_temperaturef	max_dew_point_f	meandew_point_f
startdate	duration_f										
2013-08-29 09:00:00	2.90	4069	174	64	64	288	74	68	61	61	58
	2.97	4086	178	45	45	379	74	68	61	61	58
	3.63	4081	218	27	27	150	80	70	64	65	61
	4.78	4084	287	27	27	138	80	70	64	65	61
	12.73	4080	764	66	69	315	74	68	61	61	58

5 rows × 36 columns

In this table, we are looking at hourly data by landmark.

```
hourlytry = dmerge4.groupby([pd.Grouper(freq='H',key='startdate'),'landmark'],as\
_index=False)[['duration_f']].mean()
hourlytry
```

	startdate	landmark	duration_f
0	2013-08-29 09:00:00	Mountain View	4.205000
1	2013-08-29 09:00:00	Redwood City	63.820000
2	2013-08-29 09:00:00	San Francisco	14.695556
3	2013-08-29 10:00:00	Mountain View	6.325000
4	2013-08-29 10:00:00	Palo Alto	18.775000
5	2013-08-29 10:00:00	San Francisco	37.356250
6	2013-08-29 10:00:00	San Jose	21.225000
7	2013-08-29 11:00:00	San Francisco	40.284737
8	2013-08-29 11:00:00	San Jose	2.957500
9	2013-08-29 12:00:00	Palo Alto	11.658000
10	2013-08-29 12:00:00	Redwood City	8.353333
11	2013-08-29 12:00:00	San Francisco	16.223173
12	2013-08-29 12:00:00	San Jose	12.387500
13	2013-08-29 13:00:00	Mountain View	1628.550000
14	2013-08-29 13:00:00	Palo Alto	23.544286
15	2013-08-29 13:00:00	San Francisco	53.158529
16	2013-08-29 13:00:00	San Jose	26.937273
17	2013-08-29 14:00:00	Mountain View	15.880000
18	2013-08-29 14:00:00	San Francisco	24.205366
19	2013-08-29 14:00:00	San Jose	6.595000

Group operations on a datetime index

```
dmerge4.groupby(dmerge4.index.month).agg(['count','sum','mean','min','max'])[['d\
uration_f']]
```

	duration_f				
	count	sum	mean	min	max
1	24428	412751.48	16.896655	1.00	9772.60
2	19024	332260.54	17.465335	1.02	3077.22
8	2102	90672.74	43.136413	1.03	4022.23
9	25243	663946.54	26.302204	1.00	9958.62
10	29105	578249.15	19.867691	1.00	7156.40
11	24219	500883.27	20.681418	1.00	12037.27
12	19894	375728.66	18.886532	1.00	10322.03

Let’s make a table of average dockcount by each precipitation level and subscription type.

```
i = pd.DataFrame(dmerge4.groupby(['precipitation_in','subscriptiontype']).dockco\
unt.mean()).reset_index()
```

```
i
```

	precipitation_in	subscriptiontype	dockcount
0	0	Customer	18.940762
1	0	Subscriber	19.472218
2	0.01	Customer	19.660550
3	0.01	Subscriber	19.833333
4	0	Customer	18.942500
5	0	Subscriber	19.528072
6	0.01	Customer	18.397083
7	0.01	Subscriber	19.411516
8	0.02	Customer	19.000000
9	0.02	Subscriber	18.341772
10	0.03	Customer	12.000000
11	0.03	Subscriber	17.666667
12	0.04	Customer	17.166667
13	0.04	Subscriber	18.132075
14	0.05	Subscriber	17.545455
15	0.06	Customer	19.915663
16	0.06	Subscriber	18.818182
17	0.07	Customer	11.000000
18	0.07	Subscriber	15.000000

What are the total number of observations in each of the initial six months?

Other than first month of operation, the distribution of observations distribution is pretty evenly spread out.

```
dmerge4.groupby(dmerge4.index.month).size()
```

```
1     24428
2     19024
8      2102
9     25243
10    29105
11    24219
12    19894
dtype: int64
```

```
dmerge4.groupby(dmerge4.index.month).size().order(ascending=False)[:5]
```

```
10    29105
9     25243
1     24428
11    24219
12    19894
dtype: int64
```

Working with multi-level indexes

Let’s create a multi-level index without grouping.

We’ll often want to perform queries on the full dataset.

```
dmerge4twolevels = dmerge4.set_index(['landmark','subscriptiontype'])
dmerge4twolevels
```

		lubstartdate	startstation	tripid	duration	startdate	startterminal	enddate	endstation	endterminal	bike	...	day	hour	month	timeofday	diff	hourdiff	durationminutes
landmark	subscriptiontype																		
	Subscriber	2013-08-29	2nd at Folsom	4636	186	2013-08-29 15:11:00	62	2013-08-29 15:14:00	2nd at Townsend	61	366	...	29	15	8	evening	00:03:00	0	3.100000
	Subscriber	2013-08-29	2nd at Folsom	4820	813	2013-08-29 17:35:00	62	2013-08-29 17:48:00	Townsend at 7th	65	409	...	29	17	8	evening	00:13:00	0	13.550000
	Customer	2013-08-29	2nd at Folsom	5001	236	2013-08-29 20:00:00	62	2013-08-29 20:04:00	Market at Sansome	77	567	...	29	20	8	night	00:04:00	0	3.933333
	Subscriber	2013-08-29	2nd at Folsom	4812	186	2013-08-29 17:30:00	62	2013-08-29 17:33:00	2nd at Folsom	62	409	...	29	17	8	evening	00:03:00	0	3.100000
	Customer	2013-08-29	2nd at Folsom	4946	827	2013-08-29 19:07:00	62	2013-08-29 19:21:00	Embarcadero at Vallejo	48	342	...	29	19	8	evening	00:14:00	0	13.783333
	Subscriber	2013-08-29	2nd at Folsom	4390	825	2013-08-29 12:33:00	62	2013-08-29 12:47:00	2nd at South Park	64	331	...	29	12	8	mid_day	00:14:00	0	13.750000
	Subscriber	2013-08-29	2nd at Folsom	4939	816	2013-08-29 19:01:00	62	2013-08-29 19:14:00	Harry Bridges Plaza (Ferry Building)	50	539	...	29	19	8	evening	00:13:00	0	13.600000
	Customer	2013-08-29	2nd at Folsom	4949	1524	2013-08-29 19:11:00	62	2013-08-29 19:36:00	Commercial at Montgomery	45	400	...	29	19	8	evening	00:25:00	0	25.400000
	Subscriber	2013-08-29	2nd at Folsom	4582	335	2013-08-29 14:14:00	62	2013-08-29 14:20:00	2nd at Folsom	62	342	...	29	14	8	mid_day	00:06:00	0	5.583333

Creates a groupby object dataframe on which aggregate actions can be performed and can group by the multi-indexes.

```
grouplevel0 = dmerge4twolevels.groupby(level=0)
grouplevel0
```

```
grouplevel0.mean()
#same result as dmerge4.groupby('landmark').mean()
```

	tripid	duration	startterminal	endterminal	bike	max_temperature_f	mean_temperature_f	min_temperaturef	max_dew_point_f	meandew_point_f
landmark										
Mountain View	113506.415689	2681.418988	28.782991	28.980572	266.556085	67.732405	54.907991	44.681085	48.479106	42.678886
Palo Alto	92832.010551	4635.689918	35.669988	35.193435	208.521688	69.908558	59.389215	48.392145	48.810082	44.058617
Redwood City	82721.219420	2945.804540	23.134931	24.148802	237.013871	68.611602	59.397226	50.197982	51.931904	45.466583
San Francisco	101092.908674	1133.965253	61.825657	61.868821	455.360123	66.115962	58.201636	49.776278	49.848767	45.052167
San Jose	98496.783100	1394.662899	7.371684	7.563290	284.057862	69.322104	58.393397	46.986682	47.791046	42.897146

5 rows × 37 columns

```
grouplevel1 = dmerge4twolevels.groupby(level=1)
grouplevel1
```

```
grouplevel1.mean()
```

	tripid	duration	startterminal	endterminal	bike	max_temperature_f	mean_temperature_f	min_temperaturef	max_dew_point_f	meandew_point_f
subscriptiontype										
Customer	79116.721714	3629.519659	56.400586	56.781909	433.907238	68.987487	60.303115	51.149137	51.473953	46.698992
Subscriber	106806.906130	589.969564	57.538334	57.505486	437.863692	65.713868	57.602277	49.049971	49.218783	44.373692

2 rows × 37 columns

The level can also be specified by name.

```
dmerge4twolevels.groupby(level='subscriptiontype').sum()
dmerge4twolevels.sum(level='subscriptiontype')  #same as above
```

	tripid	duration	startterminal	endterminal	bike	max_temperature_f	mean_temperature_f	min_temperaturef	max_dew_point_f	meandew_point_f
subscriptiontype										
Customer	2402616605	110221253	1712773	1724353	13176895	2095012	1831285	1553297	1563161	1418155
Subscriber	12138284461	67048271	6539059	6535326	49761895	7468184	6546326	5574382	5593567	5042937

2 rows × 37 columns

Create the level without dropping it as a column

```
dmerge4twolevels.xs('Subscriber', level='subscriptiontype',drop_level=False)
```

		lubstartdate	startstation	tripid	duration	startdate	startterminal	enddate	endstation	endterminal	bike	...	day	hour	month	timeofday	diff	hourdiff	durationminutes
landmark	subscriptiontype																		
	Subscriber	2013-08-29	2nd at Folsom	4636	186	2013-08-29 15:11:00	62	2013-08-29 15:14:00	2nd at Townsend	61	366	...	29	15	8	evening	00:03:00	0	3.100000
	Subscriber	2013-08-29	2nd at Folsom	4820	813	2013-08-29 17:35:00	62	2013-08-29 17:48:00	Townsend at 7th	65	409	...	29	17	8	evening	00:13:00	0	13.550000
	Subscriber	2013-08-29	2nd at Folsom	4812	186	2013-08-29 17:30:00	62	2013-08-29 17:33:00	2nd at Folsom	62	409	...	29	17	8	evening	00:03:00	0	3.100000
	Subscriber	2013-08-29	2nd at Folsom	4390	825	2013-08-29 12:33:00	62	2013-08-29 12:47:00	2nd at South Park	64	331	...	29	12	8	mid_day	00:14:00	0	13.750000
	Subscriber	2013-08-29	2nd at Folsom	4939	816	2013-08-29 19:01:00	62	2013-08-29 19:14:00	Harry Bridges Plaza (Ferry Building)	50	539	...	29	19	8	evening	00:13:00	0	13.600000
	Subscriber	2013-08-29	2nd at Folsom	4582	335	2013-08-29 14:14:00	62	2013-08-29 14:20:00	2nd at Folsom	62	342	...	29	14	8	mid_day	00:06:00	0	5.583333
	Subscriber	2013-08-29	2nd at Folsom	4938	840	2013-08-29 19:01:00	62	2013-08-29 19:15:00	Harry Bridges Plaza (Ferry Building)	50	607	...	29	19	8	evening	00:14:00	0	14.000000
	Subscriber	2013-08-29	2nd at South Park	4557	130	2013-08-29 13:57:00	64	2013-08-29 13:59:00	2nd at South Park	64	371	...	29	13	8	mid_day	00:02:00	0	2.166667
	Subscriber	2013-08-29	2nd at South Park	4069	174	2013-08-29 09:08:00	64	2013-08-29 09:11:00	2nd at South Park	64	288	...	29	9	8	morning	00:03:00	0	2.900000

Querying data

By setting up the initial query, We can then change the inputs for interactive reporting of results. We can answer very specific questions by querying.

Let’s created a grouped object, apply aggregate functions, and then query the data.

In this table, we have the average duration for subscription type of customer only.

```
grouped1 = dmerge4.groupby(['landmark','subscriptiontype'])
group2 = grouped1[['duration']].agg([np.mean, np.std])
group2
query1 = group2.query('subscriptiontype == "Customer"')
query1
```

		duration	
		mean	std
landmark	subscriptiontype		
Mountain View	Customer	10679.844133	35553.063019
Palo Alto	Customer	8856.419476	39184.301869
Redwood City	Customer	11584.882716	23257.019736
San Francisco	Customer	3212.968434	9861.482292
San Jose	Customer	4675.431818	17693.747427

Return a dataset from a multi-index dataframe that contains only startterminal number 62.

```
#subseting and indexing a non-grouped multi index:
dmerge4twolevels.query('startterminal == 62')
```

		lubstartdate	startstation	tripid	duration	startdate	startterminal	enddate	endstation	endterminal	bike	...	day	hour	month	timeofday	diff	hourdiff	durationminutes
landmark	subscriptiontype																		
	Subscriber	2013-08-29	2nd at Folsom	4636	186	2013-08-29 15:11:00	62	2013-08-29 15:14:00	2nd at Townsend	61	366	...	29	15	8	evening	00:03:00	0	3.100000
	Subscriber	2013-08-29	2nd at Folsom	4820	813	2013-08-29 17:35:00	62	2013-08-29 17:48:00	Townsend at 7th	65	409	...	29	17	8	evening	00:13:00	0	13.550000
	Customer	2013-08-29	2nd at Folsom	5001	236	2013-08-29 20:00:00	62	2013-08-29 20:04:00	Market at Sansome	77	567	...	29	20	8	night	00:04:00	0	3.933333
	Subscriber	2013-08-29	2nd at Folsom	4812	186	2013-08-29 17:30:00	62	2013-08-29 17:33:00	2nd at Folsom	62	409	...	29	17	8	evening	00:03:00	0	3.100000
	Customer	2013-08-29	2nd at Folsom	4946	827	2013-08-29 19:07:00	62	2013-08-29 19:21:00	Embarcadero at Vallejo	48	342	...	29	19	8	evening	00:14:00	0	13.783333
	Subscriber	2013-08-29	2nd at Folsom	4390	825	2013-08-29 12:33:00	62	2013-08-29 12:47:00	2nd at South Park	64	331	...	29	12	8	mid_day	00:14:00	0	13.750000
	Subscriber	2013-08-29	2nd at Folsom	4939	816	2013-08-29 19:01:00	62	2013-08-29 19:14:00	Harry Bridges Plaza (Ferry Building)	50	539	...	29	19	8	evening	00:13:00	0	13.600000
	Customer	2013-08-29	2nd at Folsom	4949	1524	2013-08-29 19:11:00	62	2013-08-29 19:36:00	Commercial at Montgomery	45	400	...	29	19	8	evening	00:25:00	0	25.400000
	Subscriber	2013-08-29	2nd at Folsom	4582	335	2013-08-29 14:14:00	62	2013-08-29 14:20:00	2nd at Folsom	62	342	...	29	14	8	mid_day	00:06:00	0	5.583333

Query a multi-index dataframe for subscriptiontype of subscriber.

```
dmerge4twolevels.query('subscriptiontype == "Subscriber"')
```

		lubstartdate	startstation	tripid	duration	startdate	startterminal	enddate	endstation	endterminal	bike	...	day	hour	month	timeofday	diff	hourdiff	durationminutes
landmark	subscriptiontype																		
	Subscriber	2013-08-29	2nd at Folsom	4636	186	2013-08-29 15:11:00	62	2013-08-29 15:14:00	2nd at Townsend	61	366	...	29	15	8	evening	00:03:00	0	3.100000
	Subscriber	2013-08-29	2nd at Folsom	4820	813	2013-08-29 17:35:00	62	2013-08-29 17:48:00	Townsend at 7th	65	409	...	29	17	8	evening	00:13:00	0	13.550000
	Subscriber	2013-08-29	2nd at Folsom	4812	186	2013-08-29 17:30:00	62	2013-08-29 17:33:00	2nd at Folsom	62	409	...	29	17	8	evening	00:03:00	0	3.100000
	Subscriber	2013-08-29	2nd at Folsom	4390	825	2013-08-29 12:33:00	62	2013-08-29 12:47:00	2nd at South Park	64	331	...	29	12	8	mid_day	00:14:00	0	13.750000
	Subscriber	2013-08-29	2nd at Folsom	4939	816	2013-08-29 19:01:00	62	2013-08-29 19:14:00	Harry Bridges Plaza (Ferry Building)	50	539	...	29	19	8	evening	00:13:00	0	13.600000
	Subscriber	2013-08-29	2nd at Folsom	4582	335	2013-08-29 14:14:00	62	2013-08-29 14:20:00	2nd at Folsom	62	342	...	29	14	8	mid_day	00:06:00	0	5.583333
	Subscriber	2013-08-29	2nd at Folsom	4938	840	2013-08-29 19:01:00	62	2013-08-29 19:15:00	Harry Bridges Plaza (Ferry Building)	50	607	...	29	19	8	evening	00:14:00	0	14.000000
	Subscriber	2013-08-29	2nd at South Park	4557	130	2013-08-29 13:57:00	64	2013-08-29 13:59:00	2nd at South Park	64	371	...	29	13	8	mid_day	00:02:00	0	2.166667

What is the average duration in descending order by endstation when the startterminal is number 62?

We can get this type of granular data by querying. By setting up the initial query, we can then change the inputs for interactive reporting of results.

```
duration62 = dmerge4twolevels.query('startterminal == 62').groupby('endstation')\
['duration_f'].mean().order(ascending=False)
```

```
duration62
```

```
endstation
2nd at Folsom                                84.510405
Beale at Market                              28.968750
Civic Center BART (7th at Market)            25.788571
Embarcadero at Vallejo                       22.056000
Powell at Post (Union Square)                21.702200
Embarcadero at Sansome                       20.633906
Mechanics Plaza (Market at Battery)          18.325526
Golden Gate at Polk                          18.093333
San Francisco Caltrain (Townsend at 4th)     17.642372
South Van Ness at Market                     16.725909
Commercial at Montgomery                     15.988511
Market at 10th                               15.887500
Townsend at 7th                              15.233402
Grant Avenue at Columbus Avenue              15.224000
San Francisco City Hall                      14.389286
Davis at Jackson                             11.523784
Broadway St at Battery St                    11.013333
Washington at Kearney                        10.850811
Harry Bridges Plaza (Ferry Building)         10.394581
```

```
5th at Howard                                          9.919462
Powell Street BART                                     9.627881
Clay at Battery                                        9.399346
Post at Kearney                                        8.780957
Steuart at Market                                      8.627549
Embarcadero at Bryant                                  8.034000
Yerba Buena Center of the Arts (3rd @ Howard)          7.567119
Market at 4th                                          7.431879
San Francisco Caltrain 2 (330 Townsend)                7.380276
Embarcadero at Folsom                                  5.832000
Temporary Transbay Terminal (Howard at Beale)          5.819636
Spear at Folsom                                        5.556141
2nd at South Park                                      5.551429
2nd at Townsend                                        4.662283
Market at Sansome                                      4.334730
Howard at 2nd                                          3.326883
Name: duration_f, dtype: float64
```

Merging and concatenating

Let’s concatenate the morning monthly and evening monthly created in the time series section into a dataframe for the plotting section.

```
concatenated = pd.concat([morning_monthly,evening_monthly], keys=['morning', 'e\
vening'],axis=1)
concatenated
```

	morning	evening
	duration_f	duration_f
startdate		
2013-08-31	37.648913	37.240278
2013-09-30	17.973624	24.621531
2013-10-31	13.302877	16.752767
2013-11-30	13.209249	17.277542
2013-12-31	15.384694	14.980330
2014-01-31	11.679421	14.962542
2014-02-28	11.441120	15.718395

Merge same named column from two datasets but keep both columns but rename them with a suffix

```
merge = pd.merge(morning_monthly,evening_monthly,how='outer',left_index=True,\ r\
ight_index=True,suffixes=['_morning', '_evening'])
merge
```

	duration_f_morning	duration_f_evening
startdate		
2013-08-31	37.648913	37.240278
2013-09-30	17.973624	24.621531
2013-10-31	13.302877	16.752767
2013-11-30	13.209249	17.277542
2013-12-31	15.384694	14.980330
2014-01-31	11.679421	14.962542
2014-02-28	11.441120	15.718395

Ad-hoc data summary or string operations

Use a lambda function to get summary statistics of average duration by landmark.

The maximum average duration was in San Jose, the highest average duration was in Palo Alto and the lowest was in San Francisco.

```
dmerge4.groupby("landmark")[['duration_f']].apply(lambda x: x.describe()).unstac\
k()
```

	duration_f							
	count	**mean**	**std**	**min**	**25%**	**50%**	**75%**	**max**
landmark								
Mountain View	2728	44.690359	283.766147	1.03	4.100	5.18	13.080	9772.60
Palo Alto	1706	77.261483	459.154620	1.10	6.055	15.57	34.865	12037.27
Redwood City	793	49.096608	233.252426	1.00	3.800	4.73	7.830	3831.90
San Francisco	129853	18.899424	81.849843	1.00	6.000	8.98	13.280	7156.40
San Jose	8935	23.244376	173.987471	1.03	4.770	7.42	11.320	10322.03

Another use of the lambda function is to map a function.

For example sake, to check the length of a string.

```
#apply a lambda function
f = lambda x: len(str(x))
dmerge4[['landmark']].applymap(f)[:3]
```

	landmark
startdate	
2013-08-29 15:11:00	13
2013-08-29 17:35:00	13
2013-08-29 20:00:00	13

Here is another string operation by way of mapping a lambda function. Very useful.

```
dmerge4['endstation'].map(lambda x: x.startswith('Grant')).head()
```

```
startdate
2013-08-29 15:11:00    False
2013-08-29 17:35:00    False
```

```
2013-08-29 20:00:00    False
2013-08-29 17:30:00    False
2013-08-29 19:07:00    False
Name: endstation, dtype: bool
```

Make a grouped object without making it the index.

```
dmerge4.groupby('landmark', as_index=False).mean()
```

	landmark	tripid	duration	startterminal	endterminal	bike	max_temperature_f	mean_temperature_f	min_temperaturef	max_dew_point_f	...	lat	long	dockcount	day	hour
0	Mountain View	113506.415689	2681.418988	28.782991	28.980572	266.556085	67.732405	54.907991	44.681085	48.479106	...	37.394312	-122.083341	18.888563	15.521261	13.07
1	Palo Alto	92832.010551	4635.689918	35.669988	35.193435	208.521688	69.908558	59.389215	48.392145	48.810082	...	37.440460	-122.156431	15.171161	15.759086	13.46
2	Redwood City	82721.219420	2945.804540	23.134931	24.148802	237.013871	68.611602	59.397226	50.197982	51.931904	...	37.486049	-122.229363	19.766709	15.145019	12.63
3	San Francisco	101092.908674	1133.965253	61.825657	61.868821	455.360123	66.115962	58.201636	49.776278	49.848767	...	37.787144	-122.400225	19.565963	15.746290	13.17
4	San Jose	98496.783100	1394.662899	7.371684	7.563290	284.057862	69.322104	58.393397	46.986682	47.791046	...	37.334148	-121.892161	17.977952	15.714605	13.63

5 rows × 38 columns

Data Science Programming in Python - Visualization

Data visualization

Let’s make some visualizations as part of the iterative process of data munging, aggregating, and visualizing. Visualizations help us understand the data better. At best, exploratory data visualization inspires questions and informs our analysis while identifying trends and patterns to further learn from the data.

We will see that displaying similar information in a variety of ways and using different types and styles of plots can reveal even more about the data.

```
%matplotlib inline
```

```
import numpy as np
import pandas as pd
from datetime import datetime, time
```

```
from ggplot import *
import seaborn as sns
```

```
print pd.__version__
print np.__version__
```

```
0.14.1
1.9.0
```

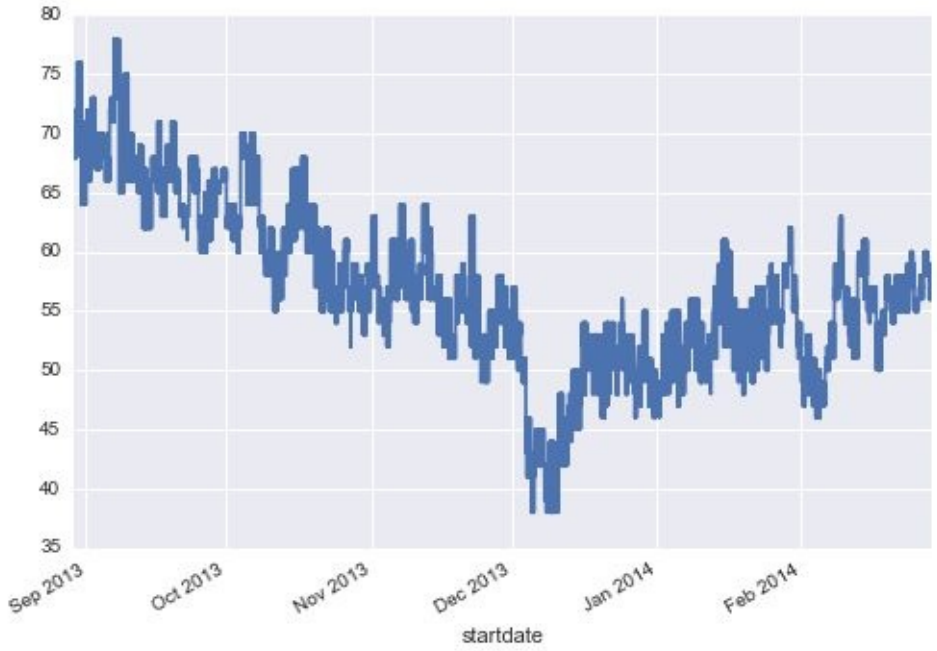
```
%load_ext ipycache
```

Exploratory data visualization

In practical data analysis, we want to make our plots in the most efficient and succinct way possible following an iterative data analysis process. A visualization will form more questions that lead to further visualizations using pandas, ggplot, and seaborn. In exploratory visualization, grouping & aggregating data and plotting are part of that iterative process. We want to learn from our data what further visualizations will provide insight and hopefully lead to more statistical questions and more visualizations.

