HOUSE PRICE PREDICTION

By: Mitali Saini

OUTLINE:

Introduction:

Increasing population in the world people looking to buy a new house as per their budget seem to be conservative and need more market strategy for house agents. As house price increase dramatically every year, there should a system to predict new house price according to the demand of people for house size, Bedroom size and location. This could really help real estate agents to decide house price for their clients. There are several methods proposed to determine house price ranges.

In the new technologically advanced world, there are several proposed methodologies used to predict price. It has been seen that machine learning is the reliable method to achieve this project. It aims to estimate the value of residential properties based on the dataset used which includes Bedrooms, Bathrooms, Size of the house, Zip Code, etc. Linear Regression and its techniques have been used which includes several steps to match needs and can give perfect prediction for the model.

Problem Statement:

The problem statement for the project involves developing model that can accurately estimate the selling price. The goal is to create a reliable tool that can assist homeowners, buyerseal estate agents and investors in making informed decisions about buying, selling or investing in properties.

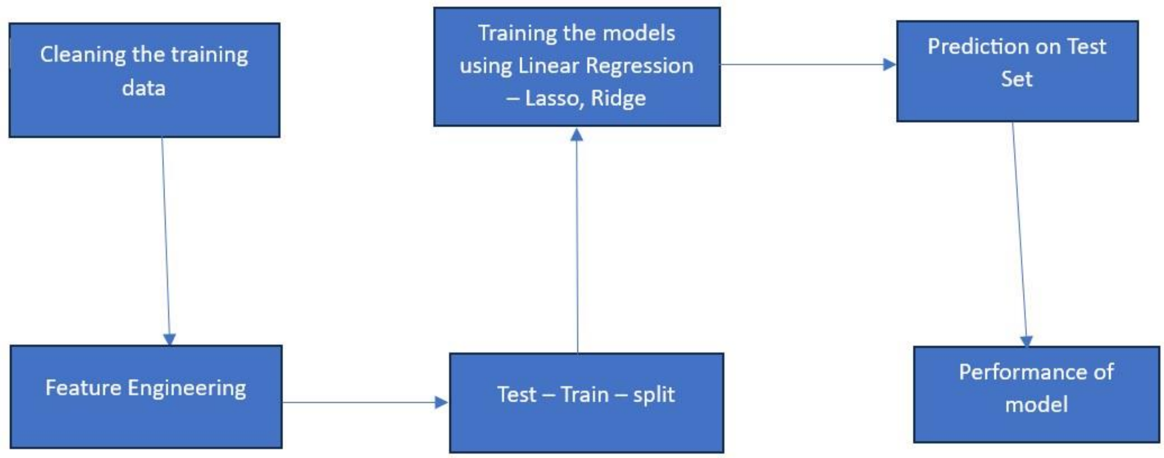
Tools used:

- **Python**: modules: numpy, pandas, matplotlib, seaborn, scikit-learn, pickle, etc.
  framework: Flask

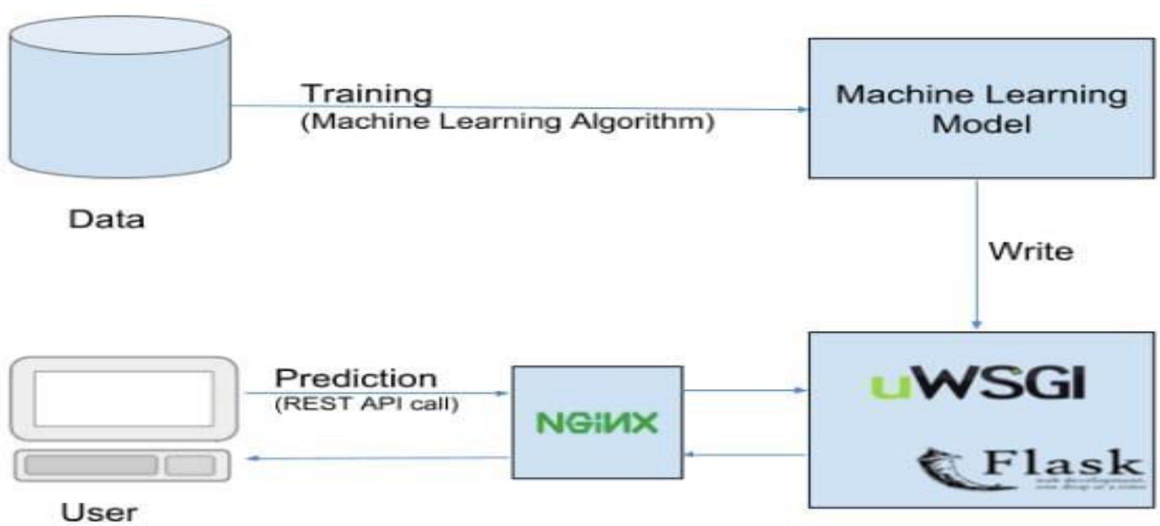
- **Web Development tools**:
  Html, css, javascript

