House Prices

House Prices: Advanced Prediction Techniques (Kaggle)

Ritesh Kumar

2024WI_MS_DSP_422-DL_SEC61: Practical Machine Learning

Module 3 Assignment

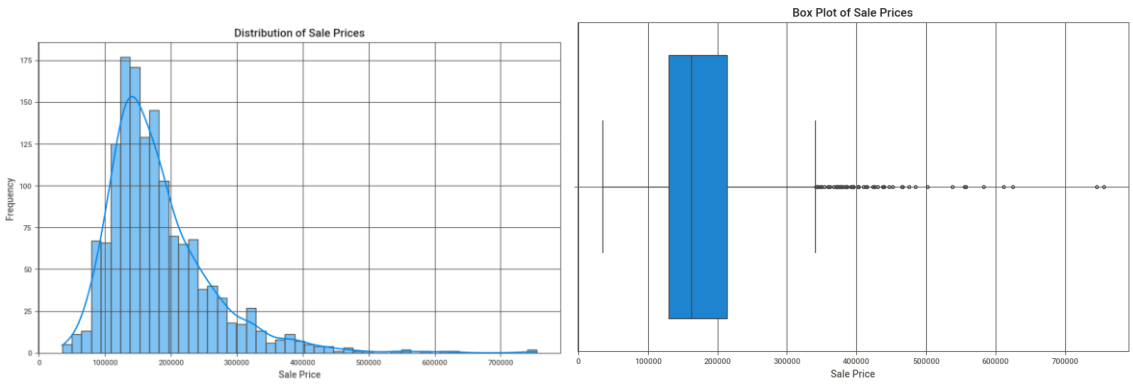
House Prices: Advanced Prediction Techniques

Donald Wedding and Narayana Darapaneni

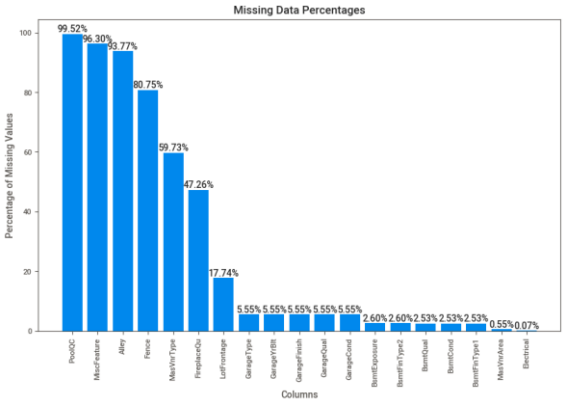
January 21, 2024

Approach

We start with a comprehensive Exploratory Data Analysis (EDA), commencing with the extraction of descriptive statistics for the 'SalePrice' column, serving as the target variable for prediction. This entails computing the count, mean, standard deviation, minimum, quartiles, and maximum values.



The script meticulously identifies columns with missing data, presenting the count and names of these columns. We remove all features that exhibit over 90% missing values. For the remaining features, we infer that rows with missing values indicate the absence of those features in the respective properties. Consequently, we substitute all missing values with 0.

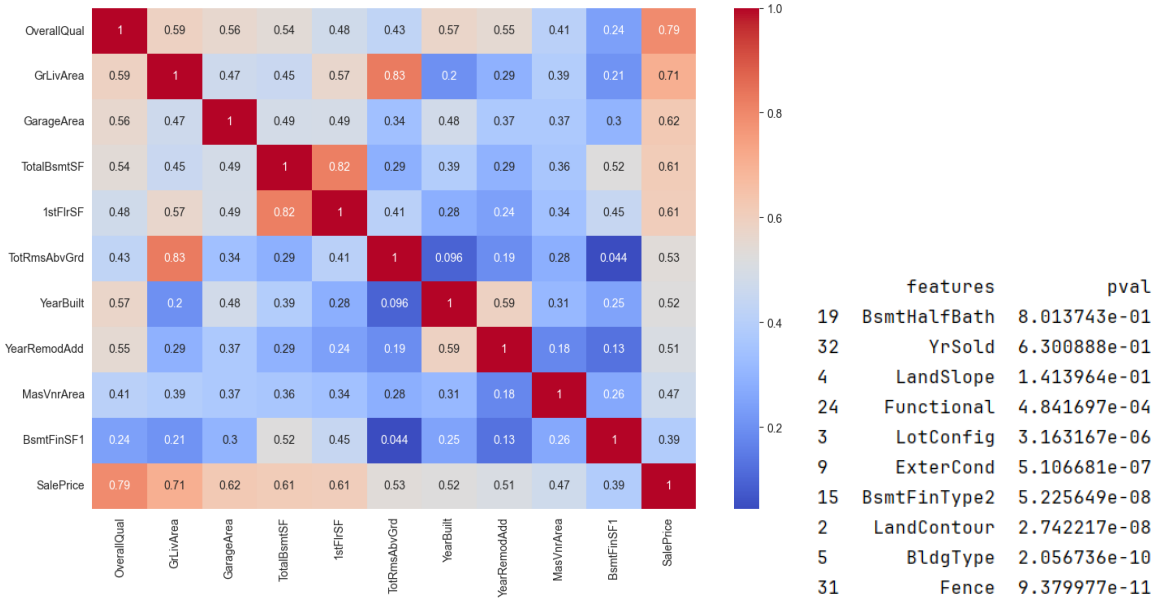


Post-handling missing data, categorical features are recognized based on unique values, with those containing eight or fewer unique values considered as such. Further refinement

involves dropping columns with less than 1% of values for more than half of the categories, a strategic dimensionality reduction.

The script endeavors to engineer new features pertinent to prediction, summing up square footage and porch-related columns. A Chi-Square test evaluates categorical variable correlation, ensuring independence.

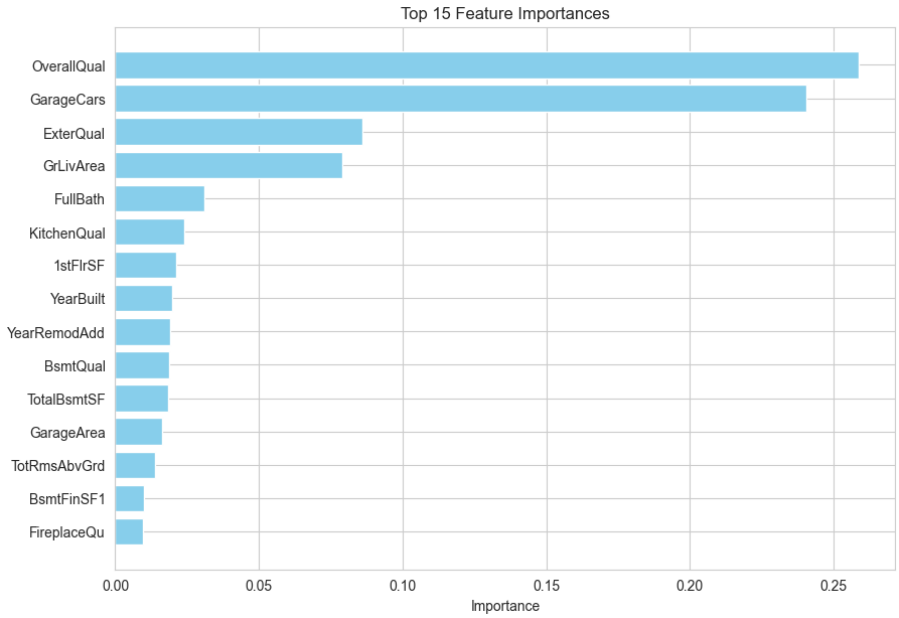
Relationship of SalePrice with the most important categorical and numerical features is explored.



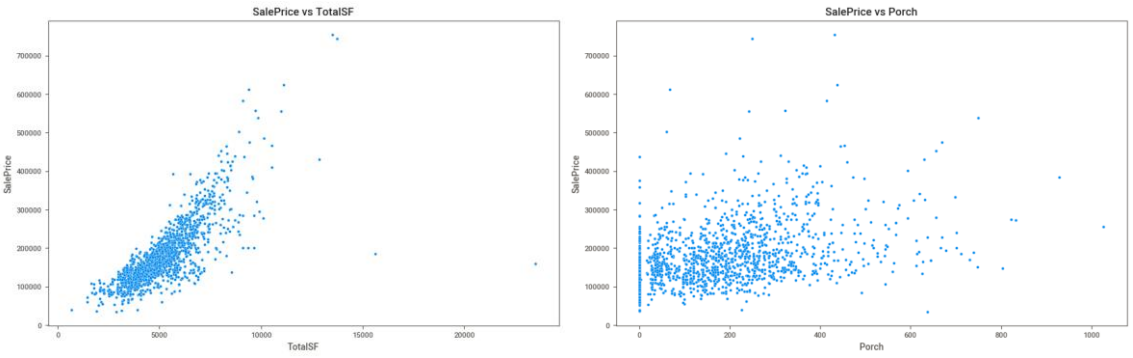
	features	pval
19	BsmtHalfBath	8.013743e-01
32	YrSold	6.300888e-01
4	LandSlope	1.413964e-01
24	Functional	4.841697e-04
3	LotConfig	3.163167e-06
9	ExterCond	5.106681e-07
15	BsmtFinType2	5.225649e-08
2	LandContour	2.742217e-08
5	BldgType	2.056736e-10
31	Fence	9.379977e-11

All numerical features are analyzed for outliers, and only two of 28 numerical features are found to be having more than 10% outliers.

Random Forest algorithm is used for identification of the most influential features, encompassing both numerical and categorical variables, that exert the strongest impact on the 'SalePrice.



Two new variables – TotalSF and Porch are created by adding all the variables for the covered area and porch (in sf), respectively. And, the relationship of SalePrice with both is explored.



Highly correlated features, both categorical and numerical, are identified, and removed.

Feature list is refined. The script specifies features for removal, optimizing the dataset for model training.

Application of the 'OneHotEncoder' from sklearn.preprocessing converts categorical features to a one-hot numeric array, enhancing compatibility with classification and regression algorithms.

Ridge regression, Lasso regression, and ElasticNet models are trained, incorporating 5-fold cross-validation to gauge accuracy and prevent overfitting.

Subsequently, the script prepares test data ('test.csv') mirroring operations applied to the training data. Trained models predict test data outcomes, stored in CSV files.

These files were uploaded on Kaggle (https://www.kaggle.com/riteshrk) , and the scores are:

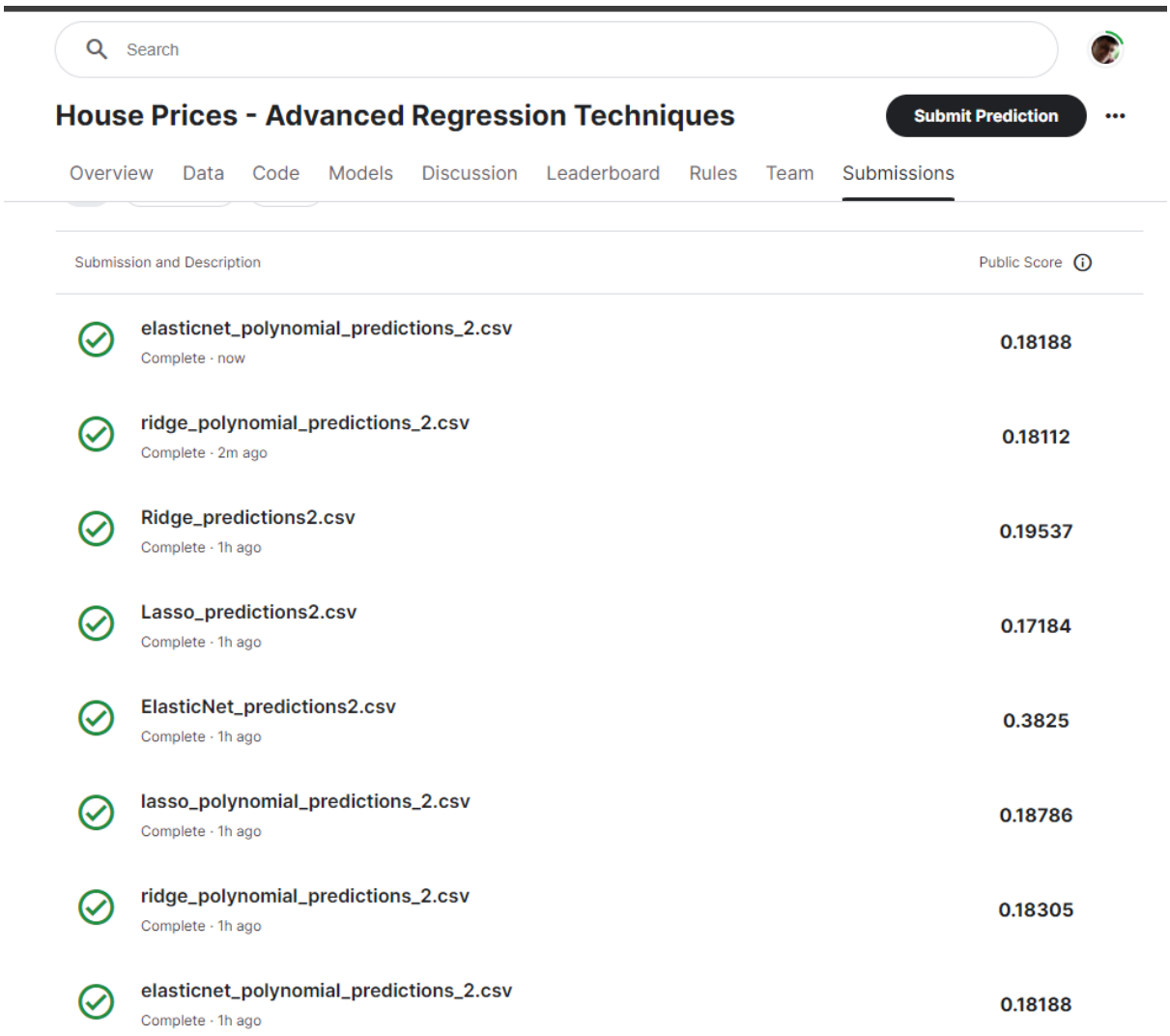
- Elastic Net: 0.3825
- Ridge: 0.19537
- **Lasso: 0.17184**

Next, we used to scikit-learn on Lasso to perform a grid-search for hyperparameter tuning (polynomial_degree: 1 to 3, alpha [50, 100, 250]), and Cross-Validation = 5, and the best parameters were found to be alpha = 250 and polynomial_degree = 1. Kaggle submission score is 0.18786.

After that, we used to scikit-learn on Ridge to perform a grid-search for hyperparameter tuning (polynomial_degree: 1 to 3, alpha [10, 15, 20, 25, 30, 35, 40]), and Cross-Validation = 5, and the best parameters were found to be alpha = 25 and polynomial_degree = 1. Kaggle submission score is 0.18112.

Finally, we used to scikit-learn on ElasticNet to perform a grid-search for hyperparameter tuning (polynomial_degree: 1 to 3, alpha [0.01, 0.05, 0.1], l1_ratio [0.4, 0.5, 0.6], and Cross-Validation = 5, and the best parameters were found to be alpha = 0.05, l1_ratio = 0.5 and polynomial_degree = 1. Kaggle submission score is 0.18188.

The submission results can be viewed at https://www.kaggle.com/competitions/house-prices-advanced-regression-techniques/submissions.



The best score 0.17184 on Kaggle is returned by our Lasso model.

The jupyter notebook has been uploaded on Kaggle and can be viewed at https://www.kaggle.com/riteshrk/lasso-ridge-elasticnet/edit.

We have identified the 15 features with the highest coefficients from the optimal model i.e. the Lasso:

	Feature	Coefficient
2	LotArea	23512.849789
8	KitchenAbvGr	12363.232246
108	BedroomAbvGr_8	9043.688564
10	GarageYrBlt	8417.092221
96	FullBath_2	7657.397947
6	YearRemodAdd	7196.729958
1	LotFrontage	5185.720666
121	Fireplaces_1	5132.109243
13	MiscVal	4369.607298
52	MasVnrType_Stone	3394.404878
63	Foundation_CBlock	3309.211871
33	LandSlope_Gtl	3004.900549
98	HalfBath_0	2931.432311
3	OverallQual	2765.820458
124	FireplaceQu_0	2707.349560

Coefficients: The coefficients returned by a LASSO regression model represent the strength and type of the relationship between each feature and the target variable, SalePrice.

Breakdown of the coefficients:

1. LotArea: A larger lot area significantly increases property values.
2. KitchenAbvGr: A higher number of above-ground kitchens (like in a duplex or apartment building) seems to have a very high positive influence on property value.
3. BedroomAbvGr_8: Properties with eight bedrooms above ground are valued higher, possibly indicating large family homes or commercial residential investments like B&Bs.
4. GarageYrBlt: Newer garages seem to add to property value, suggesting that modern construction or renovations are favorable.
5. FullBath_2: Having two full bathrooms above grade is a significant positive for property value.
6. YearRemodAdd: Recently remodeled or added constructions boost the property's value.
7. LotFrontage: Having a larger street connected to the property marginally increases value.
8. Fireplaces_1: The presence of at least one fireplace adds value, indicating a preference for this feature.

9. MiscVal: Miscellaneous values also play a role in increasing property value, suggesting unique features or amenities are important.
10. MasVnrType_Stone: A stone masonry veneer is highly valued, indicating a preference for stone finishes.
11. Foundation_CBlock: Cinder block foundations are valued, possibly for their durability or the property style they signify.
12. LandSlope_Gtl: Properties with a gentle slope have a slight positive impact on value.
13. HalfBath_0: No half baths surprisingly increase value, which may indicate a preference for more full baths or simpler layouts.
14. OverallQual: Overall quality has a strong positive impact on value, as expected.
15. FireplaceQu_0: Having no fireplace or a low-quality fireplace reduces the value, indicating the desirability of higher-quality fireplaces.

Management Recommendations: Given these findings, several strategic recommendations can be made:

1. Focus on Quality and Modern Features: Prioritize high-quality construction and modern amenities like new garages and stone masonry.
2. Maximize Value-Adding Features: Include at least two full bathrooms and ensure there are fireplaces in the properties, as they have shown to increase value significantly.
3. Highlight Unique Features: Any unique or miscellaneous features that a property possesses should be emphasized in listings, as they can add substantial value.
4. Invest in Kitchen and Bedroom Optimizations: Properties with multiple kitchens or a larger number of bedrooms fetch higher prices and might appeal to investors in rental properties or large family homes.

5. Consider Lot Characteristics: Maintain or improve lot frontage and ensure the property has a gentle slope, as these aspects have positive influences on value.
6. Capitalize on Renovations: Market the value of recent remodels or additions, as they are shown to increase property values.

By leveraging these insights, property managers, real estate agents, and developers can make informed decisions to enhance property appeal and market value effectively.

