Data Problems/Solutions:

Lesson one

2) Find mean of bronze metals when there is at least one gold

My Solution (correct answer, but fewer significant digits)

avg_bronze_at_least_one_gold=df[(df.gold>0)][['bronze']].apply(numpy.mean)

Their solution

Bronze_at_least_one_gold=Olympic_medal_counts_df['bronze'][Olympic_medal_counts_df['gold']>0] **makes array of bronze metals where there is at least one gold

avg_bronze_at_least_one_gold=numpy.mean(bronze_at least_one_gold) **finds average of bronze metals (after gold=zero are removed)

3) Using Pandas, NymPy and … create a new series that indicates the average number of gold, silver and bronze medals earned amongst countries who earned at least one medal of any kind.

ave_medal_count=df[(df.gold>0)|(df.silver>0)|(df.bronze>0)][['gold','silver','bronze']].apply(numpy.mean) ** actual correct answer

ave_medal_count = df[['gold','silver','bronze']].apply(numpy.mean) ** another correct answer due to no country not achieving one medal of some sort.

4) 4 gold, 2 silver, 1 bronze

LESSON 2

SIMPLE QUERY

```
import pandas

import pandasql


def select_first_50(filename):

    # Read in our aadhaar_data csv to a pandas dataframe.  Afterwards, we rename the columns

    # by replacing spaces with underscores and setting all characters to lowercase, so the
```

```
    # column names more closely resemble columns names one might find in a table.
    aadhaar_data = pandas.read_csv(filename)
    aadhaar_data.rename(columns = lambda x: x.replace(' ', '_').lower(), inplace=True)

    # Select out the first 50 values for "registrar" and "enrolment_agency"
    # in the aadhaar_data table using SQL syntax.
    #
    # Note that "enrolment_agency" is spelled with one l. Also, the order
    # of the select does matter. Make sure you select registrar then enrolment agency
    # in your query.
    q = """
    SELECT
    registrar, enrolment_agency
    FROM
    aadhaar_data
    LIMIT 50;
    """

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

Complex query

```
import pandas
import pandasql
```

```
def aggregate_query(filename):
    # Read in our aadhaar_data csv to a pandas dataframe.  Afterwards, we rename the columns
    # by replacing spaces with underscores and setting all characters to lowercase, so the
    # column names more closely resemble columns names one might find in a table.

    aadhaar_data = pandas.read_csv(filename)
    aadhaar_data.rename(columns = lambda x: x.replace(' ', '_').lower(), inplace=True)

    # Write a query that will select from the aadhaar_data table how many men and how
    # many women over the age of 50 have had aadhaar generated for them in each district
    #
    # Note that in this quiz, the SQL query keywords are case sensitive.
    # For example, if you want to do a sum make sure you type 'sum' rather than 'SUM'.
    #

    # The possible columns to select from aadhaar data are:
    #     1) registrar
    #     2) enrolment_agency
    #     3) state
    #     4) district
    #     5) sub_district
    #     6) pin_code
    #     7) gender
    #     8) age
    #     9) aadhaar_generated
```

```
    #     10) enrolment_rejected
    #     11) residents_providing_email,
    #     12) residents_providing_mobile_number
    #
    # You can download a copy of the aadhaar data that we are passing
    # into this exercise below:
    # https://www.dropbox.com/s/vn8t4uulbsfmalo/aadhaar_data.csv


    q = """
    SELECT
    gender, district, sum(aadhaar_generated)
    FROM
    aadhaar_data
    WHERE
    age>50
    GROUP BY
    gender, district;
    """


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

API EXERCIZE

```
import json
import requests
```

```
def api_get_request(url):
    # In this exercise, you want to call the last.fm API to get a list of the
    # top artists in Spain.
    #
    # Once you've done this, return the name of the number 1 top artist in Spain.

    data = requests.get(url).text
    data=json.loads(data)
    print type(data)
    Q="""
    SELECT
    topartistcountry='Spain' & artist_rank='1'
    FROM data
    name=name;
    """
    return (name) # return the top artist in Spain
```

