Student and Class Information

Dorothy Kessler
db4871@us.att.com

Udacity Intro to Data Science, UD359: enrolled 1/27/15, coupon UMCCHJT8ARDHIFN
https://www.udacity.com/course/ud359

Final Project: Analyze NY Subway System; 3/14/15
I used both the class interface; and I used the Anaconda 2.1.0 install of python 2.7 with the Spyder 2.3.1 IDE for windows 64 bit.

Analyzing the NYC Subway Dataset

Short Questions Overview

This project consists of two parts. In Part 1 of the project, you should have completed the questions in Problem Sets 2, 3, 4, and 5 in the Introduction to Data Science course.

Questions completed, and all quizzes were completed by Dorothy Kessler.

I used several of the libraries, including but not limited to:
pandas (DataFrame, Series), numpy, json, request, pprint, pandasql, scipt, scipy.stats, ggplot, logging, sys, string, util, re, statsmoels.api, datetime, csv, matplotlib

This document addresses part 2 of the project. Please use this document as a template and answer the following questions to explain your reasoning and conclusion behind your work in the problem sets. You will attach a document with your answers to these questions as part of your final project submission.

My document is submitted, namely DorothyKessler.UD359.FinalProject.pdf

Data Set

For my exploratory analysis, I used turnstile_data_master_with_weather.csv, a regular csv file containing subway data from May 2011; with the data being read into a pandas DataFrame. I continued with that data set for my final Project: https://www.dropbox.com/s/meyki2wl9xfa7yk/turnstile_data_master_with_weather.csv

```
fl="C:/Users/dak/Documents/Udacity.IntroDataSciences/turnstile_data_master_with_weather.csv"
turnstile_weather = pandas.read_csv(fl)
with_rain = turnstile_weather['ENTRIESn_hourly'][turnstile_weather.rain == 1]
without_rain = turnstile_weather['ENTRIESn_hourly'][turnstile_weather.rain == 0]
print "records with rain", len(with_rain)
print "records without rain", len(without_rain)

Result:
records with rain 44104
records without rain 87847
```

We see that the data has:
131,951 rows; columns A through V inclusive
44,104 with rain
87,847 without rain

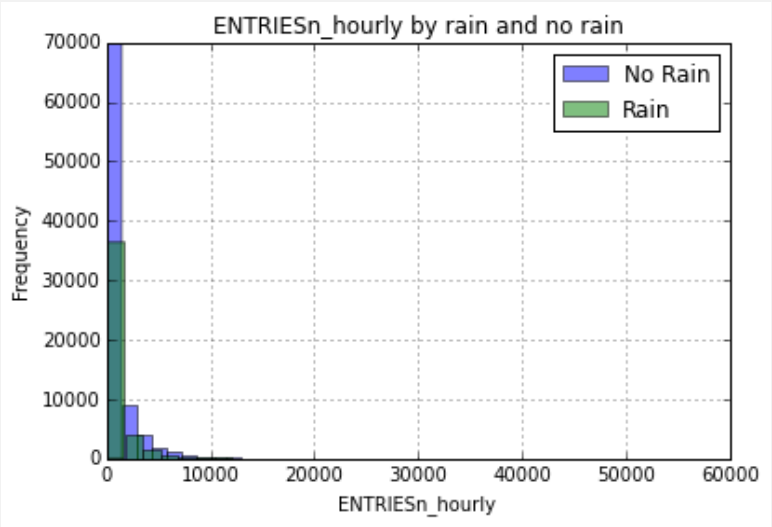
Section 1. Statistical Test

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?

Which statistical test did you use to analyze the NYC subway data?

Before performing any analysis, I took a look at the data to examine the hourly entries in our NYC subway data to determine what distribution the data follows. I plotted two histograms on the same axes to show hourly entries when raining vs not raining (Problem set 3.1)

```
import numpy as np
import pandas
import matplotlib.pyplot as plt
def entries_histogram(turnstile_weather):
    #plt.figure()
    #turnstile_weather['ENTRIESn_hourly'][turnstile_weather.rain ==
1].hist()
    #turnstile_weather['ENTRIESn_hourly'][turnstile_weather.rain==
0].hist()


    # use this logic to get the labels printed
    ax = turnstile_weather[turnstile_weather.rain ==
0]['ENTRIESn_hourly'].plot(kind='hist',
    bins = 30, label='No Rain', alpha = 0.5)

    ax = turnstile_weather[turnstile_weather.rain ==
1]['ENTRIESn_hourly'].plot(kind='hist',
    bins = 30, label='Rain', alpha = 0.5)

    ax.legend()
    ax.set_ylabel('Frequency')
    ax.set_xlabel('ENTRIESn_hourly')
    ax.set_title('ENTRIESn_hourly by rain and no rain')
```



Next I attempted the Shapiro-Wilk Test (Problem set 3.3) which tests the null hypothesis that data was drawn from a normal distribution. Note this gave a warning that the p-value may not be accurate:

```
print "shapiro with rain result", (scipy.stats.shapiro(with_rain))
print "shapiro without rain result", (scipy.stats.shapiro(without_rain))

Result
C:\Users\dak\Anaconda\lib\site-packages\scipy\stats\morestats.py:997:
UserWarning: p-value may not be accurate for N > 5000.
shapiro with rain result (0.4715914726257324, 0.0)
shapiro without rain result (0.47661787271499634, 0.0)
```

So then, I used scipy normaltest. This test calculates a p-value, that, if low enough (standard value is <= 0.05), then it indicates that there is a low chance the distribution is normal; my P critical was 0.05.

```
with_rain = turnstile_weather['ENTRIESn_hourly'][turnstile_weather.rain
== 1]
without_rain =
turnstile_weather['ENTRIESn_hourly'][turnstile_weather.rain == 0]

(k, pvalue) =
scipy.stats.normaltest(turnstile_weather["ENTRIESn_hourly"])
print "pvalue for ENTRIESn_hourly", pvalue
(k, pvalue) = scipy.stats.normaltest(with_rain)
print "pvalue for with_rain", pvalue
(k, pvalue) = scipy.stats.normaltest(without_rain)
print "pvalue for without _rain", pvalue

Result
pvalue for ENTRIESn_hourly 0.0
pvalue for with_rain 0.0
pvalue for without _rain 0.0
```

Thus I concluded that the turnstile_data_master_with_weather.csv data was non-normal. This means I could not use the Welch T-test, because that test assumes a normalized data set.

I continued by using nonparametric statistical procedures, the Mann-Whitney Wilcoxon U test. Non parametric tests apply to observations that are difficult to quantify when you have 2 independent samples. Mann-Whitney U is significant if the u-obtained is LESS THAN or equal to the critical value of U, which for my test was 0.05.

Information on the Mann-Whitney Wilcoxon U Test can be found here.
http://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.mannwhitneyu.html

```
# run the Mann Whitney U test
with_rain = turnstile_weather['ENTRIESn_hourly'][turnstile_weather.rain
== 1]
without_rain =
turnstile_weather['ENTRIESn_hourly'][turnstile_weather.rain == 0]

with_rain_mean = np.mean(with_rain)
without_rain_mean = np.mean(without_rain)
U, p = scipy.stats.mannwhitneyu(with_rain, without_rain)
return with_rain_mean, without_rain_mean, U, p
```

Result

```
(1105.4463767458733,          1090.278780151855,          1924409167.0,
0.024999912793489721)
```

Did you use a one-tail or a two-tail P value?

I used the two-tail P-value, because I did not have a prediction of which group would have a higher frequency. I had no prior assumptions. The direction of the difference could go either way.

A two-tailed test assigns half of the alpha to testing the significance in one direction and half of the alpha to testing significance in the other direction. So using the standard value of 0.05, this means .025 is in each tail of the distribution. I am testing for the possibility of the relationship in both directions.

```
if( p_value * 2 ) <= p-critical
      Reject the null hypothesis
```

Mann Whitney returns a 1-sided p value, and my test calculated.

```
0.024999912793489721
```

To get a two-sided p-value, multiply the returned p-value by 2

```
0.049999825586979442
```

What is the null hypothesis?

The null hypothesis is that the distributions are identical, that there is a 50% probability that an observation from a value randomly selected from one group exceeds an observation randomly selected from the other group. That is to say, an observed discrepancy is due to chance.

Conventional wisdom to formulate the Null Hypothesis H_0

H_0 = The ridership in the NY subway is not likely to be different on rainy days and non-rainy days. (no effect)

Alternative Hypotheses H_a

H_a = Ridership in the NY subway is impacted by rainy days (an effect)
2-sided alternative includes both > and < H_0 value

What is your p-critical value?

