Data Analyst Nanodegree - Project 1

Analyzing the New York Subway Dataset

References:

- Book: Artificial Intelligence: A Modern Approach (Stuart Russell, Peter Norvig)
- Lecture notes: http://cs229.stanford.edu/notes/cs229-notes1.pdf
- Online book (MIT book in preparation on Deep Learning): http://www.iro.umontreal.ca/~bengioy/dlbook/
- Linear regression (gradient Descent) - http://spin.atomicobject.com/2014/06/24/gradient-descent-linear-regression/
- Mann-Whitney U test: http://en.wikipedia.org/wiki/Mann%E2%80%93Whitney_U_test
- Project reference for verification: https://github.com/allanbreyes/udacity-data-science/tree/master/p1
- Python Coding reference: https://github.com/allanbreyes/udacity-data-science/blob/master/p1/ps3/analyze.py

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?

For the given input data (turnstile_data_master_with_weather.csv) I used the two tailed Mann-Whitney U test.

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.

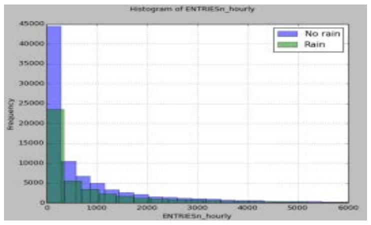

Figure 1

As shown in figure 1 - the “hourly entries” does not follow a normal distribution. The Mann-Whitney U test has greater efficiency than the t-test on non-normal distributions, such as a mixture of normal distributions, and it is nearly as efficient as the t-test on normal distributions.

Code snippet (full version attached – Project1_S1_exploredata.py)

```
    df['ENTRIESn_hourly'][df['rain'] == 0].hist(bins=bins, alpha=alpha)
    df['ENTRIESn_hourly'][df['rain'] == 1].hist(bins=bins, alpha=alpha)

    plt.axis([xmin, xmax, ymin, ymax])
    plt.suptitle('Histogram of ENTRIESn_hourly')
    plt.xlabel('ENTRIESn_hourly')
    plt.ylabel('Frequency')
    plt.legend(['No rain', 'Rain'])
```

1.2 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.

Results are as following:

```
Mann-Whitney U test:
(1105.4463767458733, 1090.278780151855, 1924409167.0, 0.019309634413792565)
```

Mean_non_rainy: 1090.28

Mean_rainy: 1105.45

U = 1924409167

p = 1.93%

Code snippet (full version attached – Project1_S1_exploredata.py)

```
    df_wet = df['ENTRIESn_hourly'][df['rain'] == 1]
    df_dry = df['ENTRIESn_hourly'][df['rain'] == 0]

    with_rain_mean = df_wet.mean()
    without_rain_mean = df_dry.mean()

    U, p = scipy.stats.mannwhitneyu(df_wet, df_dry)

    return with_rain_mean, without_rain_mean, U, p
```

1.4 What is the significance and interpretation of these results?

I find that the computed U value (1924409167) gives significant evidence that the two distributions differ (p < 0.05, two tailed). Therefore there is a significant difference between the number of hourly entries on rainy hours and non-rainy hours. We reject the H0 hypothesis - two samples come from the same population – since p (1.93%) is smaller than 5%

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?

I used Gradient Descent:

Code snippet (full version attached – Project1_S1_exploredata.py)

```
def compute_cost(features, values, theta):
    m = len(values)
    sum_of_square_errors = (np.square(np.dot(features, theta) - values)).sum()
    return sum_of_square_errors/2*m

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

    for i in range(num_iterations):
        predicted_values = np.dot(features, theta) - values
        theta = theta - (alpha/m) * np.dot(predicted_values, features)
        cost = compute_cost(features, values, theta)
        cost_history.append(cost)
    return theta, pandas.Series(cost_history
```

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

As input variables I used dataframe[['rain', 'precipi', 'meanwindspdi', 'meantempi']]. In addition I used 'UNIT' as dummy variables.

