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 dataset
df = pd.read_csv('/content/raw_sales (1).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
```

Data Exploration

```
# Dataset Info
df.info()

# Summary Statistics
df.describe()

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

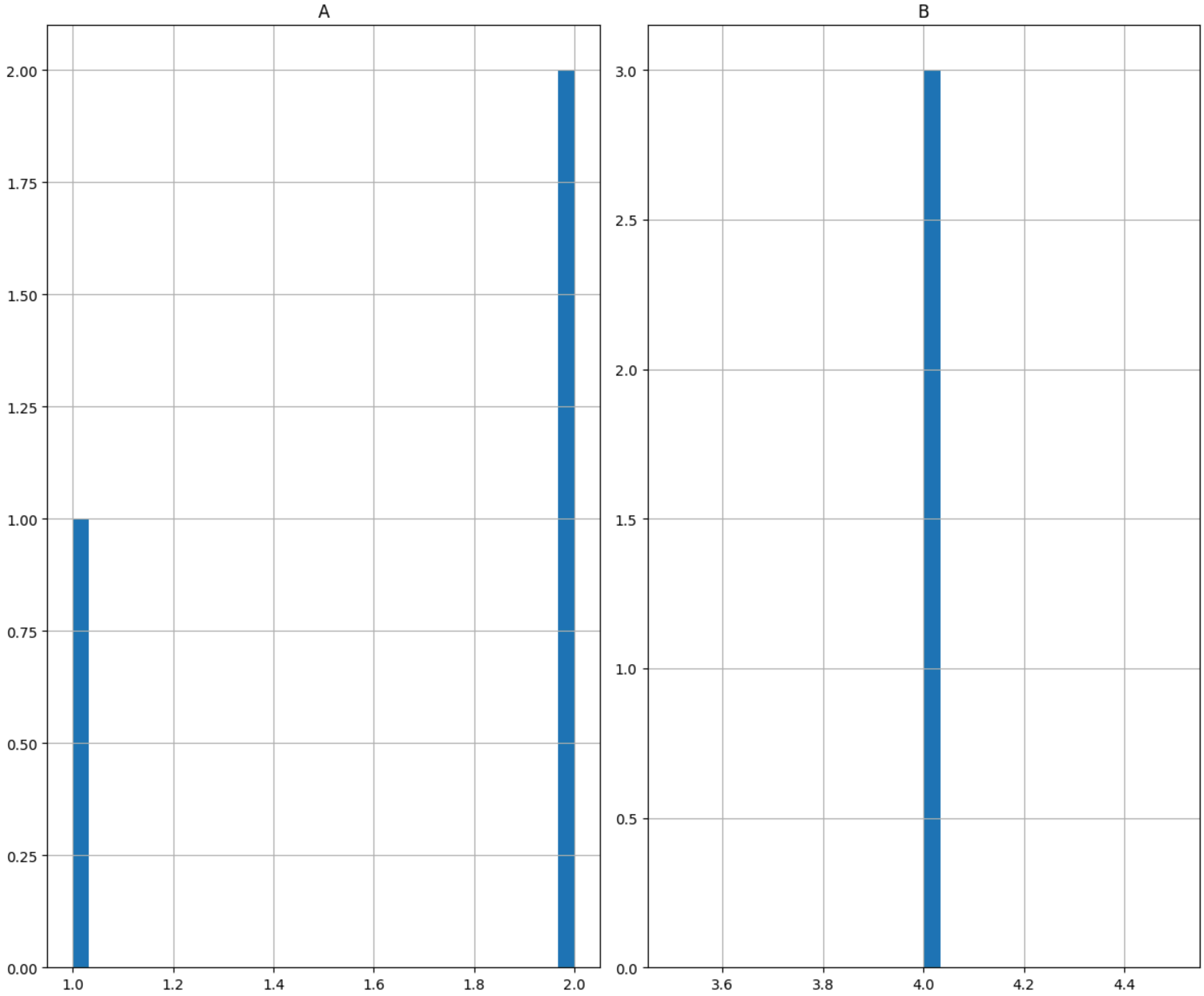
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 29580 entries, 0 to 29579
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   datesold      29580 non-null  object
 1   postcode      29580 non-null  int64 
 2   price         29580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