Code

In [1]:
```
# Import Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
from scipy.stats import pearsonr
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import LabelEncoder

import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
df = pd.read_csv('train.csv')
```

In [3]:
```
df.shape
```

Out[3]:
```
(1460, 81)
```

In [4]:
```
df.columns
```

Out[4]:
```
Index(['Id', 'MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'Street',
       'Alley', 'LotShape', 'LandContour', 'Utilities', 'LotConfig',
       'LandSlope', 'Neighborhood', 'Condition1', 'Condition2', 'BldgType',
       'HouseStyle', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd',
       'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrType',
       'MasVnrArea', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual',
       'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinSF1',
       'BsmtFinType2', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'Heating',
       'HeatingQC', 'CentralAir', 'Electrical', '1stFlrSF', '2ndFlrSF',
       'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath',
       'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'KitchenQual',
       'TotRmsAbvGrd', 'Functional', 'Fireplaces', 'FireplaceQu', 'GarageType',
       'GarageYrBlt', 'GarageFinish', 'GarageCars', 'GarageArea', 'GarageQual',
       'GarageCond', 'PavedDrive', 'WoodDeckSF', 'OpenPorchSF',
       'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'PoolQC',
       'Fence', 'MiscFeature', 'MiscVal', 'MoSold', 'YrSold', 'SaleType',
       'SaleCondition', 'SalePrice'],
      dtype='object')
```

In [5]:
```
# Drop the 'Id' column
df = df.drop('Id', axis=1)
df.shape
```

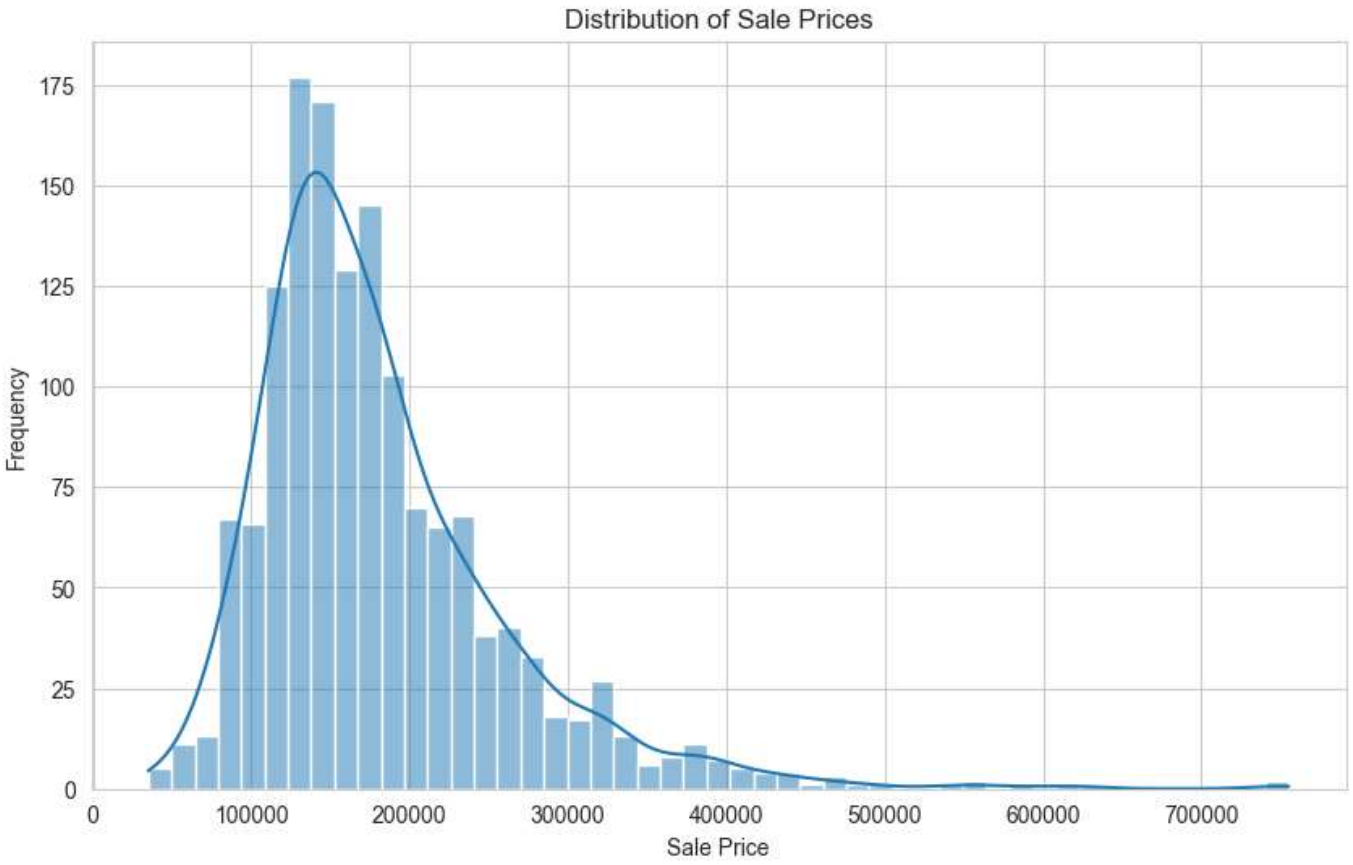
Out[5]:
```
(1460, 80)
```

In [6]:
```
# Descriptive statistics for SalePrice
desc_stats = df['SalePrice'].describe()
print("Descriptive Statistics for SalePrice:\n", desc_stats)
```

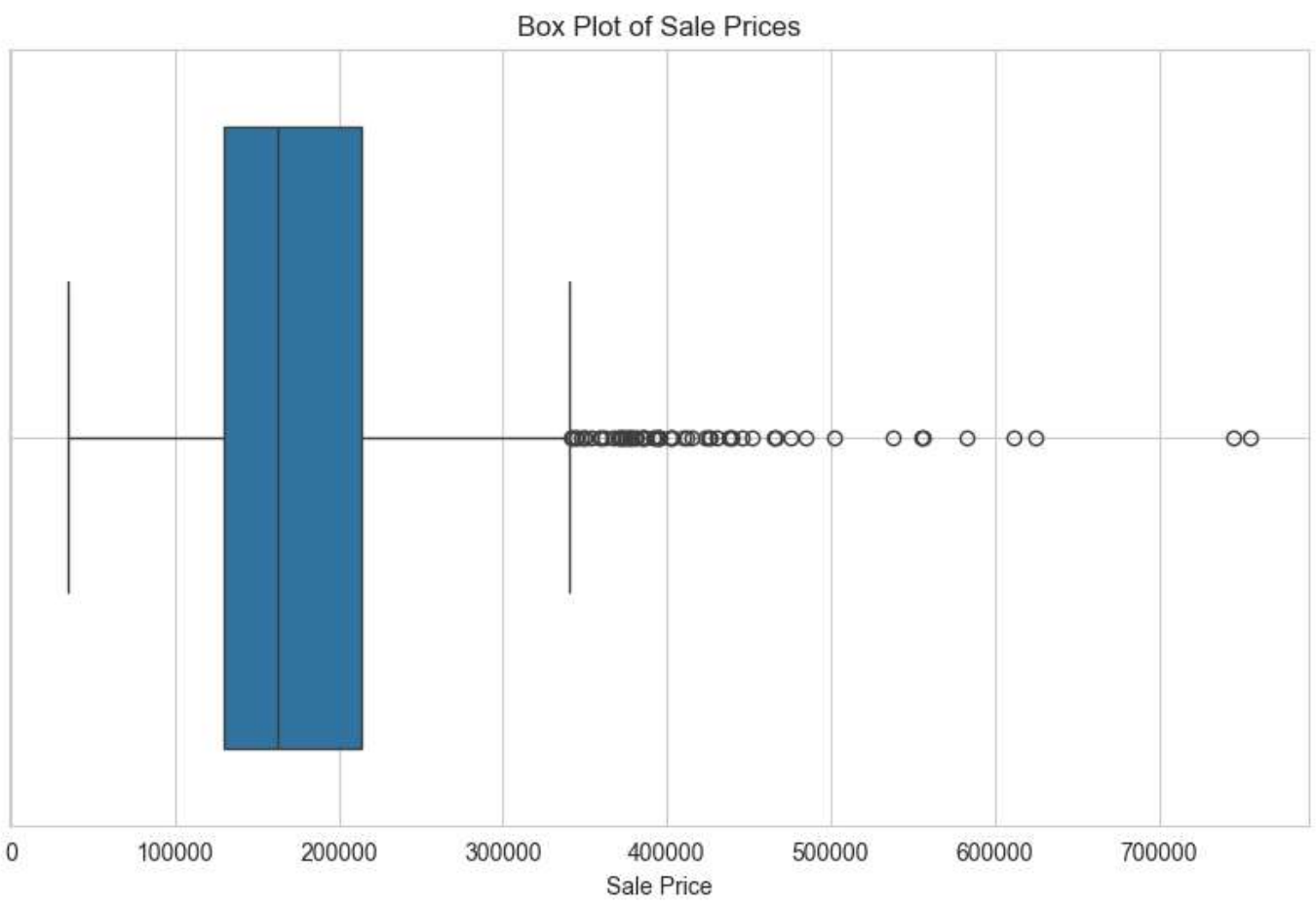
```
Descriptive Statistics for SalePrice:
 count      1460.000000
mean     180921.195890
std       79442.502883
min       34900.000000
25%      129975.000000
50%      163000.000000
75%      214000.000000
max      755000.000000
Name: SalePrice, dtype: float64
```

In [7]:
```
# Histogram with Kernel Density Estimate (KDE) for SalePrice
plt.figure(figsize=(10, 6))
```

```
sns.histplot(df['SalePrice'], kde=True)
plt.title('Distribution of Sale Prices')
plt.xlabel('Sale Price')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```



In [8]:
```
# Box plot for SalePrice
plt.figure(figsize=(10, 6))
sns.boxplot(x=df['SalePrice'])
plt.title('Box Plot of Sale Prices')
plt.xlabel('Sale Price')
plt.grid(True)
plt.show()
```



In [9]:
```
# Identify columns with missing values
columns_with_missing_values = df.columns[df.isnull().any()].tolist()

print(f"Number of columns with missing values: {len(columns_with_missing_values)}")
print("Names of Columns with Missing Data:\n", columns_with_missing_values)
```

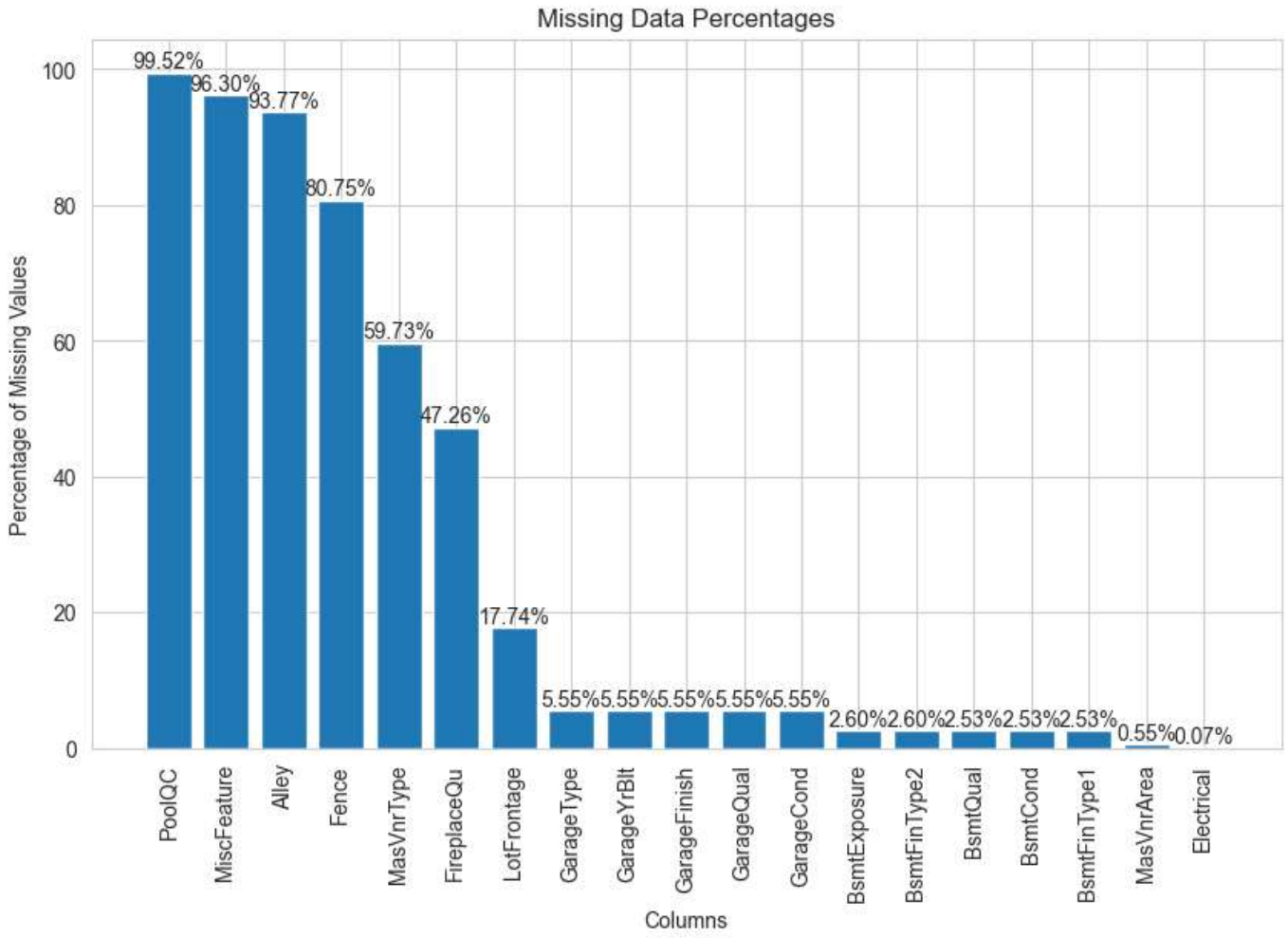
```
Number of columns with missing values: 19
Names of Columns with Missing Data:
 ['LotFrontage', 'Alley', 'MasVnrType', 'MasVnrArea', 'BsmtQual', 'BsmtCond', 'BsmtExposure',
'BsmtFinType1', 'BsmtFinType2', 'Electrical', 'FireplaceQu', 'GarageType', 'GarageYrBlt', 'Gar
ageFinish', 'GarageQual', 'GarageCond', 'PoolQC', 'Fence', 'MiscFeature']
```

In [10]:
```
# Calculate percentage of missing values for each column
missing_percentage_dict = {}
for column in columns_with_missing_values:
    missing_count = df[column].isnull().sum()
    total_count = len(df)
    missing_percentage = (missing_count / total_count) * 100
    missing_percentage_dict[column] = missing_percentage

# Sort and print the percentage of missing values in descending order
sorted_missing_percentage = sorted(missing_percentage_dict.items(), key=lambda x: x[1], rever
for column, missing_percentage in sorted_missing_percentage:
    print(f"Percentage of missing values in {column}: {missing_percentage:.2f}%")
```

```
Percentage of missing values in PoolQC: 99.52%
Percentage of missing values in MiscFeature: 96.30%
Percentage of missing values in Alley: 93.77%
Percentage of missing values in Fence: 80.75%
Percentage of missing values in MasVnrType: 59.73%
Percentage of missing values in FireplaceQu: 47.26%
Percentage of missing values in LotFrontage: 17.74%
Percentage of missing values in GarageType: 5.55%
Percentage of missing values in GarageYrBlt: 5.55%
Percentage of missing values in GarageFinish: 5.55%
Percentage of missing values in GarageQual: 5.55%
Percentage of missing values in GarageCond: 5.55%
Percentage of missing values in BsmtExposure: 2.60%
Percentage of missing values in BsmtFinType2: 2.60%
Percentage of missing values in BsmtQual: 2.53%
Percentage of missing values in BsmtCond: 2.53%
Percentage of missing values in BsmtFinType1: 2.53%
Percentage of missing values in MasVnrArea: 0.55%
Percentage of missing values in Electrical: 0.07%
```

In [11]:

```python
# Segregate the values
sorted_columns = [item[0] for item in sorted_missing_percentage]
sorted_percentages = [item[1] for item in sorted_missing_percentage]

# Plot the sorted missing data percentages
plt.figure(figsize=(10, 6))
plt.bar(sorted_columns, sorted_percentages)
plt.xlabel('Columns')
plt.ylabel('Percentage of Missing Values')
plt.title('Missing Data Percentages')
plt.xticks(rotation=90)

# Add data labels to the bar plot
for i, v in enumerate(sorted_percentages):
    plt.text(i, v, f"{v:.2f}%", ha='center', va='bottom')

plt.show()
```



In [12]:
```python
# Create a dictionary to save value counts of all features with missing values
value_counts_dict = {}

# Calculate the value counts for all features with missing values
for feature in sorted_missing_percentage:
    feature_name = feature[0]
    feature_value_counts = df[feature_name].value_counts().to_dict()
    value_counts_dict[feature_name] = feature_value_counts

# Print the dictionary
for key, value in value_counts_dict.items():
    print("Feature: ", key)
    print(value)
    print('\n')
```

```
Feature:  PoolQC
{'Gd': 3, 'Ex': 2, 'Fa': 2}


Feature:  MiscFeature
{'Shed': 49, 'Gar2': 2, 'Othr': 2, 'TenC': 1}


Feature:  Alley
{'Grvl': 50, 'Pave': 41}


Feature:  Fence
{'MnPrv': 157, 'GdPrv': 59, 'GdWo': 54, 'MnWw': 11}


Feature:  MasVnrType
{'BrkFace': 445, 'Stone': 128, 'BrkCmn': 15}


Feature:  FireplaceQu
{'Gd': 380, 'TA': 313, 'Fa': 33, 'Ex': 24, 'Po': 20}


Feature:  LotFrontage
{60.0: 143, 70.0: 70, 80.0: 69, 50.0: 57, 75.0: 53, 65.0: 44, 85.0: 40, 78.0: 25, 21.0: 23, 9
0.0: 23, 68.0: 19, 64.0: 19, 24.0: 19, 73.0: 18, 55.0: 17, 63.0: 17, 79.0: 17, 72.0: 17, 100.
0: 16, 66.0: 15, 74.0: 15, 51.0: 15, 52.0: 14, 59.0: 13, 82.0: 12, 71.0: 12, 67.0: 12, 43.0: 1
2, 40.0: 12, 57.0: 12, 76.0: 11, 69.0: 11, 92.0: 10, 34.0: 10, 53.0: 10, 86.0: 10, 88.0: 10, 8
4.0: 9, 35.0: 9, 62.0: 9, 44.0: 9, 77.0: 9, 93.0: 8, 98.0: 8, 96.0: 8, 61.0: 8, 95.0: 7, 58.0:
7, 120.0: 7, 107.0: 7, 94.0: 6, 81.0: 6, 36.0: 6, 30.0: 6, 48.0: 6, 91.0: 6, 41.0: 6, 54.0: 6,
89.0: 6, 110.0: 6, 105.0: 6, 32.0: 5, 47.0: 5, 87.0: 5, 56.0: 5, 37.0: 5, 83.0: 5, 102.0: 4, 4
2.0: 4, 49.0: 4, 104.0: 3, 103.0: 3, 99.0: 3, 45.0: 3, 108.0: 3, 130.0: 3, 124.0: 2, 114.0: 2,
109.0: 2, 313.0: 2, 134.0: 2, 115.0: 2, 121.0: 2, 116.0: 2, 118.0: 2, 129.0: 2, 101.0: 2, 174.
0: 2, 97.0: 2, 122.0: 2, 112.0: 1, 168.0: 1, 182.0: 1, 111.0: 1, 138.0: 1, 160.0: 1, 149.0: 1,
153.0: 1, 152.0: 1, 33.0: 1, 128.0: 1, 144.0: 1, 39.0: 1, 150.0: 1, 106.0: 1, 137.0: 1, 141.0:
1, 38.0: 1, 140.0: 1, 46.0: 1}


Feature:  GarageType
{'Attchd': 870, 'Detchd': 387, 'BuiltIn': 88, 'Basment': 19, 'CarPort': 9, '2Types': 6}


