In [1]:

```
import forecastio
import numpy as np
import pandas as pd
import seaborn as sns
from itertools import combinations
from delorean import Delorean
%pylab inline

sns.set_context("notebook")
```

```
Populating the interactive namespace from numpy and matplotlib
```

Integrating Weather Data

Motivation

Many of the use cases for our project entail making inferences about variables that are plausibly effected by weather conditions. Some examples include:

- **Equipment lifetime**
- **Job execution time**
- **Reducible idling time**

If we can incorporate hi-resolution weather data into our models, our insights will be more accurate and we will be able to make more confident inferences.

Forecast.io

Forecast.io is application that purports to offer present and historical queries for rich weather information, plus forecasts, from any coordinates on the globe. We will evaluate this claim below. The platform incorporates data from a large variety of sources, including several from the NCDC and other sources around the globe.

Forecast.io is accessed via a well-documented REST API (https://developer.forecast.io/) and is free for the first 1000 calls / day. After that they start charging a fraction of a cent.

They also make a pretty sweet iPhone app called Dark Sky (http://darkskyapp.com/), which will ping you ten minutes before it starts raining.

NOAA / NCDC

The National Oceanic and Atmospheric Administration is really good about opening up data collected by its weather stations positioned around the country to the public. A wide variety of data products are available for consumption, from very coarse (day-level quality-controlled summaries) to very fine-grained (minute-level weather data from ASOS (http://www.nws.noaa.gov/ost/asostech.html) stations maintained by the NWS, FAA, and DoD).

The primary NCDC source that we have proposed to use for fine-grained US weather data is the ISD (http://www.ncdc.noaa.gov/isd) (Integrated Surface Database). The ISD goes down to 20-minute timesteps in most places and itself incorporates data from several different types of weather stations, located thorughout the country. ISD is sort of the "Gold Standard" for NCDC weather data -- comprehensive, pretty well quality-controlled, reliable.

You can pull data from their ftp server, located at ftp.ncdc.noaa.gov (ftp://ftp.ncdc.noaa.gov). There are usually PDFs that describe the format (sometimes arcane) of whatever you're looking at.

The Questions at Hand

- **Is Forecast.io a suitable proxy for raw ISD data?**
  This would be nice, because Forecast.io is much easier to get at than ISD data, is pre-organized and -processed, covers more territory (the whole earth(!)) and supports real-time querying.
- **Do Forecast.io's interpolation strategies produce credible estimates?** Forecast.io's innovation is that they are able to produce a "smooth surface" describing the weather conditions over a given geo area and time. Using historical data from ISD, we can design an experiment to make sure they're not pulling our leg.

To business!

We'll begin by looking at the ISD data, and getting it into shape so that we can compare with Forecast.io.

In [2]:
```
# I've pulled a bunch of historical ISD data from Colorado, and cleaned it up for inspection.
# The git repo containing the R script and bash scripts used to download and clean is here:
# https://github.com/hack-c/weathergrab
# No promises of portability for the shell scripting, but the R script that pulls the data doesn't have any dependencies.

stations = pd.read_csv("data/stations.csv")          # my script generates this "index" file with meta info about stations.
stations.columns = map(str.lower, stations.columns)  # lowercase the column names
```

In [3]:
```
stations.head()
```

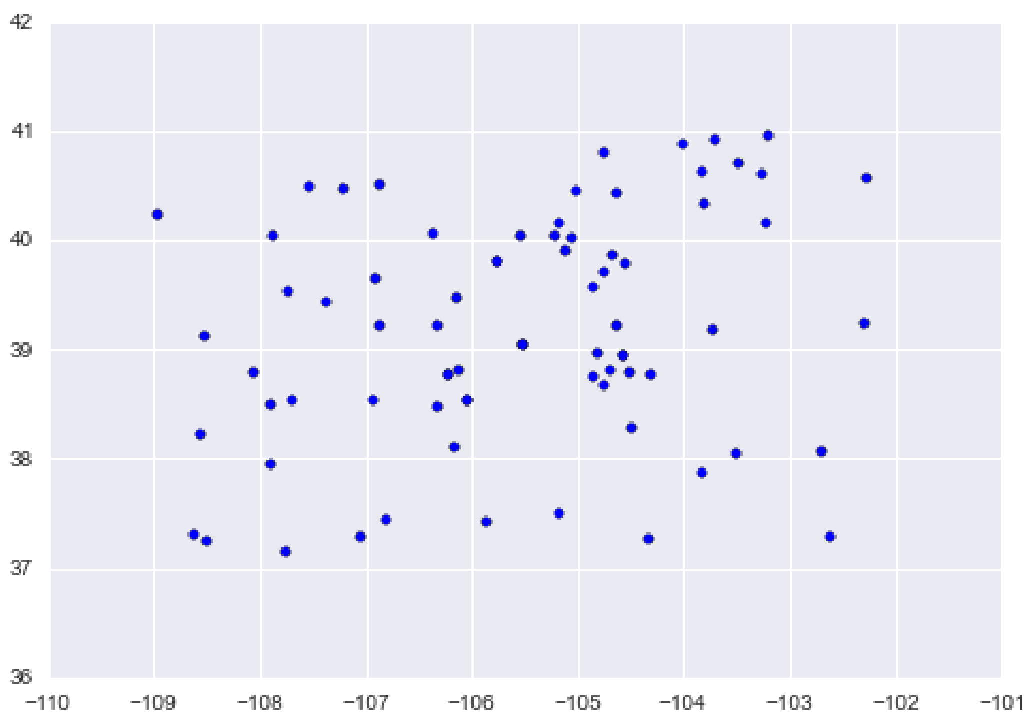
Out[3]:

