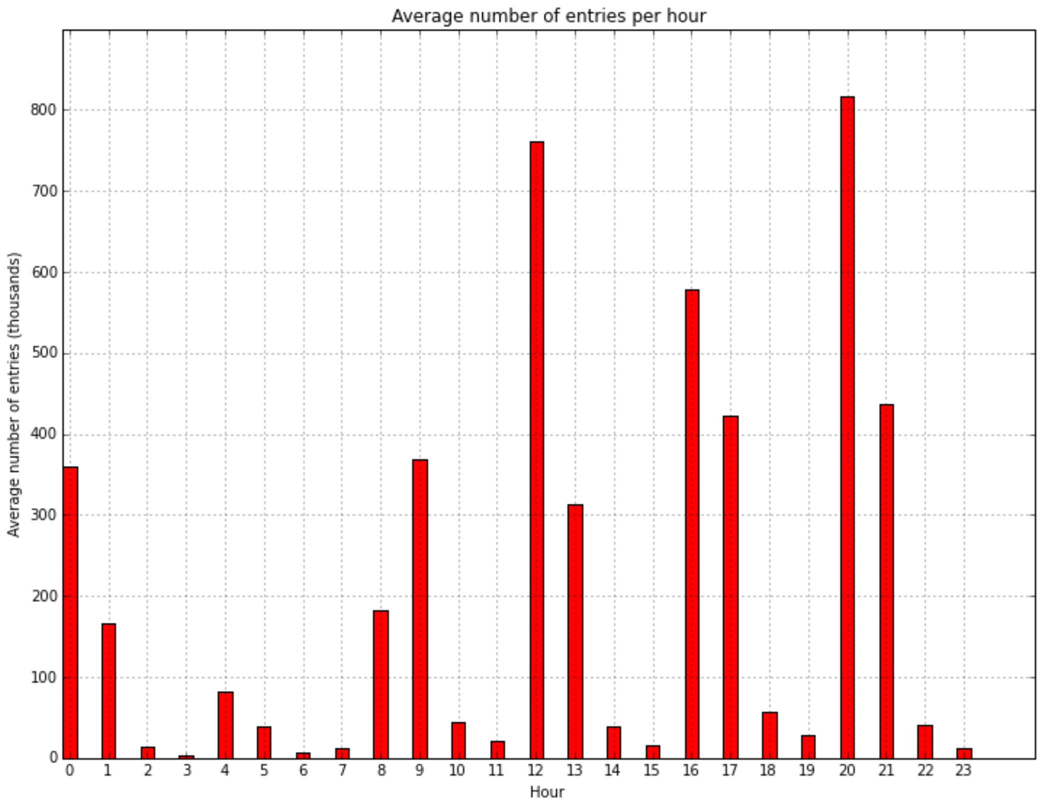
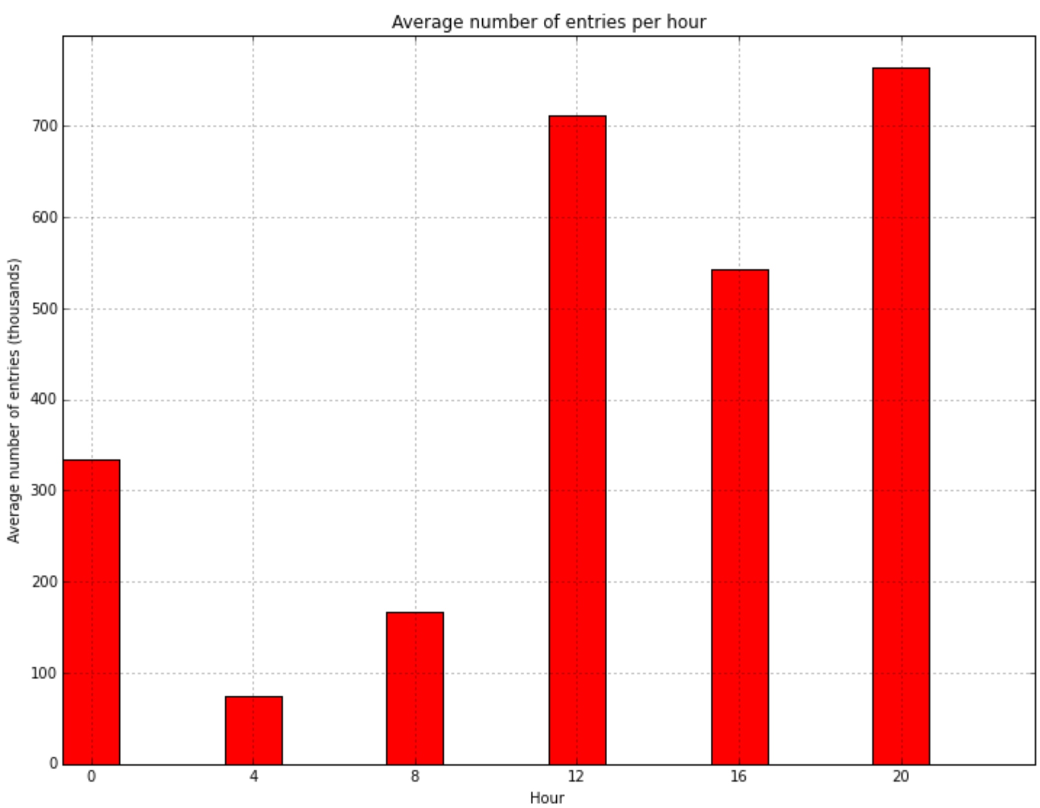
In [48]:
```
import numpy as np
import pandas as pd
import scipy
import scipy.stats
from ggplot import *
import matplotlib.pyplot as plt
from matplotlib.ticker import NullFormatter
from mpl_toolkits.axes_grid1 import make_axes_locatable
from mpl_toolkits.axes_grid1.axes_divider import make_axes_area_auto_adju
stable

%matplotlib inline

#original version of the combined turnstile and weather data
turnstile_weather = pd.read_csv('/Users/rclement2/Projects/workspace/udac
ity/data_science1/turnstile_data_master_with_weather.csv')

#version 2 of the combined turnstile and weather data
turnstile_weather_v2 = pd.read_csv('/Users/rclement2/Projects/workspace/u
dacity/data_science1/improved-dataset/turnstile_weather_v2.csv')
```

In [28]:
```
def time_bar(df):
    """
    find the average number of entries per hour across all units for the
time period of
    data provided. In the test case this is data for the month of May 201
1.

    get the data grouped by date and hour and sum up the results across a
ll units
    and then calculate the mean
    """
    df = df.set_index([pd.to_datetime(df['DATEn'] + ' ' + df['TIMEn'])])
    df_grouped = df.groupby([df.index.date, df.index.hour])[['ENTRIESn_ho
urly', 'EXITSn_hourly']].aggregate(sum)
    df_grouped = df_grouped.reset_index()
    df_grouped.columns = ['Date', 'Hour', 'ENTRIESn_hourly', 'EXITSn_hour
ly']
    df_grouped = df_grouped.groupby('Hour').mean()
    #print df_grouped.head(24)

    #setup the plot
    fig, ax = plt.subplots(figsize=(12,9))
    plt.grid(True)
    ind = np.arange(len(df_grouped))  # the x locations for the groups
    width = 0.35 # the width of the bars

    #create bar plot for average # entries per hour
    rects1 = ax.bar(ind, df_grouped['ENTRIESn_hourly'], width, color='r')

    #rects2 = ax.bar(ind+width, df_grouped['EXITSn_hourly'], width, color
='y')

    ax.set_xlabel('Hour')
    ax.set_ylabel('Average number of entries (thousands)')
    ax.set_title('Average number of entries per hour')
    start, end = ax.get_xlim()
    plt.xticks(ind+width/2., df_grouped.index)

    start, end = ax.get_ylim()
    ax.set_yticklabels(range(0, int(end/1000.0), 100))
    #ax.legend( (rects1[0], rects2[0]), ('Entries', 'Exits') )
    #ax.axis([0, 23.75, 0, 25000000])
    #plt.legend()

    return plt
```

In [29]:
```
print time_bar(turnstile_weather) #original data
print time_bar(turnstile_weather_v2) #version 2 of the data
```

```
<module 'matplotlib.pyplot' from '/Users/rclement2/anaconda/lib/python2.7/site-packages/matplotlib/pyplot.pyc'>
<module 'matplotlib.pyplot' from '/Users/rclement2/anaconda/lib/python2.7/site-packages/matplotlib/pyplot.pyc'>
```

