Capstone Project (Feature Engineering)

Load the data:

```
df = pd.read_csv('gurgaon_properties_cleaned_v1.csv')
```

Columns to work on:

- areaWithType
- additionalRoom
- agePossession
- furnishDetails
- features

areaWithType

3 Areas:

- Carpet Area: Area of all rooms
- Build-up area : Carpet Area + Thickness of wall + Balcony
- Super Build-up area: Any additional common area (eg. Stairs + Garden)
- Select 5 random sample

```
df.sample(5)[['price','area','areaWithType']]
```

	price	area	areaWithType
3094	1.07	1350.0	Super Built up area 1350(125.42 sq.m.)
1216	0.34	629.0	Carpet area: 629 (58.44 sq.m.)
1816	1.25	1423.0	Built Up area: 1423 (132.2 sq.m.)Carpet area: 1190 sq.ft. (110.55 sq.m.)
143	1.40	2208.0	Built Up area: 1735 (161.19 sq.m.)
205	0.55	1130.0	Carpet area: 1130 (104.98 sq.m.)

- The `area` column is not reliable.

💡 **We can make 3 Columns with `areaWithType`**

```
# This function extracts the Super Built up area
def get_super_built_up_area(text):
    match = re.search(r'Super Built up area[:\s]?(\d+\.?\d*)', text)
    if match:
        return float(match.group(1))
    return None
```

`match = re.search(r'Super Built up area (\d+\.?\d*)', text)`

- `re.search()` is a method from the `re` **(regular expressions) module**. It is used to search for a pattern in a given string (`text` in this case).

`[:\s]?` :

- This part makes the colon (`:`) **optional**.

- `\s` matches any whitespace character (like space, tab, etc.)

`(\d+\.?\d*)`: This part is a **capture group** that matches a number. Here's how it works:

- `\d+`: Matches one or more digits (i.e., `1`, `10`, `100`, etc.).
- `\.?`: Matches an optional decimal point (`.`). The `?` means it is optional, so it can match both integer numbers (like `100`) and floating-point numbers (like `100.5`).
- `\d*`: Matches zero or more digits after the decimal point. This allows the number to have digits after the decimal (e.g., `100.5`).

So, the whole pattern `r'Super Built up area (\d+\.?\d*)'` will match strings like:

- `"Super Built up area 150"`
- `"Super Built up area 200.5"`

And it will capture the numeric part (e.g., `150` or `200.5`) as a match group.

`match = re.search(..., text)`:

This searches for the first occurrence of this pattern in the input `text`. If a match is found, it will store the result in `match`. If no match is found, `match` will be `None`.

`match.group(1)`: If a match was found, `group(1)` returns the first **captured group** from the regular expression, which is the number (e.g., `150` or `200.5`).

> A "**captured group**" is a part of the regular expression that is enclosed in parentheses `()`

- `match.group(0)`: Returns the entire match — `"Super Built up area 150.5"`.

- `match.group(1)`: Returns the first captured group, which is the number — `"150.5"`

```
# This function extracts the Built Up area or Carpet area
def get_area(text, area_type):
    match = re.search(area_type + r'\s*:\s*(\d+\.?\d*)', text)
    if match:
        return float(match.group(1))
    return None
```

- `re.search()`: Finds the first match of the given pattern in the text.
- **Pattern (`area_type + r'\s*:\s*(\d+\.?\d*)'`)**:
  - `area_type`: The dynamic part, such as "Built Up area" or "Carpet area".
  - `\s*`: Matches any spaces around the colon.
  - `(\d+\.?\d*)`: Matches and captures the number (with optional decimal), which represents the area.
- `match.group(1)`: Retrieves the captured area value (e.g., `1350`).
- `float()`: Converts the area value to a float.
- Returns `None` if no match is found.

Ex. *Extracting "Built Up area":*

```
text = "Built Up area: 1350 Carpet area: 1190"
area = get_area(text, "Built Up area")
print(area)  # Output: 1350.0
```

Convert the area into sqft if needed:

```
# This function checks if the area is provided in sq.m. and converts it to sqft if needed

def convert_to_sqft(text, area_value):
    # If area_value is None, return None immediately
    if area_value is None:
        return None

    # Look for the specific area_value in the text followed by (some number) sq.m.
    match = re.search(r'{} \((\d+\.?\d*) sq.m.\)'.format(area_value), text)

    # If a match is found (i.e., area is in sq.m.), convert it to sqft
    if match:
        sq_m_value = float(match.group(1))  # Get the numerical value of sq.m.
        return sq_m_value * 10.7639  # Convert sq.m. to sqft using the conversion factor

    # If no match found, return the original area_value
    return area_value
```

`r'{} \((\d+\.?\d*) sq.m.\)'.format(area_value)`

- The regular expression is constructed dynamically to search for the `area_value` followed by a space, then a number (which represents the area), and the unit `sq.m.` within parentheses.
- Example: If `area_value = "Built Up area"`, it will search for text like `"Built Up area (150.5 sq.m.)"`.

If a Match is Found:

- `match.group(1)`: This retrieves the matched number (the area in `sq.m.`) as a string.
- `float()`: Converts the string to a float for calculations.
- `sq_m_value * 10.7639`: This converts the value from **square meters** to **square feet** using the conversion factor of 1 sq.m. = 10.7639 sqft.

Return the Original Value:

- If the area is not in `sq.m.` (i.e., the pattern is not matched), the function returns the original `area_value` as is, without any changes.

Example:

```
text = "Built Up area (150.5 sq.m.)"
area_value = "Built Up area"
converted_value = convert_to_sqft(text, area_value)
print(converted_value)  # Output: 1614.14395 (converted to sqft)
```

Make 3 columns for 3 areas:

```
# Extract Super Built up area and convert to sqft if needed
df['super_built_up_area'] = df['areaWithType'].apply(get_super_built_up_area)
df['super_built_up_area'] = df.apply(lambda x: convert_to_sqft(x['areaWithTyp
e'], x['super_built_up_area']), axis=1)

# Extract Built Up area and convert to sqft if needed
df['built_up_area'] = df['areaWithType'].apply(lambda x: get_area(x, 'Built Up a
rea'))
df['built_up_area'] = df.apply(lambda x: convert_to_sqft(x['areaWithType'], x
['built_up_area']), axis=1)

# Extract Carpet area and convert to sqft if needed
df['carpet_area'] = df['areaWithType'].apply(lambda x: get_area(x, 'Carpet are
a'))
df['carpet_area'] = df.apply(lambda x: convert_to_sqft(x['areaWithType'], x['ca
rpet_area']), axis=1)
```

```
df[['price','property_type','area','areaWithType','super_built_up_area','built_up_area','carpet_area']].sample(5)
```

	price	property_type	area	areaWithType	super_built_up_area	built_up_area	carpet_area
3419	1.24	flat	1000.0	Super Built up area 1150(106.84 sq.m.)Built Up area: 1050 sq.ft. (97.55 sq.m.)Carpet area: 1000 sq.ft. (92.9 sq.m.)	1150.0	1050.0	1000.0
3725	1.95	flat	1400.0	Super Built up area 1852(172.06 sq.m.)Carpet area: 1400 sq.ft. (130.06 sq.m.)	1852.0	NaN	1400.0
3278	0.85	flat	1530.0	Super Built up area 1350(125.42 sq.m.)	1350.0	NaN	NaN
2416	NaN	house	NaN	Plot area 360(301.01 sq.m.)	NaN	NaN	NaN
240	0.99	flat	1623.0	Super Built up area 1621(150.6 sq.m.)	1621.0	NaN	NaN