OFFICIAL ANSWER

```
import json
import requests
if_name_==’_main_’:
#Provide a URL we should make an API call to.
url =
‘http://ws.audioscrobbler.com/2.0/?method=artist.gettopalbums&artist=cher&api_key=ab10a0cf075a484e1baa3308de63ea7e&format=json’

url=’http://ws.audioscrobbler.com/2.0/?method=gettopartists&country=spain&api_key=4beab33cc6d65b05800d51f5e83bde1b&format=json’
```

```
#make our API call using the requests library and load results into a dict.
data = requests.get(url).text
   data=json.loads(data)
#Print our the name of the #1 top artist.
Print data['topartists']['artist'][0]['name']
```

NEXT

```
from pandas import *
import numpy

def imputation(filename):
   # Pandas dataframes have a method called 'fillna(value)', such that you can
   # pass in a single value to replace any NAs in a dataframe or series. You
   # can call it like this:
   #     dataframe['column'] = dataframe['column'].fillna(value)
   #
   # Using the numpy.mean function, which calculates the mean of a numpy
   # array, impute any missing values in our Lahman baseball
   # data sets 'weight' column by setting them equal to the average weight.
   #
   # You can access the 'weight' colum in the baseball data frame by
   # calling baseball['weight']
```

```
    baseball = pandas.read_csv(filename)



    #YOUR CODE GOES HERE

    weight_ave=numpy.mean(baseball['weight'])

    baseball['weight']=baseball['weight'].fillna(weight_ave)


    return baseball
```

Their answer a one liner:

Baseball[‘weight’]=baseball[‘weight’].fillna(numpy.mean(baseball[‘weight’]))

Assignment #2

1) Acquire weather data via weather underground API
   A month in 2011 and write to file
2) Get a sense of the data using SQL queries
3) Clean and process the data

PROBLEM SET NUMBER 2

Problem Set 2: Wrangling Subway Data > 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

You might also find that interpreting numbers as integers or floats may not
work initially. In order to get around this issue, it may be useful to cast
these numbers as integers. This can be done by writing cast(column as integer).
So for example, if we wanted to cast the maxtempi column as an integer, we would actually
write something like where cast(maxtempi as integer) = 76, as opposed to simply
where maxtempi = 76.

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
count(*)
FROM
weather_data
WHERE
```

```
    rain=1
    GROUP BY rain

    """

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

Problem Set 2: Wrangling Subway Data > 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 might also find that interpreting numbers as integers or floats may not
    work initially.  In order to get around this issue, it may be useful to cast
```

these numbers as integers. This can be done by writing cast(column as integer).

So for example, if we wanted to cast the maxtempi column as an integer, we would actually write something like where cast(maxtempi as integer) = 76, as opposed to simply where maxtempi = 76.

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

output

```
Good job! Your code worked perfectly.
Output by your program below.
```

```
   fog  max(cast (maxtempi as integer))
0    0                               86
1    1                               81
```

Problem Set 2: Wrangling Subway Data > 2 - > 3 - Mean Temp on Weekends

```
import pandas

import pandasql


def avg_weekend_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'll need to provide  the SQL query.

    You might also find that interpreting numbers as integers or floats may not
    work initially.  In order to get around this issue, it may be useful to cast
    these numbers as integers.  This can be done by writing cast(column as integer).
    So for example, if we wanted to cast the maxtempi column as an integer, we would actually
    write something like where cast(maxtempi as integer) = 76, as opposed to simply
    where maxtempi = 76.

    Also, you can convert dates to days of the week via the 'strftime' keyword in SQL.
```

For example, cast (strftime('%w', date) as integer) will return 0 if the date

is a Sunday or 6 if the date is a Saturday.

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)=6

or cast(strftime('%w', date)as integer)=0;

"""

#Execute your SQL command against the pandas frame

mean_temp_weekends = pandasql.sqldf(q.lower(), locals())

return mean_temp_weekends

output

```
Good job! Your code worked perfectly.
Output by your program below.

   avg(cast (meantempi as integer))
0                         65.111111
```

Problem Set 2: Wrangling Subway Data > 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 might also find that interpreting numbers as integers or floats may not
    work initially.  In order to get around this issue, it may be useful to cast
    these numbers as integers.  This can be done by writing cast(column as integer).
    So for example, if we wanted to cast the maxtempi column as an integer, we would actually
    write something like where cast(maxtempi as integer) = 76, as opposed to simply
    where maxtempi = 76.

    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
    rain=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
```

output

```
Good job! Your code worked perfectly.
Output by your program below.

   avg(cast (mintempi as integer))
0                            61.25
```

