Capstone Project (Data Visualization)

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

```
df = pd.read_csv('gurgaon_properties_missing_value_imputation.csv')
df.head()
```

	property_type	society	sector	price	price_per_sqft	bedRoom	bathroom	balcony	floorNum	agePos
0	flat	signature global park 4	sector 36	0.82	7586.0	3.0	2.0	2	2.0	New
1	flat	smart world gems	sector 89	0.95	8597.0	2.0	2.0	2	4.0	New
2	flat	breez global hill view	sohna road	0.32	5470.0	2.0	2.0	1	17.0	New
3	flat	bestech park view sanskruti	sector 92	1.60	8020.0	3.0	4.0	3+	10.0	Relativ
4	flat	suncity avenue	sector 102	0.48	9023.0	2.0	2.0	1	5.0	Relativ

- Data with `price_per_sqft` column

```
latlong = pd.read_csv('latlong.csv')
latlong
```

	sector	coordinates
0	sector 1	28.3663° N, 76.9456° E
1	sector 2	28.5095° N, 77.0320° E
2	sector 3	28.4909° N, 77.0176° E
3	sector 4	28.4738° N, 77.0107° E
4	sector 5	28.4794° N, 77.0176° E
...	...	...
124	sector 113	28.5287° N, 77.0233° E
125	sector 114	28.5334° N, 77.0118° E
126	sector 115	28.5385° N, 77.0061° E
127	gwal pahari	28.4484° N, 77.0210° E
128	manesar	28.3515° N, 76.9428° E

- Coordinates of each sector

- Now separate the **latitude & longitude**

```
latlong['latitude'] = latlong['coordinates'].str.split(',').str.get(0).str.split('°').str.get(0).astype('float')

latlong['longitude'] = latlong['coordinates'].str.split(',').str.get(1).str.split('°').str.get(0).astype('float')

latlong.head()
```

	sector	coordinates	latitude	longitude
0	sector 1	28.3663° N, 76.9456° E	28.3663	76.9456
1	sector 2	28.5095° N, 77.0320° E	28.5095	77.0320
2	sector 3	28.4909° N, 77.0176° E	28.4909	77.0176
3	sector 4	28.4738° N, 77.0107° E	28.4738	77.0107
4	sector 5	28.4794° N, 77.0176° E	28.4794	77.0176

- Merge this with `df`

```
new_df = df.merge(latlong, on='sector')
```

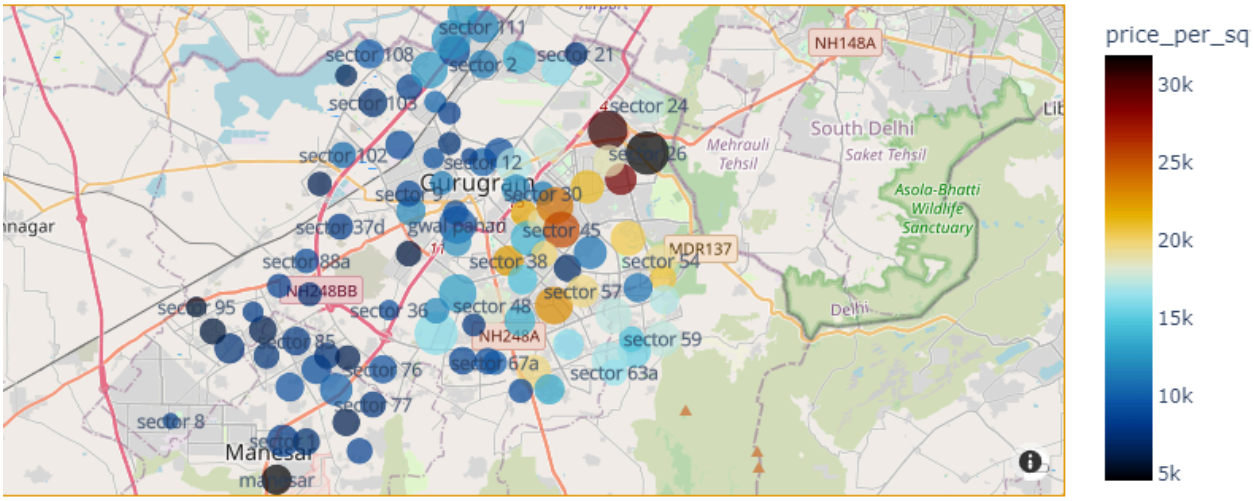
```
group_df = new_df.groupby('sector')[['price','price_per_sqft','built_up_area','latitude','longitude']].mean()

group_df
```