My p-critical value is the standard value <= 0.05
α (significance level) conventionally at 0.05
5% change of observing a result as extreme
reject the null even if p is 0.04999999

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.

```
Two unpaired, independent groups
Samples are large
Data measured on a continuous scale, every value is unique
"Distribution free", does not rely on mathematical normal distribution
```

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.

np.mean with rain	np mean no rain	U	2-sided p
1105.4463767458733	1090.278780151855	1924409167.0	.049999825586979442

1.4 What is the significance and interpretation of these results?

If the P value is less than or equal to the p critical value, you can reject the null hypothesis. The smaller the P-value, the stronger the evidence against H_0. Because the two sided p_value is .049, and it is less than 0.05, the null hypothesis is rejected and I accept the Alternative Hypotheses H_a that ridership variations on rainy days are not due to chance.

```
Ha = Ridership in the NY subway is impacted by rainy days
```

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?

Using linear regression to predict how many total riders the New York City subway will have on a given day with a given a variety of factors.

A linear relationship is used to predict the numerical value of *Y* for a given value of *X* using a straight line (*regression line*).If the slope of the line is known, and if we know the *y* intercept of that line, then we can put a value for *X* and predict the value for *Y*. In most cases, Y is the variable we are trying to predict. So we are trying to predict the number on entries per hour, our Y; using variables from our data set X.

1. Gradient descent (as implemented in exercise 3.5)

Gradient Descent to compute coefficients theta used for the predictions.

```
def predictions(dataframe):
    # Select Features (try different features!)
    features = dataframe[['rain', 'precipi', 'Hour', 'meantempi']]

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

    # Values
    values = dataframe['ENTRIESn_hourly']
    m = len(values)

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

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

    # Set values for alpha, number of iterations.
    alpha = 0.1 # please feel free to change this value
    num_iterations = 75 # please feel free to change this value

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

    plot = None
    # Uncomment the next line to see your cost history
    plot = plot_cost_history(alpha, cost_history)
    # Please note, there is a possibility that plotting
    # this in addition to your calculation will exceed
    # the 30 second limit on the compute servers.
    predictions = np.dot(features_array, theta_gradient_descent)
    return predictions, plot
```

We are going to multiply features for X, input variables X_1 to X, by a set of coefficients, theta1 through thetaN. We do this to weight our model. I selected these features to predict Y.

```
features = dataframe[['rain', 'precipi', 'Hour', 'meantempi']]
```

In my predictions function, I setup dummy Units. Categorical data 'R001' to 'R552', which cannot be used in a mathematical formula.

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

Then I normalized the features in the data set.

```
def normalize_features(df):
    """
    Normalize the features in the data set.
    """
    mu = df.mean()
    sigma = df.std()
    if (sigma == 0).any():
        raise Exception("One or more features had the same value for
all samples, and thus could " + "not be normalized. Please do not
include features with only a single value " + "in your model.")
    df_normalized = (df - df.mean()) / df.std()

    return df_normalized, mu, sigma
```

Then I called my gradient descent function,

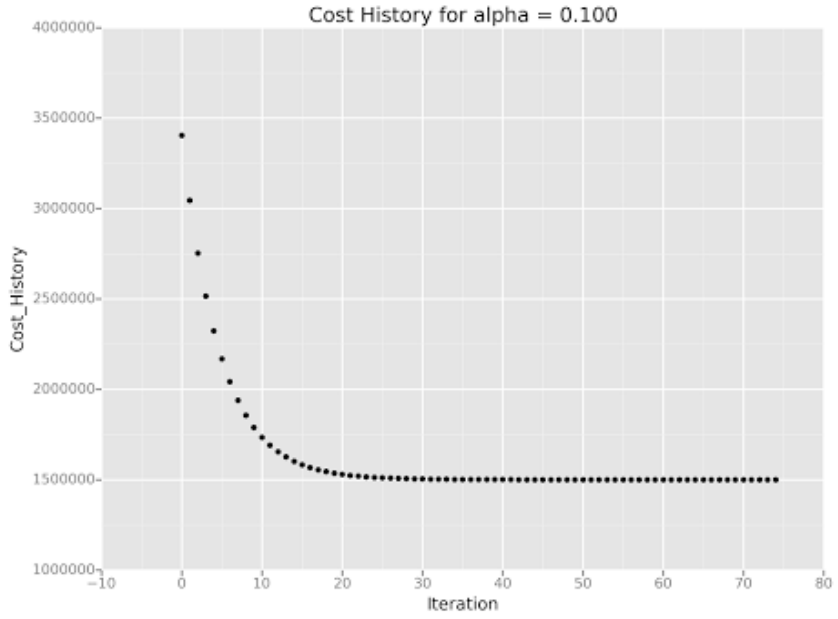
```
def gradient_descent(features, values, theta, alpha, num_iterations):
    cost_history = []
    m = len(values) *1.0
    alpha - alpha * 1.0
```

```
    for i in range(num_iterations):
        cost_history.append( (compute_cost(features, values, theta)) )

        Y = values
        hXi = np.dot(features, theta)
        mySummation = np.dot( (Y - hXi), features )
        theta = theta + ((alpha/m) * (mySummation))

    return theta, pandas.Series(cost_history)
```

We use values of theta to map to a single value. And perform gradient descent given a data set with an arbitrary number of features. My cost function is to measure how our thetas are doing at modeling the data. To minimize cost, I start with theta with zero, and performed a gradient descent 75 times.

```
def compute_cost(features, values, theta):
    """
    Compute the cost function given a set of features / values, and the
values for our thetas.
    """
    m = len(values)
    sum_of_square_errors = np.square(np.dot(features, theta) -
                                                    values).sum()
    cost = sum_of_square_errors / (2*m)
    return cost
```

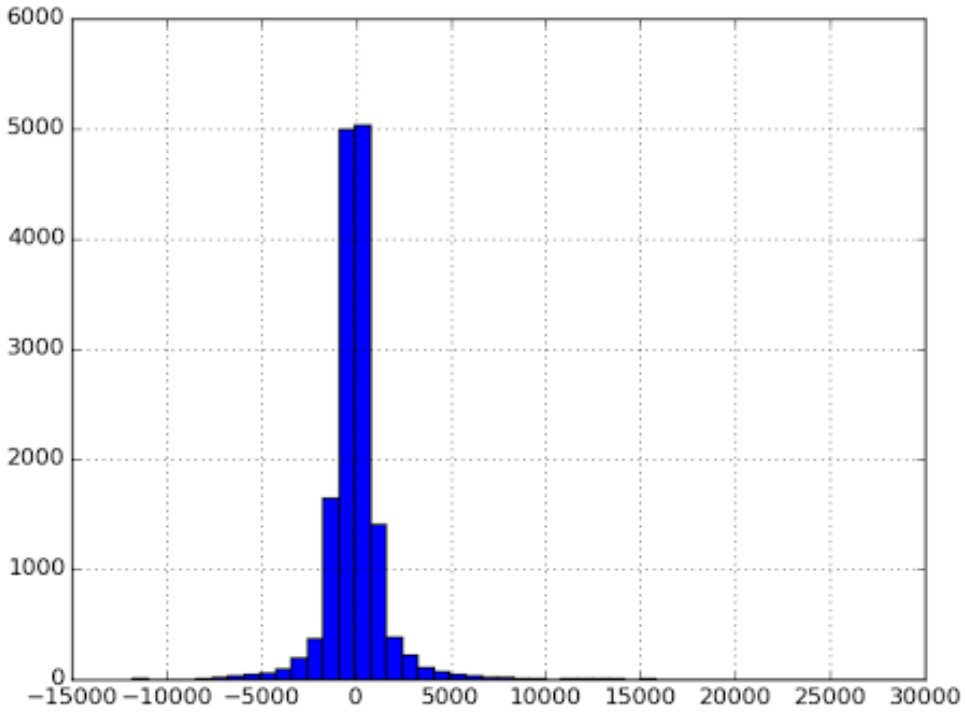
Looking at the plot we can see the cost is going down with each iteration, so we know we are heading toward minimum.



