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.

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.

Section 1. Statistical Test

1.1

- ❖ **Which statistical test did you use to analyse the NYC subway data?**
  - I used a Mann—Whitney *U* test comparing the distributions of ENTRIESn_hourly variable between `rain == 1` and `rain == 0` data of the `turnstile_weather.csv` dataset.
  - I also conducted a *T* test; dealing with the non-normal distribution of rain/norain by adjusting the data <- log(data + 1).
    (a) + 1 was to prevent +-inf values from values of 0
    (b) The data looked more like a normal distribution, and the *T* test yielded an even more significant result (below – Section 1.3)

- ❖ **Did you use a one-tail or a two-tail P value?**
  - I used a two-tail P value because we are not defining a difference in any particular direction (less/greater than), instead, we are looking to see if there are differences at all.

(continued)

- **What is the null hypothesis?**
  - The null hypothesis is that both populations (rain and no-rain) are the same.

- **What is your p-critical value?**
  - I used a p-critical value of 0.05

1.2

- **Why is this statistical test applicable to the dataset?**
  - After creating histograms for both distributions, it was apparent that the distribution is non-normal; skewed towards zero. The *U* test is ideal on non-normal distributions over the *T* test by using the sum of the ranks on both sets of observations and comparing them, rather than taking the observations at face-value.

- **In particular, consider the assumptions that the test is making about the distribution of ridership inthe two samples.**
  - The observations from both groups are independent of each other
  - The responses are ordinal
  - Under the null hypothesis, the distributions are equal
  - Under the alternative hypothesis, the probability of one group's observation exceeding the others is not equal to 0.5

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.**
  - mean(rain) : 1105.446
  - mean(norain) : 1090.279
  - *U* test – two-sided p-value (R) : 0.04988
  - *U test* – two-sided p-value (Python) : 0.03862
  - *T test* – two-sided p-value (R, Python; log(data + 1) adjusted) : 0.002519

1.4

- **What is the significance and interpretation of these results?**
  - The critical value is set to be the point at which you can reject the null hypothesis that the two distributions are the same
  - The critical value was set to 0.05 and both results (R, Python) yields results that reject the null hypothesis.
  - The distributions are not the same.

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 Ordinary Least Squares to make the model

2.2

- ❖ **What features (input variables) did you use in your model? Did you use any dummy variables as part of your features?**
  - UNIT, DATEn, Hour
  - All of my variables were dummy variables

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 removed variables that had no variance (DESCn, thunder), that were redundant (TIMEn is represented by Hour), and the indexing variable (X).
  - We are left with [UNIT, DATEn, Hour, maxpressurei, maxdewpti, mindewpti, minpressurei, meandewpti, meanpressurei, fog, rain, meanwindspdi, mintempi]
  - Studying the coefficients of the OLS model on the variables reveal multicollinearity issues

    a) While [UNIT, DATEn, Hour] had coefficients, the rest returned NA for coefficients

    b) This means…
      - that the these variables can be linearly predicted from the other variables in the model
      - these variables are highly correlated to other variables and provide no significant improvement to the accuracy of prediction

2.4

- ❖ **What are the coefficients (or weights) of the non-dummy features in your linear regression model?**
  - I did not use non-dummy features

2.5

- ❖ **What is your model's R^2 (coefficients of determination) value?**
  - 0.5155325

2.6

- ❖ **What does this R^2 value mean for the goodness of fit for your regression model?**
  - The model explains ~51% of the total variation in the data
- ❖ **Do you think this linear model to predict ridership is appropriate for this dataset, given this R^2 value?**
  - Depends on the level of accuracy required
  - I believe this is the maximum R^2 achievable considering none of the other variables in the set adds to the predictability of the model
  - If the required model only needed to predict a range of entries, i.e.3000-4000, I would break those values in to categories and run a classification tree based model which would most definitely increase the accuracy because only a 'ballpark' is needed.