Feature:  GarageYrBlt
{2005.0: 65, 2006.0: 59, 2004.0: 53, 2003.0: 50, 2007.0: 49, 1977.0: 35, 1998.0: 31, 1999.0: 3
0, 1976.0: 29, 2008.0: 29, 2000.0: 27, 1968.0: 26, 2002.0: 26, 1950.0: 24, 1993.0: 22, 1958.0:
21, 1966.0: 21, 2009.0: 21, 1962.0: 21, 1965.0: 21, 1970.0: 20, 1996.0: 20, 1957.0: 20, 2001.
0: 20, 1978.0: 19, 1954.0: 19, 1960.0: 19, 1997.0: 19, 1974.0: 18, 1964.0: 18, 1994.0: 18, 199
5.0: 18, 1959.0: 17, 1956.0: 16, 1963.0: 16, 1990.0: 16, 1980.0: 15, 1979.0: 15, 1969.0: 15, 1
967.0: 15, 1973.0: 14, 1988.0: 14, 1920.0: 14, 1972.0: 14, 1940.0: 14, 1992.0: 13, 1961.0: 13,
1971.0: 13, 1955.0: 13, 1953.0: 12, 1948.0: 11, 1987.0: 11, 1925.0: 10, 1985.0: 10, 1981.0: 1
0, 1989.0: 10, 1941.0: 10, 1975.0: 9, 1991.0: 9, 1939.0: 9, 1930.0: 8, 1984.0: 8, 1949.0: 8, 1
983.0: 7, 1926.0: 6, 1986.0: 6, 1951.0: 6, 1936.0: 5, 1916.0: 5, 1922.0: 5, 1935.0: 4, 1946.0:
4, 1928.0: 4, 1931.0: 4, 1982.0: 4, 1945.0: 4, 1910.0: 3, 2010.0: 3, 1932.0: 3, 1952.0: 3, 193
8.0: 3, 1921.0: 3, 1924.0: 3, 1923.0: 3, 1947.0: 2, 1937.0: 2, 1942.0: 2, 1915.0: 2, 1918.0:
2, 1934.0: 2, 1914.0: 2, 1929.0: 2, 1927.0: 1, 1900.0: 1, 1906.0: 1, 1908.0: 1, 1933.0: 1}


Feature:  GarageFinish
{'Unf': 605, 'RFn': 422, 'Fin': 352}


Feature:  GarageQual
{'TA': 1311, 'Fa': 48, 'Gd': 14, 'Ex': 3, 'Po': 3}
```

Feature: GarageCond
{'TA': 1326, 'Fa': 35, 'Gd': 9, 'Po': 7, 'Ex': 2}

Feature: BsmtExposure
{'No': 953, 'Av': 221, 'Gd': 134, 'Mn': 114}

Feature: BsmtFinType2
{'Unf': 1256, 'Rec': 54, 'LwQ': 46, 'BLQ': 33, 'ALQ': 19, 'GLQ': 14}

Feature: BsmtQual
{'TA': 649, 'Gd': 618, 'Ex': 121, 'Fa': 35}

Feature: BsmtCond
{'TA': 1311, 'Gd': 65, 'Fa': 45, 'Po': 2}

Feature: BsmtFinType1
{'Unf': 430, 'GLQ': 418, 'ALQ': 220, 'BLQ': 148, 'Rec': 133, 'LwQ': 74}

Feature: MasVnrArea
{0.0: 861, 180.0: 8, 72.0: 8, 108.0: 8, 120.0: 7, 16.0: 7, 200.0: 6, 340.0: 6, 106.0: 6, 80.0:
6, 132.0: 5, 320.0: 5, 360.0: 5, 84.0: 5, 170.0: 5, 220.0: 4, 336.0: 4, 183.0: 4, 252.0: 4, 19
6.0: 4, 300.0: 4, 268.0: 4, 40.0: 4, 168.0: 4, 100.0: 4, 270.0: 4, 210.0: 4, 76.0: 4, 288.0:
4, 216.0: 4, 160.0: 4, 178.0: 4, 246.0: 4, 456.0: 4, 74.0: 3, 70.0: 3, 50.0: 3, 44.0: 3, 272.
0: 3, 226.0: 3, 104.0: 3, 116.0: 3, 90.0: 3, 145.0: 3, 148.0: 3, 42.0: 3, 110.0: 3, 128.0: 3,
85.0: 3, 136.0: 3, 420.0: 3, 186.0: 3, 256.0: 3, 166.0: 3, 350.0: 3, 130.0: 3, 99.0: 3, 176.0:
3, 174.0: 3, 208.0: 3, 312.0: 3, 169.0: 3, 240.0: 3, 82.0: 3, 289.0: 2, 232.0: 2, 95.0: 2, 18.
0: 2, 338.0: 2, 298.0: 2, 236.0: 2, 206.0: 2, 30.0: 2, 125.0: 2, 113.0: 2, 144.0: 2, 305.0: 2,
423.0: 2, 362.0: 2, 147.0: 2, 328.0: 2, 304.0: 2, 164.0: 2, 424.0: 2, 513.0: 2, 243.0: 2, 233.
0: 2, 192.0: 2, 351.0: 2, 660.0: 2, 335.0: 2, 158.0: 2, 140.0: 2, 245.0: 2, 250.0: 2, 182.0:
2, 189.0: 2, 45.0: 2, 54.0: 2, 117.0: 2, 60.0: 2, 157.0: 2, 92.0: 2, 94.0: 2, 285.0: 2, 1.0:
2, 75.0: 2, 105.0: 2, 215.0: 2, 425.0: 2, 149.0: 2, 442.0: 2, 172.0: 2, 344.0: 2, 472.0: 2, 6
6.0: 2, 284.0: 2, 203.0: 2, 153.0: 2, 260.0: 2, 68.0: 2, 205.0: 2, 98.0: 2, 318.0: 2, 480.0:
2, 112.0: 2, 184.0: 2, 135.0: 2, 302.0: 2, 266.0: 2, 171.0: 2, 650.0: 2, 212.0: 2, 306.0: 2, 1
62.0: 2, 143.0: 2, 101.0: 2, 238.0: 2, 281.0: 2, 368.0: 1, 137.0: 1, 194.0: 1, 673.0: 1, 115.
0: 1, 579.0: 1, 65.0: 1, 705.0: 1, 223.0: 1, 387.0: 1, 438.0: 1, 894.0: 1, 38.0: 1, 426.0: 1,
816.0: 1, 11.0: 1, 237.0: 1, 310.0: 1, 375.0: 1, 294.0: 1, 448.0: 1, 564.0: 1, 366.0: 1, 860.
0: 1, 123.0: 1, 244.0: 1, 96.0: 1, 1047.0: 1, 408.0: 1, 603.0: 1, 731.0: 1, 337.0: 1, 274.0:
1, 63.0: 1, 81.0: 1, 32.0: 1, 151.0: 1, 975.0: 1, 450.0: 1, 230.0: 1, 571.0: 1, 24.0: 1, 766.
0: 1, 53.0: 1, 554.0: 1, 234.0: 1, 51.0: 1, 14.0: 1, 324.0: 1, 295.0: 1, 396.0: 1, 67.0: 1, 15
4.0: 1, 202.0: 1, 163.0: 1, 218.0: 1, 410.0: 1, 796.0: 1, 428.0: 1, 122.0: 1, 165.0: 1, 1378.
0: 1, 88.0: 1, 228.0: 1, 760.0: 1, 391.0: 1, 27.0: 1, 361.0: 1, 632.0: 1, 86.0: 1, 342.0: 1, 7
88.0: 1, 621.0: 1, 451.0: 1, 541.0: 1, 359.0: 1, 567.0: 1, 114.0: 1, 415.0: 1, 333.0: 1, 651.
0: 1, 481.0: 1, 64.0: 1, 922.0: 1, 142.0: 1, 290.0: 1, 127.0: 1, 506.0: 1, 297.0: 1, 604.0: 1,
254.0: 1, 36.0: 1, 102.0: 1, 399.0: 1, 41.0: 1, 46.0: 1, 348.0: 1, 315.0: 1, 299.0: 1, 31.0:
1, 34.0: 1, 1600.0: 1, 365.0: 1, 56.0: 1, 150.0: 1, 278.0: 1, 262.0: 1, 138.0: 1, 275.0: 1, 74
8.0: 1, 286.0: 1, 380.0: 1, 640.0: 1, 412.0: 1, 1031.0: 1, 573.0: 1, 287.0: 1, 167.0: 1, 1115.
0: 1, 576.0: 1, 443.0: 1, 468.0: 1, 22.0: 1, 48.0: 1, 28.0: 1, 600.0: 1, 768.0: 1, 1129.0: 1,
309.0: 1, 436.0: 1, 664.0: 1, 653.0: 1, 491.0: 1, 225.0: 1, 370.0: 1, 388.0: 1, 188.0: 1, 584.
0: 1, 292.0: 1, 207.0: 1, 97.0: 1, 459.0: 1, 280.0: 1, 204.0: 1, 156.0: 1, 452.0: 1, 261.0: 1,
259.0: 1, 209.0: 1, 263.0: 1, 381.0: 1, 528.0: 1, 258.0: 1, 464.0: 1, 57.0: 1, 1170.0: 1, 293.
0: 1, 630.0: 1, 466.0: 1, 109.0: 1, 479.0: 1, 219.0: 1, 175.0: 1, 594.0: 1, 296.0: 1, 146.0:
1, 616.0: 1, 870.0: 1, 530.0: 1, 500.0: 1, 510.0: 1, 247.0: 1, 255.0: 1, 432.0: 1, 126.0: 1, 4
73.0: 1, 376.0: 1, 161.0: 1, 224.0: 1, 248.0: 1, 772.0: 1, 435.0: 1, 378.0: 1, 562.0: 1, 89.0:
1, 921.0: 1, 762.0: 1, 119.0: 1}

Feature: Electrical
{'SBrkr': 1334, 'FuseA': 94, 'FuseF': 27, 'FuseP': 3, 'Mix': 1}

We delete all features with more than 90% missing values. And, for other features we assume that rows with missing values represent features that are absent from the respective properties, we replace all the missing values with 0.

In [13]:
```
# Define the missing values threshold
missing_values_threshold = 90

for column, missing_percentage in sorted_missing_percentage:
    # If missing percentage > missing_values_threshold, drop the column
    if missing_percentage > missing_values_threshold:
        df.drop(column, axis=1, inplace=True)
    # Else, if the percentage of missing values <= missing_values_threshold, replace missing
    else:
        df[column].fillna(0, inplace=True)
```

In [14]:
```
# Identify columns with missing values
columns_with_missing_values = df.columns[df.isnull().any()].tolist()

print(f"Number of columns with missing values: {len(columns_with_missing_values)}")
print("Names of Columns with Missing Data:\n", columns_with_missing_values)
```

```
Number of columns with missing values: 0
Names of Columns with Missing Data:
 []
```

Identifying Categorical and Numerical Features

In [15]:
```
# Assuming all the columns with less than 8 unique values are categorical
threshold = 8  # Threshold
categorical_features = []

for column in df.columns:
    if df[column].nunique() <= threshold :
        categorical_features.append(column)

print(f"Number of Categorical Features: {len(categorical_features)}")
print("Categorical Features:", categorical_features)

# Identify Numerical Features
numerical_features = df.select_dtypes(include=['int64', 'float64']).columns.tolist()
numerical_features = [x for x in numerical_features if x not in categorical_features]

print(f"Number of Numerical Features: {len(numerical_features)}")
print("Numerical Features:", numerical_features)
```

```
Number of Categorical Features: 45
Categorical Features: ['MSZoning', 'Street', 'LotShape', 'LandContour', 'Utilities', 'LotConfi
g', 'LandSlope', 'Condition2', 'BldgType', 'HouseStyle', 'RoofStyle', 'RoofMatl', 'MasVnrTyp
e', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinTy
pe1', 'BsmtFinType2', 'Heating', 'HeatingQC', 'CentralAir', 'Electrical', 'BsmtFullBath', 'Bsm
tHalfBath', 'FullBath', 'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'KitchenQual', 'Functiona
l', 'Fireplaces', 'FireplaceQu', 'GarageType', 'GarageFinish', 'GarageCars', 'GarageQual', 'Ga
rageCond', 'PavedDrive', 'PoolArea', 'Fence', 'YrSold', 'SaleCondition']
Number of Numerical Features: 27
Numerical Features: ['MSSubClass', 'LotFrontage', 'LotArea', 'OverallQual', 'OverallCond', 'Ye
arBuilt', 'YearRemodAdd', 'MasVnrArea', 'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtS
F', '1stFlrSF', '2ndFlrSF', 'LowQualFinSF', 'GrLivArea', 'TotRmsAbvGrd', 'GarageYrBlt', 'Garag
eArea', 'WoodDeckSF', 'OpenPorchSF', 'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'MiscVal',
'MoSold', 'SalePrice']
```

In [16]:

```
# Identifying the categorical variables where the count of records within a category is below

threshold_percent = 1/100.0
columns_with_small_categories = []
columns_with_half_or_less_categories = []

for col in categorical_features:
    value_counts_proportions = df[col].value_counts(normalize=True)
    value_counts_numbers = df[col].value_counts()

    small_categories_proportions = value_counts_proportions[value_counts_proportions < thresh

    num_of_total_categories = df[col].nunique()
    num_of_small_categories = len(small_categories_proportions)

    num_and_names_of_small_categories = [(cat, value_counts_numbers[cat]) for cat in small_ca

    if num_of_small_categories > 0:
        columns_with_small_categories.append(col)
        print(f"For '{col}', the total number of categories is {num_of_total_categories}, and

    # Check if the number of "small" categories is more than half the total number of categor
    if num_of_small_categories >= num_of_total_categories / 2:
        columns_with_half_or_less_categories.append(col)

print(f"The columns that contain categories with less than 1% of values are: {columns_with_sm
print()
print(f"The columns for which more than half of the categories each have less than 1% values
```

```
For 'MSZoning', the total number of categories is 5, and the categories with less than 1% valu
es are: [('C (all)', 10)]
For 'Street', the total number of categories is 2, and the categories with less than 1% values
are: [('Grvl', 6)]
For 'LotShape', the total number of categories is 4, and the categories with less than 1% valu
es are: [('IR3', 10)]
For 'Utilities', the total number of categories is 2, and the categories with less than 1% val
ues are: [('NoSeWa', 1)]
For 'LotConfig', the total number of categories is 5, and the categories with less than 1% val
ues are: [('FR3', 4)]
For 'LandSlope', the total number of categories is 3, and the categories with less than 1% val
ues are: [('Sev', 13)]
For 'Condition2', the total number of categories is 8, and the categories with less than 1% va
lues are: [('Feedr', 6), ('Artery', 2), ('RRNn', 2), ('PosN', 2), ('PosA', 1), ('RRAn', 1),
('RRAe', 1)]
For 'HouseStyle', the total number of categories is 8, and the categories with less than 1% va
lues are: [('1.5Unf', 14), ('2.5Unf', 11), ('2.5Fin', 8)]
For 'RoofStyle', the total number of categories is 6, and the categories with less than 1% val
ues are: [('Flat', 13), ('Gambrel', 11), ('Mansard', 7), ('Shed', 2)]
For 'RoofMatl', the total number of categories is 8, and the categories with less than 1% valu
es are: [('Tar&Grv', 11), ('WdShngl', 6), ('WdShake', 5), ('Metal', 1), ('Membran', 1), ('Rol
l', 1), ('ClyTile', 1)]
For 'ExterQual', the total number of categories is 4, and the categories with less than 1% val
ues are: [('Fa', 14)]
For 'ExterCond', the total number of categories is 5, and the categories with less than 1% val
ues are: [('Ex', 3), ('Po', 1)]
For 'Foundation', the total number of categories is 6, and the categories with less than 1% va
lues are: [('Stone', 6), ('Wood', 3)]
For 'BsmtCond', the total number of categories is 5, and the categories with less than 1% valu
es are: [('Po', 2)]
For 'BsmtFinType2', the total number of categories is 7, and the categories with less than 1%
values are: [('GLQ', 14)]
For 'Heating', the total number of categories is 6, and the categories with less than 1% value
s are: [('Grav', 7), ('Wall', 4), ('OthW', 2), ('Floor', 1)]
For 'HeatingQC', the total number of categories is 5, and the categories with less than 1% val
ues are: [('Po', 1)]
For 'Electrical', the total number of categories is 6, and the categories with less than 1% va
lues are: [('FuseP', 3), ('Mix', 1), (0, 1)]
For 'BsmtFullBath', the total number of categories is 4, and the categories with less than 1%
values are: [(3, 1)]
For 'BsmtHalfBath', the total number of categories is 3, and the categories with less than 1%
values are: [(2, 2)]
For 'FullBath', the total number of categories is 4, and the categories with less than 1% valu
es are: [(0, 9)]
For 'HalfBath', the total number of categories is 3, and the categories with less than 1% valu
es are: [(2, 12)]
For 'BedroomAbvGr', the total number of categories is 8, and the categories with less than 1%
values are: [(6, 7), (0, 6), (8, 1)]
For 'KitchenAbvGr', the total number of categories is 4, and the categories with less than 1%
values are: [(3, 2), (0, 1)]
For 'Functional', the total number of categories is 7, and the categories with less than 1% va
lues are: [('Maj1', 14), ('Maj2', 5), ('Sev', 1)]
For 'Fireplaces', the total number of categories is 4, and the categories with less than 1% va
lues are: [(3, 5)]
For 'GarageType', the total number of categories is 7, and the categories with less than 1% va
lues are: [('CarPort', 9), ('2Types', 6)]
For 'GarageCars', the total number of categories is 5, and the categories with less than 1% va
lues are: [(4, 5)]
For 'GarageQual', the total number of categories is 6, and the categories with less than 1% va
lues are: [('Gd', 14), ('Ex', 3), ('Po', 3)]
For 'GarageCond', the total number of categories is 6, and the categories with less than 1% va
lues are: [('Gd', 9), ('Po', 7), ('Ex', 2)]
For 'PoolArea', the total number of categories is 8, and the categories with less than 1% valu
es are: [(512, 1), (648, 1), (576, 1), (555, 1), (480, 1), (519, 1), (738, 1)]
For 'Fence', the total number of categories is 5, and the categories with less than 1% values
are: [('MnWw', 11)]
```

```
For 'SaleCondition', the total number of categories is 6, and the categories with less than 1% values are: [('Alloca', 12), ('AdjLand', 4)]
The columns that contain categories with less than 1% of values are: ['MSZoning', 'Street', 'LotShape', 'Utilities', 'LotConfig', 'LandSlope', 'Condition2', 'HouseStyle', 'RoofStyle', 'RoofMatl', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtCond', 'BsmtFinType2', 'Heating', 'HeatingQC', 'Electrical', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath', 'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'Functional', 'Fireplaces', 'GarageType', 'GarageCars', 'GarageQual', 'GarageCond', 'PoolArea', 'Fence', 'SaleCondition']