	usafid	wban	yr	lat	long	elev
0	720262	94076	2015	40.054	-106.368	2259
1	720385	419	2015	39.800	-105.767	4113
2	720385	99999	2015	39.800	-105.767	4113
3	720528	99999	2015	38.817	-106.117	2423
4	720529	429	2015	38.233	-108.567	1811

Let's look for a few stations that are closely clustered.

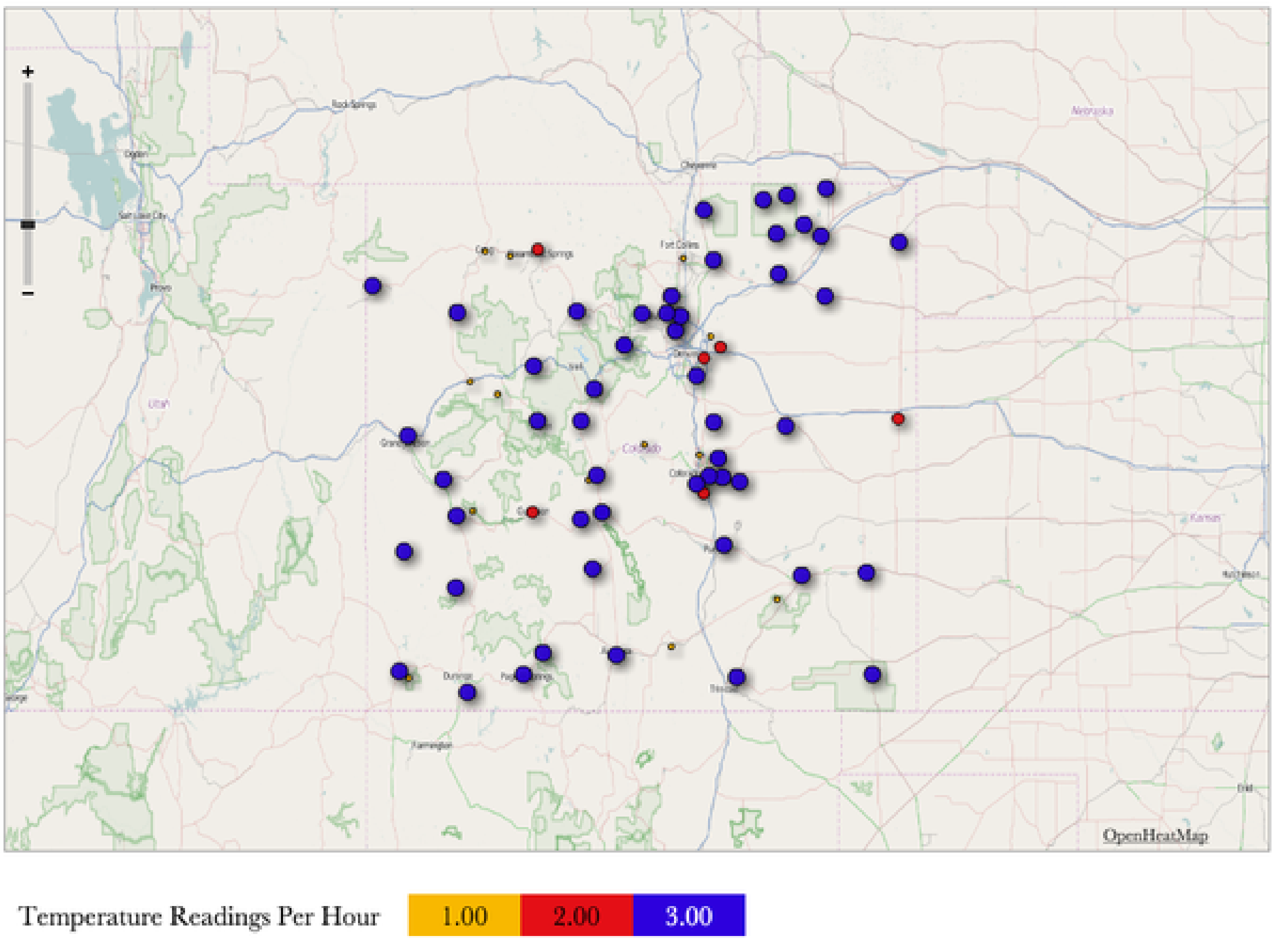
In [4]:
```
plt.scatter(stations["long"], stations["lat"])
```

