```
# Importing necessary libraries for data analysis and modeling
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Data Pre-processing

```
# Loading the dataset
data = pd.read_csv("/content/Bengaluru_House_Data.csv")
# Display the first few rows of the dataset to understand its structure
data.head()
```

	area_type	availability	location	size	society	total_sqft	bath	balcor
0	Super built-up Area	19-Dec	Electronic City Phase II	2 BHK	Coomee	1056	2.0	1
1	Plot Area	Ready To Move	Chikka Tirupathi	4 Bedroom	Theanmp	2600	5.0	3
2	Built-up	Ready To	Uttarahalli	3 BHK	NaN	1440	2.0	3

Next steps: Generate code with data View recommended plots New interactive sheet

```
# Checking the Shape of our dataset, number of rows and columns
data.shape
```

```
(13320, 9)
```

```
#Checking Data-type of each column/feature
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13320 entries, 0 to 13319
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   area_type     13320 non-null  object 
 1   availability  13320 non-null  object 
 2   location      13319 non-null  object 
 3   size          13304 non-null  object 
 4   society       7818 non-null   object 
 5   total_sqft    13320 non-null  object 
 6   bath          13247 non-null  float64
 7   balcony       12711 non-null  float64
 8   price         13320 non-null  float64
dtypes: float64(3), object(6)
memory usage: 936.7+ KB
```

```
#Dropping Columns which are not required
data = data.drop(columns=['society','availability'],axis=1)
```

```
data.head(1)
```

	area_type	location	size	total_sqft	bath	balcony	price
0	Super built-up Area	Electronic City Phase II	2 BHK	1056	2.0	1.0	39.07

Next steps: Generate code with data View recommended plots New interactive sheet

```
# The below function will simply remove the '-' from each row in total_sqft column and convert it's data type to float
def convert_sqft_to_num(x):
    tokens = x.split('-')
    if len(tokens) == 2:
        return (float(tokens[0])+float(tokens[1]))/2
    try:
        return float(x)
    except:
        return None
```

```
data.total_sqft = data.total_sqft.apply(convert_sqft_to_num) # Calling our function
data = data[data.total_sqft.notnull()]
data.head()
```

	area_type	location	size	total_sqft	bath	balcony	price
0	Super built-up Area	Electronic City Phase II	2 BHK	1056.0	2.0	1.0	39.07
1	Plot Area	Chikka Tirupathi	4 Bedroom	2600.0	5.0	3.0	120.00
2	Built-up Area	Uttarahalli	3 BHK	1440.0	2.0	3.0	62.00
3	Super built-up Area	Lingadheeranahalli	3 BHK	1521.0	3.0	1.0	95.00

Next steps: Generate code with data View recommended plots New interactive sheet

```
data.describe()
```

	total_sqft	bath	balcony	price
count	13274.000000	13201.000000	12669.000000	13274.000000
mean	1559.626694	2.691160	1.585682	112.453654
std	1238.405258	1.338867	0.816734	149.070368
min	1.000000	1.000000	0.000000	8.000000
25%	1100.000000	2.000000	1.000000	50.000000
50%	1276.000000	2.000000	2.000000	72.000000
75%	1680.000000	3.000000	2.000000	120.000000
max	52272.000000	40.000000	3.000000	3600.000000

here we can see

```
data.isnull().sum() # Checking for null values
```

```
area_type       0
location        1
size           16
total_sqft      0
bath           73
balcony       605
price           0
dtype: int64
```

```
#Data clean: handling null values
data = data.dropna()
```

```
data.area_type.unique()
```

```
array(['Super built-up Area', 'Plot Area', 'Built-up Area', 'Carpet Area'],
      dtype=object)
```

```
data.location.unique()
```

```
array(['Electronic City Phase II', 'Chikka Tirupathi', 'Uttarahalli', ...,
       '12th cross srinivas nagar banshankari 3rd stage',
       'Havanur extension', 'Abshot Layout'], dtype=object)
```

```
len(data.location.unique())
```

```
1259
```

```
location_count = data.groupby('location').size().sort_values(ascending=False) # Grouping Locations to get the count of each.
location_count
```

```
location
Whitefield              513
Sarjapur  Road          372
Electronic City         300
```