The columns for which more than half of the categories each have less than 1% values are: ['Street', 'Utilities', 'Condition2', 'RoofStyle', 'RoofMatl', 'Heating', 'Electrical', 'KitchenAbvGr', 'GarageQual', 'GarageCond', 'PoolArea']
```

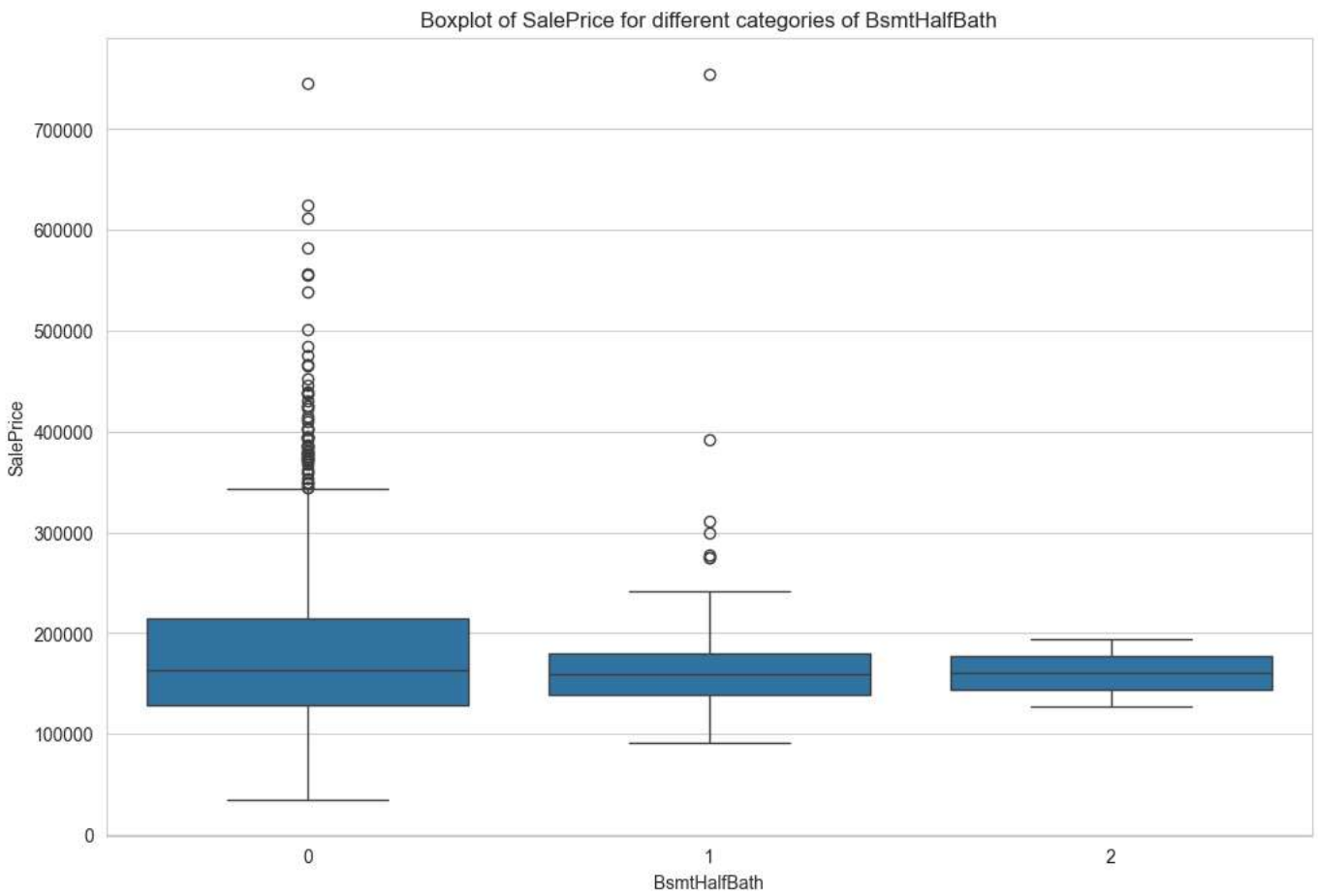
In [18]:
```python
# Dropping the categorical variables where the count of records within a category is below 1%
categorical_features = [x for x in categorical_features if x not in columns_with_half_or_less
print(f'Number of Categorical Features: {len(categorical_features)}')
```

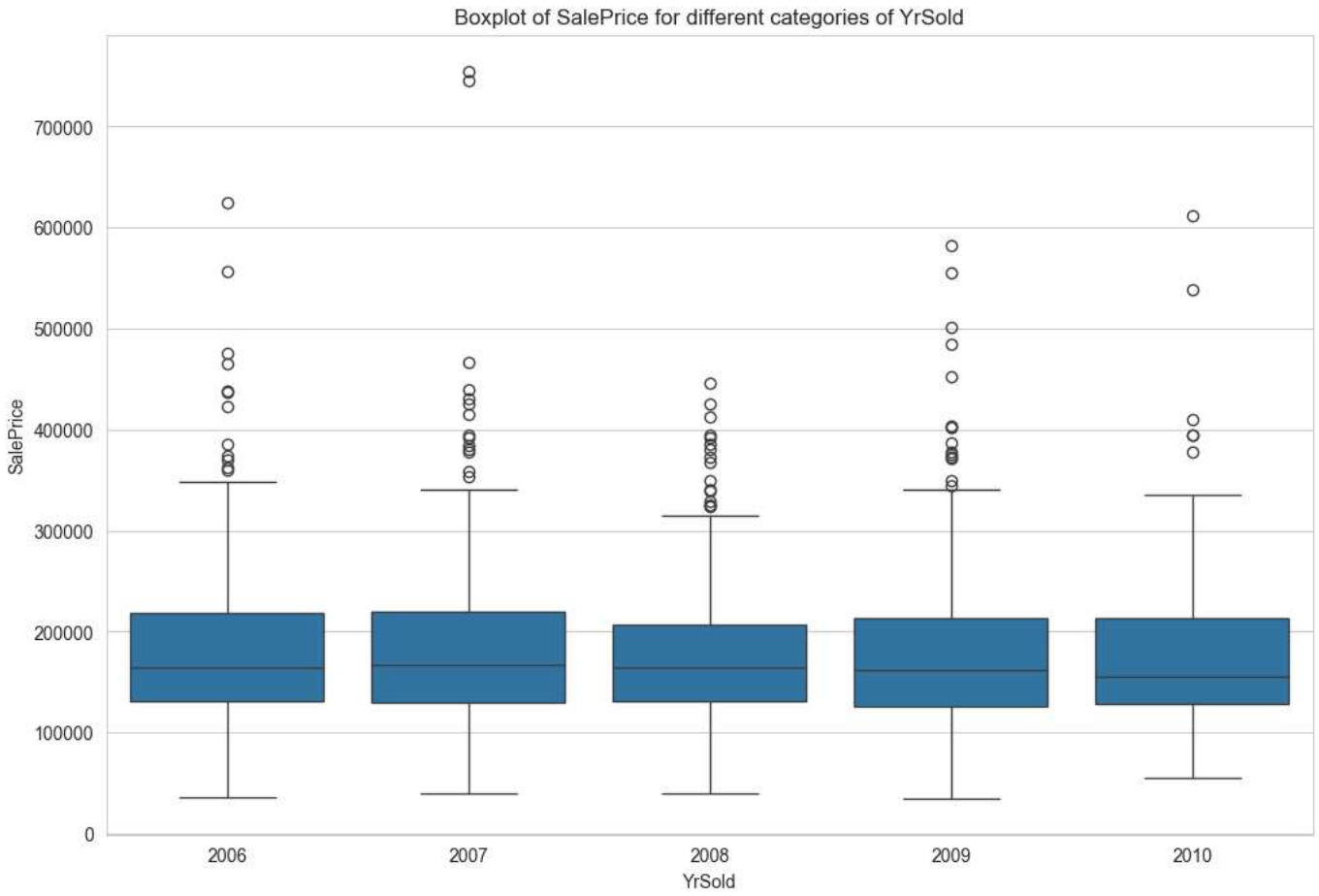
```
Number of Categorical Features: 34
```

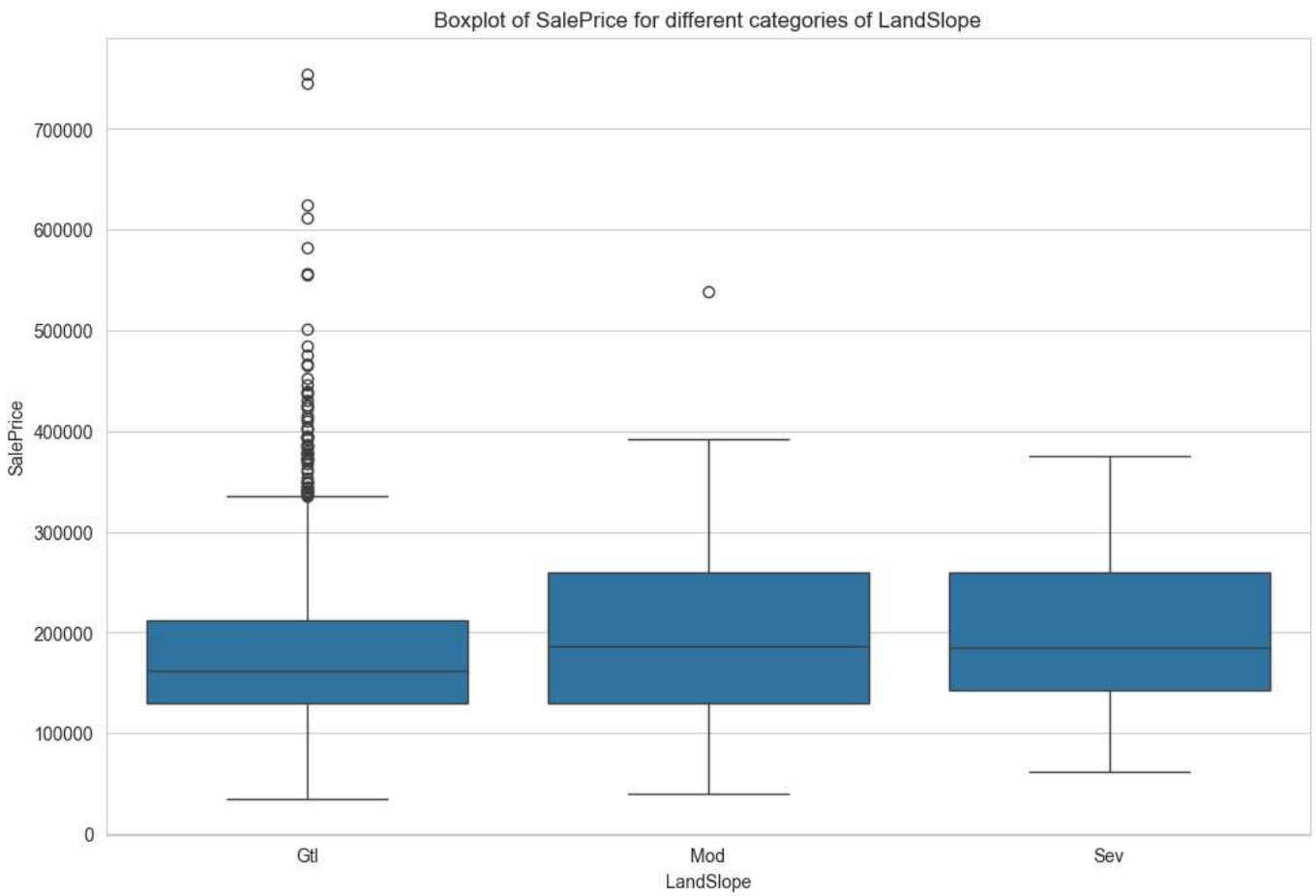
In [19]:
```python
# Identifying the top ten categorical features with the maximum impact on SalePrice
def anova(frame):
    anv = pd.DataFrame()
    anv['features'] = categorical_features
    pvals = []
    for c in categorical_features:
        samples = []
        for cls in frame[c].unique():
            s = frame[frame[c] == cls]['SalePrice'].values
            samples.append(s)
        pval = stats.f_oneway(*samples)[1]
        pvals.append(pval)
    anv['pval'] = pvals
    return anv.sort_values('pval', ascending=False)

k = anova(df)
top_ten = k.head(10)
print(top_ten)
```

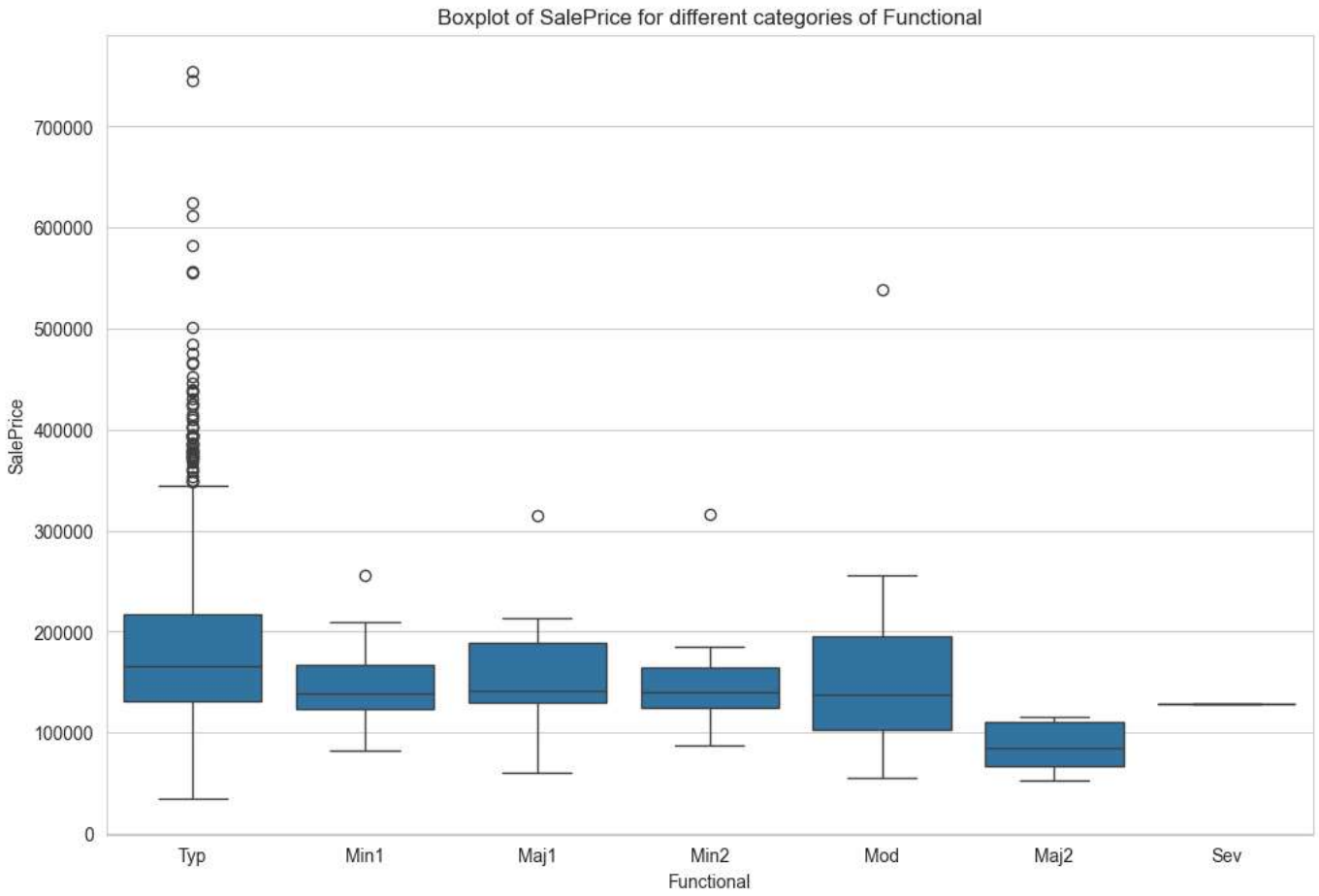
```
       features          pval
19  BsmtHalfBath  8.013743e-01
32        YrSold  6.300888e-01
4      LandSlope  1.413964e-01
24    Functional  4.841697e-04
3      LotConfig  3.163167e-06
9      ExterCond  5.106681e-07
15  BsmtFinType2  5.225649e-08
2    LandContour  2.742217e-08
5       BldgType  2.056736e-10
31         Fence  9.379977e-11
```

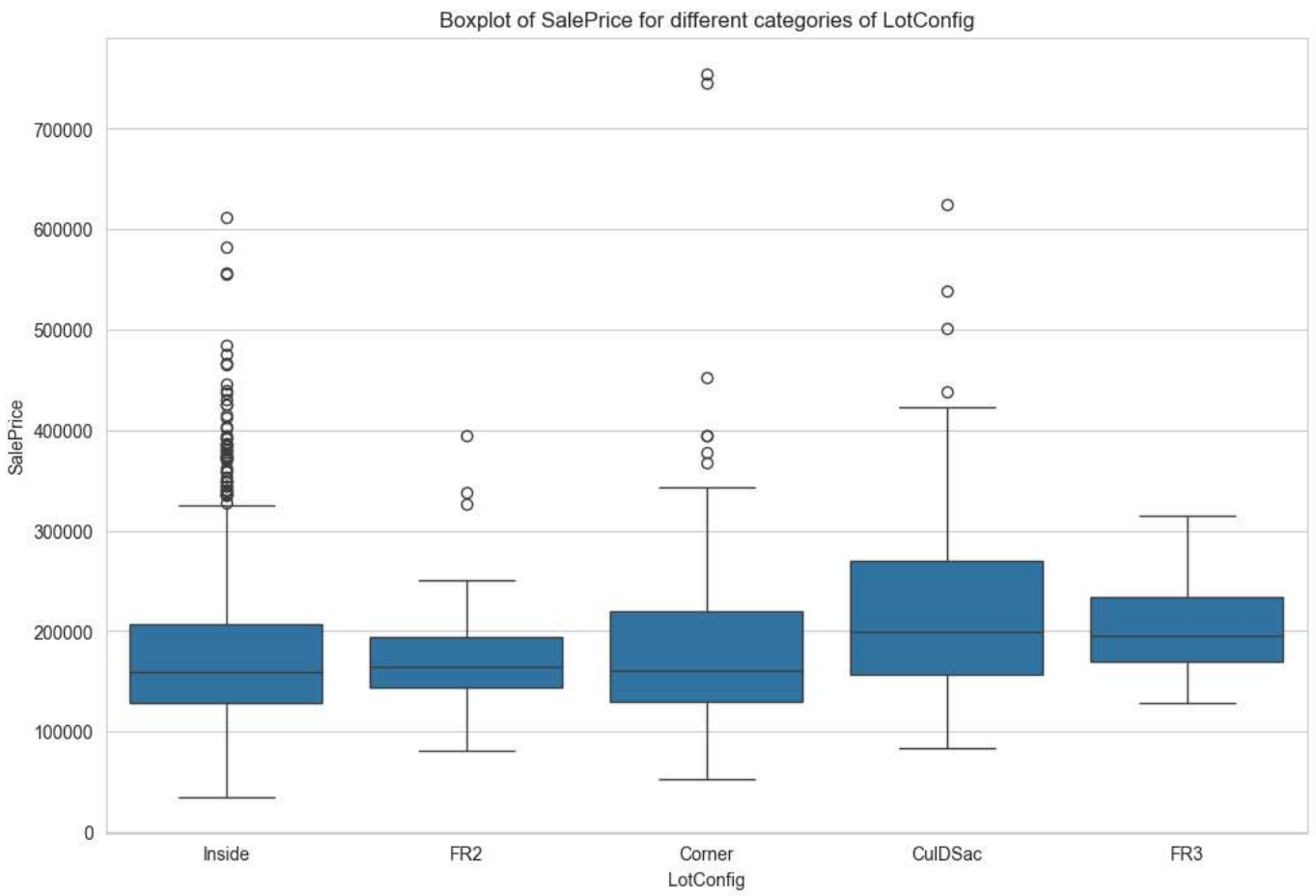
In [20]:
```python
# Exploring the relationship of SalePrice with the top five categorical features
top_5_categorical_features = k.head(5).features.tolist()