Out[4]: <matplotlib.collections.PathCollection at 0x109a52fd0>



Looks about right. (Interactive map [here (http://www.openheatmap.com/view.html?map=TrochlearsSpladKickboxer)](http://www.openheatmap.com/view.html?map=TrochlearsSpladKickboxer))

Seems there's a solid cluster in the NE corner of the map. We'll use that in a minute.



Now to look at some of the actual time series weather data for these stations.

The ISD data I pulled is organized by year and station in csv files, with filenames formatted like the below.

We'll build a new column `filename` for our "index" dataframe (`stations`), and then load in timeseries data from the stations in the northeast corner of the map.

In [5]:
```
ls data/csv/ | head -n5
```

```
720262-94076-2015.csv
720385-00419-2015.csv
720385-99999-2015.csv
720528-99999-2015.csv
720529-00429-2015.csv
```

So let's construct a column with the filename for each station, for easy accessing.

In [6]:
```
def construct_filename(s):
    """
    Format the filename for a given station,
    coercing types and padding with zeros where necessary.
    """
    usafid = "{0:06d}".format(int(s.usafid))  # use new python string fo
rmatting
    wban   = "{0:05d}".format(int(s.wban))
    return "{usafid}-{wban}-2015.csv".format(usafid=usafid, wban=wban)

stations['filename'] = stations.apply(construct_filename, axis=1)
stations.index = stations['filename']  # use this as the rowkey
del stations['filename']               # no need to keep the duplicate c
olumn

# this ought to do for an example subset.
# restrict to the NE corner of the map.
subset = stations[stations.apply(lambda s: s.long > -103.5 and s.lat > 4
0.5, axis=1)]

subset
```

Out[6]:

	usafid	wban	yr	lat	long	elev
filename						
720537-99999-2015.csv	720537	99999	2015	40.569	-102.273	1137
720544-00168-2015.csv	720544	168	2015	40.615	-103.265	1231
720987-99999-2015.csv	720987	99999	2015	40.967	-103.200	1380
720991-99999-2015.csv	720991	99999	2015	40.700	-103.483	1336

In [7]:
```
dfs = [pd.read_csv("data/csv/" + filename) for filename in subset.index]
# load them in
df = pd.concat(dfs)
# stack them on top of one another
df.columns = map(str.lower, df.columns)
# lowercase column names
df['filename'] = df.apply(construct_filename, axis=1)
# make filename column for easy reference


# let's use pandas builtin datetime indexing capabilities to make life e
asier.
df.index = pd.DatetimeIndex(df.apply(lambda s: pd.datetime(s["yr"],
s["m"], s["d"], s["hr"], s["min"]), axis=1))

df.head()
```

Out[7]:

	usafid	wban	yr	m	d	hr	min	lat	long	elev	wind.dir	wind.spd	te
2015-01-01 00:15:00	720537	99999	2015	1	1	0	15	40.569	-102.273	1137	290	4.6	-1
2015-01-01 00:35:00	720537	99999	2015	1	1	0	35	40.569	-102.273	1137	290	4.1	-8
2015-01-01 00:55:00	720537	99999	2015	1	1	0	55	40.569	-102.273	1137	290	4.1	-1
2015-01-01 01:15:00	720537	99999	2015	1	1	1	15	40.569	-102.273	1137	290	5.1	-1
2015-01-01 01:35:00	720537	99999	2015	1	1	1	35	40.569	-102.273	1137	300	5.1	-1

Forecast.io API

Now that we have a manageable piece of ISD data in good shape for testing out some queries, let's make some comparisons with what we get back from Forecast.io.

