# HOUSE PRICE PREDICTION

- Algorithm: Linear Regression,random forest regression ,decision tree regression, gradient boosting regressor
- Description: Predict house prices based on features like area, number of bedrooms, and location.
- For dataset-here

In [359…

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.preprocessing import LabelEncoder,StandardScaler
from sklearn.metrics import mean_squared_error
import warnings
warnings.filterwarnings('ignore')
```

In [360…

```
#loading the datasets
df=pd.read_csv('data.csv')
df
```

Out[360]:

| | date | price | bedrooms | bathrooms | sqft_living | sqft_lot | floors | waterfront | view | condition | sqft_above | sqft_basement | yr_ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 2014-05-02 00:00:00 | 3.130000e+05 | 3.0 | 1.50 | 1340 | 7912 | 1.5 | 0 | 0 | 3 | 1340 | 0 | |
| **1** | 2014-05-02 00:00:00 | 2.384000e+06 | 5.0 | 2.50 | 3650 | 9050 | 2.0 | 0 | 4 | 5 | 3370 | 280 | |
| **2** | 2014-05-02 00:00:00 | 3.420000e+05 | 3.0 | 2.00 | 1930 | 11947 | 1.0 | 0 | 0 | 4 | 1930 | 0 | |
| **3** | 2014-05-02 00:00:00 | 4.200000e+05 | 3.0 | 2.25 | 2000 | 8030 | 1.0 | 0 | 0 | 4 | 1000 | 1000 | |
| **4** | 2014-05-02 00:00:00 | 5.500000e+05 | 4.0 | 2.50 | 1940 | 10500 | 1.0 | 0 | 0 | 4 | 1140 | 800 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **4595** | 2014-07-09 00:00:00 | 3.081667e+05 | 3.0 | 1.75 | 1510 | 6360 | 1.0 | 0 | 0 | 4 | 1510 | 0 | |
| **4596** | 2014-07-09 00:00:00 | 5.343333e+05 | 3.0 | 2.50 | 1460 | 7573 | 2.0 | 0 | 0 | 3 | 1460 | 0 | |
| **4597** | 2014-07-09 00:00:00 | 4.169042e+05 | 3.0 | 2.50 | 3010 | 7014 | 2.0 | 0 | 0 | 3 | 3010 | 0 | |
| **4598** | 2014-07-10 00:00:00 | 2.034000e+05 | 4.0 | 2.00 | 2090 | 6630 | 1.0 | 0 | 0 | 3 | 1070 | 1020 | |
| **4599** | 2014-07-10 00:00:00 | 2.206000e+05 | 3.0 | 2.50 | 1490 | 8102 | 2.0 | 0 | 0 | 4 | 1490 | 0 | |

4600 rows × 18 columns

In [361…

```
df.head()
```

Out[361]:

| | date | price | bedrooms | bathrooms | sqft_living | sqft_lot | floors | waterfront | view | condition | sqft_above | sqft_basement | yr_built |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 2014-05-02 00:00:00 | 313000.0 | 3.0 | 1.50 | 1340 | 7912 | 1.5 | 0 | 0 | 3 | 1340 | 0 | 1955 |
| **1** | 2014-05-02 00:00:00 | 2384000.0 | 5.0 | 2.50 | 3650 | 9050 | 2.0 | 0 | 4 | 5 | 3370 | 280 | 1921 |
| **2** | 2014-05-02 00:00:00 | 342000.0 | 3.0 | 2.00 | 1930 | 11947 | 1.0 | 0 | 0 | 4 | 1930 | 0 | 1966 |
| **3** | 2014-05-02 00:00:00 | 420000.0 | 3.0 | 2.25 | 2000 | 8030 | 1.0 | 0 | 0 | 4 | 1000 | 1000 | 1963 |
| **4** | 2014-05-02 00:00:00 | 550000.0 | 4.0 | 2.50 | 1940 | 10500 | 1.0 | 0 | 0 | 4 | 1140 | 800 | 1976 |

In [362…

```
df.tail()
```

Out[362]:

| | date | price | bedrooms | bathrooms | sqft_living | sqft_lot | floors | waterfront | view | condition | sqft_above | sqft_basement | y |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **4595** | 2014-07-09 00:00:00 | 308166.666667 | 3.0 | 1.75 | 1510 | 6360 | 1.0 | 0 | 0 | 4 | 1510 | 0 | |
| **4596** | 2014-07-09 00:00:00 | 534333.333333 | 3.0 | 2.50 | 1460 | 7573 | 2.0 | 0 | 0 | 3 | 1460 | 0 | |
| **4597** | 2014-07-09 00:00:00 | 416904.166667 | 3.0 | 2.50 | 3010 | 7014 | 2.0 | 0 | 0 | 3 | 3010 | 0 | |
| **4598** | 2014-07-10 00:00:00 | 203400.000000 | 4.0 | 2.00 | 2090 | 6630 | 1.0 | 0 | 0 | 3 | 1070 | 1020 | |
| **4599** | 2014-07-10 00:00:00 | 220600.000000 | 3.0 | 2.50 | 1490 | 8102 | 2.0 | 0 | 0 | 4 | 1490 | 0 | |

In [363…

