property_price_data_model

December 11, 2025

[ ]:
```
# Problem Statement
# A major challenge for property sellers is determining the right sale price␣
↪for a property.
# Accurate price prediction benefits investors in assessing returns and helps␣
↪buyers plan
# their finances in line with market trends.
# Property prices are influenced by several factors such as:
#   Property area
#   Basement square footage
#   Year of construction
#   Number of bedrooms
#   And many other features
# By applying Regression Analysis, we can model these features to predict the␣
↪price of
# a property with better accuracy.
```

[27]:
```
# import libraries:
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import joblib
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

[28]:
```
# load the data
data = pd.read_csv("property_price_data (1).csv")
data.head(10)
```

[28]:
```
           \  MSSubClass MSZoning  LotFrontage  LotArea Street Alley LotShape  \
0    PR0504          20       RL        100.0    15537   Pave   NaN      IR1
1    PR0102          60       RL         77.0     9534   Pave   NaN      Reg
2    PR0609          70       RL          NaN    12781   Pave   NaN      Reg
3   PR01090         120       FV         37.0     3728   Pave  Pave      IR1
4    PR0820         120       RL         44.0     6606   Pave   NaN      IR1
```

```
5    PR0685           60        RL          58.0     18002    Pave    NaN       IR2
6   PR01281           20        RL          67.0      9769    Pave    NaN       IR1
7    PR0921           60        RL          70.0      8653    Pave    NaN       IR1
8   PR01454           20        RL          90.0     17720    Pave    NaN       Reg
9    PR0541           20        RL          85.0     15369    Pave    NaN       Reg

  LandContour Utilities  … 3SsnPorch ScreenPorch PoolArea PoolQC Fence  \
0         Lvl    AllPub  …         0         161        0    NaN  GdWo
1         Lvl    AllPub  …         0           0        0    NaN   NaN
2         HLS    AllPub  …         0           0        0    NaN   NaN
3         Lvl    AllPub  …         0           0        0    NaN   NaN
4         Lvl    AllPub  …         0           0        0    NaN   NaN
5         Lvl    AllPub  …         0           0        0    NaN   NaN
6         Lvl    AllPub  …         0           0        0    NaN   NaN
7         Lvl    AllPub  …         0           0        0    NaN   NaN
8         Lvl    AllPub  …         0           0        0    NaN   NaN
9         Lvl    AllPub  …         0           0        0    NaN   NaN

  MiscFeature MiscVal  YrSold  SaleCondition  SalePrice
0         NaN       0    2010         Normal     288330
1         NaN       0    2010         Normal     183164
2         NaN       0    2007         Alloca     362145
3         NaN       0    2006         Normal     196079
4         NaN       0    2010        Partial     228515
5         NaN       0    2010         Normal     224119
6         NaN       0    2009         Normal     228107
7         NaN       0    2007         Normal     203953
8         NaN       0    2006        Abnorml      87468
9         NaN       0    2009         Normal     313697

[10 rows x 69 columns]
```

[3]:
```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 970 entries, 0 to 969
Data columns (total 69 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   \               970 non-null    object
 1   MSSubClass      970 non-null    int64
 2   MSZoning        970 non-null    object
 3   LotFrontage     789 non-null    float64
 4   LotArea         970 non-null    int64
 5   Street          970 non-null    object
 6   Alley           56 non-null     object
 7   LotShape        970 non-null    object
 8   LandContour     970 non-null    object
```

9	Utilities	970 non-null	object
10	LotConfig	970 non-null	object
11	LandSlope	970 non-null	object
12	Neighborhood	970 non-null	object
13	Condition1	970 non-null	object
14	Condition2	970 non-null	object
15	BldgType	970 non-null	object
16	PropStyle	970 non-null	object
17	OverallQual	970 non-null	int64
18	OverallCond	970 non-null	int64
19	YearBuilt	970 non-null	int64
20	YearRemodAdd	970 non-null	int64
21	RoofStyle	970 non-null	object
22	RoofMatl	970 non-null	object
23	Exterior1st	970 non-null	object
24	Exterior2nd	970 non-null	object
25	ExterQual	970 non-null	object
26	ExterCond	970 non-null	object
27	Foundation	970 non-null	object
28	BsmtQual	947 non-null	object
29	BsmtCond	947 non-null	object
30	BsmtExposure	946 non-null	object
31	TotalBsmtSF	970 non-null	int64
32	Heating	970 non-null	object
33	HeatingQC	970 non-null	object
34	CentralAir	970 non-null	object
35	Electrical	970 non-null	object
36	GrLivArea	970 non-null	int64
37	BsmtFullBath	970 non-null	int64
38	BsmtHalfBath	970 non-null	int64
39	FullBath	970 non-null	int64
40	HalfBath	970 non-null	int64
41	BedroomAbvGr	970 non-null	int64
42	KitchenAbvGr	970 non-null	int64
43	KitchenQual	970 non-null	object
44	TotRmsAbvGrd	970 non-null	int64
45	Functional	970 non-null	object
46	Fireplaces	970 non-null	int64
47	FireplaceQu	533 non-null	object
48	GarageType	918 non-null	object
49	GarageYrBlt	918 non-null	float64
50	GarageFinish	918 non-null	object
51	GarageCars	970 non-null	int64
52	GarageArea	970 non-null	int64
53	GarageQual	918 non-null	object
54	GarageCond	918 non-null	object
55	PavedDrive	970 non-null	object
56	WoodDeckSF	970 non-null	int64

```
 57  OpenPorchSF    970 non-null    int64
 58  EnclosedPorch  970 non-null    int64
 59  3SsnPorch      970 non-null    int64
 60  ScreenPorch    970 non-null    int64
 61  PoolArea       970 non-null    int64
 62  PoolQC         7 non-null      object
 63  Fence          169 non-null    object
 64  MiscFeature    29 non-null     object
 65  MiscVal        970 non-null    int64
 66  YrSold         970 non-null    int64
 67  SaleCondition  970 non-null    object
 68  SalePrice      970 non-null    int64
dtypes: float64(2), int64(27), object(40)
memory usage: 523.0+ KB
```

[4]:
```
data.shape
```

[4]:
```
(970, 69)
```

[5]:
```
data.isnull().sum()
```

[5]:
```
\                  0
MSSubClass         0
MSZoning           0
LotFrontage      181
LotArea            0
                ...
MiscFeature      941
MiscVal            0
YrSold             0
SaleCondition      0
SalePrice          0
Length: 69, dtype: int64
```

[6]:
```
data[data.isnull().any(axis=1)]        # show the rows that is having null values
```

[6]:
```
            \  MSSubClass MSZoning  LotFrontage  LotArea Street Alley LotShape  \
0      PR0504          20       RL        100.0    15537   Pave   NaN      IR1
1      PR0102          60       RL         77.0     9534   Pave   NaN      Reg
2      PR0609          70       RL          NaN    12781   Pave   NaN      Reg
3     PR01090         120       FV         37.0     3728   Pave  Pave      IR1
4      PR0820         120       RL         44.0     6606   Pave   NaN      IR1
..        …           …        …          …        …      …    …       …
965    PR0162          60       RL        110.0    14777   Pave   NaN      IR1
966    PR0693          60       RL         42.0    26949   Pave   NaN      IR1
967    PR0674          20       RL        110.0    15573   Pave   NaN      Reg
968    PR0750          50       RL         50.0     8367   Pave   NaN      Reg
969   PR01032          75       RL        102.0    16765   Pave   NaN      Reg
```

```
    LandContour Utilities  … 3SsnPorch ScreenPorch PoolArea PoolQC Fence  \
0           Lvl    AllPub  …         0         161        0    NaN  GdWo
1           Lvl    AllPub  …         0           0        0    NaN   NaN
2           HLS    AllPub  …         0           0        0    NaN   NaN
3           Lvl    AllPub  …         0           0        0    NaN   NaN
4           Lvl    AllPub  …         0           0        0    NaN   NaN
..          …         …  …         …           …        …    …
965         Lvl    AllPub  …         0           0        0    NaN   NaN
966         Lvl    AllPub  …         0           0        0    NaN   NaN
967         Lvl    AllPub  …         0         200        0    NaN   NaN
968         Lvl    AllPub  …         0           0        0    NaN   NaN
969         Lvl    AllPub  …         0           0        0    NaN   NaN

    MiscFeature MiscVal  YrSold  SaleCondition  SalePrice
0           NaN       0    2010         Normal     288330
1           NaN       0    2010         Normal     183164
2           NaN       0    2007         Alloca     362145
3           NaN       0    2006         Normal     196079
4           NaN       0    2010        Partial     228515
..          …     …     …            …     …
965         NaN       0    2008         Normal     413403
966         NaN       0    2006         Normal     336805
967         NaN       0    2007         Normal     259324
968         NaN       0    2009         Normal      96478
969         NaN       0    2009         Normal     197073

[970 rows x 69 columns]
```

[7]:
```python
data.duplicated().sum()
```

[7]:
```
np.int64(0)
```

[29]:
```python
# filling the missing values

