Upload the Dataset

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

Choose Files Housing.csv.xlsx

- **Housing.csv.xlsx**(application/vnd.openxmlformats-officedocument.spreadsheetml.sheet) - 40874 bytes, last modified: 5/8/2025 - 100% done

```
Saving Housing.csv.xlsx to Housing.csv (2).xlsx
```

```
import pandas as pd
from google.colab import files

# Upload the file
uploaded = files.upload()

# Assuming the uploaded file is named 'Housing.csv.xlsx'
df = pd.read_excel('Housing.csv.xlsx')  # No need for sep

# Display the first few rows
df.head()
```

Choose Files Housing.csv.xlsx

- **Housing.csv.xlsx**(application/vnd.openxmlformats-officedocument.spreadsheetml.sheet) - 40874 bytes, last modified: 5/8/2025 - 100% done

```
Saving Housing.csv.xlsx to Housing.csv (5).xlsx
```

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	furn
0	13300000	7420	4	2	3	yes	no	no	no	yes	2	yes	
1	12250000	8960	4	4	4	yes	no	no	no	yes	3	no	
2	12250000	9960	3	2	2	yes	no	yes	no	no	2	yes	
3	12215000	7500	4	2	2	yes	no	yes	no	yes	3	yes	
4	11410000	7420	4	1	2	yes	yes	yes	no	yes	2	no	

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

Data Exploration

```
# Display first few rows
df.head()
```

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	furn
0	13300000	7420	4	2	3	yes	no	no	no	yes	2	yes	
1	12250000	8960	4	4	4	yes	no	no	no	yes	3	no	
2	12250000	9960	3	2	2	yes	no	yes	no	no	2	yes	
3	12215000	7500	4	2	2	yes	no	yes	no	yes	3	yes	
4	11410000	7420	4	1	2	yes	yes	yes	no	yes	2	no	

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

```
# Cell 2 - Display information about the DataFrame
print("Shape:", df.shape)
print("Columns:", df.columns.tolist())
df.info()
df.describe()
```

```
Shape: (545, 13)
Columns: ['price', 'area', 'bedrooms', 'bathrooms', 'stories', 'mainroad', 'guestroom', 'basement', 'hotwaterheating', 'airconditioning
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 545 entries, 0 to 544
Data columns (total 13 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   price             545 non-null    int64
 1   area              545 non-null    int64
 1   area              545 non-null    int64
```

```
# Shape of the dataset
print("Shape:", df.shape)
# Column names
print("Columns:", df.columns.tolist())
# Data types and non-null values
df.info()
# Summary statistics for numeric features
df.describe()
```

```
memory usage: 55.5+ KB
Shape: (545, 13)
Columns: ['price', 'area', 'bedrooms', 'bathrooms', 'stories', 'mainroad', 'guestroom', 'basement', 'hotwaterheating', 'airconditioning
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 545 entries, 0 to 544
Data columns (total 13 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   price             545 non-null    int64
 1   area              545 non-null    int64
 2   bedrooms          545 non-null    int64
 3   bathrooms         545 non-null    int64
 4   stories           545 non-null    int64
 5   mainroad          545 non-null    object
 6   guestroom         545 non-null    object
 7   basement          545 non-null    object
 8   hotwaterheating   545 non-null    object
 9   airconditioning   545 non-null    object
 10  parking           545 non-null    int64
 11  prefarea          545 non-null    object
 12  furnishingstatus  545 non-null    object
dtypes: int64(6), object(7)
memory usage: 55.5+ KB
```

	price	area	bedrooms	bathrooms	stories	parking
count	5.450000e+02	545.000000	545.000000	545.000000	545.000000	545.000000
mean	4.766729e+06	5150.541284	2.965138	1.286239	1.805505	0.693578
std	1.870440e+06	2170.141023	0.738064	0.502470	0.867492	0.861586
min	1.750000e+06	1650.000000	1.000000	1.000000	1.000000	0.000000
25%	3.430000e+06	3600.000000	2.000000	1.000000	1.000000	0.000000
50%	4.340000e+06	4600.000000	3.000000	1.000000	2.000000	0.000000
75%	5.740000e+06	6360.000000	3.000000	2.000000	2.000000	1.000000
max	1.330000e+07	16200.000000	6.000000	4.000000	4.000000	3.000000

	price	area	bedrooms	bathrooms	stories	parking
count	5.450000e+02	545.000000	545.000000	545.000000	545.000000	545.000000
mean	4.766729e+06	5150.541284	2.965138	1.286239	1.805505	0.693578
std	1.870440e+06	2170.141023	0.738064	0.502470	0.867492	0.861586
min	1.750000e+06	1650.000000	1.000000	1.000000	1.000000	0.000000
25%	3.430000e+06	3600.000000	2.000000	1.000000	1.000000	0.000000
50%	4.340000e+06	4600.000000	3.000000	1.000000	2.000000	0.000000
75%	5.740000e+06	6360.000000	3.000000	2.000000	2.000000	1.000000
max	1.330000e+07	16200.000000	6.000000	4.000000	4.000000	3.000000

