*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)](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': 'Ne
vada', 'WY': 'Wyoming', 'NA': 'National', 'AL': 'Alabama', 'MD': 'Maryland'
, 'AK': 'Alaska', 'UT': 'Utah', 'OR': 'Oregon', 'MT': 'Montana', 'IL': 'Ill
inois', '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': 'Connecti
cut', 'WV': 'West Virginia', 'SC': 'South Carolina', 'LA': 'Louisiana', 'K
S': 'Kansas', 'NY': 'New York', 'NE': 'Nebraska', 'OK': 'Oklahoma', 'FL':
'Florida', 'CA': 'California', 'CO': 'Colorado', 'PA': 'Pennsylvania', 'DE'
: 'Delaware', 'NM': 'New Mexico', 'RI': 'Rhode Island', 'MN': 'Minnesota',
'VI': 'Virgin Islands', 'NH': 'New Hampshire', 'MA': 'Massachusetts', 'GA':
'Georgia', 'ND': 'North Dakota', 'VA': 'Virginia'}
```

**YC 答案**

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 c
haracter '\n'. '''
    with open('university_towns.txt') as file:
        data = []
        for line in file:
            data.append(line[:-1])

    state_down = []
    for line in data:
        if line[-6:] == '[edit]':
            state = line[:-6]
        elif '(' in line:
            town = line[:(line.index('(')-1)]
            # 因为每次州开始时，那行没有 town 名
            # 如果把 state_down.append 写在最后
            # 则新的州，后面会添加之前的 town 名
            # 所以将 append 写在每次生成 town 之后
            state_down.append([state, town])
        else:
            town = line.strip()
            state_down.append([state, town])
    return pd.DataFrame(state_down, columns = ['State', 'RegionName'])
```

计算过程

In [4]:
```
with open('university_towns.txt') as file:
    data = []
    for line in file:
        data.append(line[:-1])
data[:2]
```

Out[4]:
```
['Alabama[edit]', 'Auburn (Auburn University)[1]']
```

In [5]:
```
state_down = []
for line in data:
    if line[-6:] == '[edit]':
        state = line[:-6]
        #print("State :", state)
    elif '(' in line:
        town = line[:(line.index('(')-1)]
        #print("Town :", town)
        # 因为每次州开始时，那行没有 town 名
        # 如果把 state_down.append 写在最后
        # 则新的州，后面会添加之前的 town 名
        # 所以将 append 写在每次生成 town 之后
        state_down.append([state, town])
    else:
        town = line.strip()
        #print("Town 2 :", town)
        state_down.append([state, town])
state_down[:2]
```

Out[5]:
```
[['Alabama', 'Auburn'], ['Alabama', 'Florence']]
```

In [6]:
```
df = pd.DataFrame(state_down, columns = ['State', 'RegionName'])
df.head(2)
```

Out[6]:

| | State | RegionName |
|---|---|---|
| **0** | Alabama | Auburn |
| **1** | Alabama | Florence |

**YC 答案**

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

    # only look at GDP data from the first quarter of 2000 onward.
    df = pd.read_excel('gdplev.xls', skiprows = 220, header = None)
    df = df.iloc[:,[4,5]]

    recession_start = []
    for i in range(len(df) - 4):
        if ((df.iloc[i][5] > df.iloc[i+1][5]) and (df.iloc[i+1][5] > df.ilo
c[i+2][5])):
            recession_start.append(df.iloc[i][4])

    return recession_start[0]
```

计算过程

In [8]:
```
# only look at GDP data from the first quarter of 2000 onward.
df = pd.read_excel('gdplev.xls', skiprows = 220, header = None)
df = df.iloc[:,[4,5]]
df.head()
```

Out[8]:

| | 4 | 5 |
|---|---|---|
| **0** | 2000q1 | 10031.0 |
| **1** | 2000q2 | 10278.3 |
| **2** | 2000q3 | 10357.4 |
| **3** | 2000q4 | 10472.3 |
| **4** | 2001q1 | 10508.1 |

In [9]:
```
recession_start = []
for i in range(len(df) - 4):
    if ((df.iloc[i][5] > df.iloc[i+1][5]) and (df.iloc[i+1][5] > df.iloc[i+
2][5])):
        recession_start.append(df.iloc[i][4])
recession_start
```

Out[9]: ['2008q3', '2008q4']

In [10]:
```
recession_start[0]
```

Out[10]: '2008q3'

**YC** 答案

In [11]:
```
def get_recession_end():
    '''Returns the year and quarter of the recession end time as a
    string value in a format such as 2005q3'''
    # only look at GDP data from the first quarter of 2000 onward.
    df = pd.read_excel('gdplev.xls', skiprows = 220, header = None)
    df = df.iloc[:,[4,5]]