sector	price	price_per_sqft	built_up_area	latitude	longitude
gwal pahari	3.192222	9585.777778	3056.166667	28.4484	77.0210
manesar	0.962258	4608.064516	2027.367742	28.3515	76.9428
sector 1	1.860000	8249.833333	2327.833333	28.3663	76.9456
sector 10	2.092857	11866.571429	1908.857143	28.4537	77.0009
sector 102	1.696636	10603.822430	1556.130841	28.4750	76.9715
...	...	...	...	...	...
sector 91	1.648235	7586.117647	2028.647059	28.4014	76.9225
sector 92	0.934000	5928.290000	1571.341800	28.4079	76.9153
sector 93	0.848889	8009.888889	1017.000000	28.4153	76.9326
sector 95	0.480545	5602.509091	995.981818	28.4172	76.9081
sector 99	1.008095	6412.166667	1364.214286	28.4640	76.9614

Plot a graph

```
fig = px.scatter_mapbox(group_df, lat="latitude", lon="longitude", color="price
_per_sqft", size='built_up_area',
                        color_continuous_scale=px.colors.cyclical.IceFire, zoom=10,
                        mapbox_style="open-street-map",text=group_df.index)
fig.show()
```



Parameter	Explanation
`px.scatter_mapbox()`	Creates a scatter plot on an interactive Mapbox map.
`group_df`	DataFrame containing location and real estate data.
`lat="latitude"`	Specifies the latitude column for plotting points.
`lon="longitude"`	Specifies the longitude column for plotting points.
`color="price_per_sqft"`	Colors the points based on price per square foot (continuous scale).
`size="built_up_area"`	Sizes the points based on built-up area (larger homes → bigger dots).
`color_continuous_scale=px.colors.cyclical.IceFire`	Uses a cyclical color scale (blue to red) for price differences.
`zoom=10`	Initial zoom level of the map.
`mapbox_style="open-street-map"`	Uses OpenStreetMap for the map's background style.

Parameter	Explanation
`text=group_df.index`	Displays index values when hovering over points.
`fig.show()`	Displays the interactive plot.

Alternative Map Styles (`mapbox_style` Options)

Style Name	Description
`"open-street-map"`	Default OpenStreetMap tiles
`"carto-positron"`	Light-colored map with labels
`"carto-darkmatter"`	Dark-themed map
`"stamen-terrain"`	Topographic-style map
`"stamen-toner"`	High-contrast black & white map

Export:

```
new_df.to_csv('data_viz1.csv',index=False)
```

```
df1 = pd.read_csv('gurgaon_properties.csv')

df1.head()
```

	property_name	property_type	society	price	price_per_sqft	area	areaWithType	bedRoom	bathroom	balcony	additionalRoom	address	floor
0	3 BHK Flat in Sector 65 Gurgaon	flat	m3m heights	2.86	14000.0	2043.0	Carpet area: 2040 (189.52 sq.m.)	3	3	3	servant room	Sector 65 Gurgaon, Gurgaon, Haryana	
1	5 Bedroom House for sale in Sector 66 Gurgaon	house	emaar mgf marbella	19.00	31666.0	6000.0	Plot area 9000(836.13 sq.m.)Carpet area: 6000 ...	5	6	3+	pooja room,servant room,store room	Belinda 14, Sector 66 Gurgaon, Gurgaon, Haryana	
2	3 BHK Flat in Sector 37D Gurgaon	flat	ramprastha primera	1.08	6000.0	1800.0	Super Built up area 1800(167.23 sq.m.)Built Up...	3	3	3	not available	901, Sector 37D Gurgaon, Gurgaon, Haryana	
3	4 Bedroom House for sale in Malibu Town	house	independent	0.99	17187.0	576.0	Plot area 64(53.51 sq.m.)	4	4	3	not available	Malibu Town, Gurgaon, Haryana	

Merge `df` & `df1`

```
wordcloud_df = df1.merge(df, left_index=True, right_index=True)[['features','sector']]
wordcloud_df.head()
```

	features	sector
0	NaN	sector 36
1	['Feng Shui / Vaastu Compliant', 'Private Gard...	sector 89
2	['Centrally Air Conditioned', 'Water purifier'...	sohna road
3	['Maintenance Staff', 'Rain Water Harvesting']	sector 92
4	['Centrally Air Conditioned', 'Water purifier'...	sector 102

- Joins `df1` and `df` using their **index** as the key (instead of a column).
- Keeps only the matching rows from both DataFrames (default is an **inner join**).

merge(df)	Merges df1 with df
`left_index=True`	Uses df1's index for joining
`right_index=True`	Uses df's index for joining
`[['features', 'sector']]`	Selects only two columns

- If you **do not** specify `left_index=True, right_index=True`, pandas will assume that you want to merge **based on common column values** instead of index values.
- This will lead to different behavior depending on whether your DataFrames have a common column with the same name.

```
import ast
main = []
for item in wordcloud_df['features'].dropna().apply(ast.literal_eval):
    main.extend(item)
```

`wordcloud_df['features'].dropna()`

- `wordcloud_df['features']`: Selects the `features` column from `wordcloud_df`.
- `.dropna()`: Removes any rows where `features` is `NaN` (missing). This prevents errors when applying `ast.literal_eval`.

`ast.literal_eval(item)`: Converts a string that looks like a list into a real Python list.

- Example:

```python
item = "['house', 'garden', 'pool']"  # This is a string, not a real list
ast.literal_eval(item)  # Converts to a real list: ['house', 'garden', 'pool']
```

What does `.extend()` do?

- It **adds each individual word** from `item` into `main`.
- ✨Unlike `append()`, which would add the entire list as a single element, extend() flattens the list and adds each element individually.

- First iteration: `item = ['gym', 'pool']` → `main.extend(['gym', 'pool'])` → `main = ['gym', 'pool']`.
- Second iteration: `item = ['park', 'club']` → `main.extend(['park', 'club'])` → `main = ['gym', 'pool', 'park', 'club']`.

main

```
['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',
 'Bank Attached Property',
 'Piped-gas',
 'Visitor Parking',
 'Swimming Pool',
 'Park',
 'Security Personnel',
 'Internet/wi-fi connectivity',
 'Low Density Society',
 'Fitness Centre / GYM',
 'Waste Disposal',
 'Rain Water Harvesting',
```

- It’s a giant list.

WordCloud

```
from wordcloud import WordCloud
```

```
feature_text = ' '.join(main)
```

This code
joins all words in `main` **into a single long string**, where words are separated by a space.


```
import pickle
pickle.dump(feature_text, open('feature_text.pkl','wb'))
```

What is Pickle?

- Pickle is a Python module used for **serializing** (saving) and **deserializing** (loading) Python objects.

- It converts Python objects (lists, dictionaries, text, models, etc.) into a format that can be saved to a file and reloaded later.

- Here, `pickle` is used to save the `feature_text` object to a **file**.

python Copy Edit

```
pickle.dump(object, file)
```

- `feature_text`: The object you want to save (in this case, a large string of text).
- `open('feature_text.pkl', 'wb')`: Opens a file in **write-binary (`wb`) mode** for saving the object.

After this line executes, `feature_text` is saved as a binary file (`feature_text.pkl`).

```
plt.rcParams["font.family"] = "Arial"

wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = set(['s']),  # Any stopwords you'd like to exclude
                min_font_size = 10).generate(feature_text)

plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.tight_layout(pad = 0)
plt.show()
```


```
plt.rcParams["font.family"] = "Arial"

wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = set(['s']),  # Any stopwords you'd like to exclude
                min_font_size = 10).generate(feature_text)

plt.figure(figsize = (8, 8), facecolor = None)

plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.tight_layout(pad = 0)
plt.show()
```

- This sets the font for Matplotlib plots to "Arial".

- Ensures consistency in text rendering.

`stopwords=set(['s'])`: Specifies a set of words to exclude from the WordCloud. Here, it excludes the word "s". This is useful for removing common or irrelevant words

`.generate(feature_text)` :

- Generates the WordCloud from the feature_text string.
- Based on earlier code, `feature_text` is a space-separated string of features (e.g., `"gym pool park club ...`") created from a list of features:

`plt.tight_layout(pad=0)`: Removes extra spacing around the figure.

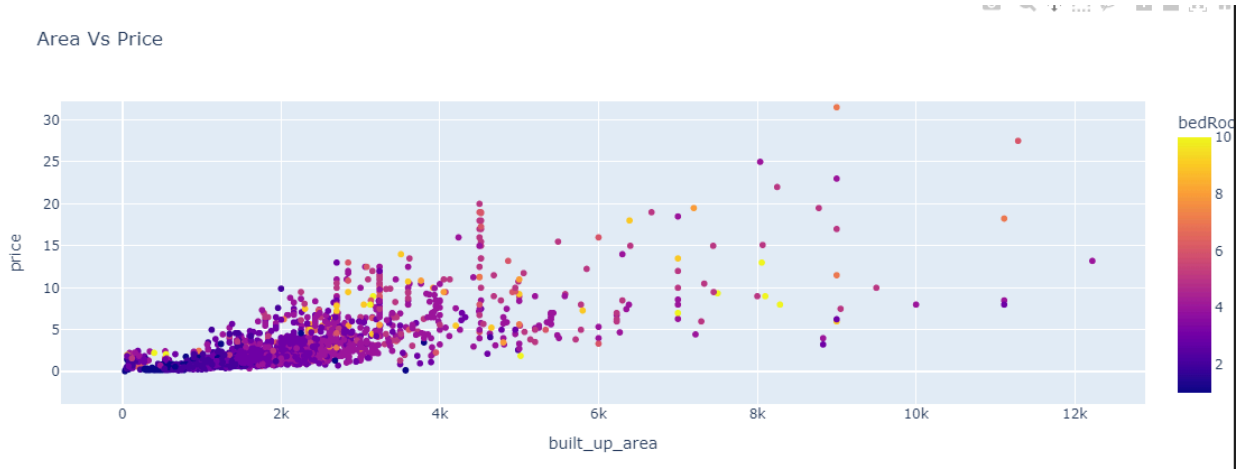
`plt.imshow(wordcloud, interpolation='bilinear'):`

- Displays the ***WordCloud*** image in the **Matplotlib** figure.
- `wordcloud` is a **WordCloud** object, but when passed to `plt.imshow()`, it's automatically converted to an **image** (a NumPy array representing the pixel data).
- `interpolation='bilinear'` applies bilinear interpolation to smooth the image, making the edges of the words less pixelated.

Plot Graphs:

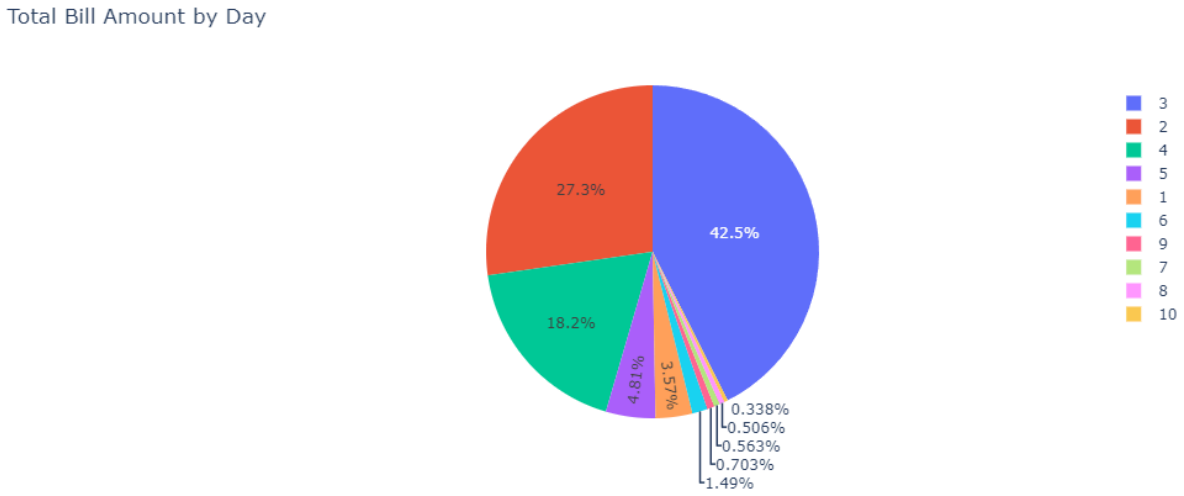
```
fig = px.scatter(df, x="built_up_area", y="price", color="bedRoom", title="Area Vs Price")

# Show the plot
fig.show()
```



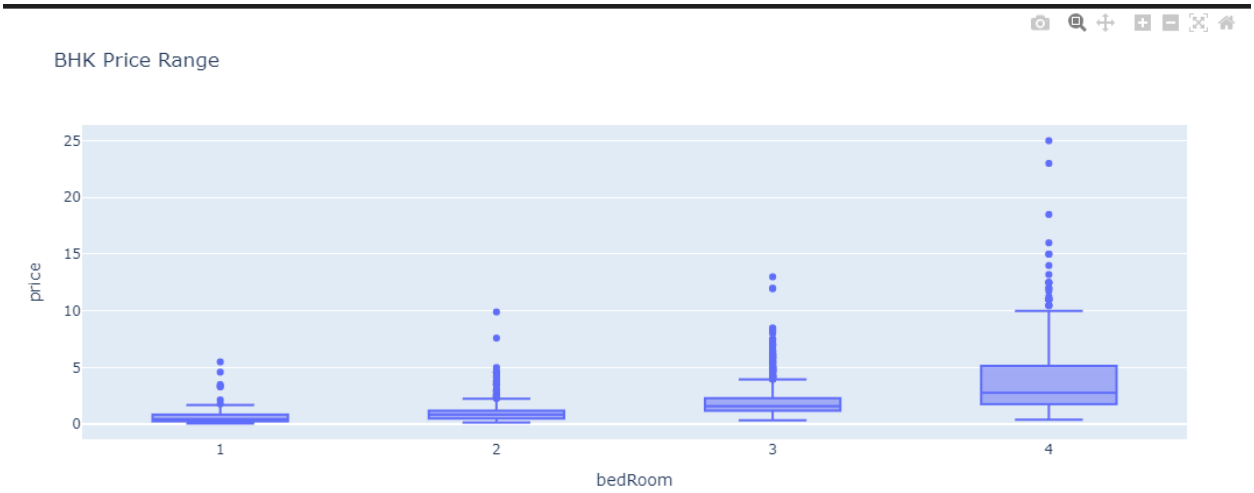

```
fig = px.pie(df, names='bedRoom', title='Total Bill Amount by Day')

# Show the plot
fig.show()
```

