


## Upload the Dataset

 Generate

10 random numbers using numpy



Close

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



Choose files raw\_sales.csv

- **raw\_sales.csv**(text/csv) - 1505497 bytes, last modified: 08/05/2025 - 100% done  
Saving raw\_sales.csv to raw\_sales.csv

## Load the Dataset

```
import pandas as pd
```

```
# Load dataset
df = pd.read_csv('/content/raw_sales.csv')
df.head()
```



	datesold	postcode	price	propertyType	bedrooms
0	2007-02-07 00:00:00	2607	525000	house	4
1	2007-02-27 00:00:00	2906	290000	house	3
2	2007-03-07 00:00:00	2905	328000	house	3
3	2007-03-09 00:00:00	2905	380000	house	4
4	2007-03-21 00:00:00	2906	310000	house	3

Next steps:

[Generate code with df](#)[View recommended plots](#)[New interactive sheet](#)

## Data Exploration

```
# Dataset Info
df.info()
```

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

```
# First few records
df.head()
```



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 29580 entries, 0 to 29579
Data columns (total 5 columns):
#   Column          Non-Null Count  Dtype
---  -
0   datesold         29580 non-null  object
1   postcode         29580 non-null  int64
2   price            29580 non-null  int64
3   propertyType     29580 non-null  object
4   bedrooms         29580 non-null  int64
dtypes: int64(3), object(2)
memory usage: 1.1+ MB
```

	datesold	postcode	price	propertyType	bedrooms
0	2007-02-07 00:00:00	2607	525000	house	4
1	2007-02-27 00:00:00	2906	290000	house	3
2	2007-03-07 00:00:00	2905	328000	house	3
3	2007-03-09 00:00:00	2905	380000	house	4
4	2007-03-21 00:00:00	2906	310000	house	3

Next steps:

[Generate code with df](#)[View recommended plots](#)[New interactive sheet](#)

## Check for Missing Values and Duplicates

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

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

```
# Drop duplicates
df = df.drop_duplicates()
```

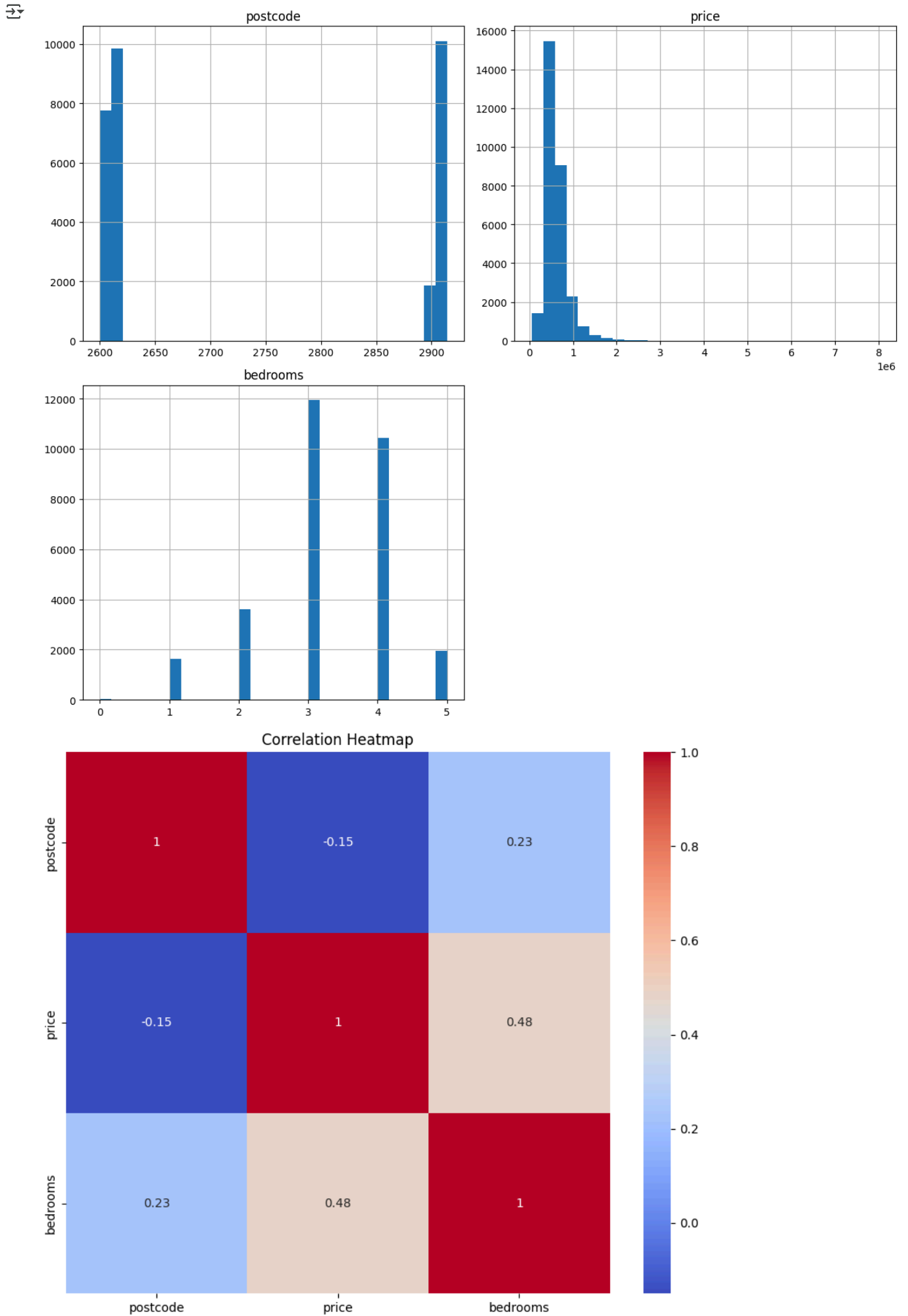
```
Missing Values:
  datesold      0
  postcode      0
  price         0
  propertyType  0
  bedrooms      0
  dtype: int64
  Duplicates: 0
```

### Visualize a Few Features

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