RESIDUALS:

Next, I plotted the residuals. The differences between the predicted and the actual values. If you have a good model, you expect to see the residuals as a normal distribution.

```
def plot_residuals(turnstile_weather, predictions):

    ########################################################################
    # A bin is range that represents the width of a single bar of the
    # histogram along the X-axis. You could also call this the interval
    ########################################################################
    # The Numpy histogram function doesn't draw the histogram, but it
    # computes the occurrences of input data that fall within each bin,
    # which in turns determines the area (not necessarily the height if
    # bins aren't equal width) of each bar. EQUAL WIDTH left to right
    ########################################################################
    # If bins=5, for example, it will use 5 bins of equal width
    # spread between minimum input value and the maximum input value
    ########################################################################

    # turnstile_weather[..] is observe, original observed hourly data
    # turnstile_weather['ENTRIESn_hourly']
    # tried bins, 5, 10, 50, 80.
    # As you increase bins, the plot
    # takes on more of a normal curve look
    plt.figure()
    (turnstile_weather['ENTRIESn_hourly'] - predictions).hist(bins=50)
    return plt
```



2. OLS using Statsmodels

Ordinary Least Squares is another method for predictions. For OLS, I included more of the weather features. This gave me the best R^2 value, a bit better than my gradient descent model. (see below section 2.5).

```
def predictions(weather_turnstile):
    #feature set 1
    #features = weather_turnstile[['rain', 'precipi', 'Hour',
    #'meantempi']]

    # features set 2
    features=weather_turnstile[['rain','fog','thunder',
    'meandewpti','meanwindspdi','precipi','Hour',
    'meantempi','meanpressurei']]


    # UNIT is the location of the turnstile, not integer type
    # getDummies Convert categorical variable into dummy
    # indicator variables
    dummy_units = pandas.get_dummies(weather_turnstile['UNIT'],
                                                    prefix='unit')
    features = features.join(dummy_units)

    # Values
    values = weather_turnstile['ENTRIESn_hourly']

    # OLS = Ordinary Least Squares OSL (Y, X)
    model = sm.OLS(values,features)
    prediction = model.fit()

    #print prediction.summary()
    # model.fit().predict wants a DataFrame where the columns
    # have the same names as the predictors
    prediction = prediction.predict(features)
    return prediction
```

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

Features for Gradient Descent

```
features = dataframe[['rain', 'precipi', 'Hour', 'meantempi']]
And I joined UNIT as dummy
```

Then I tried without Hour, I was not sure Hour was a good choice to include in the feature set. I decided to keep hour.

```
features = dataframe[['rain', 'precipi', 'meantempi']]
And I joined UNIT as dummy
```

Features for OLS

OLS using same set of features as data set 1 for gradient descent.

```
features = weather_turnstile[['rain', 'precipi', 'Hour', 'meantempi']]
And I joined UNIT as dummy
```

Then I tried with more weather related variables.

```
Features = weather_turnstile [['rain','fog','thunder','meandewpti',
'meanwindspdi','precipi','Hour','meantempi','meanpressurei']]
And I joined UNIT as dummy
```

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.

I selected these values because I believed that weather related events affect the ridership in a linear fashion. I also included Hour, I was not sure Hour would be a good value because it may not follow a linear progression. My R value was closer to 1 with Hour included. I was not convinced the features impacted ridership in a linear fashion. I tried several combinations, but could not get an r^2 value over .50

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

theta1 through thetaN in the predictions function using gradient descent code and are the weights of the non-dummy features. They tell us how important the feature is in our prediction model, if theta is small, then X is not an important value in predicting our output variable.

My feature set is 18850 by 469 columns, mostly the units

```
features is length 18850 by 469 columns, .. Continuing columns for each unit…
   rain  precipi  Hour  meantempi  unit_R001  unit_R002  unit_R003  unit_R004
0     0     0.00     4         74          0          0          0          0
1     1     0.07     4         67          0          0          0          0
2     0     0.00    20         65          0          0          0          0
[18850 rows x 469 columns]
```

Theta, one value per column

```
length of theta is  470
[  2.92398062e+00   1.46526720e+01   4.67708502e+02  -6.22179395e+01
   1.50048977e+02  -1.51041835e+01  -2.78178403e+01  -1.78111613e+01
  -3.82710650e+00  -1.51064914e+01  -3.03405205e+01  -2.81397316e+01
  -2.72977045e+01   1.66072856e+02   2.13089119e+02   2.67066331e+02
   5.16152726e+01   8.82106563e+01   1.45131256e+01  -6.62061400e+00
   1.50151940e+02   1.81381142e+02   9.12824534e+01   1.66301402e+02
   1.95134027e+02   3.31670190e+02   2.52955523e+02   6.95647250e+01
   1.23023878e+02   4.96124717e+01   6.63699366e+01   2.71275019e+02
   3.91722520e+01   9.66980021e+01   1.13481504e+02   2.03923524e+02
   6.25995495e-01   1.05820526e+02  -1.37963537e+01  -1.80013053e+00
  -3.27954482e+01  -1.57338803e+01   2.40551279e+01   9.27339967e+01
  -5.49078674e+00  -3.80039668e+01   1.01906404e+02   4.57223135e+01
   2.48437131e+02   2.31882935e+02   2.83139436e+01   7.94475649e+01
   1.31061801e+02   1.31300761e+02   4.18095222e+00   7.96368978e+01
   2.29892511e+00   2.06037678e+02   2.14411380e+01   1.19340398e+02
  -1.16613447e+01   1.20077190e+01  -6.26584061e+00  -1.54628210e+01
   1.10967381e+02  -8.84860608e-01  -1.17517123e+01  -6.88583093e+00
  -2.55555792e+01  -1.11363180e+00  -1.80672170e+01  -1.29962705e+01
   1.48668764e+01   7.63598907e+01   1.31615020e+02   8.81024335e+01
   7.10038324e+00   8.62353968e+01   2.55269667e+02   2.70952300e+01
   4.57764855e+01   5.51407232e+00  -1.77230574e+01  -1.96240474e+01
  -2.49486008e+01   1.09359230e+01   2.48741776e+01   1.92440070e+01
   1.94854288e+01   2.66630818e+01   4.30533932e+01   6.94323103e+01
   5.00310592e+01   4.39223928e+01  -1.76023459e+01   7.04055675e+01
   1.38167300e+02   1.06162644e+01   3.59649577e+01   7.88106275e+01
  -3.53738569e-01  -3.29696161e+00   1.55927816e+02   1.69053707e+01
   7.18854459e+01   5.17933579e+01   2.25060137e+01   6.21093692e+01
  -3.54641138e+00   1.05705688e+00   8.42167769e+01  -3.86817297e+00
   1.22408491e+01   2.29820639e+01   1.66715245e+01   1.60787783e+01
   5.10221678e+01   5.38004964e+01  -1.49439231e+01  -1.16135349e+01
   2.89759258e+01   1.29025343e+02  -2.06271758e+01   1.31320798e+01
  -1.65639606e+01   9.10323193e+01   8.02906034e+01   1.11730913e+01
  -1.92383723e+01   7.82542916e+00   2.99968282e+01   5.03477387e+01
   1.65183577e+02   1.65060583e+01   1.17815897e+01   1.01895050e+02
   7.44241689e+01   4.91928463e+01   8.38284634e+01  -1.83678785e+01
   1.55278282e+01   4.21211046e+01  -1.03792582e+01   2.62002267e+00
  -2.80226986e+00   3.33085928e+01   2.09961672e+01   5.03164568e+01
   1.31932560e+00  -1.36045026e-01   3.62466188e-02   1.11634842e+01
```

```
 1.22038517e+02   1.28670551e+01   6.66731439e+01   1.33745672e+01
 9.60670502e+01   9.01344169e+01  -4.85764483e+01   5.62686462e+01
 6.26727112e+01   1.48694711e+02   3.35068505e+01   5.00414940e+02
 8.68066204e+00  -2.10257955e+00  -2.56847487e+00   1.64480655e+01
 1.44297465e+02   6.63098422e+01   1.09567641e+02   1.91881373e+02
 2.78125290e+02   3.83518594e+01   2.03687137e+01   5.09909874e+01
-5.78887965e-01  -5.46397265e+00   5.06683758e+00   7.42743335e+00
 3.16116046e+01   5.22556678e+01   1.97528601e+01   4.66587287e+01
 3.24473270e+01   3.13032551e+01   1.63326803e+01   2.90693854e+01
 1.60710579e+02   6.53772896e+00   2.06010322e+01   5.48789870e+01
-1.03006272e+01   3.52467369e+01   7.09050598e+01   5.47745857e+01
 9.79737628e+00   2.67633781e+01   2.66819218e+01   2.87386871e+01
 3.99245233e+01   7.37477117e+01  -3.00408986e+00  -1.44469574e+01
 2.79650790e+01   2.95782867e+01   1.19019670e+01  -1.08724998e+01
 1.60719642e+01  -1.33071187e+01   2.40376541e+01   1.71926617e+01
-1.13588499e+01  -3.24544547e+00   6.72658276e+00   7.06604331e+01
 4.23561090e+01  -8.01175225e+00  -1.49990063e+01  -8.78100551e+00
-2.37672549e+00   6.72927256e+00  -1.51755343e+01  -1.75937192e+01
-3.64694329e+00   1.04490832e+01  -3.06820340e+00  -2.13828360e+01
 1.15044764e+02   2.90499794e+01  -1.28082169e+01   4.80749249e+01
-6.43462805e+00   1.02582351e+02  -2.50690566e+01  -1.63062202e+01
 2.86130420e+01   1.69578970e+01  -7.18142286e+00  -2.52714068e+01
 8.31163186e+01   2.86389022e+01  -3.35894490e-01   1.12344963e+01
-6.46018722e+00  -1.74766967e+01   6.13660826e+01  -8.63953967e+00
 1.72718038e+00   3.98900587e+01   3.66611207e+01  -6.17733830e+00
 1.77550634e+01   1.04675884e+01  -7.18810321e+00  -3.56617357e+01
-2.24938334e+01  -8.52652410e+00   7.59911032e-01   4.62456576e+00
 5.57623298e+01  -3.87211542e+00  -1.71708270e+01  -1.89741611e+01
 3.16461361e+01   1.14515983e+01  -5.26717163e+00  -7.39687157e+00
 1.84135764e+01  -5.93125845e+00  -2.45820155e+01  -1.57118031e+01
-1.11708169e+01   1.64393144e+01   1.73837628e+01  -1.10113821e+01
-5.21955466e+00  -1.81140993e+01  -2.26540482e+01  -2.76647704e+00
 3.56673714e+01  -1.60159757e+01   2.98461187e+01   2.29884191e+01
-2.37929126e+01   2.45429952e+02  -2.57666107e+00  -1.55981971e+01
-1.85741875e+01  -9.11463740e+00  -9.77392606e+00  -2.59040332e+01
 6.47952316e+01   3.17516647e+00   4.49564729e+01   2.27908025e+01
-2.30662784e+00  -3.60378307e+01  -1.79093664e+01  -3.70940451e+00
-4.22550003e+00   1.97467272e+01  -1.81003256e+01  -2.67692378e+01
-2.52117889e+01  -2.09685883e+01  -1.53484328e+01  -2.95207357e+01
-1.50109966e+01  -1.77590991e+01   4.19232583e+01  -1.48460812e+01
-5.33563180e+00   2.89895748e+01   1.15778138e+01   2.60615941e+01
-2.54795330e+01  -2.14884481e+01  -3.22301004e+00  -2.55735966e+01
-2.68592085e+01   2.50492296e+00  -3.14189352e+01  -1.38648866e+01
-1.17163150e+01   2.38271400e-01  -1.69316116e+01  -2.65934826e+01
-3.30000680e+01  -4.09388736e+01  -2.37005601e+01  -2.66587511e+01
-2.52111821e+01  -1.89995888e+01  -7.43492697e+00  -1.77254514e+01
-1.84646960e+01   6.88263540e-02  -6.43422348e+00  -3.02489437e+01
-8.63456850e+00  -2.65950082e+01  -2.45560331e+01  -1.76573585e+01
-2.75102374e+01  -3.20560426e+01  -7.38477272e+00  -3.52012537e+01
-2.58174990e+01   5.57228671e+01  -3.13940428e+01  -3.96081119e+00
-9.82232561e+00  -2.06761669e+01  -8.64652986e+00  -1.93162138e+01
-1.98290293e+01  -2.22590197e+01  -1.76862690e+01  -1.45367911e+01
-1.59402972e+01  -1.14422889e+01  -1.46165877e+01  -1.22320571e+01
-8.57764806e+00  -2.04223404e+01  -2.22987259e+01   7.41479002e+00
 8.21672738e+00  -2.02835359e+01  -6.57019285e+00  -7.64812931e+00
-4.26073602e-01  -2.60150458e+01  -2.77392980e+01  -2.74593788e+00
-2.06771622e+00  -1.43276684e+00  -4.76246626e+00  -2.72040690e+01
 2.46829569e+01   5.43973047e+00  -4.72477894e+00  -1.57853028e+01
 1.18737442e-01  -1.25911777e+01  -1.91415337e+01  -2.87506199e+01
-1.19308332e+01  -2.52758365e+01  -3.29973179e+01  -2.56932053e+01
-2.69425117e+01  -2.31969460e+01  -1.04640497e+01  -1.63455270e+01
 1.10341777e+00  -3.50342263e+01   7.72466058e+00  -1.77872512e+01
-2.27666219e+01  -2.87829997e+01  -2.53520955e+01  -3.40200700e+01
-4.67921051e+01  -3.00631021e+01  -3.03755890e+01  -3.66659085e+01
-3.94676403e+01  -1.85643229e+01  -2.41963528e+01  -3.24984311e+01
-2.33215131e+01  -2.57757079e+01  -2.33830335e+01  -3.70482702e+01
-3.04501205e+01  -3.12934992e+01  -1.19141325e+01  -1.54377113e+01
-1.80132550e+01  -2.26994061e+01  -2.61288907e+01  -1.95216924e+01
-2.86648011e+01  -3.15231402e+01  -1.69257037e+01  -2.03155413e+01
-1.55249279e+01  -2.39118819e+01  -2.16670019e+01  -2.37745319e+01
-4.27971316e+00  -2.63205000e+01  -3.84155187e+00  -1.36450122e+01
-2.01281768e+01  -4.14871372e+01  -1.43512544e+01  -2.85127638e+01
```

```
  -7.33504374e+00   1.69717762e+02   1.91311452e+01  -2.94194679e+01
  -2.50590366e+01  -2.79625975e+01  -2.61591921e+01   1.43779868e+01
   1.03360548e+02   3.41207957e+01   6.78632689e+01  -3.13487939e+01
  -2.14761798e+01  -2.15114698e+01  -2.64818431e+01  -2.47460624e+01
  -1.62082154e+02  -2.06914281e+02  -1.08074254e+02  -1.78889027e+02
  -1.33497294e+02  -1.25293661e+02  -1.07118617e+02  -7.86617495e+01
  -6.48198796e+01  -3.18393401e+02  -2.41395539e+02  -1.50927305e+02
  -1.51524518e+02   1.10060866e+03]
```

