```
from google.colab import files
uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving house_price.csv to house_price.csv

```
import pandas as pd

# Load the uploaded dataset
df = pd.read_csv("house_price.csv")
df.head()
print("shape:",df.shape)
print("columns:",df.columns.tolist())
df.info()
df.describe()
```

```
shape: (20, 6)
columns: ['size', 'bedroom', 'bathrooms', 'location', 'year_build', 'price']
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20 entries, 0 to 19
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   size        20 non-null     int64
 1   bedroom     20 non-null     int64
 2   bathrooms   20 non-null     float64
 3   location    20 non-null     object
 4   year_build  20 non-null     int64
 5   price       20 non-null     int64
dtypes: float64(1), int64(4), object(1)
memory usage: 1.1+ KB
```

	size	bedroom	bathrooms	year_build	price
count	20.000000	20.00000	20.000000	20.000000	20.000000
mean	1615.000000	2.85000	1.950000	1999.200000	247750.000000
std	328.913682	0.74516	0.666886	13.213231	54179.890424
min	1100.000000	2.00000	1.000000	1970.000000	170000.000000
25%	1337.500000	2.00000	1.500000	1993.750000	203750.000000
50%	1625.000000	3.00000	2.000000	2002.500000	245000.000000
75%	1862.500000	3.00000	2.500000	2008.250000	292500.000000
max	2200.000000	4.00000	3.000000	2018.000000	340000.000000

Start coding or generate with AI.

Start coding or generate with AI.

```
# Check missing values print(df.isnull().sum())
df_cleaned = df.dropna() # removes rows with missing values
print(df_cleaned)
```

```
    size  bedroom  bathrooms     location  year_build   price
0   1500        3        2.0     suburb A        1995  225000
1   1200        2        1.0  city center        2002  180000
2   2000        4        2.5   Rural area        1980  250000
3   1800        3        2.0     suburb B        2010  310000
4   1400        2        1.5  city center        1998  210000
5   2200        4        3.0   Rural area        1975  280000
6   1600        3        2.0     suburb A        2005  240000
7   1100        2        1.0     suburb C        1990  170000
8   1900        3        2.5   Rural area        2015  330000
9   1700        3        2.0     suburb B        2008  290000
10  1300        2        1.5  city center        2001  200000
11  2100        4        3.0     suburb A        1970  260000
12  1550        3        2.0     suburb C        1997  230000
13  1250        2        1.0   Rural area        2004  190000
14  1850        3        2.5     suburb B        2012  320000
15  1650        3        2.0  city center        2007  250000
16  1150        2        1.0     suburb A        1985  175000
17  1950        4        3.0   Rural area        2018  340000
18  1750        3        2.0     suburb C        2009  300000
19  1350        2        1.5     suburb B        2003  205000
```

```
import pandas as pd

data = {
        "Size (sqft)" : [1500,2000,1200,1800,1000,2200,1600],
        "Bedrooms "   : [ 3,4,2,3,2,4,3],
        "Bathrooms"   : [2,3,1,2,1,3,2],
        "Location"    :["suburb","city","suburb","city","rural","suburb","City"],
        "Year Built"  : [2005,2010,1995,2008,1980,2015,2012],
        "Price"       : [250000,400000,180000,350000,120000,420000,300000]
    }

df = pd.DataFrame(data)
print(df)
```

```
    Size (sqft)  Bedrooms   Bathrooms Location  Year Built   Price
0         1500          3          2   suburb        2005  250000
1         2000          4          3     city        2010  400000
2         1200          2          1   suburb        1995  180000
3         1800          3          2     city        2008  350000
4         1000          2          1    rural        1980  120000
5         2200          4          3   suburb        2015  420000
6         1600          3          2     City        2012  300000
```