Forecast.io exposes a REST API, which also has a nice Python wrapper available on PyPi and [GitHub (https://github.com/ZeevG/python-forecast.io)](https://github.com/ZeevG/python-forecast.io).

In [8]:
```
# set the parameters we'll use to fetch weather data from the API.
api_key  = "d35a1f70a0f565c5b6015c47140e2a28"
lat, lng = subset.lat.mean(), subset.long.mean()  # find a point in the
middle of the four stations
time = pd.datetime(2015, 1, 1, 1)                 # 1am on New Year's Da
y

# make the API call.
forecast = forecastio.load_forecast(api_key, lat, lng, time=time, unit
s="si")
```

In [9]:
```
# current conditions from the API response object look like this
forecast.currently().d
```

Out[9]:
```
{u'apparentTemperature': -25.05,
 u'cloudCover': 0,
 u'dewPoint': -20.7,
 u'humidity': 0.66,
 u'icon': u'clear-night',
 u'precipIntensity': 0,
 u'precipProbability': 0,
 u'pressure': 1029.57,
 u'summary': u'Clear',
 u'temperature': -15.88,
 u'time': 1420092000,
 u'visibility': 16.09,
 u'windBearing': 276,
 u'windSpeed': 5.31}
```

Experiment

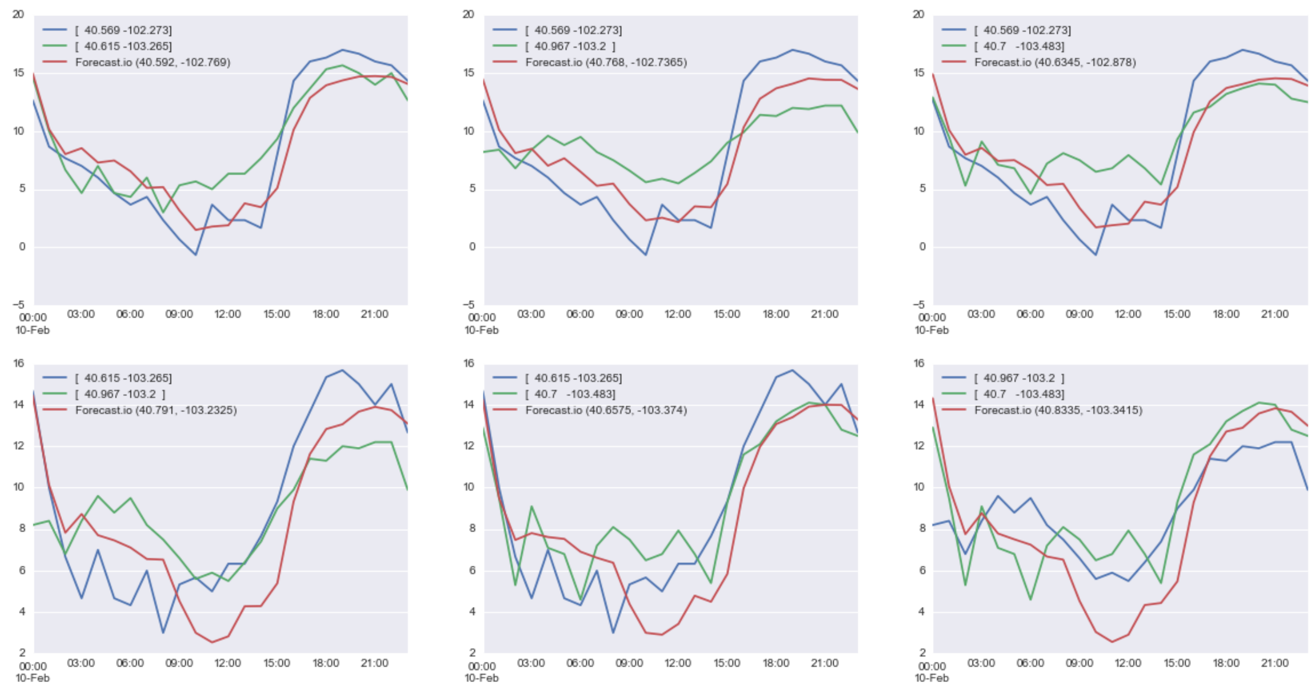
Here we choose each pair from the subset of ISD stations we chose, and compare the temperature at those stations to Forecast.io's estimate at the midpoint between them.