- **Machine Learning Techniques:**
  Regression: Linear Regression, Lasso, Ridge

System Architecture:



System Deployment Approach:



Code and Analysis:

Data Loading:

```
import numpy as np
import pandas as pd
```

```
data = pd.read_csv("train.csv")
data.head()
```

	beds	baths	size	size_units	lot_size	lot_size_units	zip_code	price
0	3	2.5	2590.0	sqft	6000.00	sqft	98144	795000.0
1	4	2.0	2240.0	sqft	0.31	acre	98106	915000.0
2	4	3.0	2040.0	sqft	3783.00	sqft	98107	950000.0
3	4	3.0	3800.0	sqft	5175.00	sqft	98199	1950000.0
4	2	2.0	1042.0	sqft	NaN	NaN	98102	950000.0

```
data.shape
```

```
(2016, 8)
```

EDA:

Data Preprocessing and EDA

```
for column in data.columns:
    print(data[column].value_counts())
    print("*"*20)
```

```
beds
3     645
2     560
4     398
1     256
5     123
6      22
9       5
7       3
8       2
15      1
14      1
Name: count, dtype: int64
********************
baths
2.0    627
1.0    493
2.5    282
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2016 entries, 0 to 2015
Data columns (total 8 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   beds            2016 non-null   int64  
 1   baths           2016 non-null   float64
 2   size            2016 non-null   float64
 3   size_units      2016 non-null   object 
 4   lot_size        1669 non-null   float64
 5   lot_size_units  1669 non-null   object 
 6   zip_code        2016 non-null   int64  
 7   price           2016 non-null   float64
dtypes: float64(4), int64(2), object(2)
memory usage: 126.1+ KB
```

```
data.isna().sum()
```

```
beds                0
baths               0
size                0
size_units          0
lot_size          347
lot_size_units    347
zip_code            0
price               0
dtype: int64
```

```
data.drop(columns=['lot_size','lot_size_units'],inplace=True)
```

```
data.describe()
```

	beds	baths	size	zip_code	price
count	2016.000000	2016.000000	2016.000000	2016.000000	2.016000e+03
mean	2.857639	2.159970	1735.740575	98123.638889	9.636252e+05
std	1.255092	1.002023	920.132591	22.650819	9.440954e+05
min	1.000000	0.500000	250.000000	98101.000000	1.590000e+05
25%	2.000000	1.500000	1068.750000	98108.000000	6.017500e+05
50%	3.000000	2.000000	1560.000000	98117.000000	8.000000e+05
75%	4.000000	2.500000	2222.500000	98126.000000	1.105250e+06
max	15.000000	9.000000	11010.000000	98199.000000	2.500000e+07

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2016 entries, 0 to 2015
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   beds        2016 non-null   int64  
 1   baths       2016 non-null   float64
 2   size        2016 non-null   float64
 3   size_units  2016 non-null   object 
 4   zip_code    2016 non-null   int64  
 5   price       2016 non-null   float64
dtypes: float64(3), int64(2), object(1)
memory usage: 94.6+ KB
```

```
data.head()
```

	beds	baths	size	size_units	zip_code	price
0	3	2.5	2590.0	sqft	98144	795000.0
1	4	2.0	2240.0	sqft	98106	915000.0
2	4	3.0	2040.0	sqft	98107	950000.0
3	4	3.0	3800.0	sqft	98199	1950000.0
4	2	2.0	1042.0	sqft	98102	950000.0

```
# Price per sq feet
```

```
data['price_per_sqft'] = data['price']*100000 / data['size']
```

```
data['price_per_sqft']
```

```
0       3.069498e+07
1       4.084821e+07
2       4.656863e+07
3       5.131579e+07
4       9.117083e+07
            ...
2011    6.642336e+07
2012    6.186727e+07
2013    5.373832e+07
2014    7.421384e+07
2015    3.853801e+07
Name: price_per_sqft, Length: 2016, dtype: float64
```

```
data.describe()
```

	beds	baths	size	zip_code	price	price_per_sqft
count	2016.000000	2016.000000	2016.000000	2016.000000	2.016000e+03	2.016000e+03
mean	2.857639	2.159970	1735.740575	98123.638889	9.636252e+05	5.915851e+07
std	1.255092	1.002023	920.132591	22.650819	9.440954e+05	8.327952e+07
min	1.000000	0.500000	250.000000	98101.000000	1.590000e+05	6.796117e+06
25%	2.000000	1.500000	1068.750000	98108.000000	6.017500e+05	4.452221e+07
50%	3.000000	2.000000	1560.000000	98117.000000	8.000000e+05	5.529762e+07
75%	4.000000	2.500000	2222.500000	98126.000000	1.105250e+06	6.595389e+07
max	15.000000	9.000000	11010.000000	98199.000000	2.500000e+07	3.424658e+09