```
high_Price = df[df["Price"] > 120000]
print(high_Price)
```

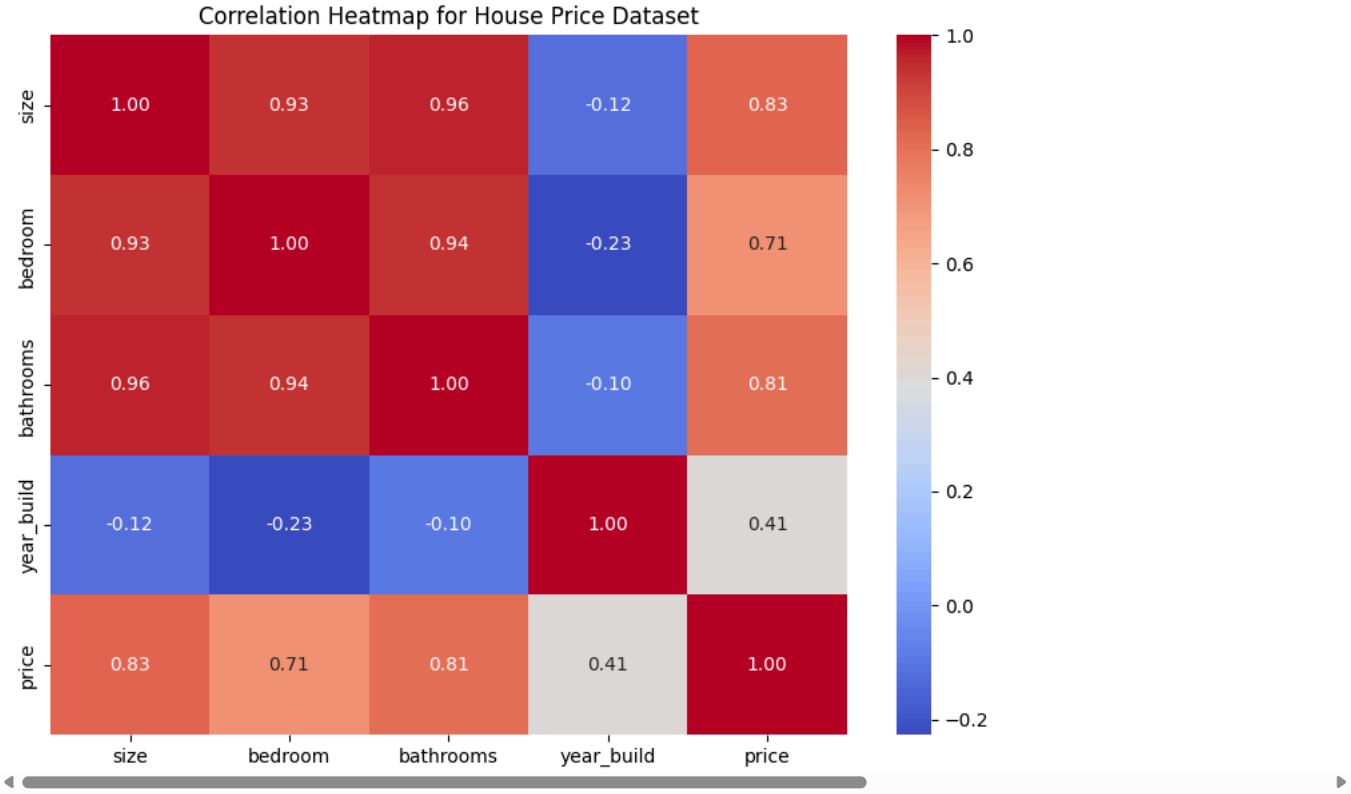
```
    Size (sqft)  Bedrooms   Bathrooms Location  Year Built   Price
0         1500          3          2   suburb        2005  250000
1         2000          4          3     city        2010  400000
2         1200          2          1   suburb        1995  180000
3         1800          3          2     city        2008  350000
5         2200          4          3   suburb        2015  420000
6         1600          3          2     City        2012  300000
```

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df["Location"] = le.fit_transform(df["Location"])
print(df)
```

