Intro To Data Science Project 1. Analyzing the NYC Subway Dataset

Russell Fischer

Intro to Data Science, March 29, 2015 (revised April 17, 2015)

Note: the answers to the "Short Questions" document are directly below. Supporting code and visualizations follow those answers.

Section 1: Statistical Test - Does rain effect subway ridership?

1.1. Choice of statistical test - Which statistical test did you use to analyze the NYC subway data? Did you use a one-tail or two-tail p-value? What is the null hypothesis? What is your critical p-value?

The data did not appear to have a recognizable distribution (see Figure 3), so a non-parametric test was selected. In this case, we use the Mann-Whitney U test, two-tailed, as we will be interested if ridership falls or increases significantly on rainy days (e.g., ridership might increase on rainy days because people take the train instead of walking, or ridership might decrease on rainy days because people are likely to stay home).

The null hypothesis is: the ridership on rainy and non-rainy days is the same and any difference we see is due to random sampling (we assume the statistical distribution of ridership on rainy and non-rainy days is the same). If the p-value we calculate is less than our critical value we will reject the hypothesis. If the p-value is larger we cannot reject the null hypothesis and there is no evidence that ridership differs depending on rain. I chose a conventional significance level, 0.05.

The 'rain' and 'precipi' variables both address rainfall. The former variable is a simple binary indication of whether it rained on the given day. 'Precipi' seems to be the quantity of rain that has fallen, measured with a 4-hour periodicity. For this analysis, I decided it was more appropriate to utilize the daily value, understanding that a decision to ride the subway because of rain does not simply depend on precipitation at the moment the decision is made, but on the **expectation** of rain and whether it will rain during the traveler's anticipated trip time. We do not have variables that address these considerations, so the best course of action is the simplest: an initial analysis with a timescale of days, not hours.

1.2. Why is this statistical test applicable to the dataset? In particular consider the assumptions that the test is making about the distribution of ridership in the two samples.

The Mann-Whitney U test is a non-parametric test, so we do not have to characterize the data as being drawn from any specific type of statistical distribution. We do have to assume the populations are drawn at random, which I think in this case probably means that there is a 50% chance of rain on any given day. We did not try to establish whether this is true.

1.3. What results did you get from this statistical test? These should include the following numerical values: p-values as well as the means for each of the two samples under test.

Mean daily turnstile entries, with rain: 2562036.29.

Mean daily entries, without rain: 2611790.04.

p-value is: 0.33

1.4 What is the significance and interpretation of these results?

We have no evidence that ridership differs between rainy and non-rainy days. I repeated this test under different conditions (weekdays only, weekends only) and reached the same conclusion.

Section 2: Linear Regression

2.1 What approach did you use to compute the coefficients theta and produce predictions for ENTRIESn_hourly in your regression model?

The gradient descent algorithm was used to fit a linear model to the data.

2.2 What features (input variables) did you use in your model? Did you use any dummy variables as part of your features?

The features used in this regression are the hour-of-the-day, the turnstile entries 24-hours-ago, the turnstile ID (categorical variable) and the day-of-the-week (categorical variable). I performed a lot of experimentation with other variables, including the weather variables, but felt this selection of features acheived the best fit while keeping the feature set relatively small.

2.3 Why did you select these features in your model? We are looking for specific reasons that lead you to believe that the selected features will contribute to the predictive power of your model.

Our previous hypothesis test seemed to indicate that rain had no significant affect on subway ridership. We therefore chose to ignore the weather variables while building our model. Factors such as mean temperature might certainly play a role in subway ridership, but since our data is for only one (relatively warm) month, we don't expect weather to influence ridership to the same degree as date and time-of-day.

The motivation for feature selection was as follows:

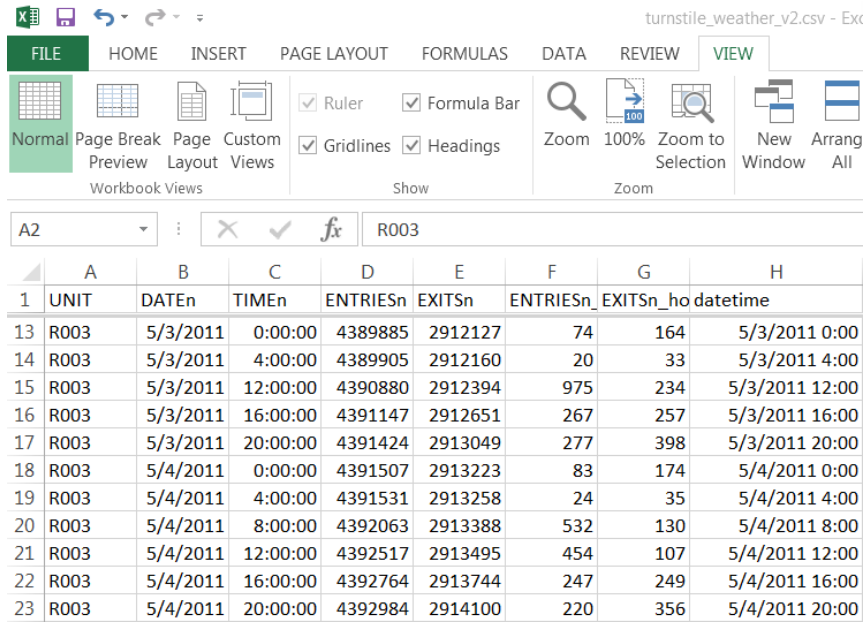
- Hour-of-day and day-of-week are included as variables because we expect a strong diurnal pattern to the data (people tend to sleep and work at regular times during the day), and business schedules are typically different between weekdays and weekends. We confirmed this assumption with data visualizations (Figure 4-5).
- Subway schedules vary between weekends, weekdays and holidays. A priori, we expect knowing what type of day we are dealing with will be important to building a model.

- We include turnstile ID as a categorical variable. Our assumption here is that there are consistent differences in ridership between turnstiles; we have no reason to expect rideship to be constant across turnstiles.
- **One of the most important discoveries I made in my data explorations was that one of the best predictors of today's turnstile entries was yesterday's turnstile entries.** In other words there is a some level of serial correlation in the data set. I characterized only one correlation in the data, the 24-hour-lagged value of turnstile entries.

Because the original data set did not provide values at every four-hour timestep, it was difficult to get the 24-hour lagged value by simply shifting through rows of the dataset. In the original data shown below, you can see there is no 8:00 value for turnstile R003 on May 3rd. So on May 4th for turnstile R003, if I would like to find the 8:00 value for May 3rd, simply shifting back five rows (24 hours) would be incorrect, because the 8:00 May 3rd value is absent.

In [60]:
```
from IPython.display import Image
Image(filename='turnstile_weather_v2.csv - Excel.png', width=400)
```

Out[60]:

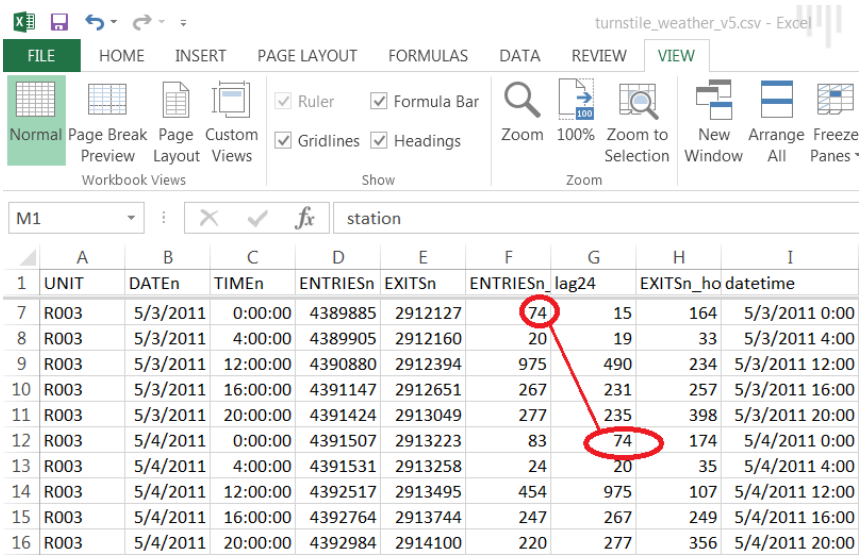


	A	B	C	D	E	F	G	H
1	UNIT	DATEn	TIMEn	ENTRIESn	EXITSn	ENTRIESn_	EXITSn_ho	datetime
13	R003	5/3/2011	0:00:00	4389885	2912127	74	164	5/3/2011 0:00
14	R003	5/3/2011	4:00:00	4389905	2912160	20	33	5/3/2011 4:00
15	R003	5/3/2011	12:00:00	4390880	2912394	975	234	5/3/2011 12:00
16	R003	5/3/2011	16:00:00	4391147	2912651	267	257	5/3/2011 16:00
17	R003	5/3/2011	20:00:00	4391424	2913049	277	398	5/3/2011 20:00
18	R003	5/4/2011	0:00:00	4391507	2913223	83	174	5/4/2011 0:00
19	R003	5/4/2011	4:00:00	4391531	2913258	24	35	5/4/2011 4:00
20	R003	5/4/2011	8:00:00	4392063	2913388	532	130	5/4/2011 8:00
21	R003	5/4/2011	12:00:00	4392517	2913495	454	107	5/4/2011 12:00
22	R003	5/4/2011	16:00:00	4392764	2913744	247	249	5/4/2011 16:00
23	R003	5/4/2011	20:00:00	4392984	2914100	220	356	5/4/2011 20:00

So, external to Python, I generated a variation of the dataset where there was a value of hourly turnstile entries at a regular 4-hour sampling. Any missing data was treated as "NA". Now you can see in the screenshot below, I have a new column 'lag24' which contains the 24-hour lagged value for turnstile entries by a simple lookback over the rows of the dataset. Any row that did not have a 24-hour lagged value (because it was NA in the original data) was dropped from analysis. This enhancement of the dataset did not change any data other than adding the new 24-hour lagged value, and dropping any rows for which this value was not present. I am including the enhanced file 'turnstile_weather_v5.csv' with my project if a review is necessary.

In [61]:
```
from IPython.display import Image
Image(filename='turnstile_weather_v5.csv - Excel.png', width=400)
```

Out[61]:



	A	B	C	D	E	F	G	H	I
1	UNIT	DATEn	TIMEn	ENTRIESn	EXITSn	ENTRIESn_	lag24	EXITSn_ho	datetime
7	R003	5/3/2011	0:00:00	4389885	2912127	74	15	164	5/3/2011 0:00
8	R003	5/3/2011	4:00:00	4389905	2912160	20	19	33	5/3/2011 4:00
9	R003	5/3/2011	12:00:00	4390880	2912394	975	490	234	5/3/2011 12:00
10	R003	5/3/2011	16:00:00	4391147	2912651	267	231	257	5/3/2011 16:00
11	R003	5/3/2011	20:00:00	4391424	2913049	277	235	398	5/3/2011 20:00
12	R003	5/4/2011	0:00:00	4391507	2913223	83	74	174	5/4/2011 0:00
13	R003	5/4/2011	4:00:00	4391531	2913258	24	20	35	5/4/2011 4:00
14	R003	5/4/2011	12:00:00	4392517	2913495	454	975	107	5/4/2011 12:00
15	R003	5/4/2011	16:00:00	4392764	2913744	247	267	249	5/4/2011 16:00
16	R003	5/4/2011	20:00:00	4392984	2914100	220	277	356	5/4/2011 20:00

2.4 What are the coefficients (or weights) of the non-dummy features in your linear regression model?

The weights for the 'hour-of-the-day' and 'turnstile entries 24-hours-ago' features are: [262.06980169 2173.095726]

2.5 What is your model's R-sq (coeff of determination) value?

The R^2 of the model is 0.76

2.6 What does this R-sq mean for the goodness of fit of your regression model? Do you think this linear model to predict ridership is appropriate for this dataset, given this R-sq value?

At the grader's request I included all turnstiles. Previously, I used an outlier detection scheme to remove some turnstiles and I achieved a relatively good coefficient of determination, 0.85. Now the R-sq is 0.76. Before I discovered the 'turnstile entries 24-hours-ago' feature in my modeling attempts, my R-sq values were always less than 0.5. I conclude that it is possible to predict ridership using a linear regression model, provided the modeler has a good understanding of the dependencies in the dataset. However there is room to improve the goodness of fit, either with more sophisticated modeling techniques, or perhaps by a deeper look into the quality of the data to verify the input data is well-matched to the modeling assumptions.

Section 3: Visualization

3.1 One visualization should contain two histograms: one of the ENTRIESn_hourly for rainy days and one of ENTIRESn_hourly for non-rainy days.

In [62]:
```
import pandas as pd
import pandasql
from pandasql import sqldf
import seaborn as sns
import numpy as np

# all days
improved_df = pd.read_csv('C:/Users/rf6994/Documents/udacity/data science nanodegre
e/improved-dataset/turnstile_weather_v5.csv')

print "The number of entries in the data set is: %d" % len(improved_df)
print "The number of turnstiles is: %d" % len(improved_df.UNIT.unique())

#q = """
#    SELECT rain, DATEn, SUM(cast (ENTRIESn_hourly as integer)) as total
#    FROM improved_df
#    WHERE UNIT='R003'
#    GROUP BY DATEn
#    """
q = """
    SELECT rain, DATEn, ENTRIESn_hourly as total
    FROM improved_df
    WHERE day_week IN ('0','1','2','3','4')
    """
selection = pandasql.sqldf(q, locals())

num_bins = 100
start_bin = 0 #min(mondays['total'])
stop_bin = 1000
bin_width = (stop_bin - start_bin)/num_bins
bin_list = np.arange(start_bin, stop_bin, bin_width)

%pylab inline
fig = plt.figure(figsize=[20,5])
ax = fig.add_subplot(111)
ax = selection[selection.rain == 0]['total'].plot(kind='hist', bins = bin_list, lab
el='no_rain', alpha = 0.5, color='g')
ax = selection[selection.rain == 1]['total'].plot(kind='hist', bins = bin_list, lab
el='rain', alpha = 0.7, color='b')
ax.legend()
ax.set_ylabel('Frequency')
ax.set_xlabel('Entries')
ax.set_title('Histrogram, ENTRIESn_hourly')
plt.show()
```

```
The number of entries in the data set is: 40060
The number of turnstiles is: 240
Populating the interactive namespace from numpy and matplotlib
```

