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.


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

In [2]:
```
import pandas as pd
import numpy as np
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 [3]:
```
# First, make a variable called url_LondonHousePrices, and assign it the following link, enclosed in quotation-marks as
# https://data.london.gov.uk/download/uk-house-price-index/70ac0766-8902-4eb5-aab5-01951aaed773/UK%20House%20price%20ind

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

# The dataset we're interested in contains the Average prices of the houses, and is actually on a particular sheet of th
# 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)
```

import pandas as pd

properties =pd.read_excel(https://data.london.gov.uk/download/uk-house-price-index/70ac0766-8902-4eb5-aab5-01951aaed773/UK%20House%20price%20index.xls, sheet_name='Average price', index_col= None) properties.head()

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 [4]:
```
properties.shape
properties.head()
```

Out[4]:

	Unnamed: 0	City of London	Barking & Dagenham	Barnet	Bexley	Brent	Bromley	Camden	Croydon	Ealing	...	N
0	NaT	E09000001	E09000002	E09000003	E09000004	E09000005	E09000006	E09000007	E09000008	E09000009	...	E12
1	1995-01-01	91448.98487	50460.2266	93284.51832	64958.09036	71306.56698	81671.47692	120932.8881	69158.16225	79885.89069	...	43958
2	1995-02-01	82202.77314	51085.77983	93190.16963	64787.92069	72022.26197	81657.55944	119508.8622	68951.09542	80897.06551	...	43925
3	1995-03-01	79120.70256	51268.96956	92247.52435	64367.49344	72015.76274	81449.31143	120282.2131	68712.44341	81379.86288	...	4443
4	1995-04-01	77101.20804	53133.50526	90762.87492	64277.66881	72965.63094	81124.41227	120097.899	68610.04641	82188.90498	...	4426

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 = properties.transpose()
properties = properties.reset_index()
properties.columns = properties.iloc[0]
properties = properties.drop(0)
properties
```

Out[5]:

	Unnamed: 0	NaT	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	...	00
1	City of London	E09000001	91448.98487	82202.77314	79120.70256	77101.20804	84409.14932	94900.51244	110128.0423	112329.4376	...	9
2	Barking & Dagenham	E09000002	50460.2266	51085.77983	51268.96956	53133.50526	53042.24852	53700.34831	52113.12157	52232.19868	...	3
3	Barnet	E09000003	93284.51832	93190.16963	92247.52435	90762.87492	90258.00033	90107.23471	91441.24768	92361.31512	...	5
4	Bexley	E09000004	64958.09036	64787.92069	64367.49344	64277.66881	63997.13588	64252.32335	63722.70055	64432.60005	...	4
5	Brent	E09000005	71306.56698	72022.26197	72015.76274	72965.63094	73704.04743	74310.48167	74127.03788	73547.0411	...	5
6	Bromley	E09000006	81671.47692	81657.55944	81449.31143	81124.41227	81542.61561	82382.83435	82898.52264	82054.37156	...	5
7	Camden	E09000007	120932.8881	119508.8622	120282.2131	120097.899	119929.2782	121887.4625	124027.5768	125529.8039	...	8
8	Croydon	E09000008	69158.16225	68951.09542	68712.44341	68610.04641	68844.9169	69052.51103	69142.48112	68993.42545	...	4
9	Ealing	E09000009	79885.89069	80897.06551	81379.86288	82188.90498	82077.05525	81630.66181	82352.2226	82706.65927	...	5
10	Enfield	E09000010	72514.69096	73155.19746	72190.44144	71442.92235	70630.77955	71348.31147	71837.54011	72237.94562	...	4
11	Greenwich	E09000011	62300.10169	60993.26863	61377.83464	61927.7246	63512.99103	64751.56404	65486.34112	65076.43195	...	4
12	Hackney	E09000012	61296.52637	63187.08332	63593.29935	65139.64403	66193.99212	66921.17101	68390.753	68096.79385	...	6
13	Hammersmith & Fulham	E09000013	124902.8602	122087.718	120635.9467	121424.6241	124433.539	126175.1513	124381.5134	123625.3196	...	7
14	Haringey	E09000014	76287.56947	78901.21036	78521.94855	79545.57477	79374.0349	79956.3621	80746.34881	81217.69074	...	6
15	Harrow	E09000015	84769.52599	83396.10525	83416.23759	83567.88439	83853.65615	84173.24689	84226.69844	84430.61796	...	5
16	Havering	E09000016	68000.13774	69393.51294	69368.02407	69444.26215	68534.52248	68464.60664	68680.83996	69023.36482	...	4
17	Hillingdon	E09000017	73834.82964	75031.0696	74188.66949	73911.40591	73117.12416	74005.00585	74671.13263	74967.86534	...	4
18	Hounslow	E09000018	72231.70537	71051.55852	72097.99411	71890.28339	72877.47219	72331.08116	73717.78844	74479.94802	...	4
19	Islington	E09000019	92516.48557	94342.37334	93465.86407	93344.49305	94346.39917	97428.94311	98976.14077	98951.20791	...	7
20	Kensington & Chelsea	E09000020	182694.8326	182345.2463	182878.8231	184176.9168	191474.1141	197265.7602	197963.3169	198037.4218	...	13
21	Kingston upon Thames	E09000021	80875.84843	81230.13524	81111.48848	81672.80476	82123.51084	82205.66822	82525.793	83342.84552	...	5

