Udacity Nanodegree Project 1

NYC Subway Ridership and Weather Analysis

By: Taylor Somma

This is a project analyzing data obtained from the NYC subway system and provided by Udacity. The objective of this analyis is to determine what facets of the weather, time of day, and day of week most influence the number of people that ride the subway in NYC. After determining the variables with the strongest correlation we will produce a predictive model using linear regression with gradient descent to best predict the number of subway riders.

Sources
Udacity (http://www.udacity.com)
Pandas Documentation (http://pandas.pydata.org/pandas-docs/stable/index.html)
Statistical Test Flowchart (http://abacus.bates.edu/~ganderso/biology/resources/stats_flow_chart_v2014.pdf)
Parametric vs non-parametric data (http://www.csse.monash.edu.au/~smarkham/resources/param.htm)
Mann-Whitney U (https://en.wikipedia.org/wiki/Mann%E2%80%93Whitney_U_test)
Interpretting R-Squared Value (http://blog.minitab.com/blog/adventures-in-statistics/regression-analysis-how-do-i-interpret-r-squared-and-assess-the-goodness-of-fit)

Back to magicfilebox.com (http://www.magicfilebox.com)

Section 1. Statistical Test

1.1 Which statistical test did you use to analyze the NYC subway data? Did you use a one-tail or a two-tail P value? What is the null hypothesis? What is your p-critical value?
I used a two tail p-value because we are testing if there is a difference in means between the datasets without a presumption of which mean is larger.

Assuming a null hypothesis that there is no meaningful difference between the mean number of riders on rainy and non rainy days. I chose to use a p-critical value of 0.01.

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.

A Mann-Whitney test was chosen because the sample sizes of rainy and non-rainy days are different and the frequency distribution shown in figure 1 (http://localhost:8888/notebooks/Desktop/nanoDegree/project_1.ipynb#Figure-1Histogram-of-ENTRIESn_hourly-when-raining-vs-not-raining) is not normally distributed.

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.

As seen here (http://localhost:8888/notebooks/Desktop/nanoDegree/project_1.ipynb#Mann-Whitney-U-Test-Results) the p value of the Mann-Whitney U test is 2.74e-06 which is much less than 0.05. Since this output gives us a one-tailed result we simply need to double it to get the two-tailed p-value of 5.482e-06. The mean number of hourly riders for rainy and non rainy days are 2028 and 1845 respectively.

1.4 What is the significance and interpretation of these results?

Since the p-value is far less than the p-critical value we can safely reject the null hypothesis of any difference in means of ENTRIESn_hourly during rainy and non rainy days being due to chance. Given the p-value and the mean number of ENTRIESn_hourly during rainy days being more than not rainy days we can make the claim that more people ride the subway during rainy days than days with no rain. We can not make any claims of why this is yet but one guess is that people who would have otherwise walked to where they needed to go instead chose to take the subway.

Section 2. Linear Regression

2.1 What approach did you use to compute the coefficients theta and produce prediction for ENTRIESn_hourly in your regression model: OLS using Statsmodels or Scikit Learn Gradient descent using Scikit Learn Or something different?

I used gradient descent using Scikit Learn

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

I used rain, weekday, and hour as input variables in my model. My model used used unit as a dummy variable.

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. Your reasons might be based on intuition. For example, response for fog might be: "I decided to use fog because I thought that when it is very foggy outside people might decide to use the subway more often." Your reasons might also be based on data exploration and experimentation, for example: "I used feature X because as soon as I included it in my model, it drastically improved my R2 value."

I started by testing several input variables independently. The selection of these variables was based partly on intuition and partly on other visualizations I made. These variables were rain, weekday, fog, hour, and latitude and longitude. Looking at the heatmap (='http://localhost:8888/notebooks/Desktop/nanoDegree/project_1.ipynb#Google-Maps-Heatmap-Using-Mean-ENTRIESn_hourly-as-Weight-for-Each-Station') using mean ENTRIESn_hourly as weights it appears that location is a good indicator of ridership. Latitude and Longitude turned out to not be the best predictors of ridership with r-squared values of 0.375 when using OLS linear regression. As seen in this graph (http://localhost:8888/notebooks/Desktop/nanoDegree/project_1.ipynb#Bar-Graph-Showing-Mean-ENTRIESn_hourly-by-Day-of-Week-Split-into-Rain-and-No-Rain-Groups) showing mean ENTRIESn_hourly split by day of week and rain and no rain groups it is apparent that there are less subway riders on the weekend and some minor variations throughout the week. I also chose to use the time of day as a feature in my model because as shown in the graph displaying ENTRIESn_hourly by hour (http://localhost:8888/notebooks/Desktop/nanoDegree/project_1.ipynb#Plot-Showing-Average-Subyway-Riders-Throughout-the-Day-Split-by-Day-of-Week) there is easily apparent variation in entries throughout the day. I dropped fog from the final prediction model because it lowered the r-squared value when paired with the other features. I only tested fog because I thought people might not want to go out on a foggy day. I think fog probably didn't have a significant impact because there were not many foggy days.

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

For some reason I am getting 243 coefficients, they can be seen here (http://localhost:8888/notebooks/Desktop/nanoDegree/project_1.ipynb#Coefficient-Output)

2.5 What is your model's R2 (coefficients of determination) value?

My models R2 is 0.408394170194 when using gradient descent. It is curious that the R2 value with OLS is greated at ~0.48 but appears to have a wider dispersion in the residuals histogram (http://localhost:8888/notebooks/Desktop/nanoDegree/project_1.ipynb#Residual-Plot-Using-OLS-Linear-Regression-and-Linear-Regression-With-Gradient-Descent).

2.6 What does this R2 value mean for the goodness of fit for your regression model? Do you think this linear model to predict ridership is appropriate for this dataset, given this R2 value?

An R-squared value of ~0.408 is not the best and I believe there is still room for improvement with my model. However, we are trying to predict human behavior based on the weather which is difficult becase humans can be unpredictable. Also when looking at the plot of residuals it appears that most of the time the model predicts ENTRIESn_hourly within a few hundred of the actual value.

Section 3. Visualization

Please include two visualizations that show the relationships between two or more variables in the NYC subway data. Remember to add appropriate titles and axes labels to your plots. Also, please add a short description below each figure commenting on the key insights depicted in the figure.

3.1 One visualization should contain two histograms: one of ENTRIESn_hourly for rainy days and one of ENTRIESn_hourly for non-rainy days. You can combine the two histograms in a single plot or you can use two separate plots. If you decide to use to two separate plots for the two histograms, please ensure that the x-axis limits for both of the plots are identical. It is much easier to compare the two in that case. For the histograms, you should have intervals representing the volume of ridership (value of ENTRIESn_hourly) on the x-axis and the frequency of occurrence on the y-axis. For example, each interval (along the x-axis), the height of the bar for this interval will represent the number of records (rows in our data) that have ENTRIESn_hourly that falls in this interval. Remember to increase the number of bins in the histogram (by having larger number of bars). The default bin width is not sufficient to capture the variability in the two samples.

The histogram visualization can be seen here (http://localhost:8888/notebooks/Desktop/nanoDegree/project_1.ipynb#Figure-1Histogram-of-ENTRIESn_hourly-when-raining-vs-not-raining).

3.2 One visualization can be more freeform. You should feel free to implement something that we discussed in class (e.g., scatter plots, line plots) or attempt to implement something more advanced if you'd like. Some suggestions are: Ridership by time-of-day Ridership by day-of-week

Here (http://localhost:8888/notebooks/Desktop/nanoDegree/project_1.ipynb#Plot-Showing-Average-Subyway-Riders-Throughout-the-Day-Split-by-Day-of-Week) is a visualization showing the mean ENTRIESn_hourly with time of day on the x-axis broken up by day of week. Also here (http://localhost:8888/notebooks/Desktop/nanoDegree/project_1.ipynb#Google-Maps-Heatmap-Using-Mean-ENTRIESn_hourly-as-Weight-for-Each-Station) is a heatmap I made using google maps that shows the locations of all of the stations with the weights of the heatmap the mean ENTRIESn_hourly for that station.

Section 4. Conclusion

Code Below

Please address the following questions in detail. Your answers should be 1-2 paragraphs long.

4.1 From your analysis and interpretation of the data, do more people ride the NYC subway when it is raining or when it is not raining?
4.2 What analyses lead you to this conclusion? You should use results from both your statistical tests and your linear regression to support your analysis.

Yes, more people ride the subway on rainy days than when it is not raining. I am using ENTRIESn_hourly to represent ridership. The mean number of riders on rainy days is 2028 versus an average of 1985 on non rainy days. This alone is not statistically significant. This is why I ran a Mann-Whitney U test to assess for a statistically significant difference in the distribution of the datasets representing rainy days and not rainy days. As shown in the histogram plotting the frequency of ENTRIESn_hourly for rainy and non rainy days it is very apparent that the data is not normally distributed. Also apparent is the large difference in the size of the data sets, there are many more non rainy days than rainy days. This is why the Mann-Whitney U test was chosen to test for a difference between the data sets. A Mann-Whitney U test is better suited to data fitting the previously described critera than a standard t-test which requires normally distributed data of relatively equal sample sizes.
After finding that there are in fact more subway riders on rainy days the next step was to create a predictive model using this fact as a starting point. I then used scikit learn to create a linear regression model using rain as the only feature to predict ENTRIESn_hourly. This gave a R-squared value of 0.375. I think we can do better than that. Especially since there is probably variation in ridership on different days of the week and different times of the day. Since our dataset included these variables I added them to the linear regression model which achieved a R-squared value of 0.481 using OLS and 0.408 Using gradient descent. When plotting the residuals in a histogram we can see that over 16000 times the model correctly predicted ENTRIESn_hourly within 400 riders.
There is alot of room for improvement. The predictions that were the most accurate could have just been the times that there were not many riders which when compared to times that had alot of riders could look very accurate if they were off by the same number of riders. Also I think that the station could be used as a good predictor. My net step is to group the dataset by station and take the mean ENTRIESn_hourly for each station. Then I will sort the stations by ENTRIESn_hourly and use the rank as a feature. I think that this will be a good way to further improve the model.

