Real Estate Price Prediction Model: Using Supervised Regression

Load The Dataset

```
import pandas as pd

# Load the dataset
data = pd.read_csv('https://raw.githubusercontent.com/Jaden2802/Real_Estate_Prediction/main/House_Prices.csv')

# Display the first few rows
data.head()
```

	ID	Price	Bedrooms	Bathrooms	Sqft_living	Sqft_lot	Floors	Waterfront	View
0	1	280000.0	6	3.00	2400	9373	2.0	0	0
1	2	300000.0	6	3.00	2400	9373	2.0	0	0
2	3	647500.0	4	1.75	2060	26036	1.0	0	0
3	4	400000.0	3	1.00	1460	43000	1.0	0	0
4	5	235000.0	3	1.00	1430	7599	1.5	0	0

Next steps: Generate code with `data` View recommended plots

Data Exploration

```
# Get basic information about the dataset
data.info()

# Summary statistics
data.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 20 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   ID             21613 non-null  int64
 1   Price          21613 non-null  float64
 2   Bedrooms       21613 non-null  int64
 3   Bathrooms      21613 non-null  float64
 4   Sqft_living    21613 non-null  int64
 5   Sqft_lot       21613 non-null  int64
 6   Floors         21613 non-null  float64
 7   Waterfront     21613 non-null  int64
 8   View           21613 non-null  int64
 9   Condition      21613 non-null  int64
 10  Grade          21613 non-null  int64
 11  Sqft_above     21613 non-null  int64
 12  Sqft_basement  21613 non-null  int64
 13  Yr_built       21613 non-null  int64
 14  Yr_renovated   21613 non-null  int64
 15  zipcode        21613 non-null  int64
 16  Lat            21613 non-null  float64
 17  Long           21613 non-null  float64
 18  Sqft_living15  21613 non-null  int64
 19  Sqft_lot15     21613 non-null  int64
dtypes: float64(5), int64(15)
memory usage: 3.3 MB
```

	ID	Price	Bedrooms	Bathrooms	Sqft_living	Sqft_lo
count	21613.00000	2.161300e+04	21613.000000	21613.000000	21613.000000	2.161300e+0
mean	10807.00000	5.401822e+05	3.370842	2.114757	2079.899736	1.510697e+0
std	6239.28002	3.673622e+05	0.930062	0.770163	918.440897	4.142051e+0
min	1.00000	7.500000e+04	0.000000	0.000000	290.000000	5.200000e+0
25%	5404.00000	3.219500e+05	3.000000	1.750000	1427.000000	5.040000e+0
50%	10807.00000	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+0
75%	16210.00000	6.450000e+05	4.000000	2.500000	2550.000000	1.068800e+0
max	21613.00000	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+0

Data Cleaning

```
# Check for missing values
print(data.isnull().sum())

# Handle missing values (example: fill with median)
data = data.fillna(data.median())

# Remove duplicates if any
data = data.drop_duplicates()

#Remove unnecessary columns
#data = data.drop(columns=['ID', 'Sqft_living', 'Sqft_living15', 'Sqft_lot15', 'Grade', 'View', 'Sqft_lot', 'Sqft_above', 'Sqft_basement
```

```
ID               0
Price            0
Bedrooms         0
Bathrooms        0
Sqft_living      0
Sqft_lot         0
Floors           0
Waterfront       0
View             0
Condition        0
Grade            0
Sqft_above       0
Sqft_basement    0
Yr_built         0
Yr_renovated     0
zipcode          0
Lat              0
Long             0
Sqft_living15    0
Sqft_lot15       0
dtype: int64
```

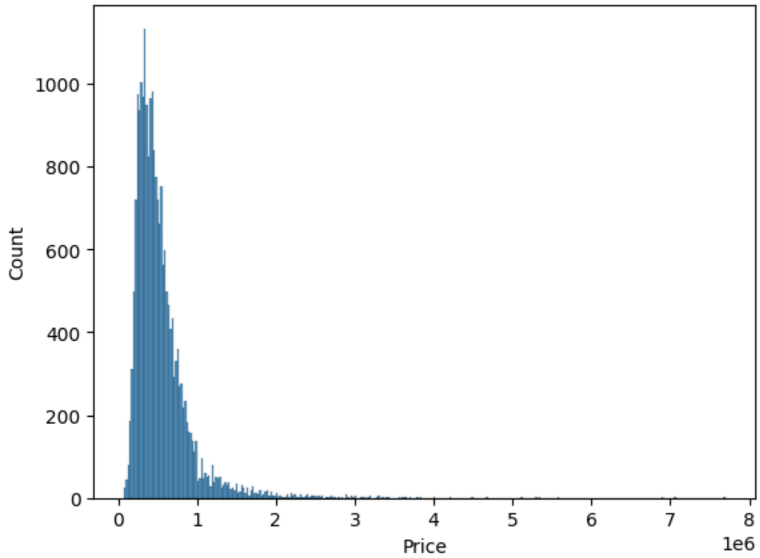
Exploratory Data Analysis (EDA)