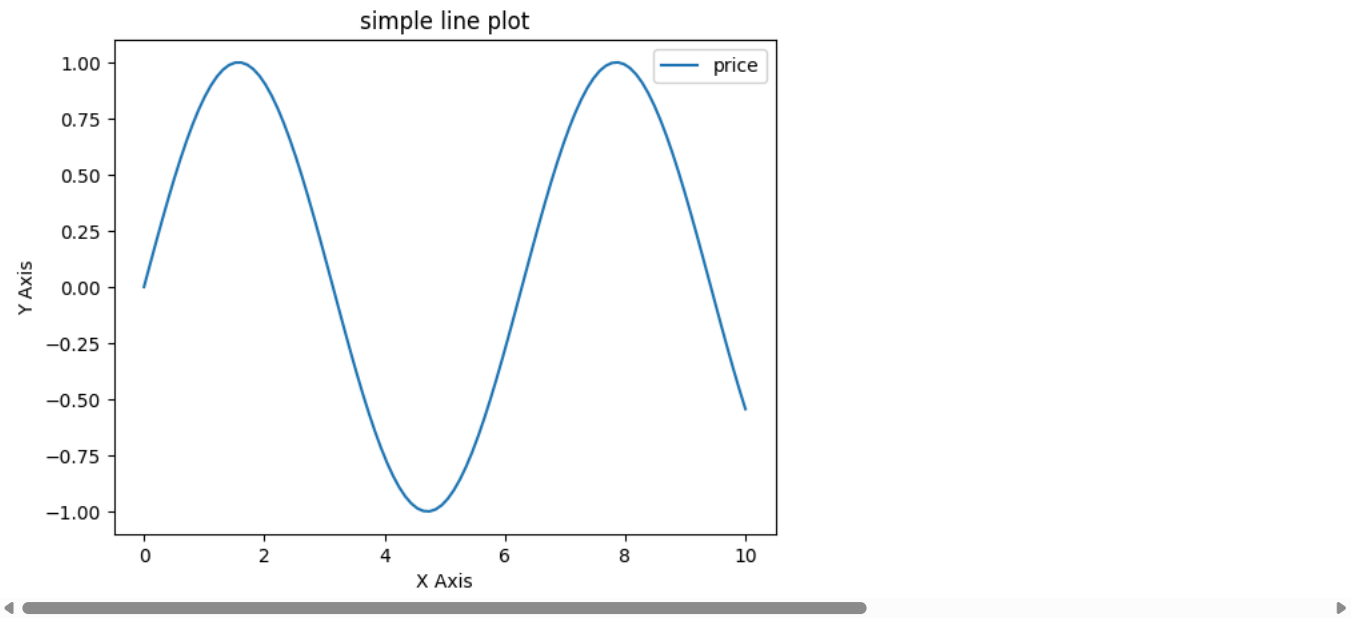
```
    Size (sqft)  Bedrooms   Bathrooms  Location  Year Built   Price
0         1500          3          2         3        2005  250000
1         2000          4          3         1        2010  400000
2         1200          2          1         3        1995  180000
3         1800          3          2         1        2008  350000
4         1000          2          1         2        1980  120000
5         2200          4          3         3        2015  420000
6         1600          3          2         0        2012  300000
```

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

# Load the dataset
df = pd.read_csv('house_price.csv')

# Clean column names
df.columns = df.columns.str.strip().str.lower()  # Standardize names

# Display column names to verify
print("Columns in dataset:", df.columns.tolist())

# Select only numeric columns for correlation
numeric_df = df.select_dtypes(include=['int64', 'float64'])

# Check if there are enough numerical columns
if numeric_df.shape[1] < 2:
    print("Not enough numeric columns to generate a correlation heatmap.")
else:
    # Generate correlation matrix
    correlation_matrix = numeric_df.corr()

    # Plot heatmap
    plt.figure(figsize=(8, 6))
    sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f")
    plt.title("Correlation Heatmap for House Price Dataset")
    plt.tight_layout()
    plt.show()
```