In [20]:
```
# Set notebook to display matplotlib plots in notebook
%matplotlib inline
```

In [21]:
```
#Set size of plots
import pylab
pylab.rcParams['figure.figsize'] = (10.0, 8.0)
```

In [22]:
```
#Import statements
import pandas as pd
import numpy as np
import scipy.stats
import sklearn
import matplotlib.pyplot as plt
from sklearn import linear_model
from sklearn.metrics import r2_score
from IPython.display import HTML, Javascript, display
```

In [23]:
```
# Load regular and improved datasets
improved_dataset = pd.read_csv('improved-dataset/turnstile_weather_v2.csv')
regular_dataset = pd.read_csv('turnstile_data_master_with_weather.csv')
```

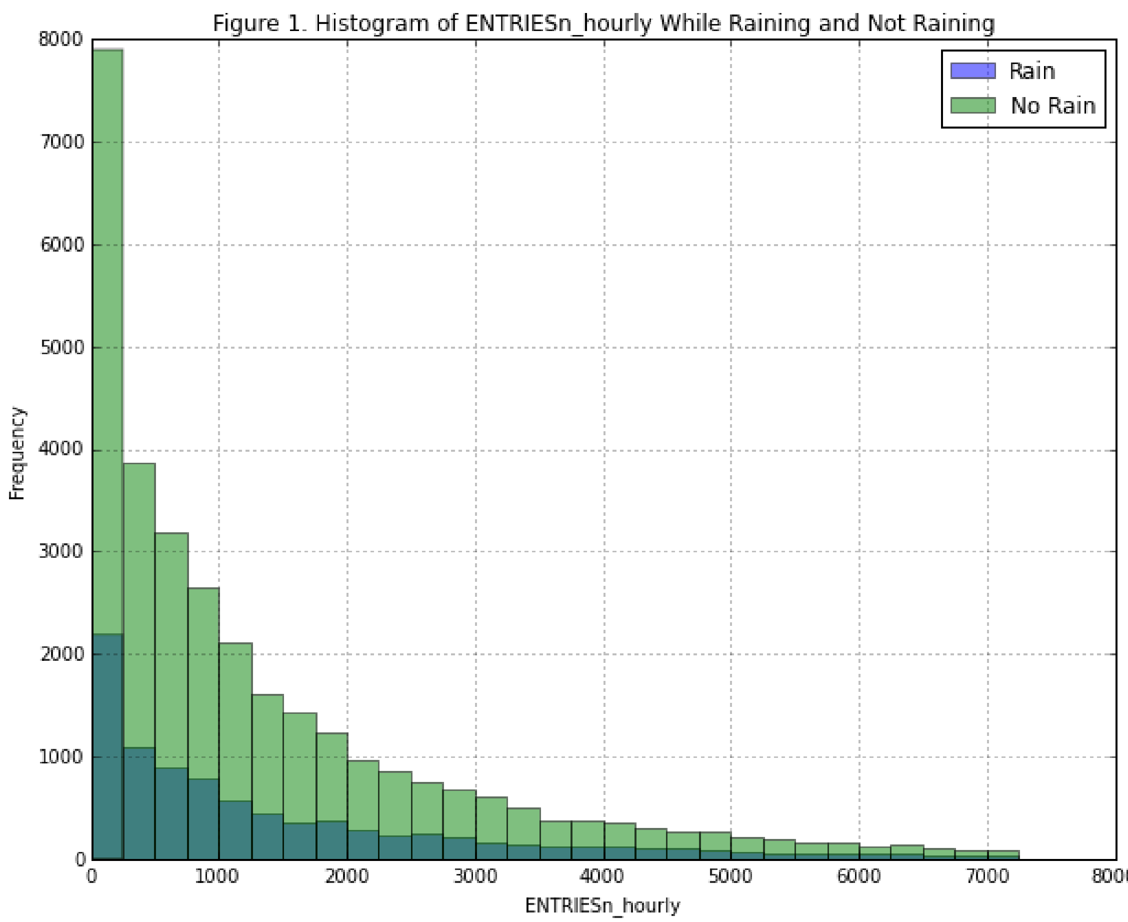
In [24]:
```
#Seperate rainy and non-rainy days
rainyDays = improved_dataset[improved_dataset['rain']==1]
nonRainyDays = improved_dataset[improved_dataset['rain']==0]
```

Figure 1
Histogram of ENTRIESn_hourly when raining vs not raining

[Back to Answers (http://localhost:8888/notebooks/Desktop/nanoDegree/project_1.ipynb#Figure-1Histogram-of-ENTRIESn_hourly-when-raining-vs-not-raining)](http://localhost:8888/notebooks/Desktop/nanoDegree/project_1.ipynb#Figure-1Histogram-of-ENTRIESn_hourly-when-raining-vs-not-raining)

In [25]:
```
binLabels = []
for x in range(30):
    binLabels.append(x*250)
plt.figure()
plt.title('Figure 1. Histogram of ENTRIESn_hourly While Raining and Not Rainin
g')
plt.xlabel('ENTRIESn_hourly')
plt.ylabel('Frequency')
rainyDays['ENTRIESn_hourly'].hist(bins=binLabels, alpha=0.5,label=['Raining'])
nonRainyDays['ENTRIESn_hourly'].hist(bins=binLabels,alpha=0.5,label=['No Rain'])
plt.legend(['Rain','No Rain'])
```

Out[25]: <matplotlib.legend.Legend at 0x10f9ca4d0>



Mann-Whitney U Test Results

In [26]:
```
mannWhitneyOutput = scipy.stats.mannwhitneyu(rainyDays['ENTRIESn_hourly'],nonRai
nyDays['ENTRIESn_hourly'])
rainyDaysMean = rainyDays['ENTRIESn_hourly'].mean()
nonRainyDaysMean = nonRainyDays['ENTRIESn_hourly'].mean()
print mannWhitneyOutput
print 'U : ',mannWhitneyOutput[0]
print 'p one-tailed : ',mannWhitneyOutput[1]
print 'p two-tailed : ',mannWhitneyOutput[1]*2
print 'ENTRIESn_hourly Mean Rainy : ',rainyDaysMean
print 'ENTRIESn_hourly Mean Not Rainy : ',nonRainyDaysMean
```

```
(153635120.5, 2.7410695712437496e-06)
U :  153635120.5
p one-tailed :  2.74106957124e-06
p two-tailed :  5.48213914249e-06
ENTRIESn_hourly Mean Rainy :  2028.19603547
ENTRIESn_hourly Mean Not Rainy :  1845.53943866
```

Pivot Tables Defined

Also prints min,max,and mean temperatures and amount of precipitation

pivot_1 creates [table (http://localhost:8888/notebooks/Desktop/nanoDegree/project_1.ipynb#ENTRIESn_hourly-broken-down-by-day-of-week-and-hour)] to show mean ENTRIESn_hourly broken down by day of week and hour
pivot_2 creates table grouping the dataset into mean ENTRIESn_hourly by meantempi
pivot_3 creates table grouping the dataset into mean ENTRIESn_hourly by meanprecipi
pivot_4 creates [table (http://localhost:8888/notebooks/Desktop/nanoDegree/project_1.ipynb#Average-ENTRIESn_hourly-by-Day-of-Week-and-Rain-or-No-Rain)] to show mean ENTRIESn_hourly broken down by day of week and split into not rainy and rainy days

In [27]:
```
# Build pivot tables for further analysis
improved_dataset['meantempi']=improved_dataset['meantempi'].round(decimals=0)
pivot_1 = pd.pivot_table(improved_dataset,columns='day_week',index='hour',aggfun
c=np.mean)
pivot_2 = pd.pivot_table(improved_dataset,index='meantempi',aggfunc=np.mean)
pivot_3 = pd.pivot_table(improved_dataset,index='meanprecipi',aggfunc=np.mean)
pivot_4 = pd.pivot_table(improved_dataset,columns='rain',index='day_week',aggfun
c=np.mean)

# Get locations of all stations
grouped_by_station = improved_dataset.groupby(['station'])
grouped_by_station_mean = grouped_by_station.aggregate(np.mean)
#Get locations of all units
grouped_by_unit = regular_dataset.groupby(['UNIT'])
grouped_by_unit_mean = grouped_by_station.aggregate(np.mean)
grouped_by_unit_sum = grouped_by_station.aggregate(np.sum)
print 'Min temp: ',improved_dataset['meantempi'].min()
print 'Mean temp: ',improved_dataset['meantempi'].mean()
print 'Max temp: ',improved_dataset['meantempi'].max()
print 'Min precip: ',improved_dataset['meanprecipi'].min()
print 'Mean precip: ',improved_dataset['meanprecipi'].mean()
print 'Max precip: ',improved_dataset['meanprecipi'].max()
```

```
Min temp:  49.0
Mean temp:  63.0922882131
Max temp:  80.0
Min precip:  0.0
Mean precip:  0.00461769326362
Max precip:  0.1575
```

In [28]:
```
#Finds center latitude and longitude for google maps api call
lat_max = grouped_by_station_mean['latitude'].max()
lat_min = grouped_by_station_mean['latitude'].min()
long_max = grouped_by_station_mean['longitude'].max()
long_min = grouped_by_station_mean['longitude'].min()
print 'Latitude Max, Min',lat_max,lat_min,'\nLongitude Max, Min ',long_max,lon
g_min
print 'AVG: ',((lat_max+lat_min)/2),',',((long_max+long_min)/2)

lat_max_unit = grouped_by_unit_mean['latitude'].max()
lat_min_unit = grouped_by_unit_mean['latitude'].min()
long_max_unit = grouped_by_unit_mean['longitude'].max()
long_min_unit = grouped_by_unit_mean['longitude'].min()
print 'Latitude Max, Min',lat_max_unit,lat_min_unit,'\nLongitude Max, Min ',lon
g_max_unit,long_min_unit
print 'AVG: ',((lat_max_unit+lat_min_unit)/2),',',((long_max_unit+long_min_uni
t)/2)
```