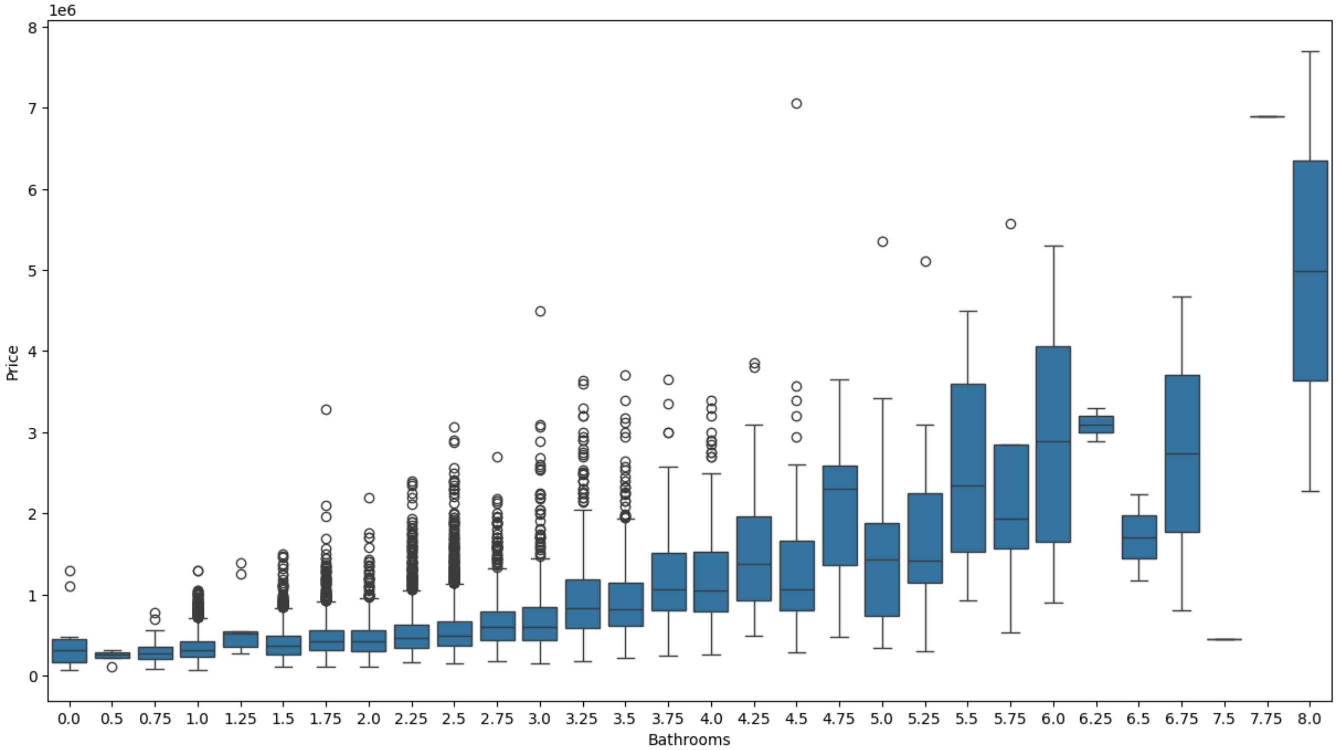
```
import seaborn as sns
import matplotlib.pyplot as plt
```

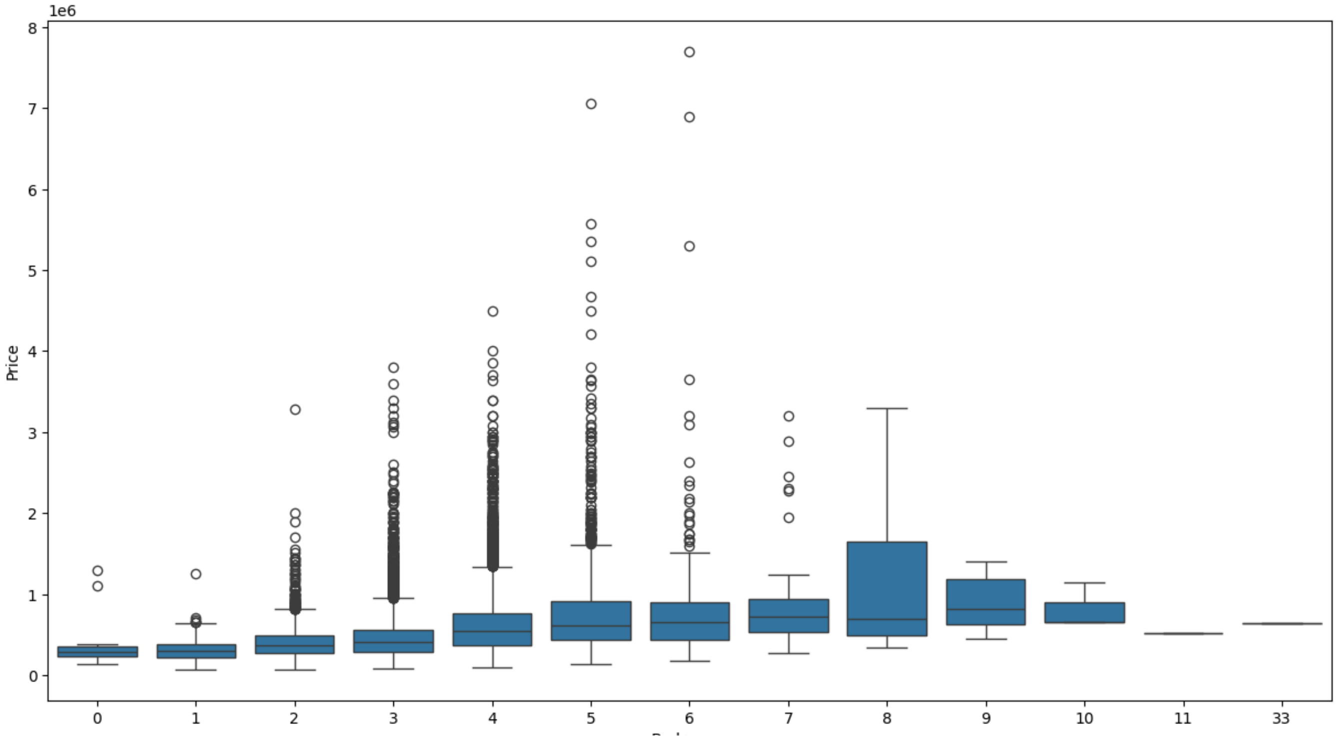
```
#Distribution of target variable (price)
sns.histplot(data['Price'])
plt.show()

#Boxplot for each feature
plt.figure(figsize=(15,8))
sns.boxplot(x="Bathrooms", y="Price", data=data)

plt.figure(figsize=(15,8))
sns.boxplot(x="Bedrooms", y="Price", data=data)

plt.figure(figsize=(15,8))
sns.boxplot(x="Floors", y="Price", data=data)

#ScatterPlot
plt.figure(figsize=(15,8))
sns.scatterplot(x="Floors", y="Price", data=data)
#CountPlot
plt.figure(figsize=(15,8))
sns.countplot(x="Bathrooms", data=data)

plt.figure(figsize=(15,8))
sns.countplot(x="Bedrooms", data=data)

plt.figure(figsize=(15,8))
sns.countplot(x="Floors", data=data)
```

```
features = ['Price', 'Bedrooms', 'Bathrooms', 'Floors']
data[features].hist(bins=20, figsize=(14, 10))
plt.show()
```