for feature in top_5_categorical_features:
    plt.figure(figsize=(12,8))
    sns.boxplot(x=feature, y='SalePrice', data=df)
    plt.title(f'Boxplot of SalePrice for different categories of {feature}')
    plt.show()
```


Boxplot of SalePrice for different categories of BsmtHalfBath
SalePrice
0
100000
200000
300000
400000
500000
600000
700000
0
1
2
BsmtHalfBath


Boxplot of SalePrice for different categories of YrSold
SalePrice
0
100000
200000
300000
400000
500000
600000
700000
2006
2007
2008
2009
2010
YrSold


Boxplot of SalePrice for different categories of LandSlope
SalePrice
0
100000
200000
300000
400000
500000
600000
700000
Gtl
Mod
Sev
LandSlope

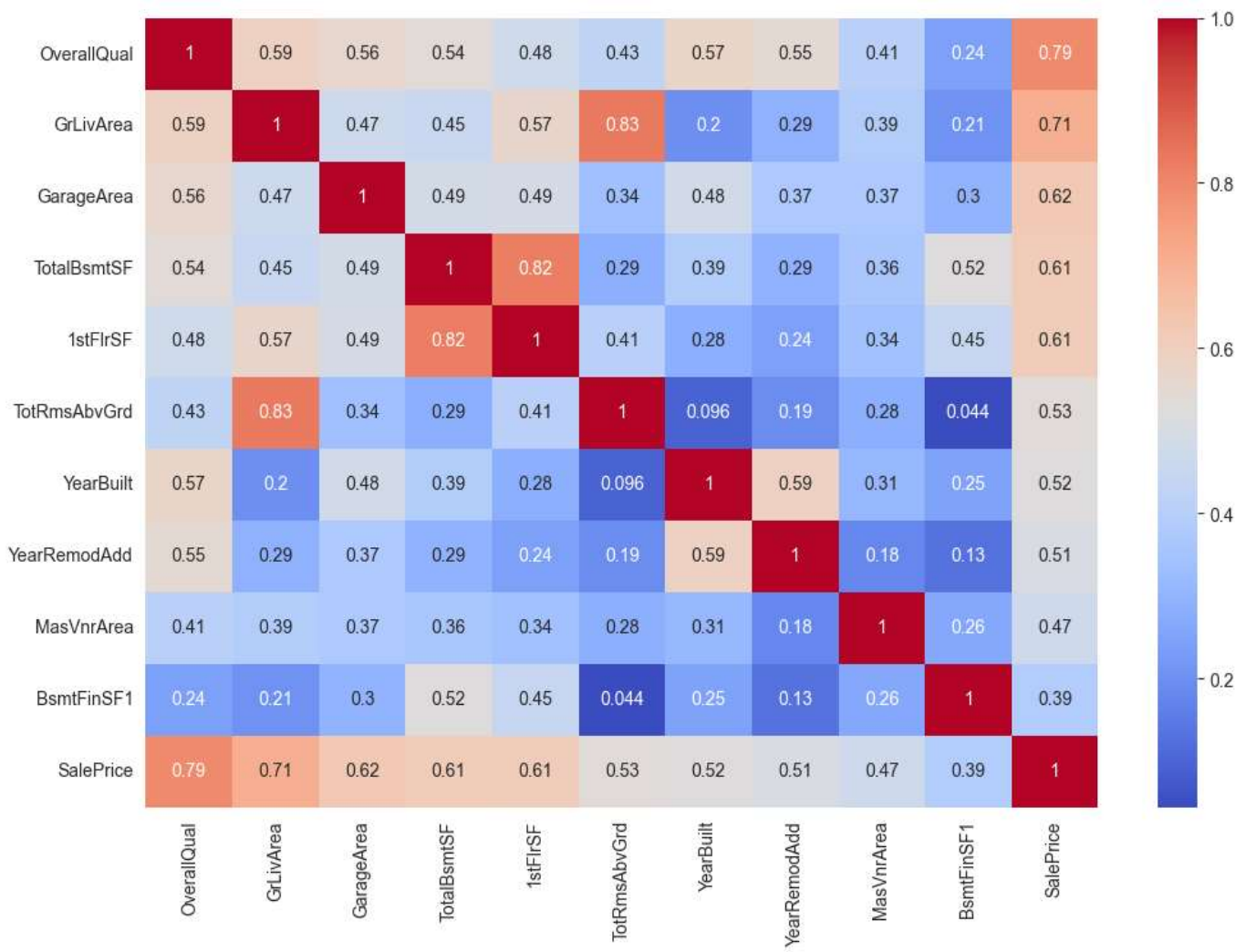

Boxplot of SalePrice for different categories of Functional
SalePrice
0
100000
200000
300000
400000
500000
600000
700000
Typ
Min1
Maj1
Min2
Mod
Maj2
Sev
Functional



In [21]:
```
# Identify Numerical Features
numerical_features = df.select_dtypes(include=['int64', 'float64']).columns.tolist()
numerical_features = [x for x in numerical_features if x not in categorical_features]
numerical_features.remove('SalePrice')
numerical_features.remove('MoSold')
categorical_features.append('MoSold')
print(f'Number of Numerical Features: {len(numerical_features)}')
print(f'Number of Categorical Features: {len(categorical_features)}')
```

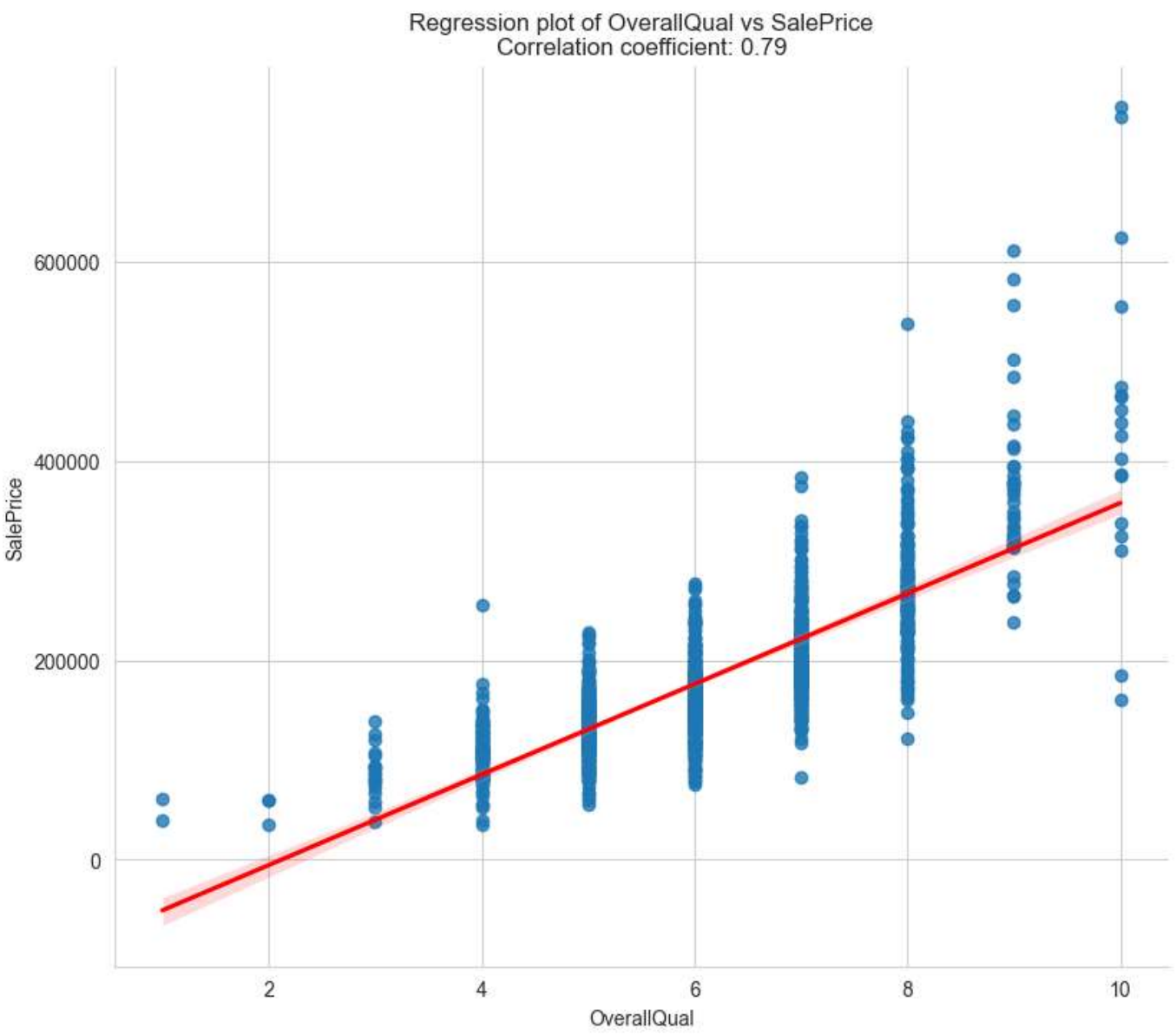
```
Number of Numerical Features: 27
Number of Categorical Features: 35
```

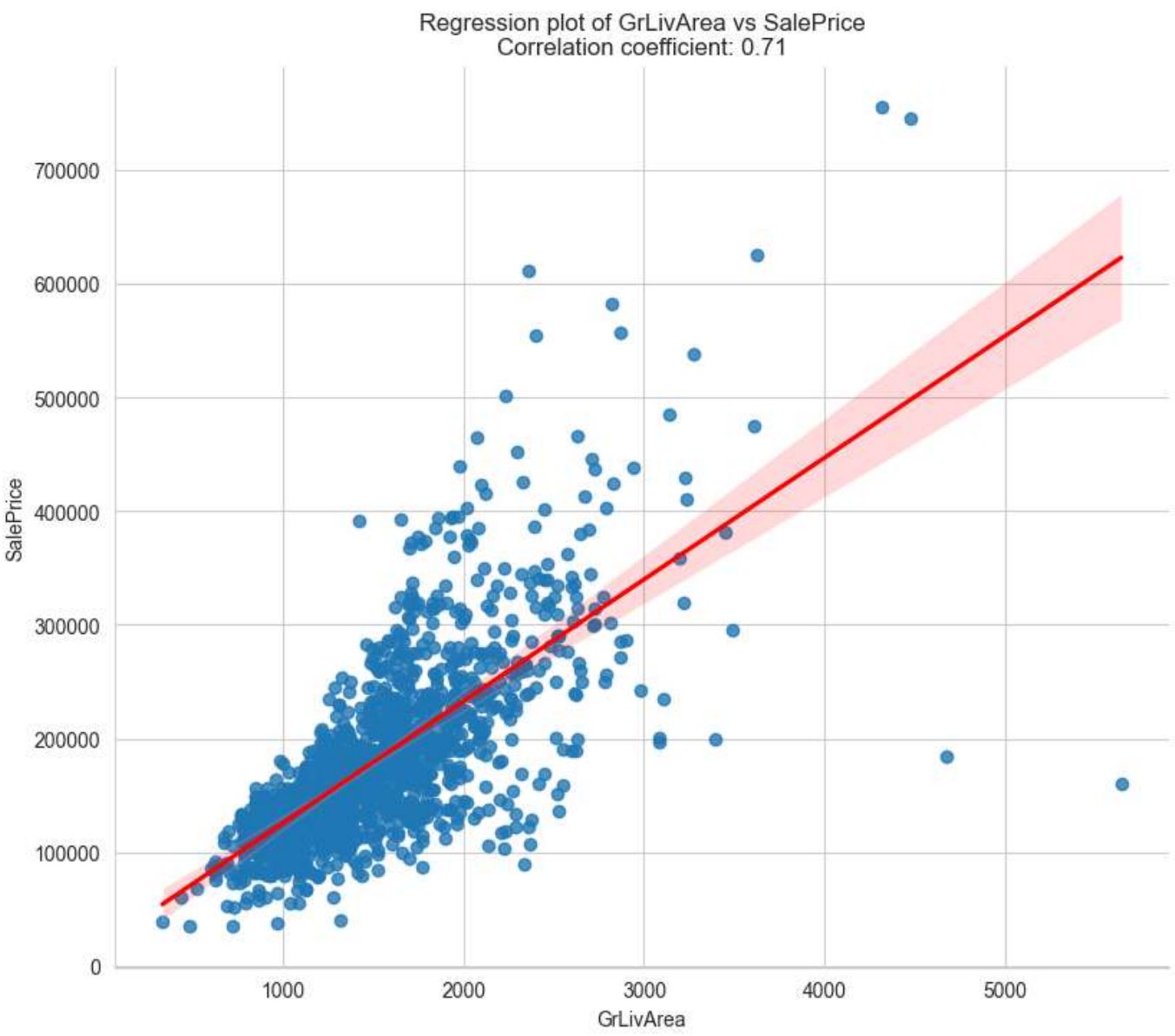
In [22]:
```
target_variable ='SalePrice'

# Compute correlations
correlations = df[numerical_features + ['SalePrice']].corr()[target_variable]

# Sort by absolute value and take top 10
top_10_numerical_features = correlations.abs().sort_values(ascending=False).head(11).index.to

# We remove 'SalePrice' from top_10_numerical_features since it will obviously be perfectly c
top_10_numerical_features.remove(target_variable)

# Calculate correlation of top 10 numerical features
top_corr_matrix = df[top_10_numerical_features + [target_variable]].corr()

# Use seaborn to generate a heatmap
plt.figure(figsize=(12,8))
sns.heatmap(top_corr_matrix, annot=True, cmap='coolwarm')

plt.show()
```



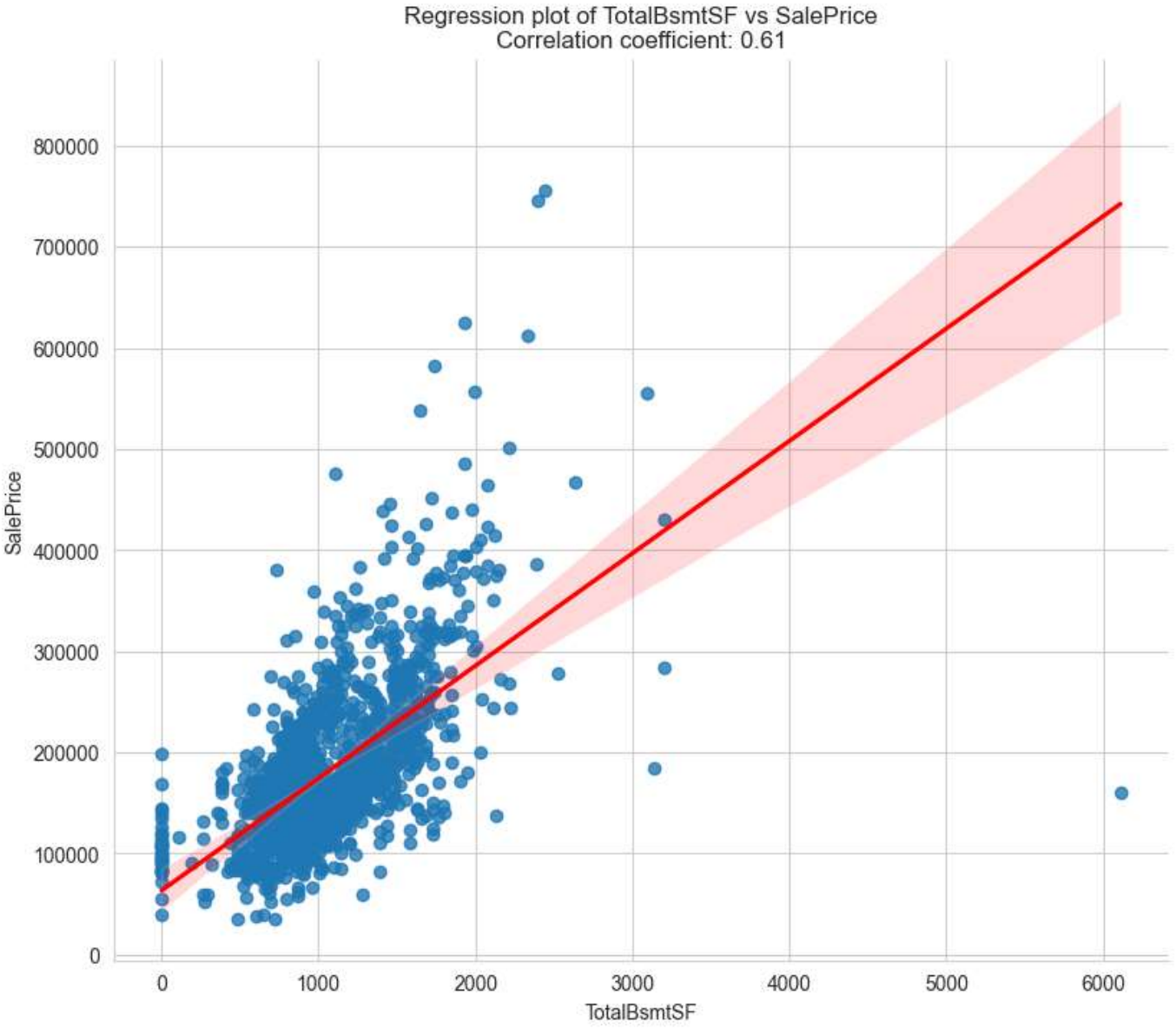
In [23]:

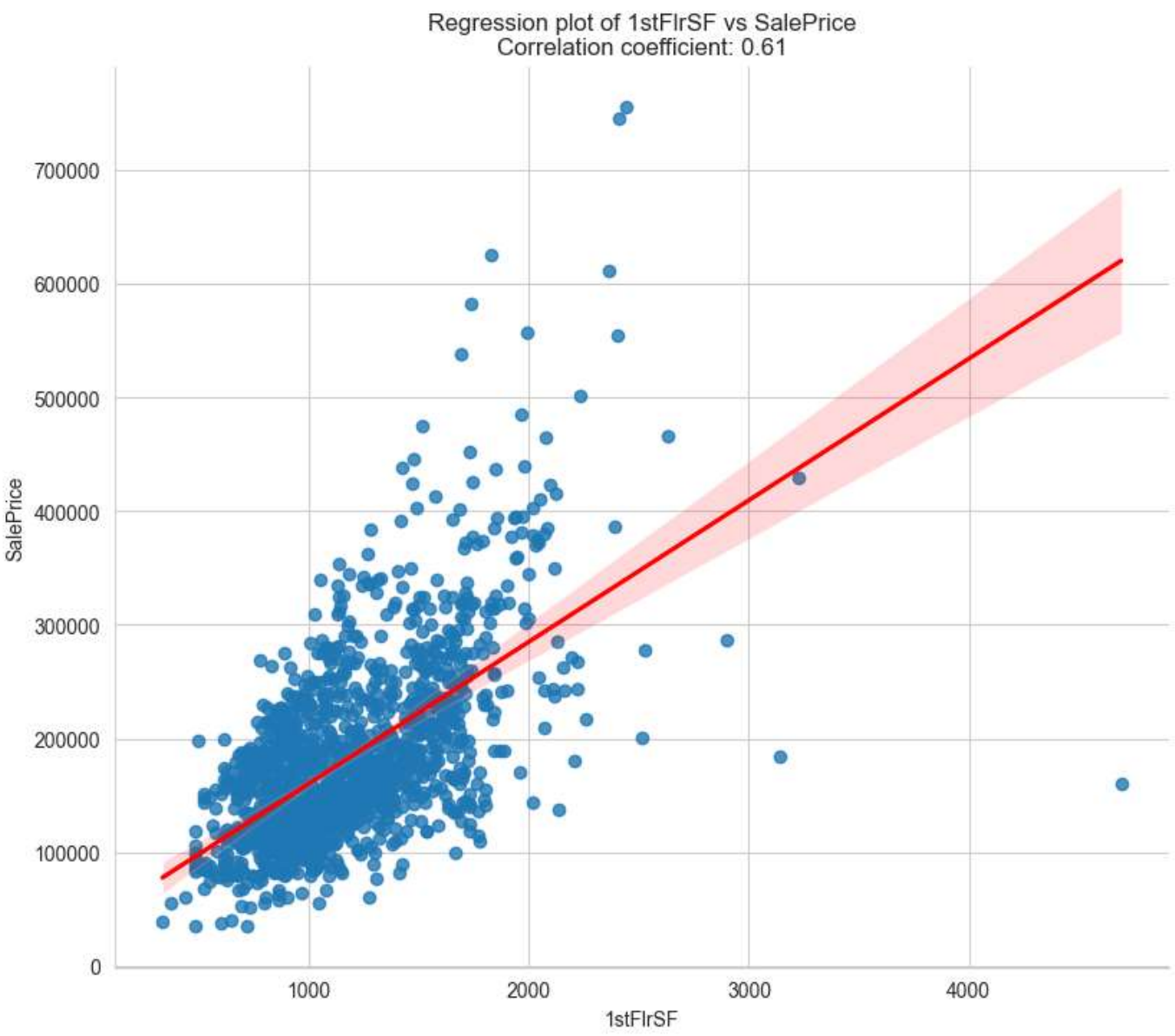
```
# Plot the Scatter Plots with SalePrice