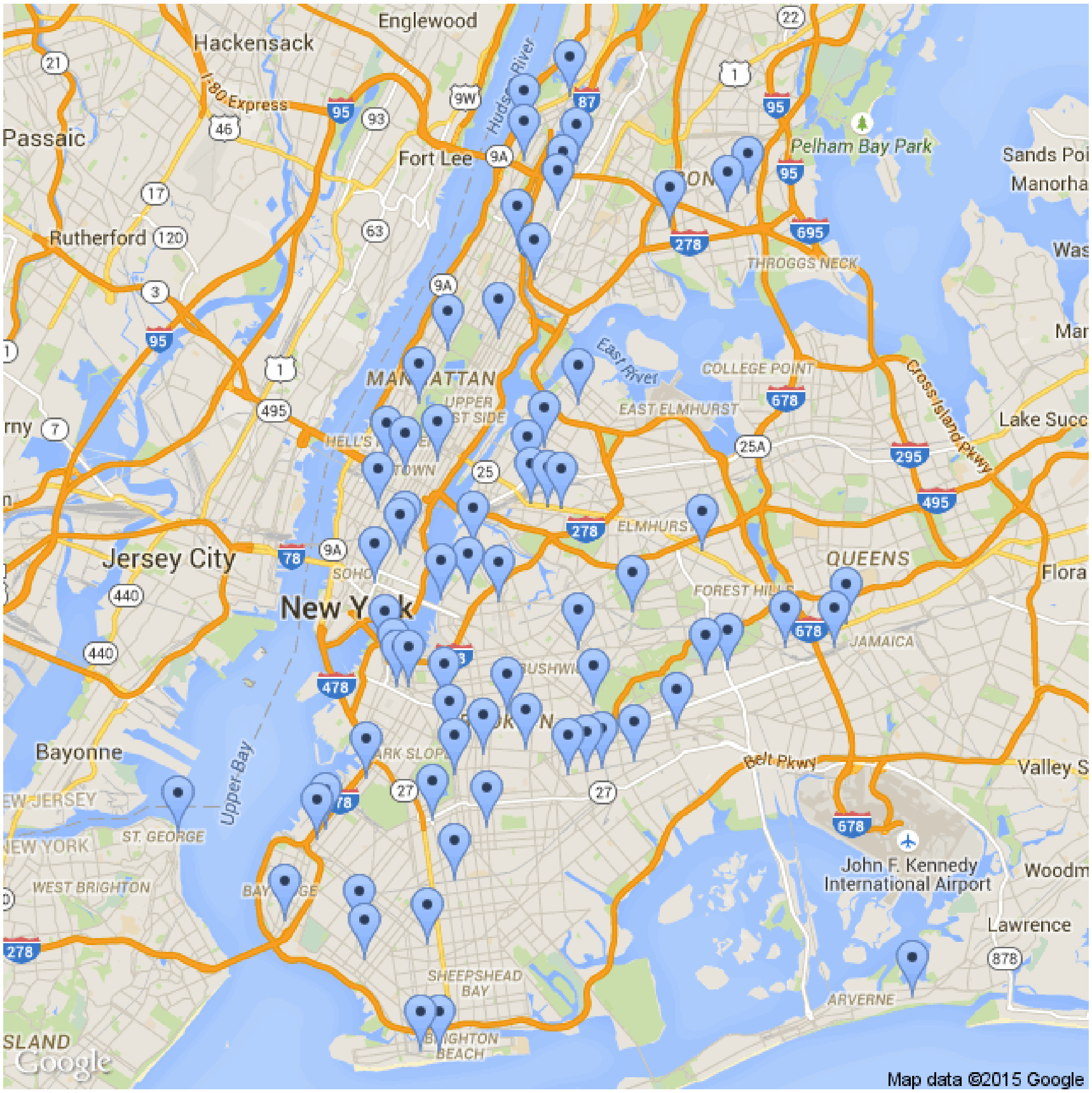
```
Latitude Max, Min 40.889185 40.576152
Longitude Max, Min  -73.755383 -74.073622
AVG:  40.7326685 , -73.9145025
Latitude Max, Min 40.889185 40.576152
Longitude Max, Min  -73.755383 -74.073622
AVG:  40.7326685 , -73.9145025
```

In [29]:
```
# Produce url for google maps api call
#Create google heat map lat,long objects in an array
#new google.maps.LatLng(37.782551, -122.445368),
#{location: new google.maps.LatLng(37.782, -122.447), weight: 0.5},
markerString = ''
latitudeArray = grouped_by_station_mean['latitude']
longitudeArray = grouped_by_station_mean['longitude']
meanEntries = grouped_by_station_mean['ENTRIESn_hourly']

lat_long_objects=''
lat_long_objects_weighted=''
for x in range(len(latitudeArray)):
    lat_long_objects+='new google.maps.LatLng('+str(latitudeArray[x])+','+str(lo
ngitudeArray[x])+'),'
    lat_long_objects_weighted+='{location: new google.maps.LatLng('+str(latitude
Array[x])+','+str(longitudeArray[x])+'), weight: '+str(meanEntries[x])+'},'
    if x%3==0:
        markerString += str(latitudeArray[x])+','+str(longitudeArray[x])+'|'
google_maps_marker_shell='https://maps.googleapis.com/maps/api/staticmap?size=80
0x800&maptype=roadmap&markers=color:blue%7C'
google_maps_callurl = google_maps_marker_shell+markerString[0:len(markerStrin
g)-1]
"""The below lines are commented out because they are the print statements whose
outputs were used to provide the latitude,longitude, and weights for google map
s. The weights are simply the mean ENTRIESn_hourly"""
#print google_maps_callurl
#print lat_long_objects
#print lat_long_objects_weighted
```

Out[29]:
```
'The below lines are commented out because they are the print statements whose o
utputs were used to provide the latitude,longitude, and weights for google maps.
The weights are simply the mean ENTRIESn_hourly'
```

Google Maps Plot Showing Some Subway Station Locations

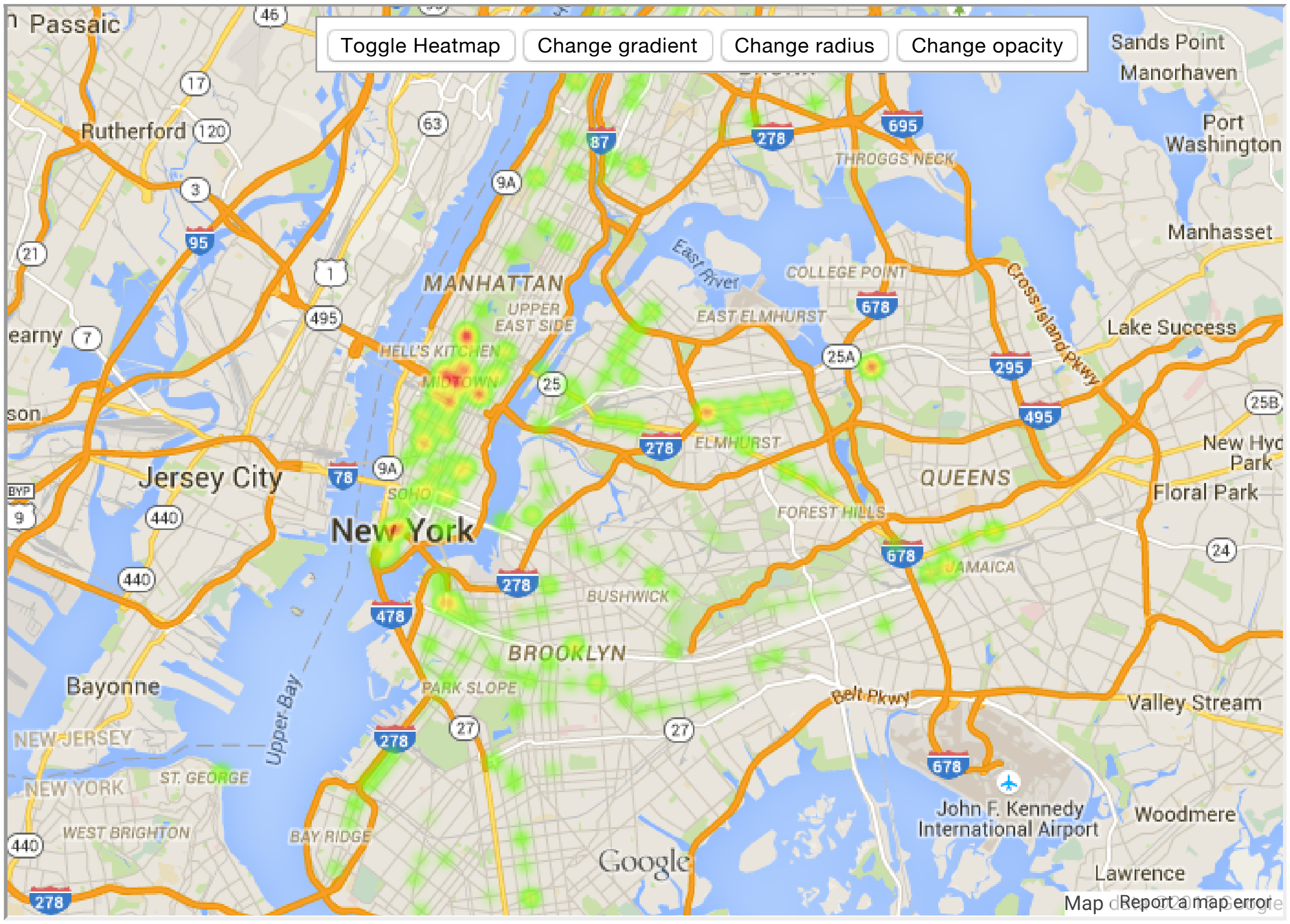


Link to Heatmap built using locations of stations (http://magicfilebox.com/google_map.html)

Google Maps Heatmap Using Mean ENTRIESn_hourly as Weight for Each Station

In [30]:
```
from IPython.core.display import HTML
HTML('<iframe src=http://magicfilebox.com/google_map.html width=700 height=500>
</iframe>')
```

Out[30]:



ENTRIESn_hourly broken down by day of week and hour

In [31]:
```
pivot_1.ENTRIESn_hourly
```

Out[31]:

day_week	0	1	2	3	4	5	6
hour							
0	889.673950	1341.702092	1600.429778	1553.824764	1612.522608	1790.108421	1225.34
4	192.592965	306.079698	339.003151	338.877487	350.583333	329.701994	347.992
8	1063.412801	1207.877963	1082.971963	1049.712534	1006.063274	272.110942	222.387
12	3080.615709	3619.568394	3959.696938	3972.372240	3816.129237	1679.344245	1229.79
16	2266.823830	2535.647210	2577.870213	2651.118393	2849.359275	1880.176842	1541.78
20	3405.477712	3990.406809	3981.178458	4068.230444	3757.201699	2013.174370	1575.41