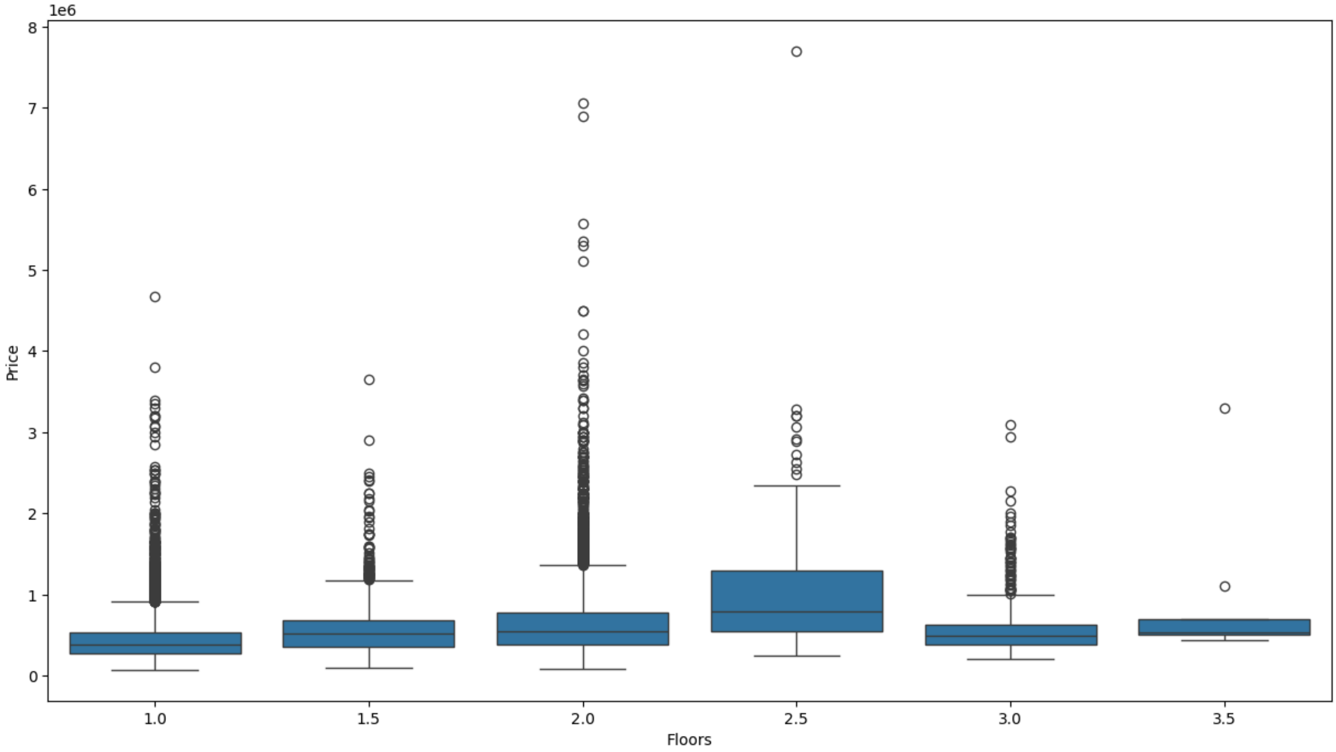
```
#Correlation heatmap
corr_matrix = data.corr()
plt.figure(figsize=(15, 10))

sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.show()
```