```
print("The Number of rows",df.shape[0])
print("The Number of columns",df.shape[1])
```

```
The Number of rows 4600
The Number of columns 18
```

In [364…

```
df.size
```

Out[364]:

```
82800
```

In [365…

```
#lets checks the mumerical and categorical feature
Categorical_feature=[feature for feature in df.columns if df[feature].dtype=='O']
Numerical_feature=[feature for feature in df.columns if df[feature].dtype!='O']
print("The Categorical feature is =",Categorical_feature)
print("The Numerical feature is =",Numerical_feature)
```

```
The Categorical feature is = ['date', 'street', 'city', 'statezip', 'country']
The Numerical feature is = ['price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors', 'waterfront'
, 'view', 'condition', 'sqft_above', 'sqft_basement', 'yr_built', 'yr_renovated']
```

In [366…

```
df.isnull().sum()
```

Out[366]:

```
date             0
price            0
bedrooms         0
bathrooms        0
sqft_living      0
sqft_lot         0
floors           0
waterfront       0
view             0
condition        0
sqft_above       0
sqft_basement    0
yr_built         0
yr_renovated     0
street           0
city             0
statezip         0
country          0
dtype: int64
```

In [367…

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4600 entries, 0 to 4599
Data columns (total 18 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   date           4600 non-null   object 
 1   price          4600 non-null   float64
 2   bedrooms       4600 non-null   float64
 3   bathrooms      4600 non-null   float64
 4   sqft_living    4600 non-null   int64  
 5   sqft_lot       4600 non-null   int64  
 6   floors         4600 non-null   float64
 7   waterfront     4600 non-null   int64  
 8   view           4600 non-null   int64  
 9   condition      4600 non-null   int64  
 10  sqft_above     4600 non-null   int64  
 11  sqft_basement  4600 non-null   int64  
 12  yr_built       4600 non-null   int64  
 13  yr_renovated   4600 non-null   int64  
 14  street         4600 non-null   object 
 15  city           4600 non-null   object 
 16  statezip       4600 non-null   object 
 17  country        4600 non-null   object 
dtypes: float64(4), int64(9), object(5)
memory usage: 647.0+ KB
```

In [368…

```python
print(df.duplicated().sum())
```

```
0
```

In [369…

```python
df.describe()
```

Out[369]:

| | price | bedrooms | bathrooms | sqft_living | sqft_lot | floors | waterfront | view | condition | sqft_abov |
|---|---|---|---|---|---|---|---|---|---|---|
| **count** | 4.600000e+03 | 4600.000000 | 4600.000000 | 4600.000000 | 4.600000e+03 | 4600.000000 | 4600.000000 | 4600.000000 | 4600.000000 | 4600.0000 |
| **mean** | 5.519630e+05 | 3.400870 | 2.160815 | 2139.346957 | 1.485252e+04 | 1.512065 | 0.007174 | 0.240652 | 3.451739 | 1827.2654 |
| **std** | 5.638347e+05 | 0.908848 | 0.783781 | 963.206916 | 3.588444e+04 | 0.538288 | 0.084404 | 0.778405 | 0.677230 | 862.1689 |
| **min** | 0.000000e+00 | 0.000000 | 0.000000 | 370.000000 | 6.380000e+02 | 1.000000 | 0.000000 | 0.000000 | 1.000000 | 370.0000 |
| **25%** | 3.228750e+05 | 3.000000 | 1.750000 | 1460.000000 | 5.000750e+03 | 1.000000 | 0.000000 | 0.000000 | 3.000000 | 1190.0000 |
| **50%** | 4.609435e+05 | 3.000000 | 2.250000 | 1980.000000 | 7.683000e+03 | 1.500000 | 0.000000 | 0.000000 | 3.000000 | 1590.0000 |
| **75%** | 6.549625e+05 | 4.000000 | 2.500000 | 2620.000000 | 1.100125e+04 | 2.000000 | 0.000000 | 0.000000 | 4.000000 | 2300.0000 |
| **max** | 2.659000e+07 | 9.000000 | 8.000000 | 13540.000000 | 1.074218e+06 | 3.500000 | 1.000000 | 4.000000 | 5.000000 | 9410.0000 |

