Upload the Dataset

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

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving full_house_price_dataset.csv to full_house_price_dataset.csv
```

Load the Dataset

```
import pandas as pd
df = pd.read_csv("full_house_price_dataset.csv")
df.head()
```

	ID	Area	Bedrooms	Bathrooms	Floors	YearBuilt	Location	Garage	DistanceToCityCenter	P
0	1	2146	5	1	1	1997	Urban	1	26.75	514
1	2	1418	5	4	3	1974	Suburban	0	34.45	328
2	3	3297	2	3	1	1985	Rural	1	37.46	830
3	4	2031	1	2	1	1957	Rural	0	10.74	127
4	5	3845	4	3	2	2022	Suburban	1	43.37	335

Data Exploration

```
df.info()
df.describe()
df.columns
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 10 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   ID                    1000 non-null   int64  
 1   Area                  1000 non-null   int64  
 2   Bedrooms              1000 non-null   int64  
 3   Bathrooms             1000 non-null   int64  
 4   Floors                1000 non-null   int64  
 5   YearBuilt             1000 non-null   int64  
 6   Location              1000 non-null   object 
 7   Garage                1000 non-null   int64  
 8   DistanceToCityCenter  1000 non-null   float64
 9   Price                 1000 non-null   int64  
dtypes: float64(1), int64(8), object(1)
memory usage: 78.3+ KB
```

```
Index(['ID', 'Area', 'Bedrooms', 'Bathrooms', 'Floors', 'YearBuilt',
       'Location', 'Garage', 'DistanceToCityCenter', 'Price'],
      dtype='object')
```

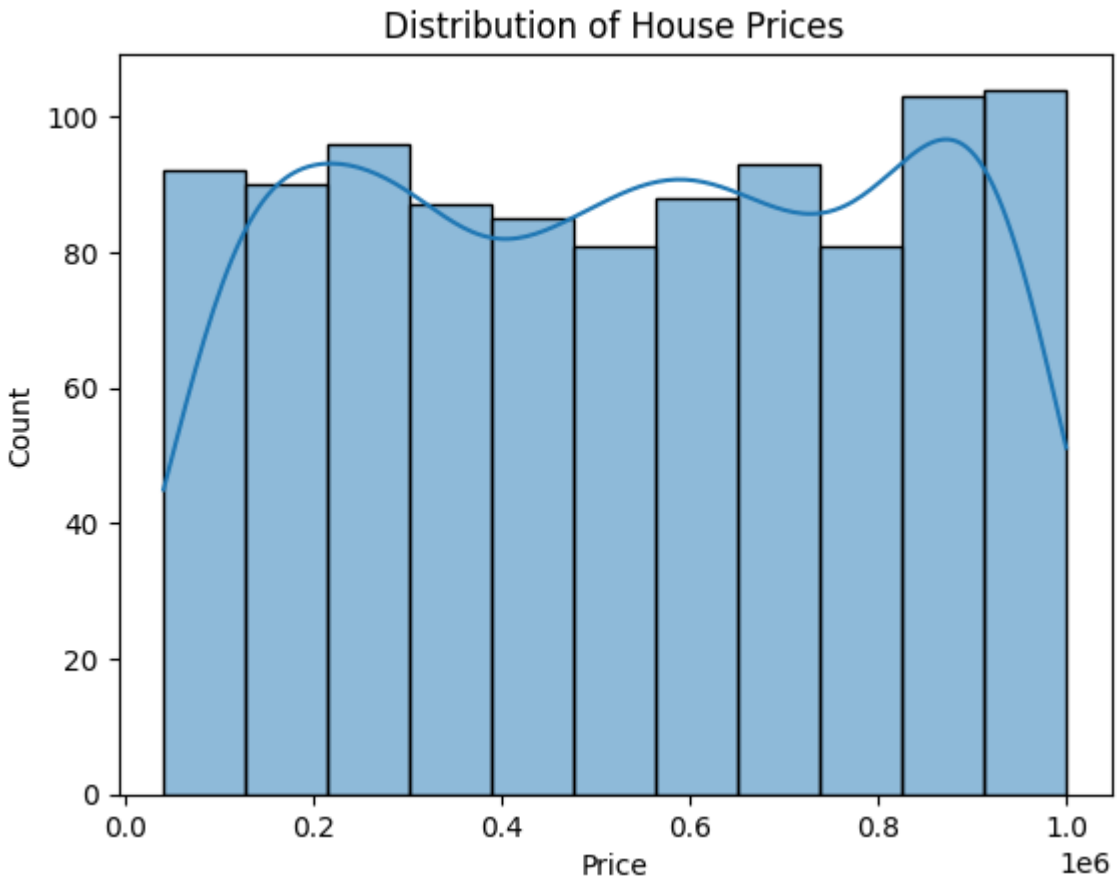
Check for Missing Values and Duplicates