Mean temperature

```
dmerge4.mean_temperature_f.plot()
```

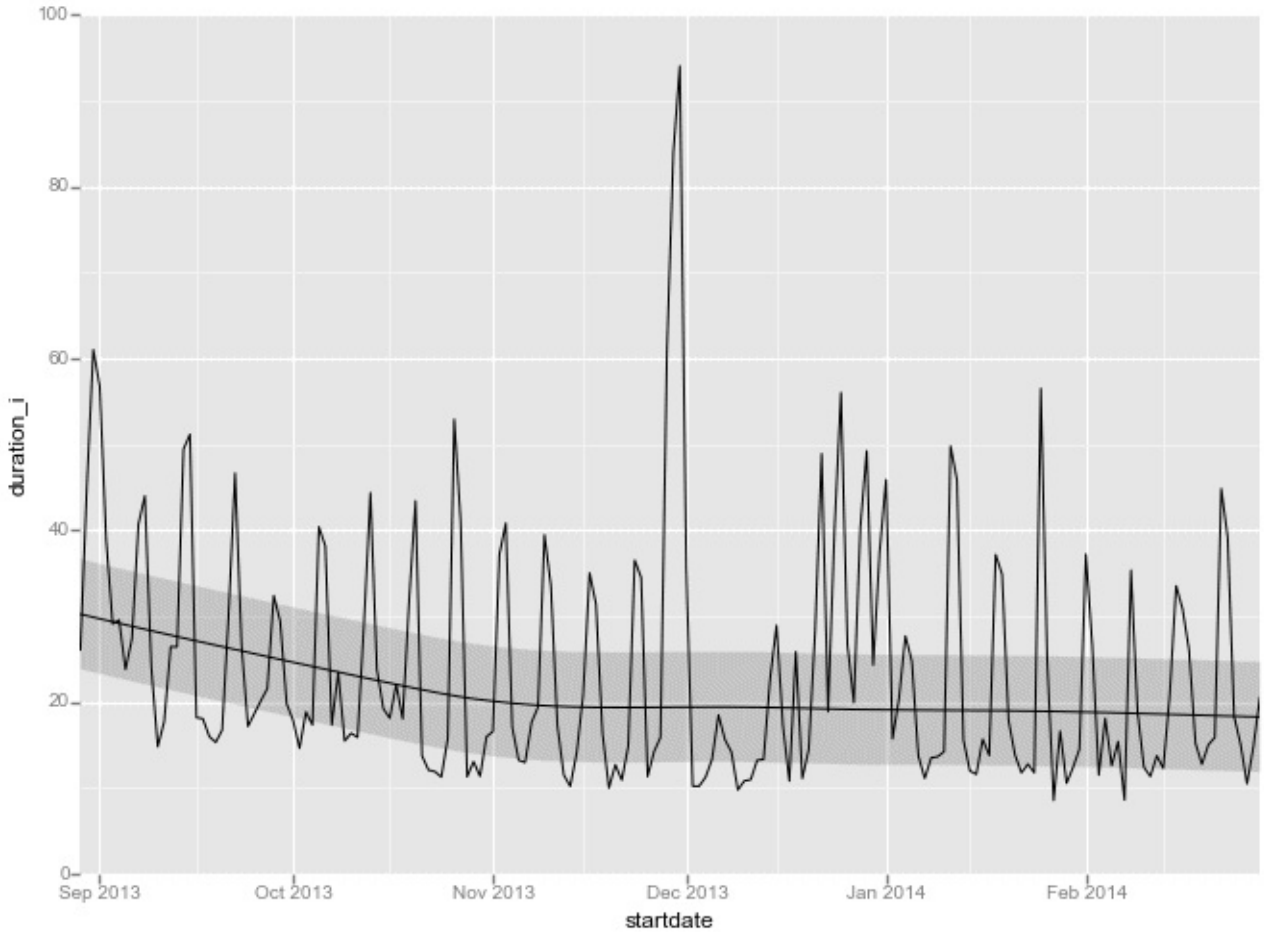

80
75
70
65
60
55
50
45
40
35
Sep 2013
Oct 2013
Nov 2013
Dec 2013
Jan 2014
Feb 2014
startdate

Plotting time-series data

Daily average duration for the first six months of operation of the bike sharing system

In this plot, we can see that most trips were under sixty minutes and fluctuated for the most part between under twenty minutes up to around fifty minutes depending on the time of day.

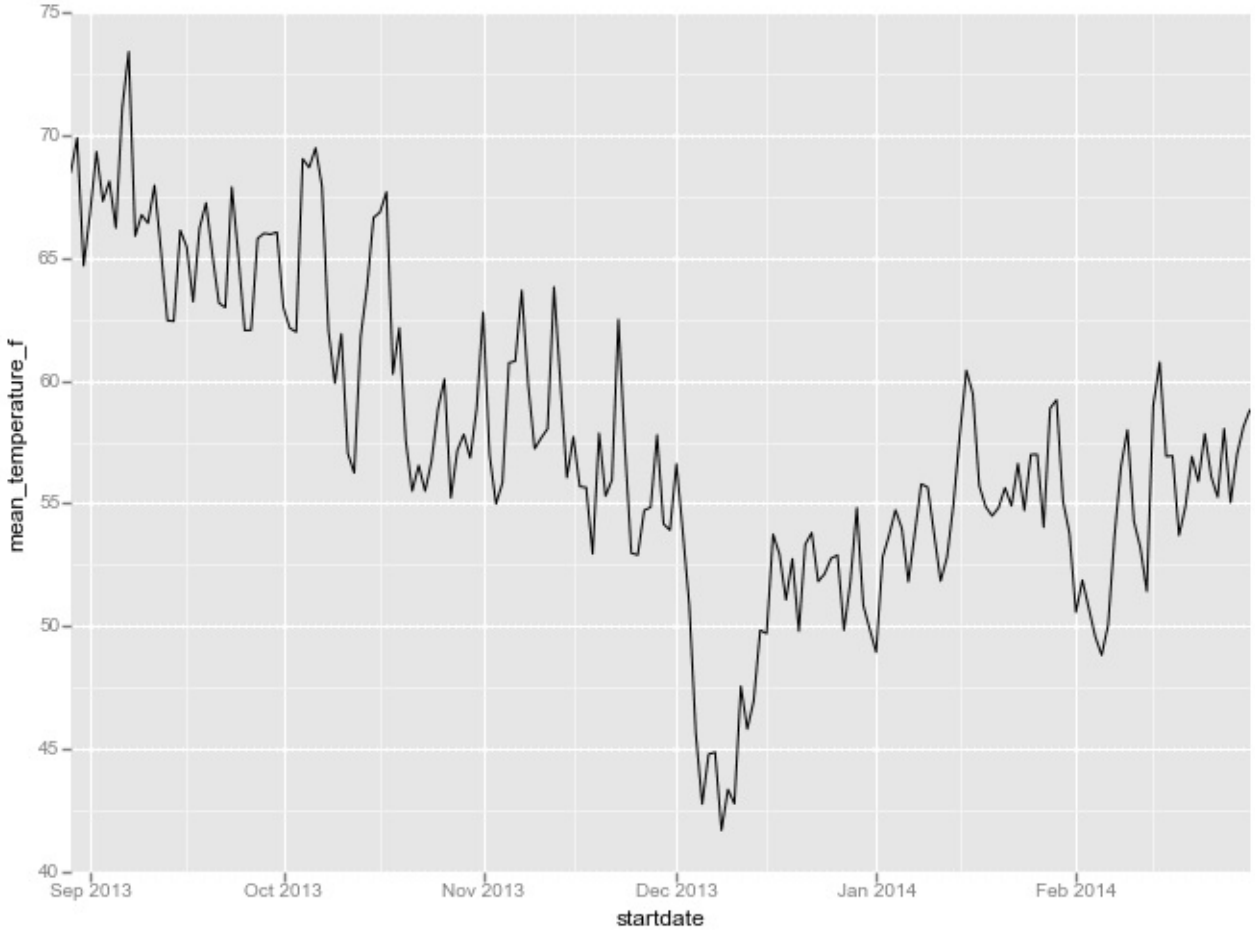
```
print (ggplot(aes(x='startdate', y='duration_i'), data=daily_means) + \
geom_line()) + geom_smooth()
```



Daily average mean temperature

Since San Francisco doesn’t have typically harsh winters, temperature did not seem to have much effect on length of bicycle trips

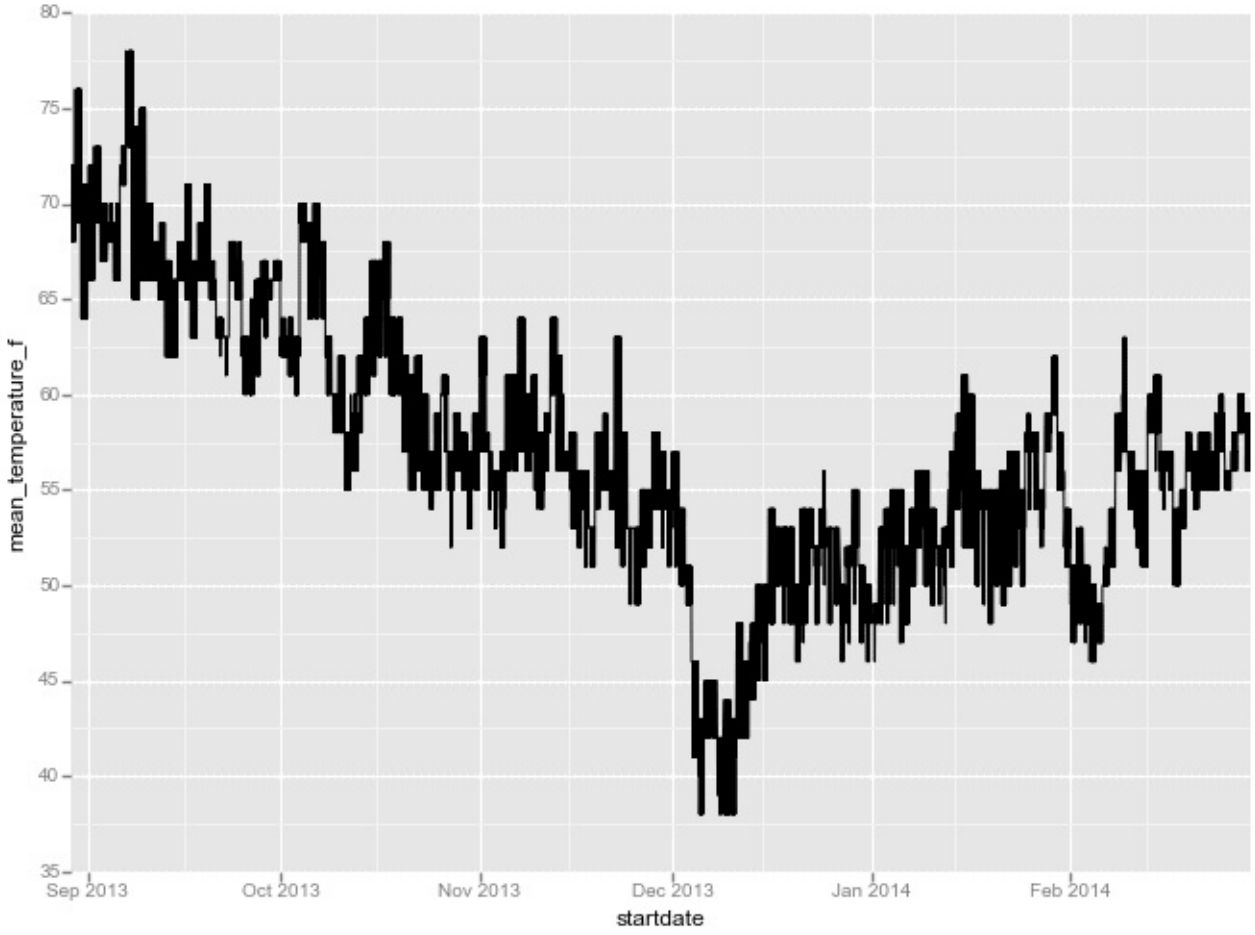
```
print (ggplot(aes(x='startdate', y='mean_temperature_f'), data=daily_means) + \
geom_line())
```



Mean temperatures for the entire dataset

This plot is included here to compare the above graph with plotting the entire dataset. Taking the daily mean in the previous plots allows us to consolidate multiple observations on a single day to give a less noisy graph. In this graph, every single observation is plotted. Taking the daily mean is a way to see the trend more cleanly and also will allow for further analysis on a daily data.

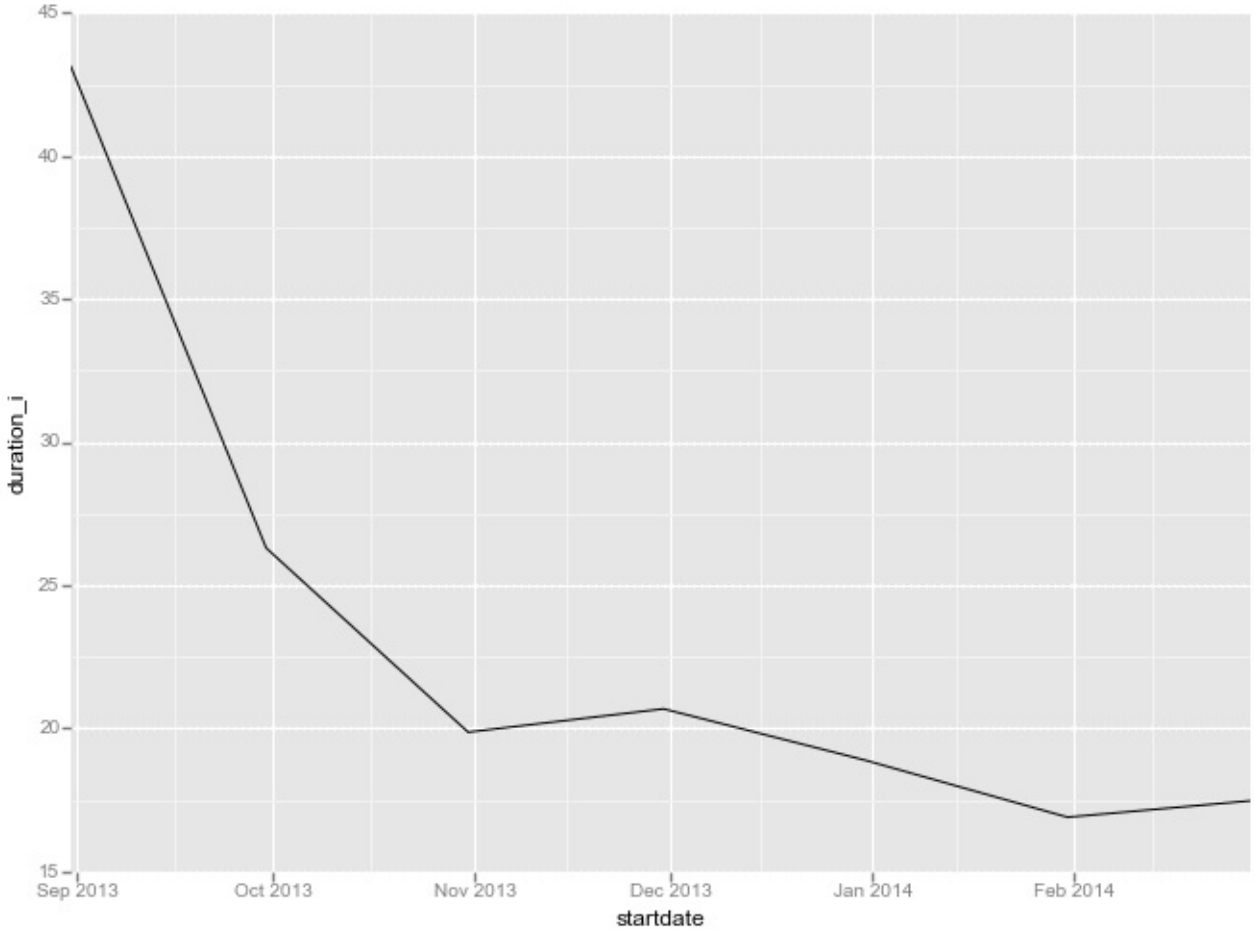
```
print (ggplot(aes(x='startdate', y='mean_temperature_f'), data=dmerge4) + \
geom_line())
```



Average duration on a monthly basis for the six month period

The monthly duration takes away some of the noise indicated by the daily data. We do actually see a declining trend as we go from Autumn to Winter.

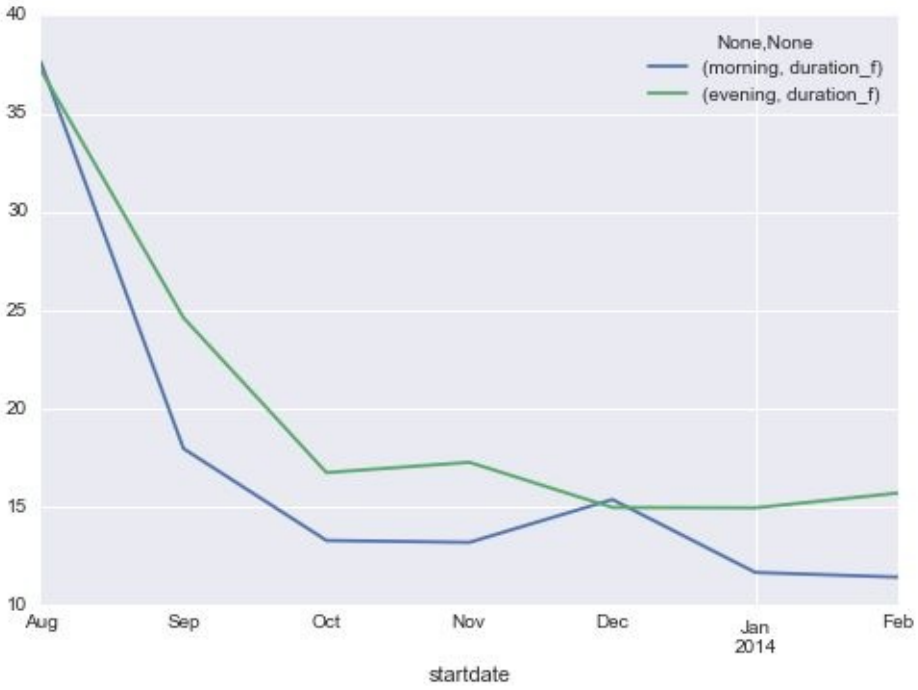
```
print (ggplot(aes(x='startdate', y='duration_i'), data=monthly_means) + \
geom_line())
```



Remember that we concatenated two time series intervals for the morning and evening commute. We can now see in this graph that the average duration by month is mostly higher in the evening than in the morning. Keeping in mind that August was the first month of operation, the plot also indicates that on average, the bicycle was returned to a dock within the initial thirty minute from start time from September to February.

Average duration by month faceted by morning and evening commuting hours

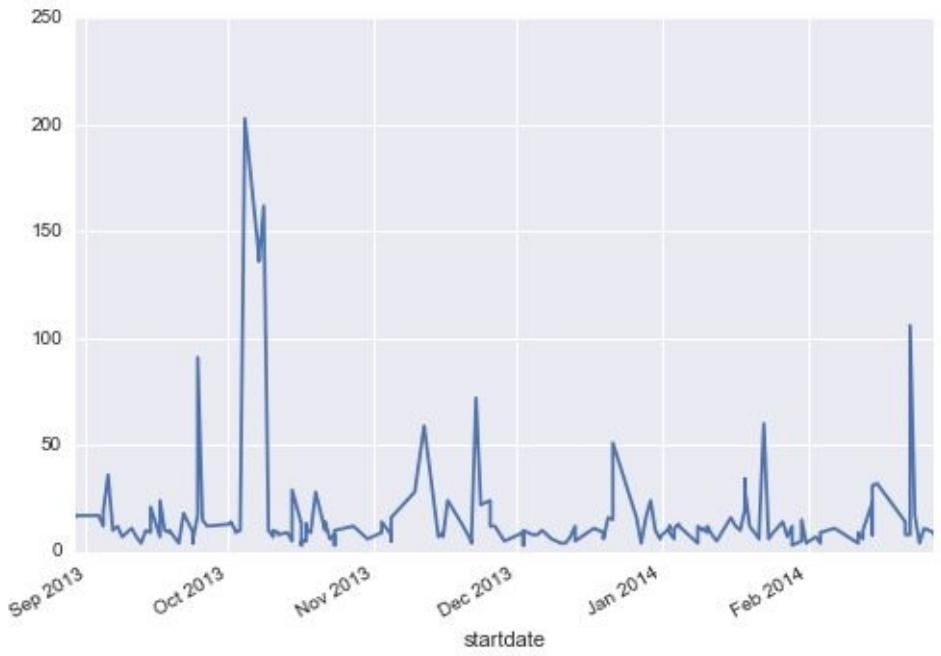
```
concatenated.plot()
```



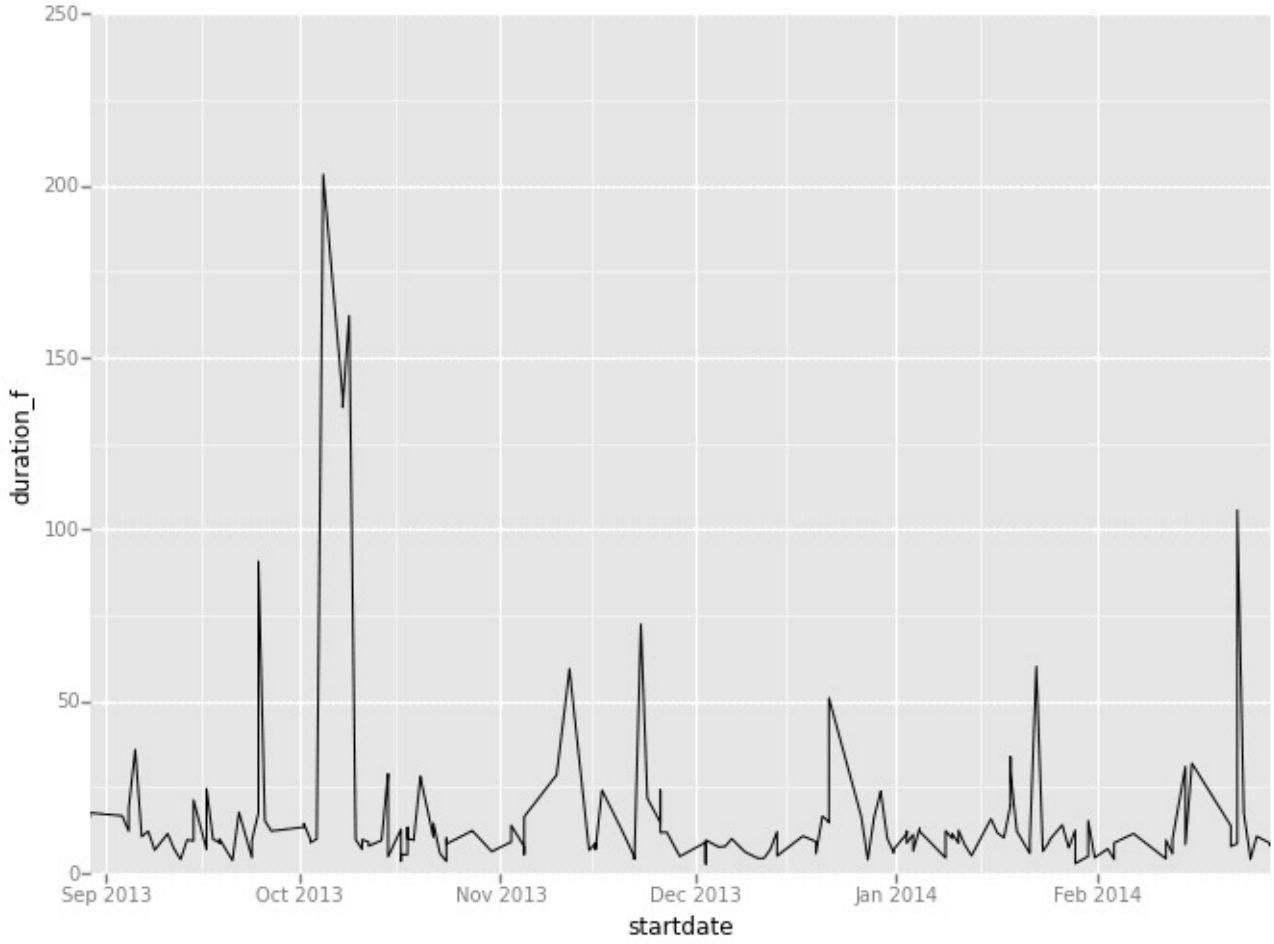
Average duration at 4pm for the six month period

