Data Mining Project

Marine George Zarif 20221443430

Carine Emad Sanad 20221440878

Eng\Yassmina Ayman

Comparative Analysis of K-medoids and Hierarchical Clustering for House Pricing Dataset

In this project, we explore two popular clustering algorithms, K-medoids and hierarchical clustering, to analyze a house pricing dataset. Clustering is a fundamental unsupervised learning technique used to group similar data points together based on their characteristics. The aim of this project is to compare the performance of these two clustering algorithms in segmenting houses into distinct price categories.

Dataset Description: The house pricing dataset used in this project contains various features related to house characteristics such as area, number of bedrooms, city, level, etc., along with the corresponding prices.

Methodology:

1. Download data "Egypt House Pricing"

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

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

```
df.head()
```

	Type	Price	Bedrooms	Bathrooms	Area	Furnished	Level	Compound	Payment_Option	Delivery_Date	Delivery_Term	City
0	Duplex	4000000	3	3	400	No	7	Unknown	Cash	Ready to move	Finished	Nasr City
1	Apartment	4000000	3	3	160	No	10+	Unknown	Cash	Ready to move	Finished	Camp Caesar
2	Apartment	2250000	3	2	165	No	1	Unknown	Cash	Ready to move	Finished	Smoha
3	Apartment	1900000	3	2	230	No	10	Unknown	Cash	Ready to move	Finished	Nasr City
4	Apartment	5800000	2	3	160	No	Ground	Eastown	Cash	Ready to move	Semi Finished	New Cairo - El Tagamoa

Data Preprocessing:

We removed rows with missing values and duplicate entries to ensure data cleanliness and integrity. This step involved checking for missing values in each column and dropping rows with null values. Additionally, duplicate rows were identified and eliminated to prevent redundancy in the dataset.

Fixing the Dtype for the columns

```
[ ] df.info()

<class 'pandas.core.frame.DataFrame'>
Index: 4528 entries, 4 to 18448
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Type            4528 non-null   object
 1   Price           4528 non-null   object
 2   Bedrooms        4528 non-null   object
 3   Bathrooms       4528 non-null   object
 4   Area            4528 non-null   object
 5   Furnished       4528 non-null   object
 6   Level           4528 non-null   object
 7   Compound        4528 non-null   object
 8   Payment_Option  4528 non-null   object
 9   Delivery_Date   4528 non-null   object
 10  Delivery_Term   4528 non-null   object
 11  City            4528 non-null   object
dtypes: object(12)
memory usage: 459.9+ KB
```

```
[ ] df['Bedrooms'] = df['Bedrooms'].astype(float).astype(int)
    df['Bathrooms'] = df['Bathrooms'].astype(float).astype(int)
    df['Area'] = df['Area'].astype(float).astype(int)
    df['Price'] = df['Price'].astype(float).astype(int)
```

```
[ ] df.dropna(inplace=True)
```

```
[ ] df.duplicated().sum()

377
```

```
[ ] df.drop_duplicates(inplace=True)
```

```
df.shape

(4528, 12)
```

Drop null and duplicates

```
[ ] df['Type'] = df['Type'].replace('Unknown', np.nan)
    df['Furnished'] = df['Furnished'].replace('Unknown', np.nan)
    df['Price'] = df['Price'].replace('Unknown', np.nan)
    df['Level'] = df['Level'].replace('Unknown', np.nan)
    df['Compound'] = df['Compound'].replace('Unknown', np.nan)
    df['Area'] = df['Area'].replace('Unknown', np.nan)
    df['Bedrooms'] = df['Bedrooms'].replace('Unknown', np.nan)
    df['Bathrooms'] = df['Bathrooms'].replace('Unknown', np.nan)
    df['Payment_Option'] = df['Payment_Option'].replace('Unknown Payment', np.nan)
    df['Delivery_Date'] = df['Delivery_Date'].replace('Unknown', np.nan)
    df['Delivery_Term'] = df['Delivery_Term'].replace('Unknown', np.nan)
    df['City'] = df['City'].replace('Unknown', np.nan)
```