Check the rows where all the 3 values are present

```
df[~((df['super_built_up_area'].isnull()) | (df['built_up_area'].isnull()) | (df['carpet_area'].isnull()))][['price','property_type','area','areaWithType','super_built_up_area','built_up_area','carpet_area']]
```

	price	property_type	area	areaWithType	super_built_up_area	built_up_area	carpet_area
2	1.08	flat	1800.0	Super Built up area 1800(167.23 sq.m.)Built Up area: 1700 sq.ft. (157.94 sq.m.)Carpet area: 1600 sq.ft. (148.64 sq.m.)	1800.0	1700.00	1600.00
5	0.90	flat	1446.0	Super Built up area 1285(119.38 sq.m.)Built Up area: 1185 sq.ft. (110.09 sq.m.)Carpet area: 975 sq.ft. (90.58 sq.m.)	1285.0	1185.00	975.00
9	1.54	flat	1670.0	Super Built up area 1929(179.21 sq.m.)Built Up area: 1780 sq.ft. (165.37 sq.m.)Carpet area: 1670 sq.ft. (155.15 sq.m.)	1929.0	1780.00	1670.00
17	1.75	flat	2200.0	Super Built up area 2200(204.39 sq.m.)Built Up area: 1900 sq.ft. (176.52 sq.m.)Carpet area: 1700 sq.ft. (157.94 sq.m.)	2200.0	1900.00	1700.00
25	2.99	flat	2527.0	Super Built up area 2527(234.77 sq.m.)Built Up area: 2200 sq.ft. (204.39 sq.m.)Carpet area: 2100 sq.ft. (195.1 sq.m.)	2527.0	2200.00	2100.00
40	1.55	flat	2191.0	Super Built up area 2191(203.55 sq.m.)Built Up area: 2100 sq.ft. (195.1 sq.m.)Carpet area: 1800 sq.ft. (167.23 sq.m.)	2191.0	2100.00	1800.00
45	0.50	flat	1050.0	Super Built up area 1050(97.55 sq.m.)Built Up area: 850 sq.ft. (78.97 sq.m.)Carpet area: 645 sq.ft. (59.92 sq.m.)	1050.0	850.00	645.00
46	2.25	flat	2103.0	Super Built up area 2103(195.38 sq.m.)Built Up area: 1600 sq.ft. (148.64 sq.m.)Carpet area: 1257 sq.ft. (116.78 sq.m.)	2103.0	1600.00	1257.00
77	1.13	flat	1764.0	Super Built up area 1760(163.51 sq.m.)Built Up area: 1186 sq.ft. (110.18 sq.m.)Carpet area: 1130 sq.ft. (104.98 sq.m.)	1760.0	1186.00	1130.00
84	1.54	flat	1400.0	Super Built up area 1400(130.06 sq.m.)Built Up area: 1200 sq.ft. (111.48 sq.m.)Carpet area: 1100 sq.ft. (102.19 sq.m.)	1400.0	1200.00	1100.00
110	0.85	flat	950.0	Super Built up area 1245(115.66 sq.m.)Built Up area: 1100 sq.ft. (102.19 sq.m.)Carpet area: 950 sq.ft. (88.26 sq.m.)	1245.0	1100.00	950.00

```
df[~((df['super_built_up_area'].isnull()) | (df['built_up_area'].isnull()) | (df['carpet_area'].isnull()))][['price','property_type','area','areaWithType','super_built_up_area','built_up_area','carpet_area']].shape
```

Output: (534, 7)

- `df['super_built_up_area'].isnull()`: Checks if the `'super_built_up_area'` is `null`.
- `|`: Combines conditions with OR. If any of the conditions are `True`, the overall result will be `True`.
- `~`: Negates the result.
- **This means we're selecting rows where none of the conditions are `True` (i.e., none of the specified columns are null).**

- Rows with `plot` **values:**

```
df[df['areaWithType'].str.contains('Plot')][['price','property_type','area','areaWithType','super_built_up_area','built_up_area','carpet_area']].head(5)
```

	price	property_type	area	areaWithType	super_built_up_area	built_up_area	carpet_area
1	19.00	house	6000.0	Plot area 9000(836.13 sq.m.)Carpet area: 6000 sq.ft. (557.42 sq.m.)	NaN	NaN	6000.0
3	0.99	house	576.0	Plot area 64(53.51 sq.m.)	NaN	NaN	NaN
16	1.25	house	1080.0	Plot area 120(100.34 sq.m.)Built Up area: 120 sq.yards (100.34 sq.m.)	NaN	120.0	NaN
20	0.55	house	1350.0	Plot area 1350(125.42 sq.m.)	NaN	NaN	NaN
21	4.60	house	1460.0	Plot area 1460(135.64 sq.m.)	NaN	NaN	NaN

```
df[df['areaWithType'].str.contains('Plot')][['price','property_type','area','areaWithType','super_built_up_area','built_up_area','carpet_area']].shape
```

Output: (682, 7)

```
df.isnull().sum()
```

```
property_type              0
society                    1
sector                     0
price                     18
price_per_sqft            18
area                      18
areaWithType               0
bedRoom                    0
bathroom                   0
balcony                    0
additionalRoom             0
floorNum                  19
facing                  1105
agePossession              1
nearbyLocations          177
furnishDetails           981
features                 635
super_built_up_area     1888
built_up_area           2616
carpet_area             1859
dtype: int64
```

Create a df where all 👆these 3 values are null

```
all_nan_df = df[((df['super_built_up_area'].isnull()) & (df['built_up_area'].isnull()) & (df['carpet_area'].isnull()))][['price','property_type','area','areaWithType','super_built_up_area','built_up_area','carpet_area']]
```

```
all_nan_df.head()
```

	price	property_type	area	areaWithType	super_built_up_area	built_up_area	carpet_area
3	0.99	house	576.0	Plot area 64(53.51 sq.m.)	NaN	NaN	NaN
20	0.55	house	1350.0	Plot area 1350(125.42 sq.m.)	NaN	NaN	NaN
21	4.60	house	1460.0	Plot area 1460(135.64 sq.m.)	NaN	NaN	NaN
31	9.85	house	3323.0	Plot area 418(349.5 sq.m.)	NaN	NaN	NaN
34	3.10	house	2250.0	Plot area 250(209.03 sq.m.)	NaN	NaN	NaN