In [370…

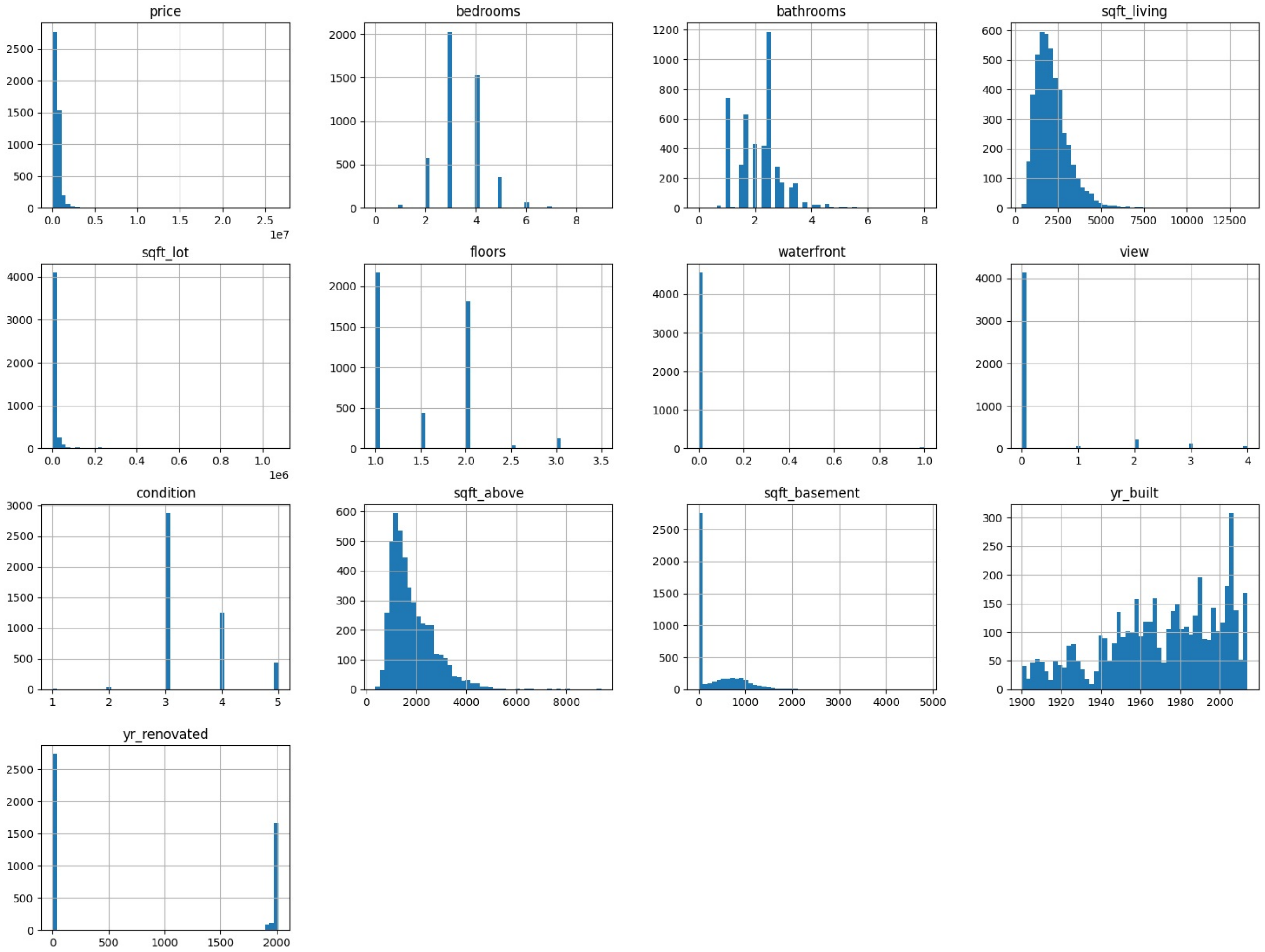
```python
df.columns
```

Out[370]:

```
Index(['date', 'price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot',
       'floors', 'waterfront', 'view', 'condition', 'sqft_above',
       'sqft_basement', 'yr_built', 'yr_renovated', 'street', 'city',
       'statezip', 'country'],
      dtype='object')
```