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

Type                 36
Price                39
Bedrooms            239
Bathrooms           207
Area                507
Furnished          8528
Level             10439
Compound          11068
Payment_Option     3048
Delivery_Date     10108
Delivery_Term      4706
City                  0
dtype: int64
```

Show executed code history

```
print(df['Type'].unique())
print(df['Level'].unique())
```

```
['Apartment' 'Penthouse' 'Duplex' 'Studio' 'Chalet' 'Standalone Villa'
 'Twin house' 'Town House']
['Ground' '1' '2' '3' '4' '9' 'Highest' '5' '8' '10' '10+' '7' '6']
```

```
df.loc[(df['Level']=='10+'),'Level'] = 11
df.loc[(df['Level']=='Highest'),'Level'] = 12
df.loc[(df['Level']=='Ground'),'Level'] = 0
```

```
df.reset_index(inplace=True)
df.drop(['index'],axis=1,inplace=True)
```

```
df.head()
```

	Type	Price	Bedrooms	Bathrooms	Area	Furnished	Level	Compound	Payment_Option	Delivery_Date	Delivery_Term	City
0	Apartment	5800000	2	3	160	No	0	Eastown	Cash	Ready to move	Semi Finished	New Cairo - El Tagamoa
1	Apartment	1844900	4	3	222	No	1	Beit Al Watan	Cash or Installment	2024	Semi Finished	New Cairo - El Tagamoa
2	Apartment	309825	4	3	153	No	1	Beit Al Watan	Cash or Installment	2024	Semi Finished	New Cairo - El Tagamoa
3	Apartment	2350000	3	3	178	No	2	La Mirada	Cash	Ready to move	Finished	New Cairo - El Tagamoa
4	Apartment	1050000	3	1	108	No	3	Maadi V	Cash or Installment	2023	Semi Finished	Zahraa Al Maadi

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4528 entries, 0 to 4527
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Type            4528 non-null   object
 1   Price           4528 non-null   int64
 2   Bedrooms        4528 non-null   int64
 3   Bathrooms       4528 non-null   int64
 4   Area            4528 non-null   int64
 5   Furnished       4528 non-null   object
 6   Level           4528 non-null   object
 7   Compound        4528 non-null   object
 8   Payment_Option  4528 non-null   object
 9   Delivery_Date   4528 non-null   object
 10  Delivery_Term   4528 non-null   object
 11  City            4528 non-null   object
dtypes: int64(4), object(8)
memory usage: 424.6+ KB
```

```
df.describe()
```

	Price	Bedrooms	Bathrooms	Area
count	4.528000e+03	4528.000000	4528.000000	4528.000000
mean	3.435122e+06	2.773189	2.345406	162.810292
std	3.721786e+06	0.858444	0.956277	86.802356
min	3.000000e+04	1.000000	1.000000	12.000000
25%	1.550000e+06	2.000000	2.000000	112.000000
50%	2.575500e+06	3.000000	2.000000	145.000000
75%	4.000000e+06	3.000000	3.000000	189.000000
max	6.500000e+07	9.000000	10.000000	950.000000

Visualization of Preprocessed Data:

- After preprocessing, we visualized the preprocessed data to gain insights into the distribution and relationships between features.
- Visualization techniques such as scatter plots, histograms, or pair plots were employed to explore the data's characteristics and identify any patterns or outliers.