```
data.shape
```

```
(2016, 7)
```

```
data
```

	beds	baths	size	size_units	zip_code	price	price_per_sqft
0	3	2.5	2590.0	sqft	98144	795000.0	3.069498e+07
1	4	2.0	2240.0	sqft	98106	915000.0	4.084821e+07
2	4	3.0	2040.0	sqft	98107	950000.0	4.656863e+07
3	4	3.0	3800.0	sqft	98199	1950000.0	5.131579e+07
4	2	2.0	1042.0	sqft	98102	950000.0	9.117083e+07

```
data.drop(columns=['size_units'],inplace=True)
```

```
data.drop(columns=['price_per_sqft'],inplace=True)
```

```
data.head()
```

	beds	baths	size	zip_code	price
0	3	2.5	2590.0	98144	795000.0
1	4	2.0	2240.0	98106	915000.0
2	4	3.0	2040.0	98107	950000.0
3	4	3.0	3800.0	98199	1950000.0
4	2	2.0	1042.0	98102	950000.0

```
data.to_csv("final_dataset.csv")
```

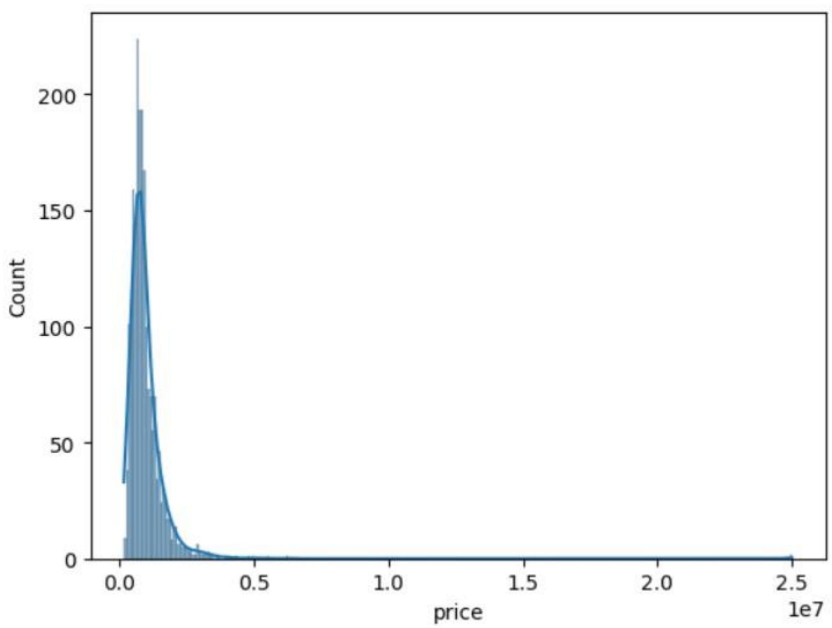
```
one_hot_encoded = pd.get_dummies(data['beds'], prefix='beds')
data_encoded = pd.concat([data, one_hot_encoded], axis=1)
print(data_encoded)
```

```
      beds  baths    size  zip_code      price  beds_1  beds_2  beds_3  \
0        3    2.5  2590.0     98144   795000.0   False   False    True
1        4    2.0  2240.0     98106   915000.0   False   False   False
2        4    3.0  2040.0     98107   950000.0   False   False   False
3        4    3.0  3800.0     98199  1950000.0   False   False   False
4        2    2.0  1042.0     98102   950000.0   False    True   False
...    ...    ...     ...       ...        ...     ...     ...     ...
2011     3    2.0  1370.0     98112   910000.0   False   False    True
2012     1    1.0   889.0     98121   550000.0    True   False   False
2013     4    2.0  2140.0     98199  1150000.0   False   False   False
2014     2    2.0   795.0     98103   590000.0   False    True   False
2015     3    2.0  1710.0     98133   659000.0   False   False    True
```

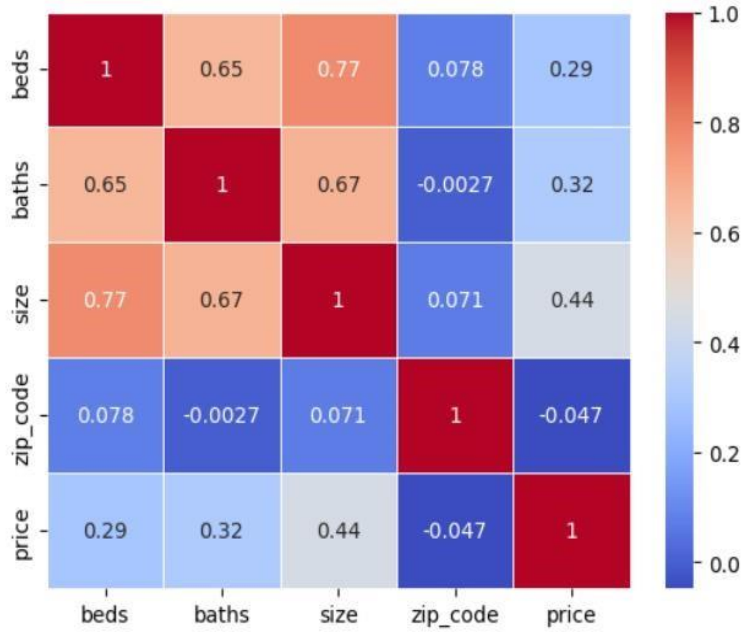
```
import matplotlib.pyplot as plt
import seaborn as sns
```

