Intro to Data Science: Code Problem Sets 2-5

Problem Set 2. Wrangling Data

Problem 2.1. Number of Rainy Days

```
import pandas
import pandasql
def num_rainy_days(filename):
    '''
    This function should run a SQL query on a dataframe of
    weather data.  The SQL query should return one column and
    one row - a count of the number of days in the dataframe where
    the rain column is equal to 1 (i.e., the number of days it
    rained).  The dataframe will be titled 'weather_data'. You'll
    need to provide the SQL query.  You might find SQL's count function
    useful for this exercise.  You can read more about it here:

    https://dev.mysql.com/doc/refman/5.1/en/counting-rows.html
    weather_data = pandas.read_csv(filename)
    '''
    q = """
    SELECT COUNT(*) FROM weather_data WHERE rain=1;
    """
    #Execute your SQL command against the pandas frame
    rainy_days = pandasql.sqldf(q.lower(), locals())
    return rainy_days
```

Problem 2.2. Temp on Foggy and Nonfoggy Days

```
import pandas
import pandasql
def max_temp_aggregate_by_fog(filename):
    '''
    This function should run a SQL query on a dataframe of
    weather data.  The SQL query should return two columns and
    two rows - whether it was foggy or not (0 or 1) and the max
    maxtempi for that fog value (i.e., the maximum max temperature
    for both foggy and non-foggy days).  The dataframe will be
    titled 'weather_data'. You'll need to provide the SQL query.

    You can see the weather data that we are passing in below:
    https://www.dropbox.com/s/7sf0yqc9ykpq3w8/weather_underground.csv
    '''
    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 2.3. Mean Temp on Weekends

```
import pandas
import pandasql
def avg_min_temperature(filename):
    '''
    This function should run a SQL query on a dataframe of
    weather data.  The SQL query should return one column and
    one row - the average meantempi on days that are a Saturday
    or Sunday (i.e., the the average mean temperature on weekends).
    The dataframe will be titled 'weather_data' and you can access
    the date in the dataframe via the 'date' column.

    You can see the weather data that we are passing in below:
    https://www.dropbox.com/s/7sf0yqc9ykpq3w8/weather_underground.csv
```

```
    '''
    weather_data = pandas.read_csv(filename)
    q = """
    SELECT avg(cast (meantempi as integer)) FROM weather_data WHERE
        cast (strftime('%w', date) as integer) in (0,6)
    """
    #Execute your SQL command against the pandas frame
    mean_temp_weekends = pandasql.sqldf(q.lower(), locals())
    return mean_temp_weekends
```

Problem 2.4. Mean Temp on Rainy Days

```
import pandas
import pandasql
def avg_min_temperature(filename):
    '''
    This function should run a SQL query on a dataframe of
    weather data. More specifically you want to find the average
    minimum temperature on rainy days where the minimum temperature
    is greater than 55 degrees.

    You can see the weather data that we are passing in below:
    https://www.dropbox.com/s/7sf0yqc9ykpq3w8/weather_underground.csv
    '''
    weather_data = pandas.read_csv(filename)
    q = """
    SELECT avg(cast (mintempi as integer)) FROM weather_data WHERE
        mintempi > 55 and rain=1;
    """
    #Execute your SQL command against the pandas frame
    mean_temp_weekends = pandasql.sqldf(q.lower(), locals())
    return mean_temp_weekends
```

Problem 2.5. Fixing Turnstile Data

```
import csv
def fix_turnstile_data(filenames):
    '''
    Filenames is a list of MTA Subway turnstile text files. A link to an example
    MTA Subway turnstile text file can be seen at the URL below:
    http://web.mta.info/developers/data/nyct/turnstile/turnstile_110507.txt

    As you can see, there are numerous data points included in each row of the
    a MTA Subway turnstile text file.

    You want to write a function that will update each row in the text
    file so there is only one entry per row. A few examples below:
    A002,R051,02-00-00,05-28-11,00:00:00,REGULAR,003178521,001100739
    A002,R051,02-00-00,05-28-11,04:00:00,REGULAR,003178541,001100746
    A002,R051,02-00-00,05-28-11,08:00:00,REGULAR,003178559,001100775

    Write the updates to a different text file in the format of "updated_" +
    filename.
    For example:
        1) if you read in a text file called "turnstile_110521.txt"
        2) you should write the updated data to "updated_turnstile_110521.txt"

    The order of the fields should be preserved.