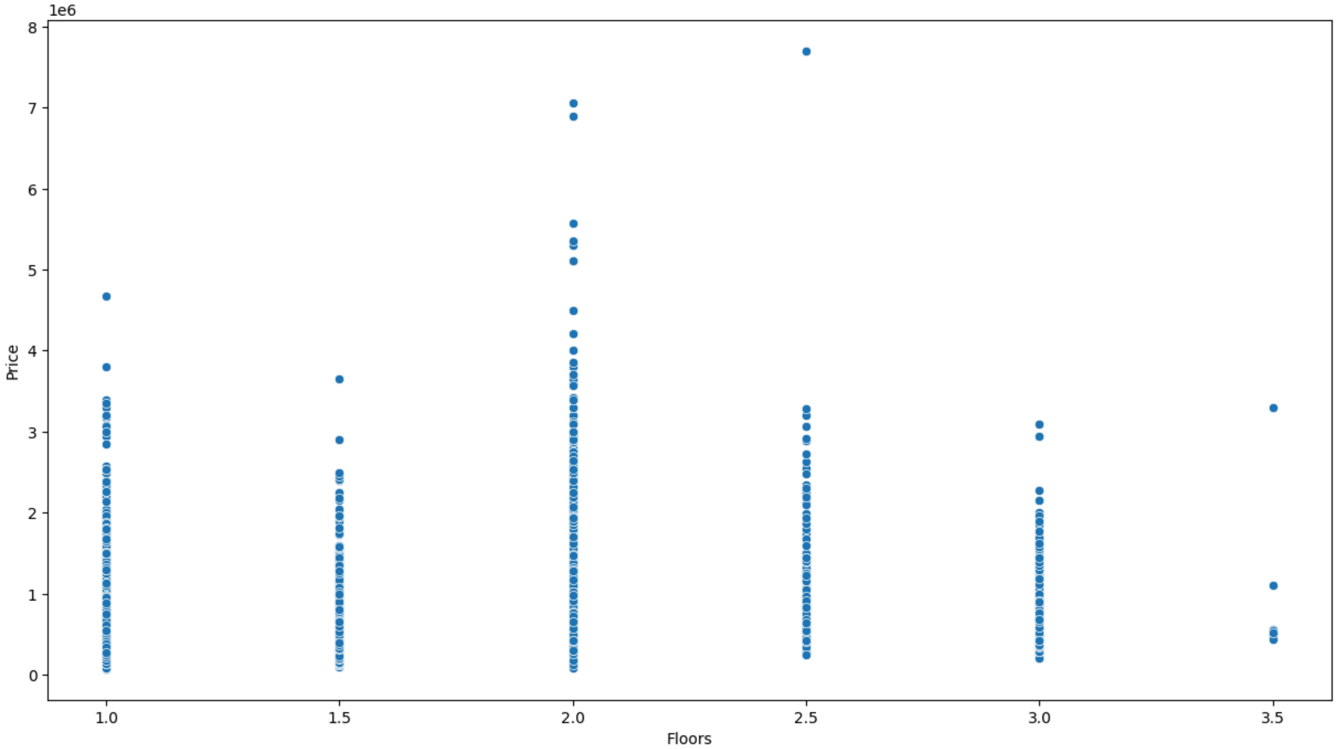

Count
Price
1e6

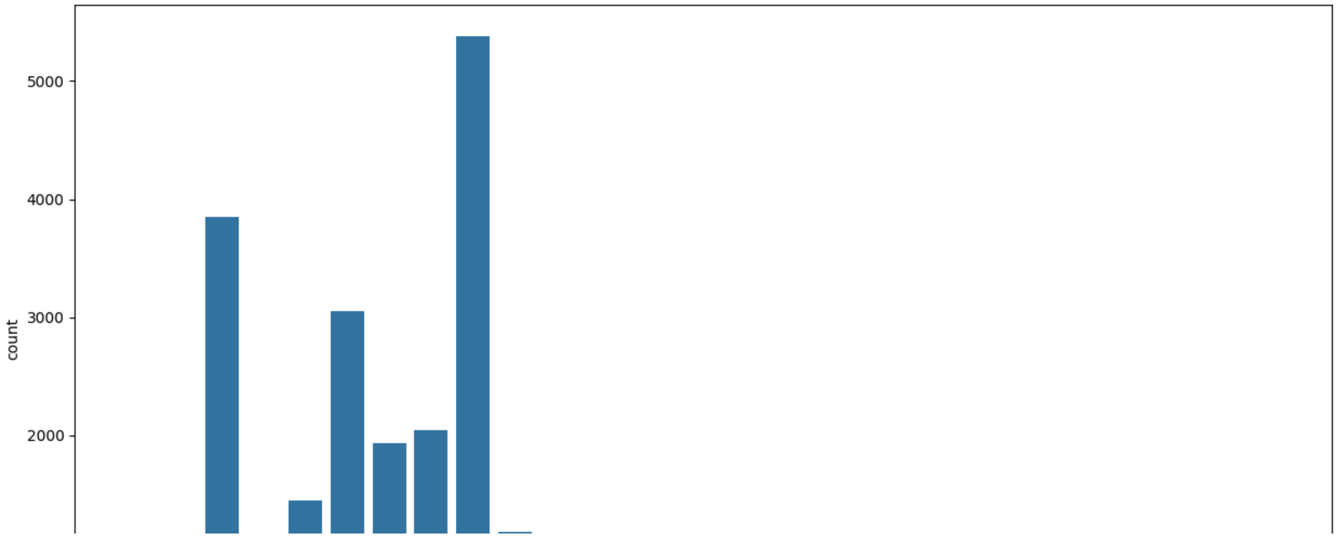

1e6
Price
Bathrooms

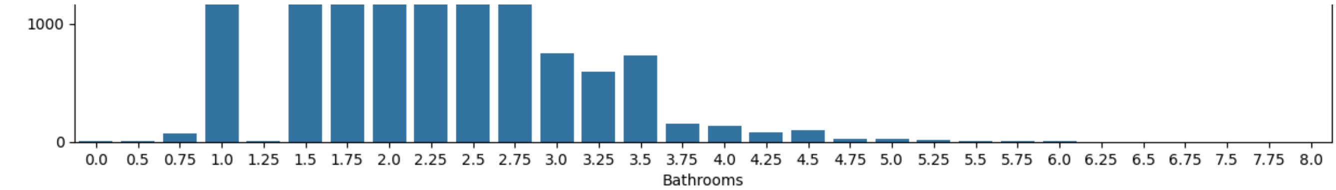

1e6
Price

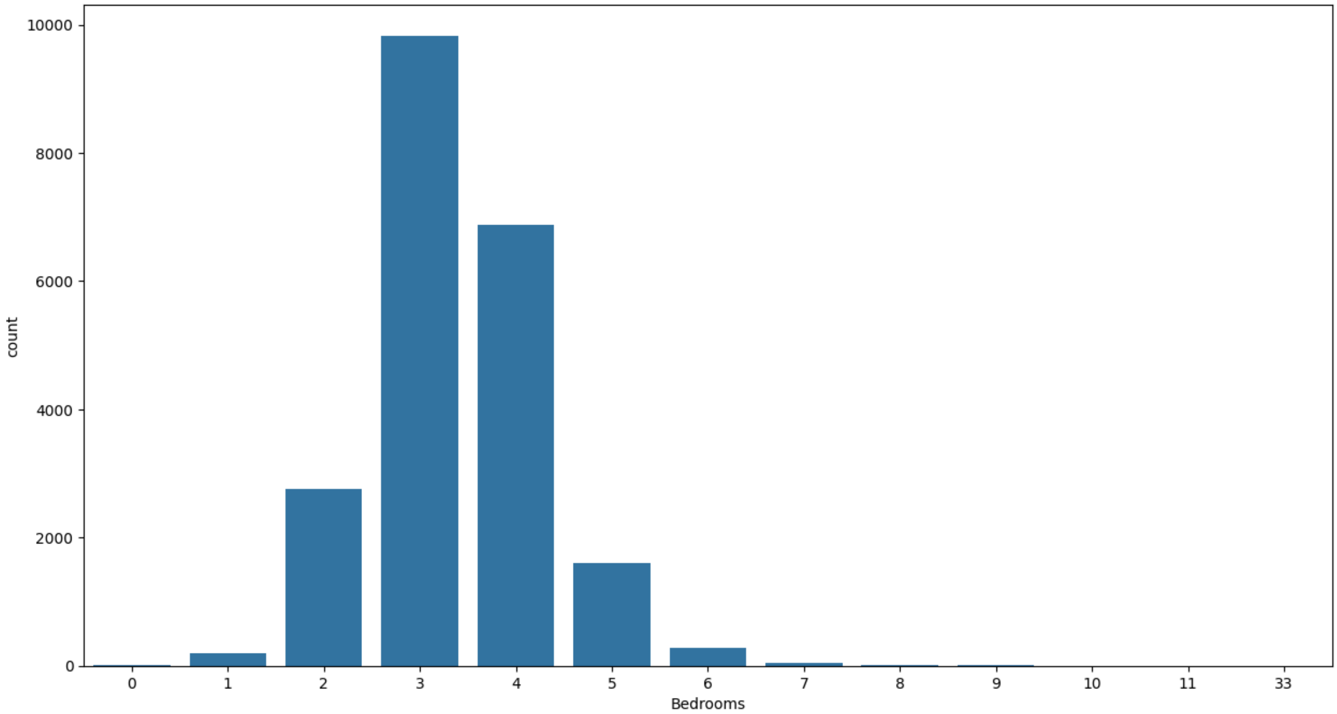
Bedrooms