```
sns.histplot(data['price'], kde=True)
```

```
<AxesSubplot: xlabel='price', ylabel='Count'>
```

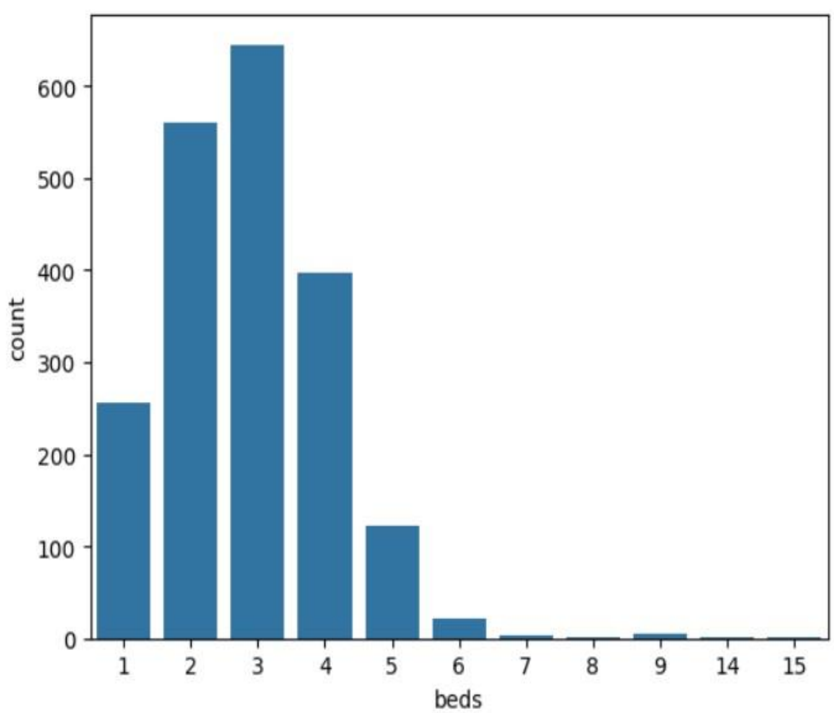


```
sns.heatmap(data.corr(),annot=True, cmap='coolwarm', linewidth=0.5)
plt.show()
```

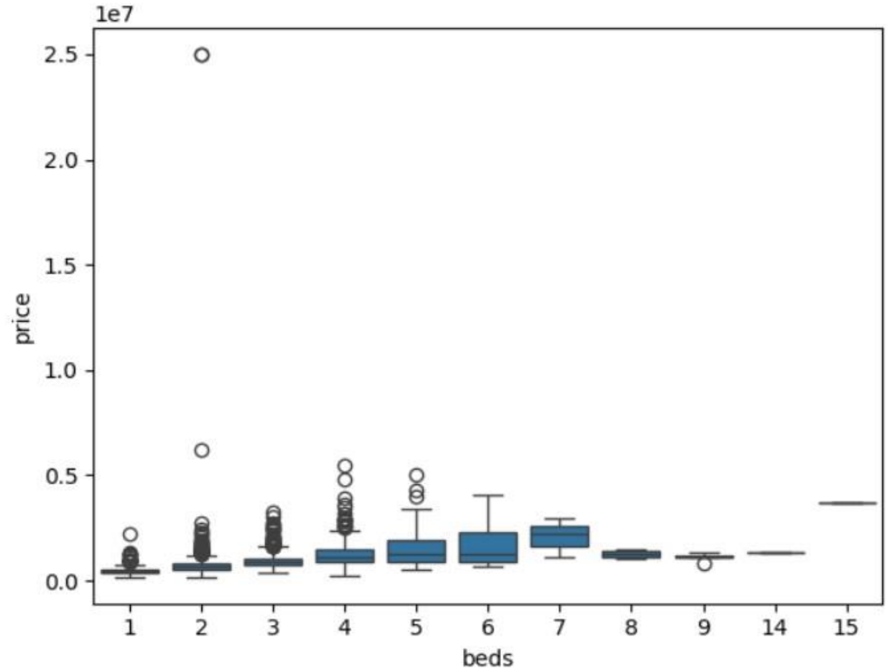