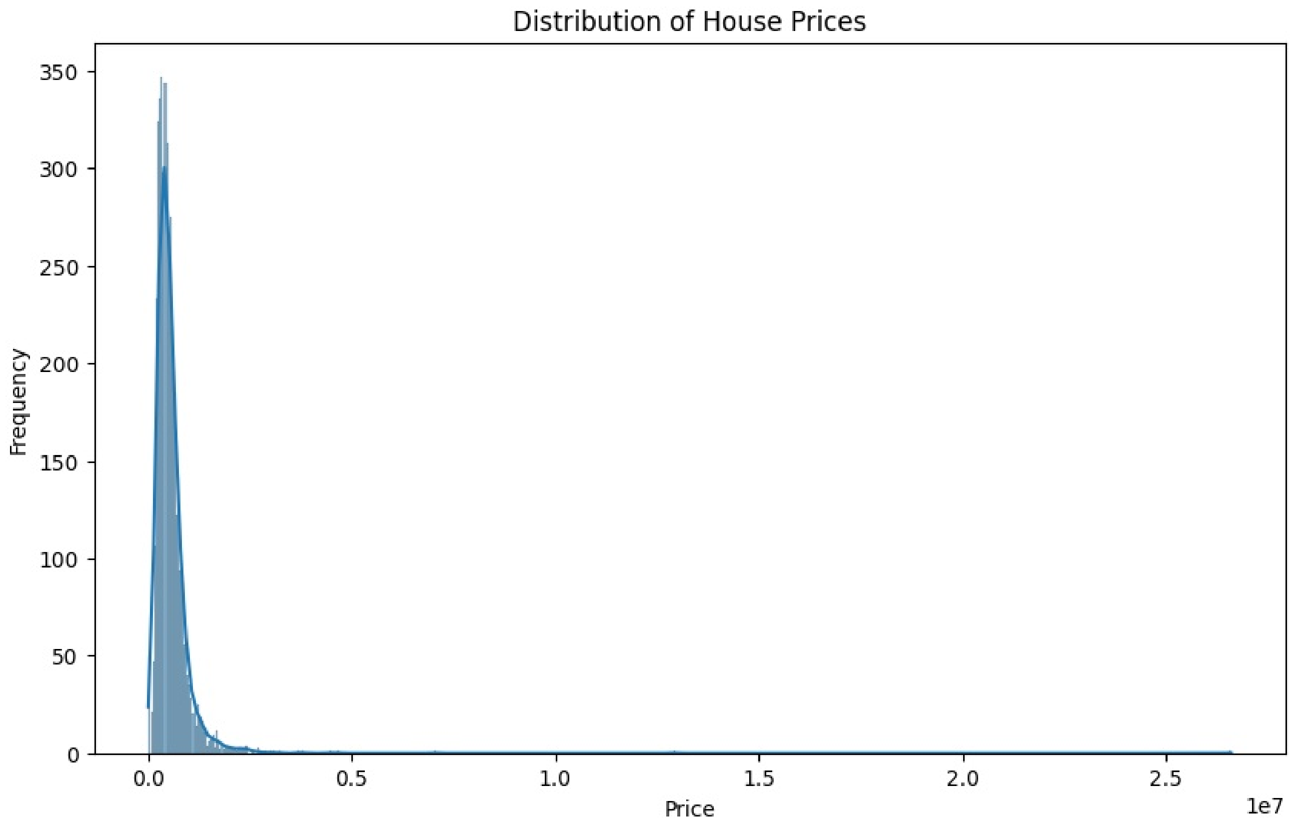
In [371…

```python
df.hist(bins=50, figsize=(20,15))
plt.show()
```




In [372…

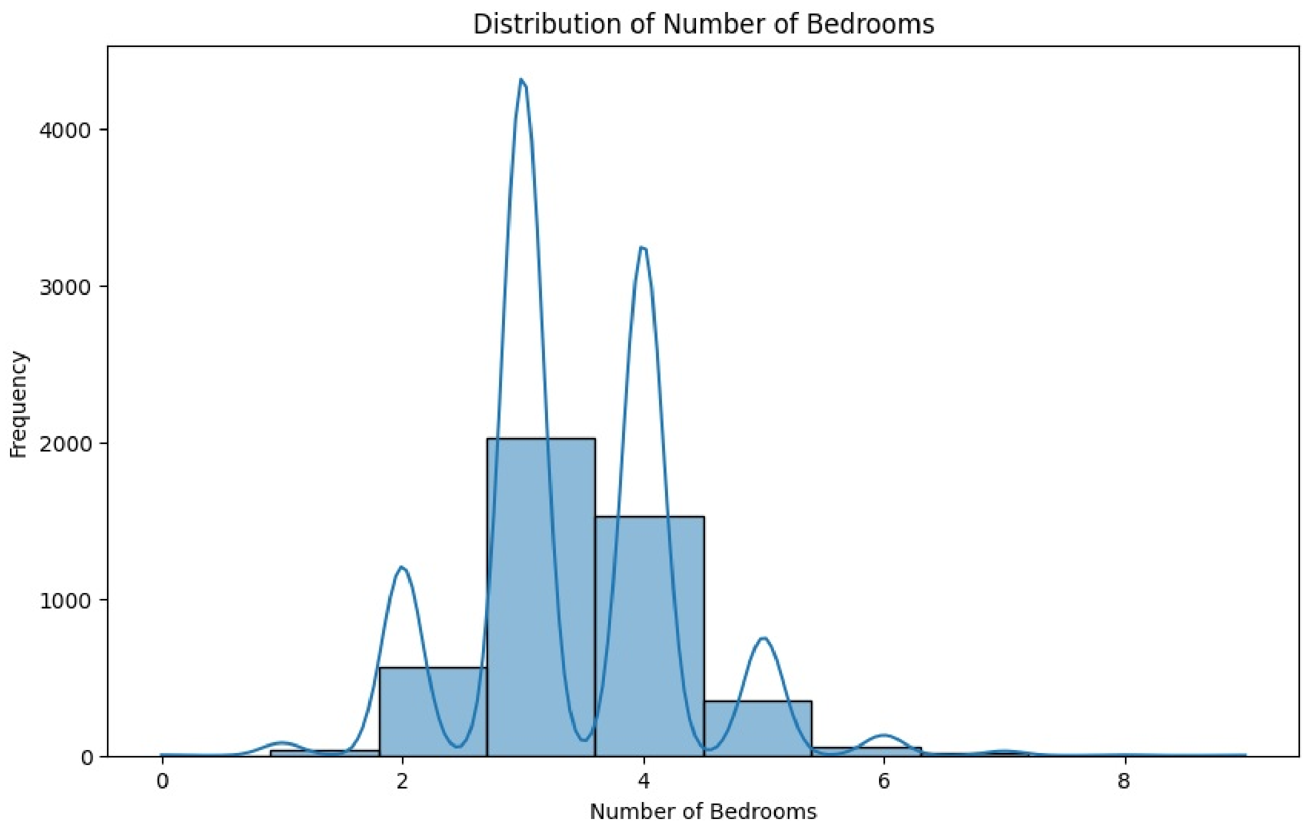
```
# Visualize the distribution of house prices
plt.figure(figsize=(10, 6))
sns.histplot(df['price'], kde=True)
plt.title('Distribution of House Prices')
plt.xlabel('Price')
plt.ylabel('Frequency')
plt.show()
```