What’s really nice about Pandas time series functionality is that we can also look at plots at specific times as well. Here is a plot of total duration at 4pm for the entire six month period. The plot indicates that a user had a bicycle for a really long time at 4pm in October but mostly the trip durations were under fifty minutes.

```
#total duration at particular time
dmerge4.duration_i.at_time(time(16, 0)).plot()
```



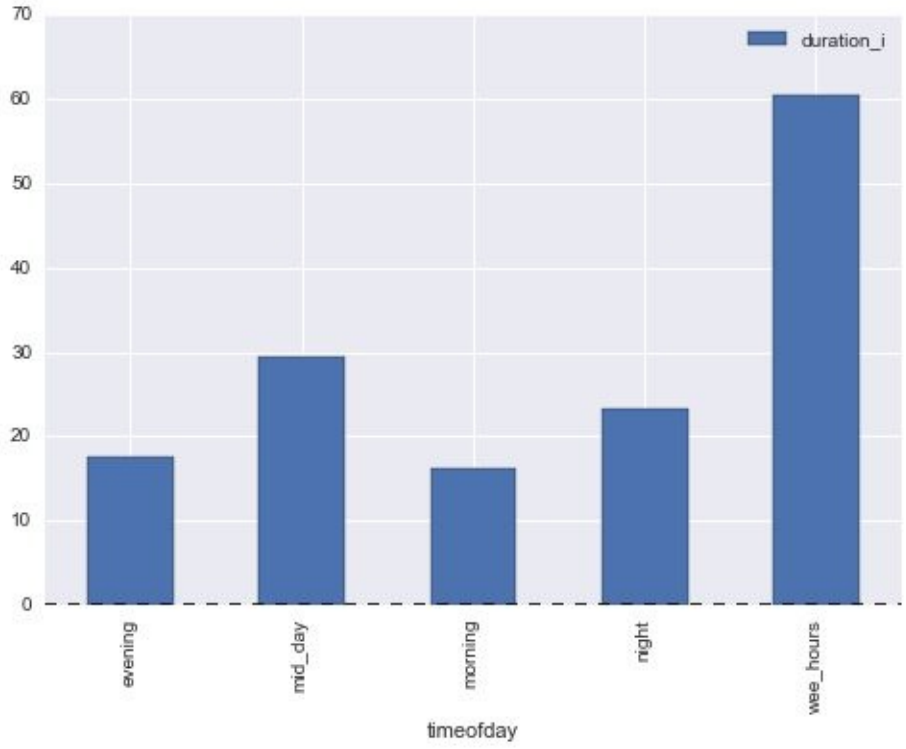
```
four_pm_df=pd.DataFrame(dmerge4.duration_f.at_time(time(16, 0))).reset_index()
ggplot(four_pm_df,aes(x='startdate',y='duration_f')) + geom_line()
#duration at 4pm for everyday the entire six month period
```



Average duration by time of day

Durations are shorter during the morning and evening commuting hours and slightly higher during the mid-day and at night. The data also indicates that people are keeping bicycles for a longer period of time after midnight until four am.

```
dmerge4.groupby('timeofday')[['duration_i']].mean().plot(kind='bar')
```

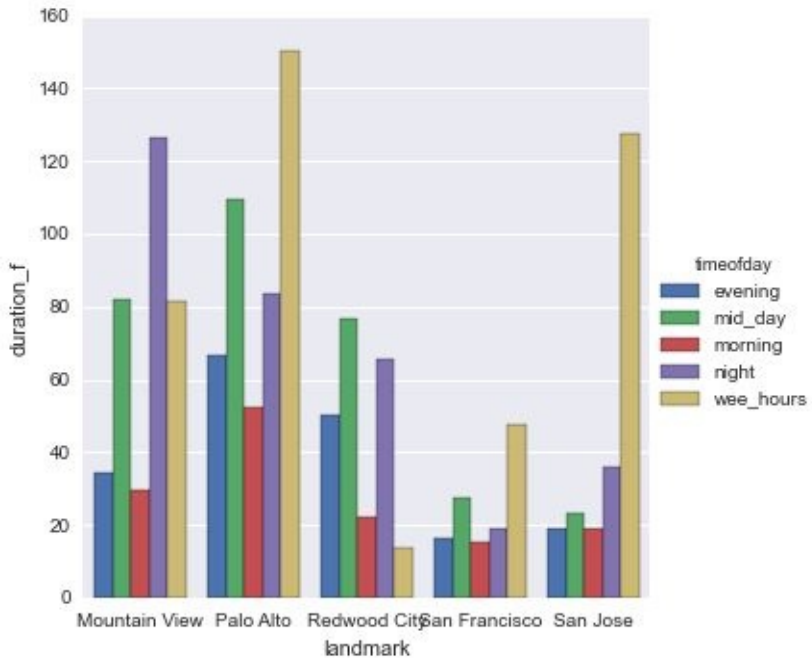


Dodged bar plot of average duration by landmark and time of day

This plot gives a really nice visual of average duration by landmark and time of day. Across the board, morning durations are shortest which would make sense for morning commuters. Morning commutes are highest in Palo Alto. Perhaps there is more traffic or greater distances to travel. In addition, we know that there are fewer bicycle docks in Palo Alto. Mid-day start times have longer trip duration except for Mountain View where evening bicycle trips are the longest.

Duration in San Francisco is the shortest and this is also the area that the highest number of docks in the dataset as shown in the grouping and aggregating section of the analysis.

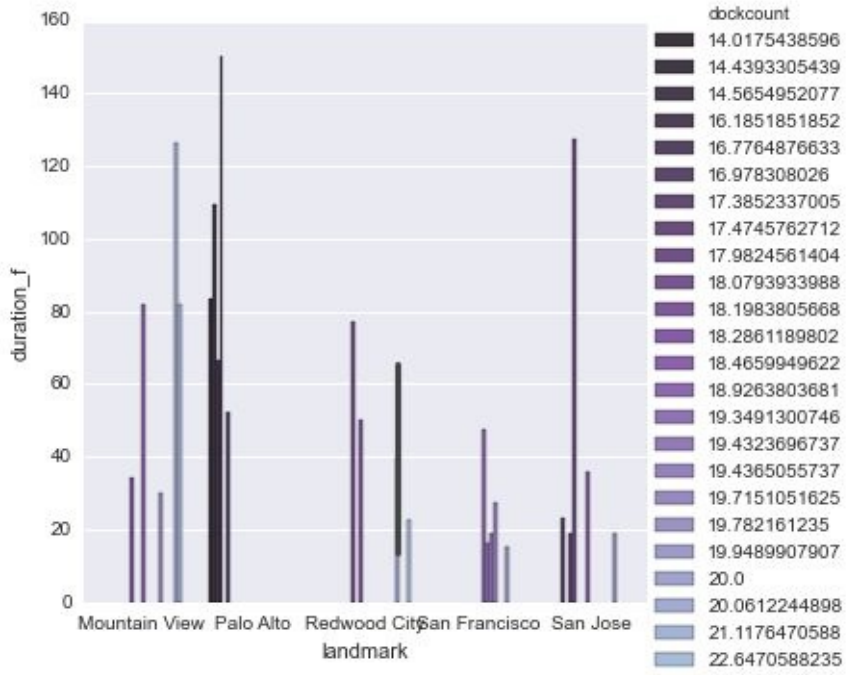
```
sns.factorplot("landmark", "duration_f", "timeofday", data=timelandmark);
```



Dockcounts by landmark and duration

It is evident in this plot that dockcounts were lowest in Palo Alto where average durations were also the highest.

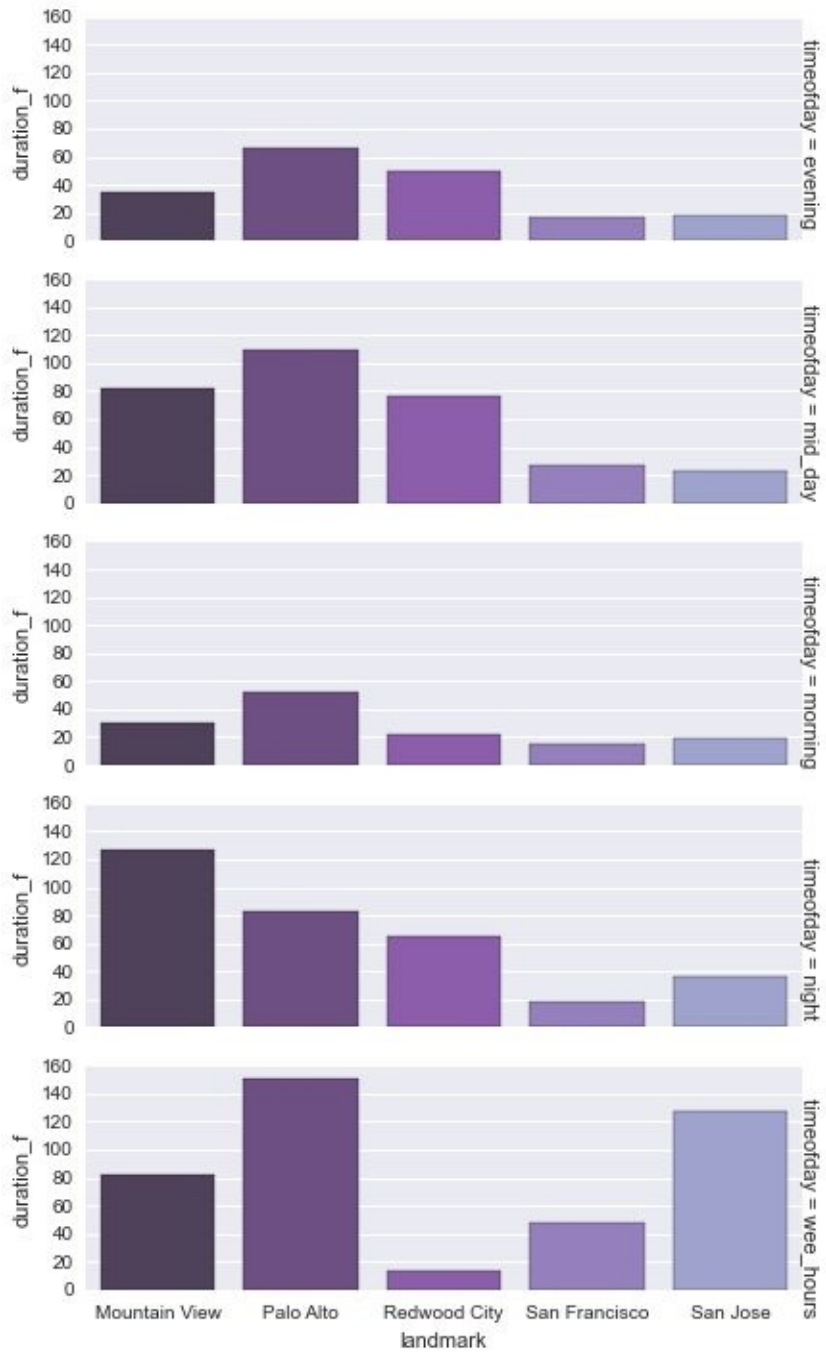
```
sns.factorplot("landmark", "duration_f", "dockcount", data=timelandmark,palette=\
"BuPu_d")
```



Average duration by landmark faceted by timeofday

This plot is nice because it allows us to visually see the differences in duration by landmark by categorical time of day.

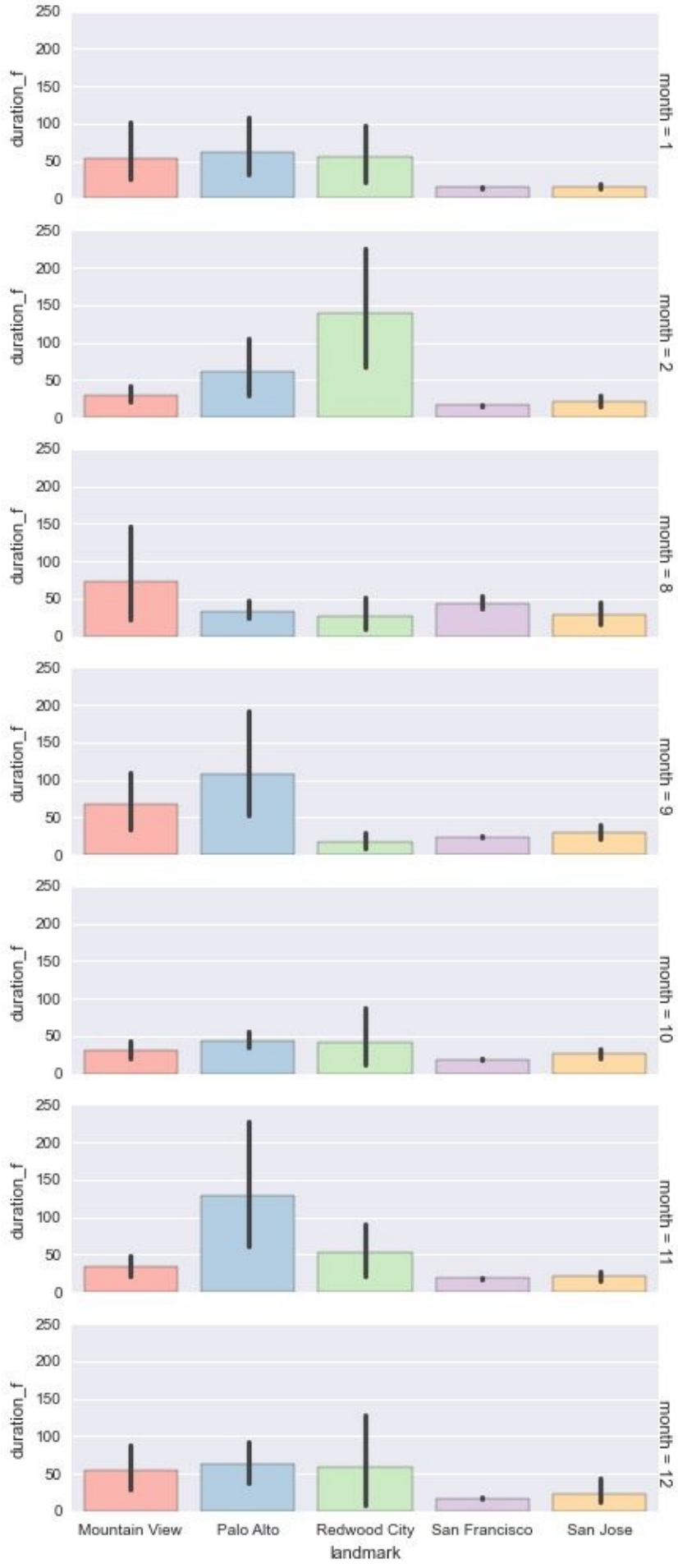
```
sns.factorplot("landmark","duration_f", data=timelandmark, row="timeofday",
margin_titles=True, aspect=3, size=2,palette="BuPu_d");
```



Average duration for each landmark faceted by month

Palo Alto had highest duration in almost every month except for February when Mountain View had the highest average duration.

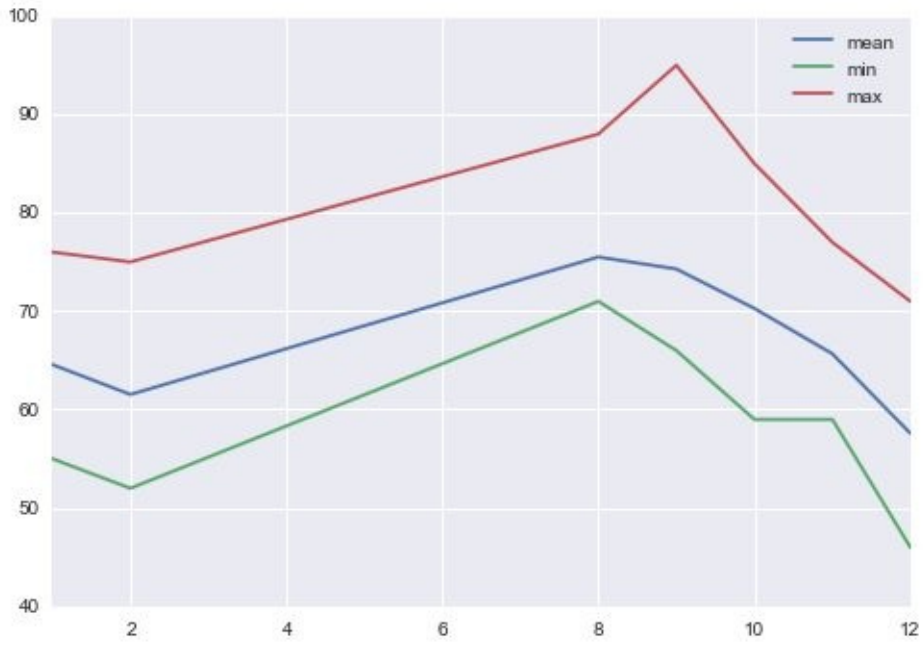
```
sns.factorplot("landmark","duration_f", data=dmerge4, row="month",
margin_titles=True, aspect=3, size=2,palette="Pastel1");
```


duration_f
250
200
150
100
50
0
month = 1
month = 2
month = 8
month = 9
month = 10
month = 11
month = 12
Mountain View
Palo Alto
Redwood City
San Francisco
San Jose
landmark

Plot of mean, minimum, and maximum temperature by month

This plot indicates plots for mean,minimum,and maximum summary statistics

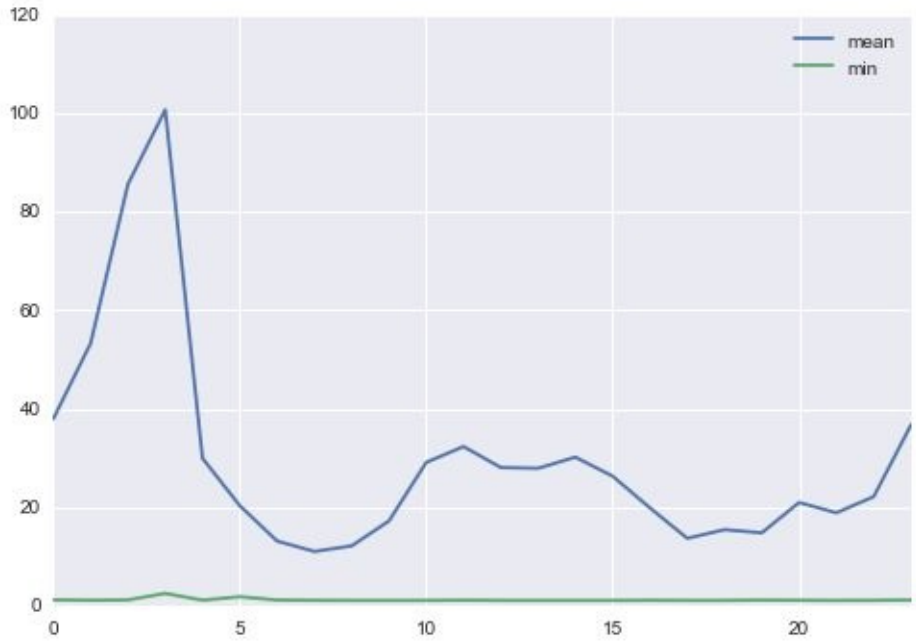
```
dmerge4.groupby(dmerge4.index.month)
['max_temperature_f'].agg(['mean','min','max']).plot()
```



Mean and minimum duration by hour of the day

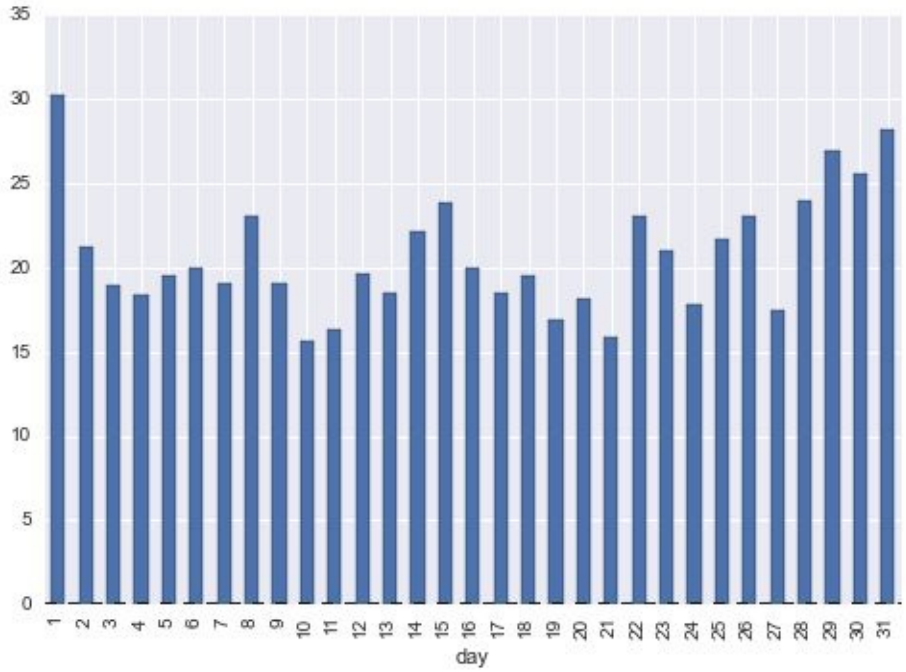
Duration is lowest in the morning and evening commute hours and highest in the evening and wee hours.

```
dmerge4.groupby(dmerge4.index.hour)['duration_f'].agg(['mean','min']).plot()
```



What are the average durations by day of the month?

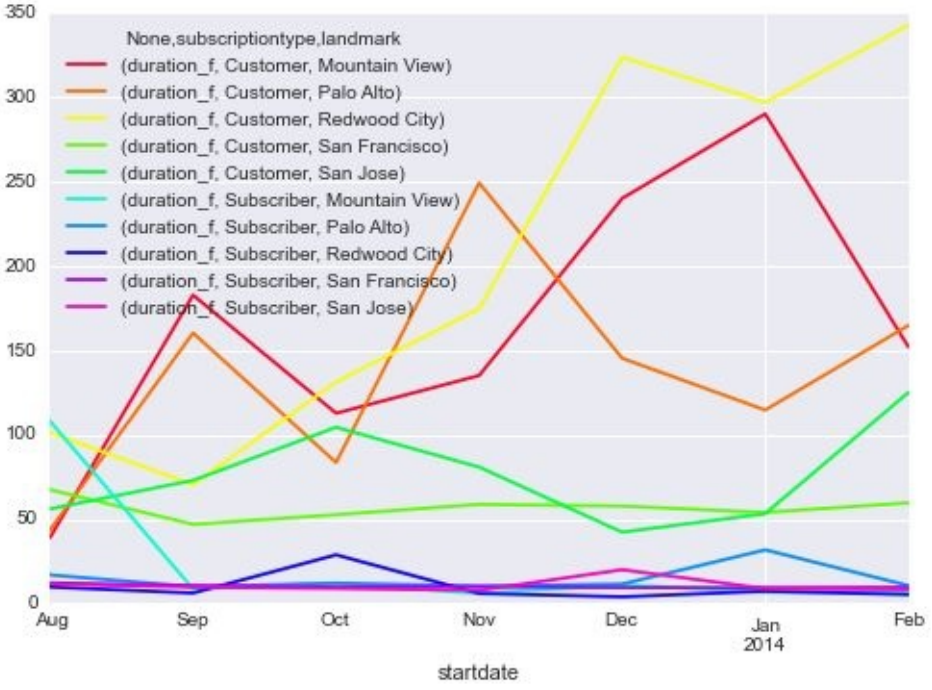
```
day_means['duration_f'].plot(kind='bar')
```



In what month did the highest duration occur and what was the subscription type?