```
Kanakpura Road          259
Thanisandra             230
                       ...
Kalhalli                  1
Kalkere Channasandra      1
 Banaswadi                1
Kamdhenu Nagar            1
whitefiled                1
Length: 1259, dtype: int64
```

```
len(location_count[location_count<=10])
```

```
1024
```

```
location_less_10 = location_count[location_count<=10]
```

```
#if the location have less than 10 or 10 houses than that location are known as other location
data.location = data.location.apply(lambda x: 'other' if x in location_less_10 else x)
```

```
<ipython-input-18-bd7f53f11902>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-cc
  data.location = data.location.apply(lambda x: 'other' if x in location_less_10 else x)
```

```
len(data.location.unique())
```

```
236
```

```
data['size'].unique()
```

```
array(['2 BHK', '4 Bedroom', '3 BHK', '3 Bedroom', '1 BHK', '1 RK',
       '4 BHK', '1 Bedroom', '2 Bedroom', '6 Bedroom', '8 Bedroom',
       '7 Bedroom', '5 BHK', '7 BHK', '6 BHK', '5 Bedroom', '11 BHK',
       '9 BHK', '9 Bedroom', '27 BHK', '11 Bedroom', '43 Bedroom',
       '14 BHK', '8 BHK', '12 Bedroom', '10 Bedroom', '13 BHK'],
      dtype=object)
```

```
data['size'] = data['size'].apply(lambda x: int(x.split(' ')[0])) #Removing non-numeric values from size column
```

```
data['size'].unique
```

```
pandas.core.series.Series.unique
def unique() -> ArrayLike

/usr/local/lib/python3.10/dist-packages/pandas/core/series.py
Return unique values of Series object.

Uniques are returned in order of appearance. Hash table-based unique,
therefore does NOT sort.
```

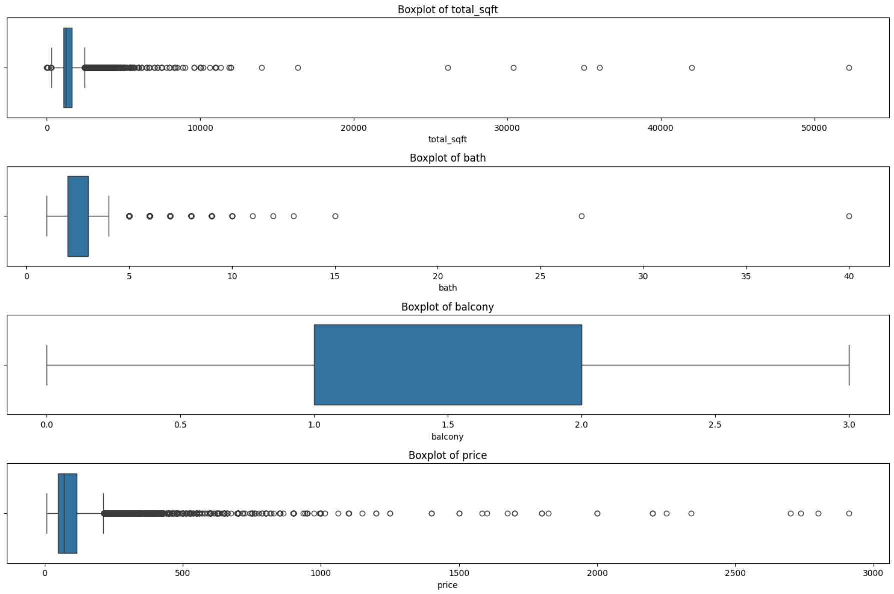
```
data.head(1)
```

	area_type	location	size	total_sqft	bath	balcony	price
0	Super built-up Area	Electronic City Phase II	2	1056.0	2.0	1.0	39.07

Next steps: Generate code with data | View recommended plots | New interactive sheet

```
import seaborn as sns
```

```
# List of numerical columns to visualize outliers
numeric_cols = ['total_sqft', 'bath', 'balcony', 'price']

# Set the plot size
plt.figure(figsize=(15, 10))

# Create boxplots for each numeric column
for i, col in enumerate(numeric_cols, 1):
    plt.subplot(len(numeric_cols), 1, i)
    sns.boxplot(x=data[col])
    plt.title(f'Boxplot of {col}')