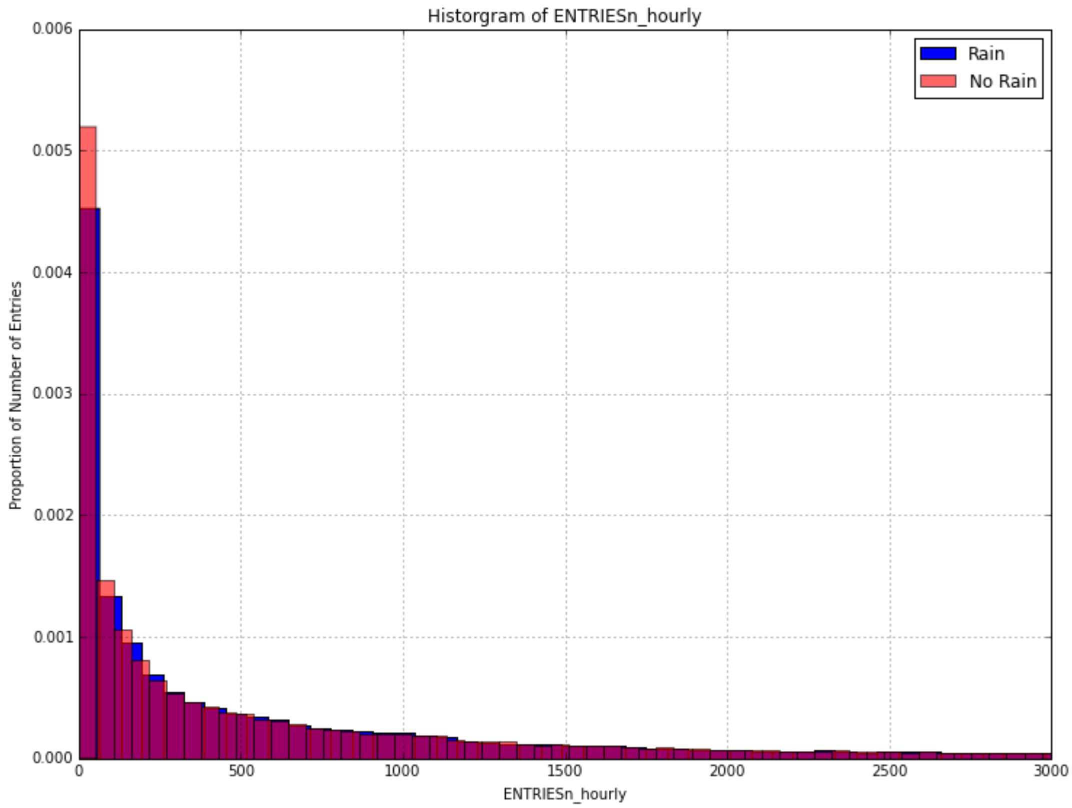


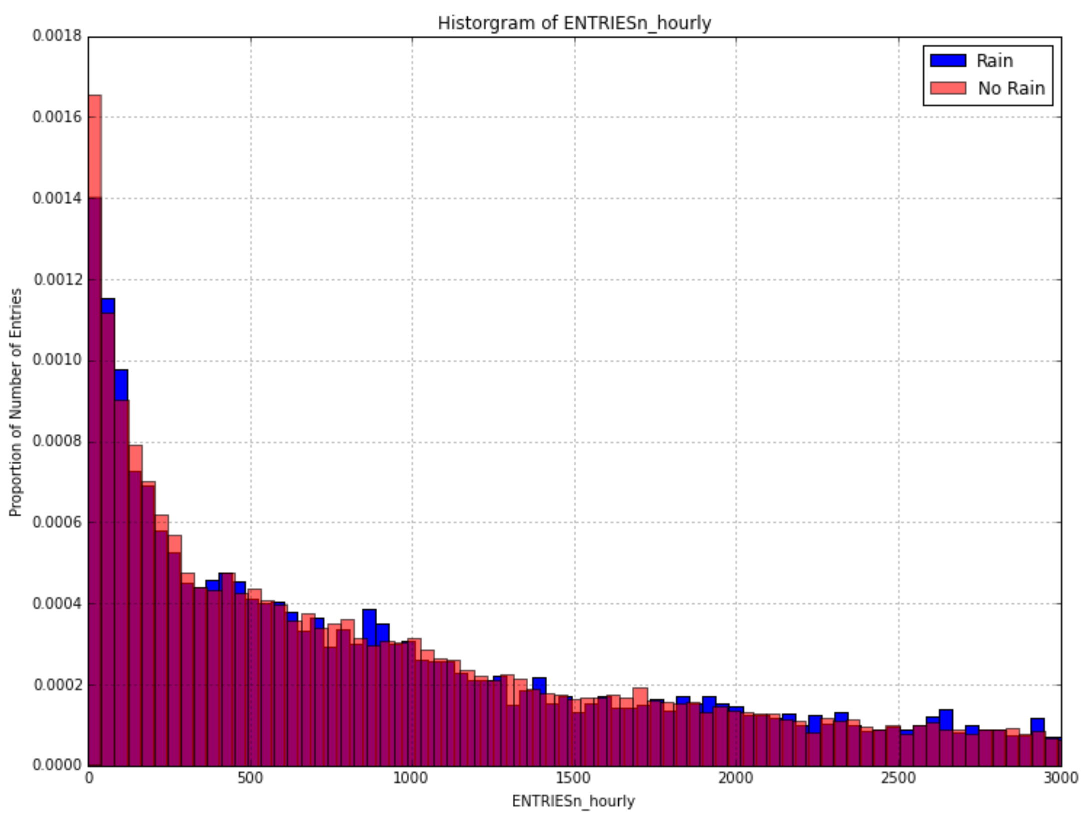


In [105]:
```
def entries_histogram(df):
    """
    compare the distribution of the number of entries for the two
    populations (raining versus not raining) via histograms. Normalize th
e histograms for an easier
    comparison of the two sets of results
    """

    df_rain = df[df['rain'] == 1] #dataframe contains the entries when it
 is raining
    df_norain = df[df['rain'] == 0] #dataframe contains the entries when
it is not raining

    #print df['ENTRIESn_hourly']

    #setup the histogram plots
    fig = plt.figure(figsize=(12,9))
    plt.title('Historgram of ENTRIESn_hourly')
    plt.ylabel('Proportion of Number of Entries')
    plt.xlabel('ENTRIESn_hourly')
    plt.grid(True)

    #plot a historgram for hourly entries when it is raining.
    hist1 = df_rain['ENTRIESn_hourly'].hist(bins=800, histtype='bar', col
or='b', normed=1, label='Rain')

    #plot a historgram for hourly entries when it is not raining
    hist2 = df_norain['ENTRIESn_hourly'].hist(alpha=0.6, bins=800, histty
pe='bar', color='r', normed=1, label='No Rain')

    #plt.axvline(np.mean(df_norain['ENTRIESn_hourly']), color='r', linest
yle='dashed', linewidth=2) # no rain mean
    #plt.axvline(np.mean(df_rain['ENTRIESn_hourly']), color='b', linestyl
e='dashed', linewidth=2) # rain mean
    #plt.axvline(np.median(df_norain['ENTRIESn_hourly']), color='r', line
style='dotted', linewidth=2)
    #plt.axvline(np.median(df_rain['ENTRIESn_hourly']), color='b', linest
yle='dotted', linewidth=2)
    #plt.text(845, .0075, r'No Rain $\mu=1,090$')
    #plt.text(1150, .0075, r'Rain $\mu=1,105$')


    plt.xlim(0, 3000) #focus the plot away from the long tail to provide
better visualization
    plt.legend()
    return plt
```

In [106]:
```
print entries_histogram(turnstile_weather) #original data
print entries_histogram(turnstile_weather_v2) #version 2 of the data
```

```
<module 'matplotlib.pyplot' from '/Users/rclement2/anaconda/lib/python2.7/
site-packages/matplotlib/pyplot.pyc'>
<module 'matplotlib.pyplot' from '/Users/rclement2/anaconda/lib/python2.7/
site-packages/matplotlib/pyplot.pyc'>
```