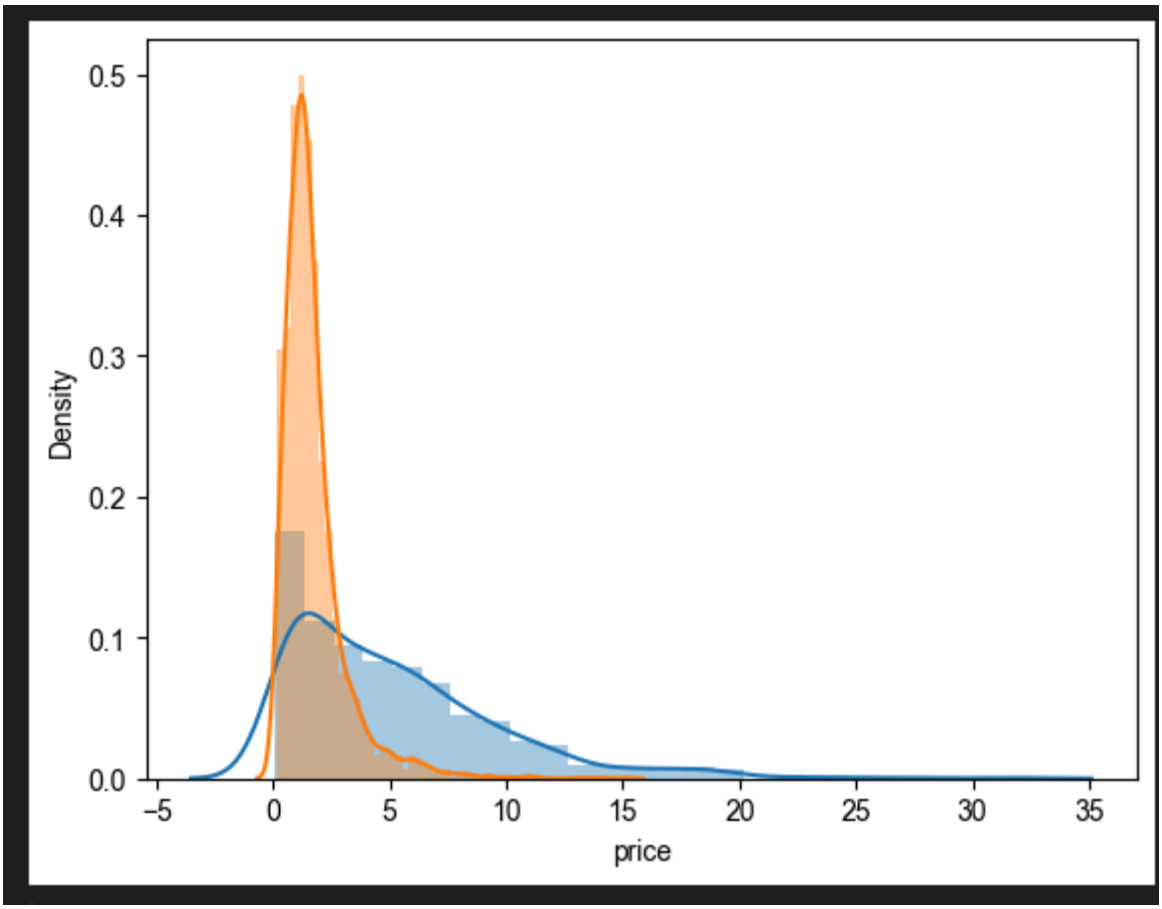



```
temp_df = df[df['bedRoom'] <= 4]
# Create side-by-side boxplots of the total bill amounts by day
fig = px.box(temp_df, x='bedRoom', y='price', title='BHK Price Range')
```

```
# Show the plot
fig.show()
```




```
sns.distplot(df[df['property_type'] == 'house']['price'])
sns.distplot(df[df['property_type'] == 'flat']['price'])
```


0.5
0.4
0.3
0.2
0.1
0.0
Density
−5
0
5
10
15
20
25
30
35
price