```
sns.countplot(x='beds', data=data)
```

```
<AxesSubplot: xlabel='beds', ylabel='count'>
```



```
sns.boxplot(x='beds', y='price', data=data)
plt.show()
```



Modeling:

```
X = data.drop(columns=['price'])
y = data['price']
```

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression,Lasso,Ridge
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import make_column_transformer
from sklearn.pipeline import make_pipeline
from sklearn.metrics import r2_score
```

```
X_train,X_test,y_train,y_test = train_test_split(X,y, test_size=0.2, random_state=0)
```

```
print(X_train.shape)
print(y_train.shape)
```

```
(1612, 4)
(1612,)
```

```
column_trans = make_column_transformer((OneHotEncoder(), ['beds']), remainder='passthrough')
```

```
scaler = StandardScaler()
```

```
lr = LinearRegression()
```

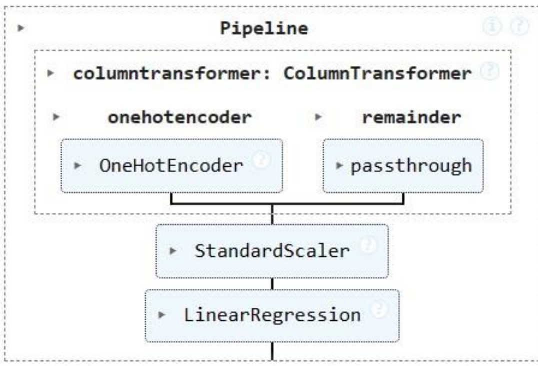
```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
```

```python
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler

# X being feature matrix
scaler = StandardScaler(with_mean=False)
X_scaled = scaler.fit_transform(X)

lr = LinearRegression()
lr.fit(X_scaled, y)
```

```
LinearRegression
LinearRegression()
```

```python
pipe = make_pipeline(column_trans,scaler, lr)
```

```python
pipe.fit(X_train,y_train,)
```




```python
y_pred_lr = pipe.predict(X_test)
```

```python
r2_score(y_test,y_pred_lr)
```

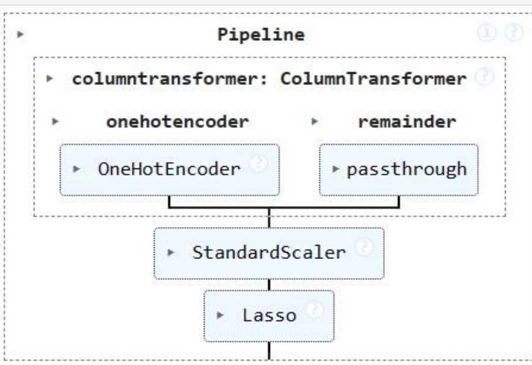
```
0.5746822864697891
```

using Lasso

```python
lasso = Lasso()
```

```python
pipe = make_pipeline(column_trans,scaler, lasso)
```

```python
pipe.fit(X_train,y_train)
```



```python
y_pred_lasso = pipe.predict(X_test)
r2_score(y_test,y_pred_lasso)
```

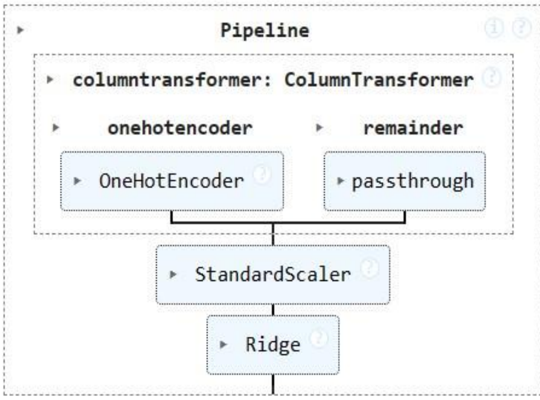
```
0.5746817917321105
```

using Ridge

```python
ridge = Ridge()
```

```python
pipe = make_pipeline(column_trans,scaler, ridge)
```

```python
pipe.fit(X_train,y_train)
```




```python
y_pred_ridge = pipe.predict(X_test)
r2_score(y_test,y_pred_ridge)
```

```
0.5746891139050041
```

```python
print("No Regularization: ", r2_score(y_test,y_pred_lr))
print("Lasso: ", r2_score(y_test,y_pred_lasso))
print("Ridge: ", r2_score(y_test,y_pred_ridge))
```

```
No Regularization:  0.5746822864697891
Lasso:  0.5746817917321105
Ridge:  0.5746891139050041
```

```python
import pickle
```

```python
pickle.dump(pipe, open('RidgeModel.pkl','wb'))
```