# Adjust layout
plt.tight_layout()

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



```
#Removing outliers using standard deviation from price column
upper_limit = data.price.mean() + data.price.std()
lower_limit = data.price.mean() - data.price.std()
data = data[data.price<upper_limit]
data = data[data.price > lower_limit]
```

```
#Removing outliers using standard deviation from price column
upper_limit = data.total_sqft.mean() + data.total_sqft.std()
lower_limit = data.total_sqft.mean() - data.total_sqft.std()
data = data[data.total_sqft<upper_limit]
data = data[data.total_sqft > lower_limit]
```

```
data.head(1)
```

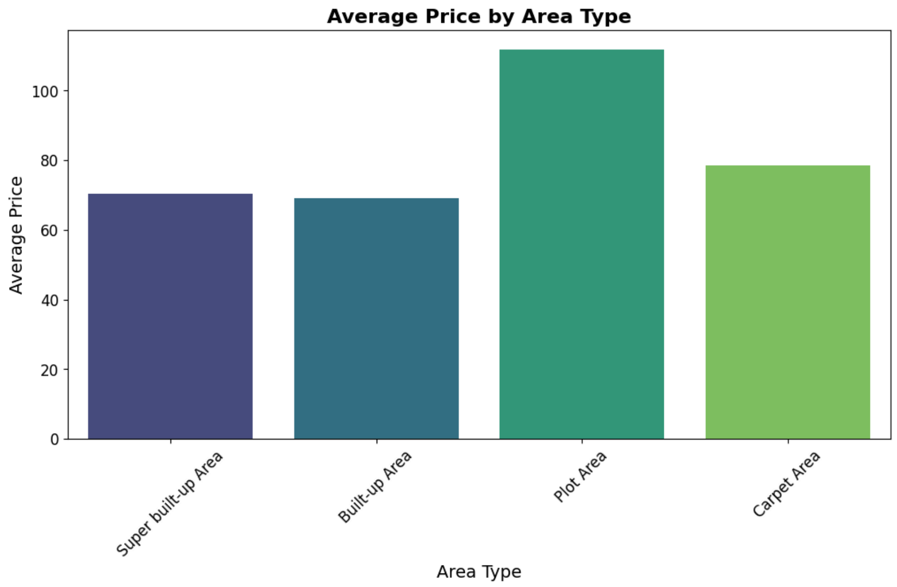
	area_type	location	size	total_sqft	bath	balcony	price
0	Super built-up Area	Electronic City Phase II	2	1056.0	2.0	1.0	39.07

Next steps: Generate code with data | View recommended plots | New interactive sheet

```
import warnings
warnings.filterwarnings("ignore")
```

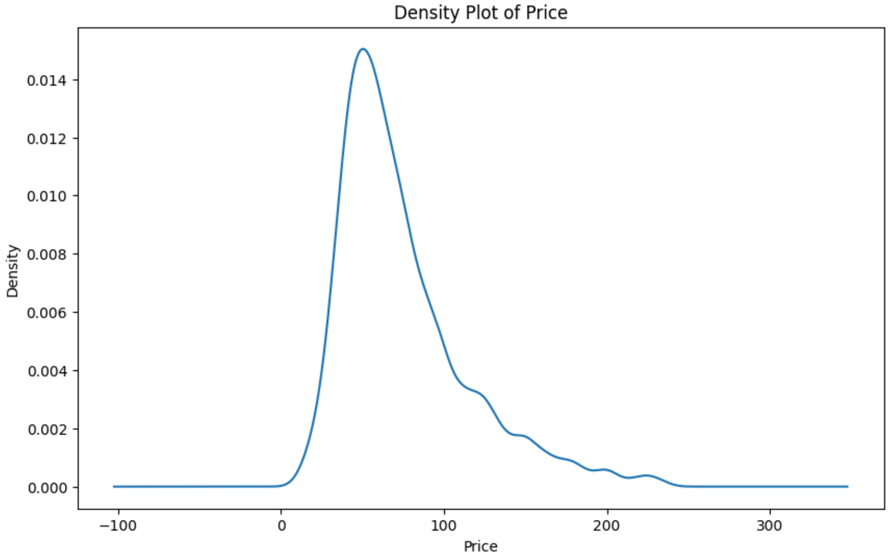
Data Visualization

```
# Setting the figure size for better readability
plt.figure(figsize=(12, 6))


palette = sns.color_palette("viridis", len(data['area_type'].unique()))

# Creating a bar plot to show the average price for each area type with different colors
sns.barplot(x='area_type', y='price', data=data, estimator=np.mean, ci=None, palette=palette)


plt.title('Average Price by Area Type', fontsize=16, fontweight='bold')
plt.xlabel('Area Type', fontsize=14)
plt.ylabel('Average Price', fontsize=14)


plt.xticks(rotation=45, fontsize=12)
plt.yticks(fontsize=12)
plt.show()
```