```
# Histogram of numerical columns
df.hist(bins=30, figsize=(12, 10))
plt.tight_layout()
plt.show()
```

```
# Correlation heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(df.corr(numeric_only=True), annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```



## Identify Target and Features

```
import pandas as pd

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

# Display column names
print("Column Names:")
print(df.columns.tolist())
```

Column Names:  
['datesold', 'postcode', 'price', 'propertyType', 'bedrooms']

## Convert Categorical Columns to Numerical

```
# Identify categorical features
categorical_cols = X.select_dtypes(include=['object']).columns.tolist()
print("Categorical Columns:", categorical_cols)
```

Categorical Columns: ['datesold', 'propertyType']

## One-Hot Encoding

```
# One-hot encode categorical features
X_encoded = pd.get_dummies(X, columns=categorical_cols, drop_first=True)
X_encoded.head()
```

5 rows × 3585 columns

	postcode	price	bedrooms	datesold_2007-02-27 00:00:00	datesold_2007-03-07 00:00:00	datesold_2007-03-09 00:00:00	datesold_2007-03-21 00:00:00	datesold_2007-04-04 00:00:00	datesold_2007-04-24 00:00:00	datesold_2007-04-30
0	2607	525000	4	False	False	False	False	False	False	
1	2906	290000	3	True	False	False	False	False	False	
2	2905	328000	3	False	True	False	False	False	False	
3	2905	380000	4	False	False	True	False	False	False	
4	2906	310000	3	False	False	False	True	False	False	

## Feature Scaling

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_encoded)
```

## Train-Test Split

```
# Target and features
target = 'price'
features = ['datesold', 'postcode', 'propertyType', 'bedrooms']

X = df[features]
y = df[target]
```

## Model Building

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
```

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

# Target and features
target = 'price'
features = ['datesold', 'postcode', 'propertyType', 'bedrooms']

X = df[features]
y = df[target]


# Convert 'datesold' to datetime and extract useful features
X['datesold'] = pd.to_datetime(X['datesold'])
X['year'] = X['datesold'].dt.year
X['month'] = X['datesold'].dt.month
X = X.drop('datesold', axis=1)

# One-hot encode categorical column
X = pd.get_dummies(X, columns=['propertyType'], drop_first=True)

# Scale features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42
)

# Train model
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
```

 <ipython-input-28-14e9053df833>:17: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead

See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\\_guide/indexing.html#returning-a-view-versus-a-copy](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)  
X['datesold'] = pd.to\_datetime(X['datesold'])


RandomForestRegressor ⓘ ?  
RandomForestRegressor(random\_state=42)

## Evaluation

```
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("Mean Squared Error:", mean_squared_error(y_test, y_pred))
```

 R² Score: 0.596013553913987  
Mean Squared Error: 27298040328.29305

## Make Predictions from New Input