- Store the index

```
all_nan_index = df[((df['super_built_up_area'].isnull()) & (df['built_up_area'].isnull()) & (df['carpet_area'].isnull()))][['price','property_type','area','areaWithType','super_built_up_area','built_up_area','carpet_area']].index
```

```
Index([   3,   20,   21,   31,   34,   36,   38,   52,   57,   59,
       ...
       3679, 3682, 3701, 3735, 3748, 3763, 3775, 3780, 3781, 3795],
      dtype='int64', length=546)
```

✨**Note: The values aren't populated because there is plot area present.**

✅ **We can add plot area to build-up area.**

- They are similar👆

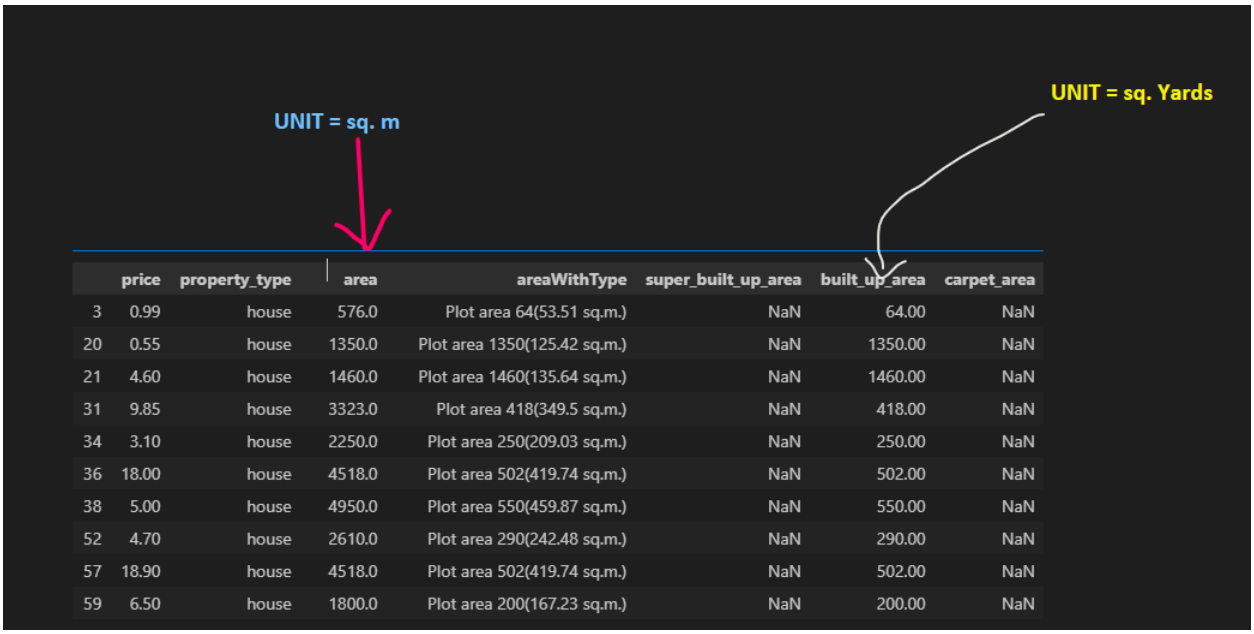
```
# Function to extract plot area from 'areaWithType' column
def extract_plot_area(area_with_type):
    match = re.search(r'Plot area (\d+\.?\d*)', area_with_type)
    return float(match.group(1)) if match else None
```

- **Update the area in** `all_nan_df:`

```
all_nan_df['built_up_area'] = all_nan_df['areaWithType'].apply(extract_plot_area)

# Update the original dataframe
#gurgaon_properties.update(filtered_rows)
```

```
all_nan_df
```



	price	property_type	area	areaWithType	super_built_up_area	built_up_area	carpet_area
3	0.99	house	576.0	Plot area 64(53.51 sq.m.)	NaN	64.00	NaN
20	0.55	house	1350.0	Plot area 1350(125.42 sq.m.)	NaN	1350.00	NaN
21	4.60	house	1460.0	Plot area 1460(135.64 sq.m.)	NaN	1460.00	NaN
31	9.85	house	3323.0	Plot area 418(349.5 sq.m.)	NaN	418.00	NaN
34	3.10	house	2250.0	Plot area 250(209.03 sq.m.)	NaN	250.00	NaN
36	18.00	house	4518.0	Plot area 502(419.74 sq.m.)	NaN	502.00	NaN
38	5.00	house	4950.0	Plot area 550(459.87 sq.m.)	NaN	550.00	NaN
52	4.70	house	2610.0	Plot area 290(242.48 sq.m.)	NaN	290.00	NaN
57	18.90	house	4518.0	Plot area 502(419.74 sq.m.)	NaN	502.00	NaN
59	6.50	house	1800.0	Plot area 200(167.23 sq.m.)	NaN	200.00	NaN

Convert yard → m

```
def convert_scale(row):
    if np.isnan(row['area']) or np.isnan(row['built_up_area']):
        return row['built_up_area']
    else:
        if round(row['area']/row['built_up_area']) == 9.0:
            return row['built_up_area'] * 9
        elif round(row['area']/row['built_up_area']) == 11.0:
            return row['built_up_area'] * 10.7
        else:
            return row['built_up_area']
```

1 sq yd = 9 sq ft

- ***9 * sq.yard = 1 sq.ft***

- ***10.7 * sq.mt =1 sq.ft***

```
all_nan_df['built_up_area'] = all_nan_df.apply(convert_scale,axis=1)
all_nan_df.head()
```

	price	property_type	area	areaWithType	super_built_up_area	built_up_area	carpet_area
3	0.99	house	576.0	Plot area 64(53.51 sq.m.)	NaN	576.0	NaN
20	0.55	house	1350.0	Plot area 1350(125.42 sq.m.)	NaN	1350.0	NaN
21	4.60	house	1460.0	Plot area 1460(135.64 sq.m.)	NaN	1460.0	NaN
31	9.85	house	3323.0	Plot area 418(349.5 sq.m.)	NaN	418.0	NaN
34	3.10	house	2250.0	Plot area 250(209.03 sq.m.)	NaN	2250.0	NaN

Update the original df:

```
df.update(all_nan_df)
```

- The `update()` method in Pandas updates the values in df with values from another DataFrame (`all_nan_df`) where the indices and columns align.

