Task 2: Data Visualization

Goal

Visualize the given dataset (https://www.kaggle.com/datasets/shree1992/housedata)

Requirements:

1. Load the dataset
2. Perform basic data cleaning (if needed) and explore the dataset (using .describe(), .info(), etc.).

Create the following visualizations:

1. A scatter plot of features (yr_built,floors) vs price and explain which one impacts the price more.
2. A box plot for a single feature to identify outliers.
3. A heatmap to visualize the correlation between features.
4. A line graph to show a trend.

```
#Importing relevant python libraries
import kagglehub
import plotly.express as px
import pandas as pd
from scipy import stats
import numpy as np
from kagglehub import KaggleDatasetAdapter

#Set the path to the file in the dataset from Kaggle
file_path = "data.csv"
```

```
#Load the Dataset
df = kagglehub.load_dataset(
  KaggleDatasetAdapter.PANDAS,
  "shree1992/housedata",
  file_path,
  # we can add further arguments to import as required (sql, etc.). See documenation for more information:
  # https://github.com/Kaggle/kagglehub/blob/main/README.md#kaggledatasetadapterpandas
)
```

```
<ipython-input-37-063d27ed61af>:2: DeprecationWarning:

load_dataset is deprecated and will be removed in a future version.
```

```
#Exploring the dataset
df.head()
```

	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	sqft_above	sqft_basement	yr_buil
0	2014-05-02 00:00:00	313000.0	3.0	1.50	1340	7912	1.5	0	0	3	1340	0	195
1	2014-05-02 00:00:00	2384000.0	5.0	2.50	3650	9050	2.0	0	4	5	3370	280	192
2	2014-05-02 00:00:00	342000.0	3.0	2.00	1930	11947	1.0	0	0	4	1930	0	196
3	2014-05-02 00:00:00	420000.0	3.0	2.25	2000	8030	1.0	0	0	4	1000	1000	196
4	2014-05-02 00:00:00	550000.0	4.0	2.50	1940	10500	1.0	0	0	4	1140	800	197

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

```
#Exploring the dataset
df.info()
df.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4600 entries, 0 to 4599
Data columns (total 18 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   date           4600 non-null   object 
 1   price          4600 non-null   float64
 2   bedrooms       4600 non-null   float64
 3   bathrooms      4600 non-null   float64
 4   sqft_living    4600 non-null   int64  
 5   sqft_lot       4600 non-null   int64  
 6   floors         4600 non-null   float64
 7   waterfront     4600 non-null   int64  
 8   view           4600 non-null   int64  
 9   condition      4600 non-null   int64  
 10  sqft_above     4600 non-null   int64  
 11  sqft_basement  4600 non-null   int64  
 12  yr_built       4600 non-null   int64  
 13  yr_renovated   4600 non-null   int64  
 14  street         4600 non-null   object 
 15  city           4600 non-null   object 
 16  statezip       4600 non-null   object 
 17  country        4600 non-null   object 
dtypes: float64(4), int64(9), object(5)
memory usage: 647.0+ KB
```

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	sqft_above	s
count	4.600000e+03	4600.000000	4600.000000	4600.000000	4.600000e+03	4600.000000	4600.000000	4600.000000	4600.000000	4600.000000	
mean	5.519630e+05	3.400870	2.160815	2139.346957	1.485252e+04	1.512065	0.007174	0.240652	3.451739	1827.265435	
std	5.638347e+05	0.908848	0.783781	963.206916	3.588444e+04	0.538288	0.084404	0.778405	0.677230	862.168977	
min	0.000000e+00	0.000000	0.000000	370.000000	6.380000e+02	1.000000	0.000000	0.000000	1.000000	370.000000	
25%	3.228750e+05	3.000000	1.750000	1460.000000	5.000750e+03	1.000000	0.000000	0.000000	3.000000	1190.000000	
50%	4.609435e+05	3.000000	2.250000	1980.000000	7.683000e+03	1.500000	0.000000	0.000000	3.000000	1590.000000	
75%	6.549625e+05	4.000000	2.500000	2620.000000	1.100125e+04	2.000000	0.000000	0.000000	4.000000	2300.000000	
max	2.659000e+07	9.000000	8.000000	13540.000000	1.074218e+06	3.500000	1.000000	4.000000	5.000000	9410.000000	

