Analyzing the New York Subway Dataset

Data Analyst Nano Degree

Intro to Data Science Course

Project 2

Bingson Huang

TABLE OF CONTENTS

A. PROJECT QUESTIONS

1. REFERENCES

1. **Kung, Sheng.** Questions about assumption in Mann-Whitney U Test. *Udacity Discussion Forums*. [Online] https://discussions.udacity.com/t/questions-about-assumption-in-mann-whitney-u-test/8498/7.

2. **Turner, Charlie.** Problem Set 3-3 Interpreting Mann-Whitney U-Test (repost). *Udacity Discussion Forums*. [Online] https://discussions.udacity.com/t/problem-set-3-3-interpreting-mann-whitney-u-test-repost/25403.

3. **jaroslav_105547.** Problem set 3.8 - sklearn.linear_model.SGDRegressor. *Udacity Discussion Forums*. [Online] https://goo.gl/CsfUMd.

4. **Ranjit, Priyanshu.** Mathematical Optimization: Why would someone use gradient descent for a convex function? *Quora*. [Online] Jun 25, 2012. http://www.quora.com/Mathematical-Optimization/Why-would-someone-use-gradient-descent-for-a-convex-function.

5. **Udacity.** Understanding the MannWhitney U Test. [Online] 2014. https://goo.gl/Bhe2Up.

6. **Wikipedia.** Dummy variable (statistics). *Wikipedia*. [Online] [Cited: Aug 11, 2015.] https://en.wikipedia.org/wiki/Dummy_variable_(statistics).

7. **McKinney, Wes.** *Python for Data Analysis*. Sebastopol : O'Reilly Media, Inc., 2013. ISBN: 978-1-449-31979-3.

8. **Essa, Alfred.** Python Pandas Cookbook. *Youtube*. [Online] Aug 11, 2013. [Cited: Jul 30, 2015.] https://goo.gl/46x9Kc.

9. **Diez, David M, Barr, Christopher D and Rundel-Cetinkaya, Mine.** *OpenIntro Statistics*. 3rd. 2015.

10. **Kormanik, Katie, Laraway, Sean and Rogers, Ronald.** Udacity Inferential and Descriptive Stats. *Udacity*. [Online] 2015. https://www.udacity.com/course/intro-to-inferential-statistics--ud201.

11. **Evans, Dave.** Udacity Intro to Computer Science. *Udacity*. [Online] 2015. https://www.udacity.com/course/intro-to-computer-science--cs101.

2. STATISTICAL TEST

2.1. 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?

To assess whether or not rain has an influence on subway ridership, I conducted a Mann-Whitney U test (Figure 5, p. 9).

I used a two-tailed hypothesis test and P value. Mathematically, my null and alternative hypothesis is expressed as

H_0: Prob(ENTRIESn_Hourly | rain > ENTRIESn_Hourly | no rain) = 0.5

H_A: Prob(ENTRIESn_Hourly | rain > ENTRIESn_Hourly | no rain) ≠ 0.5

In other words, my null hypothesis asserts that if we divided historic NYC subway traffic data (turnstile entries/hour) into two groups—hours with rain and hours without rain—the odds of drawing an observation from one group that is larger than the other is 50:50.

To test this hypothesis I settled on a p-critical value threshold of 0.05.

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

The Mann-Whitney U test is appropriate because it is a non-parametric test that can be applied to skewed data. This is important because NYC subway traffic during both raining and not raining periods in Figure 5 (p. 9) exhibits a right skewed distribution (exhibits a disproportionately large number of low turnstile count observations) due to the fact that subway traffic drops-off precipitously outside of normal business hours (see Figure 6, p. 9). The Mann-Whitney test makes no assumptions about population distributions and simply checks if there's a value bias when comparing random draws between two groups of observations [1].

2.3. 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.

Scipy (turnstile_data_master_with_weather.csv):

min(Ux, Uy) =	1924409167.0
two-tailed p-value =	0.0386192688276
with_rain_mean =	1105.4463767458733
without_rain_mean =	1090.278780151855

Sample code: Appendix 1.3, p. 14

Note: the p-value above was calculated using python's scipy library which corrects for ties and uses a continuity correction. I had to use the original turnstile_data_master_with_weather.csv file as opposed to the turnstile_weather_v2.csv file because of a hardware-related bug in scipy that would give me a Mann Whitney p-value of NaN (System: Windows 7, 64-bit).

Using the normal approximation method with the enhanced turnstile_weather_v2.csv data, I got the following test results.

Normal approximation (turnstile_weather_v2.csv):

min(Ux, Uy) =	153635120.5
two-tailed p-value =	5.48269387142e-06
with_rain_mean =	2028.19603547
without_rain_mean =	1845.53943866

2.4. What is the significance and interpretation of these results?

Because the Mann-Whitney p-value is below the 0.05 critical value in both tests variations above, we can reject the null hypothesis. The above results suggest that rain has an effect on subway traffic. However, the test tells us nothing about whether the effect is positive or negative. This information was thrown away during the step in the calculation where the minimum of Ux and Uy is taken [2].