    You can see a sample of the turnstile text file that's passed into this
    function and the the corresponding updated file in the links below:

    Sample input file:
    https://www.dropbox.com/s/mpin5zv4hgrx244/turnstile_110528.txt
    Sample updated file:
    https://www.dropbox.com/s/074xbgio4c39b7h/solution_turnstile_110528.txt
    '''
    for name in filenames:
        with open(name, 'r') as fin:
            reader = csv.reader(fin)
            with open('updated_' + name, 'w') as fout:
                writer = csv.writer(fout)
```

```
                for inrow in reader:
                    for i in range(3, len(inrow), 5):
                        outrow = []
                        outrow.append(inrow[0])
                        outrow.append(inrow[1])
                        outrow.append(inrow[2])
                        for j in range(i, i + 5):
                            outrow.append(inrow[j].strip())
                        writer.writerow(outrow)
```

Problem 2.6. Combining Turnstile Data

```
def create_master_turnstile_file(filenames, output_file):
    '''
    Write a function that takes the files in the list filenames, which all have the
    columns 'C/A, UNIT, SCP, DATEn, TIMEn, DESCn, ENTRIESn, EXITSn', and
    consolidates them into one file located at output_file.  There should be ONE
    row with the column headers, located at the top of the file.

    For example, if file_1 has:
    'C/A, UNIT, SCP, DATEn, TIMEn, DESCn, ENTRIESn, EXITSn'
    line 1 ...
    line 2 ...

    and another file, file_2 has:
    'C/A, UNIT, SCP, DATEn, TIMEn, DESCn, ENTRIESn, EXITSn'
    line 3 ...
    line 4 ...
    line 5 ...

    We need to combine file_1 and file_2 into a master_file like below:
     'C/A, UNIT, SCP, DATEn, TIMEn, DESCn, ENTRIESn, EXITSn'
    line 1 ...
    line 2 ...
    line 3 ...
    line 4 ...
    line 5 ...
    '''
    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:
           with open(filename, 'r') as inpfile:
               row = inpfile.readline(-1)
               row = inpfile.readline(-1)
               while row <> '':
                   master_file.write(row)
                   row = inpfile.readline(-1)
               inpfile.close()
       master_file.close()
```

Problem 2.7. Filtering Irregular Data

```
import pandas
def filter_by_regular(filename):
    '''
    This function should read the csv file located at filename into a pandas
    dataframe, and filter the dataframe to only rows where the 'DESCn' column has
    the value 'REGULAR'.

    For example, if the pandas dataframe is as follows:
    ,C/A,UNIT,SCP,DATEn,TIMEn,DESCn,ENTRIESn,EXITSn
    0,A002,R051,02-00-00,05-01-11,00:00:00,REGULAR,3144312,1088151
    1,A002,R051,02-00-00,05-01-11,04:00:00,DOOR,3144335,1088159
    2,A002,R051,02-00-00,05-01-11,08:00:00,REGULAR,3144353,1088177
    3,A002,R051,02-00-00,05-01-11,12:00:00,DOOR,3144424,1088231

    The dataframe will look like below after filtering to only rows where DESCn
    column has the value 'REGULAR':
    0,A002,R051,02-00-00,05-01-11,00:00:00,REGULAR,3144312,1088151
    2,A002,R051,02-00-00,05-01-11,08:00:00,REGULAR,3144353,1088177
    '''
    turnstile_data = pandas.read_csv(filename)
```

```
    turnstile_data = turnstile_data[turnstile_data['DESCn'] == 'REGULAR']
    return turnstile_data
```

Problem 2.8. Get Hourly Entries

```
import pandas
def get_hourly_entries(df):
    '''
    The data in the MTA Subway Turnstile data reports on the cumulative
    number of entries and exits per row.  Assume that you have a dataframe
    called df that contains only the rows for a particular turnstile machine
    (i.e., unique SCP, C/A, and UNIT).  This function should change
    these cumulative entry numbers to a count of entries since the last reading
    (i.e., entries since the last row in the dataframe).

