ChandonnetPythonProject

December 16, 2022

1 Python Project

1.0.1 Ray Chandonnet

1.0.2 12/16/2022

[3]:
```
import pandas as pd
import numpy as np
# from os import chdir, getcwd
# wd=getcwd
path = "/users/raychandonnet/Dropbox (Personal)/Merrimack College - MS in Data␣
 ↪Science/DSE5002/Python Project/"
cost_of_living=pd.read_csv(path + "cost_of_living.csv")
# Step 1 - Break the City up - requires a lot of cleaning because city and␣
 ↪country are in the same field separated by
# a comma, and US cities also have the state in form city, st, country.  I want␣
 ↪to break these up into separate columns
# to link to the country code data, then pull in job info for those city /␣
 ↪country combos
# First, split the city into three separate fields
cost_of_living[['City','State','Country']]=cost_of_living['City'].str.
 ↪split(pat=",",n=2,expand=True)
# This leaves the city state and country correct for US cities, but has the␣
 ↪country in the city column for
# non-US cities and "None" for country, so we have to fix that  It's two steps:
# First, repalce all the "None" in Country column with the State value which is␣
 ↪where the Country is
# Then eliminate the dups by putting "None" in the State field for alkl those. ␣
 ↪So we basically just swapped values
# to get the country in the right place
cost_of_living['Country']=cost_of_living['Country'].
 ↪fillna(cost_of_living['State']) # Puts the state value in country where NA
cost_of_living['State']=np.
 ↪where(cost_of_living['State']==cost_of_living['Country'],"-",cost_of_living['State'])
# It took me more time than I care to admit to realize that my states and␣
 ↪countries had padded whitespace and so weren't
```

```
# matching when I tried to merge in the country code data!!! Sigh....using the strip method fixes that
cost_of_living['Country']=cost_of_living['Country'].str.strip()
cost_of_living['State']=cost_of_living['State'].str.strip()
# This leaves us with City, State and Country, with country spelled out
# Now I can merge in the country codes from that file into my table.  This will let me connect to data in the salary
# and jobs files once I do some stuff with that data, if I need to use a code instead of the country spelled out
countrycodes=pd.read_excel(path+"country_codes.xlsx")
cost_of_living=pd.merge(left=cost_of_living,right=countrycodes,how='left',on='Country')
# Now read in the salaries file and do some data wrangling here as well to convert the time stamp to a date,
# and put the cities and countries in the right columns
salaries=pd.read_csv(path+"Levels_Fyi_Salary_Data.csv") # read data
salaries=salaries[salaries['title']=="Data Scientist"] # Extract all the Data Scientist jobs
salaries['timestamp'] = pd.to_datetime(salaries['timestamp']) # convert date
salaries=salaries.rename({"timestamp":"date"},axis=1) #rename date column
salaries[['City','State','Country']]=salaries['location'].str.split(pat=",",n=2,expand=True)
# Now fill in "United States" where country is blank, delete the state for non-US locations, and remove whitespace
salaries['Country']=salaries['Country'].fillna("United States") # Fills in 'United States' for all blank countries
salaries['State']=np.where(salaries['Country']!='United States',"-",salaries['State']) # delete erroneous states for non-US
salaries['Country']=salaries['Country'].str.strip()
salaries['State']=salaries['State'].str.strip()
# Now I'm going to enrich the data a bit to make it more granular and make more sense:
# First, I calculate "cashcomp" as cash compensation, = base _+ bonus, because when you are looking at affordability,
# stock grants don't count; They're a way to build wealth, not pay bills.  (Yuu could argue I should only count base
# salary since bonuses are discretionary and lumpy)
salaries['cashcomp']=salaries['basesalary']+salaries['bonus'] # calculate total cash comp
salaries=salaries[salaries['cashcomp']!=0]  # eliminate zeros where salary and bonus not listed
# Now, I create a new field standardized column called explevel which groups jobs based on years of experience,
# with people with 5 years or less labeled "Junior" and those with more than that labeled "Senior".
# This will add granularity to my salary indexes amd reduce the risk that jobs␣
 ↪are not evenly distributed
# across levels of experience for every city and country combination.
# I could make it more and more granular but there isn't enough data to really␣
 ↪dig much deeper and subset further than that
salaries['explevel']=np.
 ↪where(salaries['yearsofexperience']<=5,"Junior","Senior")
#
# Now comes the heavy lifting.  I'm going to aggregate by country and␣
 ↪experience level, and again by city, country
# and experience level, calculating mean salary for each subset.   Then, since␣
 ↪in the cost of living data, the indexes
# are calculated using NYC as baseline (100 index), I will calculate a "salary␣
 ↪index" for each country/city and
# experience combination, relative to that same experience level in NYC
sal_by_ctry=salaries.groupby(['Country','explevel'],as_index=False).agg(
    avg_salary=('cashcomp', np.mean)) # Calculate mean salaries by country and␣
 ↪job level
sal_by_city=salaries.
 ↪groupby(['Country','City','State','explevel'],as_index=False).agg(
    avg_salary=('cashcomp', np.mean)) # Calculate mean salaires by city,␣
 ↪country and level
NYCsal=salaries[salaries['City']=="New York"] # Extract all the NYC jobs
NYC_by_level=NYCsal.groupby('explevel').agg(
    NYC_avg_salary=('cashcomp', np.mean))  # Calculate mean salaries by level␣
 ↪for NYC
#
# Here comes the magic!  Now that I have average salaries for NYC by level, I␣
 ↪can use those to create a
# salary index, with NYC as the baseline at 100, for each city/country/level␣
 ↪combo I found.
# Once that cost of living index is calculated, I can derive an "affordability"␣
 ↪index
# dividing the salary index for that city/country/level into the cost  of␣
 ↪living index for that city/country
# So for example if the salary index is 80 (80% of NYC) but the cost of living␣
 ↪index is 40 (50% of NYC),
# then that location for that experience level is twice as affordable as NYC␣
 ↪(80/40=2x) or affordability = 200
# meaning your money goes twice as far
# Let's start with cities because they're easier because they map directly to␣
 ↪the cost_of_living data
afford_city_level=pd.
 ↪merge(left=sal_by_city,right=NYC_by_level,how='left',on='explevel') # Pull␣
 ↪in NYC salarary by level
```

```
afford_city_level['sal_index']=afford_city_level['avg_salary']/
 ↪afford_city_level['NYC_avg_salary']*100 # Calculate salary index
afford_city_level=pd.
 ↪merge(left=afford_city_level,right=cost_of_living,how='left',on=['City','State','Country'])
afford_city_level['affordability']=afford_city_level['sal_index']/
 ↪afford_city_level['Cost of Living Plus Rent Index']*100
```

[ ]:

```
```