import glob

# There are four columns in each of the CSV files.  They are the 'first sale

# price' (purchase price), 'second sale price' (sale price), 'first sale date'

# (purchase data) and 'second sale date' (sale date).  The date in the last

# two columns is indicated in quarters, which will be further explained in the

# following instructions.  There are no headers in these documents.

# 1. Complete the function below to read multiple files into a single pandas

#    data frame.  This function takes three arguments:

#    - a string 'dir\_path'

#    - an array of strings that will be used as column names

#    - a string name of a new column

#    Your function should return a single DataFrame with the column names

#    as specified in the column\_names argument.  The resulting frame should

#    be indexed with unique integers, starting at 0.  For now, you can ignore

#    the third argument.

#    A sample data row is:

#    0     2550     3300       38   54

def read\_all\_files(dir\_path, column\_names, source\_column) :

data\_files = glob.glob(dir\_path)

frames = []

for f in data\_files:

#FILL THIS IN

#CREATE A NEW DATAFRAME CALLED new\_frame FROM FILE f and columns

frames.append(new\_frame)

return pd.concat(frames, ignore\_index=True)

# 2.  In a comment in this file, describe in your own words what this function

#     does, assuming file\_path is a string file path.

def add\_source\_column(frame, col\_name, file\_path):

frame[col\_name] = file\_path.split('/')[-1].split('.')[0]

# 3.  Now call the function in (2) from inside the for loop of the function in

#     (1) such that a sample row of your DataFrame is:

#    0     2550     3300       38   54  <something else here>

# 4.  Fill in the following function to add a year column.

#     This dataset contains data columns that are the

#     number of the quarter since 1970.  We want to know which year the quarter

#     number indicates.  For instance, '54' means the 54th quarter starting

#     from Q1:1970, i.e. year 1983.  Fill in the function below where the

#     arguments are the frame and the string name of an existing column.

#     Your function should return the frame with a new column called whose name

#     depends on 'col.'  The new column should be called XXXX\_year where XXXX is

#     the value of 'col'  If col is quarter\_num then the new column should be

#     quarter\_num\_year

#def append\_year(frame, col):

# 5.  Fill in the following function to find fast sales.  The function should

#     take a DataFrame 'frame' and an integer 'period' and return all rows

#     for homes that were sold within period quarters of their purchase.

#     Your function can add columns to frame if you wish.

#     Your function can have hard-coded column names.

#def fast\_sales(frame, period):

# 6.  Fill in the following function to find the profit on fast sales.

#     The funtcion should return the mean profit (a single number) for homes that

#     that were sole within period quarters of their purchase.  It should

#     take a DataFrame 'frame' and an integer 'period'.

#     Your function can add columns to frame if you wish.  You can

#     call the function from (5).

#     Your function can have hard-coded column names.

#def profit\_on\_fast\_sales(frame, period)

# 7.  Fill in the following function to count the sales by group.

#     The function should

#     take a DataFrame 'frame' and a string 'col\_name' which is the name

#     of the column to group by.

#     Your function should return the count of sales per group.  For our

#     datasets, using 'city' as the col\_name,

#     the function should return 4 rows like:

#  chicago  15530

#     Your function CANNOT have hard-coded column names.

#def sales\_by\_column(frame, col\_name) :