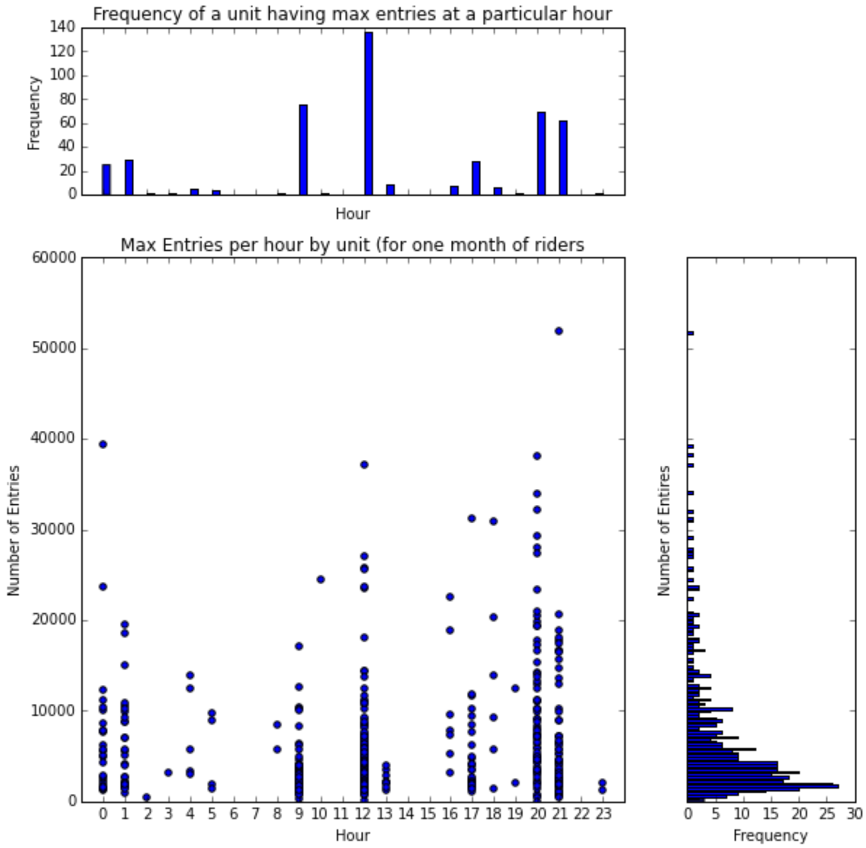


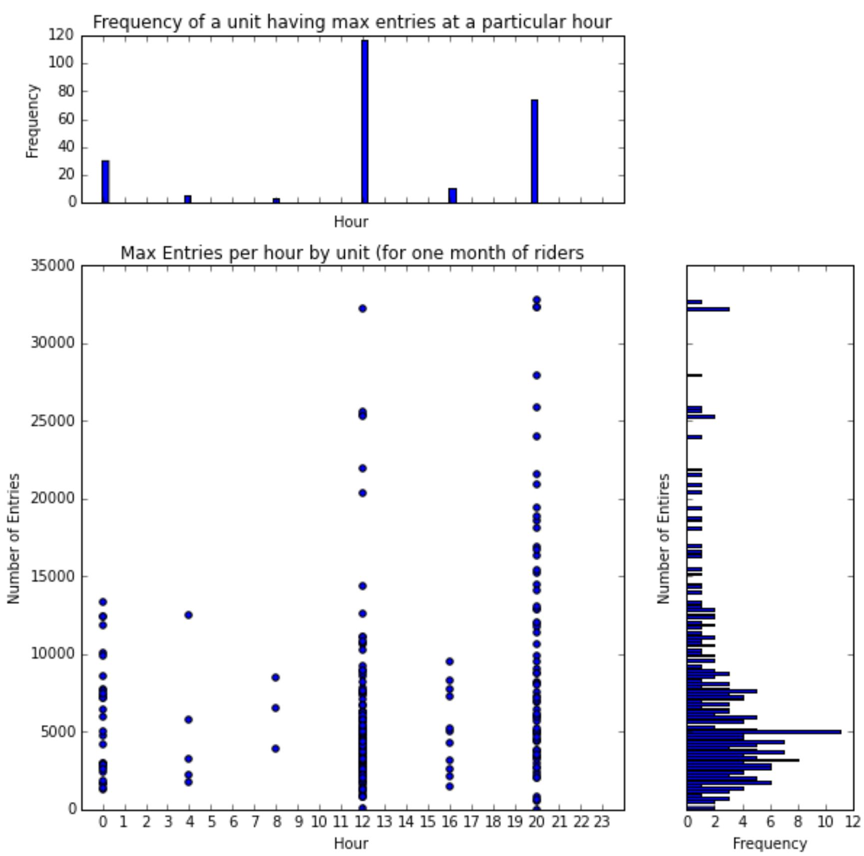


In [120]:

```
def time_unit_bar(df):

    #the data. Find entry maximums by unit and make sure to track the hou
r in which it occurred
    df = df.set_index([pd.to_datetime(df['DATEn'] + ' ' + df['TIMEn'])])
    df['Hour'] = df.index.hour
    df = df[['UNIT', 'Hour', 'ENTRIESn_hourly']]
    #print df.head()

    idx = df.groupby(['UNIT'])['ENTRIESn_hourly'].transform(max) == df['E
NTRIESn_hourly']
    df = df[idx].sort('UNIT') #finds the maximum entries but still contai
ns duplicates, i.e. 0's
    df = df.groupby(['UNIT']).aggregate(max) #finds the maximum entries p
er unit
    #print df.info()

    #plotting setup
    nullfmt   = NullFormatter()         # no labels

    #definitions for the axes
    left, width = 0.1, 0.65
    bottom, height = 0.1, 0.65
    bottom_h = left_h = left+width+0.075

    rect_scatter = [left, bottom, width, height]
    rect_histx = [left, bottom_h, width, 0.2]
    rect_histy = [left_h, bottom, 0.2, height]

    #start with a rectangular Figure
    plt.figure(1, figsize=(8,8))

    axScatter = plt.axes(rect_scatter)
    axHistx = plt.axes(rect_histx)
    axHisty = plt.axes(rect_histy)

    #no labels
    axHistx.xaxis.set_major_formatter(nullfmt)
    axHisty.yaxis.set_major_formatter(nullfmt)

    #the scatter plot:
    #the scatter plot shows the distribution of the maximum number of ent
ries per unit by hour
    #its the largest of the 3 plots
    axScatter.scatter(df['Hour'], df['ENTRIESn_hourly'])
    axScatter.set_xlabel('Hour')
    axScatter.set_ylabel('Number of Entries')
    axScatter.set_title('Max Entries per hour by unit (for one month of r
iders')

    #now determine nice limits by hand:
    N = 24
    ind = np.arange(N)  # the x locations for the groups
    width = 0.25        # the width of the bars

    axScatter.set_xlim( (-1, 24) )
    axScatter.set_xticks(range(0, 24))
    Start, End = axScatter.get_ylim()
    axScatter.set_ylim( (0, End) )

    #the histogram plots:
    #the histograms are looking at a few things. The upper plot shows the
 frequency at which units
    #have maximum entries per hour. The plot to the side shows the freque
ncy at which
    #different maximum entry numbers occur.
    axHistx.hist(df['Hour'], bins=69, align='mid') #upper histogram
    axHistx.set_xlabel('Hour')
    axHistx.set_ylabel('Frequency')
    axHistx.set_title('Frequency of a unit having max entries at a partic
ular hour')
    axHisty.hist(df['ENTRIESn_hourly'], bins=200, orientation='horizontal
') #right histogram
    axHisty.set_ylabel('Number of Entires')
    axHisty.set_xlabel('Frequency')

    axHistx.set_xlim( (-1, 24) )
    axHistx.set_xticks(range(0, 24))
    plt.show()
    #return df
```

In [121]:
```
time_unit_bar(turnstile_weather) #original data
time_unit_bar(turnstile_weather_v2) #version 2 of the data
```

