*You are currently looking at **version 1.1** of this notebook. To download notebooks and datafiles, as well as get help on Jupyter notebooks in the Coursera platform, visit the Jupyter Notebook FAQ (https://www.coursera.org/learn/python-data-analysis/resources/0dhYG) course resource.*

In [1]:
```
import pandas as pd
import numpy as np
from scipy.stats import ttest_ind
```

Assignment 4 - Hypothesis Testing

This assignment requires more individual learning than previous assignments - you are encouraged to check out the pandas documentation (http://pandas.pydata.org/pandas-docs/stable/) to find functions or methods you might not have used yet, or ask questions on Stack Overflow (http://stackoverflow.com/) and tag them as pandas and python related. And of course, the discussion forums are open for interaction with your peers and the course staff.

Definitions:

- A *quarter* is a specific three month period, Q1 is January through March, Q2 is April through June, Q3 is July through September, Q4 is October through December.
- A *recession* is defined as starting with two consecutive quarters of GDP decline, and ending with two consecutive quarters of GDP growth.
- A *recession bottom* is the quarter within a recession which had the lowest GDP.
- A *university town* is a city which has a high percentage of university students compared to the total population of the city.

Hypothesis: University towns have their mean housing prices less effected by recessions. Run a t-test to compare the ratio of the mean price of houses in university towns the quarter before the recession starts compared to the recession bottom. (`price_ratio=quarter_before_recession/recession_bottom`)

The following data files are available for this assignment:

- From the Zillow research data site (http://www.zillow.com/research/data/) there is housing data for the United States. In particular the datafile for all homes at a city level (http://files.zillowstatic.com/research/public/City/City_Zhvi_AllHomes.csv), `City_Zhvi_AllHomes.csv`, has median home sale prices at a fine grained level.
- From the Wikipedia page on college towns is a list of university towns in the United States (https://en.wikipedia.org/wiki/List_of_college_towns#College_towns_in_the_United_States) which has been copy and pasted into the file `university_towns.txt`.
- From Bureau of Economic Analysis, US Department of Commerce, the GDP over time (http://www.bea.gov/national/index.htm#gdp) of the United States in current dollars (use the chained value in 2009 dollars), in quarterly intervals, in the file `gdplev.xls`. For this assignment, only look at GDP data from the first quarter of 2000 onward.

Each function in this assignment below is worth 10%, with the exception of `run_ttest()`, which is worth 50%.

In [2]:
```
# Use this dictionary to map state names to two letter acronyms
states = {'OH': 'Ohio', 'KY': 'Kentucky', 'AS': 'American Samoa', 'NV': 'Nevada
', 'WY': 'Wyoming', 'NA': 'National', 'AL': 'Alabama', 'MD': 'Maryland', 'AK': '
Alaska', 'UT': 'Utah', 'OR': 'Oregon', 'MT': 'Montana', 'IL': 'Illinois', 'TN':
'Tennessee', 'DC': 'District of Columbia', 'VT': 'Vermont', 'ID': 'Idaho', 'AR':
'Arkansas', 'ME': 'Maine', 'WA': 'Washington', 'HI': 'Hawaii', 'WI': 'Wisconsin
', 'MI': 'Michigan', 'IN': 'Indiana', 'NJ': 'New Jersey', 'AZ': 'Arizona', 'GU':
'Guam', 'MS': 'Mississippi', 'PR': 'Puerto Rico', 'NC': 'North Carolina', 'TX':
'Texas', 'SD': 'South Dakota', 'MP': 'Northern Mariana Islands', 'IA': 'Iowa', '
MO': 'Missouri', 'CT': 'Connecticut', 'WV': 'West Virginia', 'SC': 'South Caroli
na', 'LA': 'Louisiana', 'KS': 'Kansas', 'NY': 'New York', 'NE': 'Nebraska', 'OK
': 'Oklahoma', 'FL': 'Florida', 'CA': 'California', 'CO': 'Colorado', 'PA': 'Pen
nsylvania', 'DE': 'Delaware', 'NM': 'New Mexico', 'RI': 'Rhode Island', 'MN': 'M
innesota', 'VI': 'Virgin Islands', 'NH': 'New Hampshire', 'MA': 'Massachusetts',
'GA': 'Georgia', 'ND': 'North Dakota', 'VA': 'Virginia'}
```