The highest duration occurred in December for a subscription type of customer in Redwood City

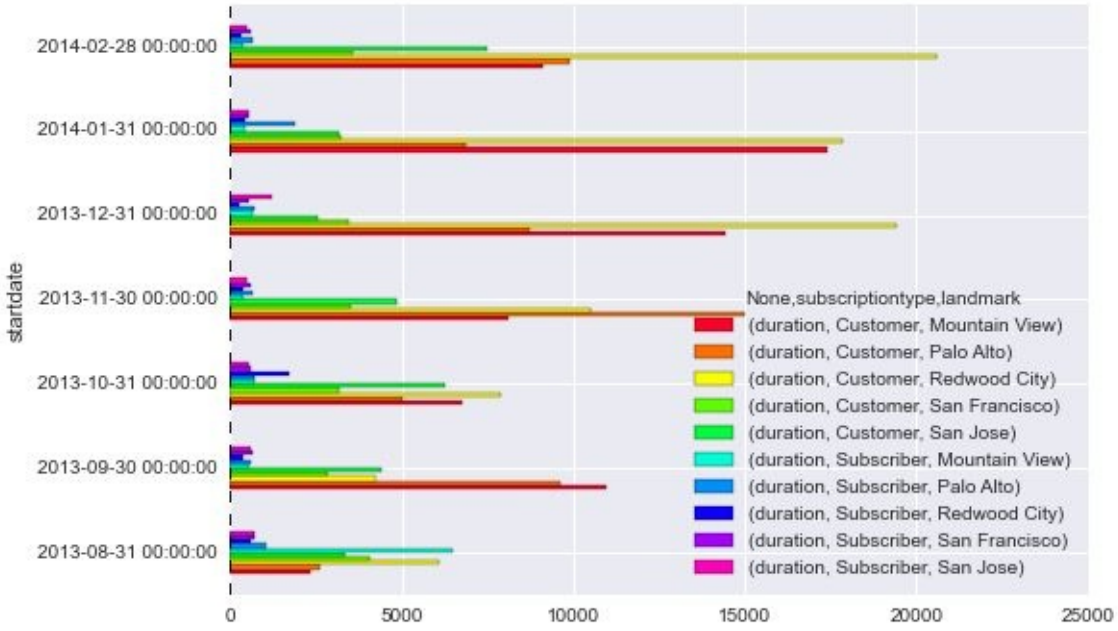
```
dmerge4.groupby([pd.Grouper(freq='M',key='startdate'),"landmark",\
"subscriptiontype"])[['duration_f']].mean().unstack().unstack().plot(colormap='g\
ist_rainbow')
```



Dodged bar plots by month of average duration by landmark and subscription type

In December, January, and February, the highest average duration was for a customer in Redwood City.

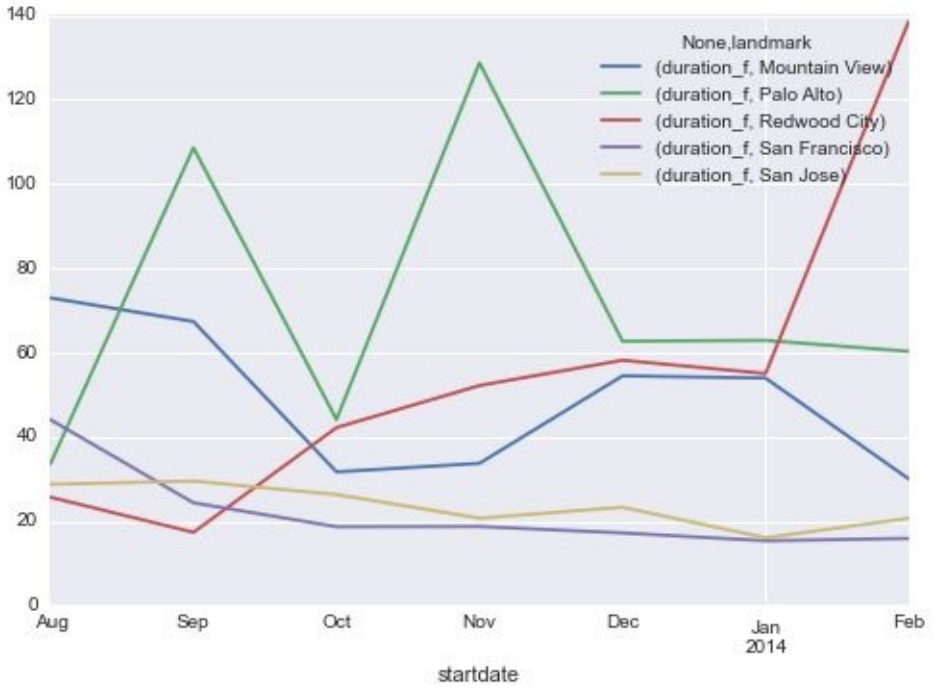
```
three_grouper_unstack_twice =
dmerge4.groupby([pd.Grouper(freq='M',key='startdate'),"landmark",\
"subscriptiontype"])[['duration']].mean().unstack().unstack()
three_grouper_unstack_twice.plot(kind="barh",colormap='gist_rainbow')
```



Monthly average duration by landmark

From September until December, Palo Alto had the highest average duration by month while San Jose and San Francisco had the lowest. Average duration by month increased in Redwood City from September to February.

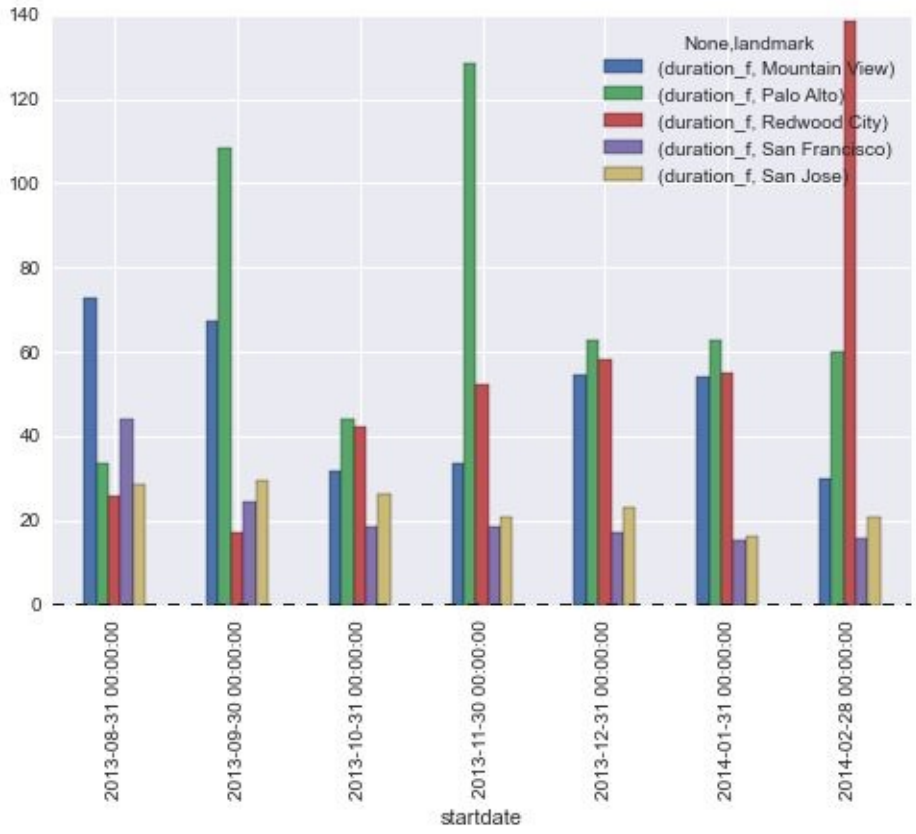
```
grouperl = dmerge4.groupby([pd.Grouper(freq='M',key='startdate'),\
'landmark']).mean()[['duration_f']]
grouperl.unstack().plot()
```



Dodged bar plot of monthly average duration by landmark

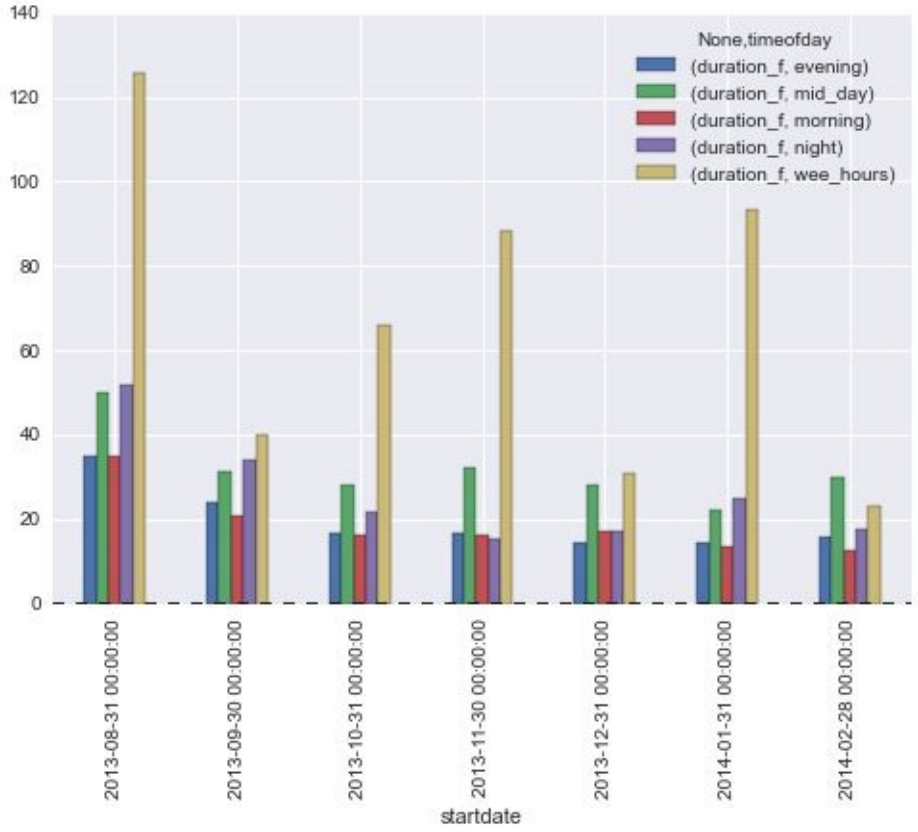
This dodged barplot shows that duration was highest for Redwood City in February and highest for Palo Alto in November. In every month, durations were lowest for both San Francisco and San Jose. We could hypothesize this is due to commuting, traffic patterns, and demand for bicycles. In any case, knowing where durations are higher will influence demand and dockcount.

```
grouperl.unstack().plot(kind="bar")
```



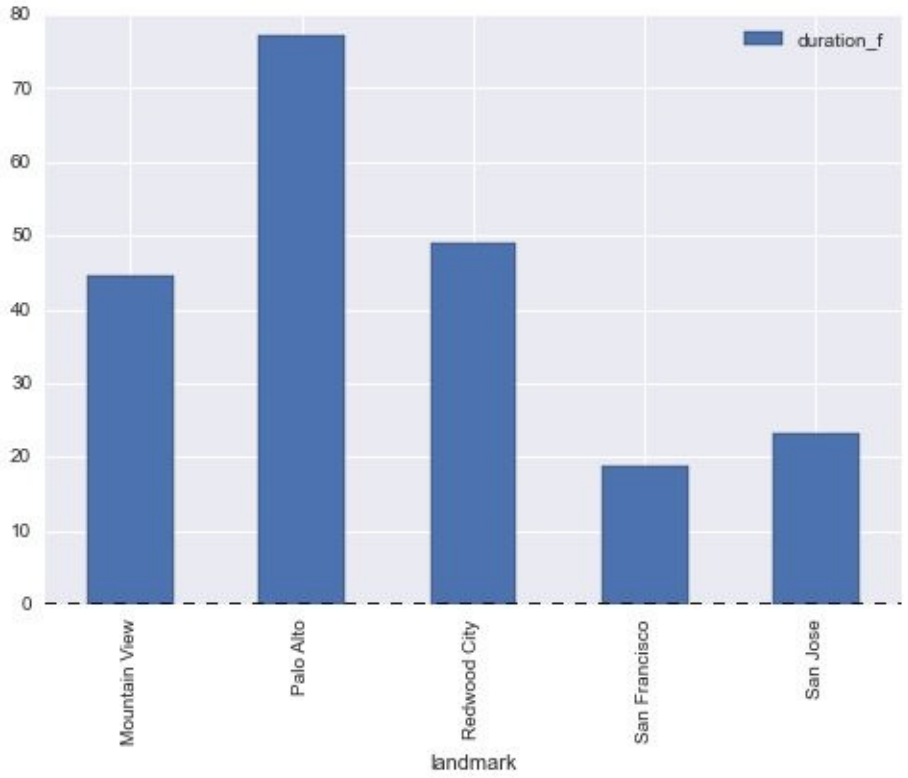
Average monthly duration faceted by time of day

```
grouperm = dmerge4.groupby([pd.Grouper(freq='M',key='startdate'),'timeofday'])\
.mean()[['duration_f']]
grouperm.unstack().plot(kind="bar")
```



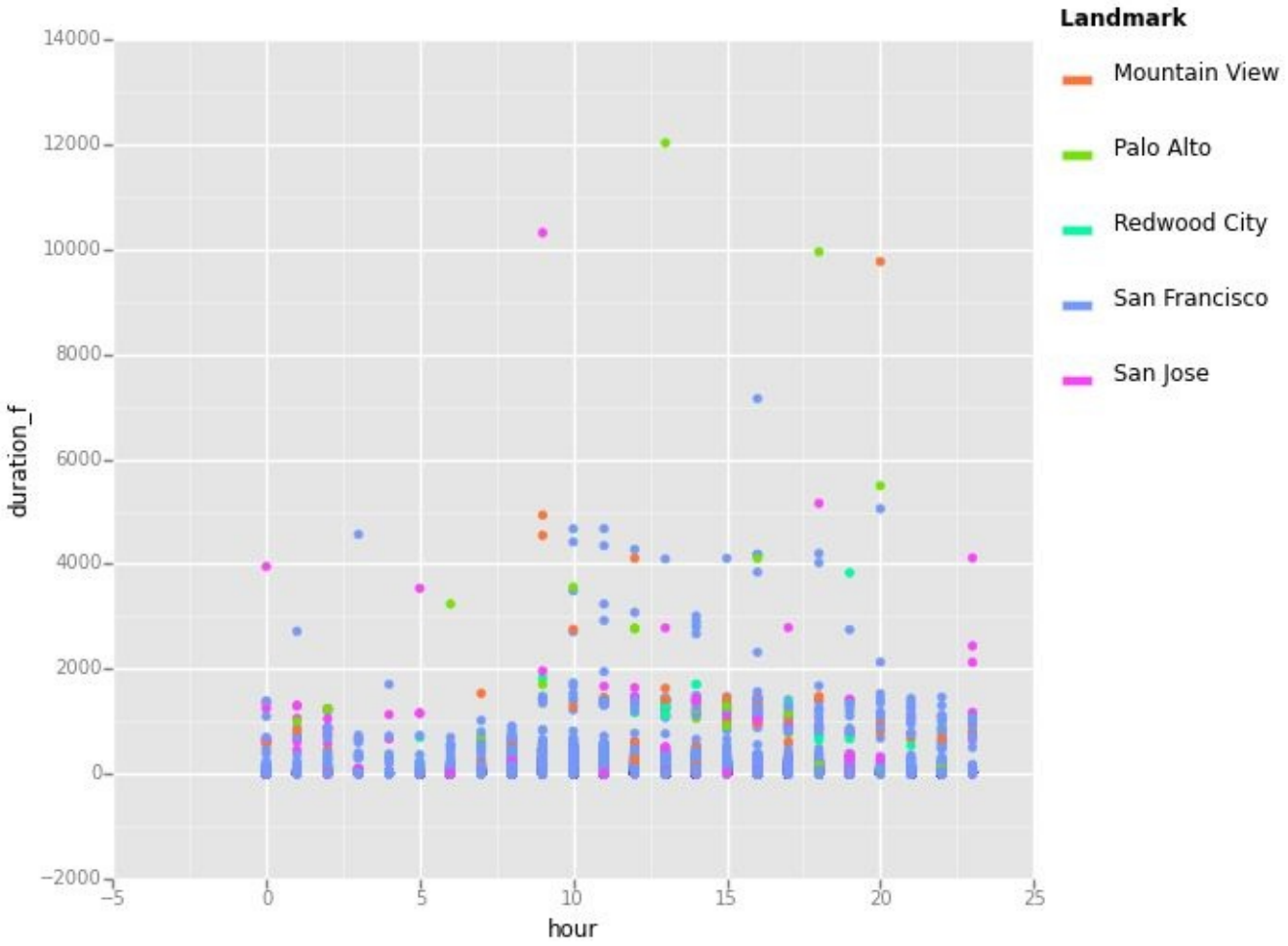
Average duration by landmark

```
dmerge4.groupby("landmark").mean()[['duration_f']].plot(kind='bar')
```



Average duration by hour of the day for the entire six month period

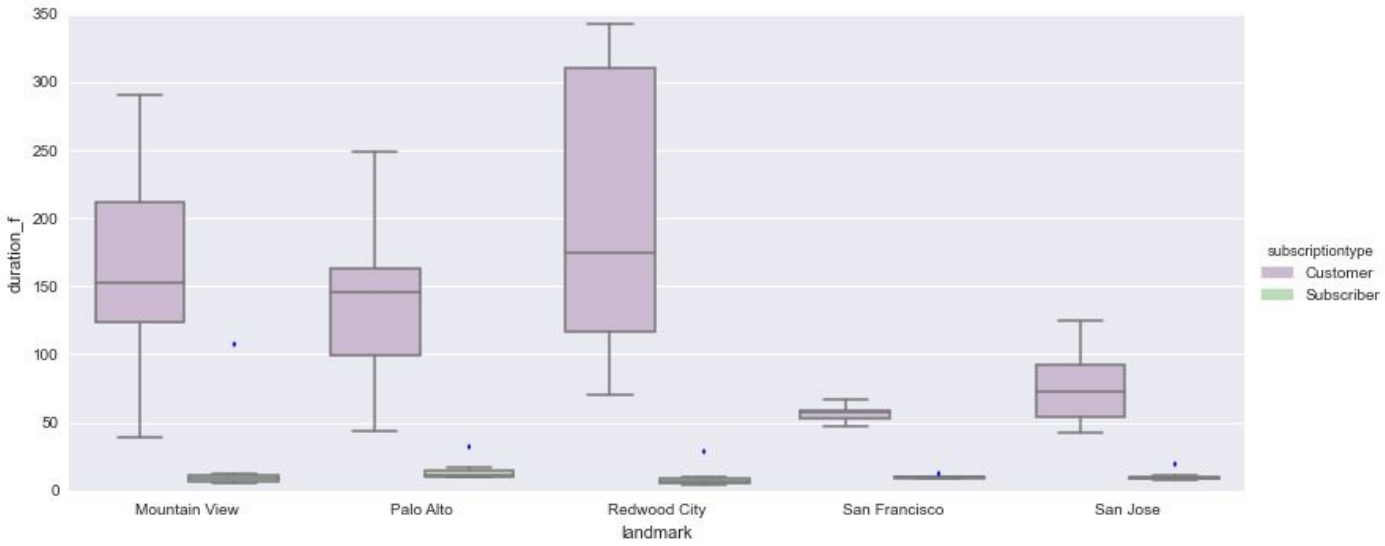
```
ggplot(dmerge4, aes('hour', 'duration_f',color='landmark')) + geom_point(stat="i\
dentity")
```



Boxplot of monthly average duration by landmark

Subscribers for the most part are below sixty minute durations while customers have higher durations and the widest range of duration in Redwood City followed by Mountain View.

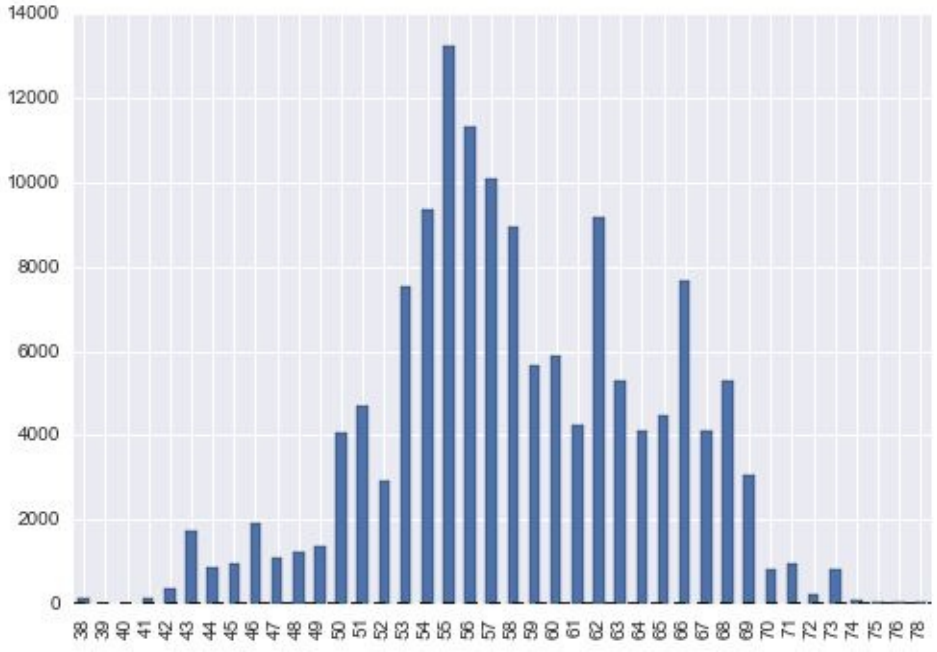
```
three_grouper = dmerge4.groupby([pd.Grouper(freq='M',key='startdate'),"landmark"\
,"subscriptiontype"])[['duration_f']].mean().reset_index()
import seaborn as sns
sns.factorplot("landmark", "duration_f", "subscriptiontype", three_grouper,\
kind="box",palette="PRGn", aspect=2.25)
```



Distribution of mean temperature

It doesn’t get too warm or too cold in Northern California between August and February.