Html code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>House Price Prediction</title>
    <style>
        body{
            font-family: Arial, sans-serif;
            margin: 0;
            padding: 0;
            background-color: #f4f4;
        }
        header{
            background-color: #333;
            color: #fff;
            padding: 10px;
            text-align: center;
        }
        main{
            max-width: 800px;
            margin: 20px auto;
            padding: 20px;
            background-color: #fff;
            box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
        }
        footer{
            text-align: center;
            padding: 10px;
            background-color: #333;
            color: #fff;
            position: fixed;
            bottom: 0;
            width: 100px;
        }
        form{
            margin-top: 20px;
        }
        label{
            display: block;
            margin-bottom: 16px;
        }
        select{
```

```
        }
        select{
            width: 100%;
            padding: 8px;
            margin-bottom: 16px;
        }
        button{
            background-color: #333;
            color: #fff;
            padding: 10px;
            border: none;
            cursor: pointer;
        }
        #predictedPrice{
            margin-top: 20px;
            font-weight: bold;
        }
    </style>
</head>
<body>
    <header>
        <h1>House Price Prediction</h1>
    </header>
    <main>
        <p>Welcome to House Price Prediction</p>
<form id="predictionForm">
    <label for="beds">Beds:</label>
    <select id="beds" name="beds">
        <option value"" disabled selected>Select no of bedrooms</option>
        {% for bedroom in bedrooms %}
            <option value="{{ bedroom }}">{{ bedroom }}</option>
        {% endfor %}
    </select>
    <label for="baths">Baths:</label>
    <select id="baths" name="baths">
        <option value"" disabled selected>Select no of bathrooms</option>
        {% for bathroom in bathrooms %}
            <option value="{{ bathroom }}">{{ bathroom }}</option>
        {% endfor %}
    </select>
    <label for="size">Size:</label>
```