How It Works:

- It matches rows by index and columns by name between df and **`all_nan_df`**.
- For each matching cell, it replaces the value in df with the value from `all_nan_df`, but only if the value in **`all_nan_df`** is not **NaN**.
- If the value in `all_nan_df` is **NaN**, the corresponding value in df remains unchanged.
- **In-Place:** `update()` modifies df directly (doesn't return a new DataFrame).

additionalRoom

```
df['additionalRoom'].value_counts()
```

```
not available                                  1587
servant room                                    705
study room                                      250
others                                          225
pooja room                                      165
store room                                       99
study room,servant room                          99
pooja room,servant room                          82
pooja room,study room,servant room,store room    72
servant room,others                              60
pooja room,study room,servant room               55
pooja room,study room,servant room,others        54
servant room,pooja room                          38
servant room,store room                          33
study room,others                                29
pooja room,study room                            22
pooja room,others                                17
pooja room,store room                            15
pooja room,store room,study room,servant room    12
servant room,study room                          12
study room,servant room,store room               11
pooja room,servant room,others                   11
study room,pooja room                            10
servant room,study room,pooja room,store room    10
study room,servant room,pooja room,store room     8
...
store room,servant room,study room,pooja room     1
servant room,pooja room,study room                1
pooja room,store room,servant room                1
store room,pooja room,servant room,study room     1
```

- We can divide this into 5 columns

```
# additional room
# List of new columns to be created
new_cols = ['study room', 'servant room', 'store room', 'pooja room', 'others']
```

```
# Populate the new columns based on the "additionalRoom" column
for col in new_cols:
    df[col] = df['additionalRoom'].str.contains(col).astype(int)
```

```
df.sample(5)[['additionalRoom','study room', 'servant room', 'store room', 'poo
ja room', 'others']]
```

	additionalRoom	study room	servant room	store room	pooja room	others
1968	not available	0	0	0	0	0
1735	not available	0	0	0	0	0
2654	study room	1	0	0	0	0
2059	not available	0	0	0	0	0
264	not available	0	0	0	0	0

`.str.contains(col):`

- Checks if the string col (e.g., "study room") is present in each value of additionalRoom.
- Returns a Series of True/False values.
- **Example:** If additionalRoom is "study room, pooja room", then .str.contains('study room') returns True.

`.astype(int):`

- Converts True to 1 and False to 0.
- Example: True → 1, False → 0.

	additionalRoom	study room	servant room	store room	pooja room	others
1968	not available	0	0	0	0	0
1735	not available	0	0	0	0	0
2654	study room	1	0	0	0	0
2059	not available	0	0	0	0	0
264	not available	0	0	0	0	0

💡 👆This is **one-hot encoding**

agePossession

```
df['agePossession'].value_counts()
```

agePossession	
1 to 5 Year Old	1676
5 to 10 Year Old	575
0 to 1 Year Old	530
undefined	332
10+ Year Old	310
Under Construction	90
Within 6 months	70
Within 3 months	26
23-Dec	20
By 2023	19
By 2024	17
24-Dec	15
24-Mar	14
24-Oct	7
23-Aug	7
24-Jan	7
25-Dec	7
24-Jun	5
23-Nov	5
By 2025	4
24-Jul	4
24-Aug	4
23-Sep	4
24-Feb	3

- This column tells you when you'll get the possession/age of the property.

```
def categorize_age_possession(value):
    if pd.isna(value):
        return "Undefined"
    if "0 to 1 Year Old" in value or "Within 6 months" in value or "Within 3 months"
        return "New Property"
    if "1 to 5 Year Old" in value:
        return "Relatively New"
    if "5 to 10 Year Old" in value:
        return "Moderately Old"
    if "10+ Year Old" in value:
        return "Old Property"
```

```
    if "Under Construction" in value or "By" in value:
        return "Under Construction"
    try:
        # For entries like 'May 2024'
        int(value.split(" ")[-1])
        return "Under Construction"
    except:
        return "Undefined"
```

`value.split(" ")[-1]:`

- `value.split(" ")`: Splits the string value into a list of substrings using a space (" ") as the delimiter.
    - Example: If value = "May 2024", this becomes ["May", "2024"].
- `[-1]`: Takes the last element of the list.
    - Example: From [**"May", "2024"**], it takes **"2024"**.

`int(...)`:

- Tries to convert the last element (e.g., "2024") into an integer.
- If successful (e.g., **`int("2024")`** → **`2024`**), the try block succeeds.
- If it fails (e.g., `int("May")` raises a `ValueError`), the except block runs.

`return "Under Construction":`

- If the **`int()`** conversion succeeds, it assumes value is a date-like string (e.g., "May 2024") indicating a future possession date, so it categorizes it as "Under Construction".

`except: return "Undefined"`:

- If the `int()` conversion fails (e.g., value = "Ready to Move" → last element "Move" → `int("Move")` fails), it categorizes the value as "Undefined".

```
df['agePossession'] = df['agePossession'].apply(categorize_age_possession)
```

- Applied the above function to the `agePossession` column

```
df['agePossession'].value_counts()
```

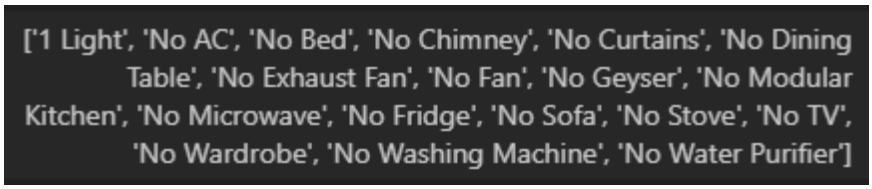
```
agePossession
Relatively New        1676
New Property           626
Moderately Old         575
Undefined              476
Old Property           310
Under Construction     140
Name: count, dtype: int64
```

furnishDetails

```
df.sample(5)[['furnishDetails','features']]
```

	furnishDetails	features
1372	NaN	['Centrally Air Conditioned', 'Security / Fire Alarm', 'Feng Shui / Vaastu Compliant', 'Intercom Facility', 'Lift(s)', 'High Ceiling Height', 'Maintenance Staff', 'Water Storage', 'Separate entry for servant room', 'No open drainage around', 'Internet/wi-fi connectivity', 'Recently Renovated', 'Visitor Parking', 'Swimming Pool', 'Park', 'Security Personnel', 'Natural Light', 'Airy Rooms', 'Spacious Interiors', 'Waste Disposal', 'Rain Water Harvesting', 'Water softening plant', 'Shopping Centre', 'Fitness Centre / GYM', 'Club house / Community Center']
1383	NaN	['Power Back-up', 'Intercom Facility', 'Lift(s)', 'High Ceiling Height', 'Swimming Pool', 'Maintenance Staff', 'Park', 'Visitor Parking', 'Internet/wi-fi connectivity', 'Fitness Centre / GYM', 'Club house / Community Center', 'Water softening plant']
322	NaN	['Security / Fire Alarm', 'Feng Shui / Vaastu Compliant', 'Intercom Facility', 'Lift(s)', 'High Ceiling Height', 'Maintenance Staff', 'False Ceiling Lighting', 'Water Storage', 'Separate entry for servant room', 'No open drainage around', 'Bank Attached Property', 'Piped-gas', 'Internet/wi-fi connectivity', 'Visitor Parking', 'Swimming Pool', 'Park', 'Security Personnel', 'Natural Light', 'Airy Rooms', 'Spacious Interiors', 'Low Density Society', 'Waste Disposal', 'Water softening plant', 'Shopping Centre', 'Fitness Centre / GYM', 'Club house / Community Center']
1685	['1 Light', 'No AC', 'No Bed', 'No Chimney', 'No Curtains', 'No Dining Table', 'No Exhaust Fan', 'No Fan', 'No Geyser', 'No Modular Kitchen', 'No Microwave', 'No Fridge', 'No Sofa', 'No Stove', 'No TV', 'No Wardrobe', 'No Washing Machine', 'No Water Purifier']	NaN
2156	['3 Wardrobe', '1 Exhaust Fan', '3 Geyser', '14 Light', '5 AC', '1 Modular Kitchen', '1 Chimney', 'No Bed', 'No Curtains', 'No Dining Table', 'No Fan', 'No Microwave', 'No Fridge', 'No Sofa', 'No Stove', 'No TV', 'No Washing Machine', 'No Water Purifier']	['Security / Fire Alarm', 'Power Back-up', 'Feng Shui / Vaastu Compliant', 'Intercom Facility', 'Lift(s)', 'Water purifier', 'Centrally Air Conditioned', 'Maintenance Staff', 'Separate entry for servant room', 'No open drainage around', 'Recently Renovated', 'Bank Attached Property', 'Piped-gas', 'Visitor Parking', 'Swimming Pool', 'Park', 'Security Personnel', 'Natural Light', 'Airy Rooms', 'Spacious Interiors', 'Low Density Society', 'Fitness Centre / GYM', 'Rain Water Harvesting', 'Club house / Community Center']

