# Analyzing the NYC Subway Dataset

# Intro to Data Science: Final Project 1, Part 2

# (Short Questions)

## Data Exploration Supplement

Austin J. Alexander

---

## Import Directives and Initial DataFrame Creation

In [40]:
```
import inflect # for string manipulation
import numpy as np
import pandas as pd
import scipy as sp
import scipy.stats as st
import matplotlib.pyplot as plt
%matplotlib inline

filename = '/Users/excalibur/py/nanodegree/intro_ds/final_project/i
mproved-dataset/turnstile_weather_v2.csv'

# import data
data = pd.read_csv(filename)
```

## Initial Data Exploration

### Data Shape

In [41]:
```
print "SHAPE: " + str(data.shape)
data.head(1)
```

```
SHAPE: (42649, 27)
```

Out[41]:

| | UNIT | DATEn | TIMEn | ENTRIESn | EXITSn | ENTRIESn_hourly | EXITSn_hourly |
|---|---|---|---|---|---|---|---|
| **0** | R003 | 05-01-11 | 00:00:00 | 4388333 | 2911002 | 0 | 0 |

1 rows × 27 columns

## Data Types

In [42]:
```
data.dtypes
```

Out[42]:
```
UNIT                  object
DATEn                 object
TIMEn                 object
ENTRIESn               int64
EXITSn                 int64
ENTRIESn_hourly      float64
EXITSn_hourly        float64
datetime              object
hour                   int64
day_week               int64
weekday                int64
station               object
latitude             float64
longitude            float64
conds                 object
fog                    int64
precipi              float64
pressurei            float64
rain                   int64
tempi                float64
wspdi                float64
meanprecipi          float64
meanpressurei        float64
meantempi            float64
meanwspdi            float64
weather_lat          float64
weather_lon          float64
dtype: object
```

### ENTRIESn_hourly Statistics

In [43]:
```
data['ENTRIESn_hourly'].describe()
```

Out[43]:
```
count    42649.000000
mean      1886.589955
std       2952.385585
min          0.000000
25%        274.000000
50%        905.000000
75%       2255.000000
max      32814.000000
Name: ENTRIESn_hourly, dtype: float64
```

# Functions for Getting, Mapping, and Plotting Data

[ N.B. Due to decisions described in the *Unit_Entries* supplement (IntroDS-ProjectOne-Unit_Entries-Supplement.ipynb), in the current analysis, unless otherwise noted, *entries* will refer to a summation of **ENTRIESn_hourly** per **UNIT** (i.e., not, as might be expected, simply alues in the **ENTRIESn** column). ]

In [44]:
```
entries_hourly_by_row = data['ENTRIESn_hourly'].values
```

In [45]:
```
def map_column_to_entries_hourly(column):
    instances = column.values # e.g., longitude_instances = data['l
ongitude'].values

    # reduce
    entries_hourly = {} # e.g., longitude_entries_hourly = {}
    for i in np.arange(len(instances)):
        if instances[i] in entries_hourly:
            entries_hourly[instances[i]] += float(entries_hourly_b
y_row[i])
        else:
            entries_hourly[instances[i]] = float(entries_hourly_b
y_row[i])

    return entries_hourly # e.g., longitudes, entries
```

In [46]:
```
def display_basic_stats(entries_hourly_dict, column1name):
    # e.g, longitude_df = pd.DataFrame(data=longitude_entries_hourl
y.items(), columns=['longitude','entries'])
    df = pd.DataFrame(data=entries_hourly_dict.items(), columns=[co
lumn1name,'entries'])

    p = inflect.engine()
    print "{0} AND THEIR ENTRIES".format(p.plural(column1name.uppe
r()))
    print df.head(3)

    print
    print pd.DataFrame(df['entries']).describe()
    print "{:<7}".format('range') + "{:0<14}".format(str(np.ptp(ent
ries_hourly_dict.values())))

    return df # e.g, longitude_df
```

In [47]:
```
def plot_data(df, column1name, plot_kind, xaxis_labeled):

    p = inflect.engine()
    if xaxis_labeled == True:
        df.plot(x=column1name, y='entries', title="{0} AND THEIR EN
TRIES".format(p.plural(column1name.upper())), kind=plot_kind, alph
a=0.5, color='green')
        plt.xlabel(column1name)
    else:
        df.plot(title="{0} AND THEIR ENTRIES".format(p.plural(colum
n1name.upper())), kind=plot_kind, alpha=0.5, color='green')
        plt.xlabel("{0} row index".format(column1name))

    plt.ylabel('{0} entries'.format(column1name))
    plt.legend(['entries'])
    plt.show()
```

In [48]:
```
def plot_histogram(df, column_name, num_of_bins):
    df[column_name].plot(kind='hist', bins=num_of_bins, alpha=0.5,
color='green')
    plt.ylabel('frequency')
    plt.show()
```

## Function for Basic Statistics

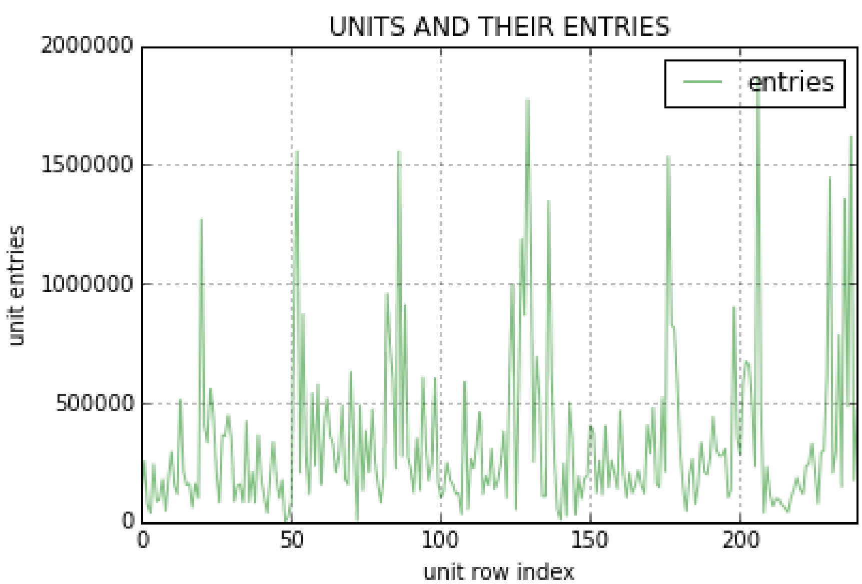
```
In [49]: def describe_samples(sample1, sample2):
             size1, min_max1, mean1, var1, skew1, kurt1 = st.describe(sample
         1)
             size2, min_max2, mean2, var2, skew2, kurt2 = st.describe(sample
         2)

             med1 = np.median(sample1)
             med2 = np.median(sample2)

             std1 = np.std(sample1)
             std2 = np.std(sample2)

             print "Sample 1 (rainy days):\n  min = {0}, max = {1},\n  mean
         = {2:.2f}, median = {3}, var = {4:.2f}, std = {5:.2f}".format(min_m
         ax1[0], min_max1[1], mean1, med1, var1, std1)
             print "Sample 2 (non-rainy days):\n  min = {0}, max = {1},\n  m
         ean = {2:.2f}, median = {3}, var = {4:.2f}, std = {5:.2f}".format(m
         in_max2[0], min_max2[1], mean2, med2, var2, std2)
```

