**PART1 TEST**

df = get\_list\_of\_university\_towns()

cols = ["State", "RegionName"]

print('Shape test: ', "Passed" if df.shape ==

(517, 2) else 'Failed')

print('Index test: '

"Passed" if df.index.tolist() == list(range(517))

else 'Failed')

print('Column test: ',

"Passed" if df.columns.tolist() == cols else 'Failed')

print('\\n test: ',

"Passed" if len(df[df[cols[0]].str.contains(

'\n') | df[cols[1]].str.contains(

'\n')].values) == 0

else 'Failed')

print('Trailing whitespace test:',

"Passed" if len(df[df[cols[0]].str.contains(

'\s+$') | df['RegionName'].str.contains(

'\s+$')].values) == 0

else 'Failed')

print('{"(","["} test:',

"Passed" if len(df[df['State'].str.contains(

'\(|\[') | df['RegionName'].str.contains(

'\(|\]')].values) == 0

else 'Failed')

# from troubleshooting learners code, using incorrect regex

# for RegionName usually generates these common mismatches

rgn\_rgx\_mstch = [

'Pomona', 'Mankato', 'Fulton', 'Sewanee']

rgn\_loc = (33, 218, 237, 442)

print ('RegionName regex test: ', "Passed" if all(df.loc[

rgn\_loc, 'RegionName'] == rgn\_rgx\_mstch)

else "Failed")

# when using split or find to extract regionName, these are

# common mismatches

rgn\_splt\_msmtch = [

'The Five College Region of Western Massachusetts:',

'Faribault, South Central College']

rgn\_loc = (184, 217)

print ('RegionName regex test: ', "Passed" if all(df.loc[

rgn\_loc, 'RegionName'] == rgn\_splt\_msmtch)

else "Failed")

# use the values of the states dictionary precoded to verify

# state names are as expected

states\_vlist = list(sorted(states.values()))

mismatchedStates = df[~df['State'].isin(

states\_vlist)].loc[:, 'State'].unique()

print ('State regex test: ', "Passed" if len(

mismatchedStates) == 0 else "Failed")

if len(mismatchedStates) > 0:

print()

print('The following states failed the equality test:')

print()

print('\n'.join(mismatchedStates))

#output should be

Shape test: Passed

Index test: Passed

Column test: Passed

\n test: Passed

Trailing whitespace test: Passed

{"(","["} test: Passed

RegionName regex test: Passed

RegionName split/find test: Passed

State test: Passed

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 character '\n'. '''

cols=["State", "RegionName"]

with open('university\_towns.txt')as file:

lines=file.readlines()

lines=[x.strip() for x in lines]

# Remove redudant of RegionName

lines2=[x.split('(')[0].strip() for x in lines]

res=pd.DataFrame(columns=cols)

count=0

for x in lines2:

if '[edit]' in x:

state=x.split('[edit]')[0].strip()

res.loc[count,'State']=state

else:

res.loc[count,'RegionName']=x

res.loc[count,'State']=state

count+=1

return res

#get\_list\_of\_university\_towns()

def get\_recession\_start():

'''Returns the year and quarter of the recession start time as a

string value in a format such as 2005q3'''

cols=['date','GDP of 2009 chained dollar']

df=pd.read\_excel(io='gdplev.xls',skiprows=219,parse\_cols="E,G",names=cols)

df.date=df.date.astype(str)

for index in range(len(df)-2):

if (df.loc[index+1,'GDP of 2009 chained dollar']<df.loc[index,'GDP of 2009 chained dollar'] and

df.loc[index+2,'GDP of 2009 chained dollar']<df.loc[index+1,'GDP of 2009 chained dollar']):

date\_start=df.loc[index,'date']

return date\_start

get\_recession\_start()

def get\_recession\_end():

'''Returns the year and quarter of the recession end time as a

string value in a format such as 2005q3'''

cols=['date','GDP of 2009 chained dollar']

df=pd.read\_excel(io='gdplev.xls',skiprows=219,parse\_cols="E,G",names=cols)

df.date=df.date.astype(str)

for index in range(len(df)-2):

if (df.loc[index+1,'GDP of 2009 chained dollar']>df.loc[index,'GDP of 2009 chained dollar'] and

df.loc[index+2,'GDP of 2009 chained dollar']>df.loc[index+1,'GDP of 2009 chained dollar']):

date\_end=df.loc[index,'date']

return date\_end

get\_recession\_end()

def get\_recession\_bottom():

'''Returns the year and quarter of the recession bottom time as a

string value in a format such as 2005q3'''

cols=['date','GDP of 2009 chained dollar']

df=pd.read\_excel(io='gdplev.xls',skiprows=219,parse\_cols="E,G",names=cols)

df.date=df.date.astype(str)

df['GDP of 2009 chained dollar']=df['GDP of 2009 chained dollar'].astype(float)

for index in range(len(df)-2):

if (df.loc[index+1,'GDP of 2009 chained dollar']<df.loc[index,'GDP of 2009 chained dollar'] and

df.loc[index+2,'GDP of 2009 chained dollar']<df.loc[index+1,'GDP of 2009 chained dollar']):

index\_start=index

if (df.loc[index+1,'GDP of 2009 chained dollar']>df.loc[index,'GDP of 2009 chained dollar'] and

df.loc[index+2,'GDP of 2009 chained dollar']>df.loc[index+1,'GDP of 2009 chained dollar']):

index\_end=index

min\_val=min(df.loc[index,'GDP of 2009 chained dollar'] for index in range(index\_start,index\_end))

ind=df[df['GDP of 2009 chained dollar']==min\_val].index.tolist()

index=ind[0]

date=df.loc[index,'date']

return date

get\_recession\_bottom()

df=pd.read\_csv('City\_Zhvi\_AllHomes.csv')

df

# Get the columns

cols=['State','RegionName']

for i in range(16):

for j in range(4):

year=str(2000+i)

quater=str(j+1)

cols.append(year+'q'+quater)

cols.append('2016q1')

cols.append('2016q2')

cols.append('2016q3')

res=pd.DataFrame(columns=cols)

res

sum\_q=0

sum\_d=0

for index in range(len(df.index)):

for i in range(17):

year=str(2000+i)

year\_q=[]

year\_q.append(year+'q1');year\_q.append(year+'q2');year\_q.append(year+'q3');year\_q.append(year+'q4')

for year in df.columns:

res.loc[index,'State']=df.loc[index,'State']

res.loc[index,'RegionName']=df.loc[index,'RegionName']