Section 3. Visualization

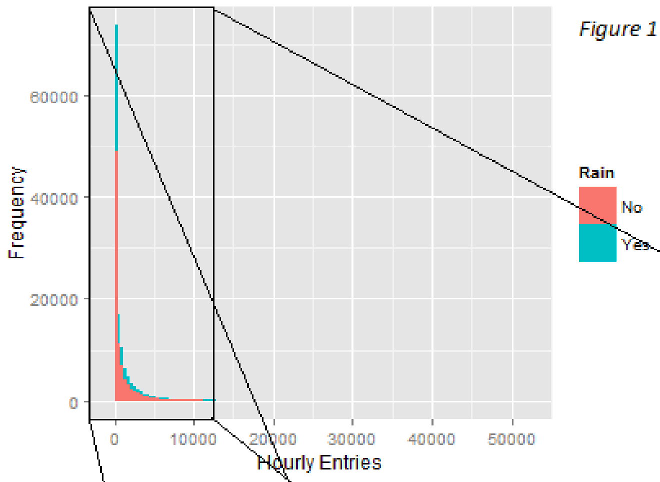


Figure 1

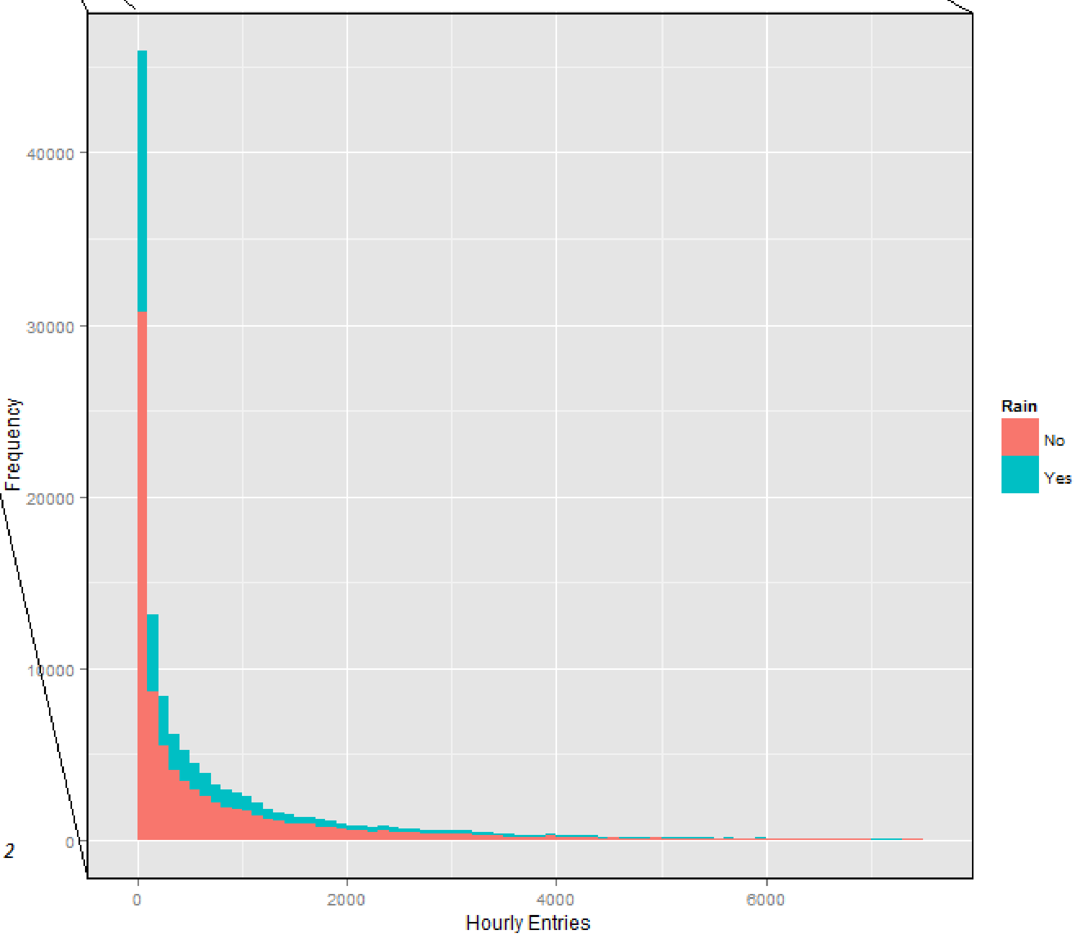


Figure 2

Figure 1 shows a histogram of the dataset colored by 'Rain'.

Figure 2 zooms in on *Figure 1* to show how skewed the distribution is.

This key feature of the data led me to use Mann-Whitney U test over the T test.