```
    <select id="size" name="size">
        <option value"" disabled selected>Select size of the House</option>
        {% for house_size in sizes %}
            <option value="{{ house_size }}">{{ house_size }}</option>
        {% endfor %}
    </select>
    <label for="zip_code">Zip Code:</label>
    <select id="zip_code" name="zip_code">
        <option value"" disabled selected>Select zip code</option>
        {% for zip_code in zip_codes %}
            <option value="{{ zip_code }}">{{ zip_code }}</option>
        {% endfor %}
    </select>
    <button type="button" onclick="sendData()">Predict Price</button>
    <div id="predictedPrice"></div>
</form>
    </main>
    <footer>
        <p>&copy; 2024 House Price Prediction. All Rights Reserved.</p>
    </footer>
    <script>
        function fetchOptions(endpoint, dropdownId){
            fetch(endpoint)
                .then(response => response.json())
                .then(data => {
                    const dropdown = document.getElementById(dropdownId);
                    dropdown.innerHTML = '<option value="" disabled selected>Select an option</option>';
                    data.forEach(option => {
                        const optionElement = document.createElement('option');
                        optionElement.value = option;
                        optionElement.textContent = option;
                        dropdown.appendChild(optionElement);
                    });
                });
        }
        window.onload = function(){
            fetchOptions('/bedrooms', 'beds');
            fetchOptions('/bathrooms', 'baths');
            fetchOptions('/sizes', 'size');
            fetchOptions('/zip_codes', 'zip_code');
```

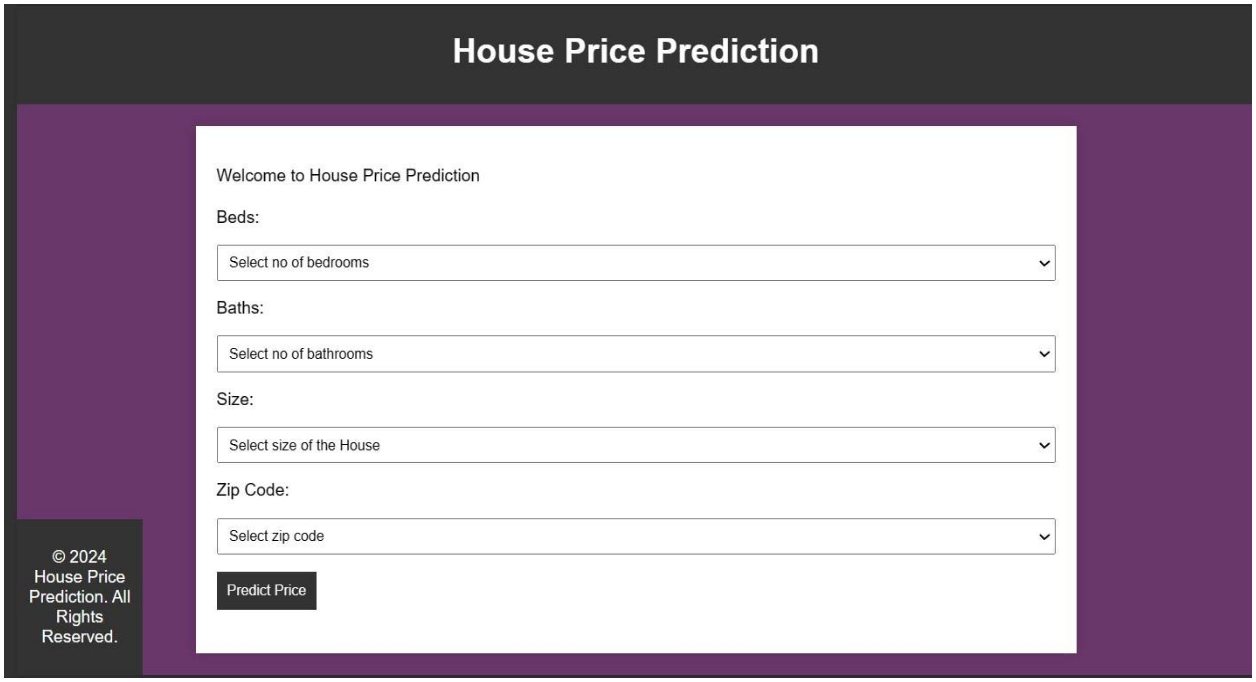
```
            fetchOptions('/zip_codes', 'zip_code');
        };
        function sendData() {
            const form = document.getElementById('predictionForm');
            const formData = new FormData(form);
            fetch('/predict', {
                method: 'POST',
                body: formData
            })
            .then(response => response.text())
            .then(price => {
                document.getElementById("predictedPrice").innerHTML = "Price: INR " + price;
            });
        }
    </script>
</body>
</html>--
```

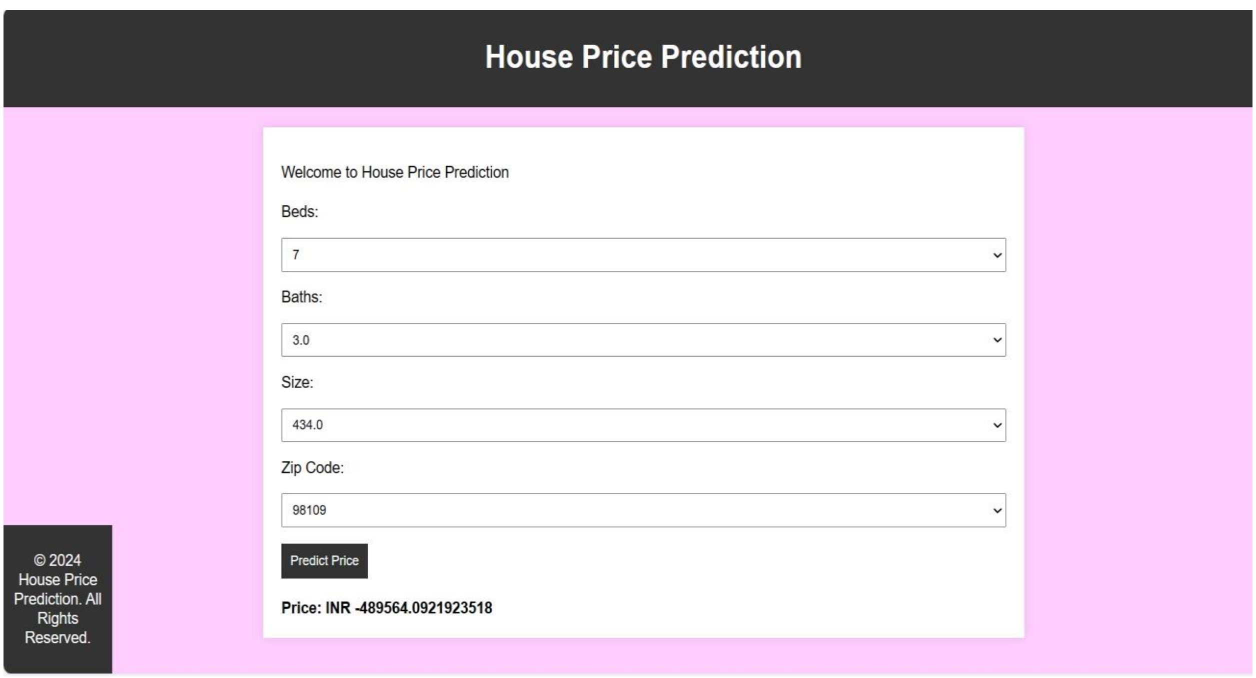
Main file:

```
from flask import Flask, render_template, request, jsonify
import pandas as pd
import pickle

app = Flask(__name__, template_folder='templates')
data = pd.read_csv('final_dataset.csv')
pipe = pickle.load(open("RidgeModel.pkl", 'rb'))

@app.route('/index')
def index():
    bedrooms = sorted(data['beds'].unique())
    bathrooms = sorted(data['baths'].unique())
    sizes = sorted(data['size'].unique())
    zip_codes = sorted(data['zip_code'].unique())

    return render_template('house.html', bedrooms=bedrooms, bathrooms=bathrooms, sizes=sizes, zip_codes=zip_codes)

@app.route('/predict',methods=['POST'])
def predict():
    bedrooms = request.form.get('beds')
    bathrooms = request.form.get('baths')
    size = request.form.get('size')
    zipcode = request.form.get('zip_code')

    #create a dataframe with input data
    input_data = pd.DataFrame([[bedrooms, bathrooms, size, zipcode]], columns=['beds', 'baths', 'size', 'zip_code'])
    print("Input data: ")
    print(input_data)

    # Handle unknown categories in the input data
    for column in input_data.columns:
        unknown_categories = set(input_data[column]) - set(data[column].unique())
        if unknown_categories:
            input_data[column] = input_data[column].replace(unknown_categories, data[column].mode()[0])
    print("Preprocessed Input Data: ")
    print(input_data)


    # predict the price
    prediction = pipe.predict(input_data)[0]

    return str(prediction)
from flask import Flask, render_template, request, jsonify
import pandas as pd
```

```python
app = Flask(__name__)
data = pd.read_csv('final_dataset.csv')
pipe = pickle.load(open("RidgeModel.pkl", 'rb'))

@app.route('/')
def index():
    bedrooms = sorted(data['beds'].unique())
    bathrooms = sorted(data['baths'].unique())
    sizes = sorted(data['size'].unique())
    zip_codes = sorted(data['zip_code'].unique())

    return render_template('house.html', bedrooms=bedrooms, bathrooms=bathrooms, sizes=sizes, zip_codes=zip_codes)

@app.route('/predict',methods=['POST'])
def predict():
    bedrooms = request.form.get('beds')
    bathrooms = request.form.get('baths')
    size = request.form.get('size')
    zipcode = request.form.get('zip_code')

    #create a dataframe with input data
    input_data = pd.DataFrame([[bedrooms, bathrooms, size, zipcode]], columns=['beds', 'baths', 'size', 'zip_code'])
    print("Input data: ")
    print(input_data)

    # convert 'baths' column to numeric with errors='coerce'
    input_data['baths'] = pd.to_numeric(input_data['baths'], errors='coerce')

    # convert input data to numeric types
    input_data = input_data.astype({'beds': int, 'baths': float, 'size': float, 'zip_code': int})

    # Handle unknown categories in the input data
    for column in input_data.columns:
        unknown_categories = set(input_data[column]) - set(data[column].unique())
        if unknown_categories:
            print(f"Unknown categories in {column}: {unknown_categories}")
            input_data[column] = input_data[column].replace(unknown_categories, data[column].mode()[0])
    print("Preprocessed Input Data: ")
    print(input_data)


    # predict the price
    prediction = pipe.predict(input_data)[0]

    return str(prediction)

if __name__ == "__main__":
    app.run(debug=True)

if __name__ == "__main__":
    app.run(debug=True)
```

Result:


House Price Prediction
Welcome to House Price Prediction
Beds:
Select no of bedrooms
Baths:
Select no of bathrooms
Size:
Select size of the House
Zip Code:
Select zip code
Predict Price
© 2024 House Price Prediction. All Rights Reserved.


House Price Prediction
Welcome to House Price Prediction
Beds:
7
Baths:
3.0
Size:
434.0
Zip Code:
98109
Predict Price
Price: INR -489564.0921923518
© 2024 House Price Prediction. All Rights Reserved.

Conclusion:

The results have shown the predicted price of a house with a particular zip code, number of bedrooms, no of bathrooms and size of house.

Future Scope:

- Geospatial Analysis and Visualization
- Integration with Blockchain Technology
- Fine-Tuning of models with Time Series
- Integration of External APIs
- Expansion of International Markets