Average ENTRIESn_hourly by Day of Week and Rain or No Rain

In [32]:
```
pivot_4.ENTRIESn_hourly
```

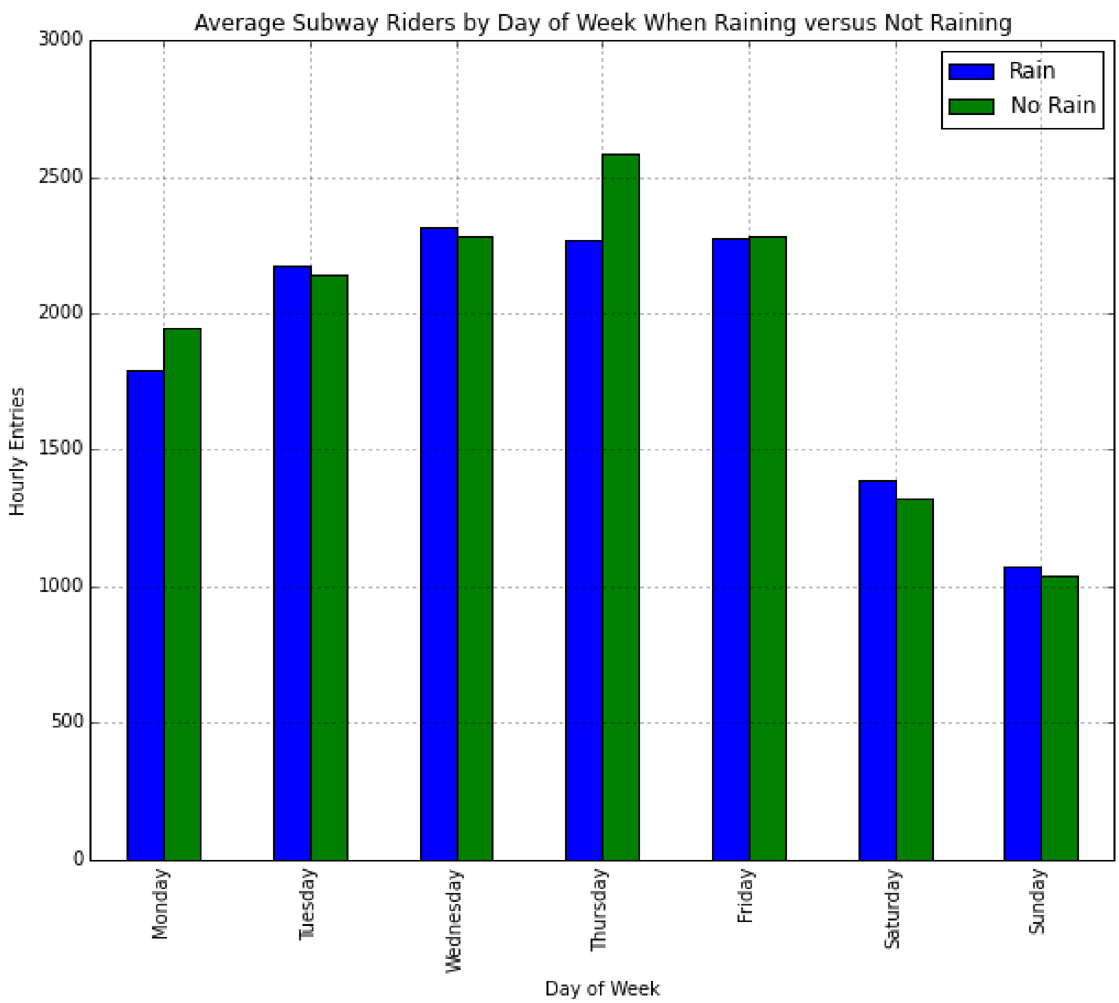
Out[32]:

rain	0	1
day_week		
0	1792.143970	1948.177419
1	2171.126911	2139.860974
2	2313.388767	2280.795255
3	2264.874673	2584.054627
4	2274.958944	2284.644330
5	1388.219575	1317.855422
6	1074.015694	1036.095344

Bar Graph Showing Mean ENTRIESn_hourly by Day of Week Split into Rain and No Rain Groups

In [33]:
```
plot_2 = pivot_4['ENTRIESn_hourly'].plot(kind='bar')
plot_2.set_title('Average Subway Riders by Day of Week When Raining versus Not R
aining')
plot_2.set_ylabel('Hourly Entries')
plot_2.set_xlabel('Day of Week')
plot_2.set_xticklabels(['Monday','Tuesday','Wednesday','Thursday','Friday','Satu
rday','Sunday'])
plot_2.legend(['Rain','No Rain'])
```

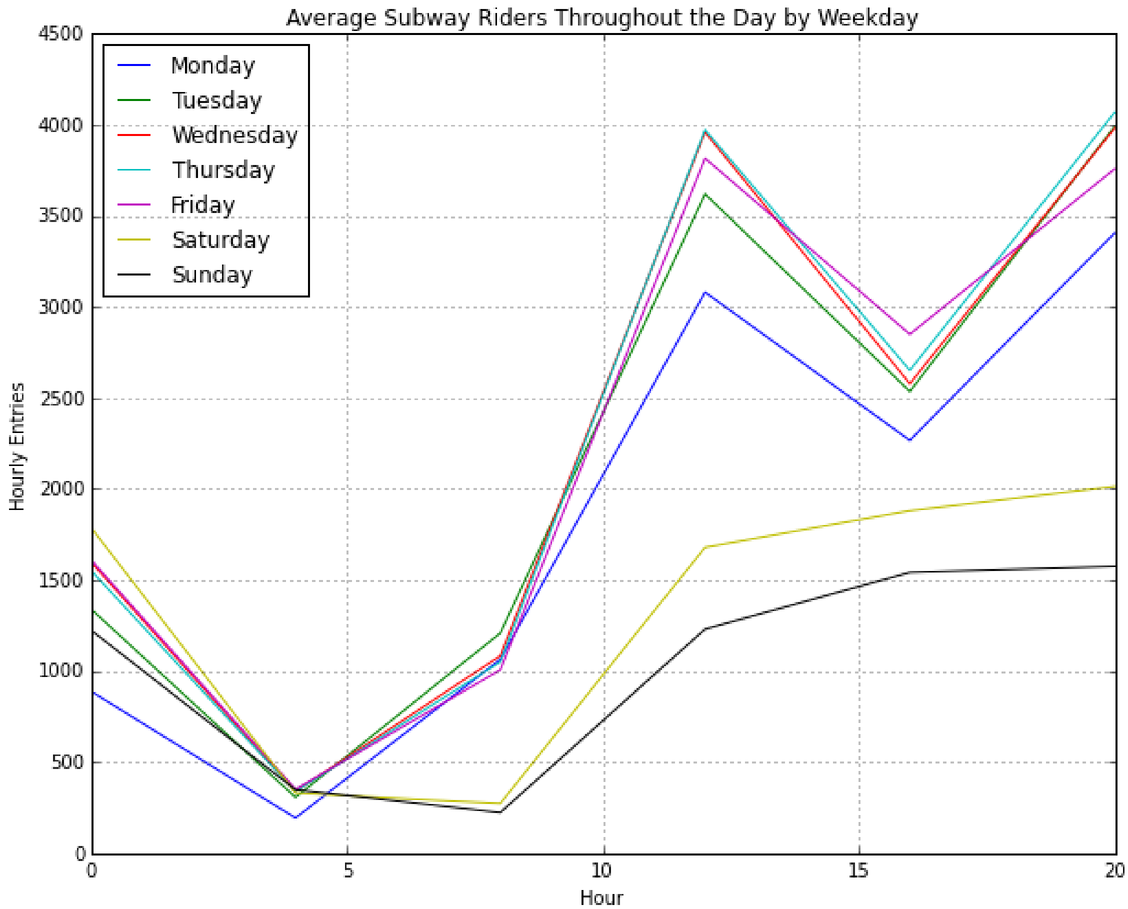
Out[33]: <matplotlib.legend.Legend at 0x10f9ca410>



Plot Showing Average Subyway Riders Throughout the Day Split by Day of Week

In [34]:
```
plot_1 = pivot_1['ENTRIESn_hourly'].plot(title='Average Subway Riders Throughout
the Day by Weekday')
plot_1.set_ylabel('Hourly Entries')
plot_1.set_xlabel('Hour')
plot_1.legend(['Monday','Tuesday','Wednesday','Thursday','Friday','Saturday','Su
nday'],loc='upper left')
```