Double-click (or enter) to edit

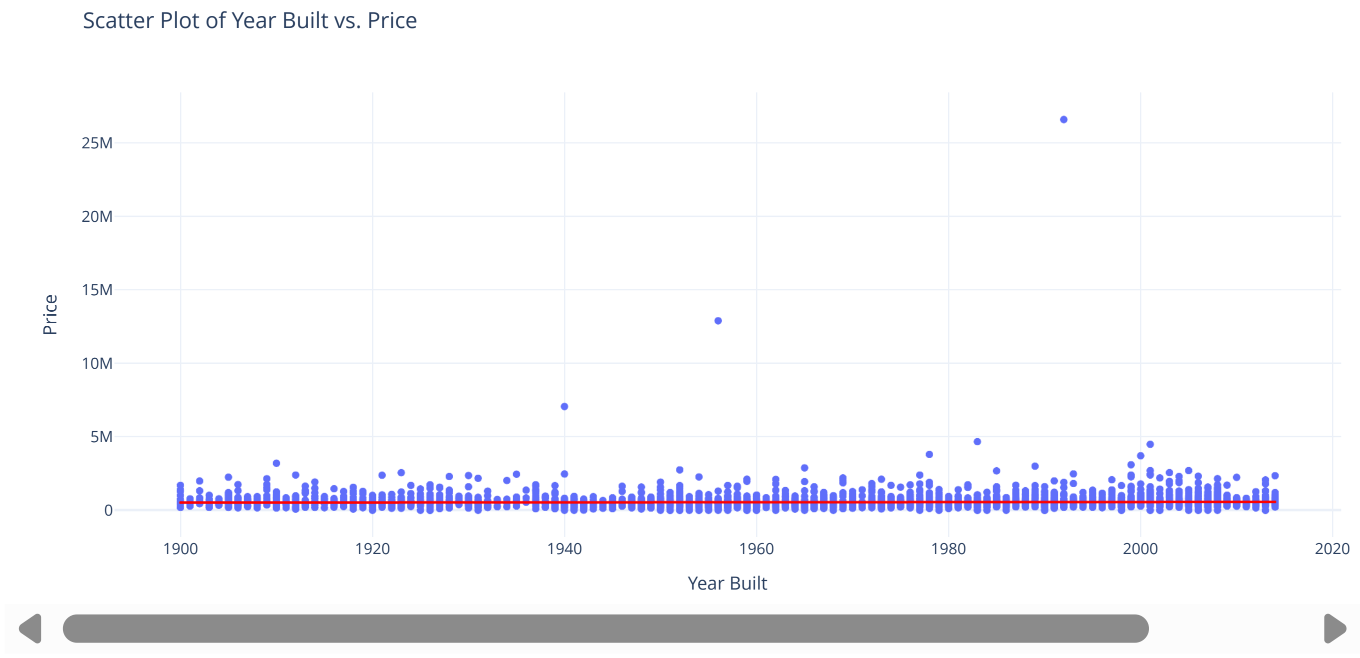
```
#Check for missing values
print(df.isnull().sum())
```

```
date             0
price            0
bedrooms         0
bathrooms        0
sqft_living      0
sqft_lot         0
floors           0
waterfront       0
view             0
condition        0
sqft_above       0
sqft_basement    0
yr_built         0
yr_renovated     0
street           0
city             0
statezip         0
country          0
dtype: int64
```