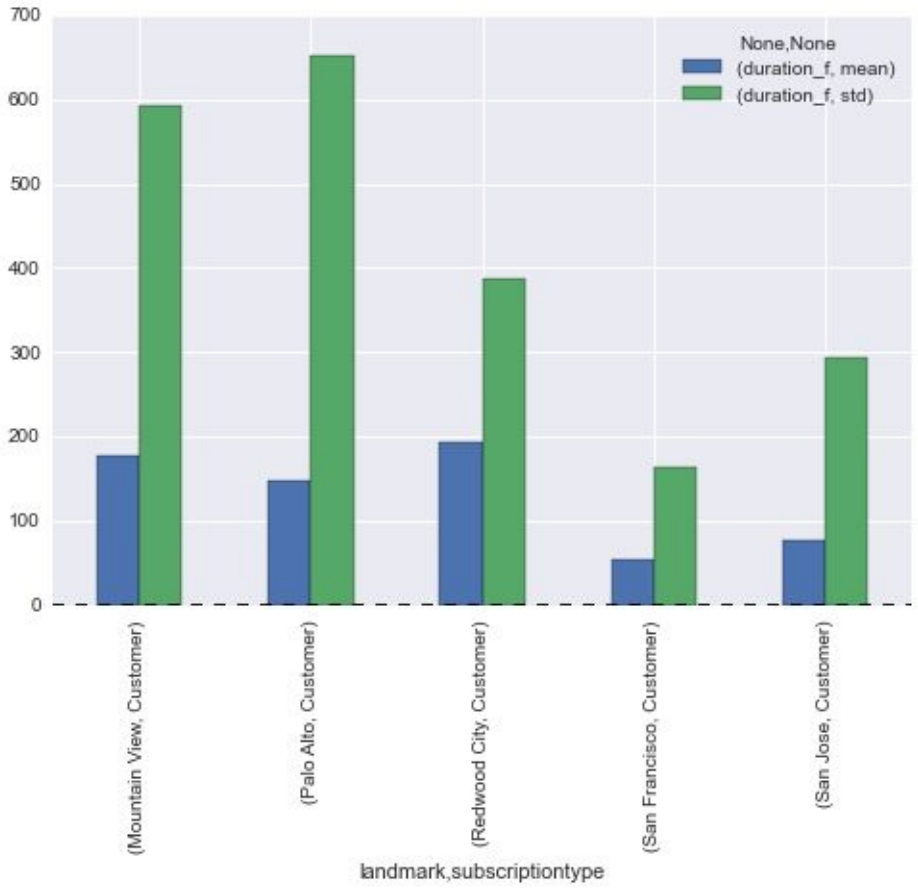
```
dmerge4['mean_temperature_f'].value_counts().sort_index().plot(kind='bar')
```



Average duration and standard deviation for subscription type of Customer by landmark

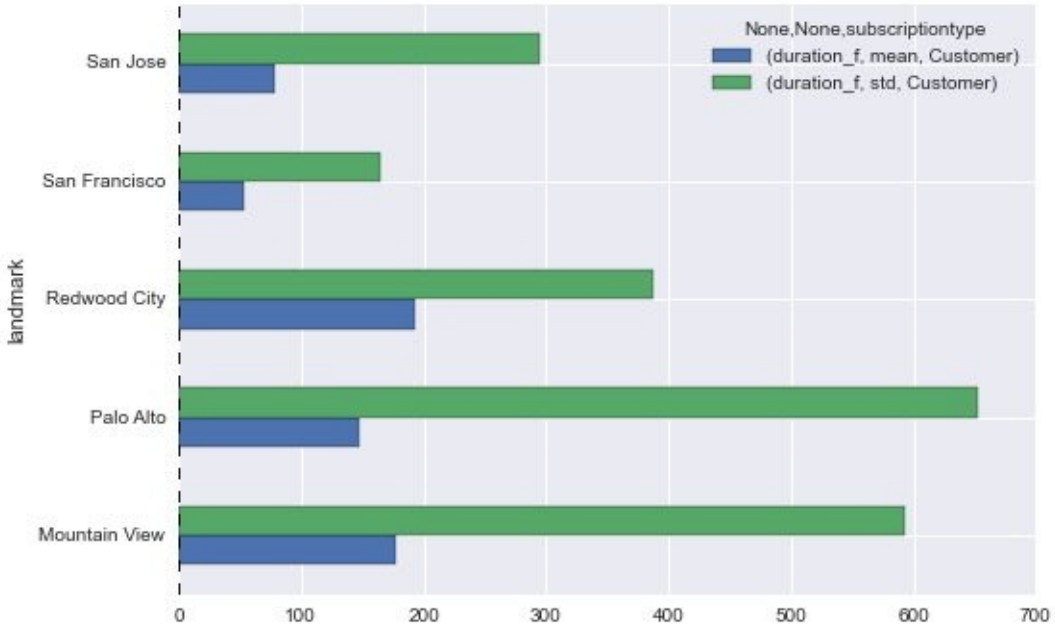
Bar plot of average duration by landmark and subscription type. In this case, we only wanted to compare duration by one type of customer.

```
dmerge4.groupby(['landmark','subscriptiontype'])[['duration_f']].agg([np.mean, n\
p.std]).query('subscriptiontype == "Customer"').plot(kind="bar")
```



Note, this is the exact same plot as above except that the unstacking does not affect the result because we specified the subscription type value.

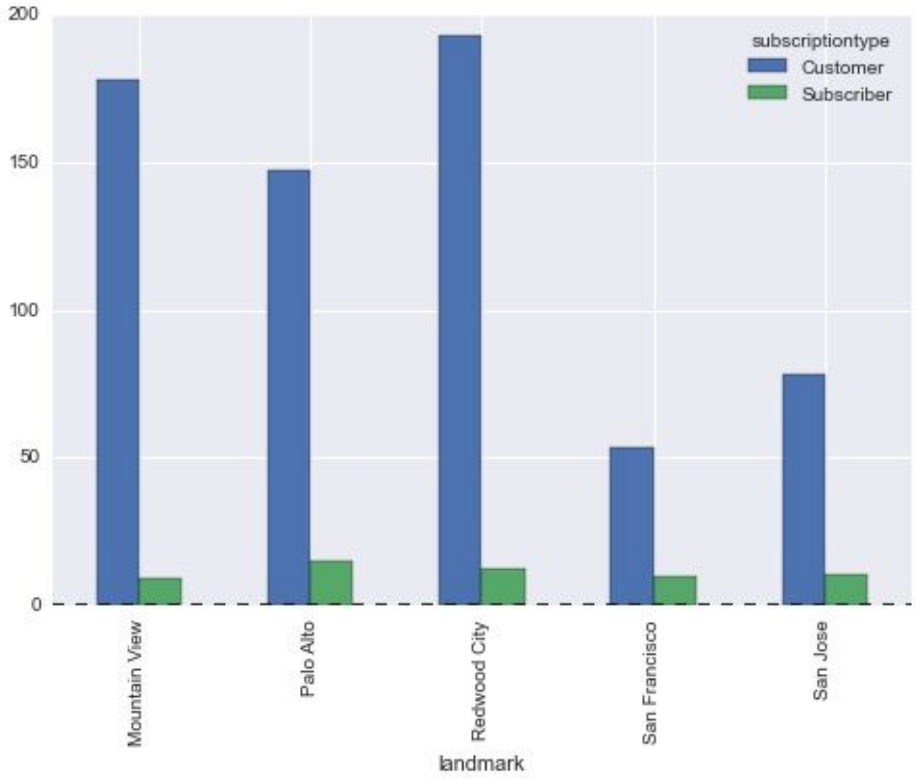
```
dmerge4.groupby(['landmark','subscriptiontype'])[['duration_f']].agg([np.mean, n\
p.std]).query('subscriptiontype == "Customer"').unstack().plot(kind="barh")
```


None,None,subscriptiontype
(duration_f, mean, Customer)
(duration_f, std, Customer)
landmark
San Jose
San Francisco
Redwood City
Palo Alto
Mountain View
0
100
200
300
400
500
600
700

Average duration by subscription type

Looking at the mean duration by subscription type shows that subscribers are on average within the thirty minutes while ‘customers’ often exceed that time period. Remember that seventy-nine percent of total observations in the dataset are subscribers. Subscribers have an annual membership while customers are defined as either twenty-four hour or three day pass.

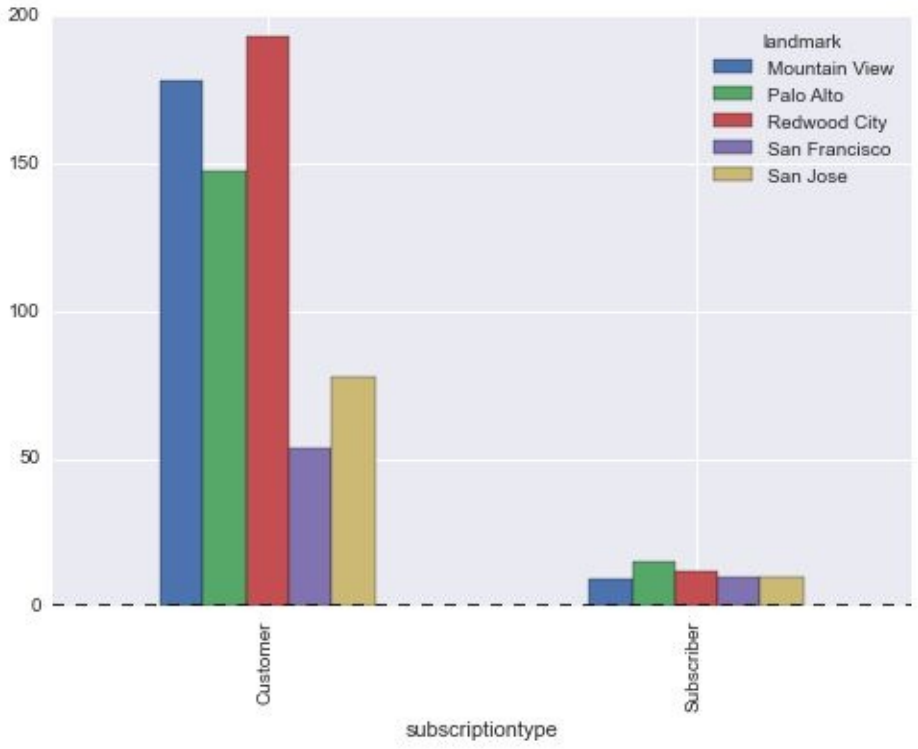
```
dmerge4.groupby(['landmark','subscriptiontype'])['duration_f'].mean().unstack(\
).plot(kind="bar")
```



Plot that indicates the differences in average duration between subscribers and customers by landmark

We love this plot. It gives a very clear idea of the differences in average duration between customers and subscribers with dodged plots by landmark.

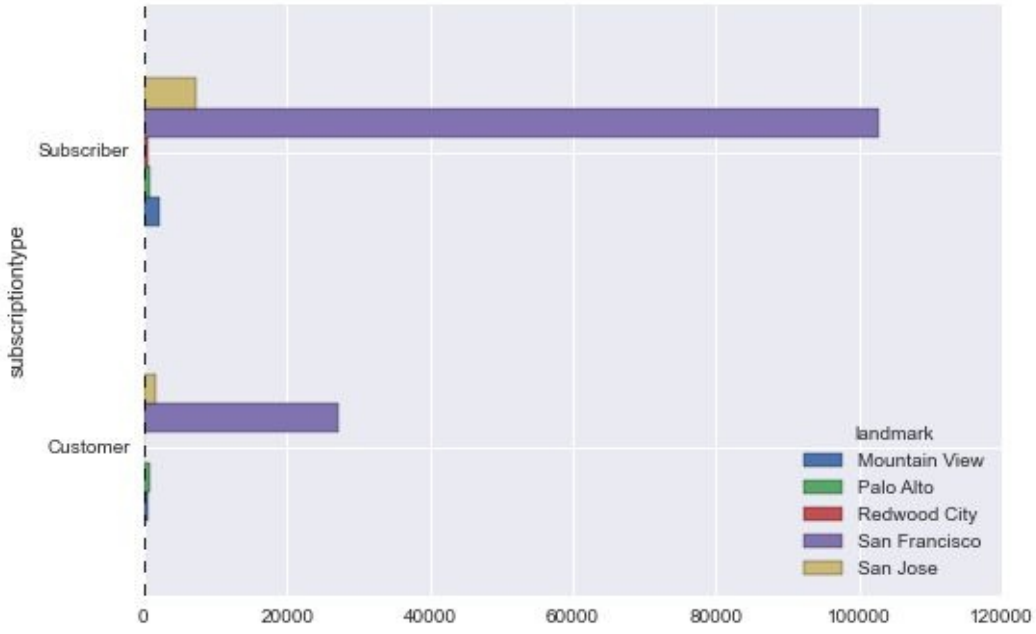
```
dmerge4.groupby(['subscriptiontype','landmark'])['duration_f'].mean().unstack().\
plot(kind="bar")
```



Number of observations in the dataset by landmark and subscription type

The plot above reveals really interesting insights when compared to this plot of the number of subscribers versus the number of customers in the bikesharing system in the first six months of operation. This plot indicates that the majority of bike trips were taken by subscribers in San Francisco. The previous plot indicated that they had the shortest duration time. Looking at these plots together tells a story.

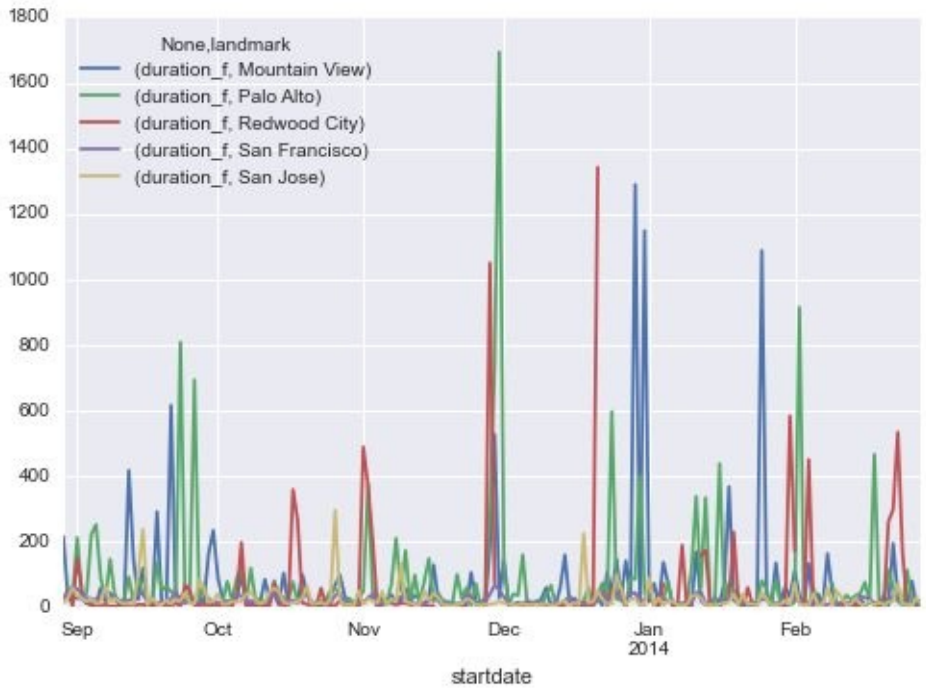
```
dmerge4.groupby(['subscriptiontype', 'landmark']).size().unstack().plot(kind='ba\
rh')
```



Daily average duration by landmark for the six month period

The highest daily average duration occurred in December in Palo Alto.

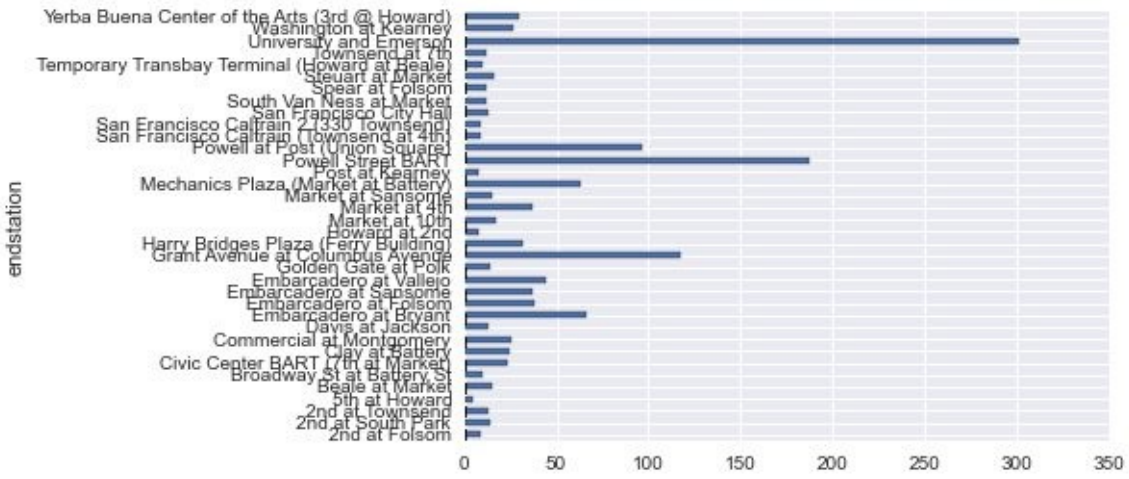
```
dmerge4.groupby([pd.Grouper(freq='D',key='startdate'),'landmark'])[['duration_f'\
]].mean().unstack().plot()
```



What is the most common end station when a bicyclist starts at the Powell Street BART?

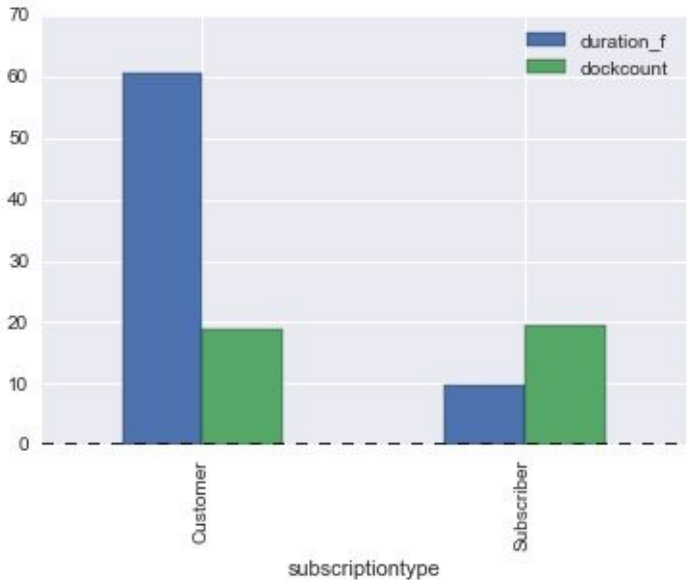
The most common end station when a bicyclist starts at the Powell Street BART is the University and Emerson.

```
dmerge4twolevels.query('startterminal == 53').groupby('endstation')['duration_f'\
].mean().plot(kind='barh')
```



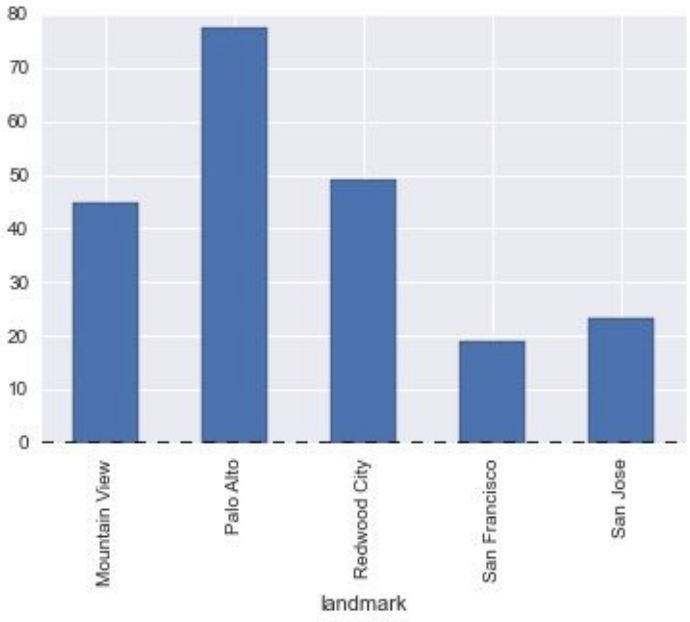
Plot duration and dockcount by subscription type

```
grouplevel1 = dmerge4twolevels.groupby(level=1)
grouplevel1.mean()[['duration_f','dockcount']].plot(kind='bar')
```



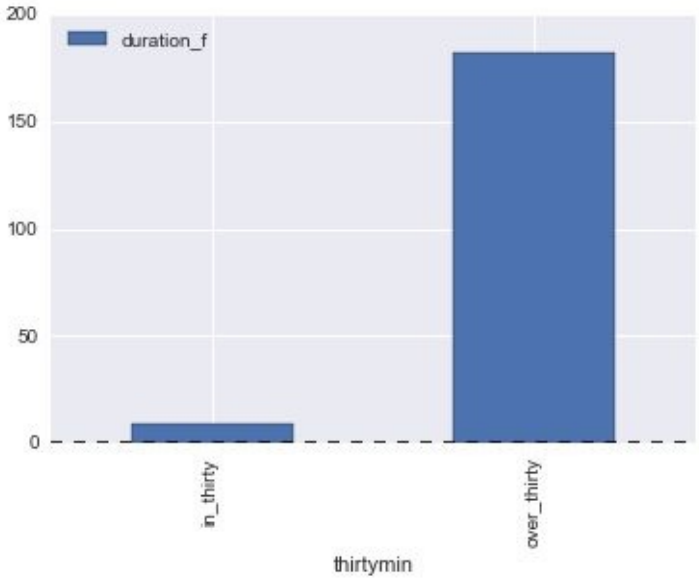
When working with a mulit-level index, this is one way of accessing a particular level.

```
 dmerge4twolevels.groupby(level=['landmark']).mean()['duration_f']\
.plot(kind="bar")
```



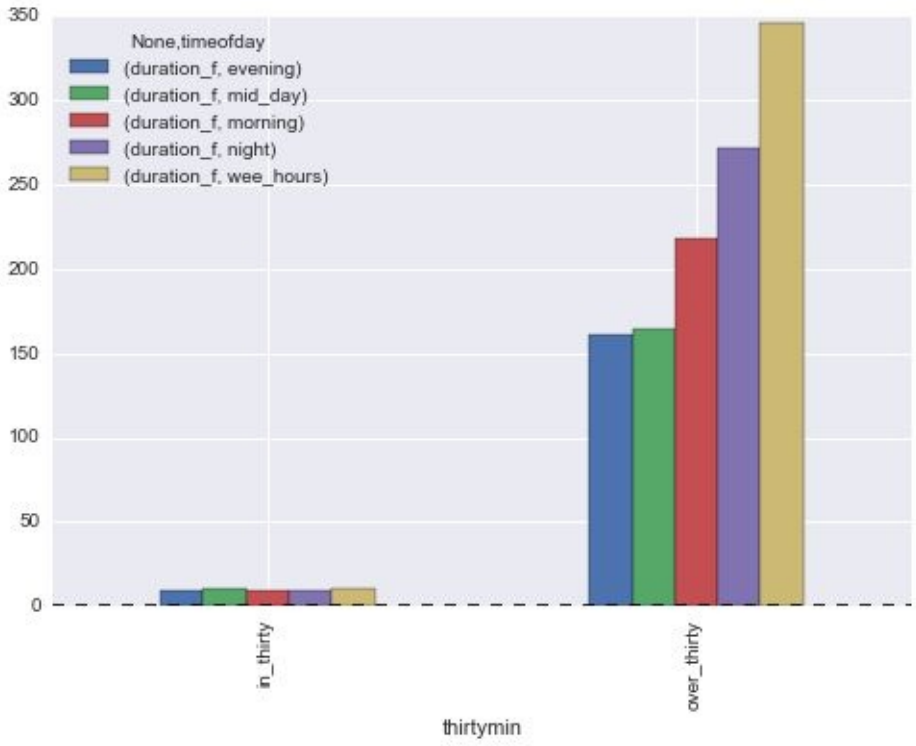
Plot the average duration for trips that were thirty minutes and under versus trips over thirty minutes

```
dmerge4.groupby('thirtymin')[['duration_f']].mean().plot(kind='bar')
```



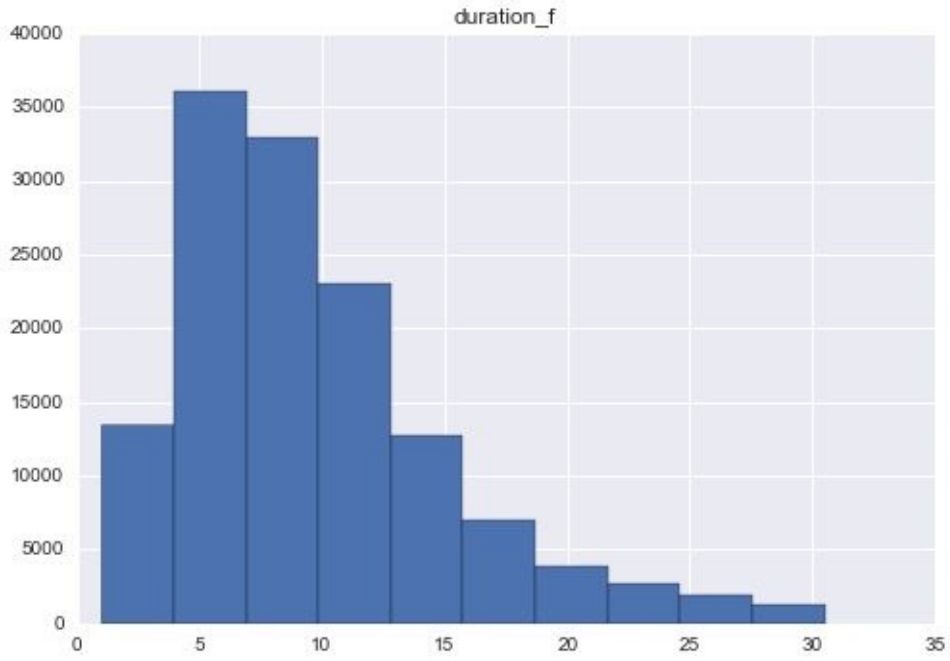
Average duration for trips thirty minutes and under compared to trips over thirty minutes faceted by time of day

```
dmerge4.groupby(['thirtymin','timeofday'])[['duration_f']].mean().unstack().plot\
(kind="bar")
```