2.5 What is your model's R^2 (coefficients of determination) value?

Your prediction should have an R^2 value of 0.20 or better. The closer R squared is to 1, the better the model. It varies from 0 to 1 and bigger values are better. This gives us a way to evaluate how good our model is. We can use numpy mean in our python code to help.

```
def compute_r_squared(data, predictions):
    r_squared = 1.0 - ( (np.sum((data - predictions)**2)) /
    (np.sum((data - (np.mean(data)))**2)) )
    return r_squared
```

R^2 Feature Set 1 with Gradient Descent

```
features = dataframe[['rain', 'precipi', 'Hour', 'meantempi']]
```

```
Your r^2 value is 0.463968815042
```

R^2 Feature Set 2 with Gradient Descent, without hour

```
features = dataframe[['rain', 'precipi', 'meantempi']]
```

```
Your r^2 value is 0.425845325332
```

R^2 Feature Set 1 for OLS, using the same features as the gradient descent set 1

```
features = weather_turnstile [['rain', 'precipi', 'Hour', 'meantempi']]
```

```
Your R^2 value is: 0.483404937509
```

R^2 Feature Set 2 for OLS using more weather related variables

```
features=weather_turnstile[['rain','fog','thunder','meandewpti','meanwi
ndspdi','precipi','Hour','meantempi','meanpressurei']]
```

```
Your r^2 value is 0.48518596312
```

2.6 What does this R^2 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 thisR^2 value?

I believe the R^2 values are telling me my model is a fair fit for predicting ridership given a variety of weather factors.

Section 3. Visualization

Please include two visualizations that show the relationships between two or more variables in the NYC subway data. You should feel free to implement something that we discussed in class (e.g., scatter plots, line plots, or histograms) or attempt to implement something more advanced if you'd like.

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.

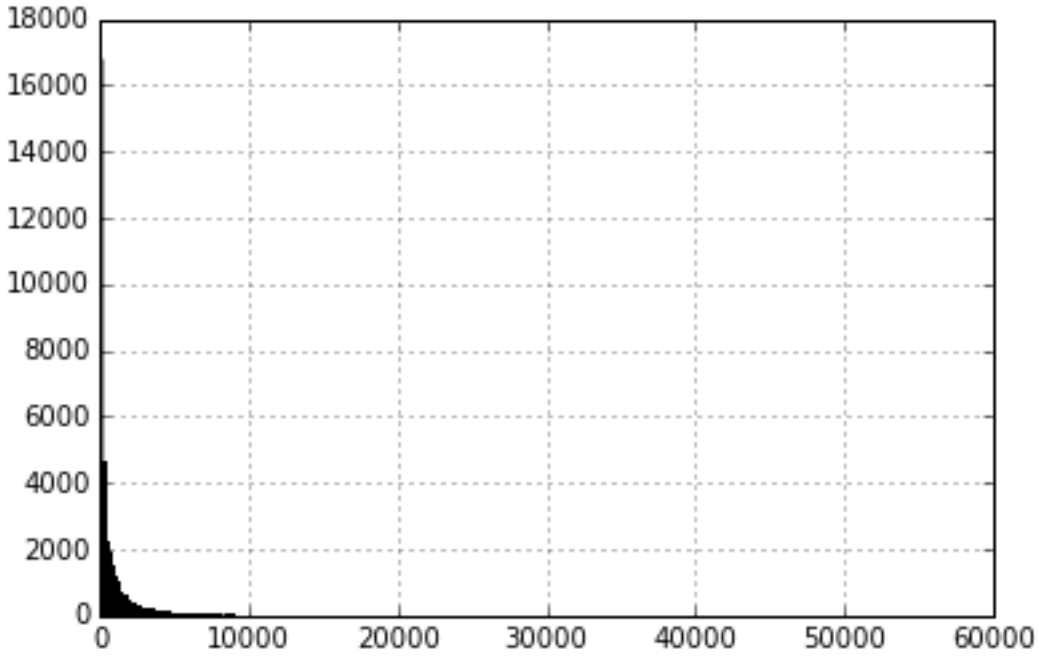
3.1 One visualization should contain two histograms: one of ENTRIESn_hourly for rainy days and one of ENTRIESn_hourly for non-rainy days.