3. LINEAR REGRESSION

3.1. What approach did you use to compute the coefficients theta and produce prediction for ENTRIESn_hourly in your regression model:

I computed θ coefficients and the associated ENTRIESn_hourly predictions using ordinary least squares (OLS) regression implemented through python's statsmodels library. This library was chosen because it provides useful summary statistics that help me better evaluate the fit of my model. The underlying data set was sourced from turnstile_weather_v2.csv, and was chosen over the original csv file because it provides ENTRIESn_hourly data across turnstiles in comparable 4 hour time intervals (ignoring truncation related error). In addition, because of the extreme right skewed quality of the data (p. 9), a logarithmic transformation was applied to the response variable in an attempt to linearize its relationship with test predictors.

Note: Although OLS offers a clear closed form solution for solving multifactor regression problems, for very large numbers of observations and feature sets (predictor variables), OLS can be prohibitively costly to implement[1]. As an alternative, Stochastic Gradient Descent (SGD) offers a numerical method for solving linear regression models by iteratively changing parameters (using first-

1) As illustrated in the following formula, an OLS solution requires computationally expensive matrix calculations to simultaneously solve a system of n equations corresponding to partial derivatives with respect to each predictor variable: $\theta = (X^T X)^{-1} X^T y$ — requires three matrix multiplications and one matrix inversion to solve [3]; where X is a n by n matrix of predictors, X^T is the inverse matrix of X, and y is the response vector. The challenge with this approach is that calculation complexity grows cubically as we add predictor variables (n).

order partial differential optimization) to find θ that minimizes the objective cost function (i.e. squared sum of residual errors).

For linear regression in particular, which has a simple convex cost function with a unique minimum (assuming linearly independent features), SGD can provide faster θ estimation, depending on how one calibrates the algorithm (e.g., specifying a learning rate that decreases as we move closer to the minimum). For completeness, I have also provided sample code for a SGD approach to estimating θ coefficients on page 15 [3].

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

I used only time and location related dummy variables for my model.

Dummy variable groups:

- *UNIT*: Remote unit that collects turnstile information.
- *Hour*: Hour of the timestamp from TIMEn. Truncated rather than rounded (4 hour intervals).
- *day_week*: Integer (0 - 6 Mon - Sun) corresponding to the day of the week.

Note: One dummy indicator was removed from each group (i.e. incorporated into the base case) to avoid the dummy variable trap[1].

3.3. 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.

My feature selection was driven by an intuitive model of the traffic patterns presented in my heat map in Figure 7 on page 10. Dummy variables related to day of the week (*day_week*) and time interval (*Hour*) were chosen to separate busy NYC subway traffic periods tied to business operating schedules from low traffic periods tied to sleep cycles.

Dummy variables describing individual turnstile units were also added to account for traffic differences related to turnstile location[2]. For example, turnstiles in New York's financial district are expected to be less busy on weekends (when stock markets are closed) than weekdays, while turnstiles located in Times Square, may record more consistent (or possibly elevated) weekend tourist-related traffic.

Note: The above categorical features were modeled as mutually exclusive dummy variables because their numerical values have no intrinsic meaning on their own. For example, it makes no difference if Tuesday is represented by a value of 2 or 4. A transformation of time

1) If we did not remove a dummy from each group, the system of equations formed by the regression does not have a unique solution [6].

2) Turnstile unit rather than station was selected, to account for turnstile traffic differences related to specific office buildings.

and location related features to binary indicator variables allows us to better model the effects of cyclical (e.g., regime switching) time-varying relationships.

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

I considered adding non-dummy variables describing a variety of temperature and weather related data points, but found few with significant theta coefficients or p-values. On the assumption that my predictive model would be used out of sample to predict turnstile traffic for months other than May, I decided to keep my feature set limited to station location and time schedules. Because of the limitations of a May only training set, I felt that this specification would reduce the risk of overfitting my model to non-stationary May temperature and weather relationships. If I had a larger training set to calibrate theta parameters (i.e., 12 months of data), I'd likely revisit the idea of adding weather and temperature related features after studying how these aforementioned relationships evolve over a four season cycle.

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

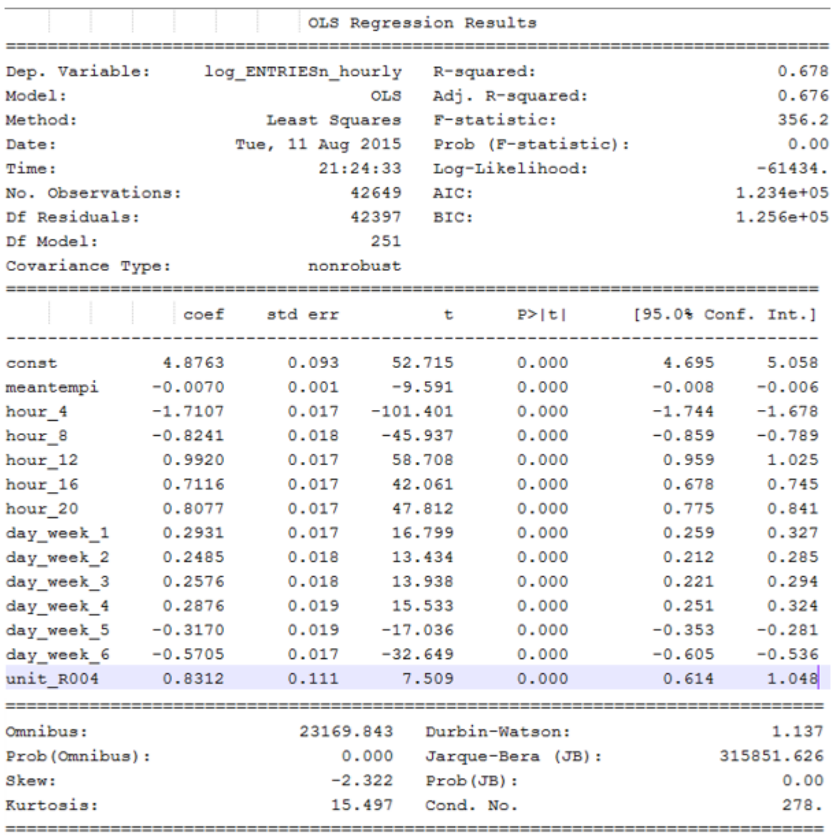
Looking at the inverse transformed (exponentiated) log linear time & location based turnstile traffic prediction model, I achieved an in-sample R^2 value of around 0.68, a 24% improvement over an R^2 of 0.544 generated by an OLS regression of an unlogged response variable (ENTRIESn_hourly).