In [373…

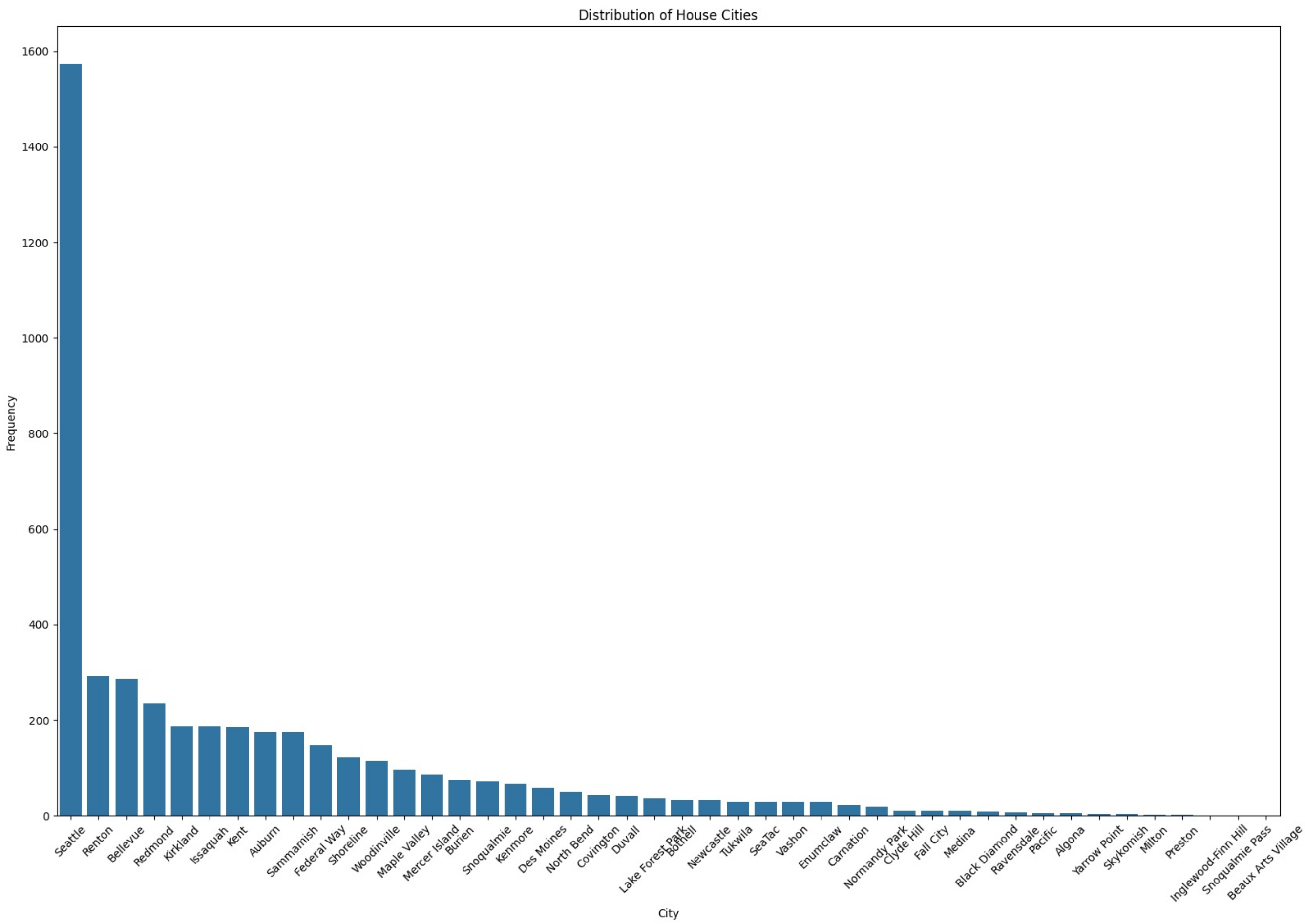
```
# Visualize the distribution of number of bedrooms
plt.figure(figsize=(10, 6))
sns.histplot(df['bedrooms'], kde=True, bins=df['bedrooms'].nunique())
```

```
plt.title('Distribution of Number of Bedrooms')
plt.xlabel('Number of Bedrooms')
plt.ylabel('Frequency')
plt.show()
```