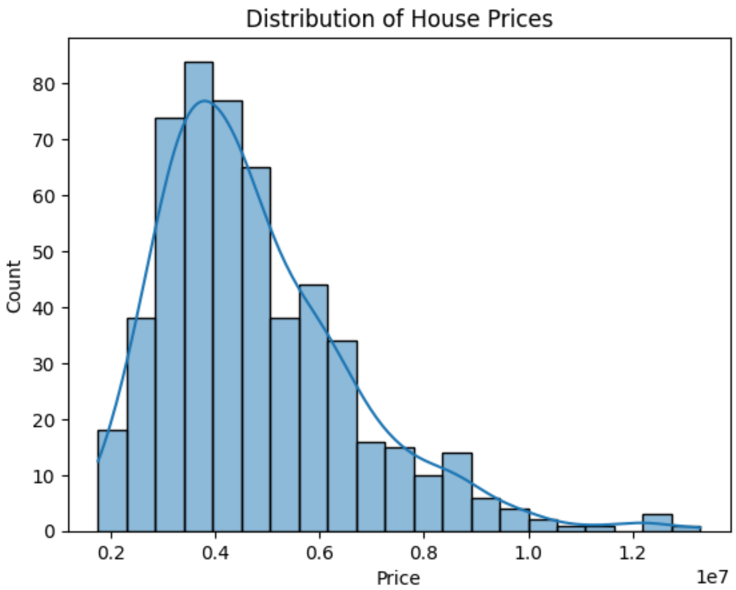
Check for Missing Values and Duplicates

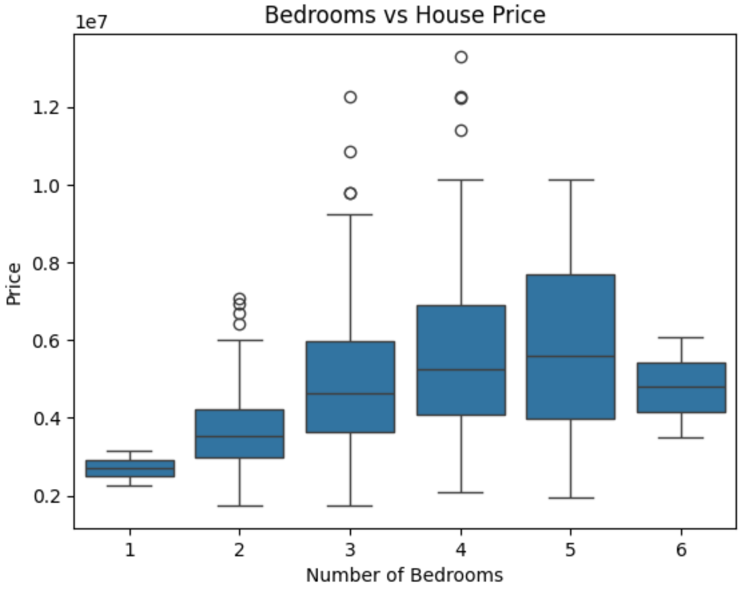
```
# Check for missing values
print(df.isnull().sum())
# Check for duplicates
print("Duplicate rows:", df.duplicated().sum())
```

```
price               0
area                0
bedrooms            0
bathrooms           0
stories             0
mainroad            0
guestroom           0
basement            0
hotwaterheating     0
airconditioning     0
parking             0
prefarea            0
furnishingstatus    0
dtype: int64
Duplicate rows: 0
```

Visualize a Few Features

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

# Distribution of house prices
sns.histplot(df['price'], kde=True)
plt.title('Distribution of House Prices')
plt.xlabel('Price')
plt.show()

# Relationship between number of bedrooms and price
sns.boxplot(x='bedrooms', y='price', data=df)
plt.title('Bedrooms vs House Price')
plt.xlabel('Number of Bedrooms')
plt.ylabel('Price')
plt.show()
```





Identify Target and Features

```
import pandas as pd # Make sure pandas is imported

# ... (Your other code)

target = 'hotwaterheating'

# Reload or recreate the DataFrame if necessary
# df = pd.read_csv('Housing.csv', sep=';')  # Assuming Housing.csv is your data fil
```

```
features = df.columns.drop(target)
```

```
Features: Index(['price', 'area', 'bedrooms', 'bathrooms', 'stories', 'mainroad',
       'guestroom', 'basement', 'airconditioning', 'parking', 'prefarea',
       'furnishingstatus'],
      dtype='object')
```

One-Hot Encoding