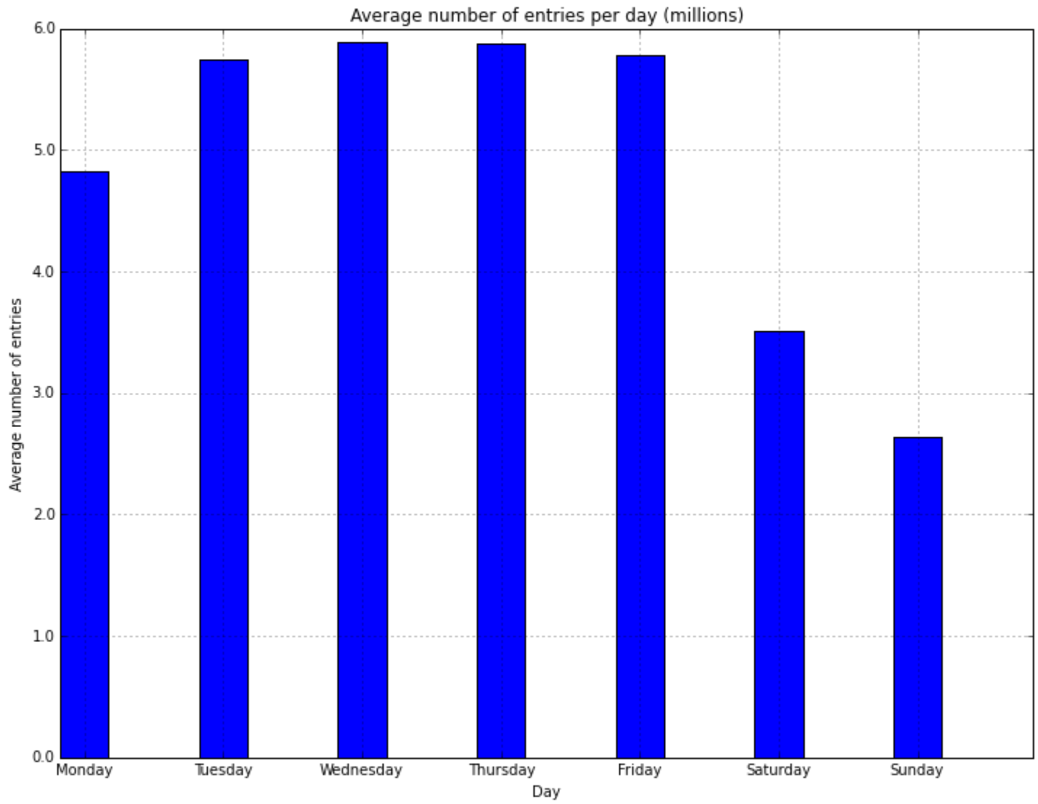


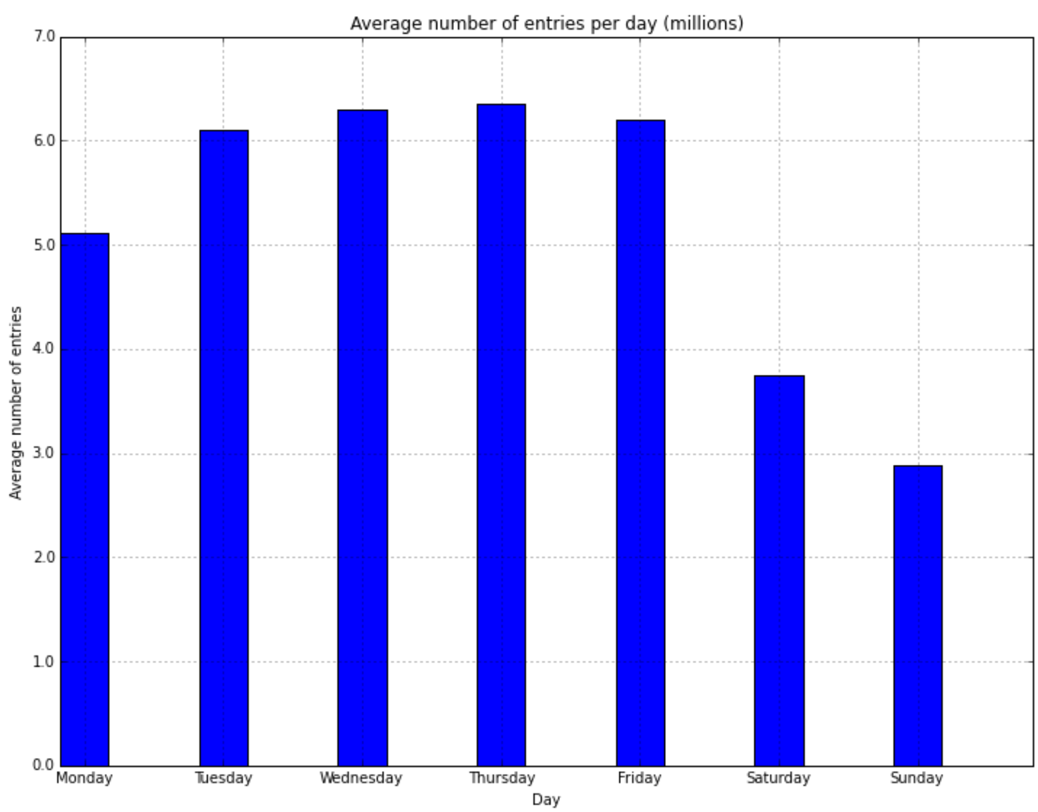


In [46]:
```
def entries_by_day_histogram(df):

    """
    developed to take a look at the average number of entries per day
    taking all units and a months worth of data (May, 2011)

    the data aggregated by day of the week
    """
    df = df.set_index([pd.to_datetime(df['DATEn'] + ' ' + df['TIMEn'])])
    df_grouped = df.groupby(df.index.date)['ENTRIESn_hourly', 'EXITSn_hou
rly'].aggregate(sum)

    df_grouped = df_grouped.set_index([pd.to_datetime(df_grouped.index)])
    df_grouped['weekday'] = df_grouped.index.weekday
    df_grouped = df_grouped.groupby('weekday').mean()

    #print df_grouped.head(20)
    #print df_grouped.info()

    #setup the plot
    N = 7
    ind = np.arange(N)  # the x locations for the groups
    width = 0.35       # the width of the bars

    fig, ax = plt.subplots(figsize=(12,9))
    plt.grid(True)
    rects1 = ax.bar(ind, df_grouped['ENTRIESn_hourly'], width, color='b')
 #bar plot
    #rects2 = ax.bar(ind+width, df_grouped['EXITSn_hourly'], width, color
='y')

    ax.set_xlabel('Day')
    ax.set_ylabel('Average number of entries')
    ax.set_title('Average number of entries per day (millions)')
    start, end = ax.get_ylim()
    plt.xticks(ind+width/2., ('Monday', 'Tuesday', 'Wednesday', 'Thursday
', 'Friday', 'Saturday', 'Sunday'))
    ax.set_ylim(0, int(end))
    ax.set_yticklabels(np.arange(end))
    #ax.legend( (rects1[0], rects2[0]), ('Entries', 'Exits') )
    #plt.legend()

    return plt
```

In [47]:
```
print entries_by_day_histogram(turnstile_weather) #original data
print entries_by_day_histogram(turnstile_weather_v2) #version 2 of the da
ta
```

```
<module 'matplotlib.pyplot' from '/Users/rclement2/anaconda/lib/python2.7/
site-packages/matplotlib/pyplot.pyc'>
<module 'matplotlib.pyplot' from '/Users/rclement2/anaconda/lib/python2.7/
site-packages/matplotlib/pyplot.pyc'>
```

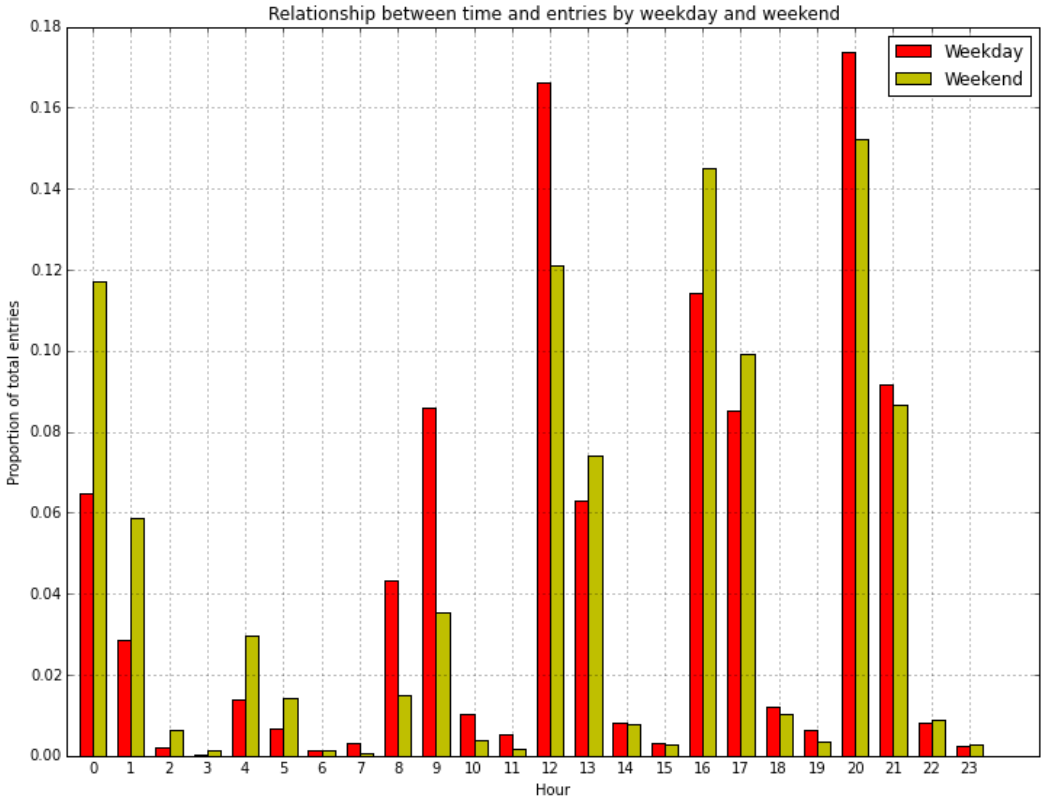


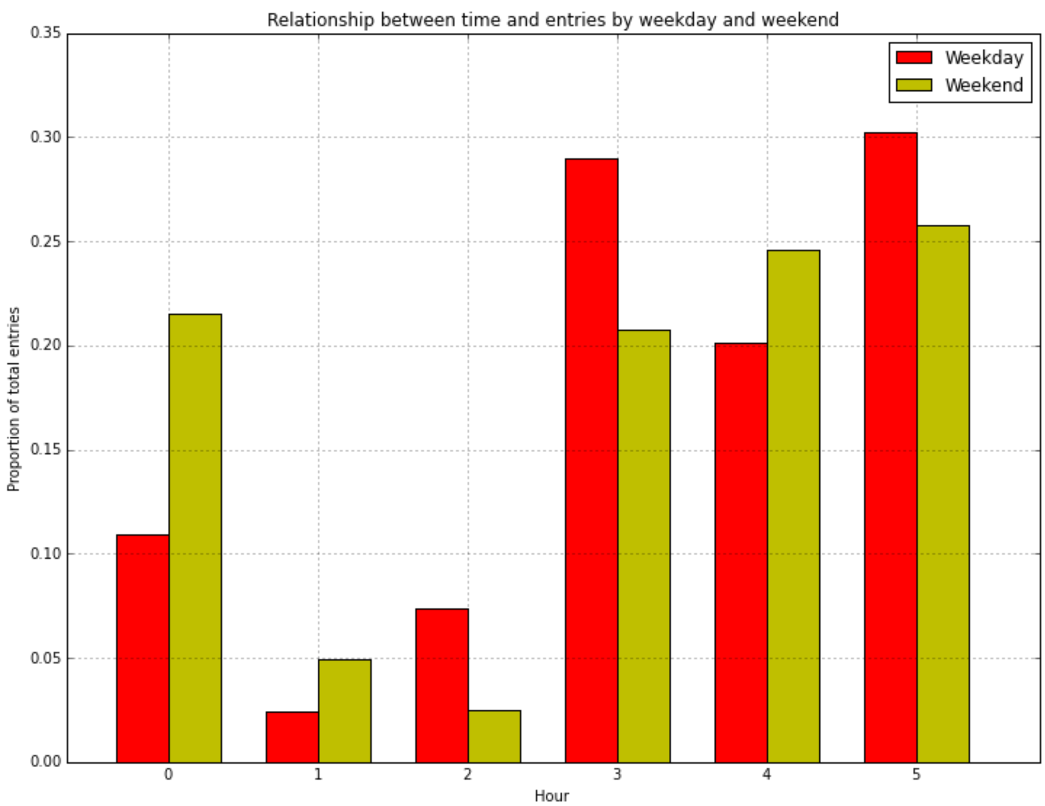


In [162]:
```
def time_bar_weekday(df):

    """
    developed to take a look at the average number of entries by hour for
 weekdays and weekends.
    plots are based on looking at data from the month of May 2011 and sho
w proportions rather than
    absolute numbers
    """

    #the data
    df = df.set_index([pd.to_datetime(df['DATEn'] + ' ' + df['TIMEn'])])
    df_grouped = df.groupby([df.index.date, df.index.hour, df.index.weekd
ay]).aggregate(sum)
    df_grouped.index.names = ['datei', 'houri', 'dayi']
    df_grouped = df_grouped.reset_index()

    #print df_grouped.info()

    #classify dates as either weekdays or weekend
    df_weekday = df_grouped[df_grouped['dayi'] < 5][['houri', 'dayi','ENT
RIESn_hourly', 'EXITSn_hourly']]
    df_weekend = df_grouped[df_grouped['dayi'] >= 5][['houri', 'dayi','EN
TRIESn_hourly', 'EXITSn_hourly']]

    #take the mean for each hour by either weekday or weekend
    df_weekday = df_weekday.groupby('houri')[['ENTRIESn_hourly', 'EXITSn_
hourly']].mean()
    df_weekend = df_weekend.groupby('houri')[['ENTRIESn_hourly', 'EXITSn_
hourly']].mean()
    df_weekday_count = df_weekday.sum() #calculate the totals for all wee
kday hours
    df_weekend_count = df_weekend.sum() #calculate the totals for all wee
kend hours

    #print df_weekday.head(30)
    #print df_weekend.head(30)
    #print df_weekday_count
    #print df_weekend_count

    #setup the plots
    fig, ax = plt.subplots(figsize=(12,9))
    plt.grid(True)
    width = 0.35 #the width of the bars
    ind = np.arange(len(df_weekday['ENTRIESn_hourly']))  #the x locations
 for the groups

    #create the bar plots. In this case the data is normalized to create
proportions in order
    #to facilitate comparisons
    rects1 = ax.bar(ind-.5*width, df_weekday['ENTRIESn_hourly']/df_weekda
y_count['ENTRIESn_hourly'], width, color='r')
    rects2 = ax.bar(ind+.5*width, df_weekend['ENTRIESn_hourly']/df_weeken
d_count['ENTRIESn_hourly'], width, color='y')

    ax.set_xlabel('Hour')
    ax.set_ylabel('Proportion of total entries')
    ax.set_title('Relationship between time and entries by weekday and we
ekend')
    start, end = ax.get_xlim()
    ax.set_xlim(-.5, end)
    plt.xticks(ind+width/2., range(0, int(end), 1))
    ax.legend( (rects1[0], rects2[0]), ('Weekday', 'Weekend') )
    start, end = ax.get_ylim()
    ax.set_ylim(0, end)
    plt.legend()

    #digressed into figuring out how to do 2 plots next to each other
    #fig = plt.figure(3)
    #ax1 = plt.axes([0,0,1,1])
    #divider = make_axes_locatable(ax1)

    #ax2 = divider.new_horizontal(size="100%", pad=0.3)
    #fig = ax1.get_figure()
    #ax2.tick_params(labelleft="off")
    #fig.add_axes(ax2)

    #ax1.set_title("Title")
    #ax1.set_yticks([0.5])
    #ax1.set_yticklabels(["very long label"])
    #ax1.set_xlabel("X - Label")
```