# fill numeric columns with median
data = data.fillna(data.median(numeric_only=True))

# fill categorical columns with mode
for col in data.select_dtypes(include='object'):
    data[col] =  data[col].fillna(data[col].mode()[0])
```

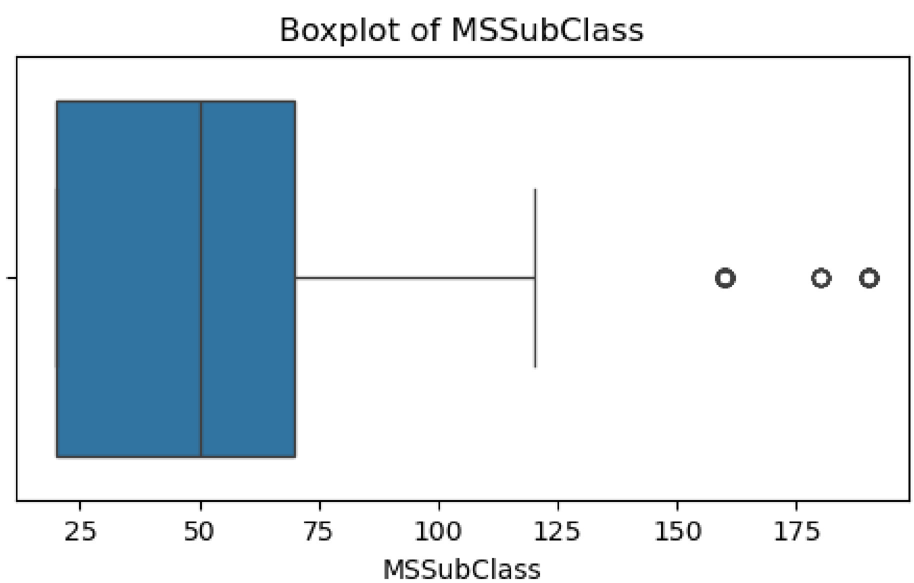
[30]:
```python
data.isnull().sum()
```

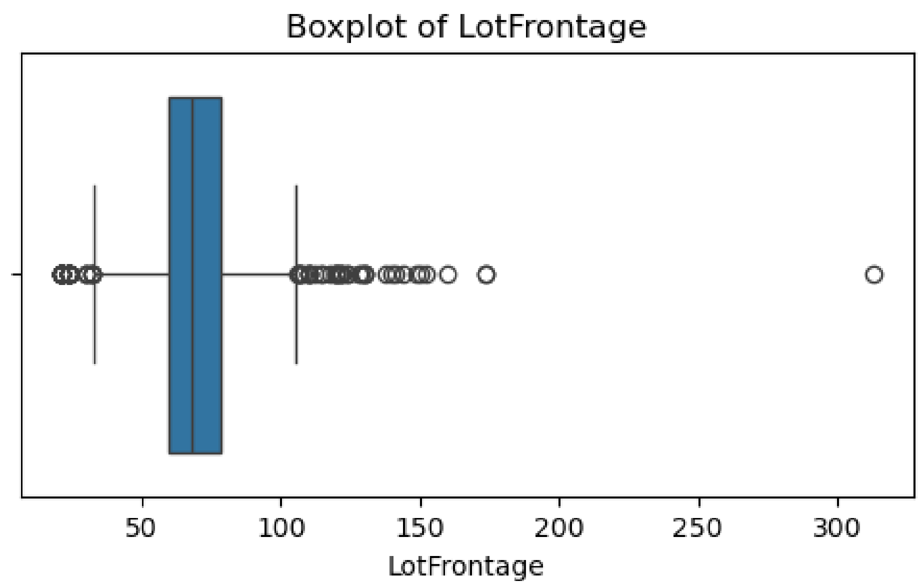
[30]:
```
\                0
MSSubClass       0
MSZoning         0
```

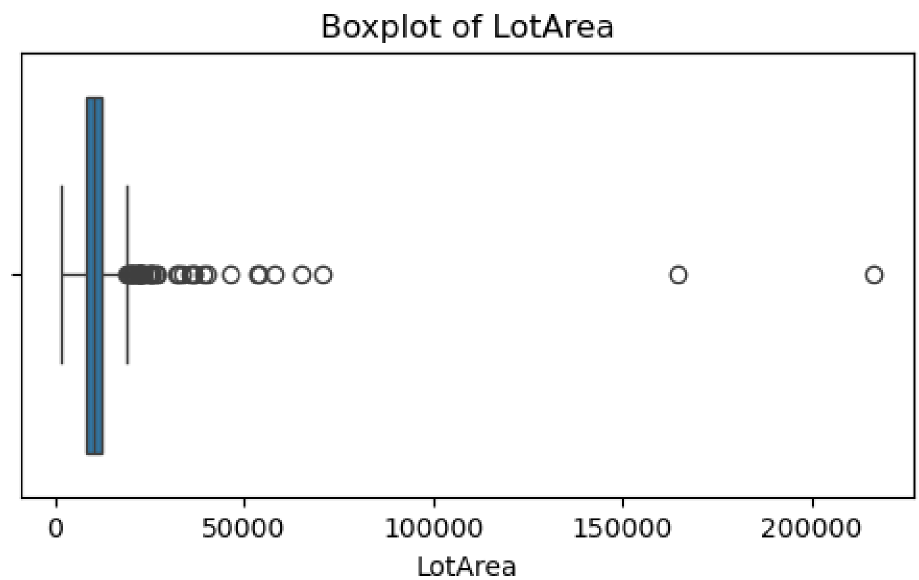
```
LotFrontage      0
LotArea          0
                ..
MiscFeature      0
MiscVal          0
YrSold           0
SaleCondition    0
SalePrice        0
Length: 69, dtype: int64
```

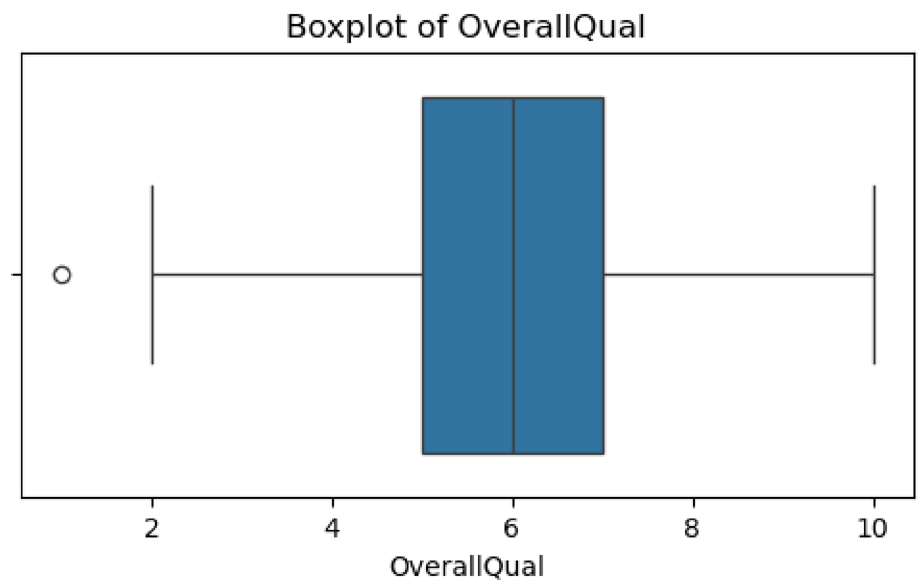
[69]:

```
# outlier detection and handling:

numeric_cols = data.select_dtypes(include=['int64', 'float64']).columns

for col in numeric_cols:
    plt.figure(figsize=(6,3))
    sns.boxplot(x=data[col])
    plt.title(f"Boxplot of {col}")
    plt.show()
```

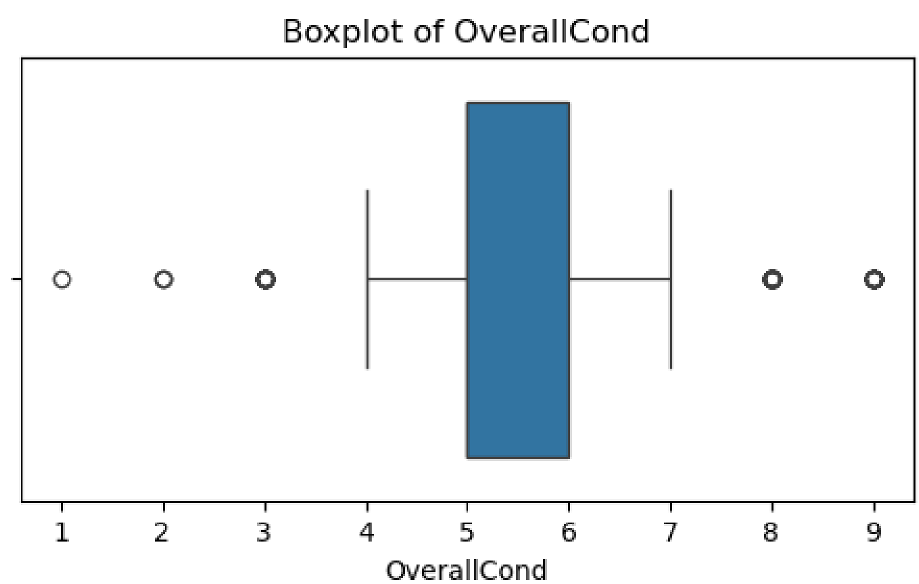



Boxplot of LotFrontage
50
100
150
200
250
300
LotFrontage

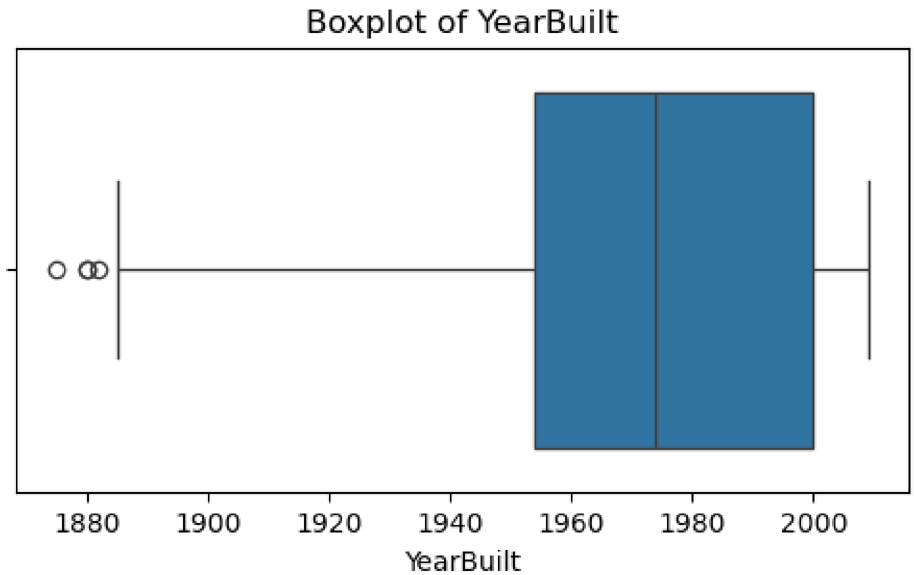

Boxplot of LotArea
0
50000
100000
150000
200000
LotArea

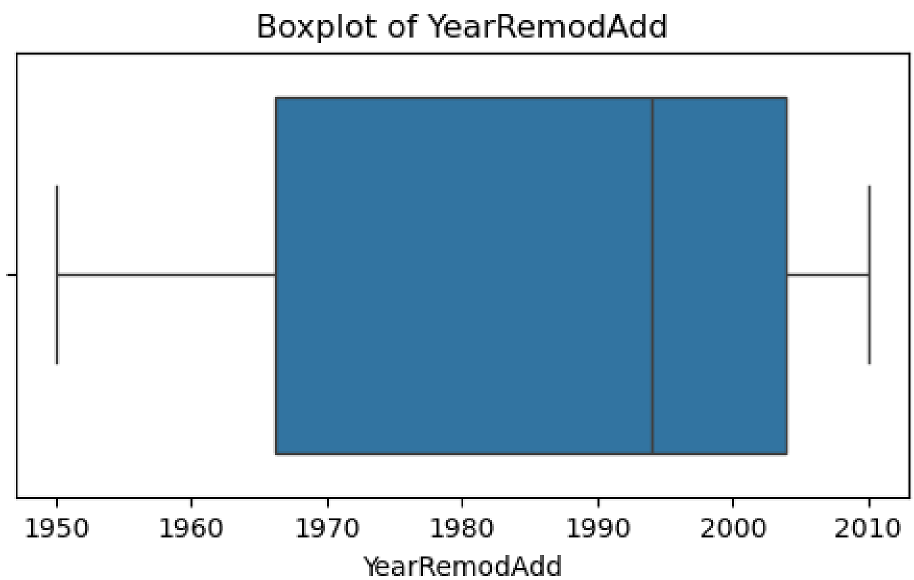


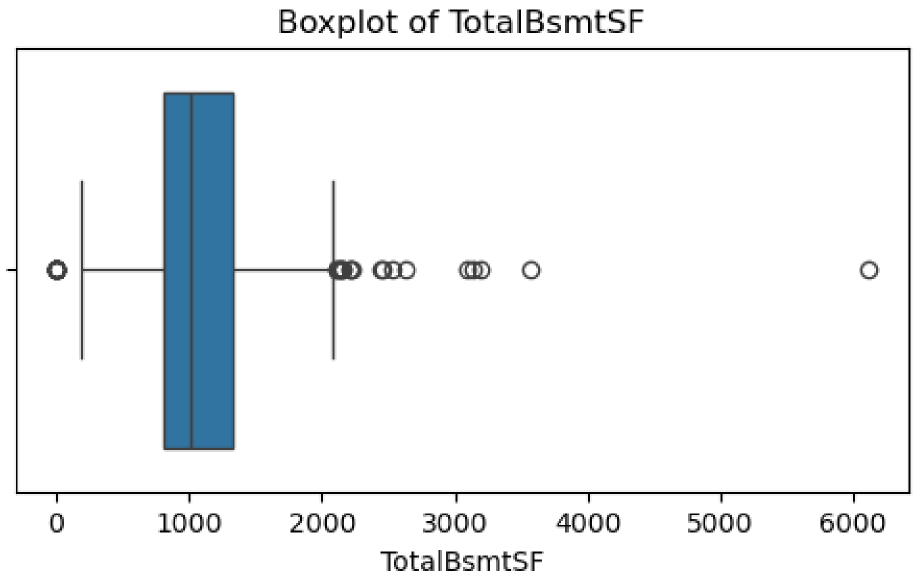
Boxplot of OverallQual
2
4
6
8
10
OverallQual



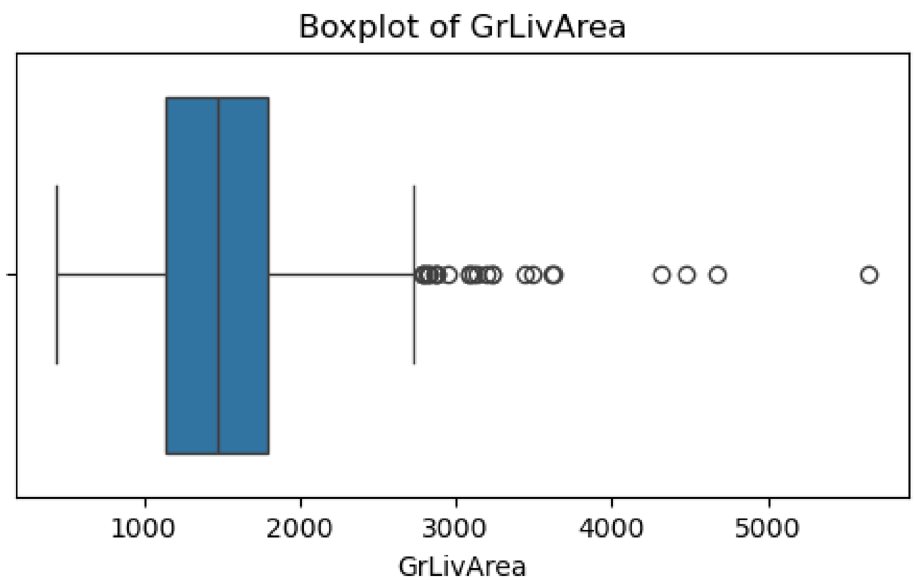
Boxplot of OverallCond
1
2
3
4
5
6
7
8
9
OverallCond


Boxplot of YearBuilt
1880
1900
1920
1940
1960
1980
2000
YearBuilt

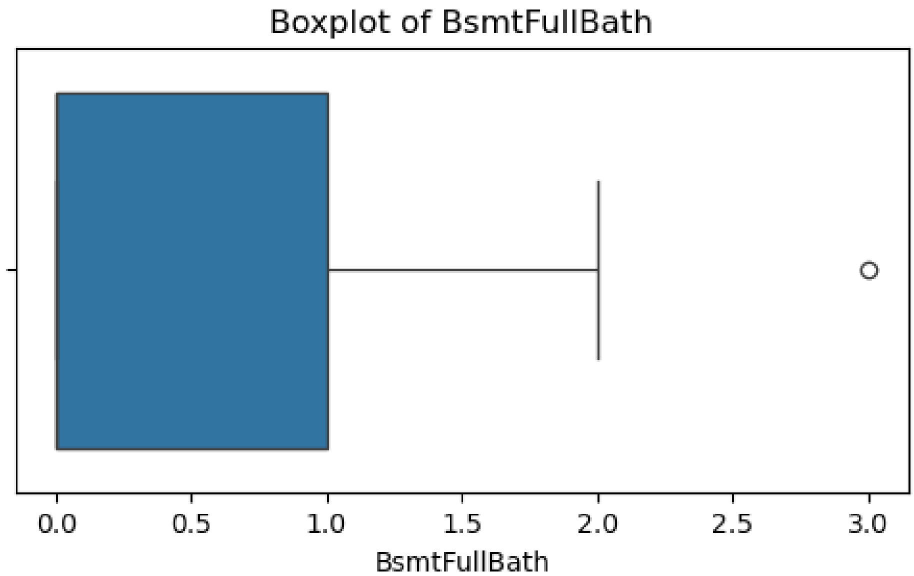