# Sort by absolute value and take top 5
top_5_numerical_features = correlations.abs().sort_values(ascending=False).head(6).index.toli
top_5_numerical_features.remove(target_variable)

# Plot scatter plots with regression line
for feature in top_5_numerical_features:
    # Adjust the height and aspect parameters to your needs
    sns.lmplot(x=feature, y=target_variable, data=df, line_kws={'color': 'red'}, height=7, as
    plt.title(f'Regression plot of {feature} vs {target_variable}\n'
              f'Correlation coefficient: {pearsonr(df[feature], df[target_variable])[0]:.2f}'
    plt.show()
```


Regression plot of OverallQual vs SalePrice
Correlation coefficient: 0.79
SalePrice
600000
400000
200000
0
2
4
6
8
10
OverallQual


Regression plot of GrLivArea vs SalePrice
Correlation coefficient: 0.71
SalePrice
GrLivArea
0
100000
200000
300000
400000
500000
600000
700000
1000
2000
3000
4000
5000


Regression plot of GarageArea vs SalePrice
Correlation coefficient: 0.62
SalePrice
GarageArea
0
100000
200000
300000
400000
500000
600000
700000
0
200
400
600
800
1000
1200
1400


Regression plot of TotalBsmtSF vs SalePrice
Correlation coefficient: 0.61
SalePrice
TotalBsmtSF
0
100000
200000
300000
400000
500000
600000
700000
800000
0
1000
2000
3000
4000
5000
6000



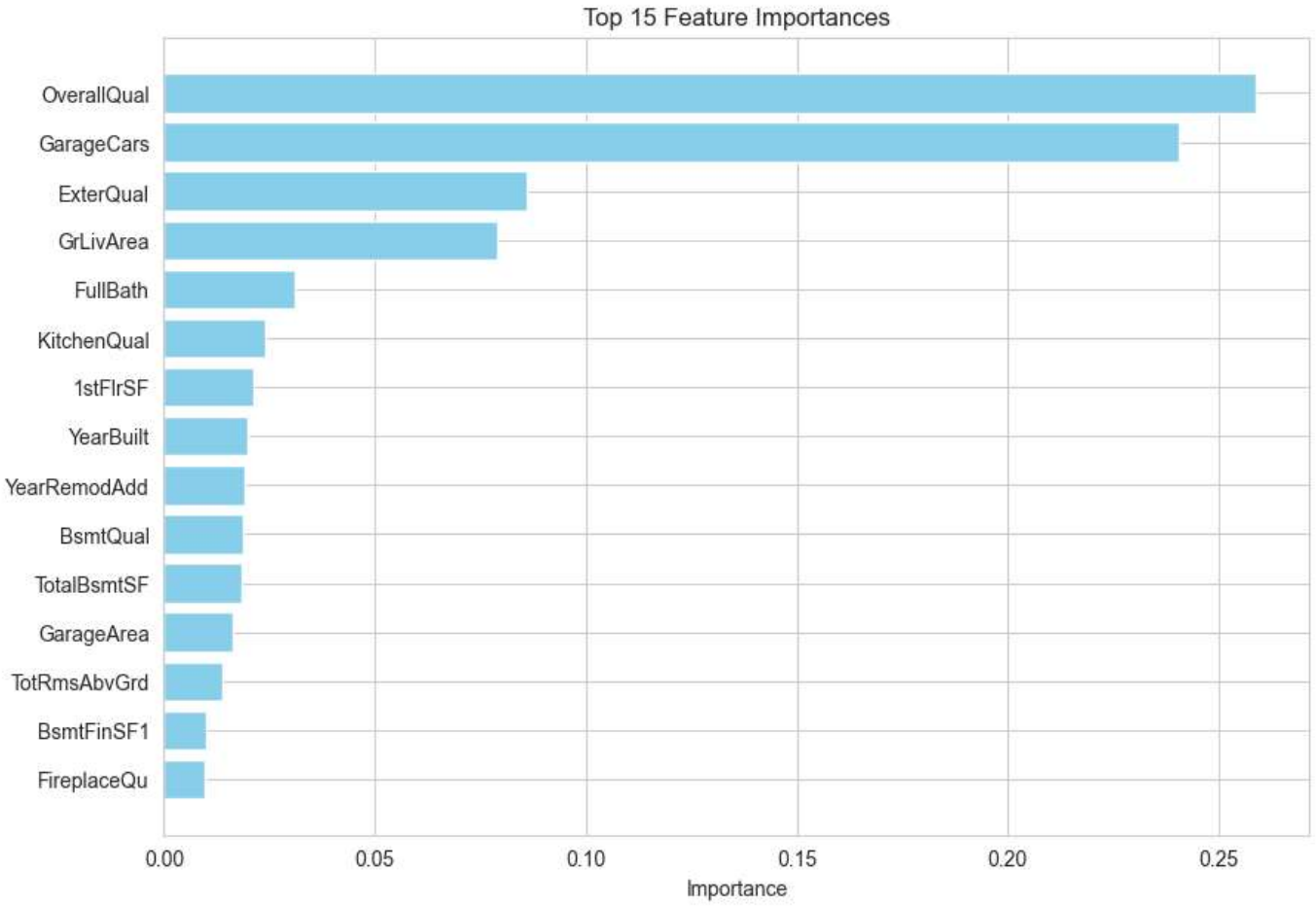
Identify Outliers

In [24]:
```python
# Set the threshold for the number of outliers
outlier_threshold = len(df) * 0.1  # 10% of the length of df

# Identify columns with more than 10% outliers
outlier_features = []
for column in numerical_features:
    q1 = df[column].quantile(0.25)
    q3 = df[column].quantile(0.75)
    iqr = q3 - q1
    lower_threshold = q1 - 1.5 * iqr
    upper_threshold = q3 + 1.5 * iqr
    outliers = df[(df[column] < lower_threshold) | (df[column] > upper_threshold)]
    outliers_count = outliers.shape[0]
    if outliers_count > outlier_threshold:
        outlier_features.append((column, outliers_count))

# Step 3: Print the columns with more than 10% outliers
print("Columns with more than 10% outliers:")
for feature, count in outlier_features:
    print(f"{feature} has {count} outliers.")
```

```
Columns with more than 10% outliers:
BsmtFinSF2 has 167 outliers.
EnclosedPorch has 208 outliers.
```

Of the 28 numerical features only two features have more than 10% outliers

In [25]:
```
# Apply the Random Forest algorithm to identify the most influential features, encompassing bo

# Copy the original DataFrame
df_encoded = df.copy()

# Label encoding for categorical features
le = LabelEncoder()
for feature in categorical_features+numerical_features:
    df_encoded[feature] = le.fit_transform(df_encoded[feature].astype(str))

# Define input and target variables
X = df_encoded[categorical_features + numerical_features]
y = df_encoded[target_variable]


# Fit Random Forest on your dataset
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X, y)

# get the feature importances
importances = rf.feature_importances_

# Create a DataFrame for visualization
feature_importances = pd.DataFrame({"Feature": X.columns, "Importance": importances})
top_15_features = feature_importances.sort_values(by="Importance", ascending=False).head(15)

# Plot
plt.figure(figsize=(10, 7))
plt.barh(top_15_features['Feature'], top_15_features['Importance'], color='skyblue')
plt.xlabel('Importance')
plt.title('Top 15 Feature Importances')
plt.gca().invert_yaxis()  # This line is to invert the y-axis to start with the highest value
plt.show()
```



In [26]:
```
print(numerical_features)
```

```
['MSSubClass', 'LotFrontage', 'LotArea', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemod
Add', 'MasVnrArea', 'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF', '2ndFl
rSF', 'LowQualFinSF', 'GrLivArea', 'KitchenAbvGr', 'TotRmsAbvGrd', 'GarageYrBlt', 'GarageAre
a', 'WoodDeckSF', 'OpenPorchSF', 'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'Mis
cVal']
```

In [27]:
```
print(categorical_features)
```

```
['MSZoning', 'LotShape', 'LandContour', 'LotConfig', 'LandSlope', 'BldgType', 'HouseStyle', 'M
asVnrType', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual', 'BsmtCond', 'BsmtExposure', 'B
smtFinType1', 'BsmtFinType2', 'HeatingQC', 'CentralAir', 'BsmtFullBath', 'BsmtHalfBath', 'Full
Bath', 'HalfBath', 'BedroomAbvGr', 'KitchenQual', 'Functional', 'Fireplaces', 'FireplaceQu',
'GarageType', 'GarageFinish', 'GarageCars', 'PavedDrive', 'Fence', 'YrSold', 'SaleCondition',
'MoSold']
```

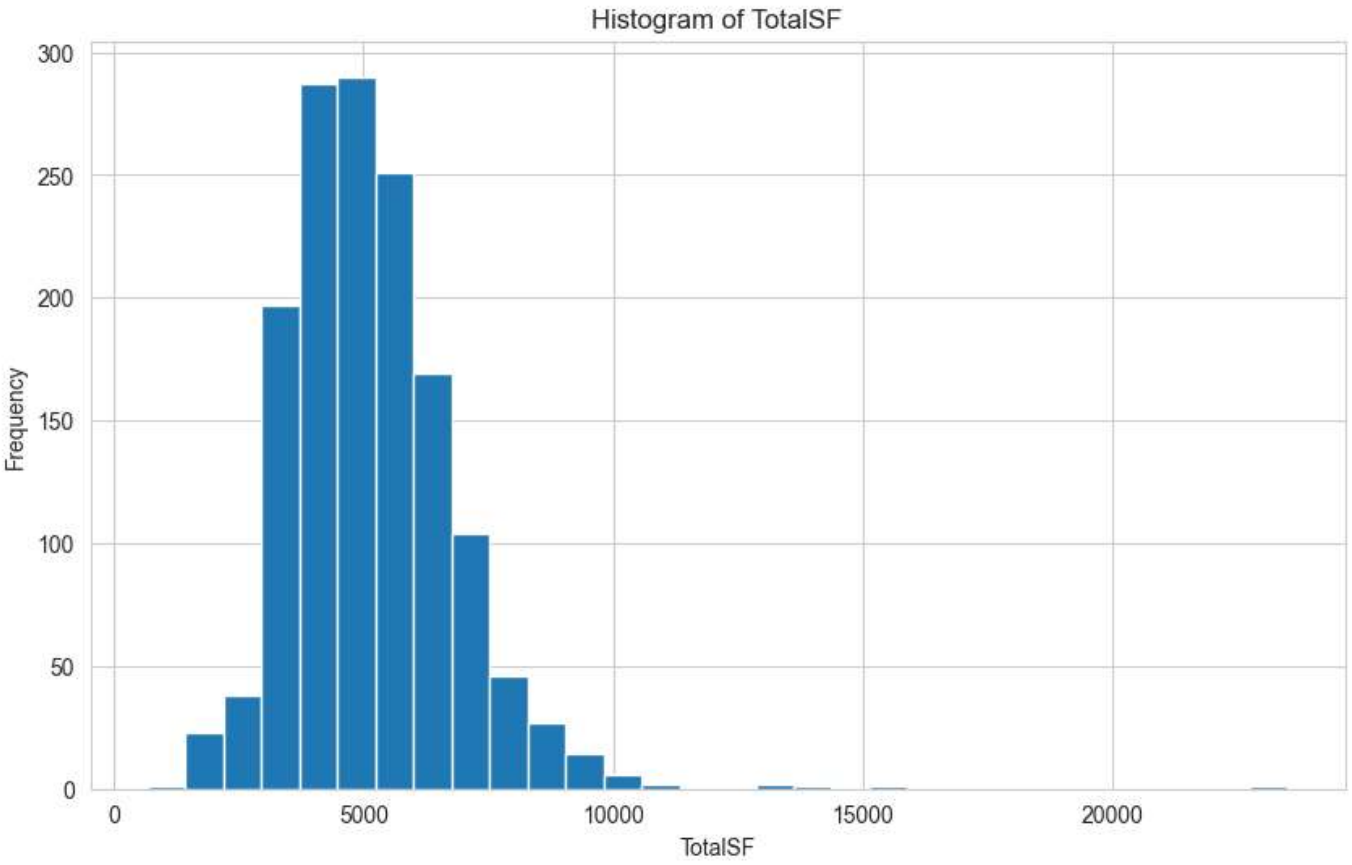
Consolidating the house size-related features into a single variable, we sum the following attributes: 'GrLivArea' (above-grade living area square feet), '1stFlrSF' (first floor square feet), '2ndFlrSF' (second floor square feet), 'BsmtFinSF1: Type 1 finished square feet', and 'LowQualFinSF' (low-quality finished square feet across all floors). This new aggregated variable represents the total living area of the house in square feet, capturing the comprehensive size across all floors and quality levels.

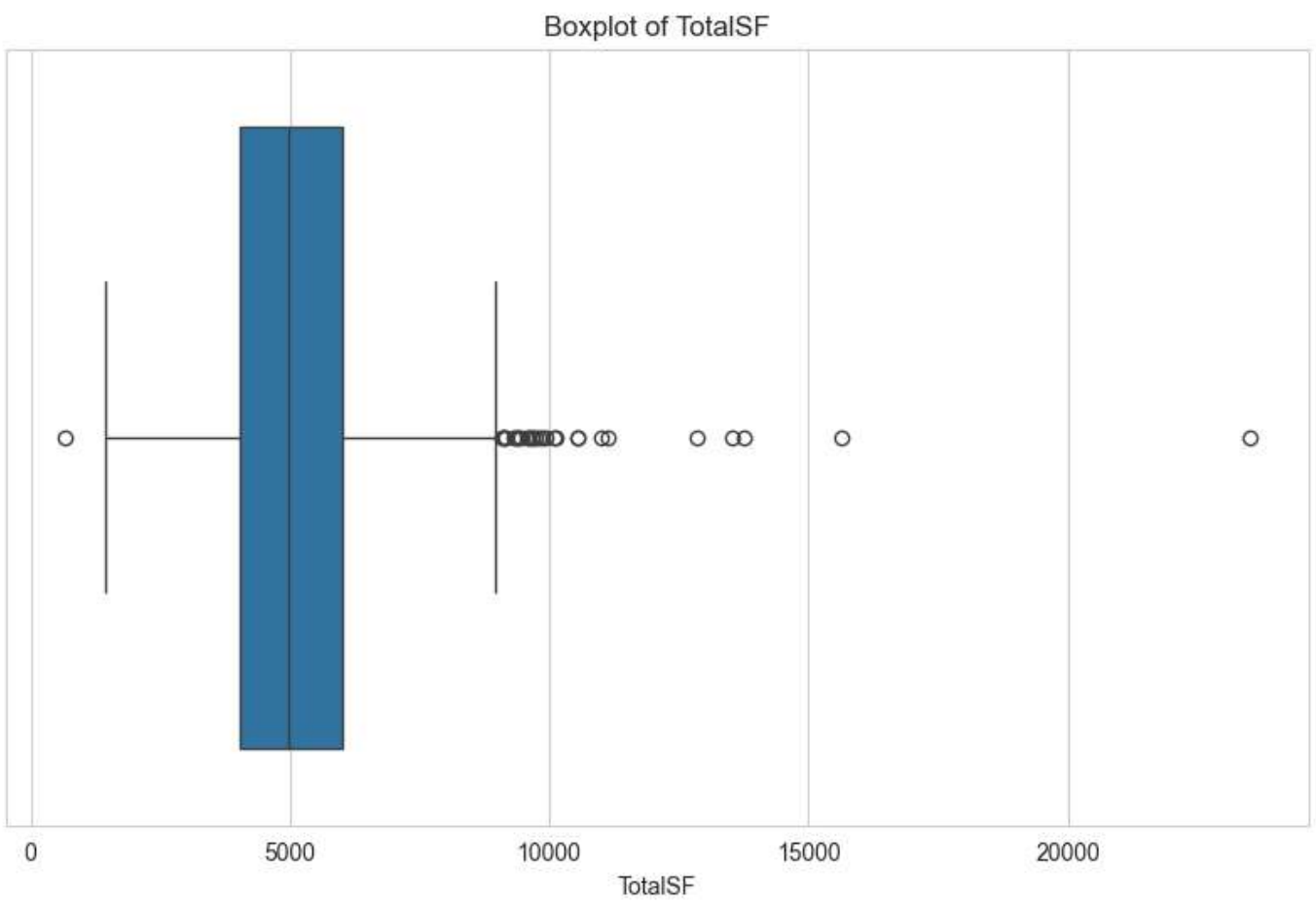
In [28]:
```
# Adding the variables
df['TotalSF'] = df['GrLivArea'] + df['1stFlrSF'] + df['2ndFlrSF'] + df['BsmtFinSF1'] + df['Lo
```

In [29]:
```
# Exploring TotalSF
print("Statistical Summary:", 'TotalSF')
print(df['TotalSF'].describe())

plt.figure(figsize=(10, 6))
df['TotalSF'].hist(bins=30)
plt.title(f'Histogram of {'TotalSF'}')
plt.xlabel('TotalSF')
plt.ylabel('Frequency')
plt.show()
```

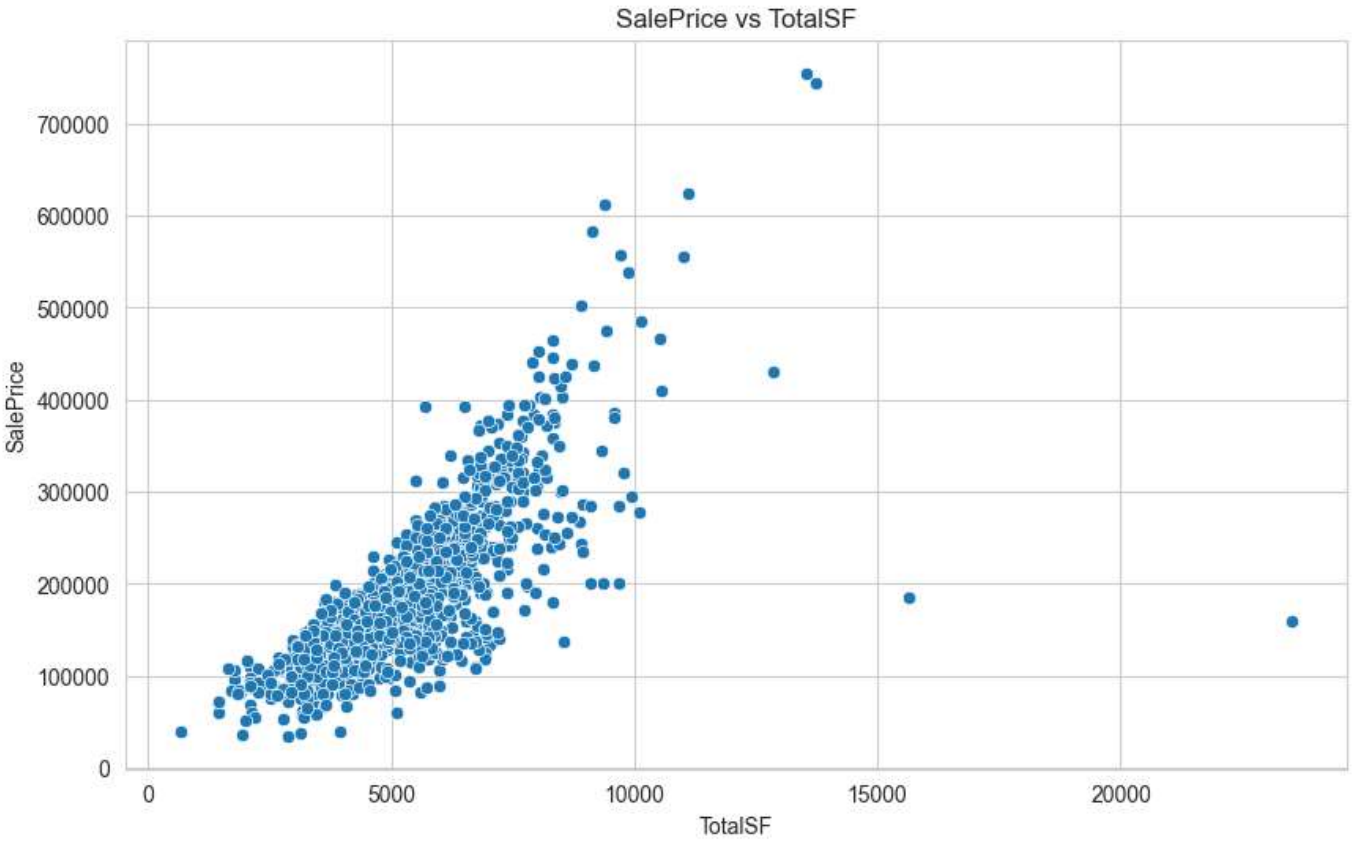
```
plt.figure(figsize=(10, 6))
sns.boxplot(data=df, x='TotalSF')
plt.title(f'Boxplot of {'TotalSF'}')
plt.show()
```