- There is no structure in this data
- Total categories = 18



- We can make 3 columns from this data:
  - Furnished
  - Semi-furnished
  - Unfurnished

```
# Extract all unique furnishings from the furnishDetails column
all_furnishings = []
for detail in df['furnishDetails'].dropna():
    furnishings = detail.replace('[', '').replace(']', '').replace("'", "").split(', ')
    all_furnishings.extend(furnishings)
unique_furnishings = list(set(all_furnishings))

# Define a function to extract the count of a furnishing from the furnishDetail
def get_furnishing_count(details, furnishing):
    if isinstance(details, str):
        if f"No {furnishing}" in details:
            return 0
        pattern = re.compile(f"(\d+) {furnishing}")
        match = pattern.search(details)
        if match:
            return int(match.group(1))
        elif furnishing in details:
            return 1
    return 0

# Simplify the furnishings list by removing "No" prefix and numbers
columns_to_include = [re.sub(r'No |\d+', '', furnishing).strip() for furnishing in
columns_to_include = list(set(columns_to_include))  # Get unique furnishings
```

```
columns_to_include = [furnishing for furnishing in columns_to_include if furni

# Create new columns for each unique furnishing and populate with counts
for furnishing in columns_to_include:
    df[furnishing] = df['furnishDetails'].apply(lambda x: get_furnishing_count(x

# Create the new dataframe with the required columns
furnishings_df = df[['furnishDetails'] + columns_to_include]
```

`all_furnishings = []...`

- The code processes the furnishDetails column (likely containing lists of furnishings as strings, e.g., ['Fan', 'Light']) to create a list of all unique furnishings.

`for detail in df['furnishDetails'].dropna()`:

- Loops over non-NaN values in the furnishDetails column.
- dropna() skips rows where furnishDetails is missing (NaN).

`furnishings = detail.replace('[', '').replace(']', '').replace("'", "").split(', ')`:

- Cleans the string and splits it into a list:
  - `replace('[', '').replace(']', '')`: Removes square brackets (e.g., ['Fan', 'Light'] → Fan', 'Light).
  - `replace("'", "")`: Removes single quotes (e.g., Fan', 'Light → Fan, Light).
  - `split(', ')`: Splits on comma + space into a list (e.g., Fan, Light → ['Fan', 'Light']

`all_furnishings.extend(furnishings):`

- Adds the list of furnishings to all_furnishings.
- `extend()` adds each item individually (e.g., ['Fan', 'Light'] adds Fan and Light to the list).

`unique_furnishings = list(set(all_furnishings)):`

- `set(all_furnishings)`: Removes duplicates (e.g., if Fan appears multiple times, it's kept once).
- `list(...)`: Converts the set back to a list of unique furnishings.

Example

Input (`df['furnishDetails']`):

Index	furnishDetails
0	['Fan', 'Light']
1	['Fan', 'Geyser']
2	NaN

- `dropna()` → `['Fan', 'Light']`, `['Fan', 'Geyser']`.
- After cleaning and splitting:
  - `['Fan', 'Light']` → `['Fan', 'Light']`.
  - `['Fan', 'Geyser']` → `['Fan', 'Geyser']`.
- `all_furnishings.extend()` → `['Fan', 'Light', 'Fan', 'Geyser']`.
- `unique_furnishings = list(set(...))` → `['Fan', 'Light', 'Geyser']`.

```
# Define a function to extract the count of a furnishing from the furnishDetails
def get_furnishing_count(details, furnishing):

    if isinstance(details, str): # Checks if details is a string
        if f"No {furnishing}" in details:
            return 0
```

```
        pattern = re.compile(f"(\d+) {furnishing}")
        match = pattern.search(details)
        if match:
            return int(match.group(1))
        elif furnishing in details:
            return 1
    return 0
```

What Are `details` and `furnishing`?

- `details`: A string from the furnishDetails column in your DataFrame (df). It lists furnishings for a property, like `['1 Light', 'No AC', '2 Fan']`.
  - Example: `details = "['1 Light', 'No AC', '2 Fan']"` **(one row's value)**.
- `furnishing`: The specific item you're looking for, like "`Light`" or "`Fan`".
  - Example: `furnishing = "Light"`.

`if isinstance(details, str)`:

- Checks if `details` is a string (since `furnishDetails` might have `NaN` or other types).
- If details isn't a string (e.g., `NaN`), the function skips to the end and returns `0`

`if f"No {furnishing}" in details`:

- Checks if the string contains "`No {furnishing}`" (e.g., `"No Light"` if `furnishing = "Light"`).
- If found, returns `0` (indicating the furnishing is explicitly not present).

`pattern = re.compile(f"(\d+) {furnishing}")`

- Creates a regex pattern to match a number followed by the furnishing (e.g., "`1 Light`").
  - `(\d+)`: Matches one or more digits (e.g., `1, 2`) and captures them in a group.
  - `{furnishing}`: Matches the furnishing name (e.g., "`Light`").

- Example: For `furnishing = "Light"`, the pattern is `(\d+) Light` (matches `"1 Light"`, "`2 Light"`).

`match = pattern.search(details)`

- Searches for the pattern in details.
- If found, match is a match object; if not, match is `None`

`if match: return int(match.group(1))`

- If a match is found (e.g., `"1 Light"`):
  - `match.group(1)`: Gets the captured digits (e.g., `"1"`).
  - `int(...)`: Converts it to an integer (e.g., `1`).
  - Returns the count (e.g., `1`).