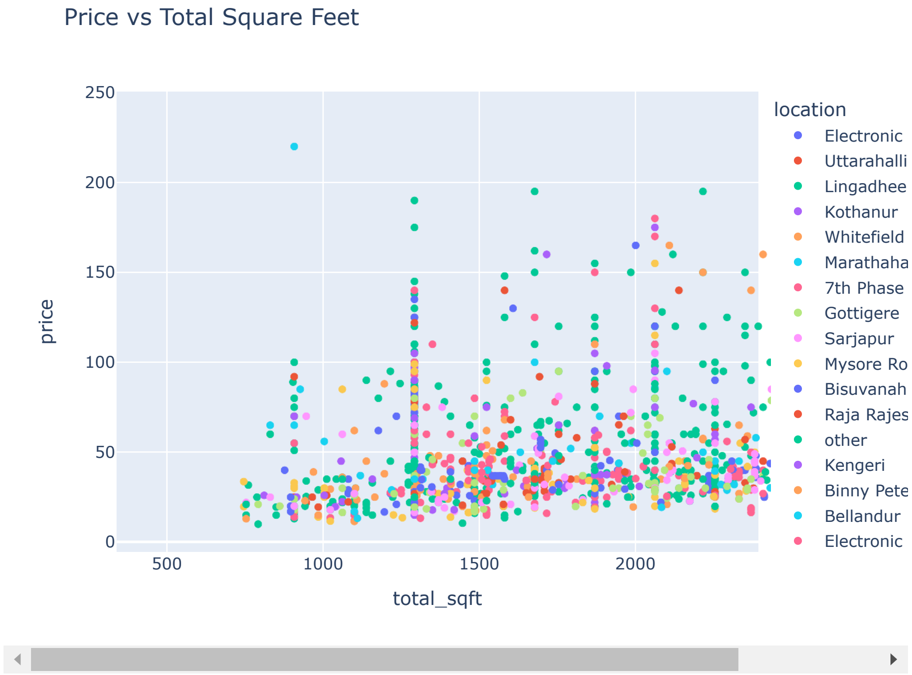
This bar chart displays the average price of properties categorized by different area types. The 'Plot Area' type shows the highest average price, followed by 'Super built-up Area', 'Built-up Area', and 'Carpet Area'. This suggests that properties classified under 'Plot Area' are generally more expensive compared to the other types.

```
#Density Plot using Matplotlib
plt.figure(figsize=(10, 6))
data['price'].plot(kind='density')
plt.title('Density Plot of Price')
plt.xlabel('Price')
plt.show()
```