    More specifically, you want to do two things:
       1) Create a new column called ENTRIESn_hourly
       2) Assign to the column the difference between ENTRIESn of the current row
          and the previous row. If there is any NaN, fill/replace it with 1.

    You may find the pandas functions shift() and fillna() to be helpful in this exercise.

    Examples of what your dataframe should look like at the end of this exercise:

           C/A  UNIT       SCP     DATEn     TIMEn    DESCn  ENTRIESn    EXITSn  ENTRIESn_hourly
    0     A002  R051  02-00-00  05-01-11  00:00:00  REGULAR   3144312   1088151                1
    1     A002  R051  02-00-00  05-01-11  04:00:00  REGULAR   3144335   1088159               23
    2     A002  R051  02-00-00  05-01-11  08:00:00  REGULAR   3144353   1088177               18
    3     A002  R051  02-00-00  05-01-11  12:00:00  REGULAR   3144424   1088231               71
    4     A002  R051  02-00-00  05-01-11  16:00:00  REGULAR   3144594   1088275              170
    5     A002  R051  02-00-00  05-01-11  20:00:00  REGULAR   3144808   1088317              214
    6     A002  R051  02-00-00  05-02-11  00:00:00  REGULAR   3144895   1088328               87
    7     A002  R051  02-00-00  05-02-11  04:00:00  REGULAR   3144905   1088331               10
    8     A002  R051  02-00-00  05-02-11  08:00:00  REGULAR   3144941   1088420               36
    9     A002  R051  02-00-00  05-02-11  12:00:00  REGULAR   3145094   1088753              153
    10    A002  R051  02-00-00  05-02-11  16:00:00  REGULAR   3145337   1088823              243
    ...
    ...

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

Problem 2.9. Get Hourly Exits

```
import pandas
def get_hourly_exits(df):
    '''
    The data in the MTA Subway Turnstile data reports on the cumulative
    number of entries and exits per row.  Assume that you have a dataframe
    called df that contains only the rows for a particular turnstile machine
    (i.e., unique SCP, C/A, and UNIT).  This function should change
    these cumulative exit numbers to a count of exits since the last reading
    (i.e., exits since the last row in the dataframe).

    More specifically, you want to do two things:
       1) Create a new column called EXITSn_hourly
       2) Assign to the column the difference between EXITSn of the current row
          and the previous row. If there is any NaN, fill/replace it with 0.

    You may find the pandas functions shift() and fillna() to be helpful in this exercise.

    Example dataframe below:

          Unnamed: 0   C/A  UNIT       SCP     DATEn     TIMEn    DESCn  ENTRIESn    EXITSn
ENTRIESn_hourly  EXITSn_hourly
    0              0  A002  R051  02-00-00  05-01-11  00:00:00  REGULAR   3144312   1088151
0             0
    1              1  A002  R051  02-00-00  05-01-11  04:00:00  REGULAR   3144335   1088159
23             8
```

```
    2              2  A002  R051  02-00-00  05-01-11  08:00:00  REGULAR   3144353   1088177
18             18
    3              3  A002  R051  02-00-00  05-01-11  12:00:00  REGULAR   3144424   1088231
71             54
    4              4  A002  R051  02-00-00  05-01-11  16:00:00  REGULAR   3144594   1088275
170             44
    5              5  A002  R051  02-00-00  05-01-11  20:00:00  REGULAR   3144808   1088317
214             42
    6              6  A002  R051  02-00-00  05-02-11  00:00:00  REGULAR   3144895   1088328
87             11
    7              7  A002  R051  02-00-00  05-02-11  04:00:00  REGULAR   3144905   1088331
10              3
    8              8  A002  R051  02-00-00  05-02-11  08:00:00  REGULAR   3144941   1088420
36             89
    9              9  A002  R051  02-00-00  05-02-11  12:00:00  REGULAR   3145094   1088753
153            333
    '''

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

Problem 2.10. Time to Hour

```
import pandas
def time_to_hour(time):
    '''
    Given an input variable time that represents time in the format of:
    "00:00:00" (hour:minutes:seconds)