```
df_encoded = pd.get_dummies('df', drop_first=True)
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler

# Check that 'price' exists in the DataFrame
# assert 'price' in df_encoded.columns, "'price' column not found in df_encoded"

# Scale the feature columns (excluding the target 'price')
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df_encoded.drop('price', axis=1))

# Extract the target variable
y = df_encoded['price']
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-32-ccdfd5d00768> in <cell line: 0>()
      6 # Scale the feature columns (excluding the target 'price')
      7 scaler = StandardScaler()
----> 8 X_scaled = scaler.fit_transform(df_encoded.drop('price', axis=1))
      9 
     10 # Extract the target variable

                              3 frames
/usr/local/lib/python3.11/dist-packages/pandas/core/indexes/base.py in drop(self, labels, errors)
   7068         if mask.any():
   7069             if errors != "ignore":
-> 7070                 raise KeyError(f"{labels[mask].tolist()} not found in axis")
   7071             indexer = indexer[~mask]
   7072         return self.delete(indexer)

KeyError: "['price'] not found in axis"
```

Next steps: Explain error

Train-Test Split

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
# Split data
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-33-c0276d08d785> in <cell line: 0>()
      3 from sklearn.metrics import mean_squared_error, r2_score
      4 # Split data
----> 5 X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

NameError: name 'X_scaled' is not defined
```

Next steps: Explain error

Model Building

```
# Train model
model = LinearRegression()
```

```
model.fit(X_train, y_train)
# Predict
y_pred = model.predict(X_test)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-34-fb6bf145dda5> in <cell line: 0>()
      1 # Train model
      2 model = LinearRegression()
----> 3 model.fit(X_train, y_train)
      4 # Predict
      5 y_pred = model.predict(X_test)

NameError: name 'X_train' is not defined
```

Next steps: Explain error

Evaluation

```
print("MSE:", mean_squared_error(y_test, y_pred))
print("R² Score:", r2_score(y_test, y_pred))
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-35-e2d208fc2f54> in <cell line: 0>()
----> 1 print("MSE:", mean_squared_error(y_test, y_pred))
      2 print("R² Score:", r2_score(y_test, y_pred))

NameError: name 'y_test' is not defined
```

Next steps: Explain error

Make Predictions from New Input

```
# Sample input (replace values with any other valid values from the original dataset)
new_student = {
'school': 'GP', # 'GP' or 'MS'
'sex': 'F', # 'F' or 'M'
'age': 17, # Integer
'address': 'U', # 'U' or 'R'
'famsize': 'GT3', # 'LE3' or 'GT3'
'Pstatus': 'A', # 'A' or 'T'
'Medu': 4, # 0 to 4
'Fedu': 3, # 0 to 4
'Mjob': 'health', # 'teacher', 'health', etc.
'Fjob': 'services',
'reason': 'course',
'guardian': 'mother',
'traveltime': 2,
'studytime': 3,
'failures': 0,
'schoolsup': 'yes',
'famsup': 'no',
'paid': 'no',
'activities': 'yes',
'nursery': 'yes',
'higher': 'yes',
'internet': 'yes',
'romantic': 'no',
'famrel': 4,
'freetime': 3,
'goout': 3,
'Dalc': 1,
'Walc': 1,
'health': 4,
'absences': 2,
4/26/25, 12:08 PM sample project.ipynb - Colab
https://colab.research.google.com/drive/1LHSouQeD_tA9J58hn8Q1-yEM77VgZi3U#scrollTo=5BYaJj5jmg8c&printMode=true 9/14
'G1': 14,
'G2': 15
}
```

```
File "<ipython-input-36-b7b44068fba0>", line 33
  4/26/25, 12:08 PM sample project.ipynb - Colab
             ^
SyntaxError: leading zeros in decimal integer literals are not permitted; use an 0o prefix for octal integers
```

Next steps: Fix error

Convert to DataFrame and Encode

```
import numpy as np
# Convert to DataFrame
new_df = pd.DataFrame([new_student])
# Combine with original df to match columns
df_temp = pd.concat([df.drop('G3', axis=1), new_df], ignore_index=True)
# One-hot encode
df_temp_encoded = pd.get_dummies(df_temp, drop_first=True)
# Match the encoded feature order
df_temp_encoded = df_temp_encoded.reindex(columns=df_encoded.drop('G3', axis=1).columns, fill_value=0)
# Scale (if you used scaling)
new_input_scaled = scaler.transform(df_temp_encoded.tail(1))
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-37-99899f534580> in <cell line: 0>()
      1 import numpy as np
      2 # Convert to DataFrame
----> 3 new_df = pd.DataFrame([new_student])
      4 # Combine with original df to match columns
      5 df_temp = pd.concat([df.drop('G3', axis=1), new_df], ignore_index=True)

NameError: name 'new_student' is not defined
```