3.6. 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 this R^2 value?

The R^2 means that, in-sample, my model's line of best fit explains 67.8% of the variation in subway turnstile traffic as measured by log_ENTRIESn_hourly, which can be exponentiated (inverse log-transformed) to produce predictions for ENTRIESn_hourly (Figure 4).

A side by side comparison of the log linear probability plot (Figure 2) and the histogram of logged ENTRIESn_hourly (Figure 6, p. 9) suggest that my model tends to under predict the number of low count turnstile observations. Therefore, to assess the appropriateness of my model, I would first need to know the cost of under predicting low count observations (especially zero values) to establish an acceptable accuracy threshold for rejecting or adopting the model.

Figure 1: Statsmodels summary Output



```
                            OLS Regression Results
==============================================================================
Dep. Variable:      log_ENTRIESn_hourly   R-squared:                       0.678
Model:                              OLS   Adj. R-squared:                  0.676
Method:                   Least Squares   F-statistic:                     356.2
Date:                  Tue, 11 Aug 2015   Prob (F-statistic):               0.00
Time:                          21:24:33   Log-Likelihood:                -61434.
No. Observations:                 42649   AIC:                         1.234e+05
Df Residuals:                     42397   BIC:                         1.256e+05
Df Model:                           251
Covariance Type:              nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
const          4.8763      0.093     52.715      0.000         4.695     5.058
meantempi     -0.0070      0.001     -9.591      0.000        -0.008    -0.006
hour_4        -1.7107      0.017   -101.401      0.000        -1.744    -1.678
hour_8        -0.8241      0.018    -45.937      0.000        -0.859    -0.789
hour_12        0.9920      0.017     58.708      0.000         0.959     1.025
hour_16        0.7116      0.017     42.061      0.000         0.678     0.745
hour_20        0.8077      0.017     47.812      0.000         0.775     0.841
day_week_1     0.2931      0.017     16.799      0.000         0.259     0.327
day_week_2     0.2485      0.018     13.434      0.000         0.212     0.285
day_week_3     0.2576      0.018     13.938      0.000         0.221     0.294
day_week_4     0.2876      0.019     15.533      0.000         0.251     0.324
day_week_5    -0.3170      0.019    -17.036      0.000        -0.353    -0.281
day_week_6    -0.5705      0.017    -32.649      0.000        -0.605    -0.536
unit_R004      0.8312      0.111      7.509      0.000         0.614     1.048
==============================================================================
Omnibus:                    23169.843   Durbin-Watson:                   1.137
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           315851.626
Skew:                          -2.322   Prob(JB):                         0.00
Kurtosis:                      15.497   Cond. No.                         278.
==============================================================================
```

Note: For length considerations, row entries for the other 238 unit dummy variables were excluded. Out of those results, only five had p-values above 0.05