Problem set 2: Wrangling Subway Data > 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"

```
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. Remember to read through the

Instructor Notes below for more details on the task.

In addition, here is a CSV reader/writer introductory tutorial:

http://goo.gl/HBbvyy

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:
    # your code here
    # create file input object f_in to work with in_data.csv file
```

```
    f_in = open(name,'r')
    # create file output object f_out to write to the new 'out_data.csv'
    f_out = open('updated_' + name,'w')

    # create csv readers and writers based on our file objects
    reader_in = csv.reader(f_in, delimiter=',')
    writer_out = csv.writer(f_out, delimiter=',')

    for line in reader_in:
       # initial reader_in row field counter to 3 when repeating sets of 5 data starts & define first three fields as header
       k=3
       header=line[0:3]
       # determine length of reader_in row to limit loop (
       length = len(line)
       while k<length:
          # write out rows in sets of 8: first three fields + sets of next five
          link = header+line[k:k+5]
          writer_out.writerow(link)
          #increment k by 5 to grab the next set of five fields
          k=k+5

    f_in.close()
    f_out.close()
```

```
Good job. Your code worked perfectly.
Your code produced the following output:

updated_turnstile_110528.txt
```

```
A002,R051,02-00-00,05-21-11,00:00:00,REGULAR,003169391,001097585

A002,R051,02-00-00,05-21-11,04:00:00,REGULAR,003169415,001097588

A002,R051,02-00-00,05-21-11,08:00:00,REGULAR,003169431,001097607

A002,R051,02-00-00,05-21-11,12:00:00,REGULAR,003169506,001097686
```

Problem Set 2: Wrangling Subway Data > 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:
        fln=open(filename,'r')
        for line in fln:
            master_file.write(line)
        master_file.close
```

OUTPUT:

```
Good job. Your code worked perfectly.
Your code produced the following output:
C/A,UNIT,SCP,DATEn,TIMEn,DESCn,ENTRIESn,EXITSn

A002,R051,02-00-00,05-21-11,00:00:00,REGULAR,003169391,001097585
```

ETC….

Problem Set 2: Wrangling Subway Data > > 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 = # your code here
    # more of your code here
    return turnstile_data
```

NEED 2-8

Problem Set 2: Wrangling Subway Data > 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()


   return df.fillna(0)
```

Problem Set 2: Wrangling Subway Data > 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.

'''

hour=time.split(':')

return (int(hour[0]))

Problem Set 2: Wrangling Subway Data > 11 – Reformat Subway Dates

import datetime

import time as t #added…?

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.

Hint:

There is a useful function in the datetime library called strptime.

More info can be seen here:

http://docs.python.org/2/library/datetime.html#datetime.datetime.strptime

'''

temp = t.strptime(date, "%m-%d-%y")#[0:3]

temp1 = temp[0]

temp2 = temp[1]

temp3 = temp[2]

dt = datetime.datetime(temp[0], temp[1], temp[2])# t.strptime(, "%Y-%m-%d")

date_formatted = dt.strftime("%Y-%m-%d")

return date_formatted

```
Good job! Your code worked perfectly. Your output below:


     Unnamed: 0  UNIT       DATEn     TIMEn    DESCn  ENTRIESn_hourly
EXITSn_hourly  Hour
```

```
0              0  R022  2011-05-01  00:00:00  REGULAR                0
0     0

1              1  R022  2011-05-01  04:00:00  REGULAR              562
173     4

2
```

Problem Set 3: Anayzing Subway Data > > 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()

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

turnstile_weather['ENTRIESn_hourly'][turnstile_weather["rain"]==1].hist(color="red") # your code here to plot a historgram for hourly entries when it is raining

return plt

OUTPUT:

```
The image produced by your code is shown below:
Does the data seem normally distributed?
Do you think we would be able to use Welch's t-test on this data?
```

Problem Set 3: Anayzing Subway Data > > 2 – Welch's t-Test

Does the data appear normal – NO

Can we run Welch's t-test? Why or why not? - No, the histogram does not appear normal, so we should try a non-parametric test. Correct

Problem Set 3: Anayzing Subway Data > > 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[turnstile_weather['rain']==1]
with_rain_mean=np.mean(with_rain['ENTRIESn_hourly'])

without_rain=turnstile_weather[turnstile_weather['rain']==0]
without_rain_mean=np.mean(without_rain['ENTRIESn_hourly'])

U,p = scipy.stats.mannwhitneyu(without_rain['ENTRIESn_hourly'],
with_rain['ENTRIESn_hourly'])

print with_rain_mean
print without_rain_mean
print U
print p

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

OUTPUT

```
1105.44637675
1090.27878015
1924409167.0
0.0249999127935
Good job! Your calculations are correct.
```

```
Here's your output:
(1105.4463767458733, 1090.278780151855, 1924409167.0, 0.024999912793489721)
Here's the correct output:
(1105.4463767458733, 1090.278780151855, 1924409167.0, 0.024999912793489721)
```

Problem Set 3: Anayzing Subway Data > > 4 – Ridership on Rainy and non-Rainy Days

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

Describe your results and the methods used.

The Mann Whitney Test returned a p value of .02. This means that it is less than 2% likely that the null hypothesis is true. The probability that the result that the means are different due to chance is less than 2%.

Problem Set 3: Anayzing Subway Data > > 5 – Linear Regression

```
import numpy as np

import pandas

from ggplot import *


"""

In this question, you need to:

1) implement the compute_cost() and gradient_descent() procedures

2) Select features (in the predictions procedure) and make predictions.


"""
```

```
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):
        p_values = np.dot(features, theta)
        theta=theta-alpha/m*(np.dot((p_values - values), features))
        cost=compute_cost(features,values,theta)
        cost_history.append(cost)
        #cost=cost_history.append(compute_cost(features, values, theta))
    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.

    Note: Due to the memory and CPU limitation of our Amazon EC2 instance, we will
    give you a random subet (~15%) of the data contained in
    turnstile_data_master_with_weather.csv. You are encouraged to experiment with
    this computer on your own computer, locally.


    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 receive a "server has encountered an error" message, that means you are hitting the 30-second limit that's placed on running your program. Try using a smaller number for num_iterations if that's the case.

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', 'precipi', 'Hour', 'meantempi', 'mintempi']]

# 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 = 0.1 # please feel free to change this value
num_iterations = 75 # 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 )
```

OUTPUT:

```
Your r^2 value is 0.46430082406
```

Problem Set 3: Anayzing Subway Data > > 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()
    return plt
```

Problem Set 3: Anayzing Subway Data > > 7 – Compute R^2

```
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

'''

data_average=np.mean(data)

numerator = np.sum(np.square(data-predictions))

denominator = np.sum(np.square(data-data_average))

r_squared= 1.0- (numerator/denominator)

return r_squared

OUTPUT

```
You calculated R^2 value correctly!
Your calculated R^2 value is: 0.318137233709
```

Problem Set 3: Anayzing Subway Data > > 8 – More Linear Regression (Optional)

-*- coding: utf-8 -*-

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

This is your playground. Go wild!

How does your choice of linear regression compare to linear regression

with gradient descent computed in Exercise 3.5?

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

Note: due to the memory and CPU limitation of our amazon EC2 instance, we will

give you a random subset (~10%) of the data contained in
turnstile_data_master_with_weather.csv

If you receive a "server has encountered an error" message, that means you are hitting

the 30 second limit that's placed on running your program. See if you can optimize your code so it

runs faster.

"""

```
def predictions(weather_turnstile):
    #
    # Your implementation goes here. Feel free to write additional
    # helper functions
    #

    y = weather_turnstile['ENTRIESn_hourly']
    x = weather_turnstile[['EXITSn_hourly','Hour',
'maxpressurei','mindewpti','minpressurei','meandewpti','meanpressurei','fog','rain','meanwinds
pdi','mintempi','meantempi', 'maxtempi', 'precipi']]
```

```
    prediction = sm.OLS(y, x).fit()
    #print prediction.summary()

    return prediction.predict()

    return prediction
```

turnstile_weather

Problem Set 4: Visualizing Subway Data > > Exercise – Visualization 1

```
from pandas import *
from ggplot import *

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
      (You can use UNIT as a proxy for subway station.)

    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.
    '''

    #plot = ggplot(turnstile_weather, aes('ENTRIESn_hourly', 'rain'))+geom_point()

  ##
    plot = ggplot(turnstile_weather, aes('ENTRIESn_hourly', fill='rain')) + geom_bar(binwidth=100) +
xlim(low=0, high=5000) + \
    xlab("Hourly Entries - Bins of Size 100") + \
    ylab("Hourly Entries - Count in each bin") + \
```

```
    ggtitle("Hourly Entries Histogram - Rain vs. No Rain (Stacked)")
    return plot
```

Problem Set 4: Visualizing Subway Data > > 2 – Make Another Visualization

```
from pandas import *
from ggplot import *

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

(You can use UNIT as a proxy for subway station.)

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.

'''

```
plot =  ggplot(turnstile_weather, aes('Hour', fill='rain')) + geom_bar(binwidth=1) + \
    xlab("Hour of the day") + ggtitle("Distribution of ridership throughout the day - Rain (Blue) / No Rain (Red)") + \
    ylab("Entries")

return plot
```

Problem Set 5: MapReduce on Subway Data > > 1 – to be continued

MAPPER:

```
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():
    """
    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'

    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")
    Note that, unlike print, logging.info will take only a single argument.
    So logging.info("my message") will work, but logging.info("my","message") will not.
```

```
The logging module can be used to give you more control over your debugging
or other messages than you can get by printing them. In this exercise, print
statements from your mapper will go to your reducer, and print statements
from your reducer will be considered your final output. By contrast, messages
logged via the loggers we configured will be saved to two files, one
for the mapper and one for the reducer. If you click "Test Run", then we
will show the contents of those files once your program has finished running.
The logging module also has other capabilities; see
https://docs.python.org/2/library/logging.html for more information.
"""
Header_row=0
key=0
entries=0
for line in sys.stdin: #cycle through lines of code
   next=line.strip().split(",") #separate data components within each line of data


   if Header_row == 0: #skip header line
      Header_row=Header_row+1
      continue


   key=next[1]
   entries=next[6]


   if len(next) !=22:  #remove lines without 22 items
```

```
        continue


    print '{0}\t{1}'.format(key,entries) #tokenize data into UNIT (key) and ENTRIESn_hourly


mapper()
```

REDUCER:

```
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.

    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")
    Note that, unlike print, logging.info will take only a single argument.
    So logging.info("my message") will work, but logging.info("my","message") will not.
    '''
Tot_ENTR_hr = 0.0
old_key = None
this_key= None
count = 0.0

for line in sys.stdin:
  data=line.split('\t') #separate contents of each data line into data elements
  if len(data) != 2:
    continue

  this_key=data[0]
  count = data[1]

  if old_key != None:    #all lines of data go through 1st condition (except 1st line)
    if old_key != this_key:  #when the new line contains a new turnstyle print
      print '{0}\t{1}'.format(old_key,Tot_ENTR_hr)
      Tot_ENTR_hr=float(count)
      old_key = this_key
    else: #when the new line is the same turn style add
      Tot_ENTR_hr +=float(count)
```

```
        old_key = this_key
    else:       #1st line of data goes through this condition to simply add
        Tot_ENTR_hr +=float(count)
        old_key = this_key


print '{0}\t{1}'.format(old_key, Tot_ENTR_hr)


reducer()
```

Problem Set 5: MapReduce on Subway Data > > 2 – Ridership by Weather Type

MAPPER:

```
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'

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")
Note that, unlike print, logging.info will take only a single argument.
So logging.info("my message") will work, but logging.info("my","message") will not.
'''

```
# 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_row=0
#key=0
entries=0
lines=0
count=0

for line in sys.stdin:  #cycle through lines of code
    lines+=1
    next=line.strip().split(",") #separate

    if Header_row == 0: #skip header line
        logging.info("UNIT")
        Header_row+=1
```

```
    continue


if len(next) !=22:  #remove lines without 22 items
    logging.info("REMOVE")
    continue
#key=next[0]
entries=next[6]
fog=float(next[14])
rain=float(next[15])
if fog == 1.0:
   if rain == 0.0:
      key="fog-norain"
      print "{0}\t{1}".format(key,entries) #tokenize weather type and ENTRIESn_hourly
      #count+=1
   else:
      key="fog-rain"
      print "{0}\t{1}".format(key,entries) #tokenize weather type and ENTRIESn_hourly
      #count+=1
else:
   if rain == 0.0:
      key="nofog-norain"
      print "{0}\t{1}".format(key,entries) #tokenize weather type and ENTRIESn_hourly
      #count+=1
   else:
      key="nofog-rain"
```

```
        print "{0}\t{1}".format(key,entries) #tokenize weather type and ENTRIESn_hourly
        count+=1


    logging.info("count")
    logging.info(count)
mapper()
```

REDUCER:

```
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'

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")
Note that, unlike print, logging.info will take only a single argument.
So logging.info("my message") will work, but logging.info("my","message") will not.
'''

```
riders_fnr = 0      # The number of total riders fog, no rain
num_hours_fnr = 0   # The number of hours etc...
riders_nfnr = 0
num_hours_nfnr = 0
riders_fr = 0
num_hours_fr = 0
riders_nfr = 0
num_hours_nfr = 0
```

```
#old_key = None
count=0
rider_count=0.0
key=()
i=0

for line in sys.stdin:
    data=line.split("\t") #separate each line
    #logging.info(data[0])
    rider_count=float(data[1])
    #logging.info(data[1])

    count+=1
    #logging.info(count)

    if data[0] == "fog-norain":
        riders_fnr += rider_count
        num_hours_fnr+= 1
    if data[0] == "nofog-norain":
        riders_nfnr += rider_count
        num_hours_nfnr+= 1
    if data[0] == "fog-rain":
```

```
        riders_fr += rider_count
        num_hours_fr+= 1
    if data[0] == "nofog-rain":
        riders_nfr += rider_count
        num_hours_nfr+= 1

key="fog-norain"
ave_riders=riders_fnr/num_hours_fnr
#logging.info(ave_riders)
print "{0}\t{1}".format(key,ave_riders)

key="fog-rain"
ave_riders=riders_fr/num_hours_fr
#logging.info(ave_riders)
print "{0}\t{1}".format(key,ave_riders)

key="nofog-norain"
ave_riders=riders_nfnr/num_hours_nfnr
#logging.info(ave_riders)
print "{0}\t{1}".format(key,ave_riders)

key="nofog-rain"
ave_riders=riders_nfr/num_hours_nfr
#logging.info(ave_riders)
print "{0}\t{1}".format(key,ave_riders)
```

reducer()

Problem Set 5: MapReduce on Subway Data > > 3 – Busiest Hour

MAPPER:

```
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():
    """
    In this exercise, for each turnstile unit, you will determine the date and time
    (in the span of this data set) at which the most people entered through the unit.

    The input to the 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, the mapper should return the UNIT, ENTRIESn_hourly, DATEn, and
    TIMEn columns, separated by tabs. For example:
    'R001\t100000.0\t2011-05-01\t01:00:00'
```

```
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")
Note that, unlike print, logging.info will take only a single argument.
So logging.info("my message") will work, but logging.info("my","message") will not.
"""

Header_row=0
unit="HI"
entries=0
lines=0.0
date=5/4/2011
#time=1:00:00
for line in sys.stdin: #cycle through lines of code
   lines+=1.0
   #logging.info(lines)
   next=line.strip().split(",") #separate

   if Header_row == 0: #skip header line
      #logging.info("UNIT")
      Header_row=Header_row+1
      continue
```

```
        if len(next) !=22:  #remove lines without 22 items
            logging.info("REMOVE")
            continue

        unit=next[1]
        #logging.info("unit")
        #logging.info(unit)

        entries=next[6]
        #logging.info("entries")
        #logging.info(entries)

        date=next[2]
        #logging.info("date")
        #logging.info(date)

        time=next[3]
        #logging.info("time")
        #logging.info(time)

        print "{0}\t{1}\t{2}\t{3}".format(unit,entries,date,time) #tokenize data into UNIT (key) and
ENTRIESn_hourly
    #logging.info("lines")
    #logging.info(lines)
```

mapper()

REDUCER:

```
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")

Note that, unlike print, logging.info will take only a single argument.

So logging.info("my message") will work, but logging.info("my","message") will not.

'''

old_key = None

datetime = ''

most_entries = 0.0

entires=0.0

```
for line in sys.stdin:
    data=line.strip().split("\t")

    if len(data)!=4:
        continue
        logging.info("remove")

    key=data[0]
    entries=float(data[1])



    if old_key != None:    #all lines of data go through 1st condition (except 1st line)
        if old_key and key!= old_key:
            print "{0}\t{1}\t{2}".format(old_key,datetime,most_entries)
            old_key=key
            most_entries = 0.0
            datetime=''
    else:
        old_key=key
        most_entries=entries
        datetime = data[2] + " " + data[3]

    if most_entries<=entries:
        most_entries=entries
```

```
            datetime = data[2]+ " " +data[3]


    print "{0}\t{1}\t{2}".format(old_key,datetime,most_entries)

reducer()
```