In [163]:
```
time_bar_weekday(turnstile_weather) #original data
time_bar_weekday(turnstile_weather_v2) #version 2 of the data
```

Out[163]:
```
<module 'matplotlib.pyplot' from '/Users/rclement2/anaconda/lib/python2.7
/site-packages/matplotlib/pyplot.pyc'>
```



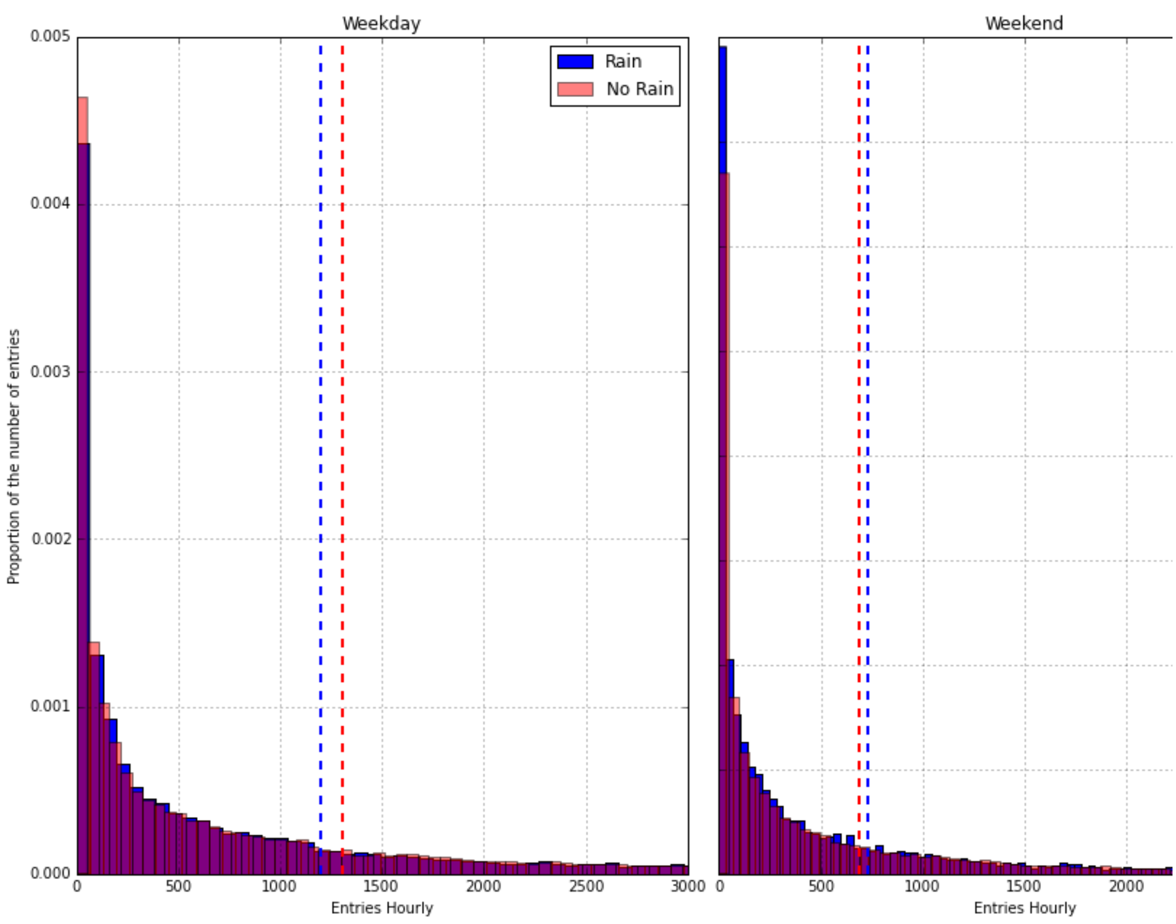


In [172]:
```
def weekday_histogram(df):

    """
    compare the distribution of the number of entries for the two
    populations (raining versus not raining) via histograms. Do so by fur
ther dividing the data into info
    for weekdays and info for weekends. Normalize the histograms for an e
asier
    comparison of the two sets of results
    """

    #the data
    df = df.set_index([pd.to_datetime(df['DATEn'] + ' ' + df['TIMEn'])])
    df['weekday'] = df.index.weekday
    #print df.info()

    df_weekday = df[df['weekday'] < 5][['ENTRIESn_hourly', 'EXITSn_hourly
', 'weekday', 'rain']]
    df_weekend = df[df['weekday'] >= 5][['ENTRIESn_hourly', 'EXITSn_hourl
y', 'weekday', 'rain']]
    #print df_weekday.info()
    #print df_weekend.info()

    #identification of rain and no rain datasets for total, weekday and w
eekend situations
    df_rain_weekday = df_weekday[df_weekday['rain'] == 1]
    df_rain_weekend = df_weekend[df_weekend['rain'] == 1]
    df_norain_weekday = df_weekday[df_weekday['rain'] == 0]
    df_norain_weekend = df_weekend[df_weekend['rain'] == 0]

    df_rain = df[df['rain'] == 1]
    df_norain = df[df['rain'] == 0]

    #print df_rain_weekday.head()
    #print df_rain_weekend.head()
    #print df_norain_weekday.head()
    #print df_norain_weekend.head()

    #output some characteristics of the data
    print '\n'
    print 'Number of riders when it rains: ' + str(len(df_rain))
    print 'Number of riders with no rain: ' + str(len(df_norain))
    print 'Number of riders when it rains (weekday): ' + str(len(df_rain_
weekday))
    print 'Number of riders with no rain (weekday): ' + str(len(df_norain
_weekday))
    print 'Number of riders when it rains (weekend): ' + str(len(df_rain_
weekend))
    print 'Number of riders with no rain (weekend): ' + str(len(df_norain
_weekend))
    print 'U1 + U2: ' + str(1/2. * (len(df_rain_weekday) * len(df_norain_
weekday)))

    with_rain_mean = np.mean(df_rain['ENTRIESn_hourly'])
    without_rain_mean = np.mean(df_norain['ENTRIESn_hourly'])
    with_rain_mean_weekday = np.mean(df_rain_weekday['ENTRIESn_hourly'])
    with_rain_mean_weekend = np.mean(df_rain_weekend['ENTRIESn_hourly'])
    without_rain_mean_weekday = np.mean(df_norain_weekday['ENTRIESn_hourl
y'])
    without_rain_mean_weekend = np.mean(df_norain_weekend['ENTRIESn_hourl
y'])

    #output some characteristics of the data
    print '\n'
    print 'With rain mean: ' + str(with_rain_mean)
    print 'Without rain mean: ' + str(without_rain_mean)
    print 'With rain mean (weekday): ' + str(with_rain_mean_weekday)
    print 'Without rain mean (weekday): ' + str(without_rain_mean_weekday
)
    print 'With rain mean (weekend): ' + str(with_rain_mean_weekend)
    print 'Without rain mean (weekend): ' + str(without_rain_mean_weekend
)
    print '\n'

    #setup the plots
    plt.figure(figsize=(12,9))
    fig = plt.figure()

    ax1 = plt.axes([0,0,2,2])
    divider = make_axes_locatable(ax1)
    ax1.set_xlim([0, 3000])
```