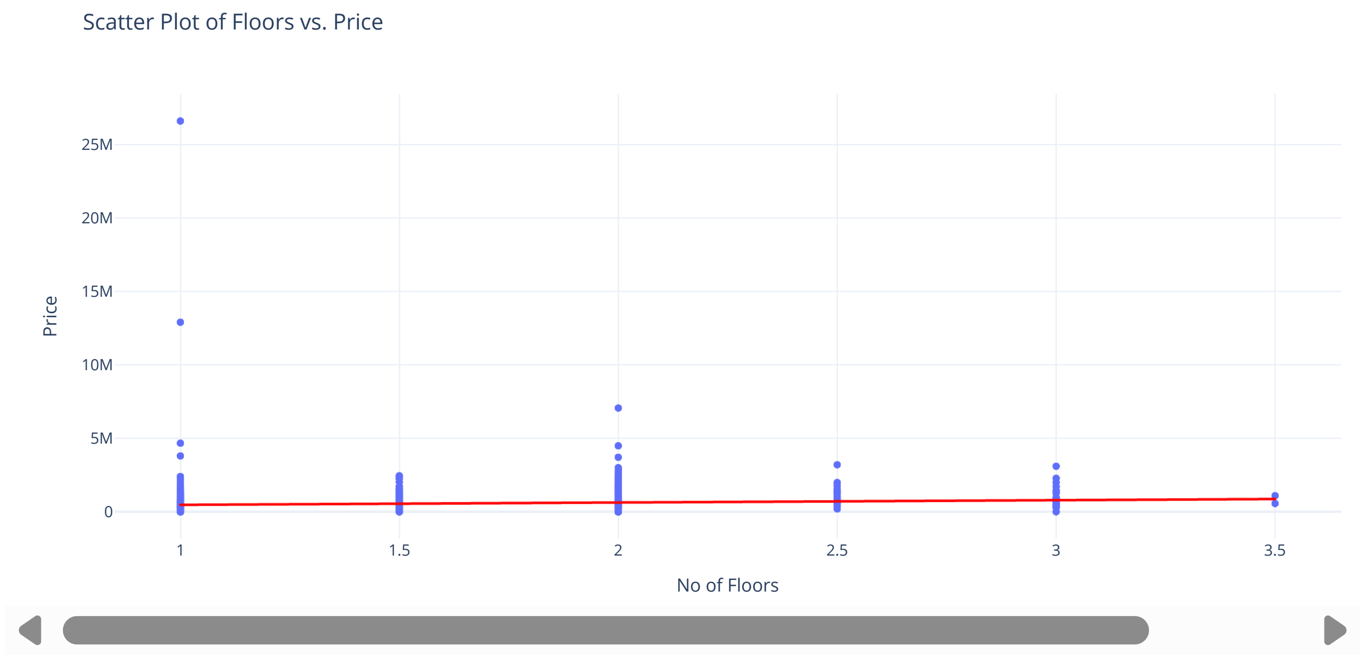
No data cleaning is required because we have found no missing/erroneous values from exploring the dataset above.

Visualizations

```
from re import template
#Scatter plot of features yr_built vs price
fig_1 = px.scatter(df, x='yr_built', y='price',
                 title='Scatter Plot of Year Built vs. Price',
                 trendline='ols',
                 labels={'price': 'Price', 'yr_built': 'Year Built'},
                 trendline_color_override="red",
                 template="plotly_white",
                )
fig_1.show()
```




```
#Scatter plot of features floors vs price
fig_2 = px.scatter(df, x='floors', y='price',
                 title='Scatter Plot of Floors vs. Price',
                 trendline='ols',
                 labels={'price': 'Price', 'floors': 'No of Floors'},
                 trendline_color_override="red",
                 template="plotly_white",
                 )
fig_2.show()
```




```
#get trendline data for yr_built vs price
results_1 = px.get_trendline_results(fig_1)
r_squared_1 = results_1.px_fit_results.iloc[0].rsquared
slope_1 = results_1.px_fit_results.iloc[0].params[1]

#get trendline data for floors vs price
results_2 = px.get_trendline_results(fig_2)
r_squared_2 = results_2.px_fit_results.iloc[0].rsquared
slope_2 = results_2.px_fit_results.iloc[0].params[1]

print(f"R-squared for yr_built vs price: {r_squared_1}")
print(f"Slope for yr_built vs price: {slope_1}")
print(f"R-squared for floors vs price: {r_squared_2}")
print(f"Slope for floors vs price: {slope_2}")
```

```
R-squared for yr_built vs price: 0.0004777210349372618
Slope for yr_built vs price: 414.49287992337895
R-squared for floors vs price: 0.022940374099241656
Slope for floors vs price: 158648.89345565005
```

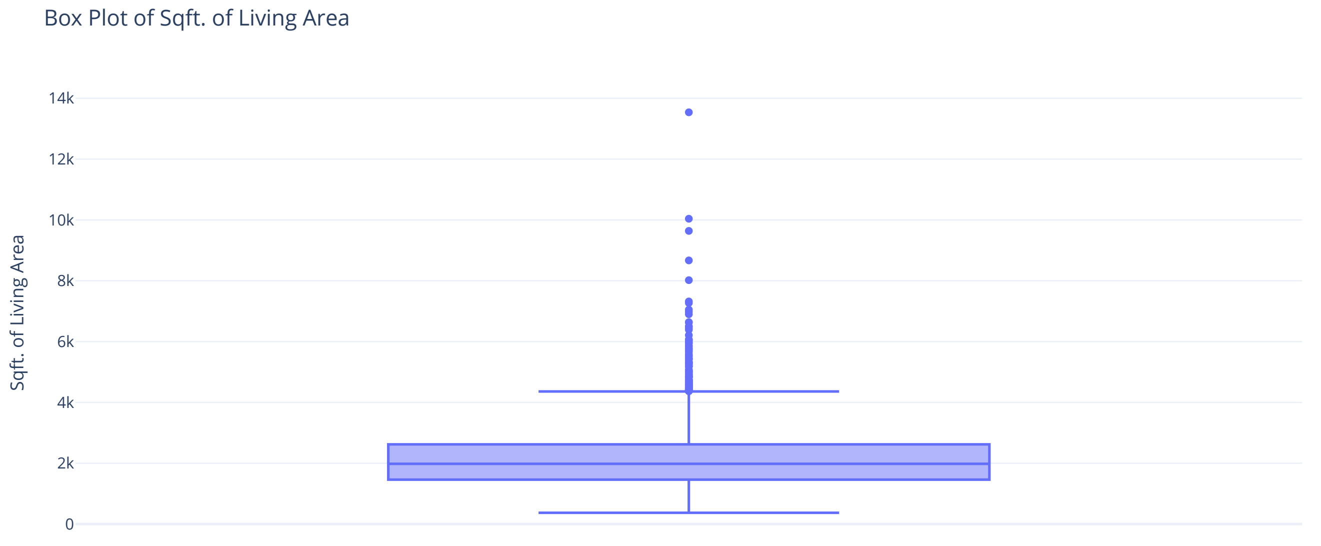
R-squared is a statistical measure that represents the proportion of the variance for a dependent variable (price) that's explained by an independent variable (yr_built or floors). Higher R-squared values indicate a stronger linear relationship, i.e. a better model fit. On the other hand, the slope (or gradient) represents the change in the dependent variable (price) for a one-unit change in the independent variable (yr_built or floors).

In our case, the R-squared metric for floors vs price is 0.0229 which is significantly higher than that of yr_built vs price, which is 0.0004777. This suggests that floors has a stronger linear relationship with price compared to yr_built.

The slope/gradient for floors vs price (158648.89) is much larger than the slope for yr_built vs price (414.49) which indicates a high unit increase impact on price of floors in comparison to yr_built.

Therefore, according to both the statistical metrics, no. of floors has a greater impact on price than year built.

```
# Box plot for sqft_living to identify outliers
fig_3 = px.box(df, y='sqft_living',
               title='Box Plot of Sqft. of Living Area',
               labels={'sqft_living': 'Sqft. of Living Area'},
               template="plotly_white",
               )
fig_3.show()
```



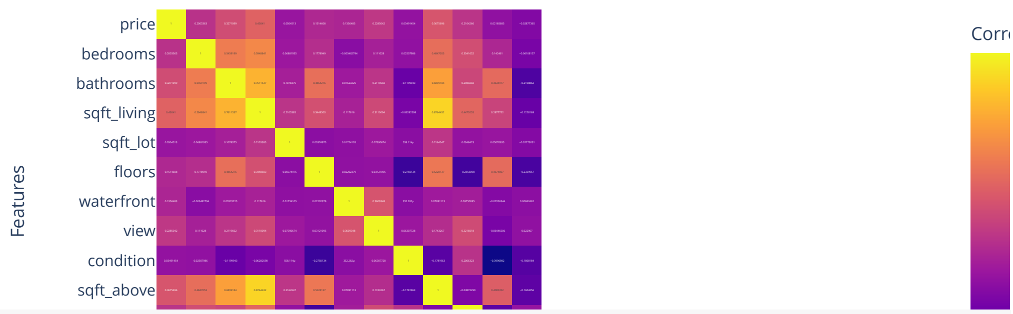
The houses outside the box plot are outliers as can be seen above the box plot.

```
#Heatmap to visualize the correlation between features

#Select number data types (float64 and int64) only
df_num = df.select_dtypes(include=[np.number])

#Create correlation matrix and plot the heatmap
correlation_matrix = df_num.corr()
fig_4 = px.imshow(correlation_matrix, text_auto=True,
                  title='Correlation Matrix Heatmap',
                  labels=dict(x="Features", y="Features", color="Correlation"),
                  template="plotly_white",)
fig_4.show()
```

Correlation Matrix Heatmap




```
# Group the data by 'yr_built' and count the number of houses built per year
yr_built_counts = df.groupby('yr_built')['yr_built'].count().reset_index(name='count')

# Create the line graph
fig_5 = px.line(yr_built_counts, x='yr_built', y='count',
                title='Line Graph of Year Built vs. Count',
                labels={'count': 'No. of Houses', 'yr_built': 'Year Built'},
                template="plotly_white",
                )
fig_5.show()
```

Line Graph of Year Built vs. Count