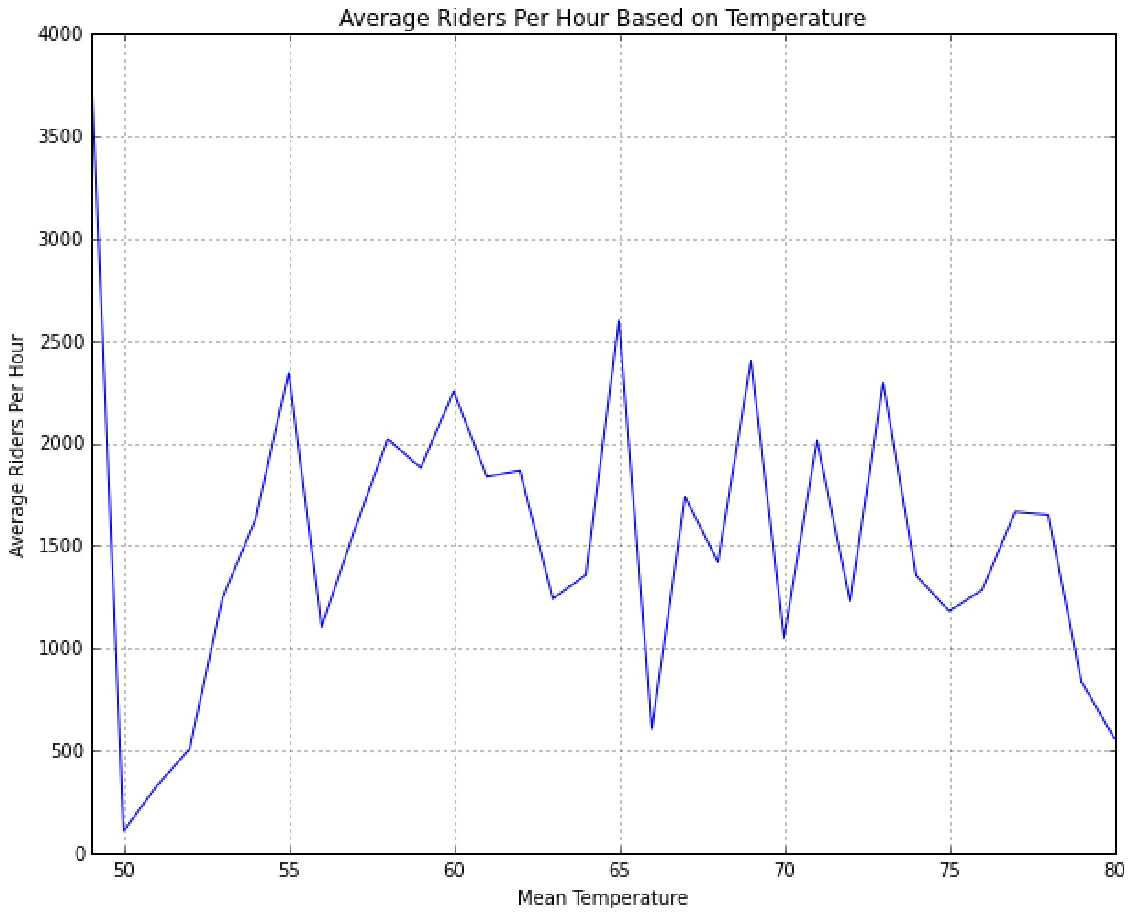
Out[34]: <matplotlib.legend.Legend at 0x10cd45410>



This graph shows the number or subway riders throughout the day broken down by day of the week. It shows that the number of riders declines in the early morning and increases throughout the day. This chart also shows that the overall number or riders is less during weekends than during the week.
I used the pandas pivot table (http://pandas.pydata.org/pandas-docs/stable/reshaping.html) function coupled with numpy.mean (http://docs.scipy.org/doc/numpy/reference/generated/numpy.mean.html) to produce the pivot table this plot was created from

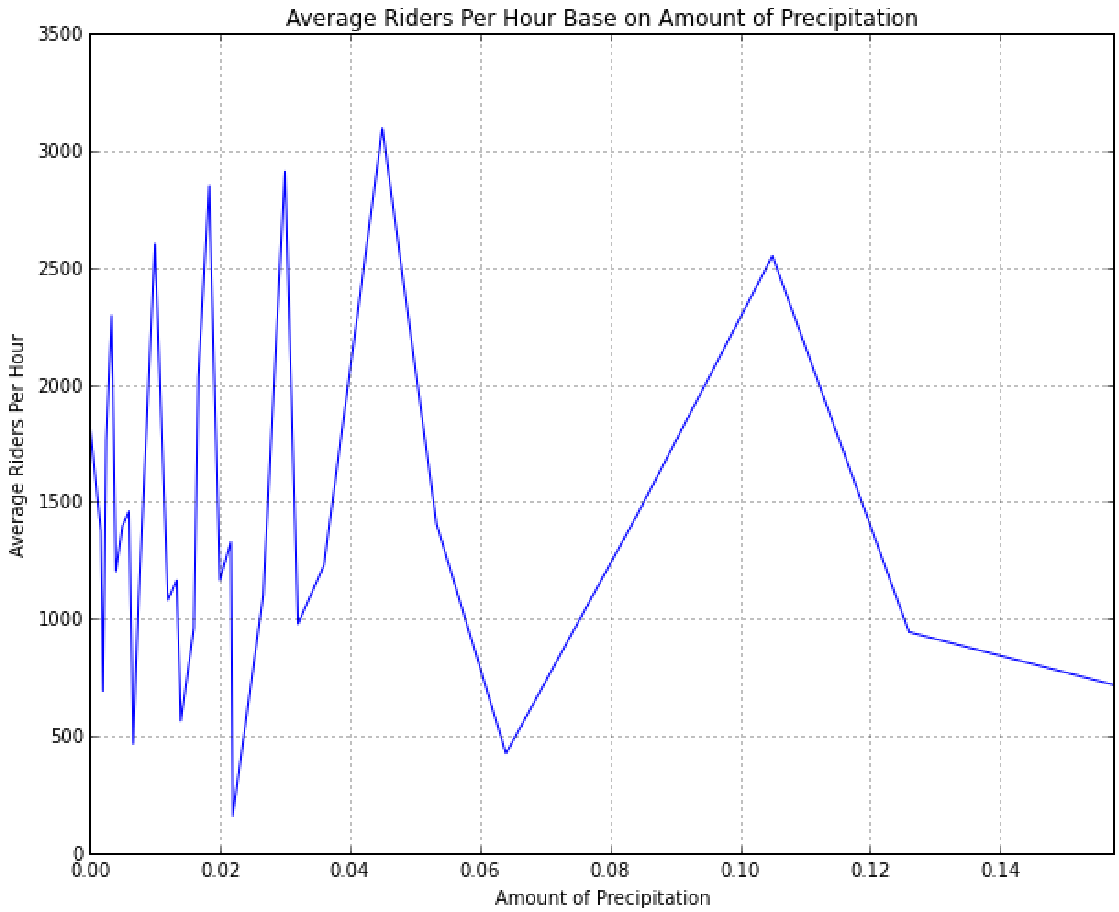
In [35]:
```
plot_3 = pivot_2['ENTRIESn_hourly'].plot()
plot_3.set_ylabel('Average Riders Per Hour')
plot_3.set_xlabel('Mean Temperature')
plot_3.set_title('Average Riders Per Hour Based on Temperature')
```

Out[35]: <matplotlib.text.Text at 0x10953da50>



In [36]:
```
plot_4 = pivot_3['ENTRIESn_hourly'].plot()
plot_4.set_title('Average Riders Per Hour Base on Amount of Precipitation')
plot_4.set_ylabel('Average Riders Per Hour')
plot_4.set_xlabel('Amount of Precipitation')
```

Out[36]: <matplotlib.text.Text at 0x103a804d0>



In [37]:
```
#Seperate Data by Weekday/Weekend and by each day of the week
weekdays = improved_dataset[improved_dataset['day_week']<=4]
weekends = improved_dataset[improved_dataset['day_week']>4]
monday = improved_dataset[improved_dataset['day_week']==0]
tuesday = improved_dataset[improved_dataset['day_week']==1]
wednesday = improved_dataset[improved_dataset['day_week']==2]
thursday = improved_dataset[improved_dataset['day_week']==3]
friday = improved_dataset[improved_dataset['day_week']==4]
saturday = improved_dataset[improved_dataset['day_week']==5]
sunday = improved_dataset[improved_dataset['day_week']==6]
```

In [38]:
```
#Run Mann-Whitney U Comparing Weekdays to Weeken
print 'Mann-Whitney U Test Output ',mann_whitney_u_week_weekend
print 'U value : ',mann_whitney_u_week_weekend[0]
print 'One-Tailed p-value : ',mann_whitney_u_week_weekend[1]
print 'Two-Tailed p-value : ',(mann_whitney_u_week_weekend[1]*2)
print 'Mean ENTRIESn_hourly on Weekends : ',(weekends['ENTRIESn_hourly'].mean())
print 'Mean ENTRIESn_hourly on Weekdays : ',(weekdays['ENTRIESn_hourly'].mean())
```

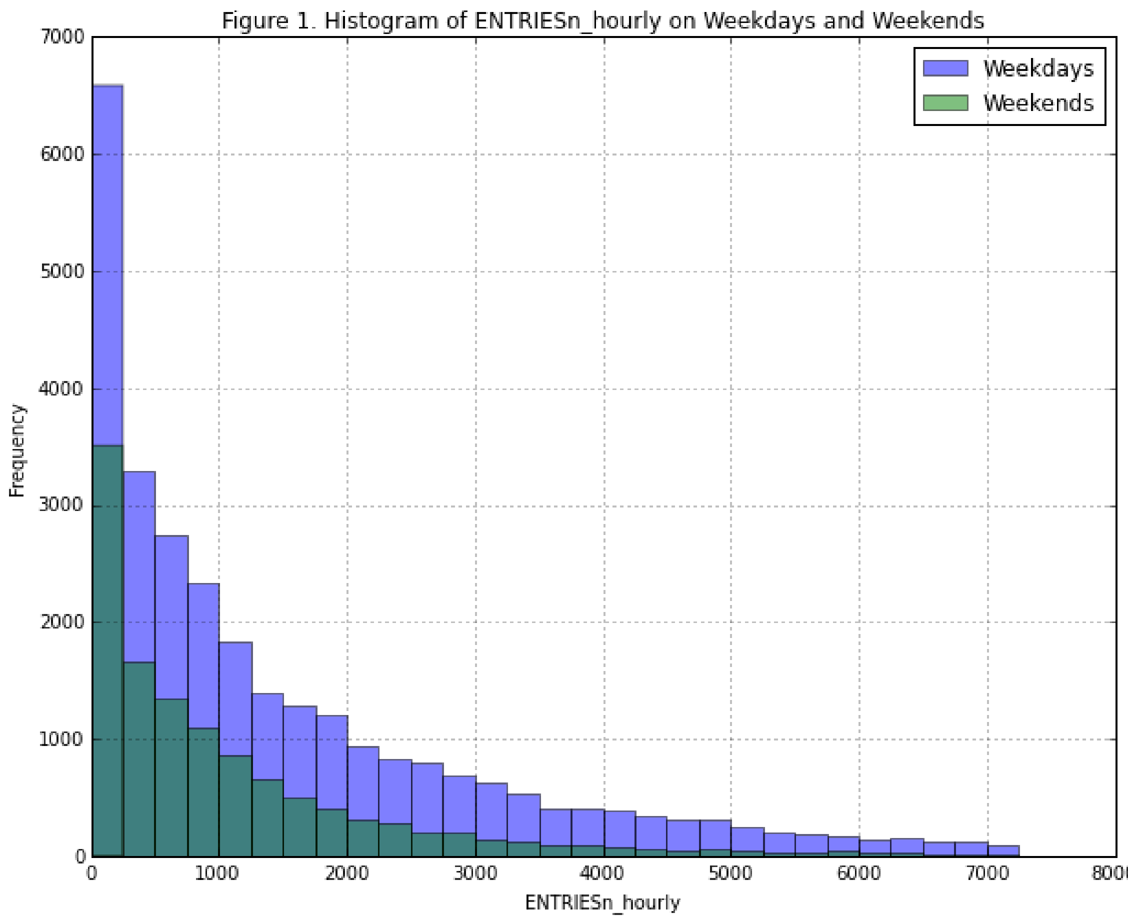
```
Mann-Whitney U Test Output

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-38-27c622b97396> in <module>()
      1 #Run Mann-Whitney U Comparing Weekdays to Weeken
----> 2 print 'Mann-Whitney U Test Output ',mann_whitney_u_week_weekend
      3 print 'U value : ',mann_whitney_u_week_weekend[0]
      4 print 'One-Tailed p-value : ',mann_whitney_u_week_weekend[1]
      5 print 'Two-Tailed p-value : ',(mann_whitney_u_week_weekend[1]*2)

NameError: name 'mann_whitney_u_week_weekend' is not defined
```