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')

# seperate time columns and convert their names to datetime

# 2000-2016

tdf = df[df.columns[51:]].rename(columns=pd.to\_datetime)

cols=tdf.columns

set\_cols=cols[(cols > '1999-12-31') & (cols < '2017')]

set\_cols

mdf=tdf[set\_cols].resample('Q',axis=1).mean().rename(

columns=lambda x: '{:}q{:}'.format(x.year, x.quarter))

mdf['State']=df['State']

mdf['RegionName']=df['RegionName']

for key in states.keys():

for index in range(len(mdf.index)):

if(mdf.loc[index,'State']==key):

mdf.loc[index,'State']=states[key]

mdf.set\_index(['State','RegionName'],inplace=True)

return mdf

convert\_housing\_data\_to\_quarters()

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 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(). 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()

ul = get\_list\_of\_university\_towns()

hdf['PriceRatio']=np.nan

rows=len(hdf.index)

cols=len(hdf.columns)

for i in range(rows):

start=0

end=0

for j in range(cols-2):

if(hdf.iloc[i,j]>hdf.iloc[i,j+1]>hdf.iloc[i,j+2]):

start=j+1

# print('i',i)

# print(hdf.iloc[i,j])

# print('start',start)

break

for j in range(start,cols-2):

if(hdf.iloc[i,j]<hdf.iloc[i,j+1]<hdf.iloc[i,j+2]):

end=j+2

# print('end',end)

break

if(start!=0 and end!=0):

min\_val=min(hdf.iloc[i,k] for k in range(start,end))

hdf.iloc[i,67]=hdf.iloc[i,start-1]/(min\_val)

uni\_list1=ul['State'].tolist()

uni\_list2=ul['RegionName'].tolist()

uni\_list\_tuple=list(zip(uni\_list1,uni\_list2))

uni\_list\_tuple

uni\_towns=hdf.loc[uni\_list\_tuple]

non\_uni\_towns=hdf[~hdf.index.isin(uni\_list\_tuple)]

static,p\_val=ttest\_ind(uni\_towns['PriceRatio'],non\_uni\_towns['PriceRatio'],nan\_policy='omit')

uni\_town\_mean=uni\_towns['PriceRatio'].mean(skipna=True)

non\_uni\_town\_mean=non\_uni\_towns['PriceRatio'].mean(skipna=True)

different=True if p\_val<0.01 else False

better="university town" if uni\_town\_mean< non\_uni\_town\_mean else "non-university town"

return (different,p\_val,better)

run\_ttest()

**LAST Q reference**

hdf=convert\_housing\_data\_to\_quarters()

start='2008q2'

hdf=(hdf.loc[:,[start,get\_recession\_bottom()]])

hdf['priceRatio']=hdf.loc[:,start]/hdf.loc[:,get\_recession\_bottom()]

hdf=hdf['priceRatio']

utdf=(get\_list\_of\_university\_towns())

s1=pd.merge(utdf,

hdf.reset\_index(),

how='inner',

left\_on=['State','RegionName'],

right\_on=['State','RegionName']).set\_index(['State','RegionName']).dropna()

s2=pd.merge(utdf,

hdf.reset\_index(),

how='right',

left\_on=['State','RegionName'],

right\_on=['State','RegionName']).set\_index(['State','RegionName']).dropna()

s2=s2.loc[(s2.index.difference(s1.index))]

print(s1.size)

print(s2.size)

**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 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(). 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()**

**ul = get\_list\_of\_university\_towns()**

**start=get\_recession\_start()**

**# Get the index of start of the recession**

**col=hdf.columns.get\_loc(start)**

**# Get the name of quarter before the start**

**start\_before=hdf.iloc[:,col-1].name**

**# Get the name of the bottom of the recession**

**bottom=get\_recession\_bottom()**

**# print(start\_before,bottom)**

**hdf['PriceRatio']=hdf[start\_before].div(hdf[bottom])**

**uni\_list1=ul['State'].tolist()**

**uni\_list2=ul['RegionName'].tolist()**

**uni\_list\_tuple=list(zip(uni\_list1,uni\_list2))**

**uni\_list\_tuple**

**uni\_towns=hdf.loc[uni\_list\_tuple]**

**non\_uni\_towns=hdf[~hdf.index.isin(uni\_list\_tuple)]**

**static,p\_val=ttest\_ind(uni\_towns['PriceRatio'],non\_uni\_towns['PriceRatio'],nan\_policy='omit')**

**uni\_town\_mean=uni\_towns['PriceRatio'].mean(skipna=True)**

**non\_uni\_town\_mean=non\_uni\_towns['PriceRatio'].mean(skipna=True)**

**different=True if p\_val<0.01 else False**

**better="university town" if uni\_town\_mean< non\_uni\_town\_mean else "non-university town"**

**return (different,p\_val,better)**

**run\_ttest()**