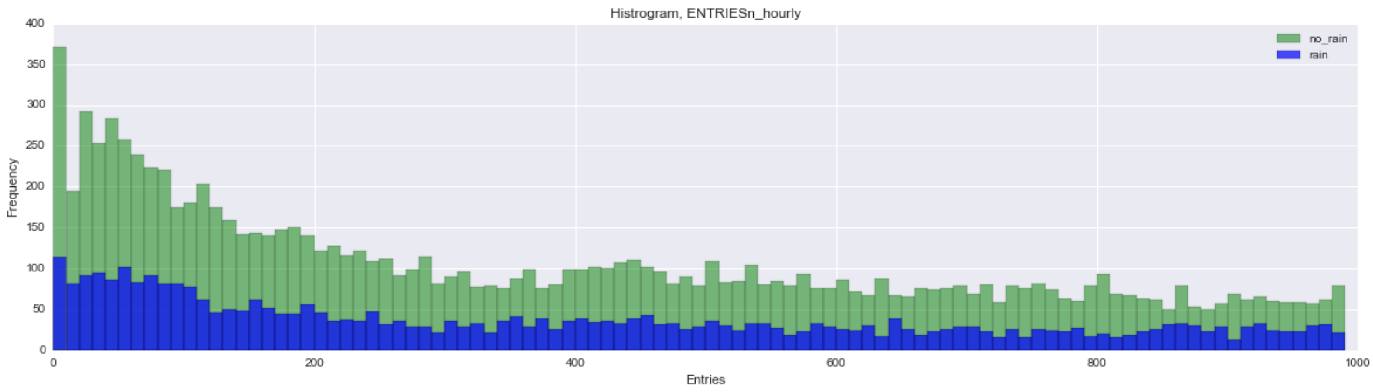


Figure 3. Histogram visualization of the number of turnstile entries on rainy and non-rainy weekdays. By visual inspection, it was decided that a normal distribution was not an appropriate model for the data, and a non-parametric test was consequently selected for statistical testing.

3.2 One visualization can be more freeform. You should feel free to implement something that we discussed in class (eg scatter plots, line plots) or attempt to implement something more advanced if you'd like.

Let's compare ridership data for weekdays to ridership on weekends and holidays. We will compare in two ways, 1. histogram plot of the total daily ridership on weekdays on non-weekdays, 2. total daily ridership displayed as a timeseries. We are exploring our data's dependency on time to get a sense of what underlying variables influence our data. If the distribution of our data appears to depend strongly on day-of-week or hour-of-day for example, it will likely be important to build those dependencies into our regression model

In [63]:
```
import pandas
import pandasql
from pandasql import sqldf
import seaborn as sns

# we suspect ridership differs depending whether the day is weekday, weekend, holid
ay
saturdays=["7","14","21","28"]
sundays  =["1","8","15","22","29"]
holidays =["30"]
days_off =saturdays+sundays+holidays
days_off =map(lambda x: "5/"+x+"/2011",days_off)

#grouping by day
q = """
    SELECT DATEn, SUM(cast (ENTRIESn_hourly as integer)) as total
    FROM improved_df
    GROUP BY DATEn
    """
df_bytype = pandasql.sqldf(q, locals())


# two day classifications: weekday and otherwise
weekdays = df_bytype.loc[-df_bytype['DATEn'].isin(days_off)]
non_weekdays = df_bytype.loc[df_bytype['DATEn'].isin(days_off)]

num_bins = 30
start_bin = 0 #min(non_weekdays['total'])
stop_bin = max(weekdays['total'])
bin_width = (stop_bin - start_bin)/num_bins
bin_list = np.arange(start_bin, stop_bin, bin_width)

%pylab inline
fig = plt.figure(figsize=[20,5])
ax = fig.add_subplot(111)
ax = weekdays['total'].plot(kind='hist', bins = bin_list, label='weekday', alpha =
0.5, color='g')
ax = non_weekdays['total'].plot(kind='hist', bins = bin_list, label='other', alpha
= 0.7, color='b')
ax.legend()
ax.set_ylabel('Frequency')
ax.set_xlabel('ENTRIESn_hourly')
ax.set_title('Ridership Distribution on Weekdays and Weekends')
plt.show()
```

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

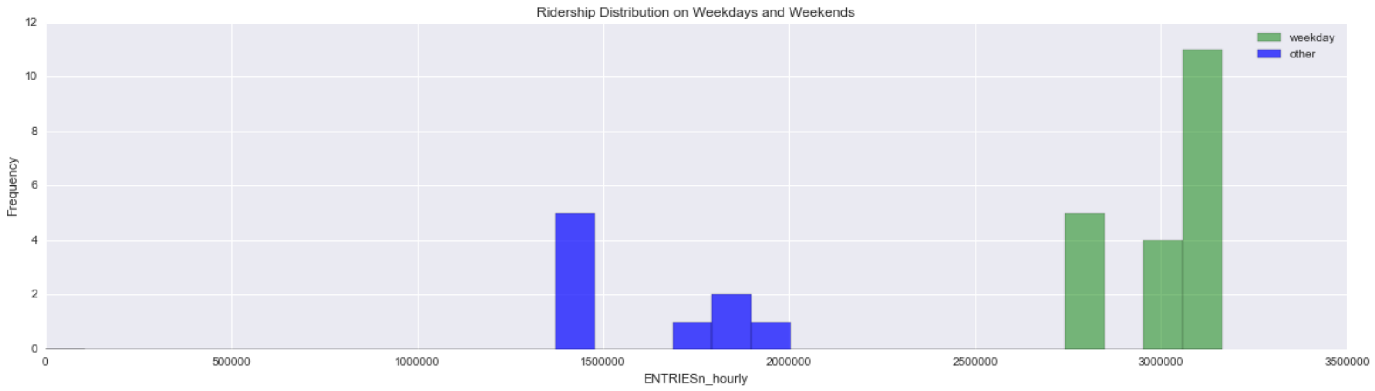


Figure 4. Comparison of ridership data for weekdays to ridership on weekends and holidays.