    recession_end = []

    for i in range(len(df) - 4):
        if ((df.iloc[i][5] > df.iloc[i+1][5]) and (df.iloc[i+1][5] > df.ilo
c[i+2][5]) and
           (df.iloc[i+2][5] < df.iloc[i+3][5]) and (df.iloc[i+3][5] < df.il
oc[i+4][5])):
            recession_end.append([df.iloc[i][4], df.iloc[i+1][4],
                                  df.iloc[i+2][4], df.iloc[i+3][4], df.iloc
[i+4][4]])

    return recession_end[0][4]
```

计算过程

In [12]:
```
# only look at GDP data from the first quarter of 2000 onward.
df = pd.read_excel('gdplev.xls', skiprows = 220, header = None)
df = df.iloc[:,[4,5]]
```

In [13]:
```
recession_end = []

for i in range(len(df) - 4):
    if ((df.iloc[i][5] > df.iloc[i+1][5]) and (df.iloc[i+1][5] > df.iloc[i+
2][5]) and
       (df.iloc[i+2][5] < df.iloc[i+3][5]) and (df.iloc[i+3][5] < df.iloc[i
+4][5])):
        recession_end.append([df.iloc[i][4], df.iloc[i+1][4],
                              df.iloc[i+2][4], df.iloc[i+3][4], df.iloc[i+
4][4]])
recession_end
```

Out[13]:
```
[['2008q4', '2009q1', '2009q2', '2009q3', '2009q4']]
```

In [14]:
```
recession_end[0][4]
```

Out[14]:
```
'2009q4'
```

**YC** 答案

In [15]:
```
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 = pd.read_excel('gdplev.xls', skiprows = 220, header = None)
    df = df.iloc[:,[4,5]]

    recession_end = []

    for i in range(len(df) - 4):
        if ((df.iloc[i][5] > df.iloc[i+1][5]) and (df.iloc[i+1][5] > df.ilo
c[i+2][5]) and
           (df.iloc[i+2][5] < df.iloc[i+3][5]) and (df.iloc[i+3][5] < df.il
oc[i+4][5])):
            recession_end.append([df.iloc[i][4], df.iloc[i+1][4],
                                  df.iloc[i+2][4], df.iloc[i+3][4], df.iloc
[i+4][4]])

    return recession_end[0][2]
```

计算过程

In [16]:
```
# 因为出现 recession 后，连续 2 个 quarter GDP 回升，意味着 recession 结束。
# 所以倒数第 3 个 quarter 的 GDP 应该是最低的。
recession_end[0][2]
```

Out[16]: '2009q2'

**YC** 答案

In [17]:
```
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.
    '''
    # 读取数据
    df = pd.read_csv('City_Zhvi_AllHomes.csv')
    # 仅提取 2000 年后，以及 State 和 RegionName 列
    list_col = [1, 2] + list(range(51,251))
    df = df.iloc[:, list_col]
    # 生成新 DataFrame，先仅包含 State 和 RegionName 列
    # 直接用 df[['RegionName', 'State']] 会生成 SettingWithCopyWarning:
    # A value is trying to be set on a copy of a slice from a DataFrame.
    # Try using .loc[row_indexer,col_indexer] = value instead
    df_qrt = df.loc[:, ['RegionName', 'State']]
    # 因为 2016 不足最后两个季度，所以单列
    for year in range(2000,2016):
        df_qrt[str(year) + 'q1'] = df[[str(year) + '-01', str(year) + '-02'
, str(year) + '-03']].mean(axis = 1)
        df_qrt[str(year) + 'q2'] = df[[str(year) + '-04', str(year) + '-05'
, str(year) + '-06']].mean(axis = 1)
        df_qrt[str(year) + 'q3'] = df[[str(year) + '-07', str(year) + '-08'
, str(year) + '-09']].mean(axis = 1)
        df_qrt[str(year) + 'q4'] = df[[str(year) + '-10', str(year) + '-11'
, str(year) + '-12']].mean(axis = 1)
    # 2016 只有 8 个月，不能完整构成第 3，4 季度
    year = 2016
    df_qrt[str(year) + 'q1'] = df[[str(year) + '-01', str(year) + '-02', st
r(year) + '-03']].mean(axis = 1)
    df_qrt[str(year) + 'q2'] = df[[str(year) + '-04', str(year) + '-05', st
r(year) + '-06']].mean(axis = 1)
    df_qrt[str(year) + 'q3'] = df[[str(year) + '-07', str(year) + '-08']].m
ean(axis = 1)
    # Use this dictionary to map state names to two letter acronyms
    df_qrt['State'] = [states[State] for State in df_qrt['State']]
    # have a multi-index in the shape of ["State","RegionName"]
    df_qrt.set_index(['State', 'RegionName'], inplace = True)