In [3]:
```
def get_list_of_university_towns():
    '''Returns a DataFrame of towns and the states they are in from the
    university_towns.txt list. The format of the DataFrame should be:
    DataFrame( [ ["Michigan", "Ann Arbor"], ["Michigan", "Yipsilanti"] ],
    columns=["State", "RegionName"]  )

    The following cleaning needs to be done:

    1. For "State", removing characters from "[" to the end.
    2. For "RegionName", when applicable, removing every character from " (" to
the end.
    3. Depending on how you read the data, you may need to remove newline charac
ter '\n'. '''

    # create basic datatframe to be filled
    df = pd.DataFrame([], columns=['State', 'RegionName'])
    # with function is used to avoid close() also helps assign it to a variable
in the same line
    with open('university_towns.txt', 'r') as f:
        state = ""
        for line in f:
            # line is each line in the txt file
            # print(line)
            if '[edit]' in line:
                # The find method gives output as index and therefore the strip
() removes everything after that index
                # line[:line.find('(')].strip() -> line[:3].strip() -> outputs a
ll value upto index 3
                state = line[:line.find('[')].strip()
                # print(state)
                # continue statement sends the loop control back to the beginnin
g, hence skipping all below lines of codes
                continue
            # n be any number; line[n].strip() -> retains the specific value at
that index and
            # strips everything else
            # print('this', line[3].strip())

            region = line.strip()
            if '(' in region:
                # print('this', region)
                region = region[:region.find('(')].strip()
                # region = region[:(region.find('(') - 1)]
                # print('here', region)
            df = df.append(pd.DataFrame([[state, region]], columns=['State', 'Re
gionName']), ignore_index=True)
    return df
get_list_of_university_towns()
```

Out[3]:

	State	RegionName
0	Alabama	Auburn
1	Alabama	Florence
2	Alabama	Jacksonville
3	Alabama	Livingston
4	Alabama	Montevallo
5	Alabama	Troy
6	Alabama	Tuscaloosa
7	Alabama	Tuskegee
8	Alaska	Fairbanks
9	Arizona	Flagstaff
10	Arizona	Tempe
11	Arizona	Tucson
12	Arkansas	Arkadelphia
13	Arkansas	Conway
14	Arkansas	Fayetteville
15	Arkansas	Jonesboro
16	Arkansas	Magnolia
17	Arkansas	Monticello
18	Arkansas	Russellville
19	Arkansas	Searcy
20	California	Angwin
21	California	Arcata
22	California	Berkeley
23	California	Chico
24	California	Claremont
25	California	Cotati
26	California	Davis
27	California	Irvine
28	California	Isla Vista
29	California	University Park, Los Angeles
...	...	...
487	Virginia	Wise
488	Virginia	Chesapeake
489	Washington	Bellingham
490	Washington	Cheney
491	Washington	Ellensburg
492	Washington	Pullman

In [4]:
```
def get_gdp_data():
    # Cleans GDP data in "gdplev.xls"

    df = pd.read_excel('gdplev.xls')
    df = (df.drop(['Current-Dollar and "Real" Gross Domestic Product',
                   'Unnamed: 1', 'Unnamed: 2', 'Unnamed: 3', 'Unnamed: 5'],
                  axis=1)
             .ix[7:] # same as iloc, can use iloc as well no problem
             .rename(columns={'Unnamed: 4': 'Quarter', 'Unnamed: 6': 'GDP'})
             .set_index('Quarter'))
    # getting index for the start of year 2000
    index = df.index.get_loc('2000q1')
    df = df.ix[index:]
    return df
get_gdp_data()
```

Out[4]:

	GDP	2016-09-29 00:00:00
Quarter		
2000q1	12359.1	NaN
2000q2	12592.5	NaN
2000q3	12607.7	NaN
2000q4	12679.3	NaN
2001q1	12643.3	NaN
2001q2	12710.3	NaN
2001q3	12670.1	NaN
2001q4	12705.3	NaN
2002q1	12822.3	NaN
2002q2	12893	NaN
2002q3	12955.8	NaN
2002q4	12964	NaN
2003q1	13031.2	NaN
2003q2	13152.1	NaN
2003q3	13372.4	NaN
2003q4	13528.7	NaN
2004q1	13606.5	NaN
2004q2	13706.2	NaN
2004q3	13830.8	NaN
2004q4	13950.4	NaN
2005q1	14099.1	NaN
2005q2	14172.7	NaN
2005q3	14291.8	NaN
2005q4	14373.4	NaN
2006q1	14546.1	NaN
2006q2	14589.6	NaN
2006q3	14602.6	NaN
2006q4	14716.9	NaN
2007q1	14726	NaN
2007q2	14838.7	NaN
...	...	...
2009q1	14375	NaN
2009q2	14355.6	NaN
2009q3	14402.5	NaN
2009q4	14541.9	NaN
2010q1	14604.8	NaN