# Non-Weather-Related Data

## Unit Statistics

In [50]:
```
unit_entries_hourly = map_column_to_entries_hourly(data['UNIT'])
unit_df = display_basic_stats(unit_entries_hourly, 'unit')
plot_data(unit_df, 'unit', 'line', False)
```

```
UNITS AND THEIR ENTRIES
   unit  entries
0  R318   112098
1  R319   254531
2  R312    73913

                entries
count       240.000000
mean     335254.895833
std      334849.388932
min           0.000000
25%      131148.000000
50%      221479.500000
75%      409285.750000
max     1868674.000000
range   1868674.000000
```




**Top-5 Units**

In [51]:
```
unit_df.sort(columns='entries', ascending=False).head(5)
```

Out[51]:

| | unit | entries |
|---|---|---|
| **206** | R084 | 1868674 |
| **129** | R022 | 1773372 |
| **237** | R012 | 1618262 |
| **52** | R046 | 1555117 |
| **86** | R055 | 1554806 |

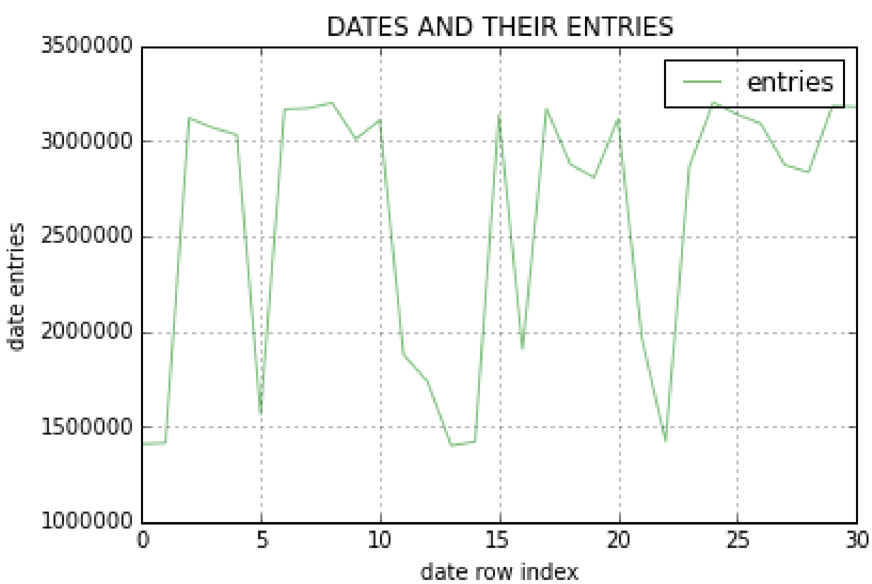
### Unit Summary

Clearly, certain units received more entries than other units.

## Date Statistics

In [52]:
```
date_entries_hourly = map_column_to_entries_hourly(data['DATEn'])
date_df = display_basic_stats(date_entries_hourly, 'date')
plot_data(date_df, 'date', 'line', False)
```

```
DATES AND THEIR ENTRIES
       date   entries
0  05-30-11  1409572
1  05-15-11  1413929
2  05-04-11  3118915

               entries
count        31.000000
mean    2595521.774194
std      710440.834289
min     1400098.000000
25%     1891834.000000
50%     3009536.000000
75%     3137683.000000
max     3201840.000000
range   1801742.000000
```




May 2011 Calendar

*Image from: calendarbar.org*

**Top-5 Dates**

In [53]:
```
date_df.sort(columns='entries', ascending=False).head(5)
```

Out[53]:

| | date | entries |
|---|---|---|
| **24** | 05-12-11 | 3201840 |
| **8** | 05-05-11 | 3199002 |
| **29** | 05-03-11 | 3183128 |
| **30** | 05-06-11 | 3179032 |
| **7** | 05-26-11 | 3172004 |

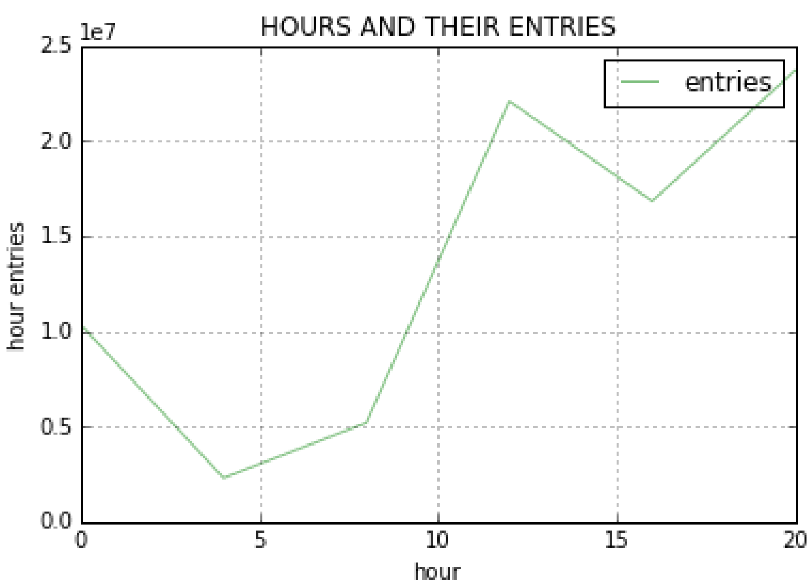
**Date Summary**

Clearly, certain dates received more entries than other dates (the top 5 included 1 Tuesday, 3 Thursdays , and 1 Friday).

## Hour Statistics

In [54]:
```
hour_entries_hourly = map_column_to_entries_hourly(data['hour'])
hour_df = display_basic_stats(hour_entries_hourly, 'hour')
plot_data(hour_df, 'hour', 'line', True)
```

```
HOURS AND THEIR ENTRIES
   hour   entries
0      0  10353167
1      4   2300788
2      8   5198583

               entries
count         6.000000
mean   13410195.833333
std     8863957.086415
min     2300788.000000
25%     6487229.000000
50%    13593103.500000
75%    20772247.000000
max    23690281.000000
range  21389493.00000
```




**Top-5 Hours**

In [55]:
```
hour_df.sort(columns='entries', ascending=False).head(5)
```

Out[55]:

| | hour | entries |
|---|---|---|
| **5** | 20 | 23690281 |
| **3** | 12 | 22085316 |
| **4** | 16 | 16833040 |
| **0** | 0 | 10353167 |
| **2** | 8 | 5198583 |