```
print(df.isnull().sum())
print("Duplicates:", df.duplicated().sum())
```

```
ID                        0
Area                      0
Bedrooms                  0
Bathrooms                 0
Floors                    0
YearBuilt                 0
Location                  0
Garage                    0
DistanceToCityCenter      0
Price                     0
dtype: int64
Duplicates: 0
```

Visualize a Few Features

```
import seaborn as sns
import matplotlib.pyplot as plt
sns.histplot(df['Price'], kde=True)
plt.title("Distribution of House Prices")
plt.show()
```



Identify Target and Features

```
target = 'Price'
features = df.drop(columns=[target]).columns
```

Convert Categorical Columns to Numerical

```
df.select_dtypes(include='object').columns
```

```
Index(['Location'], dtype='object')
```

One-Hot Encoding

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

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaled_features = scaler.fit_transform(df.drop(columns=[target]))
```

Train-Test Split

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(scaled_features, df[target], te
```

Model Building

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

Evaluation

Make Predictions from New Input

```
from sklearn.metrics import mean_squared_error, r2_score
y_pred = model.predict(X_test)
print("R² Score:", r2_score(y_test, y_pred))
print("MSE:", mean_squared_error(y_test, y_pred))
```

```
R² Score: -0.017629100142256382
MSE: 83500859652.60652
```

```python
# Save trained columns after preprocessing
trained_columns = df.drop(columns='Price').columns

# Example input (must match the format used in training)
original_input = {
    'Bedrooms': 3,
    'Bathrooms': 2,
    'Area': 1500,
    'Garage': 1,
    'Location': 'Suburb'  # Replace with actual category used in your dataset
}

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

# One-hot encode and align columns
new_df_encoded = pd.get_dummies(new_df)
new_df_encoded = new_df_encoded.reindex(columns=trained_columns, fill_value=0)

# Scale input
scaled_input = scaler.transform(new_df_encoded)

# Predict
prediction = model.predict(scaled_input)
print("Predicted Price:", prediction[0])
```

```
Predicted Price: 1346256.048805823
```

Convert to DataFrame and Encode

```python
# Example new input (adjust keys based on your dataset)
new_input_data = {
    'Bedrooms': 3,
    'Bathrooms': 2,
    'Area': 1500,
    'Garage': 1,
    'Location': 'Suburb'  # Replace with actual category value from your dataset
}

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

# One-hot encode the input to match training format
new_df_encoded = pd.get_dummies(new_df)

# Ensure columns match training data
new_df_encoded = new_df_encoded.reindex(columns=trained_columns, fill_value=0)

# Scale the input using the same scaler from training
scaled_input = scaler.transform(new_df_encoded)

# Predict with the trained model
prediction = model.predict(scaled_input)
print("Predicted Price:", prediction[0])
```

```
Predicted Price: 1346256.048805823
```

```
trained_columns = df.drop(columns='Price').columns
```

Predict the Final Grade (House Price)