- 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.

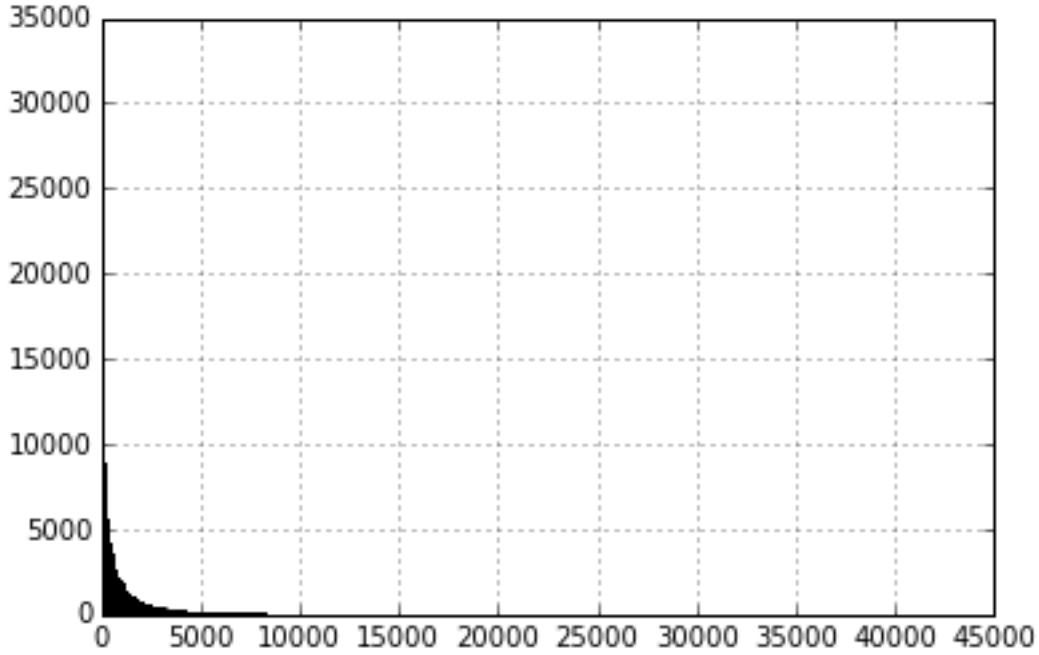
```
import numpy as np
import scipy
import scipy.stats
from pandas import *
from ggplot import *
import matplotlib.pyplot as plt

def mytry(turnstile_weather):
    ###########################################################
    # pull out weather data
    # with_rain is a list of 131951 single entries
    # without_rain is a list of 131951 single entries
    # rdf is [131951 rows x 2 columns]
    # rain without_rain
    #0 0 0
    #1 0 217
    ###########################################################
    with_rain =
turnstile_weather['ENTRIESn_hourly'][turnstile_weather.rain == 1]
    without_rain =
turnstile_weather['ENTRIESn_hourly'][turnstile_weather.rain == 0]
    #print with_rain.head(10)

    ###########################################################
    # histogram, entries rain not rain
    # histogram view is not spread enough to see differences
    ###########################################################
    plt.figure()
    (turnstile_weather['ENTRIESn_hourly'][turnstile_weather.rain ==
1]).hist(bins=400)
print plt
    plt.figure()
    (turnstile_weather['ENTRIESn_hourly'][turnstile_weather.rain ==
0]).hist(bins=400)
    print plt
```

```
    ###########################################################
    # plot spread out, entries rain not rain
    ###########################################################
    rainData = {'rain':Series(with_rain),
                'without_rain': Series(without_rain)}
    rdf = DataFrame(rainData)
    rdf['rain'] = rdf['rain'].fillna(0)
    rdf['without_rain'] = rdf['without_rain'].fillna(0)

    ###########################################################
    # reshape data from wide to long format
    # [263902 rows x 2 columns]
    # variable value
    # rain 0
    #263901 without_rain 0
    ###########################################################
    rdf = melt(rdf)
    #print type(rdf)

    ###########################################################
    # plot the data in long format on same plot
    # Note that to differentiate between multiple categories on
    # the same plot, we pass color in with the other arguments
    # to aes, rather than in our geometry functions.
    # xvar and yvar are going to be columns in the data frame
    ###########################################################
    plotD = ggplot(aes(x ='value', fill='variable', color='variable'),
data = rdf) \
      + scale_color_manual(name="legend title",values=['black', 'red']) \
      + geom_histogram( alpha=0.5, binwidth=400, position="dodge") \
      + scale_y_log() \
      + ylab("Frequency of occurrence") \
      + xlab("Entries") \
      + ggtitle("Entries vs Frequency")
    print plotD

    ###########################################################
    # plot, by hour
    ###########################################################
    plotA = ggplot(turnstile_weather, aes('Hour', 'ENTRIESn_hourly',
                                          fill ='rain', color='rain'))\
      + geom_histogram() \
      + scale_color_manual(values=['black', 'red']) \
      + xlab("Which hour of the day") \
      + ggtitle("Ridership by hour - Rain in black: No Rain in Red" )\
      + ylab("Entries")
    print plotA


file_path="C:/Users/dak/Documents/Udacity.IntroDataSciences/turnstile_data
_master_with_weather.csv"
turnstile = pandas.read_csv(file_path)
mytry(turnstile)
```

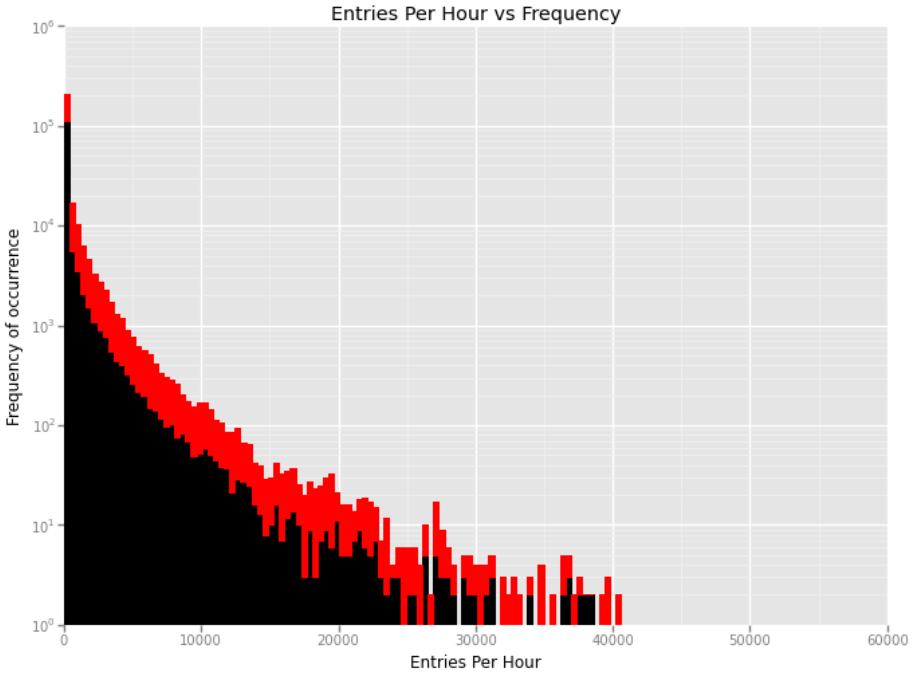
Plot 1. This is entries per hour when rain using histogram. You can see the view is not spread enough to see the differences.



Plot 2. This is entries per hour when **no rain** using histogram. You can see the view is not spread enough to see the differences.