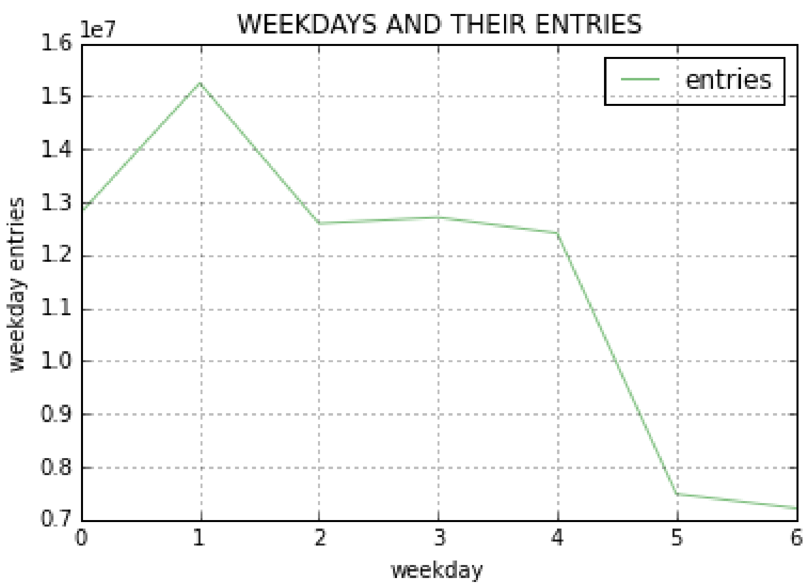
#### Hour Summary

Clearly, certain hours recorded more entries than other hours (the top 2 were, in order, 10:00pm and 12:00pm).

## Weekday Statistics

In [56]:
```
weekday_entries_hourly = map_column_to_entries_hourly(data['day_wee
k'])
weekday_df = display_basic_stats(weekday_entries_hourly, 'weekday')
plot_data(weekday_df, 'weekday', 'line', True)
```

```
WEEKDAYS AND THEIR ENTRIES
   weekday   entries
0        0  12795107
1        1  15246943
2        2  12592691

                entries
count          7.000000
mean   11494453.571429
std     2989933.638739
min     7218706.000000
25%     9949293.000000
50%    12592691.000000
75%    12752124.500000
max    15246943.000000
range  8028237.000000
```




**Top-5 Weekdays**

(0: Mon, 1: Tue, 2: Wed, 3: Thu, 4: Fri, 5: Sat, 6: Sun)

In [57]:
```
weekday_df.sort(columns='entries', ascending=False).head(5)
```

Out[57]:

| | weekday | entries |
|---|---|---|
| **1** | 1 | 15246943 |
| **0** | 0 | 12795107 |
| **3** | 3 | 12709142 |
| **2** | 2 | 12592691 |
| **4** | 4 | 12411679 |

**Weekday Summary**

Clearly, certain weekdays received more entries than other weekdays (the top 2 were, in order, Tuesday and Monday -- somewhat strange given the Date results above).

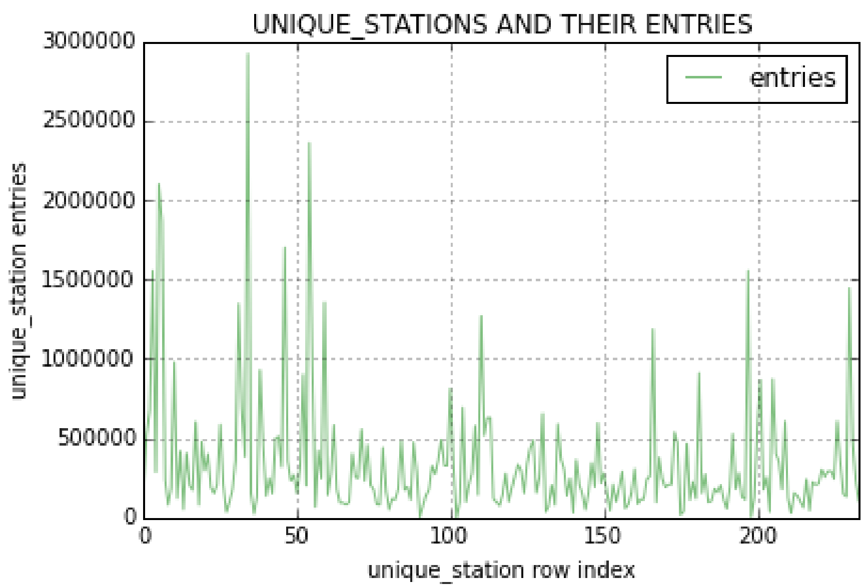
## Station Statistics

[ N.B. Some stations have the same name but different locations, so unique identifiers needed to be created. ]

In [58]:
```
data['unique_station'] = data['station'] + " (" + data['latitud
e'].map(str) + ", " + data['longitude'].map(str) + ")"
```

In [59]:
```
station_entries_hourly = map_column_to_entries_hourly(data['uniqu
e_station'])
station_df = display_basic_stats(station_entries_hourly, 'unique_st
ation')
plot_data(station_df, 'unique_station', 'line', False)
```

```
UNIQUE_STATIONS AND THEIR ENTRIES
                            unique_station  entries
0            176 ST (40.848635, -73.912497)   151399
1  168 ST-BROADWAY (40.840778, -73.940091)   521054
2       57 ST-7 AVE (40.764755, -73.980646)   674799

              entries
count      234.000000
mean    343851.175214
std     393424.158576
min          0.000000
25%     130422.000000
50%     217648.000000
75%     402551.250000
max    2920887.000000
range  2920887.000000
```



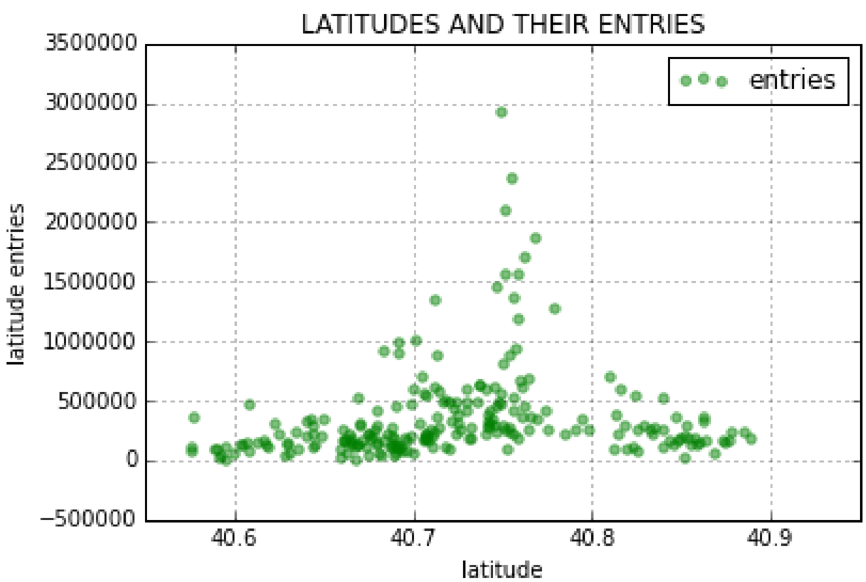

**Station Summary**

Clearly, certain stations received more entries than other stations.