```
# 1. Example new input (adjust based on your dataset's actual columns and values)
new_input_data = {
    'Bedrooms': 3,
    'Bathrooms': 2,
    'Area': 1500,
    'Garage': 1,
    'Location': 'Suburb'  # Replace with a valid category from your data
}

# 2. Convert the input dictionary to a DataFrame
new_df = pd.DataFrame([new_input_data])

# 3. One-hot encode categorical variables (same as training)
new_df_encoded = pd.get_dummies(new_df)

# 4. Align new data with the columns used during training
new_df_encoded = new_df_encoded.reindex(columns=trained_columns, fill_value=0)

# 5. Scale the new input using the trained scaler
scaled_input = scaler.transform(new_df_encoded)

# 6. Predict the house price using the trained model
final_prediction = model.predict(scaled_input)
print("Predicted Price:", final_prediction[0])
```

```
Predicted Price: 1346256.048805823
```

Deployment - Building an Interactive App

```
!pip install gradio
import gradio as gr
```

```
Collecting gradio
  Downloading gradio-5.29.0-py3-none-any.whl.metadata (16 kB)
Collecting aiofiles<25.0,>=22.0 (from gradio)
  Downloading aiofiles-24.1.0-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: anyio<5.0,>=3.0 in /usr/local/lib/python3.11/dist-
packages (from gradio) (4.9.0)
Collecting fastapi<1.0,>=0.115.2 (from gradio)
  Downloading fastapi-0.115.12-py3-none-any.whl.metadata (27 kB)
Collecting ffmpy (from gradio)
  Downloading ffmpy-0.5.0-py3-none-any.whl.metadata (3.0 kB)
Collecting gradio-client==1.10.0 (from gradio)
  Downloading gradio_client-1.10.0-py3-none-any.whl.metadata (7.1 kB)
Collecting groovy~=0.1 (from gradio)
  Downloading groovy-0.1.2-py3-none-any.whl.metadata (6.1 kB)
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) (0.30.2)
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)
Collecting pydub (from gradio)
  Downloading pydub-0.25.1-py2.py3-none-any.whl.metadata (1.4 kB)
Collecting python-multipart>=0.0.18 (from gradio)
  Downloading python_multipart-0.0.20-py3-none-any.whl.metadata (1.8 kB)
Requirement already satisfied: pyyaml<7.0,>=5.0 in /usr/local/lib/python3.11/dist-
packages (from gradio) (6.0.2)
Collecting ruff>=0.9.3 (from gradio)
  Downloading ruff-0.11.8-py3-none-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (25 kB)
Collecting safehttpx<0.2.0,>=0.1.6 (from gradio)
  Downloading safehttpx-0.1.6-py3-none-any.whl.metadata (4.2 kB)
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==1.10.0->gradio) (2025.3.2)
Requirement already satisfied: websockets<16.0,>=10.0 in /usr/local/lib/python3.11/
dist-packages (from gradio-client==1.10.0->gradio) (15.0.1)
```

Create a Prediction Function

```
def predict_price(beds, baths, area, garage):
    input_data = [[beds, baths, area, garage]]
    scaled = scaler.transform(input_data)
    return model.predict(scaled)[0]
```

Create the Gradio Interface

```
interface = gr.Interface(fn=predict_price,
                         inputs=["number", "number", "number", "number"],
                         outputs="number",
                         title="House Price Predictor")
interface.launch()
```

```
It looks like you are running Gradio on a hosted a Jupyter notebook. For the Gradio
app to work, sharing must be enabled. Automatically setting `share=True` (you can
turn this off by setting `share=False` in `launch()` explicitly).

Colab notebook detected. To show errors in colab notebook, set debug=True in
launch()
* Running on public URL: https://7d8b2270389cc99eef.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 directory to deploy to Hugging
Face Spaces (https://huggingface.co/spaces)
```