    return df_qrt
```

计算过程

In [18]:
```
# 读取数据
df = pd.read_csv('City_Zhvi_AllHomes.csv')
df.head(2)
```

Out[18]:

| | RegionID | RegionName | State | Metro | CountyName | SizeRank | 1996-04 | 1996-05 | 1996-06 |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 6181 | New York | NY | New York | Queens | 1 | NaN | NaN | NaN |
| **1** | 12447 | Los Angeles | CA | Los Angeles-Long Beach-Anaheim | Los Angeles | 2 | 155000.0 | 154600.0 | 154400.0 |

2 rows × 251 columns

In [19]:
```
# 仅提取 2000 年后，以及 State 和 RegionName 列
list_col = [1, 2] + list(range(51,251))
df = df.iloc[:, list_col]
df.head(2)
```

Out[19]:

| | RegionName | State | 2000-01 | 2000-02 | 2000-03 | 2000-04 | 2000-05 | 2000-06 | 2000-07 | 2 |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | New York | NY | NaN | NaN | NaN | NaN | NaN | NaN | NaN | |
| **1** | Los Angeles | CA | 204400.0 | 207000.0 | 209800.0 | 212300.0 | 214500.0 | 216600.0 | 219000.0 | 2 |

2 rows × 202 columns

In [20]:
```
# 生成新 DataFrame，先仅包含 State 和 RegionName 列
# 直接用 df[['RegionName', 'State']] 会生成 SettingWithCopyWarning:
# A value is trying to be set on a copy of a slice from a DataFrame.
# Try using .loc[row_indexer,col_indexer] = value instead
df_qrt = df.loc[:, ['RegionName', 'State']]
df_qrt.head(2)
```

Out[20]:

| | RegionName | State |
|---|---|---|
| **0** | New York | NY |
| **1** | Los Angeles | CA |

In [21]:

```
# 因为 2016 不足最后两个季度，所以单列
for year in range(2000,2016):
    df_qrt[str(year) + 'q1'] = df[[str(year) + '-01', str(year) + '-02', st
r(year) + '-03']].mean(axis = 1)
    df_qrt[str(year) + 'q2'] = df[[str(year) + '-04', str(year) + '-05', st
r(year) + '-06']].mean(axis = 1)
    df_qrt[str(year) + 'q3'] = df[[str(year) + '-07', str(year) + '-08', st
r(year) + '-09']].mean(axis = 1)
    df_qrt[str(year) + 'q4'] = df[[str(year) + '-10', str(year) + '-11', st
r(year) + '-12']].mean(axis = 1)
df_qrt.head(2)
```

Out[21]:

| | RegionName | State | 2000q1 | 2000q2 | 2000q3 | 2000q4 | 2001q1 |
|---|---|---|---|---|---|---|---|
| **0** | New York | NY | NaN | NaN | NaN | NaN | NaN |
| **1** | Los Angeles | CA | 207066.666667 | 214466.666667 | 220966.666667 | 226166.666667 | 233000.0 |

2 rows × 66 columns

In [22]:

```
# 2016 只有 8 个月，不能完整构成第 3，4 季度
year = 2016
df_qrt[str(year) + 'q1'] = df[[str(year) + '-01', str(year) + '-02', str(ye
ar) + '-03']].mean(axis = 1)
df_qrt[str(year) + 'q2'] = df[[str(year) + '-04', str(year) + '-05', str(ye
ar) + '-06']].mean(axis = 1)
df_qrt[str(year) + 'q3'] = df[[str(year) + '-07', str(year) + '-08']].mean(
axis = 1)
df_qrt.head(2)
```

Out[22]:

| | RegionName | State | 2000q1 | 2000q2 | 2000q3 | 2000q4 | 2001q1 |
|---|---|---|---|---|---|---|---|
| **0** | New York | NY | NaN | NaN | NaN | NaN | NaN |
| **1** | Los Angeles | CA | 207066.666667 | 214466.666667 | 220966.666667 | 226166.666667 | 233000.0 |

2 rows × 69 columns

In [23]:
```
# Use this dictionary to map state names to two letter acronyms
df_qrt['State'] = [states[State] for State in df_qrt['State']]
df_qrt.head()
```

Out[23]:

| | RegionName | State | 2000q1 | 2000q2 | 2000q3 | 2000q4 | |
|---|---|---|---|---|---|---|---|
| **0** | New York | New York | NaN | NaN | NaN | NaN | |
| **1** | Los Angeles | California | 207066.666667 | 214466.666667 | 220966.666667 | 226166.666667 | 233 |
| **2** | Chicago | Illinois | 138400.000000 | 143633.333333 | 147866.666667 | 152133.333333 | 156 |
| **3** | Philadelphia | Pennsylvania | 53000.000000 | 53633.333333 | 54133.333333 | 54700.000000 | 55 |
| **4** | Phoenix | Arizona | 111833.333333 | 114366.666667 | 116000.000000 | 117400.000000 | 119 |

5 rows × 69 columns

In [24]:
```
df_qrt.set_index(['State', 'RegionName'], inplace = True)
df_qrt.head(2)
```

Out[24]:

| | | 2000q1 | 2000q2 | 2000q3 | 2000q4 | 2001q1 |
|---|---|---|---|---|---|---|
| **State** | **RegionName** | | | | | |
| **New York** | **New York** | NaN | NaN | NaN | NaN | NaN |
| **California** | **Los Angeles** | 207066.666667 | 214466.666667 | 220966.666667 | 226166.666667 | 233000.0 |

2 rows × 67 columns

**YC** 答案

In [25]:
```
def run_ttest():
    '''First creates new data showing the decline or growth of housing pric
es
    between the recession start and the recession bottom. Then runs a ttest
    comparing the university town values to the non-university towns value
s,
    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-t
est is
    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(). Th
e
    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).'''

    college_town = get_list_of_university_towns()
    house_qrt = convert_housing_data_to_quarters()
    start = get_recession_start()
    bottom = get_recession_bottom()

    # housing prices between the recession start and the recession bottom
    house_qrt['ratio'] = house_qrt[start] - house_qrt[bottom]

    house_qrt = house_qrt.loc[:, [bottom, start, 'ratio']]
    house_qrt.reset_index(inplace = True)

    # 找出学区房以及价钱
    college_house = pd.merge(college_town, house_qrt, on = ['State', 'Regio
nName'])
    # 找出非学区房价钱，即总表排除学区房
    non_college = pd.merge(house_qrt, college_house, how = 'outer', indicat
or = True)
    # 通过 merge 中 indicator 产生新列，把不是 'both' 的排除，即得出非学区房
    non_college = non_college.loc[non_college._merge == 'left_only', :]

    t, p = ttest_ind(college_house['ratio'].dropna(), non_college['ratio'].
dropna())

    different = True if p < 0.01 else False

    better = "university town" if college_house['ratio'].mean() < non_colle
ge['ratio'].mean() else "non-university town"