## Latitude Statistics

```
In [60]: latitude_entries_hourly = map_column_to_entries_hourly(data['latitu
de'])
latitude_df = display_basic_stats(latitude_entries_hourly, 'latitud
e')
plot_data(latitude_df, 'latitude', 'scatter', True)
```

```
LATITUDES AND THEIR ENTRIES
    latitude  entries
0  40.852417     7559
1  40.707840   209745
2  40.643982   102508

                entries
count       233.000000
mean     345326.931330
std      393653.267874
min           0.000000
25%      131511.000000
50%      218938.000000
75%      402883.000000
max     2920887.000000
range   2920887.000000
```



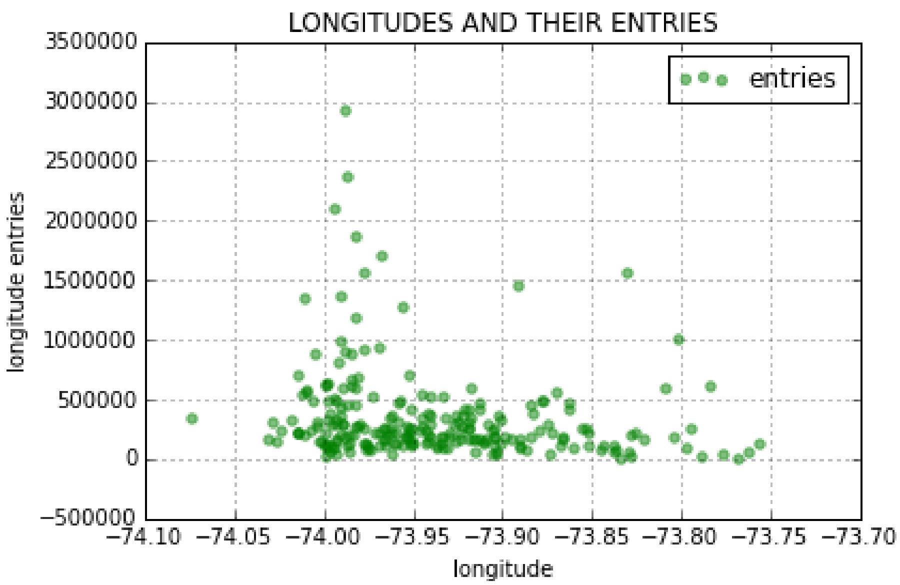

#### Latitude Summary

Clearly, certain latitudes received more entries than other latitudes (in particular, those between latitudes $40.7$ and $40.8$).

## Longitude Statistics

In [61]:
```
longitude_entries_hourly = map_column_to_entries_hourly(data['longi
tude'])
longitude_df = display_basic_stats(longitude_entries_hourly, 'longi
tude')
plot_data(longitude_df, 'longitude', 'scatter', True)
```

```
LONGITUDES AND THEIR ENTRIES
   longitude  entries
0 -73.977417   911174
1 -73.828125   193792
2 -74.014099   694605

               entries
count       234.000000
mean     343851.175214
std      393424.158576
min           0.000000
25%      130422.000000
50%      217648.000000
75%      402551.250000
max     2920887.000000
range   2920887.000000
```




**Longitude Summary**

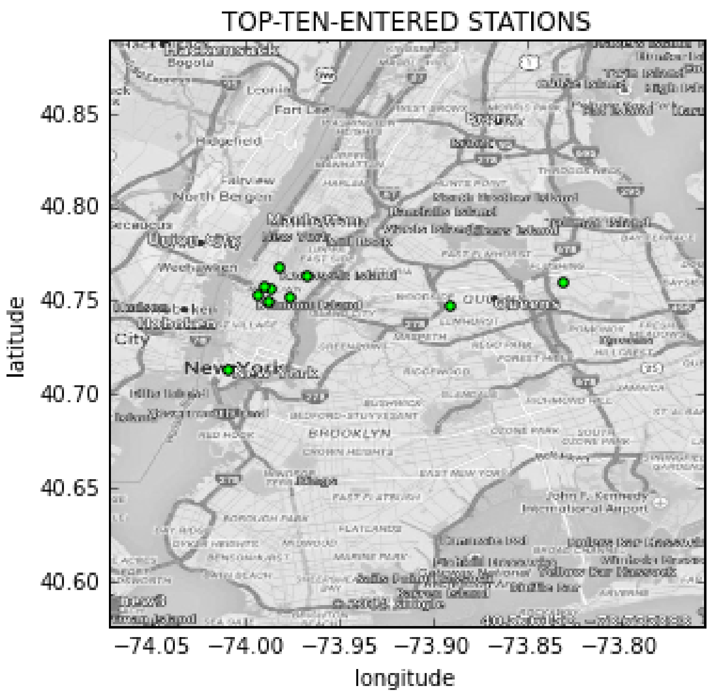
Clearly, certain longitudes received more entries than other longitudes (in particular, those between longitudes $-74.00$ and $-73.95$).

## Combining Station, Latitude, Longitude Data on a Map Layer

The top-ten most-entered stations are the ones that someone familiar with New York City might expect (with a focus on Manhattan Island):

In [62]:
```
station_location = data[['unique_station', 'latitude', 'longitud
e']]
station_location.drop_duplicates(inplace=True)
station_location_entries = pd.merge(station_location, station_df, o
n='unique_station')
station_location_entries.sort(columns='entries', ascending=False, i
nplace=True)
top_ten = station_location_entries.head(10)
print top_ten[['unique_station', 'entries']]

plt.figure(figsize = (5,5))
plt.title('TOP-TEN-ENTERED STATIONS')
plt.xlabel('longitude')
plt.ylabel('latitude')
plt.xlim(station_location_entries['longitude'].min(), station_locat
ion_entries['longitude'].max())
plt.ylim(station_location_entries['latitude'].min(), station_locati
on_entries['latitude'].max())
img = plt.imread('NYmap.png')

plt.scatter(top_ten['longitude'], top_ten['latitude'], color='#00FF
00', edgecolors='black', zorder=1)

plt.imshow(img, zorder=0, extent=[station_location_entries['longitu
de'].min(), station_location_entries['longitude'].max(), station_lo
cation_entries['latitude'].min(), station_location_entries['latitud
e'].max()])

plt.show()
```

```
                              unique_station  entries
14  34 ST-HERALD SQ (40.749533, -73.987899)  2920887
21   42 ST-TIMES SQ (40.755905, -73.986504)  2360981
8    34 ST-PENN STA (40.752247, -73.993456)  2101634
58    59 ST-COLUMBUS (40.76811, -73.981891)  1868674
34    LEXINGTON AVE (40.762796, -73.967686)  1701440
32  42 ST-GRD CNTRL (40.751849, -73.976945)  1555117
38          MAIN ST (40.759578, -73.830056)  1554806
10    ROOSEVELT AVE (40.746655, -73.891361)  1444569
7   42 ST-PA BUS TE (40.757303, -73.989787)  1355492
18  WORLD TRADE CTR (40.712557, -74.009807)  1347727
```




**Station, Latitude, and Longitude Summary**

Again, it seems quite clear that number of entries is dependent on *location, location, location*!