	Unnamed: 0	NaT	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	...	00
22	Lambeth	E09000022	67770.98843	65381.51908	66336.51868	66388.7716	69035.11076	68881.15764	69608.72242	68840.02827	...	5
23	Lewisham	E09000023	60491.26109	60869.27091	60288.03002	59471.03136	58551.38387	58041.43543	58126.37811	58151.3154	...	4
24	Merton	E09000024	82070.6133	79982.74872	80661.68279	79990.54333	80873.98643	80704.92667	81055.90335	80781.09186	...	5
25	Newham	E09000025	53539.31919	53153.88306	53458.26393	54479.75395	55803.95958	56067.76986	55458.31693	54709.35467	...	4
26	Redbridge	E09000026	72189.58437	72141.6261	72501.35502	72228.60295	72366.64122	72279.4325	72880.83974	73275.16891	...	4
27	Richmond upon Thames	E09000027	109326.1245	111103.0394	107325.4742	106875	107707.6799	112865.0542	114656.6011	112320.4096	...	7
28	Southwark	E09000028	67885.20344	64799.0648	65763.29719	63073.62117	64420.49933	64155.81449	67024.74767	65525.94434	...	5
29	Sutton	E09000029	71536.97357	70893.20851	70306.83844	69411.9439	69759.21989	70125.24728	70789.57284	69958.41918	...	4
30	Tower Hamlets	E09000030	59865.18995	62318.53353	63938.67686	66233.19383	66432.85846	66232.16372	64692.22672	63472.27558	...	4
31	Waltham Forest	E09000031	61319.44913	60252.12246	60871.08493	60971.39722	61494.16938	61547.79643	61933.52738	61916.4222	...	5
32	Wandsworth	E09000032	88559.04381	88641.01678	87124.81523	87026.00225	86518.05945	88114.3351	89830.58934	90560.68078	...	6
33	Westminster	E09000033	133025.2772	131468.3096	132260.3417	133370.2036	133911.1117	134562.1941	133450.2162	136581.5082	...	10
34	Unnamed: 34	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	
35	Inner London	E13000001	78251.9765	75885.70201	76591.59947	76851.56697	79129.19443	79969.1525	80550.47935	80597.64563	...	6
36	Outer London	E13000002	72958.79836	72937.88262	72714.53478	72591.92469	72752.99414	73189.39978	73665.90517	73691.12888	...	4
37	Unnamed: 37	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	
38	NORTH EAST	E12000001	42076.35411	42571.98949	42369.72984	42095.8436	43266.45165	42315.34372	43287.74323	41899.05494	...	1
39	NORTH WEST	E12000002	43958.48001	43925.42289	44434.8681	44267.7796	44223.61973	44112.96432	44109.58764	44193.66583	...	2
40	YORKS & THE HUMBER	E12000003	44803.42878	44528.80721	45200.46775	45614.34341	44830.98563	45392.63981	45534.99864	45111.45939	...	2
41	EAST MIDLANDS	E12000004	45544.52227	46051.57066	45383.82395	46124.23045	45878.00396	45679.99539	46037.67312	45922.53585	...	2
42	WEST	E12000005	48527.52339	49341.29029	49442.17973	49455.93299	50369.66188	50100.43023	49860.00809	49598.45969	...	2