Figure 2: log_ENTRIESn_hourly Q-Q Plot

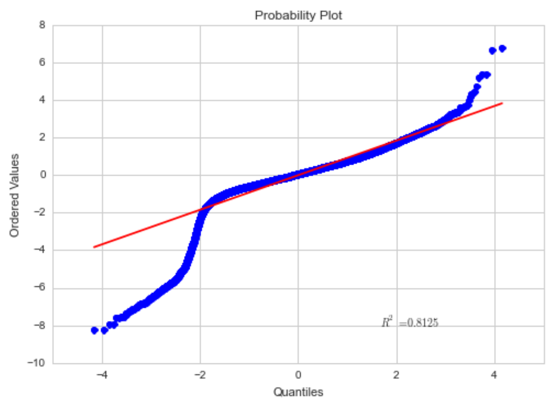


Figure 3: Log_ENTRIESn_hourly residuals histogram

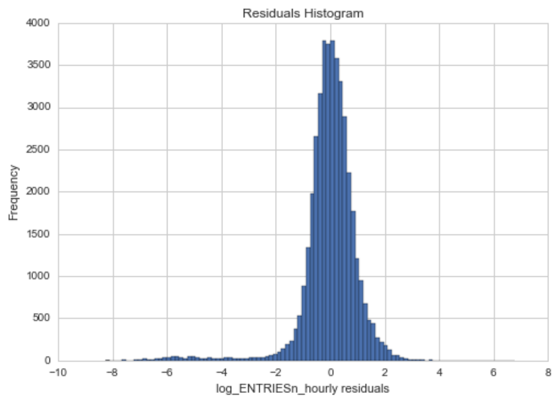
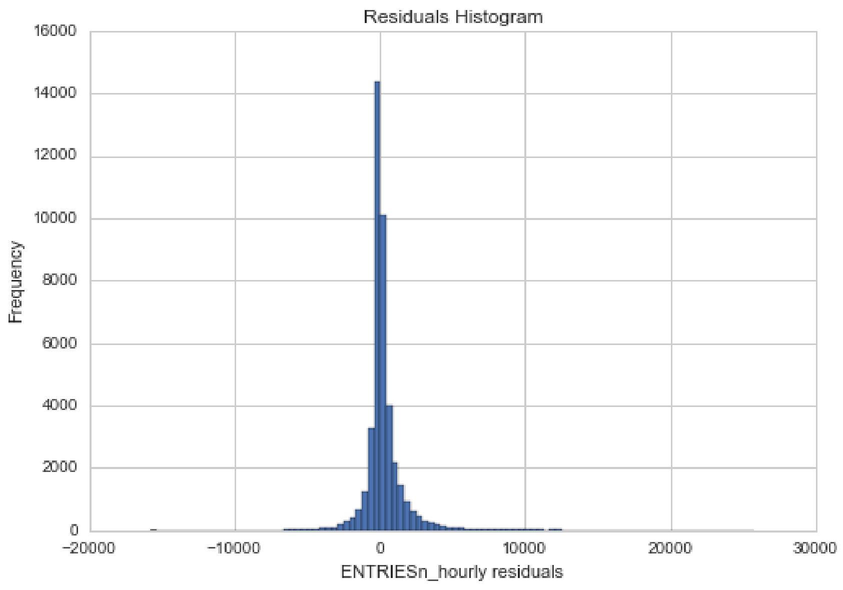


Figure 4: ENTRIESn_hourly residuals histogram

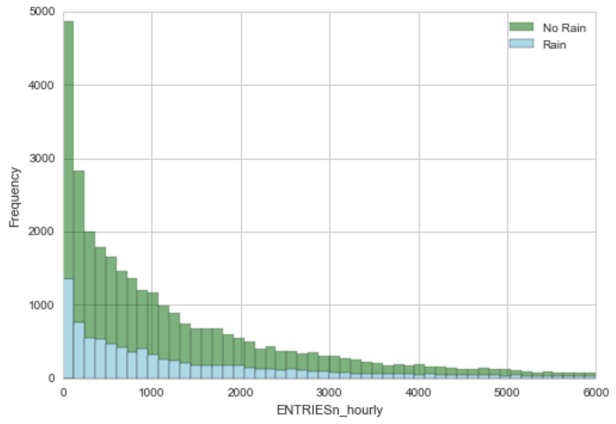


Predictions for ENTRIESn_hourly were found by taking the inverse log of log_ENTRIESn_hourly predictions (see *predictions* variable on p. 14).

4. VISUALIZATION

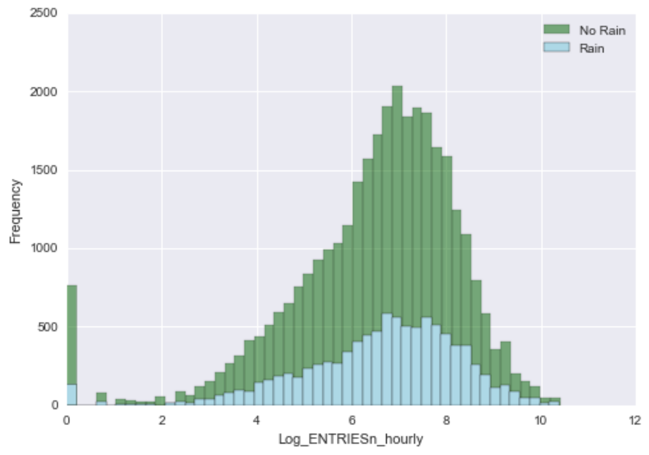
4.1. One visualization should contain two histograms: one of ENTRIESn_hourly for rainy days and one of ENTRIESn_hourly for non-rainy days.

Figure 5: Histogram of hourly NYC Subway Traffic (as measured by ENTRIESn_hourly)