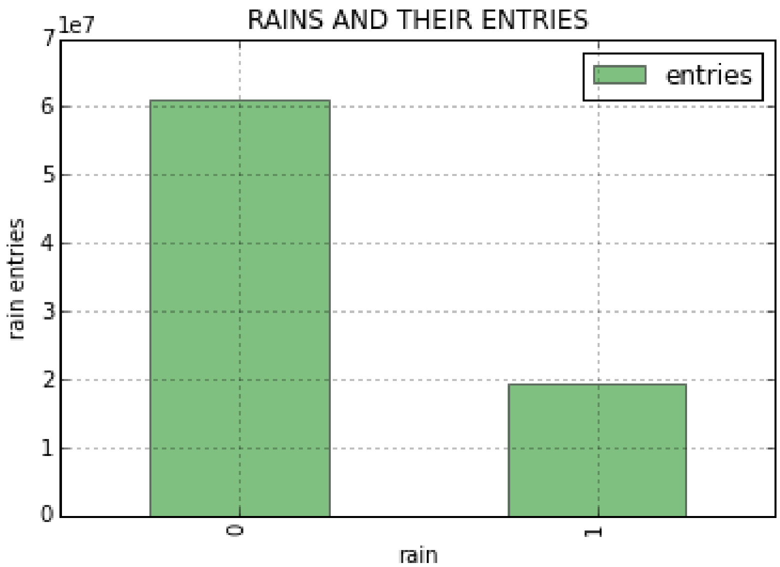
# Weather-Related Data

While the above simple observations seem to indicate that non-weather-related variables have a tremendous impact on transit usage, it would only be fair to explore possible weather-related influences as well (esp. since the project guidelines expect it). Out of the weather-related data that has been made available, the two most-sensible categories to check are the (binary) rain and temperature (in Fahrenheit) variables.

## Rain Statistics

In [63]:
```
rain_entries_hourly = map_column_to_entries_hourly(data['rain'])
rain_df = display_basic_stats(rain_entries_hourly, 'rain')
plot_data(rain_df, 'rain', 'bar', True)
```

```
RAINS AND THEIR ENTRIES
   rain   entries
0     0  61020916
1     1  19440259

               entries
count         2.000000
mean   40230587.500000
std    29401964.530892
min    19440259.000000
25%    29835423.250000
50%    40230587.500000
75%    50625751.750000
max    61020916.000000
range  41580657.00000
```



In [64]:
```
rain_days = data[data['rain'] == 1]
no_rain_days = data[data['rain'] == 0]

print "RAIN DAYS"
print rain_days['ENTRIESn_hourly'].describe()
print
print "NO-RAIN DAYS"
print no_rain_days['ENTRIESn_hourly'].describe()
```

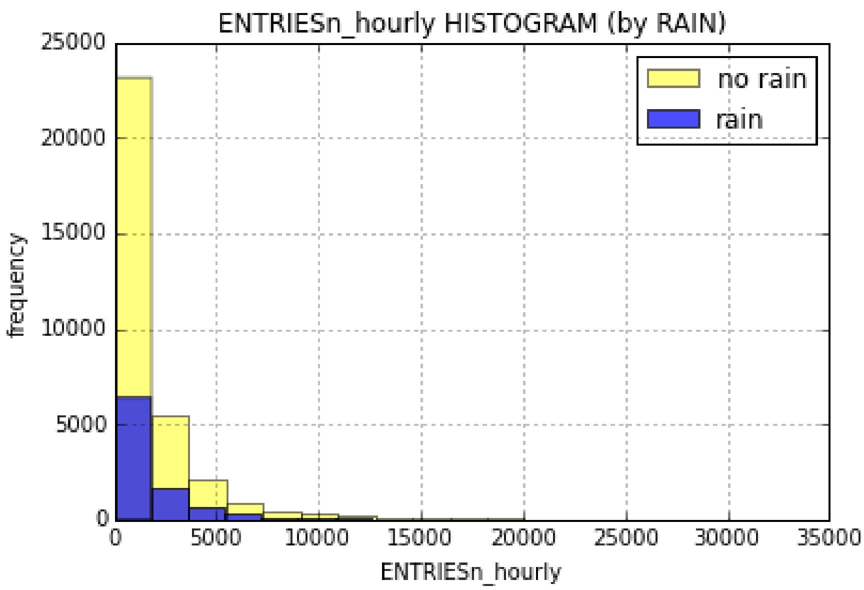
```
RAIN DAYS
count     9585.000000
mean      2028.196035
std       3189.433373
min          0.000000
25%        295.000000
50%        939.000000
75%       2424.000000
max      32289.000000
Name: ENTRIESn_hourly, dtype: float64

NO-RAIN DAYS
count    33064.000000
mean      1845.539439
std       2878.770848
min          0.000000
25%        269.000000
50%        893.000000
75%       2197.000000
max      32814.000000
Name: ENTRIESn_hourly, dtype: float64
```

### ENTRIESn_hourly HISTOGRAM (by RAIN)

In [76]:

```
print "Number of non-rainy days:", no_rain_days['ENTRIESn_hourl
y'].count()
print "Number of rainy days:",  rain_days['ENTRIESn_hourly'].coun
t()

no_rain_days['ENTRIESn_hourly'].plot(kind='hist', bins=18, alph
a=0.5, color='yellow')
rain_days['ENTRIESn_hourly'].plot(kind='hist', bins=18, alpha=0.7,
color='blue')
plt.title('ENTRIESn_hourly HISTOGRAM (by RAIN)')
plt.xlabel('ENTRIESn_hourly')
plt.ylabel('frequency')
plt.legend(['no rain', 'rain'])
plt.show()
```

