Upload the Dataset

```
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

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

# Upload the file
uploaded = files.upload()

# Check if any file was uploaded
if uploaded:
    # Get the name of the first uploaded file
    # Assuming you are uploading only one file
    file_name = list(uploaded.keys())[0]

    # Read the Excel file using the actual uploaded file name
    try:
        df = pd.read_excel(file_name)

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

    except Exception as e:
        print(f"An error occurred while reading the file: {e}")
else:
    print("No file was uploaded. Please ensure you select and upload a file.")
```

Choose Files No file chosen

```
No file was uploaded. Please ensure you select and upload a file.
```

Data Exploration

```
import pandas as pd

# Load data into a DataFrame
df = pd.read_csv('Housing.csv')

# Display the first 5 rows
print(df.head())
```

```
      price  area  bedrooms  bathrooms  stories mainroad guestroom basement  \
0  13300000  7420         4          2        3      yes        no       no
1  12250000  8960         4          4        4      yes        no       no
2  12250000  9960         3          2        2      yes        no      yes
3  12215000  7500         4          2        2      yes        no      yes
4  11410000  7420         4          1        2      yes       yes      yes

  hotwaterheating airconditioning  parking prefarea furnishingstatus
0              no             yes        2      yes        furnished
1              no             yes        3       no        furnished
2              no              no        2      yes   semi-furnished
3              no             yes        3      yes        furnished
4              no             yes        2       no        furnished
```

```
# 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', 'airconditior
<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

```
# 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()
```

```
Shape: (545, 13)
Columns: ['price', 'area', 'bedrooms', 'bathrooms', 'stories', 'mainroad', 'guestroom', 'basement', 'hotwaterheating', 'airconditio
<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