(continued)

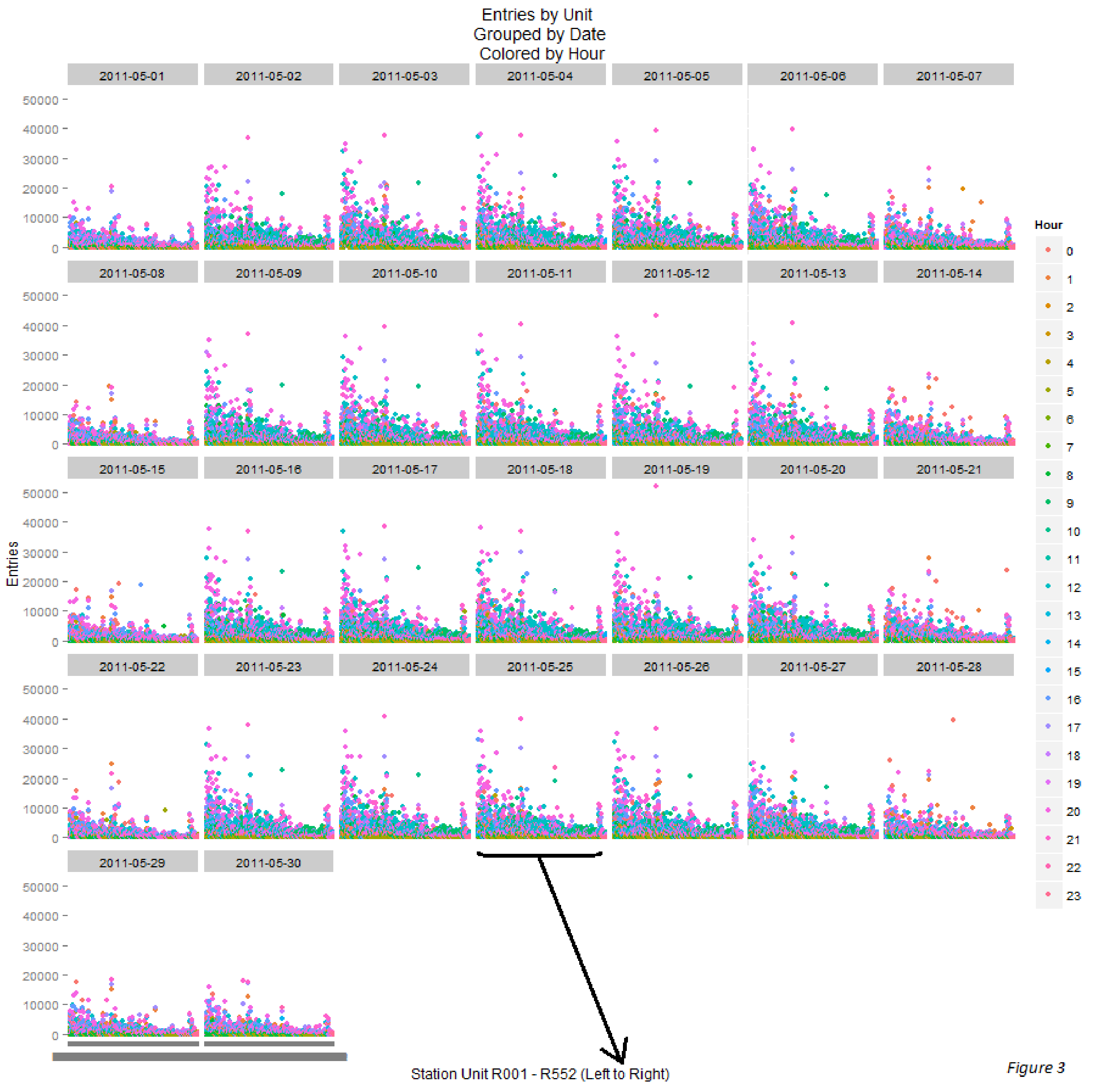


Figure 3

Figure 3 shows…

- Stations with lower R# (left of plot) had more entries
- Weekend riders rode later in the evening than Weekday riders
- More people rode in the weekday on average than the weekends
- Ridership categorically shifts between the days, not linearly

Although this visualization offered many great insights into patterns in ridership, entering Weekends as a categorical variable did not offer any increase in the model's Rsquared, so it was not included in the final model.

However, entering the Hour variable as a dummy increased accuracy considerably.

Section 4. Conclusion

Please address the following questions in detail. Your answers should be 1-2 paragraphs long.

4.1

- **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?**

  If you are considering the means of both categories (rain = 1105.446, no-rain = 1090.279), it would seem that more people on average use the subway when it is raining. If the question has to do with absolute values, 95.7 million rode when there was no rain in the month of May vs. 48.7 million when there was rain. This information alone should be used to draw any conclusions since there is close to double the observations of no-rain days over rain days.

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 Mann—Whitney *U* Test with a p-critical value of 0.05 rejected the null hypothesis that the distributions are the same with a p-value of 0.04988. I also adjusted for the skewness of the ENTRIESn_hourly frequency table by using the log(data + 1). This created a normal distribution without +-inf values. This allowed me to confidently apply a *T* test which also rejected the null hypothesis that the distributions are the same with a p-value of 0.002519. A *U* test p-value for mean(norain) < mean(rain) = 0.02494, meaning with a critical value of 0.05, mean(norain) < mean(rain) is True.

  The linear regression model pointed out that if the end goal is to simply predict the hourly entries, utilizing data pertaining to rain/norain did not offer any substantial accuracy to the data, in fact, the rain/norain variable showed strong colinearity with the categorical variables. Meaning, the values in rain/norain could be explained linearly by the other variables. In conclusion, although the occurrence of rain does increase the use of NYC subways, utilizing rain as a variable will not help the predictability of the model.

Section 5. Reflection

Please address the following questions in detail. Your answers should be 1-2 paragraphs long.

5.1

- **Please discuss potential shortcomings of the methods of your analysis, including:**
  - **1. Dataset**
    - The majority of the variables were not used to find the best prediction model
    - Intuitively, weather should affect the use of the subway, but according to the model, weather variables offered no increase in accuracy
  - **2. Analysis, such as the linear regression model or statistical test.**
    - Rsquared only accounted for ~51% of the variance
      - This would indicate that there are missing variables that would increase accuracy
    - I would have liked to have used a classification tree on this model
    - A tree was out of the question; required too much computer resource
    - In real world applications, a range for the entries predicted would be sufficient versus an exact value.
    - A classification tree would not only allow for a higher accuracy, but it would point key variables at different levels of the tree so not only would one be able to predict the entries, but potential city planners could control the entries themselves, preventing 'bottlenecks' and creating more efficient flow through the subway

5.2

- **(Optional) Do you have any other insight about the dataset that you would like to share with us?**
  - Maybe I missed a way to manipulate the data, but I would have loved to have a dataset where the majority of the variables were relevant to the final prediction model. After utilizing the dummy-variables, and identifying the rest of the variables as collinear, there seemed to be nothing left to do. At least nothing that the class taught us to look out for.
  - I would have also liked a dataset where there existed multiple relationships where multiple outcome variables could have been predicted. For example, this dataset could have added a *traffic_incidents* variable. Multiple relationships could be drawn between the existing variables and the new variable. Weather conditions could have causal effects on *traffic_incidents* and then weather becomes more relevant to the dataset rather than weather's effect on subway entry. *traffic_incidents* could mean traffic blockage causing more people to use the subway, or conversely, higher subway entries may mean less people are driving therefore there are less *traffic_incidents*. This would be a fun dataset I could probably get lost in for a while. Overall, this exercise was great and I hope there are more fun projects like this along the way.

Helpful webpages
This site, offers a lot of 'hows' of running non-parametric and parametric tests after applying linear regression
http://connor-johnson.com/2014/02/18/linear-regression-with-python/
This site goes over the basics of Ordinary Least Squares
http://www.datarobot.com/blog/ordinary-least-squares-in-python/
Everything you need to know about SQL queries
http://www.w3schools.com/sql/

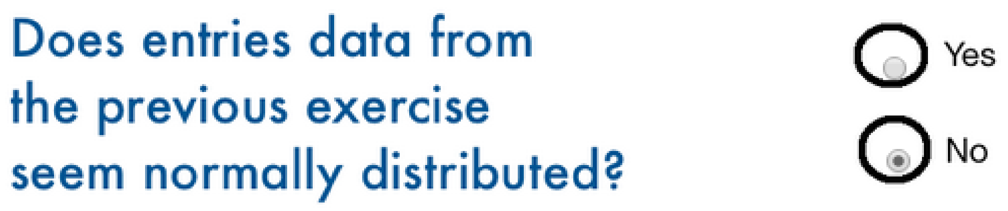
```
Problem Set 2.1

import pandas
import pandasql


def num_rainy_days(filename):
    weather_data = pandas.read_csv(filename)

    q = """
    SELECT COUNT(*)
    FROM weather_data
    WHERE cast(rain as integer) = 1;
    """

    #Execute your SQL command against the pandas frame
    rainy_days = pandasql.sqldf(q.lower(), locals())
    return rainy_days


Problem Set 2.2

import pandas
import pandasql


def max_temp_aggregate_by_fog(filename):
        weather_data = pandas.read_csv(filename)

    q = """
    SELECT fog, MAX(cast (maxtempi as integer))
    FROM weather_data
    GROUP BY fog;
    """

    #Execute your SQL command against the pandas frame
    foggy_days = pandasql.sqldf(q.lower(), locals())
    return foggy_days

Problem Set 2.3

import pandas
import pandasql

def avg_weekend_temperature(filename):
    weather_data = pandas.read_csv(filename)

    q = """
    SELECT avg(cast (meantempi as integer))
    FROM weather_data
    WHERE cast (strftime('%w', date) as integer) IN (6,0);
    """

    #Execute your SQL command against the pandas frame
    mean_temp_weekends = pandasql.sqldf(q.lower(), locals())
    return mean_temp_weekends
```

```
Problem Set 2.4

import pandas
import pandasql

def avg_min_temperature(filename):
    weather_data = pandas.read_csv(filename)

    q = """
    SELECT avg(cast (mintempi as integer))
    FROM weather_data
    WHERE cast(rain as integer) = 1
    AND cast(mintempi as integer) > 55;
    """

    #Execute your SQL command against the pandas frame
    avg_min_temp_rainy = pandasql.sqldf(q.lower(), locals())
    return avg_min_temp_rainy




Problem Set 2.5

import csv

def fix_turnstile_data(filenames):
    for name in filenames:
        f_in = open(name, 'r')
        f_out = open('updated_'+name, 'w')
        reader_in = csv.reader(f_in, delimiter=',')
        writer_out = csv.writer(f_out, delimiter=',')
        for x in reader_in:
            c1 = x[0]
            c2 = x[1]
            c3 = x[2]

            for k in range(len(x)/5):
                      out = [c1, c2, c3,
                             x[k*5 + 3],
                             x[k*5 + 4],
                             x[k*5 + 5],
                             x[k*5 + 6],
                             x[k*5 + 7]]
                      writer_out.writerow(out)



      Problem Set 2.6

      def create_master_turnstile_file(filenames, output_file):
          with open(output_file, 'w') as master_file:
             master_file.write('C/A,UNIT,SCP,DATEn,TIMEn,DESCn,ENTRIESn,EXITSn\n')
             for filename in filenames:
                  f_in = open(filename, 'r')
                  f_in.next()
                  for line in f_in:
                      master_file.write(line)
                  f_in.close


```

```
Problem Set 2.7

import pandas

def filter_by_regular(filename):

    turnstile_data = pandas.read_csv(filename)
    turnstile_data = turnstile_data[turnstile_data['DESCn'] == 'REGULAR']
    # your code here
    # more of your code here

    return turnstile_data


Problem Set 2.8

import pandas

def get_hourly_entries(df):

    df['ENTRIESn_hourly'] = (df['ENTRIESn'] -  df['ENTRIESn'].shift(1)).fillna(1)
    return df


Problem Set 2.9

import pandas

def get_hourly_exits(df):

    df['EXITSn_hourly'] = (df['EXITSn'] - df['EXITSn'].shift(1)).fillna(0)
    return df


Problem Set 2.10

import pandas

def time_to_hour(time):

    hour = int(time[0:2])# your code here
    return hour


Problem Set 2.11

import datetime

def reformat_subway_dates(date):

    date_formatted = datetime.datetime.strptime(date, "%m-%d-%y").strftime("%Y-%m-%d")# your code here
    return date_formatted
```

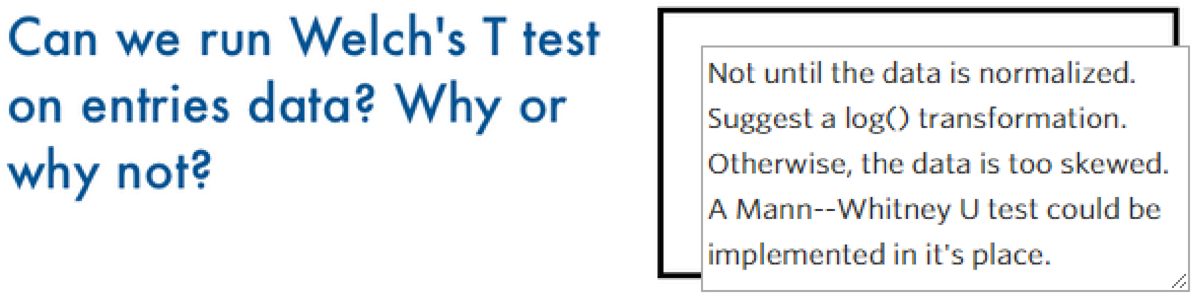
Problem Set 3: Analyzing Subway Data Miadad Rashid 2015.2.24

```
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?

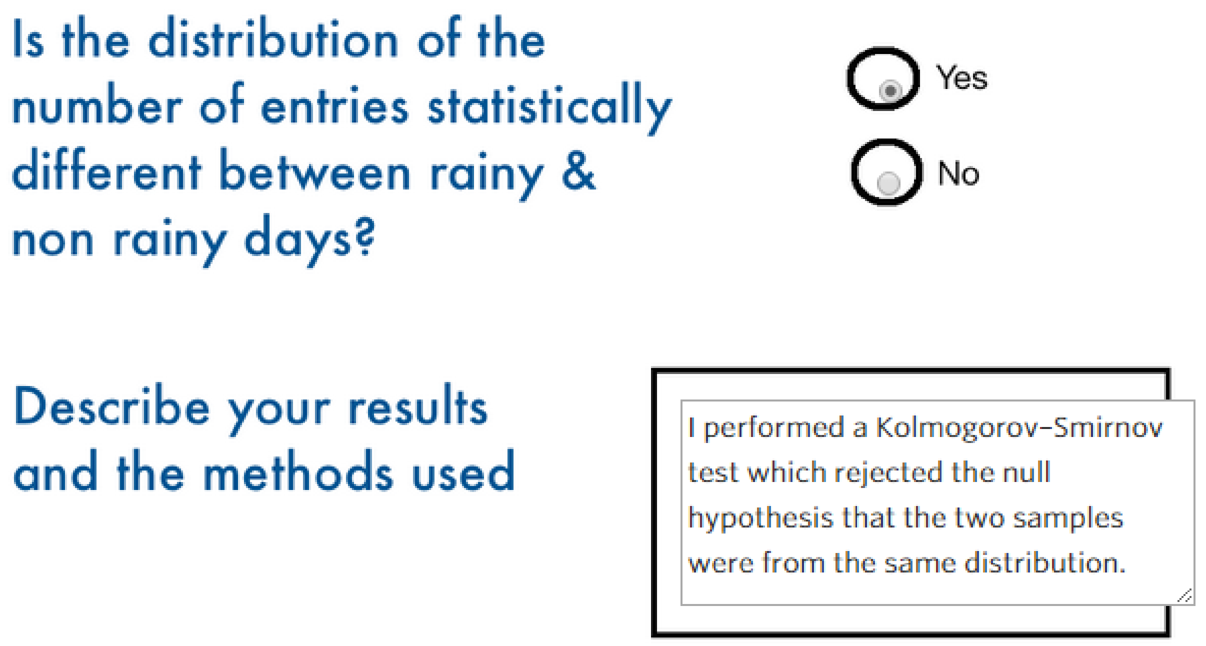
Yes [✓ No]



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

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)

        # 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
```

```
Problem Set 4.1

from pandas import *
from ggplot import *

def plot_weather_data(turnstile_weather):
    df = turnstile_weather
    plot = ggplot(aes('Hour', 'ENTRIESn_hourly', color = 'UNIT'), data = df)
    plot = plot + geom_line()
    return plot



Problem Set 4.2

from pandas import *
from ggplot import *

def plot_weather_data(turnstile_weather):
    pandas.options.mode.chained_assignment = None
    df = turnstile_weather
    UNIT2 = []
    for i in range(len(df['UNIT'])):
        UNIT2.append(int(df['UNIT'][i][1:5]))
    df['UNIT2'] = UNIT2
    plot = ggplot(aes('UNIT2', 'ENTRIESn_hourly', color = 'meantempi'), data = df)
    plot = plot + geom_line() + facet_wrap('Hour') + ggtitle('Entries by Unit per Hour') + \
    xlab('Unit') + ylab('Entries')


    return plot
```

Problem Set 5 : MapReduce on Subway Data Miadad Rashid 2015.2.24

```
Problem Set 5.1

# mapper.py

import sys
import string
import logging

from util import mapper_logfile
logging.basicConfig(filename=mapper_logfile, format='%(message)s',
                    level=logging.INFO, filemode='w')

def mapper():

    for line in sys.stdin:
        # your code here
        data = line.strip().split(',')
        if len(data) != 22 or data[1] == 'UNIT':
            continue

        out = "{}\t{}".format(data[1], data[6])
        print out


mapper()

import sys
import logging

from util import reducer_logfile
logging.basicConfig(filename=reducer_logfile, format='%(message)s',
                    level=logging.INFO, filemode='w')

def reducer():

    entries = 0
    last_unit = None

    for line in sys.stdin:
        # your code here
        data = line.strip().split('\t')
        if len(data) != 2:
            continue
        this_unit, count = data

        if last_unit and last_unit != this_unit:
            print '{0}\t{1}'.format(last_unit, entries)
            entries = 0

        last_unit = this_unit
        entries += float(count)

    if last_unit != None:
        print "{}\t{}".format(last_unit, entries)


reducer()




```

Problem Set 5.2

```python

import sys
import string
import logging

from util import mapper_logfile
logging.basicConfig(filename=mapper_logfile, format='%(message)s',
                    level=logging.INFO, filemode='w')

def mapper():
    def format_key(fog, rain):
        return '{}fog-{}rain'.format(
            '' if fog else 'no',
            '' if rain else 'no'
        )

    for line in sys.stdin:
        # your code here
        data = line.strip().split(',')
        if len(data) != 22 or data[1] == 'UNIT':
            continue
        out = "{}\t{}".format(format_key(float(data[14]), float(data[15])), data[6])
        print out



mapper()
import sys
import logging

from util import reducer_logfile
logging.basicConfig(filename=reducer_logfile, format='%(message)s',
                    level=logging.INFO, filemode='w')

def reducer():

    riders = 0      # The number of total riders for this key
    num_hours = 0   # The number of hours with this key
    old_key = None

    for line in sys.stdin:
        # your code here
        data = line.strip().split('\t')
        if len(data) != 2:
            continue
        this_unit, count = data

        if old_key and old_key != this_unit:
            print '{0}\t{1}'.format(old_key, (riders/float(num_hours)))
            riders = 0
            num_hours = 0

        old_key = this_unit
        riders += float(count)
        num_hours += 1

    if old_key != None:
        print '{0}\t{1}'.format(old_key, (riders/float(num_hours)))
reducer()
```

Problem Set 5.3

```python

import sys
import string
import logging

from util import mapper_logfile
logging.basicConfig(filename=mapper_logfile, format='%(message)s',
                    level=logging.INFO, filemode='w')

def mapper():
    for line in sys.stdin:
        data = line.strip().split(',')
        if data[1] == 'UNIT' or len(data) != 22:
            continue

        print '{0}\t{1}\t{2}\t{3}'.format(data[1],data[6], data[2], data[3])


mapper()

import sys
import logging

from util import reducer_logfile
logging.basicConfig(filename=reducer_logfile, format='%(message)s',
                    level=logging.INFO, filemode='w')
def reducer():
    max_entries = 0
    old_key = None
    datetime = ''

    for line in sys.stdin:
        # your code here
        data = line.strip().split('\t')

        if len(data) != 4:
            continue

        this_key, count, date, time  = data
        count = float(count)

        if old_key and old_key != this_key:
            print "{0}\t{1}\t{2}".format(old_key, datetime, max_entries)
            max_entries = 0
            datetime = ''

        old_key = this_key
        if count >= max_entries:
            max_entries = count
            datetime = str(date) + ' ' + str(time)
            #datetime = datetime.strptime(date, '%Y-%m-
%d') + ' ' + datetime.strptime(time, '%H:%M:%S')

    if old_key != None:
        print "{0}\t{1}\t{2}".format(old_key, datetime, max_entries)

reducer()
```