If we make the naïve assumption that the temperature gradient in Northeastern Colorado is "smooth", we would expect Forecast.io's estimate to track between the bands of the neighboring stations.

In [22]:
```
def construct_plotset(date):
    """
    build the dataset to plot
    """
    day  = df[date]
    fns  = day.filename.unique()

    # define some utility functions for extracting temperature from fore
castio and getting labels for stuff
    def get_temp(lat, lng, dt):
        """
        return temperature at the centroid at given datetime from foreca
st
        """
        time = Delorean(dt, timezone="UTC").shift("US/Eastern").datetime
# ISD data is in UTC, but Forecast takes local time
        return forecastio.load_forecast(api_key, lat, lng, time=time, un
its="si").currently().d["temperature"]

    def coords(filename):
        """
        use the `stations` meta info dataframe to get coordinates for a
filename.
        """
        return np.array((stations.ix[filename].lat, stations.ix[filenam
e].long))

    # datetime index
    dtindex = pd.date_range(start=date, periods=24, freq="h")
    times = map(lambda x: x.to_datetime(), dtindex)

    # now, for each of the six midpoints between two stations, construct
a plotdf
    # containing station 1 temp data, station 2 temp data, and forecasti
o's temp at the midpoint.
    plotdfs = []
    for s1, s2 in combinations(fns, 2):
        plotdf = pd.DataFrame(index=dtindex)
        plotdf[str(coords(s1))] = day[day.filename == s1].temp.resampl
e("1h")
        plotdf[str(coords(s2))] = day[day.filename == s2].temp.resampl
e("1h")

        midptlat, midptlng = tuple((coords(s1) + coords(s2)) / 2)  # mid
point between stations
        plotdf["Forecast.io ({}, {})".format(midptlat, midptlng)] = pd.S
eries([get_temp(midptlat, midptlng, t) for t in times],

index=dtindex)

        plotdfs.append(plotdf)

    return plotdfs
```

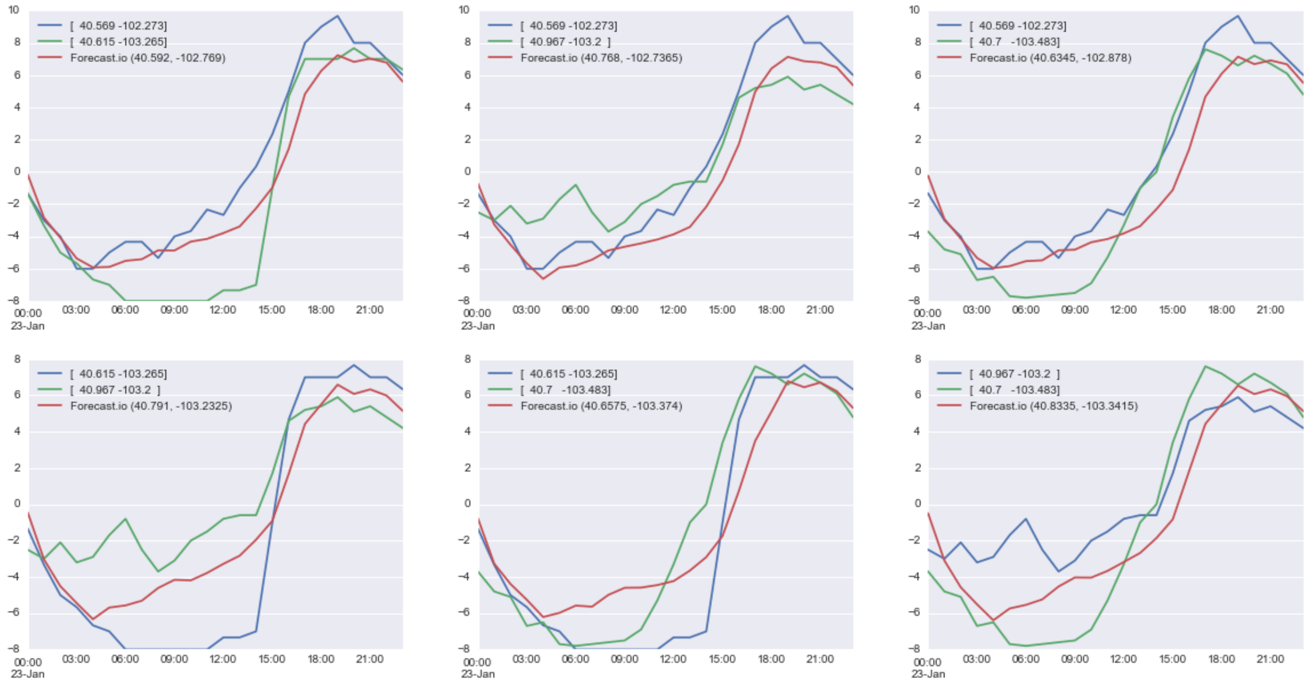
In [23]:

```
# plot

def plot_temp(date):
    """
    plot the station - forecast triplet for the given day
    """
    plotdfs = construct_plotset(date)

    fig, axes = plt.subplots(nrows=2, ncols=3)

    for i in range(6):
        plotdfs[i].plot(ax=axes[i/3,i%3], figsize=(20,10), sharex=False, sharey=False)

plot_temp("February 10, 2015")
```