```
Statistical Summary: TotalSF
count     1460.000000
mean      5145.786301
std       1647.196985
min        668.000000
25%       4028.000000
50%       4958.000000
75%       6017.000000
max      23504.000000
Name: TotalSF, dtype: float64
```





In [30]:
```
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df, x=df['TotalSF'], y='SalePrice')
plt.title(f'SalePrice vs TotalSF')
plt.xlabel('TotalSF')
plt.ylabel('SalePrice')
plt.show()
```



In [31]:
```
# Creating the final list of features to be included in the model creation
features_to_delete = ['GrLivArea', '1stFlrSF', '2ndFlrSF', 'BsmtFinSF1', 'LowQualFinSF', 'Tot
numerical_features = [x for x in numerical_features if x not in features_to_delete]
numerical_features.append('TotalSF')
```

```
print(numerical_features)
print()
print(f"Number of Numerical features: {len(numerical_features)}")
```

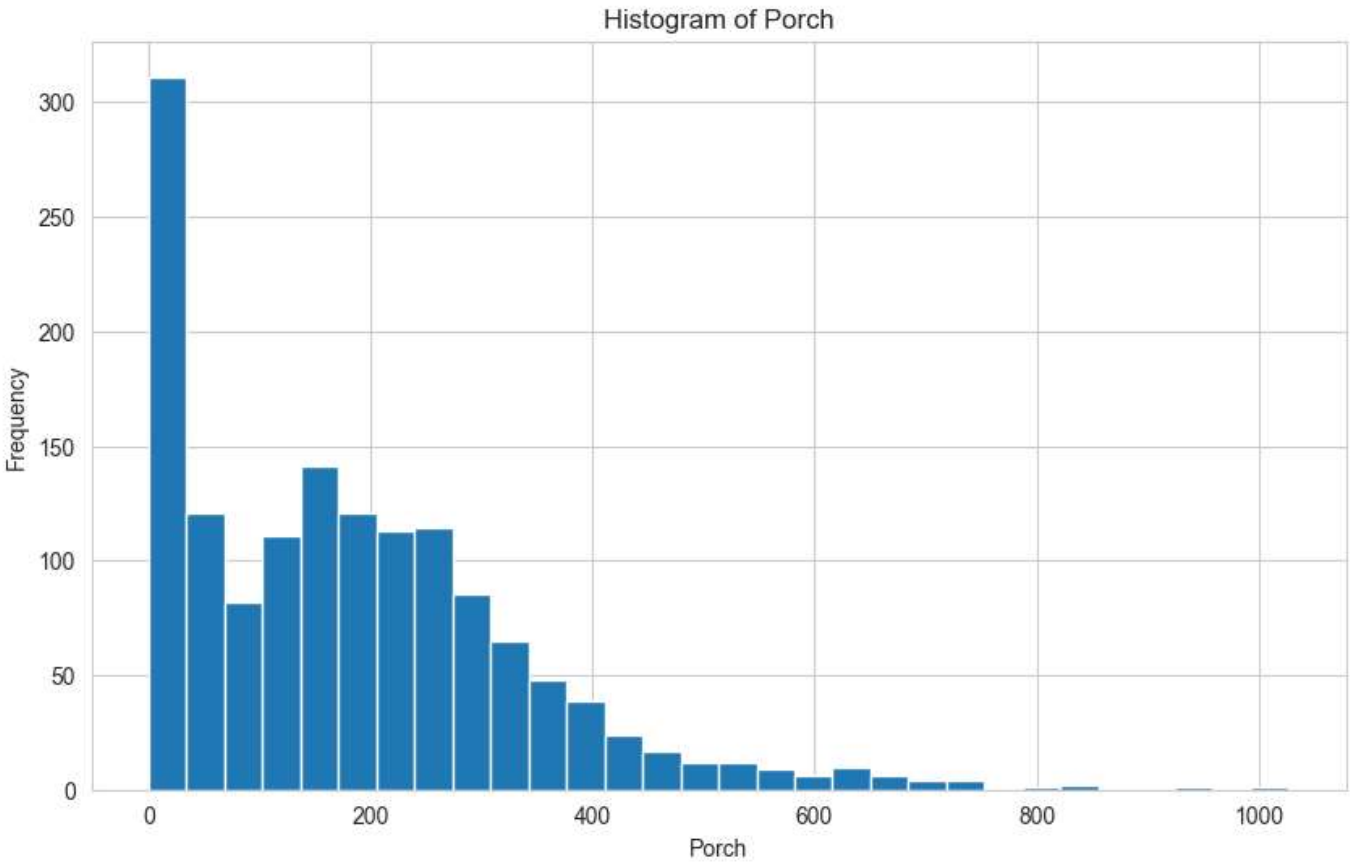
```
['MSSubClass', 'LotFrontage', 'LotArea', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemod
Add', 'MasVnrArea', 'KitchenAbvGr', 'TotRmsAbvGrd', 'GarageYrBlt', 'GarageArea', 'WoodDeckSF',
'OpenPorchSF', 'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'MiscVal', 'TotalSF']

Number of Numerical features: 20
```

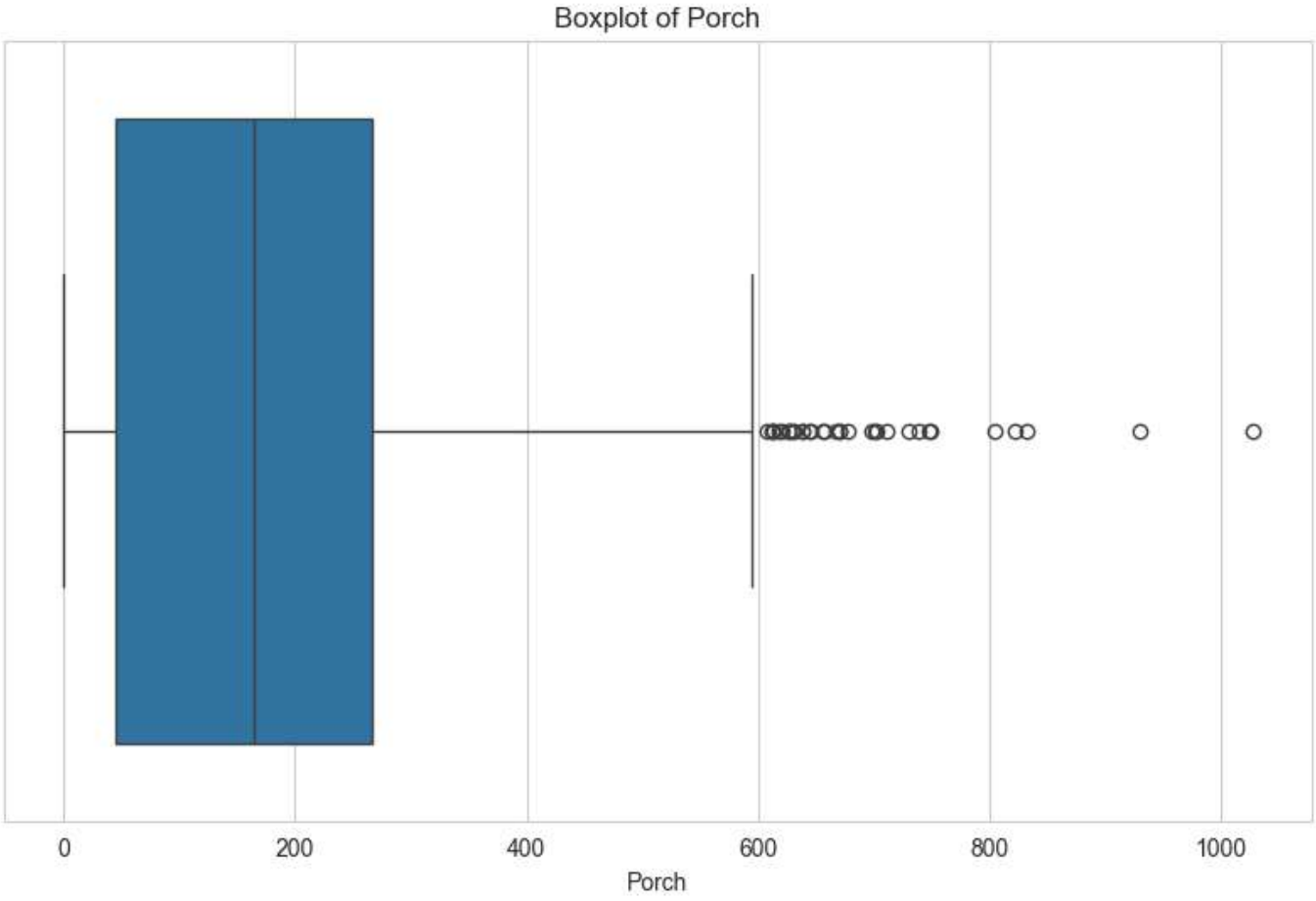
Consolidating the Porch features into a single variable, we sum the following attributes: 'WoodDeckSF: Wood deck area in square feet, OpenPorchSF: Open porch area in square feet, EnclosedPorch: Enclosed porch area in square feet, 3SsnPorch: Three season porch area in square feet, and 'ScreenPorch: Screen porch area in square feet. This new aggregated porch variable represents the total porch area of the house in square feet, capturing the comprehensive size across all floors and quality levels.

In [32]:
```
df['Porch'] = df['WoodDeckSF'] + df['OpenPorchSF'] + df['EnclosedPorch'] + df['3SsnPorch'] +
```

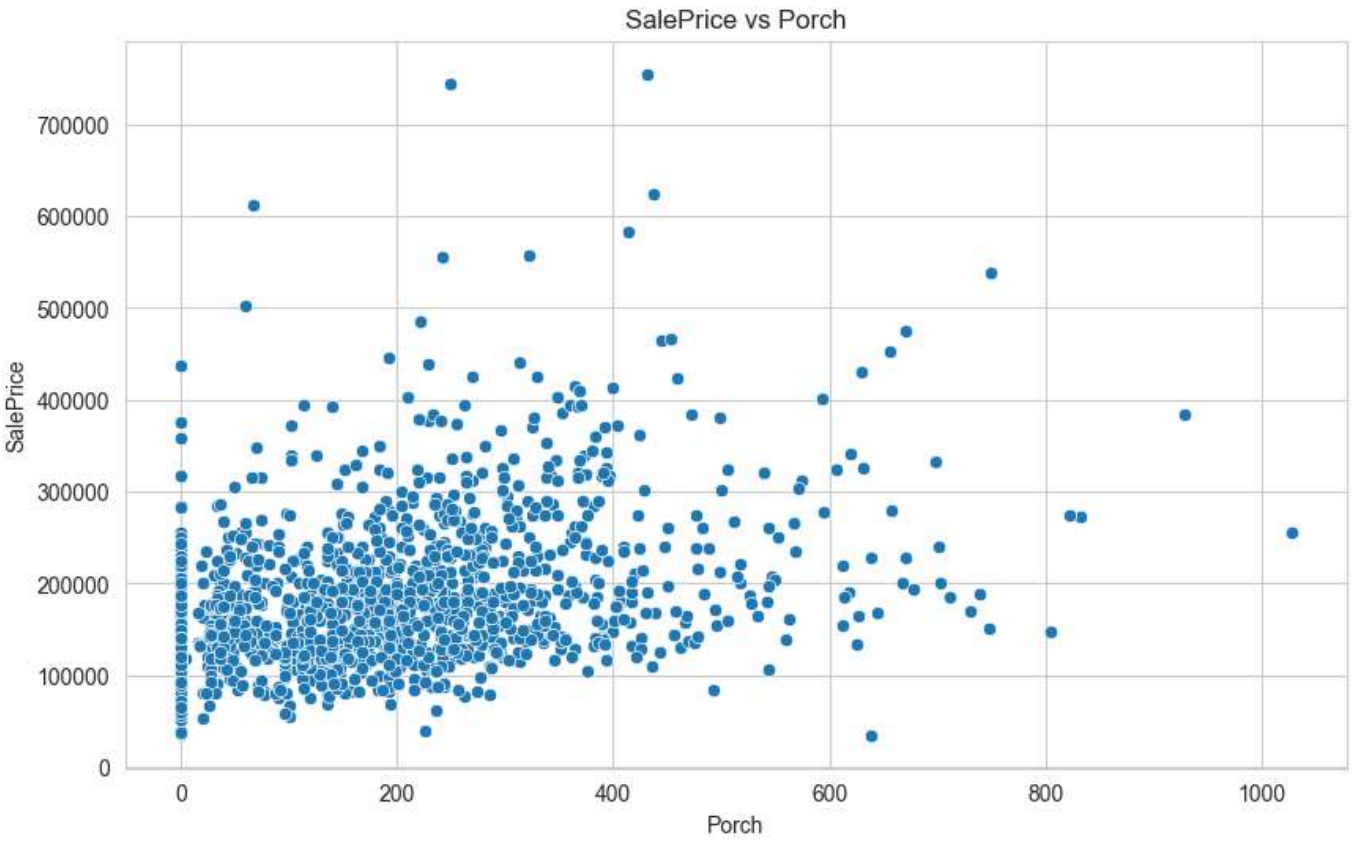
In [33]:
```
# Exploring TotalSF
print("Statistical Summary:", 'Porch')
print(df['TotalSF'].describe())

plt.figure(figsize=(10, 6))
df['Porch'].hist(bins=30)
plt.title(f'Histogram of {'Porch'}')
plt.xlabel('Porch')
plt.ylabel('Frequency')
plt.show()

plt.figure(figsize=(10, 6))
sns.boxplot(data=df, x='Porch')
plt.title(f'Boxplot of {'Porch'}')
plt.show()
```

```
Statistical Summary: Porch
count     1460.000000
mean      5145.786301
std       1647.196985
min        668.000000
25%       4028.000000
50%       4958.000000
75%       6017.000000
max      23504.000000
Name: TotalSF, dtype: float64
```





In [34]:
```
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df, x=df['Porch'], y='SalePrice')
plt.title(f'SalePrice vs Porch')
plt.xlabel('Porch')
plt.ylabel('SalePrice')
plt.show()
```



In [35]:
```
# Creating the final list of features to be included in the model creation
features_to_delete = ['WoodDeckSF', 'OpenPorchSF', 'EnclosedPorch', '3SsnPorch', 'ScreenPorch
numerical_features = [x for x in numerical_features if x not in features_to_delete]
numerical_features.append('Porch')
print(numerical_features)
print()
print(f"Number of Numerical features: {len(numerical_features)}")
```

```
['MSSubClass', 'LotFrontage', 'LotArea', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemod
Add', 'MasVnrArea', 'KitchenAbvGr', 'TotRmsAbvGrd', 'GarageYrBlt', 'GarageArea', 'PoolArea',
'MiscVal', 'TotalSF', 'Porch']

Number of Numerical features: 16
```

In [36]:
```
# Identify highly correlated features

# Calculate correlations between numerical features
correlation_matrix = df[numerical_features].corr()

# Set threshold for high correlation
threshold = 0.65

# Find pairs of highly correlated features
high_corr_pairs = []
correlation_matrix_abs = correlation_matrix.abs()

# Iterate over the columns for upper half of the matrix
for i in range(len(correlation_matrix_abs.columns)):
    for j in range(i+1, len(correlation_matrix_abs.columns)):
        if correlation_matrix_abs.iloc[i, j] >= threshold:
            colname_i = correlation_matrix_abs.columns[i]
            colname_j = correlation_matrix_abs.columns[j]
            high_corr_pairs.append((colname_i, colname_j, correlation_matrix_abs.iloc[i, j]))

print("Highly correlated pairs of numerical features:")
for pair in high_corr_pairs:
    print(f"{pair[0]} and {pair[1]}: Correlation = {pair[2]}")
```

```
Highly correlated pairs of numerical features:
OverallQual and TotalSF: Correlation = 0.6648302888837496
TotRmsAbvGrd and TotalSF: Correlation = 0.6788024533202122
```

In [37]:
```python
# We drop the highly correlated numerical features
numerical_features.remove('TotalSF')
```

In [38]:
```python
# Updated list of numerical features
print(numerical_features)
print()
print(f"Number of Numerical features: {len(numerical_features)}")
```

```
['MSSubClass', 'LotFrontage', 'LotArea', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemod
Add', 'MasVnrArea', 'KitchenAbvGr', 'TotRmsAbvGrd', 'GarageYrBlt', 'GarageArea', 'PoolArea',
'MiscVal', 'Porch']

Number of Numerical features: 15
```

In [39]:
```python
# Updated list of numerical features
print(categorical_features)
print()
print(f"Number of Categorical features: {len(categorical_features)}")
```

```
['MSZoning', 'LotShape', 'LandContour', 'LotConfig', 'LandSlope', 'BldgType', 'HouseStyle', 'M
asVnrType', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual', 'BsmtCond', 'BsmtExposure', 'B
smtFinType1', 'BsmtFinType2', 'HeatingQC', 'CentralAir', 'BsmtFullBath', 'BsmtHalfBath', 'Full
Bath', 'HalfBath', 'BedroomAbvGr', 'KitchenQual', 'Functional', 'Fireplaces', 'FireplaceQu',
'GarageType', 'GarageFinish', 'GarageCars', 'PavedDrive', 'Fence', 'YrSold', 'SaleCondition',
'MoSold']

Number of Categorical features: 35
```

In [40]:
```python
from scipy.stats import chi2_contingency
import itertools

# initialize list of lists
chi2_check = []

# loop through each pair of categorical features
for column in itertools.combinations(categorical_features, 2):
    # create a contingency table for the pair
    contingency_table = pd.crosstab(df[column[0]], df[column[1]])
    # calculate chi2 test statistic, p-value, degree of freedom and expected values
    chi2, p, dof, ex = chi2_contingency(contingency_table.values)
    chi2_check.append([column[0], column[1], p])

# convert to dataframe
res_chi2 = pd.DataFrame(data = chi2_check, columns=['Feature_A', 'Feature_B', 'P-Value'])

# sort the results by P-Value
res_chi2 = res_chi2.sort_values(by='P-Value', ascending=True)

# get the top ten most correlated categorical features
top_10_corr_cat_features = res_chi2.head(10)
# flatten the pairs to get a list of involved features
flattened_features = top_10_corr_cat_features[['Feature_A', 'Feature_B']].values.flatten()