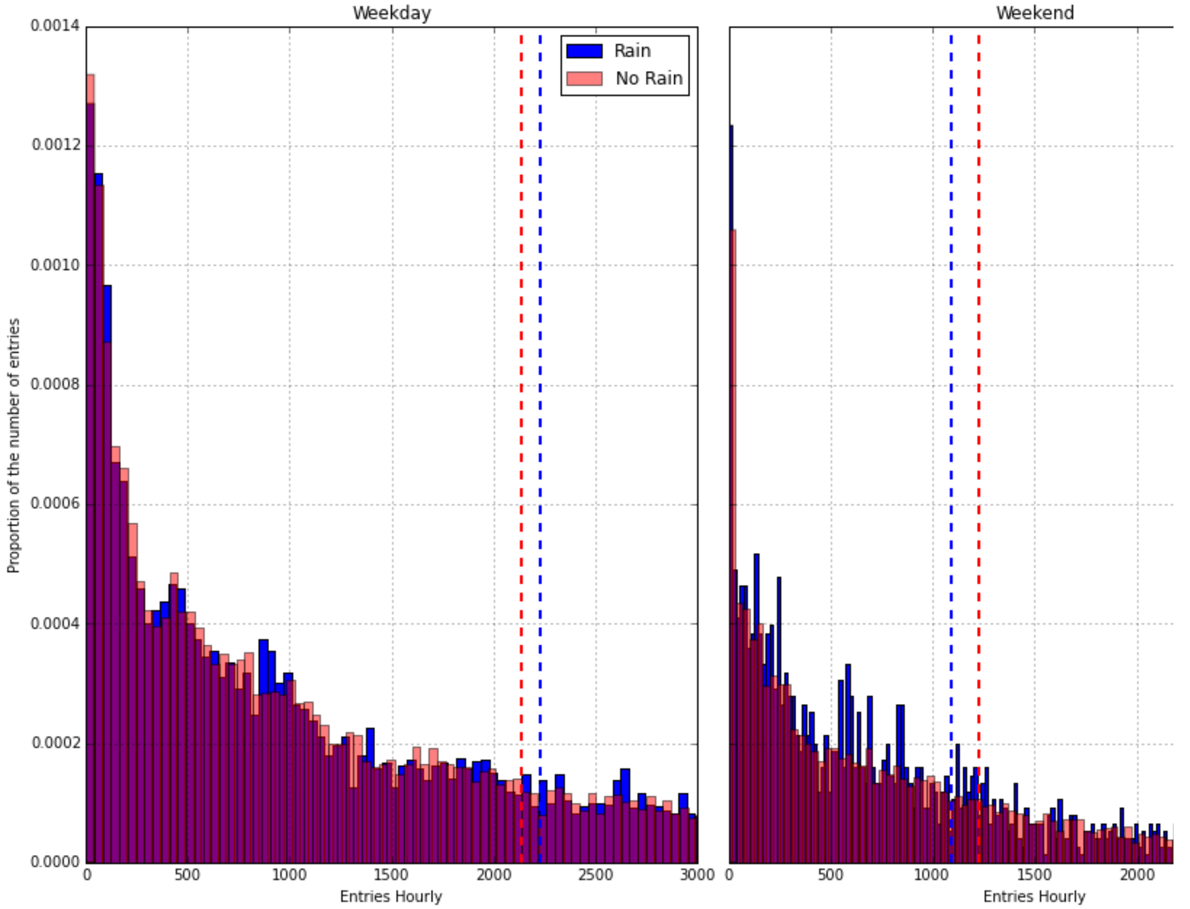
In [173]:

```
print weekday_histogram(turnstile_weather) #original data
print weekday_histogram(turnstile_weather_v2) #version 2 of the data
```

```
Number of riders when it rains: 44104
Number of riders with no rain: 87847
Number of riders when it rains (weekday): 35423
Number of riders with no rain (weekday): 57389
Number of riders when it rains (weekend): 8681
Number of riders with no rain (weekend): 30458
U1 + U2: 1016445273.5


With rain mean: 1105.44637675
Without rain mean: 1090.27878015
With rain mean (weekday): 1198.15029783
Without rain mean (weekday): 1304.53623517
With rain mean (weekend): 727.166109895
Without rain mean (weekend): 686.574627356


<module 'matplotlib.pyplot' from '/Users/rclement2/anaconda/lib/python2.7/
site-packages/matplotlib/pyplot.pyc'>


Number of riders when it rains: 9585
Number of riders with no rain: 33064
Number of riders when it rains (weekday): 7900
Number of riders with no rain (weekday): 22570
Number of riders when it rains (weekend): 1685
Number of riders with no rain (weekend): 10494
U1 + U2: 89151500.0


With rain mean: 2028.19603547
Without rain mean: 1845.53943866
With rain mean (weekday): 2227.96126582
Without rain mean (weekday): 2133.56969428
With rain mean (weekend): 1091.61127596
Without rain mean (weekend): 1226.0575567


<module 'matplotlib.pyplot' from '/Users/rclement2/anaconda/lib/python2.7/
site-packages/matplotlib/pyplot.pyc'>

<matplotlib.figure.Figure at 0x113df6c50>
```

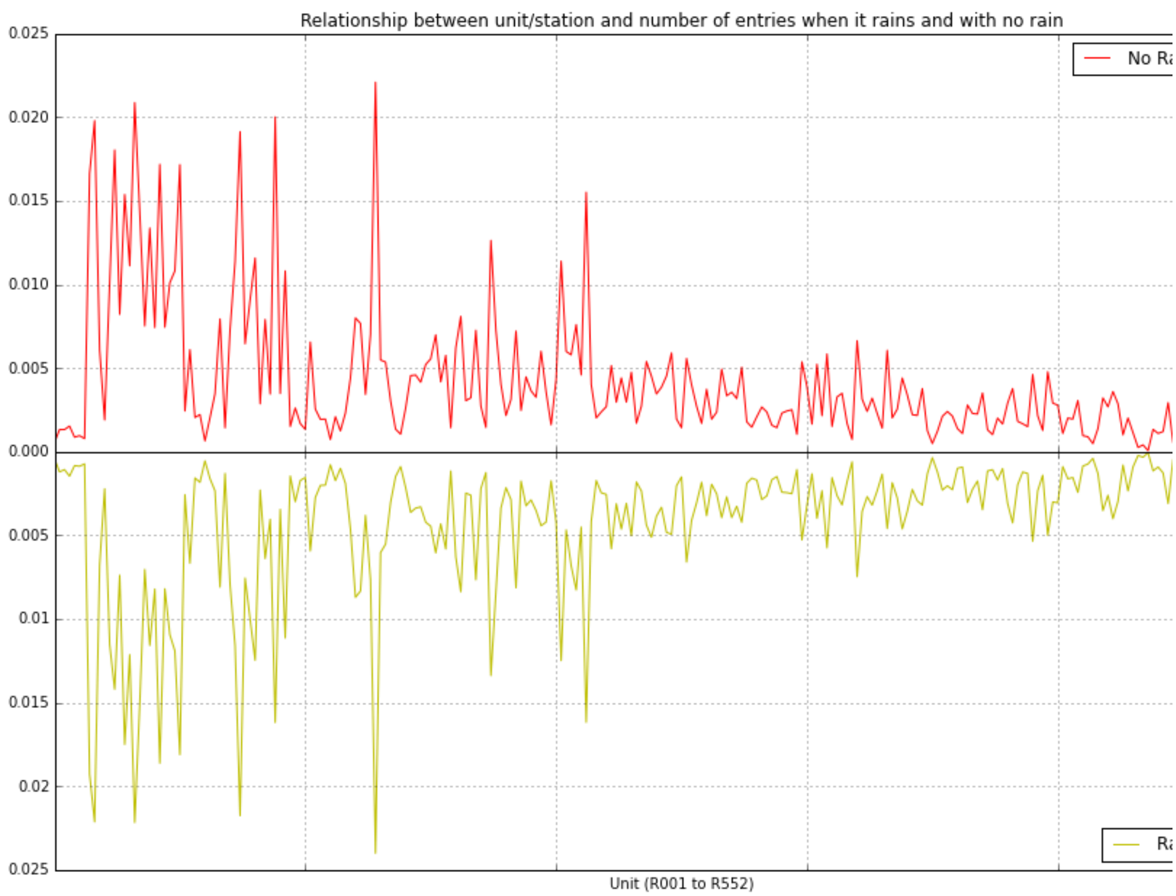


```
<matplotlib.figure.Figure at 0x11ae96890>
```



In [61]:
```
def unit_histogram(df):

    """
    Look at the relationship between Unit/Station, weekday or weekend and
 rain or no rain to see if there's
    any patterns that come out. Plots are created as mirrors of each othe
r for comparison
    """

    #the data
    df = df.set_index([pd.to_datetime(df['DATEn'] + ' ' + df['TIMEn'])])
    df['weekday'] = df.index.weekday
    #print df.info()

    df_weekday = df[df['weekday'] < 5][['UNIT', 'ENTRIESn_hourly', 'EXITS
n_hourly', 'weekday', 'rain']] #weekday results
    df_weekend = df[df['weekday'] >= 5][['UNIT', 'ENTRIESn_hourly', 'EXIT
Sn_hourly', 'weekday', 'rain']] #weekend results
    #print df_weekday.info()
    #print df_weekend.info()

    #now separate into rain or no rain for each dataframe
    df_rain_weekday = df_weekday[df_weekday['rain'] == 1]
    df_rain_weekend = df_weekend[df_weekend['rain'] == 1]
    df_norain_weekday = df_weekday[df_weekday['rain'] == 0]
    df_norain_weekend = df_weekend[df_weekend['rain'] == 0]

    #get the totals for each unit/station from the weekday vs weekend and
 rain vs rain dataframes
    df_rain_weekday = df_rain_weekday.groupby(['UNIT'])[['ENTRIESn_hourly
', 'EXITSn_hourly']].aggregate(sum)
    df_norain_weekday = df_norain_weekday.groupby(['UNIT'])[['ENTRIESn_ho
urly', 'EXITSn_hourly']].aggregate(sum)
    df_rain_weekend = df_rain_weekend.groupby(['UNIT'])[['ENTRIESn_hourly
', 'EXITSn_hourly']].aggregate(sum)
    df_norain_weekend = df_norain_weekend.groupby(['UNIT'])[['ENTRIESn_ho
urly', 'EXITSn_hourly']].aggregate(sum)

    #need to normalize the values so find the totals regardless of the un
it
    norain_weekday_sum = df_norain_weekday[['ENTRIESn_hourly']].sum()
    rain_weekday_sum = df_rain_weekday[['ENTRIESn_hourly']].sum()
    norain_weekend_sum = df_norain_weekend[['ENTRIESn_hourly']].sum()
    rain_weekend_sum = df_rain_weekend[['ENTRIESn_hourly']].sum()

    #print norain_weekday_sum
    #print rain_weekday_sum
    #print norain_weekend_sum
    #print rain_weekend_sum

    #organize the data for plotting. Make sure it goes in order for the s
tations provided
    df_rain_weekday = df_rain_weekday.sort()
    df_norain_weekday = df_norain_weekday.sort()
    df_rain_weekend = df_rain_weekend.sort()
    df_norain_weekend = df_norain_weekend.sort()

    #organize the data for plotting. Make sure an actual integer can be u
sed for the X-axis
    df_rain_weekday = df_rain_weekday.reset_index()
    df_norain_weekday = df_norain_weekday.reset_index()
    df_rain_weekend = df_rain_weekend.reset_index()
    df_norain_weekend = df_norain_weekend.reset_index()

    #take a look at the mean for each dataframe to see if it shows anythi
ng
    df_norain_weekday_mean = df_norain_weekday.mean()
    df_rain_weekday_mean = df_rain_weekday.mean()
    df_norain_weekend_mean = df_norain_weekend.mean()
    df_rain_weekend_mean = df_rain_weekend.mean()

    #print (df_norain_weekday_mean/norain_weekday_sum).head()
    #print (df_rain_weekday_mean/rain_weekday_sum).head()
    #print (df_norain_weekend_mean/norain_weekend_sum).head()
    #print (df_rain_weekend_mean/rain_weekend_sum).head()

    #print (df_norain_weekday/norain_weekday_sum)['ENTRIESn_hourly'].head
()
    #print (df_rain_weekday/rain_weekday_sum)['ENTRIESn_hourly'].head()
    #print (df_norain_weekend/norain_weekend_sum)['ENTRIESn_hourly'].head
()
    #print (df_rain_weekend/rain_weekend_sum)['ENTRIESn_hourly'].head()
```

In [62]:
```
unit_histogram(turnstile_weather) #original data
unit_histogram(turnstile_weather_v2) #version 2 data
```



In [213]:
```
def mann_whitney_plus_means(df):

    """
    The Mann-Whitney U test is used to compare differences between two in
dependent groups
    when the dependent variable is either ordinal or continuous, but not
normally distributed.
    Want to use results of the Mann-Whitney U test to see if the data pop
ulations are
    statistically different.

    First, for any Mann-Whitney U test, the theoretical range of U is fro
m 0 (complete separation
    between groups which means the null hypothesis (H0) most likely false
 and alternate hypothesis (H1)
    most likely true) to n1*n2 (little evidence in support of alternate h
ypothesis (H1)).
    In every test, we must determine whether the observed U supports the
null or alternate hypothesis.
    Specifically, we need to determine a critical value of U such that if
 the observed value of U is
    less than or equal to the critical value, we reject H0 in favor of H1
 and if the observed value of
    U exceeds the critical value we do not reject H0.

    In terms of p we are looking for small values. p < 0.05
    If the p value is large, the data do not give you any reason to rejec
t the null hypothesis.
    This is not the same as saying that the two populations are the same.
 You just have no
    compelling evidence that they differ.
    """

    #the data
    #will subdivide the data into weekday and weekend groups. We will als
o keep the entire week data
    df = df.set_index([pd.to_datetime(df['DATEn'] + ' ' + df['TIMEn'])])
    df['weekday'] = df.index.weekday
    #print df.info()

    df_weekday = df[df['weekday'] < 5][['ENTRIESn_hourly', 'EXITSn_hourly
', 'weekday', 'rain']]
    df_weekend = df[df['weekday'] >= 5][['ENTRIESn_hourly', 'EXITSn_hourl
y', 'weekday', 'rain']]
    #print df_weekday.info()
    #print df_weekend.info()

    #for each of the different dataframes used as input segment these eve
n further by rain/no rain
    df_rain_weekday = df_weekday[df_weekday['rain'] == 1]
    df_rain_weekend = df_weekend[df_weekend['rain'] == 1]
    df_norain_weekday = df_weekday[df_weekday['rain'] == 0]
    df_norain_weekend = df_weekend[df_weekend['rain'] == 0]

    df_rain = df[df['rain'] == 1]
    df_norain = df[df['rain'] == 0]

    #print df_rain_weekday.head()
    #print df_rain_weekend.head()
    #print df_norain_weekday.head()
    #print df_norain_weekend.head()
    #print 1/2. * (len(df_rain_weekday) * len(df_norain_weekday))

    #output (Note: Need to look at making a loop to take care of this rat
her than the copy/paste approach)
    #Calculated values for U, p, mean and count for each segmentation

    #entire week output
    with_rain_mean = np.mean(df_rain['ENTRIESn_hourly'])
    without_rain_mean = np.mean(df_norain['ENTRIESn_hourly'])
    U, p = scipy.stats.mannwhitneyu(df_rain['ENTRIESn_hourly'], df_norain
['ENTRIESn_hourly'])
    print '\nU: ' + str(U) + ' p: ' + str(2.0*p) + ' p is less than 0.05?
 ' + str((2.0*p < 0.050))
    print 'mean with rain: ' + str(with_rain_mean)
    print 'mean without rain: ' + str(without_rain_mean)
    print 'count with rain: ' + str(len(df_rain))
    print 'count without rain: ' + str(len(df_norain))

    #weekday output
    with_rain_mean_weekday = np.mean(df_rain_weekday['ENTRIESn_hourly'])
    without_rain_mean_weekday = np.mean(df_norain_weekday['ENTRIESn_hourl
```

In [214]:
```
mann_whitney_plus_means(turnstile_weather) #original data
```

```
U: 1924409167.0 p: 0.049999825587 p is less than 0.05? True
mean with rain: 1105.44637675
mean without rain: 1090.27878015
count with rain: 44104
count without rain: 87847

U: 985531035.5 p: 6.37487756025e-15 p is less than 0.05? True
mean with rain (weekday): 1198.15029783
mean without rain (weekday): 1304.53623517
count with rain (weekday): 35423
count without rain (weekday): 57389

U: 129024195.5 p: 0.000619366333927 p is less than 0.05? True
mean with rain (weekend): 727.166109895
mean without rain (weekend): 686.574627356
count with rain (weekend): 8681
count without rain (weekend): 30458
```

In [215]:
```
mann_whitney_plus_means(turnstile_weather_v2) #version 2 data
```

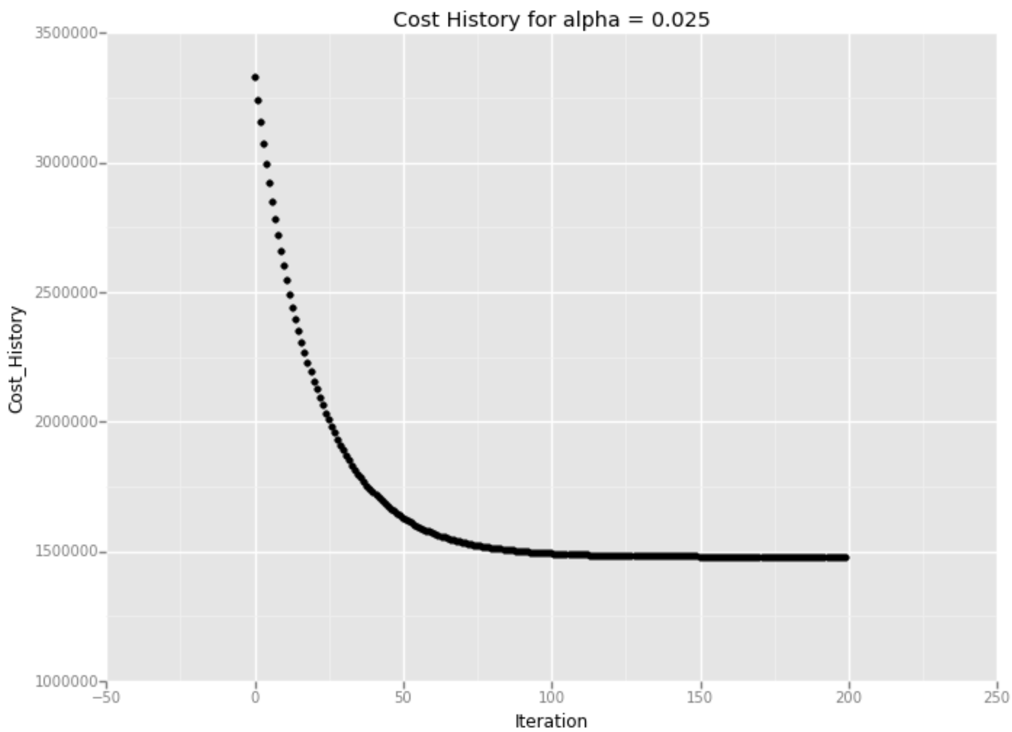
```
U: 153635120.5 p: 5.48213914249e-06 p is less than 0.05? True
mean with rain: 2028.19603547
mean without rain: 1845.53943866
count with rain: 9585
count without rain: 33064

U: 88065521.5 p: 0.106538831417 p is less than 0.05? False
mean with rain (weekday): 2227.96126582
mean without rain (weekday): 2133.56969428
count with rain (weekday): 7900
count without rain (weekday): 22570

U: 8571295.5 p: 0.043933047194 p is less than 0.05? True
mean with rain (weekend): 1091.61127596
mean without rain (weekend): 1226.0575567
count with rain (weekend): 1685
count without rain (weekend): 10494
```

In [66]:
```python
def normalize_features(array):
   """
   Normalize the features in the data set.
   """
   array_normalized = (array-array.mean())/array.std()
   mu = array.mean()
   sigma = array.std()

   return array_normalized, mu, sigma

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

    hypothesis = np.dot(features, theta)
    loss = hypothesis - values
    m = len(values)
    # avg cost per example (the 2 in 2*m doesn't really matter here.
    # But to be consistent with the gradient, I include it)
    cost = np.sum(loss ** 2) / (2 * m)

    return cost, loss

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

    cost_history = []
    m = len(values)
    xTrans = features.transpose()

    for i in range(0, num_iterations):

        cost, loss = compute_cost(features, values, theta)

        cost_history.append(cost)
        # print("Iteration %d | Cost: %f" % (i, cost))
        #
        #  avg gradient per example
        gradient = np.dot(xTrans, loss) / m

        # update
        theta = theta - alpha * gradient
        # print theta

    return theta, pd.Series(cost_history)

def set_features(df):

    """
    Need to determine which list of features to use since the datasets pr
ovided do not have the same columns.
    This is also where you would adjust the features for iterating.
    """

    #print df.columns.tolist()
    L1 = ['Hour', 'rain', 'fog', 'mintempi', 'maxtempi', 'meanpressurei',
 'meanwindspdi']
    L2 = ['Hour', 'rain', 'fog', 'meantempi', 'meanpressurei', 'meanwspdi
']

    if len([(lookfor, maybe) for lookfor in L1 for maybe in df.columns.to
list() if maybe == lookfor]) == len(L1):
        return L1
    if len([(lookfor, maybe) for lookfor in L2 for maybe in df.columns.to
list() if maybe == lookfor]) == len(L2):
        return L2

def predictions(df):
    # Select Features
    df = df.set_index([pd.to_datetime(df['DATEn'] + ' ' + df['TIMEn'])])
    df['Hour'] = df.index.hour
    df = df.reset_index()
    #print df.info()

    feature_list = set_features(df)
    print 'Feature list: ' + str(feature_list)
    features = df[feature_list]
```

In [67]:

```python
"""
First get the predictions and the plot then use the information to calculate r_squared.
This particular set of functionality is using the original data.
NOTE: depending on the alpha, number of features and number of iterations run length may vary
"""
predicted_values, plot = predictions(turnstile_weather)
r_squared = compute_r_squared(turnstile_weather['ENTRIESn_hourly'], predicted_values)

print r_squared
print plot
```

```
Feature list: ['Hour', 'rain', 'fog', 'mintempi', 'maxtempi', 'meanpressurei', 'meanwindspdi']

R^2: 0.459272328625
```

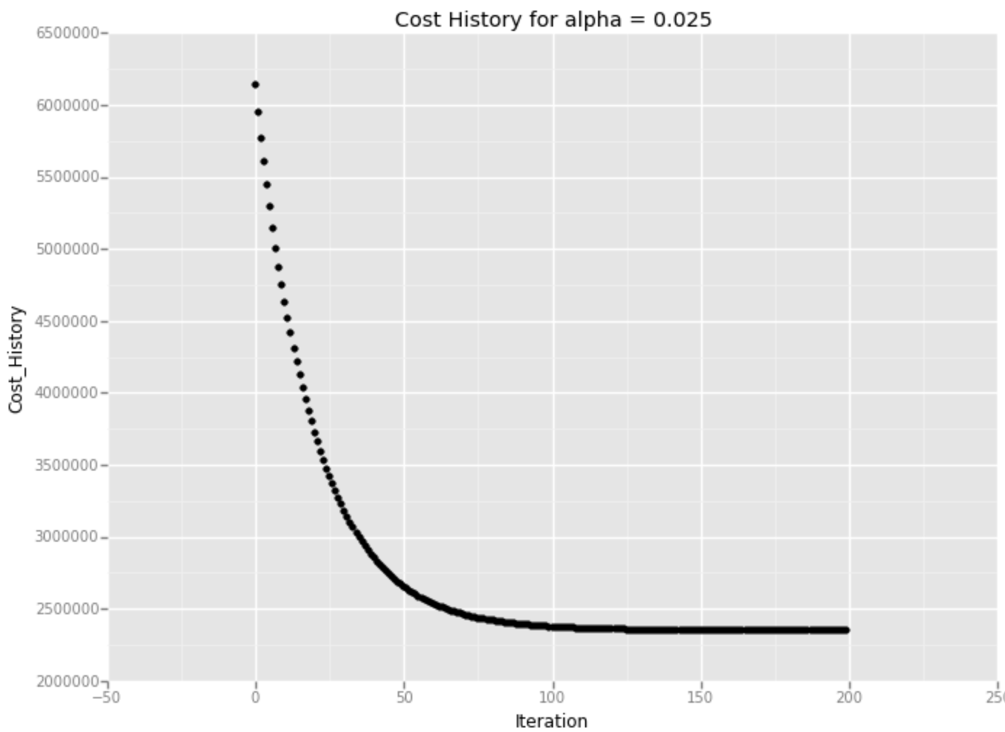



```
&lt;ggplot: (285412345)&gt;
```

In [68]:

```
"""
First get the predictions and the plot then use the information to calcul
ate r_squared.
This particular set of functionality is using the version 2 data
NOTE: depending on the alpha, number of features and number of iterations
 run length may vary
"""
predicted_values_v2, plot = predictions(turnstile_weather_v2)
r_squared = compute_r_squared(turnstile_weather_v2['ENTRIESn_hourly'], pr
edicted_values_v2)

print r_squared
print plot
```

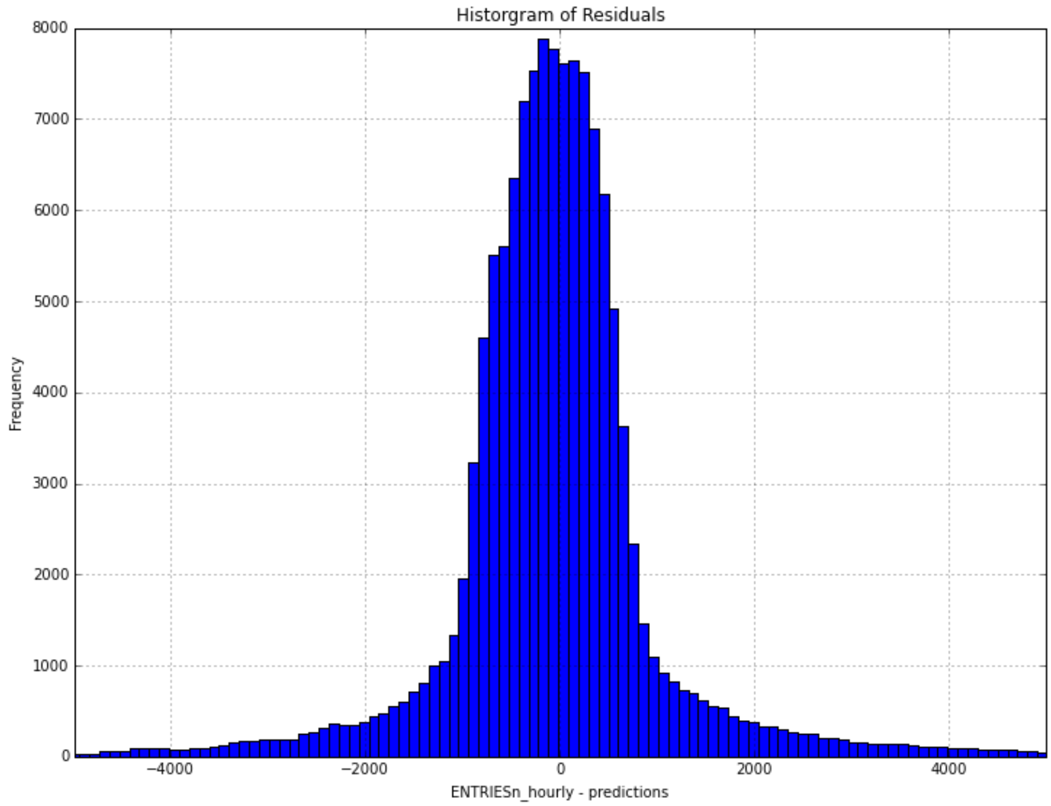
```
Feature list: ['Hour', 'rain', 'fog', 'meantempi', 'meanpressurei', 'meanw
spdi']

R^2: 0.461424565814
```




```
<ggplot: (290054793)>
```

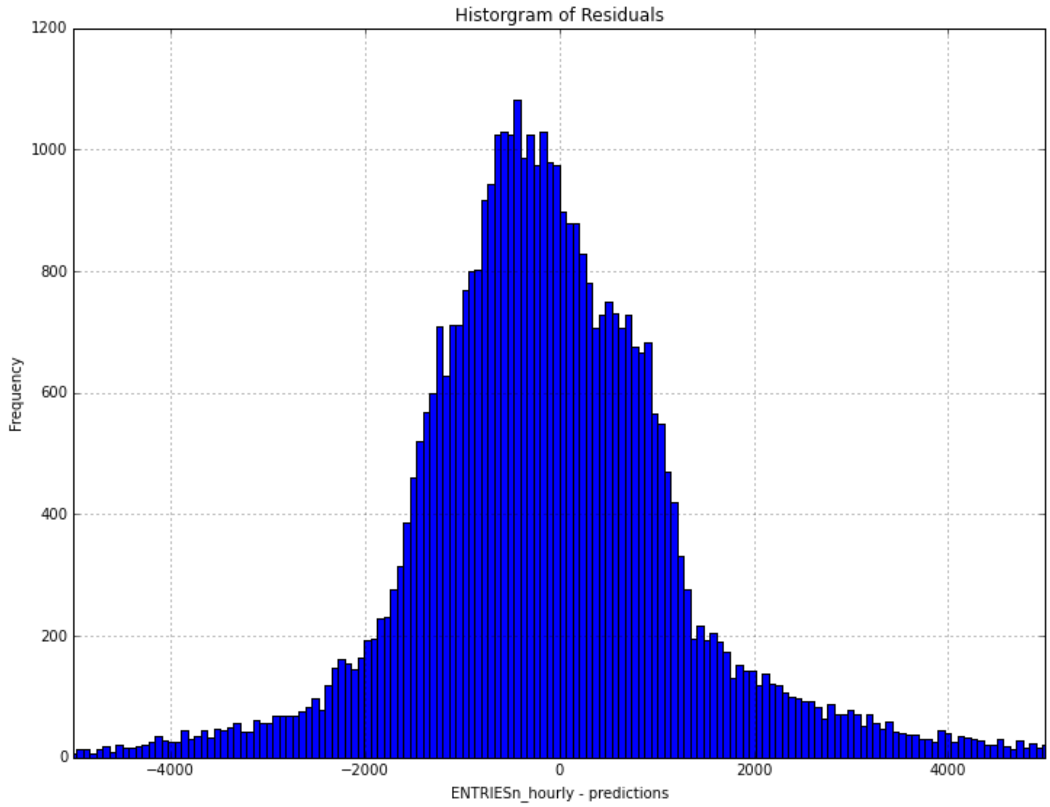
In [46]:

```
"""
Plot the actuals versus the predicted values calculated. Looking for a no
rmal distribution.
This plot is for the originsl set of data
"""

#predicted_values, plot = predictions(turnstile_weather)
plot_residuals(turnstile_weather, predicted_values)
```



In [48]:

```
"""
Plot the actuals versus the predicted values calculated. Looking for a no
rmal distribution.
This plot is for the version 2 set of data
"""
#predicted_values_v2, plot = predictions(turnstile_weather_v2)
plot_residuals(turnstile_weather_v2, predicted_values_v2)
```



In [ ]: