Lab 8

MSBA 6310, Fall 2019

Before You Begin

Before you begin, you should:

- Read Chapter 10 of your McKinney text
- Complete the following tutorials:
  - GroupBy: split-apply-combine
  - Merge, join, and concatenate
  - Time Series / Date functionality
- Watch the GroupBy Aggregation and Time Series video lecture materials
- Watch the HTML Primer and Web Scraping Example videos
- Review the following courses in DataCamp:
  - Intermediate Python for Data Science
  - pandas Foundations
  - Manipulating DataFrames with pandas
  - Manipulating Time Series Data in Python
- Complete the "Merging DataFrames with pandas" course in DataCamp

Make Steady Progress

To encourage you to make steady progress on your lab work throughout the week, you will earn up to 3 "steady progress points" as follows (you earn 1 point for meeting each bullet):

- 3 correct exercises before Tuesday at 8am
- 6 correct exercises before Thursday at 8am
- 9 correct exercises before Saturday at 8am

Exercises

Complete the following exercises and submit via GitHub. Be sure to name your files using the convention ex1.py, ex2.py, etc. The evaluation script will be expecting this convention. If you don't follow this convention, you won't earn credit for your work. Please do not compress your files!

The evaluation script will be looking at the following for each exercise:

- Correctness and adherence to the exercise requirements
- Understanding of data types and use of conversion functions
- Efficiency of code
- Proper file structure, with function definitions at the top
- Appropriate selection of data structures and types
- Do not alter any template code given to you
- Proper capture of return values from functions

Exercise 1

(3 points) The file `misspellings.csv` has a list of common English misspellings, perhaps for use in an autocorrect system. Write a function named `misspellings` that takes 2 parameters: a file name and a number n. Your function should return an array consisting of the words in the file that have n different misspellings. Here are a few example calls:

```
In [1]: misspellings('misspellings.csv', 4)
Out[1]:
array(['acquaintance', 'because', 'constitution', 'definitely',
       'essential', 'going', 'harassed', 'independently', 'maintenance',
       'necessarily', 'perhaps', 'privileges', 'successful', 'the',
       'unnecessary', 'very', 'with'], dtype=object)

In [2]: misspellings('misspellings.csv',5)
Out[2]:
array(['have', 'immediately', 'know', 'occasionally', 'occurrence',
       'parallel', 'parallelly', 'privilege', 'really', 'rhythm', 'that',
       'think', 'which', 'years'], dtype=object)

In [3]: misspellings('misspellings.csv',6)
Out[3]: array(['successfully'], dtype=object)
```

Exercise 2

(4 points) The files `manufacturers.csv` and `products.csv` contain the catalog for a simple computer store. Write a function named `avg_price_gt` that accepts 3 parameters: a prices file name, a manufacturers file name, and a price p. Your function should return a Series consisting of all manufacturers whose average product price is greater than p. Here are a few example calls:

```
In [1]: avg_price_gt('products.csv','manufacturers.csv',150)
Out[1]:
mfr_name
Fujitsu            240.0
Hewlett-Packard    168.0
Sony               240.0
Name: price, dtype: float64

In [2]: avg_price_gt('products.csv','manufacturers.csv',200)
Out[2]:
mfr_name
Fujitsu    240.0
Sony       240.0
Name: price, dtype: float64
```

Exercise 3

(4 points) Return to the `baseball.csv` file from previous labs. Write a function named `teams_w_n_players` that accepts two parameters: a file name and a number n. The function should return an array consisting of teams that have n players listed in the file. Here are some examples:

```
In [1]: teams_w_n_players('baseball.csv',7)
Out[1]: array(['Cardinals', 'Rangers', 'Tigers', 'Twins', 'Yankees'], dtype=object)

In [2]: teams_w_n_players('baseball.csv',6)
Out[2]: array(['Diamondbacks', 'Royals'], dtype=object)
```

Exercise 4

(4 points) Continue your analysis of the baseball file by writing a function named `team_avg` that accepts 3 parameters: a file name, a low batting average, and a high batting average. The function should return a Series where the index consists of all teams with a *team batting average* (defined as team total hits / team total at bats) that is within the range specified by the low batting average and high batting average parameters (inclusive). The team batting average is defined as the team's total hits / team's total at bats. Here are some example calls:

```
In [1]: team_avg('baseball.csv',.200,.250)
Out[1]:
team
Mariners    0.242
Phillies    0.250
Rays        0.241
Reds        0.249
Name: avg, dtype: float64

In [2]: team_avg('baseball.csv',.250,.275)
Out[2]:
team
Athletics    0.253
Blue Jays    0.251
Braves       0.266
Cubs         0.266
Dodgers      0.273
Indians      0.270
Nationals    0.272
Orioles      0.255
Padres       0.269
Phillies     0.250
Red Sox      0.270
Royals       0.270
White Sox    0.263
Name: avg, dtype: float64
```

Exercise 5

(4 points) The file `bank.csv` contains data about bank customers. The last column ('Personal Loan') indicates whether or not the customer was approved for a personal loan or not. Write a function named `loan_by_zip` that accepts 3 parameters: a file name, a minimum number of records, and a percentage approval rate. The function should return a DataFrame of those zip codes for which we meet the minimum number of records and the loan approval rate for that zip code (1 = approved, 0 = not approved). In this example, we return all zip codes that have at least 2 records and a loan approval rate of 50% or greater:

```
In [1]: loan_by_zip('bank.csv',2,.5)
Out[1]:
           mean  size
ZIP Code
90210       0.5     2
90254       0.5     2
91355       0.5     2
94131       0.5     2
94501       0.5     2
94704       0.5     2
94928       0.5     4
95054       1.0     2
95070       0.5     2
95211       0.5     2
95818       1.0     2
```

Exercise 6

(4 points) The file `flight_delays.csv` contains information about flight delays from the month of January. Write a function named `most_delayed_flights` that accepts 2 parameters: a file name and a percentage threshold. Your function should return a pandas Series listing all flight numbers that were delayed more often than the percentage threshold. In order to appear on the list, the flight must be regularly operated at least 3 times per week. Here is an example of all regular flights that were delayed more than 60% of the time:

```
In [1]: most_delayed_flights('flight_delays.csv',.6)
Out[1]:
CARRIER  FL_NUM
DH       7303      0.714286
MQ       4970      0.666667
Name: mean, dtype: float64
```

Exercise 7

(4 points) Your manager appreciates the work you did in the previous exercise, however she would like a small adjustment to the calculation. Modify your `most_delayed_flights` function from the previous exercise so that it only takes one parameter (the file name). Now the function should return all regular flights (operated 3x or more per week, meaning the flight is operated on at least 3 different days of the week) that are delayed more often than the overall delay rate *for that particular carrier*. Here is an example call and results:

```
In [1]: most_delayed_flights('flight_delays.csv')
Out[1]:
CARRIER  FL_NUM
CO       814       0.600000
DH       7211      0.600000
         7215      0.428571
         7302      0.333333
         7303      0.714286
         7304      0.333333
         7307      0.600000
         7812      0.285714
         7814      0.571429
DL       746       0.142857
         1766      0.333333
MQ       4752      0.285714
         4784      0.333333
         4968      0.600000
         4970      0.666667
         4976      0.600000
RU       2156      0.571429
         2261      0.400000
         2303      0.428571
         2336      0.500000
         2385      0.600000
         2403      0.333333
US       1479      0.333333
         2176      0.142857
         2178      0.250000
         2186      0.250000
dtype: float64
```

For example, Continental Airlines (CO) has an overall delay rate of almost 37%, so all CO flights delayed more often than this rate appear in the list. Discovery Airways (DH) has an overall delay rate of 25.6%, so all DH flights delayed more often than this rate appear in the list, and so on. Your manager feels that these results will be more indicative of problem flights!

Exercise 8

(4 points) Building on the analysis done in the misspellings file, you note that some misspellings are only wrong by one character (awkward vs. ackward), while some misspellings are off by many more. You are curious which words are the most egregiously misspelled.

For example, the word European has 3 misspellings: Europian (off by 1), Eurpean (off by 4), and Eurpoean (off by 2). The average character difference among misspellings is $(1 + 4 + 2)/3 = 2.33$. Write a function named `avg_char_diff` that takes 2 parameters: a file name and a number n. Your function should return a Series where the index consists of corrected words whose misspellings have an average character difference of n or more. Here are a few example calls:

```
In [1]: avg_char_diff('misspellings.csv',10)
Out[1]:
corrected word
accomplishment          11.0
accomplishments         12.0
aesthetically           10.0
anthropomorphization    11.0
autobiographic          11.0
autobiography           10.0
dissatisfaction         11.0
extraordinarily         10.0
municipalities          10.0
ophthalmologist         11.0
ophthalmology           10.0
parliamentarian         12.0
regardless              10.0
Name: diff, dtype: float64

In [2]: avg_char_diff('misspellings.csv',11)
Out[2]:
corrected word
accomplishment          11.0
accomplishments         12.0
anthropomorphization    11.0
autobiographic          11.0
dissatisfaction         11.0
ophthalmologist         11.0
parliamentarian         12.0
Name: diff, dtype: float64
```

```
In [3]: avg_char_diff('misspellings.csv',12)
Out[3]:
corrected word
accomplishments    12.0
parliamentarian    12.0
Name: diff, dtype: float64
```

NOTE: *One line in the data file is incorrectly formatted. How will you find it? How will you choose to deal with it?*

There are many ways to approach constructing this function, but here is one approach:

Step 1. Write a short helper function that accepts two words / strings and returns the character difference between the two. This function can use a loop to iterate over each word and count the differences. Here is an example of calling such a function:

```
In [1]: char_diff({'misspelled word': 'Eurpean','corrected word':'European'})
Out[1]: 4
```

Step 2. Use the DataFrame's apply method to create a new column in your DataFrame representing the char difference between the misspelled and corrected words. The intermediate DataFrame might look something like this:

```
      misspelled word   corrected word  diff
0            aberation       aberration     5
1              abondon          abandon     1
2            abondoned        abandoned     1
3             abondons         abandons     1
4           abondoning       abandoning     1
```

Step 3. Use the DataFrame's groupby method along with the appropriate aggregation function, and return the result.

Exercise 9

(4 points) The file `201406hourly.txt` (needs to be unzipped) contains hourly weather observations for Minneapolis for the month of June, 2014. Read this file into a pandas DataFrame. The index for the DataFrame should be the date/time of each observation. Note that, in the original file, the date and time are stored in separate columns. As we usually do, refer to the `read_csv` documentation for clues on how to combine these columns into a single index.

This exercise is designed to give you more practice with a large file that could take some time to load. To handle these situations, consider cutting and pasting a subset of the file into a new file. Use this new file to refine your code, then run on the full data set when you are satisfied with your results on the small data set. Another alternative that we have discussed is using the nrows parameter of `read_csv` to work on a subset of the data until you are satisfied with your logic.

Write a function to analyze this data named `four_hour_temp`. The function accepts 2 parameters: a file name and a date (as a string). Your function should return a Series containing the average temperatures ('DryBulbCelsius') for that date in 4 hour blocks. Here is an example:

```
In [1]: four_hour_temp('201406hourly.txt', '2014-06-17')
Out[1]:
Date_Time
2014-06-17 00:00:00    18.0
2014-06-17 04:00:00    18.5
2014-06-17 08:00:00    23.7
2014-06-17 12:00:00    26.2
2014-06-17 16:00:00    24.6
2014-06-17 20:00:00    20.5
Freq: 4H, Name: DryBulbCelsius, dtype: float64
```

Exercise 10

(4 points) The file `usgs_1930_2014.csv` contains data on US national Gross Domestic Product (GDP) and spending since the year 1930. Write a function named `adjusted_gdp` that accepts 4 parameters: an input file name, a file with CPI values (`cpiai.csv`), a start year, and an end year. Your function should return a pandas Series containing inflation-adjusted GDP values for the input range. Here is an example:

```
In [1]: adjusted_gdp('usgs_1930_2014.csv','cpiai.csv',2006,2008)
Out[1]:
2006-12-31    6874.24
2007-12-31    6983.76
2008-12-31    6837.03
Freq: A-DEC, Name: adjusted_gdp, dtype: float64
```

Your function will include the following basic steps:

Read in the file with a DateTimeIndex. First, read the file in to a DataFrame named `us_gdp` using the pandas `read_csv` function. You want the first column (the year) to be the index, so be sure pass the proper arguments to `read_csv`. You can read the documentation if you need a refresher.

Next, type the following at the iPython prompt:

```
us_gdp.index
```

Notice that the index consists of integer years. In order to use the time series manipulation functions, we need the index to be a DateTimeIndex instead of Int64Index. Look through the `read_csv` documentation and see if you can figure out how to make pandas interpret the first column as a date. When you are successful, you will note that the data type of your index changes from Int64Index to DateTimeIndex.

Shift the dates. By default, pandas will interpret a datetime field with only the year to be on Jan 1 of that year. However, GDP values are typically reported as of the end of the year. Write a statement to shift the index to be on Dec 31 of each year rather than Jan 1. To challenge yourself, you could consider altering the index to be a period index instead of a timestamp index.

Adjust for inflation. Any time that you deal with financial data over time, you need to be concerned with whether the data is reported in real dollars (adjusted for inflation) or absolute/nominal dollars. Assume these data are in nominal dollars. The CPI for 1982 - 1984 is 100, which is convenient, so convert each GDP value to this base currency. The video Investigating Real and Nominal Dollars (click on the title to access) gives simple explanation of how to do this using the US Consumer Price Index (CPI).

Exercise 11

(4 points) This exercise is designed to challenge you to apply the techniques from the previous exercise in a new environment. Students often ask how to scrape data from "live" web sites. This exercise will give you an opportunity to do that. We will scrape a few lines of data from the FRBG consumer credit report into a pandas DataFrame.

You can retrieve a URL from the web using the Python urllib module. Here is an example:

```
import urllib

r = urllib.request.urlopen('https://www.federalreserve.gov/releases/g19/current/default.htm')
soup = BeautifulSoup(r)
```

From here, the variable `soup` represents an object that accepts all BeautifulSoup commands, just like in the example and the previous exercise. Write a function named `total_consumer_credit` that returns the total outstanding revolving and non-revolving credit from the page linked above. Here is a sample call:

```
In [1]: total_consumer_credit()
Out[1]:
Out[17]:
                   2015    2016    2017    2018      Q3      Q4      Q1  \
Revolving         888.0   906.7   968.0  1022.1  1053.5  1040.5  1053.5
Nonrevolving 3   2424.5  2504.3  2676.2  2806.1  2956.3  2915.5  2956.3

                    Q2r     Q3p    Julr    Augr    Sepp
Revolving        1057.5  1071.2  1077.0  1080.3  1078.1
Nonrevolving 3   2995.1  3027.9  3072.3  3041.6  3061.7
```

The code to do this need not be lengthy. The key lies in understanding the structure of the web page. Spend some time with your browser's developer tools - in particular the "View Source" command - to study and understand the structure of the web site. Remember to try out BeautifulSoup commands in the iPython window before including them in your script.

Exercise 12 (Challenge Problem)

This is a designated challenge problem designed to challenge your thinking and logic skills. As you encounter challenge problems in the class, please keep things in perspective: these problems are worth only a small number of points.

(4 points) The file `results.csv` contains some of the raw results from this summer's (full-time MSBA version of) Programming Challenge 2. Write a function named `leaderboard` that accepts 2 arguments (a file name and a function name) and returns a pandas DataFrame representing the leaderboard for that particular function. In general, the leaderboard works like this:

- All teams with a valid solution appear on the leaderboard for $n = 1$ (or $pop = 101$, in the case of `new_nation_with_pop`)
- Teams advance to the next level (ex. $n = 2$ or $pop = 201$) if they returned an optimal solution for the previous level.
- If two teams return the same optimal solution for the same n (or pop), then they are ranked by time. The fastest team earns the higher rank.

Here are some examples - note that your index values might be different depending on your approach, or they might offer clues to a solution:

```
In [1]: leaderboard('results.csv', 'new_nation_n_states')
Out[1]:

       rank                                          team    n      output      time
11548     1                          Hamilton & Schumacher  100  3536078000  2.733212
8853      2                        One Ring Rules Them All  100  3534932000  3.294884
29332     3                                      Maverick   71  2555030000  0.582666
29774     4                                 Optimal Prime   71  2552619000  2.459172
26915     5  MERICA 2.0: A Supersized Order of Freedom Frie...   44  1634850000  4.844180
6762      6                                           306   38  1412255000  0.045689
23625     7                                         nunya   27  1006629000  1.764269
8629      8                             Run Flamingo Run    26   957227000  1.878788
30183     9                                      Big Bang   15   589199000  0.090104
29489    10                                     POP-si-CAL   15   589199000  0.141882
28945    11                             God Bless America !   15   589199000  0.228327
27164    12                                      Mjolnir   15   589199000  0.289198
13203    13                                   knapsackers   15   589199000  1.081757
25335    14                                Really Really    15   589199000  1.100752
26260    15                                      Blizzard   15   589199000  1.109587
30042    16                                Challenge Survivor   15   589199000  1.133468
19819    17                                   Coding Buddies   15   589199000  1.160678
2210     18                             That's what they said!   15   589199000  1.200670
29616    19                                   Bolivar Republic   13   509531000  0.153084
362      20                                    Hello World 2.0    7   261178000  0.109729
21527    21                                        Alohaha     6   237403000  0.522112
23444    22                                     King Cobra     6   226030000  0.267219
15893    23                                   take it and come    6   226030000  0.532576
29870    24                                        Cancun     6   222289000  0.007032
30274    25                                   Optimize Prime    6   222289000  0.008380
12994    26                                   The Liberators    6   222289000  0.106034
4703     27                                   Hack OâÄŽ Holics    6   212852000  0.015550
25312    28                             âĘŠCatch-Me-If-U-Can-(ÎęĬĹÎę)    5   222765000  4.684576
27075    29                                        go_cal     5   215561000  0.014742
7344     30                                       Akizuki     5   215561000  0.025458
29592    31                                   Really Really?    5   215561000  0.035513
29216    32                                     OptimusPrime    5   215561000  0.145832
29238    33                                         0.10>0.9    5   215561000  0.162742
22466    34                                      SodaGreen     5   215561000  0.734603
11033    35                                     STAT(US) QUO    5   215561000  1.029520
21085    36                                          L.Y.E.     5   215561000  4.352190
9880     37                                     ShunBorders     4   176017000  3.112592
30110    38                                         L&L Co.     3   130240000  0.014202
1949     39                                           Meow     3   118712000  0.006479
13036    40                             What would Ken Reily Do?     2    77511000  0.006855
28617    41                                   Chuck Norris     2    77511000  0.058487
27391    42                                      Yakiniku     2    77221000  0.007003
11642    43                                       KEBUKE     2    76926000  0.005794
15115    44                                          MAS     2    76926000  0.006100
25205    45                                divide_and_conquer     2    76926000  0.006119
29245    46                                      Gorillaz     2    76926000  0.006967
18960    47                                         M&M's     2    76926000  0.007256
10574    48                                    We took it!     2    76926000  0.007503
26670    49                                    Calexit 2.0     2    76926000  0.007504
23342    50                                       Sneakers     2    76926000  0.011972
21835    51                                       funcboom     2    76926000  0.012911
18138    52                                     R-B-trary     2    76926000  0.019329
27820    53                                   Pandas Express     2    76926000  0.019811
29973    54                                        Analytica     2    76926000  0.023105
4252     55                                    Optimus Prime     2    53953000  0.006548
3132     56                                         GSPedia     1    15513000  0.005904
22507    57                                Python Challenge 2     1    15513000  0.007864
22313    58                                    Slippery Hands     1    15513000  0.008879
```

```
In [1]: leaderboard('results.csv', 'new_nation_with_pop')
Out[1]:
       rank                                          team     n  output      time
11620     1                          Hamilton & Schumacher  3301      93  4.969122
14118     2                        One Ring Rules Them All  2101      56  2.319639
27789     3                                         nunya   901      24  4.293991
489       4                                           306   601      16  0.012528
26626     5                                  Chuck Norris   601      16  0.027505
28399     6                                      Big Bang   601      16  0.382320
28872     7                           God Bless America !   601      16  0.825223
29581     8                                   POP-si-CAL   601      16  1.339179
26923     9  MERICA 2.0: A Supersized Order of Freedom Frie...   601      16  1.340092
13379    10                                  knapsackers   601      16  1.393543
28386    11                                Optimal Prime   601      16  2.096339
22711    12                                      Akizuki   301       8  0.703553
27087    13                                       go_cal   301       8  1.013837
29602    14                                Really Really?   301       8  1.704582
29232    15                                      0.10>0.9   301       8  1.858044
560      16                                      Maverick   301       9  0.007725
9895     17                                  STAT(US) QUO   301       9  0.008067
15736    18                               Run Flamingo Run   301       9  0.008399
11138    19                                Hello World 2.0   301       9  0.009716
4354     20                                 Optimize Prime   301       9  0.012128
4917     21                                       Alohaha   301       9  0.013687
13996    22                               take it and come   301       9  0.014069
4652     23                                 Hack OâŽŹ Holics   301       9  0.037549
23246    24                                   OptimusPrime   301      10  0.006713
23437    25                                     King Cobra   201       5  0.767102
11743    26                                         KEBUKE   201       6  0.006574
17020    27                        âĘŠCatch-Me-If-U-Can-(ĨęĬĽĨę)   201       6  0.006774
9332     28                                           Meow   201       6  0.006778
26430    29                             divide_and_conquer   201       6  0.006854
23673    30                                      SodaGreen   201       6  0.006879
25355    31                                 Really Really    201       6  0.006957
27986    32                          That's what they said!   201       6  0.007015
26380    33                                       Yakiniku   201       6  0.007050
23196    34                                 Coding Buddies   201       6  0.007061
3841     35                               Bolivar Republic   201       6  0.007067
2695     36                                       Mjolnir   201       6  0.007115
29818    37                                        Cancun   201       6  0.007134
29981    38                             Challenge Survivor   201       6  0.007313
773      39                                      Analytica   201       6  0.007531
30061    40                                        L&L Co.   201       6  0.008004
27024    41                                       Gorillaz   201       6  0.008455
26771    42                                    Calexit 2.0   201       6  0.008676
17815    43                                       Blizzard   201       6  0.008996
27103    44                                 The Liberators   201       6  0.009107
29024    45                                          M&M's   201       6  0.009125
8940     46                        What would Ken Reily Do?   201       6  0.010574
15269    47                                       funcboom   201       6  0.012188
14592    48                                         L.Y.E.   201       6  0.097007
18635    49                                 Pandas Express   101       3  0.024040
9875     50                                    ShunBorders   101       3  1.462453
23293    51                                       Sneakers   101       4  0.006436
10191    52                                    We took it!   101       4  0.006466
3233     53                                        GSPedia   101       4  0.006591
22414    54                                 Slippery Hands   101       4  0.006639
4203     55                                  Optimus Prime   101       4  0.006845
18145    56                                     R-B-trary   101       4  0.006899
15215    57                                           MAS   101       4  0.011700
25180    58                                Python Challenge 2   101       4  0.097572
```