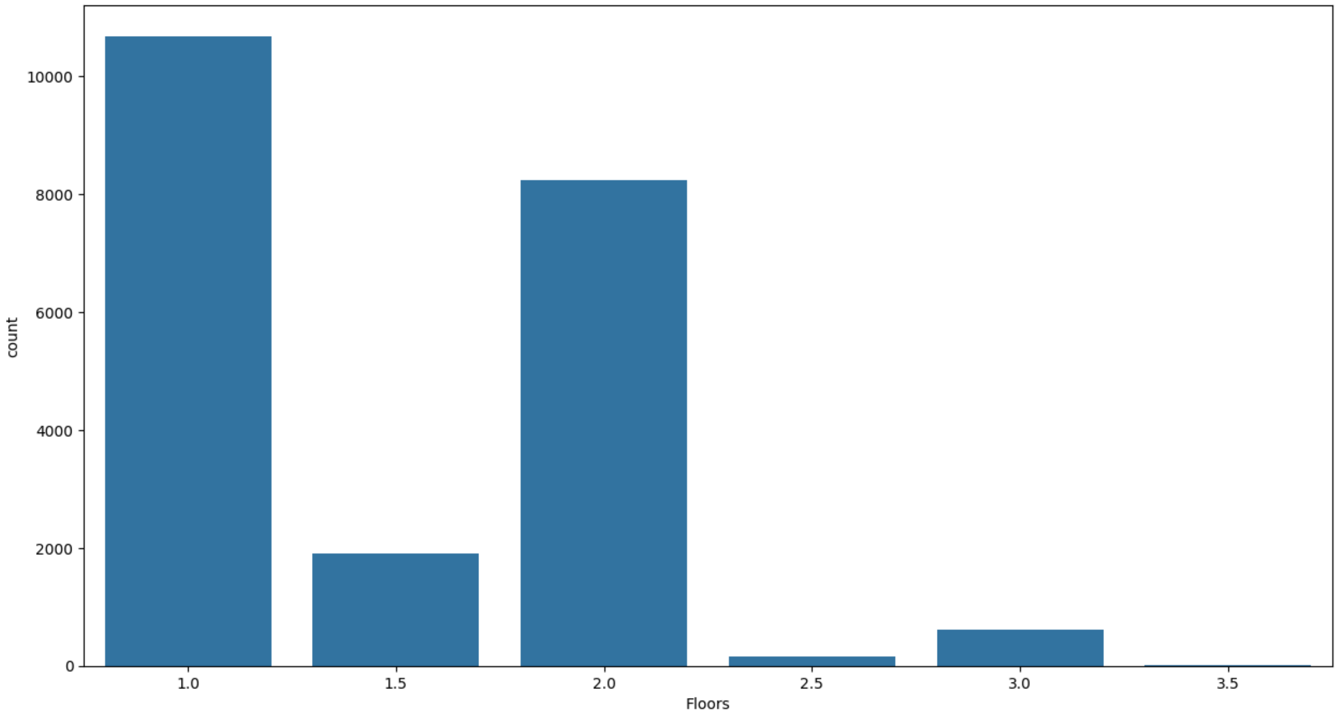

1e6
8
7
6
5
4
3
2
1
0
Price
1.0
1.5
2.0
2.5
3.0
3.5
Floors

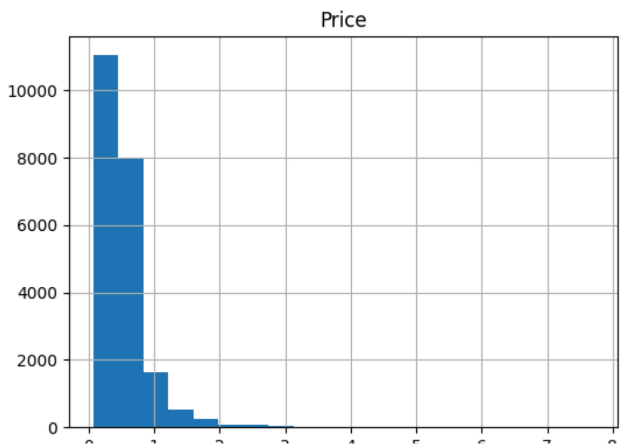

1e6
8
7
6
5
4
3
2
1
0
Price
1.0
1.5
2.0
2.5
3.0
3.5
Floors

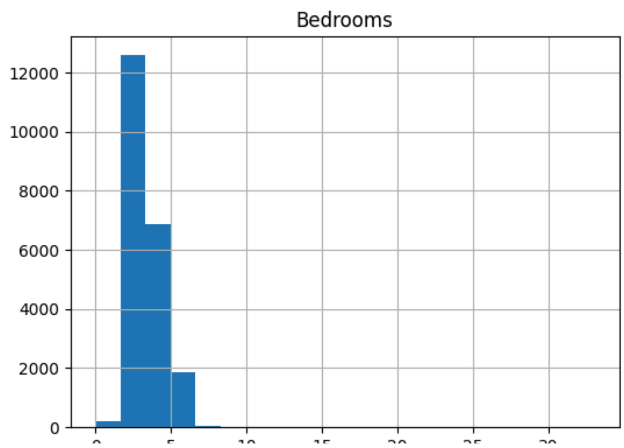

5000
4000
3000
2000
count

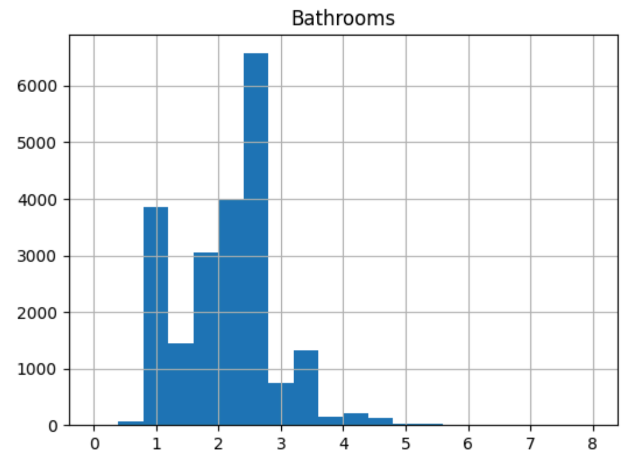

1000
0
0.0 0.5 0.75 1.0 1.25 1.5 1.75 2.0 2.25 2.5 2.75 3.0 3.25 3.5 3.75 4.0 4.25 4.5 4.75 5.0 5.25 5.5 5.75 6.0 6.25 6.5 6.75 7.5 7.75 8.0
Bathrooms