Boxplot of YearRemodAdd
1950
1960
1970
1980
1990
2000
2010
YearRemodAdd

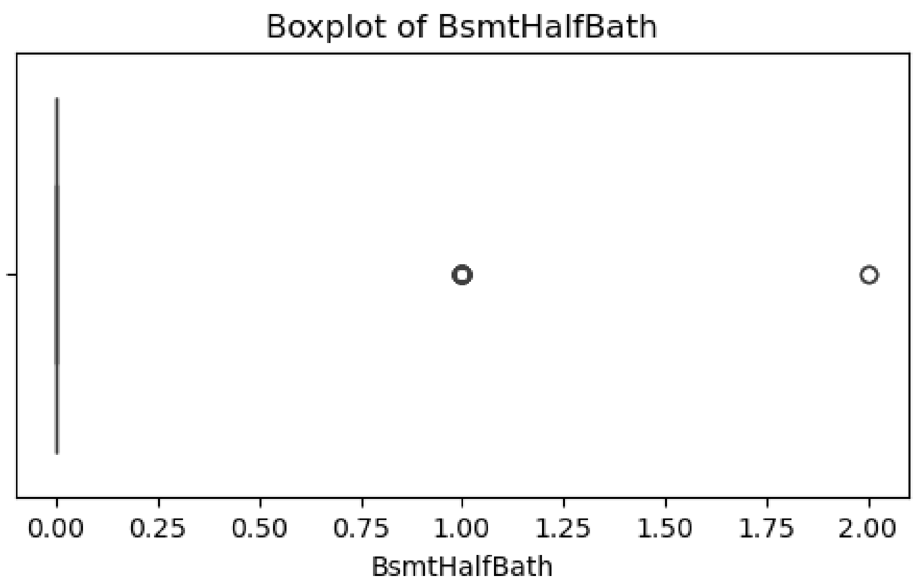


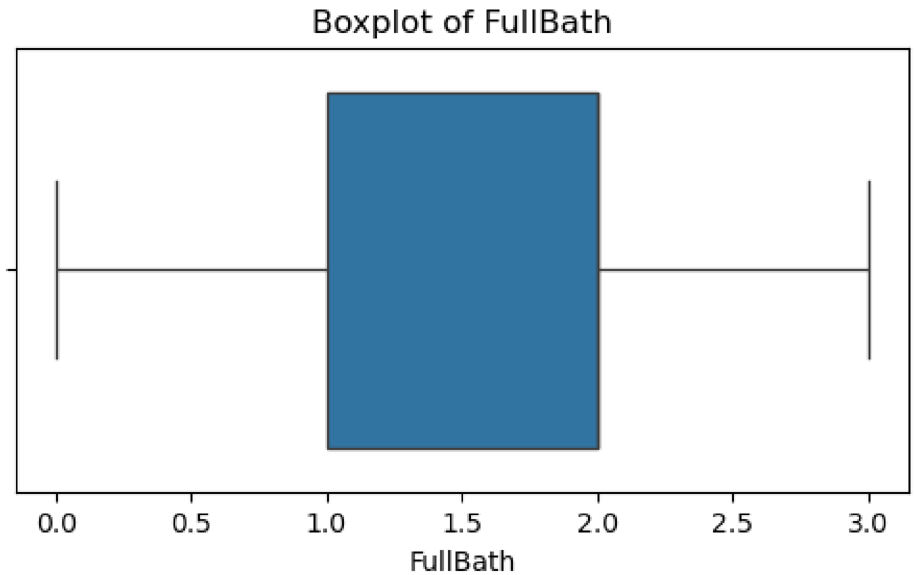
Boxplot of TotalBsmtSF
0
1000
2000
3000
4000
5000
6000
TotalBsmtSF

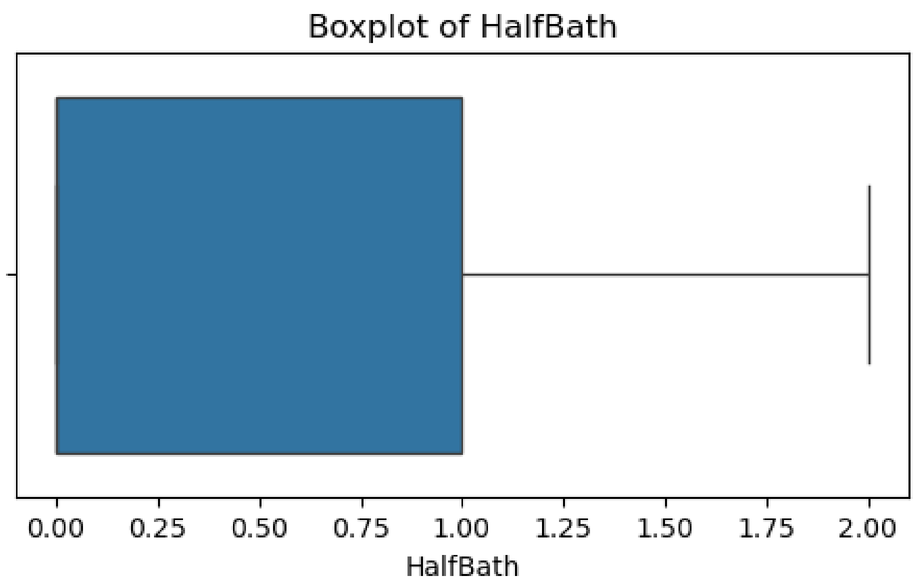


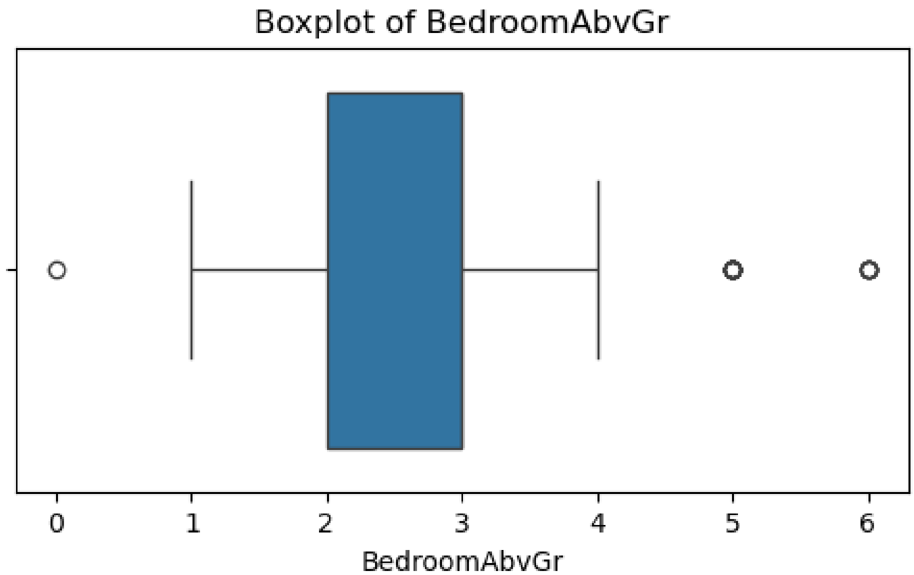
Boxplot of GrLivArea
1000
2000
3000
4000
5000
GrLivArea

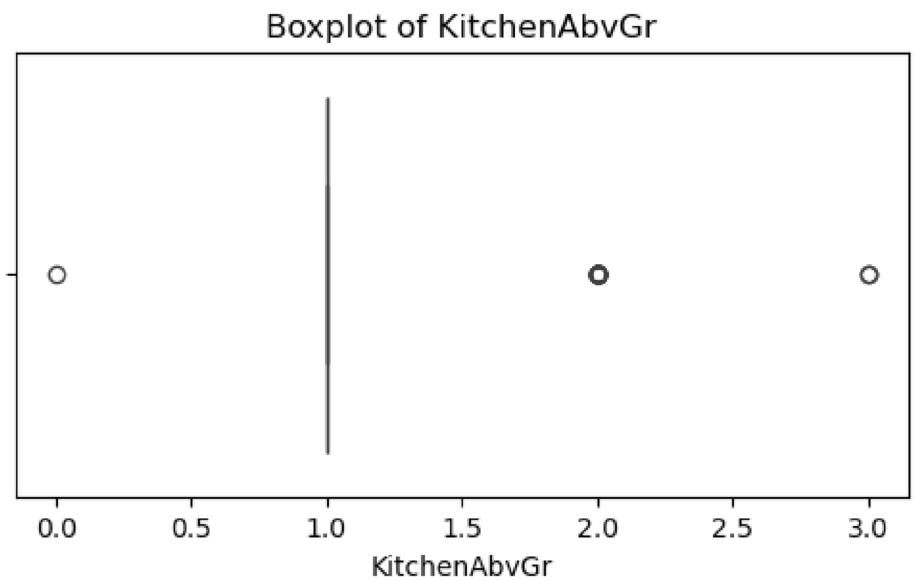

Boxplot of BsmtFullBath
0.0
0.5
1.0
1.5
2.0
2.5
3.0
BsmtFullBath

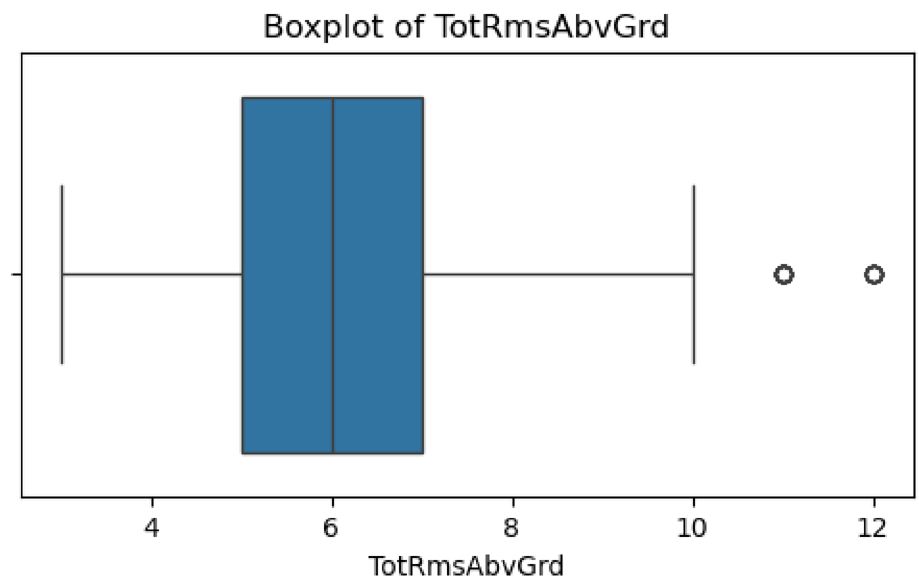

Boxplot of BsmtHalfBath
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
BsmtHalfBath

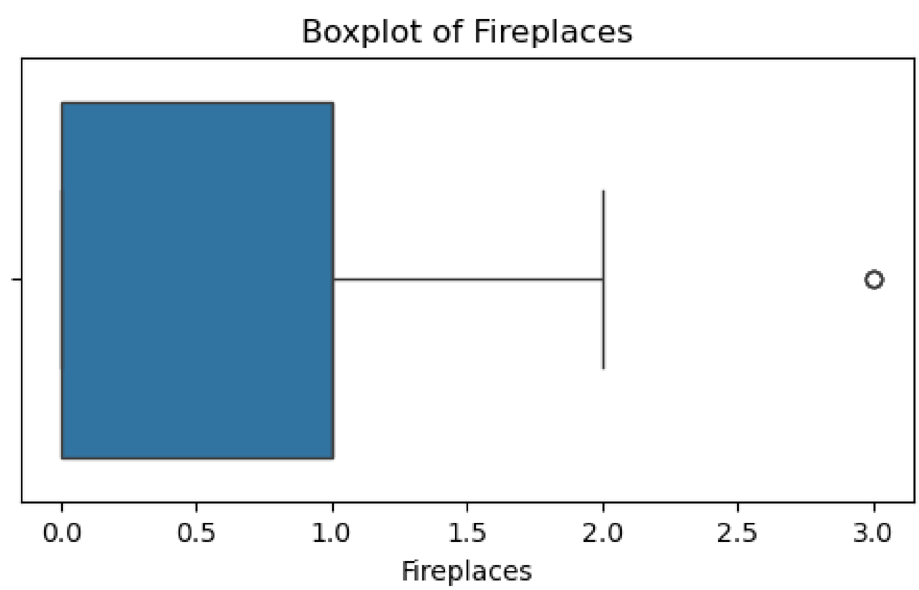

Boxplot of FullBath
0.0
0.5
1.0
1.5
2.0
2.5
3.0
FullBath

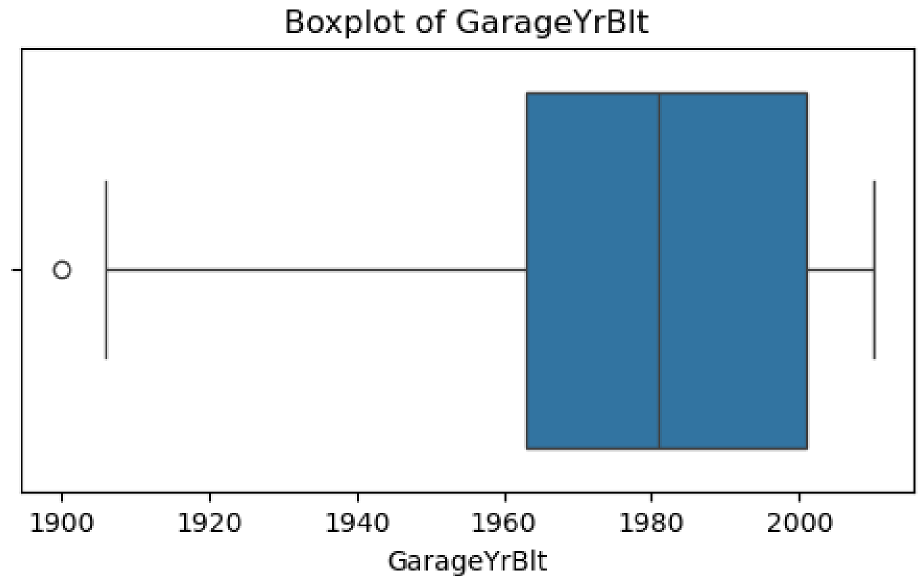