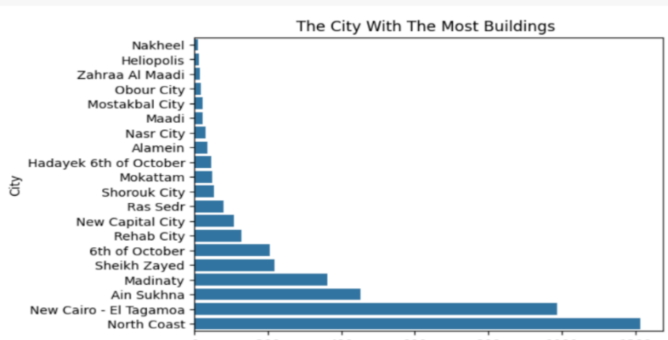
```
[ ] plt.subplots(figsize=(24, 128))
    sns.boxplot(y='City', x='Price',data=df)
```

<Axes: xlabel='Price', ylabel='City'>

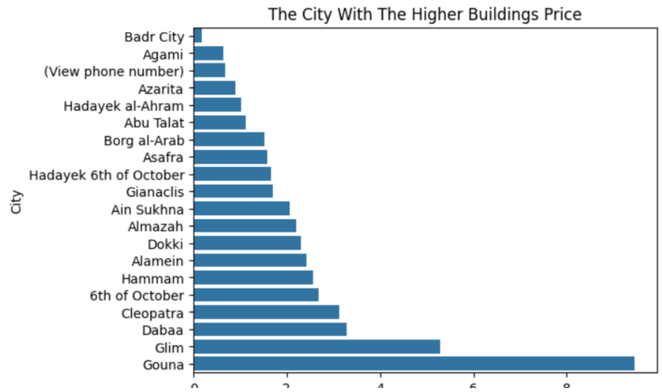

```
[ ] lcc = df['City'].value_counts().keys().tolist()
```

```
[ ] for x in lcc:
        Q1= df[(df['City']==x)]['Price'].quantile(0.25)
        Q3= df[(df['City']==x)]['Price'].quantile(0.75)
        IQR = Q3 - Q1
        upper_bound = Q3 + 1.2 * IQR
        lower_bound = Q1 - 1.2 * IQR
        df=df.drop(df[(df['City']==x)&(df['Price']>=upper_bound)].index)
        df=df.drop(df[(df['City']==x)&(df['Price']<=lower_bound)].index)
```

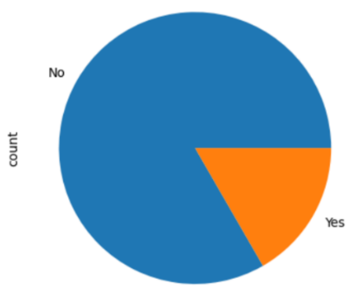
```
[ ] mp = df['City'].value_counts()[0:20].sort_values()
    sns.barplot(y=mp.index,x=mp.values);
    plt.title('The City With The Most Buildings ');
```



```python
lpm = df.groupby('City')['Price'].mean()[0:20].sort_values()
sns.barplot(y=lpm.index,x=lpm.values);
plt.title('The City With The Higher Buildings Price');
```