In [39]:
```
binLabels = []
for x in range(30):
    binLabels.append(x*250)
plt.figure()
plt.title('Figure 1. Histogram of ENTRIESn_hourly on Weekdays and Weekends')
plt.xlabel('ENTRIESn_hourly')
plt.ylabel('Frequency')
weekdays['ENTRIESn_hourly'].hist(bins=binLabels, alpha=0.5,label=['Weekdays'])
weekends['ENTRIESn_hourly'].hist(bins=binLabels,alpha=0.5,label=['Weekends'])
plt.legend(['Weekdays','Weekends'])
```

Out[39]: <matplotlib.legend.Legend at 0x10f2e34d0>



Linear Regression and R-Squared Functions

These are from completed problems on Udacity Problem set 3

In [40]:
```
# From Udacity PS3.5
def linear_regression(features, values):
    """
    Perform linear regression given a data set with an arbitrary number of featu
res.

    This can be the same code as in the lesson #3 exercise.
    """

    lr = linear_model.LinearRegression(fit_intercept=True)
    lr.fit(features,values)
```

```
    lr.fit(features,values)
    intercept = lr.intercept_
    params = lr.coef_
    return intercept, params


def predictions(dataframe,features):
    '''
    The NYC turnstile data is stored in a pandas dataframe called weather_turnst
ile.
    Using the information stored in the dataframe, let's predict the ridership o
f
    the NYC subway using linear regression with gradient descent.

    You can download the complete turnstile weather dataframe here:
    https://www.dropbox.com/s/meyki2wl9xfa7yk/turnstile_data_master_with_weathe
r.csv

    Your prediction should have a R^2 value of 0.40 or better.
    You need to experiment using various input features contained in the datafra
me.
    We recommend that you don't use the EXITSn_hourly feature as an input to the
    linear model because we cannot use it as a predictor: we cannot use exits
    counts as a way to predict entry counts.

    Note: Due to the memory and CPU limitation of our Amazon EC2 instance, we wi
ll
    give you a random subet (~10%) of the data contained in
    turnstile_data_master_with_weather.csv. You are encouraged to experiment wit
h
    this exercise on your own computer, locally. If you do, you may want to comp
lete Exercise
    8 using gradient descent, or limit your number of features to 10 or so, sinc
e ordinary
    least squares can be very slow for a large number of features.

    If you receive a "server has encountered an error" message, that means you a
re
    hitting the 30-second limit that's placed on running your program. Try using
a
    smaller number of features.

    Features is a dataframe slice containing variables of interest i.e.
    dataframe[['rain', 'meantempi']]
    '''

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

    # Values
    values = dataframe['ENTRIESn_hourly']

    # Get the numpy arrays
    features_array = features.values
    values_array = values.values

    # Perform linear regression
    intercept, params = linear_regression(features_array, values_array)

    predictions = intercept + np.dot(features_array, params)
    return predictions
```

In [41]:
```
def compute_r_squared(data, predictions):
    '''
    In exercise 5, we calculated the R^2 value for you. But why don't you try an
d
    and calculate the R^2 value yourself.

    Given a list of original data points, and also a list of predicted data poin
ts,
    write a function that will compute and return the coefficient of determinati
on (R^2)
    for this data.  numpy.mean() and numpy.sum() might both be useful here, but
    not necessary.

    Documentation about numpy.mean() and numpy.sum() below:
    http://docs.scipy.org/doc/numpy/reference/generated/numpy.mean.html
    http://docs.scipy.org/doc/numpy/reference/generated/numpy.sum.html
    '''
    def square(x):
        return x**2
    squareArray = np.vectorize(square)
    dataAverage = np.mean(data)

    ssRes = np.sum(squareArray(data-predictions))

    ssTot = np.sum(squareArray(data-dataAverage))

    r_squared=1-(ssRes/ssTot)

    return r_squared
```

Prediction based on rain and meantemp

Uses linear regression with gradient descent from sklearn.

In [42]:
```
def normalize_features(features):
    '''
    Returns the means and standard deviations of the given features, along with
a normalized feature
    matrix.
    '''
    means = np.mean(features, axis=0)
    std_devs = np.std(features, axis=0)
    normalized_features = (features - means) / std_devs
    return means, std_devs, normalized_features

def recover_params(means, std_devs, norm_intercept, norm_params):
    '''
    Recovers the weights for a linear model given parameters that were fitted us
ing
    normalized features. Takes the means and standard deviations of the original
    features, along with the intercept and parameters computed using the normali
zed
    features, and returns the intercept and parameters that correspond to the or
iginal
    features.
    '''
    intercept = norm_intercept - np.sum(means * norm_params / std_devs)
    params = norm_params / std_devs
    return intercept, params
```

```
    return intercept, params

def linear_regression_gd(features, values):
    """
    Perform linear regression given a data set with an arbitrary number of featu
res.
    """
    lr = linear_model.SGDRegressor(n_iter=10)
    lr.fit(features,values)
    intercept = lr.intercept_
    params = lr.coef_

    return intercept, params



def predictions_gd(dataframe,features):
    '''
    The NYC turnstile data is stored in a pandas dataframe called weather_turnst
ile.
    Using the information stored in the dataframe, let's predict the ridership o
f
    the NYC subway using linear regression with gradient descent.

    You can download the complete turnstile weather dataframe here:
    https://www.dropbox.com/s/meyki2wl9xfa7yk/turnstile_data_master_with_weathe
r.csv

    Your prediction should have a R^2 value of 0.40 or better.
    You need to experiment using various input features contained in the datafra
me.
    We recommend that you don't use the EXITSn_hourly feature as an input to the
    linear model because we cannot use it as a predictor: we cannot use exits
    counts as a way to predict entry counts.

    Note: Due to the memory and CPU limitation of our Amazon EC2 instance, we wi
ll
    give you a random subet (~50%) of the data contained in
    turnstile_data_master_with_weather.csv. You are encouraged to experiment wit
h
    this exercise on your own computer, locally.

    If you receive a "server has encountered an error" message, that means you a
re
    hitting the 30-second limit that's placed on running your program. Try using
a
    smaller number of features or fewer iterations.
    '''
    # Add UNIT to features using dummy variables
    dummy_units = pd.get_dummies(dataframe['UNIT'], prefix='unit')
    features = features.join(dummy_units)

    # Values
    values = dataframe['ENTRIESn_hourly']

    # Get the numpy arrays
    features_array = features.values
    values_array = values.values

    means, std_devs, normalized_features_array = normalize_features(features_arr
ay)

    # Perform gradient descent
    norm_intercept, norm_params = linear_regression_gd(normalized_features_arra
```

```
    norm_intercept, norm_params = linear_regression_gd(normalized_features_arra
y, values_array)

    intercept, params = recover_params(means, std_devs, norm_intercept, norm_par
ams)

    predictions = intercept + np.dot(features_array, params)
    # The following line would be equivalent:
    # predictions = norm_intercept + np.dot(normalized_features_array, norm_para
ms)

    return predictions
```