Boxplot of HalfBath
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
HalfBath


Boxplot of BedroomAbvGr
0
1
2
3
4
5
6
BedroomAbvGr

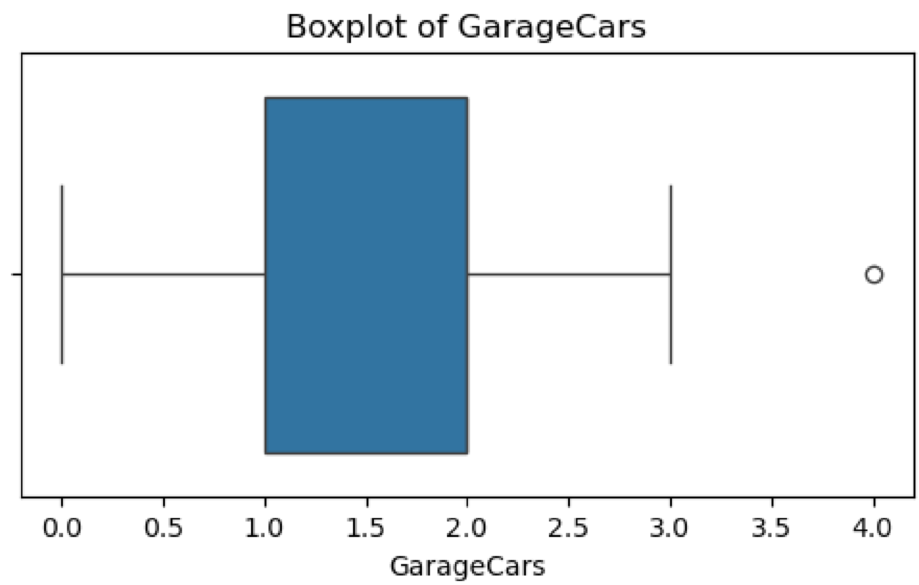

Boxplot of KitchenAbvGr
0.0
0.5
1.0
1.5
2.0
2.5
3.0
KitchenAbvGr


Boxplot of TotRmsAbvGrd
4
6
8
10
12
TotRmsAbvGrd

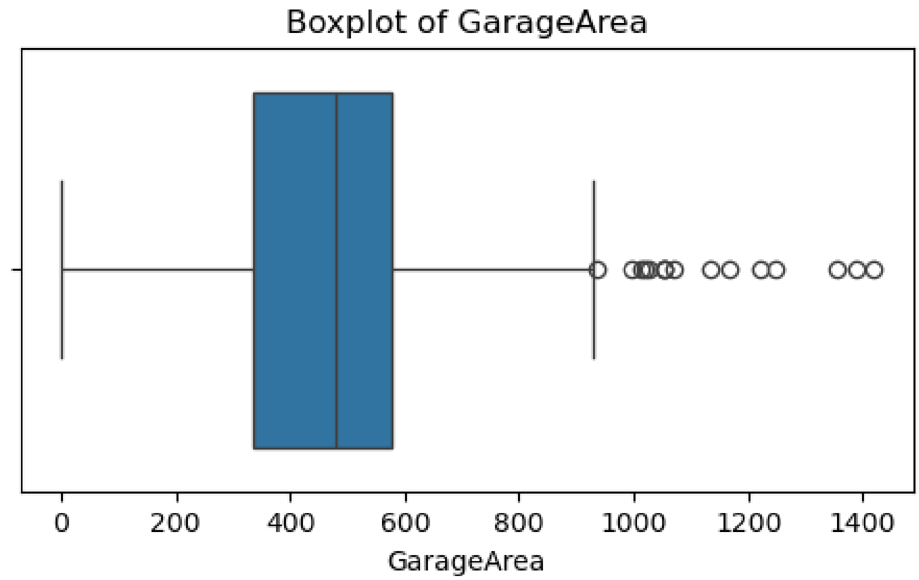

Boxplot of Fireplaces
0.0
0.5
1.0
1.5
2.0
2.5
3.0
Fireplaces

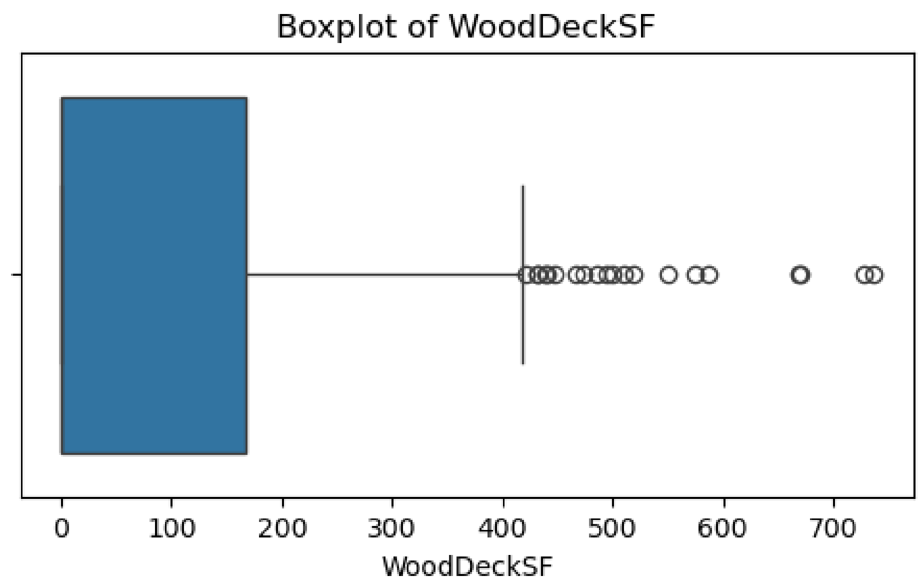


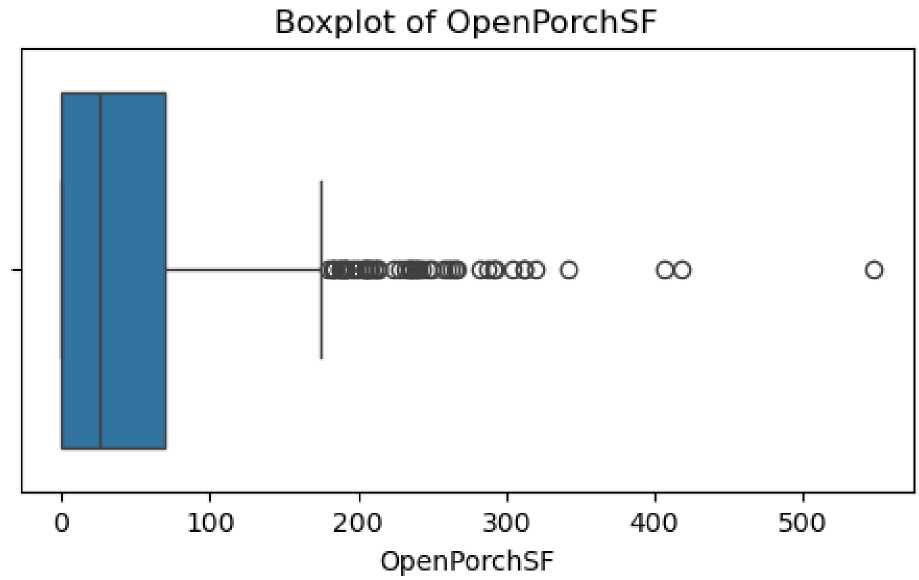
Boxplot of GarageYrBlt
1900
1920
1940
1960
1980
2000
GarageYrBlt

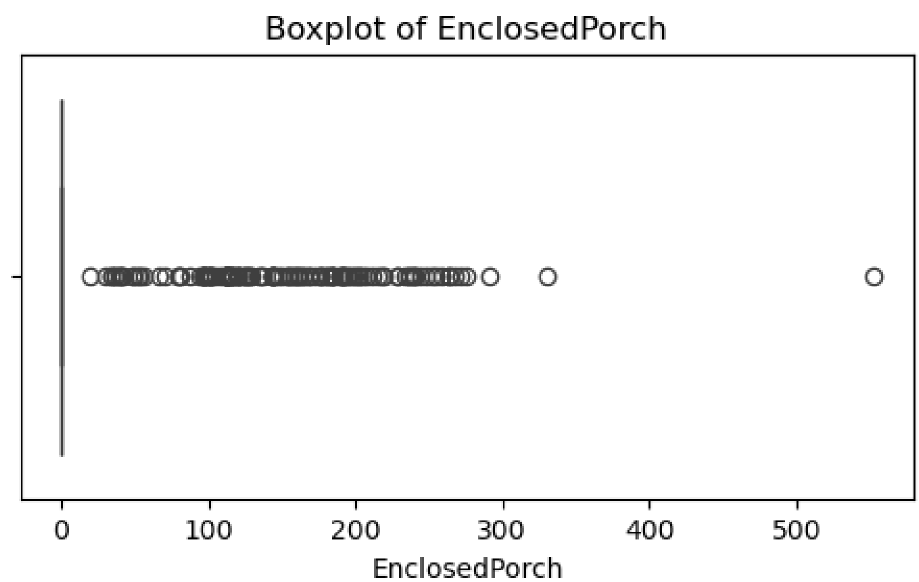


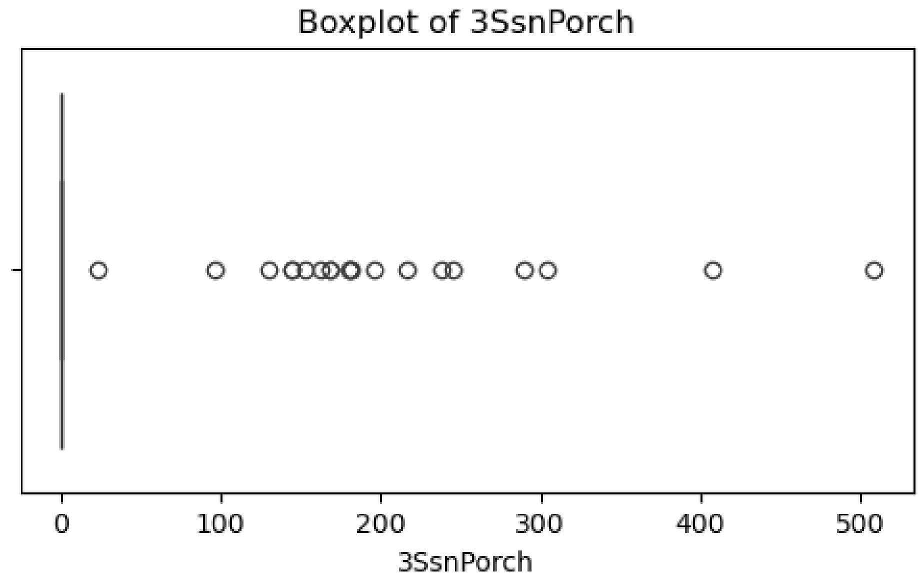
Boxplot of GarageCars
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
GarageCars

