Springboard Data Science Career Track Unit 4 Challenge - Tier 3 Complete

Objectives

Hey! Great job getting through those challenging DataCamp courses. You're learning a lot in a short span of time.

In this notebook, you're going to apply the skills you've been learning, bridging the gap between the controlled environment of DataCamp and the *slightly* messier work that data scientists do with actual datasets!

Here's the mystery we're going to solve: ***which boroughs of London have seen the greatest increase in housing prices, on average, over the last two decades?***

A borough is just a fancy word for district. You may be familiar with the five boroughs of New York… well, there are 32 boroughs within Greater London (here's some info for the curious). Some of them are more desirable areas to live in, and the data will reflect that with a greater rise in housing prices.

This is the Tier 3 notebook, which means it's not filled in at all: we'll just give you the skeleton of a project, the brief and the data. It's up to you to play around with it and see what you can find out! Good luck! If you struggle, feel free to look at easier tiers for help; but try to dip in and out of them, as the more independent work you do, the better it is for your learning!

This challenge will make use of only what you learned in the following DataCamp courses:

- Prework courses (Introduction to Python for Data Science, Intermediate Python for Data Science)
- Data Types for Data Science
- Python Data Science Toolbox (Part One)
- pandas Foundations
- Manipulating DataFrames with pandas
- Merging DataFrames with pandas

Of the tools, techniques and concepts in the above DataCamp courses, this challenge should require the application of the following:

- **pandas**
  - **data ingestion and inspection** (pandas Foundations, Module One)
  - **exploratory data analysis** (pandas Foundations, Module Two)
  - **tidying and cleaning** (Manipulating DataFrames with pandas, Module Three)
  - **transforming DataFrames** (Manipulating DataFrames with pandas, Module One)
  - **subsetting DataFrames with lists** (Manipulating DataFrames with pandas, Module One)
  - **filtering DataFrames** (Manipulating DataFrames with pandas, Module One)
  - **grouping data** (Manipulating DataFrames with pandas, Module Four)
  - **melting data** (Manipulating DataFrames with pandas, Module Three)
  - **advanced indexing** (Manipulating DataFrames with pandas, Module Four)
- **matplotlib** (Intermediate Python for Data Science, Module One)
- **fundamental data types** (Data Types for Data Science, Module One)
- **dictionaries** (Intermediate Python for Data Science, Module Two)
- **handling dates and times** (Data Types for Data Science, Module Four)
- **function definition** (Python Data Science Toolbox - Part One, Module One)
- **default arguments, variable length, and scope** (Python Data Science Toolbox - Part One, Module Two)
- **lambda functions and error handling** (Python Data Science Toolbox - Part One, Module Four)

The Data Science Pipeline

This is Tier Three, so we'll get you started. But after that, it's all in your hands! When you feel done with your investigations, look back over what you've accomplished, and prepare a quick presentation of your findings for the next mentor meeting.

Data Science is magical. In this case study, you'll get to apply some complex machine learning algorithms. But as David Spiegelhalter reminds us, there is no substitute for simply **taking a really, really good look at the data.** Sometimes, this is all we need to answer our question.

Data Science projects generally adhere to the four stages of Data Science Pipeline:

1. Sourcing and loading
2. Cleaning, transforming, and visualizing
3. Modeling
4. Evaluating and concluding

1. Sourcing and Loading

Any Data Science project kicks off by importing ***pandas***. The documentation of this wonderful library can be found here. As you've seen, pandas is conveniently connected to the Numpy and Matplotlib libraries.

Hint: This part of the data science pipeline will test those skills you acquired in the pandas Foundations course, Module One.

1.1. Importing Libraries

In [1]:
```
# Let's import the pandas, numpy libraries as pd, and np respectively.
import pandas as pd
import numpy as np

# Load the pyplot collection of functions from matplotlib, as plt
import matplotlib.pyplot as plt
```

1.2. Loading the data

Your data comes from the London Datastore: a free, open-source data-sharing portal for London-oriented datasets.