# count the frequency of each feature
feature_frequency = pd.Series(flattened_features).value_counts()

print("Individual features most involved in correlations:")
print(feature_frequency)
```

```
Individual features most involved in correlations:
BsmtQual        4
BsmtFinType1    3
BsmtExposure    2
GarageType      2
GarageFinish    2
GarageCars      2
BsmtFinType2    1
Foundation      1
BsmtCond        1
Fireplaces      1
FireplaceQu     1
Name: count, dtype: int64
```

In [41]:
```python
# Dropping the highly correlated features
features_to_delete = ['BsmtQual', 'BsmtFinType1', 'GarageType', 'GarageCars', 'GarageFinish',
categorical_features = [x for x in categorical_features if x not in features_to_delete]
print(f'Number of Categorical Features: {len(categorical_features)}')
print()
print(f'Categorical Features are: {categorical_features}')
```

```
Number of Categorical Features: 29

Categorical Features are: ['MSZoning', 'LotShape', 'LandContour', 'LotConfig', 'LandSlope', 'B
ldgType', 'HouseStyle', 'MasVnrType', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtCond', 'Bsm
tFinType2', 'HeatingQC', 'CentralAir', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath', 'HalfBath',
'BedroomAbvGr', 'KitchenQual', 'Functional', 'Fireplaces', 'FireplaceQu', 'PavedDrive', 'Fenc
e', 'YrSold', 'SaleCondition', 'MoSold']
```

In [42]:
```python
print(f'Number of Numerical Features: {len(numerical_features)}')
print()
print(f'Numerical Features are: {numerical_features}')
```

```
Number of Numerical Features: 15

Numerical Features are: ['MSSubClass', 'LotFrontage', 'LotArea', 'OverallQual', 'OverallCond',
'YearBuilt', 'YearRemodAdd', 'MasVnrArea', 'KitchenAbvGr', 'TotRmsAbvGrd', 'GarageYrBlt', 'Gar
ageArea', 'PoolArea', 'MiscVal', 'Porch']
```

In [43]:
```python
# Get a combined list of all numerical and categorical features
all_features = numerical_features + categorical_features
all_features.append(target_variable)

# Find columns to drop - those not in all_features
drop_columns = [col for col in df.columns if col not in all_features]

# Drop these columns from df
df = df.drop(columns=drop_columns)

print(df.head())
```

```
  MSSubClass MSZoning  LotFrontage  LotArea LotShape LandContour LotConfig  \
0         60       RL         65.0     8450      Reg         Lvl    Inside
1         20       RL         80.0     9600      Reg         Lvl       FR2
2         60       RL         68.0    11250      IR1         Lvl    Inside
3         70       RL         60.0     9550      IR1         Lvl    Corner
4         60       RL         84.0    14260      IR1         Lvl       FR2

  LandSlope BldgType HouseStyle  ...  GarageArea  PavedDrive  PoolArea  Fence  \
0       Gtl     1Fam     2Story  ...         548           Y         0      0
1       Gtl     1Fam     1Story  ...         460           Y         0      0
2       Gtl     1Fam     2Story  ...         608           Y         0      0
3       Gtl     1Fam     2Story  ...         642           Y         0      0
4       Gtl     1Fam     2Story  ...         836           Y         0      0

  MiscVal  MoSold YrSold SaleCondition SalePrice Porch
0       0       2   2008        Normal    208500    61
1       0       5   2007        Normal    181500   298
2       0       9   2008        Normal    223500    42
3       0       2   2006       Abnorml    140000   307
4       0      12   2008        Normal    250000   276

[5 rows x 45 columns]
```

In [44]:
```
# Apply one-hot encoding to each column with categorical data in your DataFrame
df_encoded = pd.get_dummies(df, columns=categorical_features)

print(df_encoded.head())
```

```
  MSSubClass  LotFrontage  LotArea  OverallQual  OverallCond  YearBuilt  \
0         60         65.0     8450            7            5       2003
1         20         80.0     9600            6            8       1976
2         60         68.0    11250            7            5       2001
3         70         60.0     9550            7            5       1915
4         60         84.0    14260            8            5       2000

   YearRemodAdd  MasVnrArea  KitchenAbvGr  TotRmsAbvGrd  ...  MoSold_3  \
0          2003       196.0             1             8  ...     False
1          1976         0.0             1             6  ...     False
2          2002       162.0             1             6  ...     False
3          1970         0.0             1             7  ...     False
4          2000       350.0             1             9  ...     False

   MoSold_4  MoSold_5  MoSold_6  MoSold_7  MoSold_8  MoSold_9  MoSold_10  \
0     False     False     False     False     False     False      False
1     False      True     False     False     False     False      False
2     False     False     False     False     False      True      False
3     False     False     False     False     False     False      False
4     False     False     False     False     False     False      False

   MoSold_11  MoSold_12
0      False      False
1      False      False
2      False      False
3      False      False
4      False       True

[5 rows x 162 columns]
```

In [45]:
```
# Print all column names
print(df_encoded.columns.tolist())
```

```
['MSSubClass', 'LotFrontage', 'LotArea', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemod
Add', 'MasVnrArea', 'KitchenAbvGr', 'TotRmsAbvGrd', 'GarageYrBlt', 'GarageArea', 'PoolArea',
'MiscVal', 'SalePrice', 'Porch', 'MSZoning_C (all)', 'MSZoning_FV', 'MSZoning_RH', 'MSZoning_R
L', 'MSZoning_RM', 'LotShape_IR1', 'LotShape_IR2', 'LotShape_IR3', 'LotShape_Reg', 'LandContou
r_Bnk', 'LandContour_HLS', 'LandContour_Low', 'LandContour_Lvl', 'LotConfig_Corner', 'LotConfi
g_CulDSac', 'LotConfig_FR2', 'LotConfig_FR3', 'LotConfig_Inside', 'LandSlope_Gtl', 'LandSlope_
Mod', 'LandSlope_Sev', 'BldgType_1Fam', 'BldgType_2fmCon', 'BldgType_Duplex', 'BldgType_Twnh
s', 'BldgType_TwnhsE', 'HouseStyle_1.5Fin', 'HouseStyle_1.5Unf', 'HouseStyle_1Story', 'HouseSt
yle_2.5Fin', 'HouseStyle_2.5Unf', 'HouseStyle_2Story', 'HouseStyle_SFoyer', 'HouseStyle_SLvl',
'MasVnrType_0', 'MasVnrType_BrkCmn', 'MasVnrType_BrkFace', 'MasVnrType_Stone', 'ExterQual_Ex',
'ExterQual_Fa', 'ExterQual_Gd', 'ExterQual_TA', 'ExterCond_Ex', 'ExterCond_Fa', 'ExterCond_G
d', 'ExterCond_Po', 'ExterCond_TA', 'Foundation_BrkTil', 'Foundation_CBlock', 'Foundation_PCon
c', 'Foundation_Slab', 'Foundation_Stone', 'Foundation_Wood', 'BsmtCond_0', 'BsmtCond_Fa', 'Bs
mtCond_Gd', 'BsmtCond_Po', 'BsmtCond_TA', 'BsmtFinType2_0', 'BsmtFinType2_ALQ', 'BsmtFinType2_
BLQ', 'BsmtFinType2_GLQ', 'BsmtFinType2_LwQ', 'BsmtFinType2_Rec', 'BsmtFinType2_Unf', 'Heating
QC_Ex', 'HeatingQC_Fa', 'HeatingQC_Gd', 'HeatingQC_Po', 'HeatingQC_TA', 'CentralAir_N', 'Centr
alAir_Y', 'BsmtFullBath_0', 'BsmtFullBath_1', 'BsmtFullBath_2', 'BsmtFullBath_3', 'BsmtHalfBat
h_0', 'BsmtHalfBath_1', 'BsmtHalfBath_2', 'FullBath_0', 'FullBath_1', 'FullBath_2', 'FullBath_
3', 'HalfBath_0', 'HalfBath_1', 'HalfBath_2', 'BedroomAbvGr_0', 'BedroomAbvGr_1', 'BedroomAbvG
r_2', 'BedroomAbvGr_3', 'BedroomAbvGr_4', 'BedroomAbvGr_5', 'BedroomAbvGr_6', 'BedroomAbvGr_
8', 'KitchenQual_Ex', 'KitchenQual_Fa', 'KitchenQual_Gd', 'KitchenQual_TA', 'Functional_Maj1',
'Functional_Maj2', 'Functional_Min1', 'Functional_Min2', 'Functional_Mod', 'Functional_Sev',
'Functional_Typ', 'Fireplaces_0', 'Fireplaces_1', 'Fireplaces_2', 'Fireplaces_3', 'FireplaceQu
_0', 'FireplaceQu_Ex', 'FireplaceQu_Fa', 'FireplaceQu_Gd', 'FireplaceQu_Po', 'FireplaceQu_TA',
'PavedDrive_N', 'PavedDrive_P', 'PavedDrive_Y', 'Fence_0', 'Fence_GdPrv', 'Fence_GdWo', 'Fence
_MnPrv', 'Fence_MnWw', 'YrSold_2006', 'YrSold_2007', 'YrSold_2008', 'YrSold_2009', 'YrSold_201
0', 'SaleCondition_Abnorml', 'SaleCondition_AdjLand', 'SaleCondition_Alloca', 'SaleCondition_F
amily', 'SaleCondition_Normal', 'SaleCondition_Partial', 'MoSold_1', 'MoSold_2', 'MoSold_3',
'MoSold_4', 'MoSold_5', 'MoSold_6', 'MoSold_7', 'MoSold_8', 'MoSold_9', 'MoSold_10', 'MoSold_1
1', 'MoSold_12']
```

In [46]:

```python
from sklearn.linear_model import RidgeCV, LassoCV, ElasticNetCV
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline

# Separate features and target variable
X = df_encoded.drop([target_variable], axis=1)
y = df_encoded[target_variable]

# Initialize models with their CV-variants
models = {
    'Ridge': make_pipeline(StandardScaler(), RidgeCV(cv=5)),
    'Lasso': make_pipeline(StandardScaler(), LassoCV(cv=5, random_state=0)),
    'ElasticNet': make_pipeline(StandardScaler(), ElasticNetCV(cv=5, random_state=0))
}

# Loop over models
for model_name, model in models.items():
    print(f"Fitting {model_name}...")

    # Fit the model
    model.fit(X, y)

    # Perform cross-validation
    scores = cross_val_score(model, X, y, cv=5)

    # Print cross-validation scores
    print(f"{model_name} Cross-validation scores: ", scores)
```

```
Fitting Ridge...
Ridge Cross-validation scores:  [0.86029598 0.76357456 0.83405198 0.82928329 0.70995818]
Fitting Lasso...
Lasso Cross-validation scores:  [0.85306594 0.79128531 0.81538715 0.83316668 0.73125862]
Fitting ElasticNet...
ElasticNet Cross-validation scores:  [0.21435521 0.19953592 0.17694457 0.19104858 0.1769559 ]
```

In [47]:
```
# Load the test data
df_test_raw = pd.read_csv('test.csv')
df_test = df_test_raw.copy()  # Creating a copy to preserve raw data

# Save 'Id'
df_test_id = df_test_raw[['Id']]

# Add the Porch Variable
df_test['Porch'] = df_test['WoodDeckSF'] + df_test['OpenPorchSF'] + df_test['EnclosedPorch'] +

# Add the TotalSF Variable
df_test['TotalSF'] = df_test['GrLivArea'] + df_test['1stFlrSF'] + df_test['2ndFlrSF'] + df_te

# Ensures that the test DataFrame has the necessary features for prediction
if target_variable in all_features:
    all_features.remove(target_variable)  # removing target variable from the feature list

df_test = df_test[all_features]

# Test DataFrame also needs to go from categorical to one-hot encoding
# And has the same columns as the training DataFrame
df_test_encoded = pd.get_dummies(df_test, columns=categorical_features)

# Ensure the test DataFrame has the same columns as the training DataFrame
missing_cols = set(df_encoded.columns) - set(df_test_encoded.columns)
for c in missing_cols:
    df_test_encoded[c] = 0
df_test_encoded = df_test_encoded[df_encoded.columns]

# Remove the target_variable if it is still in df_test_encoded
df_test_encoded = df_test_encoded.drop(target_variable, errors='ignore', axis=1)

# Filling NaN values with median of the columns
df_test_encoded = df_test_encoded.fillna(df_test_encoded.median())

# Loop over models
for model_name, model in models.items():
    # Generate predictions
    y_pred = model.predict(df_test_encoded)

    # Create a DataFrame for predictions and add the 'Id' column
    df_pred = pd.DataFrame()
    df_pred['Id'] = df_test_id
    df_pred['SalePrice'] = y_pred

    # Write to a CSV file
    df_pred.to_csv(f'{model_name}_predictions2.csv', index=False, header=True, float_format='

    # Print a status update
    print(f"{model_name} predictions have been written to {model_name}_predictions2.csv")
```

```
Ridge predictions have been written to Ridge_predictions2.csv
Lasso predictions have been written to Lasso_predictions2.csv
ElasticNet predictions have been written to ElasticNet_predictions2.csv
```

In [48]:
```
from sklearn.linear_model import Lasso
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import make_pipeline

# Define the pipeline
pipe = make_pipeline(PolynomialFeatures(), Lasso())

# Define the parameter search range
param_grid = {
```

```
    'polynomialfeatures__degree': range(1, 4),  # Trying degrees 1 to 3
    'lasso__alpha': [50, 100, 250]   # Trying alpha values 50, 100, 250
}

# Define GridSearchCV object
grid = GridSearchCV(pipe, param_grid=param_grid, cv=5, n_jobs=-1, verbose=1)

# Fit the GridSearchCV object to the data
grid.fit(X, y)

# Extract the best model
model_1 = grid.best_estimator_

# Generate predictions with the best model
y_pred = model_1.predict(df_test_encoded)

# Create a DataFrame for predictions and add the 'Id' column
df_pred = pd.DataFrame()
df_pred['Id'] = df_test_id
df_pred['SalePrice'] = y_pred

# Write to a CSV file
df_pred.to_csv('lasso_polynomial_predictions_2.csv', index=False, header=True, float_format='
```

```
# Print a status update
print("Lasso Polynomial predictions have been written to lasso_polynomial_predictions_2.csv")
print('Best parameters found: ', grid.best_params_)
```

```
Fitting 5 folds for each of 9 candidates, totalling 45 fits
Lasso Polynomial predictions have been written to lasso_polynomial_predictions_2.csv
Best parameters found:  {'lasso__alpha': 250, 'polynomialfeatures__degree': 1}
```

In [53]:

```
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import make_pipeline

# Define the pipeline
pipe = make_pipeline(PolynomialFeatures(), Ridge())

# Define the parameter search range
param_grid = {
    'polynomialfeatures__degree': range(1, 4),  # Trying degrees 1 to 3
    'ridge__alpha': [10, 15, 20, 25, 30, 35, 40]   # Trying alpha values 10, 15, 20, 25, 30,
}

# Define GridSearchCV object
grid = GridSearchCV(pipe, param_grid=param_grid, cv=5, n_jobs=-1, verbose=1)

# Fit the GridSearchCV object to the data
grid.fit(X, y)

# Extract the best model
model_2 = grid.best_estimator_

# Generate predictions with the best model
y_pred = model_2.predict(df_test_encoded)

# Create a DataFrame for predictions and add the 'Id' column
df_pred = pd.DataFrame()
df_pred['Id'] = df_test_id
df_pred['SalePrice'] = y_pred

# Write to a CSV file
df_pred.to_csv('ridge_polynomial_predictions_2.csv', index=False, header=True, float_format='

# Print a status update
```

```
print("Ridge Polynomial predictions have been written to ridge_polynomial_predictions_2.csv")
print('Best parameters found: ', grid.best_params_)
```

```
Fitting 5 folds for each of 21 candidates, totalling 105 fits
Ridge Polynomial predictions have been written to ridge_polynomial_predictions_2.csv
Best parameters found:  {'polynomialfeatures__degree': 1, 'ridge__alpha': 25}
```

In [50]:
```
from sklearn.linear_model import ElasticNet
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import make_pipeline

# Define the pipeline
pipe = make_pipeline(PolynomialFeatures(), ElasticNet())

# Define the parameter search range
param_grid = {
    'polynomialfeatures__degree': range(1, 4),  # Trying degrees 1 to 3
    'elasticnet__alpha': [0.01, 0.05, 0.1],   # Trying alpha values 0.1, 1, and 10
    'elasticnet__l1_ratio': [0.4, 0.5, 0.6]  # Trying L1 ratio (mixing parameter for L1 and L
}

# Define GridSearchCV object
grid = GridSearchCV(pipe, param_grid=param_grid, cv=5, n_jobs=-1, verbose=1)

# Fit the GridSearchCV object to the data
grid.fit(X, y)

# Extract the best model
model_3 = grid.best_estimator_

# Generate predictions with the best model
y_pred = model_3.predict(df_test_encoded)

# Create a DataFrame for predictions and add the 'Id' column
df_pred = pd.DataFrame()
df_pred['Id'] = df_test_id
df_pred['SalePrice'] = y_pred

# Write to a CSV file
df_pred.to_csv('elasticnet_polynomial_predictions_2.csv', index=False, header=True, float_for

# Print a status update
print("ElasticNet Polynomial predictions have been written to elasticnet_polynomial_predictio
print('Best parameters found: ', grid.best_params_)
```

```
Fitting 5 folds for each of 27 candidates, totalling 135 fits
ElasticNet Polynomial predictions have been written to elasticnet_polynomial_predictions_2.csv
Best parameters found:  {'elasticnet__alpha': 0.05, 'elasticnet__l1_ratio': 0.5, 'polynomialfe
atures__degree': 1}
```

The best score 0.17184 on Kaggle is returned by our Lasso model.

In [54]:
```
# Re-creating the Lasso Model

# Initialize Lasso model
lasso_model = make_pipeline(StandardScaler(), LassoCV(cv=5, random_state=0))

# Fit the Lasso model
lasso_model.fit(X, y)

# Perform cross-validation
scores = cross_val_score(lasso_model, X, y, cv=5)
```

```
# Print cross-validation scores
print("Lasso Cross-validation scores: ", scores)
```

```
Lasso Cross-validation scores:  [0.85306594 0.79128531 0.81538715 0.83316668 0.73125862]
```

In [57]:
```
# Code for identifying the most impactful features
# Get the coefficients
coefs = lasso_model.named_steps['lassocv'].coef_

# If Lasso fits intercept, we need to adjust handling of coefficients
if lasso_model.named_steps['lassocv'].fit_intercept:
        coefs = coefs[1:]  # drop first coef

# Your original feature names are here
feature_names = X.columns

# Create a DataFrame for coefficients
df_coefs = pd.DataFrame(list(zip(feature_names, coefs)), columns=["Feature", "Coefficient"])
df_coefs.sort_values(by="Coefficient", ascending=False, inplace=True)

print(df_coefs[:15])
```

```
              Feature   Coefficient
2             LotArea  23512.849789
8         KitchenAbvGr  12363.232246
108     BedroomAbvGr_8   9043.688564
10         GarageYrBlt   8417.092221
96          FullBath_2   7657.397947
6         YearRemodAdd   7196.729958
1          LotFrontage   5185.720666
121       Fireplaces_1   5132.109243
13             MiscVal   4369.607298
52    MasVnrType_Stone   3394.404878
63   Foundation_CBlock   3309.211871
33       LandSlope_Gtl   3004.900549
98          HalfBath_0   2931.432311
3          OverallQual   2765.820458
124      FireplaceQu_0   2707.349560
```

In [ ]:
```
# Code for creating and html file
!jupyter nbconvert --to html Lasso_Ridge_ElasticNet.ipynb
```