London Housing — Data Wrangling Mini Project

Goal: Clean, standardize, and merge monthly and yearly London housing datasets into a single, analysis-ready dataset.

Datasets:

- `/mnt/data/housing_in_london_monthly_variables.csv`
- `/mnt/data/housing_in_london_yearly_variables.csv`

Deliverables:

- Cleaned CSV: `london_housing_cleaned.csv`
- This notebook with step-by-step transformations and before/after samples

In [3]:
```python
# 1) Imports
import pandas as pd
import numpy as np
from pathlib import Path

pd.set_option('display.max_columns', None)
DATA_DIR = Path('/mnt/data')
MONTHLY_PATH = DATA_DIR / 'housing_in_london_monthly_variables.csv'
YEARLY_PATH = DATA_DIR / 'housing_in_london_yearly_variables.csv'
```

2) Load and Inspect Data

In [5]:
```python
import pandas as pd

monthly = pd.read_csv('housing_in_london_monthly_variables.csv')
yearly = pd.read_csv('housing_in_london_yearly_variables.csv')

print("Monthly shape:", monthly.shape)
print("Yearly shape:", yearly.shape)

display(monthly.head(3))
display(yearly.head(3))

print("\nMonthly .info():")
monthly.info()
print("\nYearly .info():")
yearly.info()
```

```
Monthly shape: (13549, 7)
Yearly shape: (1071, 12)
```

	date	area	average_price	code	houses_sold	no_of_crimes	borough_flag
0	1995-01-01	city of london	91449	E09000001	17.0	NaN	1
1	1995-02-01	city of london	82203	E09000001	7.0	NaN	1
2	1995-03-01	city of london	79121	E09000001	14.0	NaN	1

	code	area	date	median_salary	life_satisfaction	mean_salary	recyclir
0	E09000001	city of london	1999-12-01	33020.0	NaN	48922	
1	E09000002	barking and dagenham	1999-12-01	21480.0	NaN	23620	
2	E09000003	barnet	1999-12-01	19568.0	NaN	23128	

```
Monthly .info():
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13549 entries, 0 to 13548
Data columns (total 7 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   date           13549 non-null  object
 1   area           13549 non-null  object
 2   average_price  13549 non-null  int64
 3   code           13549 non-null  object
 4   houses_sold    13455 non-null  float64
 5   no_of_crimes   7439 non-null   float64
 6   borough_flag   13549 non-null  int64
dtypes: float64(2), int64(2), object(3)
memory usage: 741.1+ KB

Yearly .info():
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1071 entries, 0 to 1070
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   code               1071 non-null   object
 1   area               1071 non-null   object
 2   date               1071 non-null   object
 3   median_salary      1049 non-null   float64
 4   life_satisfaction  352 non-null    float64
 5   mean_salary        1071 non-null   object
 6   recycling_pct      860 non-null    object
 7   population_size    1018 non-null   float64
 8   number_of_jobs     931 non-null    float64
 9   area_size          666 non-null    float64
 10  no_of_houses       666 non-null    float64
 11  borough_flag       1071 non-null   int64
dtypes: float64(6), int64(1), object(5)
memory usage: 100.5+ KB
```

3) Standardize Column Names

In [6]:

```python
def to_snake(s: str) -> str:
    return s.strip().lower().replace(' ', '_')

monthly.columns = [to_snake(c) for c in monthly.columns]
yearly.columns  = [to_snake(c) for c in yearly.columns]

monthly.head(1), yearly.head(1)
```

Out[6]:
```
(         date            area  average_price       code  houses_sold  \
 0  1995-01-01  city of london          91449  E09000001         17.0

    no_of_crimes  borough_flag
 0           NaN             1  ,
          code            area        date  median_salary  life_satisfaction
\
 0  E09000001  city of london  1999-12-01        33020.0                NaN

   mean_salary recycling_pct  population_size  number_of_jobs  area_size  \
 0       48922             0           6581.0             NaN        NaN

    no_of_houses  borough_flag
 0           NaN             1  )
```

4) Fix Data Types

In [7]:
```
# Convert date columns to datetime
monthly['date'] = pd.to_datetime(monthly['date'], errors='coerce')
yearly['date']  = pd.to_datetime(yearly['date'], errors='coerce')

# Coerce obviously numeric fields that might be 'object' to numeric
numeric_like_cols_yearly = ['median_salary','mean_salary','life_satisfactior
                            'population_size','number_of_jobs','area_size','

for col in numeric_like_cols_yearly:
    if col in yearly.columns:
        yearly[col] = pd.to_numeric(yearly[col], errors='coerce')

# Show dtypes summary
monthly.dtypes, yearly.dtypes
```

Out[7]:
```
(date             datetime64[ns]
 area                     object
 average_price             int64
 code                     object
 houses_sold             float64
 no_of_crimes            float64
 borough_flag              int64
 dtype: object,
 code                         object
 area                         object
 date                 datetime64[ns]
 median_salary               float64
 life_satisfaction           float64
 mean_salary                 float64
 recycling_pct               float64
 population_size             float64
 number_of_jobs              float64
 area_size                   float64
 no_of_houses                float64
 borough_flag                  int64
 dtype: object)
```

5) Clean Categorical Data

In [8]:
```
# Standardize area names to title case and strip whitespace
for df in [monthly, yearly]:
    if 'area' in df.columns:
        df['area'] = df['area'].astype(str).str.strip().str.title()

# Quick spot-check
monthly['area'].head(10).to_frame().T
```

Out[8]:

	0	1	2	3	4	5	6	7	8	
area	City Of London	City Of London	City Of London	City Of London	City Of London	City Of London	City Of London	City Of London	City Of London	City Lonc