In [64]:
```
import calendar
import time
from calendar import timegm
from datetime import datetime
from matplotlib import dates

weekday_number = weekdays.apply(lambda r: int(time.mktime(time.strptime(r['DATEn'],
'%m/%d/%Y'))), axis=1)
nonweekday_number = non_weekdays.apply(lambda r: int(time.mktime(time.strptime(r['D
ATEn'], '%m/%d/%Y'))), axis=1)

# convert epoch to matplotlib float format
weekday_formatted = dates.date2num(map(datetime.fromtimestamp, weekday_number))
nonweekday_formatted = dates.date2num(map(datetime.fromtimestamp, nonweekday_numbe
r))

# matplotlib date format object
hfmt = dates.DateFormatter('%m/%d %H:%M')

%pylab inline
fig = plt.figure(figsize=[20,5])
ax = fig.add_subplot(111)
#ax.vlines(fds, y2, y1)
ax.plot(weekday_formatted, weekdays.total,marker='o', linestyle='None', color='r')
ax.plot(nonweekday_formatted, non_weekdays.total,marker='o', linestyle='None', colo
r='b')
ax.xaxis.set_major_locator(dates.MinuteLocator())
ax.xaxis.set_major_formatter(hfmt)
ax.set_ylim(bottom = 0)
plt.xticks(np.arange(min(weekday_formatted), max(weekday_formatted)+1, 10.0), fonts
ize=16)
plt.yticks(np.arange(0, 4000000, 1000000.0), fontsize=16)
#plt.xticks(rotation='vertical')
#plt.subplots_adjust(bottom=.3)
plt.xlabel('Time (date and hour:minute)', fontsize=16)
plt.ylabel('Total Riders', fontsize=16)
plt.show()
```

```
Populating the interactive namespace from numpy and matplotlib

WARNING: pylab import has clobbered these variables: ['datetime']
`%matplotlib` prevents importing * from pylab and numpy
```

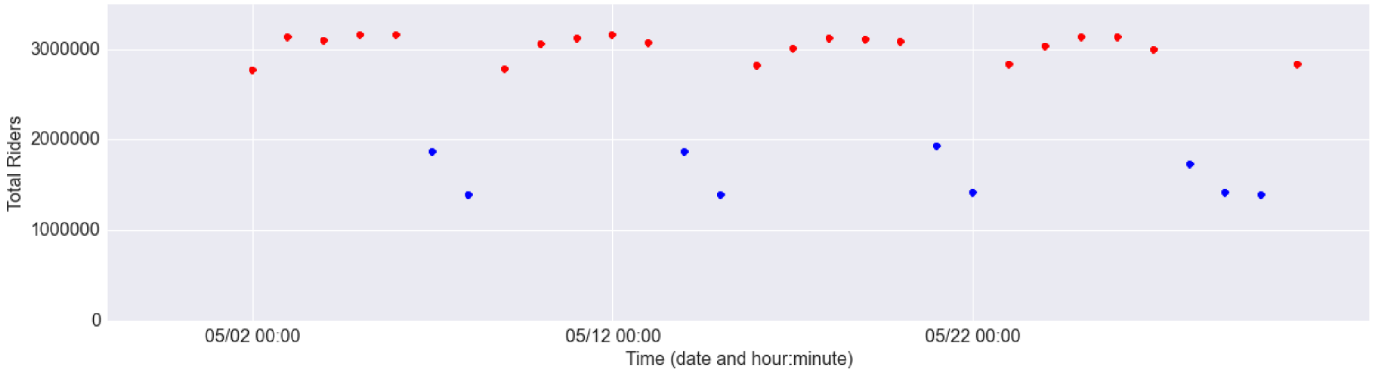


Figure 5. Total daily ridership, displayed as a timeseries. The distribution of our data appears to differ strongly between weekdays and non-weekdays. (weekends and holidays).

Section 4: Conclusions

The analysis presented in this report does not provide any evidence to support the claim that subway ridership changes during rainy weather. Further, an effective model of subway ridership was presented that did not require any weather variables at all.

1.3. Results of statistical test

In [44]:
```
import scipy
import scipy.stats

q = """
    SELECT rain, DATEn, SUM(cast (ENTRIESn_hourly as integer)) as total
    FROM improved_df
    GROUP BY DATEn
    """
#    WHERE day_week IN ('0','1','2','3','4')

selection = pandasql.sqldf(q, locals())

mean_rain = np.mean(selection[selection.rain == 1]['total'])
mean_norain = np.mean(selection[selection.rain == 0]['total'])

print "Mean daily turnstile entries, with rain: %.2f\nMean daily entries, without r
ain: %.2f" % (mean_rain, mean_norain)



[U, p] = scipy.stats.mannwhitneyu(selection[selection.rain == 1]['total'],
                                  selection[selection.rain == 0]['total'])

crit_value = 0.05
print "P value is: %.2f" % (p)
# two-tailed test
if (p * 2) < crit_value:
    print 'Reject the null hypothesis'
else:
    print 'Fail to reject null hypothesis'
```

```
Mean daily turnstile entries, with rain: 2562036.29
Mean daily entries, without rain: 2611790.04
P value is: 0.33
Fail to reject null hypothesis
```