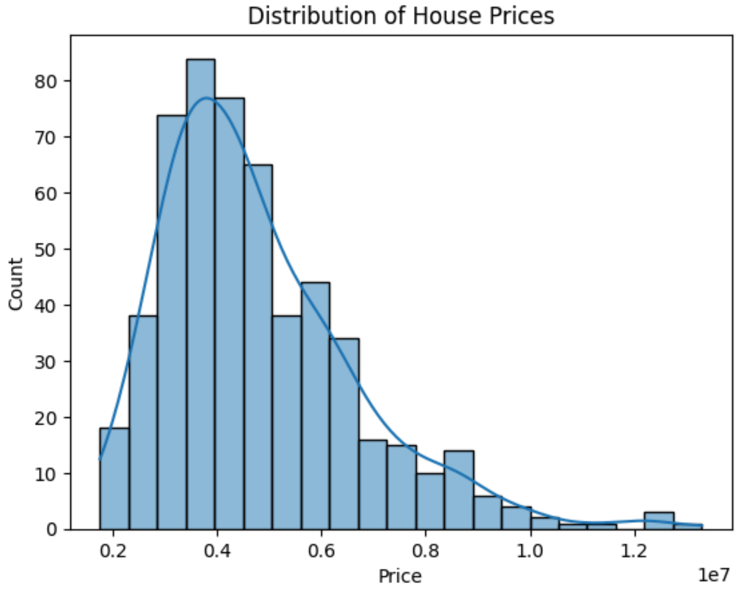
Check for Missing Values and Duplicates

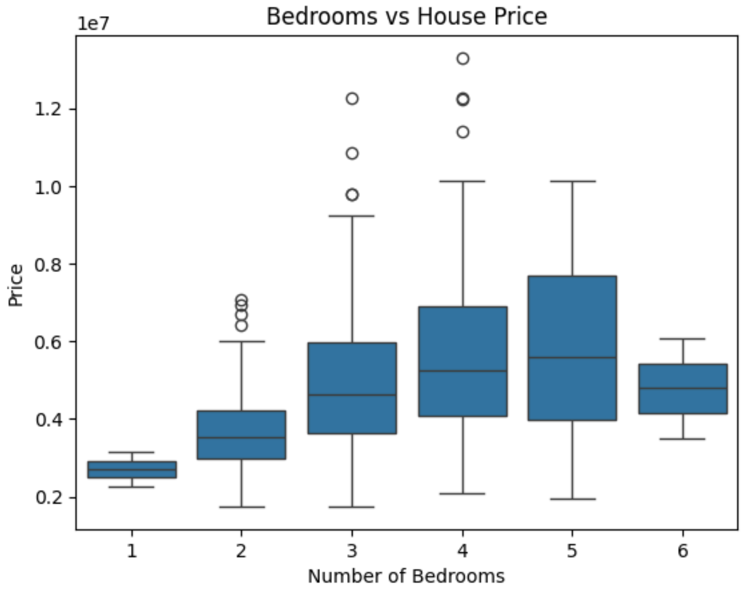
```
# Check for missing values
print(df.isnull().sum())
# Check for duplicates
print("Duplicate rows:", df.duplicated().sum())
```

```
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 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()
```


Distribution of House Prices


Bedrooms vs House Price

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 file

features =df.columns.drop(target)
print("Features:", features)
```

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

One-Hot Encoding

```
import pandas as pd # Ensure pandas is imported in this cell

# Add a check to confirm df is defined
if 'df' in locals() or 'df' in globals():
    # Apply one-hot encoding to the DataFrame object df
    df_encoded = pd.get_dummies(df, drop_first=True)
else:
    print("Error: DataFrame 'df' is not defined. Please run the data loading cell first.")
```

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']
```

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)
```

Model Building

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

Evaluation

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

```
MSE: 1754318687330.6675
R² Score: 0.6529242642153177
```

Make Predictions from New Input

```
import pandas as pd

# Sample new student data
new_student = {
    'school': 'GP',
    'sex': 'F',
    'age': 17,
    'address': 'U',
    'famsize': 'GT3',
    'Pstatus': 'A',
    'Medu': 4,
    'Fedu': 3,
    'Mjob': 'health',
    '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,
    'G1': 14,
    'G2': 15
}

# Convert to DataFrame
new_student_df = pd.DataFrame([new_student])

# Apply preprocessing (ensure this matches the training preprocessing)
# For example: encoding categorical variables, scaling, etc.
# This step will vary based on how the model was trained.
```

Convert to DataFrame and Encode

```
# Sample new student data
new_student = {
    'school': 'GP',
    'sex': 'F',
    'age': 17,
    'address': 'U',
    'famsize': 'GT3',
    'Pstatus': 'A',
    'Medu': 4,
    'Fedu': 3,
    'Mjob': 'health',
    '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,
    'G1': 14,
    'G2': 15
}

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

Predict the Final Grade

```
# 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_score # Import metrics if you want to evaluate later

# Define and train the model
model = LinearRegression()
model.fit(X_train, y_train)

# Now you can proceed to the prediction cell:
# predicted_grade = model.predict(new_input_scaled)
# print("🎓 Predicted Final Grade (G3):", round(predicted_grade[0], 2))
```

```
▾ LinearRegression ⓘ ?
LinearRegression()
```

Deployment-Building an Interactive App

```
!pip install gradio
```

```
Collecting semantic-version~=2.0 (from gradio)
  Downloading semantic_version-2.10.0-py2.py3-none-any.whl.metadata (9.7 kB)
Collecting starlette<1.0,>=0.40.0 (from gradio)
  Downloading starlette-0.46.2-py3-none-any.whl.metadata (6.2 kB)
Collecting tomlkit<0.14.0,>=0.12.0 (from gradio)
  Downloading tomlkit-0.13.2-py3-none-any.whl.metadata (2.7 kB)
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)
Collecting uvicorn>=0.14.0 (from gradio)
  Downloading uvicorn-0.34.2-py3-none-any.whl.metadata (6.5 kB)
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->grad
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) (
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) (4.6
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas<3.0,>=1.0->gradio) (
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
Requirement already satisfied: pydantic-core==2.33.2 in /usr/local/lib/python3.11/dist-packages (from pydantic<2.12,>=2.0->gradio)
Requirement already satisfied: typing-inspection>=0.4.0 in /usr/local/lib/python3.11/dist-packages (from pydantic<2.12,>=2.0->grad
Requirement already satisfied: click>=8.0.0 in /usr/local/lib/python3.11/dist-packages (from typer<1.0,>=0.12->gradio) (8.2.0)
Requirement already satisfied: shellingham>=1.3.0 in /usr/local/lib/python3.11/dist-packages (from typer<1.0,>=0.12->gradio) (1.5
Requirement already satisfied: rich>=10.11.0 in /usr/local/lib/python3.11/dist-packages (from typer<1.0,>=0.12->gradio) (13.9.4)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas<3.0,>=1.0-
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.11/dist-packages (from rich>=10.11.0->typer<1.0,>=0
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.11/dist-packages (from rich>=10.11.0->typer<1.0,>
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->huggingface-hub
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests->huggingface-hub>=0.28
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.11/dist-packages (from markdown-it-py>=2.2.0->rich>=10.11.0->t
Downloading gradio-5.29.1-py3-none-any.whl (54.1 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 54.1/54.1 MB 12.5 MB/s eta 0:00:00
Downloading gradio_client-1.10.1-py3-none-any.whl (323 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 323.1/323.1 kB 18.9 MB/s eta 0:00:00
Downloading aiofiles-24.1.0-py3-none-any.whl (15 kB)
Downloading fastapi-0.115.12-py3-none-any.whl (95 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 95.2/95.2 kB 7.8 MB/s eta 0:00:00
Downloading groovy-0.1.2-py3-none-any.whl (14 kB)
Downloading python_multipart-0.0.20-py3-none-any.whl (24 kB)
Downloading ruff-0.11.10-py3-none-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (11.6 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 11.6/11.6 MB 71.8 MB/s eta 0:00:00
Downloading safehttpx-0.1.6-py3-none-any.whl (8.7 kB)
Downloading semantic_version-2.10.0-py2.py3-none-any.whl (15 kB)
Downloading starlette-0.46.2-py3-none-any.whl (72 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 72.0/72.0 kB 5.8 MB/s eta 0:00:00
Downloading tomlkit-0.13.2-py3-none-any.whl (37 kB)
Downloading uvicorn-0.34.2-py3-none-any.whl (62 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 62.5/62.5 kB 5.2 MB/s eta 0:00:00
Downloading ffmpy-0.5.0-py3-none-any.whl (6.0 kB)
Downloading pydub-0.25.1-py2.py3-none-any.whl (32 kB)
Installing collected packages: pydub, uvicorn, tomlkit, semantic-version, ruff, python-multipart, groovy, ffmpy, aiofiles, starlet
Successfully installed aiofiles-24.1.0 fastapi-0.115.12 ffmpy-0.5.0 gradio-5.29.1 gradio-client-1.10.1 groovy-0.1.2 pydub-0.25.1 p
```

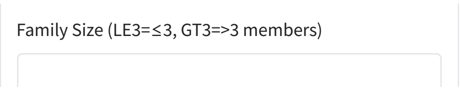
Create a Prediction Function

```
import gradio as gr
```

Create the Gradio Interface

```
import gradio as gr

# Define your prediction function
def predict_final_grade(
    school, sex, age, address, famsize, Pstatus, Medu, Fedu,
    Mjob, Fjob, reason, guardian, traveltime, studytime, failures,
    schoolsup, famsup, paid, activities, nursery, higher,
    internet, romantic, famrel, freetime, goout, Dalc, Walc,
    health, absences, G1, G2
):
    # Your preprocessing and prediction logic here
    # For demonstration, returning a placeholder value
    return f"🎯 Predicted Final Grade (G3): {15.0:.2f}"

# Define the input components
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)"),
    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)")
]

# Create the Gradio interface
demo = gr.Interface(
    fn=predict_final_grade,
    inputs=inputs,
    outputs="text",
    title="🎓 Student Performance Predictor",
    description="Enter academic and demographic info to predict the final grade (G3) of a student."
)

# Launch the interface
demo.launch()
```

```
It looks like you are running Gradio on a hosted a Jupyter notebook. For the Gradio app to work, sharing must be enabled. Automatica

Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://7bd16a579a489717f9.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
```



Family Size (LE3=≤3, GT3=>3 members)

Upload the Dataset

```
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

Load the Dataset

```
import pandas as pd

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

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

Data Exploration

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

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

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

# Summary statistics
df.describe()
```

```
Shape of the dataset: (545, 13)
Columns: ['price', 'area', 'bedrooms', 'bathrooms', 'stories', 'mainroad', 'guestroom', 'basement', 'hotwaterheating', 'airconditior
<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

Data Cleaning

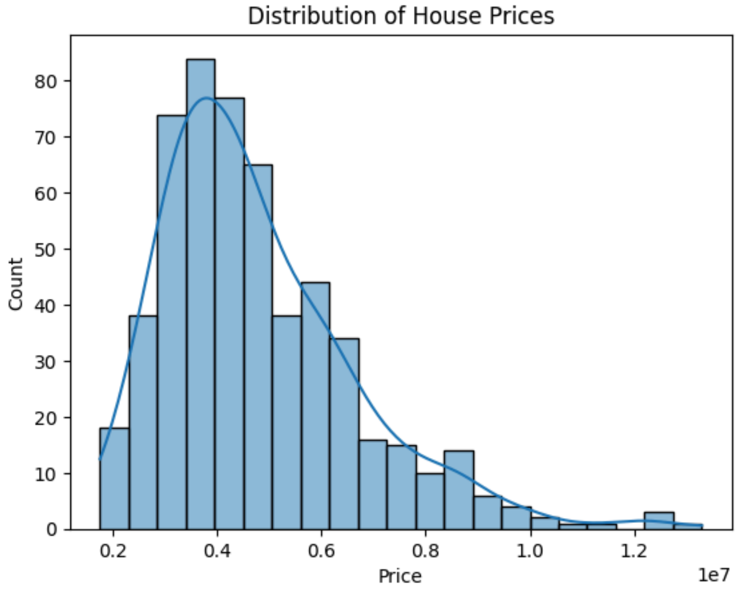
```
# Check for missing values
print("Missing values:\n", df.isnull().sum())
```

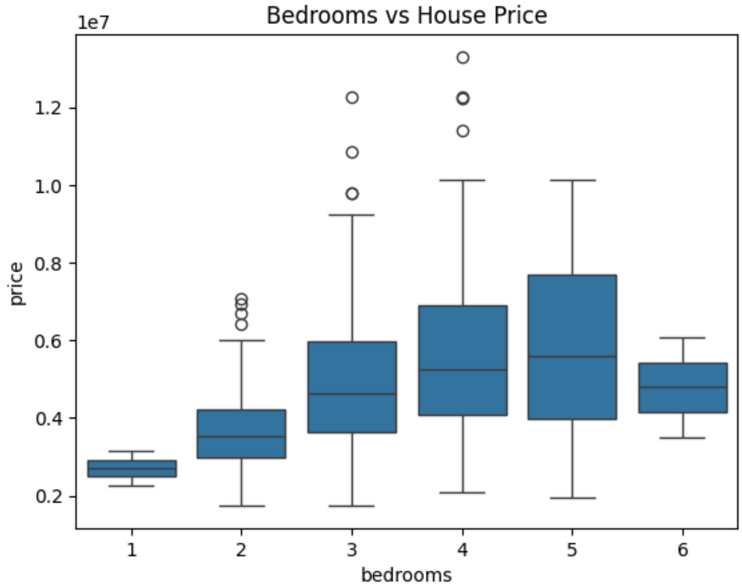
```
# Check for duplicates
print("Duplicate rows:", df.duplicated().sum())
```