	Unnamed: 0	NaT	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	...	00
	MIDLANDS											
43	EAST OF ENGLAND	E12000006	56701.5961	56593.59475	56171.18278	56567.89582	56479.80183	56288.94557	57242.30186	56732.40547	...	3
44	LONDON	E12000007	74435.76052	72777.93709	73896.84204	74455.28754	75432.02786	75606.24501	75984.24079	75529.34488	...	5
45	SOUTH EAST	E12000008	64018.87894	63715.02399	64113.60858	64623.22395	64530.36358	65511.008	65224.88465	64851.60429	...	3
46	SOUTH WEST	E12000009	54705.1579	54356.14843	53583.07667	54786.01938	54698.83831	54420.15939	54265.86368	54365.71495	...	3
47	Unnamed: 47	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	
48	England	E92000001	53202.77128	53096.1549	53201.2843	53590.8548	53678.24041	53735.15475	53900.60633	53600.31975	...	3

48 rows × 347 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 = properties.rename(columns = {'Unnamed: 0':'London_Borough', pd.NaT: 'ID'})
properties
```

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	...
1	City of London	E09000001	91448.98487	82202.77314	79120.70256	77101.20804	84409.14932	94900.51244	110128.0423	112329.4376	...
2	Barking & Dagenham	E09000002	50460.2266	51085.77983	51268.96956	53133.50526	53042.24852	53700.34831	52113.12157	52232.19868	...
3	Barnet	E09000003	93284.51832	93190.16963	92247.52435	90762.87492	90258.00033	90107.23471	91441.24768	92361.31512	...
4	Bexley	E09000004	64958.09036	64787.92069	64367.49344	64277.66881	63997.13588	64252.32335	63722.70055	64432.60005	...
5	Brent	E09000005	71306.56698	72022.26197	72015.76274	72965.63094	73704.04743	74310.48167	74127.03788	73547.0411	...
6	Bromley	E09000006	81671.47692	81657.55944	81449.31143	81124.41227	81542.61561	82382.83435	82898.52264	82054.37156	...
7	Camden	E09000007	120932.8881	119508.8622	120282.2131	120097.899	119929.2782	121887.4625	124027.5768	125529.8039	...
8	Croydon	E09000008	69158.16225	68951.09542	68712.44341	68610.04641	68844.9169	69052.51103	69142.48112	68993.42545	...
9	Ealing	E09000009	79885.89069	80897.06551	81379.86288	82188.90498	82077.05525	81630.66181	82352.2226	82706.65927	...
10	Enfield	E09000010	72514.69096	73155.19746	72190.44144	71442.92235	70630.77955	71348.31147	71837.54011	72237.94562	...