In [374…

```
# Visualize the distribution of cities using barplot
city_counts = df['city'].value_counts()
plt.figure(figsize=(20,13))
sns.barplot(x=city_counts.index, y=city_counts.values)
plt.title('Distribution of House Cities')
plt.xlabel('City')
plt.ylabel('Frequency')
plt.xticks(rotation=45)
plt.show()
```



In [375…
```
df.keys()
```
Out[375]:
```
Index(['date', 'price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot',
       'floors', 'waterfront', 'view', 'condition', 'sqft_above',
       'sqft_basement', 'yr_built', 'yr_renovated', 'street', 'city',
       'statezip', 'country'],
      dtype='object')
```
In [376…
```
label = LabelEncoder()
df['city']=label.fit_transform(df['city'])
```
In [377…
```
# Feature selection
x = df[["bedrooms","bathrooms","city"]]
y = df['price']
```
In [378…
```
x
```
Out[378]:

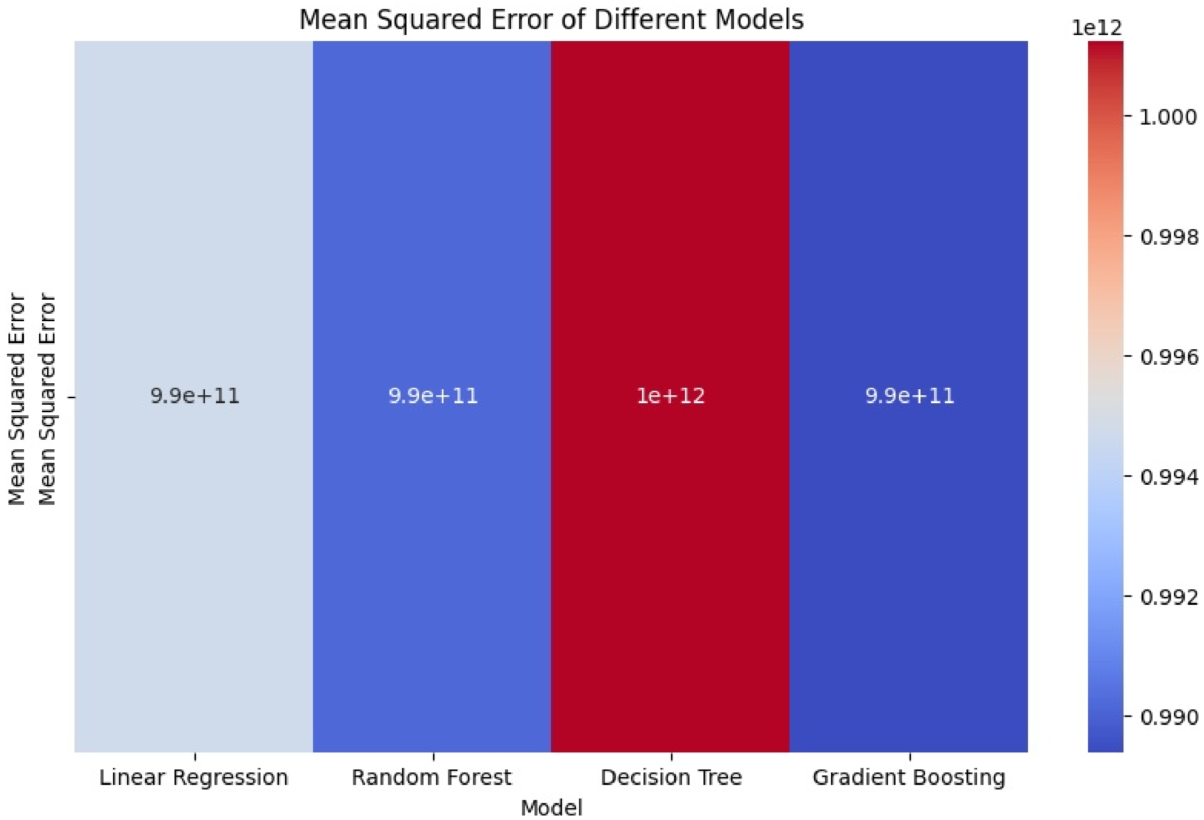
| | bedrooms | bathrooms | city |
|---|---|---|---|
| **0** | 3.0 | 1.50 | 36 |
| **1** | 5.0 | 2.50 | 35 |
| **2** | 3.0 | 2.00 | 18 |
| **3** | 3.0 | 2.25 | 3 |
| **4** | 4.0 | 2.50 | 31 |
| **...** | ... | ... | ... |
| **4595** | 3.0 | 1.75 | 35 |
| **4596** | 3.0 | 2.50 | 3 |
| **4597** | 3.0 | 2.50 | 32 |
| **4598** | 4.0 | 2.00 | 35 |
| **4599** | 3.0 | 2.50 | 9 |