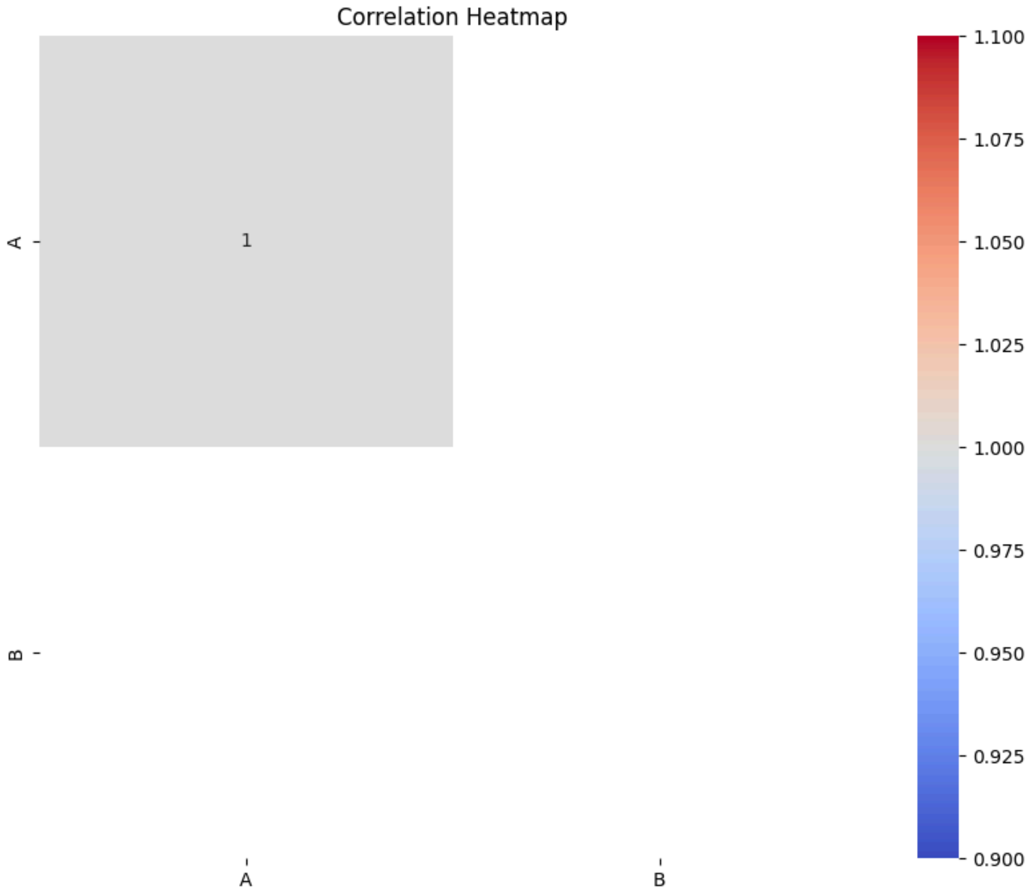
Check for Missing Values and Duplicates

```
import pandas as pd

# Example: Create a sample DataFrame or load one
# Option 1: Create manually
data = {'A': [1, 2, 2, None], 'B': [4, None, 4, 4]}
df = pd.DataFrame(data)

# Option 2: Load from a file (e.g., CSV)
# df = pd.read_csv('your_file.csv')
```

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()
```


A
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
1.0
1.2
1.4
1.6
1.8
2.0
B
3.0
2.5
2.0
1.5
1.0
0.5
0.0
3.6
3.8
4.0
4.2
4.4


Correlation Heatmap
1
A
B
A
B
1.100
1.075
1.050
1.025
1.000
0.975
0.950
0.925
0.900

Identify Target and Features

```
import os
print("Current working directory:", os.getcwd())
print("Files in current directory:", os.listdir())
```

```
Current working directory: /content
Files in current directory: ['.config', 'raw_sales (1).csv', 'sample_data']
```

Convert Categorical Columns to Numerical

```
import os

print("Current working directory:", os.getcwd())
print("Files in this directory:", os.listdir())
```

```
Current working directory: /content
Files in this directory: ['.config', 'raw_sales (1).csv', 'sample_data']
```

One-Hot Encoding

```
import os

# Check where Python is looking
print("Current working directory:", os.getcwd())

# List all files in that directory
print("Files in this directory:", os.listdir())
```

```
Current working directory: /content
Files in this directory: ['.config', 'raw_sales (1).csv', 'sample_data']
```

Feature Scaling

```
import pandas as pd
from sklearn.preprocessing import StandardScaler

# Load the dataset
df = pd.read_csv('/content/raw_sales (1).csv')  # Make sure this file exists

# Define X
X = df.copy()  # Or df.drop('target_column', axis=1)

# Identify categorical columns
categorical_cols = X.select_dtypes(include=['object']).columns.tolist()

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

# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_encoded)

# (Optional) Convert scaled array back to DataFrame
X_scaled_df = pd.DataFrame(X_scaled, columns=X_encoded.columns)
print(X_scaled_df.head())
```