Linear Regression Runs with R-Squared Values

In [43]:

```
#Run Linear Regression functions and calculate R-Squared values comparing predic
ted ENTRIESn_hourly to actual
features_1 = improved_dataset[['rain']]
features_2 = improved_dataset[['weekday']]
features_3 = improved_dataset[['hour']]
features_4 = improved_dataset[['fog']]
features_5 = improved_dataset[['latitude','longitude']]
features_best = improved_dataset[['rain','weekday','fog','hour']]
features_best_nofog = improved_dataset[['rain','weekday','hour']]

predictions_out_1 = predictions(improved_dataset,features_1)
predictions_out_1_gd= predictions_gd(improved_dataset,features_1)
prediction_1_r_squared = compute_r_squared(improved_dataset['ENTRIESn_hourly'],p
redictions_out_1)
prediction_1_r_squared_gd = compute_r_squared(improved_dataset['ENTRIESn_hourl
y'],predictions_out_1_gd)

predictions_out_2 = predictions(improved_dataset,features_2)
predictions_out_2_gd = predictions_gd(improved_dataset,features_2)
prediction_2_r_squared = compute_r_squared(improved_dataset['ENTRIESn_hourly'],p
redictions_out_2)
prediction_2_r_squared_gd = compute_r_squared(improved_dataset['ENTRIESn_hourl
y'],predictions_out_2_gd)

predictions_out_3 = predictions(improved_dataset,features_3)
predictions_out_3_gd = predictions_gd(improved_dataset,features_3)
prediction_3_r_squared = compute_r_squared(improved_dataset['ENTRIESn_hourly'],p
redictions_out_3)
prediction_3_r_squared_gd = compute_r_squared(improved_dataset['ENTRIESn_hourl
y'],predictions_out_3_gd)

predictions_out_4 = predictions(improved_dataset,features_4)
predictions_out_4_gd = predictions_gd(improved_dataset,features_4)
prediction_4_r_squared = compute_r_squared(improved_dataset['ENTRIESn_hourly'],p
redictions_out_4)
prediction_4_r_squared_gd = compute_r_squared(improved_dataset['ENTRIESn_hourl
y'],predictions_out_4_gd)

predictions_out_5 = predictions(improved_dataset,features_5)
predictions_out_5_gd = predictions_gd(improved_dataset,features_5)
prediction_5_r_squared = compute_r_squared(improved_dataset['ENTRIESn_hourly'],p
redictions_out_5)
prediction_5_r_squared_gd = compute_r_squared(improved_dataset['ENTRIESn_hourl
y'],predictions_out_5_gd)
```

```
predictions_out_best = predictions(improved_dataset,features_best)
predictions_out_best_gd = predictions_gd(improved_dataset,features_best)
prediction_best_r_squared = compute_r_squared(improved_dataset['ENTRIESn_hourl
y'],predictions_out_best)
prediction_best_r_squared_gd = compute_r_squared(improved_dataset['ENTRIESn_hour
ly'],predictions_out_best_gd)

predictions_out_best_nofog = predictions(improved_dataset,features_best_nofog)
predictions_out_best_gd_nofog = predictions_gd(improved_dataset,features_best_no
fog)
prediction_best_r_squared_nofog = compute_r_squared(improved_dataset['ENTRIESn_h
ourly'],predictions_out_best_nofog)
prediction_best_r_squared_gd_nofog = compute_r_squared(improved_dataset['ENTRIES
n_hourly'],predictions_out_best_gd_nofog)

print 'Features : Rain'
print 'R-Squared : ',prediction_1_r_squared
print 'R-Squared with Gradient Descent : ',prediction_1_r_squared_gd,'\n'

print 'Features : Weekday'
print 'R-Squared : ',prediction_2_r_squared
print 'R-Squared with Gradient Descent : ',prediction_2_r_squared_gd,'\n'

print 'Features : Hour'
print 'R-Squared : ',prediction_3_r_squared
print 'R-Squared with Gradient Descent : ',prediction_3_r_squared_gd,'\n'

print 'Features : Fog'
print 'R-Squared : ',prediction_4_r_squared
print 'R-Squared with Gradient Descent : ',prediction_4_r_squared_gd,'\n'

print 'Features : Latitude, Longitude'
print 'R-Squared : ',prediction_5_r_squared
print 'R-Squared with Gradient Descent : ',prediction_5_r_squared_gd,'\n'

print 'Features : Rain, Weekday, fog, hour'
print 'R-Squared : ',prediction_best_r_squared
print 'R-Squared with Gradient Descent : ',prediction_best_r_squared_gd,'\n'

print 'Features : Rain, Weekday, hour'
print 'R-Squared : ',prediction_best_r_squared_nofog
print 'R-Squared with Gradient Descent : ',prediction_best_r_squared_gd_nofo
g,'\n'
```

```
Features : Rain
R-Squared :  0.375820464297
R-Squared with Gradient Descent :  0.370503151513

Features : Weekday
R-Squared :  0.397283558908
R-Squared with Gradient Descent :  0.379664106411

Features : Hour
R-Squared :  0.45883641483
R-Squared with Gradient Descent :  0.400441722831

Features : Fog
R-Squared :  0.375292766898
R-Squared with Gradient Descent :  0.368362019853

Features : Latitude, Longitude
R-Squared :  0.375210459854
R-Squared with Gradient Descent :  0.368468807703

Features : Rain, Weekday, fog, hour
R-Squared :  0.481783370977
R-Squared with Gradient Descent :  0.408361043688

Features : Rain, Weekday, hour
R-Squared :  0.481396426979
R-Squared with Gradient Descent :  0.408394170194
```

In [44]:
```
#Latitude and Longitude evaluated for linear regression seperately
''' A better way to input location into linear regression would be as follows
Group dataset by station
Sort stations from smallest to largest means.
Use numerical rank as new value for stations.
Normalize data
Use new numerical rank in linear regression model.
'''
features_6 = improved_dataset[['latitude']]
features_7 = improved_dataset[['longitude']]

predictions_out_6 = predictions(improved_dataset,features_6)
predictions_out_6_gd = predictions_gd(improved_dataset,features_6)
prediction_6_r_squared = compute_r_squared(improved_dataset['ENTRIESn_hourly'],p
redictions_out_6)
prediction_6_r_squared_gd = compute_r_squared(improved_dataset['ENTRIESn_hourl
y'],predictions_out_6_gd)

predictions_out_7 = predictions(improved_dataset,features_7)
predictions_out_7_gd = predictions_gd(improved_dataset,features_7)
prediction_7_r_squared = compute_r_squared(improved_dataset['ENTRIESn_hourly'],p
redictions_out_7)
prediction_7_r_squared_gd = compute_r_squared(improved_dataset['ENTRIESn_hourl
y'],predictions_out_7_gd)

print 'Features : Latitude'
print 'R-Squared : ',prediction_6_r_squared
print 'R-Squared with Gradient Descent : ',prediction_6_r_squared_gd,'\n'

print 'Features : Longitude'
print 'R-Squared : ',prediction_7_r_squared
print 'R-Squared with Gradient Descent : ',prediction_7_r_squared_gd,'\n'
```

```
Features : Latitude
R-Squared :  0.375214816677
R-Squared with Gradient Descent :  0.368423603012

Features : Longitude
R-Squared :  0.375215695582
R-Squared with Gradient Descent :  0.368469097282
```

In [45]:
```
def linear_regression_test(features, values):
    """
    Perform linear regression given a data set with an arbitrary number of featu
res.
    """
    lr = linear_model.SGDRegressor(n_iter=10)
    lr.fit(features,values)
    intercept = lr.intercept_
    params = lr.coef_
    print values.shape
    print lr.coef_.shape
    print 'Coefficients :'
    print lr.coef_
    return intercept, params

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

    # Values
    values = dataframe['ENTRIESn_hourly']

    # Get the numpy arrays
    features_array = features.values
    values_array = values.values

    means, std_devs, normalized_features_array = normalize_features(features_arr
ay)

    # Perform gradient descent
    norm_intercept, norm_params = linear_regression_test(normalized_features_arr
ay, values_array)

    intercept, params = recover_params(means, std_devs, norm_intercept, norm_par
ams)

    # The following line would be equivalent:
    # predictions = norm_intercept + np.dot(normalized_features_array, norm_para
ms)

    return params

def get_coefficients_ols(dataframe,features):

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

    # Values
    values = dataframe['ENTRIESn_hourly']

    # Get the numpy arrays
    features_array = features.values
    values_array = values.values

    # Perform linear regression
    intercept, params = linear_regression(features_array, values_array)

    return params
```

Coefficient Output

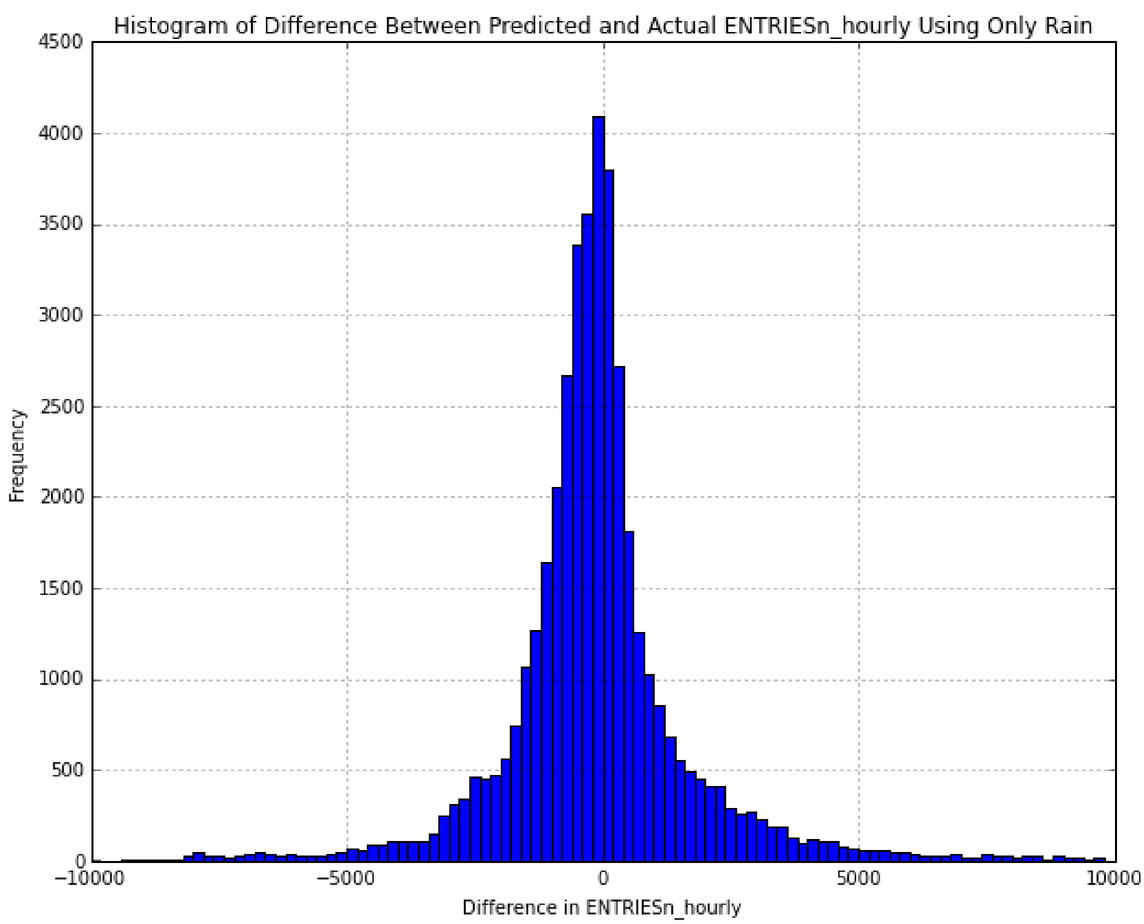
In [ ]:
```
model_coefficients_gd = get_coefficients_gd(improved_dataset,features_best_nofo
g)
model_coefficients_ols = get_coefficients_ols(improved_dataset,features_best_nof
og)
#print model_coefficients_ols
#print model_coefficients_gd
```