2.2-2.3. Feature selection

In [45]:
```
import calendar
import time
from calendar import timegm
from datetime import datetime

improved_df['dateTime'] = improved_df.apply(lambda r: pd.datetime.strptime(r['datet
ime'], '%m/%d/%Y %H:%M'), axis=1)
improved_df['hour_week'] = improved_df.day_week * 24. + improved_df.hour
improved_df['dateTimeString'] = improved_df.apply(lambda r: int(time.mktime(time.st
rptime(r['datetime'], '%m/%d/%Y %H:%M'))), axis=1)

features = improved_df[['hour', 'lag24']]
turnstile_master = improved_df[['DATEn', 'hour', 'lag24']]

# Add UNIT to features using dummy variables
dummy_units = pd.get_dummies(improved_df['UNIT'], prefix='unit')
features = features.join(dummy_units)


# use lambda function to get weekday from 'DATEn' field
f = lambda x: datetime.strptime(x, "%m/%d/%Y").weekday()

dummy_units = pd.get_dummies(turnstile_master['DATEn'].apply(f), prefix='weekday')
features = features.join(dummy_units)
```

In [46]:
```
# Values
values = improved_df['ENTRIESn_hourly']
m = len(values)

#print features.iloc[[0]]

features, mu, sigma = normalize_features(features)
features['ones'] = np.ones(m) # Add a column of 1s (y intercept)

# Convert features and values to numpy arrays
features_array = np.array(features)
values_array = np.array(values)

#print features_array[[0], :]
```

2.4-2.5. Model coeeficients and the coefficient of determination

In [47]:
```
# Set values for alpha, number of iterations.
alpha = 0.1 # please feel free to change this value
num_iterations = 200 # please feel free to change this value


# Initialize theta, perform gradient descent
theta_gradient_descent = np.zeros(len(features.columns))
theta_gradient_descent, cost_history = gradient_descent(features_array,
                                                        values_array,
                                                        theta_gradient_descent,
                                                        alpha,
                                                        num_iterations)

print "The weights for the 'hour-of-the-day' and 'turnstile entries 24-hours-ago' f
eatures are:"
print theta_gradient_descent[[0,1]]


predictions = np.dot(features_array, theta_gradient_descent)

SST = ((improved_df['ENTRIESn_hourly'] - np.mean(improved_df['ENTRIESn_hourl
y']))**2).sum()
SSReg = ((predictions - np.mean(improved_df['ENTRIESn_hourly']))**2).sum()
r_squared = SSReg / SST

print "The R^2 of the model is %.2f" % (r_squared)
```

```
The weights for the 'hour-of-the-day' and 'turnstile entries 24-hours-ago' features
are:
[  262.06980169  2173.095726  ]
The R^2 of the model is 0.76
```

2.6. Interpretation of results

I generated some timeseries plots of the actual data to the predicted values so I could visualize the model fit and determine if there were any problems with the fit. I was very happy with the result.

In [48]:
```
%pylab inline
fig = plt.figure(figsize=[20,5])
ax = fig.add_subplot(111)
ax.plot(improved_df.query('UNIT == "R084"').dateTimeString, improved_df.query('UNIT
== "R084"').ENTRIESn_hourly, 'b-', lw=0.5, label='Observed')
ax.plot(improved_df.query('UNIT == "R084"').dateTimeString, predictions[(improved_d
f.UNIT == "R084").values], 'ko-', label='Predicted')
ax.legend()
ax.set_ylabel('Entries')
ax.set_xlabel('Time Index')
ax.set_title('Predicted Ridership Vs. Observed, Turnstile R084')
plt.show()
```

```
Populating the interactive namespace from numpy and matplotlib

WARNING: pylab import has clobbered these variables: ['f', 'datetime']

`%matplotlib` prevents importing * from pylab and numpy
```

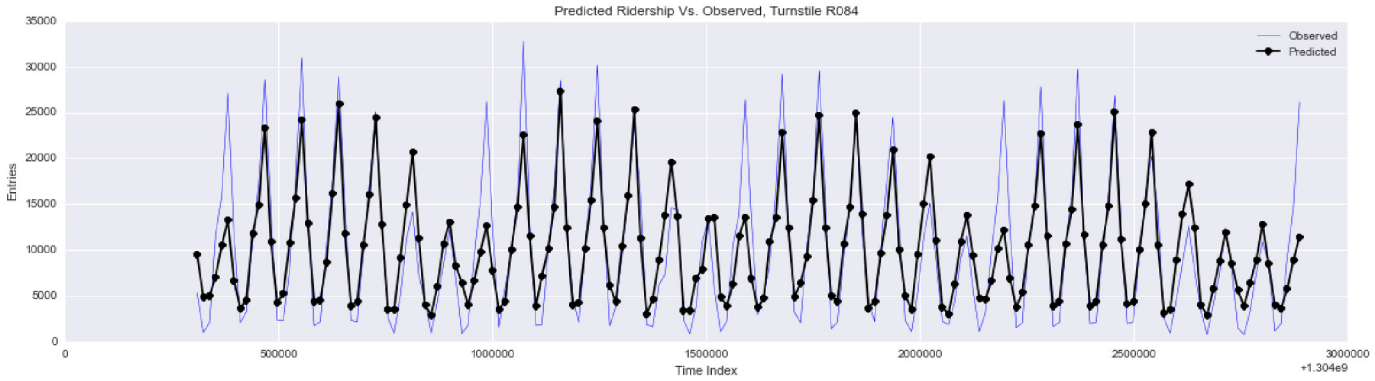


Figure 6. Predicted ridership versus observed for turnstile R084.

The fit for turnstile R084 was excellent. Here is a not-so-excellent fit. In particular we predict negative ridership (!). Given more time, I would look at ways to constrain ridership predictions to positive values.

In [49]:
```
%pylab inline
fig = plt.figure(figsize=[20,5])
ax = fig.add_subplot(111)
ax.plot(improved_df.query('UNIT == "R263"').dateTimeString, improved_df.query('UNIT
== "R263"').ENTRIESn_hourly, 'b-', lw=0.5, label='Observed')
ax.plot(improved_df.query('UNIT == "R263"').dateTimeString, predictions[(improved_d
f.UNIT == "R263").values], 'ko-', label='Predicted')
ax.legend()
ax.set_ylabel('Entries')
ax.set_xlabel('Time Index')
ax.set_title('Predicted Ridership Vs. Observed, Turnstile R263')
plt.show()
```

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

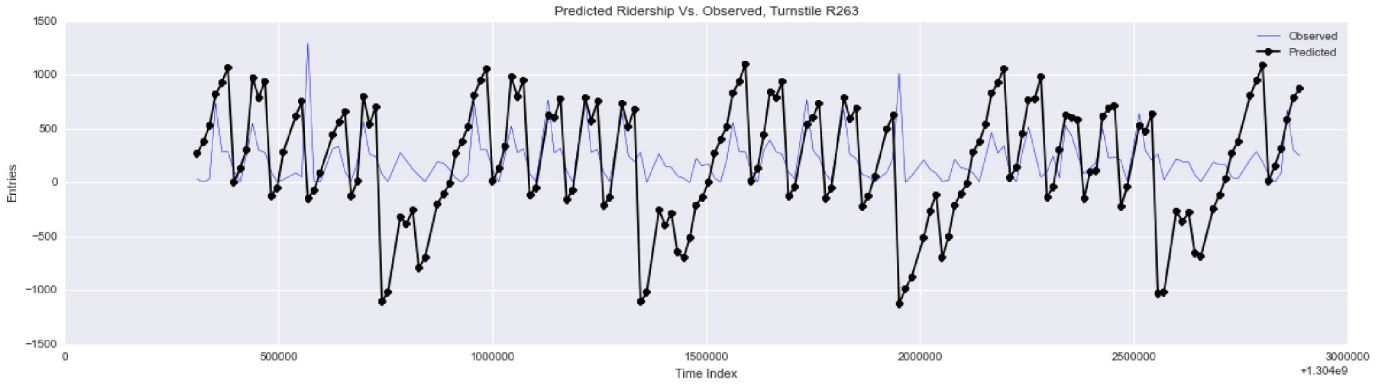


Figure 7. Predicted ridership versus observed for turnstile R263.

2.7. Fit diagnostics

Visualizing the residuals of the fit, and predictions vs. actual values allow us to judge quality of the regression model. In this case, the distribution of the residuals appears to be heavy-tailed (ref http://socserv.socsci.mcmaster.ca/jfox/Courses/Brazil-2009/slides-handout.pdf), so improvements to the regression model or data trasformations may improve the fit further.

In [50]:
```
%pylab inline

plt.figure(figsize=[12,12])
ax1 = plt.subplot(221)
ax1.set_xlim([-3,3])
#ax1.set_ylim([-15000,150000])
c = scipy.stats.probplot(improved_df.ENTRIESn_hourly - predictions, dist="norm", pl
ot=pylab)
ax2 = plt.subplot(222)
axes = plt.gca()
axes.set_xlim([-5000,5000])
d = plt.hist(improved_df.ENTRIESn_hourly - predictions, bins=200)
ax2.set_xlabel('Residuals', fontsize=14)
ax2.set_ylabel('Counts', fontsize=14)
ax2.set_title('Residual Distribution', fontsize=14)
ax3 = plt.subplot(223)
plt.plot(predictions, improved_df.ENTRIESn_hourly, 'b.', ms=4)
ax3.set_xlabel('Predicted Ridership', fontsize=14)
ax3.set_ylabel('Actual Ridership', fontsize=14)
ax3.set_title('Actual Ridership vs Predicted', fontsize=14)
ax4 = plt.subplot(224)
```

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

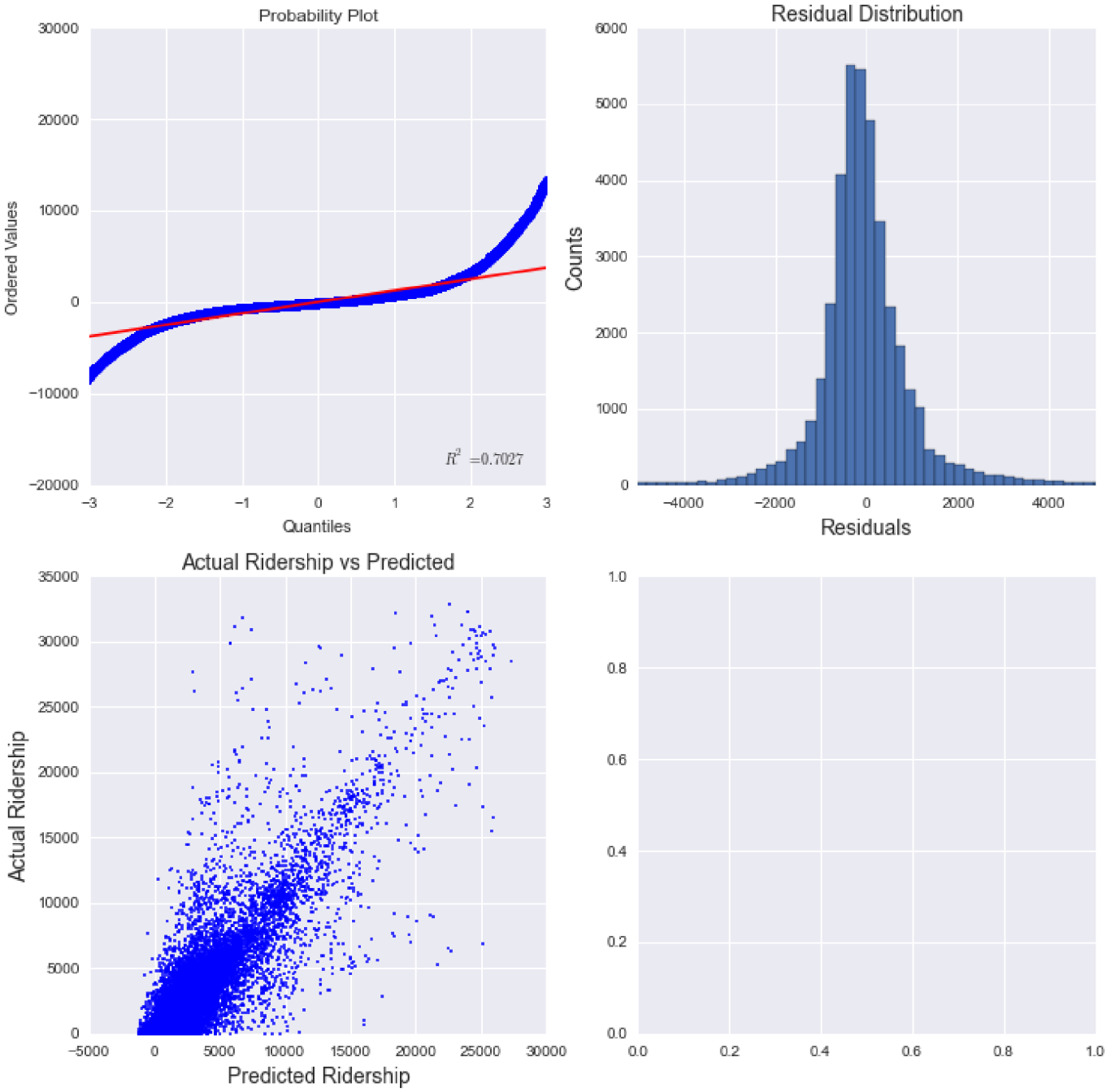


Figure 8. Fit diagnostics.

In [51]:
```
def normalize_features(df):
    """
    Normalize the features in the data set.
    """
    mu = df.mean()
    sigma = df.std()

    if (sigma == 0).any():
        raise Exception("One or more features had the same value for all samples, a
nd thus could " + \
                        "not be normalized. Please do not include features with on
ly a single value " + \
                        "in your model.")
    df_normalized = (df - df.mean()) / df.std()

    return df_normalized, mu, sigma

def gradient_descent(features, values, theta, alpha, num_iterations):
    """
    Perform gradient descent given a data set with an arbitrary number of features.

    This can be the same gradient descent code as in the lesson #3 exercises,
    but feel free to implement your own.
    """

    m = len(values)
    cost_history = []
    for j in range(num_iterations):
        prediction = np.dot(features, theta)
        theta = theta - alpha/m*np.dot((prediction - values), features)
        cost_history.append(compute_cost(features, values, theta))

    return theta, pd.Series(cost_history) # leave this line for the grader

def compute_cost(features, values, theta):
    """
    Compute the cost function given a set of features / values,
    and the values for our thetas.

    This can be the same code as the compute_cost function in the lesson #3 exercis
es,
    but feel free to implement your own.
    """

    m = len(values)
    sum_of_square_errors = np.square(np.dot(features, theta) - values).sum()
    cost = sum_of_square_errors / (2*m)

    return cost
```