Next steps: Explain error

Predict the Final Grade

```
predicted_grade = model.predict(new_input_scaled)
print("  Predicted Final Grade (G3):", round(predicted_grade[0], 2))
```

Deployment-Building an Interactive App

```
!pip install gradio
```

Create a Prediction Function

```
import gradio as gr
```

Create the Gradio Interface

```
inputs = [
gr.Dropdown(['GP', 'MS'], label="School (GP=Gabriel Pereira, MS=Mousinho da Silveira)"),
gr.Dropdown(['M', 'F'], label="Gender (M=Male, F=Female)"),
gr.Number(label="Student Age"),
gr.Dropdown(['U', 'R'], label="Residence Area (U=Urban, R=Rural)"),
gr.Dropdown(['LE3', 'GT3'], label="Family Size (LE3=≤3, GT3=>3 members)"),
gr.Dropdown(['A', 'T'], label="Parent Cohabitation Status (A=Apart, T=Together)"),
gr.Number(label="Mother's Education Level (0-4)"),
gr.Number(label="Father's Education Level (0-4)"),
gr.Dropdown(['teacher', 'health', 'services', 'at_home', 'other'], label="Mother's Job"),
gr.Dropdown(['teacher', 'health', 'services', 'at_home', 'other'], label="Father's Job"),
gr.Dropdown(['home', 'reputation', 'course', 'other'], label="Reason for Choosing School"),
gr.Dropdown(['mother', 'father', 'other'], label="Guardian"),
gr.Number(label="Travel Time to School (1-4)"),
gr.Number(label="Weekly Study Time (1-4)"),
gr.Number(label="Past Class Failures (0-3)"),
gr.Dropdown(['yes', 'no'], label="Extra School Support"),
gr.Dropdown(['yes', 'no'], label="Family Support"),
gr.Dropdown(['yes', 'no'], label="Extra Paid Classes"),
gr.Dropdown(['yes', 'no'], label="Participates in Activities"),
gr.Dropdown(['yes', 'no'], label="Attended Nursery"),
gr.Dropdown(['yes', 'no'], label="Aspires Higher Education"),
gr.Dropdown(['yes', 'no'], label="Internet Access at Home"),
gr.Dropdown(['yes', 'no'], label="Currently in a Relationship"),
gr.Number(label="Family Relationship Quality (1-5)"),
gr.Number(label="Free Time After School (1-5)"),
gr.Number(label="Going Out Frequency (1-5)"),
gr.Number(label="Workday Alcohol Consumption (1-5)"),
4/26/25, 12:08 PM sample project.ipynb - Colab
https://colab.research.google.com/drive/1LHSouQeD_tA9J58hn8Q1-yEM77VgZi3U#scrollTo=5BYaJj5jmg8c&printMode=true 13/14
It looks like you are running Gradio on a hosted a Jupyter notebook. For the Gradio app to work, sharing must be enabled. Automatically se
Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://37518063c688a89403.gradio.live
This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the working direc
 Student Performance Predictor
Enter academic and demographic info to predict the final grade (G3) of a student.
GP 0
School (GP=Gabriel Pereira, MS=Mousinho da Silveira)  Predicted Final Grade (G3)
gr.Number(label="Weekend Alcohol Consumption (1-5)"),
gr.Number(label="Health Status (1=Very Bad to 5=Excellent)"),
gr.Number(label="Number of Absences"),
gr.Number(label="Grade in 1st Period (G1: 0-20)"),
gr.Number(label="Grade in 2nd Period (G2: 0-20)")
]
```

Upload the Dataset

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

Choose Files Housing.csv.xlsx

- **Housing.csv.xlsx**(application/vnd.openxmlformats-officedocument.spreadsheetml.sheet) - 40874 bytes, last modified: 5/8/2025 - 100% done

```
Saving Housing.csv.xlsx to Housing.csv (6).xlsx
```

Load the Dataset

```
import pandas as pd

# Load the Excel file
df = pd.read_excel('/Housing.csv.xlsx')

# Preview the data
df.head()
```

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	furn
0	13300000	7420	4	2	3	yes	no	no	no	yes	2	yes	
1	12250000	8960	4	4	4	yes	no	no	no	yes	3	no	
2	12250000	9960	3	2	2	yes	no	yes	no	no	2	yes	
3	12215000	7500	4	2	2	yes	no	yes	no	yes	3	yes	
4	11410000	7420	4	1	2	yes	yes	yes	no	yes	2	no	