`elif furnishing in details: return 1`

- If no numbered match is found but the furnishing is present (e.g., `"Light"` without a number):
  - Returns `1` (assumes one instance of the furnishing).

`return 0`

- Default case: If details isn't a string, or the furnishing isn't found, or it's `"No {furnishing}"`, returns `0`.

Example

Input:

- `details = "['1 Light', 'No AC', '2 Fan']"`
- `furnishing = "Light"`

Execution:

1. `isinstance(details, str)` → `True` (it's a string).
2. `f"No Light" in details` → `False` (not present).
3. `pattern = re.compile(r"(\d+) Light")`.
4. `match = pattern.search(details)` → Matches `"1 Light"`.
5. `match.group(1)` → `"1"`, `int("1")` → `1`.
6. Returns `1`.

Example

Your DataFrame (`df`):

Index	furnishDetails
0	['1 Light', 'No AC', '2 Fan']

- **Call:** `get_furnishing_count(df['furnishDetails'][0], "Light")`
  - `details = "['1 Light', 'No AC', '2 Fan']"`
  - `furnishing = "Light"`
  - Finds `"1 Light"`, returns `1`.
- **Call:** `get_furnishing_count(df['furnishDetails'][0], "AC")`
  - Sees `"No AC"`, returns `0`.
- **Call:** `get_furnishing_count(df['furnishDetails'][0], "Fan")`
  - Finds `"2 Fan"`, returns `2`.

```
columns_to_include = [re.sub(r'No |\d+', '', furnishing).strip() for furnishing in unique_furnishings]
```

- **What It Does?:** Cleans each item in `unique_furnishings` (e.g., ['1 Light', 'No AC', '2 Fan']) to **remove** `"No"` **and numbers**, leaving just the furnishing name.

`re.sub(r'No |\d+', '', furnishing)` : Removes `No` and numeric values

- `r'No |d+'` : A regex pattern.
  - `No` : Matches "No " (e.g., in "No AC").

- `\d+`: Matches one or more digits (e.g., "1" in "1 Light").
- `''` : Replaces matches with an empty string.
  - Example: "1 Light" → "Light", "No AC" → "AC".

`.strip():`

- Removes extra spaces (e.g., " AC " → "AC").
- **List comprehension:** Applies this to every item in `unique_furnishings`.

Result: `['1 Light', 'No AC', '2 Fan']` → `['Light', 'AC', 'Fan']`.

```
columns_to_include = list(set(columns_to_include))
```

- Removes duplicates from `columns_to_include`.

`set(columns_to_include)`: Converts the list to a set (removes duplicates).

`list(...)`: Converts back to a list

```
columns_to_include = [furnishing for furnishing in columns_to_include if furnishing]
```

- Removes any empty strings (`''`) from `columns_to_include`.
- `if furnishing`: Filters out `''`
- Example: `['Light', 'AC', '', 'Fan']` → `['Light', 'AC', 'Fan']`.

```
for furnishing in columns_to_include:
    df[furnishing] = df['furnishDetails'].apply(lambda x: get_furnishing_count(x, furnishing))
```

- **Creates a new column** in `df` for each furnishing (e.g., Light, AC, Fan) and **fills it with counts**.

- Loops over `columns_to_include` (e.g., `['Light', 'AC', 'Fan']`).
- `df[furnishing]` : Creates a new column named furnishing (e.g., `df['Light']`).
- `df['furnishDetails'].apply(...)` : Applies `get_furnishing_count` to each row in `furnishDetails`.
- `get_furnishing_count(x, furnishing)` : Returns the count of furnishing in x (e.g., 1 for "1 Light", 0 for "No AC").

```
furnishings_df = df[['furnishDetails'] + columns_to_include]
```

- Creates a new DataFrame (`furnishings_df`) with only the `furnishDetails` column and the new furnishing columns.

- `['furnishDetails'] + columns_to_include` : Combines the list `['furnishDetails']` with `columns_to_include` (e.g., ['furnishDetails', 'Light', 'AC', 'Fan']).
- `df[...]` : Selects these columns from `df`.

```
furnishings_df.shape

Output: (3803, 19)
```

Drop the `furnishDetails` column from above dataframe

```
furnishings_df.drop(columns=['furnishDetails'],inplace=True)
```

```
furnishings_df.sample(5)
```

	Fan	Washing Machine	Modular Kitchen	Chimney	Geyser	Bed	Light	Water Purifier	TV	Sofa	Exhaust Fan	Wardrobe	Microwave	Dining Table	Curtains	Fridge	AC	Stove
1371	13	0	1	0	0	0	17	0	0	0	0	0	0	0	0	0	0	0
1335	5	0	1	0	2	0	12	0	0	0	1	3	0	0	0	0	2	0
521	3	0	0	1	0	0	6	0	0	0	0	1	0	0	0	0	3	0
117	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2701	7	0	0	0	4	1	26	0	0	0	1	0	0	0	0	0	8	0

Categorize with K-means Clustering:

Scale the above df:

```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaled_data = scaler.fit_transform(furnishings_df)
```

```
wcss_reduced = []

for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=42)
    kmeans.fit(scaled_data)
    wcss_reduced.append(kmeans.inertia_)
```

WCSS : Within-Cluster Sum of Squares

- Creates an empty list `wcss_reduced` to store WCSS values.

- Loops through i from 1 to 10 (testing 1 to 10 clusters).
- For each `i`, runs K-Means clustering on `scaled_data` with `i` clusters.
- Calculates the WCSS (a measure of how "tight" the clusters are) and adds it to `wcss_reduced`.

`init='k-means++'`: A smart way to choose initial cluster centers to get better results.

`kmeans.fit(scaled_data)`

- Fits the K-Means model to `scaled_data`

- `kmeans.inertia_`: This is the **Within-Cluster-Sum of Squares (WCSS)** for the current clustering.
  - WCSS measures the sum of squared distances between each data point and its assigned cluster centroid.
  - A lower WCSS indicates tighter clusters.
- The WCSS value is appended to the `wcss_reduced` list

Example

Assume:

- `scaled_data` is your data (e.g., property features like price, area, scaled to be comparable).

Execution:

- `i = 1`: K-Means with 1 cluster → `kmeans.inertia_` = 1000 (example value) → `wcss_reduced = [1000]`.
- `i = 2`: K-Means with 2 clusters → `kmeans.inertia_` = 600 → `wcss_reduced = [1000, 600]`.
- `i = 3`: K-Means with 3 clusters → `kmeans.inertia_` = 400 → `wcss_reduced = [1000, 600, 400]`.
- Continues up to `i = 10`.

Output:

- `wcss_reduced` might look like: `[1000, 600, 400, 300, 250, 200, 180, 160, 150, 140]`.

Why This Is Useful?