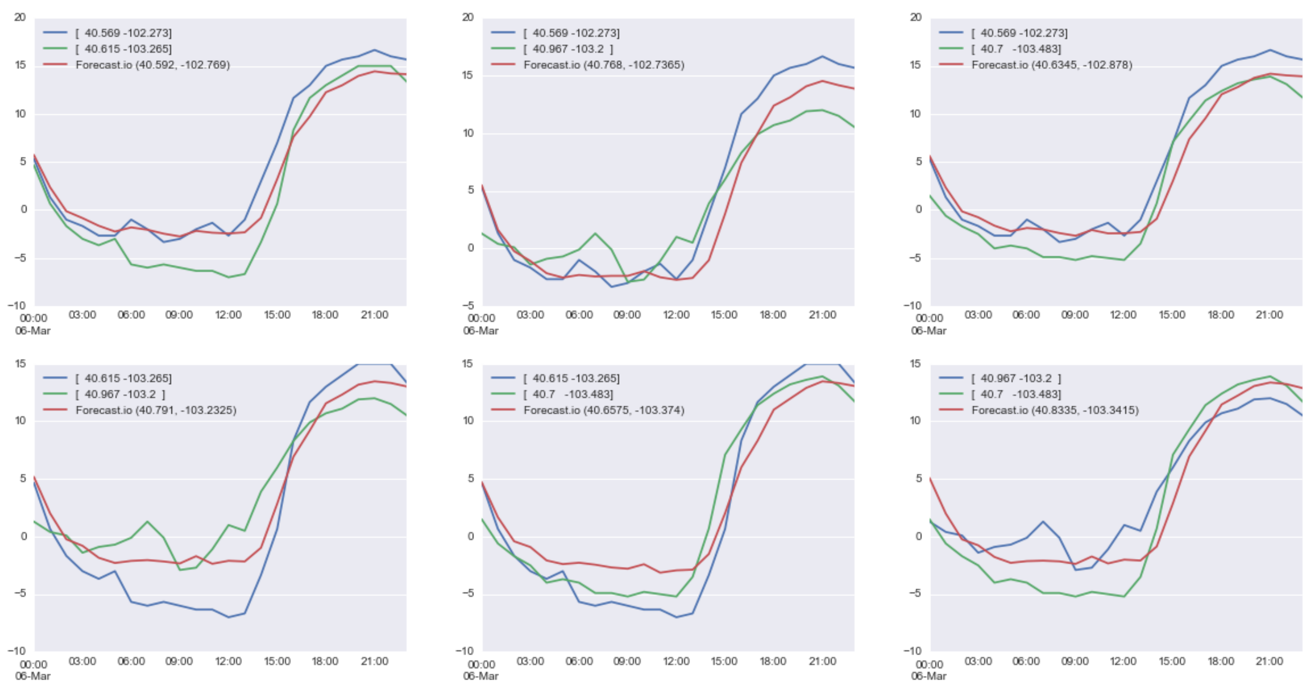
Forecast is "leading" ever so slightly.