In [2]:
```
# First, make a variable called url_LondonHousePrices, and assign it the following link, enclosed in quotation-marks as a string:
# https://data.london.gov.uk/download/uk-house-price-index/70ac0766-8902-4eb5-aab5-01951aaed773/UK%20House%20price%20index.xls

url_LondonHousePrices= "https://data.london.gov.uk/download/uk-house-price-index/70ac0766-8902-4eb5-aab5-01951aaed773/UK%20House%20price%20index.xl

# The dataset we're interested in contains the Average prices of the houses, and is actually on a particular sheet of the Excel file.
# As a result, we need to specify the sheet name in the read_excel() method.
# Put this data into a variable called properties.
properties = pd.read_excel(url_LondonHousePrices, sheet_name='Average price', index_col= None)
```

2. Cleaning, transforming, and visualizing

This second stage is arguably the most important part of any Data Science project. The first thing to do is take a proper look at the data. Cleaning forms the majority of this stage, and can be done both before or after Transformation.

The end goal of data cleaning is to have tidy data. When data is tidy:

1. Each variable has a column.
2. Each observation forms a row.

Keep the end goal in mind as you move through this process, every step will take you closer.

Hint: This part of the data science pipeline should test those skills you acquired in:

- Intermediate Python for data science, all modules.
- pandas Foundations, all modules.
- Manipulating DataFrames with pandas, all modules.
- Data Types for Data Science, Module Four.
- Python Data Science Toolbox - Part One, all modules

2.1. Exploring your data

Think about your pandas functions for checking out a dataframe.

In [3]:
```
properties.shape
properties.head()
```

Out[3]:

	Unnamed: 0	City of London	Barking & Dagenham	Barnet	Bexley	Brent	Bromley	Camden	Croydon	Ealing	...	NORTH WEST	YORKS & THE HUMBER	EAST MIDLANDS	WEST MIDLANDS	EAST OF ENGLAND
0	NaT	E09000001	E09000002	E09000003	E09000004	E09000005	E09000006	E09000007	E09000008	E09000009	...	E12000002	E12000003	E12000004	E12000005	E12000006
1	1995-01-01	91449	50460.2	93284.5	64958.1	71306.6	81671.5	120933	69158.2	79885.9	...	43958.5	44803.4	45544.5	48527.5	56701.6
2	1995-02-01	82202.8	51085.8	93190.2	64787.9	72022.3	81657.6	119509	68951.1	80897.1	...	43925.4	44528.8	46051.6	49341.3	56593.6
3	1995-03-01	79120.7	51269	92247.5	64367.5	72015.8	81449.3	120282	68712.4	81379.9	...	44434.9	45200.5	45383.8	49442.2	56171.2
4	1995-04-01	77101.2	53133.5	90762.9	64277.7	72965.6	81124.4	120098	68610	82188.9	...	44267.8	45614.3	46124.2	49455.9	56567.9

5 rows × 49 columns

2.2. Cleaning the data

You might find you need to transpose your dataframe, check out what its row indexes are, and reset the index. You also might find you need to assign the values of the first row to your column headings . (Hint: recall the .columns feature of DataFrames, as well as the iloc[] method).

Don't be afraid to use StackOverflow for help with this.

In [5]:
```
properties_T=properties.T
properties_T.head()
properties_T.index
properties_T=properties_T.reset_index()
properties_T.index
properties_T.head()
properties_T.columns
properties_T.iloc[0]
properties_T.columns = properties_T.iloc[0]
properties_T.head()
properties_T=properties_T.drop(0)
properties_T.head()
```

Out[5]:

	Unnamed: 0	NaN	1995-01-01 00:00:00	1995-02-01 00:00:00	1995-03-01 00:00:00	1995-04-01 00:00:00	1995-05-01 00:00:00	1995-06-01 00:00:00	1995-07-01 00:00:00	1995-08-01 00:00:00	...	2020-04-01 00:00:00	2020-05-01 00:00:00	2020-06-01 00:00:00	2020-07-01 00:00:00	2020-08-01 00:00:00	2020-09-01 00:00:00	2020-10-01 00:00:00	2020-11-01 00:00:00
1	City of London	E09000001	91449	82202.8	79120.7	77101.2	84409.1	94900.5	110128	112329	...	920444	918209	882872	786627	827659	802639	841259	785325
2	Barking & Dagenham	E09000002	50460.2	51085.8	51269	53133.5	53042.2	53700.3	52113.1	52232.2	...	293603	293816	300526	304556	304924	302467	305283	307227
3	Barnet	E09000003	93284.5	93190.2	92247.5	90762.9	90258	90107.2	91441.2	92361.3	...	526689	526033	518175	523280	529660	535671	532217	533279
4	Bexley	E09000004	64958.1	64787.9	64367.5	64277.7	63997.1	64252.3	63722.7	64432.6	...	341553	339353	340893	344091	346680	344895	345812	349116
5	Brent	E09000005	71306.6	72022.3	72015.8	72965.6	73704	74310.5	74127	73547	...	470601	482808	484160	482303	497729	519982	524109	516904

5 rows × 315 columns

2.3. Cleaning the data (part 2)

You might we have to **rename** a couple columns. How do you do this? The clue's pretty bold...

In [6]:
```
properties_T = properties_T.rename(columns={'Unnamed: 0': 'London_Borough', pd.NaT:'ID'})
properties_T.head()
```

Out[6]:

	London_Borough	ID	1995-01-01 00:00:00	1995-02-01 00:00:00	1995-03-01 00:00:00	1995-04-01 00:00:00	1995-05-01 00:00:00	1995-06-01 00:00:00	1995-07-01 00:00:00	1995-08-01 00:00:00	...	2020-04-01 00:00:00	2020-05-01 00:00:00	2020-06-01 00:00:00	2020-07-01 00:00:00	2020-08-01 00:00:00	2020-09-01 00:00:00	2020-10-01 00:00:00	2 0
1	City of London	E09000001	91449	82202.8	79120.7	77101.2	84409.1	94900.5	110128	112329	...	920444	918209	882872	786627	827659	802639	841259	
2	Barking & Dagenham	E09000002	50460.2	51085.8	51269	53133.5	53042.2	53700.3	52113.1	52232.2	...	293603	293816	300526	304556	304924	302467	305283	
3	Barnet	E09000003	93284.5	93190.2	92247.5	90762.9	90258	90107.2	91441.2	92361.3	...	526689	526033	518175	523280	529660	535671	532217	
4	Bexley	E09000004	64958.1	64787.9	64367.5	64277.7	63997.1	64252.3	63722.7	64432.6	...	341553	339353	340893	344091	346680	344895	345812	
5	Brent	E09000005	71306.6	72022.3	72015.8	72965.6	73704	74310.5	74127	73547	...	470601	482808	484160	482303	497729	519982	524109	

5 rows × 315 columns

2.4.Transforming the data

Remember what Wes McKinney said about tidy data?

You might need to **melt** your DataFrame here.

In [7]:
```
properties_T=pd.melt(properties_T, id_vars= ['London_Borough','ID'])
properties_T.head()
properties_T=properties_T.rename(columns = {0:'Month','value':'Avg_Price'})
properties_T.head()
```

Out[7]:

	London_Borough	ID	Month	Avg_Price
0	City of London	E09000001	1995-01-01	91449
1	Barking & Dagenham	E09000002	1995-01-01	50460.2
2	Barnet	E09000003	1995-01-01	93284.5
3	Bexley	E09000004	1995-01-01	64958.1
4	Brent	E09000005	1995-01-01	71306.6

Remember to make sure your column data types are all correct. Average prices, for example, should be floating point numbers...

In [8]:
```
properties_T.dtypes
properties_T['Avg_Price']=pd.to_numeric(properties_T['Avg_Price'])
properties_T.dtypes
```

Out[8]:
```
London_Borough            object
ID                        object
Month             datetime64[ns]
Avg_Price                float64
dtype: object
```

2.5. Cleaning the data (part 3)

Do we have an equal number of observations in the ID, Average Price, Month, and London Borough columns? Remember that there are only 32 London Boroughs. How many entries do you have in that column?