11	Greenwich	E09000011	62300.10169	60993.26863	61377.83464	61927.7246	63512.99103	64751.56404	65486.34112	65076.43195	...
12	Hackney	E09000012	61296.52637	63187.08332	63593.29935	65139.64403	66193.99212	66921.17101	68390.753	68096.79385	...
13	Hammersmith & Fulham	E09000013	124902.8602	122087.718	120635.9467	121424.6241	124433.539	126175.1513	124381.5134	123625.3196	...
14	Haringey	E09000014	76287.56947	78901.21036	78521.94855	79545.57477	79374.0349	79956.3621	80746.34881	81217.69074	...
15	Harrow	E09000015	84769.52599	83396.10525	83416.23759	83567.88439	83853.65615	84173.24689	84226.69844	84430.61796	...
16	Havering	E09000016	68000.13774	69393.51294	69368.02407	69444.26215	68534.52248	68464.60664	68680.83996	69023.36482	...
17	Hillingdon	E09000017	73834.82964	75031.0696	74188.66949	73911.40591	73117.12416	74005.00585	74671.13263	74967.86534	...
18	Hounslow	E09000018	72231.70537	71051.55852	72097.99411	71890.28339	72877.47219	72331.08116	73717.78844	74479.94802	...
19	Islington	E09000019	92516.48557	94342.37334	93465.86407	93344.49305	94346.39917	97428.94311	98976.14077	98951.20791	...
20	Kensington & Chelsea	E09000020	182694.8326	182345.2463	182878.8231	184176.9168	191474.1141	197265.7602	197963.3169	198037.4218	...
21	Kingston upon Thames	E09000021	80875.84843	81230.13524	81111.48848	81672.80476	82123.51084	82205.66822	82525.793	83342.84552	...
22	Lambeth	E09000022	67770.98843	65381.51908	66336.51868	66388.7716	69035.11076	68881.15764	69608.72242	68840.02827	...

	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	...
23	Lewisham	E09000023	60491.26109	60869.27091	60288.03002	59471.03136	58551.38387	58041.43543	58126.37811	58151.3154	...
24	Merton	E09000024	82070.6133	79982.74872	80661.68279	79990.54333	80873.98643	80704.92667	81055.90335	80781.09186	...
25	Newham	E09000025	53539.31919	53153.88306	53458.26393	54479.75395	55803.95958	56067.76986	55458.31693	54709.35467	...
26	Redbridge	E09000026	72189.58437	72141.6261	72501.35502	72228.60295	72366.64122	72279.4325	72880.83974	73275.16891	...
27	Richmond upon Thames	E09000027	109326.1245	111103.0394	107325.4742	106875	107707.6799	112865.0542	114656.6011	112320.4096	...