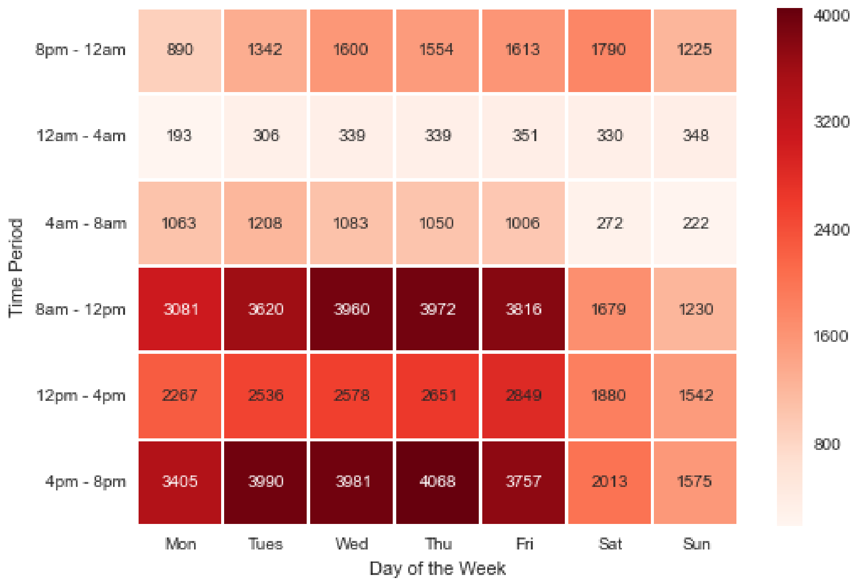
NYC subway traffic during both raining and not raining periods in Figure 5 (p. 9) exhibits a right skewed distribution. This shape can be attributed to the fact that subway traffic drops-off dramatically outside of the hours of 8 am – 8 pm (Figure 7). Since it is easier to work with relationships that are approximately linear (normal distribution), versus highly curved (skewed distributions), mathematical transformations are sometimes applied to data to help linearized otherwise non-linear relationships. After applying a logarithmic transformation to our response variable (Figure 6), we see that ENTRIESn_hourly has an approximately log normal distribution (Figure 6), with the exception of a non-normally distributed occurrence of zero-value response variable observations (for more on inflated zero-value observations see section 6.1, p. 12).

Figure 6: Histogram of the log of hourly NYC Subway Traffic (as measured by log_ENTRIESn_hourly)



4.2. 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.

Figure 7: NYC Subway Time Period Heat Map
(as measured by average turnstile entries per four hour interval)



Code: Appendix 0, p. 16

In Figure 7 I constructed a heat map showing average ENTRIESn_Hourly at different time intervals throughout the week. As expected, subway traffic peaks during morning and evening rush hours, and weekends are generally less busy than weekdays. Figure 7 suggest that variations in turnstile traffic are better explained by work schedules and cyclical ridership trends than weather patterns.

5. CONCLUSION

5.1. 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?

According to my analysis, slightly more people ride the NYC subway when it is raining.

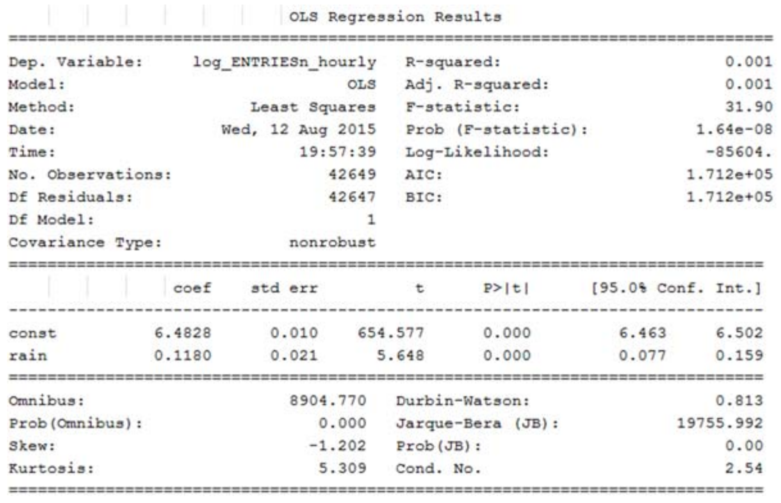
5.2. What analyses lead you to this conclusion? You should use results from both your statistical tests and your linear regression to support your analysis.

I came to the above conclusion from looking at my Mann-Whitney U test on the enhanced dataset which gave me a two-tailed p-value of 5.48269387142e-06. This indicates that if we sorted historic NYC subway traffic data (turnstile entries/hour) into two groups—observations with rain and observations without rain—the odds of drawing an observation from one group that is larger than the other is not 50:50.

To determine the incremental impact on subway ridership of rain versus no rain, I conducted two single variable regressions (with rain as the only predictor), one with log_ENTRIESn_hourly as the response variable and the other with ENTRIESn_hourly as the response variable (Figure 8 and Figure 9). Both models, yielded small positive coefficients for the rain coefficient, suggesting that more people tend to ride the subway when it rains. The log linear model predicted that we can expect on average 82 more turnstile entries for periods with rain (per 4 hour interval), while the linear model predicted an additional 182 more turnstile entries for periods with rain.

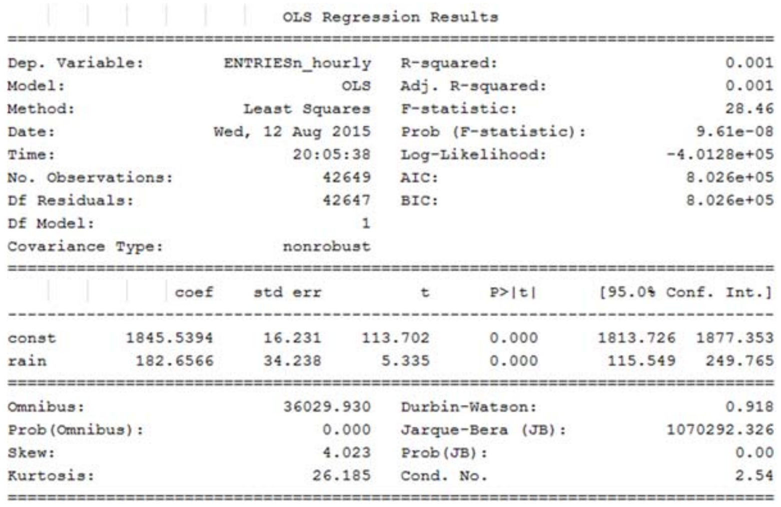
However, it's important not to lose sight of the fact that the small rain coefficients and R^2 in both models tell us that rain has a minor influence on ridership patterns when compared to the time and location related features discussed above.