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

Data Exploration

```
# Dataset shape
print("Shape of the dataset:", df.shape)

# Column names
print("Columns:", df.columns.tolist())

# Info about data types and missing value
df.info()
```

```
# Summary statistics
df.describe()
```

```
Shape of the dataset: (545, 13)
Columns: ['price', 'area', 'bedrooms , 'bathrooms', 'stories', 'mainroad', 'guestroom', 'basement', 'hotwaterheating', 'airconditioning
<class 'pandas.core.frame.DataFrame'
RangeIndex: 545 entries, 0 to 544
Data columns (total 13 columns):
 #   Column            Non-Null Cour   Dtype
---  ------            -------------   -----
 0   price             545 non-null    int64
 1   area              545 non-null    int64
 2   bedrooms          545 non-null    int64
 3   bathrooms         545 non-null    int64
 4   stories           545 non-null    int64
 5   mainroad          545 non-null    object
 6   guestroom         545 non-null    object
 7   basement          545 non-null    object
 8   hotwaterheating   545 non-null    object
 9   airconditioning   545 non-null    object
 10  parking           545 non-null    int64
 11  prefarea          545 non-null    object
 12  furnishingstatus  545 non-null    object
dtypes: int64(6), object(7)
memory usage: 55.5+ KB
```

	price	area	bedrooms	bathrooms	stories	parking
count	5.450000e+02	545.000000	545.000000	545.000000	545.000000	545.000000
mean	4.766729e+06	5150.541284	2.965138	1.286239	1.805505	0.693578
std	1.870440e+06	2170.141023	0.738064	0.502470	0.867492	0.861586
min	1.750000e+06	1650.000000	1.000000	1.000000	1.000000	0.000000
25%	3.430000e+06	3600.000000	2.000000	1.000000	1.000000	0.000000
50%	4.340000e+06	4600.000000	3.000000	1.000000	2.000000	0.000000
75%	5.740000e+06	6360.000000	3.000000	2.000000	2.000000	1.000000
max	1.330000e+07	16200.000000	6.000000	4.000000	4.000000	3.000000

Data Cleaning

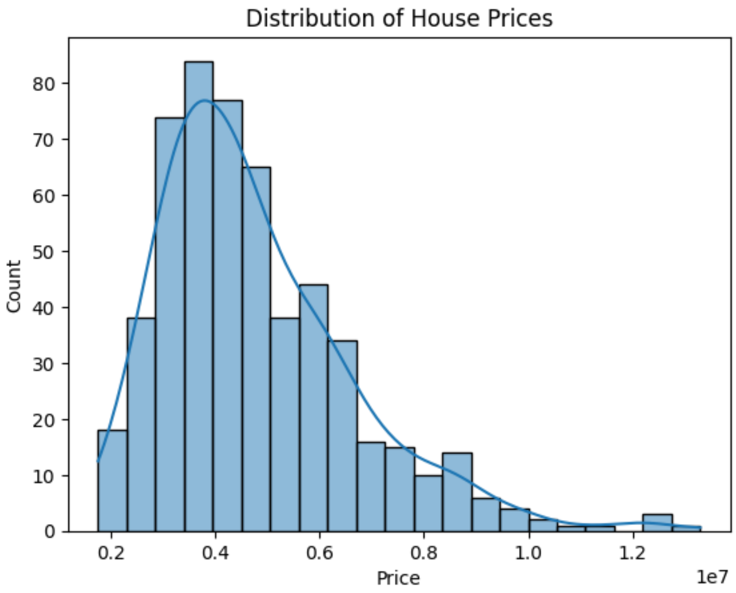
```
# Check for missing values
print("Missing values:\n", df.isnull().sum())

# Check for duplicates
print("Duplicate rows:", df.duplicated().sum())
```

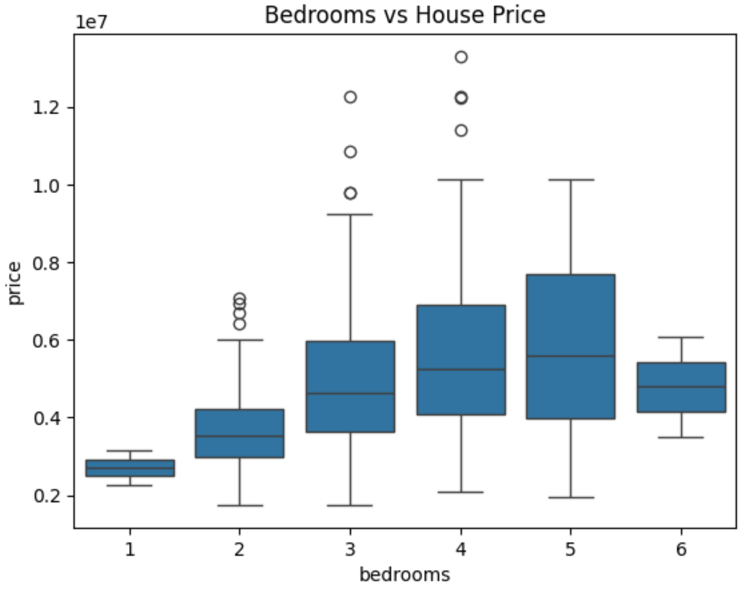
```
Missing values:
 price               0
area                0
bedrooms            0
bathrooms           0
stories             0
mainroad            0
guestroom           0
basement            0
hotwaterheating     0
airconditioning     0
parking             0
prefarea            0
furnishingstatus    0
dtype: int64
Duplicate rows: 0
```