28	Southwark	E09000028	67885.20344	64799.0648	65763.29719	63073.62117	64420.49933	64155.81449	67024.74767	65525.94434	...
29	Sutton	E09000029	71536.97357	70893.20851	70306.83844	69411.9439	69759.21989	70125.24728	70789.57284	69958.41918	...
30	Tower Hamlets	E09000030	59865.18995	62318.53353	63938.67686	66233.19383	66432.85846	66232.16372	64692.22672	63472.27558	...
31	Waltham Forest	E09000031	61319.44913	60252.12246	60871.08493	60971.39722	61494.16938	61547.79643	61933.52738	61916.4222	...
32	Wandsworth	E09000032	88559.04381	88641.01678	87124.81523	87026.00225	86518.05945	88114.3351	89830.58934	90560.68078	...
33	Westminster	E09000033	133025.2772	131468.3096	132260.3417	133370.2036	133911.1117	134562.1941	133450.2162	136581.5082	...
34	Unnamed: 34	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...
35	Inner London	E13000001	78251.9765	75885.70201	76591.59947	76851.56697	79129.19443	79969.1525	80550.47935	80597.64563	...
36	Outer London	E13000002	72958.79836	72937.88262	72714.53478	72591.92469	72752.99414	73189.39978	73665.90517	73691.12888	...
37	Unnamed: 37	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...
38	NORTH EAST	E12000001	42076.35411	42571.98949	42369.72984	42095.8436	43266.45165	42315.34372	43287.74323	41899.05494	...
39	NORTH WEST	E12000002	43958.48001	43925.42289	44434.8681	44267.7796	44223.61973	44112.96432	44109.58764	44193.66583	...
40	YORKS & THE HUMBER	E12000003	44803.42878	44528.80721	45200.46775	45614.34341	44830.98563	45392.63981	45534.99864	45111.45939	...
41	EAST MIDLANDS	E12000004	45544.52227	46051.57066	45383.82395	46124.23045	45878.00396	45679.99539	46037.67312	45922.53585	...
42	WEST MIDLANDS	E12000005	48527.52339	49341.29029	49442.17973	49455.93299	50369.66188	50100.43023	49860.00809	49598.45969	...
43	EAST OF ENGLAND	E12000006	56701.5961	56593.59475	56171.18278	56567.89582	56479.80183	56288.94557	57242.30186	56732.40547	...
44	LONDON	E12000007	74435.76052	72777.93709	73896.84204	74455.28754	75432.02786	75606.24501	75984.24079	75529.34488	...

	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	...
45	SOUTH EAST	E12000008	64018.87894	63715.02399	64113.60858	64623.22395	64530.36358	65511.008	65224.88465	64851.60429	...
