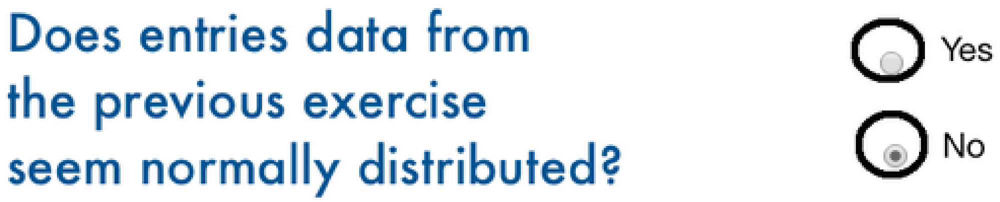
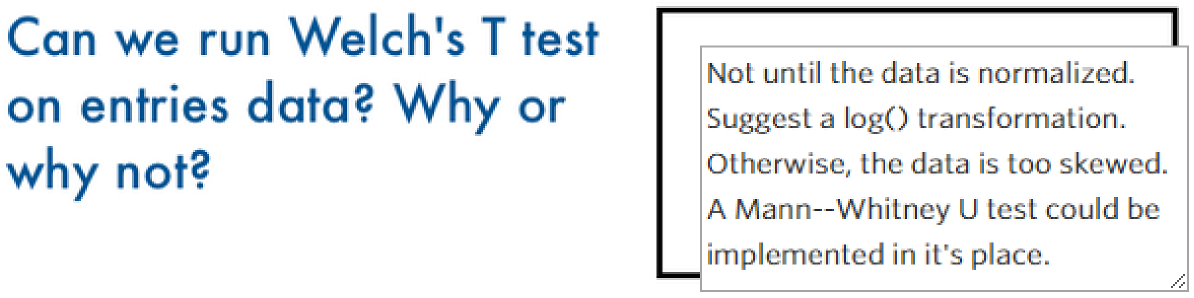
Problem Set 3: Analyzing Subway Data   Miadad Rashid   2015.2.24

```
Problem Set 3.1
import numpy as np
import pandas
import matplotlib.pyplot as plt

def entries_histogram(turnstile_weather):


    plt.figure()
    turnstile_weather[turnstile_weather['rain'] == 1]['ENTRIESn_hourly'].hist() # your code here to plot a historgram for hourly entries when it is raining
    turnstile_weather[turnstile_weather['rain'] == 0]['ENTRIESn_hourly'].hist() # your code here to plot a historgram for hourly entries when it is not raining
    return plt


Problem Set 3.2
```



Does entries data from the previous exercise seem normally distributed?

Yes [✓ No]



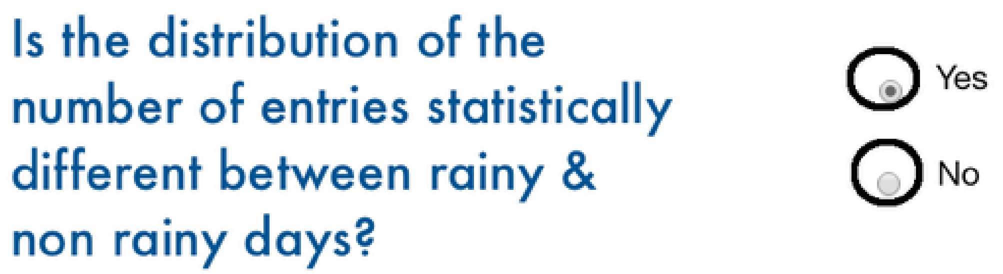
Can we run Welch's T test on entries data? Why or why not?

Not until the data is normalized. Suggest a log() transformation. Otherwise, the data is too skewed. A Mann--Whitney U test could be implemented in it's place.