Histogram of duration of trips thirty minutes and less

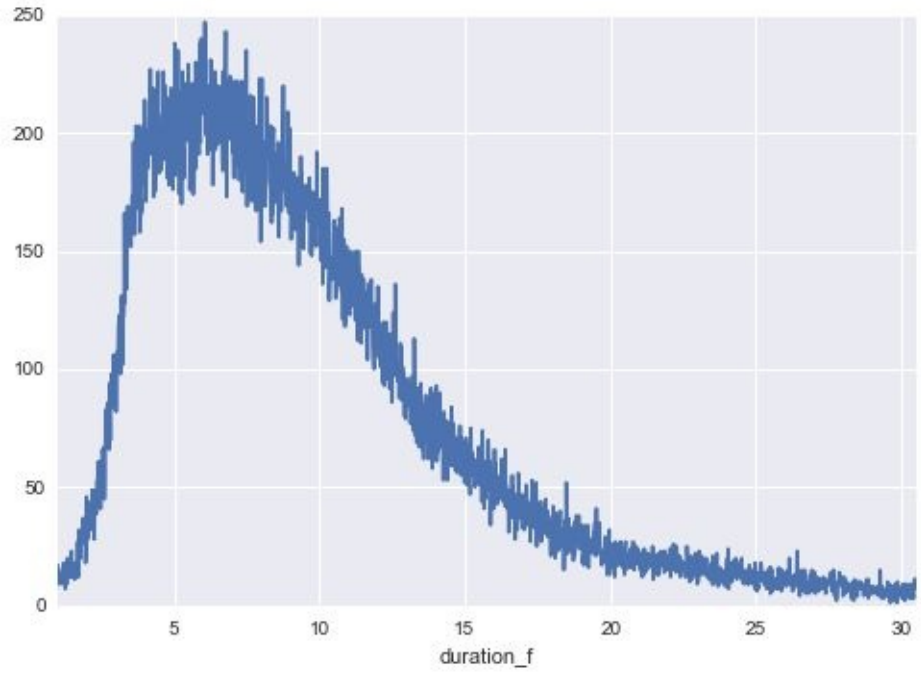
```
dmerge4.query('thirtymin == "in_thirty"')[['duration_f']].hist()
```



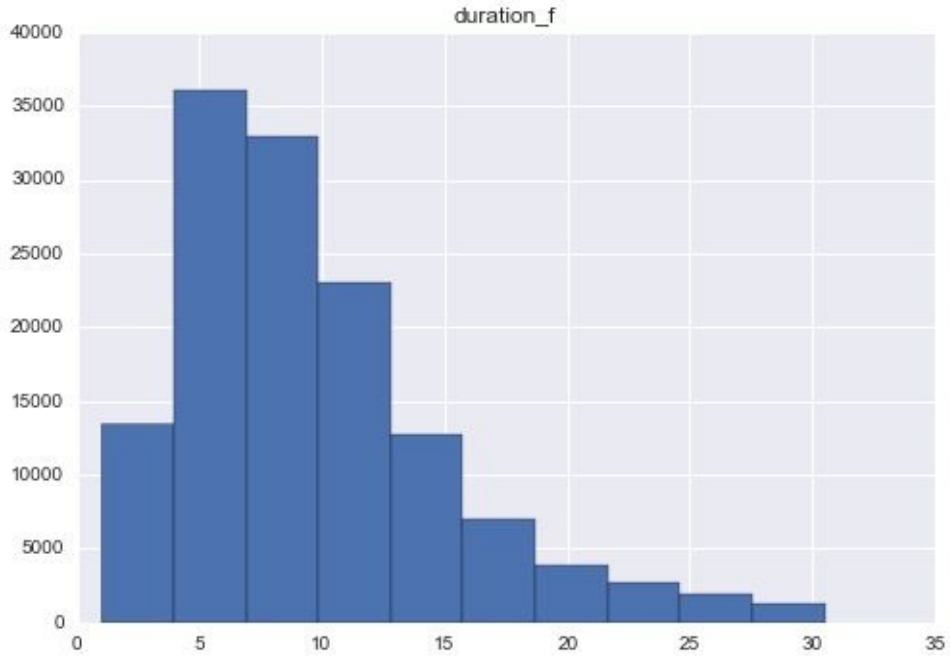
Number of observations for each duration of thirty minutes and under

It’s very interesting that the majority of rides thirty minutes and less are between five and 10 minutes! So, perhaps the system is mainly used for short commutes as intended rather than for recreational purpose. We have seen this to be the case for rides originating in San Francisco where the majority of trips began and docks are located.

```
dmerge4.query('thirtymin == "in_thirty"').groupby('duration_f').size().plot()
```

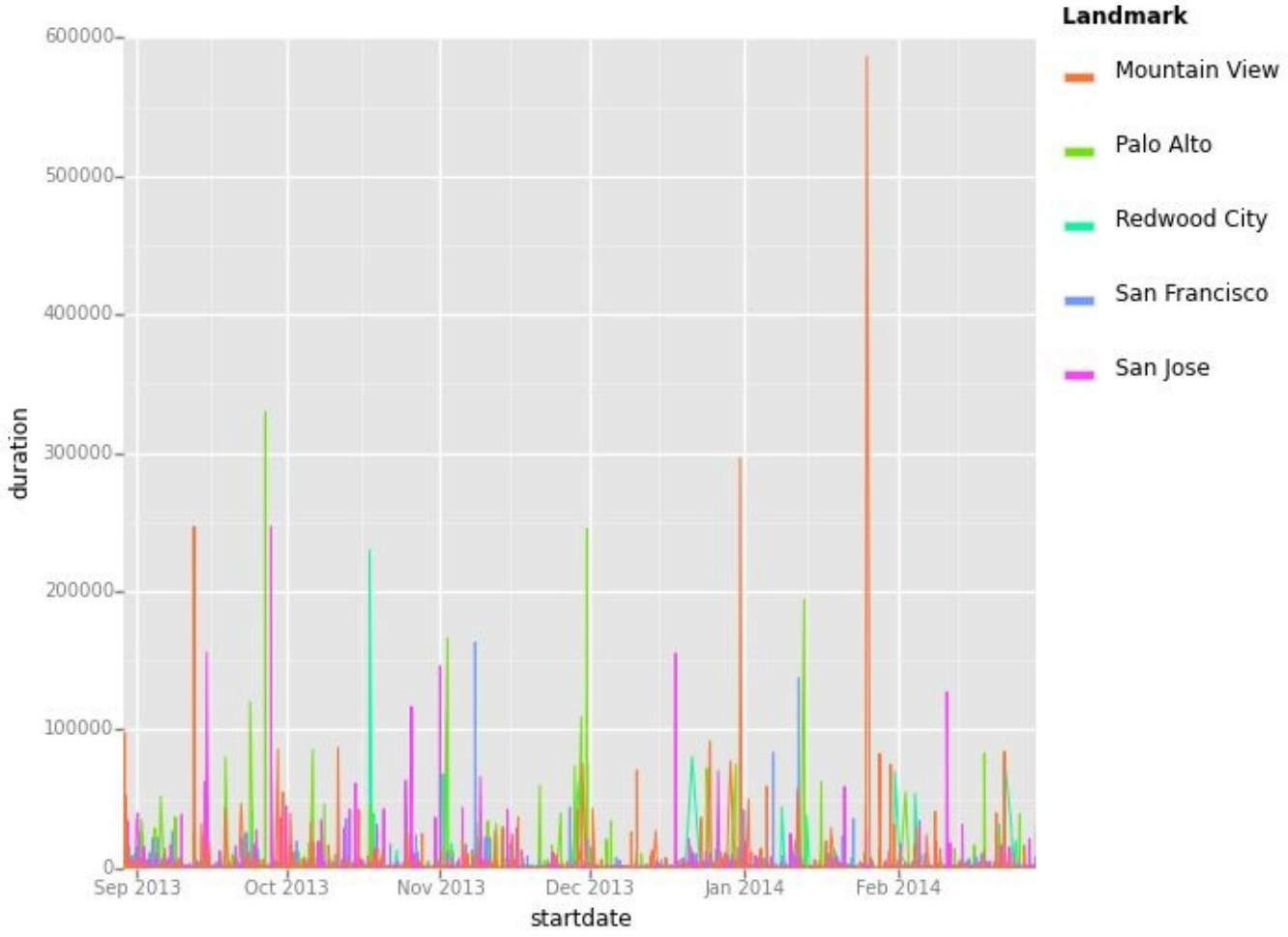



```
dmerge4.query('thirtymin == "in_thirty"')[['duration_f']].hist()
```



Hourly average duration for the six month period

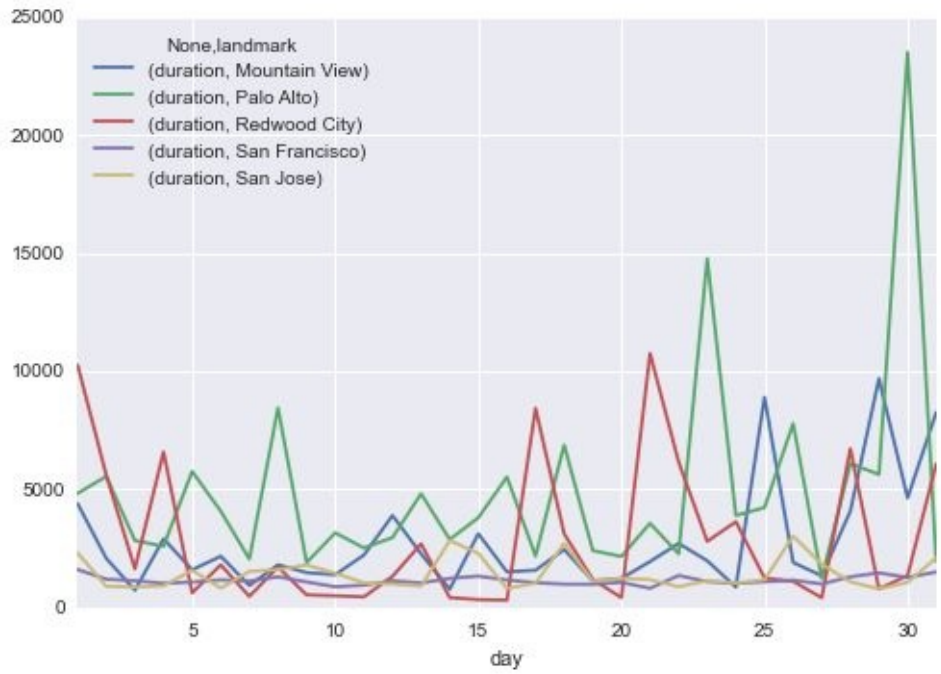
```
hourly2 =  dmerge4.groupby([pd.Grouper(freq='H',key='startdate'),'landmark'],\
as_index=False).mean()
ggplot(hourly2, aes('startdate', 'duration',color='landmark')) + geom_line()
```



What days of the week are bicycles being used for the longest average duration by landmark?

It’s nice to see the differences by landmark. For instance, it appears that durations are higher in Palo Alto near the end of the month.

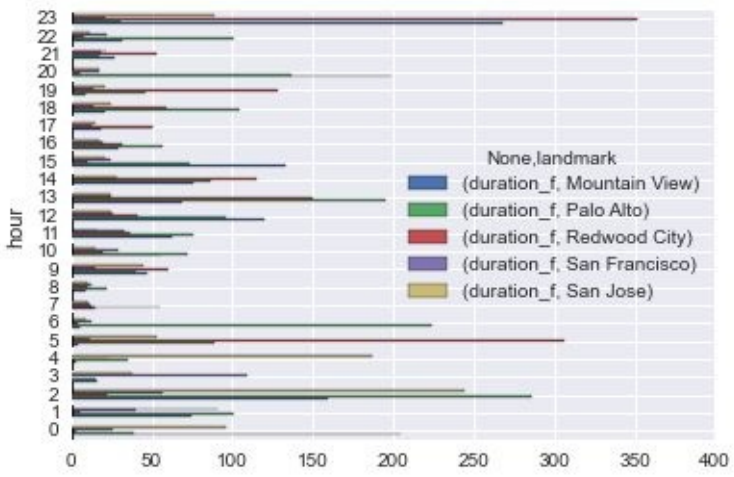
```
#mean duration by day of the week
dmerge4.groupby(['day','landmark']).aggregate(np.mean)[['duration']].unstack()\
.plot()
```



What hour of the day has the longest duration by landmark?

Redwood City in the eleven pm hour has the longest average duration.

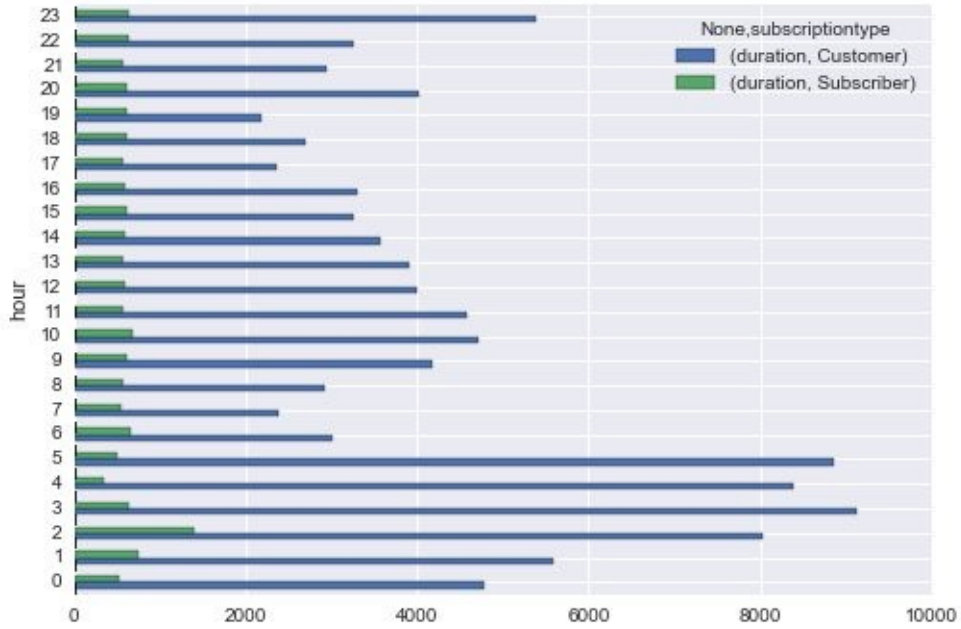
```
#mean duration by hour of the day
dmerge4.groupby(['hour','landmark']).aggregate(np.mean)[['duration_f']].\
unstack().plot(kind='barh')
```



Average duration by hour of the day faceted by subscriber type

For every hour of the day, Customers have longer average duration than Subscribers especially in the wee hours.

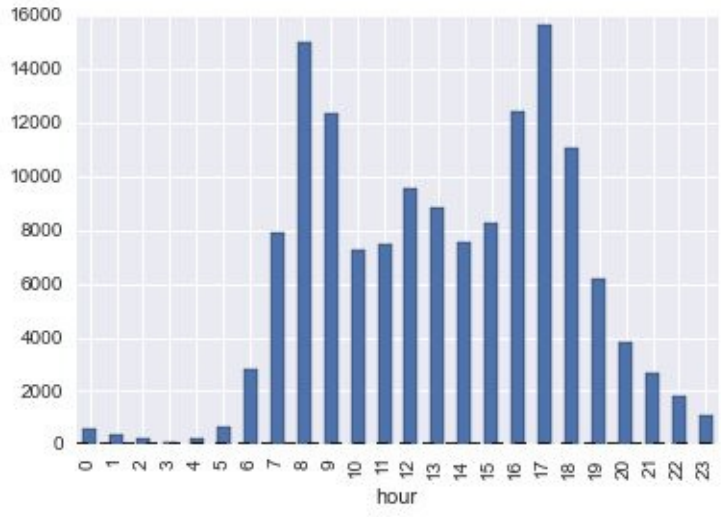
```
#mean duration by hour of the day faceted by subscriber type
dmerge4.groupby(['hour','subscriptiontype']).aggregate(np.mean)[['duration']].un\
stack().plot(kind='barh')
```



Distribution of number of observations in the dataset by hour of the day

There were more bicycle trips taken during the morning commute and the evening commute.

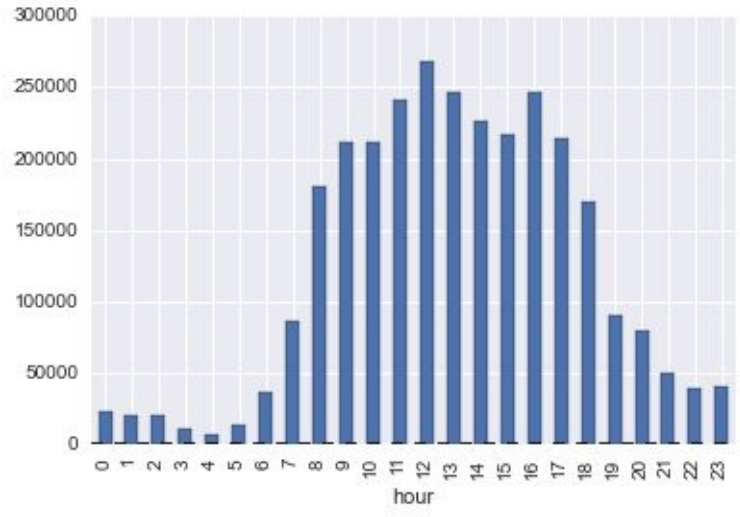
```
hour_counts = dmerge4.groupby('hour').aggregate(np.size)
hour_counts['duration_i'].plot(kind='bar')
```



Distribution of total duration by hour of the day

Total duration was highest from eight am to six pm which also corresponds to working hours.

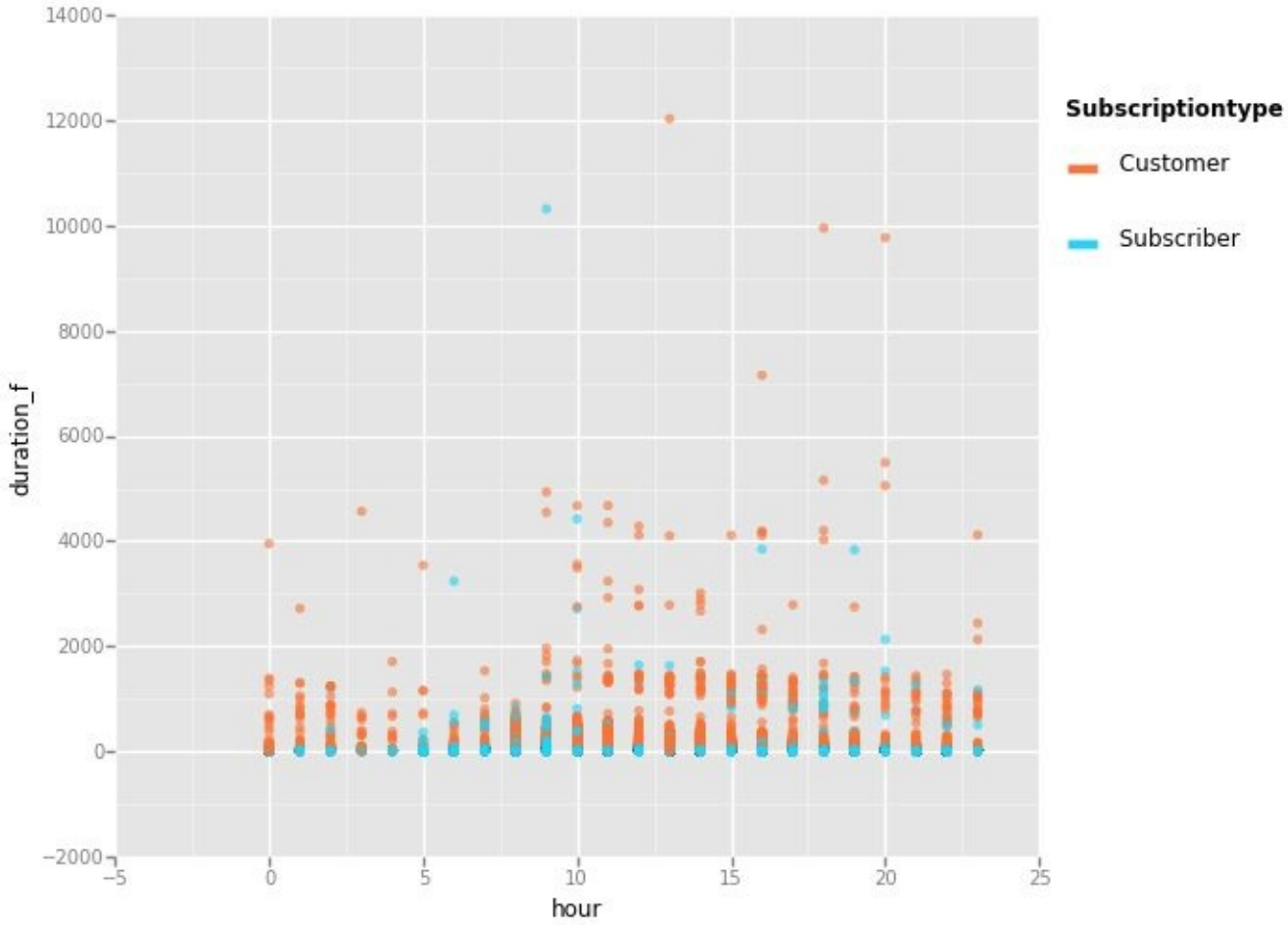
```
hour_sum = dmerge4.groupby('hour').aggregate(np.sum)
hour_sum['duration_i'].plot(kind='bar')
```



Average duration by hour of the day and subscription type.