In [5]:
```
def get_recession_start():
    '''Returns the year and quarter of the recession start time as a
    string value in a format such as 2005q3'''

    df = pd.read_excel('gdplev.xls')
    df = (df.drop(['Current-Dollar and "Real" Gross Domestic Product',
                   'Unnamed: 1', 'Unnamed: 2', 'Unnamed: 3', 'Unnamed: 5'],
                  axis=1)
            .ix[7:] # same as iloc, can use iloc as well no problem
            .rename(columns={'Unnamed: 4': 'Quarter', 'Unnamed: 6': 'GDP'})
            .set_index('Quarter'))
    # getting index for the start of year 2000
    index = df.index.get_loc('2000q1')
    df = df.ix[index:]

    for i in range(1, len(df) - 1):
        if (df.iloc[i]['GDP'] < df.iloc[i - 1]['GDP']) and (df.iloc[i + 1]['GDP
'] < df.iloc[i]['GDP']):
            return df.iloc[i].name
    return None

get_recession_start()
```

Out[5]: '2008q3'

In [6]:
```
def get_recession_end():
    '''Returns the year and quarter of the recession end time as a
    string value in a format such as 2005q3'''

    df = get_gdp_data()
    recession_start = get_recession_start()
    index = df.index.get_loc(recession_start)
    for i in range(index + 2, len(df)):
        if (df.iloc[i]['GDP'] > df.iloc[i - 1]['GDP']) and (df.iloc[i - 1]['GDP
'] > df.iloc[i - 2]['GDP']):
            return df.iloc[i].name
    return None

get_recession_end()
```

Out[6]: '2009q4'

In [7]:
```
def get_recession_bottom():
    '''Returns the year and quarter of the recession bottom time as a
    string value in a format such as 2005q3'''



    df = get_gdp_data()
    start = df.index.get_loc(get_recession_start())
    end = df.index.get_loc(get_recession_end())
    table = df['GDP'][start:end + 1]
    year = df[df['GDP'] == np.min(table)].iloc[0].name
    return year

get_recession_bottom()
```

Out[7]: '2009q2'

In [8]:
```
def get_quarter(year, month):
    if month <= 3:
        quarter = 1
    elif month <= 6:
        quarter = 2
    elif month <= 9:
        quarter = 3
    elif month <= 12:
        quarter = 4
    return (str(year) + 'q' + str(quarter))
```

In [9]:
```
def convert_housing_data_to_quarters():
    '''Converts the housing data to quarters and returns it as mean
    values in a dataframe. This dataframe should be a dataframe with
    columns for 2000q1 through 2016q3, and should have a multi-index
    in the shape of ["State","RegionName"].

    Note: Quarters are defined in the assignment description, they are
    not arbitrary three month periods.

    The resulting dataframe should have 67 columns, and 10,730 rows.
    '''



    df = pd.read_csv('City_Zhvi_AllHomes.csv')
    #print(df)
    df = (df.drop(['RegionID', 'Metro', 'CountyName', 'SizeRank'], axis=1)
             .replace({'State': states})
             .set_index(['State', 'RegionName'])
             .replace(to_replace='NaN', value=np.NaN)
             .convert_objects(convert_numeric=True)
             .sort())
    index = list(df.columns.values).index('2000-01')
    df = df.drop(df.columns[:index], axis=1)
    l = len(df.columns)
    # print(l) -> 200
    i = 0
    while i < l:
        col_name = df.iloc[:, i].name
        year = int(col_name.split('-')[0])
        month = int(col_name.split('-')[1])
        quarter = get_quarter(year, month)
        if i + 3 < l:
            split = df.iloc[:, i:i + 3]
        else:
            split = df.iloc[:, i:l]
        df[quarter] = split.mean(axis=1)
        i += 3
    df = df.drop(df.columns[:l], axis=1)
    return df

convert_housing_data_to_quarters()
```

Out[9]:

		2000q1	2000q2	2000q3	2000q4	2001q1
State	**RegionName**					
Alabama	**Adamsville**	69033.333333	69166.666667	69800.000000	71966.666667	73466.6666
	Alabaster	122133.333333	123066.666667	123166.666667	123700.000000	123233.333
	Albertville	73966.666667	72600.000000	72833.333333	74200.000000	75900.0000
	Arab	83766.666667	81566.666667	81333.333333	82966.666667	84200.0000
	Ardmore	NaN	NaN	NaN	NaN	NaN
	Axis	NaN	NaN	NaN	NaN	NaN
	Baileyton	NaN	NaN	NaN	NaN	NaN
	Bay Minette	81700.000000	78533.333333	79133.333333	81300.000000	85700.0000
	Bayou La Batre	44066.666667	44500.000000	44266.666667	43666.666667	42500.0000
	Bessemer	NaN	NaN	NaN	NaN	NaN
	Birmingham	54033.333333	54400.000000	54966.666667	56066.666667	56833.3333
	Boaz	70866.666667	70266.666667	70300.000000	71466.666667	72833.3333
	Brent	92933.333333	94333.333333	96166.666667	98333.333333	96533.3333
	Brighton	NaN	NaN	NaN	NaN	NaN
	Brookwood	92566.666667	95100.000000	98866.666667	99966.666667	101666.666
	Buhl	90800.000000	94600.000000	96500.000000	96566.666667	99266.6666
	Calera	108933.333333	110366.666667	108000.000000	107933.333333	109333.333
	Center Point	80966.666667	81233.333333	81500.000000	82233.333333	83366.6666
	Centreville	95300.000000	96566.666667	98000.000000	99366.666667	98100.0000
	Chalkville	94100.000000	94433.333333	94433.333333	94433.333333	96066.6666
	Chancellor	NaN	NaN	NaN	NaN	NaN
	Chelsea	162066.666667	167033.333333	166900.000000	167400.000000	170633.333
	Chickasaw	51200.000000	53666.666667	54933.333333	56066.666667	55133.3333
	Chunchula	80266.666667	81766.666667	82200.000000	83400.000000	85400.0000
	Citronelle	64833.333333	66633.333333	68066.666667	67766.666667	67866.6666
	Clay	120900.000000	122266.666667	123966.666667	126500.000000	128033.333
	Coden	62600.000000	64800.000000	66866.666667	68566.666667	68933.3333
	Coker	118100.000000	120766.666667	118166.666667	115333.333333	114066.666
	Concord	78600.000000	78700.000000	80133.333333	82800.000000	85133.3333
	Cottondale	100833.333333	102633.333333	104766.666667	105300.000000	106733.333
...	**...**	...	...	...	...	...
	Vernon	183200.000000	178200.000000	174300.000000	177466.666667	186233.333
	Vienna	178033.333333	181533.333333	182433.333333	186300.000000	190600.000
	Vinland	119800.000000	126766.666667	134933.333333	137833.333333	144300.000
	Wales	NaN	NaN	NaN	NaN	NaN

In [10]:
```
def run_ttest():
    '''First creates new data showing the decline or growth of housing prices
    between the recession start and the recession bottom. Then runs a ttest
    comparing the university town values to the non-university towns values,
    return whether the alternative hypothesis (that the two groups are the same)
    is true or not as well as the p-value of the confidence.

    Return the tuple (different, p, better) where different=True if the t-test i
s
    True at a p<0.01 (we reject the null hypothesis), or different=False if
    otherwise (we cannot reject the null hypothesis). The variable p should
    be equal to the exact p value returned from scipy.stats.ttest_ind(). The
    value for better should be either "university town" or "non-university town"
    depending on which has a lower mean price ratio (which is equivilent to a
    reduced market loss).'''



    hdf = convert_housing_data_to_quarters()
    # print(hdf)
    start_index = hdf.columns.get_loc(get_recession_start())
    bottom_index = hdf.columns.get_loc(get_recession_bottom())
    hdf['Ratio'] = hdf.iloc[:, start_index - 1] / hdf.iloc[:, bottom_index]
    # print(hdf.iloc[:, start_index - 1])
    hdf = pd.DataFrame(hdf.loc[:, 'Ratio'])
    # print(hdf)
    ul = get_list_of_university_towns()
    ul = ul.set_index(['State', 'RegionName'])
    univ_prices = pd.merge(hdf, ul, how="inner", left_index=True, right_index=Tr
ue)
    non_univ = pd.merge(hdf, ul, how="outer", left_index=True, right_index=True,
indicator=True)
    non_univ = non_univ[non_univ['_merge'] == 'left_only']
    non_univ = non_univ.drop('_merge', axis=1)
    univ_prices = univ_prices.dropna()
    non_univ = non_univ.dropna()
    s, p = ttest_ind(univ_prices['Ratio'], non_univ['Ratio'])
    s2, p2 = ttest_ind(non_univ['Ratio'], univ_prices['Ratio'])
    ans = True, p, "university town"
    return ans

run_ttest()
```

Out[10]:
```
(True, 0.002724063704761164, 'university town')
```

In [ ]:
```

```