4600 rows × 3 columns

In [379…
```
y
```
Out[379]:
```
0       3.130000e+05
1       2.384000e+06
2       3.420000e+05
3       4.200000e+05
4       5.500000e+05
            ...
4595    3.081667e+05
4596    5.343333e+05
4597    4.169042e+05
4598    2.034000e+05
4599    2.206000e+05
Name: price, Length: 4600, dtype: float64
```
In [380…
```
# Split the dataset into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
```
In [381…
```
#scaling
scale = StandardScaler()
x_train = scale.fit_transform(x_train)
x_test = scale.transform(x_test)
```
In [382…
```
# Initialize models
models = {
    'Linear Regression': LinearRegression(),
    'Random Forest': RandomForestRegressor(random_state=42),
    'Decision Tree': DecisionTreeRegressor(random_state=42),
    'Gradient Boosting': GradientBoostingRegressor(random_state=42)
}
```
In [383…
```
# Train and evaluate models
results = {}
for name, model in models.items():
    model.fit(x_train, y_train)
    y_pred = model.predict(x_test)
    mse = mean_squared_error(y_test, y_pred)
    results[name] = mse
    print(f'{name} Mean Squared Error: {mse}')
```
```
Linear Regression Mean Squared Error: 994739551125.2224
Random Forest Mean Squared Error: 990153673453.1986
Decision Tree Mean Squared Error: 1001240109347.2585
Gradient Boosting Mean Squared Error: 989400668255.3132
```
In [384…
```
# Create a DataFrame for the results
results_df = pd.DataFrame(list(results.items()), columns=['Model', 'Mean Squared Error'])
```

In [385…

```
# Visualize the results using a heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(results_df.set_index('Model').T, annot=True, cmap='coolwarm')
plt.title('Mean Squared Error of Different Models')
plt.xlabel('Model')
plt.ylabel('Mean Squared Error')
plt.show()
```