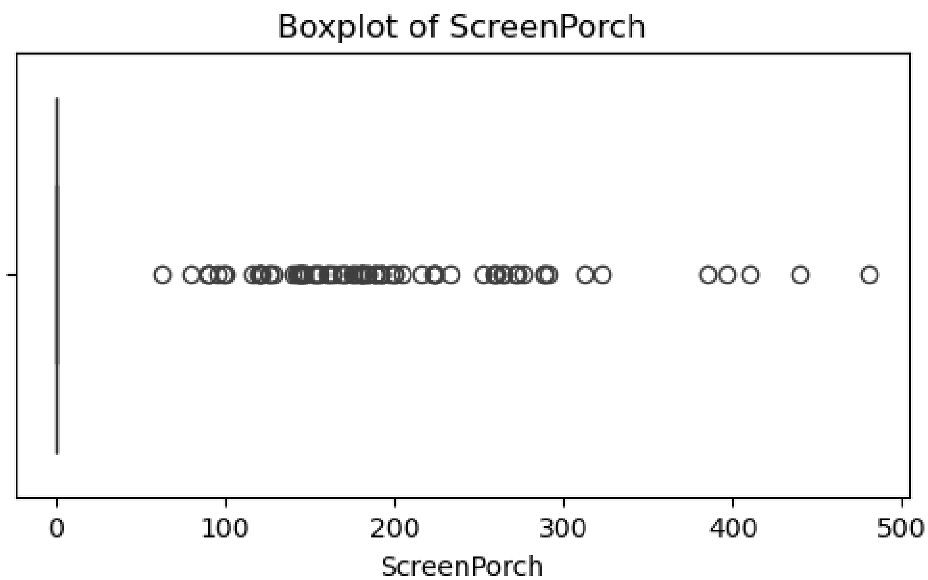

Boxplot of GarageArea
0
200
400
600
800
1000
1200
1400
GarageArea

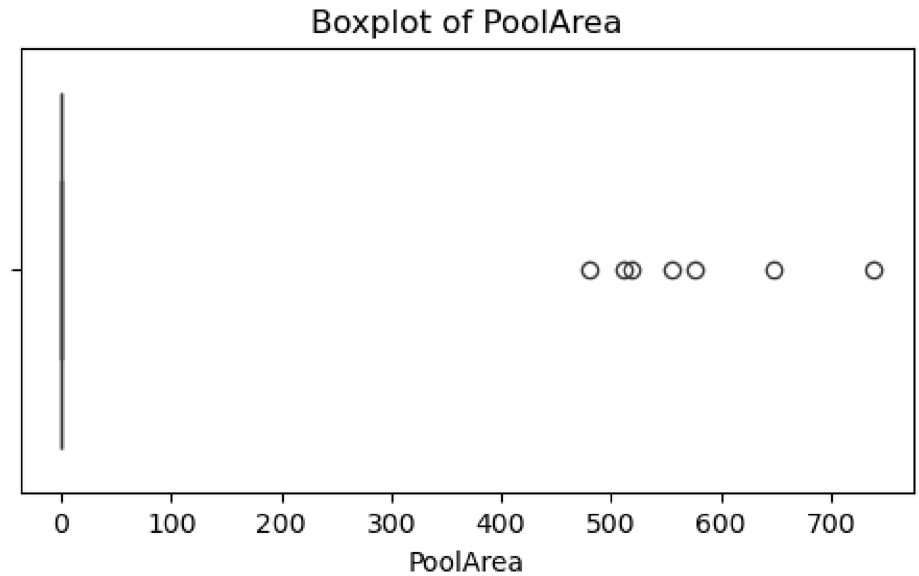

Boxplot of WoodDeckSF
0
100
200
300
400
500
600
700
WoodDeckSF

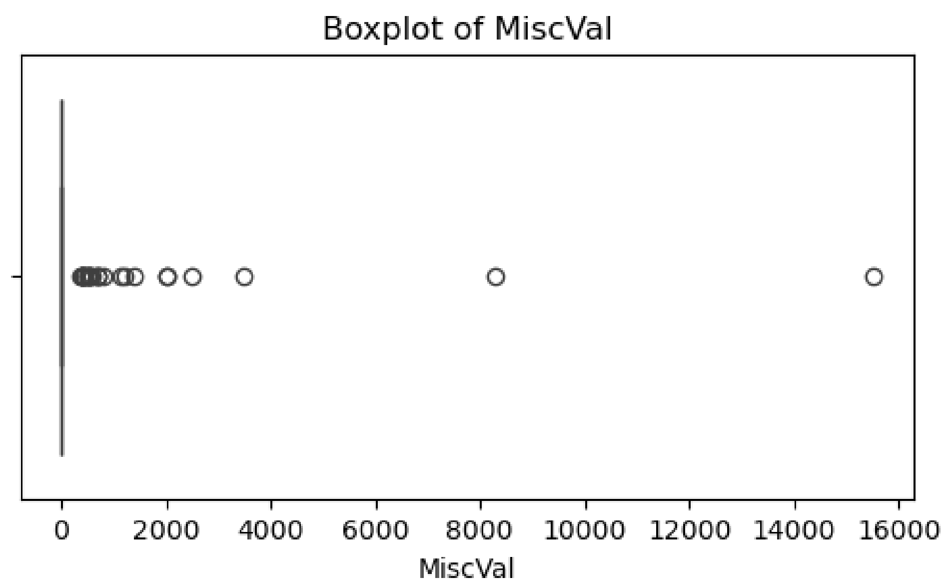

Boxplot of OpenPorchSF
0
100
200
300
400
500
OpenPorchSF

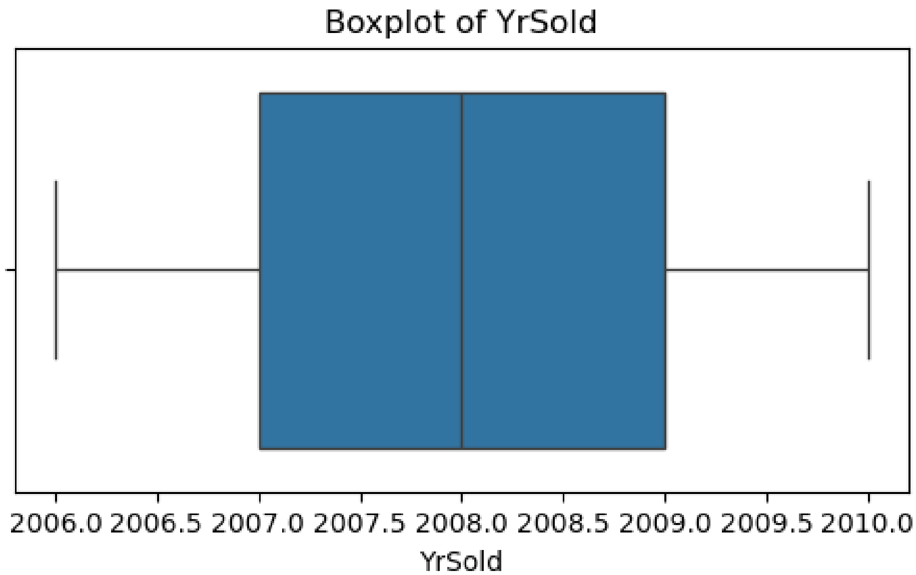

Boxplot of EnclosedPorch
0
100
200
300
400
500
EnclosedPorch

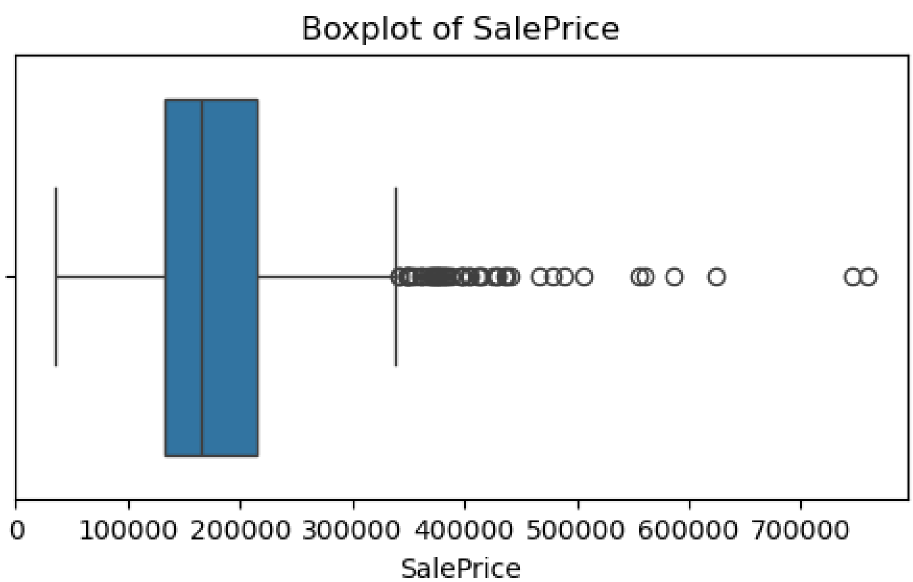

Boxplot of 3SsnPorch
0
100
200
300
400
500
3SsnPorch


Boxplot of ScreenPorch
0
100
200
300
400
500
ScreenPorch


Boxplot of PoolArea
0
100
200
300
400
500
600
700
PoolArea


Boxplot of MiscVal
0
2000
4000
6000
8000
10000
12000
14000
16000
MiscVal





[70]:
```
# removing outliers