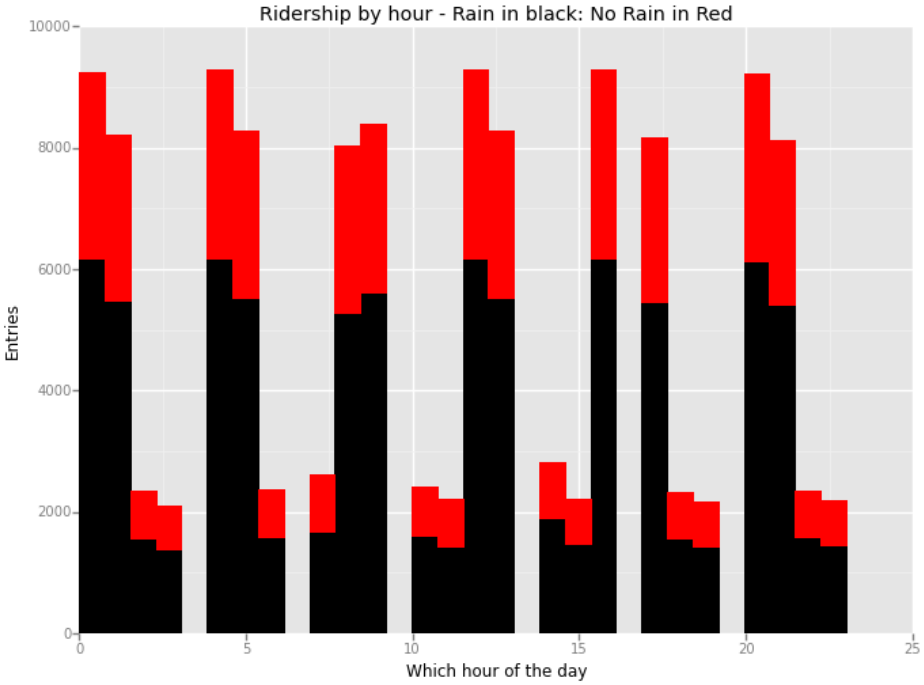


Plot 3 For this plot, I have intervals representing volume of ridership (value of ENTRIESn_hourly) on the x-axis and the frequency of occurrence on the y-axis. For example, each interval (along the x-axis), the height of the bar for this interval will represent the number of records (rows in our data) that have ENTRIESn_hourly that falls in this interval.



Rain = black, no-rain = orange.
The graph shows ridership decreases as weather events include rain.

Plot 4. This is entries by hour of the day, rain versus no rain. I used this plot to help me understand the aes section of ggplot.



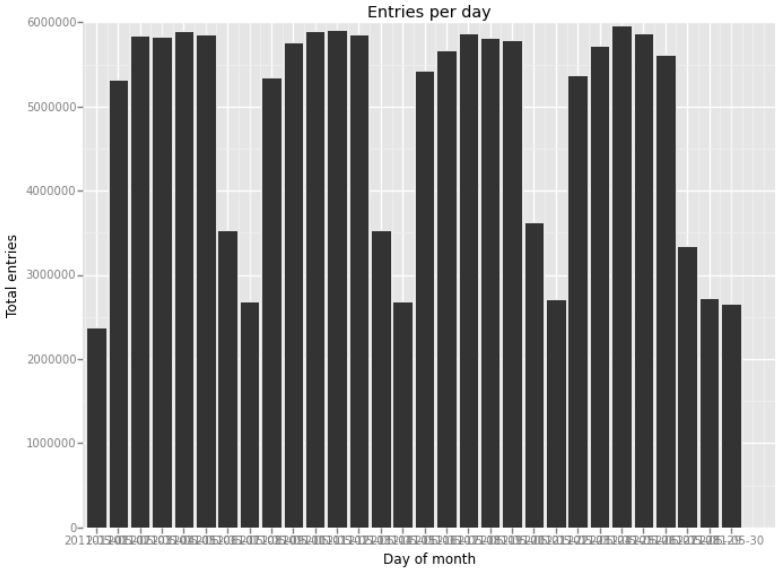
3.2 One visualization can be more freeform. Some suggestions are:

A. I looked at ridership per day of the month, using gemo_bar

```
from pandas import *
from ggplot import *

def getday(date):
    return datetime.strftime(datetime.strptime(date,'%Y-%m-%d'),'%a')

def plot_weather_data(turnstile_weather):
    days = []
    for mydate in turnstile_weather['DATEn']:
   days.append(getday(mydate))

    turnstile_weather.is_copy = False
    turnstile_weather.loc['DATEn'] = Series(days,
                    index=turnstile_weather.index)

    mygrouped = turnstile_weather.groupby('DATEn',
                            as_index=False).sum()

    plot = ggplot(mygrouped, aes('DATEn','ENTRIESn_hourly')) + \
    geom_bar(stat= "bar") + \
    ggtitle("Entries per day") + \
    xlab("Day of month") + \
    ylab("Total entries")
```

return plot

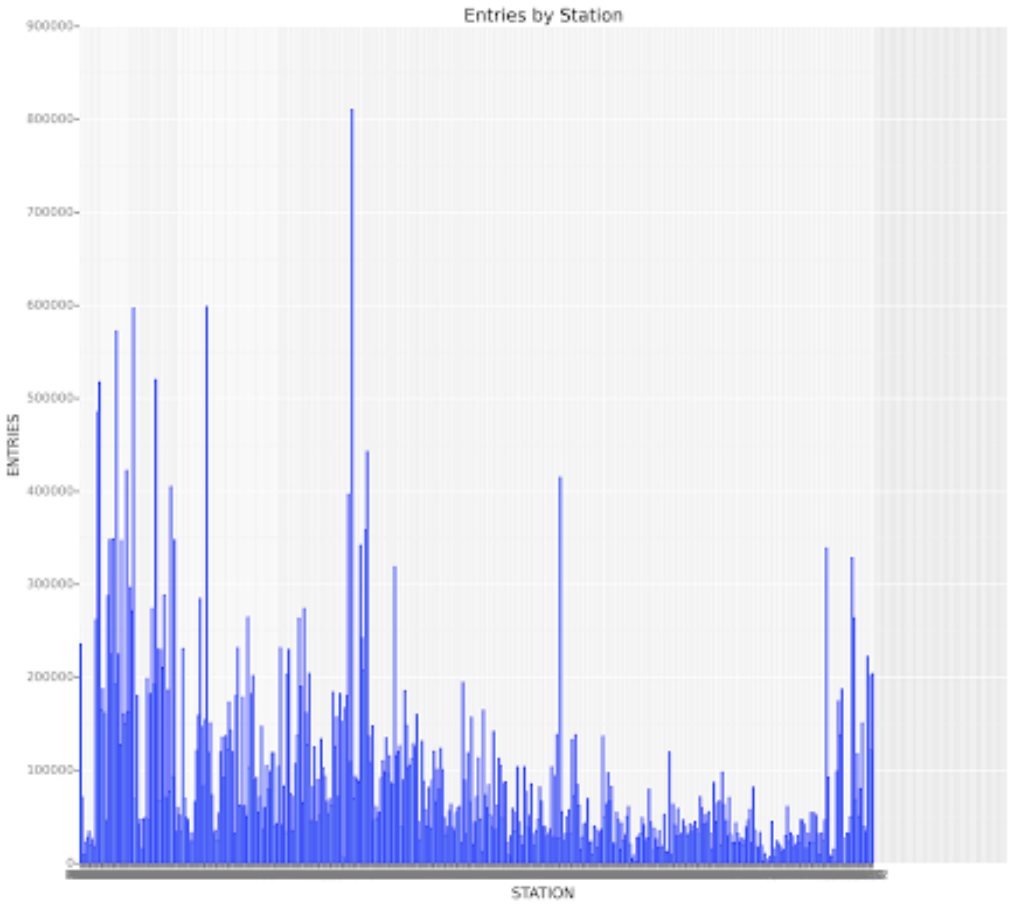
```
filename="C:/Users/dak/Documents/Udacity.IntroDataSciences/turnstile_da
ta_master_with_weather.csv"
turnstile = pandas.read_csv(filename)
result = plot_weather_data(turnstile)
print result
```



This shows ridership impacted by the pattern of weekends or weekdays.