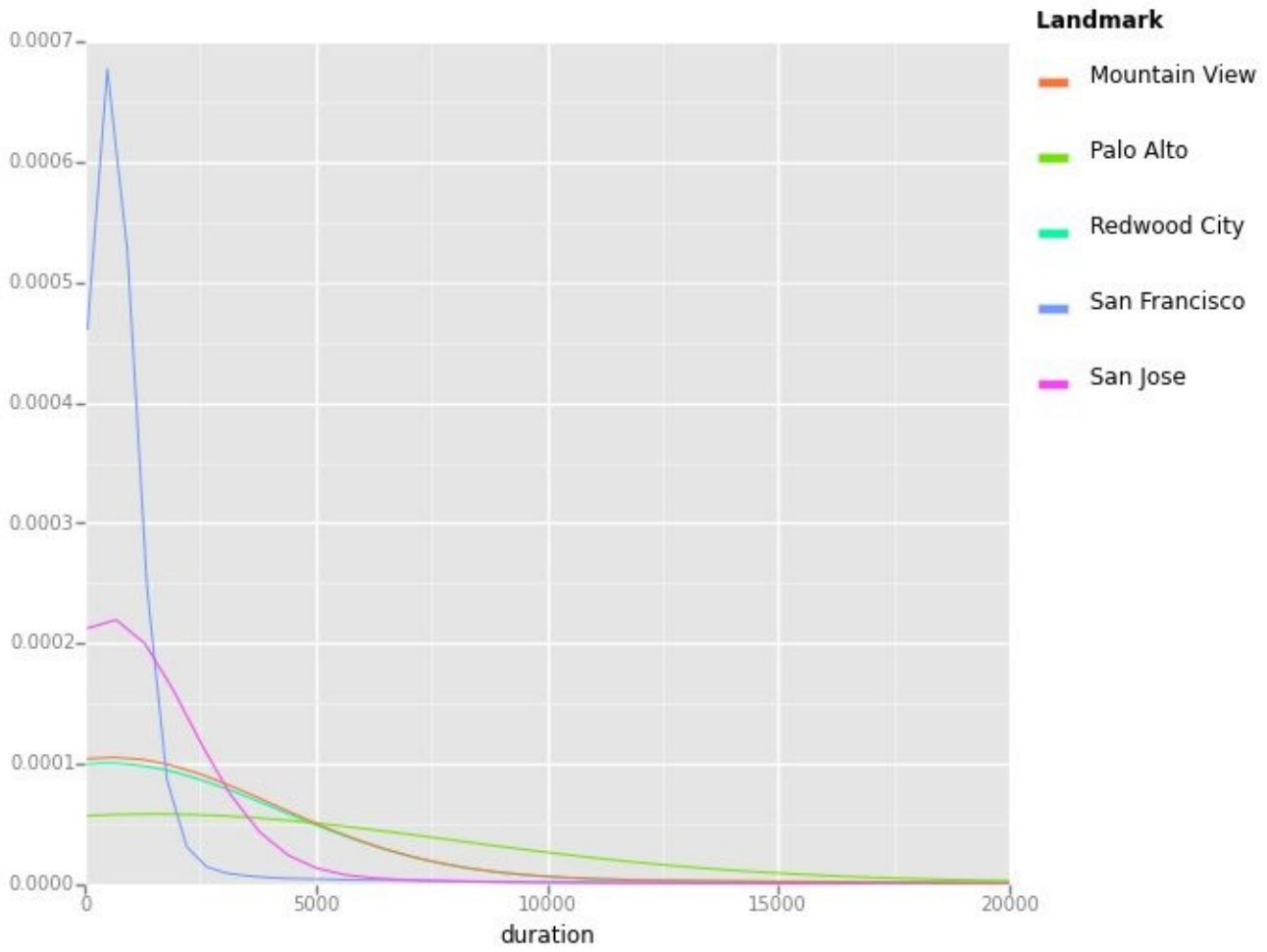
This plot indicates that subscriber trips were generally shorter while customer trips had higher durations

```
from ggplot import *
ggplot(aes(x='hour', y='duration_f',color='subscriptiontype'), data=dmerge4) + \
geom_point(alpha=.6)
```



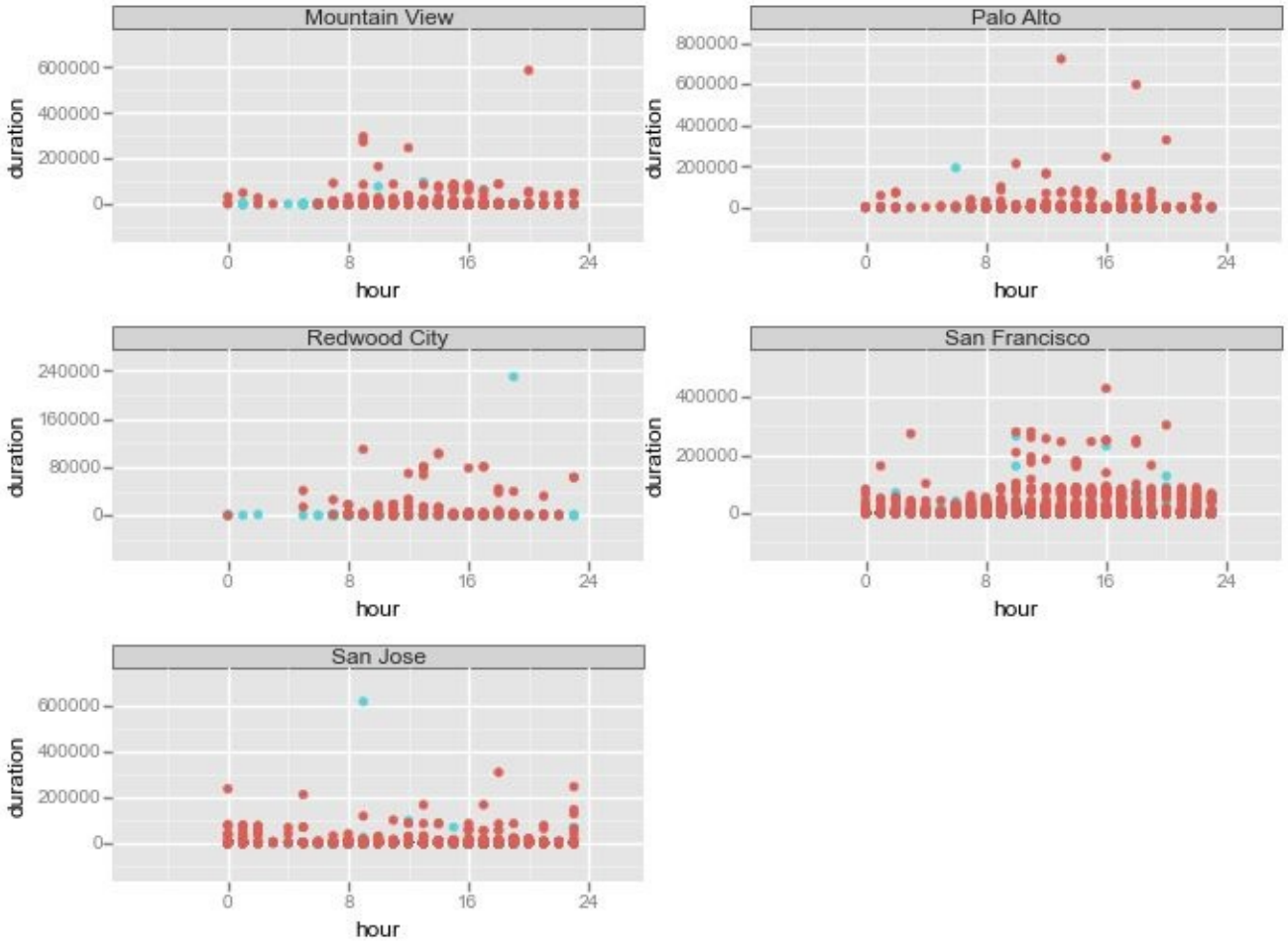
Density plot of average duration by landmark

```
ggplot(dmerge4, aes(x='duration', color='landmark')) + \
geom_density() + xlim(0,20000)
```



Duration by hour for both subscription types faceted by landmark

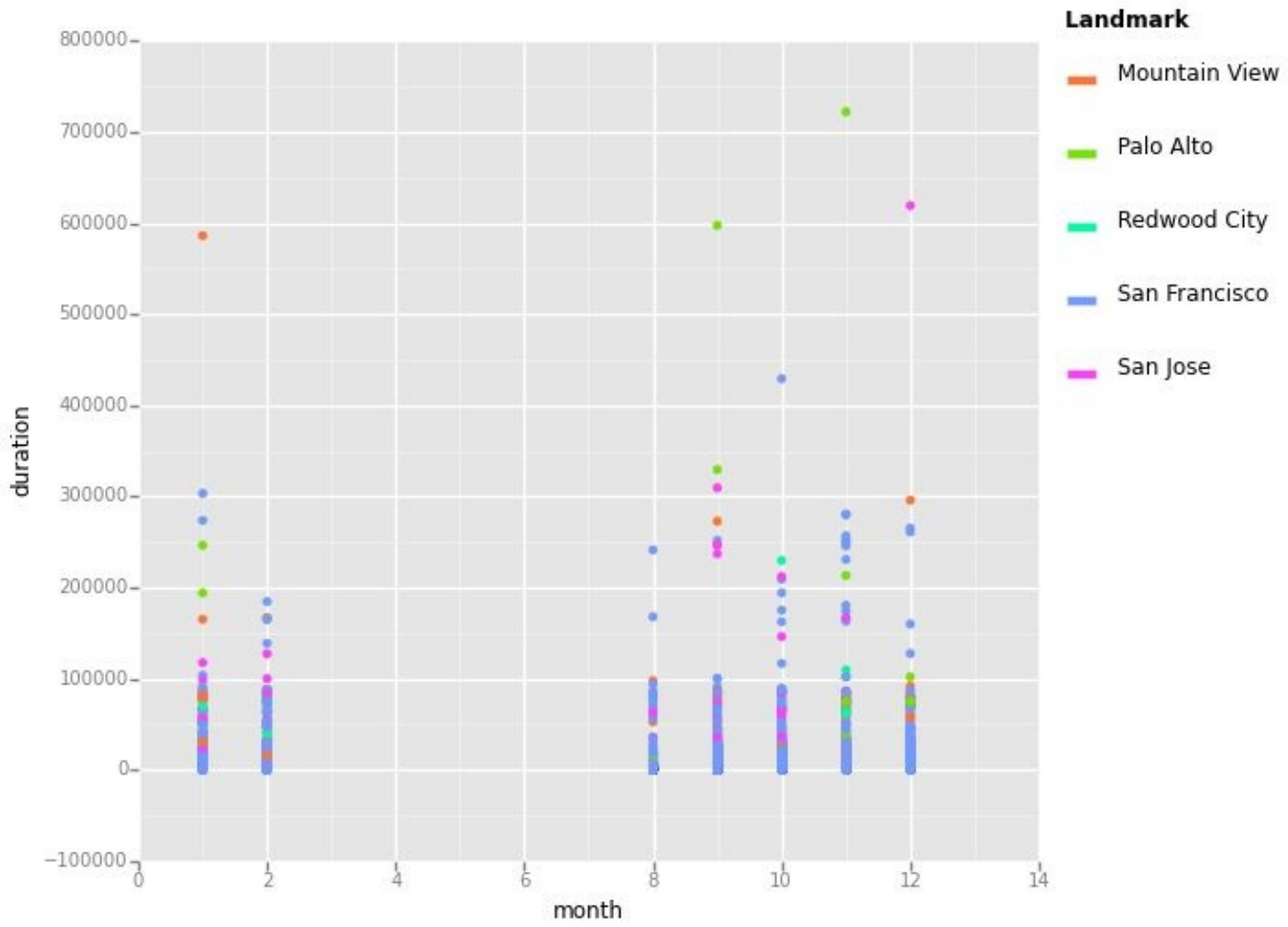
```
ggplot(aes(x='hour', y='duration', colour='subscriptiontype'), data=dmerge4) + \
geom_point() + facet_wrap("landmark")
```



Monthly total duration by landmark.

This plots illustrates that trips in San Francisco for the most part had shorter total durations while there were some outliers in Palo Alto.

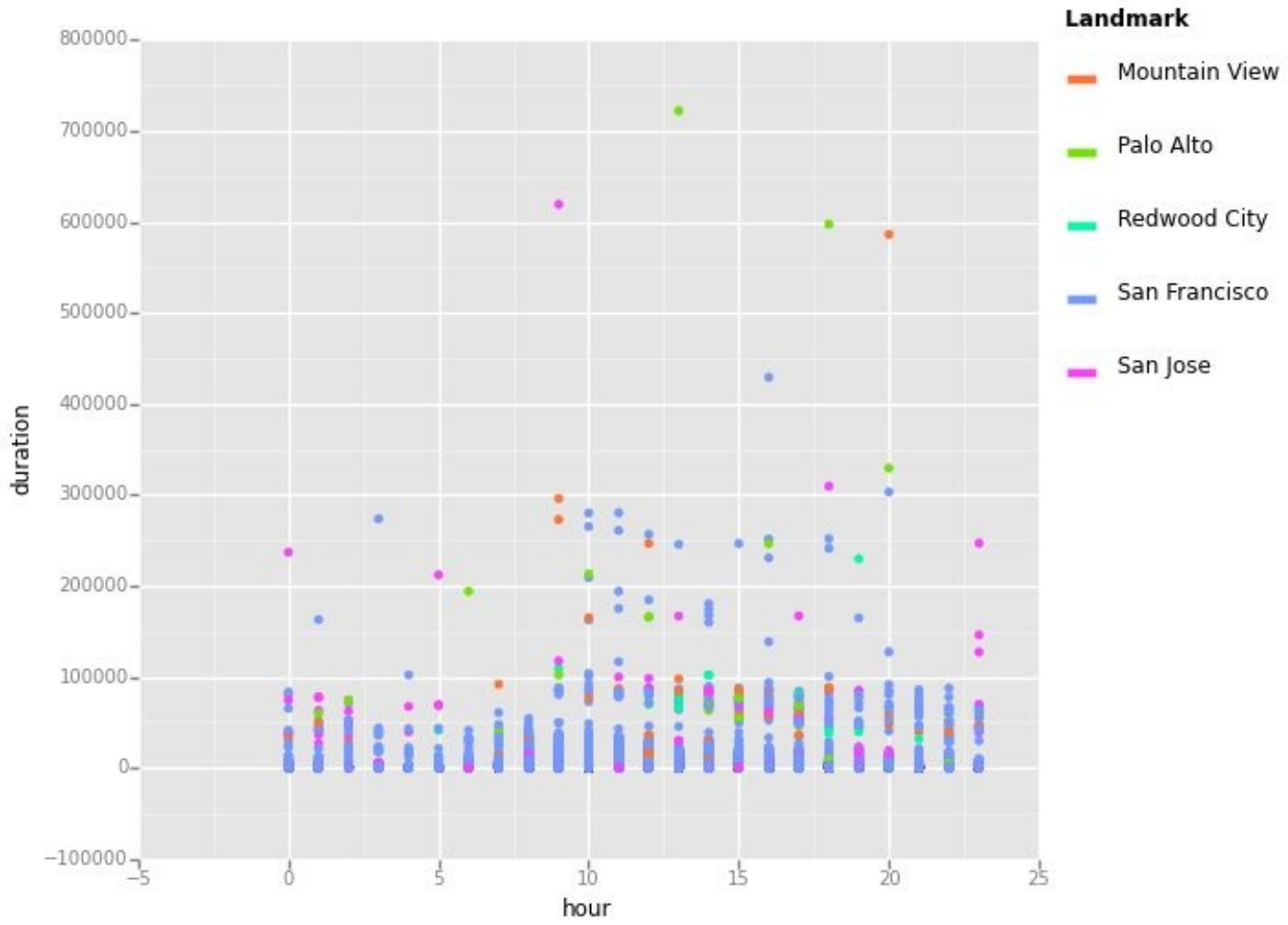
```
ggplot(aes(x='month', y='duration', color='landmark'), data=dmerge4) + \
geom_point()
```



Total duration by hour of day faceted by landmark

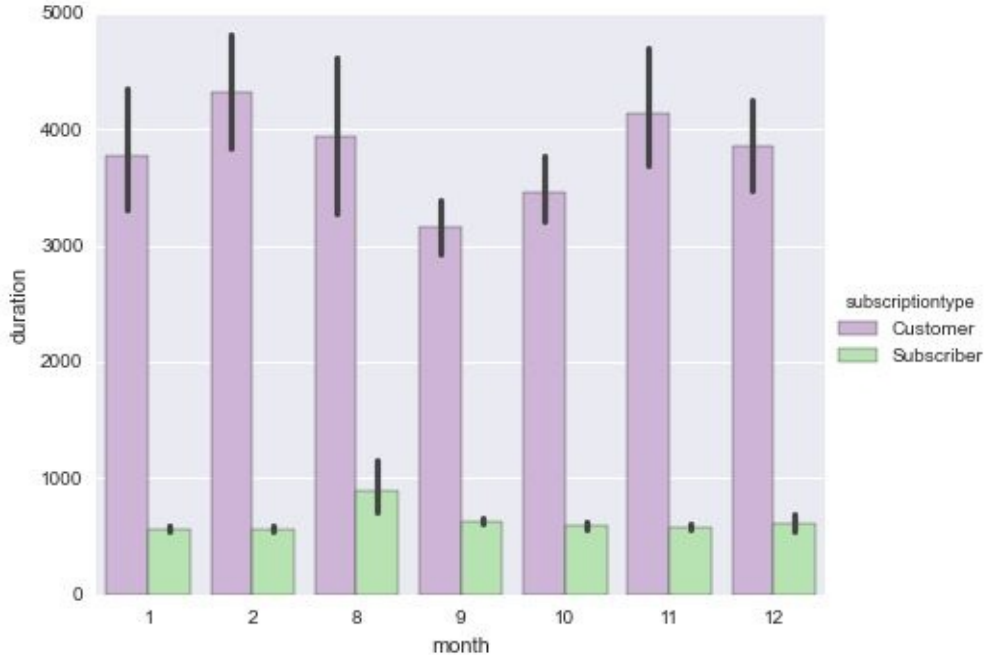
This type of plot makes it easier to pinpoint outliers by time of day.

```
ggplot(aes(x='hour', y='duration', color='landmark'), data=dmerge4) + \
geom_point()
```



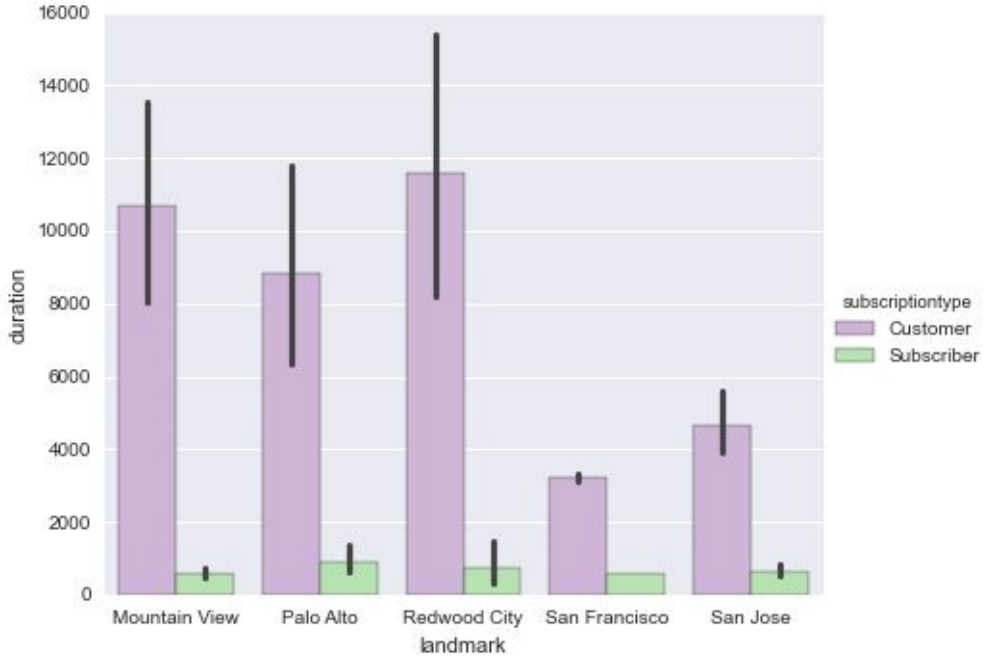
Plot of duration by month faceted by subscription type

```
sns.factorplot("month", "duration",data=dmerge4,hue="subscriptiontype",
palette="PRGn", aspect=1.25)
```



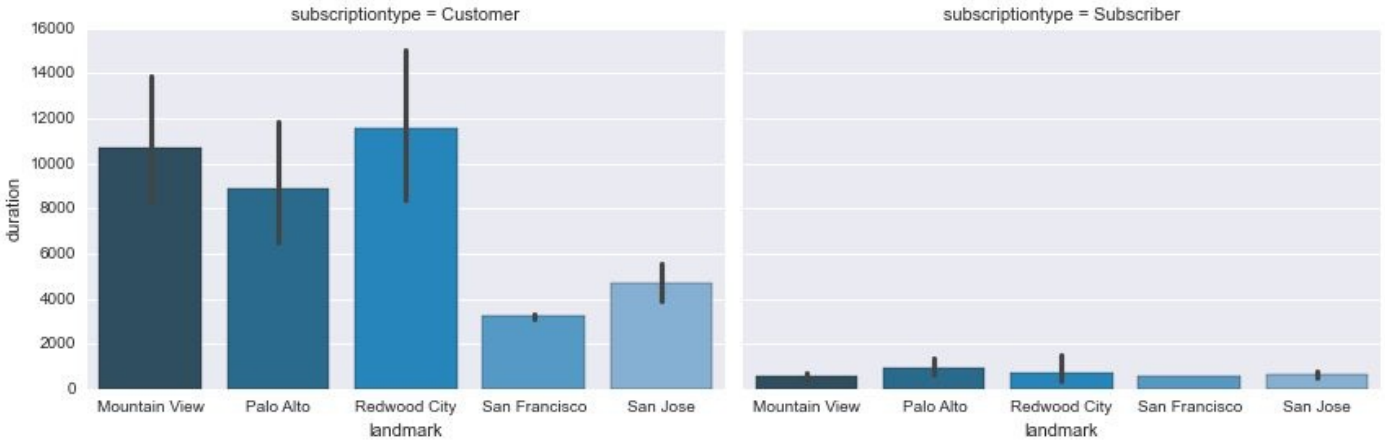
Plot of duration by landmark faceted by subscription type

```
sns.factorplot("landmark", "duration",data=dmerge4,hue="subscriptiontype",
palette="PRGn", aspect=1.25)
```



Plot of total duration by landmark faceted by subscription type

```
sns.factorplot("landmark", "duration", col="subscriptiontype",data=dmerge4, pale\
tte="PuBu_d",size=4,aspect=1.5);
```



Plot of average duration by landmark faceted by subscription type

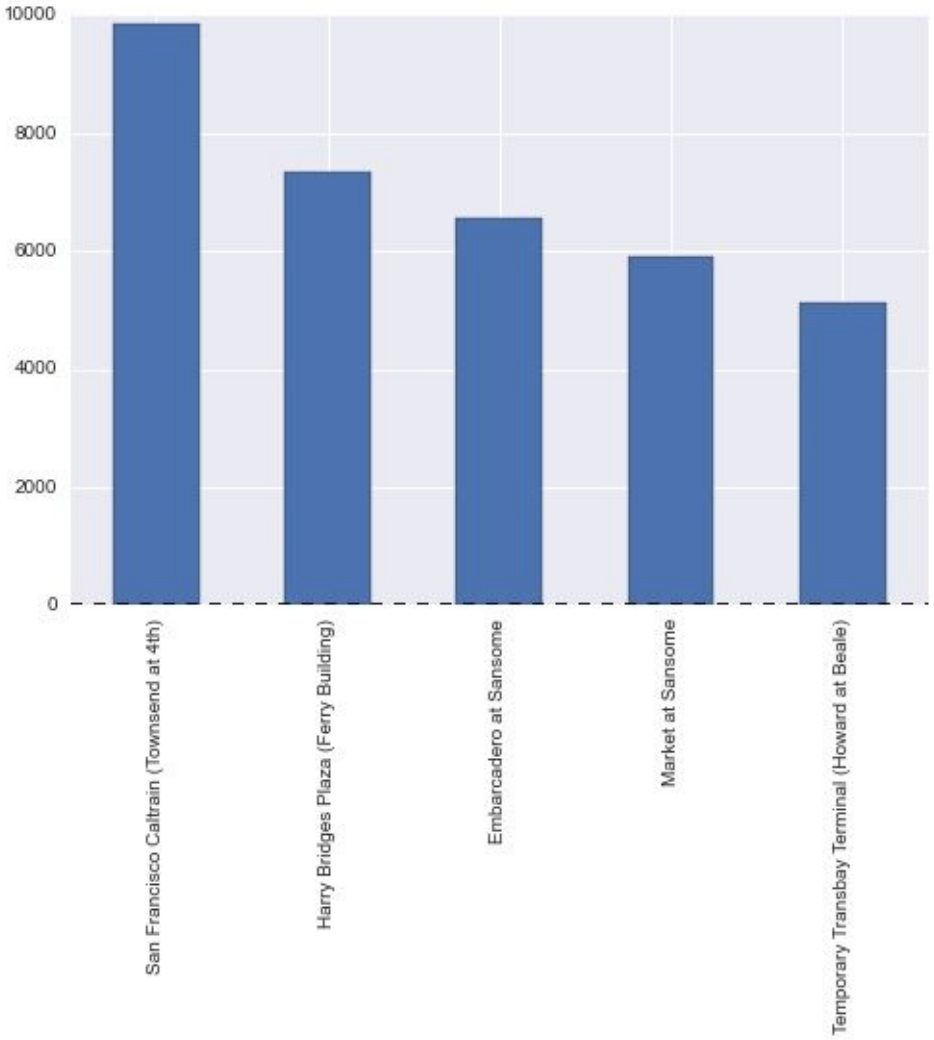
```
sns.factorplot("subscriptiontype", "duration_f", col="landmark", data=dmerge4, p\
alette="PuBu_d",size=4,aspect=.5);
```