Problem Set 3.3

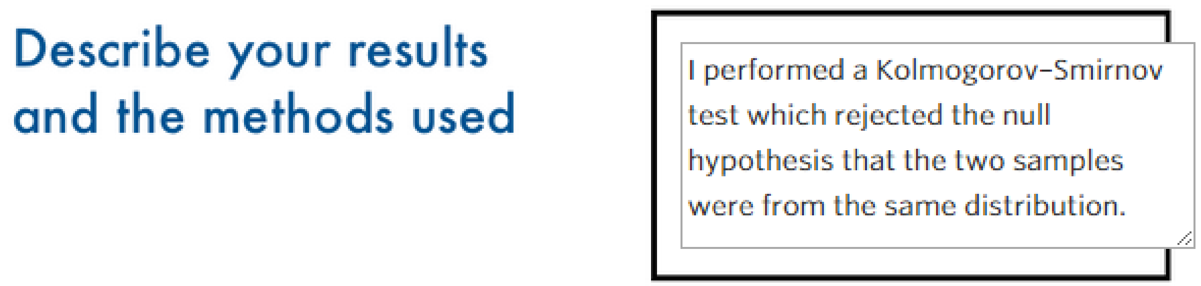
```python
import numpy as np
import scipy
import scipy.stats
import pandas

def mann_whitney_plus_means(turnstile_weather):

    ### YOUR CODE HERE ###
    with_rain_mean = np.mean(turnstile_weather[turnstile_weather['rain'] == 1]['ENTRIESn_hourly'])
    without_rain_mean = np.mean(turnstile_weather[turnstile_weather['rain'] == 0]['ENTRIESn_hourly'])
    U, p = scipy.stats.mannwhitneyu(turnstile_weather[turnstile_weather['rain'] == 1]['ENTRIESn_hourly'],
                                    turnstile_weather[turnstile_weather['rain'] == 0]['ENTRIESn_hourly'])


    return with_rain_mean, without_rain_mean, U, p # leave this line for the grader
```

Problem Set 3.4



Is the distribution of the number of entries statistically different between rainy & non rainy days?

[✓ Yes] No



Describe your results and the methods used

I performed a Kolmogorov-Smirnov test which rejected the null hypothesis that the two samples were from the same distribution.

```
Problem Set 3.5

import numpy as np
import pandas
from ggplot import *


def normalize_features(df):
    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

def compute_cost(features, values, theta):

    # your code here
    cost = np.square(np.dot(features, theta) - values).sum()

    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 can be 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):
              # your code here
              update = compute_cost(features, values, theta)
              cost_history.append(update)
              theta = theta + (1/m) * alpha * np.dot((values - np.dot(features, theta)), features)
          return theta, pandas.Series(cost_history)

      def predictions(dataframe):
          # Select Features (try different features!)
          features = dataframe[['Hour', 'maxdewpti', 'maxtempi', 'mindewpti']]

          # 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)
```

```python

    # 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 = 0.3 # please feel free to change this value
    num_iterations = 100 # 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


def plot_cost_history(alpha, cost_history):
    """This function is for viewing the plot of your cost history.
    You can run it by uncommenting this

        plot_cost_history(alpha, cost_history)

    call in predictions.

    If you want to run this locally, you should print the return value
    from this function.
    """
    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 )
```

```
Problem Set 3.6

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

def plot_residuals(turnstile_weather, predictions):

    plt.figure()
    (turnstile_weather['ENTRIESn_hourly'] - predictions).hist()
    return plt


Problem Set 3.7

import numpy as np
import scipy
import matplotlib.pyplot as plt
import sys
def compute_r_squared(data, predictions):
   r_squared = 1 - ((np.square(data - predictions).sum())/(np.square(data - np.mean(data)).sum()))

    return r_squared


Problem Set 3.8

import numpy as np
import pandas
import scipy
import statsmodels.api as sm
import datetime

def predictions(weather_turnstile):
    #
    # Your implementation goes here. Feel free to write additional
    # helper functions
    #
    df = weather_turnstile
    dummyunit = pandas.get_dummies(df['UNIT'])
    dummyhour = pandas.get_dummies(df['Hour'])
    features = dummyunit.join([dummyhour])

    entries = df['ENTRIESn_hourly']
    prediction = sm.OLS(entries, features).fit().predict(features)

    return prediction
```