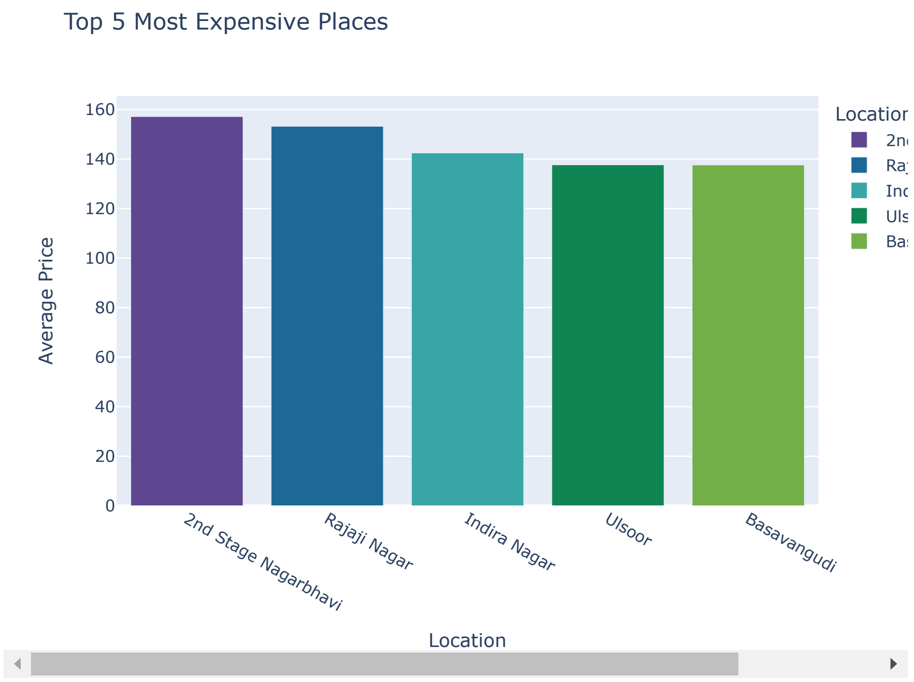
This density plot illustrates the distribution of property prices in the dataset. The peak indicates that most property prices cluster around a certain value, with fewer properties priced significantly higher or lower. The right-skewed nature of the plot suggests that there are some high-priced properties, but they are less common compared to lower-priced ones.

```
import plotly.express as px


fig_scatter = px.scatter(data, x='total_sqft', y='price', color='location', title='Price vs Total Square Feet')
fig_scatter.show()
```



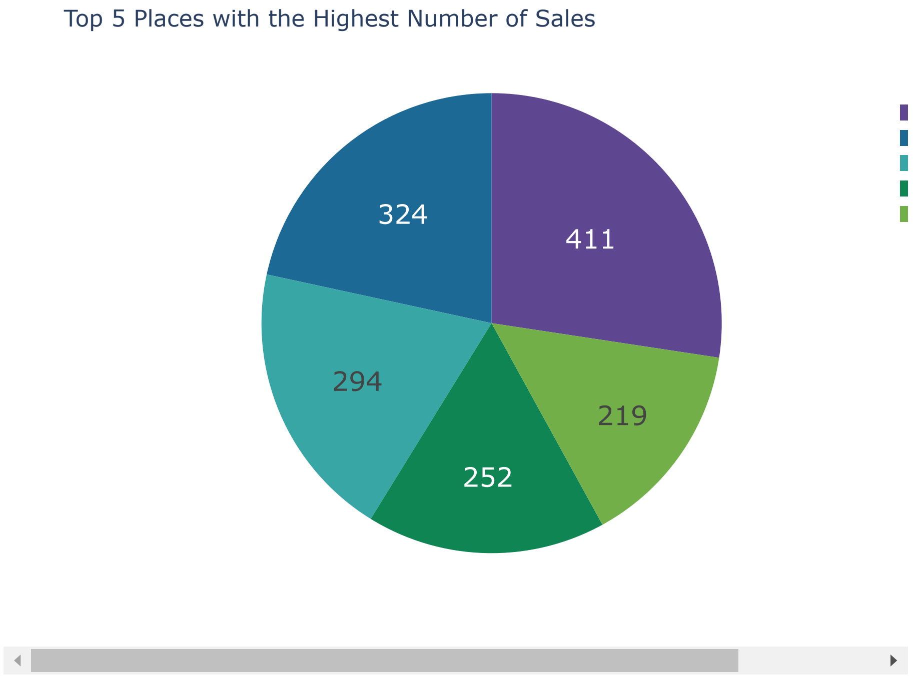
This scatter plot depicts the relationship between property prices and their total square footage, with different colors representing various locations. Generally, there is a positive correlation: as the total square feet of a property increases, the price tends to increase. However, the spread of points also indicates significant variation in prices for similar square footage, likely influenced by the location.

```
# Grouping the locations and calculating average price
expensive_places = data.groupby('location')['price'].mean().sort_values(ascending=False).head(5).reset_index()


fig1 = px.bar(expensive_places, x='location', y='price', color='location',
              title='Top 5 Most Expensive Places',
              labels={'location': 'Location', 'price': 'Average Price'},
              color_discrete_sequence=px.colors.qualitative.Prism)

fig1.show()
```



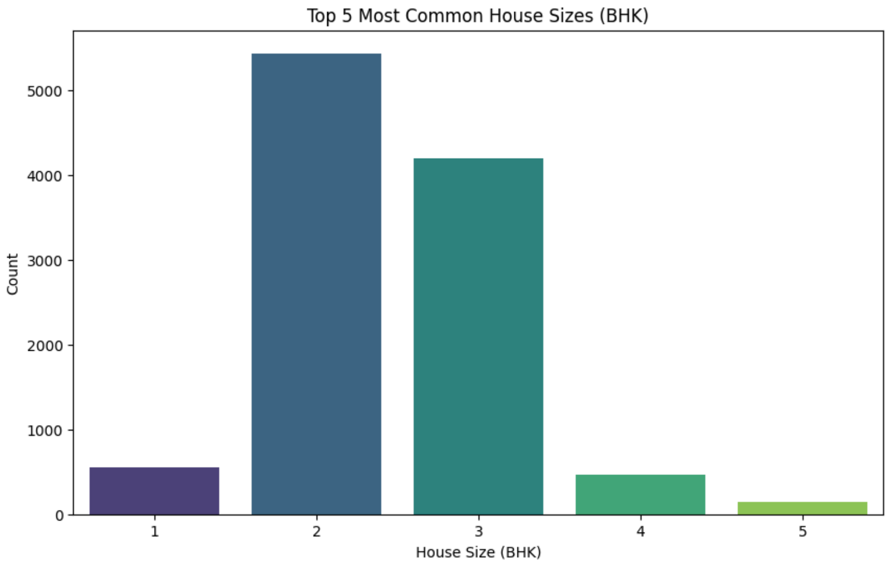
The bar chart illustrates the top 5 most expensive locations based on average property prices. Each bar represents one of these locations, with the height of the bar indicating how high the average price is.

```
import plotly.graph_objects as go
import plotly.express as px

# Group by location and counting the number of sales.
most_selling_places = data[data['location'] != 'other']['location'].value_counts().head(5).reset_index()
most_selling_places.columns = ['location', 'count']


fig2 = go.Figure(data=[go.Pie(labels=most_selling_places['location'], values=most_selling_places['count'],
                              hoverinfo='label+percent', textinfo='value', textfont_size=20,
                              marker=dict(colors=px.colors.qualitative.Prism))])

fig2.update_layout(title='Top 5 Places with the Highest Number of Sales')
fig2.show()
```



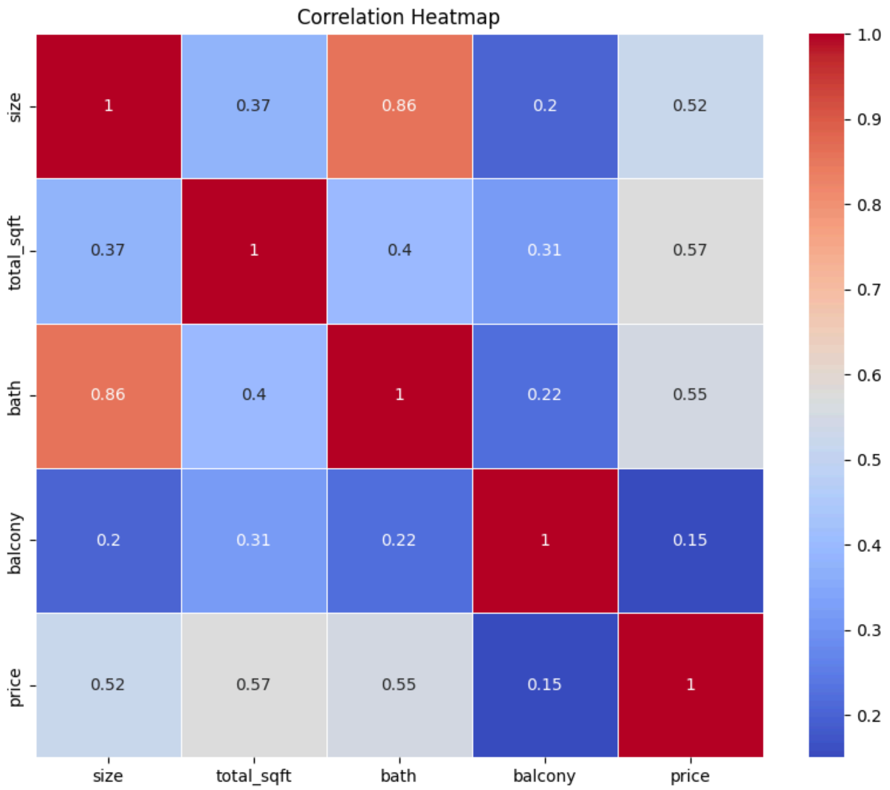
The pie chart shows the top 5 locations with the most property sales. Each slice represents one location and its share of the total sales, making it easy to see how each location contributes to the overall sales volume.

```
# Counting the most common house sizes based on size in BHK
most_common_sizes = data['size'].value_counts().head(5).reset_index()
most_common_sizes.columns = ['size', 'count']


plt.figure(figsize=(10, 6))
sns.barplot(data=most_common_sizes, x='size', y='count', palette='viridis')
plt.title('Top 5 Most Common House Sizes (BHK)')
plt.xlabel('House Size (BHK)')
plt.ylabel('Count')
plt.show()
```