6) Missing Value Summary

In [9]:
```
def missing_summary(df: pd.DataFrame) -> pd.DataFrame:
    total = df.isna().sum()
    pct = 100 * total / len(df)
    return pd.DataFrame({'missing_count': total, 'missing_pct': pct.round(2)})

ms_monthly = missing_summary(monthly)
ms_yearly = missing_summary(yearly)

ms_monthly.head(10), ms_yearly.head(10)
```

Out[9]:
```
(               missing_count  missing_pct
 no_of_crimes            6110        45.10
 houses_sold               94         0.69
 date                       0         0.00
 average_price              0         0.00
 area                       0         0.00
 code                       0         0.00
 borough_flag               0         0.00,
                    missing_count  missing_pct
 life_satisfaction            719        67.13
 area_size                    405        37.82
 no_of_houses                 405        37.82
 recycling_pct                212        19.79
 number_of_jobs               140        13.07
 population_size               53         4.95
 median_salary                 22         2.05
 mean_salary                   17         1.59
 date                           0         0.00
 area                           0         0.00)
```

7) Outlier Detection (IQR) — Flag Only (Non-Destructive)

In [ ]:
```
def flag_outliers_iqr(df, cols):
    flagged = pd.DataFrame(index=df.index)
    for c in cols:
        if c in df.columns:
            series = pd.to_numeric(df[c], errors='coerce')
            q1, q3 = series.quantile(0.25), series.quantile(0.75)
            iqr = q3 - q1
            low, high = q1 - 1.5*iqr, q3 + 1.5*iqr
            flagged[f'{c}_is_outlier'] = (series < low) | (series > high)
    return flagged

monthly_outliers = flag_outliers_iqr(monthly, ['average_price','houses_sold'
yearly_outliers  = flag_outliers_iqr(yearly, ['median_salary','mean_salary',

monthly_outliers.sum(), yearly_outliers.sum()
```

8) Handle Missing Values (Example Strategies)

In [ ]:
```
# We'll do conservative imputations:
# - 'houses_sold': fill with group median by area (monthly), then global med
# - 'no_of_crimes': leave as NaN (too sparse), but you can interpolate withi
# - yearly numeric gaps: fill small gaps with area medians where sensible

monthly_clean = monthly.copy()
if 'houses_sold' in monthly_clean.columns:
    monthly_clean['houses_sold'] = monthly_clean.groupby('area')['houses_sol
    monthly_clean['houses_sold'] = monthly_clean['houses_sold'].fillna(month

# Example: do not fill 'no_of_crimes' aggressively; keep NaN (documented cho
# yearly: fill selected columns by area median
yearly_clean = yearly.copy()
cols_to_fill_area_median = ['median_salary','mean_salary','recycling_pct','p
for col in cols_to_fill_area_median:
    if col in yearly_clean.columns:
        yearly_clean[col] = yearly_clean.groupby('area')[col].transform(lamb
        yearly_clean[col] = yearly_clean[col].fillna(yearly_clean[col].media

# Show before/after samples
display(monthly[['area','date','houses_sold','no_of_crimes']].sample(5, rand
display(monthly_clean[['area','date','houses_sold','no_of_crimes']].sample(5
display(yearly[['area','date','mean_salary','recycling_pct','population_size
display(yearly_clean[['area','date','mean_salary','recycling_pct','populatio
```

9) Merge Monthly & Yearly Data

In [ ]:
```
# We'll merge on exact month date. If yearly is year-only, align by year.
# First create year/month columns on monthly; and year-only on yearly
monthly_clean['year'] = monthly_clean['date'].dt.year
monthly_clean['month'] = monthly_clean['date'].dt.month

yearly_clean['year'] = yearly_clean['date'].dt.year
```

```
# Merge yearly attributes by (area, year) onto monthly granularity
merge_cols = ['area','year']
to_add = [c for c in yearly_clean.columns if c not in merge_cols + ['date','
merged = monthly_clean.merge(yearly_clean[merge_cols + to_add], on=merge_col

print("Merged shape:", merged.shape)
merged.head(3)
```

10) Add Derived Columns

In [ ]:
```
# Derived metrics
if 'population_size' in merged.columns:
    merged['crime_rate_per_1k'] = (merged['no_of_crimes'] / merged['populati
else:
    merged['crime_rate_per_1k'] = np.nan

# Example: price to salary ratio (if salaries available)
if 'mean_salary' in merged.columns and 'average_price' in merged.columns:
    merged['price_to_mean_salary'] = merged['average_price'] / merged['mean_

merged[['area','date','average_price','no_of_crimes','population_size','crim
```

11) Final Checks & Save

In [ ]:
```
# Drop exact duplicate rows if any
before = len(merged)
merged = merged.drop_duplicates()
after = len(merged)
print(f"Dropped {before - after} duplicate rows. Final rows: {after}")

# Save
OUTPUT_PATH = DATA_DIR / 'london_housing_cleaned.csv'
merged.to_csv(OUTPUT_PATH, index=False)
OUTPUT_PATH
```

12) Notes & Decisions

- **Area standardization:** Converted to title case to align names across files.
- **Type coercion:** Converted `date` to datetime; coerced numeric-looking columns to numeric with `errors='coerce'`.
- **Missing values:** Conservatively imputed:
  - `houses_sold` via area median, then global median.
  - Left `no_of_crimes` as NaN due to sparsity (documented trade-off).
  - Yearly fields imputed via area median, then global median as fallback.
- **Outliers:** Only flagged using IQR; no capping/removal performed (non-destructive). You can add capping if needed.
- **Merge:** Joined yearly attributes to monthly granularity on `(area, year)`.

- **Derived features:** Added `crime_rate_per_1k` and `price_to_mean_salary` as examples.