In [24]:
```
plot_temp("January 23, 2015")
```

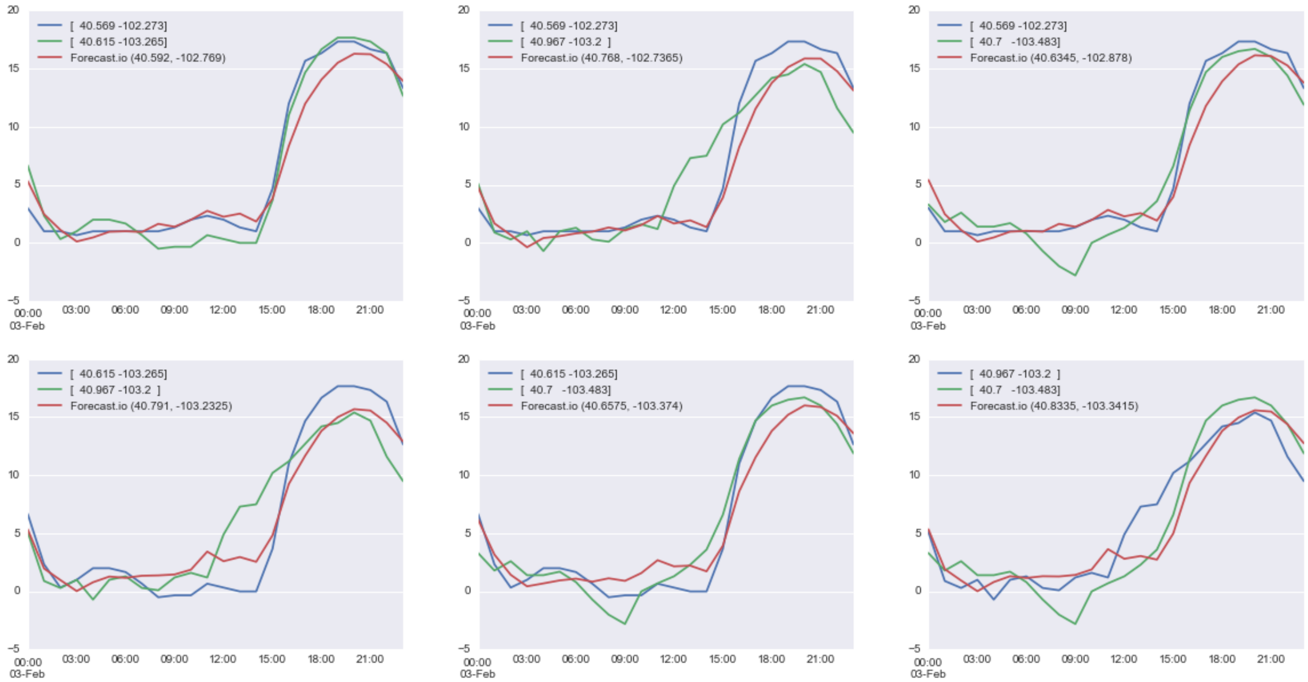


Here it's tracking between the bands pretty well.

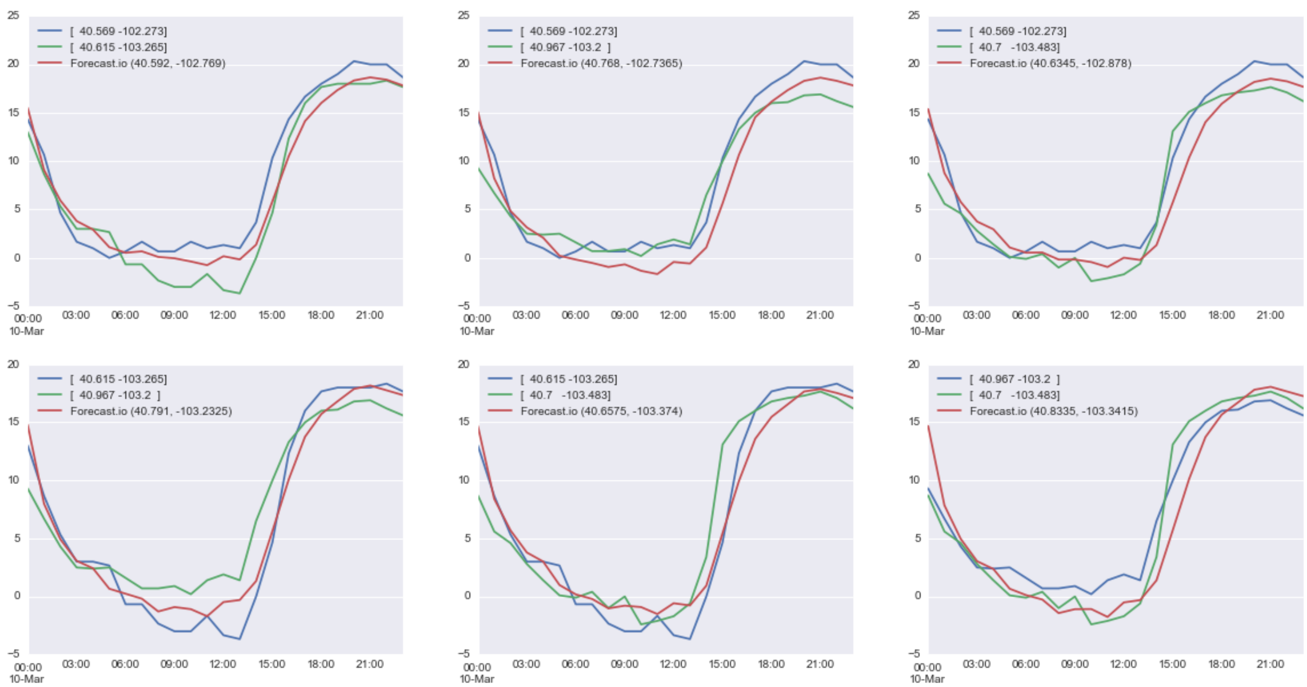
In [25]:
```
plot_temp("March 6, 2015")
```