The bar chart shows the top 5 most popular house sizes in terms of bedrooms (BHK). It helps us to see which sizes are most common, making it easier for buyers and sellers to understand market trends and make informed decisions.

```
# For Correlation Matrix, to get a better idea about the relationship between numerical features with each other.
plt.figure(figsize=(10, 8))
corr_matrix = data.select_dtypes(include=['float64', 'int64']).corr()
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Heatmap')
plt.show()
```



Data Modeling

```
from sklearn.compose import make_column_transformer
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import OneHotEncoder
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split, cross_val_score
import numpy as np

# Splitting data into dependent and independent features
# 'X' contains all features except 'price', which is our target variable 'y'
X = data.drop('price', axis=1)
y = data['price']

# Splitting the dataset into training and testing sets
# Using 80% of the data for training and 20% for testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

# One-Hot Encoding for categorical features
# This step transforms 'area_type' and 'location' columns into one-hot encoded format
ohe = OneHotEncoder()
column_trans = make_column_transformer(
    (OneHotEncoder(), ['area_type', 'location']),
    remainder='passthrough'
)

# Define a dictionary of models to evaluate
# Including Linear Regression, Decision Tree, Random Forest, and XGBoost
models = {
    'Linear Regression': LinearRegression(),
    'Decision Tree': DecisionTreeRegressor(),
    'Random Forest': RandomForestRegressor(random_state=0),
    'XGBoost': XGBRegressor(random_state=0)
}

# Iterate through each model, train it, and evaluate its performance
for name, model in models.items():
    # Create a pipeline that applies column transformations and then fits the model
    pipe = make_pipeline(column_trans, model)

    # Fit the model on the training data
    pipe.fit(X_train, y_train)

    # Predict on the test data
    y_pred = pipe.predict(X_test)

    # Calculate the R-squared score to evaluate model performance
    score = r2_score(y_test, y_pred)

    # Print the model name and its R-squared score
    print(f"{name} R-squared: {score}")

# Now we will train our meta-model using predictions from the base models
# Obtain predictions from the base models for the test set
y_pred_lr = models['Linear Regression'].predict(column_trans.transform(X_test))
y_pred_dt = models['Decision Tree'].predict(column_trans.transform(X_test))
y_pred_rf = models['Random Forest'].predict(column_trans.transform(X_test))

# Stack the predictions as new features for the meta-model
meta_features = np.column_stack((y_pred_lr, y_pred_dt, y_pred_rf))

# Train the meta-model using the stacked features
meta_model = XGBRegressor(random_state=0)
meta_model.fit(meta_features, y_test)

# Predict using the meta-model
y_pred_stacked = meta_model.predict(meta_features)

# Calculating and printing the R-squared score for the stacked model
r2_stacked = r2_score(y_test, y_pred_stacked)
print("Stacked Model R-squared:", r2_stacked)
```

```
Linear Regression R-squared: 0.6310029465726306
Decision Tree R-squared: 0.45390045842396876
```

```
     Random Forest R-squared: 0.60284969197938
```

```
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred_stacked, color='blue', label='Predicted vs Actual')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], linestyle='--', color='red', label='Ideal Fit')
plt.title('Regression Plot: Stacked Model Predictions')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```