46	SOUTH WEST	E12000009	54705.1579	54356.14843	53583.07667	54786.01938	54698.83831	54420.15939	54265.86368	54365.71495	...
47	Unnamed: 47	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...
48	England	E92000001	53202.77128	53096.1549	53201.2843	53590.8548	53678.24041	53735.15475	53900.60633	53600.31975	...

48 rows × 347 columns

2.4.Transforming the data

Remember what Wes McKinney said about tidy data?

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

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

Out[7]:

	London_Borough	ID	Month	Average_price
0	City of London	E09000001	1995-01-01	91448.98487
1	Barking & Dagenham	E09000002	1995-01-01	50460.2266
2	Barnet	E09000003	1995-01-01	93284.51832
3	Bexley	E09000004	1995-01-01	64958.09036
4	Brent	E09000005	1995-01-01	71306.56698

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

In [8]:
```
melt_properties['Average_price'] = pd.to_numeric(melt_properties['Average_price'])
melt_properties.dtypes
melt_properties.count()
```

Out[8]:
```
London_Borough    16560
ID                15525
Month             16560
Average_price     15525
dtype: int64
```

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.

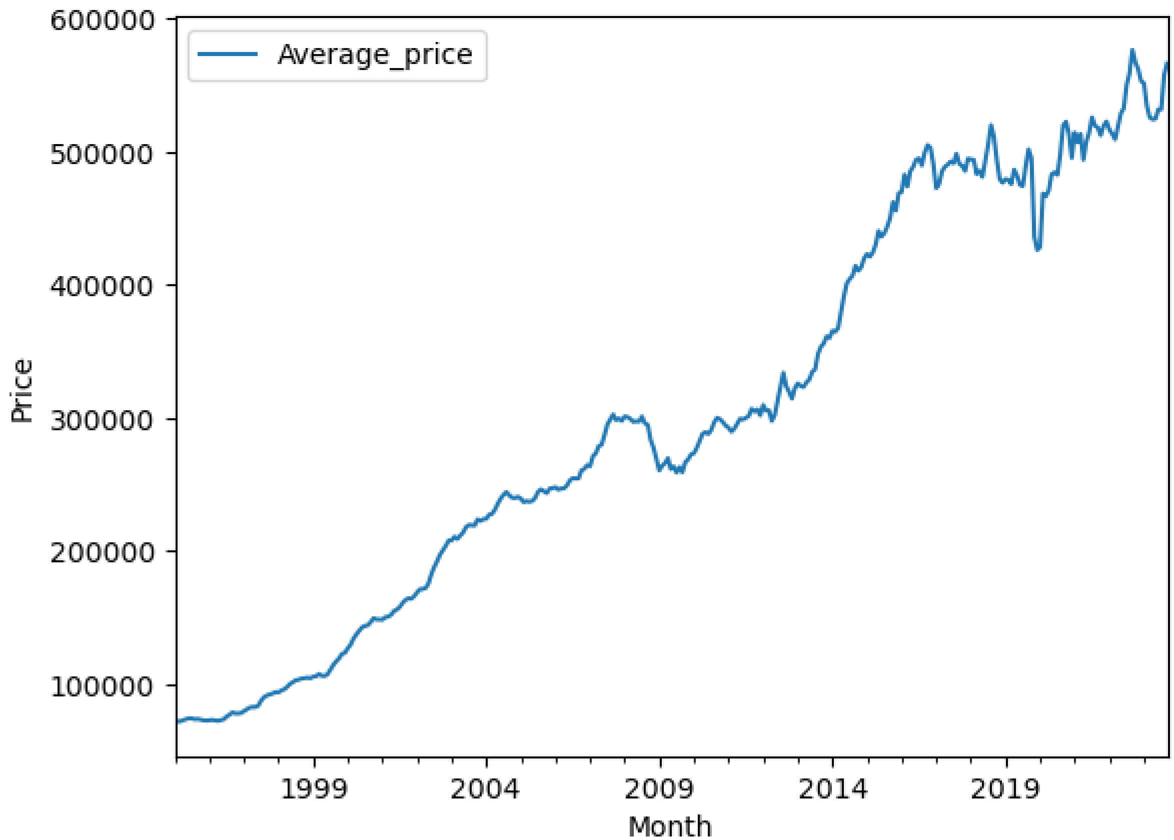
In [9]:
```
melt_properties['London_Borough'].unique()
melt_properties1 = melt_properties[melt_properties['Average_price'].notna()]
melt_properties2 = melt_properties.dropna()
melt_properties2['London_Borough'].unique()
nonBoroughs = ['Inner London',
       'Outer London', 'NORTH EAST', 'NORTH WEST', 'YORKS & THE HUMBER',
       'EAST MIDLANDS', 'WEST MIDLANDS', 'EAST OF ENGLAND', 'LONDON',
       'SOUTH EAST', 'SOUTH WEST', 'England']
melt_properties2 = melt_properties2[~melt_properties2.London_Borough.isin(nonBoroughs)]
df= melt_properties2
df.head()
df.dtypes
```

Out[9]:
```
London_Borough            object
ID                        object
Month             datetime64[ns]
Average_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]:
```
brent_prices = df[df['London_Borough'] == 'Brent']
x = brent_prices.plot(kind='line', y= 'Average_price', x= 'Month')
x.set_ylabel('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 dt: dt.year)
df.tail()
```