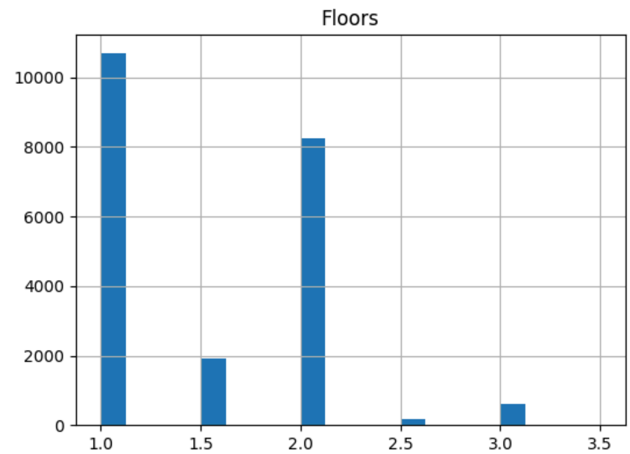

count
10000
8000
6000
4000
2000
0
0 1 2 3 4 5 6 7 8 9 10 11 33
Bedrooms

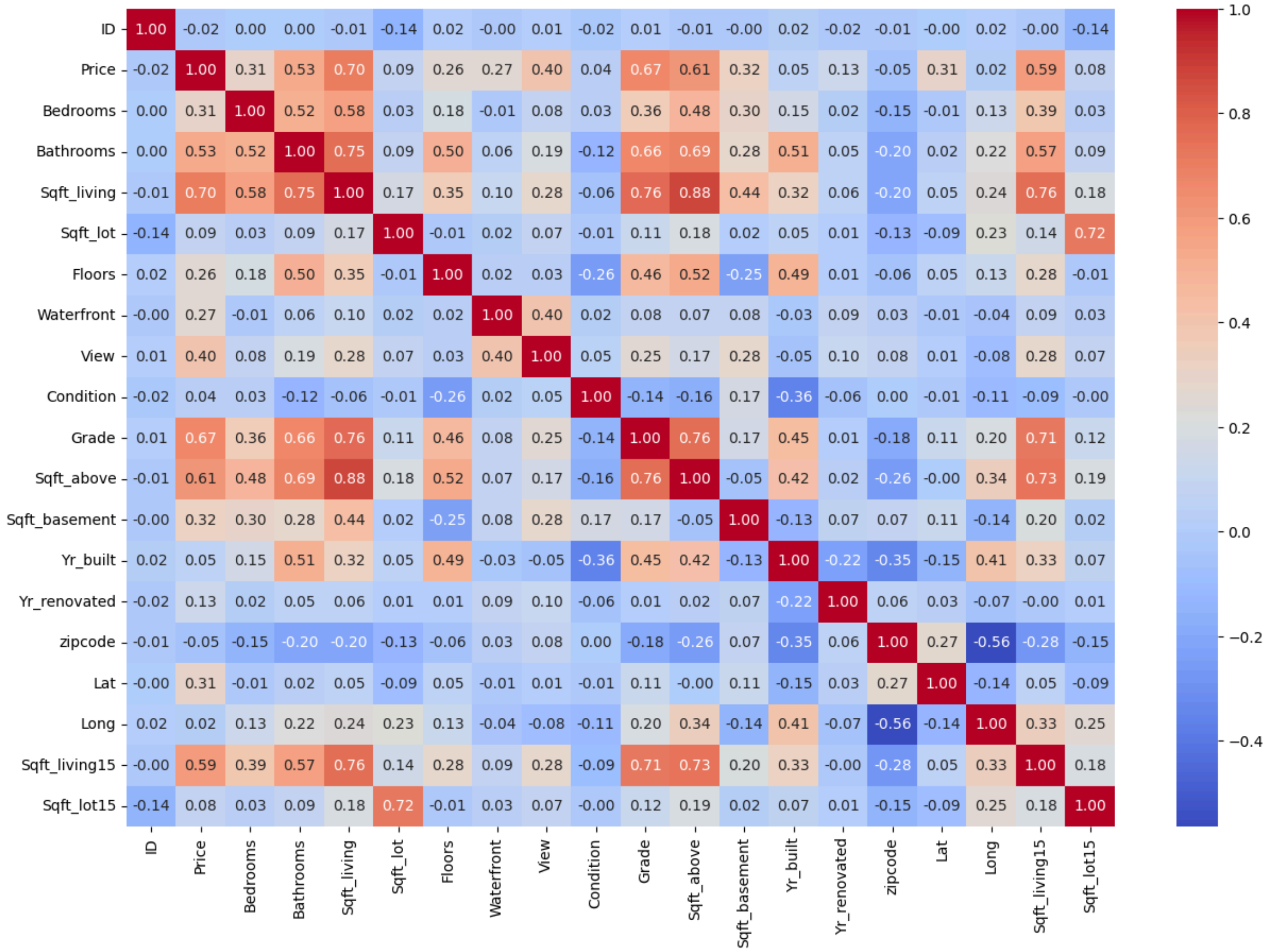

count
10000
8000
6000
4000
2000
0
1.0 1.5 2.0 2.5 3.0 3.5
Floors


Price
10000
8000
6000
4000
2000
0


Bedrooms
12000
10000
8000
6000
4000
2000
0


Bathrooms
0 1000 2000 3000 4000 5000 6000
0 1 2 3 4 5 6 7 8


Floors
0 2000 4000 6000 8000 10000
1.0 1.5 2.0 2.5 3.0 3.5


ID 1.00 -0.02 0.00 0.00 -0.01 -0.14 0.02 -0.00 0.01 -0.02 0.01 -0.01 -0.00 0.02 -0.02 -0.01 -0.00 0.02 -0.00 -0.14
Price -0.02 1.00 0.31 0.53 0.70 0.09 0.26 0.27 0.40 0.04 0.67 0.61 0.32 0.05 0.13 -0.05 0.31 0.02 0.59 0.08
Bedrooms 0.00 0.31 1.00 0.52 0.58 0.03 0.18 -0.01 0.08 0.03 0.36 0.48 0.30 0.15 0.02 -0.15 -0.01 0.13 0.39 0.03
Bathrooms 0.00 0.53 0.52 1.00 0.75 0.09 0.50 0.06 0.19 -0.12 0.66 0.69 0.28 0.51 0.05 -0.20 0.02 0.22 0.57 0.09
Sqft_living -0.01 0.70 0.58 0.75 1.00 0.17 0.35 0.10 0.28 -0.06 0.76 0.88 0.44 0.32 0.06 -0.20 0.05 0.24 0.76 0.18
Sqft_lot -0.14 0.09 0.03 0.09 0.17 1.00 -0.01 0.02 0.07 -0.01 0.11 0.18 0.02 0.05 0.01 -0.13 -0.09 0.23 0.14 0.72
Floors 0.02 0.26 0.18 0.50 0.35 -0.01 1.00 0.02 0.03 -0.26 0.46 0.52 -0.25 0.49 0.01 -0.06 0.05 0.13 0.28 -0.01
Waterfront -0.00 0.27 -0.01 0.06 0.10 0.02 0.02 1.00 0.40 0.02 0.08 0.07 0.08 -0.03 0.09 0.03 -0.01 -0.04 0.09 0.03
View 0.01 0.40 0.08 0.19 0.28 0.07 0.03 0.40 1.00 0.05 0.25 0.17 0.28 -0.05 0.10 0.08 0.01 -0.08 0.28 0.07
Condition -0.02 0.04 0.03 -0.12 -0.06 -0.01 -0.26 0.02 0.05 1.00 -0.14 -0.16 0.17 -0.36 -0.06 0.00 -0.01 -0.11 -0.09 -0.00
Grade 0.01 0.67 0.36 0.66 0.76 0.11 0.46 0.08 0.25 -0.14 1.00 0.76 0.17 0.45 0.01 -0.18 0.11 0.20 0.71 0.12
Sqft_above -0.01 0.61 0.48 0.69 0.88 0.18 0.52 0.07 0.17 -0.16 0.76 1.00 -0.05 0.42 0.02 -0.26 -0.00 0.34 0.73 0.19
Sqft_basement -0.00 0.32 0.30 0.28 0.44 0.02 -0.25 0.08 0.28 0.17 0.17 -0.05 1.00 -0.13 0.07 0.07 0.11 -0.14 0.20 0.02
Yr_built 0.02 0.05 0.15 0.51 0.32 0.05 0.49 -0.03 -0.05 -0.36 0.45 0.42 -0.13 1.00 -0.22 -0.35 -0.15 0.41 0.33 0.07
Yr_renovated -0.02 0.13 0.02 0.05 0.06 0.01 0.01 0.09 0.10 -0.06 0.01 0.02 0.07 -0.22 1.00 0.06 0.03 -0.07 -0.00 0.01
zipcode -0.01 -0.05 -0.15 -0.20 -0.20 -0.13 -0.06 0.03 0.08 0.00 -0.18 -0.26 0.07 -0.35 0.06 1.00 0.27 -0.56 -0.28 -0.15