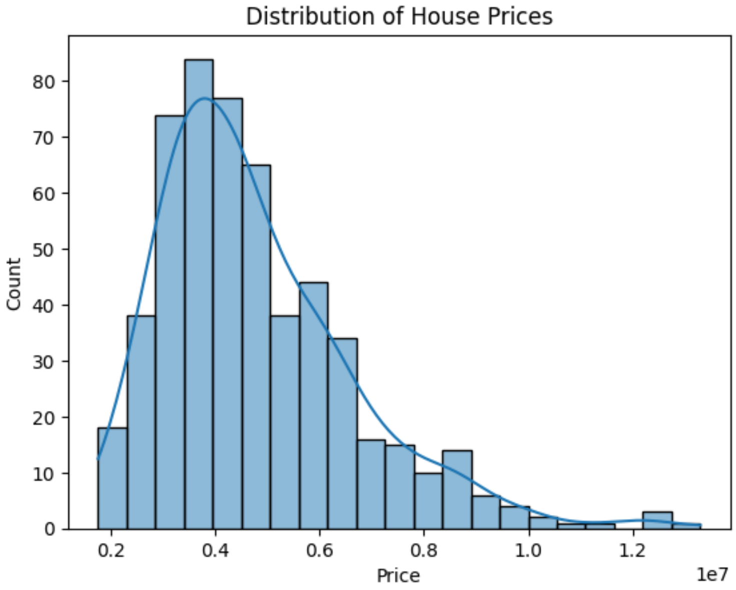
Data Visualization (Modify column names as per your dataset)

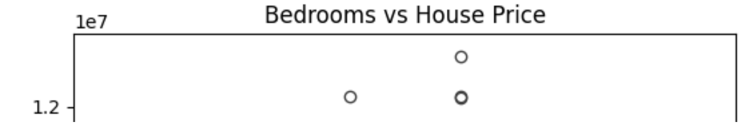
```
import seaborn as sns
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.pyplot as plt

# Example: Distribution of target column (e.g., price)
sns.histplot(df['price'], kde=True)
```

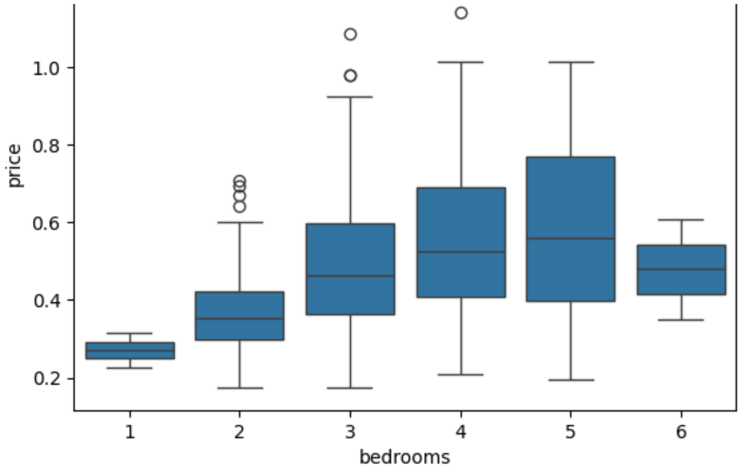
```
plt.title('Distribution of House Prices')
plt.xlabel('Price')
plt.show()

# Example: Boxplot of a feature vs price (adjust columns)
sns.boxplot(x='bedrooms', y='price', data=df)
plt.title('Bedrooms vs House Price')
plt.show()

# Example: Distribution of target column (e.g., price)
sns.histplot(df['price'], kde=True)
plt.title('Distribution of House Prices')
plt.xlabel('Price')
plt.show()

# Example: Boxplot of a feature vs price (adjust columns)
sns.boxplot(x='bedrooms', y='price', data=df)
plt.title('Bedrooms vs House Price')
plt.show()
```


Distribution of House Prices
Count
0
10
20
30
40
50
60
70
80
0.2
0.4
0.6
0.8
1.0
1.2
Price
1e7


Bedrooms vs House Price
1e7
price
0.2
0.4
0.6
0.8
1.0
1.2
1
2
3
4
5
6
bedrooms


Distribution of House Prices
Count
0
10
20
30
40
50
60
70
80
0.2
0.4
0.6
0.8
1.0
1.2
Price
1e7