    return (different, p, better)
```

计算过程

In [26]:
```
college_town = get_list_of_university_towns()
college_town.head(2)
```

Out[26]:

| | State | RegionName |
|---|---|---|
| **0** | Alabama | Auburn |
| **1** | Alabama | Florence |

In [27]:
```
house_qrt = convert_housing_data_to_quarters()
house_qrt.head(2)
```

Out[27]:

| | | 2000q1 | 2000q2 | 2000q3 | 2000q4 | 2001q1 |
|---|---|---|---|---|---|---|
| **State** | **RegionName** | | | | | |
| **New York** | **New York** | NaN | NaN | NaN | NaN | NaN |
| **California** | **Los Angeles** | 207066.666667 | 214466.666667 | 220966.666667 | 226166.666667 | 233000.0 |

2 rows × 67 columns

In [28]:
```
start = get_recession_start()
start
```

Out[28]: '2008q3'

In [29]:
```
bottom = get_recession_bottom()
bottom
```

Out[29]: '2009q2'

In [30]:
```
# housing prices between the recession start and the recession bottom
house_qrt['ratio'] = house_qrt[start] - house_qrt[bottom]
house_qrt.head(2)
```

Out[30]:

| | | 2000q1 | 2000q2 | 2000q3 | 2000q4 | 2001q1 |
|---|---|---|---|---|---|---|
| **State** | **RegionName** | | | | | |
| **New York** | **New York** | NaN | NaN | NaN | NaN | NaN |
| **California** | **Los Angeles** | 207066.666667 | 214466.666667 | 220966.666667 | 226166.666667 | 233000.0 |

2 rows × 68 columns

In [31]:
```
house_qrt = house_qrt.loc[:, [bottom, start, 'ratio']]
house_qrt.reset_index(inplace = True)
house_qrt.head(2)
```

Out[31]:

| | State | RegionName | 2009q2 | 2008q3 | ratio |
|---|---|---|---|---|---|
| **0** | New York | New York | 465833.333333 | 499766.666667 | 33933.333333 |
| **1** | California | Los Angeles | 413900.000000 | 469500.000000 | 55600.000000 |

In [32]:
```
college_house = pd.merge(college_town, house_qrt, on = ['State', 'RegionNam
e'])
college_house.head(2)
```

Out[32]:

| | State | RegionName | 2009q2 | 2008q3 | ratio |
|---|---|---|---|---|---|
| **0** | Alabama | Montevallo | 125200.000000 | 127266.666667 | 2066.666667 |
| **1** | Alabama | Tuscaloosa | 136933.333333 | 139600.000000 | 2666.666667 |

In [33]:
```
# 找出非学区房价钱，即总表排除学区房
non_college = pd.merge(house_qrt, college_house, how = 'outer', indicator =
True)
non_college.head(8)
```

Out[33]:

| | State | RegionName | 2009q2 | 2008q3 | ratio | _merge |
|---|---|---|---|---|---|---|
| **0** | New York | New York | 465833.333333 | 499766.666667 | 33933.333333 | left_only |
| **1** | California | Los Angeles | 413900.000000 | 469500.000000 | 55600.000000 | left_only |
| **2** | Illinois | Chicago | 219700.000000 | 232000.000000 | 12300.000000 | left_only |
| **3** | Pennsylvania | Philadelphia | 116166.666667 | 116933.333333 | 766.666667 | left_only |
| **4** | Arizona | Phoenix | 168233.333333 | 193766.666667 | 25533.333333 | left_only |
| **5** | Nevada | Las Vegas | 164333.333333 | 213366.666667 | 49033.333333 | both |
| **6** | California | San Diego | 389500.000000 | 424666.666667 | 35166.666667 | both |
| **7** | Texas | Dallas | 105100.000000 | 112166.666667 | 7066.666667 | both |

In [34]:
```
# 通过 merge 中 indicator 产生新列，把不是 'both' 的排除，即得出非学区房
non_college = non_college.loc[non_college._merge == 'left_only', :]
non_college.head(2)
```

Out[34]:

| | State | RegionName | 2009q2 | 2008q3 | ratio | _merge |
|---|---|---|---|---|---|---|
| **0** | New York | New York | 465833.333333 | 499766.666667 | 33933.333333 | left_only |
| **1** | California | Los Angeles | 413900.000000 | 469500.000000 | 55600.000000 | left_only |

In [35]:
```
t, p = ttest_ind(college_house['ratio'].dropna(), non_college['ratio'].drop
na())

different = True if p < 0.01 else False

better = "university town" if college_house['ratio'].mean() < non_college[
'ratio'].mean() else "non-university town"

print(different, p, better)
```

```
True 0.004325214853511201 university town
```