Code snippet (full version attached – Project1_S1_exploredata.py)

```
def predictions(dataframe):
    '''
    runs predictions via gradient descent on turnstile dataframe
    '''
    # Select Features (try different features!)
    features = dataframe[['rain', 'precipi', 'meanwindspdi', 'meantempi']]

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

    # 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 = 40 # 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)
    print theta_gradient_descent
    plot = plot_cost_history(alpha, cost_history)
    predictions = np.dot(features_array, theta_gradient_descent)
    return predictions, plot
```

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 did not use any Machine Learning techniques to test the quality of the selected features via training and test sets. Since the available amount of features was very limited. I used my common sense and selected basically all weather related features, assuming that these once will affect ridership lineally.

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

```
-4.20433898e+00, 7.78193688e+00, 4.55076972e+01, -3.27732228e+01
```

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

```
R-squared value: 0.419025381476
```

```
def compute_r_squared(data, predictions):

    r_squared = 1-(np.sum(np.square(data-predictions))/\
                np.sum(np.square(data-np.mean(data)))

    return r_squared
```

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

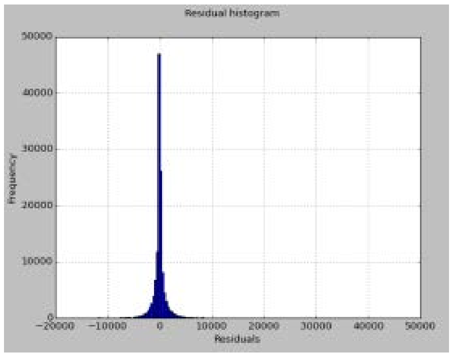

Figure 2

R^2 is a quantitative measure of the "goodness of fit", the closet R^2 is to 1 the more accurate is our statistical model. Based on a value of 0.42 for R^2 I would not think that this model is appropriate, but it depends on the use case. Nevertheless the residual plot (figure 2) shows that most of residuals are close to =/- 5000. Meaning for a "rough" estimate the linear model is sufficient.

Section 3: Visualization

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

Figure 3 shows that both distributions are not normally distributed. One could draw the conclusion from this graph that there is less ridership on rainy days, but since we look at aggregated data and we have less rainy days than non-rainy days – the conclusion would be wrong