Bedrooms vs House Price
1e7
1.2



Feature Selection and Target Definition

```
# Choose target and features
target = 'price'
features = df.columns.drop(target)

print("Target:", target)
print("Features:", features)
```

```
Target: price
Features: Index(['area', 'bedrooms', 'bathrooms', 'stories', 'mainroad', 'guestroom',
       'basement', 'hotwaterheating', 'airconditioning', 'parking', 'prefarea',
       'furnishingstatus'],
      dtype='object')
```

Encoding Categorical Variables

```
# Identify categorical columns
categorical_cols = df.select_dtypes(include='object').columns
print("Categorical columns:", categorical_cols.tolist())

# Apply one-hot encoding
df_encoded = pd.get_dummies(df, drop_first=True)
```

```
Categorical columns: ['mainroad', 'guestroom', 'basement', 'hotwaterheating', 'airconditioning', 'prefarea', 'furnishingstatus']
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_scaled = scaler.fit_transform(df_encoded.drop(target, axis=1))
y = df_encoded[target]
```

Train-Test Split

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
```

Model Building (Linear Regression)

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_scor

model = LinearRegression()
```

```
model.fit(X_train, y_train)

# Prediction
y_pred = model.predict(X_test)
```

Model Evaluation

```
print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
print("R² Score:", r2_score(y_test, y_pred))
```

```
Mean Squared Error: 1754318687330.6675
R² Score: 0.6529242642153177
```

Predict from New Input

```
# Example: Replace with values from your actual dataset structure
new_data = {
    'area': 1500,
    'bedrooms': 3,
    'bathrooms': 2,
    'stories': 2,
    'mainroad': 'yes',
    'guestroom': 'no',
    'basement': 'yes',
    'hotwaterheating': 'no',
    'airconditioning': 'yes',
    'parking': 1,
    'prefarea': 'yes',
    'furnishingstatus': 'semi-furnished'
}

# Convert to DataFrame
new_df = pd.DataFrame([new_data])

# Combine with original for consistent encoding
temp_df = pd.concat([df.drop(columns=[target]), new_df], ignore_index=True)

# One-hot encode
temp_encoded = pd.get_dummies(temp_df, drop_first=True)

# Align with training features
temp_encoded = temp_encoded.reindex(columns=df_encoded.drop(target, axis=1).columns, fill_value=0)

# Scale
new_scaled = scaler.transform(temp_encoded.tail(1))

# Predict
prediction = model.predict(new_scaled)
print(f"  Predicted House Price:  {prediction[0]:,.2f}")
```

```
  Predicted House Price:  6,125,612.95
```

Optional: Try Another Model (Random Forest)

```
from sklearn.ensemble import RandomForestRegressor

rf_model = RandomForestRegressor(random_state=42)
rf_model.fit(X_train, y_train)

rf_pred = rf_model.predict(X_test)

print("Random Forest MSE:", mean_squared_error(y_test, rf_pred))
print("Random Forest R²:", r2_score(y_test, rf_pred))
```

```
Random Forest MSE: 1959406221695.9854
Random Forest R²: 0.6123495913214113
```