def remove_outliers_iqr(data, cols):
    cleaned = data.copy()
    for col in cols:
        Q1 = cleaned[col].quantile(0.25)
        Q3 = cleaned[col].quantile(0.75)
```

```
        IQR = Q3 - Q1

        lower = Q1 - 1.5*IQR
        upper = Q3 + 1.5*IQR

        cleaned = cleaned[(cleaned[col] >= lower) & (cleaned[col] <= upper)]
    return cleaned

df_clean = remove_outliers_iqr(data, numeric_cols)
print(df_clean.shape)
```

```
(421, 1180)
```

[54]:
```
# converting categorical columns into numeric

data_model = pd.get_dummies(df_clean, drop_first=True)

data_model.head()
```

[54]:
```
   MSSubClass  LotFrontage  LotArea  OverallQual  OverallCond  YearBuilt  \
1          60         77.0     9534            6            5       1985
3         120         37.0     3728            8            5       2005
4         120         44.0     6606            7            5       2009
5          60         58.0    18002            7            5       1998
6          20         67.0     9769            7            5       2002

   YearRemodAdd  TotalBsmtSF  GrLivArea  BsmtFullBath  …  Fence_MnPrv  \
1          1985          741       1732             0  …         True
3          2005         1247       1247             1  …         True
4          2010         1358       1358             1  …         True
5          1998         1195       1839             0  …         True
6          2002         1573       1573             1  …         True

   Fence_MnWw  MiscFeature_Othr  MiscFeature_Shed  MiscFeature_TenC  \
1       False             False              True             False
3       False             False              True             False
4       False             False              True             False
5       False             False              True             False
6       False             False              True             False

   SaleCondition_AdjLand  SaleCondition_Alloca  SaleCondition_Family  \
1                  False                 False                 False
3                  False                 False                 False
4                  False                 False                 False
5                  False                 False                 False
6                  False                 False                 False

   SaleCondition_Normal  SaleCondition_Partial
```

```
1                   True                      False
3                   True                      False
4                  False                       True
5                   True                      False
6                   True                      False

[5 rows x 1180 columns]
```

[55]:
```python
# feature correlation analysis: understand which numeric features strongly␣
↪correlate with
# the target SalePrice.

corr = data_model.corr()

# sort correlation of all features with Saleprice
corr_with_target = corr["SalePrice"].sort_values(ascending=False)
print(corr_with_target.head(10))
```

```
SalePrice          1.000000
OverallQual        0.841627
GrLivArea          0.781242
GarageCars         0.705403
GarageArea         0.685849
YearBuilt          0.663157
FullBath           0.645180
GarageYrBlt        0.624319
Foundation_PConc   0.617840
TotRmsAbvGrd       0.607961
Name: SalePrice, dtype: float64
```

[44]:
```python
# Explaination of feature correlation analysis:

# higher correlation - stronger impact on price
# feature like overallqual, grlivearea, garagecars usually rank highest
```

[68]:
```python
# define independent(x) and dependent(y) variable

x = data_model.drop("SalePrice", axis=1)
y = data_model["SalePrice"]
```

[66]:
```python
# split the data

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,␣
↪random_state=42)

# initialise the linear regression model on the training dataset

model = LinearRegression()
```

```
model.fit(x_train, y_train)

print("model training completed")
```

```
model training completed
```

[67]:
```
#  predicting on the test data

y_pred = model.predict(x_test)