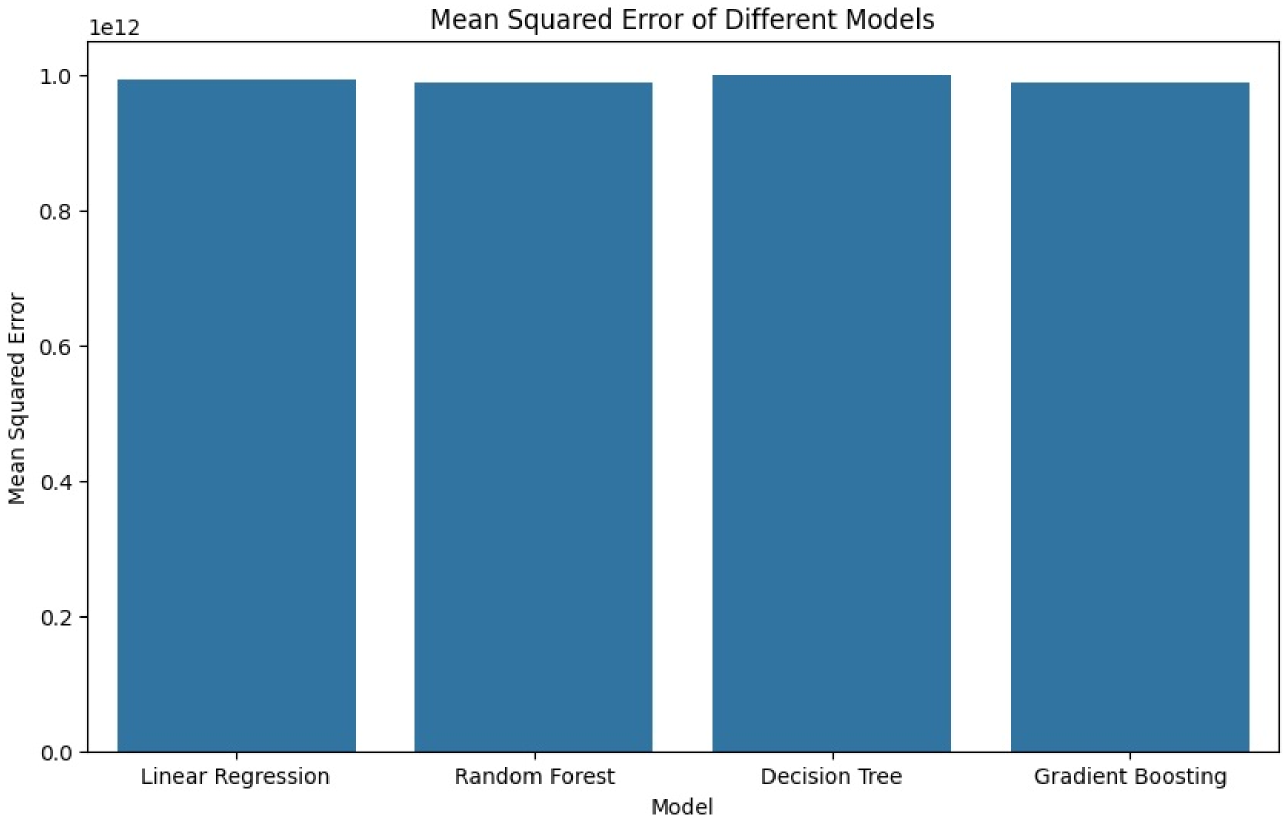

In [386…

```
# Visualize the results
model_names = list(results.keys())
mse_values = list(results.values())
print(model_names)
print(mse_values)
```

```
['Linear Regression', 'Random Forest', 'Decision Tree', 'Gradient Boosting']
[np.float64(994739551125.2224), np.float64(990153673453.1986), np.float64(1001240109347.2585), np.float64(98940
0668255.3132)]
```

In [387…

```
plt.figure(figsize=(10, 6))
sns.barplot(x=model_names, y=mse_values)
plt.title('Mean Squared Error of Different Models')
plt.xlabel('Model')
plt.ylabel('Mean Squared Error')
plt.show()
```




In [388…

```
#Take new datasets for prediction purpose
df=pd.read_csv('output.csv')
df.head()
```

Out[388]:

| | date | price | bedrooms | bathrooms | sqft_living | sqft_lot | floors | waterfront | view | condition | sqft_above | sqft_basement | yr_built |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 2014-05-02 00:00:00 | 313000.0 | 3.0 | 1.50 | 1340 | 7912 | 1.5 | 0 | 0 | 3 | 1340 | 0 | 1955 |
| **1** | 2014-05-02 00:00:00 | 2384000.0 | 5.0 | 2.50 | 3650 | 9050 | 2.0 | 0 | 4 | 5 | 3370 | 280 | 1921 |
| **2** | 2014-05-02 00:00:00 | 342000.0 | 3.0 | 2.00 | 1930 | 11947 | 1.0 | 0 | 0 | 4 | 1930 | 0 | 1966 |
| **3** | 2014-05-02 00:00:00 | 420000.0 | 3.0 | 2.25 | 2000 | 8030 | 1.0 | 0 | 0 | 4 | 1000 | 1000 | 1963 |
| **4** | 2014-05-02 00:00:00 | 550000.0 | 4.0 | 2.50 | 1940 | 10500 | 1.0 | 0 | 0 | 4 | 1140 | 800 | 1976 |

In [389…

```
label1 = LabelEncoder()
df['city']=label1.fit_transform(df['city'])
```

In [390…

```
df.keys()
```

Out[390]:

```
Index(['date', 'price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot',
       'floors', 'waterfront', 'view', 'condition', 'sqft_above',
       'sqft_basement', 'yr_built', 'yr_renovated', 'street', 'city',
       'statezip', 'country'],
      dtype='object')
```

In [391…

```
new_df=df[["bedrooms","bathrooms","city"]]
new_df
```

Out[391]:

| | bedrooms | bathrooms | city |
|---|---|---|---|
| **0** | 3.0 | 1.50 | 36 |
| **1** | 5.0 | 2.50 | 35 |
| **2** | 3.0 | 2.00 | 18 |
| **3** | 3.0 | 2.25 | 3 |
| **4** | 4.0 | 2.50 | 31 |
| **...** | ... | ... | ... |
| **4595** | 3.0 | 1.75 | 35 |
| **4596** | 3.0 | 2.50 | 3 |
| **4597** | 3.0 | 2.50 | 32 |
| **4598** | 4.0 | 2.00 | 35 |
| **4599** | 3.0 | 2.50 | 9 |

4600 rows × 3 columns

In [392…

```
# Scale the new sample data
test_data = scale.transform(new_df)
```

In [393…

```
# Predictions on new sample data
new_predictions = {}
for name, model in models.items():
    new_predictions[name] = model.predict(test_data)
    print(f"{name} Predictions on New Data:", new_predictions[name])
```

```
Linear Regression Predictions on New Data: [410373.01969841 691687.87750807 477770.37191101 ... 628418.92546631
 549729.64858226 568582.12450125]
Random Forest Predictions on New Data: [331453.3424912  726480.39768855 296844.940275   ... 399556.33280588
 551040.55813699 278664.60714286]
Decision Tree Predictions on New Data: [327000.         714807.69230769 293664.28571429 ... 400055.96666668
 547474.57763974 275750.        ]
Gradient Boosting Predictions on New Data: [359418.35056968 680980.46587945 302807.88126357 ... 482004.18226315
 575506.47496632 408843.43443016]
```

In [ ]:

In [ ]:

In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js