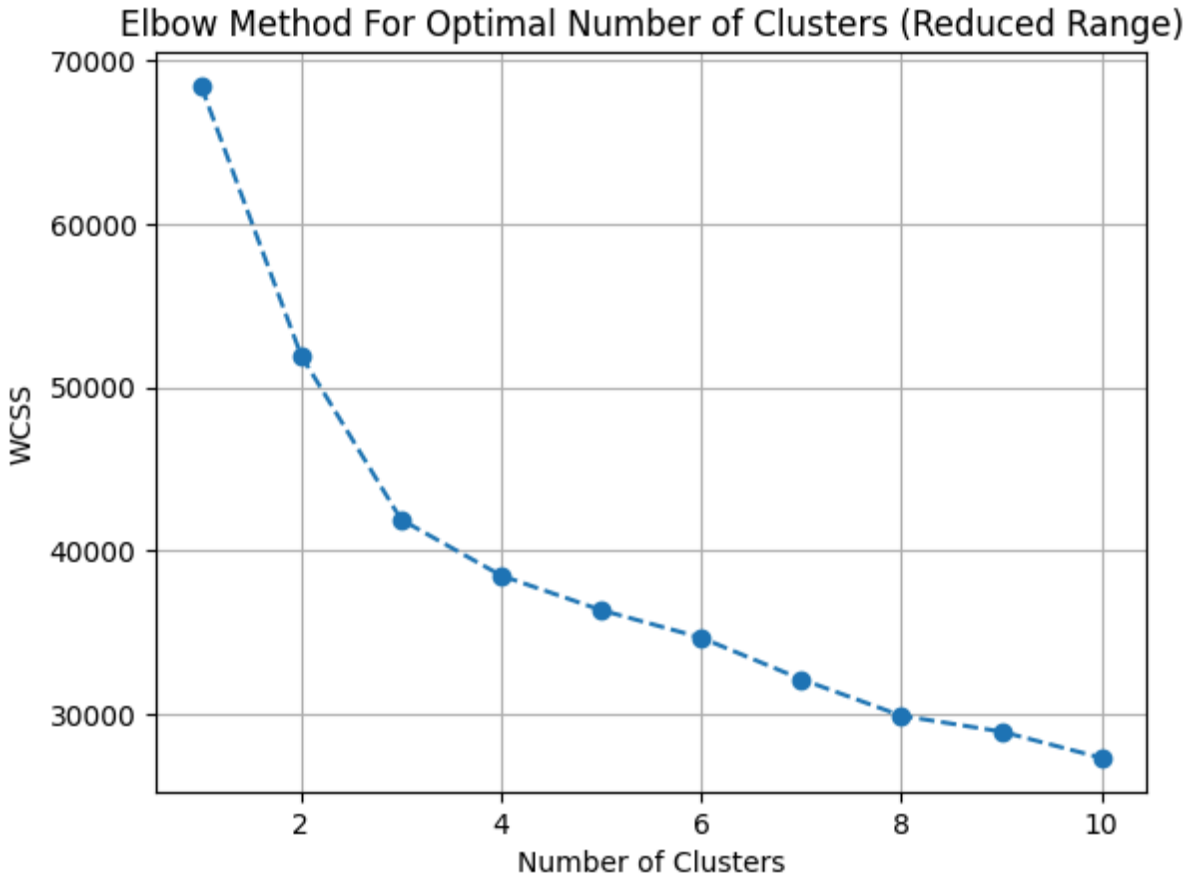
- Elbow Method: You can plot wcss_reduced against the number of clusters (1 to 10) to find the "**elbow point**"—where adding more clusters doesn't reduce WCSS much. That's the optimal number of clusters.
- Example: If WCSS drops a lot from 1 to 3 clusters but flattens after 3, you might choose 3 clusters.

Plot the graph:

```
# Plot the results

plt.plot(range(1,11), wcss_reduced, marker='o', linestyle='--')
plt.title('Elbow Method For Optimal Number of Clusters (Reduced Range)')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
```

```
plt.grid(True)
plt.show()
```



- From the above graph, we can cluster the data in 3 clusters.

```
n_clusters = 3

# Fit the KMeans model
kmeans = KMeans(n_clusters=n_clusters, random_state=42)
kmeans.fit(scaled_data)

# Predict the cluster assignments for each row
cluster_assignments = kmeans.predict(scaled_data)
```

```
df = df.iloc[:,:-18]
df['furnishing_type'] = cluster_assignments
```

```
df.sample(5)[['furnishDetails','furnishing_type']]
# 0 → unfurnished
# 1 → semifurnished
# 2 → furnished
```

	furnishDetails	furnishing_type
1565	['1 Light', 'No AC', 'No Bed', 'No Chimney', 'No Curtains', 'No Dining Table', 'No Exhaust Fan', 'No Fan', 'No Geyser', 'No Modular Kitchen', 'No Microwave', 'No Fridge', 'No Sofa', 'No Stove', 'No TV', 'No Wardrobe', 'No Washing Machine', 'No Water Purifier']	0
1518	['1 Wardrobe', '1 Fan', '1 Light', 'No AC', 'No Bed', 'No Chimney', 'No Curtains', 'No Dining Table', 'No Exhaust Fan', 'No Geyser', 'No Modular Kitchen', 'No Microwave', 'No Fridge', 'No Sofa', 'No Stove', 'No TV', 'No Washing Machine', 'No Water Purifier']	0
390	['3 Fan', '1 Fridge', '1 Washing Machine', '1 Microwave', '3 Light', '1 Chimney', '3 AC', '1 Modular Kitchen', 'No Bed', 'No Curtains', 'No Dining Table', 'No Exhaust Fan', 'No Geyser', 'No Sofa', 'No Stove', 'No TV', 'No Wardrobe', 'No Water Purifier']	2
246	['4 Fan', '1 Exhaust Fan', '2 Geyser', '11 Light', 'No AC', 'No Bed', 'No Chimney', 'No Curtains', 'No Dining Table', 'No Modular Kitchen', 'No Microwave', 'No Fridge', 'No Sofa', 'No Stove', 'No TV', 'No Wardrobe', 'No Washing Machine', 'No Water Purifier']	0
3430	['1 Water Purifier', '9 Fan', '1 Exhaust Fan', '3 Geyser', '1 Stove', '21 Light', '6 AC', '1 Modular Kitchen', '1 Chimney', '1 Curtains', '7 Wardrobe', '1 Microwave', 'No Bed', 'No Dining Table', 'No Fridge', 'No Sofa', 'No TV', 'No Washing Machine']	1

Features

```
df[['society','features']].sample(5)
```

	society	features
872	independent	['Feng Shui / Vaastu Compliant', 'Private Garden / Terrace', 'High Ceiling Height', 'Maintenance Staff', 'False Ceiling Lighting', 'Water Storage', 'Separate entry for servant room', 'No open drainage around', 'Visitor Parking', 'Park', 'Low Density Society', 'Waste Disposal', 'Rain Water Harvesting']
3536	ats triumph	['Security / Fire Alarm', 'Intercom Facility', 'Lift(s)', 'Maintenance Staff', 'Swimming Pool', 'Park', 'Security Personnel', 'Internet/wi-fi connectivity', 'Fitness Centre / GYM', 'Club house / Community Center', 'Rain Water Harvesting', 'Water softening plant']
519	godrej air	NaN
336	umang monsoon breeze	NaN
2989	experion windchants	['Private Garden / Terrace', 'High Ceiling Height', 'Maintenance Staff', 'Swimming Pool', 'Piped-gas', 'Visitor Parking', 'Natural Light', 'Airy Rooms', 'Fitness Centre / GYM', 'Club house / Community Center']