    Write a function to extract the hour part from the input variable time
    and return it as an integer. For example:
        1) if hour is 00, your code should return 0
        2) if hour is 01, your code should return 1
        3) if hour is 21, your code should return 21

    Please return hour as an integer.
    '''

    hourstr = time.split(':')[0]
    if hourstr[0] == '0':
        hourstr = hourstr[1:2]
    hour = int(hourstr)
    return hour
```

Problem 2.11. Reformat Subway Dates

```
import datetime
import time
def reformat_subway_dates(date):
    '''
    The dates in our subway data are formatted in the format month-day-year.
    The dates in our weather underground data are formatted year-month-day.

    In order to join these two data sets together, we'll want the dates formatted
    the same way.  Write a function that takes as its input a date in the MTA Subway
    data format, and returns a date in the weather underground format.
    '''
    mtadate = time.strptime(date, '%m-%d-%y')
    date_formatted = time.strftime('%Y-%m-%d', mtadate)
    return date_formatted
```

Problem Set 3. Analyzing Subway Data

Problem 3.1. Exploratory Data Analysis

```
import numpy as np
import pandas
import matplotlib.pyplot as plt
def entries_histogram(turnstile_weather):
```

```
    '''
    Before we perform any analysis, it might be useful to take a
    look at the data we're hoping to analyze. More specifically, let's
    examine the hourly entries in our NYC subway data and determine what
    distribution the data follows. This data is stored in a dataframe
    called turnstile_weather under the ['ENTRIESn_hourly'] column.

    Let's plot two histograms on the same axes to show hourly
    entries when raining vs. when not raining. Here's an example on how
    to plot histograms with pandas and matplotlib:
    turnstile_weather['column_to_graph'].hist()

    Your histograph may look similar to bar graph in the instructor notes below.

    You can read a bit about using matplotlib and pandas to plot histograms here:
    http://pandas.pydata.org/pandas-docs/stable/visualization.html#histograms

    You can see the information contained within the turnstile weather data here:
    https://www.dropbox.com/s/meyki2wl9xfa7yk/turnstile_data_master_with_weather.csv
    '''
    plt.figure()
    df = pandas.DataFrame({
      'No rain' : turnstile_weather['ENTRIESn_hourly']\
           [turnstile_weather['rain']==0][turnstile_weather['ENTRIESn_hourly']<6000],
      'Rain' : turnstile_weather['ENTRIESn_hourly']\
           [turnstile_weather['rain']==1][turnstile_weather['ENTRIESn_hourly']<6000]
       }
    )
    ax = df.plot(kind='hist',stacked=True, bins=50)
    ax.set_ylabel('Frequency')
    ax.set_xlabel('ENTRIESn_hourly')
    ax.set_title('Histogram of ENTRIESn_hourly')
    return plt
```

Problem 3.3. Mann-Whitney U-Test

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

def mann_whitney_plus_means(turnstile_weather):
    '''
    This function will consume the turnstile_weather dataframe containing
    our final turnstile weather data.

    You will want to take the means and run the Mann Whitney U-test on the
    ENTRIESn_hourly column in the turnstile_weather dataframe.

    This function should return:
        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

    You should feel free to use scipy's Mann-Whitney implementation, and you
    might also find it useful to use numpy's mean function.

    Here are the functions' documentation:
    http://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.mannwhitneyu.html
    http://docs.scipy.org/doc/numpy/reference/generated/numpy.mean.html

    You can look at the final turnstile weather data at the link below:
    https://www.dropbox.com/s/meyki2wl9xfa7yk/turnstile_data_master_with_weather.csv
    '''
    with_rain = turnstile_weather['ENTRIESn_hourly'][turnstile_weather['rain']==1]
    with_rain_mean = np.mean(with_rain)
    without_rain = turnstile_weather['ENTRIESn_hourly'][turnstile_weather['rain']==0]
    without_rain_mean = np.mean(without_rain)
    (U,p)=scipy.stats.mannwhitneyu(with_rain, without_rain)
    return with_rain_mean, without_rain_mean, U, p # leave this line for the grader
```

Problem 3.5. Linear Regression

```
import numpy as np
import pandas
from ggplot import *
def normalize_features(df):
    """
    Normalize the features in the data set.
    """
    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):
    """
    Compute the cost function given a set of features / values,
    and the values for our thetas.

    This can be the same code as the compute_cost function in the lesson #3 exercises,
    but feel free to implement your own.
    """

    m = len(values)
    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 can be the same gradient descent code as in the lesson #3 exercises,
    but feel free to implement your own.
    """

    m = len(values)
    cost_history = []

    for i in range(num_iterations):
        error = values - np.dot(features, theta)
        overstep = np.dot(np.transpose(features), error)
        delta = (alpha/len(features)) * overstep
        theta = theta + delta
        jcost = compute_cost(features, values, theta)
        cost_history.append(jcost)
    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, let's predict the ridership of
    the NYC subway using linear regression with gradient descent.

    You can download the complete turnstile weather dataframe here:
    https://www.dropbox.com/s/meyki2wl9xfa7yk/turnstile_data_master_with_weather.csv

    Your prediction should have a R^2 value of 0.20 or better.
    You need to experiment using various input features contained in the dataframe.
    We recommend that you don't use the EXITSn_hourly feature as an input to the
    linear model because we cannot use it as a predictor: we cannot use exits
    counts as a way to predict entry counts.
```

```
    If you'd like to view a plot of your cost history, uncomment the call to
    plot_cost_history below. The slowdown from plotting is significant, so if you
    are timing out, the first thing to do is to comment out the plot command again.

    If you are using your own algorithm/models, see if you can optimize your code so
    that it runs faster.
    '''
    # Select Features (try different features!)
    features = dataframe[['rain', 'Hour', 'meanwindspdi', 'mintempi', 'maxtempi', 'minpressurei']]

    # 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 = 0.2 # please feel free to change this value
    num_iterations = 20 # 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)
    #

    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 3.6. Plotting Residuals

```
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).
    Try different binwidths for your histogram.

    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=\
        [-2500, -2000, -1500, -1000, -500, 0, 500, 1000, 1500, 2000, 2500])
    plt.axis([-2500, 2500, 0, 7500])
    return plt
```

Problem 3.7. Compute R^2

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

Problem 3.8. More Linear Regression

```
import numpy as np
import pandas
import scipy
import statsmodels.api as sm
"""
In this optional exercise, you should complete the function called
predictions(turnstile_weather). This function takes in our pandas
turnstile weather dataframe, and returns a set of predicted ridership values,
based on the other information in the dataframe.

In exercise 3.5 we used Gradient Descent in order to compute the coefficients
theta used for the ridership prediction. Here you should attempt to implement
another way of computing the coeffcients theta. You may also try using a reference
implementation such as:
http://statsmodels.sourceforge.net/devel/generated/statsmodels.regression.linear_model.OLS.html

One of the advantages of the statsmodels implementation is that it gives you
easy access to the values of the coefficients theta. This can help you infer relationships
between variables in the dataset.

You may also experiment with polynomial terms as part of the input variables.

The following links might be useful:
http://en.wikipedia.org/wiki/Ordinary_least_squares
http://en.wikipedia.org/w/index.php?title=Linear_least_squares_(mathematics)
http://en.wikipedia.org/wiki/Polynomial_regression

You can look at the information contained in the turnstile_weather dataframe below:
https://www.dropbox.com/s/meyki2wl9xfa7yk/turnstile_data_master_with_weather.csv
"""
def predictions(weather_turnstile):
    # Select Features (try different features!)
    features = weather_turnstile[['rain', 'precipi', 'Hour', 'meantempi', 'meanwindspdi']]

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

```
    # Values
    values = weather_turnstile['ENTRIESn_hourly']
    # Convert features and values to numpy arrays
    features_array = np.array(features)
    values_array = np.array(values)
    m = len(values)
    features['ones'] = np.ones(m)
    model = sm.OLS(values_array, features_array )
    results = model.fit()
    theta = results.params
    prediction = np.dot(features_array, theta)
    return prediction
```