Check out the contents of the London Borough column, and if you find null values, get rid of them however you see fit.

In [9]:
```
properties_T.count()
properties_T['London_Borough'].unique()
properties_T[properties_T['London_Borough']=='Unnamed: 37'].head()
properties_T[properties_T['London_Borough']=='Unnamed: 47'].head()
NaNFreeDF = properties_T.dropna()
NaNFreeDF.count()
NaNFreeDF.head()
NaNFreeDF['London_Borough'].unique()
nonBoroughs=['Inner London','Outer London','NORTH EAST','NORTH WEST','YORKS & THE HUMBER','EAST MIDLANDS','WEST MIDLANDS',
        'EAST OF ENGLAND','LONDON','SOUTH EAST','SOUTH WEST','England']
NaNFreeDF[NaNFreeDF.London_Borough.isin(nonBoroughs)]
NaNFreeDF[~NaNFreeDF.London_Borough.isin(nonBoroughs)]
NaNFreeDF = NaNFreeDF[~NaNFreeDF.London_Borough.isin(nonBoroughs)]
df=NaNFreeDF
df.head()
df.dtypes
```

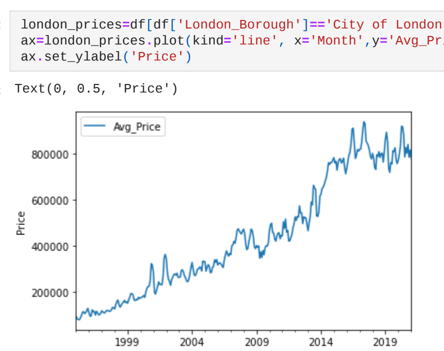
Out[9]:
```
London_Borough            object
ID                        object
Month             datetime64[ns]
Avg_Price                float64
dtype: object
```

2.6. Visualizing the data

To visualize the data, why not subset on a particular London Borough? Maybe do a line plot of Month against Average Price?

In [10]:
```
london_prices=df[df['London_Borough']=='City of London']
ax=london_prices.plot(kind='line', x='Month',y='Avg_Price')
ax.set_ylabel('Price')
```

Out[10]:
```
Text(0, 0.5, 'Price')
```



To limit the number of data points you have, you might want to extract the year from every month value your *Month* column.

To this end, you *could* apply a ***lambda function***. Your logic could work as follows:

1. look through the `Month` column
2. extract the year from each individual value in that column
3. store that corresponding year as separate column.

Whether you go ahead with this is up to you. Just so long as you answer our initial brief: which boroughs of London have seen the greatest house price increase, on average, over the past two decades?

In [11]:
```
df['Year']=df['Month'].apply(lambda t: t.year)
df.tail()
dfg=df.groupby(by=['London_Borough','Year']).mean()
dfg.sample(10)
dfg=dfg.reset_index()
dfg.head()
```

Out[11]:

	London_Borough	Year	Avg_Price
0	Barking & Dagenham	1995	51817.969390
1	Barking & Dagenham	1996	51718.192690
2	Barking & Dagenham	1997	55974.262309
3	Barking & Dagenham	1998	60285.821083
4	Barking & Dagenham	1999	65320.934441

3. Modeling

Consider creating a function that will calculate a ratio of house prices, comparing the price of a house in 2018 to the price in 1998.

Consider calling this function create_price_ratio.

You'd want this function to:

1. Take a filter of dfg, specifically where this filter constrains the London_Borough, as an argument. For example, one admissible argument should be: dfg[dfg['London_Borough']=='Camden'].
2. Get the Average Price for that Borough, for the years 1998 and 2018.
3. Calculate the ratio of the Average Price for 1998 divided by the Average Price for 2018.
4. Return that ratio.

Once you've written this function, you ultimately want to use it to iterate through all the unique London_Boroughs and work out the ratio capturing the difference of house prices between 1998 and 2018.

Bear in mind: you don't have to write a function like this if you don't want to. If you can solve the brief otherwise, then great!