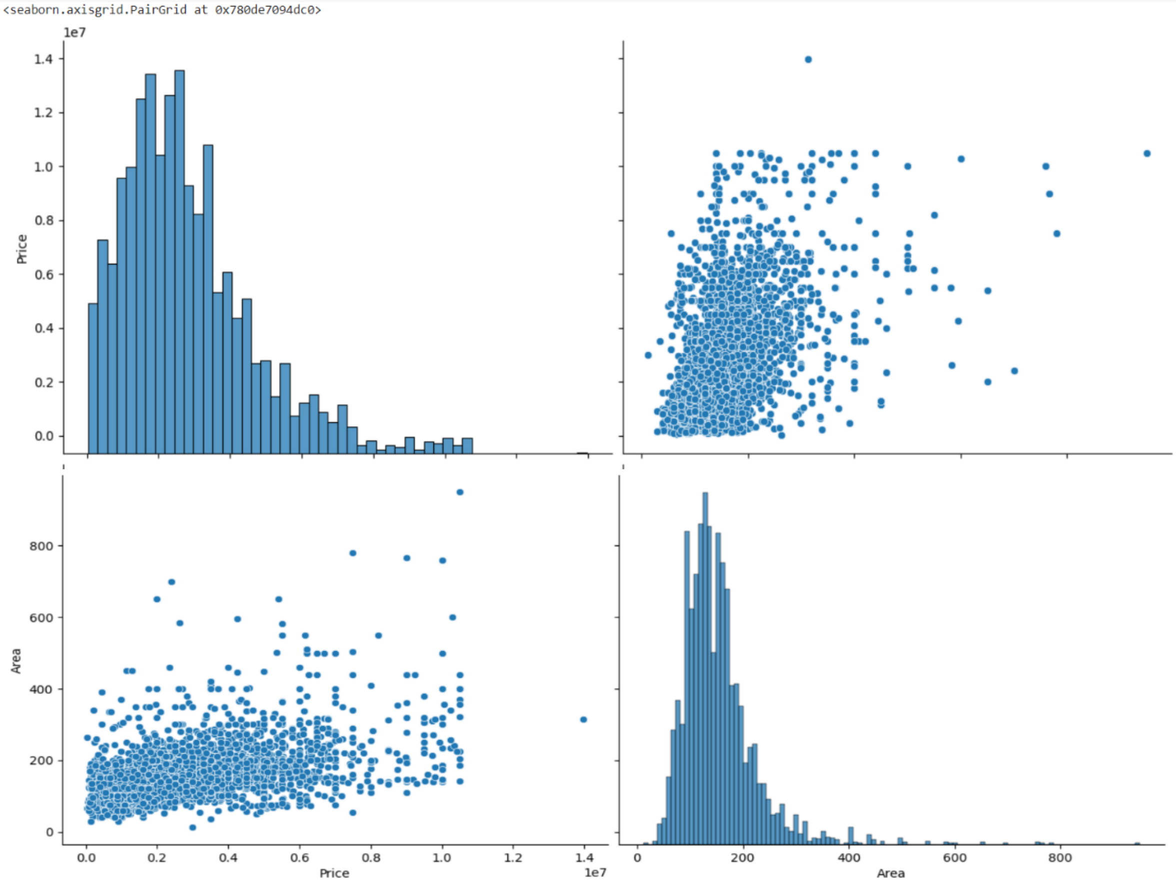



```python
df['Furnished'].value_counts().plot(kind='pie');
```

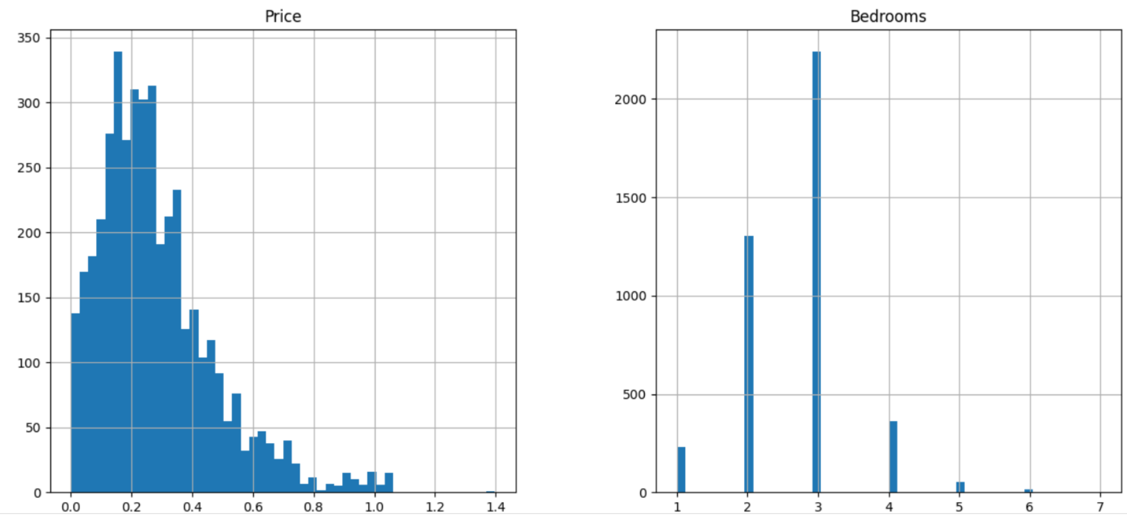


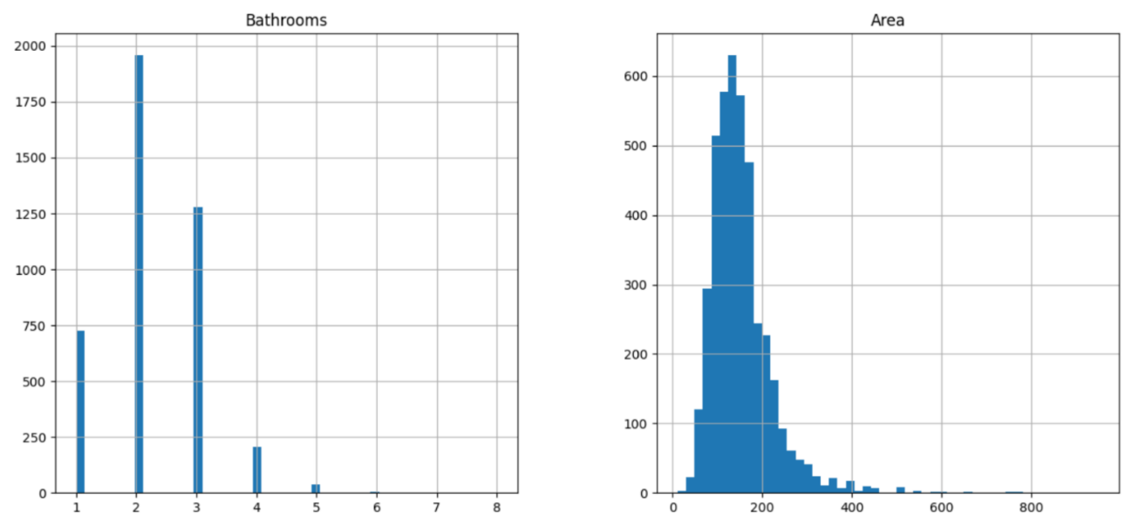

```python
sns.pairplot(df, vars = ['Price', 'Area'], height=5, aspect=1.3)
```

<seaborn.axisgrid.PairGrid at 0x780de7094dc0>



```
[ ] df.hist(bins=50, figsize=(15, 15));
```



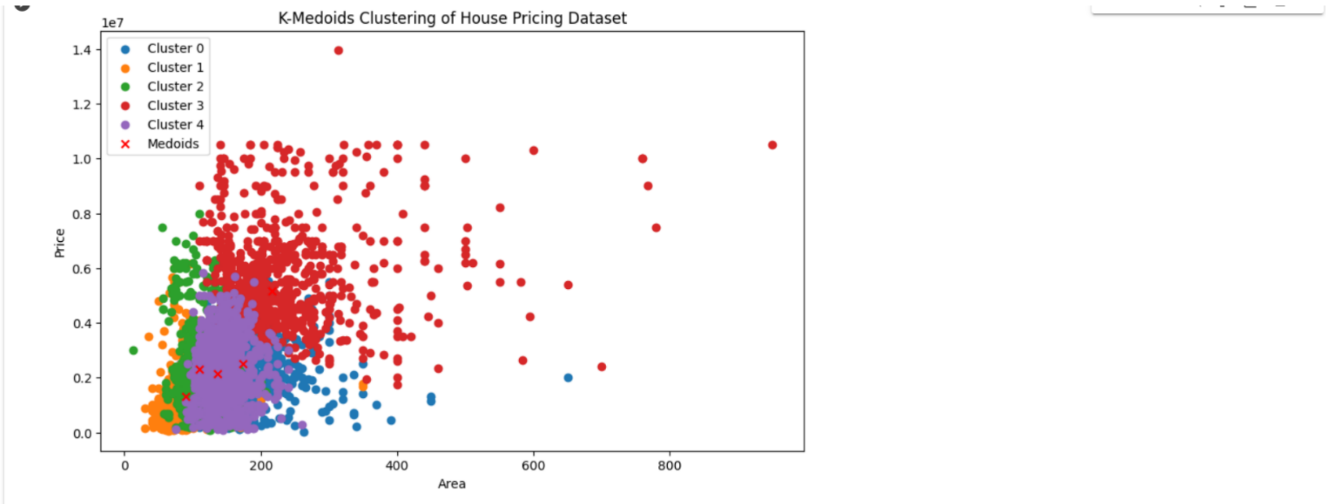


- **K-medoids Clustering:**
  - We applied the K-medoids algorithm to the preprocessed dataset to cluster houses into K distinct groups based on their characteristics.
  - The number of clusters (K) was determined using techniques like the elbow method or silhouette analysis.
  - Visualization: We visualized the clustering results using scatter plots, where each point represents a house, colored according to its assigned cluster. Additionally, we marked the medoids of each cluster for better interpretation.

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn_extra.cluster import KMedoids
from sklearn.preprocessing import StandardScaler

# Assuming 'df' is your DataFrame containing the dataset
X = df[['Area', 'Price', 'Bedrooms', 'Bathrooms', 'Level']]

# Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

n_clusters = 5
kmedoids = KMedoids(n_clusters=n_clusters, random_state=0).fit(X_scaled)

# Add cluster labels to the original DataFrame
df['cluster'] = kmedoids.labels_

# Print the cluster centers (medoids)
medoids = scaler.inverse_transform(kmedoids.cluster_centers_)
print("Cluster medoids:")
print(pd.DataFrame(medoids, columns=X.columns))

# Print the cluster sizes
print("Cluster sizes:")
print(df['cluster'].value_counts())

# Plotting the clusters
plt.figure(figsize=(10, 6))
for cluster_label in range(n_clusters):
    cluster_data = X[df['cluster'] == cluster_label]
```

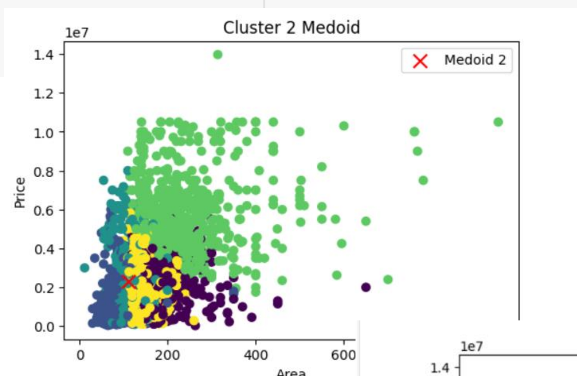
```
    plt.scatter(cluster_data['Area'], cluster_data['Price'], label=f'Cluster {cluster_label}')