```
Columns in dataset: ['size', 'bedroom', 'bathrooms', 'location', 'year_build', 'price']
```




```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 10, 100)
y = np.sin(x)

plt.plot(x,y,label="price")
plt.xlabel("X Axis")
plt.ylabel("Y Axis")
plt.title("simple line plot")
plt.legend()
plt.show()
```



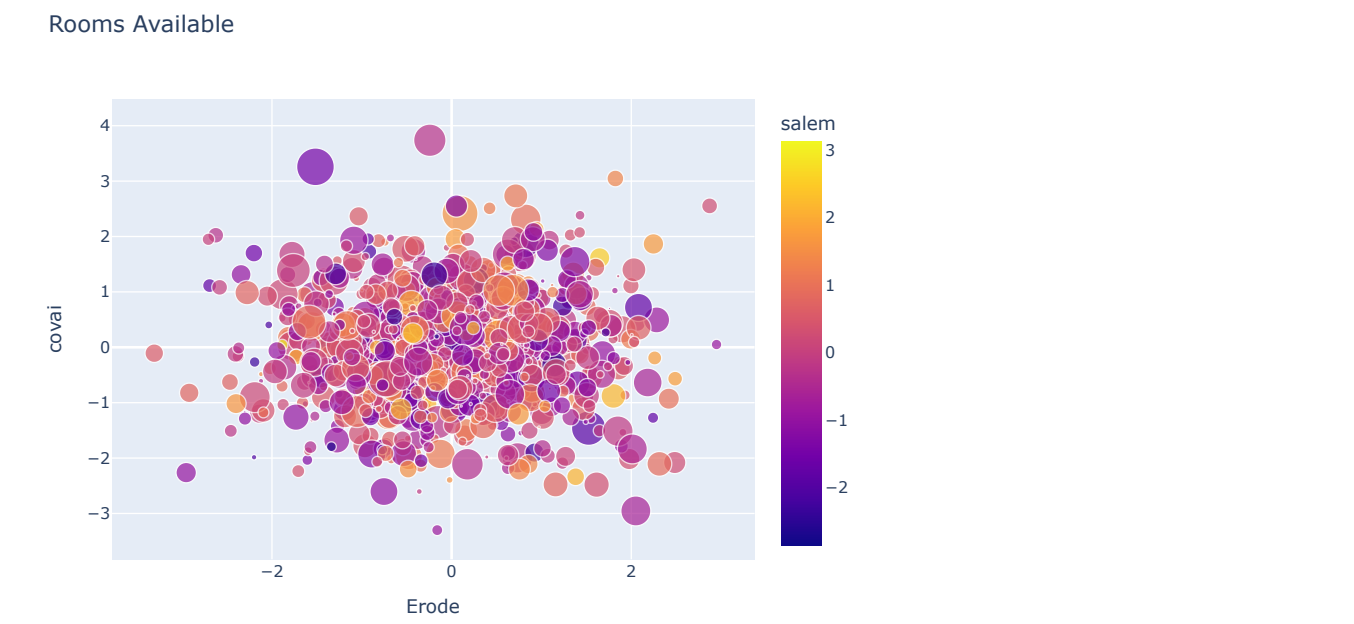

```
import pandas as pd

Data = {
    "Location": ["Erode", "covai", "Madurai", "salem"],
    "price": [42000, 30000, 35000, 18000]
}
df = pd.DataFrame(Data)  # Corrected: 'DataFrame', not 'Dataframe'

def performance_category(price):
    if price >= 42000:
```

```
        return "High"  # Corrected: Use standard quotes
    elif price >= 30000:
        return "medium"  # Corrected: Use standard quotes, consistent indentation
    else:
        return "low"  # Corrected: Use standard quotes, consistent indentation

df["performance"] = df["price"].apply(performance_category)  # Corrected: Use standard quotes
print(df)
```

```
  Location  price performance
0    Erode  42000        High
1    covai  30000      medium
2  Madurai  35000      medium
3    salem  18000         low
```

```
!pip install plotly
import plotly.express as px
import pandas as pd
import numpy as np

# Creating sample data
data = np.random.randn(1000, 4)

# Create the DataFrame
df = pd.DataFrame(data, columns=["Erode", "covai", "Madurai", "salem"])

# Ensure 'Madurai' column (used for size) has only positive values
df['Madurai'] = abs(df['Madurai'])  # Taking the absolute value

# Create the bubble chart
fig = px.scatter(df, x="Erode", y="covai", size="Madurai",
                 color="salem",  # Use 'salem' for color variation
                 hover_name="Erode",  # Display 'Erode' on hover
                 title="Rooms Available")
fig.show()
```

```
Requirement already satisfied: plotly in /usr/local/lib/python3.11/dist-packages (5.24.1)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.11/dist-packages (from plotly) (9.1.2)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from plotly) (24.2)
```

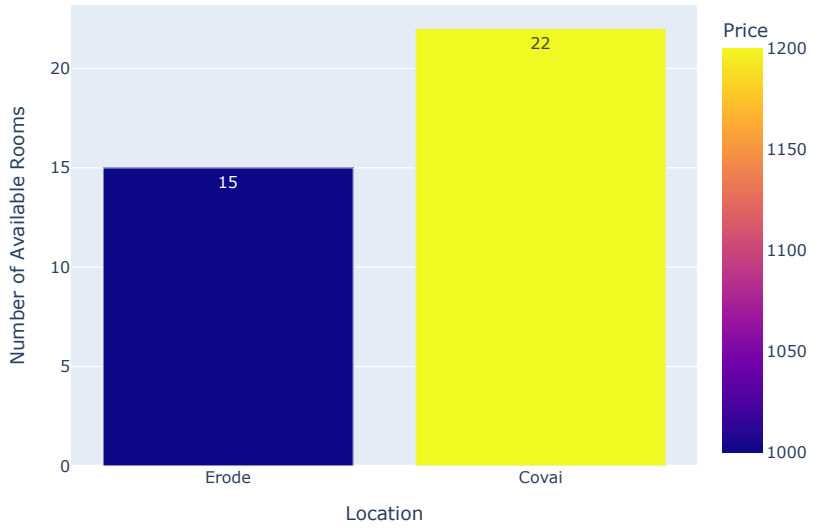



```
# prompt: create a high available house list

import plotly.express as px

# Sample data (replace with your actual data)
data = {
    "Location": ["Erode", "Covai", "Madurai", "Salem"],
    "Available Houses": [15, 22, 10, 8]  # Number of houses available in each location
}

df = pd.DataFrame(data)

# Create the bar chart
fig = px.bar(df, x="Location", y="Available Houses",
```

```
            title="House Availability by Location",
            color="Location",  # Color bars by location
            text="Available Houses"  # Show the number of houses on the bars
           )
fig.update_layout(xaxis_title="Location", yaxis_title="Number of Houses")
fig.show()
```


```
# prompt: create a low cost available rooms

import pandas as pd
import plotly.express as px

# Sample data (replace with your actual data)
data = {
    "Location": ["Erode", "Covai", "Madurai", "Salem"],
    "Available Rooms": [15, 22, 10, 8],  # Number of rooms available
    "Price": [1000, 1200, 800, 700] # Price per room
}

df = pd.DataFrame(data)

# Calculate affordability score (example: lower price = higher score)
df['Affordability'] = 1 / df['Price']


# Create the bar chart with affordability as color
fig = px.bar(df, x="Location", y="Available Rooms",
            title="Low-Cost Available Rooms by Location",
            color="Affordability",  # Color bars by affordability
            text="Available Rooms",
            hover_data=["Price"] # Show price on hover
           )
fig.update_layout(xaxis_title="Location", yaxis_title="Number of Available Rooms")
fig.show()
```


```
# prompt: create a high cost available rooms list

# Sample data (replace with your actual data)
data = {
    "Location": ["Erode", "Covai", "Madurai", "Salem"],
    "Available Rooms": [15, 22, 10, 8],  # Number of rooms available
    "Price": [1000, 1200, 800, 700] # Price per room
}

df = pd.DataFrame(data)

# Define a threshold for "high cost"
high_cost_threshold = 900  # Example: Rooms with price above 900 are high cost

# Filter for high-cost available rooms
high_cost_rooms = df[df["Price"] > high_cost_threshold]

# Create the bar chart for high-cost rooms
fig = px.bar(high_cost_rooms, x="Location", y="Available Rooms",
             title="High-Cost Available Rooms by Location",
             color="Price",  # Color bars by price
             text="Available Rooms",
             hover_data=["Price"] # Show price on hover
            )
fig.update_layout(xaxis_title="Location", yaxis_title="Number of Available Rooms")
fig.show()
```




```
# prompt: create a lexgury rooms hotel names

import pandas as pd

# Sample data (replace with your actual data)
data = {
    "Room Type": ["Presidential Suite", "Luxury Suite", "Deluxe Room", "Executive Suite", "Royal Suite"],
    "Price": [2000, 1500, 1000, 1200, 1800],  # Price per night
    "Features": ["Ocean view, private pool", "City view, jacuzzi", "Comfortable bed, balcony", "Business amenities, large workspace", "S
}

luxury_hotel_df = pd.DataFrame(data)

# Print the DataFrame
luxury_hotel_df
```

	Room Type	Price	Features
0	Presidential Suite	2000	Ocean view, private pool
1	Luxury Suite	1500	City view, jacuzzi
2	Deluxe Room	1000	Comfortable bed, balcony
3	Executive Suite	1200	Business amenities, large workspace
4	Royal Suite	1800	Spacious living area, butler service

```
# prompt: create a lower cost and foods provider

# Sample data (replace with your actual data)
data = {
    "Food Provider": ["Provider A", "Provider B", "Provider C", "Provider D"],
    "Average Cost per Meal": [8, 6, 10, 7],  # Average cost of a meal
    "Customer Rating": [4.2, 4.5, 3.8, 4.0],  # Customer rating out of 5
    "Delivery Time (minutes)": [30, 25, 40, 35]  # Average delivery time
}

food_providers_df = pd.DataFrame(data)

# Calculate a composite score (example: higher rating, lower cost, faster delivery = higher score)
food_providers_df['Composite Score'] = (
    food_providers_df['Customer Rating'] * 0.5 +  # Weighting for customer rating
    (1 / food_providers_df['Average Cost per Meal']) * 0.3 +  # Weighting for cost (inverse)
    (1 / food_providers_df['Delivery Time (minutes)']) * 0.2 # Weighting for delivery time (inverse)
)

# Sort by composite score (descending) to find the best providers
sorted_providers = food_providers_df.sort_values(by='Composite Score', ascending=False)

# Print the sorted DataFrame
print(sorted_providers)
```

```
# Create the bar chart for high-cost rooms
fig = px.bar(sorted_providers, x="Food Provider", y="Composite Score",
             title="Food Providers Ranking by Composite Score",
             color="Average Cost per Meal",  # Color bars by average cost
             text="Customer Rating", # Show customer ratings
             hover_data=["Delivery Time (minutes)"] # Show delivery time on hover
            )
fig.update_layout(xaxis_title="Food Provider", yaxis_title="Composite Score")
fig.show()
```

```
  Food Provider  Average Cost per Meal  Customer Rating  \
1    Provider B                     6              4.5
0    Provider A                     8              4.2
3    Provider D                     7              4.0
2    Provider C                    10              3.8

   Delivery Time (minutes)  Composite Score
1                       25         2.308000
0                       30         2.144167
3                       35         2.048571
2                       40         1.935000
```