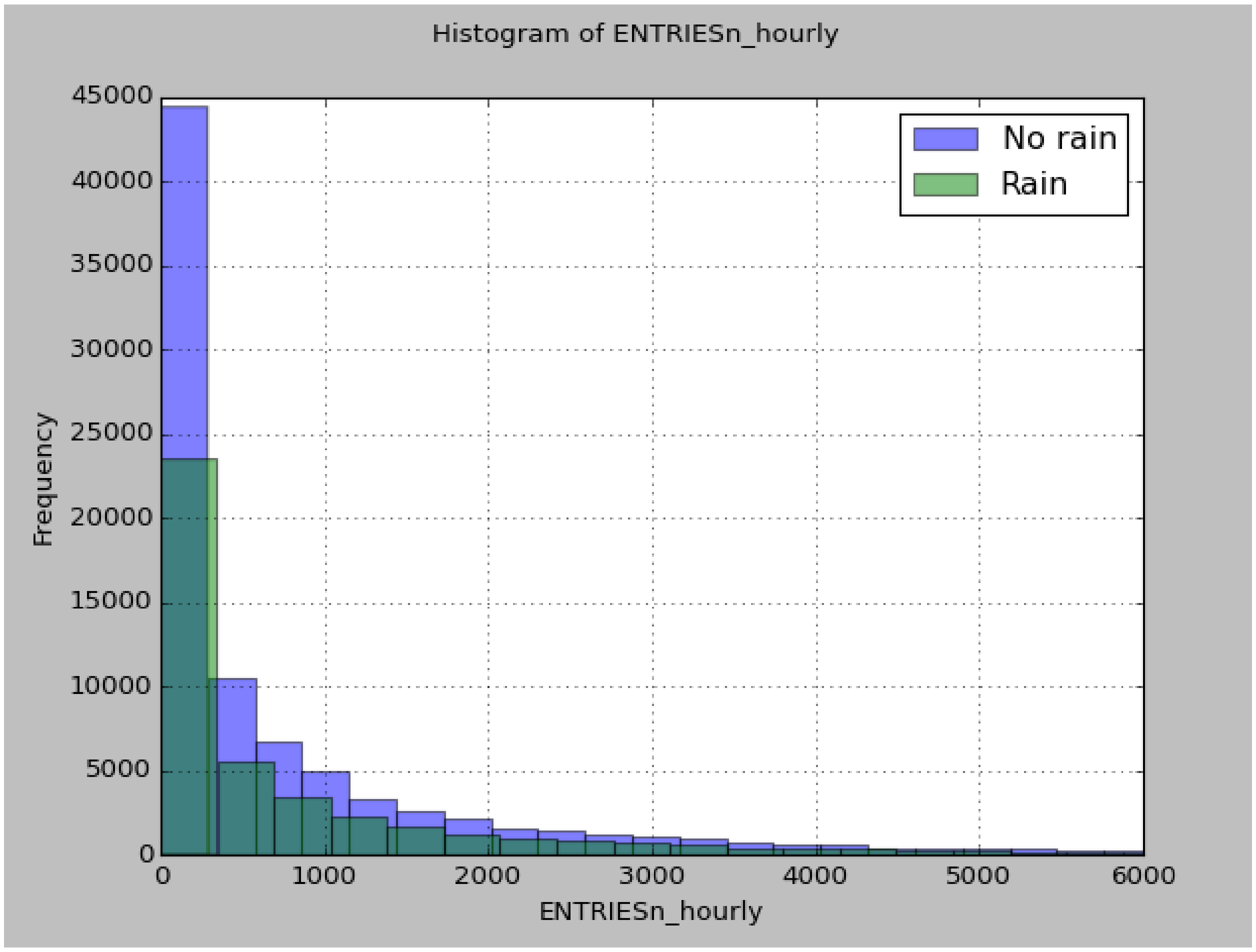

Figure 3

3.2 One visualization can be more freeform

Figure 4 shows the average number of subway entries at each hour. There are several peaks throughout the day, which raises some interesting questions:

- NY the cities which never sleeps? – seems not correct, if you define “sleeping” as entries below 10% of peak, NY sleeps a couples of hours.
- Peaks around 9am and 05:00pm are expected considering a “normal” working schedule, but peaks at 12:00pm at 08:00pm would need further investigation? Explanation could be people go out for lunch and dinner
- …

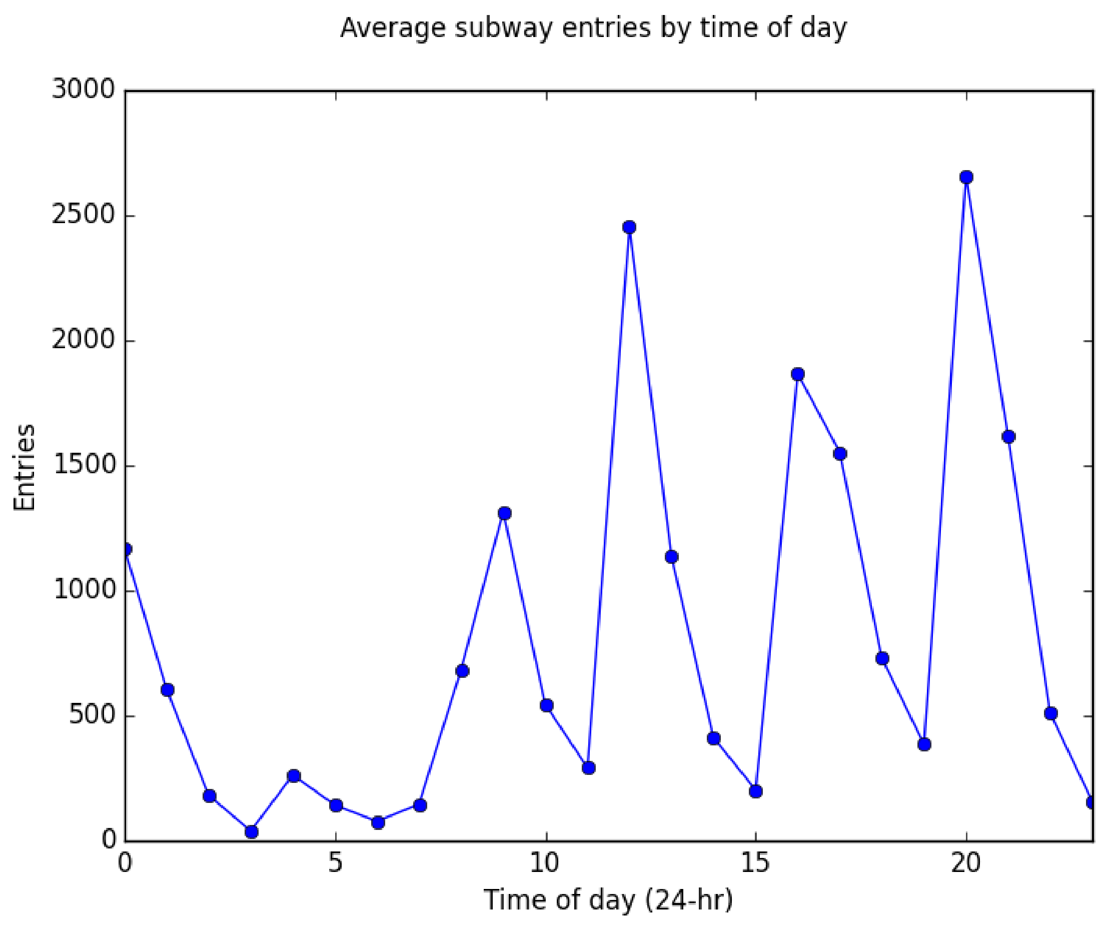

Figure 4

Section 4: Conclusion

4.1 & 4.2: 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? What analyses lead you to this conclusion? You should use results from both your statistical tests and your linear regression to support your analysis.

From the test I have done I can derive the following conclusions:

1) Based on the Mann-Whitney U test (p-value 0.0193) we can confirm that the two data sets are statically different and reject our H0 hypothesis. I can say with a high level of certainty that there are more people taking the NYC subway during rainy days.

2) If we include the R^2 value of 42% it remains unclear if rain and ridership have a linear relationship. This may not be valid for all data points. Looking at figure 2 the small residuals show however a good accuracy given our prediction objective.
   Looking at the means of both data sets we do not see a big difference, but the Mann-Whitney U test did show a statically significant difference. Therefore I stay with the conclusion that rain increases subway ridership.

Section 5: Reflection

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

One observation is that there is a big difference between entries and exits:

Sum of ENTRIESn_hourly	Sum of EXITSn_hourly
144532327	117026133

This indicates a miscount or data collection issues. But we can assume that the impact might be the same on rainy and non-rainy day.

Another observation is that we are struggling to find a clear correlation between the given variables. One explanation could be that we are mixing different station and hours of the day all together. Increasing the sample size and normalization by station would likely increase our confidence in the Mann-Whitney U test and linear regression model.