# Plotting the medoids
plt.scatter(medoids[:, 0], medoids[:, 1], c='red', marker='x', label='Medoids')

plt.xlabel('Area')
plt.ylabel('Price')
plt.title('K-Medoids Clustering of House Pricing Dataset')
plt.legend()
plt.show()
```

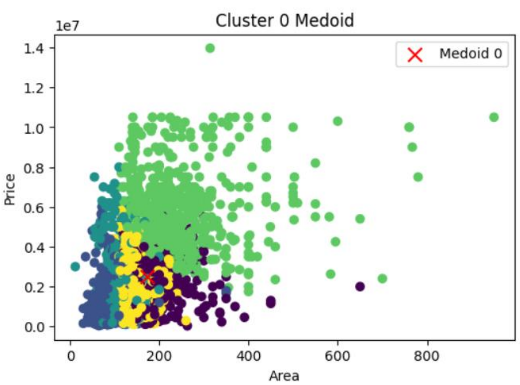
```
Cluster medoids:
    Area      Price  Bedrooms  Bathrooms  Level
0  174.0  2500000.0       3.0        3.0    2.0
1   90.0  1300000.0       2.0        1.0    2.0
2  110.0  2310000.0       2.0        2.0    1.0
3  217.0  5150000.0       3.0        3.0    1.0
4  137.0  2150000.0       3.0        2.0    2.0
Cluster sizes:
cluster
4    1104
0     848
2     810
3     777
1     669
Name: count, dtype: int64
```

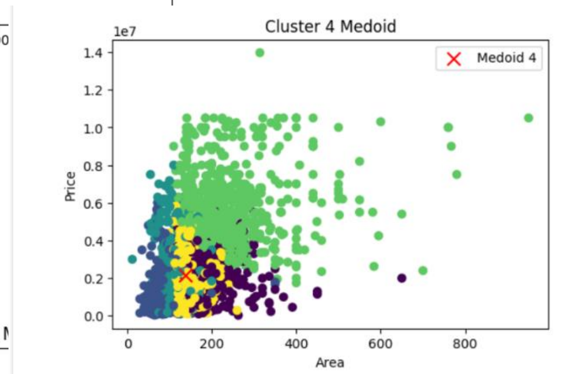


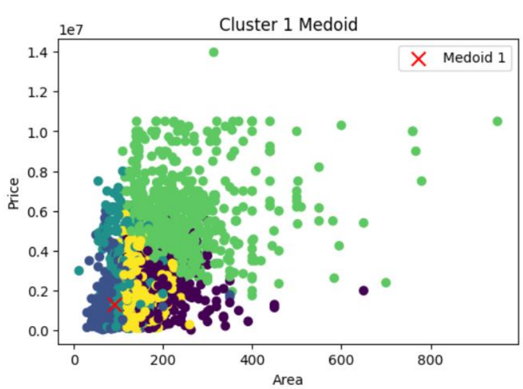
Another plotting:

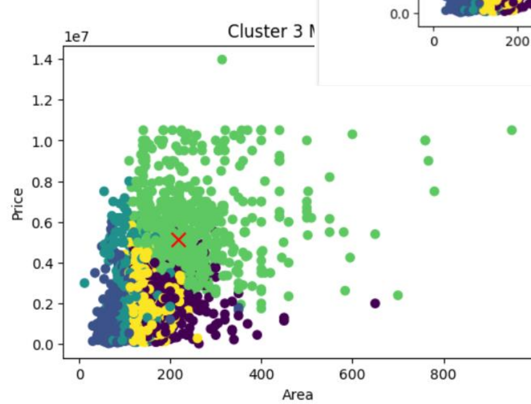
```python
# Plotting each medoid alone
for i, medoid in enumerate(medoids):
    plt.figure(figsize=(6, 4))
    plt.scatter(X['Area'], X['Price'], c=df['cluster'], cmap='viridis')
    plt.scatter(medoid[0], medoid[1], c='red', marker='x', label=f'Medoid {i}', s=100)
    plt.xlabel('Area')
    plt.ylabel('Price')
    plt.title(f'Cluster {i} Medoid')
    plt.legend()
    plt.show()
```







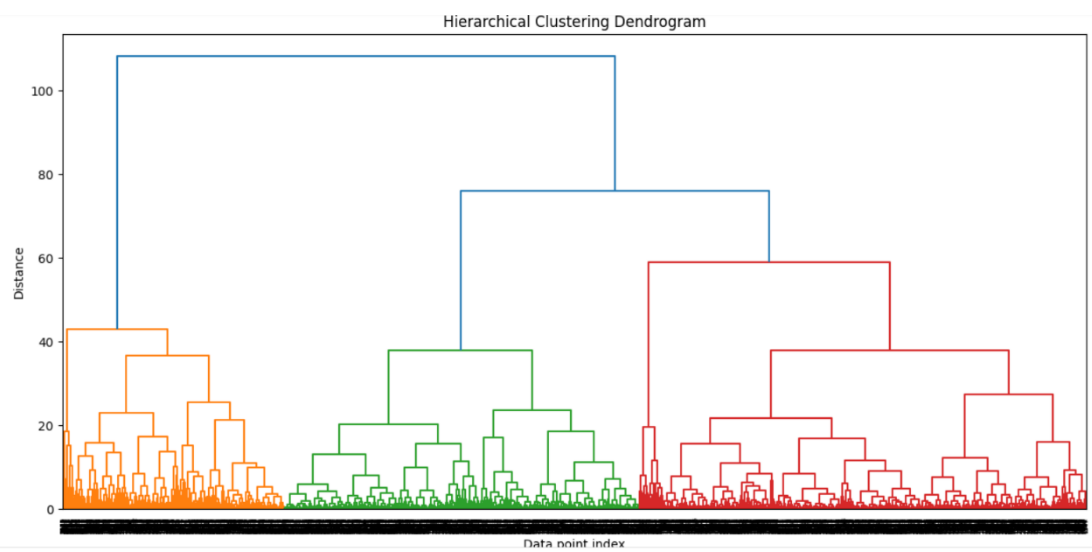




- **Hierarchical Clustering:**
  - We also employed hierarchical clustering, a bottom-up approach that forms a hierarchy of clusters, to segment the houses based on their similarity.
  - Different linkage methods such as complete, average, and ward linkage were explored to measure the distance between clusters.
  - Visualization: We visualized the hierarchical clustering dendrogram, illustrating the hierarchical structure of the clusters. Additionally, we created dendrograms for each linkage method to compare the clustering results visually.

Hierarchical clustering