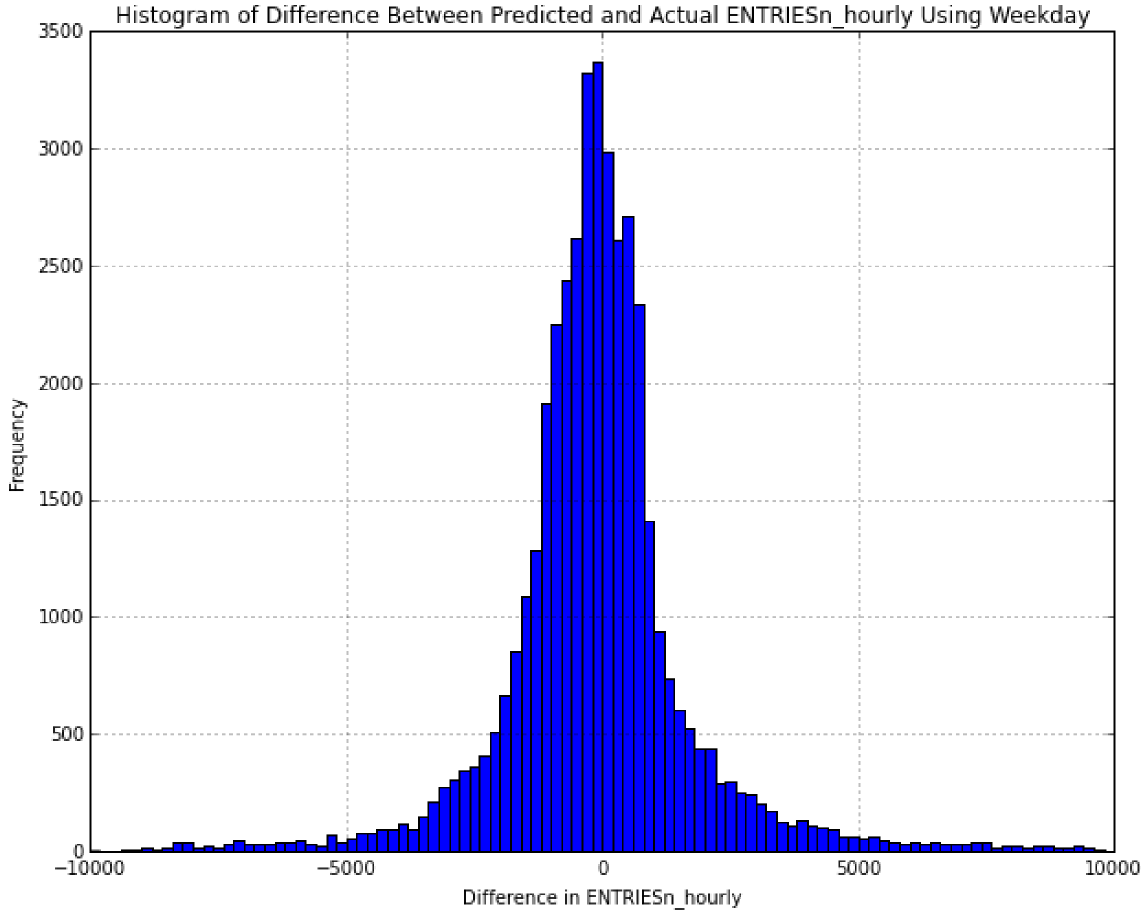
```
(42649,)
(243,)
Coefficients :
[ -29.56435321  199.25262378  240.50039783  -87.40544791  -67.81968833
  -66.79618424  -61.01323903  -77.50707251  -75.89859894  -79.84997656
  350.72037281  444.45482258   47.0542978   -65.31362109  137.25870273
  382.64592381   75.89659556  260.69099991  160.20736384  496.55101402
  263.86618926   92.23473263  195.80117557   56.16575013  301.3166379
   29.57551781  148.4084839   133.25475088  395.43276941  -57.98455828
   36.60361524  -77.75032228  -74.90825511 -109.23539939  -58.07837874
  -24.66743801   75.22386591  -77.94226276   56.05039899  165.1414673
  395.28052366   36.09014962  114.50661266  173.91759816  -53.86471435
   68.59336605  -34.96899617  408.55269955  -38.41406716  202.0065343
  -87.34057071  -52.55637957  -76.25339958  -80.75862736   45.5225721
  -50.53484037  -69.23801519  -64.44303949  -98.41503017  -56.78461664
  -88.4219609   -51.74383596   -6.112638    104.28605523   94.39253818
  -24.27572044   69.52333274  468.25672967   23.00880927   59.91049578
  -42.07133702  -91.29219326  -95.61301889  -53.24844503  -12.50080118
  -30.9324703     8.27737586   17.89033918   32.64101701   80.54556725
    1.76855644   28.15092047  -79.76221933   55.59743797  113.91774377
  -19.80059181  -30.6486407    79.76140055  -41.5452349   -77.41479213
  191.38354882   80.8791877   -11.41220644  -58.91651015  -33.9010425
   73.02720276  -35.06614455    3.5311573   -22.66723869  -18.45525126
   52.30029395  -41.02282739  -65.07274057    8.88180974  154.6663004
   30.01011713   31.80769838   78.60689558   44.65464815  303.99316463
   -0.78912476  -65.56039717  -46.06664608  -54.18309092   26.57199846
  -30.93259775   -2.906383    -23.54260338   19.99681919  -62.1987691
  -44.19442701   30.01031792  -41.92581173  -30.22477479  -28.79734909
   11.41217361   47.23449507  -55.58333192  -76.48415982   53.62727197
   -2.45013824  -33.24383882  -63.2458346    -7.05889409  -60.95181065
  -49.34267011   26.6364816   -31.76619075  -16.06800311  -20.98918132
   30.02053006  -61.34507173  -72.19160055  -56.99122683  -34.72891336
  -38.35936407  -66.45286161  -69.05034315  -48.57012793  -34.6911645
  -23.92036976  -74.42440069   47.94034266  -26.22044917  -59.52638598
   40.25247999  -52.12988181   75.25176163  -65.99265804  -19.82858503
  -66.58459477  -59.79814098  -83.90235322   74.61754695  -33.79999527
  -67.46386903  -29.28088643  -44.02366765  -65.2830277    62.44424668
  -56.35052384  -48.47598767   18.91686885   -8.22050869  -42.50047895
  -22.18141704   -4.94111904  -69.48920198  -95.38101479  -69.87959798
  -57.4606695   -58.73321149  -47.74631968  -67.80095706  -78.17936829
  -17.68051381  -76.90859817  -45.63758829  -22.69616977  -67.1155389
  -76.06962398  -67.30458777  -56.2793822   -21.77313268   -9.41226761
  -57.45352009  -62.74911827  -63.55641366   -5.19755543  -55.68522877
  -70.06515674   28.80807893  -27.47441386  -51.73670807  -76.89787292
  -46.64955441  -54.33321752  -16.46220056  -74.22040745  -56.11682781
  -92.13874994  -70.66860994  -30.23695724  -42.22264044   -2.18024681
  -23.0480638   -79.39112473  -48.09893855  -69.85496975  -92.33838537
  -86.10005596  -98.21498984  -70.16749568  -69.76816362  -71.35117601
  -30.17413625  -90.9746132   -87.52963674  -36.74100654  -77.52986326
  -64.0194032   -51.18561329  -50.02455891  -58.76302677  -44.43092366
  -72.57323665  -25.09497915   -5.3130838   -89.99032595  -91.21824883
  -86.99784948  -65.86052717 -100.54763556]
```

Histograms of Difference Between Predicted and Actual ENTRIESn_hourly

In [ ]:
```
binLabels = []
for x in range(-50,50):
    binLabels.append(x*200)
plt.figure()
plt.title('Histogram of Difference Between Predicted and Actual ENTRIESn_hourly Using Only Rain')
plt.xlabel('Difference in ENTRIESn_hourly')
plt.ylabel('Frequency')
(improved_dataset['ENTRIESn_hourly'] - predictions_out_1).hist(bins = binLabels)
plt.figure()
plt.title('Histogram of Difference Between Predicted and Actual ENTRIESn_hourly Using Weekday')
plt.xlabel('Difference in ENTRIESn_hourly')
plt.ylabel('Frequency')
(improved_dataset['ENTRIESn_hourly'] - predictions_out_2).hist(bins = binLabels)
plt.figure()
plt.title('Histogram of Difference Between Predicted and Actual ENTRIESn_hourly Using Hour')
plt.xlabel('Difference in ENTRIESn_hourly')
plt.ylabel('Frequency')
(improved_dataset['ENTRIESn_hourly'] - predictions_out_3).hist(bins = binLabels)
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x10f96e350>





Residual Plot Using OLS Linear Regression and Linear Regression With Gradient Descent

Input Variables are: Rain, Weekday, Fog, and Hour

In [ ]:
```
plt.figure()
plt.title('Histogram of Difference Between Predicted and Actual ENTRIESn_hourly Using Rain, Weekday, and Hour Using OLS')
plt.xlabel('Difference in ENTRIESn_hourly')
plt.ylabel('Frequency')
(improved_dataset['ENTRIESn_hourly'] - predictions_out_best).hist(bins = binLabels)
plt.figure()
plt.title('Histogram of Difference Between Predicted and Actual ENTRIESn_hourly Using Rain, Weekday, and Hour using Gradient Descent')
plt.xlabel('Difference in ENTRIESn_hourly')
plt.ylabel('Frequency')
(improved_dataset['ENTRIESn_hourly'] - predictions_out_best_gd).hist(bins = binLabels)
```

Kruskall Wallis Test

The below output shows the output for the scipy Kruskall Wallace (http://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.stats.mstats.kruskalwallis.html) test which is testing for a significant difference of riders between all days of the week. The output is the H-statistic and the p-value.

In [ ]:
```
kruskallwallis_days_of_week=scipy.stats.mstats.kruskalwallis(monday['ENTRIESn_hourly'],tuesday['ENTRIESn_hourly'],wednesday['ENTRIESn_hourly'],thursday['ENTRIESn_hourly'],friday['ENTRIESn_hourly'],saturday['ENTRIESn_hourly'],sunday['ENTRIESn_hourly'])
print kruskallwallis days of week
```

In [ ]: