# Problem Set2: Wrangling Subway Data

## 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 CAST(rain as integer) = 1
 """
 #Execute your SQL command against the pandas frame
 rainy_days = pandasql.sqldf(q.lower(), locals())
 return rainy_days
```

```
count(*)
0       10
```

## 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 by your program below.


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

## 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) = 0 or CAST(strftime('%w', date) as integer) = 6
 """
#Execute your SQL command against the pandas frame
mean_temp_weekends = pandasql.sqldf(q.lower(), locals())
return mean_temp_weekends
```

```
Output by your program below.
```

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

## 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 mintempi > 55
"""
#Execute your SQL command against the pandas frame
avg_min_temp_rainy = pandasql.sqldf(q.lower(), locals())
return avg_min_temp_rainy
```

```
Output by your program below.


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

## 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. 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:
        f_in = open(name, 'r')
        f_out = open("updated_" + name, 'w')
        reader_in = csv.reader(f_in)
        reader_out = csv.writer(f_out)

        for line in reader_in:
            index = 3
            header=line[0:3]
            length = len(line)
            length2 = length - 3

            for index in range(3, length2 + 1):
                fw = header + line[index:(index+5)]
                reader_out.writerow(fw)
                index = index + 5
    return filenames
```

```
Good job. Your code worked perfectly.

Your code produced the following output:


updated_turnstile_110528.txt
```

## 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. The input files do not have column header rows of their own.

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:
        # your code here
        with open(filename) as infile:
            for line in infile:
                master_file.write(line)
```

```
Good job. Your code worked perfectly.

Your code produced the following output:

C/A,UNIT,SCP,DATEn,TIMEn,DESCn,ENTRIESn,EXITSn
```

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

***return turnstile_data***

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



       C/A  UNIT       SCP     DATEn     TIMEn    DESCn  ENTRIESn    EXITSn

0     A002  R051  02-00-00  05-01-11  00:00:00  REGULAR   3144312   1088151
```

## 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
    ...
#your code here
df['ENTRIESn_hourly'] = df['ENTRIESn'] - df['ENTRIESn'].shift(periods=1)
df = df.fillna(1)

return df
```

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



      Unnamed: 0   C/A  UNIT       SCP     DATEn     TIMEn    DESCn  ENTRIESn    EXIT
Sn  ENTRIESn_hourly

0              0  A002  R051  02-00-00  05-01-11  00:00:00  REGULAR   3144312   10881
51                1
```

## 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
'''
```

#your code here
***df['EXITSn_hourly'] = df['EXITSn'] - df['EXITSn'].shift(periods=1)***
***df = df.fillna(0)***
***return df***

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



     Unnamed: 0   C/A  UNIT       SCP     DATEn     TIMEn    DESCn  ENTRIESn    EXIT
Sn  ENTRIESn_hourly  EXITSn_hourly

0              0  A002  R051  02-00-00  05-01-11  00:00:00  REGULAR   3144312   10881
51                0              0
```

## 10 - Time to Hour:

***import pandas***

***import time as ti***

***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.
    '''
    # your code here
```

***hour = ti.strptime(time, "%H:%M:%S")[3]***

***return hour***

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




     Unnamed: 0  UNIT     DATEn     TIMEn    DESCn  ENTRIESn_hourly  EXITSn_hourly  H
our

0             0  R022  05-01-11  00:00:00  REGULAR                0              0
0

1             1  R022  05-01-11  04:00:00  REGULAR              562            173
4
```

## 11 - Reformat Subway Dates:

```
import datetime
import time as timer

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
   '''
# your code here
temp = timer.strptime(date, "%m-%d-%y")
t1 = temp[0]
t2 = temp[1]
t3 = temp[2]

dt = datetime.datetime(temp[0], temp[1], temp[2])

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
```