Out[11]:

	London_Borough	ID	Month	Average_price	Year
16540	Sutton	E09000029	2023-09-01	437958.0	2023
16541	Tower Hamlets	E09000030	2023-09-01	509454.0	2023
16542	Waltham Forest	E09000031	2023-09-01	510471.0	2023
16543	Wandsworth	E09000032	2023-09-01	637929.0	2023
16544	Westminster	E09000033	2023-09-01	967277.0	2023

In [12]:

```
dfg = df.groupby(by=['London_Borough', 'Year']).mean(numeric_only = True)
dfg.sample(10)
```

Out[12]:

		Average_price
London_Borough	**Year**	
Bexley	**2011**	2.006723e+05
Richmond upon Thames	**2020**	6.809452e+05
Enfield	**2008**	2.458393e+05
Hillingdon	**2018**	4.102661e+05
Barking & Dagenham	**2004**	1.581760e+05
Lewisham	**1997**	6.615070e+04
Bromley	**2012**	2.820250e+05
Hackney	**2007**	2.903967e+05
Kensington & Chelsea	**2023**	1.332891e+06
Haringey	**2015**	4.806361e+05

In [13]:

```
dfg = dfg.reset_index()
dfg.head()
```

Out[13]:

	London_Borough	Year	Average_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 [14]:
```
def create_price_ratio(d):
    y1998 = float(d['Average_price'][d['Year']==1998])
    y2018 = float(d['Average_price'][d['Year']==2018])
    ratio = [y2018/y1998]
    return ratio
create_price_ratio(dfg[dfg['London_Borough'] == 'Barking & Dagenham']);
```

```
final = {}
for b in dfg['London_Borough'].unique():
    borough = dfg[dfg['London_Borough']== b]
    final[b] = create_price_ratio(borough)
print(final)
```

```
{'Barking & Dagenham': [4.89661861291754], 'Barnet': [4.358195917538044], 'Bexley': [4.248977046127877], 'Brent': [4.89
45544971392865], 'Bromley': [4.094784685333876], 'Camden': [4.935353408884261], 'City of London': [5.30162037758761],
'Croydon': [4.201100280024766], 'Ealing': [4.311450902121834], 'Enfield': [4.263471583495811], 'Greenwich': [4.76303634
73291935], 'Hackney': [6.198285561008663], 'Hammersmith & Fulham': [4.13779810193623], 'Haringey': [5.134624964136042],
'Harrow': [4.0591964329643195], 'Havering': [4.325230371335307], 'Hillingdon': [4.2002730803844575], 'Hounslow': [3.976
409106143329], 'Islington': [4.844048012802297], 'Kensington & Chelsea': [5.082465066092464], 'Kingston upon Thames':
[4.270549521484271], 'Lambeth': [4.957751163514062], 'Lewisham': [5.449221041059686], 'Merton': [4.741273313294603], 'N
ewham': [5.305390437201879], 'Redbridge': [4.471182006097364], 'Richmond upon Thames': [4.005161895721457], 'Southwar
k': [5.516485302379378], 'Sutton': [4.118522608573157], 'Tower Hamlets': [4.62670104006116], 'Waltham Forest': [5.83475
580932281], 'Wandsworth': [4.75770934773927], 'Westminster': [5.353565392605412]}
```

In [15]:

```
df_ratios = pd.DataFrame(final)
df_ratios.head()
```

Out[15]:

	Barking & Dagenham	Barnet	Bexley	Brent	Bromley	Camden	City of London	Croydon	Ealing	Enfield	...	Merton	Newham	Redbridge	Ri
0	4.896619	4.358196	4.248977	4.894554	4.094785	4.935353	5.30162	4.2011	4.311451	4.263472	...	4.741273	5.30539	4.471182	4

1 rows × 33 columns

In [16]:

```
df_ratios_T = df_ratios.T
df_ratios = df_ratios_T.reset_index()
df_ratios.head()
```

Out[16]:

	index	0
0	Barking & Dagenham	4.896619
1	Barnet	4.358196
2	Bexley	4.248977
3	Brent	4.894554
4	Bromley	4.094785

In [17]:
```
df_ratios.rename(columns={'index':'London_Borough', 0:'2018'}, inplace=True)
df_ratios.head(10)
```

Out[17]:

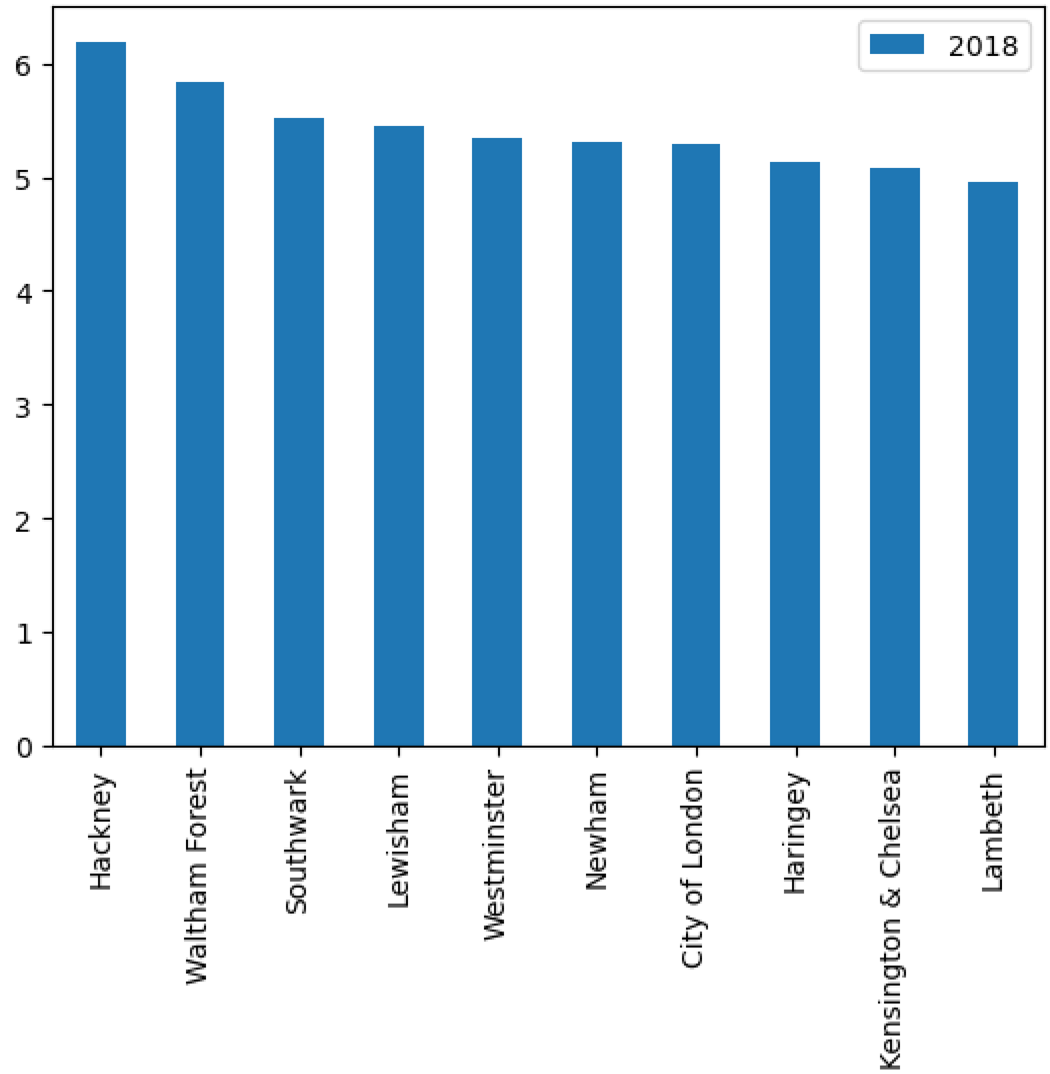
	London_Borough	2018
0	Barking & Dagenham	4.896619
1	Barnet	4.358196
2	Bexley	4.248977
3	Brent	4.894554
4	Bromley	4.094785
5	Camden	4.935353
6	City of London	5.301620
7	Croydon	4.201100
8	Ealing	4.311451
9	Enfield	4.263472

In [18]:
```
top_10 = df_ratios.sort_values(by='2018', ascending = False).head(10)

print(top_10)
```

```
          London_Borough      2018
11               Hackney  6.198286
30        Waltham Forest  5.834756
27             Southwark  5.516485
22              Lewisham  5.449221
32           Westminster  5.353565
24                Newham  5.305390
6         City of London  5.301620
13              Haringey  5.134625
19  Kensington & Chelsea  5.082465
21               Lambeth  4.957751
```

In [19]:
```
ax = top_10[['London_Borough', '2018']].plot(kind ='bar')

ax.set_xticklabels(top_10.London_Borough)
```

Out[19]:
```
[Text(0, 0, 'Hackney'),
 Text(1, 0, 'Waltham Forest'),
 Text(2, 0, 'Southwark'),
 Text(3, 0, 'Lewisham'),
 Text(4, 0, 'Westminster'),
 Text(5, 0, 'Newham'),
 Text(6, 0, 'City of London'),
 Text(7, 0, 'Haringey'),
 Text(8, 0, 'Kensington & Chelsea'),
 Text(9, 0, 'Lambeth')]
```



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 [20]:
```
# In the order, Hackney shows almost 620% increase in 2018 compared to that 1998 where other follows and I have plotted
```