Appendix:

Prerequisites

Background: Mathematics, Statistics and Intro to Data Science.
Understanding of the NYC Ridership data and weather data is required.

Required Packages:

- Pandas
- Numpy
- Scipy
- Statsmodels
- ggplot
- Matplotlib
- Pandasql

Data:

- MTA Subway turnstile text (from May 2011 period) file can be seen at the URL below:
  http://web.mta.info/developers/data/nyct/turnstile/turnstile_110507.txt
- For the same period you can obtain the weather data that we are passing in below:
  https://www.dropbox.com/s/7sf0yqc9ykpq3w8/weather_underground.csv
- Getting and cleaning raw data is a time consuming process, so to save us time we are given the cleaned up combined dataset here:
  https://www.dropbox.com/s/meyki2wl9xfa7yk/turnstile_data_master_with_weather.csv
- Data definition: http://web.mta.info/developers/resources/nyct/turnstile/ts_Field%20Description.txt

References:

Coefficient of determination → http://en.wikipedia.org/wiki/Coefficient_of_determination
Residual Plot → http://www.itl.nist.gov/div898/handbook/pri/section2/pri24.htm
MapReduce → http://en.wikipedia.org/wiki/MapReduce
Information about normal distribution → http://en.wikipedia.org/wiki/Normal_distribution
Polynomial Regression → http://en.wikipedia.org/wiki/Polynomial_regression

Definitions:

Null hypothesis →
The null hypothesis is that the distributions of both groups are identical, so that there is a 50% probability that an observation from a value randomly selected from one population exceeds an observation randomly selected from the other population.

The P value answers this question:
If the groups are sampled from populations with identical distributions, what is the chance that random sampling would result in the mean ranks being as far apart (or more so) as observed in this experiment?
If the P value is small, you can reject the null hypothesis that the difference is due to random sampling, and conclude instead that the populations are distinct.

Code Section:

Linear regression by gradient descent :

```
import numpy as np
import pandas

def normalize_features(array):
  """
  Normalize the features in our data set.
  """
```

```
    array_normalized = (array-array.mean())/array.std()
    mu = array.mean()
    sigma = array.std()
    return array_normalized, mu, sigma

def compute_cost(features, values, theta):
    """
    Compute the cost function given a set of features / values, and the values for our thetas.
    This should be the same code as the compute_cost function in the lesson #3 exercises. But     feel free to implement your own.
    """
    m = len(values) * 1.0
    sum_of_square_errors = np.square(np.dot(features, theta) - values).sum()
    cost = sum_of_square_errors / (2*m)
    return cost

def gradient_descent(features, values, theta, alpha, num_iterations):
    """
    Perform gradient descent given a data set with an arbitrary number of features.
    This is the same gradient descent code as in the lesson #3 exercises. But feel free to implement your own.
    """
    m = len(values) * 1.0
    cost_history = []

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

def predictions(dataframe):
    '''
    The NYC turnstile data is stored in a pandas dataframe called weather_turnstile. Using the information stored in the dataframe, lets predict the ridership of the
NYC subway using linear regression with gradient descent.
     You can look at information contained in the turnstile weather dataframe  at the link below:
    https://www.dropbox.com/s/meyki2wl9xfa7yk/turnstile_data_master_with_weather.csv
     '''
    dummy_units = pandas.get_dummies(dataframe['UNIT'], prefix='unit')
    features = dataframe[['rain', 'precipi', 'Hour', 'meantempi']].join(dummy_units)
    values = dataframe[['ENTRIESn_hourly']]
    m = len(values)
    features, mu, sigma = normalize_features(features)
    features['ones'] = np.ones(m)
    features_array = np.array(features)
    values_array = np.array(values).flatten()

    #Set values for alpha, number of iterations.
    alpha = 0.5 # please feel free to play with this value
    num_iterations = 75 # please feel free to play with 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)
    predictions = np.dot(features_array, theta_gradient_descent)
    return predictions
```

Residual Plot

```
import numpy as np
import scipy
import matplotlib.pyplot as plt
```

```
def plot_residuals(turnstile_weather, predictions):
    '''
    Using the same methods that we used to plot a histogram of entries per hour for our data, why don't you make a histogram of the residuals
    (that is, the difference between the original hourly entry data and the predicted values).
    Based on this residual histogram, do you have any insight into how our model performed?  Reading a bit on this webpage might be useful:
    http://www.itl.nist.gov/div898/handbook/pri/section2/pri24.htm
    '''
     plt.figure()
    (turnstile_weather['ENTRIESn_hourly'] - predictions).hist(bins=50)

    residuals = turnstile_weather['ENTRIESn_hourly'] - predictions
    (zscore, pvalue) = scipy.stats.normaltest(residuals)
    print "p-value of normaltest on residuals: %s" % pvalue
    print "mean of residuals: %s" % residuals.mean()
    plt.xlabel("Residuals")
    plt.ylabel("Frequency")
    plt.title("Residual Plot")
```

Computing r-squared:

```
import numpy as np
import scipy
import matplotlib.pyplot as plt
import sys

def compute_r_squared(data, predictions):
    '''
    In exercise 5, we calculated the R^2 value for you. But why don't you try and and calculate the R^2 value yourself.
    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
    '''
    _mean = np.mean(data)
    _y    = np.sum((predictions - data) ** 2)
    _f    = np.sum((data - _mean) ** 2)
    r_squared = 1 - _y/_f
    return r_squared
```

Computing Polynomial Regression:

**Along with below mentioned functions, the polynomial regression also uses normalize_features(array),gradient_descent(features, values, theta, alpha, num_iterations), compute_cost(features, values, theta) and compute_r_squared(data, predictions)*

```
def predictions(features, values, alpha, num_iterations):
    m = len(values) * 1.0
    theta = np.zeros(features.shape[1])
    theta, cost_history = gradient_descent(features, values, theta, alpha, num_iterations)
    predictions = features.dot(theta)
    predictions[predictions<0] = 0
    return pd.Series(predictions)


def add_poly_features(df, degree, add_sqrt):
    for i in xrange(2, degree + 1):
        for combination in itertools.combinations_with_replacement(df.columns,i):
            name = " ".join(combination)
            value = np.prod(df[list(combination)], axis=1)
```

```
            df[name] = value
    if add_sqrt:
        for column in df.columns:
            df["%s_sqrt" % column] = np.sqrt(df[column])

dummy_units = pd.get_dummies(turnstile_data['UNIT'], prefix='unit')
features = turnstile_data[['Hour','minpressurei','meanpressurei', 'maxpressurei','rain', 'meanwindspdi','mintempi','meantempi', 'maxtempi','mindewpti',
'maxdewpti','meandewpti', 'precipi' ]]
add_poly_features(features, 2, add_sqrt=True)
features = features.join(dummy_units)
features, mu, sigma = normalize_features(features)
features = np.array(features)
features = np.insert(features, 0, 1, axis=1)
values = np.array(turnstile_data[['ENTRIESn_hourly']]).flatten()
pred = predictions(features=np.array(features),values=values,alpha=0.025,num_iterations=300)
print("by adding more of existing features with polynomial combinations R^2: %.3f" %compute_r_squared(turnstile_data["ENTRIESn_hourly"], pred))
```