B. I wanted to look at entries by Station, using geom_bar

```
from pandas import *
from ggplot import *

def plot_weather_data(turnstile_weather):
    # plot hourly entries based on Subway Station
    # pull out only those two fields
    turnstile_weather = turnstile_weather[['UNIT','ENTRIESn_hourly']]

    # add up the entries for each station
    grouped = turnstile_weather.groupby('UNIT', as_index=False).sum()

    plot = ggplot(grouped, aes(x='UNIT', y='ENTRIESn_hourly'))  + \
            geom_bar(stat='identity', colour="blue", fill='lightblue')
+ \
            ggtitle('Entries by Station')  + \
            xlab('STATION') + ylab('ENTRIES')
    return plot
```



This graph indicates ridership is clearly impacted by station choice. Upon examination of the data, the station with the max entries is 2887918 from UNIT R170; and the min is 0 from UNIT 447. The bar graph corroborates that.

```
       ENTRIESn_hourly
       count        465.000000
       mean      310822.208602
       std       330968.910573
       min            0.000000
       25%       111335.000000
       50%       194635.000000
       75%       382576.000000
       max      2887918.000000

       MAX row 159 is UNIT   R170
       ENTRIESn_hourly    2887918

       MIN row 447 is UNIT   R464
       ENTRIESn_hourly          0
```

```
  # add up the entries for each station
    grouped = turnstile_weather.groupby('UNIT',
                                        as_index=False).sum()
    print grouped.describe()

    # which station has the max entries
    myresult = grouped['ENTRIESn_hourly'].argmax()
    print "MAX row is ",grouped.ix[myresult,:]

    # which station has the min entries
    myresult = grouped['ENTRIESn_hourly'].argmin()
    print "MIN row is ",grouped.ix[myresult,:]
```

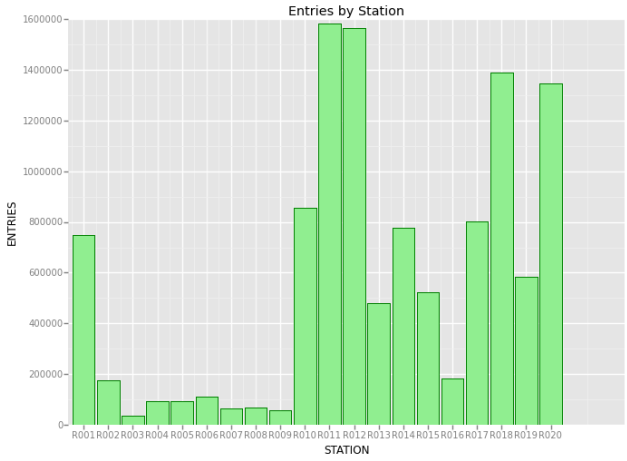
C. I wanted to look at histograms vs bar graphs using this same data.

I also plotted a smaller section of the data using both a histogram and a bar graph, because I was wondering about the differences in the two charts and the difference between bar and identity in the stat value. For this, I was not looking to make determinations about the data itself, but to examine geom_histogram and geom_bar. I learned that geom_histogram is an alias for geom_bar plus stat_bin. Also in this case, using stat=bar, stat=identity gave the same result.

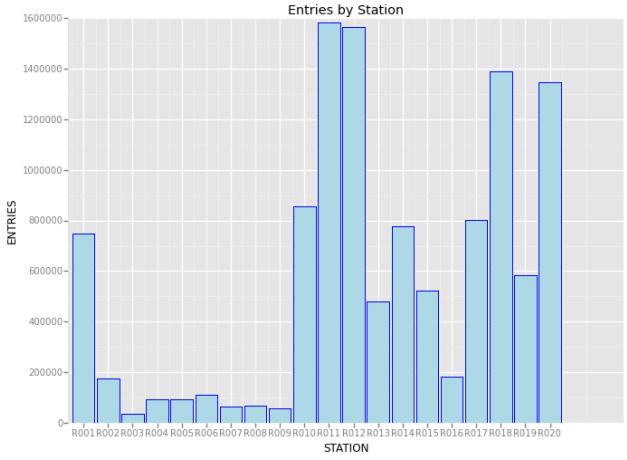
```
# plot hourly entries based on Subway Station

# pull out only those two fields
turnstile_weather = turnstile_weather[['UNIT','ENTRIESn_hourly']]

# add up the entries for each station
grouped = turnstile_weather.groupby('UNIT', as_index=False).sum()

# get a smaller set of the data
groupedSmaller=grouped.head(20)

# stat='bar' is a count of cases in each group
# stat='identity' is the values in a column of the data frame

################################################
# HISTOGRAM
################################################
# geom_histogram is an alias for geom_bar plus stat_bin
# this makes the bin off, need to understand what the correct value for
```

```
# bin width is. Also, if you do not use stat=identity, the
# histogram, is a very different picture too
plot = ggplot(groupedSmaller, aes(x='UNIT', y='ENTRIESn_hourly')) + \
geom_histogram(stat='bar', colour="green", fill='lightgreen') + \
ggtitle('Entries by Station') + \
xlab('STATION') + ylab('ENTRIES')
print plot

##############################################
# GEOM_BAR
##############################################
plot = ggplot(groupedSmaller, aes(x='UNIT', y='ENTRIESn_hourly')) + \
geom_bar(stat='bar', colour="blue", fill='lightblue') + \
ggtitle('Entries by Station') + \
xlab('STATION') + ylab('ENTRIES')
```

GREEN: This is geom_histogram with a smaller set of the data



BLUE: This is geom_bar with a smaller set of the data

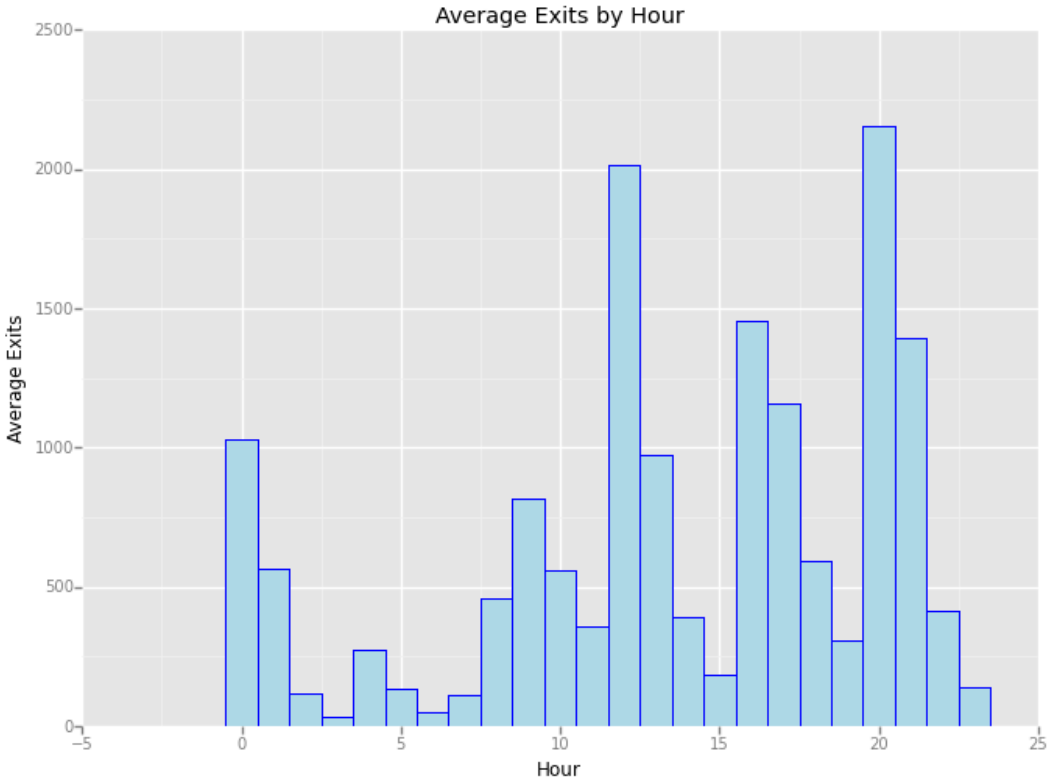


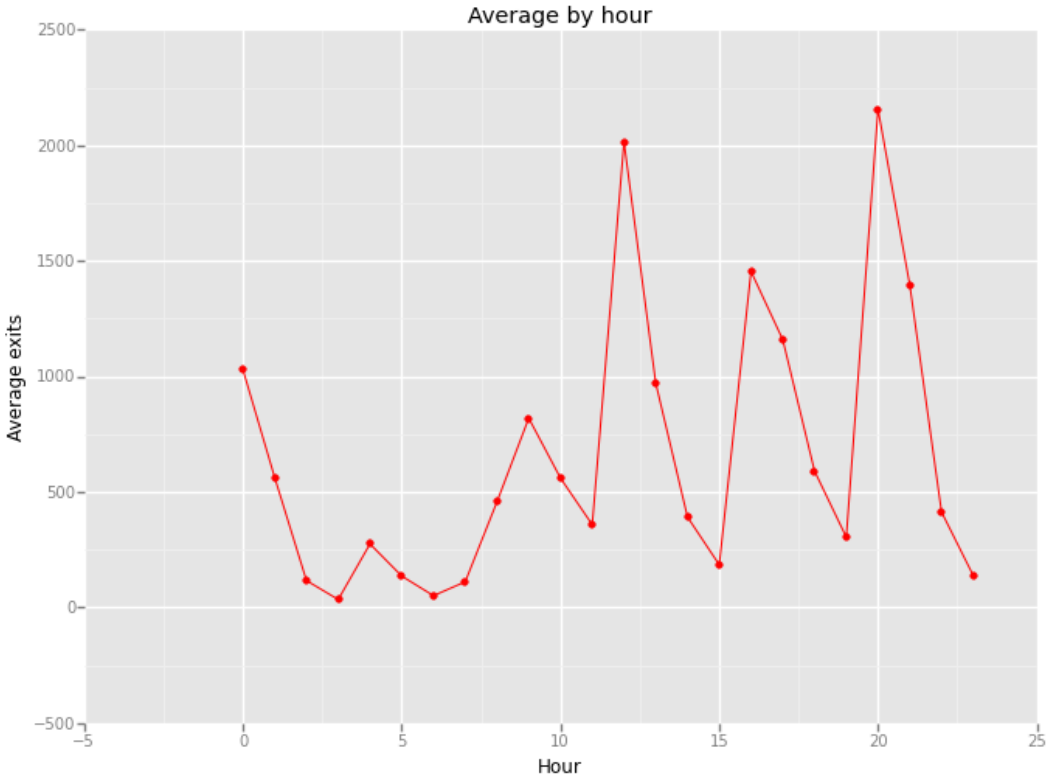
Remember, I was just learning about the plots here, this particular task was not to analyze the data. I concluded that either method was reasonable and likely to yield the same results. Both graphs were basically the same thing.