Figure 8: log_ENTRIESn_hourly regression against rain dummy



```
                            OLS Regression Results
==============================================================================
Dep. Variable:     log_ENTRIESn_hourly   R-squared:                       0.001
Model:                             OLS   Adj. R-squared:                  0.001
Method:                  Least Squares   F-statistic:                     31.90
Date:                 Wed, 12 Aug 2015   Prob (F-statistic):           1.64e-08
Time:                         19:57:39   Log-Likelihood:                -85604.
No. Observations:                42649   AIC:                         1.712e+05
Df Residuals:                    42647   BIC:                         1.712e+05
Df Model:                            1
Covariance Type:             nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
const          6.4828      0.010    654.577      0.000         6.463     6.502
rain           0.1180      0.021      5.648      0.000         0.077     0.159
==============================================================================
Omnibus:                     8904.770   Durbin-Watson:                   0.813
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            19755.992
Skew:                          -1.202   Prob(JB):                         0.00
Kurtosis:                       5.309   Cond. No.                         2.54
==============================================================================
```

Figure 9: ENTRIESn_hourly regression against rain dummy



```
                            OLS Regression Results
==============================================================================
Dep. Variable:         ENTRIESn_hourly   R-squared:                       0.001
Model:                             OLS   Adj. R-squared:                  0.001
Method:                  Least Squares   F-statistic:                     28.46
Date:                 Wed, 12 Aug 2015   Prob (F-statistic):           9.61e-08
Time:                         20:05:38   Log-Likelihood:            -4.0128e+05
No. Observations:                42649   AIC:                         8.026e+05
Df Residuals:                    42647   BIC:                         8.026e+05
Df Model:                            1
Covariance Type:             nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
const       1845.5394     16.231    113.702      0.000      1813.726  1877.353
rain         182.6566     34.238      5.335      0.000       115.549   249.765
==============================================================================
Omnibus:                    36029.930   Durbin-Watson:                   0.918
Prob(Omnibus):                  0.000   Jarque-Bera (JB):          1070292.326
Skew:                           4.023   Prob(JB):                         0.00
Kurtosis:                      26.185   Cond. No.                         2.54
==============================================================================
```

6. REFLECTION

6.1. Please discuss potential shortcomings of the methods of your analysis, including:

The primary issue with my model is that it under predicts the occurrence of low value turnstile readings. Some of these prediction errors are related to seasonal/lurking variables that are not explicitly identified in the data. For example, modelling differences in off-peak subway schedules, changes in neighborhood economics/zoning, temporary line closures (mechanical failure), public holidays, and construction related building/station closures would likely improve my model's ability to predict low value turnstile readings.

Another issue mentioned above is the risk of overfitting my model to May historical data. Without data for a full 12 month cycle, my model will likely miss seasonal variations in the strength of response-predictor relationships (i.e., non-stationary correlations). For example, average turnstile traffic next to central park would likely be higher in July than January because outside temperatures usually fall to sub-zero levels in winter months.

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

For the original dataset, one had to standardize the time interval used in the denominator of the ENTRIESn_hourly calculation before conducting a regression. I think this would be a useful topic to cover in the next iteration of the course.

B. APPENDIX: PYTHON CODE

0. DATA SET HEADERS

turnistile_weather_v2.csv
rows = 42649
Header cols = 27

UNIT	Remote unit that collects turnstile information. Can collect from multiple banks of turnstiles. Large subway stations can have more than one unit.
DATEn	Date in “yyyy-mm-dd” (2011-05-21) format.
TIMEn	Time in “hh:mm:ss” (08:05:02) format.
ENTRIESn	Raw reading of cummulative turnstile entries from the remote unit. Occasionally resets to 0.
EXITSn	Raw reading of cummulative turnstile exits from the remote unit. Occasionally resets to 0.
ENTRIESn_hourly	Difference in ENTRIES from the previous REGULAR reading.
EXITSn_hourly	Difference in EXITS from the previous REGULAR reading.
datetime	Date and time in “yyyy-mm-dd hh:mm:ss” format (2011-05-01 00:00:00). Can be parsed into a Pandas datetime object without modifications.
hour	Hour of the timestamp from TIMEn. Truncated rather than rounded.
day_week	Integer (0 - 6 Mon - Sun) corresponding to the day of the week.
weekday	Indicator (0 or 1) if the date is a weekday (Mon - Fri).
station	Subway station corresponding to the remote unit.
latitude	Latitude of the subway station corresponding to the remote unit.
longitude	Longitude of the subway station corresponding to the remote unit.
conds	Categorical variable of the weather conditions (Clear, Cloudy etc.) for the time and location.
fog	Indicator (0 or 1) if there was fog at the time and location.
precipi	Precipitation in inches at the time and location.
pressurei	Barometric pressure in inches Hg at the time and location.
rain	Indicator (0 or 1) if rain occurred within the calendar day at the location.
tempi	Temperature in ℉ at the time and location.
wspdi	Wind speed in mph at the time and location.
meanprecipi	Daily average of precipi for the location.
meanpressurei	Daily average of pressurei for the location.
meantempi	Daily average of tempi for the location.
meanwspdi	Daily average of wspdi for the location.
weather_lat	Latitude of the weather station the weather data is from.
weather_lon	Longitude of the weather station the weather data is from.

1. STATISTICAL TEST

1.3. Mann Whitney U Test (scipy.stats)

```
import numpy as np
import scipy.stats as stats
import pandas as pd

def mann_whitney_plus_means(turnstile_weather):
    # load data and calculate sample means
    rain_ENTRIESn_hourly= turnstile_weather[turnstile_weather.rain==1].ENTRIESn_hourly
    no_rain_ENTRIESn_hourly = turnstile_weather[turnstile_weather.rain==0].ENTRIESn_hourly
    with_rain_mean = np.mean(rain_ENTRIESn_hourly)
    without_rain_mean = np.mean(no_rain_ENTRIESn_hourly)

    # calculate test statistic
    U, p = stats.mannwhitneyu(rain_ENTRIESn_hourly,no_rain_ENTRIESn_hourly)
    return with_rain_mean, without_rain_mean, U, 2*p
```

1.3. Mann Whitney U Test (normal approximation method)

```
import numpy as np
import scipy.stats as stats
import pandas as pd

def mann_whitney_plus_means(turnstile_weather):
    rain_ENTRIESn_hourly = turnstile_weather[turnstile_weather['rain']==1].ENTRIESn_hourly
    with_rain_mean = np.mean(rain_ENTRIESn_hourly)       # the mean of entries with rain
    without_rain_mean = np.mean(no_rain_ENTRIESn_hourly) # the mean of entries without rain
    U, p = stats.mannwhitneyu(rain_ENTRIESn_hourly, no_rain_ENTRIESn_hourly)
    n1, n2 = ((len(rain_ENTRIESn_hourly), len(no_rain_ENTRIESn_hourly)))
    m_u = n1*n2/2.0
    s_u = (n1*n2*(n1+n2+1)/12)**0.5
    z = (U-m_u)/s_u
    p = 2*stats.norm.cdf(z) #2-tailed p-value
    return with_rain_mean, without_rain_mean, U, p
```

2. LINEAR REGRESSION

2.1. Ordinary Least Squares log-linear model

```
import numpy as np
import pandas as pd
import scipy.stats as stats
import statsmodels.api as sm

def linear_regression(features, values):
    y = pd.DataFrame(values)                 # response variable
    x = pd.DataFrame(features)               # predictors  or features
    x = sm.add_constant(x)                   # adds a constant term to the predictors dataframe
    est = sm.OLS(y,x)                        # perform the regression of the predictors on the response
    est = est.fit()                          # estimate the parameters?
    with open('results_OLS_loglinear.txt', 'w') as f:
          f.write(str(est.summary()))        # write summary results to txt file
    intercept = est.params['const']          # assign constant
    params = est.params.ix[1:]               # assign features
    return intercept, params

def predictions(dataframe):
    dataframe['log_ENTRIESn_hourly'] = np.log1p(dataframe.ENTRIESn_hourly) # log transformation of response variable
    features = dataframe[[]] # option 2: features = dataframe[['meantempi', 'rain']]
    dummy_unit = pd.get_dummies(dataframe['UNIT'], prefix='unit')
    dummy_hour = pd.get_dummies(dataframe['hour'], prefix='hour')
    dummy_day_week = pd.get_dummies(dataframe['day_week'], prefix='day_week')
    features = features.join(dummy_hour).join(dummy_day_week).join(dummy_unit) #join(dummy_rain).

    # remove one dummy from each group to avoid non-unique solution dummy variable trap
    features.drop(['unit_R003'], axis = 1, inplace = True)
    features.drop(['hour_0'], axis = 1, inplace = True)
    features.drop(['day_week_0'], axis = 1, inplace = True)
    values = dataframe['ENTRIESn_hourly']
    values_log = dataframe['log_ENTRIESn_hourly']

    # Perform linear regression
    intercept, params = linear_regression(features, values_log)
    log_predictions = intercept + np.dot(features, params)
    log_predictions[log_predictions<0] = 1 # specify zero value prediction minimum
    predictions = np.expm1(log_predictions) # inverse logarithmic transformation to produce ENTRIESn_hourly
    residuals = values - predictions        # residuals_log = values_log - log_predictions

    return predictions
df = pd.read_csv(r"data/turnstile_weather_v2.csv")
predictions(df)
```

2.1. Stochastic Gradient Descent (SGD) with normalized features

```
import numpy as np
import pandas as pd
import scipy.stats as stats
from datetime import datetime
import matplotlib.pyplot as plt
import pylab
from sklearn.linear_model import SGDRegressor
import seaborn as sns

def normalize_features(features):           #decreases time it takes for SGD convergence
    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):
    intercept = norm_intercept - np.sum(means * norm_params / std_devs)
    params = norm_params / std_devs
    return intercept, params

def linear_regression_SGD(features, values):
    clf = SGDRegressor(loss='squared_loss', alpha = 0.00001, n_iter = 1000,
                       shuffle = True, random_state = 2000).fit(features,values)
    intercept = clf.intercept_
    params = clf.coef_
    return intercept, params

def predictions(dataframe):
    dataframe['log_ENTRIESn_hourly'] = np.log1p(dataframe.ENTRIESn_hourly) # log transformation
    features = dataframe[[]]    # option 2: features = dataframe[['meantempi', 'rain']]
    dummy_unit = pd.get_dummies(dataframe['UNIT'], prefix='unit')
    dummy_hour = pd.get_dummies(dataframe['hour'], prefix='hour')
    dummy_day_week = pd.get_dummies(dataframe['day_week'], prefix='day_week')
    features = features.join(dummy_hour).join(dummy_day_week).join(dummy_unit) #join(dummy_rain).

    # removing one dummy from each group to avoid dummy variable trap
    features.drop(['unit_R003'], axis = 1, inplace = True)
    features.drop(['hour_0'], axis = 1, inplace = True)
    features.drop(['day_week_0'], axis = 1, inplace = True)
    values = dataframe['ENTRIESn_hourly']
    values_log = dataframe['log_ENTRIESn_hourly']

    # Perform linear regression
    intercept, params = linear_regression_SGD(features, values_log)
    log_predictions = intercept + np.dot(features, params)
    log_predictions[log_predictions<0] = 1
    predictions = np.expm1(log_predictions) # inverse logarithmic transformation to produce ENTRIESn_hourly
    residuals = values - predictions

    return predictions

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

df = pd.read_csv(r"data/turnstile_weather_v2.csv")
print "r^2: ", compute_r_squared(df['ENTRIESn_hourly'], predictions(df))
```

3. VISUALIZATION

3.1. ENTRIESn_hourly Histogram Code

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

# import csv and separate data into rain and no rain dataframes
df = pd.read_csv("data/turnstile_weather_v2.csv", index_col=0)
df[df.ENTRIESn_hourly >= 6000] = 6000       # limit outliers
rain_df = df[df.rain==1]                    # create rainy days df
no_rain_df = df[df.rain==0]                 # create non rainy days df
```

```
# plot rain and no rain NYC subway traffic histograms
no_rain_df.ENTRIESn_hourly.hist(alpha=.5, bins=50, label='No Rain', color = 'darkgreen')
rain_df.ENTRIESn_hourly.hist(alpha=1, bins=50, label='Rain', color = 'lightblue')
plt.ylabel("Frequency")                     # add label to the y-axis
plt.xlabel("ENTRIESn_hourly")               # add label to the x-axis
plt.legend()                                # add legend
```

3.1. Log_ENTRIESn_hourly Histogram Code

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

# import csv, munge data, apply logarithmic transformation to ENTRIESn_hourly
df_zero = df[df.ENTRIESn_hourly == 0]
print df_zero.ENTRIESn_hourly.value_counts()
zero_hours = df_zero.hour.unique()
#df = df[df.ENTRIESn_hourly != 0]
df['log_ENTRIESn_hourly'] = np.log1p(df.ENTRIESn_hourly) #log transformation
pd.options.mode.chained_assignment = None
df.is_copy = False
rain_df = df[df.rain==1]
no_rain_df = df[df.rain==0]
df = pd.read_csv("data/turnstile_weather_v2.csv", index_col=0)


# plot rain and no rain NYC subway traffic histograms
no_rain_df.log_ENTRIESn_hourly.hist(alpha=.5, bins=50, label='No Rain', color = 'darkgreen')
rain_df.log_ENTRIESn_hourly.hist(alpha=1, bins=50, label='Rain', color = 'lightblue')
plt.ylabel("Frequency")                         # add label to the y-axis
plt.xlabel("Log_ENTRIESn_hourly")               # add label to the x-axis
plt.legend()                                    # add legend
```

3.2. Heat Map

```
# import csv, add a day of the week column, and groupy by
turnstile_df = pd.read_csv(r"data/turnstile_weather_v2.csv", index_col=0)
turnstile_df.rename(columns={'day_week': 'Day of the Week', 'hour': 'Time Period' }, inplace=True)

# assign more descriptive axis labels
grouped_hour_weekday = turnstile_df['ENTRIESn_hourly'].groupby(
  [turnstile_df['Day of the Week'],turnstile_df['Time Period']]).median() # average hourly entries grouped by hour
weekly_heat_map = grouped_hour_weekday.unstack().transpose()  # transpose matrix to put day of week on the x-axis

# assign alternate x and y axis titles
old_day = weekly_heat_map.columns
new_day = ['Mon','Tues','Wed','Thu','Fri','Sat','Sun']                                    # new x-axis labels
weekly_heat_map.rename(columns=dict(zip(weekly_heat_map.columns, new_day)), inplace=True)   # change column headers
old_hour = weekly_heat_map.index
new_hour = ['8pm - 12am','12am - 4am', '4am - 8am', '8am - 12pm', '12pm - 4pm', '4pm - 8pm']
weekly_heat_map.rename(index=dict(zip(weekly_heat_map.index, new_hour)), inplace=True)      # change row headers

# plot heatmap
sns.heatmap(weekly_heat_map, annot=True,  fmt='.0f', linewidths=.5, cmap = "Reds")
```