```
from sklearn.preprocessing import StandardScaler
from scipy.cluster.hierarchy import dendrogram, linkage

# Assuming 'df' is your DataFrame containing the dataset
X = df[['Area', 'Price', 'Bedrooms', 'Bathrooms', 'Level']]

# Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Perform hierarchical clustering
Z = linkage(X_scaled, method='ward')

# Plot the dendrogram
plt.figure(figsize=(15, 7))
dendrogram(Z)
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('Data point index')
plt.ylabel('Distance')
plt.show()
```



Evaluation:

- We evaluated the clustering results of both algorithms using metrics such as silhouette score, Davies–Bouldin index, and visual inspection of cluster separation.
- The silhouette score measures the cohesion and separation of clusters, with higher scores indicating better-defined clusters.
- The Davies–Bouldin index assesses

```
from sklearn.metrics import silhouette_score, davies_bouldin_score

def evaluate_clustering(clustering_algorithm, data):
    # Fit clustering algorithm
    cluster_labels = clustering_algorithm.fit_predict(data)

    # Silhouette score
    silhouette_avg = silhouette_score(data, cluster_labels)

    # Davies-Bouldin index
    db_index = davies_bouldin_score(data, cluster_labels)

    return silhouette_avg, db_index

# Example usage:
# Assuming 'kmedoids' is the K-medoids clustering algorithm and 'hierarchical' is the hierarchical clustering algorithm
# Assuming 'scaled_features' is the preprocessed and scaled feature data

# Evaluate K-medoids clustering
kmedoids_silhouette, kmedoids_db = evaluate_clustering(kmedoids, scaled_features)
print("K-medoids Silhouette Score:", kmedoids_silhouette)
print("K-medoids Davies-Bouldin Index:", kmedoids_db)

# Evaluate Hierarchical clustering
hierarchical_silhouette, hierarchical_db = evaluate_clustering(hierarchical, scaled_features)
print("\nHierarchical Silhouette Score:", hierarchical_silhouette)
print("Hierarchical Davies-Bouldin Index:", hierarchical_db)
```

Notebook Link :

https://colab.research.google.com/drive/15qBezbJfrh4F8cdN6ZFYPap-PGmfIzVU?usp=sharing

Conclusion:

- In conclusion, we compared the effectiveness of K-medoids and hierarchical clustering algorithms in segmenting a house pricing dataset.
- Both algorithms provided valuable insights into the underlying structure of the dataset, enabling us to identify distinct groups of houses based on their characteristics and prices.
- The choice of clustering algorithm depends on factors such as dataset size, structure, and the desired level of interpretability.