I used Aptana Studio 3 (PyDev) to run the code on my local machine, attached is the program:

```
'''
Created on May 30, 2015

@author: Georg Glantschnig
'''
from ggplot import *
import matplotlib.pyplot as plt
import numpy as np
import pandas
import scipy
import scipy.stats

# ps3.1
def entries_histogram(turnstile_weather, csv=False):
    '''
    plots two histograms on the same axes to show hourly entries when raining
    vs. when not raining.
    '''

    if csv:
        df = pandas.read_csv(turnstile_weather)
    else:
        df = turnstile_weather

    bins = 150
    alpha = 0.5
    xmin = ymin = 0
    xmax = 6000
    ymax = 45000

    plt.figure()

    df['ENTRIESn_hourly'][df['rain'] == 0].hist(bins=bins, alpha=alpha)
    df['ENTRIESn_hourly'][df['rain'] == 1].hist(bins=bins, alpha=alpha)

    plt.axis([xmin, xmax, ymin, ymax])
    plt.suptitle('Histogram of ENTRIESn_hourly')
    plt.xlabel('ENTRIESn_hourly')
    plt.ylabel('Frequency')
    plt.legend(['No rain', 'Rain'])

    return plt

# ps3.3
def mann_whitney_plus_means(turnstile_weather, csv=False):
    '''
    consumes the turnstile_weather dataframe (or csv file), and returns:
        1) the mean of entries with rain
        2) the mean of entries without rain
```

```
        3) the Mann-Whitney U-statistic and p-value comparing the number
           of entries with rain and the number of entries without rain
    '''
    if csv:
        df = pandas.read_csv(turnstile_weather)
    else:
        df = turnstile_weather

    df_wet = df['ENTRIESn_hourly'][df['rain'] == 1]
    df_dry = df['ENTRIESn_hourly'][df['rain'] == 0]

    with_rain_mean = df_wet.mean()
    without_rain_mean = df_dry.mean()

    U, p = scipy.stats.mannwhitneyu(df_wet, df_dry)

    return with_rain_mean, without_rain_mean, U, p

def normalize_features(array):
   '''
   normalizes the features in the data set.
   '''
   mu = array.mean()
   sigma = array.std()
   array_normalized = (array - mu)/sigma

   return array_normalized, mu, sigma

def compute_cost(features, values, theta):
    '''
    computes 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()
    return sum_of_square_errors/2*m

def gradient_descent(features, values, theta, alpha, num_iterations):
    '''
    performs gradient descent given a data set with an arbitrary number
    of features.
    '''
    m = len(values)
    cost_history = []

    for i in range(num_iterations):
        predicted_values = np.dot(features, theta) - values
        theta = theta - (alpha/m) * np.dot(predicted_values, features)
        cost = compute_cost(features, values, theta)
        cost_history.append(cost)
    return theta, pandas.Series(cost_history)
```

```python
def predictions(dataframe):
    '''
    runs predictions via gradient descent on turnstile dataframe
    '''
    # Select Features (try different features!)
    features = dataframe[['rain', 'precipi', 'meanwindspdi', 'meantempi']]

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

    # 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 = 40 # 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)
    print theta_gradient_descent
    plot = plot_cost_history(alpha, cost_history)
    predictions = np.dot(features_array, theta_gradient_descent)
    return predictions, plot


def plot_cost_history(alpha, cost_history):
    '''
    returns plot of the cost history
    '''
    cost_df = pandas.DataFrame({
        'Cost_History': cost_history,
        'Iteration': range(len(cost_history))
    })
    return ggplot(cost_df, aes('Iteration', 'Cost_History')) + \
        geom_point() + ggtitle('Cost History for alpha = %.3f' % alpha )

def plot_residuals(turnstile_weather, predictions):
```

```python
    '''
    makes a histogram of the residuals, the difference between the original
    hourly entry data and the predicted values)
    '''

    plt.figure()
    (turnstile_weather['ENTRIESn_hourly'] - predictions).hist(bins=150)
    plt.suptitle('Residual histogram')
    plt.xlabel('Residuals')
    plt.ylabel('Frequency')
    return plt

def compute_r_squared(data, predictions):
    '''
    Calculate R square -- the coefficient of determination. The closer to one,
    the better the model.
    Given a list of original data points, and also a list of predicted data
    points, write a function that will compute and return the coefficient of
    determination (R^2) for this data.  numpy.mean() and numpy.sum() might both
    be useful here, but not necessary.
    Documentation about numpy.mean() and numpy.sum() below:
    http://docs.scipy.org/doc/numpy/reference/generated/numpy.mean.html
    http://docs.scipy.org/doc/numpy/reference/generated/numpy.sum.html
    '''

    r_squared = 1-(np.sum(np.square(data-predictions)))/\
                np.sum(np.square(data-np.mean(data)))

    return r_squared

if __name__ == '__main__':
    filename = 'turnstile_data_master_with_weather.csv'
    df = pandas.DataFrame.from_csv(filename)

    print "Histogram of turnstile data:"
    entries_histogram(filename, True)
    plt.show()
    raw_input("Press enter to continue...")

    print "Mann-Whitney U test:"
    print mann_whitney_plus_means(filename, csv=True)
    raw_input("Press enter to continue...")

    print "Linear regression predictions via gradient descent:"
    predicted, plot = predictions(df)
    print plot
    raw_input("Press enter to continue...")

    print "Plotting residuals:"
    plot_residuals(df, predicted)
    plt.show()
    raw_input("Press enter to continue...")
```

```
    print "R-squared value:"
    print compute_r_squared(df['ENTRIESn_hourly'], predicted)
```