- `features` : Amenities

- Check missing values

```
df['features'].isnull().sum()

Output: 635
```

- We can fill these missing values
  - We have a dataset called `apartments`
  - We can match the columns of both the datasets and fill the missing values

```
import pandas as pd
app_df = pd.read_csv('appartments.csv')
app_df.head(2)
```

	PropertyName	PropertySubName	NearbyLocations	LocationAdvantages	Link	PriceDetails	TopFacilities
0	Smartworld One DXP	2, 3, 4 BHK Apartment in Sector 113, Gurgaon	['Bajghera Road', 'Palam Vihar Halt', 'DPSG Palam Vihar', 'Park Hospital', 'Gurgaon Railway Station']	{'Bajghera Road': '800 Meter', 'Palam Vihar Halt': '2.5 KM', 'DPSG Palam Vihar': '3.1 KM', 'Park Hospital': '3.1 KM', 'Gurgaon Railway Station': '4.9 KM', 'The NorthCap University': '5.4 KM', 'Dwarka Expy': '1.2 KM', 'Hyatt Place Gurgaon Udyog Vihar': '7.7 KM', 'Dwarka Sector 21, Metro Station': '7.2 KM', 'Pacific D21 Mall': '7.4 KM', 'Indira Gandhi International Airport': '14.7 KM', 'Hamoni Golf Camp': '6.2 KM', 'Fun N Food Waterpark': '8.8 KM', 'Accenture DDC5': '9 KM'}	https://www.99acres.com/smartworld-one-dxp-sector-113-gurgaon-npxid-r400415	{'2 BHK': {'building_type': 'Apartment', 'area_type': 'Carpet Area', 'area': '1,370 sq.ft.', 'price-range': '₹ 2 - 2.4 Cr'}, '3 BHK': {'building_type': 'Apartment', 'area_type': 'Carpet Area', 'area': '1,850 - 2,050 sq.ft.', 'price-range': '₹ 2.25 - 3.59 Cr'}, '4 BHK': {'building_type': 'Apartment', 'area_type': 'Carpet Area', 'area': '2,600 sq.ft.', 'price-range': '₹ 3.24 - 4.56 Cr'}}	['Swimming Pool', 'Salon', 'Restaurant', 'Spa', 'Cafeteria', 'Sun Deck', '24x7 Security', 'Club House', 'Gated Community']
1	M3M Crown	3, 4 BHK Apartment in Sector 111, Gurgaon	['DPSG Palam Vihar Gurugram', 'The NorthCap University', 'Park Hospital, Palam Vihar', 'Pacific D21 Mall', 'Palam Vihar Halt Railway Station']	{'DPSG Palam Vihar Gurugram': '1.4 Km', 'The NorthCap University': '4.4 Km', 'Park Hospital, Palam Vihar': '1.4 Km', 'Pacific D21 Mall': '8.2 Km', 'Palam Vihar Halt Railway Station': '1.2 Km', 'Dwarka Sector 21 Metro Station': '8.1 Km', 'Dwarka Expressway': '450 m', 'Fun N Food Water Park': '8.1 Km', 'Indira Gandhi International Airport': '14.1 Km', 'Tau DeviLal Sports Complex': '11.2 Km', 'Hamoni Golf Camp': '5 Km', 'Hyatt Place': '6.1 Km', 'Altrade Business Centre': '11.2 Km'}	https://www.99acres.com/m3m-crown-sector-111-gurgaon-npxid-r404068	{'3 BHK': {'building_type': 'Apartment', 'area_type': 'Super Built-up Area', 'area': '1,605 - 2,170 sq.ft.', 'price-range': '₹ 2.2 - 3.03 Cr'}, '4 BHK': {'building_type': 'Apartment', 'area_type': 'Super Built-up Area', 'area': '2,248 - 2,670 sq.ft.', 'price-range': '₹ 3.08 - 3.73 Cr'}}	['Bowling Alley', 'Mini Theatre', 'Manicured Garden', 'Swimming Pool', 'Flower Garden', 'Reading Lounge', 'Golf Course', 'Barbecue', 'Sauna']

```
app_df['PropertyName'] = app_df['PropertyName'].str.lower()

temp_df = df[df['features'].isnull()]
```

- We converted `PropertyName` into lower case
- Fetched the null values in `features` column

Merge the 2 datasets:

```
x = temp_df.merge(app_df,left_on='society',right_on='PropertyName',how='lef
t')['TopFacilities']
```

1. Merge Operation

- `temp_df.merge(app_df, ...)`:
  - This merges two DataFrames: `temp_df` (left DataFrame) and `app_df` (right DataFrame).
  - The result is a new DataFrame that combines rows from both DataFrames based on a common key.

2. Specifying the Key Columns

- `left_on='society'`:
  - This specifies the column in the **left DataFrame (`temp_df`)** that will be used as the key for the merge.
  - Here, the column `'society'` in `temp_df` is used as the key.
- `right_on='PropertyName'`:
  - This specifies the column in the **right DataFrame (`app_df`)** that will be used as the key for the merge.
  - Here, the column `'PropertyName'` in `app_df` is used as the key.

3. Type of Merge

- `how='left'`:
  - This specifies the type of merge to perform. A **left merge** means:
    - All rows from the **left DataFrame (`temp_df`)** will be included in the result.
    - Only rows from the **right DataFrame (`app_df`)** that match the key in the left DataFrame will be included.

    - If there is no match in the right DataFrame, the columns from the right DataFrame will contain `NaN` (missing values).

4. Selecting a Column After the Merge

- `['TopFacilities']`:
    - After the merge, the resulting DataFrame will have all columns from both `temp_df` and `app_df`.
    - `['TopFacilities']` selects only the `'TopFacilities'` column from the merged DataFrame.
    - This column is assigned to the variable `x`.

```
df.loc[temp_df.index,'features'] = x.values
```

1. **`df.loc[temp_df.index, 'features']`**:
    - Selects the `'features'` column in `df` for the rows that match the index of `temp_df`.
2. **`x.values`**:
    - `x` pandas Series or a list-like object.
    - `.values` extracts the actual data (as a NumPy array) from `x`. It's not a column; it's just the raw values inside `x`.
3. **Assignment (`=`)**:
    - The values from `x` are assigned to the selected rows in the `'features'` column of `df`.

Export to csv:

```
df.to_csv('gurgaon_properties_cleaned_v2.csv',index=False)
```