```
# Example dictionary (adjust keys to match your actual features)
new_data = {
    'Bedrooms': 3,
    'Bathrooms': 2,
    'SqFt': 1500,
    'Location': 'Downtown', # Example categorical
    # Add other features as needed...
}
new_df = pd.DataFrame([new_data])
```

## Convert to DataFrame and Encode

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
```

```
# Load and preprocess
df = pd.read_csv("raw_sales.csv")
df['datesold'] = pd.to_datetime(df['datesold'])
df['year'] = df['datesold'].dt.year
df['month'] = df['datesold'].dt.month
df = df.drop(columns=['datesold'])

# Features and target
X = df.drop(columns=['price'])
y = df['price']

# Define categorical and numerical columns
categorical_cols = ['propertyType']
numerical_cols = _
```

### Predict the Final Price

```
import pandas as pd
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split

# Load and preprocess
df = pd.read_csv("raw_sales.csv")
df['datesold'] = pd.to_datetime(df['datesold'])
df['year'] = df['datesold'].dt.year
df['month'] = df['datesold'].dt.month
df = df.drop(columns=['datesold'])

# Features and target
X = df.drop(columns=['price'])
y = df['price']

# Define preprocessing
categorical_cols = ['propertyType']
numerical_cols = ['bedrooms', 'postcode', 'year', 'month']

preprocessor = ColumnTransformer([
    ('num', StandardScaler(), numerical_cols),
    ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_cols)
])

# Build pipeline
pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('model', RandomForestRegressor(n_estimators=100, random_state=42))
])

# Split and train
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
pipeline.fit(X_train, y_train)

# ✅ Predict new house price
new_data = pd.DataFrame([
    {'bedrooms': 4,
     'postcode': 3000,
     'propertyType': 'Unit',
     'year': 2025,
     'month': 5}
])

predicted_price = pipeline.predict(new_data)
print("Predicted House Price: ${:,.2f}".format(predicted_price[0]))
```

 Predicted House Price: \$687,765.85

### Deployment - Building an Interactive App (Gradio)

```
!pip install gradio

import gradio as gr

def predict_price(**kwargs):
    input_df = pd.DataFrame([kwargs])
    input_encoded = pd.get_dummies(input_df)
    input_encoded = input_encoded.reindex(columns=X_encoded.columns, fill_value=0)
```

```
input_scaled = scaler.transform(input_encoded)
prediction = model.predict(input_scaled)
return "${:.2f}".format(prediction[0])
```

```
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) (13.1)
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) (0.14.0)
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) (2.32.0)
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: 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) (0.7.0)
Requirement already satisfied: pydantic-core==2.33.2 in /usr/local/lib/python3.11/dist-packages (from pydantic<2.12,>=2.0->gradio) (2.33.2)
Requirement already satisfied: typing-inspection>=0.4.0 in /usr/local/lib/python3.11/dist-packages (from pydantic<2.12,>=2.0->gradio) (0.10.0)
Requirement already satisfied: click>=8.0.0 in /usr/local/lib/python3.11/dist-packages (from typer<1.0,>=0.12->gradio) (8.1.8)
Requirement already satisfied: shellingham>=1.3.0 in /usr/local/lib/python3.11/dist-packages (from typer<1.0,>=0.12->gradio) (1.5.4)
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->gradio) (1.17.0)
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,>=0.12->gradio) (3.0.0)
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,>=0.12->gradio) (2.19.2)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->huggingface-hub>=0.28.1->gradio) (3.4.0)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests->huggingface-hub>=0.28.1->gradio) (2.3.0)
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->typer<1.0,>=0.12->gradio) (0.1.2)
Downloading gradio-5.29.0-py3-none-any.whl (54.1 MB)
  54.1/54.1 MB 13.8 MB/s eta 0:00:00
Downloading gradio_client-1.10.0-py3-none-any.whl (322 kB)
  322.9/322.9 kB 21.1 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 6.9 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-0.11.8-py3-none-manylinux_2_17_x86_64_muslmanylinux2014_x86_64.whl (11.5 MB)
  11.5/11.5 MB 98.5 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 5.0 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 4.0 MB/s eta 0:00:00
Downloading ffmpeg-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, ffmpeg, aiofiles, starlette
Successfully installed aiofiles-24.1.0 fastapi-0.115.12 ffmpeg-0.5.0 gradio-5.29.0 gradio-client-1.10.0 groovy-0.1.2 pydub-0.25.1
```

## Create a Prediction Function (Gradio UI)

```
# Customize input widgets for your specific features
input_fields = [
    gr.Textbox(label="Bedrooms"),
    gr.Textbox(label="Bathrooms"),
    gr.Textbox(label="SqFt"),
    gr.Textbox(label="Location"), # Example categorical feature
]

# Add more fields as per your dataset

gr.Interface(fn=predict_price, inputs=input_fields, outputs="text", title="House Price Predictor").launch()
```