The top five start stations

```
dmerge4["startstation"].value_counts()[:10]
```

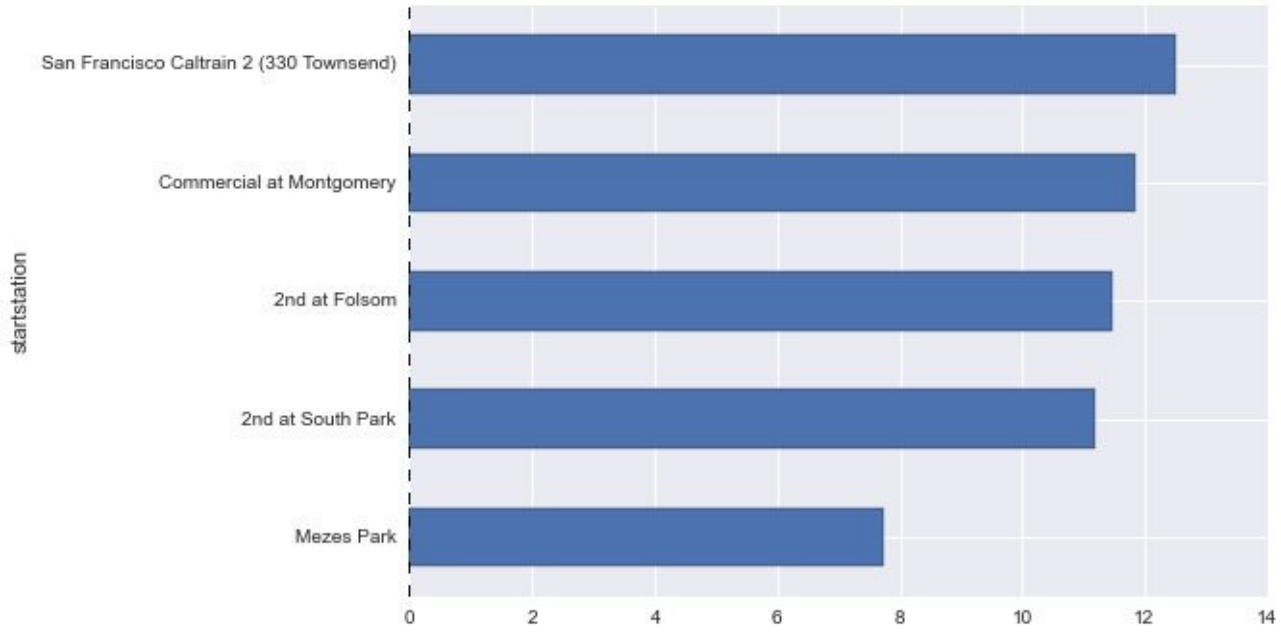
```
San Francisco Caltrain (Townsend at 4th)          9838
Harry Bridges Plaza (Ferry Building)              7343
Embarcadero at Sansome                            6545
Market at Sansome                                 5922
Temporary Transbay Terminal (Howard at Beale)     5113
Market at 4th                                     5030
2nd at Townsend                                   4987
San Francisco Caltrain 2 (330 Townsend)           4976
Steuart at Market                                 4913
Townsend at 7th                                   4493
dtype: int64
```

```
dmerge4["startstation"].value_counts()[:5].plot(kind="bar")
```



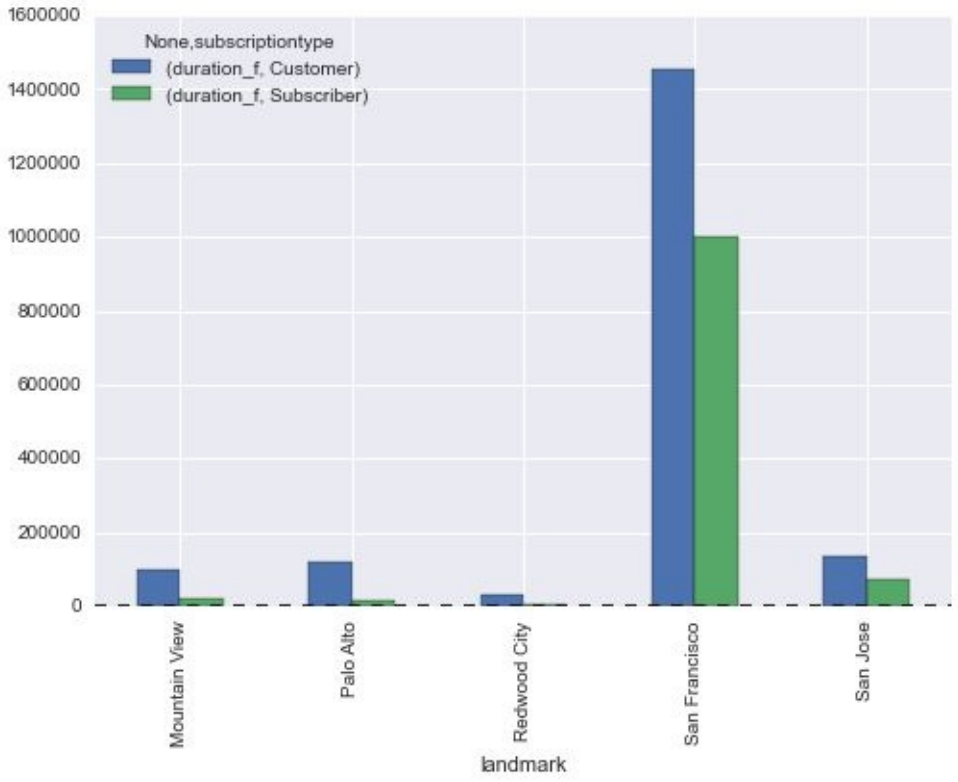
Plot of the five lowest average durations by start station

```
dmerge4.groupby('startstation').duration_f.mean().order(ascending=True)[:5].plot\
(kind='barh')
```



Plot of the total duration by landmark faceted by subscriber type

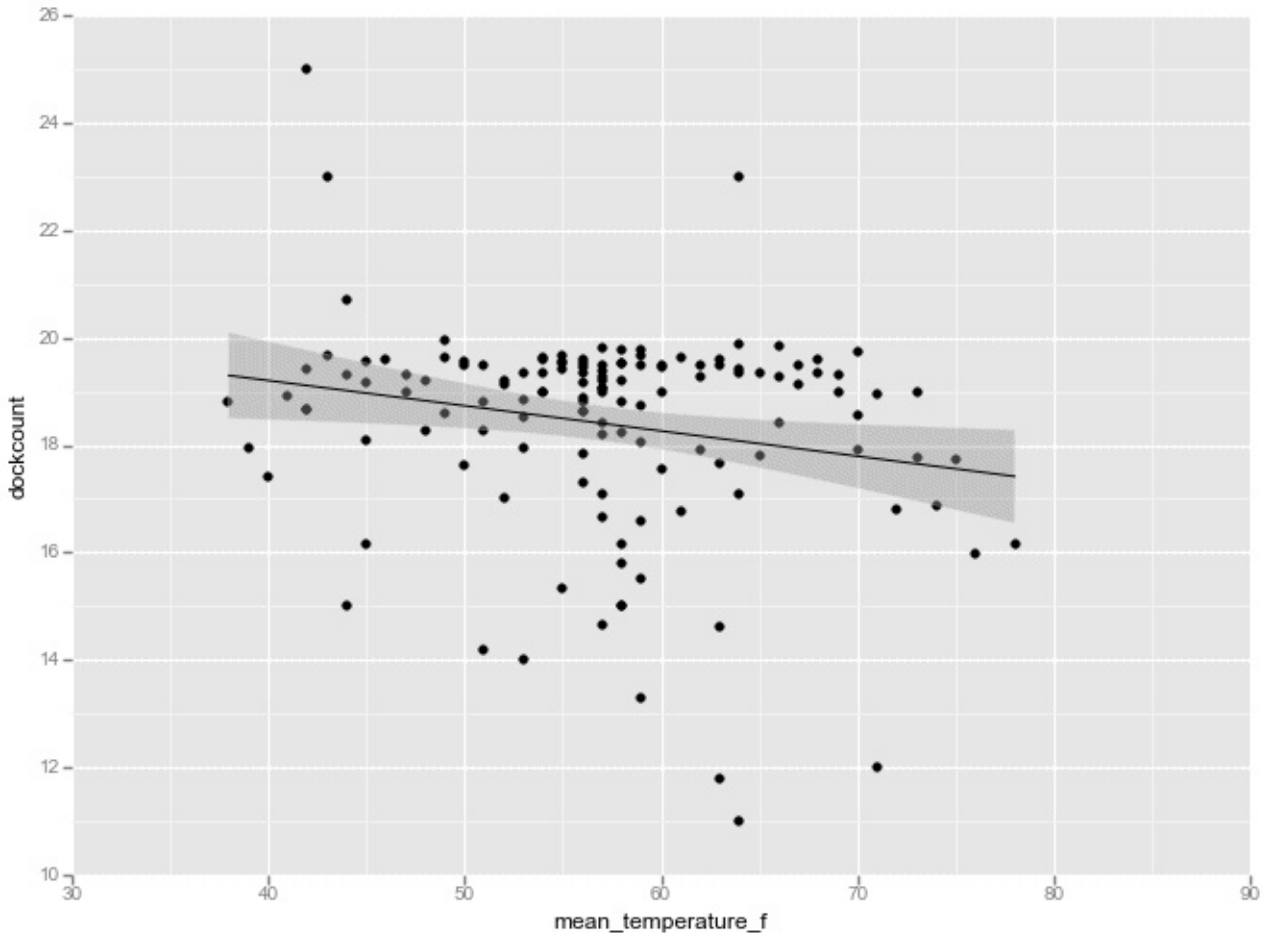
```
dmerge4.groupby(['landmark','subscriptiontype']).agg({'duration_f' : np.sum}).un\
stack().plot(kind="bar")
```



Scatter plot with a smoothing line

This plot shows that the average dock count has a decreasing trend as mean temperature increases

```
ggplot(h, aes(x = 'mean_temperature_f', y ='dockcount')) + geom_point() + geom_s\
mooth(method = 'lm')
```



Data Science Programming in Python - Time Series

Working with time-series data

Time-series data is fun and interesting. Working with time-series data has already been covered extensively in the data munging and grouping & aggregating sections. This chapter aims to cover ad-hoc time-series topics relevant for data analysis. The time- series plots produced by the datasets in these chapters is presented in the visualization section.

We have implemented and shown varying strategies for working with datetime data to gain meaning and insight from the dataset. Now, we'll go a little further in depth into looking at specific dates and times, categorical time intervals, timedeltas, and general properties of datetime and timestamp data.

We can see data analytics over longer periods of time or dates, at a specific time or date, or even just in a particular time interval. In part, this particular dataset has several nice properties and it was chosen to illustrate working with time-series data.

Preliminary work with getting date and time data into a nice format for data analysis has been covered in the data munging section. Now, let's take a deeper look at working with time-series data to gain meaning and insight from our bike sharing dataset.

```
import numpy as np
import pandas as pd
from datetime import datetime, time
```

```
%matplotlib inline
import seaborn as sns
from ggplot import *
```

```
print pd.__version__
print np.__version__
```

```
0.14.1
1.9.0
```

```
%load_ext ipycache
```

Resampling

Resampling allows us to resample to different time periods and show summary statistics.

Resample the dataset to daily means so each row will have the average means for trips started on the same day

Daily means.

```
daily_means = dmerge4.resample('D', how='mean').reset_index(drop=False)
daily_means.head(2)
```

	startdate	tripid	duration	startterminal	endterminal	bike	max_temperature_f	mean_temperature_f	min_temperatureref	max_dew_point_f	...	lat	long	dockcount	day	hour
0	2013-08-29	4661.505348	1561.296791	54.679144	55.017380	431.470588	74.871658	68.491979	61.318182	61.201872	...	37.728908	-122.345651	18.970588	29	15.381016
1	2013-08-30	5837.312325	2703.133053	54.168067	54.177871	439.878151	79.400560	69.901961	60.456583	61.390756	...	37.724274	-122.337827	19.235294	30	14.277311

2 rows × 38 columns

Resample dataset to monthly means so each row will have the average means for trips started in the same month

Monthly Means.

```
monthly_means =  dmerge4.resample('M', how='mean').reset_index(drop=False)
monthly_means.head(2)
```

	startdate	tripid	duration	startterminal	endterminal	bike	max_temperature_f	mean_temperature_f	min_temperaturef	max_dew_point_f	...	lat	long	dockcount	day
0	2013-08-31	5763.824453	2588.182683	55.104186	55.247859	433.639391	75.504757	67.814462	59.870599	60.127973	...	37.730096	-122.346329	19.064700	29.948620
1	2013-09-30	24370.800578	1578.130729	56.731054	56.811908	438.652102	74.284079	65.888048	56.996989	56.373965	...	37.745550	-122.357832	19.261142	16.282415

2 rows × 38 columns

Working with time data

The timeseries functionality in pandas allows granularity down to the second, minute, or hour. For instance, one can look at average duration at four pm on a monthly basis and compare it to eight am. However, it may be more instructive to look at a range such as a time interval like rush hour and the morning commute.

Average duration at eight am and four pm resampled to monthly mean.

```
#the datetime index at 8am resampled to monthly
eight_am = dmerge4.at_time(time(8,0)).resample('M', how='mean')[['duration_f']]
eight_am
```

	duration_f
startdate	
2013-08-31	10.980000
2013-09-30	10.226316
2013-10-31	10.027674
2013-11-30	10.429714
2013-12-31	9.469492
2014-01-31	9.413051
2014-02-28	8.042712

```
four_pm = dmerge4.at_time(time(16,0)).resample('M', how='mean')[['duration_f']]
four_pm
```

	duration_f
startdate	
2013-08-31	16.785000
2013-09-30	14.976786
2013-10-31	26.602927
2013-11-30	19.147391
2013-12-31	11.453214
2014-01-31	12.583846
2014-02-28	14.857083

In every month, duration at four pm is higher than at eight am. When looking at a range of times such as the monthly morning duration and the monthly evening duration, the differences are less delineated. This is interesting because knowing at what hour or range of times bicycles will not be available based on duration is useful for demand and infrastructure planning. Also, this information can inform marketing strategies. Perhaps,

there are incentives that can be given to change duration times now that we know more based on the time-series data.

Look at average duration between 5am and 10am on a monthly basis.

```
morning_monthly = dmerge4.between_time(time(5, 0), time(10, 0)).resample('M', ho\
w='mean')[['duration_f']]
morning_monthly
```

	duration_f
startdate	
2013-08-31	37.648913
2013-09-30	17.973624
2013-10-31	13.302877
2013-11-30	13.209249
2013-12-31	15.384694
2014-01-31	11.679421
2014-02-28	11.441120

Summary statistics for morning time interval.

```
dmerge4.between_time(time(5, 0), time(10, 0)).describe()
```

	tripid	duration	startterminal	endterminal	bike	max_temperature_f	mean_temperature_f	min_temperatureF	max_dew_point_f	meandew_point_f
count	38885.000000	38885.000000	38885.000000	38885.000000	38885.000000	38885.000000	38885.000000	38885.000000	38885.000000	38885.000000
mean	108075.706699	822.261926	59.107522	57.958313	440.186139	65.528764	57.483477	49.009850	49.158262	44.304565
std	54154.593638	4807.622205	16.421185	16.264455	134.382220	7.346052	6.013163	5.988127	5.980818	6.967008
min	4069.000000	60.000000	2.000000	2.000000	9.000000	46.000000	38.000000	25.000000	23.000000	16.000000
25%	62969.000000	339.000000	53.000000	50.000000	353.000000	60.000000	54.000000	45.000000	46.000000	41.000000
50%	108193.000000	491.000000	63.000000	61.000000	448.000000	65.000000	57.000000	49.000000	50.000000	45.000000
75%	155471.000000	688.000000	70.000000	70.000000	546.000000	71.000000	62.000000	53.000000	53.000000	49.000000
max	198297.000000	619322.000000	83.000000	82.000000	717.000000	95.000000	76.000000	66.000000	66.000000	61.000000

8 rows × 37 columns

Look at average duration between three pm and seven pm on a monthly basis.

```
evening_monthly = dmerge4.between_time(time(15, 0), time(19, 0)).resample('M', h\
ow='mean')[['duration_f']]
evening_monthly
```

	duration_f
startdate	
2013-08-31	37.240278
2013-09-30	24.621531
2013-10-31	16.752767
2013-11-30	17.277542
2013-12-31	14.980330
2014-01-31	14.962542
2014-02-28	15.718395

Summary statistics for evening time interval which was chosen to encompass the 'rush hour' time period.

```
dmerge4.between_time(time(15, 0), time(19, 0)).describe()
```

	tripid	duration	startterminal	endterminal	bike	max_temperature_f	mean_temperature_f	min_temperaturef	max_dew_point_f	meandew_point_f
count	47592.000000	47592.000000	47592.000000	47592.000000	47592.000000	47592.000000	47592.000000	47592.000000	47592.000000	47592.000000
mean	101102.863128	1070.586275	56.689212	58.069213	436.845583	66.428854	58.189738	49.515360	49.735649	44.907926
std	56272.442662	6060.799465	16.969014	17.303616	138.118301	7.600397	6.295636	6.244173	6.174970	7.177395
min	4621.000000	60.000000	2.000000	2.000000	9.000000	46.000000	38.000000	25.000000	23.000000	16.000000
25%	51994.500000	359.000000	50.000000	50.000000	348.000000	61.000000	54.000000	45.000000	46.000000	41.000000
50%	99347.000000	542.000000	60.000000	62.000000	446.000000	66.000000	57.000000	50.000000	50.000000	46.000000
75%	150233.250000	794.000000	69.000000	70.000000	545.000000	71.000000	63.000000	54.000000	54.000000	49.000000
max	198701.000000	597517.000000	82.000000	82.000000	717.000000	95.000000	78.000000	66.000000	66.000000	61.000000

8 rows × 37 columns

So, by slicing and dicing the datetime object into morning and evening commuting hours, we've learned that there are 47,592 observations in the evening ; 38,885 in the morning ; 86,477 in this combined time period out of the total, and 10% more observations in the evening than in the morning.

Categorical time intervals

When we munged the data, we created a categorical field for time intervals by extracting the hour component of the time stamp which will be useful for grouping, aggregating, and plotting based on the labelled ranges.

Here is a table of average duration by time of day category.

```
dmerge4.groupby('timeofday')[['duration_f']].mean()
```

	duration_f
timeofday	
evening	17.499759
mid_day	29.401846
morning	16.110753
night	23.363483
wee_hours	60.553483

The results are very interesting. We were curious as to why the data suggested that trips were primarily shorter rides. Splitting the data into labelled time of day intervals reveals some interesting insights. Perhaps evening and morning rides are commutes while non-rush hour times of day are closer to the thirty minute time limit for each trip. But,why is the average duration longer in the wee hours? In the group and aggregation portion of the analysis, we run further queries by time of day to learn more.

Understanding timedeltas

The column we created named diff takes the difference between the datetime objects of end date and start date

```
dmerge4['diff'].head()
```

```
startdate
2013-08-29 15:11:00   00:03:00
2013-08-29 17:35:00   00:13:00
2013-08-29 20:00:00   00:04:00
2013-08-29 17:30:00   00:03:00
2013-08-29 19:07:00   00:14:00
Name: diff, dtype: timedelta64[ns]
```

This will extract minutes from the datetime object column named diff. Also note that this will be different than using hour, minute, extracted columns because those are based on start date only.

```
np.round((dmerge4['diff']/ np.timedelta64(1, 'm') % 60))
```

```
startdate
2013-08-29 15:11:00     3
2013-08-29 17:35:00    13
2013-08-29 20:00:00     4
2013-08-29 17:30:00     3
2013-08-29 19:07:00    14
2013-08-29 12:33:00    14
2013-08-29 19:01:00    13
2013-08-29 19:11:00    25
2013-08-29 14:14:00     6
2013-08-29 19:13:00    23
2013-08-29 19:08:00    13
2013-08-29 19:01:00    14
2013-08-29 13:57:00     2
2013-08-29 09:08:00     3
2013-08-29 14:04:00    28
...
2014-02-28 09:45:00    15
2014-02-28 18:34:00    16
2014-02-28 08:28:00     9
2014-02-28 13:07:00    12
2014-02-28 17:03:00     5
2014-02-28 16:44:00     5
2014-02-28 10:05:00     9
2014-02-28 09:50:00    10
2014-02-28 16:19:00     5
2014-02-28 19:13:00     6
2014-02-28 18:48:00     4
2014-02-28 07:42:00     2
2014-02-28 18:13:00     6
2014-02-28 17:20:00     5
2014-02-28 13:22:00     6
Name: diff, Length: 144015
```

This returns the minutes component of the timestamp which is not what we want rather than total minutes which is what we want for this analysis. We mention it here because this can be useful for when working with financial time-series data in particular.

```
np.round((dmerge4['diff']/ np.timedelta64(1, 'm') % 60)).max()
```

```
59.0
```

By looking at the hour component, notice that most of the trips were under one hour.

```
deltahour = np.round((dmerge4['diff']/ np.timedelta64(1, 'h')))
deltahour
```

```
startdate
2013-08-29 15:11:00    0
2013-08-29 17:35:00    0
2013-08-29 20:00:00    0
2013-08-29 17:30:00    0
2013-08-29 19:07:00    0
2013-08-29 12:33:00    0
2013-08-29 19:01:00    0
2013-08-29 19:11:00    0
2013-08-29 14:14:00    0
2013-08-29 19:13:00    0
2013-08-29 19:08:00    0
2013-08-29 19:01:00    0
2013-08-29 13:57:00    0
```

```
2013-08-29 09:08:00    0
2013-08-29 14:04:00    0
...
2014-02-28 09:45:00    0
2014-02-28 18:34:00    0
2014-02-28 08:28:00    0
2014-02-28 13:07:00    0
2014-02-28 17:03:00    0
2014-02-28 16:44:00    0
2014-02-28 10:05:00    0
2014-02-28 09:50:00    0
2014-02-28 16:19:00    0
2014-02-28 19:13:00    0
2014-02-28 18:48:00    0
2014-02-28 07:42:00    0
2014-02-28 18:13:00    0
2014-02-28 17:20:00    0
2014-02-28 13:22:00    0
Name: diff, Length: 144015
```

This will give the what we want in minutes rather than just extracting the minutes element.

```
dmerge4['diff'].apply(lambda x: x / np.timedelta64(1, 'm'))
```

```
startdate
2013-08-29 15:11:00     3
2013-08-29 17:35:00    13
2013-08-29 20:00:00     4
2013-08-29 17:30:00     3
2013-08-29 19:07:00    14
2013-08-29 12:33:00    14
2013-08-29 19:01:00    13
2013-08-29 19:11:00    25
2013-08-29 14:14:00     6
2013-08-29 19:13:00    23
2013-08-29 19:08:00    13
2013-08-29 19:01:00    14
2013-08-29 13:57:00     2
2013-08-29 09:08:00     3
2013-08-29 14:04:00    28
...
2014-02-28 09:45:00    15
2014-02-28 18:34:00    16
2014-02-28 08:28:00     9
2014-02-28 13:07:00    12
2014-02-28 17:03:00     5
2014-02-28 16:44:00     5
2014-02-28 10:05:00     9
2014-02-28 09:50:00    10
2014-02-28 16:19:00     5
2014-02-28 19:13:00     6
2014-02-28 18:48:00     4
2014-02-28 07:42:00     2
2014-02-28 18:13:00     6
2014-02-28 17:20:00     5
2014-02-28 13:22:00     6
Name: diff, Length: 144015
```

The max function shows that total minutes rather than just the minute component is returned. Success.

```
dmerge4['diff'].apply(lambda x: x / np.timedelta64(1, 'm')).max()
```

```
12037.0
```

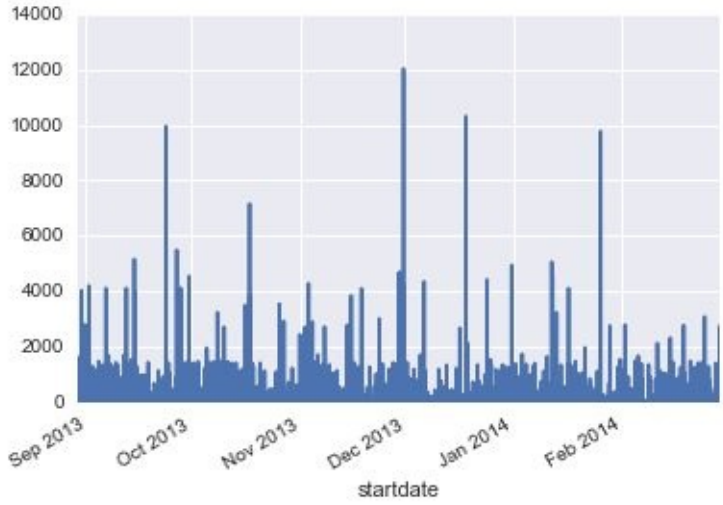
The will give the same result without a lambda function

```
np.round((dmerge4['diff']/ np.timedelta64(1, 'm'))).max()
```

```
12037.0
```

```
dmerge4['diff'].apply(lambda x: x / np.timedelta64(1, 'm')).plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe640261e10>
```



One thing to keep in mind is that pandas uses numpy datetime while there is also a datetime.datetime in python. This is something to keep in mind for general python users. The latest release of pandas also introduces a new datetime accessor for working with dates and times and is worth checking out.

Afterword

What are the next steps? This is just the beginning. Continue practicing the techniques learned in this book on your own datasets. Repetition and practice are the keys to understanding and skill. You are now on your way to analyzing data with code. Remember to keep learning and always have fun.