Hint: This section should test the skills you acquired in:

- Python Data Science Toolbox - Part One, all modules

In [32]:
```
def create_price_ratio(d):
    y1998 = float(d['Avg_Price'][d['Year']==1998])
    y2018 = float(d['Avg_Price'][d['Year']==2018])
    ratio = [y1998/y2018]
    return ratio
c=dfg[dfg['London_Borough']=='Camden']
create_price_ratio(c)
final = {}
for b in dfg['London_Borough'].unique():
    borough = dfg[dfg['London_Borough'] == b]
    final[b] = create_price_ratio(borough)

print(final)

df_ratios_Boroughs = pd.DataFrame(final)
df_ratios_Boroughs.head()

df_ratios_Boroughs_T=df_ratios_Boroughs.T
df_ratios_Boroughs=df_ratios_Boroughs_T.reset_index()
df_ratios_Boroughs.head()

df_ratios_Boroughs=df_ratios_Boroughs.rename(columns = {'index':'Boroughs',0:'2018'})
df_ratios_Boroughs.head(33)
```

```
{'Barking & Dagenham': [0.20422256235393685], 'Barnet': [0.22945274120785797], 'Bexley': [0.2353507654063011], 'Brent': [0.2043086864360114], 'Brom
ley': [0.24421308489837312], 'Camden': [0.20261973503252542], 'City of London': [0.18862157770244367], 'Croydon': [0.23803288028014047], 'Ealing':
[0.23194048191708755], 'Enfield': [0.23455064269011863], 'Greenwich': [0.20995010893854218], 'Hackney': [0.16133493530705734], 'Hammersmith & Fulha
m': [0.24167443054605853], 'Haringey': [0.19475619095546956], 'Harrow': [0.24635417785626296], 'Havering': [0.23120155787014757], 'Hillingdon': [0.
23807975835429931], 'Hounslow': [0.25148317824115635], 'Islington': [0.20643891170300285], 'Kensington & Chelsea': [0.19675491852791563], 'Kingston
upon Thames': [0.23416190234282552], 'Lambeth': [0.20170435486140822], 'Lewisham': [0.18351246764721711], 'Merton': [0.21091380604361798], 'Newham':
[0.18848754146121072], 'Redbridge': [0.2236545053715767], 'Richmond upon Thames': [0.24967779731157863], 'Southwark': [0.18127484171283462], 'Sutto
n': [0.24280551426824518], 'Tower Hamlets': [0.2161367227623553], 'Waltham Forest': [0.1713867782439487], 'Wandsworth': [0.2101851809159322], 'West
minster': [0.18679140473024677]}
```

Out[32]:

	Boroughs	2018
0	Barking & Dagenham	0.204223
1	Barnet	0.229453
2	Bexley	0.235351
3	Brent	0.204309
4	Bromley	0.244213
5	Camden	0.202620
6	City of London	0.188622
7	Croydon	0.238033
8	Ealing	0.231940
9	Enfield	0.234551
10	Greenwich	0.209950
11	Hackney	0.161335
12	Hammersmith & Fulham	0.241674
13	Haringey	0.194756
14	Harrow	0.246354
15	Havering	0.231202
16	Hillingdon	0.238080
17	Hounslow	0.251483
18	Islington	0.206439
19	Kensington & Chelsea	0.196755
20	Kingston upon Thames	0.234162
21	Lambeth	0.201704
22	Lewisham	0.183512
23	Merton	0.210914
24	Newham	0.188488
25	Redbridge	0.223655
26	Richmond upon Thames	0.249678
27	Southwark	0.181275
28	Sutton	0.242806
29	Tower Hamlets	0.216137
30	Waltham Forest	0.171387
31	Wandsworth	0.210185
32	Westminster	0.186791

4. Conclusion

What can you conclude? Type out your conclusion below.

Look back at your notebook. Think about how you might summarize what you have done, and prepare a quick presentation on it to your mentor at your next meeting.

We hope you enjoyed this practical project. It should have consolidated your data hygiene and pandas skills by looking at a real-world problem involving just the kind of dataset you might encounter as a budding data scientist. Congratulations, and looking forward to seeing you at the next step in the course!

In [ ]:

In [ ]:

In [ ]: