Capstone Project (Recommender System)

```
import numpy as np
import pandas as pd
import re
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
```

- `TfidfVectorizer`: Converting text into numerical representations (TF-IDF) that capture word importance.
- `cosine_similarity`: Measuring how similar two text documents are based on their TF-IDF representations

TF-IDF Vectorization:

- Converts text documents into numerical vectors using **Term Frequency-Inverse Document Frequency (TF-IDF)**.
- Each document is represented as a vector where each dimension corresponds to the importance of a word in the document.

1. **Cosine Similarity**:
    - Computes the similarity between pairs of documents based on their TF-IDF vectors.
    - Cosine similarity ranges from `-1` (completely dissimilar) to `1` (identical).

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

# Example text data
documents = [
```

```
    "I love machine learning",
    "Machine learning is fun",
    "I enjoy coding in Python"
]

# Step 1: Convert text to TF-IDF vectors
vectorizer = TfidfVectorizer()
tfidf_matrix = vectorizer.fit_transform(documents)

# Step 2: Compute cosine similarity between documents
similarity_matrix = cosine_similarity(tfidf_matrix)

print(similarity_matrix)
```

```
[[1.         0.44333251 0.        ]
 [0.44333251 1.         0.        ]
 [0.         0.         1.        ]]
```

- Diagonal values are `1` because each document is identical to itself.

```
df = pd.read_csv('appartments.csv').drop(22)
```

`.drop(22)`: Removes the row at index 22 from the DataFrame `df`.

22	PropertyName	PropertySubName	NearbyLocations	LocationAdvantages	Link	PriceDetails	TopFacilities

```
df.iloc[2].NearbyLocations

Output:
"['AIPL Business Club Sector 62', 'Heritage Xperiential Learning School', 'CK B
irla Hospital', 'Paras Trinity Mall Sector 63', 'Rapid Metro Station Sector 56']"
```

```
df.iloc[2].LocationAdvantages

Output:
"{'AIPL Business Club Sector 62': '2.7 Km', 'Heritage Xperiential Learning Sch
ool': '2 Km', 'CK Birla Hospital': '2.5 Km', 'Paras Trinity Mall Sector 63': '3.5 K
m', 'Rapid Metro Station Sector 56': '3.8 Km', 'De Adventure Park': '6.8 Km', 'G
olf Course Ext Rd': '99 Meter', 'DoubleTree by Hilton Hotel Gurgaon': '3.6 Km',
'KIIT College of Engineering Sohna Road': '8.4 Km', 'Mehrauli-Gurgaon Road':
'11.8 Km', 'Indira Gandhi International Airport': '21.1 Km', 'Nirvana Rd': '160 Met
er', 'TERI Golf Course': '8.7 Km'}"
```

- The locations in `NearbyLocations` are present in `LocationAdvantages` column
    - `NearbyLocations` is a subset of `LocationAdvantages`
- **Therefore, we don't need the column `NearbyLocations`**

```
df.iloc[1].PriceDetails

Output:
"{'3 BHK': {'building_type': 'Apartment', 'area_type': 'Super Built-up Area', 'are
a': '1,605 - 2,170 sq.ft.', 'price-range': '₹ 2.2 - 3.03 Cr'}, '4 BHK': {'building_typ
e': 'Apartment', 'area_type': 'Super Built-up Area', 'area': '2,248 - 2,670 sq.ft.',
'price-range': '₹ 3.08 - 3.73 Cr'}}"
```

```
df.iloc[0].TopFacilities

Output:
"['Swimming Pool', 'Salon', 'Restaurant', 'Spa', 'Cafeteria', 'Sun Deck', '24×7 S
ecurity', 'Club House', 'Gated Community']"
```

Recommendation Strategy:

- We will build 3 Recommendation Systems on the columns:
    - `LocationAdvantages`
    - `PriceDetails`
    - `TopFacilities`
- We'll combine them & display the results
- Assign weights to each

Process:

- Convert the `list` → into `str`
- Vectorize it
- Find out the 5 nearest vectors
- Recommend these👆

Recommendation system on the basis of Facilities

Convert str to list:

```
"['Swimming Pool', 'Salon', 'Restaurant', 'Spa', 'Cafeteria', 'Sun Deck', '24x7 Security', 'Club House', 'Gated Community']"

['Swimming Pool', 'Salon', 'Restaurant', 'Spa', 'Cafeteria', 'Sun Deck', '24x7 Security', 'Club House', 'Gated Community']
```

```
def extract_list(s):
    return re.findall(r"'(.*?)'", s)

df['TopFacilities'] = df['TopFacilities'].apply(extract_list)
```

`re.findall()`

- `re.findall(pattern, string)` is a function in Python's `re` module that finds all non-overlapping matches of the pattern in the string and returns them as a list.
- In this case:
    - pattern = `r"'(.*?)'"`: The regular expression pattern to match.
    - `string = s`: The input string to search in.
- `re.findall()` will return a list of all matches found in `s`.
    - `'`: Matches a **single quote**.
    - `(.*?)`: This is the **capturing group**:
        - `.`: Matches any character except a newline.
        - `?`: Matches zero or more occurrences of the preceding element (`.`), but **non-greedily** (stops at the first closing quote).
    - `'`: Matches the closing **single quote**.

How It Works?

1. The regex looks for patterns that start with a single quote (`'`).
2. It captures all characters inside the quotes (non-greedily) until it encounters the next single quote (`'`).
3. The `re.findall` function returns all matches as a list.

```python
import re

s = "I have 'apple' and 'banana' in my 'cart'"
matches = re.findall(r"'(.*?)'", s)
print(matches)
```

- **Output:**

```
['apple', 'banana', 'cart']
```

```
df.head()
```

Link	PriceDetails	TopFacilities
orld-sec...	{'2 BHK': {'building_type': 'Apartment', 'area...	[Swimming Pool, Salon, Restaurant, Spa, Cafete...
own-l-g...	{'3 BHK': {'building_type': 'Apartment', 'area...	[Bowling Alley, Mini Theatre, Manicured Garden...
lani-a-v...	{'3 BHK': {'building_type': 'Independent Floor...	[Terrace Garden, Gazebo, Fountain, Amphitheatr...
city-08-...	{'2 BHK': {'building_type': 'Apartment', 'area...	[Swimming Pool, Volley Ball Court, Aerobics Ce...
ure-ty-...	{'2 BHK': {'building_type': 'Independent Floor...	[Mini Theatre, Doctor on Call, Concierge Servi...

- Now, it's a python list

Now convert it into a long `str` without comma:

```
df['FacilitiesStr'] = df['TopFacilities'].apply(' '.join)

df['FacilitiesStr'][0]

Output:
'Swimming Pool Salon Restaurant Spa Cafeteria Sun Deck 24×7 Security Club House Gated Community'
```

```
tfidf_vectorizer = TfidfVectorizer(stop_words='english', ngram_range=(1, 2))
```

stop_words='english'

- **Purpose**: Removes common English stop words (e.g., "the", "is", "and") from the text.
- **Why?** Stop words add little meaning to the text and can be safely removed to reduce the dimensionality of the data and improve model performance.

ngram_range=(1, 2)

- **Purpose**: Specifies the range of n-grams to be extracted from the text.
- **What are n-grams?**
  - **Unigrams (1-grams)**: Single words (e.g., "machine", "learning").
  - **Bigrams (2-grams)**: Pairs of consecutive words (e.g., "machine learning").
- **Why?** Using n-grams helps capture context and relationships between words, which can improve the model's understanding of the text.

  - **Unigrams (1-grams)**:
    - "I", "love", "machine", "learning", "is", "fun"
  - **Bigrams (2-grams)**:
    - "I love", "love machine", "machine learning", "learning is", "is fun"

Why Use `ngram_range=(1, 2)`?

- **Captures Context**:
  - Unigrams capture individual words.
  - Bigrams capture relationships between consecutive words (e.g., "machine learning" is more meaningful than just "machine" or "learning").
- **Improves Model Performance**:
  - Including bigrams can help the model understand phrases and context better.

Simplified Explanation

- `ngram_range=(1, 2)` means:
  - Extract **single words** and **pairs of consecutive words** from the text.

```
tfidf_matrix = tfidf_vectorizer.fit_transform(df['FacilitiesStr'])
```

```
tfidf_matrix.toarray().shape

(246, 953)
```

- `tfidf_matrix` is a collection of **246 vectors** in **953D space**.

```
tfidf_matrix.toarray()[0]
```

```
array([0.        , 0.        , 0.        , 0.18809342, 0.18809342,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.16873099, 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.27811185, 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
...
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        ])
```

💡 In **high dimension**, **angular distance**✅ is more accurate representation than Euclidean distance.

- We'll calculate cosine similarity

```
cosine_sim1 = cosine_similarity(tfidf_matrix, tfidf_matrix)
```

Why Pass `tfidf_matrix` Twice?

- The line `cosine_sim1 = cosine_similarity(tfidf_matrix, tfidf_matrix)` passes `tfidf_matrix` as both the `X` and `Y` arguments to `cosine_similarity`.
- **Reason**: You want to compute the cosine similarity between all pairs of documents in `tfidf_matrix`.
  - In other words, you're comparing each property description with every other property description (including itself).

What Happens When `Y` is Not Specified?

- The `cosine_similarity` function has a default behavior: if `Y=None`, it sets `Y=X`. So:

python

```python
cosine_sim1 = cosine_similarity(tfidf_matrix, tfidf_matrix)
```

is equivalent to:

python

```python
cosine_sim1 = cosine_similarity(tfidf_matrix)
```

- This means `cosine_similarity` will compute the similarity between each pair of rows in `tfidf_matrix`.

```
cosine_sim1.shape

(246, 246)
```

Make a function:

```python
def recommend_properties(property_name, cosine_sim=cosine_sim):
    # Get the index of the property that matches the name
    idx = df.index[df['PropertyName'] == property_name].tolist()[0]

    # Get the pairwise similarity scores with that property
    sim_scores = list(enumerate(cosine_sim1[idx]))
```

```
# Sort the properties based on the similarity scores
sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

# Get the scores of the 10 most similar properties
sim_scores = sim_scores[1:6]

# Get the property indices
property_indices = [i[0] for i in sim_scores]

recommendations_df = pd.DataFrame({
    'PropertyName': df['PropertyName'].iloc[property_indices],
    'SimilarityScore': sim_scores
})

# Return the top 10 most similar properties
return recommendations_df
```

```
recommend_properties("DLF The Arbour")
```

	PropertyName	SimilarityScore
64	Ace Palm Floors	(63, 0.4529382062441955)
217	Yashika 104	(216, 0.4199606322926784)
93	JMS The Nation	(92, 0.4166584649363288)
154	India Rashtra	(153, 0.398954234680194)
0	Smartworld One DXP	(0, 0.38885046199432893)

- These are the 5 properties similar to **DLF The Arbour** on the basis of **facilities**.

This was the recommendation system on the basis of facilities.

Recommendation system on the basis of PriceDetails

```
df[['PropertyName','PriceDetails']]['PriceDetails'][1]
```

```
"{'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'}}"
```

- We will make separate columns for `building_type`, `area_type`, `area`, etc.

building_type	area_type	Area
Apartment	Super Built-up Area	1500

- We'll convert the categorical variables → Num by **OHE**
- In this was, we'll get vectors.
- Plot the vectors in high dimensions
- Then we'll find similarity of each vector with every other vector.
- We'll get an array (246, 246)
    - This array will have all the similarity scores
- On the basis of this array, we can do the recommendation

```
import pandas as pd
import json

# Load the dataset
df_appartments = pd.read_csv('appartments.csv')

# Function to parse and extract the required features from the PriceDetails col
```

```
umn
def refined_parse_modified_v2(detail_str):
    try:
        details = json.loads(detail_str.replace("'", "\""))
    except:
        return {}

    extracted = {}
    for bhk, detail in details.items():
        # Extract building type
        extracted[f'building type_{bhk}'] = detail.get('building_type')

        # Parsing area details
        area = detail.get('area', '')
        area_parts = area.split('-')
        if len(area_parts) == 1:
            try:
                value = float(area_parts[0].replace(',', '').replace(' sq.ft.', '').strip())
                extracted[f'area low {bhk}'] = value
                extracted[f'area high {bhk}'] = value
            except:
                extracted[f'area low {bhk}'] = None
                extracted[f'area high {bhk}'] = None
        elif len(area_parts) == 2:
            try:
                extracted[f'area low {bhk}'] = float(area_parts[0].replace(',', '').repla
ce(' sq.ft.', '').strip())
                extracted[f'area high {bhk}'] = float(area_parts[1].replace(',', '').repla
ce(' sq.ft.', '').strip())
            except:
                extracted[f'area low {bhk}'] = None
                extracted[f'area high {bhk}'] = None

        # Parsing price details
        price_range = detail.get('price-range', '')
        price_parts = price_range.split('-')
```

```
        if len(price_parts) == 2:
            try:
                extracted[f'price low {bhk}'] = float(price_parts[0].replace('₹', '').rep
lace(' Cr', '').replace(' L', '').strip())
                extracted[f'price high {bhk}'] = float(price_parts[1].replace('₹', '').re
place(' Cr', '').replace(' L', '').strip())
                if 'L' in price_parts[0]:
                    extracted[f'price low {bhk}'] /= 100
                if 'L' in price_parts[1]:
                    extracted[f'price high {bhk}'] /= 100
            except:
                extracted[f'price low {bhk}'] = None
                extracted[f'price high {bhk}'] = None

    return extracted
# Apply the refined parsing and generate the new DataFrame structure
data_refined = []

for _, row in df_appartments.iterrows():
    features = refined_parse_modified_v2(row['PriceDetails'])

    # Construct a new row for the transformed dataframe
    new_row = {'PropertyName': row['PropertyName']}

    # Populate the new row with extracted features
    for config in ['1 BHK', '2 BHK', '3 BHK', '4 BHK', '5 BHK', '6 BHK', '1 RK', 'Lan
d']:
        new_row[f'building type_{config}'] = features.get(f'building type_{confi
g}')
        new_row[f'area low {config}'] = features.get(f'area low {config}')
        new_row[f'area high {config}'] = features.get(f'area high {config}')
        new_row[f'price low {config}'] = features.get(f'price low {config}')
        new_row[f'price high {config}'] = features.get(f'price high {config}')

    data_refined.append(new_row)
```

```
df_final_refined_v2 = pd.DataFrame(data_refined).set_index('PropertyName')
```

```
df_final_refined_v2['building type_Land'] = df_final_refined_v2['building type_L
and'].replace({'':'Land'})
```

```
df['PriceDetails'][10]
```

"{'2 BHK': {'building_type': 'Independent Floor', 'area_type': 'Carpet Area', 'area': '1,055 sq.ft.', 'price-range': '₹ 1.05 - 1.5 Cr'}, '3 BHK': {'building_type': 'Independent Floor', 'area_type': 'Carpet Area', 'area': '1,325 - 1,525 sq.ft.', 'price-range': '₹ 1.35 - 1.84 Cr'}}"

```
df_final_refined_v2
```

PropertyName	building type_1 BHK	area low 1 BHK	area high 1 BHK	price low 1 BHK	price high 1 BHK	building type_2 BHK	area low 2 BHK	area high 2 BHK	price low 2 BHK	price high 2 BHK	...	building type_1 RK	area low 1 RK	area high 1 RK	price low 1 RK	price high 1 RK	building type_Land	area low Land	area high Land	pri lo Lar
Smartworld One DXP	None	NaN	NaN	NaN	NaN	Apartment	1370.0	1370.0	2.0000	2.40	...	None	NaN	NaN	NaN	NaN	None	NaN	NaN	Na
M3M Crown	None	NaN	NaN	NaN	NaN	None	NaN	NaN	NaN	NaN	...	None	NaN	NaN	NaN	NaN	None	NaN	NaN	Na
Adani Brahma Samsara Vilasa	None	NaN	NaN	NaN	NaN	None	NaN	NaN	NaN	NaN	...	None	NaN	NaN	NaN	NaN	Land	500.0	4329.0	2.
Sobha City	None	NaN	NaN	NaN	NaN	Apartment	1381.0	1692.0	1.5500	3.21	...	None	NaN	NaN	NaN	NaN	None	NaN	NaN	Na
Signature Global City 93	None	NaN	NaN	NaN	NaN	Independent Floor	981.0	1118.0	0.9301	1.06	...	None	NaN	NaN	NaN	NaN	None	NaN	NaN	Na
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
DLF Princeton Estate	None	NaN	NaN	NaN	NaN	Apartment	964.0	964.0	NaN	NaN	...	None	NaN	NaN	NaN	NaN	None	NaN	NaN	Na
Pyramid Urban Homes 2	Apartment	335.0	398.0	23.45	0.2786	Apartment	500.0	625.0	NaN	NaN	...	None	NaN	NaN	NaN	NaN	None	NaN	NaN	Na
Satya The Hermitage	None	NaN	NaN	NaN	NaN	Apartment	1450.0	1450.0	NaN	NaN	...	None	NaN	NaN	NaN	NaN	None	NaN	NaN	Na
BPTP Spacio	None	NaN	NaN	NaN	NaN	Apartment	1000.0	1079.0	NaN	NaN	...	None	NaN	NaN	NaN	NaN	None	NaN	NaN	Na
SS The Coralwood	None	NaN	NaN	NaN	NaN	Apartment	1320.0	1425.0	NaN	NaN	...	None	NaN	NaN	NaN	NaN	None	NaN	NaN	Na

Extract CAT columns for OHE

```
categorical_columns = df_final_refined_v2.select_dtypes(include=['object']).columns.tolist()

categorical_columns
```

```
['building type_1 BHK',
 'building type_2 BHK',
 'building type_3 BHK',
 'building type_4 BHK',
 'building type_5 BHK',
 'building type_6 BHK',
 'building type_1 RK',
 'building type_Land']
```

```
ohe_df = pd.get_dummies(df_final_refined_v2, columns=categorical_columns, drop_first=True)
```

```
ohe_df.fillna(0,inplace=True)

ohe_df
```

	area low 1 BHK	area high 1 BHK	price low 1 BHK	price high 1 BHK	area low 2 BHK	area high 2 BHK	price low 2 BHK	price high 2 BHK	area low 3 BHK	area high 3 BHK	...	building type_2 BHK_Independent Floor	building type_2 BHK_Service Apartment	building type_3 BHK_Independent Floor	building type_3 BHK_Service Apartment	building type_3 BHK_Villa
PropertyName																
Smartworld One DXP	0.0	0.0	0.00	0.0000	1370.0	1370.0	2.0000	2.40	1850.0	2050.0	...	False	False	False	False	False
M3M Crown	0.0	0.0	0.00	0.0000	0.0	0.0	0.0000	0.00	1605.0	2170.0	...	False	False	False	False	False
Adani Brahma Samsara Vilasa	0.0	0.0	0.00	0.0000	0.0	0.0	0.0000	0.00	1800.0	3150.0	...	False	False	True	False	False
Sobha City	0.0	0.0	0.00	0.0000	1381.0	1692.0	1.5500	3.21	1711.0	2343.0	...	False	False	False	False	False
Signature Global City 93	0.0	0.0	0.00	0.0000	981.0	1118.0	0.9301	1.06	1235.0	1530.0	...	True	False	True	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
DLF Princeton Estate	0.0	0.0	0.00	0.0000	964.0	964.0	0.0000	0.00	1127.0	1127.0	...	False	False	False	False	False
Pyramid Urban Homes 2	335.0	398.0	23.45	0.2786	500.0	625.0	0.0000	0.00	0.0	0.0	...	False	False	False	False	False
Satya The Hermitage	0.0	0.0	0.00	0.0000	1450.0	1450.0	0.0000	0.00	1991.0	1991.0	...	False	False	False	False	False
BPTP Spacio	0.0	0.0	0.00	0.0000	1000.0	1079.0	0.0000	0.00	1225.0	1865.0	...	False	False	False	False	False
SS The Coralwood	0.0	0.0	0.00	0.0000	1320.0	1425.0	0.0000	0.00	1750.0	1890.0	...	False	False	False	False	False

Scale the data:

```
from sklearn.preprocessing import StandardScaler

# Initialize the scaler
scaler = StandardScaler()

# Apply the scaler to the entire dataframe
ohe_df_normalized = pd.DataFrame(scaler.fit_transform(ohe_df), columns=oh
e_df.columns, index=ohe_df.index)
```

```
ohe_df_normalized.head()
```

PropertyName	area low 1 BHK	area high 1 BHK	price low 1 BHK	price high 1 BHK	area low 2 BHK	area high 2 BHK	price low 2 BHK	price high 2 BHK	area low 3 BHK	area high 3 BHK	...	building type_2 BHK_Independent Floor	building type_2 BHK_Service Apartment	B
Smartworld One DXP	-0.251723	-0.169231	-0.104981	-0.082164	1.227625	1.024144	-0.172887	1.161995	0.558393	0.375750	...	-0.288675	-0.063758	
M3M Crown	-0.251723	-0.169231	-0.104981	-0.082164	-0.890292	-0.893391	-0.282926	-0.387081	0.298215	0.477455	...	-0.288675	-0.063758	
Adani Brahma Samsara Vilasa	-0.251723	-0.169231	-0.104981	-0.082164	-0.890292	-0.893391	-0.282926	-0.387081	0.505295	1.308051	...	-0.288675	-0.063758	
Sobha City	-0.251723	-0.169231	-0.104981	-0.082164	1.244631	1.474835	-0.197645	1.684808	0.410782	0.624081	...	-0.288675	-0.063758	
Signature Global City 93	-0.251723	-0.169231	-0.104981	-0.082164	0.626260	0.671430	-0.231752	0.297094	-0.094708	-0.064975	...	3.464102	-0.063758	

Now find out the cosine similarity:

```
from sklearn.metrics.pairwise import cosine_similarity

# Compute the cosine similarity matrix
cosine_sim2 = cosine_similarity(ohe_df_normalized)
```

```
cosine_sim2.shape

(246, 246)
```

Make a recommendation system:

```
def recommend_properties_with_scores(property_name, top_n=5):

    # Get the similarity scores for the property using its name as the index
    sim_scores = list(enumerate(cosine_sim2[ohe_df_normalized.index.get_loc
(property_name)]))

    # Sort properties based on the similarity scores
    sorted_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    # Get the indices and scores of the top_n most similar properties
    top_indices = [i[0] for i in sorted_scores[1:top_n+1]]
    top_scores = [i[1] for i in sorted_scores[1:top_n+1]]

    # Retrieve the names of the top properties using the indices
    top_properties = ohe_df_normalized.index[top_indices].tolist()

    # Create a dataframe with the results
    recommendations_df = pd.DataFrame({
        'PropertyName': top_properties,
        'SimilarityScore': top_scores
    })

    return recommendations_df

# Test the recommender function using a property name
recommend_properties_with_scores('M3M Golf Hills')
```

	PropertyName	SimilarityScore
0	AIPL The Peaceful Homes	0.955462
1	Smartworld One DXP	0.954670
2	Unitech Escape	0.953092
3	M3M Capital	0.951156
4	BPTP Terra	0.943128

Recommendation system on the basis of LocationAdvantages

```
df[['PropertyName','LocationAdvantages']]
```

✓ 0.0s

	PropertyName	LocationAdvantages
0	Smartworld One DXP	{'Bajghera Road': '800 Meter', 'Palam Vihar Ha...
1	M3M Crown	{'DPSG Palam Vihar Gurugram': '1.4 Km', 'The N...
2	Adani Brahma Samsara Vilasa	{'AIPL Business Club Sector 62': '2.7 Km', 'He...
3	Sobha City	{'The Shikshiyan School': '2.9 KM', 'WTC Plaza...
4	Signature Global City 93	{'Pranavananda Int. School': '450 m', 'DLF Sit...
...	...	...
242	DLF Princeton Estate	{'Sector 42-43 Metro Station': '1.8 Km', 'Para...
243	Pyramid Urban Homes 2	{'Aarvy Healthcare Super Speciality': '1.8 KM'...
244	Satya The Hermitage	{'Dwarka Expressway': '1.2 Km', 'S N Internati...
245	BPTP Spacio	{'Suncity School': '0.2 Km', 'Gurugram Road': ...
246	SS The Coralwood	{'Sector 84 Road': '600 Meter', 'Delhi Public ...

246 rows × 2 columns

```
df[['PropertyName','LocationAdvantages']]['LocationAdvantages'][0]
```

```
Output:
"{'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'}"
```

- We'll make each landmark a column
- Go to each apartment and Ask: ***How are you from this landmark?***

	L1	L2	L3	. . .	Ln
A1	0-8	2.5	3.1		Null
A2	0.7	Null	—		—
A3					
⋮					

```
def distance_to_meters(distance_str):
    try:
        if 'Km' in distance_str or 'KM' in distance_str:
            return float(distance_str.split()[0]) * 1000
        elif 'Meter' in distance_str or 'meter' in distance_str:
            return float(distance_str.split()[0])
        else:
            return None
```

```
    except:
        return None
```

```
# Extract distances for each location

import ast
location_matrix = {}
for index, row in df.iterrows():
    distances = {}
    for location, distance in ast.literal_eval(row['LocationAdvantages']).items():
        distances[location] = distance_to_meters(distance)
    location_matrix[index] = distances

# Convert the dictionary to a dataframe
location_df = pd.DataFrame.from_dict(location_matrix, orient='index')

# Display the first few rows
location_df.head()
```

	Bajghera Road	Palam Vihar Halt	DPSG Palam Vihar	Park Hospital	Gurgaon Railway Station	The NorthCap University	Dwarka Expy	Hyatt Place Gurgaon Udyog Vihar	Dwar Sect 2 Met Statio
0	800.0	2500.0	3100.0	3100.0	4900.0	5400.0	1200.0	7700.0	7200
25	550.0	NaN	NaN	NaN	NaN	6700.0	3800.0	NaN	Na
37	5300.0	NaN	NaN	NaN	2500.0	8800.0	NaN	NaN	Na
69	1500.0	NaN	NaN	NaN	6500.0	6700.0	5100.0	NaN	Na
9	NaN	NaN	NaN	5500.0	NaN	NaN	NaN	NaN	Na

5 rows × 1070 columns

- Fill NaN values

```
location_df.fillna(54000,inplace=True)
```

- We just replace NaN with the biggest value because we cannot fill it with 0
- If we fill it with 0, it's meaning will be: "The distance of the place from apartment/flat is zero."

Scale:

```
from sklearn.preprocessing import StandardScaler
# Initialize the scaler
scaler = StandardScaler()

# Apply the scaler to the entire dataframe
location_df_normalized = pd.DataFrame(scaler.fit_transform(location_df), colu
mns=location_df.columns, index=location_df.index)
```

```
location_df_normalized
```

	Bajghera Road	Palam Vihar Halt	DPSG Palam Vihar	Park Hospital	Gurgaon Railway Station	The NorthCap University	Dwarka Expy	Hyatt Place Gurgaon Udyog Vihar	Dwarka Sector 21, Metro Station	Pacific D21 Mall	...	MCC Cricket Ground Dhankot	The Shri Ram School Aravali	Taj City Centre Gurugram
PropertyName														
Smartworld One DXP	-7.960979	-15.652476	-15.652476	-3.149592	-2.966108	-3.147217	-3.726615	-10.231739	-15.652476	-6.023233	...	0.0	0.063888	0.063888
M3M Crown	-7.998993	0.063888	0.063888	0.328277	0.368941	-3.054053	-3.529275	0.090308	0.063888	-6.009941	...	0.0	0.063888	0.063888
Adani Brahma Samsara Vilasa	-7.276720	0.063888	0.063888	0.328277	-3.129124	-2.903557	0.280891	0.090308	0.063888	0.171073	...	0.0	0.063888	0.063888
Sobha City	-7.854539	0.063888	0.063888	0.328277	-2.857430	-3.054053	-3.430606	0.090308	0.063888	-5.916893	...	0.0	0.063888	0.063888
Signature Global City 93	0.128476	0.063888	0.063888	-2.985606	0.368941	0.335688	0.280891	0.090308	0.063888	0.171073	...	0.0	0.063888	0.063888
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
DLF Princeton Estate	0.128476	0.063888	0.063888	0.328277	0.368941	0.335688	0.280891	0.090308	0.063888	0.171073	...	0.0	0.063888	0.063888
Pyramid Urban Homes 2	0.128476	0.063888	0.063888	0.328277	0.368941	0.335688	0.280891	0.090308	0.063888	0.171073	...	0.0	0.063888	0.063888
Satya The Hermitage	0.128476	0.063888	0.063888	0.328277	0.368941	0.335688	0.280891	0.090308	0.063888	0.171073	...	0.0	0.063888	0.063888

Calculate cosine similarity:

```
cosine_sim3 = cosine_similarity(location_df_normalized)
cosine_sim3.shape

Output:
(246, 246)
```

```
def recommend_properties_with_scores(property_name, top_n=5):

    cosine_sim_matrix = 30*cosine_sim1 + 20*cosine_sim2 + 8*cosine_sim3
    # cosine_sim_matrix = cosine_sim3

    # Get the similarity scores for the property using its name as the index
    sim_scores = list(enumerate(cosine_sim_matrix[location_df_normalized.inde
x.get_loc(property_name)]))

    # Sort properties based on the similarity scores
    sorted_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    # Get the indices and scores of the top_n most similar properties
    top_indices = [i[0] for i in sorted_scores[1:top_n+1]]
    top_scores = [i[1] for i in sorted_scores[1:top_n+1]]

    # Retrieve the names of the top properties using the indices
    top_properties = location_df_normalized.index[top_indices].tolist()

    # Create a dataframe with the results
    recommendations_df = pd.DataFrame({
        'PropertyName': top_properties,
        'SimilarityScore': top_scores
    })
```

```
    return recommendations_df

# Test the recommender function using a property name
recommend_properties_with_scores('Ireo Victory Valley')
```

	PropertyName	SimilarityScore
0	Pioneer Urban Presidia	28.021460
1	Ambience Creacions	27.787913
2	DLF The Crest	24.205986
3	Pioneer Araya	23.415308
4	Silverglades The Melia	21.007840

```
(3*cosine_sim3 + 5*cosine_sim2 + 6*cosine_sim1).shape

Output:
(246, 246)
```

👆Meaning: We are giving more weightage to `cosine_sim1`

Export Pickle (.pkl)

```
import pickle

pickle.dump(location_df, open('location_distance.pkl', 'wb'))
pickle.dump(cosine_sim1, open('cosine_sim1.pkl', 'wb'))
pickle.dump(cosine_sim2, open('cosine_sim2.pkl', 'wb'))
pickle.dump(cosine_sim3, open('cosine_sim3.pkl', 'wb'))
```