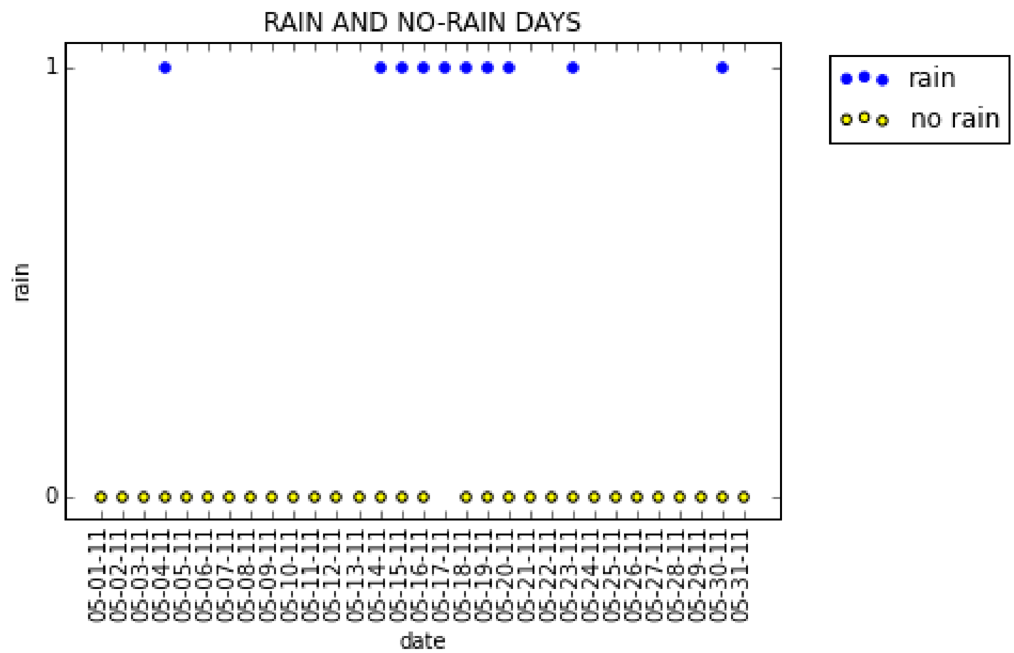
```
Number of non-rainy days: 33064
Number of rainy days: 9585
```



In [66]:

```
date_and_rain = data[['DATEn', 'rain']].drop_duplicates()
date_and_rain.sort(columns='DATEn', inplace=True)
print date_and_rain.head()

dates = data['DATEn'].unique()
rain_dates = date_and_rain[date_and_rain['rain'] == 1]['DATEn'].uni
que()
no_rain_dates = date_and_rain[date_and_rain['rain'] == 0]['DATE
n'].unique()

indices_of_rain_dates = []
for rain_date in rain_dates:
    indices_of_rain_dates.append(np.where(dates == rain_date)[0]
[0])

indices_of_no_rain_dates = []
for no_rain_date in no_rain_dates:
    indices_of_no_rain_dates.append(np.where(dates == no_rain_date)
[0][0])

plt.title('RAIN AND NO-RAIN DAYS')
plt.xticks(np.arange(len(dates)), dates, rotation='vertical')
plt.yticks([0,1])
plt.xlabel('date')
plt.ylabel('rain')

plt.scatter(indices_of_rain_dates, np.ones(len(indices_of_rain_date
s)), color='blue')
plt.scatter(indices_of_no_rain_dates, np.zeros(len(indices_of_no_ra
in_dates)), color='yellow', edgecolors='black')

plt.legend(['rain', 'no rain'], bbox_to_anchor=(1.05, 1), loc=2)
plt.show()
```

```
           DATEn  rain
0       05-01-11     0
5       05-02-11     0
11      05-03-11     0
16      05-04-11     1
32542   05-04-11     0
```

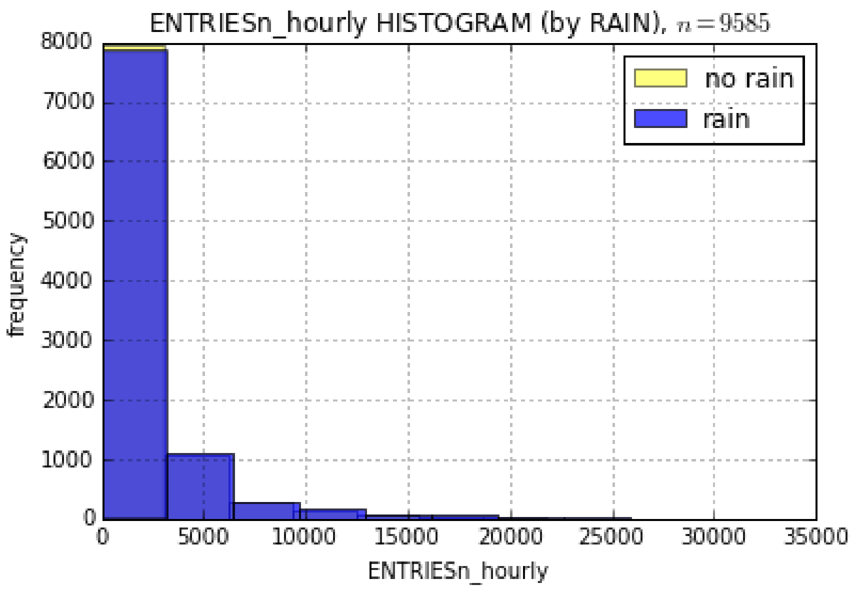



One possible explanation for the difference might be, simply, that there were more non-rainy days in May 2011, as indicated in the graph above. [ N.B. Certain days are reported as being rainy and non-rainy days. For a brief exploration of this phenomenon, see the *Rain* supplement (IntroDS-ProjectOne-Rain-Supplement.ipynb). ]

What if the number of non-rainy days was limited to the total number of rainy days in the data set?

In [67]:
```
random_row_indices = np.random.choice(no_rain_days['ENTRIESn_hourl
y'].index.values, 9585)
```

In [68]:
```
no_rain_days['ENTRIESn_hourly'].loc[random_row_indices].plot(kin
d='hist', bins=10, alpha=0.5, color='yellow')
rain_days['ENTRIESn_hourly'].plot(kind='hist', bins=10, alpha=0.7,
color='blue')
plt.title(r'ENTRIESn_hourly HISTOGRAM (by RAIN), $n = 9585$')
plt.xlabel('ENTRIESn_hourly')
plt.ylabel('frequency')
plt.legend(['no rain', 'rain'])
plt.show()
```




With the number of samples and bins equal, the similarites between the two groups are obvious.

#### Rain Summary

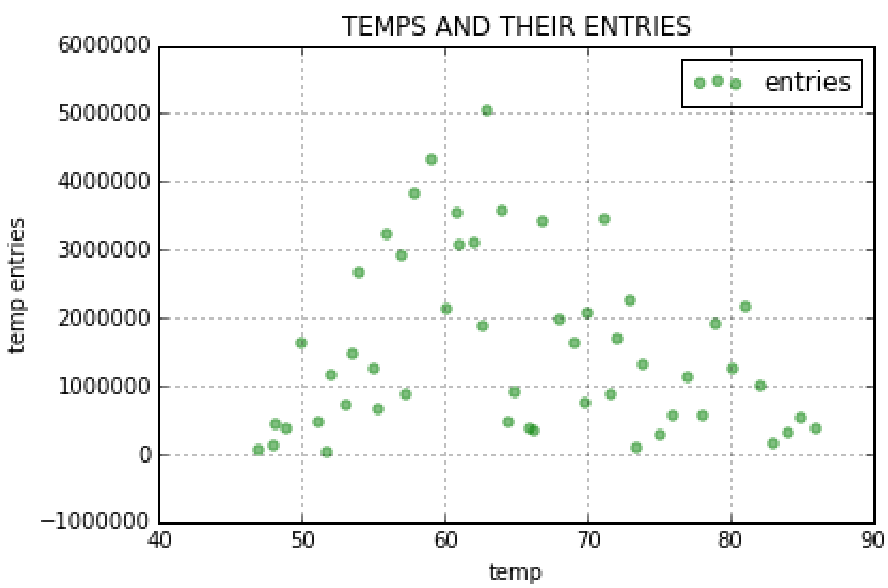
While non-rainy days occur in greater number in this data set (thus, contributing to their higher-frequency counts), the distribution of *ENTRIESn_hourly* for rainy and non-rainy days seems otherwise comparable according to the above histograms.

## Temperature Statistics

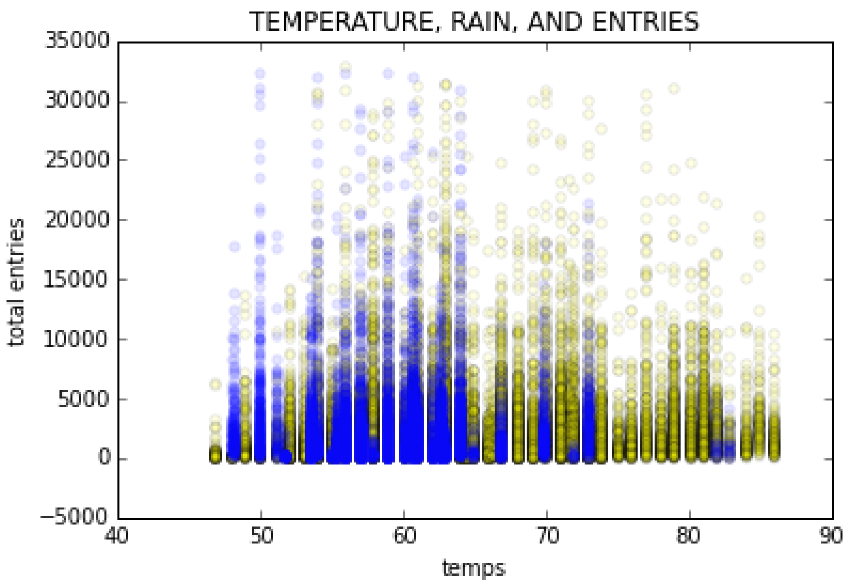
In [69]:

```
temp_entries_hourly = map_column_to_entries_hourly(data['tempi'])
temp_df = display_basic_stats(temp_entries_hourly, 'temp')
plot_data(temp_df, 'temp', 'scatter', True)
```

```
TEMPS AND THEIR ENTRIES
   temp  entries
0  51.1   471715
1  57.9  3831355
2  86.0   362824

                entries
count        52.000000
mean    1547330.288462
std     1276860.663369
min       32822.000000
25%      471789.250000
50%     1203004.000000
75%     2193937.250000
max     5037972.000000
range   5005150.000000
```



In [70]:
```
entries_rain_temp = data[['ENTRIESn_hourly', 'rain', 'tempi']].dro
p_duplicates()
entries_no_rain_temp = entries_rain_temp[entries_rain_temp['rain']
== 0]
entries_rain_temp = entries_rain_temp[entries_rain_temp['rain'] ==
1]

plt.scatter(entries_no_rain_temp['tempi'], entries_no_rain_temp['EN
TRIESn_hourly'], color='yellow', edgecolors='black', alpha=0.1)
plt.scatter(entries_rain_temp['tempi'], entries_rain_temp['ENTRIES
n_hourly'], color='blue', alpha=0.1)
plt.title('TEMPERATURE, RAIN, AND ENTRIES')
plt.xlabel('temps')
plt.ylabel('total entries')
plt.show()
```




**Temperature Summary**

Based on the above scatter plots, there does seem to be some type of relationship between temperatures and number of entries. In general, temperatures between $55$°F and $66$°F received more entries.

Visually-speaking, cold and rainy days seemed to attract approximately the same number of entries as warm and non-rainy (esp. if it's remembered that there were more non-rainy days in the data set).

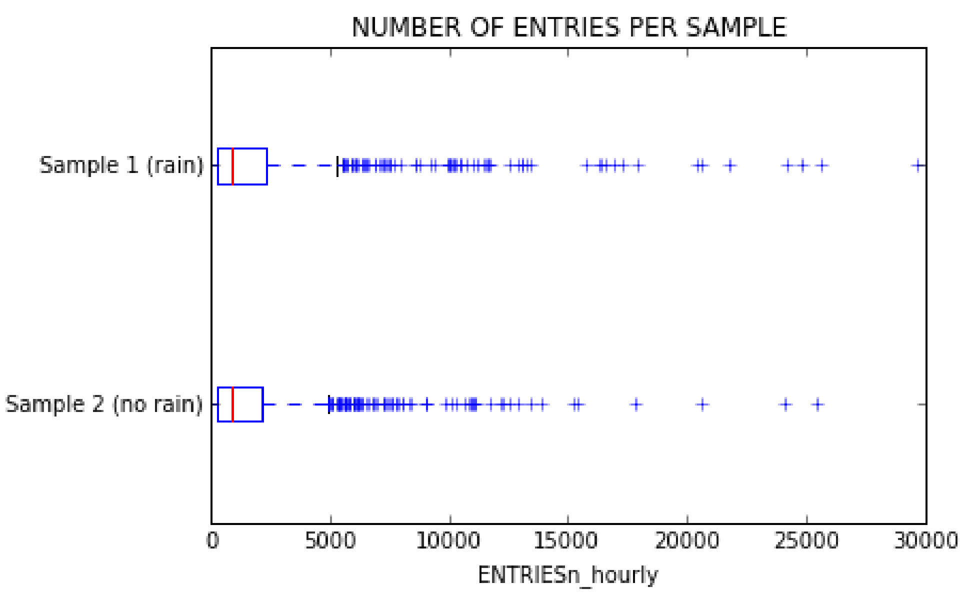
# Preparation for Statistical Tests: Looking for Normality

Taking simple random samples, where $n$: sample size, and $n = 1000$, the following basic statistics reveal themselves.

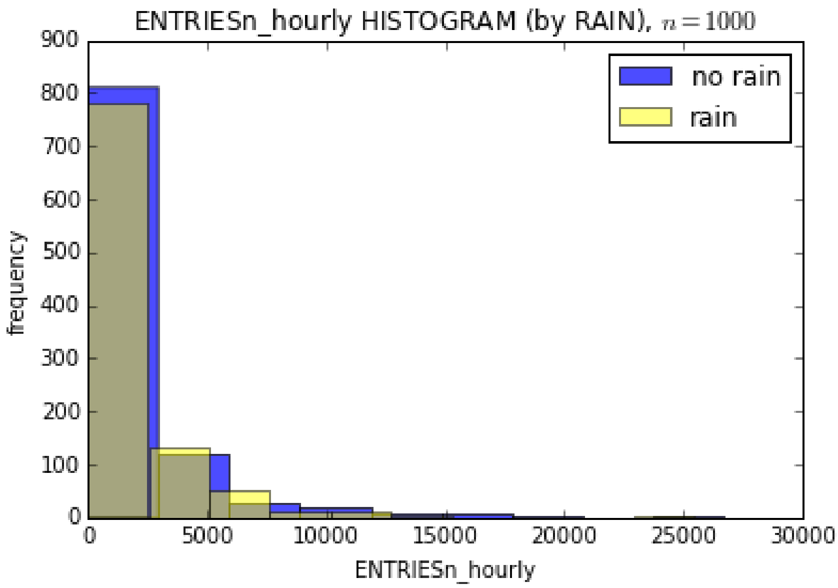
In [71]:
```
n = 1000
sample1 = np.random.choice(rain_days['ENTRIESn_hourly'], size=n, re
place=False)
sample2 = np.random.choice(no_rain_days['ENTRIESn_hourly'], size=n,
replace=False)

describe_samples(sample1,sample2)
```

```
Sample 1 (rainy days):
  min = 0.0, max = 29665.0,
  mean = 1975.33, median = 923.0, var = 10059424.13, std = 3170.07
Sample 2 (non-rainy days):
  min = 0.0, max = 25457.0,
  mean = 1803.53, median = 875.5, var = 6855341.46, std = 2616.96
```

In [72]:
```
plt.boxplot([sample2, sample1], vert=False)
plt.title('NUMBER OF ENTRIES PER SAMPLE')
plt.xlabel('ENTRIESn_hourly')
plt.yticks([1, 2], ['Sample 2 (no rain)', 'Sample 1 (rain)'])

plt.show()
```



In [77]:
```
plt.hist(sample1, color='blue', alpha=0.7, bins=10)
plt.hist(sample2, color='yellow', alpha=0.5, bins=10)
plt.title(r'ENTRIESn_hourly HISTOGRAM (by RAIN), $n = 1000$')
plt.xlabel('ENTRIESn_hourly')
plt.ylabel('frequency')
plt.legend(['no rain', 'rain'])
plt.show()
```




## Testing for Normality

Treating the rain and non-rain days as two independent populations, and although visually apparent from the above histogram, the following statistical test seeks to determine whether rainy and non-rainy day distributions are normal.

The Shapiro-Wilk normality test is a test of the null hypothesis that a sample is from a population with a normal distribution.

The test confirms the visually apparent non-normality with a small-enough sample size; so here, $n = 30$.

A $95\%$ level of confidence would suggest that $95\%$ of samples would produce similar statistical results.

For a $95\%$ level of confidence, the level of significance (i.e., the probability of making a Type I error) $\alpha = (1 - 0.05) \cdot 100\% = 0.05$.

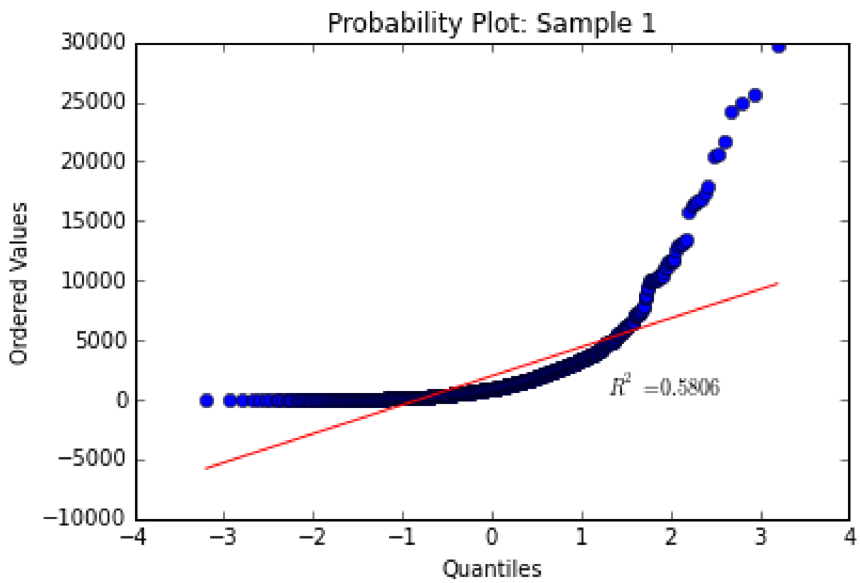
In [74]:
```
n = 30
alpha = 0.05
small_sample1 = np.random.choice(sample1, size=n, replace=False)
small_sample2 = np.random.choice(sample1, size=n, replace=False)
W1, p1 = st.shapiro(small_sample1)
W2, p2 = st.shapiro(small_sample2)

print "p1 < {0}: {1}".format(alpha, (p1 < alpha))
print "p2 < {0}: {1}".format(alpha, (p2 < alpha))
```

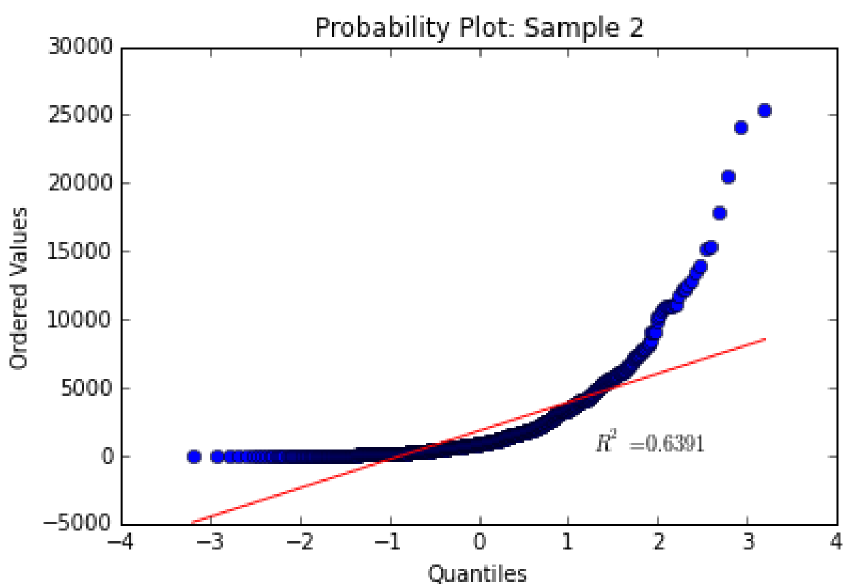
```
p1 < 0.05: True
p2 < 0.05: True
```

In both cases, $p < 0.05$, so the null hypotheses that the samples come from normally distributed populations are rejected.

Moreover, the following probability plots seal the deal (samples from normal distributions would hug the red regression line throughout the plot):

In [75]:
```
st.probplot(sample1, plot=plt)
plt.title('Probability Plot: Sample 1')
plt.show()

st.probplot(sample2, plot=plt)
plt.title('Probability Plot: Sample 2')
plt.show()
```




## Apparent Conclusions

Rainy/Non-rainy days and their number of entries are not normally distributed.