Lat -0.00 0.31 -0.01 0.02 0.05 -0.09 0.05 -0.01 0.01 -0.01 0.11 -0.00 0.11 -0.15 0.03 0.27 1.00 -0.14 0.05 -0.09
Long 0.02 0.02 0.13 0.22 0.24 0.23 0.13 -0.04 -0.08 -0.11 0.20 0.34 -0.14 0.41 -0.07 -0.56 -0.14 1.00 0.33 0.25
Sqft_living15 -0.00 0.59 0.39 0.57 0.76 0.14 0.28 0.09 0.28 -0.09 0.71 0.73 0.20 0.33 -0.00 -0.28 0.05 0.33 1.00 0.18
Sqft_lot15 -0.14 0.08 0.03 0.09 0.18 0.72 -0.01 0.03 0.07 -0.00 0.12 0.19 0.02 0.07 0.01 -0.15 -0.09 0.25 0.18 1.00
ID Price Bedrooms Bathrooms Sqft_living Sqft_lot Floors Waterfront View Condition Grade Sqft_above Sqft_basement Yr_built Yr_renovated zipcode Lat Long Sqft_living15 Sqft_lot15
1.0 0.8 0.6 0.4 0.2 0.0 −0.2 −0.4

Double-click (or enter) to edit

Split the Data

```
from sklearn.model_selection import train_test_split

# Define features and target variable
X = data.drop('Price', axis=1)
y = data['Price']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
```

Model Training

```
import numpy as np

from sklearn.model_selection import cross_val_score
def classify(model, x, y):
    x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
    model.fit(x_train, y_train)
    print("Accuracy is", model.score(x_test, y_test)*100)
    # cross validation - it is used for better validation of model
    # eg: cv-5, train-4, test-1
    score = cross_val_score(model, x, y, cv=5)
    print("Cross validation is",np.mean(score)*100)
```

Model Evaluation

Linear Regression

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
classify(model, X, y)
```

```
Accuracy is 70.3725085982858
Cross validation is 69.57368511239943
```

Decision Tree

```
from sklearn.tree import DecisionTreeRegressor
model = DecisionTreeRegressor()
classify(model, X, y)
```

```
Accuracy is 73.25108202260915
Cross validation is 71.17465615604863
```

Random Forest

```
from sklearn.ensemble import RandomForestRegressor,ExtraTreesRegressor
model = RandomForestRegressor()
classify(model, X, y)
```

```
Accuracy is 87.91164749695032
Cross validation is 86.45652778192785
```

Gradient Boosting

```
from sklearn.ensemble import GradientBoostingRegressor
model = GradientBoostingRegressor()
classify(model, X, y)
```

```
Accuracy is 87.03179210149418
Cross validation is 86.14383759131867
```