D. I wanted to looks at a visualization for average exits by hour.

I also did visualizations on exits by hour. I wanted to use a bar graph and a point graph, and observe them. Visually, I liked the point graph better.

```
from pandas import *
from ggplot import *
import matplotlib.pyplot as plt

def plot_weather_data(turnstile_weather):
    # geom_histogram is an alias for geom_bar plus stat_bin

    # average exits hourly
    # pull out only those two fields
    # this prints 131951 rows of data, two columns, hour and exits
    turnstile_weather = turnstile_weather[['Hour','EXITSn_hourly']]
    #print turnstile_weather

    # counts on the Y axis
    # this prints 23 entries, one per hour
    # and the average of exits for that hour
    grouped = turnstile_weather.groupby('Hour', as_index=False).mean()
    print grouped

    # stat='bar' is a count of cases in each group
    # stat='identity' is the values in a column of the data frame
    # bar graph

    plot = ggplot(grouped, aes('Hour','EXITSn_hourly')) + \
    geom_bar(stat='bar', colour="blue", fill='lightblue') + \
    ggtitle('Average Exits by Hour') + \
    xlab('Hour') + ylab('Average Exits')
    print plot

    # point plot
    gg = ggplot(grouped, aes('Hour','EXITSn_hourly')) + \
    geom_point(color='RED') + geom_line(color='RED') + \
    ggtitle( 'Average by hour') + \
    xlab('Hour') + ylab('Average exits')
    print gg

filename="C:/Users/dak/Documents/Udacity.IntroDataSciences/turnstile_da
ta_master_with_weather.csv"
turnstile = pandas.read_csv(filename)
result = plot_weather_data(turnstile)
```





These graphs led me to conclude that hour of the day has a definite impact on subway exits. Hour 20 has the highest number of exits. Hour 3 has the lowest amounts of exits. We see a spike in exits at noon.

```
grouped = turnstile_weather.groupby('Hour', as_index=False).mean()
print grouped.describe()

# which hour has the max average exits
myresult = grouped['EXITSn_hourly'].argmax()
print "MAX row is",grouped.ix[myresult,:]

# which hour has the min average exits
myresult = grouped['EXITSn_hourly'].argmin()
print "MIN row is",grouped.ix[myresult,:]

print grouped
```

```
       Hour  EXITSn_hourly
count  24.000000      24.000000
mean   11.500000     653.297516
std     7.071068     605.887279
min     0.000000      34.455630
25%     5.750000     173.007253
50%    11.500000     435.244929
75%    17.250000     985.187640
max    23.000000    2155.750136


MAX row is Hour   20.000000
EXITSn_hourly    2155.750136


MIN row is Hour   3.00000
EXITSn_hourly    34.45563


    Hour  EXITSn_hourly
0      0    1028.844533
1      1     562.032270
2      2     116.321155
3      3      34.455630
4      4     274.990964
5      5     135.852188
6      6      50.958950
7      7     111.271652
8      8     457.272965
9      9     819.117955
10    10     559.752902
11    11     358.255319
12    12    2013.745724
13    13     970.635342
14    14     391.124514
15    15     184.740018
16    16    1454.213247
17    17    1158.872187
18    18     590.533909
19    19     304.371124
20    20    2155.750136
21    21    1395.001848
22    22     413.216893
23    23     137.808958
```

Section 4. Conclusion

Please address the following questions in detail. Your answers should be 1-2 paragraphs long.
4.1From your analysis and interpretation of the data, do more people ride the NYC subway when it is raining or when it is not raining?

I believe the answer is that ridership is impacted by weather, and I believe the answer is yes, more people ride the subway when it is raining. I believe ridership is impacted by factors other than weather, such as station and weekday (work day) or city events. I base this conclusion on the data analysis performed, but not with confidence, in particular, the graphing on the histogram of rider on rainy days versus not rain actually shows less ridership during rain. But more importantly, my lack of confidence is more due to the fact that we only had one month of data to analyize.

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.

The first analysis, plotting histograms of entries with rain versus no rain shows entries by hour is higher without rain. However, the Whitney Wilcoxon U test, showed the average number of entries (average as opposed to sum) was indeed greater on raining days and led us to reject the hypothesis that ridership was unaffected by rain. This led us to accept the alternative hypothesis that ridership is impacted by rain.

The R^2 values for the Linear regression and OLS both were over .46 indicating 46% better than chance. These are low values, although it is an ok correlation between the features selected and the ridership.

Section 5. Reflection

Please address the following questions in detail. Your answers should be 1-2 paragraphs long.
5.1Please discuss potential shortcomings of the methods of your analysis, including:

1. **Dataset.**

   The datasets came from two different places: the Metropolitan Transportation Authority (MTA), which is responsible for transportation in New York, and the the National Climatic Data Center (NCDC), which is part of the National Oceanic and Atmospheric Administration. The weather data comes from measurements from three points in the city, JFK and LaGuardia airports, and Central Park. The subway data has turnstile information from 468 stations from all 6 Boroughs sampling on a 4 hour basis.

   The MTA Subway Turnstile data reports on cumulative number of entries and exits per row. We know the cumulative entry numbers are calculated as a count of entries since the last reading, i.e. the difference between ENTRIESn of the current row and the previous row. What if the data in the previous row is NaN? How does the data being given in 4 hour intervals affect outcomes?

	Unnamed: 0	UNIT	DATEn	TIMEn	Hour	DESCn	ENTRIESn_hourly
0	0	R001	2011-05-01	01:00:00	1	REGULAR	0
1	1	R001	2011-05-01	05:00:00	5	REGULAR	217
2	2	R001	2011-05-01	09:00:00	9	REGULAR	890
3	3	R001	2011-05-01	13:00:00	13	REGULAR	2451
4	4	R001	2011-05-01	17:00:00	17	REGULAR	4400

Furthermore, we know the subway data for entries is hourly, but it is combined to weather data which is on a daily basis, so for example, in the weather data, if it rained anytime during the day, all hours of the day are marked for rain.

https://s3.amazonaws.com/uploads.hipchat.com/23756/665149/05bgLZqSsMycnkg/turnstile-weather-variables.pdf

rain Indicator (0 or 1)
if rain occurred **within the calendar day** **at** the location.

The subway data we analyzed contained only one month of data, March 2011, which is not sufficient, and that month contains a holiday, Memorial Day. Additionally, Mother's Day falls in that time frame; as do baseball games at Yankee Stadium which draws many subway riders.

Of interest, the Daily News reported on 5/25/11, which is in our sample time, that there were many failures on Metro Cards, causing riders to swipe many time for a single admittance. How does this skew the data? What about fare-beaters?

The rain indicator is if there was rain at the location. We know that some subway locations are connected directly to transportation hubs that may not require you to go outdoors to enter the subway, thus those riders may not even be aware of the weather factors.

What about hours of operation, if not all stations are open the same hours, that certainly has impact on ridership. What about station closures.

What about express trains during particular times of day (don't go on the D to Yankee Stadium during rush hour because that train does not stop at the stadium, been there, done that!). I would expect there to be less entries from stations that trains bypass.

2. Analysis, such as the linear regression model or statistical test.

The linear regression and OLS model gave an r^2 value < .5, a bit low.

Linear regression looks at linear relationships among data that is dependent affected by independent parameters. This is not always a correct assumption. I thought predictions based on non-linear relationships might be a better fit.

5.2 (Optional) Do you have any other insight about the dataset that you would like to share with us?

I was interested in the result of entries by station, especially since there was a peak for UNIT R170; and a min of 0 from UNIT 447. I wondered where the stations were, and how this ridership affected items like nearby commerce, security, taxi ridership, and so forth.