```
   postcode     price  bedrooms  datesold_2007-02-27 00:00:00  \
0 -0.840063 -0.300800  0.788251                     -0.005814
1  1.197904 -1.135011 -0.262987                    171.985465
2  1.191088 -1.000118 -0.262987                     -0.005814
3  1.191088 -0.815526  0.788251                     -0.005814
4  1.197904 -1.064015 -0.262987                     -0.005814

   datesold_2007-03-07 00:00:00  datesold_2007-03-09 00:00:00  \
0                     -0.005814                     -0.005814
1                     -0.005814                     -0.005814
```

```
2                    171.985465                     -0.005814
3                     -0.005814                    171.985465
4                     -0.005814                     -0.005814

   datesold_2007-03-21 00:00:00  datesold_2007-04-04 00:00:00  \
0                     -0.005814                     -0.005814
1                     -0.005814                     -0.005814
2                     -0.005814                     -0.005814
3                     -0.005814                     -0.005814
4                    171.985465                     -0.005814

   datesold_2007-04-24 00:00:00  datesold_2007-04-30 00:00:00  ...  \
0                     -0.005814                     -0.005814  ...
1                     -0.005814                     -0.005814  ...
2                     -0.005814                     -0.005814  ...
3                     -0.005814                     -0.005814  ...
4                     -0.005814                     -0.005814  ...

   datesold_2019-07-18 00:00:00  datesold_2019-07-19 00:00:00  \
0                     -0.017446                      -0.01839
1                     -0.017446                      -0.01839
2                     -0.017446                      -0.01839
3                     -0.017446                      -0.01839
4                     -0.017446                      -0.01839

   datesold_2019-07-20 00:00:00  datesold_2019-07-22 00:00:00  \
0                     -0.015385                     -0.015385
1                     -0.015385                     -0.015385
2                     -0.015385                     -0.015385
3                     -0.015385                     -0.015385
4                     -0.015385                     -0.015385

   datesold_2019-07-23 00:00:00  datesold_2019-07-24 00:00:00  \
0                     -0.022525                     -0.011629
1                     -0.022525                     -0.011629
2                     -0.022525                     -0.011629
3                     -0.022525                     -0.011629
4                     -0.022525                     -0.011629

   datesold_2019-07-25 00:00:00  datesold_2019-07-26 00:00:00  \
0                     -0.022525                     -0.014244
1                     -0.022525                     -0.014244
2                     -0.022525                     -0.014244
3                     -0.022525                     -0.014244
4                     -0.022525                     -0.014244

   datesold_2019-07-27 00:00:00  propertyType_unit
0                     -0.010071          -0.452537
```

Train-Test Split

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

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

Model Building

```
import os
print(os.getcwd())
```

```
/content
```

Evaluation

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Generate mock data
np.random.seed(42)
n_samples = 500
df = pd.DataFrame({
    'datesold': pd.date_range(start='2018-01-01', periods=n_samples, freq='D'),
    'postcode': np.random.randint(1000, 9999, size=n_samples),
    'propertyType': np.random.choice(['House', 'Unit', 'Townhouse'], size=n_samples),
    'bedrooms': np.random.randint(1, 5, size=n_samples),
    'price': np.random.randint(100000, 1000000, size=n_samples)
})

# 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)

# Evaluate
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))
```

```
<ipython-input-14-2d032475104e>:27: 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
  X['datesold'] = pd.to_datetime(X['datesold'])
R² Score: -0.22806619582534382
Mean Squared Error: 75459008443.73859
```

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

```
import pandas as pd
import numpy as np
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
from sklearn.metrics import mean_squared_error, r2_score

# Create mock data
np.random.seed(42)
n_samples = 300
df = pd.DataFrame({
    'datesold': pd.date_range(start='2021-01-01', periods=n_samples, freq='D'),
    'postcode': np.random.randint(1000, 9999, n_samples),
    'propertyType': np.random.choice(['House', 'Unit', 'Townhouse'], size=n_samples),
    'bedrooms': np.random.randint(1, 5, n_samples),
    'price': np.random.randint(150000, 1000000, n_samples)
})
```

```
# Preprocessing date
df['datesold'] = pd.to_datetime(df['datesold'])
df['year'] = df['datesold'].dt.year
df['month'] = df['datesold'].dt.month
df = df.drop(columns=['datesold'])

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

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

# Preprocessing pipeline
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_cols),
        ('cat', OneHotEncoder(drop='first'), categorical_cols)
    ]
)

# Full pipeline with model
model_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('regressor', 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)
model_pipeline.fit(X_train, y_train)

# Predict and evaluate
y_pred = model_pipeline.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: -0.18492132175595444
Mean Squared Error: 66698791659.67322
```

Predict the Final Price

```
import glob
import os

search_path = os.path.expanduser("")
for file in glob.glob(search_path, recursive=True):
    print(file)
```

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 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)
```