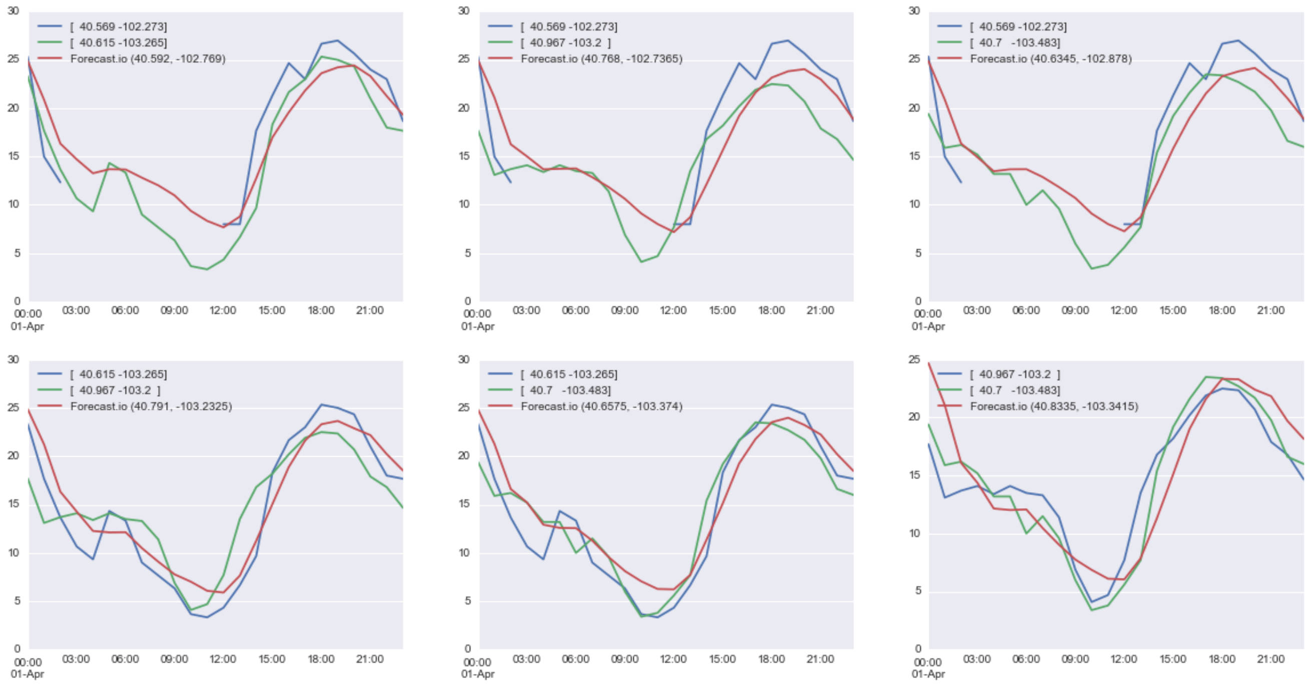
```
In [28]: plot_temp("February 3, 2015")
```




```
In [29]: plot_temp("March 10, 2015")
```



In [30]:
```
# sometimes there's a bit of missing data
plot_temp("April 1, 2015")
```



Takeaways

• ISD data comes in silly formats (fixed-width files anyone?)
• Timezone support is full of gotchas • Check your units!
• All told though, Forecast.io appears to meet our need for this specific geography and time.

Thanks for reading!

Charlie Hack
charles.hack@accenture.com (mailto:charles.hack@accenture.com)

Jordan Thomas jordan.thomas@accenture.com (mailto:jordan.thomas@accenture.com)