Deploy a Gradio App

```
!pip install gradio

import gradio as gr

def predict_house_price(area, bedrooms, bathrooms, stories, mainroad, guestroom,
                        basement, hotwaterheating, airconditioning, parking,
                        prefarea, furnishingstatus):

    input_dict = {
        'area': area,
        'bedrooms': bedrooms,
        'bathrooms': bathrooms,
        'stories': stories,
        'mainroad': mainroad,
        'guestroom': guestroom,
        'basement': basement,
        'hotwaterheating': hotwaterheating,
        'airconditioning': airconditioning,
        'parking': parking,
        'prefarea': prefarea,
        'furnishingstatus': furnishingstatus
    }

    input_df = pd.DataFrame([input_dict])
    combined_df = pd.concat([df.drop(columns=[target]), input_df], ignore_index=True)
    encoded_df = pd.get_dummies(combined_df, drop_first=True)
    encoded_df = encoded_df.reindex(columns=df_encoded.drop(target, axis=1).columns, fill_value=0)

    scaled_input = scaler.transform(encoded_df.tail(1))
    result = model.predict(scaled_input)[0]

    return f" {result:,.2f}"

# Define input fields
inputs = [
    gr.Number(label="Area (sq ft)"),
    gr.Number(label="Bedrooms"),
    gr.Number(label="Bathrooms"),
    gr.Number(label="Stories"),
    gr.Dropdown(['yes', 'no'], label="Main Road Access"),
    gr.Dropdown(['yes', 'no'], label="Guest Room"),
    gr.Dropdown(['yes', 'no'], label="Basement"),
    gr.Dropdown(['yes', 'no'], label="Hot Water Heating"),
    gr.Dropdown(['yes', 'no'], label="Air Conditioning"),
    gr.Number(label="Parking Spots"),
    gr.Dropdown(['yes', 'no'], label="Preferred Area"),
    gr.Dropdown(['furnished', 'semi-furnished', 'unfurnished'], label="Furnishing Status")
]

# Output field
output = gr.Text(label="Predicted House Price")

# Launch interface
gr.Interface(fn=predict_house_price, inputs=inputs, outputs=output,
             title="  House Price Predictor",
             description="Enter housing details to estimate the price.").launch()
```

```
Requirement already satisfied: gradio in /usr/local/lib/python3.11/dist-packages (5.29.1)
Requirement already satisfied: aiofiles<25.0,>=22.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (24.1.0)
Requirement already satisfied: anyio<5.0,>=3.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (4.9.0)
Requirement already satisfied: fastapi<1.0,>=0.115.2 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.115.12)
Requirement already satisfied: ffmpy in /usr/local/lib/python3.11/dist-packages (from gradio) (0.5.0)
Requirement already satisfied: gradio-client==1.10.1 in /usr/local/lib/python3.11/dist-packages (from gradio) (1.10.1)
Requirement already satisfied: groovy~=0.1 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.1.2)
Requirement already satisfied: httpx>=0.24.1 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.28.1)
Requirement already satisfied: huggingface-hub>=0.28.1 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.31.1)
Requirement already satisfied: jinja2<4.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (3.1.6)
Requirement already satisfied: markupsafe<4.0,>=2.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (3.0.2)
Requirement already satisfied: numpy<3.0,>=1.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (2.0.2)
Requirement already satisfied: orjson~=3.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (3.10.18)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from gradio) (24.2)
Requirement already satisfied: pandas<3.0,>=1.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (2.2.2)
Requirement already satisfied: pillow<12.0,>=8.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (11.2.1)
Requirement already satisfied: pydantic<2.12,>=2.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (2.11.4)
Requirement already satisfied: pydub in /usr/local/lib/python3.11/dist-packages (from gradio) (0.25.1)
Requirement already satisfied: python-multipart>=0.0.18 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.0.20)
Requirement already satisfied: pyyaml<7.0,>=5.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (6.0.2)
Requirement already satisfied: ruff>=0.9.3 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.11.9)
Requirement already satisfied: safehttpx<0.2.0,>=0.1.6 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.1.6)
Requirement already satisfied: semantic-version~=2.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (2.10.0)
Requirement already satisfied: starlette<1.0,>=0.40.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.46.2)
Requirement already satisfied: tomlkit<0.14.0,>=0.12.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.13.2)
Requirement already satisfied: typer<1.0,>=0.12 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.15.3)
Requirement already satisfied: typing-extensions~=4.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (4.13.2)
Requirement already satisfied: uvicorn>=0.14.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.34.2)
Requirement already satisfied: fsspec in /usr/local/lib/python3.11/dist-packages (from gradio-client==1.10.1->gradio) (2025.3.2)
Requirement already satisfied: websockets<16.0,>=10.0 in /usr/local/lib/python3.11/dist-packages (from gradio-client==1.10.1->gradio) (1
Requirement already satisfied: idna>=2.8 in /usr/local/lib/python3.11/dist-packages (from anyio<5.0,>=3.0->gradio) (3.10)
Requirement already satisfied: sniffio>=1.1 in /usr/local/lib/python3.11/dist-packages (from anyio<5.0,>=3.0->gradio) (1.3.1)
Requirement already satisfied: certifi in /usr/local/lib/python3.11/dist-packages (from httpx>=0.24.1->gradio) (2025.4.26)
Requirement already satisfied: httpcore==1.* in /usr/local/lib/python3.11/dist-packages (from httpx>=0.24.1->gradio) (1.0.9)
Requirement already satisfied: h11>=0.16 in /usr/local/lib/python3.11/dist-packages (from httpcore==1.*->httpx>=0.24.1->gradio) (0.16.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.28.1->gradio) (3.18.0)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.28.1->gradio) (2.32.3)
Requirement already satisfied: tqdm>=4.42.1 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.28.1->gradio) (4.67.1)
Requirement already satisfied: hf-xet<2.0.0,>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.28.1->gradio) (1
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas<3.0,>=1.0->gradio) (2.9.0.
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas<3.0,>=1.0->gradio) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas<3.0,>=1.0->gradio) (2025.2)
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.11/dist-packages (from pydantic<2.12,>=2.0->gradio) (0.7
Requirement already satisfied: pydantic-core==2.33.2 in /usr/local/lib/python3.11/dist-packages (from pydantic<2.12,>=2.0->gradio) (2.33
Requirement already satisfied: typing-inspection>=0.4.0 in /usr/local/lib/python3.11/dist-packages (from pydantic<2.12,>=2.0->gradio) (0
```