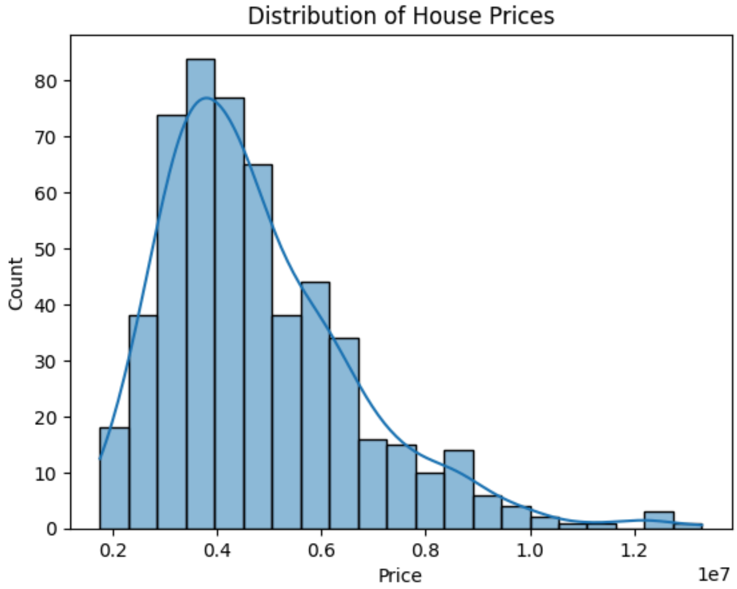
```
Missing values:
 price               0
area                0
bedrooms            0
bathrooms           0
stories             0
mainroad            0
guestroom           0
basement            0
hotwaterheating     0
airconditioning     0
parking             0
prefarea            0
furnishingstatus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

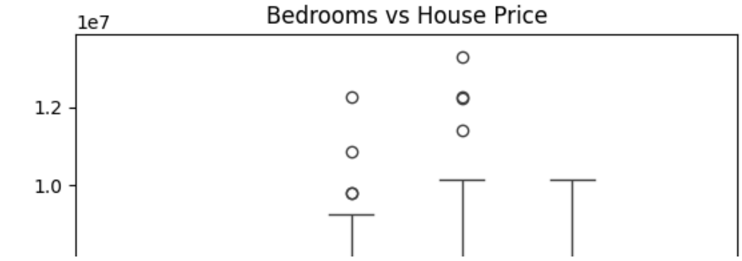
Data Visualization (Modify column names as per your dataset)

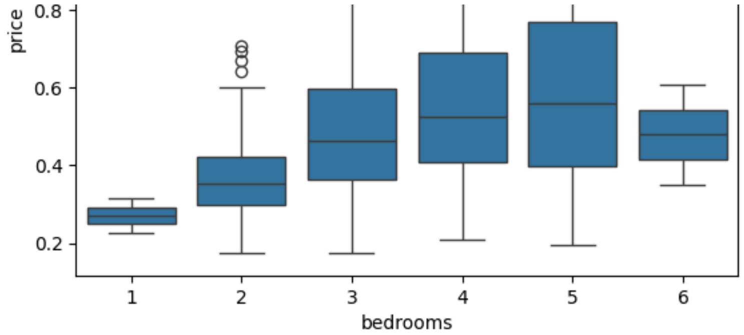
```
import seaborn as sns
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.pyplot as plt

# 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
1.0
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]
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-33-19255e302daf> in <cell line: 0>()
      2 
      3 scaler = StandardScaler()
----> 4 X_scaled = scaler.fit_transform(df_encoded.drop(target, axis=1))
      5 y = df_encoded[target]

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

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_score

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)
```