# evaluate the model
mae = mean_absolute_error(y_test,y_pred)
mse = mean_squared_error(y_test,y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test,y_pred)

# performance metrics calculation
print("model evaluation metrics:")
print(f"mean absolute error (mae): {mae:.2f}")
print(f"mean squared error (mse): {mse:.2f}")
print(f"root measured squared error (rmse): {rmse:.2f}")
print(f"r2 score: {r2:.2f}\n")
```

```
model evaluation metrics:
mean absolute error (mae): 15593.09
mean squared error (mse): 453635879.40
root measured squared error (rmse): 21298.73
r2 score: 0.87
```

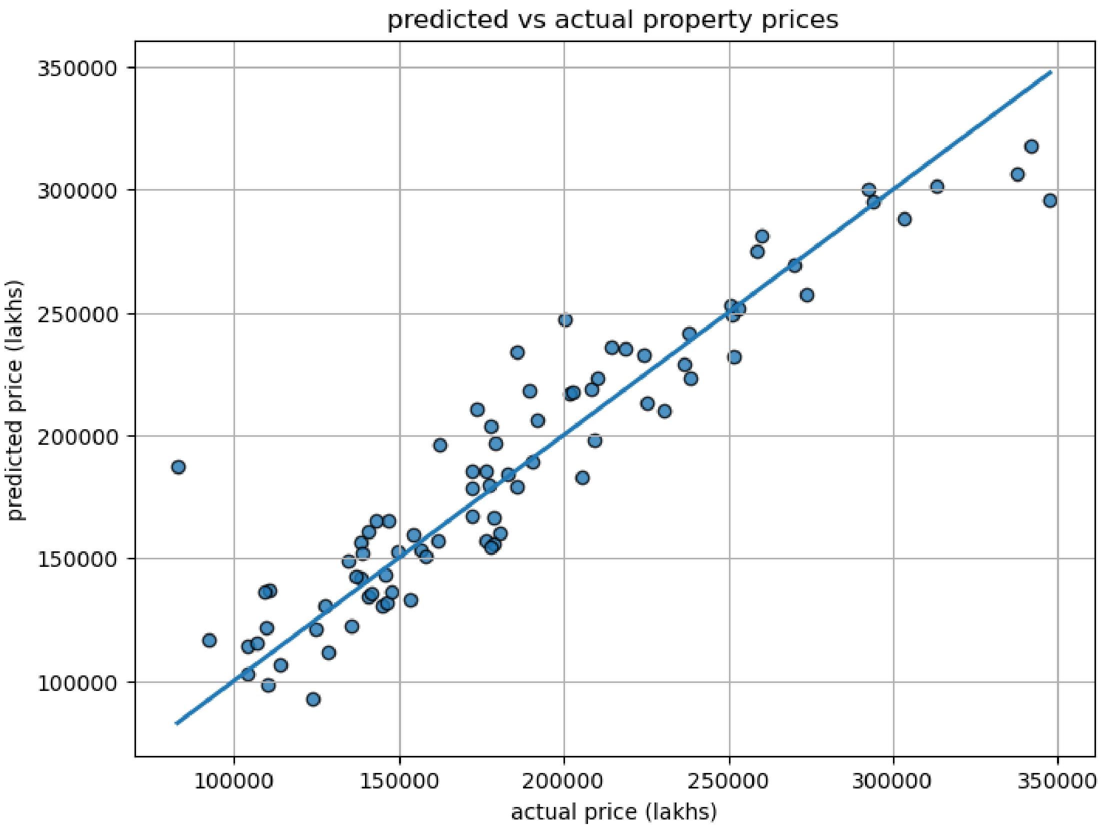
[59]:
```
# check important features (coefficients)

importance = pd.DataFrame({
    'Feature' : x.columns,
    'Coefficient': model.coef_
}).sort_values(by='Coefficient', ascending=False)

importance.head(10)
```

[59]:

	Feature	Coefficient
320	_PR014	55229.652942
921	_PR0886	33039.479336
769	_PR0681	28839.074153
913	_PR0878	28051.299698
75	_PR0108	27700.936378
174	_PR01213	26704.940316
772	_PR0684	25944.913291
42	_PR01028	24514.730709

```
264   \_PRO1320   24279.832532
68    \_PRO1066   24256.777782
```

[60]:
```
#  graphical evaluation for model performance

#  predicted vs actual plot
plt.figure(figsize=(8,6))
plt.scatter(y_test, y_pred, alpha=0.8,edgecolor="k")

line = np.linspace(min(y_test.min(), y_pred.min()), max(y_test.max(), y_pred.
 ↪max()), 100)
plt.plot(line, line, linewidth=2)

plt.xlabel("actual price (lakhs)")
plt.ylabel("predicted price (lakhs)")
plt.grid(True)
plt.title("predicted vs actual property prices")
plt.show()
```



[16]:
```
# Buisness interpretation:

# 1. Strong Positive Relationship (Model is Learning Well)
# Most points are clustered close to the diagonal line, which represents␣
↪perfect predictions.
# This indicates that the model is able to capture the overall pricing pattern␣
↪correctly.

# 2. Good Performance for Mid-Range Properties
# For properties priced between  1 lakh to  3.5 lakh (100k - 350k), the␣
↪predicted values closely match the actual values, showing:
#     Consistent model behavior
#     Low prediction error
#     Reliable accuracy in the common price range

# 3. Slight Under/Over-Prediction at Higher Prices
# For higher-priced properties (above  4 lakh / 400k), the scatter points begin␣
↪moving away from the ideal line, which means:
#     The model may under-predict some expensive properties
#     A few points are far from the line → higher error
#     The linear model may not fully capture the non-linear behavior at high␣
↪price ranges
# This is normal when using a simple linear regression model.

# 4. Business Insight
# Overall, the model can be used for:
#     Property price estimation
#     Basic valuation
#     Pricing decisions for average-priced properties
#     Market trend analysis
```

[ ]:
```
# Business Recommendations

# Based on the model insights and feature relationships, here are practical␣
↪business recommendations:

# Focus on the Top Price-Driving Features
# The model shows that certain property features have the strongest correlation␣
↪with price (e.g., area, bedrooms, location score, property age, amenities).
# Recommendation:
# Real-estate companies should prioritize acquiring and marketing properties␣
↪that score high on these features since they significantly increase selling␣
↪price.

# Improve Listings With High-Impact Features
# Properties with certain high-correlation features can be made more valuable.
```

```
# Examples:
# Add amenities (parking, lift, security)
# Renovate older properties
# Improve interior quality
# These upgrades can increase predicted selling price, leading to better␣
↪returns.

# Identify Undervalued Properties
# Since the model predicts a fair price:
# Properties selling below the model prediction can be targeted as "undervalued␣
↪opportunities".
# Investors can purchase and renovate these for profit.

# Target Marketing Based on Customer Budget
# The model helps predict price ranges.
# Real-estate companies can:
# Match customers to properties that fit their affordability
# Reduce search time
# Improve conversion rates

# Use Model Predictions for Better Negotiation
# The model provides a data-driven benchmark price.
# Agents can:
# Use predicted price as a reference in negotiations
# Avoid overpricing and losing customers
# Avoid underpricing and losing revenue
```

[ ]:
```
# Model Limitations

# Linear Regression only captures linear relationships
# Real-estate pricing often depends on non-linear factors (e.g., large area␣
↪increments increase price exponentially).
# Linear regression oversimplifies these relationships → reducing accuracy.

# Sensitive to Outliers
# Your model performance changed significantly before/after outlier removal.
# Linear Regression is not robust → extreme values distort coefficients.

# Multicollinearity may still exist
# Some features may be highly correlated (e.g., area, rooms).
# This makes model coefficients unstable and affects interpretability.

# Dataset may not include all price-driving factors
# Missing real-world factors:
# Distance to metro
# Crime rate
# Builder reputation
```

```
# Floor number
# Road width
# Because these are absent, the model cannot fully capture true price behavior.

# Scaling did not improve performance, after scaling the value of r2 is␣
↪decreasing to 0.82.
# This confirms that feature distributions are not ideal, and the data may be␣
↪skewed.

# Limited generalization
# If the dataset covers only a specific region (city/locality), the model␣
↪cannot be applied to other regions without retraining.
```

[72]:
```
# Future Work / Improvements

# Try Advanced Models (Non-linear & More Accurate)
# To achieve better accuracy, test:
# Random Forest Regressor
# XGBoost
# Gradient Boosting Regressor
# These models handle non-linearity and outliers far better than Linear␣
↪Regression.
# Accuracy can increase to 0.93-0.96.

# Build Feature Engineering Pipeline
# Improve features by adding:
# Price per square foot
# Age category (0-5 yrs, 5-15 yrs, 15+ yrs)
# Location rating
# Number of amenities
# Furnishing level category
# Better features → better predictions.

# Use Regularization Techniques
# To reduce multicollinearity:
# Ridge Regression
# Lasso Regression
# ElasticNet
# These make the model more stable and interpretable.

# Add More Real-Estate Data
# Add important features:
# Distance to schools
# Locality tier (A/B/C)
# Crime/safety score
# Connectivity score
# Floor number
```

```
# Facing direction
# These can significantly improve R².

# Build a Residual Diagnostics Report
# Check:
# Homoscedasticity
# Normality of residuals
# Autocorrelation
# These ensure the model meets all assumptions.
```

[ ]:

```

```