Problem Set 4. Visualizing Subway Data

Problem 4.1. Exercise Visualization 1

```
from pandas import *
from ggplot import *
import pandasql
def plot_weather_data(turnstile_weather):
    '''
    You are passed in a dataframe called turnstile_weather.
    Use turnstile_weather along with ggplot to make a data visualization
    focused on the MTA and weather data we used in assignment #3.
    You should feel free to implement something that we discussed in class
    (e.g., scatterplots, line plots, or histograms) or attempt to implement
    something more advanced if you'd like.

    Here are some suggestions for things to investigate and illustrate:
     * Ridership by time of day or day of week
     * How ridership varies based on Subway station
     * Which stations have more exits or entries at different times of day

    If you'd like to learn more about ggplot and its capabilities, take
    a look at the documentation at:
    https://pypi.python.org/pypi/ggplot/

    You can check out:
    https://www.dropbox.com/s/meyki2wl9xfa7yk/turnstile_data_master_with_weather.csv

    To see all the columns and data points included in the turnstile_weather
    dataframe.

    However, due to the limitation of our Amazon EC2 server, we are giving you a random
    subset, about 1/3 of the actual data in the turnstile_weather dataframe.
    '''
    # Yikes.  This is needed to change the name of the index column which is 'unnamed: 0'.
    tw = turnstile_weather.rename(columns = lambda x: x.replace(' ', '_').lower(), inplace=False)
    # Ridership by time of day
    q = """
    select cast(hour as float) as hour,cast(sum(entriesn_hourly) as float) as entriesn_hourly
    from tw group by hour;
    """
    ridershipbyhour = pandasql.sqldf(q, locals())
    plot = ggplot(ridershipbyhour,aes('hour','entriesn_hourly')) + geom_point() + geom_line() +\
        ggtitle('Ridership by hour of day') + xlab('Hour') + ylab('Ridership')
    #
    # Ridership by day of week
    #dowSeries = strDateSeriesToDayOfWeekSeries(turnstile_weather['DATEn'])
    #turnstile_weather['DayOfWeek'] = dowSeries
    #plot = ggplot(turnstile_weather,aes('DayOfWeek','ENTRIESn_hourly')) + geom_point()
    #
    return plot
```

Problem 4.2. Exploratory Data Analysis

```
from pandas import *
from ggplot import *
import pandasql
```

```
def plot_weather_data(turnstile_weather):
    '''
    plot_weather_data is passed a dataframe called turnstile_weather.
    Use turnstile_weather along with ggplot to make another data visualization
    focused on the MTA and weather data we used in Project 3.

    Make a type of visualization different than what you did in the previous exercise.
    Try to use the data in a different way (e.g., if you made a lineplot concerning
    ridership and time of day in exercise #1, maybe look at weather and try to make a
    histogram in this exercise). Or try to use multiple encodings in your graph if
    you didn't in the previous exercise.

    You should feel free to implement something that we discussed in class
    (e.g., scatterplots, line plots, or histograms) or attempt to implement
    something more advanced if you'd like.

    Here are some suggestions for things to investigate and illustrate:
     * Ridership by time-of-day or day-of-week
     * How ridership varies by subway station
     * Which stations have more exits or entries at different times of day

    If you'd like to learn more about ggplot and its capabilities, take
    a look at the documentation at:
    https://pypi.python.org/pypi/ggplot/

    You can check out the link
    https://www.dropbox.com/s/meyki2wl9xfa7yk/turnstile_data_master_with_weather.csv
    to see all the columns and data points included in the turnstile_weather
    dataframe.

    However, due to the limitation of our Amazon EC2 server, we are giving you a random
    subset, about 1/3 of the actual data in the turnstile_weather dataframe.
    '''
    tw = turnstile_weather.rename(columns = lambda x: x.replace(' ', '_').lower(), inplace=False)

    # Ridershape by station and tie of day
    q = """
    select unit,hour,sum(entriesn_hourly) as entriesn_hourly from tw group by unit, hour limit 27;
    """
    ridershipbystationhour = pandasql.sqldf(q, locals()).fillna(0)
    plot = ggplot(ridershipbystationhour,aes('hour','entriesn_hourly',color='unit')) + geom_point() +\
        geom_line() + ggtitle('Ridership by station') + xlab('Unit') + ylab('Ridership')
    return plot
```

Problem Set 5. MapReduce on Subway Data

Problem 5.1. Ridership per Station

ridership_per_station_mapper.py

```
import sys
import string
import logging
rom util import mapper_logfile
logging.basicConfig(filename=mapper_logfile, format='%(message)s',
                    level=logging.INFO, filemode='w')
def mapper():
    """
    The input to this mapper will be the final Subway-MTA dataset, the same as
    in the previous exercise.  You can check out the csv and its structure below:
    https://www.dropbox.com/s/meyki2wl9xfa7yk/turnstile_data_master_with_weather.csv

    For each line of input, the mapper output should PRINT (not return) the UNIT as
    the key, the number of ENTRIESn_hourly as the value, and separate the key and
    the value by a tab. For example: 'R002\t105105.0'
    """
    header = True
    entriesIndex = -1
    unitIndex = -1
```

```
    for line in sys.stdin:
        fields = line.split(',')
        if not header:
            unit = fields[unitIndex]
            entryCount = fields[entriesIndex]
            print unit + '\t' + entryCount
        else:
            i = 0
            for field in fields:
                if field == 'UNIT':
                    unitIndex = i
                elif field == 'ENTRIESn_hourly':
                    entriesIndex = i
                i += 1
        header = False
mapper()
```

ridership_per_station_reducer.py

```
import sys
import logging
from util import reducer_logfile
logging.basicConfig(filename=reducer_logfile, format='%(message)s',
                    level=logging.INFO, filemode='w')
def reducer():
    '''
    Given the output of the mapper for this exercise, the reducer should PRINT
    (not return) one line per UNIT along with the total number of ENTRIESn_hourly
    over the course of May (which is the duration of our data), separated by a tab.
    An example output row from the reducer might look like this: 'R001\t500625.0'

    You can assume that the input to the reducer is sorted such that all rows
    corresponding to a particular UNIT are grouped together.
    '''
    count = 0.0
    old_key = None
    for line in sys.stdin:
        kvs = line.split('\t')
        if len(kvs) != 2:
            continue
        new_key = kvs[0]
        val = kvs[1]
        if old_key and old_key != new_key:
            print "{0}\t{1}".format(old_key, count)
            count = 0
        old_key = new_key
        count += float(val)
    if old_key != None:
        print "{0}\t{1}".format(old_key, count)
reducer()
```

Problem 5.2. Ridership by Weather Type

ridership_by_weather_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 this exercise, compute the average value of the ENTRIESn_hourly column
    for different weather types. Weather type will be defined based on the
    combination of the columns fog and rain (which are boolean values).
    For example, one output of our reducer would be the average hourly entries
    across all hours when it was raining but not foggy.

    Each line of input will be a row from our final Subway-MTA dataset in csv format.
    You can check out the input csv file and its structure below:
```

```
    https://www.dropbox.com/s/meyki2wl9xfa7yk/turnstile_data_master_with_weather.csv

    Note that this is a comma-separated file.

    This mapper should PRINT (not return) the weather type as the key (use the
    given helper function to format the weather type correctly) and the number in
    the ENTRIESn_hourly column as the value. They should be separated by a tab.
    For example: 'fog-norain\t12345'
    '''

    # Takes in variables indicating whether it is foggy and/or rainy and
    # returns a formatted key that you should output.  The variables passed in
    # can be booleans, ints (0 for false and 1 for true) or floats (0.0 for
    # false and 1.0 for true), but the strings '0.0' and '1.0' will not work,
    # so make sure you convert these values to an appropriate type before
    # calling the function.
    def format_key(fog, rain):
        return '{}fog-{}rain'.format(
            '' if fog else 'no',
            '' if rain else 'no'
        )

    header = True
    entriesIndex = -1
    fogIndex = -1
    rainIndex = -1
    fogb = False
    rainb = False
    for line in sys.stdin:
        fields = line.split(',')
        if not header:
            fog = float(fields[fogIndex])
            rain = float(fields[rainIndex])
            entryCount = fields[entriesIndex]
            print format_key(fog, rain) + '\t' + entryCount
            # logging.info(format_key(fog, rain) + '\t' + entryCount)
        else:
            i = 0
            for field in fields:
                if field == 'fog':
                    fogIndex = i
                elif field == 'rain':
                    rainIndex = i
                elif field == 'ENTRIESn_hourly':
                    entriesIndex = i
                i += 1
        header = False
mapper()
```

ridership_by_weather_reducer.py

```
import sys
import logging
from util import reducer_logfile
logging.basicConfig(filename=reducer_logfile, format='%(message)s',
                    level=logging.INFO, filemode='w')
def reducer():
    '''
    Given the output of the mapper for this assignment, the reducer should
    print one row per weather type, along with the average value of
    ENTRIESn_hourly for that weather type, separated by a tab. You can assume
    that the input to the reducer will be sorted by weather type, such that all
    entries corresponding to a given weather type will be grouped together.

    In order to compute the average value of ENTRIESn_hourly, you'll need to
    keep track of both the total riders per weather type and the number of
    hours with that weather type. That's why we've initialized the variable
    riders and num_hours below. Feel free to use a different data structure in
    your solution, though.

    An example output row might look like this:
    'fog-norain\t1105.32467557'
```

```
    '''
    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:
        kvs = line.split('\t')
        if len(kvs) != 2:
            continue
        new_key = kvs[0]
        val = kvs[1]
        if old_key and old_key != new_key:
            print "{0}\t{1}".format(old_key, riders/num_hours)
            num_hours = 0
            riders = 0
        old_key = new_key
        num_hours += 1
        riders += float(val)
    if old_key != None:
        print "{0}\t{1}".format(old_key, riders/num_hours)
reducer()
```

Problem 5.3. Busiest Hour

busiest_hour_mapper.py

```
import sys
import logging
from util import reducer_logfile
logging.basicConfig(filename=reducer_logfile, format='%(message)s',
                    level=logging.INFO, filemode='w')
def reducer():
    '''
    Write a reducer that will compute the busiest date and time (that is, the
    date and time with the most entries) for each turnstile unit. Ties should
    be broken in favor of datetimes that are later on in the month of May. You
    may assume that the contents of the reducer will be sorted so that all entries
    corresponding to a given UNIT will be grouped together.

    The reducer should print its output with the UNIT name, the datetime (which
    is the DATEn followed by the TIMEn column, separated by a single space), and
    the number of entries at this datetime, separated by tabs.

    For example, the output of the reducer should look like this:
    R001    2011-05-11 17:00:00         31213.0
    R002       2011-05-12 21:00:00      4295.0
    R003       2011-05-05 12:00:00      995.0
    R004       2011-05-12 12:00:00      2318.0
    R005       2011-05-10 12:00:00      2705.0
    R006       2011-05-25 12:00:00      2784.0
    R007       2011-05-10 12:00:00      1763.0
    R008       2011-05-12 12:00:00      1724.0
    R009       2011-05-05 12:00:00      1230.0
    R010       2011-05-09 18:00:00      30916.0
    ...
    ...

    Since you are printing the output of your program, printing a debug
    statement will interfere with the operation of the grader. Instead,
    use the logging module, which we've configured to log to a file printed
    when you click "Test Run". For example:
    logging.info("My debugging message")
    '''

    max_entries = float(0)
    old_key = None
    datetime = ''
    for line in sys.stdin:
        kvs = line.split('\t')
        if len(kvs) != 4:
            continue
        new_key = kvs[0]
        val = float(kvs[1])
        dtime = kvs[2] + ' ' + kvs[3].strip()
```

```
        if old_key and old_key != new_key:
            xxx = format(max_entries,".1f").strip()
            ostr = old_key + '\t' + datetime + '\t' + xxx
            print ostr
            max_entries = val
            datetime = dtime
        old_key = new_key
        if val >= max_entries:
            max_entries = val
            datetime = dtime
    if old_key != None:
        #logging.info( "{}\t{}\t{}".format(old_key, datetime, format(max_entries,".1f")).strip())
        print "{}\t{}\t{}".format(old_key, datetime, format(max_entries, ".1f").strip())
reducer()
```